STRUBAL +™

An M6800 Compiler

By Robert D. Grappel

and

Jack E. Hemenway

Published by:
Hemenway Associates, Inc.
101 Tremont St.
Suite 208
Boston, MA 02108

ISBN 0-935026-02-9
Library of Congress Card Catalog —79-90251

The authors of the programs provided with this book have carefully reviewed them to ensure their performance in accordance with the specifications described in the book. Neither the authors nor Hemenway Associates, Inc., however, makes any warranties whatever concerning the programs, nor assumes responsibility of any kind for errors in the programs or for the consequences of any such errors.

The programs are the sole property of Hemenway Associates, Inc. and have been registered with the United States Copyright Office. Lawful users of this book may use the programs themselves, but may not make copies or translations other than as necessary to use the programs. It is a violation of the federal copyright laws, punishable by fines and/or imprisonment for anyone to copy or translate any portion of this book for any other purpose, including for purposes of resale, license or lease to others.

Copyright © 1979, Hemenway Associates, Inc.

TABLE OF CONTENTS

STRUBAL+ Manual and Language Description

Introduction to STRUBAL+ Programming..... 1

STRUBAL+ as an Extendable Language.. 1
Compiler Output Options..... 5
Symbol-table Output Format..... 5
Operating Instructions..... 6
Simple STRUBAL+ Example..... 7

Fundamental STRUBAL+ Concepts..... 8

Character Set..... 8
Constants and Variables..... 8
Label Names..... 9
Arithmetic Expressions..... 10
Transfer Statements..... 13
Input/Output..... 14
The STOP Keyword..... 14
The END Keyword..... 15

STRUBAL+ and Program Development..... 15

STRUBAL+ Variables..... 15
Arithmetic Expressions..... 15
Program Control..... 16
Loop Control..... 18
Input/Output Statements..... 19
Arrays..... 24
Strings..... 24
STRUBAL+ and Subroutines..... 26
Special Keywords..... 28

Error Messages..... 30

Assembly-pass Errors..... 31
Compiler Errors..... 32
Runtime Errors..... 33

Statement Types..... 33

Assembly Language Features..... 54

Statement Characteristics..... 55
Macro-Argument Lists..... 57
Evaluation of Expressions..... 57
Comment Field..... 57
Addressing Modes..... 58
Address Type Formats..... 59
Pseudo-instructions..... 61
MACROS..... 64
Advanced Macro Features..... 66

STRUBAL+ Internal Detail Descriptions

Modification of the STRUBAL+ Compiler..... 68

Modification of the Runtime Package..... 70

Linkage to the Runtime Packages..... 71

Compiler Routines..... 77

- Initialization and Main..... 77
- Unary Operators..... 86
- Floating-point Arithmetic..... 90
- String-operation Routines..... 91
- Parsing Routines..... 94
- Mnemonic and Symbol Routines..... 98
- Expression Evaluation Routines.....100
- Listing Routines.....106
- Opcode Processing Routines.....113
- Utility Routines.....116
- Pseudo-operation Processing.....118
- Assorted Output Routines.....133
- Mass-Storage Driver Linkages.....133

Program Linkage Examples.....134

Runtime Package Internals

Subroutines in RUNTIM.....137

- Sample Runtime Call.....138
- Logical Routines.....138
- Integer Arithmetic.....139
- Byte Input/Output Routines.....142
- Random-number Routine.....142
- String Comparison.....142
- Floating-point Move.....144
- Input/Output Routines.....144
- Error Message Routines.....147
- STRUBAL+ File Format.....149
- Sequential-file I/O Routines.....149
- File I/O Drivers.....152

Floating-point Input/Output Routines.....156

Floating-point Package Routines.....159

Scientific Function Package Routines.....162

Common Areas in Runtime Packages.....167

Program Listings

STRUBAL+ Compiler.....168

CP/68 Drivers for Compiler.....243

Compiler Memory Dump at 2000 hex.....246

Runtime Package.....252

Floating-point Package.....272

Scientific Subroutine Package.....283

CP/68 STRUBAL+ Disk Drivers.....292

INDEX.....297

Part I

INTRODUCTION TO STRUBAL+ PROGRAMMING

I.1 Microcomputer Programming

During the past four years microcomputers have been making news. The biggest news has been the stories about individual users and their personal computers. The first people to start using microcomputers effectively programmed them in assembly language. The next step in the chain brought thousands of users to the level of using BASIC. BASIC is usually an interpreter-based language that is offered in several versions, with significant differences between the versions.

A compiler converts high-level language programs into machine (object) programs. This object program can be executed or saved. The compiler is used once to generate the object program. The object program can be executed from its saved medium as often as needed and does not require the compiler's presence, thereby leaving more memory for larger programs.

I.2 The STRUBAL+ language

The name STRUBAL+ is an acronym for Structured Basic Language. It contains elements of BASIC, PL/M and assembly language. The language allows multiline user-defined STRUBAL+ functions and, in the assembly passes of the compiler (passes two and three) allows the use of Macros. This advanced function provides the mechanism to add user-defined KEYWORDS and, in fact, extend the syntax of the language. This may make some parts of the language seem very familiar, while others appear to be awkward constructions. Once you are familiar with STRUBAL+, we are sure that you will find it more powerful and easier to use than any of its predecessors.

There is, actually, no such thing as a structured language, but there are languages that will allow the use of structured programming. STRUBAL+ is one of those languages.

Designed for compilation on microcomputer systems, STRUBAL+ has a compiler that produces intermediate assembly-language code that may be optimized by the user. It provides a full set of functions and operations for manipulation of several types of data.

STRUBAL+ AS AN EXTENDABLE LANGUAGE

One of STRUBAL+'s powerful features is the ability to add new keywords to the language. The new keywords, defined in terms of existing structures, allow the programmer to customize the compiler to suit individual requirements. Keyword definition can take place in either the assembler or compiler portion of STRUBAL+. In the assembler structures called MACROs provide this flexibility. STRUBAL+ contains a complete 6800 assembler. There are, however, certain features not present in the assembler that may be desired. The pseudo-ops FCB and FDB, for example, allow only a single constant term. Some assemblers allow a list of terms, separated by commas, in these pseudo-ops. To add this capacity to STRUBAL+ you might create the macro MFCB (multiple FCB) as follows:

```
MFCB   MACRO          DEFINE MACRO
.CNT   SET &0         GET COUNT OF ARGUMENTS
       IFC .CNT       CHECK ARGUMENT COUNT
       FCB &1         FORM FIRST TERM
.CNT   SET .CNT-1
       NIFC
*
       IFC .CNT       CHECK ARGUMENT COUNT
       FCB &2         FORM SECOND TERM
.CNT   SET .CNT-1
       NIFC
*
       .....
       .....          SIMILAR LINES FOR
       .....          ADDITIONAL TERMS
       .....
*
       MEND           END OF MACRO DEFIN
```

The MACRO statement starts the macro definition and names the macro. This name will be given to the new keyword in the assembler. The MEND statement ends the macro definition. The ampersand (&) indicates a replacable parameter. When the macro name is given as a keyword, the list of arguments following the name will replace the ampersands in the order they appear with the first argument replacing &1, the second &2 and so on. The argument &0 is reserved for the value of the number of arguments used in this particular use of the keyword. For example:

```
MFCB $0D,$0A,$04
```

will cause &0 to be replaced by the number 3; the string "$0D" replaces &1 and so on. The SET pseudo-op assigns a value to a symbol. Unlike the EQU pseudo-op it allows the symbol to take on different values at different times. In the example CNT takes on the value of &0, then &0-1 and so on. The IFC (if-condition) and NIFC (end-of-condition) pseudo-ops allow macros to vary the code they produce. The block of lines between the pseudo-ops is skipped over if the argument of the IFC has the value zero. In the example the use of IFC, NIFC and SET enables the generation of the proper number of FCB pseudo-ops -- one for each argument of MFCB. Thus:

```
MFCB $0D,$0A,$04
```

produces the assembler lines

```
FCB $0D
FCB $0A
FCB $04
```

This expansion is invisible to the programmer unless the macro-expansion option M is specified. The programmer has available the keyword MFCB; a similar macro defintition could create MFDB.

Another use of macros is to define instructions that do not exist on the machine which is running STRUBAL+. The 6809, for example, will have "long branch" instructions that permit relative address branches throughout memory. Although the 6800 lacks this feature, STRUBAL+ users can assemble it using macros. The opcode for the instruction (LBRA) is 16 hex. The instruction is 3-bytes long; the second and third bytes form the relative offset for the

branch. Thus:

```
LBRA   MACRO          DEFINE LONG BRANCH
.LOC   SET *          GET CURRENT-LOCATION POINTER
       FCB $16        FORM OPCODE
       FDB &1-.LOC-3  FORM RELATIVE OFFSET
       MEND           END DEFINITION
```

Clearly macros provide a powerful addition to the assembler. They can permit one computer to produce code for another machine; they can add features to the existing assembler; they can facilitate the exchange of programs between users whose translation programs use differing syntax. By way of explaining the latter point, STRUBAL+ requires at least one blank between the opcode and register symbol on load and store instructions (LDA A or STA B). To make STRUBAL+ recognize assembly code with no blanks:

```
LDAA   MACRO          DEFINE NEW LDA
       LDA A &1       FORM STRUBAL+ FORM
       MEND
*
STAB   MACRO          DEFINE NEW STA
       STA B &1       FORM STRUBAL+ FORM
       MEND
```

In STRUBAL+ code macro capabilities are provided through the FUNCTION statement and its end statement FEND. The function name follows the keyword and STRUBAL+ statements are permitted in the definition. Suppose, for example, that a SQRT function was required. STRUBAL+ furnishes only the more general XTOY.

```
FUNCTION SQRT
&2=&1 .XTOY. 0.5
FEND
```

To use the new keyword

```
DEFINE ARG1, ARG2
INPUT/,'ENTER VALUE',ARG1
SQRT ARG1,ARG2
IF ARG2*ARG2 .EQ. ARG1 PRINT/,'PERFECT SQUARE'
STOP
END
```

Whole new sets of capabilities can be added to STRUBAL+ in this manner.

SYNTAX

The syntax of a STRUBAL+ statement makes it very straightforward to use. An asterisk in the first column indicates that the statement is strictly a comment statement, and is for documentation purposes. If a statement label is desired, it must start (with an alphabetic character) in the first column. If no label is used the first column is left blank. The next element to the right (starting in column two, if there is no label) is the keyword. The keyword defines the syntax of the rest of the line. STRUBAL+ statements must be made on a single line. There are no provisions for continuing a statement on a subsequent line.

CRUTCH CODING

The assembly-language syntax used is similar to that of Motorola. STRUBAL+ allows assembly language to be imbedded anywhere in a program. Such statements will be passed intact through the compilation and will be assembled. Imbedded assembly language statements do not require saving registers. STRUBAL+ expects no registers to be saved. Further details concerning the use of assembly language may be found in Part V of this manual.

RUNTIME

STRUBAL+ programs can call on a set of subroutine libraries to perform various operations that are not a part of the computer's basic instruction set. These libraries are linked in with the modules produced by the STRUBAL+ compiler to form the RUNTIME LOAD MODULE.

These RUNTIME libraries are called RUNTIME PACKAGES. The programmer should include in the runtime module only those packages which contain functions necessary to the program.

The LINKAGE EDITOR will flag as unresolved all runtime routines that are not loaded; however this will not cause problems as long as the program does not attempt to use the unresolved routines.

STRUBAL+ is supplied with four runtime packages.

RUNTM

This is the main package--it must be loaded with every STRUBAL+ program module. RUNTM provides the "dispatch" function which calls all other runtime routines. It contains the integer arithmetic, logical, comparison, the I/O routines and the error routines. RUNTM uses two system routines from the computer's operating system. The first, OUTEEE, takes one ASCII character from the A accumulator, and transfers it to the system console device. The second--INEEE--gets a character from the system console device, and places it in the A accumulator. All other registers must be saved in both system calls. RUNTM occupies about 3.5k bytes of RAM.

FLOT

This package provides the basic floating-point arithmetic routines. It should be included with any program that uses floating-point values. It occupies about 1.9k bytes of RAM.

SCIF

Here is where the scientific functions are brought in. Any program that uses SIN, COS, TAN, ASIN, ACOS, ATAN, LOG10, NATLOG, TENEXP, EXP or XTOY will need this package. It requires the use of FLOT. This package fits into about 2.0k bytes of RAM.

FILE DRIVERS

This package supports file I/O. It is system dependent; it uses routines as necessary to link with the particular file system in use. The drivers are called from RUNTM, and use routines from RUNTM. The file drivers must be loaded with any program that attempts file I/O. The size of the drivers depends on the system, but they typically run from 1 to 2k bytes.

Compiler Output Options

STRUBAL+ provides several forms of output as it compiles and then assembles a program. Several option switches are provided so that the user can control how much output will be produced. Options are requested at the beginning of pass one (compilation), and at the beginning of pass two (assembly).

The first and third passes can produce output files. Output file production (assembly code from pass one and object code from pass three) is controlled by the "O" (object) option switch. The first and third passes can produce an alphabetically sorted symbol table reference listing. This output is controlled by the "S" (symbols) option switch. The compilation pass can list assembly code as it is generated. This output is controlled by the "I" (intermediate) option switch. The listing of source statements (assembly or STRUBAL+) is controlled by the "L" (list) switch. A cross-reference file can be generated during compilation or assembly. This file is used to print a cross-reference listing. The "X" (XREF) switch must be specified. However, "X" and "O" options cannot be used at the same time. The result would be a garbled file, useless for XREF or further processing. The expansion of user-defined functions and assembly macros may be printed if desired. This output is controlled by the "M" (macro) option switch.

SYMBOL TABLE OUTPUT FORMAT

If the symbol table option (S) is specified, STRUBAL+ will print the symbol table at the end of the pass. The table is alphabetically sorted by the six-character symbol name. The stored symbol value and special flags are printed as described below. The value is printed in decimal and hexadecimal notation. The flags are printed as single letters.

COMPILATION

The value field is the sequence line number on which the value was defined for simple integer or floating-point symbols. The value for strings is the maximum length. The value for dimensioned variables is the first dimension. The bit flags are printed as follows:

D	Dimensioned variable
F	Floating-point variable
P	Procedure parameter
S	String variable
M	Function name

ASSEMBLY

The value is the address assigned to the symbol. The bit flags are:

C	Common Reference
N	Entry point
R	Relocatable
X	External Reference
M	Macro name

OPERATING INSTRUCTIONS

When the compiler is executed it types:"ENTER OPTIONS". Enter the options desired, separated by commas, and terminate with a carriage return.

OPTIONS: L - Listing printed
S - Sorted symbol table printed
I - Compiler-generated assembly code printed
Q - Return to operating system
O - Object code generated
X - Cross reference is created
M - Macro Expansion

NOTE: only one of O or X may be selected.

If an error is made while inputing the options, the "escape" key will return processing to the "ENTER OPTIONS" prompt, so long as the carriage return has not been made.

Next the compiler prompts for an input file. Enter the name of the input file followed by a carriage return.

If an X or an O has been selected as an option, the compiler will prompt for an output file name. Enter it followed by a carriage return.

If an error is detected by the file system, it will return to request a proper file name.

The compiler next prompts the operator to choose between compilation or assembly. The letter C indicates compilation, while A indicates assembly. The last letter entered before the carriage return is used.

If compilation is chosen, the compiler requests a name for the program. This name should be a valid STRUBAL+ name and will appear in the load map when the program is linked. The "escape" key will return processing to the prompt if an error is made. A carriage return terminates the name.

At this point, the compiler begins compilation or assembly processing of the input file. At the end of processing, the compiler returns to the "ENTER OPTIONS" prompt.

I.3 A very simple example of a STRUBAL+ program

This program will accept a positive integer from the programmer terminal. If the number is not six [6], it will print 1/2 of the input value and request another number for input.

```
     INTEGER INNUM,QUITNO,COUNT,OUTNUM
     DSTRING INAME(72)
*
* Initialize variables
*
     INNUM=0
     COUNT=0
     QUITNO=6
*
     INPUT //,'ENTER YOUR NAME',%INAME,//
     INPUT //,'ENTER AN INTEGER VALUE',INNUM,//
*
     WHILE INNUM .NE. QUITNO
          OUTNUM = INNUM/2
          PRINT //,'RESULT - ',[6],OUTNUM,//
          COUNT = COUNT+1
          INPUT //,'ENTER AN INTEGER VALUE',INNUM,//
     BLOCK
*
     PRINT //,[10],%INAME,'YOU HAVE RECEIVED',COUNT,'RESULTS',//
     STOP
     END
```

This program example presents some of STRUBAL+'S basic elements:

All of the lines that are preceeded by an asterisk in the first column are program comments. These are intended for program documentation, and have no effect on the execution of the program.

The STOP statement marks the end of the program. When this is reached during execution of the program, control is returned to the monitor.

The END statement marks the end of the compilation process.

INTEGER, DEFINE and DSTRING define the variables that will be used in the program, and specify that they may contain integer, floating-point and character variables respectively.

The expressions that follow the "Initialize Variables" comment establish the values that the variables to be used will start with.

The first INPUT statement results in the data being read from the keyboard into the variable INAME. INAME was defined by the DSTRING statement to contain a maximum of 72 characters.

The second INPUT statement reads the data keyed into the terminal into the integer variable INNUM and assures that the WHILE-BLOCK that follows will work properly on the first repetition. The maximum value for INNUM is 32767.

The PRINT statements provide the operator with feedback concerning the program. In the first PRINT statement the results of the calculation are presented, and in the second the number of times the program went through the

loop.

PART II

Fundamental STRUBAL+ Concepts

II.1 Character Set

The following characters have meaning in STRUBAL+. All other characters may appear in comments and quoted strings only. All letters are upper case.

The letters A through Z
Digits 0 through 9
Comma (,)
Quote (')
Pound Sign (#)
Percent Sign (%)
Dollar Sign ($)
Greater Than (>)
Brackets ([])
Parentheses ()
Equal Sign (=)
Plus Sign (+)
Minus Sign (-)
Asterisk/Times Sign (*)
Divide Sign (/)
Blank
Period (.)
Exclamation Point (!)
Ampersand (&)

II.2 Constants and Variables

The values that a program manipulates, such as INNUM in the sample program, are called variables.

A variable, then, is a symbolic name that the programmer gives to a quantity that is expected to change during the execution of the program.

A constant is a known value that will not change during program execution.

II.2.1 Integer variables

STRUBAL+ values are stored as 16-bit binary quantities in two's complement form, or as 9-byte BCD floating-point values. The maximum value for integers is 32767, and the minimum is -32768. Exceeding this range will cause a default to the proper limiting value, and an overflow message will be printed.

The keyword INTEGER reserves space for integer variables, which may be dimensioned if desired. If a variable name is used without being defined, it is automatically defined as an integer.

SAMPLE INTEGER VALUES:

+3 -51 1010

$F1 $0123 (HEX)

II.2.2 Floating-point Variables

The keyword DEFINE establishes storage for floating-point variables. All floating-point variables must be defined. Examples of floating point numbers would include:

8.8	1.123	.002
-1.5 E-9	3.256	33 E13

DEFINE A,B,C

for example: defines A, B and C as names for floating-point variables.

DEFINE VECTOR(10)

would define a floating-point array. Floating-point values have up to 14 significant digits, plus a 2-digit exponent. The maximum value is 1.E+99.

II.2.3 Strings

STRUBAL+ stores character strings that are generated by statements or through input/output, in the form of one-character-per-byte, with an end-string character appended. The end-string character is 04 (hex). String arrays can be formed. The maximum length of a string must be defined in a DSTRING statement prior to the use of the string. The maximum allowable string length is 32,766 characters. The definition of a character-valued variable is as follows:

DSTRING NAME (length)

NAME is any legal programmer-selected name. The length specififes the number of characters in the value. The length parameter does not include a count for the end-string character.

DSTRING S1(5),S2,S3(5,10)

would define three strings. S1 may hold up to 5 characters, S2 holds only one character, and S3 holds 10 strings, each of which holds up to 5 characters.

II.3 Label Names

Names in STRUBAL+ (whether they are for variables or keywords) must be composed of from one to seven alphanumeric characters. The first character MUST be alphabetic. The period (.) symbol is considered alphabetic in STRUBAL+, and used in compiler-generated labels and functions names. Labels have a maximum of six significant characters, with no imbedded blanks or non-alphanumeric characters being permitted. Since keywords are defined by their first four characters, a match of the first four characters of any label with the first four of any keyword will produce an invalid label.

CLOSET, for example, would be an invalid label since it shares its first four letters with the keyword CLOSE.

8TEMP would be invalid, since the first character is numeric rather than alphabetic.

ARNOLD would be a valid label.

II.4 Arithmetic Expressions

NOTE:
*****
*****

STRUBAL+ contains no precedence operators. All functions are evaluated strictly from left to right.

The available arithmetic operators are:

+	unary plus
+	addition
-	unary minus
-	subtraction
*	multiplication
/	division

There is also a series of trigonometric and logarithmic functions:

.ABS.....Absolute value
.ACOS.....Inverse Cosine (in radians)
.ASIN.....Inverse Sine (in radians)
.ATAN.....Inverse Tangent (in radians)
.COS.....Cosine (in radians)
.EXP.....Exponentiation
.FIX.....Floating-to-integer
.FLOAT.....Integer-to-floating point
.LOG10.....Base-10 Logarithm
.MOD.....Integer remainder
.NATLOG.....Natural Logarithm
.RAND.....Random Integer
.SGN.....Signum
.SINE.....Sine, (in radians)
.TAN.....Tangent (in radians)
.TENEXP.....10 to power
.XTOY.....Number to a power

Exponentiation has two forms: integer and floating. In the integer form it represents a base to a power.

I=2 .EXP. 2 returns 4

In the floating form:

R=.EXP. 5 gives e to the fifth power.

Trigonometric functions accept arguments in terms of radians, and inverse functions return values in radians. These functions give 7 to 8 decimal digits of accuracy. Note that all function names require a leading period. All single-argument functions have the syntax:

.FUNC. ARG

with the period trailing the function name optional. An example of this function type is the random integer. The RAND function yields an integer uniformly distributed over the range -32767 to +32767; the argument is ignored if its value is not zero.

.RAND 1 is ignored

If the argument is equal to zero, the random number generator will reinitialize its sequence.

.RAND 0 reinitializes

OR:

```
*
      A= .LOG10. 4
*
* RETURNS THE BASE-10 LOG OF 4
*
```

A typical example of an arithmetic expression might be:

```
*
      LET Y=.SIN(X)+7
*
* RETURNS THE VALUE OF THE SINE OF X
* RADIANS PLUS 7
*
```

Two-argument functions, such as MOD or XTOY, require an argument on either side of the function name.

EXAMPLES

```
*
      X=5 .MOD. 2
*
* RETURNS A VALUE OF 1
*
      Y = 2 .XTOY 5
*
* RETURNS A VALUE OF 32
*
```

Another set of functions, the logical functions, perform logical operations bit-wise on 16-bit integer variables. They may also combine expressions of either integer or floating type to form compound, logical expressions.

```
.AND..........Logical AND
.OR...........Logical OR
.NOT..........Logical Inverse
.XOR..........Logical Exclusive OR
```

In addition, it is often convenient to make comparisons of variables, or variables to constants, STRUBAL+ provides a series of comparison functions that make comparisons simple.

.EQ.....True if left argument equals right argument.

```
IF A .EQ. 1 THEN TELL
```

which will translate: if A is equal to 1, goto the statement labeled TELL. Similarly:

```
IF (A .EQ. 1) .OR. (B .NE. A) THEN GOOD
C = .NOT. (A .XOR. B)
```

.NE.....Not equal to
.LT.....Less than
.LE.....Less than or equal to
.GT.....Greater than
.GE.....Greater than or equal to

There are a few additional functions that will prove helpful as well.

.PEEK.....Returns the value of a byte in memory pointed to by the argument (integer type).

```
I = .PEEK. $8010
```

.EOF.....End of file. Requiring no arguments, it returns a logical true if an end-file was encountered during a file I/O operation.

.ERR.....If an error is encountered during a file operation, this will return a logical true.

Example: IF .ERR. THEN LABEL

Logical functions have uses beyond logical testing--they serve to manipulate bits in integer values. The NOT function, for example, can produce the one's complement of a value:

.NOT. $000F is equal to $FFF0

The AND function can mask a value to a certain number of bits:

$1234 .AND. $00FF is equal to $0034

The OR function can set bits in an integer:

I=IVAL .OR. 8 sets bit 3 in "I" plus any bits which were set in IVAL.

The XOR function can be used to compare integers for exact equality:

I .XOR. J equals zero if and only if I=J

II.5 Transfer Statements

Normally, a program will be executed sequentially; ie, a statement will be executed following the statement ahead of it, and before the statement after it. In some cases this may not be desirable. Consider the following example:

Suppose the programmer wishes to have the computer notify the operator if the value of any data input is greater than 12. Since the computer must check each entry to see if it exceeds 12 and then respond accordingly, a conditional branch is required. It might look like this:

```
IF X .GE. 13 THEN TELL
```

In English, this would read: If X is greater than or equal to 13, then go to the statement labelled TELL. In cases where the condition is false (ie, when X<13) the statement will be ignored, and the program proceeds to the next (sequential) statement.

There are three types of conditional branches, or IF statements. Each contains an expression which is evaluated. If the expression is true (NONZERO), then one branch is made; if the expression is false (ZERO) a different branch is made. The first form is the IF-THEN. A transfer label follows the THEN. If the expression is true, the program transfers to this label; otherwise the next statement is executed.

```
IF A .NE. B THEN LABEL
```

Reads, if A is not equal to B then go to the statement labeled LABEL.

A second form is the IF-THEN-ELSE. With this form, if the expression is false, the label following the ELSE indicates the statement to be executed next.

```
IF A .NE. B THEN LABEL ELSE NOT
```

If the expression is true, the execution is the same as before; the statement labeled LABEL is executed next. If the expression is false, the program jumps to the statement labeled NOT.

The third form has an executable statement following the expression. This statement will be executed if the expression evaluates true. The statement cannot be another IF statement.

```
IF A .NE. B LET A=A+1
```

IF A is not equal to B, A will be incremented by 1. The keyword LET is not optional here!

II.6 Input/Output

The Input/Output statements are those that allow the operator to interact with the program during its execution. The operator may be called on to supply data (e.g. the date) or to make decisions concerning program execution. Our original sample program (I.3) gives an example of the format that an INPUT statement might take. There are several steps that must be accomplished for the program to properly accept the operator's input.

1) A variable must be defined. The operator input variable in the example (INNUM) was defined as being an integer in the statement

```
INTEGER INNUM, QUITNO, COUNT
```

Actually, three variables are simultaneously defined in this statement, but only INNUM is relevant to this discussion.

2) The INPUT statement itself

```
INPUT //,'ENTER AN INTEGER VALUE',INNUM,//
```

uses the keyword INPUT to indicate that an entry from the keyboard is coming, gives an operator prompt, "ENTER AN INTEGER VALUE" (remember that someone besides the programmer may run the program), then identifies (for the program) the name of the variable that will be entered, followed by a carriage return. Program execution will be halted until the operator has entered an integer, and followed it with a carriage return.

The PRINT statement provides output to the operator's terminal. In our example, the statement is:

```
PRINT //,[10],%INAME,'YOU HAVE RECEIVED',COUNT,'RESULTS',//
```

On the terminal will appear:

```
(OPERATOR'S NAME) YOU HAVE RECEIVED (VALUE OF COUNT) RESULTS
```

The use of imbedded variables in the text to be printed allows the operator to have up-to-date information in an understandable context, in the same manner that imbedding text in the INPUT statement allowed prompting for the correct inputs.

II.7 The STOP keyword

This STRUBAL+ keyword causes code to be compiled that forces a return to the monitor. A program may have many STOP statements in it. STOP returns to warm start.

II.8 The END keyword

END stops the compilation process. There can be only one END in a program. END is a signal to the compiler and assembler; it does not generate any code, nor does it stop the processor in the manner of a STOP keyword. If STRUBAL+ reaches the end of its source file before finding an END statement, it will assume that the file is a function or macro library. The user will be prompted for a new source-file name. A library is simply a STRUBAL+ source file without an END statement. Any number of libraries may be used; libraries should contain ONLY function or macro definitions.

Part III

STRUBAL+ AND PROGRAM DEVELOPMENT

STRUBAL+ supports a set of functions, both unary and binary, for performing many arithmetic, logical comparison and other operations. Any of these functions may be used in an expression. They can be combined, nested and grouped as desired.

III.1 STRUBAL+ variables.

The earlier discussion on variables (II.2) pointed out that a variable is a symbolic name given to a quantity that is expected to change during program execution. In use, it is important to realize that there is a difference between creating variables and assigning value to a variable. The creation of a variable is a clear-cut operation performed once in a program; the assignment of a value can be accomplished through several mechanisms.

The most common methods of assigning value are either alloting a specific value: as

```
INNUM=0
```

from our sample program (strictly speaking, this statement would be written LET INNUM=0; the absence of a keyword causes the LET to be assumed), or an indirect assignment such as:

```
(LET) OUTNUM=INNUM/12
```

A curious thing about assignments is that it is possible, and even common for them to assume a form that is algebraically invalid, such as X=X+1. This is because the statement actually reads let X equal its former value, plus one.

III.1.1 Arithmetic Expressions

An expression consists of a combination of variable names, constants, and functions. Array elements are legal wherever a variable name is legal, except that the FOR/NEXT index cannot be an array element. There are no function precedence rules in STRUBAL+. All expressions are evaluated from left to right, unless parentheses are used to modify the order of evaluation. No two functions can appear sequentially, unless the second is a unary operator such as minus. Logical, arithmetic and other functions may be used within the same expression. Keywords (other than function names) may not be used in an expression. Inclusion of a keyword will produce a compilation error.

Typical simple expressions are:

```
OVERTI=(RATE*1.5) * HOURS

PAY=GROSS + BONUS
```

III.2 Program Control

There are several keywords that cause transfer to a branch. Two of the more commonly used are GOTO and GOSUB.

The GOTO statement has three forms:

The simple

The case

The computed.

The simple form consists of an unconditional branch command as in:

```
GOTO LABEL
```

which will transfer the program directly from this point to a statement labeled LABEL.

The case form provides a list of transfer points in parenthesis. The value of an expression in a preceding statement chooses the transfer point to be executed. The last transfer point will be chosen if the value is out of range.

```
ON CASE

  GOTO (L1,L2,L3,...LN)
```

The computed form computes the transfer point. The expression following the @ sign is evaluated, and that value becomes the target address. Control is passed to this address. Since STRUBAL+ integer arithmetic is signed, care must be taken with negative values. Minus one, for example, is really address FFFF (hex), and, if the value computed to minus two, the address would be FFFE (hex).

```
GOTO @$1002
```

GOSUB

This keyword produces a subroutine transfer. The program will transfer to the subroutine label and will continue there until a RETURN is encountered. GOSUB has the same three forms as GOTO.

ON
--

The ON keyword provides the CASE statement form for both GOTO and GOSUB. Its expression is evaluated, and used to structure program entry into a GOSUB or GOTO list. If the expression value does not correspond to an entry in the list, the last list entry is used. The expressions must be of an integer

type.

EXAMPLE:

```
ON I
GOTO (ARNOLD,MARVIN)
```

If I=1, the next statement that will be executed will be the one labeled ARNOLD. If I=2, the program transfers to the statement labeled MARVIN. If I is any other value, the default condition will cause the program to go to the statement labeled MARVIN.

CALL

This keyword calls programs with a parameter list. The parameters can be data names or labels in the STRUBAL+ program, but they cannot be constants. They cannot be array elements, although an array name is legal. The addresses of the parameters are passed to the called procedure (see procedure). The called program then has access to these locations in the calling program. The number of parameters in the call list must match the number of parameters in the procedure list. If a parameter is not needed in a particular call, it may be omitted. The comma must be included, even for a dummy parameter.

The actual syntax of the CALL statement will depend on the routine being called. If, for example, an assembly-language subroutine named GETRND were to be called, and it had been written to read one disk sector and put its contents in a location named RNDBUF, the CALL routine may appear in this manner:

```
     INTEGER RECNUM,DRVNUM
     DSTRING RNDBUF(127)
*
     RECNUM=5
     DRVNUM=1
*
* PUT VALUES INTO INTEGER LOCATIONS
* FOR THE PARAMETERS
*
     CALL GETRND(RECNUM,DRVNUM,RNDBUF,WRERR)
*
```

The parameters required for GETRND are: sector number, drive number, location to be read into, and, the label of the statement to go to in case of a WRITE error. Thus, in the routine above, Sector 5 of drive 1 would be read into RNDBUF. A subroutine written in STRUBAL+ would function in the same manner.

RETURN

Indicates the end of a subroutine, and causes transfer back to the main program. Subroutines may be nested as deeply as desired, limited only by the memory available to the stack.

III.2.1 Loop Control

It is almost impossible to write a computer program that is free of loops. Loops are a convenient and reasonably straightforward method of executing many common algorithms. A series of statements that must be executed on each pass through the loop may be written and placed in the body of a loop statement. A complete loop is treated in the same manner that any other statement is treated. The WHILE/BLOCK is an example of a loop structure that is employed in STRUBAL+.

WHILE BLOCK

The WHILE keyword is used to establish the condition of the loop, i.e. as long as the condition established in the WHILE statement is true, the body will be executed. A STRUBAL+ WHILE statement will take the form:

```
WHILE INNUM .NE. QUITNO
```

(from the sample program I.3)

This will cause the statements following the WHILE statement, and prior to the BLOCK statemnt to be executed as long as INNUM is not equal to QUITNO.

The BLOCK statement indicates the end of the loop. When the WHILE condition evaluates false (zero) the program will continue at the line following the BLOCK. Going back to our sample program, we can evaluate the entire loop.

```
WHILE INNUM .NE. QUITNO
      OUTNUM = INNUM/2
      PRINT//,'RESULT =',[6],OUTNUM,//
      COUNT = COUNT+1
      INPUT //,'ENTER AN INTEGER VALUE',INNUM,//
BLOCK
```

The loop will now operate in this manner: Whenever it is true that INNUM is not equal to QUITNO, INNUM will be divided by 2, and this value will be stored under the name OUTNUM. The console will print 'RESULTS' followed by the value stored in OUTNUM. The value stored in COUNT will be incremented by 1, and the operator will be requested to input another value for INNUM for the next condition test. The sequence, if the WHILE condition proves true, is then repeated until the condition proves false.

FOR/NEXT

The FOR/NEXT loop is dependent on an index variable which is assigned a value and then changed each time through the loop until a predetermined value is reached. The index variable cannot be an array element or floating point.

The FOR statement may contain two or three expressions. The first is the initial value of the index variable, and the second is its final value. The third expression is the amount that the index will change (increment or decrement) each time through the loop. If no value is specified, it will default to 1.

FOR/NEXT loops may be nested up to ten levels deep, and may be mixed with WHILE/BLOCK structures. FOR has two legal syntax forms:

Example I

```
FOR I=1 TO 5 STEP 1
```

Each time through the loop I will increment by 1, until it reaches 5. If the STEP expression is omitted, 1 will be assumed.

Example II

```
FOR I=1,5,1
```

Would produce the same results. The third expression may be omitted if it is to have a value of one (1).

```
FOR I=1,5
```

would produce the same results. The NEXT statement must end the loop. It names the index variable, and should match the FOR statement. It takes the form:

```
NEXT I
```

where I is the index variable.

To initialize an array to zeros:

```
INTEGER ARRAY(10)
FOR I=1,10
LET ARRAY (I)=0
NEXT I
```

III.3 Input/Output Statements

The I/O statement set allows communication between the operator and the program (i.e. make the program interactive) or between the program and a peripheral.

A new sample program will demonstrate how each statement type is used in the communication process.

```
*
        DSTRING LINE (80),IFNAM(30),OFNAM(30)
        DSTRING IFCB(166),OFCB(166)
*
* Allocates memory for the strings
*
        INPUT 'INPUT FILE?',%IFNAM,/
*
* Lets the operator assign the file name
*
        STRING IFNAM=IFNAM,';I'
*
        INPUT 'OUTPUT FILE NAME?',%OFNAM,/
        STRING OFNAM=OFNAM,';O'

*
```

```
* Initialize disk-system I/O points
*
        CALL INITIO
*
* THE INITIALIZATION ENTRY POINT MUST BE ESTABLISHED
* BEFORE ANY OPEN STATEMENT, BUT ONLY ONCE IN ANY
* GIVEN PROGRAM!
*
       OPEN (IFCB) IFNAM
       IF .ERR THEN INERR
*
       OPEN (OFCB) OFNAM
       IF .ERR THEN OTERR
*
LOOP   READ (IFCB) %LINE
*
       PRINT [80],%LINE,/
       WRITE (OFCB) [80],%LINE,/
       IF .NOT. .EOF THEN LOOP
*
       CLOSE (IFCB)
       CLOSE (OFCB)
*
       STOP
*
INERR  PRINT 'INPUT OPEN ERROR',/
       STOP
*
OTERR  PRINT 'OUTPUT OPEN ERROR',/
*
       STOP
       END
```

What this program does is open two files labeling one an input file and one an output file, read the input file into a variable called LINE, print the file contents for the operator to see, write the contents into the output file and close the file. An analysis of each statement type used in this program will make the sequence clear.

READ (FCB)

This keyword causes input to be read from an external file. The parenthetical expression FCB stands for File Control Block, which is a location where the specific file information is maintained. The names IFCB and OFCB represent a programmer-selected name for the location of the first byte of the control block. Going back to the sample program, the READ statement

```
READ (IFCB) %LINE
```

translates: read from the external file, using the FCB designated IFCB, the string which is defined by the name LINE.

A slash (/) in the READ list causes the file to be read until a linefeed (filemark) is found. All data up to the filemark is skipped over. These logical record filemarks are generated by using the slash in the WRITE statement which writes the file.

INPUT

An INPUT statement is used to handle input from the system console. It is similar to the READ statement, however, the INPUT statement receives its inputs from the system console rather than from the disk file. Thus, the input statement in the sample program

```
INPUT 'INPUT FILE?',%IFNAM,/
```

This statement allows the operator to designate the disk file that is to be used as an input. The percent sign before the variable IFNAM is used because the name is a string. The slash (/) produces a linefeed. When an INPUT statement is used, operator prompts may be included in the single statement as in the example, for clarity.

Two additional features of the INPUT statement are the use of Control U to produce a line delete and Control C to produce a return to monitor. If a bad input is made or a line delete is used a new prompt (>) will indicate when a new entry can be made. Backspace may be used to delete a single character just typed.

PRINT

For straightforward presentations of messages to the console, the PRINT statement provides a simple mechanism. There are two basic types of PRINT statements. The first is used for printing whatever is stored under a variable name, and the second for notifying the operator of certain conditions, such as in the case of an error message.

In the first type, the keyword is followed by a format enclosed in brackets, defining the number of characters that will be printed.

```
PRINT [80],%LINE,/
```

from our example will print -- as a character string --whatever is stored under the name LINE, provided it is less than 80 characters long. If it is longer than 80 characters, LINE will be truncated; if it is shorter than 80 characters, blanks will be added to make it 80 characters.

If a simple message is to be printed, it is handled in the same manner as a prompt: the message is enclosed in quotes (') and the format may be omitted. An example of the second type of statement is:

```
PRINT 'INPUT OPEN ERROR',/
```

The two types of print statements may be mixed in one statement line. For example:

```
PRINT //,'RESULT-',[6],OUTNUM,//
```

prints both a variable (OUTNUM) and a quoted string.

Values are printed subject to the most recent format specification. If a value cannot be printed in the specified format, it will be printed in a 14-digit exponential format.

WRITE (FCB)

This keyword will cause data to be output to external files, provided that the file designated has been previously opened. A list of arguments following the keyword describes the desired outputs. List elements should be separated by commas. Expressions may be in the list, and their values will be output. Format specifications may be included to set the number of places to the left and right of the decimal. This functions for writing only, and no data precision is lost in reformatting. When the field is defined by the value being written, leading blanks and trailing zeros are used for padding out the field. The field width for strings is also set by the format specification.

1. A slash (/) in the argument causes a linefeed to be generated, terminating a logical record in the file.

2. A greater than sign (>) followed by an expression causes a number of spaces equal to the value of the expression to be output. The expression must be of integer type.

3. A quoted string is output exactly as written. Two quotes in a row cause one quote to be output.

4. An expression in the argument list is evaluated, and its value output. The format of the value output conversion is set by the most recently executed format specification (which is an integer with six figures if no format has been executed). If the value will not fit in the specified format, it will be output in a 24-character field with 14 significant figures and a signed exponent.

5. A string name preceded by a percent sign (%) causes that string to be output. The number of characters output will be equal to the field width (Integer Part Size) format specification currently in use. Blanks will be appended to the right of the string if necessary, to fill the field. The string will be truncated if it is too long.

6. A left bracket ([) indicates a format specification. The format consists of two integer expressions separated by a comma and enclosed in brackets. The comma and second expression are optional. If omitted, STRUBAL+ assumes a value of zero for the second expression. The value of the first expression sets the number of digits to the left of the decimal point. A sign (plus is assumed) is output, and the output is padded with blanks as necessary to fill the format. Leading zeros are blanked. The second expression specifies the number of decimal places to be used. Trailing zeros are used as required. A trailing blank is ALWAYS appended. The legal range for the first expression is 1 to 128. A bad value will default to 6. The legal range for the second expression is 0 to 13, with a default to 0. In our example

```
WRITE (OFCB) [80],%LINE,/
```

will write up to 80 characters from LINE to the output file at OFCB.

OPEN (FCB)

This is the keyword that sets up the initialization of I/O files, and it must be executed before any other file commands (READ,WRITE, REWIND) are used.

```
DSTRING IFCB(166),OFCB(166)
```

provides two locations to store the names of the input and output files.

The sizes of the file control blocks (166 in the example) depend on the system being used. For example, Smoke Signal DOS 68 uses 166; SWTPC Miniflex uses 192; ICOM FDOS II uses 51.

```
INPUT 'INPUT FILE?',%IFNAM,/
```

takes the operator's choice of name for the input file and stores it under the variable name IFNAM. The choices available for file names may, however, be limited by the operating system. Consult the operating system manual for your system.

```
STRING IFNAM=IFNAM,';I'
```

will take the name chosen by the operator and append a semicolon, I to it. If, for example, the operator had named the input file "ARNOLD," the variable name IFNAM would now have been assigned a value "ARNOLD;I." This concatenation identifies the file named "ARNOLD" as an input file. Concatenating a ;O to the file name (ARNOLD;O) would designate the file as an output file.

If the programmer had wished to name the file within the context of the program, rather than letting the operator name it, he would have left out the INPUT statement, and the STRING statement would read something like:

```
STRING IFNAM='ARNOLD;I'
```

Alternatively, the file could be opened by:

```
OPEN (IFCB) 'ARNOLD;I'
```

If not performed earlier in the program, an initialization entry point in the runtime package must be called prior to any OPEN. This is accomplished by:

```
CALL INITIO
```

And finally,

```
OPEN (IFCB) IFNAM
```

opens the file according to the parameters established.

CLOSE (FCB)

Once the file is no longer needed, or in any case at the end of the program, the file must be closed. The process is simply:

```
CLOSE (IFCB)
```

REWIND (FCB)

This keyword causes an input I/O file to be rewound to its starting position. The file must have been previously opened with the same FCB.

```
OPEN (FCB) 'FNAME;I'
READ (FCB) DATA
REWIND (FCB)
```

III.4 One and two dimensional arrays

STRUBAL+ data definition statements generate one or two dimensional arrays. Each array element is either a 16-bit integer value (which could contain two characters as a part of a string), a 9-byte floating point value (which could contain nine characters as part of a string) or a string. An array can have up to 32767 elements. Each array index starts at 1 with the element values stored in sequential locations. The first index (of a two-dimensional array) changes most rapidly. If an array is dimensioned (3,3) then the elements would be stored in the following order: (1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (1,3), (2,3), (3,3).

An example of an array might be:

```
INTEGER ARRAY NAME (20)
```

One singly-subscripted variable with twenty elements; each value being a 16-bit integer. If the values were character strings whose lengths were limited to 10:

```
DSTRING ARRAY NAME (10,20)
```

III.5 STRINGS

Strings are employed for character manipulation. The keyword STRING is similar to LET, in that it assigns one string or value to another. STRUBAL+ strings are stored in one-character-per-byte with a terminating character appended (04 Hex). A string may be stored at any location, provided sufficient memory has been reserved. STRING has four forms:

1. A quoted character string is assigned to a string variable. The quote may be imbedded in the string by the use of two quotes in a row.

```
STRING A='THIS IS A STRING'

STRING B(1,2)='THIS IS AN IMBEDDED QUOTE'''
```

2. This form assigns one string variable to another. The string lengths need not be equal, however, the destination string length must be greater than or equal to the length of the source string.

```
STRING C=A
```

3. STRING may also logically compare two strings. The strings may be formed of quoted characters (like form 1) or a variable name. Any comparison operator is allowed. The second keyword, IF, indicates this form of STRING. The result of the comparison can be used in any of the IF forms.

```
STRING IF Y.EQ. 'YES' THEN GOOD
```

will compare the string Y to the word YES. If they are the same, the program will jump to the statement labeled GOOD.

4. The last form of STRING concatenates several strings into a single string. The strings are linked with commas and will be joined in order of occurance. Thus,

```
STRING S2='I AM'
STRING S3='CONCATENATION'
STRING S1=S2,' TESTING ',S3

S1 becomes:
I AM TESTING CONCATENATION
```

At some point in the program one may have no idea of how many characters are stored in a string, or may wish to somehow use the number of characters in the program. This determination is easily accomplished with the keyword LENGTH.

```
STRING A='ABC'
LENGTH I=A
```

will set the variable I equal to three, which is the number of characters in the string A. Of course, before the first statement in this little routine, the programmer would have had to set up storage locations for the variables. For the integer I, the familiar

```
INTEGER I
```

would do the trick. For the string a similar statement, DSTRING is used.

```
DSTRING A(10)
```

would have allocated enough room for 10 characters; more than enough for the three that were used.

To convert a string of characters into a value employ the keyword VALUE. The data type of the result variable determines the type of conversion attempted. If the string is not in the proper form for the value conversion attempted (floating or integer) the program will give an error messsage. Using some of the keywords described we can proceed with an example:

```
     DSTRING CHARS(10)
     INTEGER I
     DEFINE A
     STRING CHARS='213'
     VALUE I=CHARS
*
* Stores 213 as an integer value in I
*
     STRING CHARS='3.14159'
     VALUE A=CHARS
```

```
*
* Puts 3.14159 as a floating-point in A
```

But suppose only part of a string was desired? Simple, in STRUBAL+. If we add to the routine above,

```
        DSTRING S2(10)
        STRING CHARS='ABCDEFG'
        XTRACT S2=2,CHARS
*
* S2='AB'
*
```

we easily extract A and B from the string. Another simple manuever allows us to start from a position other than the first position.

```
        XTRACT S2=3,CHARS(2)
*
* S2='BCD'
*
```

CONVERT

Either integer or floating-point values or expressions can be converted into character-string form. The string produced in this manner will be identical to that produced if the value or expression were printed. The DIGITS statement is used to define the conversion format.

For Example:

```
     DSTRING CHARS(10),MORE(10)
     INTEGER IQ
     IQ=120
*
* Define conversion required with DIGITS format statement
*
     DIGITS [2,2]
     CONVERT CHARS=3.14159
*
* This assigns ' 3.14 ' to CHARS
*
     CONVERT MORE=1+5/IQ
*
* Will make MORE ' 1.00 '
*
     STOP
     END
```

III.6 STRUBAL+ and Subroutines

COMMON

Occasionally it is necessary to place certain variables in a location that will make them available for use in more than one program. This is referred to as placing the variables in COMMON, and is accomplished by the CSECT keyword. The location of COMMON may be chosen at the time that the runtime packages and program are loaded.

To place variables in common, the programmer first declares that this is what he wishes to do.

```
CSECT
```

Then the variables to be placed in common are listed:

```
INTEGER I,K(3)
```

for example. The COMMON section is then terminated by:

```
CEND
```

DUMMY

Sometimes it may be beneficial to describe the layout of data in COMMON, a storage area, or main memory without actually reserving any space. This can be done with a DUMMY section. Data structures and initial values may both be defined in a dummy section. The example begins with:

```
DSTRING RNDBUF (28)
```

which will actually reserve space.

```
DSECT
```

now identifies that the statements to follow will not actually be stored.

```
DSTRING RNAME(10),RADDR(10),RNUM(5)
```

is in the DUMMY section.

```
DEND
```

ends the DUMMY section.

Now we can do useful work with variables that aren't really there! If we say:

```
RNAME     EQU     RNDBUF
RADDR     EQU     RNDBUF+11
RNUM      EQU     RNDBUF+23
```

Then

```
PRINT %RADDR,//
```

Printing RADDR is the same as printing positions 11 through 20 of RNDBUF (RNDBUF+11). Just as easily we could have printed RNUM and seen the last 5 characters of the string. An important advantage of the renaming of the divisions of RNDBUF is the improved readability of the program. In a situation where initial values must be set, STRUBAL+ takes advantage of the assembly OPCODES FDB, FFP, FCC and FCB. The variables in the DUMMY section are assigned the values stated in the PSEUDO-OP statements. FDB assigns integer values, FFP floating-point, FCC assigns character and FCB assigns the string terminator value.

EXAMPLE

```
          DSECT
          INTEGER I,J
          DEFINE X
          DSTRING NAME(10)
          DEND
*
I         FDB 1
J         FDB 50
X         FFP 10.55
NAME      FCC 'HEMENWAY'
          FCB 4
```

The dummy variable X in the example will now have an initial value of 10.55, NAME will have an intial value of HEMENWAY, etc.

PROCEDURE

This keyword is used to begin a called-procedure. It defines the name of the procedure and the passed parameter address names. The parameters must be simple names, not constants or array elements. The number of parameters must agree with the CALL list. The parameter names are local to the procedure.

```
*
* CALLING PROGRAM
*
     INTEGER I,J,K
     INPUT /,'ENTER FIRST NUMBER',I
     INPUT /,'ENTER SECOND NUMBER',J
     CALL SUM(I,J,K)
* PROCEDURE 'SUM' ADDS I TO J AND RETURNS SUM IN K
     PRINT /,'SUM OF NUMBERS=',K
     STOP
*
*
* PROCEDURE TO ADD TWO INTEGERS
*
     PROCEDURE SUM(I1,I2,I3)
     INTEGER I1,I2,I3
     I3=I1+I2
     RETURN
*
     END
```

III.7 Special Keywords

BYTE

This keyword stores byte quantities. The integer expression to the right of the equals sign is evaluated, and its least-significant half is stored into the left argument address. No checking is done for byte overflow.

```
     DSTRING A(20)
     BYTE A(2)=3+1
```

will put a value of 4 in the second byte of A.

DELAY

Program execution can be delayed for specific time intervals by the delay statement. The expression must be of integer type, and is interpreted in milliseconds.

DELAY 25 will delay execution 25 msec.

DELAY I*J will delay execution I*J msec.

NOTE: The DELAY argument is unsigned and can go as large as 65.536 seconds.

GETAD

GETAD functions like LET, except that it forms the address of the name, rather than its value. This address can be used by imbedded assembly code, or for address calculations such as those used in constructing and accessing linked lists. Any STRUBAL+ name may appear in a GETAD statement.

EXAMPLE

```
     *
          INTEGER I(10),J(10),IADD,JADD
     *
     * MOVE CONTENTS OF J INTO I
     * USING CRUTCH CODING FOR SPEED
     *
          GETAD IADD=I
          GETAD JADD=J
          FOR K=IADD,IADD+20
     *
          LDX JADD      POINT TO J
          LDA A 0,X     GET BYTE FROM J
          INX
          STX JADD      INCREMENT JADD
          LDX K         POINT TO I
          STA A 0,X     STORE BYTE IN I
     *
          NEXT K
```

POKE

POKE stores a byte in a specific address. The fist expression after the keyword is evaluated, and its low-order byte is stored in the address given by the value of the second expression. All expressions must be of Integer type.

EXAMPLE:

```
     DSTRING BUFFER (10)
     I=3
     GETAD K=BUFFER
     POKE I,K
*
* Puts value 3 into the first byte of BUFFER
*
     POKE $04,K+1
*
```

```
* puts value 04 (hex) into the second byte of BUFFER
*
```

POP

To pop integers, decimal and strings from the stack, use the POP keyword.

```
POP CHRS     a string
POP INNUM    an integer
POP RCODE    a decimal
```

The stack is an area of memory that is set aside for special use. The computer maintains a special register, the stack pointer, (referred to as the S register) to manipulate the stack area. The stack area is used as a last-in-first-out (LIFO) structure. The stack pointer addresses the first open space in the stack, and, as data is pushed onto the stack, the stack pointer moves up.

As data is popped from the stack, the pointer moves down. The stack pointer may be set and read with the assembler LDS (load stack pointer) and STS (store stack pointer) instructions. The computer uses the stack for subroutine calls and interrupt handling. The programmer must be careful to load the stack pointer properly. It should start at the largest address assigned to the stack. Multiple stacks can exist, but there is only one hardware stack pointer.

PUSH

The reverse of POP, PUSH pushes the data items onto the stack.

III.8 ERROR MESSAGES

Errors found during the compile and assembly passes of STRUBAL+ are flagged with error messages. These messages are printed regardless of the output options chosen. Each contains the program line number in which the error occurred, a 3-digit code which describes the type of error and that portion of the line correctly processed before the error was found.

All assembly-pass errors have codes in the range 200 to 299, and the numbers are identical with those used by the Motorola assembler.

The compiler codes are in the range 300 to 399, and are defined below. It is quite possible for one line to produce more than one error message, and also posible for an error in one line to appear on a subsequent line. Assembler errors may also appear during compilation.

ASSEMBLY-PASS ERRORS

201.....NAM syntax error. Multiple NAM's in one program.

202.....OPCODE or label error. Bad character in label or OPCODE.

204.....Assembler syntax error.

205.....Bad statement label. Label too long.

206.....Redefined symbol. Name already in use.

207.....Undefined OPCODE. OPCODE not assembler mnemonic or recognized pseudo-operation.

208.....Relative branch error. Branch address out of range.

210.....Byte overflow. Value too big to fit in a byte.

211.....Undefined symbol. Does not appear in a label.

216.....Unrecognized pseudo-operation

220.....Phasing error. Pass 2 location counter does not equal pass 1 value.

221.....Symbol table overflow. Too many symbols.

223.....This pseudo-op should not have a label.

226.....Macro/Function definition syntax error.

227.....Label on MEND/FEND

228.....Macro/Function table overflow.

230.....Macro/Function expansion overflow.

251.....Macro/Function Nesting error.

254.....IFC stack overflow/underflow.

COMPILER ERRORS

300.....Bad syntax in an expression.

301.....Expression value stack overflow. Too many variables in an expression, or expression too complex.

302.....Assumed LET syntax because line has no keyword, but syntax not correct. Does not follow LET syntax.

303.....Dimension syntax error. Array has more than two dimensions, too many elements in an array, etc.

304.....Array without index. Attempt to use an array name without indexing.

305.....Blocks nested too deeply. A maximum of 10 nested blocks are allowed.

306.....Terminated blocks. A block was begun but no terminator was found.

307.....NEXT without corresponding FOR statement.

308.....IF syntax incorrect.

309.....GOTO syntax error.

310.....I/O syntax error (READ,WRITE,PRINT,INPUT)

311.....BLOCK without corresponding WHILE statement.

312.....String statement syntax error.

313.....GETAD syntax error.

333.....Type mismatch. Expression is wrong type (Integer/Floating)

334.....File-Control Block syntax error.

335.....Procedure statement syntax error.

RUNTIME ERRORS

OVERFLOW.....Results of integer arithmetic exceed 16 bits.

BAD INPUT.....Erroneous input characters or format.

TRUNCATION.....Input too long.

DIVIDE BY ZERO..Attempt to divide by zero.

ILLEGAL ARG.....Argument to runtime routine out of legal range.

NEGATIVE EXP....Exponent in integer expression less than zero.

INTEGER-PART SIZE.....First part of Format out of range.

DECIMAL-PLACES..Second part of Format out of range.

STRING LENGTH...String too long.

PARAMETER LIST MISMATCH.....Call parameter list does not have the same number of parameters as the procedure list.

PART IV

STATEMENT TYPES

ASSEMBLY LANGUAGE

STRUBAL+ allows assembly language to be imbedded anywhere in a program. Such statements will be passed intact through the compilation and will be assembled. Standard assembly-language syntax should be followed. All assembler constructs are allowed. Imbedded assembler lines may access labels and other names in the STRUBAL+ program. Labels on imbedded assembler lines may be referenced by external STRUBAL+ code. STRUBAL+ does not save registers, nor does it require any registers to be saved by imbedded assembly language statements.

Besides the entire set of Motorola opcodes, STRUBAL+ supplies a set of pseudo-ops. The RMB (Reserve Memory Bytes) allocates space. The bytes are initialized to zero. The FCB (Form Constant Byte) initializes a location. The FDB (Form Double Byte) initializes a two byte area. The FCC (Form Constant Characters) initializes a string of ASCII characters. The PAG (PAGE) formats the listing. Two pseudo-ops, ENT (ENTRY) and EXT (EXTERNAL) allow linkages between separately compiled program sections. EXT forms a two byte location which will be linked by name to that value defined by an ENT in some other program. The CMN (COMMON) pseudo-op, assigns common storage for interprogram data passing. The FFP (Form Floating Point) initializes a floating point constant (9 bytes). The IFC (if for conditional assembly) and NIFC (end if for

conditional assembly) pseudo-ops provide for conditional assembly/compilati If the operand of the IFC pseudo-op is nonzero, the code (STRUBAL+ assembler) between the IFC and following NIFC generates code. If the l operand is zero, the code is skipped. The SET pseudo-op allows redefinition a value repeatedly. The MEND and FEND pseudo-ops terminate macro and functic definitions respectively. The MAC (MACRO) pseudo-op begins a macro definition

EXAMPLE:

```
        INPUT C
    * INLINE ASSEMBLER CODE TO INCREMENT THE VALUE OF C
    *
        LDX C
        INX
        STX C
    *
    * REVERT TO STRUBAL+
    *
        PRINT C
```

BLOCK

This STRUBAL+ keyword causes code to be generated to terminate a WHILE-BLOCK. The block will be repeatedly executed until the expression in the WHILE initiating the BLOCK evaluates FALSE (zero). The program then continues at the line following the BLOCK.

EXAMPLES:

```
        A=0
        B=10
        WHILE A .NE. B
        PRINT A,/
        A=A+1
        BLOCK
  * THIS PRINTS 10 VALUES OF 'A'

      DSTRING DATA(80),FNAME(20),FIOB(166)
      DIGITS [80]
  * DEFINE DATA SPACE
  *
      OPEN (FIOB) FNAME
  * OPEN INPUT DATA FILE
  *
      READ (FIOB) %DATA
  * READ FIRST LINE OF FILE
  *
  * LOOP THROUGH HERE UNTIL END-FILE FOUND
  *
      WHILE .NOT. .EOF
      PRINT %DATA,/
      READ (FIOB) %DATA
      BLOCK
  * THIS PRINTS A FILE 'FNAME'
  * THE FILE HAS 80 CHARACTER LINES
```

BYTE

This STRUBAL+ keyword generates code for storage of byte quantities. The integer expression to the right of the 'equals' sign is evaluated and its least-significant half is stored into the left argument address. No checking is done for byte overflow.

EXAMPLES:

```
        DSTRING A(20)
        BYTE A(2)= 3 + 1
   *
   * PUTS VALUE 4 IN THE SECOND BYTE OF 'A'
   *
        BYTE A(5)= .FIX(3.14159)
   *
   * PUTS VALUE 3 IN THE FIFTH BYTE OF 'A'
   *
        BYTE A(6)=$FFF1
   *
   * PUTS VALUE F1 (HEX) IN THE SIXTH BYTE OF 'A'
   *
```

CALL

This STRUBAL+ keyword calls separately compiled programs with a parameter list. The parameters may be data names or labels in the STRUBAL+ program. They cannot be CONSTANTS!! They cannot be array elements, although an array name is legal. The addresses of the parameters are passed to the called procedure. (see PROCEDURE) The called procedure then has access to these locations in the calling program. The number of parameters in the CALL list must match the number of parameters in the PROCEDURE list. If a parameter is not needed in a particular call, it may be omitted. The comma must be included, even for a dummy parameter.

EXAMPLE:

```
        CALL MINE(P1,P2,P3,P4)
        CALL MINE(P1,,P3,P4)
        CALL MINE(,P2,P3,)
```

EXAMPLE:

```
        INTEGER RECNUM,DRVNUM
        DSTRING RNDBUF(127)
*
        .
        .
        .
        CALL GETRND(RECNUM,DRVNUM,RNDBUF,WRERR)
        .
        .
        .
WRERR   PRINT 'ERROR DURING READ',//
        .
        .
        .
        STOP
```

END

NOTE: GETRND is an assembly language subroutine that reads one disk sector and puts its contents in the RNDBUF location. WRERR is the address that GETRDN will return to if an error occurs during the read operation.

CEND

This STRUBAL+ keyword terminates a COMMON section defined by a CSECT.

CLOSE(FCBNAM)

This STRUBAL+ keyword terminates file I/O by closing the file specified by "FCBNAM". A new file may now be OPENed using FCBNAM.

COMMENTS

Comments may be inserted throughout a STRUBAL+ program for increased readability. They will appear in the compilation and assembly language listings, but no code will be generated for them. A STRUBAL+ comment line must have an asterisk (*) in the first column. The remainder of the line may contain any characters. The line will be printed in the listings exactly as it appears in the comment line. Further... imbedded assembler statements may have comments appended to the line as is usual for the assembler.

EXAMPLE:

```
*
* COMMENT LINES
* AIDJGN%SGGAG%%$#"!'%&'(`0)
*
```

EXAMPLE:

```
* COMMENT ON ASSEMBLER LINES
      LDA A VALUE   GET VALUE (COMMENT)
```

CONVERT

This STRUBAL+ keyword causes conversion of a numeric expression into character-string form. Either integer or floating-point values or expressions may be converted. The string produced by CONVERT will be identical to that produced if the value or expression were printed. The present format specification will be used to perform the conversion.

EXAMPLE:

```
      DSTRING CHARS(10),MORE(10)
      INTEGER IQ
      IQ=120
      DIGITS [2,2]
*
* THIS SETS THE FORMAT FOR CONVERSION
*
      CONVERT CHARS=3.14159
*
```

```
* THIS MAKES CHARS='  3.14 '
*
    CONVERT MORE=1+5/IQ
*
* THIS MAKES MORE='  1.00 '
*
    DIGITS [4]
*
* SET NEW FORMAT
*
    CONVERT CHARS=3.14159
*
* THIS MAKES CHARS='    3 '
*
```

CSECT

This STRUBAL+ keyword begins a COMMON section; all variables defined in a COMMON section are placed in COMMON for use by other programs. The location of the common storage may be chosen at the time that the program and runtime packages are loaded. The COMMON section must be terminated by a CEND!!

EXAMPLE:

```
CSECT
INTEGER I,K,M(3)
CEND
```

DEFINE

This STRUBAL+ keyword sets up storage for floating point variables. All floating variables must be DEFINED, they may be dimensioned if desired.

EXAMPLE:

```
  DEFINE A,B,C, VECTOR(10),MATRIX(3,3)
* THIS MAKES 'A','B',AND 'C' SIMPLE FLOATING-POINT VALUES
* THIS MAKES 'VECTOR' AND 'MATRIX' FLOATING-POINT ARRAYS
```

DELAY

This STRUBAL+ keyword causes code to be generated which will make the program wait for a number of milliseconds equal to the expression value of the DELAY statement, the expression must be of integer type.

EXAMPLE:

```
      DELAY 25
*
* WAITS 25 msec
*
      DELAY I*J/2
*
* WAITS I*J/2 msec
```

NOTE: For DELAY, all numbers are treated as unsigned quantities. Delays up to 65.535 seconds are possible.

DEND

This STRUBAL+ keyword terminates a DUMMY section defined by a DSECT.

DIGITS

This keyword is used to set the format specification for subsequent statements such as CONVERT or WRITE. The format specification consists of one or two integer expressions, enclosed in brackets. The first expression sets the field width (integer part); the second sets the number of decimal places. The field width can range from 1 to 128 characters; the decimal place count from 0 to 13.

EXAMPLES:

```
      DIGITS [2,2]
*
* 2 characters left of the decimal point, 2 decimal places
*
      DIGITS [5]
*
* 5 character field width - no decimal places
* note that the second expression may be omitted
*
```

DIGSET

Used to set the precision for floating-point calculations, the precision ranges from four figures (the minimum precision) to 14 figures. The precision is always a multiple of two, since the compiler utilizes BCD-encoded floating point. If a bad value is given in a DIGSET statement the precision defaults to 14. DIGSET does not change the amount of storage used by floating-point numbers; its purpose is to tradeoff precision for execution speed. It also does not change the format of numbers -- it is quite possible to output more figures than a calculation has precision.

EXAMPLES:

```
      DIGSET 10
*
* 10-digit precision in floating point
*
      DIGSET 7
*
* 6-digit precision; the odd value will be
* rounded down if it is within the range
* of DIGSET
*
      DIGSET 22
*
* 14-digit (default)
*
      DIGSET 3
*
* 14-digit (default)
*
```

DSECT

This STRUBAL+ keyword begins a DUMMY section. A DUMMY section is a control section that describes the layout of data in a storage area, Common or main memory, without actually reserving any storage. The DUMMY section is used with the STRUBAL+ "DATA STRUCTURE" and "INITIAL VALUE" constructs. DATA STRUCTURES and INITIAL VALUES may both be defined simultaneously in a DSECT. The DUMMY section must be terminated with a DEND!!

STRUCTURE - the assembly "EQU" instruction is used to establish addressability for STRUCTURES. The "EQU" instruction equates the variables in the DUMMY section as they actually correspond to the STRUCTURE. The displacement values in the "EQU" statements start at the zero position of the STRUCTURE variable name.

EXAMPLE:

```
     * ACTUAL DATA STORAGE SPACE (NOTE: ROOM FOR 3 END-STRINGS)
             DSTRING RNDBUF(28)
             .
             .
     * DUMMY SECTION STRUCTURE
             DSECT
             DSTRING RNAME(10),RADDR(10),RNUM(5)
             DEND
     *
     *
     * STRUCTURAL DEFINITIONS
     RNAME   EQU     RNDBUF
     RADDR   EQU     RNDBUF+11
     RNUM    EQU     RNDBUF+23
     *
             .
             .
             PRINT %RADDR,//
             .
             .
             END
```

NOTE: Program readability is enhanced by using the DUMMY variable "RADDR" instead of RNDBUF+11. "RADDR" refers to positions 11 through 20 of RNDBUF and has a string attribute with a maximum of 10.

INITIAL VALUE - The assembly pseudo-ops "FDB", "FFP", "FCC" and "FCB" reserve and initialize program variables. The variables in the DUMMY section are assigned the values stated in the pseudo-op statements. "FDB" assigns integer values, "FFP" assigns floating-point values, "FCC" assigns character values, and "FCB" assigns the string terminator value.

EXAMPLE:

```
        DSECT
         INTEGER I,J
         DEFINE X
         DSTRING NAME(10)
        DEND
        .
        .
I       FDB 1
```

```
J       FDB 50
X       FFP 10.55
NAME    FCC 'HEMENWAY'
        FCB 4          (END-STRING)
        .
        .
```

NOTE: Program changes are enhanced by using the DUMMY variable for initial values. Only the assignment values need to be updated and not the occurrences of the value in the program. The DUMMY variable "X" in the example will have an initial value of decimal 10.55; "NAME" will have an initial string value of "HEMENWAY", etc.

DSTRING

This STRUBAL+ keyword sets up storage locations for string variables. The maximum number of characters in the string is defined like a dimension. An array of strings may be formed by using double dimensions--the first dimension is the maximum length of each of the strings, and the second dimension is the number of strings in the array. If no length is specified the string is assumed to be of length one. Space for the end-string character is automatically provided for each string.

EXAMPLE:

```
            DSTRING S1(5),S2,S3(5,10)
     *
     * S1 holds up to 5 characters
     *
     * S2 holds only 1 character
     *
     * S3 holds 10 strings, each of which holds up to 5
     * characters.
     *
```

END

This STRUBAL+ keyword causes the compilation process to stop. There can be only one END in a program. If the symbolic option is selected, the Symbol table will be listed after the END line. The symbol table format is described elsewhere in this manual.

END is a signal to the compiler and assembler; it does not generate any code, nor does it stop the processor. (See STOP) If STRUBAL+ reaches the end of its source file before finding an END statement, the compiler will prompt for a new source-file name.

EXAMPLE:

```
     * PROGRAM LINES.....
     *
            END
```

FEND

This keyword terminates a function definition. In use it is identical to MEND in macro definition. See the section on macros for details.

FOR

This STRUBAL+ keyword causes code to be generated for FOR-NEXT loop structures. This is a loop based upon an index variable which is assigned a value and changed each time through the loop until a final value is reached. The index variable cannot be an array element or floating point. It may be used and/or modified inside the loop.

The FOR statement may contain two or three expressions. The first is the initial value of the index variable. The second is the final value of the index variable. It may not be changed inside the loop. The third expression is the change in the index variable for each loop. It may not be changed in the loop. This third expression is optional; if omitted, the value 1 is used. All expressions must be of integer type.

FOR-NEXT loops may be nested up to 10 levels deep and may be mixed with WHILE-BLOCK structures. FOR has two legal syntax forms, as shown in the examples.

EXAMPLE:

```
FOR I=1 TO 5
 FOR J=2 TO 10 STEP 2
  FOR K=I-1 TO J*3 STEP 1
   FOR L=1,5
    FOR M=2,77,3
     FOR N=0,-11,-1
     NEXT N
    NEXT M
   NEXT L
  NEXT K
 NEXT J
NEXT I
```

FUNCTION

This begins a function definition. It is similar in operation to the MAC (MACRO) pseudo-op; it gives the name of the function and optionally determines whether comments within the function body are to be saved and output during each expansion of the function. Functions are to STRUBAL+ code what macros are to assembly code. All features of STRUBAL+ macros may also be used in functions.

EXAMPLE:

```
        FUNCTION CHAR
*
* this comment is not saved
*
* CHAR converts integer argument 1 into ASCII
* argument 2
*
        GETAD I=&1
        BYTE &2=.PEEK(I+1)
        LDX #&2
        LDA A #4
```

```
        STA A 1,X set end-string character
        FEND
```

EXAMPLE:

```
        FUNCTION ASCII,C
*
* this comment is saved
*
* ASCII converts character arg 1 into
* integer arg 2
*
        GETAD I=&1
        &2=PEEK(I)
        FEND
```

GETAD

This STRUBAL+ keyword works like LET, except that it forms the address of the name rather than its value. This address can be used by imbedded assembly code or for address calculations such as those used in constructing and accessing linked lists. Any STRUBAL+ name (variable, array element, statement label) may appear in a GETAD statement.

EXAMPLE:

```
        INTEGER   ARRAY(22,33)
        ...
        ...

LABEL   ARRAY(I,J)=0
        ...
        ...
        ...
        GETAD A1=LABEL
        GETAD A2=ARRAY(22,1)
        GETAD ARRAY(1,1)=LABEL
```

GOSUB

This STRUBAL+ keyword causes code to be generated for a subroutine transfer. The program will transfer to the subroutine label and will continue there until a RETURN is encountered. GOSUB has three forms:

The simple
The case
The computed

The simple form is an unconditional transfer to the subroutine.

EXAMPLE

```
LABEL   GOSUB SUB1
        GOSUB SUB2
*
* PROGRAM LINES
*
SUB1    .....
        .....
```

```
            .....
            RETURN
*
SUB2        .....
            .....
            .....
            RETURN
```

The case form allows a choice of transfer points based on the value of some expression. A list of transfer names in parenthesis follows the GOSUB. A preceeding ON statement chooses the transfer point. The last transfer name will be used if the value in the preceding ON statement is out of range.

EXAMPLE

```
        ON 1+C/2
          GOSUB (SUB1,SUB2,....SUBn)
```

The computed form computes the transfer address. It has an expression preceeded by an "@" , and the integer value of the expression is the absolute address that is called. Since STRUBAL+ uses signed integer arithmetic, care must be taken with negative values.

EXAMPLE

```
        GOSUB @EXPRESSION       (INTEGER ONLY)
        GOSUB @SUB1
        GOSUB @$1001
```

GOTO

This STRUBAL+ keyword causes code to be generated for a branch. It has three forms:

The Simple
The Case
The Computed

The simple form is an unconditional branch.

EXAMPLE

```
         GOTO LABEL
```

The case form has a list of transfer points in parentheses. The value of an expression in a preceding ON statement chooses the transfer point to be executed. The last transfer point will be chosen if the value is out of range.

EXAMPLE

```
         ON CASE
           GOTO (L1,L2,L3,....Ln)
```

The computed form computes the transfer point. The expression is evaluated and that value becomes the absolute target address. Control is passed to this address. Since STRUBAL+ integer arithmetic is signed, care must be taken with negative values here.

EXAMPLE

```
        GOTO @expression        (INTEGER ONLY)
        GOTO @SUB1
        GOTO @$1002
```

IF

This STRUBAL+ keyword generates code for conditional branches. There are three types of IF statement. Each contains an expression which is evaluated. If the expression is TRUE(nonzero), then one branch is made. If the expression is FALSE(zero), then another branch is made.

The first form of IF is the IF-THEN form. A transfer label follows the THEN. The program transfers to this label if the expression is TRUE. Otherwise the next statement is executed.

EXAMPLE

```
IF A .NE. B THEN LABEL
```

The second form of IF is like the IF-THEN with the addition of ELSE. If the expression is FALSE, then the label following ELSE is transfered to.

EXAMPLE

```
IF A*C-1 .GE. 10 THEN LABEL1 ELSE LABEL2
```

The third form of IF has an executable STRUBAL+ statement following the expression. This statement will be executed if the expression evaluates TRUE. The statement cannot be another IF.

EXAMPLE:

```
IF Q .AND. (.NOT. Q) LET A=A+1
```

NOTE: The keyword LET is not optional here.

INPUT

This keyword causes code to be generated for input from the terminal. Operator prompts may be written; linefeeds and spaces may be output. Strings or values may be input. Error-correcting editing facilities are provided.

1. A slash (/) in the argument list that describes the input line causes a linefeed to be generated.

2. A quoted string in the list outputs the string to the terminal. An imbedded quote may be formed by using double quotes.

3. A variable number of spaces may be output by using a greater than sign (>) followed by an integer-type expression whose value will be the number of spaces.

4. Strings may be input by preceding the string name in the input list with a percent sign (%).

5. Values are input by giving their name in the input list.

6. String or value inputs are signalled to the user with a prompt (>). Characters are then accepted into a buffer. A character delete (backspace) allows characters just typed to be written over.

7. Control-U provides line delete.

8. Control-C quits the program and causes an immediate return to the monitor.

9. If a bad input is made, an error message is output. The user may re-input after receiving a prompt.

EXAMPLE:

```
        INTEGER DAYS
*
*
        DEFINE OPRICE,RES,YEARS
*
        INPUT 'ORIGINAL PRICE?',OPRICE,/
        INPUT 'RESALE PRICE?',RES,/
        INPUT 'YEARS?',YEARS,/
*
        INPUT/,>3,'NO OF DAYS=',DAYS,/
*
        DSTRING ADDRESS(30)
*
        INPUT 'ENTER ADDRESS',%ADDRESS
```

NOTE:
Values may be input in signed decimal form or hexadecimal form or floating-point form with variable syntax. Hexadecimal inputs are indicated by a leading dollar sign ($); four hex digits are allowed. Leading zeroes need not be typed. If there is no leading dollar sign, the input is assumed to be signed decimal. Plus is assumed unless a minus sign precedes the input. Up to five decimal digits may be input, subject to STRUBAL+'s 16-bit representation of the number. Floating-point numbers may have up to 14 significant digits with the decimal point optional; an optional exponent may be added. This base-10 exponent is formed from the letter E, an optional plus or minus sign and one or two exponent digits.

INTEGER

This STRUBAL+ keyword reserves space for integer variables. The variables may be dimensioned if desired. Non-dimensioned integers need not be declared.

EXAMPLE:

```
*
        INTEGER ARRAY(3,3),VECTOR (1000)
*
        INTEGER X
*
* THIS STATEMENT IS OPTIONAL AND SERVES
* ONLY TO DOCUMENT THE PROGRAM.  IF
* UNDECLARED, X WOULD DEFAULT TO
* INTEGER STATUS.
*
```

LENGTH

This STRUBAL+ keyword causes the number of characters that are presently stored in a string to be determined. This integer value is stored into an integer variable. The end-string character is not counted here.

EXAMPLE:

```
        DSTRING CHARS(10),S1(10),S2(10)
        STRING CHARS='ABC'
        STRING S1=CHARS,'DEF'
        STRING S2=S1,'G'
        LENGTH I=CHARS
*
* THIS SETS I=3
*
        LENGTH J=S1
*
* THIS SETS J=6
*
        LENGTH K=S2
*
* THIS SETS K=7
*
```

LET

This STRUBAL+ keyword causes code to be generated which assigns a value to a variable. The value may be any valid STRUBAL+ expression. The variable may be an array element. The keyword LET is optional; if a STRUBAL+ line has no keyword, then LET will be assumed. The value and variable must be of the same type.

EXAMPLE:

```
LABEL  LET C=C+1
       LET D=1-(TEST/C*44)/ARRAY(3,C+11)
       ARRAY(I,J)=5
       L=4/Q
       Z=ARRAY(1,1) .AND. $F000 .OR. (Q .LE. 5)
```

NEXT

This STRUBAL+ keyword terminates a FOR-NEXT block or loop. It names the index-variable of the loop, which should match the corresponding FOR statement. The index-variable must be of integer type.

EXAMPLE:

```
        FOR INDEX=1,5
         FOR JNDEX=2,10
         ....
         ....
         NEXT JNDEX
         .....
         .....
        NEXT INDEX
```

ON

This STRUBAL+ keyword causes code to be generated for use with case-form GOSUB and GOTO statements. Its expression is evaluated, and that entry in the GOSUB, or GOTO list is used. If the expression value does not correspond to an entry in the list, then the last list entry is used. The expression must be of integer type.

EXAMPLE:

```
            ON I
            GOTO (CASE1,CASE2,CASE3)
* IF I<1, THEN WILL GOTO CASE3
* IF I=1, THEN WILL GOTO CASE1
* IF I=2, THEN WILL GOTO CASE2
* IF I=3, THEN WILL GOTO CASE3
* IF I>3, THEN WILL GOTO CASE3
*
            ON J
            GOSUB (SUBR,SUBRS)
* IF J<1, THEN GOSUB SUBRS
* IF J=1, THEN GOSUB SUBR
* IF J=2, THEN GOSUB SUBRS
* IF J>2, THEN GOSUB SUBRS
```

OPEN (FCB)FILNAME

This STRUBAL+ keyword sets up the initialization of I/O files. It must be executed before any other file I/O commands (READ, WRITE, REWIND). FCB is an I/O CONTROL BLOCK and FILNAME is the name of the file to be opened. The details of the FILNAME are specific to the particular operating system in use.

The initialization entry point in the file I/O runtime package must be CALLed prior to any file I/O. It should be called only once...before the first OPEN.

EXAMPLE

```
        INTEGER INFCB(83)
*
        DSTRING INNAM(30)
*
        STRING INNAM='1:TEST.SRC;INPUT'
        CALL INITIO
*
        OPEN (INFCB) INNAM
        IF .ERR THEN INERR
*
        .
        .
        .
INERR   PRINT 'OPEN OPEN ERROR',/
        STOP
        END
```

Note: The qualifier ";INPUT" or ";OUTPUT" must be concatenated to the end of the Filname for opening INPUT or OUTPUT respectively. The qualifier may be abbreviated to ";I" or ";O" if desired.

POKE

This STRUBAL+ keyword causes code to be generated to store a byte into a specified address. The first expression after the keyword is evaluated and its low-order byte is stored in the address given by the value of the second expression. The expressions must be of integer type.

EXAMPLES:

```
        DSTRING BUFFER(10)
        I=3
        GETAD K=BUFFER
        POKE I,K
* PUTS VALUE 3 INTO THE FIRST BYTE OF 'BUFFER'

        POKE $04,K+1
* PUTS VALUE 04 (HEX) INTO SECOND BYTE OF 'BUFFER'
*
        POKE $F1,$1234
* PUTS VALUE F1 (HEX) INTO ABSOLUTE ADDRESS 1234
*
        POKE $F012,K
* PUTS VALUE 12 (HEX) INTO FIRST BYTE OF 'BUFFER'
```

POP

This STRUBAL+ to causes data items to be popped from the stack. It is the reverse of PUSH, INTEGERS, DECIMAL and STRINGS may be PUSH/POPPED from the stack.

The stack is an area of memory which is set aside for special use. The computer maintains a special register, the stack pointer, (referred to as the 'S' register) to manipulate the stack area. The stack area is used as a last-in/first-out structure. The stack pointer addresses the first open space in the stack. As data is pushed onto the stack, the stack pointer moves up. As data is popped from the stack, the stack pointer moves down. The stack pointer may be set and read with the assembler LDS (load stack pointer) and STS (store stack pointer) instructions. The computer uses the stack for subroutine calls and interrupt handling. The programmer must be careful to load the stack pointer properly. It should start at the largest address assigned to the stack. Multiple stacks can exist, but there is only one hardware stack pointer.

EXAMPLE:

```
POP CHRS        (a STRING)
POP INUM        (an INTEGER)
POP RCODE       (a DECIMAL)
```

PRINT

Used to send output to the terminal, a list of arguments following the keyword describe the desired output. The list elements should be separated by commas; string and value output is supported. Linefeeds and variable spacing are provided. Expressions may be in the list; their value will be output. Format specifications may be included to set the number of places to the left and right of the decimal point to be used for output conversion (this function can also be accomplished globally by the DIGITS statement). Leading blanks and trailing zeroes are used to fill in the field; the field width is set by the

integer part size of the format specification.

1. A slash (/) in the argument outputs a linefeed.

2. A greater-than (>) sign followed by an expression causes a number of spaces equal to the value of the expression to be output--the expression must be of the integer type.

3. A quoted string is output exactly as written. Two successive quotes cause one quote to be output.

4. An expression in the argument list is evaluated and its value output. The format of the value output conversion is set by the most recently executed format specification. If no format has been specified, the format defaults to integer and six figures to the left of the decimal. If the value cannot fit into the specified format it will be output in 24-character field with 14-significant figures and a signed exponent.

5. A string name preceded by a percent sign (%) causes that string to be output. The number of characters output will be equal to the field width of the format specification currently in use. The string will be truncated if too long.

6. Brackets ([ ]) indicates a format specification. The format consists of two integer expressions separated by a comma and enclosed in brackets. This functions the same as a DIGITS statement.

EXAMPLE:

```
STRING CHARS='ABCDEF'
PRINT [2],%CHARS
PRINT [4],%CHARS
PRINT [3],'CHARS=',%CHARS
```

OUTPUTS:

```
AB
ABCD
CHARS=ABC
```

PROCEDURE

This STRUBAL+ keyword begins a called-procedure. It defines the name of the procedure and the passed parameter address names. The parameters must be simple names, not constants or array elements. (see CALL) The number of parameters must agree with the CALL list. The parameter names are local to the procedure.

EXAMPLE:

```
* MAIN PROGRAM HERE
      INTEGER I,M(3,3)
      DEFINE R(2)
      DSTRING S(10)
      R(1)=.FLOAT(.RAND(I))
      CALL PR1(R,I,M)
      STOP
      .....
```

```
        .....

* PROCEDURE HERE
        PROCEDURE PR1(RP,IP,MP)
        INTEGER IP,MP(9)
        DEFINE RP(2)
        IF(IP .EQ. MP(1)) RETURN
        .....
        .....
        RETURN
```

NOTE: Procedures should be compiled separately from the program that calls them.

PUSH

This STRUBAL+ keyword pushes data items onto the stack. INTEGERS, STRINGS and DECIMAL numbers may be pushed onto the stack. (see POP for a description of the stack.)

EXAMPLE:

```
PUSH CHARS      (a STRING)
PUSH INUM       (an INTEGER)
PUSH RCODE      (a DECIMAL)
```

READ (FCB)

This STRUBAL+ keyword causes code to be generated for input from an external file. Strings or values may be input. A list of arguments follws the READ keyword. The list elements should be separated with commas. The FCB is the FCB used to open the file. An OPEN must be performed before any reads from the file.

1. A slash (/) in the list causes the file to be read until a record mark (linefeed) is found. The data up to the linefeed is skipped over.

2. Strings may be input by preceding the string name in the input list with a percent sign (%).

3. Values are input by giving their name in the input list.

4. If a bad input is made, an error message is output.

RETURN

This keyword causes code for a subroutine or procedure return to be generated. Subroutines may be nested as deeply as desired, limited only by the memory available to the stack. RETURN terminates a subroutine.

EXAMPLE:

```
SUB     ......
        .....
*
* STRUBAL+ statements
*
        ......
```

```
        RETURN
```

REWIND (FCB)

This STRUBAL+ keyword causes an input I/O file to be rewound to its starting position. The file must have been OPENed first with the same FCB.

EXAMPLE:

```
                OPEN (FCB) 'FNAME;INPUT'
*
                READ (FCB) DATA
                REWIND (FCB)
```

STOP

This STRUBAL+ keyword causes code to be compiled which forces a return to the monitor. A program may have many STOP statements in it. If the file I/O runtime package is included in the load module, STOP will return to the file system rather than the monitor.

EXAMPLE:

```
     * PROGRAM LINES ....
     *
     LABEL    STOP
     *
     * MORE PROGRAM LINES
     *
              STOP
     *
     * STILL MORE PROGRAM LINES
     *
              END
```

STRING

This STRUBAL+ keyword generates code for some of the character manipulation features in the language. Its basic form is like the LET, in that it assigns one string or value to another. STRUBAL+ strings are stored one character-per-byte with a terminating character appended. A string may be stored at any location, but care must be taken to see that sufficient memory has been reserved for the string.

STRING has four forms:

1. This form of STRING assigns a quoted character string to a string variable. A quote may be imbedded in the string by using two quotes in a row.

EXAMPLE:

```
STRING A='THIS IS A STRING'
STRING B(1,2)='THIS IS AN EMBEDDED QUOTE'''
```

2. This form of STRING assigns one string variable to another. The string lengths need not be equal.

EXAMPLE:

```
STRING C=A
```

3. This form of STRING logically compares two strings. The strings may be formed of quoted characters (like FORM 1), or a variable name. Any comparison operator is allowed. The second keyword IF indicates this form of STRING. The result of the comparison may be used in any of the IF forms.

EXAMPLE:

```
STRING IF 'Y' .EQ. YES THEN GOOD ELSE BAD

STRING IF OK .GE. IN LET CHAR=TEST
```

4. This form of STRING concatenates several strings into a single string. The strings may be quoted (like FORM1) or variable names. The strings are linked with commas and will be joined in order of occurrance.

EXAMPLE:

```
STRING S2='I AM '
STRING S3=' CONCATENATION'
STRING S1=S2,'TESTING',S3
* THIS MAKES S1='I AM TESTING CONCATENATION'
```

VALUE

This STRUBAL+ keyword causes conversion of a string of characters into a value. The data type of the result variable determines the type of conversion attempted. If the string is not in the proper form for the value conversion attempted, (floating or integer) the program will give an error message.

EXAMPLE:

```
        DSTRING CHARS(10)
        INTEGER I
        DEFINE A
*
        STRING CHARS='213'
        VALUE I=CHARS
* THIS MAKES I=213
*
        STRING CHARS='3.14159'
        VALUE A=CHARS
*  THIS MAKES A=3.14159
```

WHILE

This STRUBAL+ keyword causes code to be generated for a WHILE-BLOCK structure. This is a set of STRUBAL+ lines which will be repeatedly executed as long as an expression in the WHILE statement evaluates as TRUE (nonzero). The expression is evaluated at the beginning of the block. Statements within the block may change values in the expression and may transfer out of the block if desired. WHILE-BLOCKS MAY be nested up to 10 levels deep.

EXAMPLE:

```
A=4
B=10
 WHILE A*B .GE. 10
    B=B-1
      WHILE A .GE. 1
        A=A-1
      BLOCK
 BLOCK
```

WRITE (FCB)

This STRUBAL+ keyword generates code for outputs to external files. The file must have have opened prior to any writes with the same FCB. A list of arguments following the keyword describes the desired outputs. The list elements should be separated by commas. String and value outputs are supported. Linefeeds and variable spacing are provided. Expressions may be in the list; their value will be output. Format specifications may be included to set the number of places left and right of the decimal point to be used for output conversions. Leading blanks and trailing zeroes are used for fill in the field. The field width for strings is also set by the format specification (integer part size).

1. A slash (/) in the argument causes a linefeed to be generated, which terminates a logical record in the file.

2. A greater-than sign (>) followed by an expression causes a number of spaces equal to the value of the expression to be written. The expression must be of integer type.

3. A quoted string is output to external files exactly as written. Two quotes in a row causes one quote to be output.

4. An expression in the argument list is evaluated and its value written. The format of the value-output conversion is set by the most recently executed format specification. (See 6.) If no format has been specified, the format defaults to integer (no decimal places and six figures left of the decimal point). If the value cannot fit in the specified format, it will be written in a 24-character field with 14 significant figures and a signed exponent.

5. A string name preceded by a percent sign (%) causes that string to be written. The number of characters written will be equal to the field width (integer part size) format specification currently in use. Blanks will be appended to the right of the string if necessary to fill the field. The string will be truncated if it is too long.

6. A left bracket ([) indicates a format specification. The format consists of two integer expressions, separated by a comma and enclosed in brackets. The comma and second expression are optional...if omitted, STRUBAL+ assumes a value of zero for the second format expression. The value of the first expression sets the number of digits left of the decimal point (the integer part). A sign (plus is assumed) is output, and the value is padded with blanks as necessary to fill the format. Leading zeroes are blanked. The second expression in the format specifies the number of decimal places to be used. Trailing zeroes fill out the format if necessary. A trailing blank is always appended. The legal range for the first format expression is 1 to 128. A bad value will default to 6. The legal range for the second format

expression is 0 to 13. A bad value will default to 0.

EXAMPLE:

```
        WRITE 'THIS IS A STRING',%CHARS,//

        WRITE C*A-B,'=',/,.ALPHA
        WRITE [2,2],3.14159,[6],IN(3),/
```

NOTE: If A=3.14159 and I=1234, then the following formats print out as follows (_ indicates a blank):

```
WRITE [6],A,I          _____3____1234_
WRITE [5,1],A,I        _____3.1___1234.0_
WRITE [4,1],A,I        ____3.1__1234.0_
WRITE [4,3],A,I        ____3.141__1234.000_
WRITE [2,2],A,I        __3.14__0.12340000000000_E+04_
```

note that 1234 does not fit into a 2,2 format

XTRACT

This STRUBAL+ keyword causes the extraction of a substring from a given character string. The integer expression right of the "equals" sign is evaluated, and the value is used as the length of the substring. Characters are moved to the result string with an end-string (hex 04) appended. The result name must not be floating point, and the expression must be integer. A comma separates the expression from the string. The first character of a string is "one."

EXAMPLE:

```
            DSTRING S1(10),S2(10)
            STRING S1='ABCDEFG'

            XTRACT S2=2,S1
      * S2= 'AB'
            XTRACT S2=3,S1(2)
      * S2= 'BCD'
            XTRACT S2=10,S1(3)
      * S2= 'CDEFG'
```

PART V

ASSEMBLY LANGUAGE FEATURES

INSTRUCTION FORMAT

A source language statement consists of a label, an operation code, an operand, and comments. The label is used when needed as a reference by other statements. The operation code may be a mnemonic machine operation, an assembly directing pseudo-op, or a Macro call label. An operand may be an expression consisting of an alphanumeric symbol, a number, a special character, or any of these combined by arithmetic operations; or in certain instances there may be no operand at all. The comments may also be left out.

The fields in a source statement are separated by at least one space character (20 hex).

Statement Characteristics

The fields of the source statement appear in the following order:

Label Opcode Operand(s) Comments

Field Delimiters

One or more spaces separate the fields of a statement. An End-of-Statement mark, (carriage return) terminates the entire statement. A single space following an End-of-Statement mark from the previous statement is the null field indicator of the label field.

Character Set

The characters recognized by the Assembler are as follows:

A through Z
0 through 9
* (asterisk)
+ (plus)
- (minus)
/ (slash)
$ (dollar sign)
' (apostrophe)
, (comma)
. (decimal point)
@ (at sign)
# (pounds sign)
& (ampersand)
  (space)
! (exclamation point)

Any other ASCII characters may appear in the comments field. The letters A through Z, and the numbers 0 through 9 may be used in an alphanumeric symbol. In the first position of the label field, an asterisk indicates a comment line; in the operand field it represents the value of the program location counter for the current instruction (if it is in the first position), otherwise it is recognized as the multiplication operator for an expression.

The plus, minus, slash and asterisk are used as operators in arithmetic expressions.

The pounds sign is used to indicate the immediate addressing mode, the dollar to indictate hexadecimal numbers, the apostrophe to indicate ASCII strings, the ampersand to indicate substitutable parameters in Macro definitions, the comma to separate operands and the period and at sign as alpha characters.
The exclamation point serves to signal a compiler-generated label in a function or a macro.

Spaces separate fields of a statement and may also be used to format the output listing.

STATEMENT LENGTH

A statement may be up to seventy two characters long.

Label Field

The label field serves to identify the statement and may be used as a reference by other statements in the program.

The field starts immmediately following an End-of-Statement mark and is terminated by a space. A space in position one indicates that the statement is unlabeled.

Label Symbol

A label is composed of from one to six characters. The first one must be an Alpha character. The remaining characters must be alphanumeric characters. If the label is composed of more than six characters the Assembler truncates the symbol to six characters.

An asterisk in position one indicates that the entire statement is a comment. An asterisk in any position of the label field other than the first position is illegal.

Opcode Field

The operation code defines an operation to be performed by the computer or by the Assembler (opcodes, pseudo-ops or Macro calls). The opcode field follows the label field and is separated from it by at least one space. If there is no label the opcode/pseuop/Macro call may begin anywhere after position one. The Opcode field is terminated by a space immediately following the mnemonic.

Operand Field

The meaning and format of the Operand field is dependent on the type of operation code used in the source statement.

The field follows the Opcode field and is separated from it by at least one space. When Dual operand instructions are used the first operand must be either an "A" or a "B". The single characters "A" or "B" must be preceeded and followed by one or more spaces. The operand field is terminated by a space except when there are no comments; in that case it is terminated by an End-of-Statement mark.

Symbolic Terms

A symbolic term follows the same rules for the formation of labels. A symbol used in the operand field must be a symbol that is defined elsewhere in the program. An asterisk may be used to refer to the value of the location counter at the time the source statement is encountered.

Numeric Terms

A numeric term may be either decimal or hexadecimal. A decimal number is represented by one to five decimal digits within the range 0 to 65535. A hexadecimal number is indicated by one to four hexadecimal digits within the range 0 to FFFF and is preceeded by a dollar sign "$".

Expression Operators

The asterisk, symbols and numbers may be joined by the four arithmetic operators (+ - * /) to form arithmetic expressions. The Assembler evaluates expressions from left to right without regard to precedence or operator hierarchy. A fractional result, if obtained during the evaluation of an expression, is truncated to an integer value. The plus (+) and minus (-) operators may be used as unary operators.

EXAMPLE:

```
LDX -1  equivalent to LDX $FFFF
LDA A #-2 equivalent to LDA A #$FE
```

Macro-call Argument Lists

Macros are passed arguments by placing the arguments in the operand field separated by commas. The actual arguments are substituted as character strings into the positions of the corresponding dummy arguments in the Macro definition. If comments are to be included in the statement a comma must follow the last argument.

Evaluation of Symbols and Expressions

Because of the 2-pass nature of the Assembler only one level of forward referencing is legal in the use of symbols and expressions in the operand field of source statements.

Comment Field

A comment may be included in a source statement as long as it is separated by at least one space from the operand field.

ADDRESSING MODES

Dual Operands

These instructions require two operands in the operand field; first is always either an A or B. The first operand is separated from the second operand by at least one space. The second operand is formed in accordance with the rules for Direct, Extended, Immediate or Indexed addressing.

Accumulator

These instructions reference only one operand in the operand field; this operand is always either the A or B accumulator represented by the single character A or B.

Inherent

These instructions require no operands as the information needed is implied by the instruction itself.

Indexed

These instructions reference the index register X. The operand field of the instruction is evaluated and placed in the second byte of the instruction. When the instruction is executed the contents of this byte are added to the Index register to form the complete address. The format is:

```
NUMBER,X
SYMBOL,X
EXPRESSION,X
```

Where NUMBER, SYMBOL or EXPRESSION evaluates to a number between 0 and 255. If a larger number is generated only the lower eight bits are used.

Immediate

For these instructions the actual value of the operand is placed in the object code itself following the instruction machine code. The Immediate mode of addressing is indicated by preceeding the operand with a pounds sign "#". The format is:

```
#NUMBER
#SYMBOL
#EXPRESSION
#'C
#'CC
```

Relative

These instructions are all branch type instructions The branch address is taken relative to the current contents of the Program Counter. They are two bytes in length. The second byte contains a signed number in two's compliment that specifies the relative branch address. The address of the destination of the branch must be in the range of -126 to +129 relative to the address of the first byte of the branch instruction. The format is:

```
NUMBER
```

```
SYMBOL
EXPRESSION
```

Direct and Extended

For those instructions that allow it, the Assembler will select the Direct mode of addressing if the evaluated operand is in the numerical range of 0 to 255; provided that the evaluated address is not relocatable or common. In these cases the Extended mode will be selected. The Direct mode generates two bytes of machine code; the second byte contains the eight bit address in unsigned binary, the upper 8 bits of the 16-bit address are assumed to be zeroes. If the evaluated operand is greater than 255 then the Assembler generates three bytes of machine code. The second and third bytes contain the sixteen bit unsigned binary address. The format is:

```
NUMBER
SYMBOL
EXPRESSION
```

ADDRESS TYPE FORMATS

ADDRESS TYPE 1

(ADC, ADD, AND, BIT, CMP, LDA, ORA, SBC, SUB)

IMMEDIATE TWO BYTES:

```
CCC A #NUMBER            CCC B #NUMBER
CCC A #SYMBOL            CCC B #SYMBOL
CCC A #EXPRESSION        CCC B #EXPRESSION
CCC A #'C                CCC B #'C
```

DIRECT TWO BYTES, EXTENDED THREE BYTES:

```
CCC A NUMBER             CCC B NUMBER
CCC A SYMBOL             CCC B SYMBOL
CCC A EXPRESSION         CCC B SYMBOL
```

INDEXED TWO BYTES:

```
CCC A NUMBER,X           CCC B NUMBER,X
CCC A SYMBOL,X           CCC B NUMBER,X
CCC A EXPRESSION,X       CCC B EXPRESSION,X
```

ADDRESS TYPE 2 (STA)

EXTENDED THREE BYTES, DIRECT TWO BYTES:

```
STA A NUMBER             STA B NUMBER
STA A SYMBOL             STA B SYMBOL
STA A EXPRESSION         STA B EXPRESSION
```

INDEXED TWO BYTES:

```
STA A NUMBER,X          STA B NUMBER,X
STA A SYMBOL,X          STA B SYMBOL,X
STA A EXPRESSION,X      STA B EXPRESSION,X
```

ADDRESS TYPE 3

(ASL, ASR, CLR, COM, DEC, INC, LSR, NEG, ROL, ROR, TST)

ACCUMULATOR ONE BYTE:

```
CCC A                   CCC B
```

EXTENDED THREE BYTES:

```
CCC NUMBER
CCC SYMBOL
CCC EXPRESSION
```

INDEXED TWO BYTES:

```
CCC NUMBER,X
CCC SYMBOL,X
CCC EXPRESSION,X
```

ADDRESS TYPE 4 (PSH, PUL)

ACCX ONE BYTE:

```
CCC A                   CCC B
```

ADDRESS TYPE 5 (CPX, LDS, LDX)

IMMEDIATE THREE BYTES:

```
CCC #NUMBER
CCC #SYMBOL
CCC #EXPRESSION
CCC #'CC
```

DIRECT TWO BYTES, EXTENDED THREE BYTES:

```
CCC NUMBER
CCC SYMBOL
CCC EXPRESSION
```

INDEXED TWO BYTES:

```
CCC NUMBER,X
CCC SYMBOL,X
CCC EXPRESSION,X
```

ADDRESS TYPE 6 (STX, STS)

DIRECT TWO BYTES, EXTENDED THREE BYTES:

```
CCC NUMBER
CCC SYMBOL
CCC EXPRESSION
```

INDEXED TWO BYTES:

```
          CCC NUMBER,X
          CCC SYMBOL,X
          CCC EXPRESSION,X
```

ADDRESS TYPE 7 (JMP, JSR)

```
EXTENDED THREE BYTES:
          CCC NUMBER
          CCC SYMBOL
          CCC EXPRESSION

INDEXED TWO BYTES:
          CCC NUMBER,X
          CCC SYMBOL,X
          CCC EXPRESSION,X
```

ADDRESS TYPE 8

(BCC, BCS, BEQ, BGE, BGT, BHI, BLE, BLT, BMI, BRA, BSR, BVC, BVS)

```
RELATIVE TWO BYTES:
          CCC NUMBER
          CCC SYMBOL
          CCC EXPRESSION
```

ADDRESS TYPE 9 (ABA, CBA, CLC, CLI, CLV, DAA, DES, DEX, INS,

```
                INX, NOP, RTI, RTS, SBA, SEC, SEI, SEV, SWI,
                TAB, TAP, TBA, TPA, TSX, TXS, WAI)

INHERENT ONE BYTE:
          CCC
```

PSEUDO INSTRUCTIONS

CMN -

This pseudo-op is used to reserve/declare an area in Common for interprogram data communication. The syntax is:

```
CMN     SYMBOL,operand
```

The operand may be a NUMBER, a SYMBOL or an EXPRESSION. The operand is evaluated and the value obtained is used to reserve/declare that amount of bytes in the Common area. The CMN must not be labeled.

END -

This pseudo-op terminates a program. It marks the physical end of the source language program. The last statement of a program must be an END statement. The END statement must not be written with a label, generates no object code, and has no operand.

ENT -

This pseudo-op is used to declare a symbol that may be referenced by separately assembled programs. The syntax is:

```
ENT    SYMBOL
```

This pseudo-op must not be labeled and the operand field must contain a symbol that is defined elsewhere in the program.

EXT -

This pseudo-op is used to declare a symbol which may be referenced by the program but which is defined in some other program. The syntax is:

```
EXT    SYMBOL
```

This pseudo-op must not be labeled. The SYMBOL in the operand field must be declared by an ENT pseudo-op in the program in which it is defined.

EQU -

This pseudo-op assigns to a symbol a value other than the value normally assigned by the Program location counter. The syntax is:

```
Label      EQU    operand
```

The EQU statement must be labeled. The operand field may contain a NUMBER, a SYMBOL or an EXPRESSION. Symbols appearing in the operand field must be previously defined in the source program. This pseudo-op generates no object code.

FCB -

This pseudo-op generates one byte of object code. An 8-bit unsigned binary number corresponding to the value of the operand is stored in the object code. The format is:

```
[Label]        FCB    operand
```

The operand may be a NUMBER, a SYMBOL or an EXPRESSION. If the operand evaluates to a value larger than 255 only the least-significant eight bits will be used. The FCB pseudo-op may be labeled.

FCC -

This pseudo-op translates strings of characters into their seven bit ASCII code. The format is:

```
[Label]        FCC    'operand'
```

The text in the operand field is enclosed between apostrophes.

If an apostrophe is needed in the text it is represented by two apostrophes. However only one will be put into the object code. The FCC pseudo-op may be labeled.

FDB -

This pseudo-op generates two bytes of object code. A 16-bit unsigned binary number corresponding to the value of the operand is stored in the object code. The format is:

```
[Label]        FDB       operand
```

The operand may be a NUMBER, a SYMBOL or an EXPRESSION. This pseudo-op may be labeled.

FFP -

Used to generate a 9-byte floating-point constant whose value is given by the operand. The operand must be a NUMBER; the pseudo-op may be labeled.

```
[label]        FFP    Floating-point number
```

IFC -

This pseudo-op is used to cause the Assembler to process the following code normally if the value of the operand is not zero; but to ignore all source statements until a matching NIFC statement is encountered if the value of the operand is zero. The format is:

```
IFC        operand
```

The operand may be a NUMBER, a SYMBOL or an EXPRESSION. The IFC pseudo-op must not be labeled and must have an operand. IFC - NIFC blocks may be nested up to 8 levels deep.

MAC -

This pseudo-op is used in the definition of a Macro. All statements following the MAC pseudo-op up to the next MEND pseudo-op are stored in the MACRO TABLE as a Macro Definition. The syntax is:

```
Label          MAC [C]
```

A label is required on the MAC statement. The label is the Symbol (Name) by which a Macro is expanded or called. The operand field may contain a "C"; the "C" is used to specify whether or not comment lines in the Macro Definition are to be stored in the MACRO TABLE. If the "C" is present then the comment lines are stored and thus expanded with the rest of the lines in the Macro definition when it is called. By omitting the "C" the user can lower the main memory requirements needed to store the Macro definition.

NOTE: Macro definitions may not be nested but may contain calls to other Macros.

MEND -

This pseudo-op is used to indicate the end of a Macro definition. It must not have a label or an operand.

NAM -

This pseudo-op names the program. The syntax is:

```
NAM    SYMBOL
```

The SYMBOL in the operand field is passed to the Linking Loader as an Entry point. It must not be used as a label in the program. A NAM

pseudo-op must be included in each program as the first statement.

NIFC -

This pseudo-op is used as a terminator to an IFC pseudo-op. It must be unlabeled and have no operand.

PAG -

This pseudo-op causes the listing device to advance to the top of the next page. This statement does not appear on the listing, causes no object code to be generated, and must be unlabeled.

RMB -

This pseudo-op is used to reserve a block of memory whose length is the value of the operand. The syntax is:

[Label] RMB operand

The operand may be a NUMBER, a SYMBOL or an EXPRESSION. The RMB pseudo-op may be labeled and the block of memory reserved is cleared to zeroes. Symbols used in the operand field must have been previously defined in the program.

SET -

This pseudo-op assigns to a symbol a value other than the value normally assigned by the program location counter. The syntax is:

Label SET operand

The SET statement must be labeled. The operand field may contain a NUMBER, a SYMBOL or an EXPRESSION. Symbols appearing in the operand field must have been previously defined. This pseudo-op is like the EQU pseudo-op except that symbols may be defined more than once.

NOTE: STRUBAL+ allows synonyms for some of the pseudo-ops, as follows:

COMMON = CMN
ENTRY = ENT
EXTERNAL= EXT
EQUATE = EQU
MACRO = MAC
NAME = NAM
PAGE = PAG

MACROS

Macros are sections of code which are defined once at the beginning of a program and used and referenced by a mnemonic code with or without parameters.

Usually the code in a macro are statements that are repeated many times throughout a program. Macros provide a shorthand notation for repeating these sections of code.

The statements to be repeated are grouped in one place at the beginning of the program. They are preceeded by the MAC pseudo-op and followed by the MEND pseudo-op. The Macro is named by placing the name in the label field of the MAC statement. The Macro is called by placing the name of the Macro in the opcode field of a statement and any parameters, to be passed to the Macro, in the operand field separated by commas.

The effect of a Macro operation is the same as an open subroutine in that it produces in-line code. The in-line code is inserted in the normal flow of the program so that the generated statements are assembled with the rest of the program.

The Macro definition is known as the prototype. The source statements included in the prototype may be any legal Assembler or processor instruction except for another MAC pseudo-op.

Macro prototypes are of the form:

```
Header          MNAME   MAC   [C]
                ..................
Body            ..................
                ..................
Termination               MEND
```

Where:

MNAME is the name of the Macro

"C" is an optional operand to control the storing of comment lines from the prototype body.

Body is the sequence of source statements.

Termination is the line containing the pseudo-op MEND.

MEND is recognized by the Assembler as the end of the Macro definition.

Example of a Macro Prototype

```
line 1        LOOP    MAC C
line 2                LDX #$&1          LOAD X WITH ARGUMENT 1
line 3                LDA B #$&2        LOAD B WITH ARGUMENT 2
line 4        *
line 5                DEC B             B:=B-1
line 6                BNE *-1           B .ne. 0
line 7        *
line 8                DEX               X:=X-1
line 9                BNE *-6           X .ne. 0
line 10       *
line 11               RTS               ALL DONE
line 12               MEND
```

Line 1 is the header. It names the Macro as LOOP and specifies that comment lines in the body are to be stored in the Macro definition in the Macro Table.

Lines 2, 3 are source statements with substitutable arguments, (parameters) in the variable field (&1, &2). A substitutable argument is recognized by the presence of the ampersand. The character after the "&" is the argument number. A single macro may have up to 36 arguments. The first nine are indicated by the digits 1 - 9, and the rest by the letters A - Z. Argument 0 (&0) is special. This argument gives the number of arguments in the particular macro expansion.

Lines 4-11 make up the rest of the prototype body.

Line 12 is the termination line.

Lines 2 through 11 are stored in the Macro Table by the Assembler for later use. If the "C" was not on the header statement lines 4, 7 and 10 would not have been saved.

The Macro name "LOOP" is stored in the Symbol table with a pointer to the location of the Macro definition in the Macro Table.

A typical call to the Macro "LOOP" might be:

```
      LABEL       LOOP 1000,12
```

This would expand into the following:

```
            LDX  #$1000         LOAD X WITH ARGUMENT 1
            LDA B #$12          LOAD B WITH ARGUMENT 2
*
            DEC B               B:=B-1
            BNE *-1             B .ne. 0
*
            DEX                 X:=X-1
            BNE *-6             X .ne. 0
*
            RTS                 ALL DONE
```

The parameter/argument "1000" is substituted for &1 and the parameter/argument "12" is substituted for &2.

ADVANCED MACRO FEATURES

STRUBAL+ provides additional features which may be used within macros and functions to increase their power. Some of these are conditional assembly, the SET pseudo-op, argument 0 and STRUBAL+ generated labels. The following examples illustrate the convenience of these features and how they may be used.

EXAMPLE:

Suppose you require a macro to generate quite a large block of code. The usual macro expansion will generate this code every time the macro is expanded. It is more convenient to generate the code only once and generate a subroutine call to the first expansion subsequently. A flag is needed to mark the first call; conditional assembly is used to vary the code by the macro based upon the flag. The SET pseudo-op manipulates the flag. A STRUBAL+ label is used to branch around the first expansions code.

```
.FLAG         SET -1              FLAG SET TRUE HERE
```

```
                ....
                ....
                SUBCALL MYSUB        MAC EXP 1
                ....
                SUBCALL MYSUB        MAC EXP 2
                ....
*
*
*
SUBCALL         MACRO C              MACRO DEF
                JSR &1               GENERATED FOR ALL EXP
*
                IFC .FLAG            CHECK FOR FIRST EXP
                BRA !A
.FLAG           SET 0                TURN OFF FLAG FOR FUTURE EXP
&1              ...                  GEN FOR FIRST EXPAN
                ...
                ...                  MYSUB CODE
                :::
                RTS
!A              EQU *
                NIFC
                MEND                 END OF MACRO DEF
```

The macro SUBCALL generates the body of the code only once. Each subsequent expansion generates only the subroutine jump to the body of the first expansion. The first expansion must also produce the code to branch around the subroutine body for the first call. Within a function of a macro the symbol "!" signals STRUBAL+ that it is to generate a unique label. The character following the exclamation point must be alphabetic and makes the label unique within the expansion. The label is formed by the concatenation of a period, the character following the exclamation point and a 4-digit number provided by the compiler/assembler. This number changes for each macro expansion, so no duplicate labels are formed. Consider the macro prototype example again, written with labels instead of relative branches.

```
     LOOP   MAC C
            LDX #$&1
     !B     LDA B #$&2
     *
     !A     DEC B
            BNE !A
     *
            DEX
            BNE !B
     *
            RTS
            MEND
```

There are 26 possible labels in a macro or function expansion. They will be valid only within the expansion. A nested macro or function will have its own unique labels.

Suppose it is desired to build a table of up to 10 floating-point values. Each expansion of this macro may have a different number of values to build. Here, argument 0 is used with SET and conditional assembly to produce the table.

```
FTABLE          MACRO           DO NOT SAVE COMMENTS
*
* FORM TABLE OF FLOATING-POINT VALUES
*
COUNT           SET &0          GET COUNT OF ARGUMENTS
                IFC COUNT       HAVE AT LEAST ONE ARG?
COUNT           SET COUNT-1     COUNT DOWN
                FFP &1          FORM VALUE
                NIFC            END CONDITIONAL
*
                IFC COUNT       HAVE AT LEAST 2 ARG?
COUNT           SET COUNT-1     COUNT DOWN
                FFP &2          FORM VALUE
                NIFC            END CONDITIONAL
*
                IFC COUNT       HAVE AT LEAST 3 ARG?
COUNT           SET COUNT-1     COUNT DOWN
                FFP &3          FORM VALUE
                NIFC            END CONDITIONAL
*
                ....
                ....            SIMILAR FOR OTHER VALUES
                MEND            END OF MACRO DEF
```

This example shows how a variable macro can make use of argument zero. Argument zero is fixed for a particular macro expansion; nested macros will only affect the value of argument zero within their expansion.

MODIFICATION OF STRUBAL+ COMPILER

This appendix describes some locations in the compiler that might be modified by the user. All addresses are given in hexadecimal notation and are relative addresses; ie, they assume that the program origin is 0000. The user should take care to relocate the addresses as necessary for his system.

TABLE SIZING

000F

The double-byte constant found at this address gives the size of the memory space to be used for macros or functions. As supplied, the value is 1000 or 4k bytes.

0021

The double-byte constant at this address gives the size of the memory space used as a stack for macro or function expansion. As supplied, the value is

0100 or 256 bytes.

003A

The double-byte constant at this address gives the total memory space to be used by the compiler. All space not used by the object code, disk drivers or macro tables (see above) is used for symbol-table storage. As supplied, the compiler occupies 7FFF or 32k bytes.

008E

The double-byte constant stored here gives the number of symbols which can be stored in the available memory. It is calculated from the constants above. Since the compiler searches the table using hash-coding, the performance of the compiler will slow down considerably if the tables becomes too full.

INPUT/OUTPUT TO TERMINAL

070B

This is the address of an input subroutine used to get characters from the computer terminal. The character is returned in the A accumulator in ASCII form. The B accumulator and index register must be preserved.

070E

This is the address of an output subroutine used to display characters on the terminal or printer. The character is passed in the A accumulator in ASCII form. The B accumulator and index register must be preserved.

0746

Is the address of the "escape" character for option input. As supplied, it is the ASCII ESC character (1B hex).

07C1

This is the address of the "escape" character for compile/assemble.

0811

This is the address of the "escape" character for program name.

238A

This is the address of a subroutine which causes a formfeed or page eject on the listing device. As supplied, it outputs the ASCII formfeed character (0C). The formfeed character is found at 2392. If used with a listing device that has no formfeed capability, this subroutine must be modified. A suggested change is to output a number of carriage return/linefeed sequences equal to the difference between the page length and the number of lines on that page. A subroutine to output a carriage return/linefeed is found at 3E84. The count of lines output on a given page is found at 06FC. This value should be zeroed when a formfeed sequence is output.

MODIFICATION OF RUNTIME PACKAGE

Input/Output to terminal

0001

This address is the location of an input subroutine to get a character from the user terminal. The ASCII character is returned in the A accumulator, the B accumulator and index register are preserved.

0004

This address is the location of an output subroutine to put characters out to the user terminal. The ASCII character is passed in the A accumulator with the B accumulator and index register preserved.

0432

This is the prompt character for input. Normally the ASCII greater-than sign (3E hex).

0435

This is the escape character for input. It causes termination of the program and returns control to the operating system. As supplied, the ASCII Control-C (03 hex).

DELAY routine

0962

This byte is calibrated so that the delay loops used by the DELAY statement in STRUBAL+ will take one msec. As supplied, this constant is set to 150 decimal. For a 1-usec cycle processor, this constant should be about 165; for a 2-usec processor, 88. Interrupts will affect the accuracy of this calibration.

RANDOM number generator

02E3

This double-byte constant is the value to which the random seed is initialized whenever the argument passed to the subroutine is zero. The seed is also initialized to this value upon start-up. As supplied, this constant is 1357 hex.

0311

This is the RAM storage location for the random seed value.

LINKAGE TO RUNTIME PACKAGES

This section describes the linkages and calling sequences for the STRUBAL+ runtime-package subroutines. With this information the user can directly access the facilities of these packages and more easily debug compiled code.

The runtime packages utilize an area of COMMON for their link to compiled STRUBAL+ programs. The COMMON area consists of 107 bytes arranged as follows:

RUNSTK	80 BYTES	RUNTIME VALUE STACK
ACCUM1	2 BYTES	16-BIT ACCUMULATORS
ACCUM2	2 BYTES	
ACCUM3	2 BYTES	
ACCUM4	2 BYTES	
ACCUM5	9 BYTES	FLOATING-POINT ACCUM
ADRTO	2 BYTES	ADDRESS POINTER (TO)
ADRFM	2 BYTES	ADDRESS POINTER (FROM)
.EOF	2 BYTES	END-FILE FLAG FOR FILE I/O (0=NO EOF, FFFF= EOF)
.ERR	2 BYTES	ERROR FLAG FOR FILE I/O (0= NO ERR, FFFF=ERROR)
.CURIOB	2 BYTES	CURRENT FILE FCB ADDRESS

All STRUBAL+ runtime routines pass arguments using these locations and the index register. No registers are saved in the routines. Compiled programs refer to the runtime stack as .RNSTK+offset, and the accumulators as .ACCUM+offset.

Almost all of the runtime routines are called through the use of the dispatch subroutine (.RTBL). This name is declared EXTernal in every STRUBAL+ program and must be included in every procedure which intends to use runtime support. The linkage process will resolve all calls to .RTBL when the runtime packages are loaded. Each runtime routine called through the dispatcher is given a number. For example, the integer-addition routine is number 12. To call this routine, code as follows:

```
JSR .RTBL     call dispatch routine
FCB 12        runtime routine 12
```

The routine will return to the next instruction after the routine number.

RUNTIME ROUTINES

NOTES:

* indicates that the routine is in the floating-point package

** indicates that the routine is in the scientific-function package

*** indicates that the routine is in the file I/O driver package

Routine number

0) Integer bit-wise negation. Argument in index register; result in index register.

3) Integer bit-wise logical AND. First argument in ACCUM1; second argument in index register; result in index register.

6) Integer bit-wise exclusive-or. First argument in ACCUM1; second argument in index register; result in index register.

9) Integer bit-wise logical-or. First argument in ACCUM1; second argument in index register; result in index register.

12) Integer addition. First argument in ACCUM1; second argument in index register; result in index register.

15) Integer subtraction. First argument in ACCUM1; second argument in index register; result in index register.

18) Integer multiplication. First argument in ACCUM1; second argument in index register; result in index register.

21) Integer division. First argument in ACCUM1; second argument in index register; result in index register.

24) Integer compare. First argument in ACCUM1; second argument in index register; result in condition codes (works like 15).

27) Integer absolute value. Argument in index register; result in index register.

30) Integer signum function. Argument in index register; result in index register.

33) Integer modulus function. First argument in ACCUM1; second argument in index register; result in index register.

36) Integer exponential. First argument in ACCUM1; second argument in index register; result in index register.

39) Random integer. Argument in index register; result in index register.

42) Carriage-return/linefeed on terminal. No arguments.

45) Integer input from terminal. Result in index register.

48) Integer output to terminal. Argument in index register.

51) Unformatted string output to terminal. Address of string in index register; end of string marked with hex 04 character.

54) String input from terminal. Address to store string in index register.

57) Output spaces to terminal. Argument in index register.

60) String length. Address of string in index register; result in index register.

63) Convert string to integer. Address of string in index register; result in index register.

66) Convert integer to string. Address of string in index register; integer in ACCUM3.

69) Substring extraction. Address of source string in index register; address of destination string in ACCUM4; number of characters in substring in ACCUM3.

72) Transfer string. Address of source string in index register; address of destination in ACCUM1.

75) Delay. Argument in index register.

78) String comparison. Address of first string in ACCUM4; address of second string in index register; result in condition codes.

81) Byte input (PEEK). Address of byte in index register; result in index register as positive integer.

84) Byte output (POKE). Address of byte in index register; byte value in low-order of ACCUM3.

87) Floating-point output to terminal. Address of argument in index register.

90) Floating-point input from terminal. Address of result in index register.

93) Floating-point to integer conversion. Address of floating-point value in index register; integer returned in index register.

96) Integer to floating-point conversion (FLOAT). Integer in ACCUM1; address of floating-point value in index register.

99) Floating-point absolute value. Address of argument in index register.

102) Floating-point signum function. Address of argument in index register.

105) Floating-point negation. address of argument in index register.

108) Floating-point comparison. Address of first argument in index register; address of second argument in ACCUM1; result in condition codes (requires floating-point package).

111)* Move floating-point to A floating register in package. Address of argument in index register.

114)* Move floating-point to C floating register in package. Address of argument in index register.

117)* Move floating-point result to memory. Address for result in index register.

120)* Floating-point addition. Add A register to C register.

123)* Floating-point subtraction. Subtract C register from A register.

126)* Floating-point multiplication. Multiply A register by C register.

129)* Floating-point division. Divide C register by A register.

132)** Trigonometric sine in radians. Address of argument in index register.

135)** Trig cosine function in radians. Address of argument in index register.

138)** Trig tangent function in radians. Address of argument in index register.

141)** Inverse-sine function in radians. Address of argument in index register.

144)** Inverse-cosine function. Address of argument in index register.

147)** Inverse tangent function in radians. Address of argument in index register.

150)** Base-10 logarithm. Address of argument in index register.

153)** Natural (base e) log. Address of argument in index register.

156)** Base-10 to a power. Address of argument in index register.

159)** Base e to a power. Address of argument in index register.

162)** Arbitrary base to a power. Address of base in index register; address of power in ACCUM1; result at index register address.

NOTE: All scientific functions require the floating-point package routines.

165) Floating-point move. Requires floating-point package. Source address in index register; destination address in ACCUM1.

168) String to floating-point conversion. Address of string in index register; address of value in ACCUM4.

171) Floating-point to string conversion. Address of value in ACCUM3; Address of string in index register.

174)*** Open file. Address of filename string in index register; address of FCB in .CURIOB.

177)*** Close file. Address of FCB in .CURIOB.

180)*** Rewind file. Address of FCB in .CURIOB.

183)*** Integer read from file. Result returned in index register; file FCB

must have been set in .CURIOB.

186)*** Floating-point read from file. Result address in index register; file FCB must have been set in .CURIOB.

189)*** String read from file. Result address in index register; file FCB must have been set in .CURIOB.

192)*** Integer write to file. Argument in index register; file FCB must have been set in .CURIOB.

195)*** Floating-point write to file. Address of value in index register; file FCB must have been set in .CURIOB.

198)*** String write to file. Address of string in index register; string terminated with hex 04 character; file FCB must have been set in .CURIOB.

201)*** Skip to a record mark (linefeed or 0A character). File FCB must have been set in .CURIOB.

204)*** Put spaces into a file. Argument in index register; FCB must have been set in .CURIOB.

207)*** Write file record mark. File FCB must have been set into .CURIOB.

210) Set formatting constants for integer, floating point and string I/O. Number of decimal places in index register; integer part size in ACCUM4.

213)* Memory mover. Address of source in ADRFM; address of destination in ADRTO; number of bytes to move in accumulator B.

216) Push integer onto stack. Address of integer in index register.

219) Push floating-point onto stack. Address of value in index register.

222) Push string onto stack. Address of string in index register.

225) Pop integer from stack. Address of result in index register.

228) Pop floating-point from stack. Address of result in index register.

231) Pop string from stack. Address of result in index register.

234) Set .CURIOB and clear .ERR. Address of FCB in index register.

237) Stop program and return to system. If disk drivers are loaded, return to file system, otherwise return to monitor.

240) Handle parameters in procedure calls from STRUBAL+. (Let STRUBAL+ handle the setup for this routine!)

243) Formatted string output to terminal. Address of string in index register. Uses integer part size (210) to set string length; pads with blanks or truncates as necessary.

246) Formatted string output to file. Works as above. File FCB must have been set in .CURIOB.

There are a few routines which are called by name. They must be declared external in the calling program.

GETB

Reads a character from a file whose FCB address is in .CURIOB. The character is returned in the A accumulator. If the carry bit is set in the condition codes upon return, an end-file was encountered.

OUTB

Writes a character to a file whose FCB address is in .CURIOB. The character is passed in accumulator A.

DIGSET

Sets the number of digits for floating-point precision. The argument is passed in the index register.

MONTOR

This is the warm-start of the file system.

LPON

This routine revectors STRUBAL+ output to the LPT (lineprinter) device.

LPOFF

This routine revectors STRUBAL+ output to the CON (console) device.

INITIALIZATION SECTION

This section of the compiler program sets up the symbol table and other variables required by the rest of the program. It sets up the processing options and sets the processing flags appropriately. It's the only part of the compiler which interacts with the operator. The data-storage handlers are initialized before the table parameters are set. STRUBAL+ is designed to run in a minimum of 32k bytes of RAM. The program uses the space left between the end of the program code and the 32k-boundary as tables. If more memory is available, the tables can be expanded.

The first table provides space for macro-function definitions and is termed MACTBL. This table uses 4k of reserved space. A macro, or function stack of 256 bytes is then reserved (MACSTK). The rest of the space is allocated to the symbol table.

Each symbol-table entry requires 9 bytes of table space, thus 32k provides a table of aproximately 285 symbols. STRUBAL+ uses an entry point named TABLES in its I/O driver package to find the end of the program. (TABLES is not really an entry point; it is really just the last address in the package.) The space between the end of MACSTK and 7FFF (32k) is computed, the space is divided by nine--to find the number of symbols possible--and the address of the end of the table is computed. The result is

```
MACTBL=@TABLES+1          start of macro/function table
MACSTK=MACTBL+1000        start of macro/function stack
SYMTAB=MACSTK+0100        start of symbol table
NSYM=(7FFF-SYMTAB)/9      number of symbols is table
SYMEND=(NSYM*9)+SYMTAB    end of symbol table
```

All operations are performed with 16-bit arithmetic. To increase the symbol table's size, just patch the size value (7FFF) with the new value.

The compiler outputs a copyright notice and prompts the operator for processing options. You can specify almost any combination of seven options. All options remain off unless specified. They can be specified in any order and no delimiters are needed, but commas or spaces can be used between options if desired. A carriage-return ends the option list.

```
Option L: A listing to be generated
Option S: A symbol table list to be generated
Option O: Object code to be generated
          (cannot be used with X option)
Option M: Macro/Function expansion list is to be generated
Option I: Intermediate assembly output is to be generated
          (compile only)
Option X: Cross-reference output file to be generated
          (cannot be used with O option)
Option Q: return to system
```

Error messages are always generated regardless of the option setting. If an error is made while entering options, the escape character will drop all previous option settings, re-prompt for options and allow you to re-enter the proper options.

EXAMPLE OF OPTIONS: LS (CR) list, symbol table, no object code,
no intermediate, no cross-ref., etc.

The compiler prompts for an input file name. This name will be checked against the file directory--all errors are flagged. Only a proper (correct) file name is accepted. If you need an output file (effectively an X or O option), the compiler prompts for an output file name. This name will also be checked for proper syntax; if valid, its file is opened for output.

The program requests that you choose to either compile or assemble. Type an "A" to get an assembly or a "C" to invoke the compiler. The program responds to the last character typed before the carriage return. If that character isn't either an A or a C, it is ignored and the compiler re-prompts. An escape character clears the setting.

If the program is compiling, the next step is to generate the linkages for run-time routines. This requires an additional operator input--a program name. When the program issues a prompt, it accepts any character string up to 15 characters long. A carriage return terminates the name and an escape character drops any characters just typed and re-prompts. This name is used to output the runtime linkages.

```
NAM -------          program name
CMN .RNSTK,80        runtime stack has 80 bytes
CMN .ACCUM, 21       pseudo-registers
CMN .EOF,2           end-file flag
CMN .ERR,2
CMN .CURIO,2
BRA *+5              skip next line
EXT .RTBL            link to runtime routines
```

The compile flag (CFLAG) is set and the program rejoins the assembler flow.

Both the compile and assembly paths meet at PASS1A. Here the symbol table is filled with blanks and other global variables are initialized. PASS2 is the return point for the beginning of the second assembly pass.

Called by: program entry point
Calls: INITIO, SUB16, DIV16, MPY16, ADD16, PDATA1, INEEE, SPACER, STR1, ADRINT, OPENI, OPENO, MONTOR
Tables: SYMTAB, MNTAB, CHRTAB
Flags: CFLAG, PASS, OPTNS

MNTAB

This table defines all of the compiler's keywords. Each entry consists of seven bytes. The first four bytes contain the keyword's first four characters. The fifth and sixth bytes contain the address of a processing routine for that keyword. (There are exceptions such as STEP,THEN and TO. These keywords have no processing routine, so a dummy address 1234 (hex) is placed in the table entry.) The seventh byte of each entry is a flag for the processing routine.

Assembler keywords (mnemonics) have part of their machine-code in that byte. Assembler pseudo-operations have FF (hex) as a flag in the seventh byte. Compiler keywords have the flag F0 (hex) in the byte. The entries in this table are ordered alphabetically by the keyword's four characters. This allows the use of a fast binary-search algorithm to find keywords in the table. Note that the period (.) is defined as an alphabetic character here.

CHRTAB

This table defines the character set recognized by the processing routines. Each character has a byte associated with it which defines its use.

Bit 7 set..alphabetic
Bit 6 set..decimal digit
Bit 5 set..arithmetic operator
Bit 4 set..parenthesis
Bit 3 set..unused
Bit 2 set..delimiter, special character
Bit 1 set..hexadecimal digit
Bit 0 set..assembler register (A,B, or X)

The table is ordered such that the byte associated with an ASCII character may be found by using its ASCII code as a table offset. A character whose byte is zero will not be recognized by STRUBAL+ and will be skipped like a blank.

MAIN LOOP

This is the main loop for processing in both the compiler and assembler modes. Each line of program is read at MAIN1. All processing variables associated with a line are initialized. The line is then tested for an asterisk in the first column. Such lines are comments. If the listing option is set, the assembler prints comment lines. The compiler not only prints the comment but also passes the comment line to the object file (if object code is set in the options). This finishes the processing of a comment, so the program returns to MAIN1 for a new line.

Program flow now divides into two sections: one for the compile mode and another for the assembler mode.

The compiler checks whether an "IF" statement previously processed is pending. If so, then a transfer label is output to the object file. If the line's first column isn't blank, then the line has a label. The compiler checks the label's validity--if it's valid, passes it to the output file. Whether or not a label was found, a blank is now output to the object file. The compiler now checks for a keyword. If no keyword is found, the compiler looks for a user-defined function in the symbol table. A function is processed by initializing the pointer MACPTR with the address in MACTBL where the function is defined and setting MACFLG. If the function has a parameter list, PARMVE is called to move those parameters to the MACPAR buffer area. This completes the processing of the function, since subsequent calls to RDLINE (from MAIN1) will now read from the function definition. If neither a keyword nor function is found, an assumed LET (assignment) is processed. The four special-case keywords (STEP, THEN, TO and ELSE) are treated as though the were not keywords in this context.

The compiler checks the keyword table entry's seventh byte in MNTAB. If it contains the compiler keyword flag (F0 hex), the compiler jumps to the processing-routine address from the MNTAB entry bytes five and six. If the seventh byte is not F0, the compiler checks for two special cases:

The keyword END--both a compiler and assembler operator, it is processed as though it were a compiler keyword.

The assember pseudo-operation NAM is illegal in compiled code, because the computer puts out a NAM in the runtime linkage, NAMs are flagged as errors and not processed.

All other keywords are valid and therefore are passed intact to the object file. All processing routines return directly to MAIN1 to close the processing loop--except END.

Assembler processing begins at MAIN3A. If the conditional-assembly flag--CIFFLG--isn't set (not assembling) then checks are made for IFC or NIFC pseudops. Only these keywords are processed; all others are simply listed. If CIFFLG is set, processing jumps to MAIN3C where the SET pseudop is processed. If the line's first column isn't blank, the line has a label. This label is checked for validity--if valid it is stored in the symbol table. Note that register designators A, B and X are valid here. The assembler next checks for a mnemonic (keyword). If a valid keyword is found, the assembler jumps to the processing routine whose address was found in the mnemonic table entry. All processing routines (except END) return to MAIN1.

If no keyword was found, the assembler looks in the symbol table for a macro name. If this line is a macro call, the MACTBL pointer is initialized and MACFLG set. If the macro has a parameter list, PARMVE moves that list into the buffer MACPAR. This completes the processing of a macro call, because subsequent calls to RDLINE (from MAIN1) will now read the macro definition from MACTBL.

Invalid labels are handled at MAIN3B. An error message with the number 205 is output and the program resumes at MAIN1 for the next line. Invalid keywords encountered during assembly are handled at MAIN6. Error message 202 indicates that keywords are not in the proper format. If the format is valid but the character string is not found in the table, error 207 is output at MAIN8. Processing resumes at MAIN1.

Called by: (after initialization section)
Calls: RDLINE, PRINTL, CVBTD, STR1, OUTNAM, TPCHAR, WRTCHR, MNLKP, MACPSH, PRINTE, LKPSYM, PARMVE, STOSYM, ADRINT, XREF, (all processing routines)
Jumps: MAIN1
Flags: CIFFLG, LBFLG

RDLINE

This routine reads in a line of program code and stores it in INBUF. The maximum line length is 80 characters; longer lines will be truncated. Linefeeds and nulls are swallowed. If an end-file is encountered the program immediately jumps to the END keyword processing routine. The carriage-return character delimits the end of a line. All STRUBAL+ and assembler statements are one line long. RDLINE returns two pointers:

CULINE = the address of the beginning of the line,
CUCHAR = the address of the first character of the line. (Initially equal to CULINE.)

If the function/macro flag (MACFLG) is set, RDLINE reads from MACTBL instead of the source file.

RDMAC is called to get a line of macro or function expansion.

Called by: MAIN, POMAC (at RDLINA)
Calls: GETB, RDMAC, POEND
Buffers: INBUF
Pointers: CULINE, CUCHAR

RDMAC

This routine is called by RDLINE to expand a line of macro or function definition. It reads characters from MACTBL, starting at MACPTR. The ETB character (17 hex) marks the end of a definition. An "&" indicates that the number following it specifies a particular macro parameter. "&3", for example, indicates that the macro's third parameter is to be used. Parameters are handled at RDMAC3. The "!" character indicates an auto-define label. This label is expanded by taking a period, concatenating the letter which followed the "!" and adding the number contained in LABEL to it.

At RDMAC3 the parameter buffer MACPAR is searched to find the appropriate parameter. Parameter zero is special: it expands to the number of parameters in MACPAR, expressed as a hexadecimal constant. PARNUM contains the number of parameters, as set by PARMVE. Parameters other than zero move their defining string from MACPAR to the expanded line, replacing the "&" and the next character. The end result is that RDMAC returns a line of definition with special fields replaced with their actual values.

Called by: RDLINE
Calls: MACPUL, CVBTD, GCHRTB, CVHBS, PRINTE
Flags: MACFLG
Temps: MACPAR

MACPSH

This subroutine moves the current state of a macro expansion onto the macro/function stack. PARNUM, LABEL, MCLPTR, MACSAV and MACPTR are stacked. Next, MACPAR is pushed onto the stack in its reverse order. MACPSH is called when nested macros are encountered.

Called by: MAIN
Calls: PRINTE

MACPUL

This subroutine resets the state of the macro/function expansion from MACSTK. This is done whenever a nested macro or function ends. MACPTR, MACSAV, MCLPTR, LABEL and PARNUM are popped off the stack. MACPAR is popped from the stack.

Called by: RDMAC
Calls: none

PARMVE

This subroutine moves function/macro parameters from the input line to the buffer MACPAR.

Called by: MAIN
Calls: none

WRTCHR

This routine passes a specified character to the listing subroutine for listing output during compilation. If the intermediate-code option is not set, the routine does nothing. If it's set, the intermediate code is printed. The character is passed in the "A" accumulator.

Called by: MAIN, OUTCMP, LETPR, FNDDIM, DIMPR
Calls: OUTCHR

TPCHAR

This routine writes a character of object code to the output file during compilation. The desired character is passed in the "B" accumulator. If the object-code option is not set, this routine does nothing. The routine is also known as OUTBNR.

Called by: MAIN, OUTCMP, LETPR, FNDDIM, DIMPR
Calls: OUTBNR

OUTNAM

This routine writes a name (alphanumeric-character string) to list and object files during compilation. It assumes that NXTOK has set DESCRA to the address of the name and DESCRC to the number of characters in the name. It sets up for OUTCMP.

Called by: MAIN, NEXTPR, EVALS, DIMPR, READPR, DEFS, IFPR, CALLPR,
PROCPR, GTPRO, FNDDIM
Calls: OUTCMP

OUTCMP

This routine writes a character string to both listing and object files during compilation. The address of the character string is passed in the index register. A character count is passed in CMPCNT. The output can terminate in two different ways. If the carriage-return is encountered, a linefeed is appended and the routine returns. If the counter CMPCNT reaches zero, then a blank is output and the routine returns.

Called by: OUTNAM, LETPR, NEXTPR, STR1, CONVPR, STRQ, VALUPR, CALLPR, GOTOPR,
WHILPR, IFPR, QUOTPR, PEADPR, DIMPR, EVALS, OUTCR, FINDDIM, PROCPR
Calls: TPCHAR, WRTCHR, OUTCR

OUTCR

This routine writes a carriage-return to both listing and object files during compilation.

Called by: OUTCMP, LETPR, NEXTPR, EVALS, DIMPR, READPR, QUOTPR,
IFPR, CALLPR, GOTOPR, FNDDIM, PROCPR
Calls: OUTCMP, LINCK

COMPAR

This routine compares two character strings, returning the comparison's result in the condition codes. The routine is called with the address of a 5-byte parameter block in the index register. The parameter block has the first string's address in bytes one and two. The second string's address is in bytes three and four. Byte five holds the count of the number of characters to compare.

Called by: SYMCMP, MNLKP, IFPR
Calls: none

LETPR

This routine processes all STRUBAL+ assignment statements (the LET keyword is optional). If the LET keyword is present, the routine must get the next token and check it for validity as a name. If the LET keyword is absent, the name is already the present token and the routine is entered at LETPR2. If the token is invalid an error message is printed and the processing returns to MAIN1. This routine is also used by FORPR and GETAD processing, so the GAFLG2 flag determines which error message to print.

If the token is a valid name, the routine checks to see if the name is already in the symbol table. If it doesn't find it, the name is a new integer variable which must be defined. If it was found its dimension values are available. No arrays are allowed in FOR statements, thus if FORFLG is set, and the token is an array name, an error exists. STRFLG is set for legal arrays in assignment statements. The pointers to the present token are saved. If the token is floating-point mode, then FLPFLG is set. Floating-point expressions aren't allowed in FOR statements either--if FORFLG is set, this is an error. The same is true for the GETAD statements. The routine now searches along the input line for an equals sign. If none is found, this is a syntax error. Once the equal sign is found, the pointer is moved one character further.

If the routine was called to process a GETAD statement, the address to the right of the equals sign is found by code generated in the routine GETADR. Otherwise, the value of the right side of the equals sign is found by code generated in the routine EVALS. There are five possible paths for the processing: integer, integer array, floating-point, floating-point array, and integer parameter.

Integer values are handled at LET5. The code generated is simply a store-index-register instruction using the name from the left side of the equals sign. If it was a FOR statement, the routine was called as a subroutine and must return to FORPR. If it was not a FOR statement, the processing returns to MAIN1. Floating-point values are handled at LET3F. The code generated is:

```
LDX #NAME          the name left of the equals sign
STX ACCUM1         common area in runtime
LDX #RUNSTK        area in runtime where value is stored
JSR MOVER          move value to NAME address
```

If the name is a parameter, the first code line is LDX NAME. The processing returns to MAIN1.

Integer array elements are handled at LET3E. The subroutine STRDIM generates the code and processing returns to MAIN1.

Floating-point array elements are handled just after LET3D. The floating-point value from the right side of the equals sign is temporarily stored in a save-area in the runtime stack as follows:

```
LDX #SAVER         save area in runtime
STX ACCUM1         common area in runtime
LDX #RUNSTK        area in runtime where value is stored
JSR MOVER          move value
```

The EVADIM routine is then called to generate code to move the array element's address into the index register. This address receives the saved value.

```
STX ACCUM1         array-element address stored in runtime
LDX #SAVER         address of saved value
JSR MOVER          move value
```

Processing returns to MAIN1. Integer parameters are processed at LET4. The code generated is:

```
STX ACCUM3
get address of value in index register
LDA A ACCUM3
LDA B ACCUM3+1
STA A 0,X
STA B 1,X
```

Called by: MAIN, FORPR, GETAD
Calls: NXTOK, PRINTE, LKPSYM, DEFS2, GETADR, EVALS, STR1, FUNOP, EVADIM, RTBL, STRDIM, OUTCMP, WRTCHR, TPCHAR, OUTCR
Jumps: MAIN1
Flags: FORFLG, GAFLG2, STRFLG, FLPFLG

STRDIM

This routine generates code to store an integer-array element value. The value should be loaded in the index register by the previous code. The pointers DESCRC and DESCRA point to the array name in the input line. The routine first generates code to save the index register contents in the runtime area, the EVADIM routine is then called to generate code that puts the address of the array element in the index register. It then moves the value to the address.

```
LDA A ACCUM3          get high byte of value
LDA B ACCUM3+1        get low byte of value
STA A 0,X             store high byte
STA B 1,X             store low byte
```

Called by: LETPR, VALUPR, READPR
Calls: STR1, EVADIM

FORPR

This routine processes the FOR statement. It sets the FORFLG and clears the FLPFLG. LETPR then processes the "initial-value" part of the line. A new compiler-generated label is formed and the "maximum value" part of the line is processed in EVALS. This value is stored into a 4-byte storage area in the generated code which wil be labelled by the compiler. (Code generated that is not ROMable.) This label will be stored on the label stack, where it is picked up by NEXTPR. This storage area holds the value of the maximum and step expressions from the FOR statement. If the end of the line is now found, a step of one is assumed. Otherwise, EVALS is called again to evaluate the step expression. This value is also stored in the FOR storage block. If there is no room on the label stack for the new storage-area label, an error message is printed and processing returns to MAIN1.

```
        BRA *+6          skip around next 4 bytes
.LABEL  RMB 4            storage for two integers with label
```

Processing returns to MAIN1.

Called by: MAIN
Calls: LETPR, EVALS, CVBTD, PRINTE, STR1
Jumps: MAIN1
Flags: FORFLG, FLPFLG

NEXTPR

This routine handles the NEXT statement. The first step is to unstack a label from the label stack. If none is there, an error message is printed and control returns to MAIN1. Otherwise, this label will be used to refer to the storage area set up by the matching FOR statement. The next token from the line is checked for validity as a name and it is also checked in the symbol table. If it is not a simple integer variable in the table, this is an error. Otherwise, the following code adds the step value to the index variable:

```
LDX NAME            index variable
CPX .LABEL+2        compare with max value
BEQ *+18            if same, quit loop
STX ACCUM1          if not same, add step value
LDX .LABEL
JSR ADD
STX NAME            new value of index
JMP .LABEL+4        loop again
```

The processing returns to MAIN1.

Called by: MAIN
Calls: PRINTE, NXTOK, LKPSYM, OUTCMP, CVBTD, STR1, OUTCR, RTBL, OUTNAM
Jumps: MAIN1

PUSHPR

This routine generates code for the PUSH statement. PRESCN is called to set FLPFLG properly for the argument type. GETADR is called to generate code which places the address of the argument into the index register. Based on the values of FLPFLG and DSTFLG(set by GETADR), PUSHPR then calls one of three runtime routines for integer, string, or floating-point arguments. Processing returns to MAIN1.

Called by: MAIN
Calls: PRESCN, GETADR, RTBL
Jumps: MAIN1
Flags: FLPFLG, DSTFLG

POPPR

This routine generates code for the STRUBAL+ POP statement. PRESCN is called to set FLPFLG properly for the argument type. GETADR is called to generate code which places the address of the argument into the index register. Based on the values of FLPFLG and DSTFLG (set by GETADR), POPPR then forms a call to one of three runtime routines for integer, string or floating-point arguments. Processing returns to MAIN1.

Called by: MAIN
Calls: PRESCN, GETADR, RTBL
Jumps: MAIN1
Flags: FLPFLG, DSTFLG

UNARY OPERATORS

(NOTPR, ABSPR, RANPR, SIGPR, PEEKPR, FABS, ACOSPR, ASINPR, ATANPR, COSPR FEXP, FIXPR, LOGPR, NATPR, FNEG, FSIGN, SINPR, TANPR, TENPR)

All of STRUBAL+'s unary operators are similar. The integer routines (the first five) load the B accumulator with the runtime routine offset, store the value in temporary storage and call the UNOP routine to generate the code.

The floating-point routines load the B accumulator with the runtime offset and call the FUNOP routine to generate the code.

Called by: EVALS
Calls: UNOP, FUNOP

FUNOP

This routine generates code for floating-point unary operators. The desired routine's runtime offset is passed in the B accumulator.

```
LDX #RUNSTK+-----        address of argument in runtime stack
JSR .RTBL                call runtime
FCB ---                  offset
```

Called by: unary operators, LETPR, WRITPR, FLOTPR, XTOYPR
Calls: CVBTD, STR1, RTBL

FUNOP2

This routine sets up a floating-point binary function. It generates code to store the address of a floating-point value in the runtime stack location

```
LDX #RUNSTK+------       address of argument in runtime stack
STX ACCUM1               save address in runtime
```

Called by: EVALS, CMPR, XTOYPR
Calls: CVBTD, STR1

UNOP

This routine generates code for unary integer operations and is used by the binary-operation routine to generate its code. It has several entry points and two flags (UBFLAG and COMPL) which control the code generated. If entered at the entry point UNOP it generates:

```
LDX argument
JSR unary operation
STX argument
```

where "argument" is a location on the runtime stack. The operation table offset is passed in location TMPVAL. If entered at entry point UNOP2 with flag UBFLG and flag COMPL clear, then the same code is generated. If entered at UNOP2 with UBFLAG set, then the following code lines are formed:

```
LDX argument
STX ACCUM1
```

where ACCUM1 is a location in the runtime common area. If entered at entry point UNOP3 with flag COMPL clear, the following code is generated:

```
JSR unary operation
STX argument
```

(the last two lines of the code generated at UNOP). If the COMPL is set, then only the "jump-to-subroutine" code is generated.

Called by: BINOP, unary operators, FLOTPR, CMPR, ONPR
Calls: CVBTD, STR1, RTBL
Flags: UBFLAG, COMPL

BINOP

This routine generates code for binary integer operations. It generates the following code:

```
LDX  first argument
STX  ACCUM1
LDX  second argument
JSR  binary operation
STX  first argument
```

where the arguments are really locations in the runtime stack. Every binary operation frees one level on the stack—the location of the first argument is overwritten by the result of the operation.

Called by: binary operators, CMPR
Calls: STAKDN, UNOP, CVBTD, STR1
Flags: UBFLAG, COMPL, TMPVAL

CHKBIN

This routine checks the compiler's operand stack to ensure that there are two operands for a binary operator. A 301 error is printed if there aren't enough operands.

Called by: binary operators, XTOYPR, FBINOP
Calls: PRINTE

Binary operations (MULT, ADDZ, SUBTR, DIV, ORPR, ANDPR, EXOPR, MODPR)

These binary integer and logical operators generate equivalent code. They set the operator table offset into TMPVAL and check the operand stack for at least two operands. They then call BINOP.

Called by: EVALS
Calls: CHKBIN, BINOP

FLOTPR

This routine generates code for the FLOAT operation. The integer operand at the bottom of the compiler stack is to be converted to a floating-point value which will replace it on the stack.

```
LDX integer value
STX ACCUM1
LDX address in runtime stack for floating value
JSR FLOAT
```

The stack pointer--RUNCNT--must account for the length of a floating-point value. The floating-point mode flag FLPFLG is set.

Called by: EVALS
Calls: UNOP, FUNOP
Flags: FLPFLG, UBFLAG, COMPL

XTOYPR

This routine generates code for the XTOY (base to a power) operation. Both the base and power arguments are floating-point values.

```
     LDX address of first argument
     STX ACCUM1
     LDX address of second argument
     JSR XTOY
```

Called by: EVALS
Calls: CHKBIN, FUNOP, FUNOP2, STAKDN

GETAD

This routine generates code for the GETAD compiler statement. It uses LETPR with the flag GAFLG2 set.

Called by: MAIN
Calls: LETPR
Flags: GAFLG2

GETADR

This routine generates code which returns the address of a variable or expression to be used by other routines (as opposed to the value of a variable, etc). If an error was found in this process, the zero condition-code is returned cleared. If no error was found, the zero condition-code is set.

Called by: LETPR, SSETUP, VALUPR, CONVPR, XTRPR, STRCMP, LENGPR, WRITPR, PUSHPR, POPPR, IOCBPR, OPENPR
Calls: NXTOK, EVADIM
Flags: GAFLG, ERNUM

RTBL

This routine generates code for a runtime call, as follows:

```
     JSR .RTBL
     FCB offset
```

The offset is passed to this routine in the B accumulator.

Called by: LETPR, NEXTPR, FUNOP, UNOP, FBINOP, BYTEPR, CONVPR, VALUPR, LENGPR, XTRPR, STRCMP, OPENPR, READPR, WRITPR, QUOTPR, DELAY, POKEPR, FNDDIM, STRP, STRQ, PUSHPR, POPPR, PROCPR, IOCBPR OPERPR, STOPPR
Calls: CVBTD, STR1

STR1

This routine writes a line of code whose address is in the index register. The character count is set to 100--longer than any legal line--and OUTCMP is called.

Called by: initialization, main loop, LETPR, STRDIM, FORPR, NEXTPR, FUNOP, FUNOP2, UNOP, BINOP, RTBL, FBINOP, BYTEPR, CONVPR, STRP, STRQ, VALUPR, LENGPR, XTRPR, STRCMP, EVALS, CSECPR, CENDPR, DIMPR, QUOTPR, DEFS, POKEPR, POEND, CMPR, IFPR, WHILPR, BLOKPR, ONPR, GTPR, FNDDIM, PROCPR, DSECPR, OPENPR, POFUNC, RETPR
Calls:OUTCMP

Floating-point arithmetic (FADD,FSUB,FMUL,FDIV)

These routines generate code to set up for a floating-point binary arithmetic operation. Runtime routines move the arguments to the floating-point registers, as follows:

```
LDX address of argument one
JSR FTAREG
LDX address of second argument
JSR FTCREG
JSR operation
```

This routine sets up the argument pointers, accounting for the reversal of arguments necessary in division. Locations TMPVAL, TMPVAL+1 and UBFLAG pass the operation (UBFLAG) and the register pointers.

Called by: EVALS
Calls: FBINOP

FBINOP

This routine generates code for floating-point arithmetic operations as set up by the preceding routines. It then generates the following code:

```
LDX address of result in runtime stack
JSR FFDREG
```

to move the result back to the runtime stack.

Called by: floating-point arithmetic
Calls: CHKBIN, STAKDN, CVBTD, RTBL, STR1

BYTEPR

This routine generates code for the STRUBAL+ BYTE statement. The address of the left side is formed and stored in ACCUM4 in the runtime package. The right side is evaluated in integer mode and stored in ACCUM3 in the runtime package. The address (ACCUM4) is loaded into the index register and the POKE runtime routine is called to transfer the low-order byte to ACCUM3 to the address in the index register.

Called by: MAIN
Calls: SSETUP, STR1, EVALS, RTBL
Jumps: MAIN1
Flags: FLPFLG

STRING OPERATION ROUTINES

STRPR

This routine handles the STRING statement. It separates the STRING-IF form from the others, and determines the mode (integer/floating-point) of the left side of the forms with assignment. Syntax is checked and errors are flagged. (312 error number.) The right side of the equals sign is checked for a quote. If found, the processing for the quoted string is performed.

Called by: MAIN
Calls: SSETUP, NXTOK, MNLKP, PRINTE, STRQ, STRP

SSETUP

This routine performs setup for other string routines. It determines the mode of the left side of the equals sign and generates the code to form the address of the variable. Some syntax checking is performed as well.

Called by: CONVPR, VALUPR, LENGPR, XTRPR, STRPR
Calls: PRESCN, GETADR, NXTOK

CONVPR

This routine handles the CONVERT statement, converting a value into a string representation. SSETUP is called to form the address of the string. CONVPR generates code to store this address in ACCUM4 in the runtime package. The right side of the equals sign (value) is evaluated based on the mode of the left side. If the value is integer mode, the following code is generated:

```
STX ACCUM3        save value
LDX ACCUM4        recover address of string
JSR ASCII         call runtime conversion
```

If the value is floating-point mode, the following code is generated:

```
LDX #RUNSTK       address of floating-point value
JSR FASCII        call runtime conversion
```

Processing returns to the main loop at MAIN1.

Called by: MAIN
Calls: SSETUP, PRESCN, EVALS, CVBTD, STR1, OUTCMP, RTBL

STRP

This routine handles string assignments. It assumes that STRPR has generated code to form the address of the left side of the equals sign--the destination string. The address is stored in ACCUM4 in the runtime package. The address of the target string, the right side, is formed. STRP then forms the following code:

```
STX ACCUM3        save the target address
LDX ACCUM4        recover the destination address
STX ACCUM1        restore it
LDX ACCUM3        finish set-up
JSR XFER          runtime call
```

STRP checks to see if a comma follows the target string in the STRUBAL+ line. If so, string concatenation is indicated and the processing returns into STRPR with the next target string. Otherwise, processing returns to MAIN1.

Called by: STRPR
Calls: STR1, GETADR, RTBL, NXTOK, STRPR

STRQ

This routine handles assignment of a quoted string. It assumes that STRPR has generated code to form the address of the left side of the equals sign--the destination string. The following code is generated:

```
        STX ACCUM1        save address
        LDX #*+9          get address of quoted string
        JSR XFER          runtime call
        BRA .LABEL        branch around string to compiler-gen. label
        FCC '-----'       quoted string
        FCB $04           end string character
.LABEL  EQU *             dummy label
```

STRQ goes to STRP to check for possible string concatentation.

Called by: STRPR
Calls: NXTOK, STR1, OUTCMP, RTBL, QUOTPR, STRP

VALUPR

This routine handles the STRUBAL+ VALUE statement. It generates code to convert a string representation of a value into a value. It calls SSETUP to form the address of the left side of the equals sign--the value.

This address is stored in ACCUM4 in the runtime package. GETADR forms the address of the right-side, the string. If the value was integer mode, the following code is generated:

```
JSR VAL          runtime call
STX ACCUM3       save value
LDX ACCUM4       get value address
LDA A ACCUM3
LDA B ACCUM3+1   move value to address
STA A 0,X
STA B 1,X
```

If the value was floating-point mode, the following code is generated:

```
JSR FVAL         runtime call
```

The integer-mode code is formed by STRDIM. Processing returns to MAIN1.

Called by MAIN, LENGPR
Calls: SSETUP, STR1, GETADR, RTBL, OUTCMP, STRDIM
Jumps: MAIN1

LENGPR

This routine handles the STRUBAL+ LENGTH statement. It computes the length of a string. SSETUP is called to form the address of the left-side of the equals sign, the value. The length value must be integer mode. If it is not, the syntax error is returned. The code generated is the same as that for the integer mode of VALUPR, except for a different runtime call. VALUPR is called to generate part of the code lines. Processing returns to MAIN1.

Called by: MAIN
Calls: SSETUP, PRINTE, GETADR, RTBL, VALUPR
Jumps: MAIN1

XTRPR

This routine generates code for the STRUBAL+ EXTRACT statement. (Written as "XTRACT" to avoid confusion with the "EXTERNAL" statement.) The operation returns a specified substring from a given string. SSETUP is first called to form the address of the left-side of the equals sign, the destination string. The string mode must be integer. This address is stored in ACCUM4 in the runtime package. EVALS is next called to evaluate the length expression. This evaluation is done in the integer mode. GETADR is then called to form the address of the target string. A runtime routine is called to extract the desired substring. The code formed is as follows:

```
(address of destination in index register)
STX ACCUM4       save it for runtime call
(get pointer value in index register)
STX ACCUM3       save it for runtime call
JSR MID          runtime call
```

Processing returns to MAIN1.

Called by: MAIN
Calls: SSETUP, EVALS, STR1, GETADR, RTBL
Jumps: MAIN1

STRCMP

This routine handles the STRUBAL+ STRING IF statement. The statement's syntax allows logical comparison of two strings. Strings can be quoted or variable. If the first string is quoted, the following code is generated:

```
       LDX #*+8        address of quoted string
       STX ACCUM4      save in runtime
       BRA .LABEL      branch around string
       FCC '------'    quoted string
       FCB $04         end-string
.LABEL EQU *           dummy transfer label
```

If the string is a variable, GETADR is called to form the address and this is stored in ACCUM4. The next step is to search for a legal comparison operator. If a legal operator is found, its unbiased table value is saved on the compiler stack. If a legal operator is not found, this is a syntax error and the processing of the line terminates with an error message. The second string is now processed. If it is a quoted string, the following code is formed:

```
       LDX #*+5        address of quoted string
       BRA .LABEL      branch around string
       FCC '-----'     quoted string
       FCB $04         end-string
.LABEL EQU *           dummy transfer label
```

If the string is not a quoted string, GETADR is called to form the string's address. The operator table value is now recovered from the stack and used to form the appropriate conditional-branch code line. For example, if the test was for equality of the strings, the following code would be generated:

```
       BEQ  *+7
       LDX #0          false
       BRA *+5
       LDX #1          true
```

The processing finishes up by using IFPR

Called by: MAIN
Calls: NXTOK, GETADR, STR1, QUOTPR, MNLKP, RTBL, IFPR

Token-extraction Routines

These routines syntactically analyze STRUBAL+ statements by breaking each line into sub-lines, or tokens. Each token is composed of one or more sequential characters from the STRUBAL+ statement. The following are the tokens recognized in STRUBAL+:

Substrings	name, hex value, decimal value, floating-point value
Delimiters	#, comma, quote, %,$, (also used as hex indicator), greater-then sign
Parentheses	
Assignments	
Arithmetic	+,-,*,/
Machine register	A,B,X (assembly only)

Carriage return

NXTOK

This is the main token extraction routine. It is called with the input pointer CUCHAR set to the place in the STRUBAL+ line to begin. It returns four values. The starting-point of the token is returned in DESCRA (descriptor address). The number of characters in the token is returned in DESCRC (descriptor count). The type of token is returned in two values, the class (in accumulator A) and the return-code (RC) (in accumulator B). The class is the major division; substring, assignment, delimiter, etc., while the RC gives the type within a class. NXTOK strips blanks from the beginning of any token. The first nonblank character determines the type of token. (With exceptions described below.) All 1-character tokens are immediately returned.

GCHRTB determines the character type. If the first character is neither alphabetic nor numeric, the character must be a 1-character token. If it isn't one of the recognized 1-character tokens, then a syntax error exists. Most of NXTOK is concerned with substrings--multi-character tokens.

If a token's first character is a decimal digit, the token can either be a decimal number or a floating-point number. The float-mode flag FLPFLG determines which subroutine is called. DSCAN finishes the processing of a decimal number; FSCAN finishes the processing of a floating-point number.

If the first character is a delimiter, one further check must be made. If the character is a dollar sign the token might be a hexadecimal number. HSCAN is called to finish the processing of a hexadecimal value.

If the token's first character is alphabetic, the token might be a name. In assembly, the token might be a register. STRUBAL+ treats the decimal point as an alphabetic character. This means that if the first character is a decimal point, this token might be a floating-point number. Floating-point values can only appear in compilations, as defined by CFLAG. The floating-point-mode flag FLPFLG determines whether a floating-point value is legal in the present context. NSCAN is called to finish the processing of a name. A STRUBAL+ function is a name beginning with a decimal point. A STRUBAL+ compiler-generated label is a decimal point followed by a 5-digit decimal number. NXTOK must sort out all these cases.

Called by: MAIN, LETPR, NEXTPR, GETADR, STRPR, SSETUP, STRP, STRQ, STRCMP, EVALS, NSEVL, DIMPR, READPR, WRITPR, ADDR1, ADDR2, ADDR3, ADDR4, ADDR5, ADDR6, ADDR7, ADDR8, INXCK, POCMN, POENT, POEQU, POEXT, POFCB, POFCC, POFBD, PONAM, PORMB, IFPR, PRESCN, CALLPR, GOTOPR, FNDDIM, POFFP, PROCPR, IOCBPR, OPENPR, POSET, POMAC, POFUNC, POIFC

Calls: GCHRTB, FSCAN, NSCAN, DSCAN, HSCAN
Flags: CFLAG, FLPFLG
Pointers: CUCHAR, DESCRA, DESCRC

DSCAN

This routine scans substrings of decimal characters. On entry, CUCHAR points to the first character to be scanned. DSCAN continues to scan until it finds a non-decimal character. The decimal substring's address is returned in DESCRA and the substring's length in DESCRC. The B register is loaded with a type code of 09.

Calls: GCHRTB
Jumps: ENDSCN
Called by: NXTOK
Pointers: CUCHAR, DESCRC

NSCAN

This routine scans substrings of alphanumeric characters. On entry CUCHAR points to the first character to be scanned. NSCAN continues to scan until it finds a non-alphanumeric character. The alphanumeric substring's address is returned in DESCRA and its length in DESCRA. The B register is loaded with a type code of 01.

Calls: GCHRTB
Jumps: ENDSCN
Called by: NXTOK
Pointers: CUCHAR, DESCRC

HSCAN

This routine scans substrings of hexadecimal characters. On entry CUCHAR points to the first character to be scanned. HSCAN continues to scan until it finds a non-hexadecimal character. The substring's address is returned in DESCRA and its length in DESCRC. The B register is loaded with the type code 03.

Calls: GCHRTB
Called by: NXTOK
Jumps: ENDSCN
Pointers: CUCHAR, DESCRA, DESCRC

ENDSCN

This is a common return for routines: DSCAN, NSCAN and HSCAN.

FSCAN

This routine extracts a floating-point token from a STRUBAL+ line. It's called with the token's starting address in DESCRA. FSCAN also converts the token's value into the internal form and places it in the buffer FTEMP.

This internal form has nine bytes. The first two bytes are used for special flags and a 3-digit exponent in BCD form. The flags consist of the number sign, the exponent sign (0=plus) and a bit to indicate exact zero (1=number is zero). The remaining seven bytes hold up to 14 digits of mantissa in BCD form. The exponent is base 10. A floating-point value has an optional decimal point. FSCAN keeps track of the necessary exponent adjustment in EXPTMP (temporary exponent). Fraction (mantissa) digits beyond 14 are truncated. If the value has no explicit exponent, EXPTMP becomes the stored exponent. If the value has an exponent--indicated by the letter "E"--this exponent must be evaluated. The exponent has an optional sign and one or two digits in BCD. This value is converted to binary and combined with EXPTMP--with the result is converted back to BCD. It must be stored in positive form; its sign is placed in the flag bit.

FSCAN uses a subroutine--SCNCHR--to retrieve characters from the STRUBAL+ line. This subroutine moves the pointers CUCHAR and DESCRA appropriately. FSCAN does not return a character count. The B register is loaded with the RC value OF.

FSCAN returns two flags which indicate whether a floating-point or decimal (integer) value was scanned. (An integer is also a valid floating-point value.) DECFLG is set if a decimal point was encountered in the value, indicating that the value must be floating-point. The flag SIGDIG first indicates that a significant digit has been encountered; it also indicates an explicitely formed exponent. This is done by clearing SIGDIG. This is necessary because a floating-point value with such a form would return both flags cleared. PRESCN uses these flags to determine the type (floating/integer) of an expression.

Called by: NXTOK, POFFP
Calls: SCNCHR
Flags: SIGDIG, DECFLG
Buffers: FTEMP

SCNCHR

FSCAN uses this routine to extract characters from a STRUBAL+ line. It moves pointers DESCRA and CUCHAR along the line, returning the new character in accumulator A. SCNCHR moves CUCHAR one character too far at the end of the token--FSCAN moves CUCHAR back before it returns.

Called by: FSCAN
Calls: none
Pointers: DESCRA, CUCHAR

XREF

This routine outputs data blocks for use by cross-reference generator programs. Each block contains:

BYTES 1-4	Line number presently being processed. In hex-encoded ASCII form.
BYTES 5-16	Symbol name. In hex encoded ASCII form.
BYTE 17	Carriage return

These blocks are output during pass 2 of the assembler and during compilation. The "X" option bit controls this output function. The data blocks output to the object file, overriding the "O" option bit. The subroutine HASH formats the symbol name and places it in HSMBL for use by XREF.

Called by: MAIN, LKPSYM, GTPRO, POFUNC, IFPR, PROCPR
Calls: PBLOCK, OUTBIN, OUTBNR, HASH
Buffer: HSMBL
Flags: OPTNS, PASS

GCHRTB

This routine retrieves the byte in CHRTAB that is indexed by the value of the character in the A register. On return, register A contains the value of the retrieved byte for CHRTAB.

Calls: ADD16
Called by: DSCAN, HSCAN, NSCAN, NXTOK, EVADIM
Pointers: CHPTR

Mnemonic and Symbol-Table Routines

STOSYM

This routine stores a symbol and its value in the HASH coded symbol table (SYMTAB).

Hash coding is the method by which the symbol that is to be stored in the table is used to find the symbol table entry address. Hashing means simply to hash or to jumble up the ASCII characters in a symbol in such a way that a fairly unique number is generated. This number is manipulated to provide the actual address of the symbol's location in the symbol table.

On entry DESCRA contains the address of the symbol to be stored and DESCRC contains the symbol's length. HASH is called to produce an initial symbol-table address. If the entry at this address is empty, the symbol is stored there and the routine returns successfully. If the entry is not empty, SYMCMP checks that the symbol does not match this entry. If the entry matches, this is an error. If the entry doesn't match, SYMMOD forms the next sequential symbol-table address. The search continues until a vacant slot is found, the symbol is found already in the table, or the table's end is reached. These last two cases are errors.

Calls: HASH, PRINTE, SYMCMP, SYMMOD
Called by: MAIN, POCMN, POEXT, PONAM, POSET, POFUNC, DEFS, DIMPR, CALLPR, PROCPR
Pointers: SYMPTR
Temps: HSAV1, HSAV2, HSMBL

LKPSYM

This routine looks up a symbol in the symbol table and puts out cross-reference data blocks. On entry DESCRA contains the address of the symbol to be looked up and DESCRC contains its length. This routine proceeds much as the STOSYM routine except that, if the symbol is found, the value of the flag byte is returned in register B. If the symbol is not in the symbol table then a return code of FF is returned in register B.

Calls: HASH, SYMCMP, SYMMOD, XREF
Called by: NSEVL, POCMN, POENT, POEXT, LETPR, NEXTPR, DIMPR, READPR, DEFS, POSET, POFUNC, PRESCN, CALLPR, PROCPR
Pointers: HKEYA, SYMPTR

HASH

The HASH routine creates a hashed code for accessing the symbol table (SYMTAB). On entry DESCRA contains the address of the symbol to be hashed and DESCRC contains its length. The hashed value is calculated by first folding over the six bytes of the symbol (spaces are added to the right of symbols less than six bytes long) into two bytes--the six bytes are added together in groups of two bytes. This value is divided by the maximum number of symbols that the symbol table can hold (NSYM). The remainder from this division is then multiplied by nine (the entry length) and the result added to the symbol table's base address--producing a pseudo-random address. This value is returned in the index register.

Calls: ADD16, DIV16, MYP16
Called by: LKPSYM, STOSYM, MNLKP, XREF
Pointers: DESCRA, DESCRC, NSYM
Temps: HKEYA, HKEYB, HSAV1, HSAV2, HSMBL

DELSYM

Sometimes it is necessary to delete an entry from the symbol table. DELSYM can delete the last entry that has been added to the symbol table. On entry SYMPTR contains the location of the last symbol stored.

Calls: none
Called by: LBLCK
Pointers: SYMPTR

SYMMOD

This routine finds the next symbol slot in the symbol table. It increments the index register, holding the pointer value SYMPTR, nine times. If this pointer goes beyond the end of the symbol table space--SYMEND--the pointer is reset to the beginning of the table (SYMTAB).

Called by: STOSYM, LKPSYM, POEND
Calls: none
Pointers: SYMPTR, SYMTAB, SYMEND

SYMCMP

This routine compares a symbol with a symbol-table entry. On entry the index register points to the symbol-table entry and HSMBL contains the symbol. On return, the comparison's results are reflected in the condition codes (See COMPAR).

Calls: COMPAR
Called by: LKPSYM, STOSYM
Pointers: HSMBL, PCOUNT, PSTNG1, PSTNG2

MNLKP

This routine searches the mnemonic table (MNTAB). On entry DESCRA points to the mnemonic to be searched for and DESCRC contains its length. On return register A contains a return code:

A=FF Mnemonic is not in MNTAB
A=00 Mnemonic is in the table and
      Reg X contains the processing routine's address
      Reg B contains the partial opcode

A binary search is used to look for an element of an ordered table by successively halving a search interval, evaluating the remaining half. The binary search provides a much faster search than a linear search.

Calls: ADD16, COMPAR, MYP16, HASH
Called by: MAIN, STRPR, STRCMP, EVALS, PRESCN, EVADIM
Pointers: DESCRA, DESCRC, PCOUNT, PSTNG1, PSTNG2, YBADD
Temps: ENSIZ, IP, LP, MP

Expression Evaluation

These routines perform the necessary operations to evaluate the logical and arithmetic expressions found in STRUBAL+. The main routine--EVALS--generates code in the compilation pass. The routine NSEVL is used to find values in the assembly passes. EVALS allows both floating-point and integer data. It allows parentheses to set the order of evaluation. It can accept both unary plus and minus operators, as well as the set of unary and binary operations. Evaluation is performed from left to right with no operator precedence except as explicitely indicated by parentheses. NSEVL does not allow parentheses or logical operators. Only the four arithmetic operations are recognized. NSEVL performs unsigned 16-bit arithmetic only. The unary "location counter" operator (*) can be used in expressions evaluated by NSEVL.

UPSCLR

EVALS uses this routine to manipulate the operand stack (OPSTAK). The STAKUP routine is called to move the operand entries up, and the bottom of the stack (NUMBER) is cleared.

Called by: EVALS
Calls: STAKUP

STAKUP

EVALS uses STAKUP to manipulate the operand stack (OPSTAK). Each entry on the stack consists of two bytes. STAKUP moves each entry up one position. The count of entries--STKCNT--is incremented. A check is made for stack overflow; if found, an error message is produced. A maximum of 20 stack entries are permitted.

Called by: UPSLCR, EVALS
Calls: none
Counter: STKCNT

STAKDN

This routine is the inverse of STAKUP. EVALS uses it to move the entire OPSTAK down. The bottom of the stack is lost, and STKCNT is decremented.

Called by: BINOP, XTOYPR, FBINOP, FNDDIM, CMPR
Calls: none
Counter: STKCNT

EVALS

This routine performs expression evaluation for STRUBAL+ compilations. It is called to generate code for all types of expressions. The routine can call itself for evaluation of array index expressions. EVALS uses three stack structures. One stack is maintained in the runtime package. This stack contains the temporary values during evaluation. The compiled code references this stack with memory addresses of the form: .RNSTK+nnnn, where nnnn is the stack pointer value maintained in the location RUNCNT. Values are moved into the runtime stack as they are encountered, and the results of operations are returned to the stack. The final value ends at the top of the stack. The runtime stack holds 80 bytes that can be 40 integers, eight floating-point values or a mixture of both.

The second stack used by EVALS is the operand stack OPSTAK. This stack holds address pointers to the runtime stack. It is used to order the evaluation of expression which may have many levels of nested parentheses. It is manipulated by UPSCLR, STAKUP and STAKDN.

The third stack used by EVALS is the actual processor stack. This stack is used to store operator data and to retain status values for recursive calls to EVALS. A 100-byte area--EXSTK--holds this stack.

This routine may be entered at two points. Entry EVALS performs the necessary initialization for the first call to EVALS for an expression. Any other recursive calls enter at EXPR1. A flag--NXTFLG--determines if a new token must be processed. Unary operators (plus or minus) are then processed. Unary plus operators are simply skipped over. Unary minus operators must generate code. An extra zero is generated for integers, making the minus a binary operator. For floating-point values a special runtime routine changes the value's sign. If no unary operator is found, processing resumes at EXPR2.

A new token is processed here, and a set of branches based on that token determine further program flow. Terminator characters cause a branch to EVAL4. Names cause a branch to EVAL8. Integer or floating-point numbers cause a branch to EVAL7. A right parenthesis causes a branch to EVAL3. An arithmetic operator causes a branch to EVAL2A. A left parenthesis is stacked on the processor stack and a branch is made back to EXPR1. If the token is none of these, an error message is produced and evaluation is terminated. At EVAL2A an arithmetic operator is pushed onto the processor stack. Processing continues at EXPR2 (since a unary operator cannot follow an arithmetic operator).

EVAL3 checks for the completion of the processing of an array index. This is indicated by the present token being a right parenthesis and the top of the processor stack being a left parenthesis. If the dimensioned-variable flag DIMFLG is set the index is complete. Processing continues at EVAL4. If DIMFLG is not set, parentheses must be unmatched somewhere, causing an error message and termination of evaluation. If the processor stack is empty at EVAL3, this also indicates an error. One other case is handled at EVAL3. If a parenthesized expression has been evaluated fully, the right parenthesis is skipped over and processing of the outer expression continues at EVAL12.

EVAL4 is reached for a return from EVALS. It checks STKCNT to see that there are no pending operands on OPSTAK. If there are any, this is an error. It also checks the processor stack for pending values. If there are any, this is also an error. If this is a recursive call (array index) as indicated by DIMFLG, the processor stack is left. If DIMFLG is not set, the processor stack pointer is reset to its value at entry.

Constants are handled at EVAL7. Hex constants are converted to binary by CVHB and processing continues at EVAL7B. Integer (decimal) constants are converted to binary by CVDB and processing continues at EVAL7B. Token lengths are checked for validity, and errors are trapped. Code for integer and hex constants is generated at EVAL7B. The binary values is stored on the operand stack. A location on the runtime stack is reserved, and the following code is generated:

```
LDX #binary value
STX RUNSTK
```

A check is made that the runtime stack has enough room, and processing continues at EVAL12. Floating-point constants are handled at EVAL7F. The binary form of the constant is left in FTEMP by NXTOK and FSCAN. The following code is generated:

```
LDX #RUNSTK        address on runtime stack
STX ACCUM1
LDX #*+9           address of constant
JSR MOVER          runtime routine to move constant to stack
BRA *+12           branch around constant
FDB $----
FDB $----
FDB $----          floating-point constant in hex format
FDB $----
FDB $----
```

Storing the appropriate values in the operand stack and setting up the runtime location is handled at EVAL7E, as it was for integer and hex constants. The subroutine FCOUT is used to form the 'FDB' lines of code.

Names are handled at EVAL8. A check is made to see if the name is a STRUBAL+ keyword. If it isn't, processing continues at EVAL9. A compiler opcode forms a terminator and processing continues at EVAL4. If the name isn't a compiler opcode but is in the keyword table, then it might be a function name. If it begins with a period, it is a function name. If not, the name is processed at EVAL9. EVALS must then check that the floating-point mode of the function is valid in the expression context. The mode of the expression is found in FLPFLG. The mode of the function is found from the high order bit of its I D byte If the bit is set, the function is valid in a floating-point context. If the bit isn't set, the function is valid in an integer context. If the function mode does not match the expression mode a mismatch error is

indicated. There are some exceptions to this rule. The .EOF (end-file) and .ERR (file-error) functions are more like names than real functions. They are processed at EVAL9E like integer variables. Some functions have both and integer and floating-point form (logicals, SIGN and ABS). These functions are valid in either integer or floating-point context. The floating-point functions' indicator bit is stripped off and both types of functions are pushed onto the processor stack at EVAL8C.

Variable names are handled at EVAL9. The routine DEFS is called. If the variable isn't yet defined, it is defined as a simple integer at this point. In any case, the symbol-table data is found and stored in DMSAV and DMDATA. The dimension bits in DMDATA are checked. Simple variables (undimensioned) are handled at EVAL9C. Dimensioned variables are handled by stacking the parameters of this level of expression evaluation on the processor stack. The variable's mode is checked for consistency with the expression--mismatches are flagged as errors. The co-routine FNDDIM is called to generate code to form the array element's address. FNDDIM calls EVALS to evaluate the array indices. The following code is then generated to move the value into the runtime stack (Floating-point mode):

```
STX ACCUM4        save array element address
LDX #RUNSTK       address in runtime stack
STX ACCUM1
LDX ACCUM4        restore array element address
JSR MOVER         runtime call to move value
```

The operand stack value and the runtime stack space checks are handled at EVAL7E. Integer array elements are handled similarly at EVAL9A. The code generated to move the value into the runtime stack is as follows (integer mode):

```
LDX 0,X           get value from array element address
STX RUNSTK        store it into runtime stack
```

The operand stack, etc., is handled at EVAL7D. Non-dimensioned variables are handled at EVAL9C. Integers are handled at EVAL9E, floating-point variables at EVAL9D. Mode-mismatch errors are flagged here. Floating-point variables generate the following code:

```
LX #RUNSTK         runtime stack address
STX ACCUM1
LDX #variable name
JSR MOVER          runtime call to move value
```

The operand stack, etc. is finished up at EVAL7E, as for floating-point array elements. Integer variables generate the following code:

```
LDX #variable name
STX RUNSTK
```

The operand stack, etc., is finished up at EVAL7D, as for integer array elements.

Operators are pulled off the processor stack at EVAL12. If the processor stack isn't empty, processing continues at EVAL13. If the stack is empty, a check is made to see if the last token was a carriage return. If it was, this is a terminating condition...transfer to EVAL4. If the last token was a comma, this is also a terminating condition. Otherwise, continue processing at EXPR2. Unary integer minus (as indicated by UNFLAG) and left parentheses cause

branches back to EXPR2 and EVAL13. All other operators found on the processor stack are either arithmetic or functions. Their I D value is checked for validity. A bias is removed. The result is an index into one of two operator tables. One table is for integer functions, the other for floating-point functions. These tables contain the addresses of processing routines for the various functions. The unbiased function I D is an index into the appropriate table--a subroutine call is made to the processing routine. Note that some table entries are undefined and cause an error message. Processing loops back through EVAL12.

Called by: LETPR, FORPR, BYTEPR, CONVPR, XTRPR, DELAY, WRITPR, POKEPR, IFPR, WHILPR, ONPR, FNDDIM
Calls: PRINTE, NXTOK, UPSCLR, CVHB, CVDB, CVBTD, STR1, OUTCMP, OUTHL, OUTHR, MNLKP, DEFS, FNDDIM, WRTCHR, TPCHAR, OUTCR, unary operators, binary operators, FLOTPR, XTOYPR, FUNOP2, OUTNAM
Tables: OPTBL, OPTBL2
Stacks: OPSTAK, runtime stack, processor stack

DELAY

This routine generates code for the DELAY keyword. It evaluates the integer argument and calls a runtime routine to delay the specified number of milliseconds. Processing resumes at MAIN1.

Called by: MAIN
Calls: EVALS, RTBL
Jumps: MAIN1
Flags: FLPFLG

Assembly-NSEVL evaluation routine

This routine evaluates expressions during assembly processing. A straight left-to-right evaluation is performed without regard to operator precedence or hierarchy of operators. Only +, -, * and / are permitted. No parentheses, logical operators or floating-point values are allowed here.

The relocation indicator--RELFLG--is set if the final result is relocatable. A result is considered relocatable if it contains an odd number of relocatable terms. The common flag--CMNFLG--is set if a symbol is found during evaluation which was marked as common in the symbol table.

On entry the class code and type code for the first token of the expression are in registers A and B respectively--RELFLG and CMNFLG are cleared. NSEVL scans the operand field and performs the indicated operations, storing the intermediate results in VALUE and TEMP. On return, the final 16-bit unsigned result is stored in ADR1 and ADR2. The B accumulator is cleared if no errors were found. If errors were found, the index register is set to the error number and the B accumulator is set to FF hex.

Calls: ADD16, SUB16, MPY16, DIV16, CVDB, CVHB, LKPSYM, NXTOK
Called by: POCMN, POEQU, POFCB, POFDB, POIFC, PORMB, POSET
Flags: CMNFLG, RELFLG
Pointers: CLASS, CLFLG, CUCHAR, DESCRC, OPERN
Temps: TMPVAL, VALUE

CVHB

This routine converts hexadecimal character strings into a 16-bit binary value. When CVHB is called, location DESCRA contains the address of the hexadecimal string and location DESCRC contains its length. The length cannot exceed four. The converted value is returned in the index register.

Calls: CVHBS
Called by: NSEVL, EVALS
Flags: none
Pointers: DESCRA, DESCRC
Temps: HVAL

CVHBS

The CVHB routine uses this routine to convert an ASCII hexadecimal character to binary. On entry, the index register points to the character; on return the A register contains the binary value.

Calls: none
Called by: CVHB

CVDB

This routine converts decimal character strings into a 16-bit binary value. When CVDB is called, location DESCRA contains the string's address and location DESCRC contains its length. The length cannot exceed five. The converted value is returned in the index register.

Calls: MPY16
Called by: NSEVL, EVALS, DIMPR
Pointers: DESCRA, DESCRC
Temps: DCOUNT, DVAL, DXSAV, TENVL

ADD16

This routine adds together two unsigned 16-bit values. On entry the index register points to a four byte area of storage that contains the values to be added together. Bytes 1 and 2 are added to Bytes 3 and 4 and the result is stored in Bytes 1 and 2.

Calls: OVFLW
Called by: GCHRTB, HASH, MNLKP, NSEVL, initialization

SUB16

This routine subtracts two unsigned 16-bit values. On entry the index register points to a 4-byte storage area that contains the values to be subtracted. Bytes 3 and 4 are subtracted from Bytes 1 and 2 and the result is stored in Bytes 1 and 2.

Calls: OVFLW
Called by: NSEVL, initialization

MPY16

This routine multiplies two unsigned 16-bit values. On entry the first value is in registers A and B and the second value is in the two bytes pointed to by the index register. The results are truncated to 16-bits and returned in registers A and B.

Calls: OVFLW
Called by: CVDB, HASH, MNLKP, NSEVL, initialization, DIMPR

DIV16

This routine divides two unsigned 16-bit values. On entry the dividend is in registers A and B and the divisor is in the two bytes pointed to by the index register. The result is placed into registers A and B, and the remainder is returned in the index register.

Calls: none
Called by: HASH, NSEVL, initialization

OVFLW

This routine checks for overflows in 16-bit unsigned arithmetic. Some routines which use arithmetic allow overflows, others don't. The flag OVFLG is set for those routines which allow overflows. Otherwise, overflows are flagged as errors and processing is allowed to continue. The index register is preserved.

Called by: ADD16, SUB16, MPY16
Calls: PRINTE
Flags: OVFLG

Listing Routines

PRINTL

This routine, during pass 2, checks the options byte to see if the L option and the M option has been selected. PRINTL calls routine OUTL to print a line of listing if the L option has been selected. PRINTL also checks to see if the Assembler is in the Macro mode, if it is, it checks the M option to see if expansion lines from macros are to be listed.

Calls: LINCK, OUTL, SPACER
Called by: ADDR9, LCNAB1, LCN2, LCN3, MAIN, POCMN, POEND, POENT, POEQU, POEXT, POFCB, POFCC, POFDB, POIFC, POMAC, PONIFC, PORMB, POSET
FLAGS: OPTNS, PASS

LINCK

This routine is called to make sure that the output listing is formatted into pages of 60 lines each. If the line count (LCOUNT) is equal to zero, the system console is spaced to the top of the next page.

Calls: SPACER
Called by: POEND, PRINTL, OUTCR
Pointers: LCOUNT

SPACER

This routines outputs a form feed (0C hex) character to the listing if the L option is set.

Calls: OUTEEE
Called by: LINCK, PRINTL, initialization, POEND, LSTLN, POPAG

OUTL

This routine does the actual printing of the listing. Ten global values format the system.

MCOUNT	Number of bytes of machine code 0, 1, 2, or 3
POP	Pseudop to be printed 0=no, 1, 2 bytes
OPCD	Opcode in hex to be printed
ADR1, ADR2	Second and third bytes of machine code
MACFLG	Macro expansion line flag
LINEN	Line number
CMNFLG	Common flag
RELFLG	Relocatable flag
ENTFLG	Entry flag
EXTFLG	External flag

Note: For all flags 00=no FF=yes

Calls: OUT2HS, OUT4HS, OUTCHR, PDATA1, OUTS
Called by: PRINTL

CVBTD

This routine converts a 16-bit binary value into a 5-character decimal string. On entry registers A and B contain the 16-bit binary value to be converted and the index register points to an area of storage where the converted string is to be stored.

Calls: none
Called by: POEND, PRINTE, MAIN, FORPR, NEXTPR, FUNOP, FUNOP2, UNOP, BINOP, RTBL, FBINOP, CONVPR, EVALS, DIMPR, QUOTPR, IFPR, WHILPR, BLOKPR, ONPR, CALLPR, GOTOPR, FNDDIM, PROCPR
Pointers: index register
Temps: SAVEA, SAVEX, SAVEX1
Tables: KIOK

PRINTE

This routine prints error messages on the system console. Error messages are always printed as the errors occur during both PASS1 and PASS2. On entry the index register contains the error number in BCD. The routine prints the error number and the source line that caused the error, then increments the error count in ECOUNT. This count is printed at the very end of the assembly.

Calls: CVBTD, PDATA1, CRLF, OUTHL, OUTHR
Called by: ADDR1-5, 7, 8, INXCK, LBLCK, MAIN, POCMN, POEND, POENT, POEQU, POEXT, POFCB, POFCC, POFDB, PONAM, PORMB, P2ERR, STOSYM, LETPR, FORPR, NEXTPR, CHKBIN, STRPR, LENGPR, EVALS, OVFLW, READPR, EVADIM, FNDDIM, GTPRO, IOCBPR, PROCPR, POFUNC, POMAC, POIFC, POSET, RDMAC, MACPSH
Pointers: ECOUNT
Temps: ERNUM

DSECPR

This routine handles the DSECT compiler statement. A dummy code line is output for any label which may be pending. The current compile options are saved in DSCTMP. The list and object code options are then turned off. No code will be generated for all statements up to a DEND. Processing returns to MAIN1.

Called by: MAIN
Calls: STR1
Jumps: MAIN1
Flags: OPTNS
Temps: DSCTMP

DENDPR

This routine handles the STRUBAL+ DEND statement. Processor options are restored from DSCTMP and processing returns to MAIN1.

Called by: MAIN
Calls: none
Jumps: MAIN1
Flags: OPTNS
Temps: DSCTMP

CSECPR

This routine processes the STRUBAL+ CSECT keyword. It sets the common flag COMFLG which is used in DIMPR to assign variables to common. It puts out a dummy code line. Processing continues at MAIN1.

Called by: MAIN
Calls: STR1
Jumps: MAIN1
Flags: COMFLG

CENDPR

This routine processes the STRUBAL+ CEND keyword. It clears the common flag--COMFLG--used in DIMPR. This causes variables to be defined outside of common. It puts out a dummy code line. Processing continues at MAIN1.

Called by: MAIN
Calls: STR1
Jumps: MAIN1
Flags: COMFLG

DSTPR

This routine sets the "define-string" flag—DSTFLG—and calls DIMPR to handle the STRUBAL+ DSTRING keyword.

Called by: MAIN
Jumps: DIMPR
Flags: DSTFLG

INTPR

This DIMPR entry point handles the INTEGER keyword.

DEFPR

This routine handles the DEFINE STRUBAL+ keyword. It sets the floating-point flag--FLPFLG--and goes to DIMPR.

Called by: MAIN
Jumps: DIMPR
Flags: FLPFLG

DIMPR

This routine sets up storage for most STRUBAL+ variables. All arrays, floating-point variables, strings and integers defined by INTEGER statements are processed here. If the flag COMFLG is set, common references are produced; if it isn't set, data bytes are reserved in-line and code is generated to jump around the data. Parameter names generate no code, as though they were defined in a DSECT. Defined names are put into the symbol table with the appropriate attributes. Dimensioned names store the first dimension value in the table. Nondimensioned names store the line number of their definition statement. This value appears in the symbol-table's output. DIMPR computes the number of bytes to reserve for the data area. Dimensions must be decimal numbers. Floating-point values have 9 bytes per entry, integers have 2 bytes per entry and strings have single bytes per entry. Strings automatically add space for the end-string character required in strings. This means that a single character string must have at least two bytes, and that the first dimension in a dimensioned string will be incremented in DIMPR. Processing returns to MAIN1.

Called by: MAIN, INTPR, DEFPR, DSTPR
Calls: CVBTD, STR1, NXTOK, LKPSYM, STOSYM, OUTCMP, WRTCHR, TPCHAR,
PRINTE, CVDB, MPY16, OUTCR, OUTNAM
Jumps: MAIN1
Flags: DSTFLG, FLPFLG, COMFLG

PRNTPR

This routine sets the flags for WPRO to process a PRINT STRUBAL+ keyword. The read/write flag WRFLG is cleared, and the file I/O flag (FIOFLG) is cleared.

Called by: MAIN
Jumps: WPRO

Flags: WRFLG, FIOFLG

INPR

This routine sets flags for WPRO to process the INPUT keyword. FIOFLG is cleared and WRFLG is set for reading.

Called by: MAIN
Jumps: WPRO
Flags: FIOFLG, WRFLG

OPENPR

This routine handles the OPEN statement. It calls IOCBPR to process the control-block declaration. It then gets the address of a character string which is passed to the run-time package. This string is the name of the file to be opened. It is either a quoted string or a string name. Quoted strings are processed by QUOTPR (entry QUOT2) and string names are handled by GETADR. A run-time call is generated to the file I/O package and processing returns to MAIN1.

Called by: MAIN
Calls: IOCBPR, NXTOK, QUOTPR, GETADR, RTBL
Jumps: MAIN1

CLOSPR

This routine handles the CLOSE keyword--generating a runtime call to terminate sequential-file I/O. It calls IOCBPR to handle the control-block declaration. Processing returns to MAIN1.

Called by: MAIN
Calls: RTBL, IOCBPR
Jumps: MAIN1

REWPR

This routine handles REWIND keyword by generating a runtime call to rewind the sequential files. It calls IOCBPR to handle the control-block declaration. Processing returns to MAIN1.

Called by: MAIN
Calls: RTBL, IOCBPR
Jumps: MAIN1

READPR

This routine handles the READ statement and processes INPUT as well, based on the file I/O flag—FIOFLG. The entry point--READPR--sets the read/write flag WRFLG and the file I/O flag. The processing then jumps to WPRO which parses the STRUBAL+ line.

Entry point READ2 is called from WPRO if the read/write flag is set and an expression is encountered in the STRUBAL+ line. It checks that the token is a valid name. If it isn't, a syntax error exists and processing continues at WPRO with the next token. The name is searched in the symbol table for its mode. If the name isn't in the symbol table, it is assumed to be a simple integer variable and processing continues at READ4. The variable is defined there. Defined integers are processed at READ3A. Floating-point variables are handled at READ3. The subroutine EVADIM generates the code to form the address of the variable or array element in the index register. A runtime routine is then called to perform input from either a file or the console, depending on the file I/O flag. An error return from EVADIM causes processing to quit and branch back to MAIN1.

Entry point READ3A handles defined integer arrays. Simple integer variables are handled at READ4. Integer arrays are handled by generating a runtime call (either file or console) to form the value in the index register. The subroutine STRDIM then stores the value into the array element. Processing continues at WPR3B, resetting the token pointer properly.

Entry point READ4 handles simple integers. A runtime routine forms the value in the index register (either file or console input). The code generated is as follows:

```
JSR INPUT        leave value in index register
STX NAME         store in named variable
```

The variable name is placed into the symbol table if it wasn't already there. Processing continues at WPRO.

Called by: MAIN, WPRO
Calls: WPRO, PRINTE, LKPSYM, EVADIM, RTBL, STRDIM, OUTCMP, OUTCR, DEFS, OUTNAM, IOCBPR
Jumps: MAIN1
Flags: FIOFLG, WRFLG

WRITPR

This routine handles the WRITE statement. It processes the OUTPUT statement and forms part of the READ and INPUT processing. Entry point WRITPR clears the read/write flag WRFLG and sets the file I/O flag FIOFLG. Processing continues at WPRO.

Entry point WPRO is the start of the line-processing loop. The present token is found and its type determined. Right parentheses are skipped. Quoted strings are handled by a call to the subroutine QUOTPR. Processing resumes at WPRO.

At entry point WPR1, the routine checks for a slash in the line. The appropriate runtime call is generated for console (linefeed) or file (skip record) operation. Processing continues at WPRO.

At entry point WPR2, the routine checks for a carriage return. If found, the processing jumps back to MAIN1. Otherwise, it checks for a greater-than sign. This signifies a "space" operation. The integer expression is evaluated and the appropriate runtime call (file or console) generated. Processing continues at WPR3B, resetting the token pointer and returning to WPRO.

At entry point WPR3, the routine checks for a comma. If found, processing continues at WPR0. It checks next for a left bracket. A left bracket indicates a format specification whose first integer expression is processed by EVALS. If there is a second integer expression, it is also processed by EVALS. Code is generated to make a runtime call with the new format parameters. The routine next looks for a percent sign. A percent sign indicates string I/O. The subroutine GETADR is called to form the address of the string. An error return causes processing to jump back to MAIN1. If there was no error in GETADR, a runtime call is formed. There are four possible runtime calls, based on the values of WRFLG and FIOFLG. Continuing processing at WPR3B, the token pointer is reset and the routine returns to WPR0.

Entry point WPR3A handles all other cases. If WRFLG is set, processing transfers to READ2 in READPR. Otherwise, the token pointer is backed up and the subroutine PRESCN is called. This returns the floating-point mode of the expression. Evaluation of the expression is then performed by EVALS. An evaluation error causes processing to return to MAIN1. If there was no error in EVALS, a runtime call forms the output. There are four possible runtime calls, based on FIOFLG and FLPFLG. The subroutine FUNOP generates floating-point outputs. Processing falls through to WPR3B.

Called by: MAIN, READPR
Calls: READPR, NXTOK, RTBL, EVALS, GETADR, PRESCN, FUNOP, IOCBPR
Jumps: MAIN1
Flags: FIOFLG, WRFLG, FLPFLG

QUOTPR

This routine handles quoted strings in STRUBAL+ statements. It uses the FCC assembler pseudo-op processing routine. The following code is generated:

```
        LDX #*+9          point to character line
        JSR STROUT        runtime call (file/console) to output string
        BRA .LABEL        branch around string
        FCC '-----'       character string from line
        FCB $04           end-string character
.LABEL  EQU *             transfer point
```

The subroutine QPR1 is used to output the transfer labels.

Called by: WRITPR, STRQ, OPENPR
Calls: STR1, RTBL, OUTCMP, CVBTD, OUTCR, QPR1, WRTCHR, TPCHAR, FCCB(POFCC)
OUTCR
Flags: FIOFLG

QPR1

This routine outputs a transfer label from STRUBAL+.

Called by: QUOTPR, CALLPR, PROCPR
Calls: OUTCMP
Buffers: OUTLN

IOCBPR

This routine forms the code for file control-block declarations in I/O statements. It checks the file control-block declaration for errors and passes the address of the block to the runtime package.

Called by: OPENPR, CLOSPR, REWPR, READPR, WRITPR
Calls: NXTOK, GETADR, RTBL, PRINTE

POKEPR

This routine handles the POKE statement. EVALS evaluates the integer expression; an evaluation error terminates processing and causes a jump back to MAIN1. Otherwise, code is formed to store the integer value in the runtime "ACCUM3". The address expression is evaluated by EVALS. Again, evaluation errors cause termination of processing. A runtime call is formed to transfer a byte from the low half of ACCUM3 to the address. Processing returns to MAIN1.

Called by: MAIN
Calls: EVALS, STR1, RTBL
Jumps: MAIN1
Flags: FLPFLG

DEFS

This routine checks the symbol table for a variable. If found, it returns the symbol-table information from LKPSYM in the index register and accumulator B. Otherwise, the name is stored into the symbol table as a simple integer. The index register and accumulator B are returned as zero. The definition code generated is as follows:

```
        BRA *+4        branch around value
NAME    RMB 2          space for integer
```

Called by: LETPR, EVALS, READPR, EVADIM
Calls: LKPSYM, STOSYM, STR1, OUTNAM

Opcode Processing Routines

The address-processing routines all perform the same function and share common subroutines. They scan the operand field of a statement, and, based on the structure and values of the operands, generate the machine code for the instruction being processed. The processing routine also increments the location counter by the number of bytes the assembled instruction requires. Inherent Relative and Accumulator types require one byte, Indexed and Direct types require two bytes, Extended type instructions require three bytes, and Immediate types require either two or three bytes.

During Pass 1 no machine code is generated, but the operand field is scanned for errors, and the number of bytes the instruction requires is calculated.

During Pass 2 the machine code is generated and stored in the output file. Following Pass 1 and 2 processing, control passes to entry point MAIN1 in the main loop.

ADDR1

This routine processes the ADC, ADD, AND, BIT, CMP, LDA, ORA, SBC and SUB opcodes. The operand structure may be Immediate (2 bytes), Direct (2 bytes) Extended (3 bytes) or Indexed (2 bytes).

On entry register B contains the partial opcode: ADDR1 depending on the operand field, completes the opcode, generates the machine code for the address part of the instruction and outputs it to the output buffer in Pass 2.

Calls: ABRCK, ADRINT, INXCK, LCNAB2, LCNAB3, LCLCN, NSEVL, PRINTE, P2ERR
Jumps: MAIN1
Called by: MAIN
Flags: ABR, CMNFLG, IMMED, PASS, RELFLG
Pointers: ADR1, CUCHAR, ORBYA, ORBYB

ADDR2

This routine processes the STA opcode. The operand structure may be Direct (2 bytes), Extended (3 bytes), or Indexed (2 bytes).

On entry register B contains part of the opcode; ADDR2 depending on the operand field, completes the opcode, generates the machine code for the address part of the instruction and outputs it to the output buffer in Pass 2.

Calls: ABRCK, ADRINT, INXCK, LCLCN, LCNAB2, LCNAB3, NSEVL, NXTOK, PRINTE
P2ERR
Jumps: MAIN1
Called by: MAIN
Flags: ABR, CMNFLG, RELFLG
Pointers: ADR1, ORBYA, ORBYB

ADDR3

This routines processes the following opcodes: ASL, ASR, CLR, COM, DEC, INC, LSR, NEG, ROL, ROR, TST. The operand structure may be Accumulator (1 byte), Extended (3 bytes), or Indexed (2 bytes).

On entry register B contains part of the opcode; depending on the operand field, ADDR3 completes the opcode, generates the machine code for the address field of the instruction and outputs it to the output buffer.

Calls: ABRCK, ADRINT, INXCK, LCNAB1, LCN2, LCN3, NSEVL, NXTOK, PRINTE, P2ERR
Jumps: MAIN1
Called by: MAIN
Flags: ABR
Pointers: ORBYA, ORBYB

ADDR4

This routine processes the opcodes PSH and PUL. The operand structure is Accumulator (1 byte).

On entry register B contains the partial opcode. ADDR4 depending on the Accumulator in the operand field completes the opcode and outputs it to the output buffer in Pass 2.

Calls: ABRCK, ADRINT, LCLCN, LCNAB1, NXTOK, PRINTE
Jumps: MAIN1
Called by: MAIN
Flags: ABR
Pointers: ORBYA, ORBYB

ADDR5

This routine processes the opcodes CPX, LDS and LDX. The operand structure may be Immediate (3 bytes), Direct (2 bytes), Extended (3 bytes), or Indexed (2 bytes).

On entry register B contains the partial opcode. ADDR5, depending on the operand field, completes the opcode, generates the machine code for the address part of the instruction, and outputs it to the output buffer in Pass 2.

Calls: ADRINT, INXCK, LCLCN, LCN2, LCN3, NSEVL, NXTOK, PRINTE, P2ERR
Jumps: MAIN1
Called by: MAIN, ADDR6
Flags: CMNFLG
Pointers: ORBYA, ORBYB

ADDR6

This routine processes opcodes STX and STS. The operand structure may be Direct (2 bytes), Extended (3 bytes), or Indexed (2 bytes).

On entry register B contains the partial opcode. ADDR6, depending on the operand field, completes the opcode, generates the machine code for the address part of the instruction and outputs it to the output buffer in Pass 2.

Calls: ADRINT, NXTOK
Jumps: ADDR5A, ADDR5C
Called by: MAIN

ADDR7

This routine processes the JMP and JSR opcodes. The operand structure may be Extended (3 bytes), or Indexed (2 bytes).

On entry register B contains the partial opcode. ADDR7, depending on the operand field completes the opcode, generates the machine code for the address part of the instruction and outputs it ot the output buffer in Pass 2.

Calls: ADRINT, INXCK, LCLCN, LCN2, LCN3, NSEVL, NXTOK, PRINTE, P2ERR
Jumps: MAIN1
Called by: MAIN
Pointers: ORBYA

ADDR8

This routine processes the following opcodes: BCC, BCS, BEQ, BGE, BGT, BHI, BLE, BLT, BMI, BRA, BSR, BVC, BVS. The operand structure is Relative (2 bytes).

On entry register B contains the complete opcode. ADDR8 evaluates the operand field and calculates the relative offset that is to be the address part of the instruction and outputs it to the output buffer in Pass 2.

Calls: ADRINT, LCNCN, LCN2, NSEVL, NXTOK, PRINTE, P2ERR
Jumps: MAIN1
Called by: MAIN
Flags: PASS
Pointers: ADR1, 2, LC
TEMPS: LSAVE

ADDR9

This routine processes the following opcodes: ABA, CBA, CLC, CLI, CLV, DAA, DES, DEX, INS, INX, NOP, RTI, RTS, SBA, SEC, SEI, SEV, SWI, TAB, TAP, TBA, TPA, TSX, TXS, WAI. An operand structure does not exist--this is an inherent type instruction.

On entry register B contains the complete opcode, and the routine outputs this value to the output buffer in Pass 2.

Calls: ADRINT, LCLCN, OUTBIN, PRINTL
Jumps: MAIN1
Called by: MAIN
Flags: MCOUNT, PASS
Pointers: LCN

Utility Routines used by ADDR1 through 9

ADRINT

This routine initializes flags and variables used in the opcode and pseudop processing routines.

Calls: none
Called by: ADDR1-9, MAIN, POCMN, POEND, POENT, POEQU, POEXT, POFCB, POFCC, POFDB, POIFC, POMAC, PONAM, PONIFC, POPAG, PORMB, initialization, POSET
Flags: ABR, ADR1, 2, CMNFLG, ENTFLG, EXTFLG, EXTFLG IMMED, INDEX, LCN,

MCOUNT, ORBYA, ORBYB, POP, RELFLG
Pointers: OPCD

ABRCK

Checks to see what, if any, register is the first operand in the operand field of an instruction. The register is either A or B.

Calls: none
Called by: ADDR1-4
Flags: ABR

INXCK

Checks to see if an instruction is indexed.

Calls: NXTOK, PRINTE
Called by: ADDR1, 2, 3, 5, 7
Flags: INDEX

P2ERR

Prints Pass 2 errors. Some errors returned by the evaluation routine NSEVL are considered errors in Pass 2 but are not errors in Pass 1.

Calls: PRINTE
Called by: ADDR1, 2, 3, 5, 7, 8, POFCB, POFDB
Flags: PASS
Pointers: ADR1, 2

LCNAB1

Finishes up processing for 1-byte Accumulator type instructions.

Calls: OUTBIN, PRINTL
Called by: ADDR3, 4
Flags: ABR, MCOUNT, PASS
Pointers: LCN, OPCD, ORBYA, ORBYB

LCN2

Finishes up processing for 2-byte Indexed, Direct and Immediate type instructions.

Calls: OUTBIN, PRINTL
Called by: ADDR3, 5, 7, 8
Entry: LCN2A
Flags: CMNFLG, MCOUNT, PASS, RELFLG
Pointers: ADR2, LCN, OPCD, ORBYA, ORBYB

LCNAB2

Finishes up processing for 2-byte Register (A and B) Indexed, Direct and Immediate-type instructions.

Calls: none
Jumps: LCN2A
Called by: ADDR1, 2
Flags: ABR, PASS
Pointers: OPCD, ORBYA, ORBYB

LCN3

Finishes up processing for 3-byte Extended type instructions.

Calls: OUTBIN, PRINTL
Called by: ADDR3, 5, 7
Entry: LCN3A
Flags: CMNFLG, MCOUNT, PASS, RELFLG
Pointers: ADR1, ADR2, LCN, OPCD, ORBYA, ORBYB

LCNAB3

Finishes up the processing for 3-byte Register (A and B) Extended and Immediate-type instructions.

Calls: none
Jumps: LCN3A
Called by: ADDR1, 2
Flags: ABR, PASS
Pointers: OPCD, ORBYA, ORBYB

LCLCN

Adds LCN to LC (LC:=LC+LCN)

Calls: none
Called by: ADDR1-5, ADDR7, ADDR8, ADDR9, POEXT, POFCB, POFCC
Pointers: LC, LCN

Pseudo-op Processing

POCMN

This routine processes the CMN pseudop. During PASS 1 the name in the operand field is stored in the Symbol Table with:

```
RELOC. bit := off
COMMON bit := on
Value      := value of the Common LC. (CMNLC)
```

CMNLC is then incremented by the number of bytes allocated by the second

operand.

PASS 2: There is very little processing done in this pass other than setting up flags for the listing line.

Calls: ADRINT, LBLCK, LKPSYM, NSEVL, NXTOK, PRINTE, PRINTL, STOSYM
Called by: MAIN
Jumps: MAIN1
Flags: CMNFLG, MCOUNT, PASS, POP
Pointers: ADR1, ADR2, CMNLC, SYMPTR
Temps: CMNXS

POEND

This routine handles the END statement for STRUBAL+ and assembler. If POEND is called from pass 1 (assembler), this routine saves the final location counter (LC) and rewinds the input file for pass 2. Processing resumes at PASS2 in the main loop.

Processing for pass 2 continues at entry POEND1. If compiling (determined by the compiler flag CFLAG), the END line is passed to the assembly output file.

A check is made for unclosed blocks. (FOR-NEXT or WHILE-BLOCK.) An unclosed block is flagged as an error. There is no test for phasing errors in STRUBAL+ compilation, however, if this is assembly pass 2, this routine checks for phasing errors. (A different LC is saved in pass 2 than was saved in pass 1.) The listing process is completed here.

Beginning at entry ENDP3, this routine sets up and prints an alphabetical listing of the symbol table. The sorting algorithm repeatedly searches the symbol table, retaining a pointer (CBLOCK) to the smallest (first in alphabetical order) symbol name. If no symbol is found in the table, processing jumps to ENDP6. If a symbol is found, the sorting flag SORTF is set. One table entry is printed per search. As an entry is printed, a bit is set in its definition byte. This marks the entry so that it will be passed over on subsequent searches.

Printing a symbol table entry begins at SORT5. The 6-character name is printed, followed by a space. The address field is printed in decimal and hexadecimal form. In STRUBAL+, this would be either the line of definition or the first dimension of an array. In assembler, this is the address of the symbol. Finally, the symbol attributes are printed. STRUBAL+ uses the following set of attributes:

M	macro/function	(compile and assemble)
S	string	(compile only)
C	common	(compile and assemble)
N	entry point	(compile and assemble)
D	array	(compile only)
F	floating point	(compile only)
R	relocatable	(assemble only)
X	external ref.	(assemble only)

A multiply-defined symbol will be marked "redefined". A message giving the total error count and the common length is produced. The output file (if used) is end-filed and closed. The program returns to prompt for new options in the initialization section.

Called by: MAIN, RDLINE
Calls: ADRINT, LBLCK, REWIND, PASS2(main loop), STR1, PRINTE, PRINTL, SPACER, SYMMOD, COMPAR, OUTCHR, CVBTD, PDATA1, OUT4HS, CRLF, CLOSI, CLOSO
Jumps: MAIN1
Flags: CFLAG

POENT

This routine processes the ENT pseudop. PASS1: The statement is checked for syntax. PASS2: The symbol in the operand field is looked up in the symbol table. The ENT bit in the entry in the symbol table is then set. The symbol, the entry address, "R" and "N" are output to the output file providing linking information to the Linkage Editor.

Following Pass 1 and 2, processing control is passed to entry point MAIN1.

Calls: ADRINT, LBLCK, LKPSYM, NXTOK, OUTBIN, OUTBNR, PBLOCK, PRINTE, PRINTL
Called by: MAIN
Jumps: MAIN1
Flags: ENTFLG, MCOUNT, PASS, POP
Pointers: ADR1, ADR2, SYMPTR

PBLOCK

This routine is used by the POENT, POEXT and XREF routines to output the symbol name to the output file.

Calls: OUTBIN
Called by: POENT, POEXT, XREF
Pointers: SYMPTR
TEMPS: PBXS

POEQU

This routine processes the EQU pseudop. Pass 1: The operand is evaluated and stored in the symbol table entry associated with the label on the EQU statement. Pass 2: There is no processing done during this pass, other than printing the line of listing.

Following Pass 1 and 2 processing, control passes to entry point MAIN1 in the main loop.

Calls: ADRINT, NSEVL, NXTOK, PRINTE, PRINTL
Called by: MAIN
Jumps: MAIN1
Flags: LBFLG, MCOUNT, PASS, POP, RELFLG
Pointers: SYMPTR
Temps: EQUXS

POEXT

This routine processes the EXT pseudop. PASS1: The symbol in the operand field of the statement is stored in the symbol table with the EXT bit set and a value equal to the current value of the location counter. PASS2: A JMP instruction (3 bytes) is assembled, and output to the output file, along with the EXT indicator "X". This provides linking information to the Linking Loader. Following Pass 1 and 2 processing control is passed to entry point MAIN1.

Calls: ADRINT, LBLCK, LCLCN, LKPSYM, NXTOK, OUTBIN, OUTBNR, PBLOCK, PRINTE
PRINTL, STOSYM
Called by: MAIN
Jumps: MAIN1
Flags: EXTFLG, MCOUNT, PASS
Pointers: ADR1, ADR2, LCN, OPCD, SYMPTR

POFCB

This routine processes the FCB pseudop. Pass 1: During this pass POFCB simply increments the Location counter (LC). Pass 2: During this pass the LC is incremented as in pass 1 but the operand is evaluated and stored as a one byte value in the output buffer.

Following Pass 1 and 2 processing, control is passed to entry point MAIN1 in the main loop.

Calls: ADRINT, LCLCN, NSEVL, NXTOK, OUTBIN, PRINTE, PRINTL
Called by: MAIN
Jumps: MAIN1
Flags: MCOUNT, PASS, POP
Pointers: LCN

POFCC

This routine handles the assembler "Form-Constant-Characters" pseudo-operation and is used by QUOTPR to handle quoted character strings. Two quotes in a row result in a single quote being passed into the string.

Assembly pass 1 simply counts the number of characters in the string. Pass 2 moves the string of characters into the output file. Due to limitations in the output listing routine, only the first character is listed. The location counter reflects the proper number of characters.

The compiler pass (indicated by the compiler flag CFLAG) works like pass 2 of assembly, except that TPCHAR and WRTCHR are used to output list and object of the character string. Both quotes of a pair are passed through during compilation. Processing continues at MAIN1.

Called by: MAIN, QUOTPR
Calls: ADRINT, LCLCN, NXTOK, OUTBIN, PRINTE, PRINTL, TPCHAR, WRTCHR
Jumps: MAIN1
Flags: CFLAG, MCOUNT, PASS, POP

POFDB

This routine handles the assembler "Form-Double-Byte" pseudo-operation. It is also used by POFFP. It is similar to POFCB, except that two-byte values are used instead of single bytes.

Entry points FDBA, FDBD and FDBE are referred to by POFFP. FDBA is the error return. FDBD is the finish-up location. FDBE sets the flags MCOUNT and POP for listing. Processing returns to MAIN1.

Called by: MAIN, POFFP
Calls: ADRINT, LCLCN, NSEVL, NXTOK, OUTBIN, OUTBNR, PRINTE, PRINTL, P2ERR
Jumps: MAIN1
Flags: MCOUNT, PASS, POP, RELFLG

POFFP

This routine handles the assembler "Form-Floating-Point" pseudo-operation. It forms 9 bytes of object code in the STRUBAL+ floating-point format, based on the constant value in its operand. NXTOK and FSCAN form the constant value in buffer FTEMP. Subroutine PBLK2 is used to move the value into the output file. Only the first two bytes are listed. Entry points in POFDB are used.

Called by: MAIN
Calls: POFDB, ADRINT, NXTOK, PBLK2
Jumps: MAIN1
Flags: FLPFLG, PASS
Buffers: FTEMP

POFUNC

This routine processes STRUBAL+ functions (like assembler Macros). A dummy line is output and the function name is parsed from the source line. If this name is already in the symbol table an error exists. If not, the name is placed into the table and processing continues like a macro (see POMAC).

Calls: ADRINT, STR1, NXTOK, PRINTE, LKPSYM, STOSYM, XREF
Jumps: POMAC
Called by: MAIN
Flags: CMFLG

POIFC

This routine processes the conditional assembly pseudop IFC. The old value of the conditional-assembly flag CIFFLG is pushed onto the IF-stack and the operand of the pseudop determines the new value of CIFFLG. (Operand=0 sets CIFFLG=0; operand nonzero sets CIFFLG=FF.) Processing returns to MAIN1 when completed.

Calls: ADRINT, LBLCK, NXTOK, PRINTE, NSEVL, PSHIF, PRINTL
Called by: MAIN
Jumps: MAIN1
Flags: CIFFLG

POMAC

This routine handles STRUBAL+ functions and assembler Macros. The macro/function name is entered into the symbol table with the macro bit set. Subsequent lines are stored into the macro table until a MEND (for macros) or FEND (for functions) is encountered. The parameter "C" on the statement sets the comment flag CMFLG which causes comment lines to be stored into the definition of the macro/function. The definition's macro-table address is stored in the symbol value field for the macro/function name. An end-mark (17 hex) indicates the end of the definition in the macro table. Processing returns to MAIN1 on completion.

Calls: ADRINT, PRINTL, PRINTE, NXTOK, RDLINE, MACMOV, COMPAR
Called by: MAIN, POFUNC
Jumps: MAIN1
Flags: CMFLG, PASS, CFLAG, LBFLG

MACMOV

POMAC calls this subroutine to move a line of macro/function definition into the macro table.

Calls: PRINTE
Called by: POMAC

PONAM

This routine processes the NAM pseudop. The operand name is passed to the Linkage Editor as an Entry point followed by the size of the Common Block. Processing jumps to entries in POENT to finish up.

Calls: ADRINT, LBLCK, NXTOK, OUTBIN, OUTBNR, PRINTE
Jumps: POENT1, POENT3
Called by: MAIN
Flags: PASS
Pointers: CMNLC

PONIFC

This routine processes the NIFC (end conditional assembly) pseudop. The last value of the conditional assembly flag CIFFLG is pulled from the IF-stack and processing returns to MAIN1.

Calls: ADRINT, LBLCK, PULIF, PRINTL
Called by: MAIN
Jumps: MAIN1
Flags: CIFFLG

PSHIF

This subroutine pushes the value of CIFFLG onto the IF-stack.

Calls: none
Called by: POIFC

Flags: CIFFLG

PULIF

This subroutine pops the value of CIFFLG from the IF-stack.

Calls: none
Called by: PONIFC
Flags: CIFFLG

POPAG

This routine processes the PAG pseudop. During Pass 2 this routine issues a formfeed (0C hex) to the listing. Following Pass 1 and 2 processing, control is passed to entry point MAIN1 in the main loop.

Calls: ADRINT, LBLCK, SPACER
Called by: MAIN
Jumps: MAIN1
Flags: PASS
Pointers: LCOUNT

PORMB

This routine processes the RMB pseudop. Pass 1: The operand is evaluated and the Location Counter is incremented by this value. Pass 2: The LC is incremented by the value of the operand, and that number of zeros is stored in the output buffer.

Following Pass 1 and 2 processing, control passes to entry point MAIN1 in the main loop.

Calls: ADRINT, NSEVL, NXTOK, PRINTE, PRINTL, RMBOUT
Called by: MAIN
Jumps: MAIN1
Flags: MCOUNT, PASS, POP
Pointers: ADR1, ADR2, LC
Temps: LSAVE

RMBOUT

The PORMB uses this routine to output zero bytes to the output file. On entry ADR1 and 2 contain the number of bytes to be output.

Calls: OUTBIN
Called by: PORMB
Pointers: ADR1, ADR2
TEMPS: LSAVE

POSET

This routine processes the SET pseudop. The label of the line is checked in the symbol table. If it is not already there it is stored. The operand of the line is evaluated and that value is stored in the value entry for the symbol. The relocation and common bits are reset to reflect the type of operand. Processing returns to MAIN1.

Calls: ADRINT, PRINTE, NXTOK, LKPSYM, STOSYM, XREF, NSEVL, PRINTL
Called by: MAIN
Jumps: MAIN1

Comparison Operators

(EQPR, NEPR, GEPR, GTPR, LEPR, LTPR)

These routines set the index register to point to a branch code line. CHKBIN makes sure that there are enough arguments on the compiler stack. Processing for all comparisons then continues at CMPR.

Called by: EVALS
Calls: CHKBIN, CMPR

CMPR

This routine generates code for STRUBAL+ comparisons. The code formed is as follows:

```
     LDX ADDRESS 1          address of first argument
     STX ACCUM1             save it
     LDX ADDRESS 2          address of second argument
     JSR COMPAR             integer or floating compare runtime routine
     B-- *+7                branch line (BEQ,BNE,BGE,BGT,BLE,BLT)
     LDX #0                 return FALSE
     BRA *+5
     LDX #$FFFF             return TRUE
     STX ADDRESS 1          save result
```

The result of a comparison is always an integer. TRUE values are "all bits set" (-1) and FALSE values are "all bits clear" (0).

Called by: Comparison operators
Calls: FUNOP2, STAKDN, FUNOP, BINOP2, STR1, UNOP4
Flags: FLPFLG

RETPR

This routine handles the RETURN statement--forming an RTS assembler line. Processing return to MAIN1.

Called by: MAIN
Calls: STR1
Jumps: MAIN1

STOPPR

This routine handles the STOP statement by forming a runtime call. The runtime package will return control to the file system. Processing returns to MAIN1.

Called by: MAIN
Calls: .RTBL
Jumps: MAIN1

IFPR

This routine handles the IF statement. There are two basic forms of IF: the if-then-[else] and the if-statement form. IFPR first calls PRESCN to set the floating-point flag FLPFLG appropriately to the conditional expression. EVALS generates code which evaluates the expression. Any evaluation error causes processing to terminate, and the program returns to MAIN1. Otherwise, the next token after the expression is checked. If the token is "THEN", processing continues at IFPR2. Otherwise, processing continues at IFSTMT.

At IFPR2, a check is made for a valid label following THEN. If the label is not valid, processing terminates with an error message and the program branches back to MAIN1. If the label is valid, the following code is formed:

```
BEQ *+5          branch if expression false
JMP LABEL        if expression true, jump to LABEL
```

The program checks for a carriage return. If one is found, processing returns to MAIN1. If there is no carriage return, the program checks for the keyword ELSE. If not found, a syntax error is output and processing returns to MAIN1. If ELSE is found, the next token is checked for a valid label. An invalid label is a syntax error. A valid label causes the following code line to be formed:

```
JMP LABEL          jump to ELSE label
```

Processing returns to MAIN1.

The IF-statement form is handled at IFSTMT. The if flag-IFFLG-is set to produce a transfer label after the processing of the "statement" part of this line. A new compiler transfer label is produced and stored for this purpose. The following code line is generated:

```
BNE *+5          if expression false,branch to transfer label
JMP LABEL
```

A blank is output for the next code line, and processing jumps back into the MAIN loop to handle the keyword which follows the expression. This keyword cannot be another IF or a WHILE.

Called by: MAIN
Calls: PRESCN, EVALS, COMPAR, NXTOK, PRINTE, STR1, OUTCMP, OUTCR, TPCHAR, WRTCHR, MAIN (COMP2), XREF, OUTNAM
Jumps: MAIN1
Flags: IFFLG
Buffers: IFLBL

WHILPR

This routine handles the WHILE statement. It puts out a dummy first line, in case the WHILE statement has a label. A new compiler label is generated for the top of the WHILE-BLOCK. Another compiler label is formed for the BLOCK transfer point. The label is stacked on the block-stack. A check is made for available stack space, and if the stack is full a syntax error is output. The block stack (BLKSTR) holds up to 10 nested-transfer labels for FOR-NEXT and WHILE-BLOCK structures. If the stack is not full, the processing continues by calling PRESCN to set the floating-point flag FLPFLG to the mode of the expression in the WHILE statement. EVALS then generates code to evaluate the expression. The last line of this code will be a STX. WHILPR generates the following code:

```
BNE *+5            expression evaluates TRUE, so branch
JMP ENDLBL+3       otherwise, jump out of block
```

Processing continues at MAIN1.

Called by: MAIN
Calls: STR1, CVBTD, PRINTE, PRESCN, EVALS, OUTCMP
Jumps: MAIN1

PRESCN

This routine determines an expression's floating-point mode. It sets the floating-point flag FLPFLG appropriately. It starts by assuming that the expression is floating-point and searches the expression for tokens which would only be valid in one mode.

Strictly integer functions return with FLPFLG cleared. Strictly floating-point functions set FLPFLG before returning. Variable names set the flag based on their definition in the symbol table. Hexadecimal numbers can only be integer. The two flags used in NXTOK (DECFLG and SIGDIG) determine whether a strictly floating-point constant was encountered. PRESCN scans along the line until it finds either an explicitly defined mode or reaches the end of the line. If it reaches the end of the line, the expression is assumed floating-point.

A bad token causes PRESCN to quit. The token pointers are restored to the beginning of the expression before PRESCN returns to allow EVALS to process the expression properly.

Called by: IFPR, WHILPR, SSETUP, CONVPR, WRITPR, PUSHPR, POPPR
Calls: NXTOK, MNLKP, LKPSYM
Flags: FLPFLG, DECFLG, SIGDIG

BLOKPR

This routine handles the BLOCK statement. It first puts out a dummy line of code in case the BLOCK statement has a label. It then pulls the WHILE-BLOCK label from the block stack (BLKSTR). If there is no label on the stack, a syntax error exists. Processing returns to MAIN1.

The label on the WHILE corresponding to this BLOCK is one less than the present label. BLOKPR generates a code line to jump back to the start of the WHILE block. The WHILE code has a jump to just beyond this line in the BLOCK code.

Called by: MAIN
Calls: STR1, PRINTE, CVBTD
Jumps: MAIN1

ONPR

This routine handles the ON statement. It generates code to work with branch tables formed by the GOTO or GOSUB statements. The branch table has a label at its top and at it bottom. ONPR forms these labels. The expression is evaluated by EVALS; UNOP2 puts the integer result in the pseudo-accumulator in the runtime package. ONPR generates the following code:

```
LDX #TOPLBL          point to top label
LDA A VALUE          get low byte of expression value
DEC A                count down
BGT *+4              not done yet?
JMP 0,X              jump through branch table
INX
INX
INX                  5 bytes per table entry
INX
INX
CPX #BOTTOM          at bottom of table?
BEQ *-10             if so, jump to last entry
BRA *-15             otherwise, keep counting down
```

Processing returns to MAIN1. The expression is evaluated as an integer, and only the low-order byte is used.

Called by: MAIN
Calls: EVALS, UNOP2, CVBTD, STR1
Jumps: MAIN1
Flags: UBFLAG

CALLPR

This routine handles the CALL statement. This statement builds a call to an external assembly-language subroutine. Parameter's addresses are passed. The subroutine can be called several times in the program. CALLPR saves the subroutine name in the symbol table. It generates an external reference for the first call to the subroutine. The parameters are optional, but if present, they must be simple integer variables, array names or floating-point variables. The names' addresses are passed. CALLPR generates the following code:

```
      JSR SUB1            call subroutine
      BRA LABEL           if call has parameters, branch around them
      FDB Param1          address of first parameter
      FDB Param2          address of second parameter
      ....                addresses
      ....                address of last parameter
      FDB $FFFF           end marker
      (EXT SUB1)          external reference on first call of "SUB1"
LABEL EQU *               transfer point
```

CALLPR uses GOTOPR entry GTPR1 for error conditions. Processing returns to MAIN1.

Called by: MAIN
Calls: OUTCMP, NXTOK, OUTCR, GOTOPR(GTPR1), CVBTD, QPR1, LKPSYM, STOSYM
OUTNAM
Jumps: MAIN1

PROCPR

This routine handles the PROCEDURE statement. Code is generated as follows:

```
                      ENT Procedure-name
     Procedure-name   JSR runtime          move address from call
                      BRA .Label
     Parameter name-1 RMB 2                space for call addresses
     Parameter name-2 RMB 2
     ................ ......               further parameters
     ................ ......
                      FDB $FFFF            end marker
     .Label           EQU *
```

Processing resumes at MAIN1.

Called by: MAIN
Calls: OUTCMP, NXTOK, PRINTE, OUTNAM, OUTCR, STOSYM, XREF, QPR1, CVBTD,
LKPSYM, RTBL, STR1

GOTOPR

This routine handles the GOTO statement. It stores a pointer to a jump statement in OPPD and continues with GTPRO.

Called by: MAIN
Calls: GTPRO
Buffer: OPPD

GOSBPR

This routine handles the GOSUB statement. It stores a pointer to a jump-subroutine statement in OPPD and continues with GTPRO.

Called by: MAIN
Calls: GTPRO
Buffer: OPPD

GTPR0

This routine is called by GOTOPR and GOSBPR to generate code for the various forms of GOTO and GOSUB statements.

GTPR0 checks the type of the first token after the keyword. If the token is a name, processing continues at GTPR2. This is the computed form, to be used with the ON statement. If the token is an "at" sign (@), the processing is done for an indirect GOTO or GOSUB.

EVALS evaluates the integer expression following the "@." This results in the index register holding the value. The following code line is formed:

```
JMP 0,X                    for GOTO
JSR 0,X                    for GOSUB
```

depending on the pointer passed in OPPD. Processing resumes at MAIN1. If the token is none of the above, this is a syntax error. GTPR0 prints an error message and returns to MAIN1. Syntax errors in CALL statements also use this entry (GTPR1).

At GTPR2, code is generated for a branch table. A dummy line is output in case the line has a label. The computed-GOTO flag GFLAG is set to indicate that this mode is in use. A compiler label (set up by ON) is output. Processing now falls into GTPR3.

At GTPR3, processing outputs either a jump or a subroutine call to the name in the line. If the computed-GOTO flag GFLAG is not set, processing terminates and returns to MAIN1. If GFLAG is set, processing next looks for a comma or right parenthesis. If neither is found, and end transfer label is output, based on the value stored by the ON statement. Processing returns to MAIN1. If the comma was found, this indicates that more branch names follow. A branch to the end label is output and processing loops back to GTPR3 for another name. A sample of each type of GOTO code follows: (GOSUB code just substitutes "JSR" for "JMP").

```
STRUBAL+-- LABEL  GOTO  L2          Assembly-- LABEL   JMP L2

        -- LABEL  GOTO @EXPR                  LABEL   LDX #EXPR
                                               JMP 0,X

        -- LABEL  GOTO (L1,L2,L3)              LABEL   EQU *
                                               .label JMP L1
                                                      BRA .endlbl
                                                      JMP L2
                                                      BRA .endlbl
                                                      JMP L3
                                             .endlbl EQU *
```

Called by: GOTOPR, GOSBPR, CALLPR
Calls: NXTOK, EVALS, PRINTE, OUTCMP, CVBTD, STR1, XREF, OUTNAM
Jumps: MAIN1
Buffers: OPPD
Flags: GFLAG

FNDDIM

This routine evaluates array elements. It is also used to find the address an expression. It first generates the code:

```
LDX #arrayname
STX address            put base pointer into runtime stack
```

The "immediate indicator" (#) is omitted for parameter names. If the next token is a left parenthesis, processing continues with the indices. If the token is not a left parenthesis, the GETAD flag GAFLG is checked. If the flag is set--indicating a get-address call--processing is done. If the flag is not set--indicating an array reference--the lack of a left parenthesis is a syntax error and processing must quit.

After finding a left parenthesis, FNDDIM stacks the runtime stack address, the dimension information about the array taken from the symbol table and the floating-point flag on the processor stack. The dimension flag DIMFLG is set. EVALS is called at the recursive entry point EXPR1 to evaluate the first index. The information previously stacked in the processor stack is recovered. The next token is checked. If it is a comma, processing must evaluate a second index at FND3. If it is a right parenthesis, processing must finish up at FNDDONE. If it is neither of these, this is a syntax error.

At entry point FND3, the dimension data from the symbol table is checked to see that the array has a second dimension. If it does not, this is a syntax error. If it does, the runtime-stack pointer, etc. are stacked on the processor stack and EVALS is called at its recursive entry point EXPR1 to evaluate the second index expression. The following code is generated:

```
DEX                adjust origin of array (second index)
STX ACCUM1         save in runtime
LDX DIMENSION      first dimension data from symbol table
JSR MULT           runtime call to multiply index by dimension
STX ACCUM1         save in runtime
LDX address        runtime address of first index
JSR ADD            runtime call to add first index
```

to handle the second index and to combine it with the first index, whose value was on the runtime stack.

Entry point FNDDONE is reached for a single and double-dimensioned arrays. It generates the following code:

```
DEX                adjust origin of array
STX ACCUM1         save in runtime
LDX #n             n=2 for integer...n=9 for floating point
JSR MULT           adjust for size of array element
STX ACCUM1         save in runtime
LDX address        runtime address of base
JSR ADD            add indexing value to base
```

which generates the final array address. Since strings have a single byte per data entry, lines 3, 4 and 5 are omitted for strings.

Called by: EVALS, EVADIM
Calls: OUTCMP, WRTCHR, TPCHAR, OUTCR, CVBTD, STR1, PRINTE, EVALS(EXPR1), STAKDN, RTBL, OUTNAM

Flags: DIMFLG, FLPFLG, GAFLG

EVADIM

This routine sets up for FNDDIM if it is not called from EVALS. The stacks and necessary pointers are initialized. Checks are made that the present token is pointing to a valid symbol name. The symbol table pointers are set by a call to DEFS. EVADIM saves the floating-point flag state on the processor stack.

Called by: LETPR, STRDIM, GETADR, READPR
Calls: GCHRTB, MNLKP, PRINTE, DEFS, FNDDIM
Flags: DSTFLG, FLPFLG, DIMFLG

LBLCK

This routine is used by certain of the pseudop processing routines to check to see if there was a label for that source line. If there is a label, it is deleted from the Symbol table, and an error message printed.

Calls: DELSYM, PRINTE
Called by: POCMN, POEND, POENT, POEXT, POIFC, PONAM, PONIFC, POPAG
Flags: LBFLG, PASS

OUTBIN

This routine puts machine code into the output file. On entry, register B contains a byte of machine code. OUTBIN translates this byte into two bytes of ASCII hexadecimal characters and calls OUTB to output the two bytes to the object file.

Calls: OUTB, OUTHL, OUTHR
Called by: ADDR9, LCNAB1, LCN2, LCN3, PBLOCK, POENT, POEXT, POPFCB,
POFCC, POFDB, PONAM, RMBOUT, XREF
Flags: OPTNS

OUTBNR

This routine stores the single ASCII character in register B into the output file. It is identical to TPCHAR.

Calls: OUTB
Called by: POENT, POEXT, POFDB, PONAM, XREF
Flags: OPTNS

OUTHL

This routine takes the byte in the A accumulator and converts the left four bits into an ASCII character. It uses OUTHR after shifting the accumulator right four times.

OUTHR

This routine takes the byte in the A accumulator and converts the right four

bits into an ASCII character.

Called by: PRINTE, EVALS(FCOUT), output routines OUTHLL and OUTHRR
Calls: none

Assorted Output Routines

These routines produce console output for STRUBAL+. Only one routine is required of the system; OUTEEE must take the character from the A accumulator and print it. OUTEEE must preserve accumulator B and the index-register contents. All other output functions required in are handled within this set of routines.

CRLF: print a carriage-return and a linefeed

PDATA1: print a string of characters. The index register holds the address of the string. PDATA1 prints until an end-string character (04 hex) is encountered.

OUT2H: print the hexadecimal contents of a byte whose address is in the index register. Index register is incremented.

OUT2HS: call OUT2H followed by a call to OUTS.

OUT4HS: print the hexadecimal contents of a 16-bit quantity whose address is in the index register. Followed by a call to OUTS. Index register is incremented twice.

OUTS: print a space (blank)

OUTHLL: print left nibble (4 bits) of a byte in the A accumulator

OUTHRR: print right nibble (4 bits) of a byte in the A accumulator

OUTCHR: used by the other routines to print the contents of the A accumulator. Calls OUTEEE for the actual output.

Mass-storage Linkages

The compiler requires one input and one output mass-storage medium. The driver software for these devices links through the following:

TABLES: last entry in the driver package. Symbol table will begin here.
CLOSI: close input file
CLOSO: close output file
MONTOR: system monitor return point
GETB: get a byte from the input file into the A accumulator
OUTB: put a byte into the output file from the A accumulator
OPENI: open input file
OPENO: open output file
REWIND: rewind input file

The drivers use the following console output routines from STRUBAL+: PDATA1, INEEE, and CRLF.

Program Linkage

Linking pseudo-op instructions are used to provide a means of communication between a main program and its subroutines or among several subprograms that are to be linked together to run as a single program.

COMMON

CMN SYMBOL,operand

CMN reserves a block of storage locations that may be used in common by several programs. Each SYMBOL identifies a segment of the block for the subprogram in which the CMN statement appears. The operand is the length of the related segment.

Any number of CMN statements may appear in a subprogram. Storage locations, in common, are assigned contiguously. The length of the block is equal to the sum of the length of all segments names in CMN statements in the subprogram.

To refer to the Common block other subprograms must also include a CMN statement. The segment names and lengths may be the same or they may differ. Regardless of the names and lengths specified in the separate subprograms, there is only one Common block for the combined set. It has the same relative origin: the content of the nth byte of Common storage is the same for all subprograms.

EXAMPLE

```
NAM  PROG1
CMN  AAA,5      ALLOCATE 5 BYTES OF COMMON
CMN  BBB,10     ALLOCATE 10 BYTES OF COMMON
CMN  CCC,10     ALLOCATE 10 BYTES OF COMMON
.
.
.
LDA A BBB+1     LOAD BYTE 2 OF SEGMENT BBB
.
.
END


NAM PROG2
CMN DDD,2       ALLOCATE 2 BYTES OF COMMON
CMN EEE,2       ALLOCATE 2 BYTES OF COMMON
CMN FFF,1       ALLOCATE 1 BYTE OF COMMON
CMN GGG,20      ALLOCATE 20 BYTES OF COMMON
.
.
LDA A GGG+1     LOAD BYTE 2 OF SEGMENT GGG
.
.
END
```

Organization of the COMMON block

```
PROG1          PROG2          COMMON
NAME           NAME           BLOCK
AAA            DDD            BYTE 1
                              BYTE 2
               EEE            BYTE 3
                              BYTE 4
               FFF            BYTE 5
BBB            GGG            BYTE 6
                              BYTE 7
                               .
                               .
                              BYTE 15
CCC                           BYTE 16
                              BYTE 17
                               .
                               .
                               .
                              BYTE 25
```

The LDA A instruction in both subroutines refers to the same location in Common--BYTE 7.

The segment names that appear in the CMN statements can be used in the operand fields of EQU, FDB or any memory reference statement; they may not be used as labels elsewhere in the program. All references to Common are relocatable.

The user establishes the origin of the Common block when the Linking Loader is executed.

Two or more subprograms may declare Common blocks that differ in size.

ENTRY

```
ENT    SYMBOL
```

ENT defines entry points to the program or subprogram. SYMBOL is an assigned label for some statement in the program. Entry points allow another subprogram to refer to this subprogram. All entry points must be defined in the program.

EXTERNAL

```
EXT    SYMBOL
```

EXT designates labels in other subprograms that are referenced in this subprogram. SYMBOL must be defined as an ENT in some other subprogram.

EXAMPLE OF ENT AND EXT
++++++++++++++++++++++

```
          NAM PROG1
          ENT SUB1          DEFINE SUB1 AS AN ENTRY POINT
          ENT SUB2          DEFINE SUB2 AS AN ENTRY POINT
*
SUB1      LDX #$0000
           .
           .
          RTS
*
SUB2      LDA B#'C
           .
           .
          RTS
          END

          NAM PROG2
          EXT SUB1          DEFINE SUB1 AS EXTERNAL
          EXT SUB2          DEFINE SUB2 AS EXTERNAL
*
          JSR SUB1          CALL SUB1 IN PROG1
          JSR SUB2          CALL SUB2 IN PROG2
           .
           .
          END
```

SUB1 and SUB2 are referenced in PROG2 but are actually locations in PROG1.

The CMN pseudop provides data communication and the EXT and ENT pseudops are provided to allow control communication between separately assembled or compiled subprograms. These modules can then be linked together to form a single program to be executed as a unit.

LISTING OUTPUT FORMAT

Program fields have the format:

COLUMNS	CONTENT
1-6	Line number generated by STRUBAL+
7	Blank
8	Blank
9-12	Location (hexadecimal)
13	Blank
14-15	Opcode (hexadecimal)
16	Blank
17-20	Operand; either 2 or 4 characters (hexadecimal)
21	Blank
22	Type indicator; R,C,X,N
23	Blank
24-72	First 49 characters of the source statement

Lines consisting entirely of comments are listed:

1-6	Line number
7	Blank

8-24	Blanks
24-72	First 49 characters of the source statement

A Symbol Table listing has the following format:

Columns	CONTENT
1-6	Symbol name
7	Blank
8-13	Value in decimal
14	Blank
15-18	Value in hexadecimal
19	Blank
20	Symbol Type: R-Relocatable M-Macro N-Entry X-External C-Common

RUNTIME PACKAGES

The runtime packages provide sets of frequently-used subroutines which are called by STRUBAL+ programs. These subroutines overcome deficiencies in the processor instruction set and provide useful functions. STRUBAL+ is supplied with four runtime packages. Only those packages whose routines are actually being used need be loaded.

RUNTIM

This is the first and most important of the runtime packages. It will be needed by every STRUBAL+ program. It provides integer arithmetic, logical, input-output, error-trap and other functions. It also contains the linkage routine.

Subroutines in RUNTIM

.RTBL

This routine links STRUBAL+ programs to the runtime routines. STRUBAL+ programs call .RTBL to access runtime subroutines. Immediately following the call, the STRUBAL+ program has a byte with an offset value. .RTBL uses this offset to find the desired runtime routine in BRTABL. .RTBL also modifies the subroutine return on the stack to bypass the offset byte on return from the desired runtime subroutine. Since STRUBAL+ frequently uses the index register to pass arguments to runtime routines, it is saved by .RTBL.

Called by: STRUBAL+ program
Calls: all other runtime subroutines
Tables used: BRTABL

BRTABL
---

routine	offset	routine	offset	routine	offset
NOT	00	ANDZ	03	EXOR	06
OR	09	ADDZ	12	SUBZ	15
MULT	18	DIV	21	COMPAR	25
ABS	27	SIGN	30	MOD	33
EXP	36	RAND	39	LFEED	42
INPUT	45	OUTPUT	48	QOUT	51
STRIN	54	SPACES	57	LEN	60
VAL	63	ASCII	66	MID	69
XFER	72	DELAY	75	STRCMP	78
PEEK	81	POKE	84	FPRINT	87
FREAD	90	FIX	93	FLOAT	96
FABS	99	FSIGN	102	FNEG	105
FCOMPR	108	FTAREG	111	FTCREG	114
FFDREG	117	FADD	120	FSUB	123
FMUL	126	FDIV	129	SIN	132
COS	135	TAN	138	ASIN	141
ACOS	144	ATAN	147	LOG10	150
NATLOG	153	TENEXP	156	FEXP	159
XTOY	162	ZMOVER	165	FVAL	168
FASCII	171	OPEN	174	CLOSE	177
REWIND	180	IREAD	183	FLREAD	186
SREAD	189	IWRITE	192	FWRITE	195
QWRITE	198	SRECRD	201	FSPACE	204
FLFEED	207	DIGSET	210	MOVER	213
PUSHI	216	PUSHF	219	PUSHS	222
POPI	225	POPF	228	POPS	231
CURIOB	234	STOP	237	STROUT	240
SWRITE	243	GETB	246	OUTB	249

*SAMPLE RUNTIME CALL AS CODED IN A STRUBAL+ PROGRAM

```
        JSR .RTBL
        FCB 39        CALL RAND ROUTINE
```

LOGICAL ROUTINES

This set of subroutines perform the logical functions provided by STRUBAL+. They all work on 16-bit values and are applied bit-wise. In STRUBAL+, a zero value is false and a non-zero value is true. All routines which require two inputs take one argument in the common area "ACCUM1" and the second argument in the index register. Those routiness which require only one argument use only the index register.

NOT

This routine performs unary negation on a 16-bit value in the index register. The result is returned in the index register.

Called by: .RTBL
Calls: none

ANDZ

This routine forms the binary "and" of its 16-bit arguments. Auxiliary routines SETUP and FINISH are used to setup the registers on entry and to return the value to the index register on exit.

Called by: .RTBL
Calls: SETUP, FINISH

EXOR

This routine forms the binary "exclusive-or" of its two 16-bit arguments. SETUP and FINISH are used (see ANDZ).

Called by: .RTBL
Calls: SETUP, FINISH

OR
--

This routine performs the binary "or" of its two 16-bit arguments.

Called by .RTBL
Calls: SETUP, FINISH

SETUP

This routine moves the 16-bit value from the common area "ACCUM1" to the A and B accumulators. It also saves the index register into "ACCUM2".

Called by: ANDZ, EXOR, OR, ADDZ, SUBZ
Calls: none

FINISH

This routine moves the 16-bit value in the A and B accumulators into the index register.

Called by: ANDZ, EXOR, OR, ADDZ, SUBZ, PEEK
Calls: none

INTEGER ARITHMETIC ROUTINES

This set of routines performs 16-bit two's-complement arithmetic. The calling format is the same as that used by the logical functions. Overflow conditions are trapped.

ADDZ

This routine adds two 16-bit two's-complement values.

Called by: .RTBL, DECIM
Calls: SETUP, FINISH, OVERR

SUBZ

This routine subtracts two 16-bit two's-complement values.

Called by: .RTBL
Calls: SETUP, FINISH, OVERR

MULT

This routine multiplies two 16-bit two's-complement values. It first converts both values to positive numbers and computes the sign of the result. An unsigned-multiply is performed and the final sign is restored. The setup routines SETUPS (with sign) and FINISHS (with sign) are used.

Called by: .RTBL, FIX, DECIM
Calls: SETUPS, FINISHS, OVERR

DIV

This routine divides two 16-bit two's-complement values. SETUPS and FINISHS compute the sign of the result and to make both arguments positive. Attempts to divide-by-zero are trapped. The remainder of the division is stored at the location "COUNT".

Called by: .RTBL
Calls: SETUPS, FINISHS, DIVERR

SETUPS

This routine performs the setup of registers like SETUP. It also converts both input values to positive numbers using NEGATE if necessary. The exclusive-or of the signs of the input values is stored in SIG.

Called by: MULT, DIV
Calls: NEGATE

FINISHS

This routine finishes the return linkage like FINISH. It also uses the sign information from SIG to put the proper sign on the result.

Called by: MULT, DIV
Calls: NEGATE, OVERR

NEGATE

This routine changes the sign of a 16-bit two's-complement value whose address is contained in the index register upon entry. It first inverts the value's bits, then adds one.

Called by: SETUPS, FINISHS, FIX, FLOAT, DECIM
Calls: none

ABS

This routine converts a 16-bit two's-complement value in the index register to a positive value.

Called by: .RTBL
Calls: NEGATE

SIGN

This routine returns the signum of the 16-bit two's-complement value in the index register. If the value is zero then zero is returned, if the value is negative, -1 is returned, and if the value is positive, +1 is returned.

Called by: .RTBL
Calls: none

MOD

This routine returns the remainder of a division. It uses DIV.

Called by: .RTBL
Calls: DIV

EXP

This routine raises an integer to an integer power. The algorithm is repeated multiplication. An exponent of zero returns +1. A negative exponent is trapped, since a negative exponent would always return a zero result in integer arithmetic.

Called by: .RTBL
Calls: MULT, NEGERR

COMPAR

Integer comparison cannot be done with a simple compare-index (CPX) instruction with a 16-bit two's-complement values. The condition codes are not always correctly set by the single instruction. STRUBAL+ performs the comparison by an actual subtraction.

This routine is called to compare two integer values. It is identical to SUBZ.

Byte Input-Output Routines

PEEK

This routine returns the value of a byte whose address was passed in the index register. The value returned is a 16-bit integer, whose upper 8 bits are zero. The value is returned in the index register.

Called by: .RTBL
Calls: FINISH

POKE

This routine stores a byte into the address which was contained in the index register. The byte is found in the low-order byte of the common area ACCUM3.

Called by: .RTBL
Calls: none

Random-number Generation

RAND

This routine generates a 16-bit two's-complement value which varies in a pseudo-random manner with each call. The argument is ignored if it is non-zero.

If the argument is zero, the sequence is re-initialized. The shift-with-feedback algorithm is used. The value is returned in the index register.

Called by: .RTBL
Calls: none

String Comparison

STRCMP

This routine compares two character strings. The address of the first string is passed in common area ACCUM4 The second string's address is passed in the index register. Each string's bytes are compared, one at a time, until either a difference is found or one of the strings has the 04 end-string value. The result of the comparison is in the condition codes.

Called by: .RTBL
Calls: none

PUSHI

This routine pushes an integer (2-byte) value onto the processor stack. The value's address is passed in the index register. The runtime calls's return address is removed from the stack and saved in ACCUM1. The return is accomplished by an indexed jump through the saved value.

Called by: .RTBL
Calls: none

PUSHF

This routine pushes a floating-point (9-byte) value into the processor stack. The address of the value is pased in the index register. The return address of the runtime call is removed from the stack and saved in ACCUM1. The return is accomplished by an indexed jump through the saved value.

Called by: .RTBL
Calls: none

PUSHS

This routine pushes a string value onto the processor stack. The string's address is passed in the index register. The string is scanned until its end-string (hex 04) is found, then its characters are pushed onto the stack in reverse order, starting with the end-string. The return address of the runtime call is removed from the stack and stored in ACCUM1. The return is accomplished through an indexed jump using the saved value.

Called by: .RTBL
Calls: none

POPI

This routine pops an integer (2-byte) value from the processor stack. This value is stored at the address passed in the index register. The runtime call's return address is removed from the stack and stored in ACCUM1. The return is accomplished by and indexed jump through this stored value.

Called by: .RTBL
Calls: none

POPF

This routine pops a floating-point (9-byte) value from the processor stack. The value is stored at the address passed in the index register. The return address of the runtime call is removed from the stack and stored in ACCUM1. The return is accomplished by an indexed jump through this stored value.

Called by: .RTBL
Calls: none

POPS

This routine pops a string of characters from the processor stack. The characters are stored at the address passed in the index register. An end-string character (hex 04) terminate the process. The runtime return address is removed from the processor stack and stored in ACCUM1. The return is accomplished by an indexed jump through this stored value.

Called by: .RTBL
Calls: none

Floating-point Move Utility Routine

ZMOVER

This routine moves a 9-byte floating-point value from the address contained in the index register to the address contained in the common area ACCUM1.

Called by: .RTBL
Calls: MOVER

STOP

This routine returns control from a STRUBAL program to the operating system. The program jumps to the system "warmstart" entry at MONTOR.

Called by: .RTBL, INEDIT, DIVERR, ILLERR
Calls: MONTOR

Input-Output Routines

These routines perform input and output to the system console and files. All I/O is done in ASCII code. A system routine is assumed which takes the contents of the A accumulator and outputs it to the system console. A similar routine is assumed for input.

OUTHL

This routine outputs the left nibble of a byte. The algorithm is to shift right and use OUTHR. The byte is in the A accumulator.

Called by: MPRINT, FPRINT
Calls: OUTHR

OUTHR

This routine outputs the right nibble of a byte. The byte is in the A accumulator.

Called by: MPRINT, FPRINT, OUTHL

Calls: OUTEEE

OUTS

This routine outputs a space (hex 20).

Called by: SPACES, INEDIT, FPRINT, STROUT
Calls: OUTEEE

INEEE

This routine performs input from either the console or files. If the file-read flag is set, then the end-file common location .EOF is cleared and the file-input routine GETB is called. Nulls are skipped and end-file is checked. If an end-file is found, the .EOF is set. For console input, the system routine is called. The character is returned in the A accumulator.

Called by: INEDIT, SREAD
Calls: GETB, system routine INEEE
Flags: FRDFLG

OUTEEE

This routine performs output from files or the system console, depending on the flag FWRFLG. It is also used to put characters into strings. If the string-mode flag "FLAG" is set, then the character in the A accumulator is stored at the address in the string pointer "XPOINT".

Called by: PDATA, INEDIT, ASCII, FPRINT, STROUT
Calls: OUTB, system routine OUTEEE
Flags: FWRFLG, FLAG

PDATA CM ---

This routine outputs a string of characters. And end-string (hex 04) ends the string. The address of the string is passed in the index register.

Called by: PDATA1, LFEED, PERR, XFER, FPRINT, TRUNER, QOUT
Calls: OUTEEE

PDATA1

Clears the file and string mode flags and calls PDATA.

Called by: file drivers
Calls: PDATA
Flags: FWRFLG, FLAG

CRLF

This routine outputs a linefeed/carriage return to the console.

Called by: file drivers
Calls: LFEED

Flags: FLAG

OUT4H

This is a system routine which outputs four hexadecimal characters from the address contained in the index register.

Called by: PERR
Calls: none

LFEED

This routine outputs a carriage return/linefeed to a string, file, or string.

Called by: CRLF, INEDIT, PERR, TRUNER, FLFEED
Calls: PDATA
Flags: FWRFLG

INEDIT

This routine is a single line editor for input. It has an 80-character buffer which holds characters as they are typed in. In file-mode, the characters come from the file rather than the console. In console-input mode a prompt is issued (>) to the operator. Four special commands are decoded. A control-C input returns control to the system. A control-U input flushes the line buffer and prompts for a new input line. A backspace input causes the last character input to be dropped, and a backslash and the dropped character to be echoed. A carriage-return input terminates the input to the line. If the buffer is overfilled, or an attempt is made to drop more characters than are in the buffer, an error message is given and the operator prompt is given again.

Called by: INPUT, STRIN, FREAD
Calls: OUTS, INEEE, LFEED, STOP, BADERR, OUTEEE
Flags: FLAG, FRDFLG

INPUT

This routine handles integer inputs. INPUT gets a line of data by calling INEDIT. Leading blanks are flushed. It checks for a dollar-sign ($) as a leading character. If present, the dollar-sign indicates a hex value. If an error is found in the input line, the operator is prompted to try again.

Called by: .RTBL, IREAD
Calls: INEDIT, BADERR, DECIM, CVHB
Flags: FDRFLG, RTYFLG

CVHB

This routine converts a string of ASCII characters representing a hexadecimal value into a 16-bit twos-complement value which is returned in the index register. Up to four characters may be in a value.

Called by: INPUT
Calls: CVHBS, TRUNER

CVHBS

This routine converts an ASCII character into a hexadecimal value. If the character is not a valid hexadecimal digit, then an error message is given and the input may be redone.

Called by: CVHB
Calls: BADERR, INPUT
Flags: RTYFLG

DECIM

This routine handles signed-decimal input of integer values. The maximum value is 32767. Up to five digits and a sign may be input. Errors are trapped and may be retried. The value is returned in the index register.

Called by: INPUT
Calls: INPUT, TRUNER, BADERR, MULT, ADDZ, NEGATE
Flags: RTYFLG, SIG2

OUTPUT

This routine outputs integer values to files, strings, or the console depending on the settings of the flags FLAG and FWRFLG. If entered at the entry-point OUTPUT, FLAG and FWRFLG are cleared and the output goes to the console. If entered at OUTPUT+3, FWRFLG is set and output goes to the current file specified in CURIOB. If entered at OUTSS, both flags are set and output is to a string.

All forms of output first convert the integer value passed in the index register to a floating-point value in the buffer FTEMP. This value is then output by FPRINT.

Called by: .RTBL, IWRITE, ASCII
Calls: FLOAT, FPRINT
Flags: FLAG, FWRFLG
Buffers: FTEMP

ERROR MESSAGE ROUTINES

All runtime errors give an English message and most report the address of the calling statement at the time of the error. Some error routines attemped error "fixups", others trap to the system.

PERR

This is the main error-reporting routine. It prints the error message and finds the calling address by looking two return-addresses back in the stack.

Called by: DIVERR, NEGERR, BADERR, OVERR, LENERR, ILLERR, PARAM, DIGERR
Calls: LFEED, PDATA, OUT4H
Flags: FLAG, FWRFLG

DIVERR

This is the error trap for attempts to divide by zero. The program traps to the system.

Called by: DIV, FDIV
Calls: PERR, STOP

NEGERR

This is the error trap for integer exponentiation with a negative exponent. It leaves the arguments unchanged.

Called by: EXP
Calls: PERR

BADERR

This is an error trap for bad input. It may allow a retry.

Called by: INPUT, INEDIT, CVHBS, DECIM, FREAD
Calls: PERR

OVERR

This is the error trap for arithmetic overflows. The largest possible value is returned in place of the overflow value.

Called by: ADDZ, SUBZ, MULT, FINISHS, DIV, FIX
Calls: PERR

TRUNER

This is the error trap for inputs which are too long. The input line has been truncated. No address is returned.

Called by: CVHB, DECIM
Calls: LFEED, PDATA

LENERR

This is the error trap for bad string lengths. A length of zero is returned.

Called by: LEN, MID
Calls: PERR

ILLERR

This is the eror trap for illegal arguments in a function call. This routine returns to the system.

Called by: ASIN, LOG10
Calls: PERR, STOP

STRUBAL+ FILE FORMAT

STRUBAL+ sequential files are composed of strings of ASCII characters. Any output which could be printed, or any input which could be keyed in, may be stored and recovered from a file. Each output item (integer, string, floating-point) will be stored in the file with the carriage-return appended. This carriage-return character will be stripped off on input, and strings will have their normal end-string character appended. STRUBAL+ logical records are delimited with the linefeed character. The skip-record (SRECRD) routine will read through an input file until the linefeed is encountered. This will position the file in the beginning of a logical record. The write-file-mark (FLFEED) routine will place a linefeed into an output file. Logical records may contain any arbitrary mixture of data types.

LOGICAL RECORD AS STORED ON A FILE

```
L bbbb 123 C STRINGbOFbCHARACTERS C b0.315 CL
F          R                      R        RF
```

How above file was written:

```
I=123
STRING S='STRING OF CHARACTERS'
F=0.315
WRITE I,S,F,/
```

How to read file (assuming file pointer at arbitrary point preceding the first linefeed):

```
READ /,I,S,F
```

Sequential-File I/O Routines

STRUBAL+ supports sequential input and output from files, using drivers similar to those used by the compiler itself. The following routines are the STRUBAL+ linkages to those drivers. Two flags are maintained in common which allows a STRUBAL+ program to detect end-of-files and I/O errors. The end-of-file flag .EOF is cleared by each file I/O operation. It is set to hex FFFF (minus 1) if an end-file condition was sensed during the file I/O operation. The error flag .ERR is cleared when a new file control block is specified in CURIOB. It is set to hex FFFF (minus 1) when an error condition is sensed during the file I/O operation (see File I/O drivers).

IREAD

This routine reads an integer value from a file. The file-read flag FRDFLG is set and the normal input routine INPUT is called.

Called by: .RTBL
Calls: INPUT
Flags: FRDFLG

FLREAD

This routine reads a floating-point value from a file. The file-read flag FRDFLG is set and the normal input routine FREAD is called.

Called by: .RTBL
Calls: FREAD
Flags: FRDFLG

SREAD

This routine reads a character string from a file. The file-read flag FRDFLG is set and the character input routine INEEE is used to get characters from the file. Linefeed characters are ignored, and an end-string character (hex 04) is appended in place of the carriage-return which terminates strings in files.

Called by: .RTBL
Calls: INEEE
Flags: FRDFLG

CURIOB

This routine passes the desired file control block address to the I/O drivers. The address is passed to this routine in the index register and is placed in the common area .CURIO. The error flag .ERR is cleared.

Called by: .RTBL
Calls: none
Flags: .ERR

SRECRD

This routine swallows characters from a file until a filemark (linefeed character) is found.

Called by: .RTBL
Calls: GETB (driver routine)
Flags: .EOF

IWRITE

This routine writes integer values to files. The write-file flag FWRFLG is set and the normal integer output routine OUTPUT is called. A carriage return is appended to the output for an end-marker.

Called by: .RTBL, FWRITE, SWRITE, QWRITE
Calls: OUTPUT, OUTB(driver routine)
Flags: FWRFLG

FWRITE

This routine writes floating-point values to files. The file-write flag FWRFLG is set and the normal floating-point output routine FPRINT is called. A carriage return is appended as an end-marker.

Called by: .RTBL
Calls: FPRINT, IWRITE
Flags: FWRFLG

SWRITE

This routine writes a formatted character string to a file. The file-write flag FWRFLG is set and the formatted-string output routine STROUT is called. A carriage return is appended as an end-marker.

Called by: .RTBL
Calls: STROUT, IWRITE
Flags: FWRFLG

QWRITE

This routine writes an unformatted (quoted) sting to a file. The file-write flag FWRFLG is set and the unformatted-string output routine QOUT is called. A carriage return is appended as an end-marker.

Called by: .RTBL
Calls: QOUT, IWRITE
Flags: FWRFLG

FLFEED

This routine writes a carriage-return and a linefeed to a file. This forms a filemark. The file-write flag FWRFLG is set and LFEED is called to output the CR. and LF.

Called by: .RTBL
Calls: LFEED
Flags: FWRFLG

FSPACE

This routine writes a variable number of spaces into a file. The file-write flags FWRFLG is set and SPACES is called.

Called by: .RTBL
Calls: SPACES
Flags: FWRFLG

File I/O Drivers

The sequential file processing routines access a set of drivers which control the storage devices being used.(floppy disks,cassettes,etc.) These drivers are similar to those used by the STRUBAL+ compiler. They are written specifically for the system being used, however their linkages are the same and they use the same set of entry points. They use a file control block in the STRUBAL+ program for buffering. The address of the current control block is found in the common area .CURIO. Together with the sequential file I/O routines, they control the .EOF and .ERR flags.

INITIO

This driver routine is called by the STRUBAL+ program directly to initialize the drivers and the device. It may or may not be needed, depending on the particular drivers in use.

OPEN

This driver routine is called with the address of a string in the index register. This string is the file name of the file to be opened. It must have a format appropriate to the system in use. The proper information will be written into the file control block pointed to by .CURIO. In case of error (no such file, etc.)the error flag .ERR will be set.

Called by: .RTBL, REWIND
Flags: .ERR

CLOSE

This driver routine closes the file whose file control block is pointed to by .CURIO. In case of error the error flag .ERR is set.

Called by: .RTBL, REWIND
Flags: .ERR

REWIND

This routine returns an input file to its starting position. The file is closed and then opened again. In case of error, the file error flag .ERR is set.

Called by: .RTBL

Calls: OPEN, CLOSE
Flags: .ERR

GETB

This driver routine gets the next sequential character from the file whose file control block is pointed to by .CURIO. The character is returned in the A register. If an end-file condition occured, the carry bit in the condition code register is set. Subsequent calls to GETB will continue to return the carry bit set.

Called by: INEEE, SRECRD

OUTB

This driver routine writes the character in the A register to the file whose file control block is pointed to by .CURIO.

Called by: OUTEEE, IWRITE

STRING ROUTINES

STRUBAL strings are formed from arrays of ASCII characters stored in sequential bytes. The end of a string is indicated by an 04 (hex) end-string character. Any other character may appear within a string.

LEN

This routine computes the length of a string. It starts at the address given in the index register and counts characters until an end-string is found. If the count becomes larger than 32767 (the largest positive integer), then an error is indicated and a length of zero is returned. Otherwise, the length is returned in the index register.

Called by: .RTBL, VAL
Calls: LENERR

VAL

This routine converts a string representation of an integer to its binary representation. The algorithm is to treat the input string as though it were typed at the console. If the string is not a valid integer representation, then an error message is printed and the value returned is -1 by default. The address of the string is in the index register upon entry, and the value is returned in the index register.

Called by: .RTBL
Calls: INPUT
Flags: RTYFLG

ASCII

This routine converts an integer to its string representation. The algorithm is identical to that used for output of an integer, except that the output characters are stored at the address contained in the index register. The input value is contained in the common area "ACCUM3". The index register is returned pointing to the end-string, so that string concatenation can overwrite the end-string with the concatenated string. The conversion is done using the existing format specifications.

Called by: .RTBL, FASCII, XFER, TENEXP
Calls: OUTPUT, OUTEEE

FASCII

This routine converts floating-point values to their string representations. The algorithm is identical to that used for printing, except that the output is placed at the address in the index register. The address of the input value is passed in common area "ACCUM3". The index register is returned pointing to the end-string character. The conversion is done using the existing format specifications.

Called by: .RTBL
Calls: FPRINT, ASCII

MID

This routine extracts a substring from a given string. The address of the source string is passed in the index register. This address is the beginning of the substring. The address of the destination of the substring is passed in ACCUM4. The count of characters to move to the destination is passed in ACCUM3. Characters are moved from source to destination until either the character count is exhausted or an end-string is found in the source string. An end-string character is appended to the destination string. If the character count passed exceeds 32767, a string-length error message is output and a default length of one character is used.

Called by: .RTBL
Calls: LENERR

XFER

This routine moves a character string from one address to another. The "from" address is passed in the index register. The "to" address is passed in the common area "ACCUM1". Characters are moved until an end-string is encountered, and an end-string is appended at the new address.

Called by: .RTBL, STRIN
Calls: PDATA, ASCII
Flags: FLAG

STROUT

This routine outputs a formatted string of characters. The output can be to a file, another string, or to the console. The setting of the flags FLAG and FWRFLG control the choice of output. The field-width value SIZE is used as the number of characters to output. If there are not enough characters in the string, blanks are output as necessary.

Called by: .RTBL, SWRITE
Calls: OUTEEE, OUTS
Flags: FWRFLG, FLAG

QOUT

This routine performs quoted string output. Other than clearing the flags for string and file I/O, this routine is just PDATA.

Called by: .RTBL, QWRITE
Calls: PDATA
Flags: FLAG, FWRFLG

STRIN

This routine performs string output. The input editor is used to get a string of characters (maximum length of 80 characters). An end-string is appended and the string is moved to the address passed in the index register.

Called by: .RTBL
Calls: INEDIT, XFER
Flags: FRDFLG

SPACES

This routine prints a number of spaces (hex 20) equal to the count passed in the index register. If the counter is negative, then no spaces are output.

Called by: .RTBL, FSPACE
Calls: OUTS
Flags: FWRFLG

DELAY FUNCTION

This routine provides a calibrated time-delay. Delay loops are calibrated for a 1-millisecond delay, and an outer loop counts milliseconds. The number of milliseconds to delay is passed in the index register. The value in the byte DTIME should be set to suit the cycle time of the processor in use.

Called by: .RTBL
Calls: none

PARAM

This routine transfers parameter addresses from the calling program list to the called program parameter list. The addresses of the list are obtained from the subroutine return addresses on the processor stack. Checking is done to insure that the end-of-list marker (FFFF hex) occurs in both lists at the same point. If not, an error message is generated and PARAM gives up.

Called by: .RTBL
Calls: PERR

FLOATING-POINT I/O ROUTINES

STRUBAL floating-point representations have up to 14 significant digits with a 3-digit exponent. STRUBAL actually allows only two digits for exponents; the third digit allows for overflows. Values are stored in nine bytes, BCD encoded. The first nibble contains the sign bit of the number, the sign bit of the exponent, and a bit indicating that the number is exactly zero. The second, third, and fourth nibbles contain the exponent digits. All values (except zero) are normalized.

DIGSET

This routine sets the formatting constants for FPRINT and STROUT. The value for the integer-part size is passed in ACCUM4. If it is in the range 1 to 127, it is stored in SIZE. If it is out of the valid range, an error message is output and the value of SIZE defaults to 6. The value of the number of decimal places is passed in the index register. If it is in the range 0 to 13, it is stored in DECPL . If it is out of the valid range, an error message is output and the value of DECPL defaults to zero.

Called by: .RTBL
Calls: DIGERR (which calls PERR for error messages)
Buffers: SIZE, DECPL

MPRINT

This routine prints strings of BCD-encoded digits. The index register contains the address of the string of digits, and the B accumulator contains the digit count. Entry-point MPRNT2 prints digits starting with the right nibble, MPRINT normally starts with the left nibble.

Called by: FPRINT
Calls: OUTHL, OUTHR

FPRINT

This routine outputs floating-point values whose address is passed in the index register. The flags FLAG and FWRFLG determine the type of output---to a string, file, or console. The output is formatted by the values of SIZE and DECPL. If the value cannot be output in the specified format, then the output is made in full scientific notation with 14 digits of precision and a three digit exponent. Leading zeros in the exponent are blanked. Blanks are used to

pad the value if necessary to fit the format. The routine MPRINT is used to output strings of BCD-encoded digits.

Called by: .RTBL, OUTPUT, FWRITE, FASCII
Calls: OUTS, OUTEEE, MPRINT, OUTHL, OUTHR

FVAL

This routine converts a string representation of a floating-point value to a binary representation. The algorithm used is to treat the string whose address is passed in the index register as though it were an input at the console. The value is formed at the address in common area "ACCUM4". If the string is not a valid representation of a floating-point number, an error message is given and the value produced is not determined.

Called by: .RTBL
Calls: FREAD
Flags: RTYFLG

FREAD

This routine is used to input a floating-point number. The value may or may not have a sign. It might have a decimal point. There may be a signed or unsigned exponent. The exponent may be one or two digits. The mantissa of the value may have up to 14 significant digits.

FREAD first calls the input editor to get a character string. It clears the floating-point value whose address is given in the index register. Characters are read from the input string, left to right. First, plus or minus signs are handled and the sign bit of the value is set. Then, a loop is entered which extracts the mantissa of the number. A carriage-return or end-string terminates the loop. A count is made of the number of significant digits to the right or left of the decimal point, depending upon whether there were any significant digits to the left of the decimal point. Up to 14 digits will be extracted, converted to BCD form, and stored in the value. Any further digits will be lost. Syntax errors (multiple decimal points, non-numeric characters, etc.) will be trapped with an error message. Error retries may be attempted. An exponent is indicated by the letter "E". This transfers out of the mantissa loop and begins the exponent processing. The sign is extracted, and the exponent-sign bit is set in the first nibble. One or two digits are then extracted and converted to BCD form. Now this exponent must be combined with the decimal-place count formed in the mantissa extraction. The exponent is converted to binary and the decimal-place count is added. The exponent value is now forced positive and converted back to BCD form. A third exponent digit may now be required.

Called by: .RTBL, FVAL
Calls: INEDIT, INCHAR, BADERR
Flags: RTYFLG, FRDFLG

INCHAR

This routine extracts characters from the input buffer. It skips over blanks.

Called by: FREAD
Calls: none

FIX

This routine converts floating-point values to integers. The index register contains the address of the floating-point value on entry; on exit the index register contains the integer value. If the floating-point value is too large to fit an integer, than an overflow error is given and the integer defaults to the largest possible integer value. Any floating-point value of magnitude less than one is made an integer zero.

Called by: .RTBL, EXTRCT, TENEXP
Calls: OVERR, MULT, NEGATE

FLOAT

This routine converts integers to floating-point values. The integer is passed in common area "ACCUM1", and the address of the floating-point result is passed in the index register. The digits are converted to BCD form and stored into the floating-point representation. The table of powers-of-10 (KIOK) is used.

Called by: .RTBL, EXTRCT, TENEXP, OUTPUT
Calls: NEGATE

FCOMPR

This routine compares two floating-point values and returns the results of the compare in the condition codes. The address of the first number is passed in the index register, the address of the second number is passed in common area "ACCUM1". The two values are subtracted using FSUB and the condition codes are set to indicate the resulting value.

Called by: .RTBL, ATAN, ASIN
Calls: FTAREG, FTCREG, FSUB, FFDREG

FABS

This routine makes the floating-point value whose address is passed in the index register positive. This is done by clearing the number's sign bit.

Called by: .RTBL, COS, ATAN, ASIN, TENEXP
Calls: none

FSIGN

This routine performs the signum function on floating-point numbers whose address is passed in the index register. If the number is exactly zero, then it is immediately returned. If the number is positive, then plus one (+1) is returned in the floating-point value. If the number is negative, then minus one (-1) is returned.

Called by: .RTBL, SERIES, TENEXP
Calls: none

FNEG

This routine changes the sign of a floating-point number whose address is passed in the index register.

Called by: .RTBL, COS, ATAN, LOG10
Calls: none

FLOTPNT

This package contains the basic arithmetic routines for floating-point arguments. It is a modification of program 32 of the Motorola M6800 Users Group Library, originally written by Raymond Naugle. All values are stored with 14 significant digits and a three digit base 10 exponent. The digits are stored in BCD form, as follows:

BYTE 1:	Bit 7	sign of number (0=positive)
	Bit 6	sign of exponent (0=positive)
	Bit 5	number=0 exactly (1=exact zero)
	Bit 4	unused
	Bits 0-3	hundreds digit of exponent
BYTE 2:	Bits 4-7	tens digit of exponent
	Bits 0-3	ones digit of exponent
BYTES 3-9		digits of mantissa, packed as Byte 2

The mantissa, or fractional part, is always normalized...that is, it is left justified with no leading zeros.

The floating-point package uses three special "registers",labelled as the AREG, CREG, and DREG. The arguments of the arithmetic functions are placed in the A and C registers. The result is placed in the D register. The routines in this package perform the following functions:

FADD	DREG=AREG+CREG	
FSUB	DREG=AREG-CREG	
FMUL	DREG=AREG*CREG	
FDIV	DREG=CREG/AREG	(note the argument order here)

Special moving routines are provided within the package to move arguments from arbitrary addresses to the registers and to move the result from the D register to an arbitrary address. All these routines use a general-purpose "mover" routine within the package.

FTAREG

This routine moves a floating-point value from the address contained in the index register to the A register of the floating-point package.

Called by: .RTBL, FCOMPR, EXTRCT, SERIES, TAN, COS, SIN, ATAN, ACOS, ASIN, NATLOG, LOG10, FEXP, TENEXP, XTOY
Calls: MOVER

FTCREG

This routine moves a floating-point value from the address contained in the index register to the C register of the floating-point package.

Called by: .RTBL, FCOMPR, EXTRCT, SERIES, TAN, COS, SIN, ATAN, ACOS, ASIN, NATLOG, LOG10, FEXP, TENEXP, XTOY
Calls: MOVER

FFDREG

This routine moves a floating-point value from the D register in the floating-point package to the address contained in the index register.

Called by: .RTBL, FCOMPR, EXTRCT, SERIES, TAN, COS, SIN, ATAN, ACOS, ASIN, NATLOG, LOG10, FEXP, TENEXP, XTOY
Calls: MOVER

MOVER

This routine moves a string of bytes from one address to another. The address from which to move bytes is passed in pointer ADRFM. The address to which bytes are moved is passed in pointer ADRTO. The number of bytes to move is passed in accumulator B(not to be confused with BREG).

Called by: .RTBL, FTAREG, FTCREG, FFDREG, ZMOVER, TAN, COS, LOG10
Calls: none

DIGSET

This routine sets up the register pointers and counters in the floating-point package to account for the number of digits of precision desired.

```
AMOD  points to AREG  (default=AREG)
BMOD  points to BREG  (default=BREG)
CMOD  points to CREG  (default=CREG)
DMOD  points to DREG  (default=DREG)
EMOD  points to EREG  (default=EREG)
NB    no. of bytes in mantissa  (default=7)
ND    no. of digits in fraction (default=14 hex)
```

The number of digits of precision to use is passed in the index register. This value must be between 4 and 14. If not, default values are used (assumes 14 digits). The number will be made even, since there are 2 digits per byte.

Called by: .RTBL
Calls: none

ADD

This routine adds the 1-7 byte mantissas of the floating-point values in the A and C registers. The result is stored in the E register.

Called by: FADD

Calls: none

SUB

This routine subtracts the 1-7 byte mantissa of the floating-point values in the A and C registers. The result is stored in the E registers.

Called by: FADD
Calls: none

MUL

This routine multiplies the 1-7 byte mantissas of the floating-point values in the A and C registers. The result is stored in the E registers.

Called by: FMUL
Calls: none

DIV

This routine divides the 1-7 byte mantissas of the floating-point values in the A and C registers. (The C value is divided by the A value.) The result is returned in the E register.

Called by: FDIV
Calls: none

PAK

This routine normalizes the results of a floating-point computation. It deletes leading zeros from the mantissa and adjusts the exponent accordingly. The address of the floating-point value to be normalized ("PACKED") is passed in the index register. All inputs to the floating-point routines are already normalized, so packing (normalization) is only needed on exit.

Called by: FADD, FSUB, FMUL, FDIV
Calls: none

FADD

This routine adds the floating-point values in the A and C registers, returning the sum in the D register. The numbers are shifted to equalize the exponents, and the mantissas are added or subtracted, depending upon the signs of the numbers. The A register is returned unchanged, the C register is destroyed. The result is packed (normalized) and returned in the D register.

Called by: .RTBL, FSUB
Calls: ADD, SUB, PAK

FSUB

This routine subtracts the floating-point values in the A and C registers. The sign bit of the C register value is inverted and the values are then added.

Called by: .RTBL, FCOMPR
Calls: FADD

FMUL

This routine multiplies the floating-point values in the A and C registers, returning the product in the D register. The exponents are added and the mantissas are multiplied. The result is then packed (normalized) and returned in the D register. The A register is saved, but the C register is destroyed.

Called by: .RTBL
Calls: MUL, PAK

FDIV

This routine divides the floating-point values in the A and C registers, returning the result in the D register. (D=C/A) The exponents are subtracted and the mantissas divided. The result is then packed (normalized) and returned in the D register. The A register is saved.

Called by: .RTBL
Calls: DIV, PAK

SCIFUN

This scientific-subroutine package consists of a set of STRUBAL+ callable routines which perform trigonometric, logarithmic, and other mathematical functions useful in scientific programs. They are all designed around series expansions. The series used are the coefficients were taken from "Handbook of Mathematical Functions" by M. Abramowitz and A. Stegun. References given for the various routines are from this book. Except as noted, the routines should give six to seven digits of accuracy. The functions implemented in this package are:

Trigonometric...SINE, COSINE, TANGENT, ARC-SINE, ARC-COSINE, ARC-TANGENT

Logarithmic.....LOG (base 10), LOG (base e)

Exponential.....10**X, e**X, X**Y

All the trigonometric functions take arguments in radians. The inverse trig. functions return arguments in radians. All the scientific functions work with floating-point arguments and return floating-point results.

EXTRCT

This routine is used by the scientific routine to remove factors from numbers. The argument is divided by a constant, and the integer-part of the result is returned along with the floating-point remainder. The address of the argument is contained in ARGPNT, and the address of the constant is contained in CONPNT. The integer number of factors is returned in IFACT, and the remainder is left in the floating-point 'D' register.

Called by: COS
Calls: FTCREG, FTAREG, FDIV, FFDREG, FIX, FLOAT, FMUL, FSUB

SERIES

This routine performs the actual series expansion for the scientific routines. When called, the address of the argument is in the index register, the number of terms is contained in TERMS, and the address of the coefficient table is contained in ANGLE. The location SUM is initialized to the first term of the series and the final series value is returned there.

Called by: COS, ATAN, ASIN, LOG10, TENEXP
Calls: FTAREG, FTCREG, FFDREG, FMUL, FSIGN, FADD

TAN

This routine computes the tangent of an argument in radians. The algorithm is to divide the sine of the argument by the cosine of the argument. The sine and cosine are computed by SIN and COS respectively.

Called by .RTBL
Calls: MOVER, COS, SIN, FTAREG, FTCREG, FDIV, FFDREG

COS

This routine computes the cosine of the argument by series expansion. The series used is A+S formula 4.3.99 found on page 76. The coefficients are found in table COSCON.

COS (x)=1+a1(X**2)+a2(X**4)+a3(X**6)+a4(X**8)+a5(X**10)

The routine first reduces the argument to the range 0-360 degrees, then extracts factors of 90 degrees. The quadrant of the argument is used to set the proper sign for the result of the series expansion. In the quadrants 1 and 3, the cosine is computed on 90-x. The routine called with the address of the argument in the index register. The result is returned to this address.

Called by: .RTBL, TAN, SIN
Calls: FABS, EXTRCT, FFDREG, FTAREG, FTCREG, FSUB, FSIGN, SERIES, FNEG, MOVER

SIN

This routine computes the sine of the argument. It uses the relation: SIN(x)=COS (90-x) The argument address is passed in the index register, and the result is returned there.

Called by: .RTBL, TAN
Calls: FTAREG, FTCREG, FSUB, FFDREG, COS

ATAN

This routine computes the inverse-tangent of the argument. A series expansion is used, given by A+S formula 4.4.49 on page 81. The coefficients are found in the table ATNCON.

```
ATAN(X)/X=1+a1(X**2)+a2(X**4)+a3(X**6)+a4(X**8)+a5(X**10)
         +a6(X**12)+a7(X**14)+a8(X**16)
```

This formula holds for arguments whose magnitude is less than one. For arguments whose magnitude is greater than one, the argument is inverted-the series is computed-and then the relation:

ATAN(x) = 90-ATAN(1/x)

is used. The address of the argument is passed in the index register, and the result is returned there.

Called by: .RTBL
Calls: FABS, FCOMPR, FTAREG, FTCREG, FDIV, FFDREG, FSIGN, SERIES, FSUB, FNEG

ACOS

This routine computes the inverse cosine of its argument. The argument address is passed in the index register, and the result is returned to that address. The algorithm uses the following relations:

ACOS (x) = 90-ASIN (x) for positive x
ACOS(-x) = 90+ASIN(-x) for negative x

to derive the inverse cosine from the inverse sine.

Called by: .RTBL
Calls: ASIN, FTAREG, FTCREG, FSUB, FADD, FFDREG

ASIN

This routine computes the inverse sine of its argument. The address of the argument is passed in the index register, and the result is returned to the address. If the argument magnitude is greater than one, an error message is returned and the value returned will be nonsense. For arguments whose magnitude is less than or equal to one, the following series expansion is used:

ASIN(X)=X+a1(X**3)+a2(X**5)+a3(X**7)+....

This series is formula 4.4.40 from A+S page 81. The coefficients are in table ASNCON. One power of the argument is factored out, since the series-expansion subroutine generates even series. The argument is multiplied back at the end. The series gives poor accuracy for large arguments, roughly:

ABS (x) less the 0.5	8 digit accuracy
ABS (x) between 0.5 and 0.75	5 digit accuracy
ABS (x) between 0.75 and 0.9	3 digit accuracy
ABS (x) greater than 0.9	2 digit accuracy

Called by: .RTBL, ACOS
Calls: FABS, FCOMPR, ILLERR, FSIGN, SERIES, FTAREG, FTCREG, FMUL, FFDREG

NATLOG

This routine computes the natural logarithm of its argument. The address of the argument is passed in the index register, and the result is returned to that address. The algorithm uses the following relation:

LN(x)=LN(10) * LOG10(x)

to derive the natural logarithm from the base-10 logarithm.

Called by: .RTBL
Calls: LOG10, FTAREG, FTCREG, FMUL

LOG10

This routine computes the base-10 logarithm of its argument. The address of the argument is passed in the index register and the result is returned there. Arguments less than zero are flagged with an error message and the result will be erroneous. If the argument is exactly zero, the result will be the largest possible negative floating-point value. The logarithm is computed in two steps. First, the base-10 exponent of the argument is converted from its 3 digit BCD encoded form to a floating-point value. Second, the fractional part (mantissa) of the argument is series expanded using the formula from A+S number 4.1.42 on page 68:

LOG10(x)=a1(T)+a2(T**3)+a3(T**5)+a4(T**7)+a5(T**9)

where T=(x-1)/(x+1)

The coefficients of this series are in table LOGCON. Ten terms are required to get sufficient accuracy in this series. To further increase the accuracy of this series for small values of x, the argument is multiplied by 3.0 before the series is expanded and LOG10(3.0) is subtracted after the expansion to compensate. The results of the series are added to the exponent value to get the total logarithm.

Called by: .RTBL, NATLOG, XTOY
Calls: ILLERR, FNEG, FTAREG, FTCREG, FSUB, FADD, FFDREG, FDIV, MOVER, SERIES, FMUL

FEXP

This routine computes the natural exponential function of the argument. The address of the argument is passed in the index register and the result is returned there. The routine computes the exponential by using the following relation:

```
FEXP(X) = TENEXP( LOG10(e)*X)
which uses the base-10 exponential routine.
```

Called by: .RTBL
Calls: FTAREG, FTCREG, FMUL, FFDREG, TENEXP

TENEXP

This routine computes the base-10 exponential of its argument. The address of the argument is passed in the index register, and the result is returned there. The algorithm has two sections. The first section converts the integer part of the argument to a floating-point exponent with a mantissa of one. Range checks are made, and if the magnitude of the argument exceeds 255 the default values are returned. The fractional part (mantissa) of the argument is series expanded in the second step of the algorithm. Actually, two series expansions are made, one as an even series and one as an odd series. This is required by the way the series-expansion routine works. The combined series has ten terms as follows:

```
TENEXP(X)=10**X=a1+a3(X)+a5(X**3)+a7(X**5)+...
               +a2(X**2)+a4(X**4)+a6(X**6)+...
```

where the argument has been scaled by LN (10). The coefficients for the series are contained in tables TCON1 and TCON2. The two series results are added, the sum is multiplied by the power-of-ten value, and this value is corrected for the proper sign of the argument, using the relation:

```
TENEXP(-X)=1/TENEXP(X)
```

Called by: .RTBL, FEXP, XTOY
Calls: FABS, FSIGN, FCOMPR, FIX, ASCII, FLOAT, FTAREG, FTCREG, FFDREG, FSUB FMUL, SERIES, FADD, FDIV

XTOY

This is the only binary routine in the scientific package. It computes the result of an arbitrary power. Both arguments must be floating-point. The address of the base is passed in common area "ACCUM1", and the address of the exponent is passed in the index register. The result is returned to the address of the base. The algorithm used is given in the following relation:

```
X XTOY Y = (X**Y) = TENEXP(Y * LOG10(X))
```

Called by: .RTBL
Calls: LOG10, FTAREG, FTCREG, FMUL, FFDREG, TENEXP

COMMON AREAS:

STRUBAL programs use a set of common areas to link with the runtime packages. These areas serve as 16-bit pseudo-registers and as a runtime stack. There are 107 bytes total of runtime common areas, denoted as follows:

RUNSTK---length 80 bytes
This area is used as runtime stack space for expansion evaluation.

ACCUM1,ACCUM2,ACCUM3,ACCUM4---length 2 bytes each. These areas are used as 16-bit registers.

ACCUM5 length 9 bytes.
This area is used as a floating-point register.

ADRTO length 2 bytes
This area is used as an address pointer (address-to)

ADRFM length 2 bytes
This area is used as an address pointer (address from)

.EOF length 2 bytes
This area is used as an end-file flag for STRUBAL file I/O

.ERR length 2 bytes
This area is used as an error flag for STRUBAL file I/O

.CURIO length 2 bytes
This area is used as the address pointer for the current STRUBAL file I/O block.

```
0001  0000 0000   N        NAM STRUBAL+ COMPILER
0002                 *
0003                 *
0004                 *  SSS  TITTI RRRR  U   U BBBB    A   L
0005                 * S       T   R   R U   U B   B  A A  L        +
0006                 *  SSS    T   RRRR  U   U BBBB  AAAAA L       +++++
0007                 *     S   T   R R   U   U B   B A   A L        +
0008                 *  SSS    T   R  R   UUU  BBBB  A   A LLLLL        TM
0009                 *
0010                 *
0011                 *
0012                 * A LANGUAGE COMPILER/ASSEMBLER PACKAGE
0013                 * FOR MOTOROLA 6800 PROCESSORS
0014                 *
0015                 *
0016                 * COPYRIGHT 1978    ROBERT GRAPPEL, LEXINGTON, MASS
0017                 *                   AND JACK HEMENWAY, BOSTON, MASS
0018                 *                   ALL RIGHTS RESERVED
0019                 *
0020                 *
0021  0000 BF 0713 R        STS STACK    INIT STACK POINTER
0022  0003 FE 3ECA R        LDX TABLES+1
0023  0006 EE 00           LDX 0,X      GET START OF TABLES
0024  0008 FF 0084 R        STX MACTBL   MACRO TABLE ADDRESS
0025                 *
0026  000B FF 19E5 R        STX PSTNG1
0027  000E CE 1000          LDX #$1000   4K MACRO TABLE SPACE
0028  0011 FF 19E7 R        STX PSTNG2
0029  0014 CE 19E5 R        LDX #PSTNG1
0030  0017 BD 234B R        JSR ADD16
0031  001A FE 19E5 R        LDX PSTNG1
0032  001D FF 0086 R        STX MACEND   END OF MACRO TABLE
0033                 *
0034  0020 CE 0100          LDX #$0100   256 BYTE MACRO STACK
0035  0023 FF 19E7 R        STX PSTNG2
0036  0026 CE 19E5 R        LDX #PSTNG1
0037  0029 BD 234B R        JSR ADD16
0038  002C FE 19E5 R        LDX PSTNG1
0039  002F FF 0088 R        STX MACSTK   START OF MACRO STACK
0040                 *
0041  0032 08               INX
0042  0033 FF 008A R        STX SYMTAB   SYMBOL TABLE ADDRESS
0043                 *
0044  0036 FF 19E7 R        STX PSTNG2
0045  0039 CE 7FFF          LDX #$7FFF   32K SPACE
0046  003C FF 19E5 R        STX PSTNG1
0047  003F CE 19E5 R        LDX #PSTNG1
0048  0042 BD 235E R        JSR SUB16
0049  0045 B6 19E5 R        LDA A PSTNG1
0050  0048 F6 19E6 R        LDA B PSTNG1+1
0051  004B CE 0009          LDX #9       9 BYTES PER SYMBOL
0052  004E FF 19E7 R        STX PSTNG2
0053  0051 CE 19E7 R        LDX #PSTNG2
0054  0054 BD 2300 R        JSR DIV16
0055  0057 B7 008E R        STA A NSYM   NSYM=(32K-SYMTAB)/9
0056  005A F7 008F R        STA B NSYM+1
0057                 *
0058  005D CE 0009          LDX #9
0059  0060 FF 19E5 R        STX PSTNG1
0060  0063 CE 19E5 R        LDX #PSTNG1
0061  0066 BD 2273 R        JSR MPY16
0062  0069 B7 19E5 R        STA A PSTNG1
0063  006C F7 19E6 R        STA B PSTNG1+1
0064  006F FE 008A R        LDX SYMTAB
0065  0072 FF 19E7 R        STX PSTNG2
0066  0075 CE 19E5 R        LDX #PSTNG1
0067  0078 BD 234B R        JSR ADD16
0068  007B FE 19E5 R        LDX PSTNG1
0069  007E FF 008C R        STX SYMEND   SYMEND=9*NSYM+SYMTAB
0070  0081 7E 0715 R        JMP OPC1
0071                 *
0072                 * MACRO TABLE PARAMETERS
0073                 *
0074  0084 0002      MACTBL RMB 2        MACRO TABLE START
0075  0086 0002      MACEND RMB 2        TABLE END
0076  0088 0002      MACSTK RMB 2        MACRO STACK START
0077                 *
0078                 * SYMBOL TABLE PARAMETERS
0079                 *
0080  008A 0002      SYMTAB RMB 2        START OF TABLE
0081  008C 0002      SYMEND RMB 2        END OF TABLE
0082  008E 0002      NSYM   RMB 2        NUMBER OF SYMBOLS
0083                 *
0084                 *
0085                 * KEYWORD AND MNEMONIC TABLE *
0086                 *
0087  0090 2E        MNTAB  FCC ' ABS'   ABSOLUTE-VALUE FUNCTION
0088  0094 11DB   R         FDB ABSPR
0089  0096 3B               FCB $3B
0090  0097 2E               FCC ' ACO'  ARC-COSINE FUNCTION
0091  009B 11EF   R         FDB ACOSPR
0092  009D B0               FCB $B0
0093  009E 2E               FCC ' AND'  AND FUNCTION
0094  00A2 1347   R         FDB ANDPR
0095  00A4 3F               FCB $3F
0096  00A5 2E               FCC ' ASI'  ARC-SINE FUNCTION
0097  00A9 11F3   R         FDB ASINPR
0098  00AB B1               FCB $B1
0099  00AC 2E               FCC ' ATA'  ARC-TANGENT FUNCTION
0100  00B0 11F7   R         FDB ATANPR
0101  00B2 B2               FCB $B2
0102  00B3 2E               FCC ' COS'  COSINE FUNCTION
0103  00B7 11FB   R         FDB COSPR
0104  00B9 B3               FCB $B3
0105  00BA 2E               FCC '.EOF'  END-FILE FUNCTION
0106  00BE 1E2D   R         FDB EOFPR
0107  00C0 39               FCB $39
0108  00C1 2E               FCC '.EQ. '  .EQ. COMPARISON
0109  00C5 35F9   R         FDB EQPR
0110  00C7 41               FCB $41
0111  00C8 2E               FCC ' ERR'  ERROR FUNCTION
0112  00CC 1E2D   R         FDB EOFPR   (LIKE EOF)
0113  00CE 39               FCB $39
0114  00CF 2E               FCC ' EXP'  EXPONENT FUNCTION
0115  00D3 134F   R         FDB EXPR
0116  00D5 2C               FCB $2C
0117  00D6 2E               FCC '.FIX'  FIX FUNCTION
0118  00DA 1203   R         FDB FIXPR
0119  00DC 36               FCB $36
0120  00DD 2E               FCC ' FLO'  FLOAT FUNCTION
0121  00E1 1357   R         FDB FLOTPR
```

```
0122  00E3 B7          FCB $B7
0123  00E4 2E          FCC ' GE. '     GE  COMPARISON
0124  00E8 360D    R   FDB GEPR
0125  00EA 42          FCB $42
0126  00EB 2E          FCC ' GT. '     GT  COMPARISON
0127  00EF 361B    R   FDB GTPR
0128  00F1 43          FCB $43
0129  00F2 2E          FCC '.LE. '     .LE.  COMPARISON
0130  00F6 3629    R   FDB LEPR
0131  00F8 44          FCB $44
0132  00F9 2E          FCC '.LOG'      LOG10 FUNCTION
0133  00FD 120A    R   FDB LOGPR
0134  00FF B8          FCB $B8
0135  0100 2E          FCC ' LT '      .LT.  COMPARISON
0136  0104 3637    R   FDB LTPR
0137  0106 45          FCB $45
0138  0107 2E          FCC ' MOD'      MODULUS FUNCTION
0139  010B 1353    R   FDB MODPR
0140  010D 2E          FCB $2E
0141  010E 2E          FCC '.NAT'      NATURAL-LOG FUNCTION
0142  0112 120E    R   FDB NATPR
0143  0114 B9          FCB $B9
0144  0115 2E          FCC '.NE. '     .NE.  COMPARISON
0145  0119 3645    R   FDB NEPR
0146  011B 46          FCB $46
0147  011C 2E          FCC ' NOT'      NOT FUNCTION
0148  0120 11D3    R   FDB NOTPR
0149  0122 3A          FCB $3A
0150  0123 2E          FCC '.OR. '     OR FUNCTION
0151  0127 1343    R   FDB ORPR
0152  0129 3F          FCB $3E
0153  012A 2E          FCC '.PEE'      PEEK FUNCTION
0154  012E 11F7    R   FDB PEEKPR
0155  0130 38          FCB $38
0156  0131 2E          FCC ' RAN'      RANDOM FUNCTION
0157  0135 11DF    R   FDB RANPR
0158  0137 3C          FCB $3C
0159  0138 2E          FCC '.SGN'      S(I)GNUM FUNCTION
0160  013C 11F3    R   FDB SIGPR
0161  013E 3D          FCB $3D
0162  013F 2E          FCC '.SIN'      SINE FUNCTION
0163  0143 121A    R   FDB SINPR
0164  0145 B4          FCB $B4
0165  0146 2E          FCC '.TAN'      TANGENT FUNCTION
0166  014A 121E    R   FDB TANPR
0167  014C B5          FCB $B5
0168  014D 2E          FCC '.TEN'      BASE10 EXPONENT FUNCTION
0169  0151 1222    R   FDB TENPR
0170  0153 AE          FCB $AE
0171  0154 2E          FCC ' XOR'      EXCLUSIVE-OR FUNCTION
0172  0158 134B    R   FDB EXOPR
0173  015A 40          FCB $40
0174  015B 2E          FCC ' XTO'      X-TO-Y POWER FUNCTION
0175  015F 1377    R   FDB XTOYPR
0176  0161 BC          FCB $BC
0177  0162 41          FCC 'ABA '
0178  0166 2CF3    R   FDB ADDR9
0179  0168 1B          FCB $1B
0180  0169 41          FCC 'ADC '
0181  016D 2A8B    R   FDB ADDR1
0182  016F 09          FCB $09
0183  0170 41          FCC 'ADD '
0184  0174 2A8B    R   FDB ADDR1
0185  0176 0B          FCB $0B
0186  0177 41          FCC 'AND '
0187  017B 2A8B    R   FDB ADDR1
0188  017D 04          FCB $04
0189  017E 41          FCC 'ASL '
0190  0182 2B84    R   FDB ADDR3
0191  0184 08          FCB $08
0192  0185 41          FCC 'ASR '
0193  0189 2B84    R   FDB ADDR3
0194  018B 07          FCB $07
0195  018C 42          FCC 'BCC '
0196  0190 2CA2    R   FDB ADDR8
0197  0192 24          FCB $24
0198  0193 42          FCC 'BCS '
0199  0197 2CA2    R   FDB ADDR8
0200  0199 25          FCB $25
0201  019A 42          FCC 'BEQ '
0202  019E 2CA2    R   FDB ADDR8
0203  01A0 27          FCB $27
0204  01A1 42          FCC 'BGE '
0205  01A5 2CA2    R   FDB ADDR8
0206  01A7 2C          FCB $2C
0207  01A8 42          FCC 'BGT '
0208  01AC 2CA2    R   FDB ADDR8
0209  01AE 2F          FCB $2E
0210  01AF 42          FCC 'BHI '
0211  01B3 2CA2    R   FDB ADDR8
0212  01B5 22          FCB $22
0213  01B6 42          FCC 'BIT '
0214  01BA 2A8B    R   FDB ADDR1
0215  01BC 05          FCB $05
0216  01BD 42          FCC 'BLE '
0217  01C1 2CA2    R   FDB ADDR8
0218  01C3 2F          FCB $2F
0219  01C4 42          FCC 'BLOC'      BLOCK KEYWORD
0220  01C8 389F    R   FDB BLOKPR
0221  01CA F0          FCB $F0
0222  01CB 42          FCC 'BLS '
0223  01CF 2CA2    R   FDB ADDR8
0224  01D1 23          FCB $23
0225  01D2 42          FCC 'BLT '
0226  01D6 2CA2    R   FDB ADDR8
0227  01D8 2D          FCB $2D
0228  01D9 42          FCC 'BMI '
0229  01DD 2CA2    R   FDB ADDR8
0230  01DF 2B          FCB $2B
0231  01E0 42          FCC 'BNE '
0232  01E4 2CA2    R   FDB ADDR8
0233  01E6 26          FCB $26
0234  01E7 42          FCC 'BPL '
0235  01EB 2CA2    R   FDB ADDR8
0236  01ED 2A          FCB $2A
0237  01EE 42          FCC 'BRA '
0238  01F2 2CA2    R   FDB ADDR8
0239  01F4 20          FCB $20
0240  01F5 42          FCC 'BSR '
0241  01F9 2CA2    R   FDB ADDR8
0242  01FB 8D          FCB $8D
0243  01FC 42          FCC 'BVC '
```

```
0244  0200 2CA2   R    FDB ADDR8
0245  0202 28          FCB $28
0246  0203 42          FCC 'BVS '
0247  0207 2CA2   R    FDB ADDR8
0248  0209 29          FCB $29
0249  020A 42          FCC 'BYTE'   BYTE KEYWORD
0250  020E 1444   R    FDB BYTEPR
0251  0210 F0          FCB $F0
0252  0211 43          FCC 'CALL'   CALL KEYWORD
0253  0215 39EB   R    FDB CALLPR
0254  0217 F0          FCB $F0
0255  0218 43          FCC 'CBA '
0256  021C 2CF3   R    FDB ADDR9
0257  021E 11          FCB $11
0258  021F 43          FCC 'CEND'   C(OMMON-) END KEYWORD
0259  0223 25B5   R    FDB CENDPR
0260  0225 F0          FCB $F0
0261  0226 43          FCC 'CLC '
0262  022A 2CF3   R    FDB ADDR9
0263  022C 0C          FCB $0C
0264  022D 43          FCC 'CLI '
0265  0231 2CF3   R    FDB ADDR9
0266  0233 0E          FCB $0E
0267  0234 43          FCC 'CLOS'   CLOSE-FILES KEYWORD
0268  0238 27AF   R    FDB CLOSPR
0269  023A F0          FCB $F0
0270  023B 43          FCC 'CLR '
0271  023F 2B84   R    FDB ADDR3
0272  0241 0F          FCB $0F
0273  0242 43          FCC 'CLV '
0274  0246 2CF3   R    FDB ADDR9
0275  0248 0A          FCB $0A
0276  0249 43          FCC 'CMN '
0277  024D 2E5A   R    FDB POCMN
0278  024F FF          FCB $FF
0279  0250 43          FCC 'CMP '
0280  0254 2A8B   R    FDB ADDR1
0281  0256 01          FCB $01
0282  0257 43          FCC 'COM '
0283  025B 2B84   R    FDB ADDR3
0284  025D 03          FCB $03
0285  025E 43          FCC 'COMM'
0286  0262 2E5A   R    FDB POCMN
0287  0264 FF          FCB $FF
0288  0265 43          FCC 'CONV'   CONVERT KEYWORD
0289  0269 14BB   R    FDB CONVPR
0290  026B F0          FCB $F0
0291  026C 43          FCC 'CPX '
0292  0270 2BF3   R    FDB ADDR5
0293  0272 8C          FCB $8C
0294  0273 43          FCC 'CSEC'   C(OMMON-) SECTION KEYWORD
0295  0277 25A9   R    FDB CSECPR
0296  0279 F0          FCB $F0
0297  027A 44          FCC 'DAA '
0298  027E 2CF3   R    FDB ADDR9
0299  0280 19          FCB $19
0300  0281 44          FCC 'DEC '
0301  0285 2B84   R    FDB ADDR3
0302  0287 0A          FCB $0A
0303  0288 44          FCC 'DEFI'   DE(FINE-) FLOATING KEYWORD
0304  028C 25C9   R    FDB DEFPR
0305  028E F0          FCB $F0
0306  028F 44          FCC 'DELA'   DELAY KEYWORD
0307  0293 20D5   R    FDB DELAY
0308  0295 F0          FCB $F0
0309  0296 44          FCC 'DEND'   D(ATA-SECTION) END KEYWORD
0310  029A 259F   R    FDB DENDPR
0311  029C F0          FCB $F0
0312  029D 44          FCC 'DES '
0313  02A1 2CF3   R    FDB ADDR9
0314  02A3 34          FCB $34
0315  02A4 44          FCC 'DEX '
0316  02A8 2CF3   R    FDB ADDR9
0317  02AA 09          FCB $09
0318  02AB 44          FCC 'DIGI'   DIGITS KEYWORD
0319  02AF 2774   R    FDB PRNTPR   (WORKS LIKE PRINT)
0320  02B1 F0          FCB $F0
0321  02B2 44          FCC 'DIGS'   DIGSET KEYWORD
0322  02B6 2548   R    FDB DIGSPR
0323  02B8 F0          FCB $F0
0324  02B9 44          FCC 'DSEC'   D(ATA) SECTION KEYWORD
0325  02BD 258B   R    FDB DSECPR
0326  02BF F0          FCB $F0
0327  02C0 44          FCC 'DSTR'   D(ECLARE-) STRING KEYWORD
0328  02C4 25C1   R    FDB DSTPR
0329  02C6 F0          FCB $F0
0330  02C7 45          FCC 'ELSE'   ELSE KEYWORD
0331  02CB 36EC   R    FDB IFPR3
0332  02CD F0          FCB $F0
0333  02CE 45          FCC 'END '
0334  02D2 2ED4   R    FDB POEND
0335  02D4 FF          FCB $FF
0336  02D5 45          FCC 'ENT '
0337  02D9 30C0   R    FDB POENT
0338  02DB FF          FCB $FF
0339  02DC 45          FCC 'ENTR'
0340  02E0 30C0   R    FDB POENT
0341  02E2 FF          FCB $FF
0342  02E3 45          FCC 'EOR '
0343  02E7 2A8B   R    FDB ADDR1
0344  02E9 08          FCB $08
0345  02EA 45          FCC 'EQU '
0346  02EE 3138   R    FDB POEQU
0347  02F0 FF          FCB $FF
0348  02F1 45          FCC 'EQUA'
0349  02F5 3138   R    FDB POEQU
0350  02F7 FF          FCB $FF
0351  02F8 45          FCC 'EXT '
0352  02FC 3191   R    FDB POEXT
0353  02FE FF          FCB $FF
0354  02FF 45          FCC 'EXTE'
0355  0303 3191   R    FDB POEXT
0356  0305 FF          FCB $FF
0357  0306 46          FCC 'FCB '
0358  030A 31F3   R    FDB POFCB
0359  030C FF          FCB $FF
0360  030D 46          FCC 'FCC '
0361  0311 3228   R    FDB POFCC
0362  0313 FF          FCB $FF
0363  0314 46          FCC 'FDB '
0364  0318 32A3   R    FDB POFDB
0365  031A FF          FCB $FF
```

```
0366  031B 46          FCC 'FFP '
0367  031F 32F7   R    FDB POFFP
0368  0321 FF          FCB $FF
0369  0322 46          FCC 'FOR '     FOR KEYWORD
0370  0326 0FD9   R    FDB FORPR
0371  0328 F0          FCB $F0
0372  0329 46          FCC 'FUNC'     FUNCTION KEYWORD
0373  032D 331C   R    FDB POFUNC
0374  032F F0          FCB $F0
0375  0330 47          FCC 'GETA'     GET-ADDRESS KEYWORD
0376  0334 1384   R    FDB GETAD
0377  0336 F0          FCB $F0
0378  0337 47          FCC 'GOSU'     GOSUB KEYWORD
0379  033B 3B8F   R    FDB GOSBPR
0380  033D F0          FCB $F0
0381  033E 47          FCC 'GOTO'     GOTO KEYWORD
0382  0342 3B87   R    FDB GOTOPR
0383  0344 F0          FCB $F0
0384  0345 49          FCC 'IF  '     IF KEYWORD
0385  0349 36CC   R    FDB IFPR
0386  034B F0          FCB $F0
0387  034C 49          FCC 'IFC '
0388  0350 334B   R    FDB POIFC
0389  0352 FF          FCB $FF
0390  0353 49          FCC 'INC '
0391  0357 2B84   R    FDB ADDR3
0392  0359 0C          FCB $0C
0393  035A 49          FCC 'INPU'     INPUT KEYWORD
0394  035E 277E   R    FDB INPPR
0395  0360 F0          FCB $F0
0396  0361 49          FCC 'INS '
0397  0365 2CF3   R    FDB ADDR9
0398  0367 31          FCB $31
0399  0368 49          FCC 'INTE'     INTEGER KEYWORD
0400  036C 25C7   R    FDB INTPR
0401  036E F0          FCB $F0
0402  036F 49          FCC 'INX '
0403  0373 2CF3   R    FDB ADDR9
0404  0375 08          FCB $08
0405  0376 4A          FCC 'JMP '
0406  037A 2C6F   R    FDB ADDR7
0407  037C 6E          FCB $6E
0408  037D 4A          FCC 'JSR '
0409  0381 2C6F   R    FDB ADDR7
0410  0383 AD          FCB $AD
0411  0384 4C          FCC 'LDA '
0412  0388 2A8B   R    FDB ADDR1
0413  038A 06          FCB $06
0414  038B 4C          FCC 'LDS '
0415  038F 2BF3   R    FDB ADDR5
0416  0391 8E          FCB $8E
0417  0392 4C          FCC 'LDX '
0418  0396 2BF3   R    FDB ADDR5
0419  0398 CE          FCB $CE
0420  0399 4C          FCC 'LENG'     LENGTH KEYWORD
0421  039D 158F   R    FDB LENGPR
0422  039F F0          FCB $F0
0423  03A0 4C          FCC 'LET '     LET KEYWORD
0424  03A4 0E05   R    FDB LETPR
0425  03A6 F0          FCB $F0
0426  03A7 4C          FCC 'LSR '
0427  03AB 2B84   R    FDB ADDR3
0428  03AD 04          FCB $04
0429  03AF 4D          FCC 'MACR'
0430  03B2 3386   R    FDB POMAC
0431  03B4 FF          FCB $FF
0432  03B5 4E          FCC 'NAM '
0433  03B9 3495   R    FDB PONAM
0434  03BB FF          FCB $FF
0435  03BC 4E          FCC 'NAME'
0436  03C0 3495   R    FDB PONAM
0437  03C2 FF          FCB $FF
0438  03C3 4E          FCC 'NEG '
0439  03C7 2B84   R    FDB ADDR3
0440  03C9 00          FCB $00
0441  03CA 4E          FCC 'NEXT'     NEXT KEYWORD
0442  03CE 10A5   R    FDB NEXTPR
0443  03D0 F0          FCB $F0
0444  03D1 4E          FCC 'NIFC'
0445  03D5 34CA   R    FDB PONIF
0446  03D7 FF          FCB $FF
0447  03D8 4E          FCC 'NOP '
0448  03DC 2CF3   R    FDB ADDR9
0449  03DE 01          FCB $01
0450  03DF 4F          FCC 'ON  '     ON KEYWORD
0451  03E3 3965   R    FDB ONPR
0452  03E5 F0          FCB $F0
0453  03E6 4F          FCC 'OPEN'     OPEN-FILES KEYWORD
0454  03EA 2789   R    FDB OPENPR
0455  03EC F0          FCB $F0
0456  03ED 4F          FCC 'ORA '
0457  03F1 2A8B   R    FDB ADDR1
0458  03F3 0A          FCB $0A
0459  03F4 50          FCC 'PAG '
0460  03F8 3517   R    FDB POPAG
0461  03FA FF          FCB $FF
0462  03FB 50          FCC 'PAGE'
0463  03FF 3517   R    FDB POPAG
0464  0401 FF          FCB $FF
0465  0402 50          FCC 'POKE'     POKE KEYWORD
0466  0406 2A2A   R    FDB POKEPR
0467  0408 F0          FCB $F0
0468  0409 50          FCC 'POP '     POP KEYWORD
0469  040D 11B7   R    FDB POPPR
0470  040F F0          FCB $F0
0471  0410 50          FCC 'PRIN'     PRINT KEYWORD
0472  0414 2774   R    FDB PRNTPR
0473  0416 F0          FCB $F0
0474  0417 50          FCC 'PROC'     PROCEDURE KEYWORD
0475  041B 3ADE   R    FDB PROCPR
0476  041D F0          FCB $F0
0477  041E 50          FCC 'PSH '
0478  0422 2BD3   R    FDB ADDR4
0479  0424 36          FCB $36
0480  0425 50          FCC 'PUL '
0481  0429 2BD3   R    FDB ADDR4
0482  042B 32          FCB $32
0483  042C 50          FCC 'PUSH'     PUSH KEYWORD
0484  0430 119B   R    FDB PUSHPR
0485  0432 F0          FCB $F0
0486  0433 52          FCC 'READ'     READ-FILES KEYWORD
0487  0437 27BF   R    FDB READPR
```

```
0488  0439 F0              FCB $F0
0489  043A 52              FCC 'RETU'   RETURN KEYWORD
0490  043E 36B6    R       FDB RETPR
0491  0440 F0              FCB $F0
0492  0441 52              FCC 'REWI'   REWIND-FILES KEYWORD
0493  0445 27B6    R       FDB REWPR
0494  0447 F0              FCB $F0
0495  0448 52        RMB7  FCC 'RMB '
0496  044C 3530    R       FDB FORMB
0497  044E FF              FCB $FF
0498  044F 52              FCC 'ROL '
0499  0453 2B84    R       FDB ADDR3
0500  0455 09              FCB $09
0501  0456 52              FCC 'ROR '
0502  045A 2B84    R       FDB ADDR3
0503  045C 06              FCB $06
0504  045D 52              FCC 'RTI '
0505  0461 2CF3    R       FDB ADDR9
0506  0463 3B              FCB $3B
0507  0464 52              FCC 'RTS '
0508  0468 2CF3    R       FDB ADDR9
0509  046A 39              FCB $39
0510  046B 53              FCC 'SBA '
0511  046F 2CF3    R       FDB ADDR9
0512  0471 10              FCB $10
0513  0472 53              FCC 'SBC '
0514  0476 2A8B    R       FDB ADDR1
0515  0478 02              FCB $02
0516  0479 53              FCC 'SEC '
0517  047D 2CF3    R       FDB ADDR9
0518  047F 0D              FCB $0D
0519  0480 53              FCC 'SEI '
0520  0484 2CF3    R       FDB ADDR9
0521  0486 0F              FCB $0F
0522  0487 53              FCC 'SET '
0523  048B 358B    R       FDB POSET
0524  048D FF              FCB $FF
0525  048E 53              FCC 'SEV '
0526  0492 2CF3    R       FDB ADDR9
0527  0494 0B              FCB $0B
0528  0495 53              FCC 'STA '
0529  0499 2B1C    R       FDB ADDR2
0530  049B 07              FCB $07
0531  049C 53              FCC 'STEP'   STEP KEYWORD
0532  04A0 FFFF            FDB $FFFF
0533  04A2 F0              FCB $F0
0534  04A3 53              FCC 'STOP'   STOP KEYWORD
0535  04A7 36C4    R       FDB STOPPR
0536  04A9 F0              FCB $F0
0537  04AA 53              FCC 'STRI'   STRING KEYWORD
0538  04AE 1466    R       FDB STRFR
0539  04B0 F0              FCB $F0
0540  04B1 53              FCC 'STS '
0541  04B5 2C63    R       FDB ADDR6
0542  04B7 8F              FCB $8F
0543  04B8 53              FCC 'STX '
0544  04BC 2C63    R       FDB ADDR6
0545  04BE CF              FCB $CF
0546  04BF 53              FCC 'SUB '
0547  04C3 2A8B    R       FDB ADDR1
0548  04C5 00              FCB $00
0549  04C6 53              FCC 'SWI '
0550  04CA 2CF3    R       FDB ADDR9
0551  04CC 3F              FCB $3F
0552  04CD 54              FCC 'TAB '
0553  04D1 2CF3    R       FDB ADDR9
0554  04D3 16              FCB $16
0555  04D4 54              FCC 'TAP '
0556  04D8 2CF3    R       FDB ADDR9
0557  04DA 06              FCB $06
0558  04DB 54              FCC 'TBA '
0559  04DF 2CF3    R       FDB ADDR9
0560  04E1 17              FCB $17
0561  04E2 54              FCC 'THEN'   THEN KEYWORD
0562  04E6 36E5    R       FDB IFPR2
0563  04E8 F0              FCB $F0
0564  04E9 54              FCC 'TO  '   TO KEYWORD
0565  04ED FFFF            FDB $FFFF
0566  04EF F0              FCB $F0
0567  04F0 54              FCC 'TPA '
0568  04F4 2CF3    R       FDB ADDR9
0569  04F6 07              FCB $07
0570  04F7 54              FCC 'TST '
0571  04FB 2B84    R       FDB ADDR3
0572  04FD 0D              FCB $0D
0573  04FE 54              FCC 'TSX '
0574  0502 2CF3    R       FDB ADDR9
0575  0504 30              FCB $30
0576  0505 54              FCC 'TXS '
0577  0509 2CF3    R       FDB ADDR9
0578  050B 35              FCB $35
0579  050C 56              FCC 'VALU'   VALUE KEYWORD
0580  0510 155A    R       FDB VALUPR
0581  0512 F0              FCB $F0
0582  0513 57              FCC 'WAI '
0583  0517 2CF3    R       FDB ADDR9
0584  0519 3E              FCB $3E
0585  051A 57              FCC 'WHIL'   WHILE KEYWORD
0586  051E 3786    R       FDB WHILPR
0587  0520 F0              FCB $F0
0588  0521 57              FCC 'WRIT'   WRITE KEYWORD
0589  0525 2B60    R       FDB WRITPR
0590  0527 F0              FCB $F0
0591  0528 58              FCC 'XTRA'   (E)XTRACT KEYWORD
0592  052C 15B5    R       FDB XTRFR
0593  052E F0              FCB $F0
0594                     *
0595                     *
0596                     * CHARACTER TABLE
0597                     *
0598                     * BIT $80     ALPHABETIC CHARACTER
0599                     * BIT $40     DECIMAL DIGIT
0600                     * BIT $20     ARITHMETIC OPERATION
0601                     * BIT $10     PARENTHESIS,EQUALS
0602                     * BIT $08     NOT USED
0603                     * BIT $04     DELIMITER CHARACTER
0604                     * BIT $02     HEXADECIMAL DIGIT
0605                     * BIT $01     REGISTER NAME (A,B,X)
0606                     *
0607  052F 00            CHRTAB FCB  $00  SPACE
0608  0530 00                   FCB  $00  !
0609  0531 00                   FCB  $00  "
```

```
0532 04        FCB  $04  #
0533 04        FCB  $04  $
0534 04        FCB  $04  %
0535 00        FCB  $00  &
0536 04        FCB  $04  '
0537 10        FCB  $10  (
0538 10        FCB  $10  )
0539 24        FCB  $24  *
053A 24        FCB  $24  +
053B 04        FCB  $04  ,
053C 24        FCB  $24  -
053D 80        FCB  $80  .
053E 24        FCB  $24  /
053F 42        FCB  $42  0
0540 42        FCB  $42  1
0541 42        FCB  $42  2
0542 42        FCB  $42  3
0543 42        FCB  $42  4
0544 42        FCB  $42  5
0545 42        FCB  $42  6
0546 42        FCB  $42  7
0547 42        FCB  $42  8
0548 42        FCB  $42  9
0549 00        FCB  $00  :
054A 00        FCB  $00  ;
054B 00        FCB  $00  <
054C 10        FCB  $10  =
054D 04        FCB  $04  >
054E 00        FCB  $00  ?
054F 04        FCB  $04  @
0550 83        FCB  $83  A
0551 83        FCB  $83  B
0552 82        FCB  $82  C
0553 82        FCB  $82  D
0554 82        FCB  $82  E
0555 82        FCB  $82  F
0556 80        FCB  $80  G
0557 80        FCB  $80  H
0558 80        FCB  $80  I
0559 80        FCB  $80  J
055A 80        FCB  $80  K
055B 80        FCB  $80  L
055C 80        FCB  $80  M
055D 80        FCB  $80  N
055E 80        FCB  $80  O
055F 80        FCB  $80  P
0560 80        FCB  $80  Q
0561 80        FCB  $80  R
0562 80        FCB  $80  S
0563 80        FCB  $80  T
0564 80        FCB  $80  U
0565 80        FCB  $80  V
0566 80        FCB  $80  W
0567 81        FCB  $81  X
0568 80        FCB  $80  Y
0569 80        FCB  $80  Z
056A 04        FCB  $04  [
056B 00        FCB  $00  \
056C 04        FCB  $04  ]
056D 00        FCB  $00  CAROT
056E 00        FCB  $00  UNDERLINE
```

```
                 *
                 *
056F 0002        LNUM   RMB  2  LINE NUMBER
0571 0002        TSTPH  RMB  2  PHASING ERROR CHECK LOC
0573 0002        LC     RMB  2  LOCATION COUNTER
0575 0001        PASS   RMB  1  PASS; 0=1, FF=2
0576 0001        COMFLG RMB  1  CSECTION FLAG
0577 0001        LBFLG  RMB  1  0=NO LABEL
0578 0001        FLPFLG RMB  1  FLOATING-POINT SWITCH, 00=INTEGER
0579 0001        RFLFLG RMB  1  RELOCATION FLAG 00=NO,FF=YES
057A 0001        CMNFLG RMB  1  COMMON FLAG    "
057B 0001        EXTFLG RMB  1  EXTERNAL FLAG  "
057C 0001        ENTFLG RMB  1  ENTRY FLAG     "
057D 0002        MSTKPT RMB  2  MACRO STACK POINTER
057F 0002        STKSAV RMB  2  TEMP  STACK POINTER
0581 0002        MXSAV1 RMB  2  TEMP.
0583 0002        MXSAV2 RMB  2
0585 0050        MACLIN RMB 80  MACRO LINE EXPANSION SPACE
0505 0050        MACPAR RMB 80  MACRO PARAMETER SPACE
0625 0001        MACFLG RMB  1  MACRO MODE FLAG
0626 0002        MACPTR RMB  2  POINTER TO MACRO TABLE
0628 0002        MACSAV RMB  2
062A 0002        MCLPTR RMB  2  MACRO LINE POINTER
062C 0002        LABEL  RMB  2  AUTO-GEN. LINE NO.
062E 0001        PARNUM RMB  1  NO  OF PARAMETERS
062F 0010               RMB 16
063F 063F      R IFSTK  EQU  *  COND. STACK
063F 0002        @IFSTK RMB  2  COND  STACK POINTER
0641 0001        CIFFLG RMB  1  COND  FLAG
0642 0002        DESCRA RMB  2  DESCRIPTOR ADDRESS
0644 0001        DESCRC RMB  1  DESCRIPTOR COUNT
0645 0002        MYSAV  RMB  2  POINTER STORAGE
0647 0002        MYSAV2 RMB  2  """
0649 000A        OPTLN  RMB 10  BUFFER SPACE
0653 0001        CONSW  RMB 1
0654 0001        LPTSW  RMB 1
                 *
0655 0D0A        CMPLNG FDB $0D0A   CR,LF
0657 53                 FCC 'STRUBAL+ COMPILER.   VERSION 1 1'
0677 0D                 FCB $0D
0678 43                 FCC 'COPYRIGHT 1978 BY'
0689 0D                 FCB $0D
068A 52                 FCC 'ROBERT GRAPPEL, LEXINGTON, MASS '
06AA 0D                 FCB $0D
06AB 41                 FCC 'AND JACK HEMENWAY, BOSTON MASS '
06CA 0D                 FCB $0D
06CB 41                 FCC 'ALL RIGHTS RESERVED'
                 *
06DE 0D0A        OPTPMT FDB $0D0A   CR,LF
06E0 45                 FCC 'ENTER OPTIONS  '
06FF 04                 FCB $04     EOT
                 *
06F0 0001        CMPCNT RMB 1   COMPILER CHAR  COUNTER
06F1 0002        CUCHAR RMB 2   CURRENT CHAR ADDRESS
06F3 0001        CFLAG  RMB 1   COMPILING FLAG
06F4 0002        CULINE RMB 2   CURRENT LINE ADDRESS
06F6 0002        SYMPTR RMB 2   SYMTAB POINTER
06F8 0001        OPTNS  RMB 1   ASSEMBLER OPTIONS
06F9 0001        LCN    RMB 1   # BYTES IN AN INSTRUCTION
06FA 0002        LSAVE  RMB 2   LC SAVE LOCATION
06FC 0001        LCOUNT RMB 1   # LINES ON A PAGE
```

```
06FD 0001        IFFLG  RMB 1   IF-PROCESS FLAG
06FE 0002        IFLBL  RMB 2   IF-BRANCH LABEL
0700 0001        FORFLG RMB 1   FOR-PROCESS FLAG
0701 0001        PFLAG  RMB 1   PARAMETER FLAGS
0702 0001        PARFLG RMB 1
0703 0002        ECOUNT RMB 2   ERROR COUNT
0705 0002        CMNLC  RMB 2   COMMON LC
0707 0001        NXTFLG RMB 1   NEXT-TOKEN FLAG
0708 0002        LBLCNT RMB 2   COUNT OF COMPILER LABELS
                 *
                 *
070A 7E 0000 X          EXT  INEEE  INPUT A CHARACTER
070D 7E 0000 X          EXT  OUTLEE OUTPUT A CHARACTER
0710 7E 0000 X          EXT  OUTLPT OUTPUT TO LINEPRINTER
0713 0002        STACK  RMB 2
0715 0042        ES     EQU $0042     ESCAPE CHARACTER
                 *
                 *
0715 BD 3E0B R   OPC1   JSR INITIO      INIT. I/O
0718 7F 0653 R          CLR CONSW
071B 7F 0654 R          CLR LPTSW
071E 7C 0653 R          INC CONSW       OUTPUT TO CONSOLE
0721 CE 0655 R          LDX #CMPMNG     WRITE "COMPILING" PROMPT
0724 BD 3E8F R          JSR PDATA1
0727 BD 2D12 R          JSR ADRINT      CLEAR AND INITIALIZE
072A 7F 0576 R          CLR COMFLG
072D 7F 06FD R          CLR IFFLG
0730 CE 389C R          LDX #BLKEND
0733 FF 389D R          STX BLKCNT
                 *
                 * SET PROCESSING OPTIONS.
                 * (L) LIST
                 * (M) LIST MACRO EXPANSION
                 * (S) SYMBOL TABLE
                 * (O) OBJECT FILE
                 * (X) CROSS-REFERENCE OUTPUT
                 * (I) INTERMEDIATE CODE (COMPILE ONLY)
                 * (Q) QUIT---RETURN TO OPERATING SYSTEM
                 *
0736 7F 06F8 R   OPC1A  CLR OPTNS
0739 73 06F8 R          COM OPTNS       ALL OPTIONS OFF
                 *
073C BD 070A R   OPC2   JSR INEEE       READ
073F 85 40              BIT A #$40      ALPHABETIC?
0741 27 02              BEQ *+4         NO
                 *
0743 84 DF              AND A #$DF      FORCE UPPER CASE
                 *
0745 91 42              CMP A ES        ESCAPE?
0747 26 11              BNE OPC2B
                 *
0749 CE 06DE R   OPC2A  LDX #OPTPMT     RE-PROMPT FOR OPTIONS
074C 7F 0654 R          CLR LPTSW
074F 7F 0653 R          CLR CONSW
0752 7C 0653 R          INC CONSW       OUTPUT TO CONSOLE
0755 BD 3E8F R          JSR PDATA1
0758 20 DC              BRA OPC1A       TRY AGAIN
                 *
075A 81 0D       OPC2B  CMP A #$0D      CARRIAGE RETURN?
075C 27 3D              BEQ OPC3        YES, DONE WITH OPTIONS
                 *
075F 81 51              CMP A #'Q       QUIT?
0760 26 03              BNE *+5         NO
                 *
0762 7E 3ED2 R          JMP MONITOR     RETURN TO MONITOR
                 *
0765 81 4C              CMP A #'L       LIST?
0767 26 04              BNE *+6         NO
                 *
0769 86 7F              LDA A #$7F
076B 20 26              BRA OPYES       YES
                 *
076D 81 4D              CMP A #'M       MACRO EXP  LIST?
076F 26 04              BNE *+6         NO
                 *
0771 86 FB              LDA A #$FB      YES
0773 20 1E              BRA OPYES
                 *
0775 81 4F              CMP A #'O       OBJECT FILE?
0777 26 04              BNE *+6         NO
                 *
0779 86 BF              LDA A #$BF
077B 20 16              BRA OPYES       YES
                 *
077D 81 53              CMP A #'S       SYMBOL TABLE?
077F 26 04              BNE *+6         NO
                 *
0781 86 DF              LDA A #$DF
0783 20 0E              BRA OPYES       YES
                 *
0785 81 58              CMP A #'X       CROSS-REFERENCE?
0787 26 04              BNE *+6         NO
                 *
0789 86 F7              LDA A #$F7
078B 20 06              BRA OPYES       YES
                 *
078D 81 49              CMP A #'I       INTERMEDIATE CODE?
078F 26 AB              BNE OPC2        NO
                 *
0791 86 EF               LDA A #$EF
0793 B4 06F8 R   OPYES   AND A OPTNS
0796 B7 06F8 R           STA A OPTNS    TURN ON OPTION
0799 20 A1               BRA OPC2       CONTINUE
                 *
079B BD 3E84 R   OPC3    JSR CRLF
079E BD 3EC3 R           JSR OPENI      OPEN INPUT FILE
07A1 25 F8               BCS OPC3       ERROR?
07A3 29 F6               BVS OPC3       ERROR?
                 *
07A5 B6 06F8 R           LDA A OPTNS    TEST OPTIONS
07A8 43                  COM A
07A9 85 48               BIT A #$48     XREF OR OBJECT?
07AB 27 07               BEQ OPC3A      NO
                 *
07AD BD 3EC6 R           JSR OPENO      OPEN OUTPUT FILE
07B0 25 FB               BCS *-3        ERROR?
07B2 29 F9               BVS *-5        ERROR?
                 *
07B4 CE 07E2 R   OPC3A   LDX #CORASM
07B7 BD 3E8F R           JSR PDATA1     COMPILE-ASSEMBLE PROMPT
07BA 7F 0649 R           CLR OPTLN
07BD BD 070A R   OPC3B   JSR INEEE      GET CHARACTER
```

```
07C0 91 42            CMP A ES        ESCAPE?
07C2 27 F0            BEQ OPC3A       YES, REPROMPT
                *
07C4 81 0D            CMP A #$0D      C.R ?
07C6 27 05            BEQ OPC3C       YES
                *
07C8 B7 0649 R        STA A OPTLN     SAVE CHARACTER
07CB 20 F0            BRA OPC3B       LOOP BACK
                *
07CD BD 3E84 R OPC3C  JSR CRLF
07D0 B6 0649 R        LDA A OPTLN     GET CHAR
07D3 84 DF            AND A #$DF      FORCE UPPER CASE
07D5 81 41            CMP A #'A       ASSEMBLE?
07D7 26 03            BNE *+5         NO
                *
07D9 7E 08D7 R        JMP PASS1       YES, BEGIN ASSEMBLY
                *
07DC 81 43            CMP A #'C       COMPILE?
07DE 26 D4            BNE OPC3A       IF NOT, PROMPT AGAIN
                *
07E0 20 1C            BRA OPC4        YES, BEGIN COMPILATION
                *
07E2 0D0A       CORASM FDB $0D0A      CR,LF
07E4 28               FCC '(C)OMPILE OR (A)SSEMBLE? '
07F0 04               FCB $04
                *
                *
07FE CE 08C0 R OPC4   LDX #RTMSG       RUN-TIME NAME PROMPT
0801 BD 3E8F R        JSR PDATA1
0804 CE 085D R        LDX #RTL1+5     GET NAME OF PROGRAM
0807 BD 070A R OPC5   JSR INEEE       GET A CHARACTER
080A A7 00            STA A 0,X       STORE IT
080C 81 0D            CMP A #$0D      CR?
080E 27 0A            BEQ OPC6        IF SO, DONE
                *
0810 91 42            CMP A ES        ESCAPE?
0812 27 EA            BEQ OPC4        YES, DROP LINE
                *
0814 08               INX
0815 8C 0867 R        CPX #RTL1+15    TOO LONG?
0818 26 ED            BNE OPC5        NO, CONTINUE
                *
081A BD 3E84 R OPC6   JSR CRLF
081D 86 FF            LDA A #$FF
081F B7 0575 R        STA A PASS      MAKE ONE PASS
0822 B7 06F3 R        STA A CFLAG     SET COMPILE FLAG
                *
                *
                * OUTPUT RUN-TIME LINKS
                *
0825 CE 0858 R        LDX #RTL1
0828 BD 13AB R        JSR STR1
082B CE 0868 R        LDX #RTL2
082E BD 13AB R        JSR STR1
0831 CE 0877 R        LDX #RTL3
0834 BD 13AB R        JSR STR1
0837 CE 0886 R        LDX #RTL4
083A BD 13AB R        JSR STR1
083D CE 0892 R        LDX #RTL5
0840 BD 13AB R        JSR STR1
0843 CE 089E R        LDX #RTL6
0846 BD 13AB R        JSR STR1
0849 CE 08AC R        LDX #RTL7
084C BD 13AB R        JSR STR1
084F CE 08B5 R        LDX #RTL8
0852 BD 13AB R        JSR STR1
0855 7E 08DD R        JMP PASS1A
                *
0858 20         RTL1  FCC ' NAM '
085D 000A             RMB 10
0867 0D               FCB $0D
0868 20         RTL2  FCC ' CMN  RNSTK,80'
0876 0D               FCB $0D
0877 20         RTL3  FCC ' CMN .ACCUM,21'
0885 0D               FCB $0D
0886 20         RTL4  FCC ' CMN .EOF,2'
0891 0D               FCB $0D
0892 20         RTL5  FCC ' CMN  ERR,2'
089D 0D               FCB $0D
089E 20         RTL6  FCC ' CMN  CURID,2'
08AB 0D               FCB $0D
08AC 20         RTL7  FCC ' BRA *+5'
08B4 0D               FCB $0D
08B5 20         RTL8  FCC ' EXT  RTBL'
08BF 0D               FCB $0D
                *
08C0 0D0A       RTMSG FDB $0D0A
08C2 45               FCC 'ENTER PROGRAM NAME. '
08D6 04               FCB $04
                *
                *
08D7 7F 0575 R PASS1  CLR PASS        PASS.=1
08DA 7F 06F3 R        CLR CFLAG       ASSEMBLY
                *
08DD BD 238A R PASS1A JSR SPACER      TOP OF PAGE
08E0 86 20            LDA A #$20      BLANKS TO SYMTAB
08E2 FE 008A R        LDX SYMTAB      POINT TO SYMTAB
08E5 A7 00            STA A 0,X       BLANK LOCATION
08E7 08               INX             BUMP POINTER
08E8 BC 008C R        CPX SYMEND      ALL DONE ?
08EB 26 F8            BNE *-6         NO
                *
08ED CE 0000          LDX #$0000
08F0 FF 0571 R        STX TSTPH       CLEAR TSTPH
08F3 FF 0703 R        STX ECOUNT      CLEAR ECOUNT
08F6 FF 0705 R        STX CMNLC       INIT COMMON LC
08F9 FF 1BCD R        STX RUNCNT      INIT RUN-TIME STACK POINTER
08FC FF 070B R        STX LBLCNT      INIT COMPILER LABELS
08FF FF 139A R        STX GAFLG       NOT GET-ADDRESS
                *
0902 7F 06FC R PASS2  CLR LCOUNT      LCOUNT=0
0905 CE 0000          LDX #$0000
0908 FF 0573 R        STX LC          INIT LC
090B FF 056F R        STX LNUM        INIT LNUM
090E FF 0084 R        LDX MACTBL
0911 FF 0626 R        STX MACPTR      INIT MACPTR
0914 7F 0625 R        CLR MACFLG      INIT MACFLG
0917 FE 0088 R        LDX MACSTK
091A FF 057D R        STX MSTKPT      INIT MSTKPT
091D 86 FF            LDA A #$FF
091F B7 0641 R        STA A CIFFLG    INIT CIFFLG
0922 CE 063F R        LDX #IFSTK
```

```
0925 FF 063F R        STX @IFSTK      INIT @IFSTK
0928 B6 06F8 R        LDA A OPTNS     CHECK OPTIONS
092B 84 94            AND A #$94      LIST/MACRO/INTERM  ?
092D 88 94            EOR A #$94      ANY BITS ZERO?
092F 27 06            BEQ MAIN1       NO
                *
0931 7F 0653 R        CLR CONSW
0934 7C 0654 R        INC LPTSW       OUTPUT TO LINEPRINTER
                *
0937 BD 0B2E R MAIN1  JSR RDLINE      GET A LINE OF SOURCE
093A 7F 0707 R        CLR NXTFLG
093D 7F 25C6 R        CLR DSTFLG      SET FLAG TO NO STRING
0940 7F 0577 R        CLR LBFLG       SET FLAG TO NO LABEL
0943 7F 0578 R        CLR FLPFLG      SET TO INTEGER MODE
0946 7F 22FF R        CLR OVFLG       SET FLAG TO NO OVERFLOW
0949 7F 139B R        CLR GAFLG2      SET FLAG TO NO GET-ADDRESS
094C 7F 24CC R        CLR ERNUM       NO ERRORS YET
094F 7F 0700 R        CLR FORFLG      SET FLAG FOR NO "FOR" STMT.
0952 BD 2D12 R        JSR ADRINT      INITIALIZE ASSEMBLY FLAGS
                *
0955 FE 056F R MAIN2  LDX LNUM
0958 08               INX
0959 FF 056F R        STX LNUM        BUMP LNUM
095C FE 06F4 R        LDX CULINE      POINT TO LINE
095F A6 00            LDA A 0,X       GET COL 1
0961 81 2A            CMP A #'*       COMMENT?
0963 26 10            BNE MAIN3       NO
                *
0965 BD 2372 R        JSR PRINTL      PRINT THE LINE
0968 7D 06F3 R        TST CFLAG       COMPILING?
096B 27 CA            BEQ MAIN1       NO, DONE
                *
096D FE 06F4 R        LDX CULINE      RESTORE LINE POINTER
0970 BD 13AB R        JSR STR1        PASS COMMENT ACROSS
0973 20 C2            BRA MAIN1       CONTINUE
                *
0975 7D 06F3 R MAIN3  TST CFLAG       COMPILING?
0978 26 03            BNE *+5
097A 7E 0A69 R        JMP MAIN3A      NO
                *
                * PROCESSING FOR COMPILER HERE
                *
097D BD 2372 R        JSR PRINTL      PRINT LINE
0980 7D 06FD R        TST IFFLG       A PENDING IF?
0983 27 15            BEQ COMP0       NO
                *
0985 7F 06FD R        CLR IFFLG
0988 B6 06FE R        LDA A IFLBL     SET UP FOR CVBTD
098B F6 06FF R        LDA B IFLBL+1
098E CE 3B98 R        LDX #LABL+1
0991 BD 2487 R        JSR CVBTD
                *
                * PUT OUT BRANCH LABEL HERE
                * IF A PENDING "IF" STATEMENT
                *
                *  ------ EQU *
                *
0994 CE 3B97 R        LDX #LABL
0997 BD 13AB R        JSR STR1        OUTPUT CODE
099A FE 06F4 R COMP0  LDX CULINE
099D A6 00            LDA A 0,X       RESET POINTERS
099F 81 20            CMP A #$20      A LABEL?
09A1 27 13            BEQ COMP1       NO
                *
09A3 BD 172F R        JSR NXTOK       GET LABEL
09A6 C1 01            CMP B #$01      A VALID LABEL?
09A8 27 03            BEQ *+5         YES
                *
09AA 7E 0AB7 R        JMP MAIN3E      NO, ERROR
                *
09AD 7C 0577 R        INC LBFLG       SET LABEL FLAG
09B0 BD 0DA6 R        JSR OUTNAM      OUTPUT LABEL TO LIST AND OBJECT
09B3 BD 1992 R        JSR XREF        HANDLE X-REFERENCE
                *
09B6 C6 20      COMP1 LDA B #$20      BLANK TO OBJECT
09B8 BD 0DA3 R        JSR TPCHAR
09BB 86 20            LDA A #$20
09BD BD 0D98 R        JSR WRTCHR      BLANK TO LIST
09C0 BD 172F R        JSR NXTOK       GET MNEMONIC
09C3 C1 01            CMP B #1        A VALID TOKEN?
09C5 26 65            BNE COMP2E      NO, ERROR
                *
09C7 BD 1B52 R COMP2  JSR MNLKP       SEARCH MNTAB
09CA 4D               TST A           FOUND MNEMONIC?
09CB 27 50            BEQ COMP2D      YES
                *
                * LOOK FOR USER-DEFINED FUNCTION
                *
09CD BD 1A5C R        JSR LKPSYM      NAME IN SYMTAB?
09D0 C1 FF            CMP B #$FF
09D2 27 46            BEQ COMP2C      NO
                *
09D4 C5 20            BIT B #$20      NAME OF FUNCTION?
09D6 27 42            BEQ COMP2C      NO
                *
09D8 FF 24C8 R        STX SAVEX       SAVE INDEX
09DB CE 3B9D R        LDX #LABL+6     OUTPUT DUMMY LABEL
09DE BD 13AB R        JSR STR1
09E1 FE 24C8 R        LDX SAVEX       RESTORE INDEX
09E4 7D 0625 R        TST MACFLG      IN FUNCTION MODE?
09E7 27 0B            BEQ COMP2A      NO
                *
                * PUSH PRESENT FUNCTION ONTO STACK
                *
09E9 BD 0CD8 R        JSR MACPSH
09EC 7D 0625 R        TST MACFLG      ERRORS?
09EF 26 03            BNE COMP2A      NO
                *
09F1 7E 0937 R        JMP MAIN1       ERRORS, QUIT
                *
09F4 FF 0626 R COMP2A STX MACPTR      SAVE TABLE LOC
09F7 FE 0708 R        LDX LBLCNT      MAKE NEW LABEL
09FA 08               INX
09FB FF 0708 R        STX LBLCNT
09FE FF 062C R        STX LABEL
0A01 7F 062E R        CLR PARNUM      NO PARAMETERS
0A04 7C 0625 R        INC MACFLG      SET FUNCTION FLAG
0A07 BD 172F R        JSR NXTOK       LOOK FOR PARAMETERS
0A0A C1 0D            CMP B #$0D      EOL?
0A0C 26 06            BNE COMP2B      NO
                *
0A0E F7 05D5 R        STA B MACPAR    YES, NO PARAMS
```

```
0A11 7E 0937 R          JMP MAIN1        SET TO EXPAND
                  *
0A14 BD 0D6F R COMP2B JSR PARMVE       GET PARAMETER LIST
0A17 7E 0937 R          JMP MAIN1        SET TO EXPAND
                  *
0A1A 7E 0E1D R COMP2C JMP LETPR2       ASSUMED ASSIGNMENT
                  *
0A1D 8C FFFF    COMP2D CPX #$FFFF       SPECIAL NAME (STEP,TO)?
0A20 27 0A              BEQ COMP2E       YES, ERROR
                  *
0A22 8C 36E5 R          CPX #IFPR2       'THEN'?
0A25 27 05              BEQ COMP2E       YES, ERROR
                  *
0A27 8C 36EC R          CPX #IFPR3       'ELSE'?
0A2A 26 09              BNE COMP2F       NO, O K.
                  *
0A2C CE 0202    COMP2E LDX #$0202       ERROR MESSAGE
0A2F BD 24E8 R          JSR PRINTE
0A32 7E 0937 R          JMP MAIN1
                  *
0A35 C1 F0      COMP2F CMP B #$F0       COMPILER OP ?
0A37 27 2E              BEQ COMP3        YES
                  *
0A39 8C 3517 R          CPX #POPAG       PAGE DIRECTIVE?
0A3C 26 0D              BNE COMP2H       NO
                  *
0A3E F6 06FC R          LDA B LCOUNT     AT TOP OF PAGE?
0A41 27 1B              BEQ COMP2G       YES
                  *
0A43 7F 06FC R          CLR LCOUNT
0A46 BD 238A R          JSR SPACER       TOP-OF-FORM
0A49 20 13              BRA COMP2G       PASS THROUGH
                  *
0A4B 8C 2ED4 R COMP2H CPX #POEND       END STMNT?
0A4E 27 17              BEQ COMP3        YES
                  *
0A50 8C 3495 R          CPX #PONAM       NAM DIRECTIVE?
0A53 26 09              BNE COMP2G       NO
                  *
0A55 CE 0201            LDX #$0201       NAMS SKIPPED
0A58 BD 24E8 R          JSR PRINTE
0A5B 7E 0937 R          JMP MAIN1
                  *
                  * ASSEMBLER LINES ARE PASSED ACROSS
0A5E FE 0642 R COMP2G LDX DESCRA
0A61 BD 13AB R          JSR STR1
0A64 7E 0937 R          JMP MAIN1        GET NEW LINE
                  *
                  * COMPILER LINE PROCESSED HERE
0A67 6E 00      COMP3  JMP 0,X           GO TO PROCESSING
                  *
                  * PROCESSING ROUTINE RETURNS TO MAIN1
                  *
                  * PROCESSING FOR ASSEMBLER HERE
                  *
0A69 7D 0641 R MAIN3A TST CIFFLG       ASSEMBLING?
0A6C 26 20              BNE MAIN3C       YES
                  *
0A6E 81 20              CMP A #$20       LINE HAS LABEL?
0A70 27 03              BEQ MAIN3B       NO
                  *
0A72 BD 172F R          JSR NXTOK        SKIP OVER LABEL
0A75 BD 172F R MAIN3B JSR NXTOK        GET MNEMONIC
0A78 BD 1B52 R          JSR MNLKP        IN MNTAB?
0A7B 8C 334B R          CPX #POIFC       "IFC" PSEUDOP?
0A7E 27 E7              BEQ COMP3        YES, PROCESS IT
                  *
0A80 8C 34CA R          CPX #PONIF       "NIFC" PSEUDOP?
0A83 27 E2              BEQ COMP3        YES, PROCESS IT
                  *
0A85 BD 2D12 R          JSR ADRINT       INIT. FLAGS
0A88 BD 2372 R          JSR PRINTL       LIST LINE
0A8B 7E 0937 R          JMP MAIN1        DONE!
                  *
0A8E 81 20      MAIN3C CMP A #$20       LINE HAS LABEL?
0A90 27 03              BEQ MAIN3D       NO
                  *
0A92 BD 172F R          JSR NXTOK        SKIP OVER LABEL
0A95 BD 172F R MAIN3D JSR NXTOK        GET MNEMONIC
0A98 BD 1B52 R          JSR MNLKP        IN MNTAB?
0A9B 8C 358B R          CPX #POSET       "SET" PSEUDOP?
0A9E 27 C7              BEQ COMP3        YES, PROCESS IT
                  *
0AA0 FE 06F4 R          LDX CULINE
0AA3 FF 06F1 R          STX CUCHAR       RE-INIT  POINTERS
0AA6 A6 00              LDA A 0,X
                  *
0AA8 81 20              CMP A #$20       COL 1 BLANK?
0AAA 27 1E              BEQ MAIN5        YES
                  *
0AAC BD 172F R          JSR NXTOK        GET LABEL
0AAF C1 01              CMP B #$01       OK?
0AB1 27 09              BEQ MAIN4        YES
                  *
0AB3 81 01              CMP A #$01       A,B, OR X?
0AB5 27 05              BEQ MAIN4        TREAT LIKE LABEL
                  *
0AB7 CE 0205    MAIN3E LDX #$0205       ERROR
0ABA 20 69              BRA MAIN8
                  *
0ABC 7C 0577 R MAIN4  INC LBFLG        SET LABEL FLAG
0ABF 7D 0575 R          TST PASS         PASS?
0AC2 26 03              BNE MAIN4A       PASS2
                  *
0AC4 BD 19FA R          JSR STOSYM       STORE LABEL IN SYMTAB
                  *
0AC7 BD 1992 R MAIN4A JSR XREF         HANDLE CROSS-REFERENCE
                  *
0ACA BD 172F R MAIN5  JSR NXTOK        GET MNEMONIC
0ACD C1 01              CMP B #$01       VALID?
0ACF 27 05              BEQ MAIN6A       YES
                  *
0AD1 CE 0202    MAIN6  LDX #$0202       ERROR
0AD4 20 4F              BRA MAIN8
                  *
0AD6 BD 1B52 R MAIN6A JSR MNLKP        SEARCH MNTAB
0AD9 4D                 TST A            IN MNTAB?
0ADA 27 44              BEQ MAIN6D       YES
                  *
                  * LOOK FOR USER MACRO NAME
                  *
0ADC BD 1A5C R          JSR LKPSYM       NAME IN SYMTAB?
```

```
0ADF C1 FF            CMP B #$FF
0AE1 27 3F            BEQ MAIN7        NO
                  *
0AE3 C5 20            BIT B #$20       MACRO NAME?
0AE5 27 3B            BEQ MAIN7        NO
                  *
0AE7 7D 0625 R        TST MACFLG       IN MACRO MODE?
0AEA 27 0B            BEQ MAIN6B       NO
                  *
                  * PUSH PRESENT MACRO ONTO STACK
                  *
0AEC BD 0CD8 R        JSR MACPSH
0AEF 7D 0625 R        TST MACFLG       ERRORS?
0AF2 26 03            BNE MAIN6B       NO
                  *
0AF4 7E 0937 R        JMP MAIN1        YES, QUIT
                  *
0AF7 FF 0626 R MAIN6B STX MACPTR       SAVE MACRO LOC.
0AFA FE 0708 R        LDX LBLCNT       MAKE NEW AUTO-LABEL
0AFD 08               INX
0AFE FF 0708 R        STX LBLCNT
0B01 FF 062C R        STX LABEL
0B04 7F 062E R        CLR PARNUM       NO PARAMETERS
0B07 7C 0625 R        INC MACFLG       SET MACRO MODE FLAG
0B0A BD 2372 R        JSR PRINTL       LIST MACRO CALL
0B0D BD 172F R        JSR NXTOK        LOOK FOR PARAMETERS
0B10 C1 0D            CMP B #$0D       EOL?
0B12 26 06            BNE MAIN6C       NO
                  *
0B14 F7 05D5 R        STA B MACPAR     YES, NO PARAMS
0B17 7E 0937 R        JMP MAIN1        SET FOR EXPANSION
                  *
0B1A BD 0D6F R MAIN6C JSR PARMVE       GET PARAMETERS
0B1D 7E 0937 R        JMP MAIN1        SET FOR EXPANSION
                  *
0B20 6E 00     MAIN6D JMP 0,X          GO TO ROUTINE
                  * ROUTINES RETURN TO MAIN1
                  *
0B22 CE 0207   MAIN7  LDX #$0207        ERROR
0B25 BD 24E8 R MAIN8  JSR PRINTE        PRINT IT
0B28 BD 2372 R        JSR PRINTL        PRINT LINE
0B2B 7E 0937 R        JMP MAIN1        PROCESS NEXT LINE
                  *
                  *
                  * GET A LINE OF SOURCE FROM INBUF *
                  * RETURNS ADDRESS OF LINE IN CULINE *
                  * CUCHAR:=ADDRESS OF FIRST CHARACTER*
                  *
0B2E 7D 0625 R RDLINE TST MACFLG       IN MACRO MODE?
0B31 27 09            BEQ RDLINA       NO
                  *
0B33 BD 0BC7 R        JSR RDMAC        YES, EXPAND MACRO
0B36 7D 0625 R        TST MACFLG       FULLY EXPANDED?
0B39 27 01            BEQ RDLINA       YES
                  *
0B3B 39               RTS
                  *
0B3C CE 0B77 R RDLINA LDX #INBUF
0B3F FF 06F1 R        STX CUCHAR
0B42 FF 06F4 R        STX CULINE
                  *
0B45 BD 3ED5 R RDL1   JSR GETB         GET A BYTE
0B48 24 12            BCC RDL1A        OK
                  *
0B4A BD 3FCC R        JSR CLOSI        CLOSE FILE
0B4D BD 3EC3 R        JSR OPENI        OPEN NEW FILE
0B50 29 FB            BVS *-3          ERROR CHECK
0B52 25 02            BCS *+4
0B54 20 E6            BRA RDLINA
                  *
0B56 BE 0713 R        LDS STACK        EOF, FLUSH STACK
0B59 7E 2EDA R        JMP POENDO       FORCE "END"
                  *
0B5C 81 0A     RDL1A  CMP A #$0A       LF?
0B5E 27 E5            BEQ RDL1         YES
                  *
0B60 81 00            CMP A #$00       NULLS?
0B62 27 E1            BEQ RDL1         YES
                  *
0B64 BC 0BC6 R        CPX #INBUF+79    LINE TOO LONG?
0B67 27 05            BEQ RDL2         YES
                  *
0B69 A7 00            STA A 0,X        SAVE CHARACTER
0B6B 08               INX
0B6C 20 04            BRA RDL3
                  *
0B6E C6 0D     RDL2   LDA B #$0D       TRUNCATE LINE
0B70 E7 00            STA B 0,X
                  *
0B72 81 0D     RDL3   CMP A #$0D       CR?
0B74 26 CF            BNE RDL1         NO
                  *
0B76 39               RTS
                  *
0B77 0050      INBUF  RMB  80          INPUT BUFFER FOR LINE
                  *
                  * RDMAC  EXPAND MACRO OR FUNCTION CALL
                  *
0BC7 FE 0626 R RDMAC  LDX MACPTR
0BCA A6 00            LDA A 0,X        GET A CHARACTER
0BCC 81 17            CMP A #$17       ETB?
0BCE 26 0B            BNE RDMAC1       NO
                  *
0BD0 7A 0625 R        DEC MACFLG       DEC  MODE COUNT
0BD3 27 05            BEQ RDMACO       NO MORE MACROS
                  *
                  * PULL UP LAST STACKED MACRO
                  *
0BD5 BD 0D40 R        JSR MACPUL
0BD8 20 ED            BRA RDMAC        CONTINUE EXPANSION
                  *
0BDA 39        RDMACO RTS              DONE!
                  *
0BDB CE 0585 R RDMAC1 LDX #MACLIN       POINT TO MACRO EXP. SPACE
0BDE FF 06F4 R        STX CULINE       INIT  POINTERS
0BE1 FF 06F1 R        STX CUCHAR
0BE4 FF 062A R        STX MCLPTR
0BE7 FE 0626 R RDMAC2 LDX MACPTR       POINT TO MACRO DEF
0BEA A6 00            LDA A 0,X        GET A CHARACTER
0BEC 08               INX
0BED FF 0626 R        STX MACPTR       MOVE POINTER
0BF0 81 26            CMP A #'&        PARAMETER?
```

```
0BF2 27 5B            BEQ RDMAC3       YES
                 *
0BF4 81 21            CMP A #'!        AUTO-DEF  LABEL?
0BF6 26 41            BNE RDM2         NO
                 *
                 * AUTO-DEFINE LABEL ( . + ALPHACHAR + "LABEL" )
                 *
0BF8 86 2E            LDA A #'.        FORM NEW LABEL
0BFA FF 062A R        LDX MCLPTR       FIRST CHAR=.
0BFD A7 00            STA A 0,X
0BFF 08               INX
0C00 B6 062C R        LDA A LABEL
0C03 F6 062D R        LDA B LABEL+1    SET UP FOR CVBTD
0C06 BD 2487 R        JSR CVBTD        MAKE "LABEL" INTO ASCII
0C09 FE 0626 R        LDX MACPTR
0C0C A6 00            LDA A 0,X        GET CHARACTER FROM MACRO
0C0E BD 19C5 R        JSR GCHRTB       CHECK FOR ALPHABETIC?
0C11 85 80            BIT A #$80       O.K ?
0C13 26 03            BNE *+5          YES
                 *
0C15 7E 0CBF R        JMP RDMERR       ERROR
                 *
0C18 FE 0626 R        LDX MACPTR       RESTORE POINTER
0C1B A6 00            LDA A 0,X        GET CHAR
0C1D 08               INX
0C1E FF 0626 R        STX MACPTR
0C21 FE 062A R        LDX MCLPTR
0C24 A7 01            STA A 1,X        PUT CHAR.  IN LABEL
0C26 C6 06            LDA B #6         SIX-CHAR LABEL
0C28 08          RDM1 INX
0C29 FF 062A R        STX MCLPTR       MOVE POINTER
0C2C 8C 05CF R        CPX #MACLIN+74   OVERFLOW?
0C2F 26 03            BNE *+5          NO
                 *
0C31 7E 0CBF R        JMP RDMERR       YES. ERROR
                 *
0C34 5A               DEC B            DONE?
0C35 26 F1            BNE RDM1         NO
                 *
0C37 20 AE            BRA RDMAC2       YES. DONE WITH LABEL
                 *
                 * END AUTO-LABEL SECTION
                 *
0C39 FE 062A R RDM2   LDX MCLPTR       POINT TO MACLINE
0C3C A7 00            STA A 0,X        STORE CHAR
0C3E 08               INX
0C3F FF 062A R        STX MCLPTR       MOVE POINTER
0C42 8C 05CF R        CPX #MACLIN+74   OVERFLOW?
0C45 26 03            BNE *+5          NO
                 *
0C47 7E 0CBF R        JMP RDMERR       YES. ERROR
                 *
0C4A 81 0D            CMP A #$0D       EOL?
0C4C 26 99            BNE RDMAC2       NO. CONTINUE
                 *
0C4E 39               RTS              YES. DONE!
                 *
                 * SUBSTITUTE POSITIONAL PARAMETERS
                 *
0C4F BD 225A R RDMAC3 JSR CVHBS        GET POSITION NUMBER,IN BINARY
0C52 B1 062E R        CMP A PARNUM     POS  NUMBER TOO BIG?
0C55 2F 03            BLE *+5          NO. O.K.
                 *
0C57 B6 062E R        LDA A PARNUM     YES. DEFAULT POS. NUMBER
                 *
0C5A 16               TAB
0C5B 08               INX
0C5C FF 0626 R        STX MACPTR       SKIP OVER POS  NUMBER
0C5F 5D               TST B            PARAMETER ZERO?
0C60 26 1B            BNE RDM3         NO
                 *
                 * PARAMETER ZERO=NUMBER OF PARAMETERS IN CALL
                 *
0C62 86 24            LDA A #'$        OUTPUT HEX CONSTANT
0C64 FE 062A R        LDX MCLPTR
0C67 A7 00            STA A 0,X
0C69 B6 062E R        LDA A PARNUM     GET PARAMETER COUNT
0C6C BD 3E75 R        JSR OUTHL        LEFT NIBBLE ASCII
0C6F A7 01            STA A 1,X
0C71 B6 062E R        LDA A PARNUM
0C74 BD 3E79 R        JSR OUTHR        RIGHT NIBBLE ASCII
0C77 A7 02            STA A 2,X
0C79 C6 03            LDA B #3         3 CHARS  IN NUMBER
0C7B 20 AB            BRA RDM1         FINISH UP
                 *
                 * END OF PARAMETER-ZERO SECTION
                 *
0C7D CE 05D5 R RDM3   LDX #MACPAR       POINT TO PARAMETERS FROM CALL
                 *
                 * SCAN THROUGH PARAMETERS
                 *
0C80 FF 0628 R RDMAC6 STX MACSAV       SAVE INDEX
                 *
0C83 A6 00     RDMAC4 LDA A 0,X        GET A CHAR
0C85 08               INX
0C86 81 2C            CMP A #',        END OF A PARAM?
0C88 27 04            BEQ RDMAC7       YES
                 *
0C8A 81 0D            CMP A #$0D       EOL?
0C8C 26 F5            BNE RDMAC4       NO
                 *
0C8E 5A        RDMAC7 DEC B            FOUND PARAMETER YET?
0C8F 26 EF            BNE RDMAC6       NO. KEEP LOOKING
                 *
0C91 FE 0628 R        LDX MACSAV       POINT TO PARAMETER
0C94 A6 00            LDA A 0,X        GET A CHARACTER
0C96 08               INX
0C97 FF 0628 R        STX MACSAV       SAVE POINTER
0C9A FE 062A R RDMAC5 LDX MCLPTR       POINT TO MACLIN
0C9D A7 00            STA A 0,X        SAVE CHAR  IN LINE
0C9F 08               INX
0CA0 FF 062A R        STX MCLPTR       MOVE POINTER
0CA3 8C 05CF R        CPX #MACLIN+74   OVERFLOW?
0CA6 27 17            BEQ RDMERR       YES. ERROR
                 *
0CA8 FE 0628 R        LDX MACSAV       POINT TO PARAMETER
0CAB A6 00            LDA A 0,X        GET NEXT CHAR.
0CAD 08               INX
0CAE FF 0628 R        STX MACSAV       MOVE POINTER
0CB1 81 2C            CMP A #',        END OF PARAMETER?
0CB3 26 03            BNE *+5          NO
                 *
```

```
0CB5 7E 0BE7 R        JMP RDMAC2      YES
                *
0CB8 81 0D            CMP A #$0D      EOL?
0CBA 26 DE            BNE RDMAC5      NO, KEEP SCANNING
                *
0CBC 7E 0BE7 R        JMP RDMAC2      YES
                *
0CBF 86 0D      RDMERR LDA A #$0D     FORCE EOL
0CC1 FE 062A R        LDX MCLPTR      POINT TO MACRO LINE
0CC4 FF 06F1 R        STX CUCHAR      ERROR LOCATION POINTER
0CC7 A7 00            STA A 0,X
0CC9 CE 0230          LDX #$0230      ERROR MESSAGE
0CCC BD 24E8 R        JSR PRINTE
0CCF FE 06F4 R        LDX CULINE      RESTORE POINTER
0CD2 FF 06F1 R        STX CUCHAR
0CD5 7E 0BE7 R        JMP RDMAC2
                *
                * PUSH A MACRO ONTO MACRO STACK
                *
0CD8 FF 0581 R MACPSH STX MXSAV1      SAVE INDEX
0CDB 07               TPA
0CDC 36               PSH A           SAVE COND  CODES
0CDD 0F               SEI             DISABLE INTERRUPTS
0CDE BF 057F R        STS STKSAV      SAVE STACK POINTER
0CE1 BE 057D R        LDS MSTKPT      MACRO STACK POINTER
                *
                * PUSH PARNUM,LABEL,MCLPTR,MACSAV,MACPTR ONTO STACK
                *
0CE4 CE 062E R        LDX #PARNUM
0CE7 C6 09            LDA B #9        NINE BYTES TO PUSH
0CE9 A6 00      MPSH1 LDA A 0,X       GET A BYTE
0CEB 09               DEX
0CEC FF 0583 R        STX MXSAV2
0CEF 30               TSX
0CF0 09               DEX             POINT TO STACK
0CF1 BC 0086 R        CPX MACEND      STACK FULL?
0CF4 27 35            BEQ MPSH5       YES
                *
0CF6 FE 0583 R        LDX MXSAV2      RESTORE POINTER
0CF9 36               PSH A           PUSH BYTE
0CFA 5A               DEC B           DONE?
0CFB 26 EC            BNE MPSH1       NO
                *
                * PUSH MACPAR IN REVERSE ORDER
                *
0CFD CE 05D5 R        LDX #MACPAR
0D00 A6 00      MPSH2 LDA A 0,X       FIND EOL
0D02 81 0D            CMP A #$0D
0D04 27 03            BEQ MPSH3       FOUND?
                *
0D06 08               INX
0D07 20 F7            BRA MPSH2       NO, KEEP LOOKING
                *
0D09 A6 00      MPSH3 LDA A 0,X       GET A BYTE
0D0B FF 0583 R        STX MXSAV2      SAVE POINTER
0D0E 30               TSX
0D0F 09               DEX             POINT TO STACK
0D10 BC 0086 R        CPX MACEND      STACK FULL?
0D13 27 16            BEQ MPSH5       YES, ERROR
                *
0D15 FE 0583 R        LDX MXSAV2      RESTORE POINTER
0D18 36               PSH A           PUSH BYTE
0D19 09               DEX
0D1A 8C 05D4 R        CPX #MACPAR-1   DONE?
0D1D 26 EA            BNE MPSH3       NO, KEEP GOING
                *
0D1F BF 057D R        STS MSTKPT      SAVE MACRO STACK POINTER
0D22 BE 057F R        LDS STKSAV      RESTORE STACK POINTER
0D25 32               PUL A
0D26 06               TAP             RESTORE COND  CODES
0D27 FE 0581 R        LDX MXSAV1      RESTORE INDEX
0D2A 39               RTS
                *
                * MACRO NESTING OVERFLOW ERROR
                *
0D2B BE 0088 R MPSH5  LDS MACSTK      FLUSH MACRO STACK
0D2E BF 057D R        STS MSTKPT
0D31 BE 057F R        LDS STKSAV      RESTORE STACK POINTER
0D34 32               PUL A
0D35 06               TAP             RESTORE COND  CODES
0D36 CE 0251          LDX #$0251
0D39 BD 24E8 R        JSR PRINTE      ERROR MESSAGE
0D3C FE 0581 R        LDX MXSAV1      RESTORE INDEX
0D3F 39               RTS
                *
                * PULL A MACRO FROM THE MACRO STACK
                *
0D40 FF 0581 R MACPUL STX MXSAV1      SAVE INDEX
0D43 07               TPA
0D44 36               PSH A           SAVE COND.  CODES
0D45 0F               SEI             DISABLE INTERRUPTS
0D46 BF 057F R        STS STKSAV      SAVE STACK POINTER
0D49 BE 057D R        LDS MSTKPT      POINT TO MACRO STACK
                *
                * PULL MACPAR OFF MACRO STACK
                *
0D4C CE 05D5 R        LDX #MACPAR
0D4F 32         MPUL1 PUL A           PULL A CHARACTER
0D50 A7 00            STA A 0,X       SAVE IN MACPAR
0D52 08               INX
0D53 81 0D            CMP A #$0D      EOL?
0D55 26 F8            BNE MPUL1       NO, KEEP PULLING
                *
                * PULL MACPTR,MACSAV,MCLPTR,LABEL,PARNUM
                *
0D57 CE 0626 R        LDX #MACPTR
0D5A C6 09            LDA B #9        NINE BYTES TO PULL
0D5C 32         MPUL2 PUL A           PULL A BYTE
0D5D A7 00            STA A 0,X       SAVE IT
0D5F 08               INX
0D60 5A               DEC B           DONE?
0D61 26 F9            BNE MPUL2       NO
                *
0D63 BF 057D R        STS MSTKPT      SAVE MACRO STACK POINTER
0D66 BE 057F R        LDS STKSAV      RESTORE STACK POINTER
0D69 32               PUL A
0D6A 06               TAP             RESTORE COND.  CODES
0D6B FE 0581 R        LDX MXSAV1      RESTORE INDEX
0D6E 39               RTS
                *
                * PARMVE. MOVE FUNCTION/MACRO PARAMS  TO MACPAR
                *
```

```
0D6F CE 05D5 R PARMVE LDX #MACPAR
0D72 7C 062E R        INC PARNUM      COUNT FIRST PARAMETER
0D75 FF 0628 R        STX MACSAV
0D78 FE 0642 R PARMV2 LDX DESCRA      POINT TO PARAMS
0D7B A6 00            LDA A 0,X       GET A CHAR.
0D7D 08               INX
0D7E FF 0642 R        STX DESCRA
0D81 FE 0628 R        LDX MACSAV      POINT TO MACPAR
0D84 A7 00            STA A 0,X       STORE CHAR.
0D86 08               INX
0D87 FF 0628 R        STX MACSAV
0D8A 81 0D            CMP A #$0D      EOL?
0D8C 27 09            BEQ PARMV3      YES
                 *
0D8F 81 2C            CMP A #',       COMMA?
0D90 26 E6            BNE PARMV2      NO, CONTINUE SCAN
                 *
0D92 7C 062E R        INC PARNUM      COUNT PARAMETER
0D95 20 E1            BRA PARMV2      CONTINUE SCAN
                 *
0D97 39          PARMV3 RTS           DONE!
                 *
                 *
                 * WRITE A CHAR TO LIST FROM 'A' DURING COMPILE
                 *
0D98 F6 06F8 R WRTCHR  LDA B OPTNS    CHECK OPTIONS
0D9B C5 10             BIT B #$10     INTERMEDIATE?
0D9D 26 03             BNE *+5        NO
                 *
0D9F BD 3EAE R         JSR OUTCHR     WRITE CHARACTER
                 *
0DA2 39                RTS
                 *
                 * PUT CHAR IN OBJECT FROM 'B' DURING COMPILE
                 *
0DA3 7E 3E69 R TPCHAR  JMP OUTBNR
                 *
                 *
                 * OUTPUT A NAME (USING DESCRA,DESCRC)
                 *
0DA6 FE 0642 R OUTNAM  LDX DESCRA
0DA9 B6 0644 R         LDA A DESCRC
0DAC B7 06F0 R         STA A CMPCNT
                 *
                 * WRITE CHAR STRING FROM X-REG POINTER
                 * IF END ON COUNT, APPEND BLANK
                 * IF END ON CR, ADD LF
                 *
0DAF A6 00       OUTCMP LDA A 0,X      GET CHAR
0DB1 36          OTCMP1 PSH A
0DB2 8D E4             BSR WRTCHR     LIST CHAR
0DB4 33                PUL B
0DB5 37                PSH B
0DB6 8D EB             BSR TPCHAR     TAPE CHAR
0DB8 32                PUL A
0DB9 81 0D             CMP A #$0D     CR?
0DBB 26 08             BNE OTCMP2     NO
                 *
0DBD 86 0A             LDA A #$0A     LF
0DBF 8D D7             BSR WRTCHR     WRITE IT
0DC1 C6 0A             LDA B #$0A
0DC3 20 DE             BRA TPCHAR
                 *
0DC5 08          OTCMP2 INX
0DC6 7A 06F0 R         DEC CMPCNT     DEC  CHAR COUNT
0DC9 26 E4             BNE OUTCMP     LOOP UNTIL DONE
                 *
0DCB 86 20             LDA A #$20
0DCD 8D C9             BSR WRTCHR     APPEND BLANK
0DCF C6 20             LDA B #$20
0DD1 20 D0             BRA TPCHAR
                 *
                 * WRITE CR ON OBJECT,  CR-LF TO LIST
                 *
0DD3 86 0D       OUTCR  LDA A #$0D     CR
0DD5 20 DA             BRA OTCMP1     USE OUTCMP TO COMPLETE PROCESS
                 *
                 *
                 * COMPARE TWO STRINGS *
                 * ON ENTRY [X] = A PARM LIST OF 5 BYTES .
                 *       A(STRING1)
                 *       A(STRING2)
                 *       COUNT OF # BYTES TO BE COMPARED
                 *
                 * ON RETURN IF CC Z IS SET THERE IS A MATCH
                 *       EXAMPLE.
                 *             LDX  #STRING1
                 *             JSR  COMPAR
                 *             BEQ  ------    MATCH
                 *
                 *
                 *    STRING1  RMB  2
                 *    STRING2  RMB  2
                 *    COUNT    RMB  1
                 *
0DD7 36          COMPAR PSH A
0DD8 37                PSH B
0DD9 E6 04             LDA B 4,X      GET COUNT
0DDB FF 0E03 R         STX   XSAV     SAVE PARM POINTER
0DDE FE 0E03 R CMP1    LDX   XSAV     GET PARM PTR
0DE1 EE 00             LDX   0,X      GET A(STRING1)
0DE3 A6 00             LDA A 0,X      GET CHARACTER
0DE5 FE 0E03 R         LDX   XSAV     GET PTR
0DE8 6C 01             INC   1,X      PTR SET TO NEXT
0DEA 26 02             BNE   *+4      CHAR IN
                 *
0DEC 6C 00             INC   0,X      STRING1
0DEE EE 02             LDX   2,X      GET A(STRING2)
0DF0 A1 00             CMP A 0,X      COMPARE
0DF2 26 0C             BNE   CDONE    NOT EQUAL
                 *
0DF4 FE 0E03 R         LDX   XSAV     GET PARM POINTER
0DF7 6C 03             INC   3,X      PTR SET TO NEXT
0DF9 26 02             BNE   CMP2     CHAR IN
                 *
0DFB 6C 02             INC   2,X      STRING2
0DFD 5A          CMP2  DEC B          DECREMENT COUNT
0DFE 26 DE             BNE   CMP1     TRY AGAIN
                 *
0E00 33          CDONE PUL B          DONE
0E01 32                PUL A
0E02 39                RTS
```

```
                        *
0E03 0002               XSAV    RMB   2          PARM PTR SAVE AREA
                        *
                        *
                        * PROCESS "LET" COMPILER OPCODE
                        * ALSO HANDLES "ASSUMED" ASSIGNMENTS (NO "LET")
                        * HANDLES "FOR" AND "GETAD" PROCESSING TOO
                        *
0E05 BD 172F R          LETPR   JSR NXTOK          GET NEXT TOKEN
0E08 C1 01                      CMP B #1           VALID NAME?
0E0A 27 11                      BEQ LETPR2         O.K
                        *
0E0C CE 0302            LETERR  LDX #$0302
0E0F 7D 139B R                  TST GAFLG2         IN A "GETAD"?
0E12 27 03                      BEQ *+5            NO
                        *
0E14 CE 0313                    LDX #$0313         GETAD ERROR
                        *
0E17 BD 24E8 R                  JSR PRINTE         PRINT ERROR MESSAGE
0E1A 7E 0937 R                  JMP MAIN1          QUIT
                        *
0E1D BD 1A5C R          LETPR2  JSR LKPSYM         CHECK SYMBOL TABLE
0E20 7F 0701 R                  CLR PFLAG          CLEAR PARAM. FLAG
0E23 7F 0EDB R                  CLR STRFLG         CLEAR ARRAY EL. FLAG
0E26 C1 FF                      CMP B #$FF         FOUND?
0E28 26 05                      BNE LET1           YES
                        *
0E2A BD 2A56 R                  JSR DEFS2          DEFINE SYMBOL (NO DIMENS )
0E2D 20 0C                      BRA LET2
                        *
0E2F C5 03              LET1    BIT B #3           DIMENSIONED?
0E31 27 08                      BEQ LET2           NO
                        *
0E33 73 0EDB R                  COM STRFLG         SET ARRAY EL. FLAG
0E36 7D 0700 R                  TST FORFLG         IN A "FOR" STMT?
0E39 26 D1                      BNE LETERR         NO DIM  ALLOWED IN "FOR"
                        *
0E3B FE 0642 R          LET2    LDX DESCRA
0E3E FF 0645 R                  STX MYSAV          STORE TOKEN PARAMETERS
0E41 B6 0644 R                  LDA A DESCRC
0E44 B7 0649 R                  STA A OPTLN
0E47 C5 80                      BIT B #$80         PARAMETER?
0E49 27 03                      BEQ *+5            NO
                        *
0E4B 73 0701 R                  COM PFLAG          YES, SET FLAG
                        *
0E4E C4 08                      AND B #$08
0E50 F7 0578 R                  STA B FLPFLG       IF FLOAT, SET FLAG
0E53 27 0A                      BEQ LET3
                        *
0E55 7D 0700 R                  TST FORFLG         IN "FOR" STMT. ?
0E58 26 B2                      BNE LETERR         NO FLOAT ALLOWED
                        *
0E5A 7D 139B R                  TST GAFLG2         IN A "GETAD"?
0E5D 26 AD                      BNE LETERR         NO FLOAT ALLOWED
                        *
0E5F BD 172F R          LET3    JSR NXTOK          GET NEXT TOKEN
0E62 FE 0642 R                  LDX DESCRA
0E65 FF 0647 R                  STX MYSAV2
0E68 C1 3D                      CMP B #'=          AN EQUALS SIGN?
0E6A 27 18                      BEQ LET3C          YES
                        *
0E6C C1 0D                      CMP B #$0D         CR?
0E6E 27 9C                      BEQ LETERR         IF SO, ERROR
                        *
0E70 FE 0642 R                  LDX DESCRA
0E73 08                 LET3A   INX
0E74 A6 00                      LDA A 0,X          SEARCH FOR =
0E76 81 3D                      CMP A #'=
0E78 27 06                      BEQ LET3B
                        *
0E7A 81 0D                      CMP A #$0D         CR?
0E7C 27 8E                      BEQ LETERR         IF SO, ERROR
0E7E 20 F3                      BRA LET3A
                        *
0E80 08                 LET3B   INX
0E81 FF 06F1 R                  STX CUCHAR
                        *
0E84 7D 139B R          LET3C   TST GAFLG2         IN A "GETAD"?
0E87 27 05                      BEQ LET3D          NO
                        *
0E89 BD 138A R                  JSR GETADR         GET ADDRESS
0E8C 20 03                      BRA *+5
                        *
0E8E BD 1C47 R          LET3D   JSR EVALS          EVALUATE EXPRESSION
                        *
0E91 7D 0EDB R                  TST STRFLG         AN ARRAY ELEMENT?
0E94 27 46                      BEQ LET3F          NO
                        *
0E96 FE 0645 R                  LDX MYSAV          RESTORE TOKEN PARAMETERS
0E99 FF 0642 R                  STX DESCRA
0E9C FE 0647 R                  LDX MYSAV2
0E9F FF 06F1 R                  STX CUCHAR
0EA2 B6 0649 R                  LDA A OPTLN
0EA5 B7 0644 R                  STA A DESCRC
0EA8 7D 0578 R                  TST FLPFLG         IN FLOATING MODE?
0EAB 27 28                      BEQ LET3E
                        *
                        * GENERATE FLOAT CODE FOR ARRAY
                        *       LDX #SAVER
                        *       STX ACCUM1
                        *       LDX #RUNSTK
                        *       JSR MOVER
                        * GET INDEX VALUE IN X-REGISTER
                        *       STX ACCUM1
                        *       LDX #SAVER
                        *       JSR MOVER
                        *
0EAD CE 0FCA R                  LDX #SAVEL
0EB0 BD 13AB R                  JSR STR1           OUTPUT LINE 1
0EB3 CE 12EE R                  LDX #UNOPL2
0EB6 BD 13AB R                  JSR STR1           OUTPUT LINE 2
0EB9 C6 A5                      LDA B #165
0EBB BD 1224 R                  JSR FUNOP          OUTPUT LINE 3,4
0EBE BD 3DEA R                  JSR EVADIM         EVALUATE INDEX
0EC1 CE 12FE R                  LDX #UNOPL2
0EC4 BD 13AB R                  JSR STR1           OUTPUT LINE 5
0EC7 CE 0FCA R                  LDX #SAVEL
0ECA BD 13AB R                  JSR STR1           OUTPUT LINE 6
0ECD C6 A5                      LDA B #165
0ECF BD 139C R                  JSR RTB1           OUTPUT LINE 7
0ED2 7E 0937 R                  JMP MAIN1          DONE
```

```
                        *
                        * GENERATE INTEGER ARRAY ELEMENT CODE HERE
                        *
0ED5 BD 0F65 R LET3E    JSR STRDIM        GENERATE ARRAY EL. CODE
0ED8 7E 0937 R          JMP MAIN1         DONE
                        *
0EDB 0001               STRFLG  RMB 1
                        *
0EDC 7D 0578 R LET3F    TST FLPFLG        IN FLOATING MODE?
0EDF 27 37              BEQ LET4          NO
                        *
                        * GENERATE FLOATING CODE HERE
                        *       LDX #NAME
                        *       STX ACCUM1
                        *       LDX #RINSTK
                        *       JSR MOVER
                        *
0EE1 86 04              LDA A #4
0EE3 B7 06F0 R          STA A CMPCNT
0EE6 CE 12E3 R          LDX #UNOPL1
0EE9 BD 0DAF R          JSR OUTCMP        OUTPUT "LDX "
0EEC 7D 0701 R          TST PFLAG         PARAMETER?
0EEF 26 0A              BNE LET3G         YES, SKIP "#"
                        *
0EF1 86 23              LDA A #'#
0EF3 BD 0D98 R          JSR WRTCHR        OUTPUT "#"
0EF6 C6 23              LDA B #'#
0EF8 BD 0DA3 R          JSR TPCHAR
                        *
0EFB FE 0645 R LET3G    LDX MYSAV
0EFE B6 0649 R          LDA A OPTLN
0F01 B7 06F0 R          STA A CMPCNT
0F04 BD 0DAF R          JSR OUTCMP        OUTPUT "NAME"
0F07 BD 0DD3 R          JSR OUTCR
0F0A CE 12FE R          LDX #UNOPL2
0F0D BD 13AB R          JSR STR1          OUTPUT LINE 2
0F10 C6 A5              LDA B #165
0F12 BD 1224 R          JSR FUNOP         OUTPUT LINE 3,4
0F15 7E 0937 R          JMP MAIN1         DONE
                        *
                        * GENERATE INTEGER CODE HERE
                        *       STX NAME          (NOT PARAMETER)
                        * FOR PARAMETER, CODE LIKE INTEGER ARRAY
                        *
0F18 7D 0701 R LET4     TST PFLAG         PARAMETER?
0F1B 27 25              BEQ LET5          NO
                        *
0F1D CE 0F86 R          LDX #STRL1        CODE FOR PARAMETER
0F20 BD 13AB R          JSR STR1          "STX ACCUM3"
0F23 86 FF              LDA A #$FF
0F25 B7 139A R          STA A GAFLG       SET ADDRESS FLAG
0F28 FE 0645 R          LDX MYSAV         RESTORE TOKEN PARAMETERS
0F2B FF 0642 R          STX DESCRA
0F2E FE 0647 R          LDX MYSAV2
0F31 FF 06F1 R          STX CUCHAR
0F34 B6 0649 R          LDA A OPTLN
0F37 B7 0644 R          STA A DESCRC
0F3A BD 3DEB R          JSR EVADIM+3      GET ADDRESS
0F3D 8D 2F              BSR STRD2         FINISH UP
0F3F 7E 0937 R          JMP MAIN1         DONE!
                        *
0F42 CE 12FE R LET5     LDX #UNOPL2
0F45 86 04              LDA A #4
0F47 B7 06F0 R          STA A CMPCNT
0F4A BD 0DAF R          JSR OUTCMP        OUTPUT " STX"
0F4D FE 0645 R          LDX MYSAV
0F50 B6 0649 R          LDA A OPTLN
0F53 B7 06F0 R          STA A CMPCNT
0F56 BD 0DAF R          JSR OUTCMP        OUTPUT " NAME"
0F59 BD 0DD3 R          JSR OUTCR
0F5C 7D 0700 R          TST FORFLG        IN A "FOR" STMT?
0F5F 27 01              BEQ *+3           NO
0F61 39                 RTS
                        *
0F62 7E 0937 R          JMP MAIN1         DONE
                        *
                        * GENERATE ARRAY ELEMENT STORE CODE:
                        * GET VALUE IN X REGISTER
                        * DESCRA,DESCRC POINT TO ARRAY NAME
                        *       STX ACCUM3
                        * GET INDEX VALUE IN X REGISTER
                        *       LDA A ACCUM3
                        *       LDA B ACCUM3+1
                        *       STA A 0,X
                        *       STA B 1,X
                        *
0F65 CE 0F86 R STRDIM   LDX #STRL1
0F68 BD 13AB R          JSR STR1          OUTPUT LINE 1
0F6B BD 3DEA R          JSR EVADIM        EVALUATE INDEX
0F6E CE 0F94 R STRD2    LDX #STRL2
0F71 BD 13AB R          JSR STR1          OUTPUT LINE 2
0F74 CE 0FA4 R          LDX #STRL3
0F77 BD 13AB R          JSR STR1          OUTPUT LINE 3
0F7A CE 0FB4 R          LDX #STRL4
0F7D BD 13AB R          JSR STR1          OUTPUT LINE 4
0F80 CE 0FBF R          LDX #STRL5
0F83 7E 13AB R          JMP STR1          OUTPUT LINE 5
                        *
0F86 20                 STRL1   FCC ' STX  ACCUM+4'
0F93 0D                         FCB $0D
0F94 20                 STRL2   FCC ' LDA A  ACCUM+4'
0FA3 0D                         FCB $0D
0FA4 20                 STRL3   FCC ' LDA B  ACCUM+5'
0FB3 0D                         FCB $0D
0FB4 20                 STRL4   FCC ' STA A 0,X'
0FBE 0D                         FCB $0D
0FBF 20                 STRL5   FCC ' STA B 1,X'
0FC9 0D                         FCB $0D
0FCA 20                 SAVEL   FCC ' LDX #,ACCUM+8'
0FD8 0D                         FCB $0D
                        *
                        *
                        * PROCESS "FOR" COMPILER OPCODE
                        * USES "LET" PROCESSING
                        *
0FD9 73 0700 R FORPR    COM FORFLG        SET "FOR" FLAG
0FDC 7F 0578 R          CLR FLPFLG        FORCE INTEGER MODE
0FDF BD 0E05 R          JSR LETPR         NAME=EXPRESSION
0FE2 FE 0708 R          LDX LBLCNT        GENERATE NEW LABEL
0FE5 08                 INX
0FE6 FF 0708 R          STX LBLCNT
0FE9 FF 36AD R          STX BLKTOP
```

```
0FEC 7F 0578 R          CLR FLPFLG      FORCE INTEGER MODE
0FEF BD 1C47 R          JSR EVALS       EVAL. MAX  EXPRESSION
0FF2 7D 24CC R          TST FRNUM       EVALUATION ERROR?
0FF5 26 5F              BNE FORPR5      IF SO, QUIT
                *
0FF7 B6 3BAD R          LDA A BLKTOP    SET UP FOR CVBTD
0FFA F6 3BAE R          LDA B BLKTOP+1
0FFD CE 107B R          LDX #FORL1+6
1000 BD 2487 R          JSR CVBTD
1003 CE 1075 R          LDX #FORL1
1006 BD 13AB R          JSR STR1        OUTPUT "STX LABEL+2"
1009 FE 0642 R          LDX DESCRA
100C A6 00              LDA A 0,X       GET LAST TOKEN USED
100E 81 0D              CMP A #$0D      END-OF-LINE?
1010 26 08              BNE FORPR2      NO, THEN "STEP"
                *
                *DEFAULT STEP SIZE IS 1
                *GENERATE  LDX #1
1012 CE 36A2 R          LDX #COMPL4
1015 BD 13AB R          JSR STR1        OUTPUT CODE LINE
1018 20 06              BRA FORPR3
                *
101A 7F 0578 R FORPR2   CLR FLPFLG      FORCE INTEGER MODE
101D BD 1C47 R          JSR EVALS       EVAL. STEP EXPR
                *
1020 CE 1089 R FORPR3   LDX #FORL3+6    SET UP FOR CVBTD
1023 B6 3BAD R          LDA A  BLKTOP
1026 F6 3BAE R          LDA B  BLKTOP+1
1029 BD 2487 R          JSR CVBTD
102C CE 1083 R          LDX #FORL3
102F BD 13AB R          JSR STR1        OUTPUT "STX LABEL"
1032 07                 TPA             SAVE COND. CODES
1033 36                 PSH A
1034 0F                 SEI             MASK OUT INTERRUPTS
1035 BF 1BD4 R          STS STKTOP      SAVE STACK POINTER
1038 BE 389D R          LDS BLKCNT
103B B6 3BAD R          LDA A BLKTOP    STACK LABEL
103E 36                 PSH A
103F B6 3BAE R          LDA A BLKTOP+1
1042 36                 PSH A
1043 30                 TSX
1044 09                 DEX             ADJUST
1045 BE 1BD4 R          LDS STKTOP      RESTORE STACK POINTER
1048 32                 PUL A
1049 06                 TAP             RESTORE COND. CODES
104A 8C 3889 R          CPX #BLKSTR     OUT OF STACK SPACE?
104D 26 09              BNE FORPR5+3    NO
                *
104F CE 0305            LDX #$0305
1052 BD 24E8 R          JSR PRINTE      PRINT ERROR MESSAGE
1055 7E 0937 R FORPR5   JMP MAIN1       QUIT
                *
1058 FF 389D R          STX BLKCNT
105B CE 108F R          LDX #FORL4
105E BD 13AB R          JSR STR1        OUTPUT "FOR-BLOCK" LINE 1
1061 CE 1099 R          LDX #FORL5+1    SET UP FOR CVBTD
1064 B6 3BAD R          LDA A BLKTOP
1067 F6 3BAE R          LDA B BLKTOP+1
106A BD 2487 R          JSR CVBTD
106D CE 1098 R          LDX #FORL5
1070 BD 13AB R          JSR STR1        OUTPUT "FOR-BLOCK" LINE 2
1073 20 E0              BRA FORPR5      DONE
                *
1075 20         FORL1   FCC ' STX  '
1078 0005               RMB 5
1080 2B                 FCC '+2'
1082 0D                 FCB $0D
1083 20         FORL3   FCC ' STX  '
1089 0005               RMB 5
108E 0D                 FCB $0D
108F 20         FORL4   FCC ' BRA *+6'
1097 0D                 FCB $0D
1098 2E         FORL5   FCC '  '
1099 0005               RMB 5
109E 20                 FCC ' RMB 4'
10A4 0D                 FCB $0D
                *
                *
                * PROCESS "NEXT" COMPILER OPCODE
                *
10A5 07         NEXTPR  TPA             SAVE COND  CODES
10A6 36                 PSH A
10A7 0F                 SEI             MASK OUT INTERRUPTS
10A8 BF 1BD4 R          STS STKTOP      SAVE STACK POINTER
10AB BE 389D R          LDS BLKCNT
10AE 32                 PUL A           UNSTACK "FOR" LABEL
10AF B7 3BAE R          STA A BLKTOP+1
10B2 32                 PUL A
10B3 B7 3BAD R          STA A BLKTOP
10B6 30                 TSX
10B7 09                 DEX             ADJUST
10B8 BE 1BD4 R          LDS STKTOP      RESTORE STACK POINTER
10BB 32                 PUL A
10BC 06                 TAP             RESTORE COND  CODES
10BD 8C 389C R          CPX #BLKEND     ENOUGH STACK SPACE?
10C0 2C 09              BGE NXTPR2+3
                *
10C2 CE 0307    NXTERR  LDX #$0307      ERROR
10C5 BD 24E8 R          JSR PRINTE
10C8 7E 0937 R NXTPR2   JMP MAIN1       QUIT
                *
10CB FF 389D R          STX BLKCNT
10CE BD 172F R          JSR NXTOK       GET NEXT TOKEN
10D1 C1 01              CMP B #1        VALID NAME?
10D3 26 ED              BNE NXTERR      NO
                *
10D5 BD 1A5C R          JSR LKPSYM      CHECK SYMBOL TABLE
10D8 C1 FF              CMP B #$FF      FOUND?
10DA 27 E6              BEQ NXTERR      NOT FOUND, ERROR
                *
10DC C5 8B              BIT B #$8B      ARRAY,FLOAT, OR PARAMETER?
10DE 26 E2              BNE NXTERR      IF SO, ERROR
                *
                * GENERATE CODE AS FOLLOWS:
                *
                *       LDX NAME
                *       CPX LABEL+2
                *       BEQ *+18
                *       STX ACCUM1
                *       LDX LABEL
                *       JSR ADD
                *       STX NAME
```

```
                        *       JMP LABEL+4
                        *
10E0 CE 12E3 R                  LDX #UNOPL1
10E3 86 04                      LDA A #4
10E5 B7 06F0 R                  STA A CMPCNT
10L8 BD 0DAF R                  JSR OUTCMP
10EB BD 0DA6 R                  JSR OUTNAM              OUTPUT NAME
10EE BD 0DD3 R                  JSR OUTCR               LINE 1
10F1 CE 1189 R                  LDX #NXTL5+6
10F4 B6 3BAD R                  LDA A BLKTOP            SET UP FOR CVBTD
10F7 F6 3BAE R                  LDA B BLKTOP+1
10FA BD 2487 R                  JSR CVBTD
10FD CF 1183 R                  LDX #NXTL5
1100 BD 13AB R                  JSR STR1                LINE 2
1103 CE 1191 R                  LDX #NXTL6
1106 BD 13AB R                  JSR STR1                LINE 3
1109 CE 12EE R                  LDX #UNOPL2
110C BD 13AB R                  JSR STR1                LINE 4
110F CF 115D R                  LDX #NXTL1+6
1112 B6 3BAD R                  LDA A BLKTOP            SET UP FOR CVBTD
1115 F6 3BAE R                  LDA B BLKTOP+1
1118 BD 2487 R                  JSR CVBTD
111B CE 1157 R                  LDX #NXTL1
111E BD 13AB R                  JSR STR1                LINE 5
1121 C6 0C                      LDA B #12
1123 BD 139C R                  JSR RTBL                LINE 6
1126 CF 12EE R                  LDX #UNOPL2
1129 86 04                      LDA A #4
112B B7 06F0 R                  STA A CMPCNT
112E BD 0DAF R                  JSR OUTCMP
1131 BD 0DA6 R                  JSR OUTNAM
1134 BD 0DD3 R                  JSR OUTCR               LINE 7
1137 CE 3BA4 R                  LDX #JMPZ
113A 86 04                      LDA A #4
113C B7 06F0 R                  STA A CMPCNT
113F BD 0DAF R                  JSR OUTCMP
1142 CE 117B R                  LDX #NXTL4+1            SET UP FOR CVBTD
1145 B6 3BAD R                  LDA A BLKTOP
1148 F6 3BAE R                  LDA B BLKTOP+1
114B BD 2487 R                  JSR CVBTD
114E CE 117A R                  LDX #NXTL4
1151 BD 13AB R                  JSR STR1                OUTPUT LINE 8
1154 7E 0937 R                  JMP MAIN1               DONE
                        *
1157 20                 NXTL1   FCC ' LDX '
115D 0005                       RMB 5
1162 0D                         FCB $0D
1163 20                 NXTL2   FCC ' LDX '
1169 0005                       RMB 5
116E 2B                         FCC '+2'
1170 0D                         FCB $0D
1171 20                 NXTL3   FCC ' BGT *+5'
1179 0D                         FCB $0D
117A 2F                 NXTL4   FCC '/'
117B 0005                       RMB 5
1180 2B                         FCC '+4'
1182 0D                         FCB $0D
1183 20                 NXTL5   FCC ' CPX '
1189 0005                       RMB 5
118E 2B                         FCC '+2'
1190 0D                         FCB $0D
1191 20                 NXTL6   FCC ' BEQ *+18'
119A 0D                         FCB $0D
                        *
                        * HANDLE "PUSH" COMPILER OPCODE
                        * PUT ADDRESS OF ARGUMENT IN X-REGISTER
                        * GENERATE RUNTIME CALL
                        *
119B BD 3812 R          PUSHPR  JSR PRESCN      GET FLOAT TYPE
119E BD 138A R                  JSR GETADR      GET ADDRESS INTO X
11A1 C6 D8                      LDA B #216      INTEGER CALL
11A3 7D 0578 R                  TST FLPFLG      INTEGER?
11A6 27 02                      BEQ *+4         YES
                        *
11A8 C6 DB                      LDA B #219      FLOATING CALL
11AA 7D 25C6 R                  TST DSTFLG      STRING?
11AD 27 02                      BEQ *+4         NO
                        *
11AF C6 DE                      LDA B #222      STRING CALL
11B1 BD 139C R                  JSR RTBL        MAKE RUNTIME CALL
11B4 7E 0937 R                  JMP MAIN1       DONE!
                        *
                        *
                        * HANDLE "POP" COMPILER OPCODE
                        * PUT ADDRESS OF ARGUMENT INTO X-REGISTER
                        * GENERATE RUNTIME CALL
                        *
11B7 BD 3812 R          POPPR   JSR PRESCN      GET FLOAT TYPE
11BA BD 138A R                  JSR GETADR      GET ADDRESS INTO X
11BD C6 E1                      LDA B #225      INTEGER CALL
11BF 7D 0578 R                  TST FLPFLG      INTEGER?
11C2 27 02                      BEQ *+4         YES
                        *
11C4 C6 E4                      LDA B #228      FLOATING CALL
11C6 7D 25C6 R                  TST DSTFLG      STRING?
11C9 27 02                      BEQ *+4         NO
                        *
11CB C6 E7                      LDA B #231      STRING CALL
11CD BD 139C R                  JSR RTBL        RUNTIME CALL
11D0 7E 0937 R                  JMP MAIN1       DONE
                        *
                        *
                        * UNARY FUNCTION HANDLER
                        * SEVENTEEN UNARY FUNCTIONS:
                        * NOT, ABS, RAN, SIGN, PEEK
                        * SIN, COS, TAN, ASIN, ACOS, ATAN
                        * TENEXP, FEXP, NATLOG, LOG10, FIX, FNEG
                        *
11D3 C6 00              NOTPR   LDA B #0
11D5 F7 20E5 R          EVAL20  STA B TMPVAL    HANDLE FUNCTION (INTEGER)
11D8 7E 12A1 R                  JMP UNOP
                        *
11DB C6 1B              ABSPR   LDA B #27
11DD 20 F6                      BRA EVAL20
                        *
11DF C6 27              RANPR   LDA B #39
11E1 20 F2                      BRA EVAL20
                        *
11E3 C6 1E              SIGPR   LDA B #30
11E5 20 EE                      BRA EVAL20
                        *
11E7 C6 51              PEEKPR  LDA B #81
```

```
11E9 20 EA              BRA EVAL20
                 *
11EB C6 63       FABS   LDA B #99
11ED 20 35              BRA FUNOP
                 *
11EF C6 90       ACOSPR LDA B #144
11F1 20 31              BRA FUNOP
                 *
11F3 C6 8D       ASINPR LDA B #141
11F5 20 2D              BRA FUNOP
                 *
11F7 C6 93       ATANPR LDA B #147
11F9 20 29              BRA FUNOP
                 *
11FB C6 87       COSPR  LDA B #135
11FD 20 25              BRA FUNOP
                 *
11FF C6 9F       FEXP   LDA B #159
1201 20 21              BRA FUNOP
                 *
1203 7F 05/8 R FIXPR    CLR FLPFLG
1206 C6 5D              LDA B #93
1208 20 1A              BRA FUNOP
                 *
120A C6 96       LOGPR  LDA B #150
120C 20 16              BRA FUNOP
                 *
120F C6 99       NATPR  LDA B #153
1210 20 12              BRA FUNOP
                 *
1212 C6 69       FNEG   LDA B #105
1214 20 0E              BRA FUNOP
                 *
1216 C6 66       FSIGN  LDA B #102
1218 20 0A              BRA FUNOP
                 *
121A C6 84       SINPR  LDA B #132
121C 20 06              BRA FUNOP
                 *
121E C6 8A       TANPR  LDA B #138
1220 20 02              BRA FUNOP
                 *
1222 C6 9C       TENPR  LDA B #156
                 *
1224 F7 20E5 R FUNOP    STA B TMPVAL    HANDLE FUNCTION (FLOATING)
1227 B6 1C03 R          LDA A NUMBER
122A F6 1C04 R          LDA B NUMBER+1
122D CE 1264 R          LDX #FCLN1+13
1230 BD 2487 R          JSR CVBTD
1233 CE 1257 R          LDX #FCLN1
1236 BD 13AB R          JSR STR1        "LDX #ARGUMENT"
1239 F6 20E5 R          LDA B TMPVAL
123C 7E 139C R          JMP RTBL        "JSR RUNTIME"
                 *
123F B6 1C03 R FUNOP2   LDA A NUMBER
1242 F6 1C04 R          LDA B NUMBER+1
1245 CE 1264 R          LDX #FCLN1+13
1248 BD 2487 R          JSR CVBTD
124B CE 1257 R          LDX #FCLN1
124E BD 13AB R          JSR STR1        "LDX #ARGUMENT"
1251 CE 12EE R          LDX #UNOPL2
```

```
1254 7E 13AB R          JMP STR1        "STX ACCUM1"
                 *
1257 20          FCLN1  FCC ' LDX # RNSTK+'
1264 0005               RMB 5
1269 0D                 FCB $0D
126A 20          FCLN2  FCC ' STX #.RNSTK+'
1277 0005               RMB 5
127C 0D                 FCB $0D
127D 20          FCLN3  FCC ' LDX  RNSTK+'
1289 0005               RMB 5
128E 0D                 FCB $0D
128F 20          FCLN4  FCC ' STX .RNSTK+'
129B 0005               RMB 5
12A0 0D                 FCB $0D
                 *
                 *
                 * UNARY OPERATION CODE GENERATOR
                 * GENERATES THE FOLLOWING:
                 *      LDX ARGUMENT
                 *      JSR UNOPERATION
                 *      STX ARGUMENT
                 *
12A1 7F 12E2 R UNOP     CLR  UBFLAG
12A4 7F 368E R          CLR COMPL
12A7 B6 1C03 R UNOP2    LDA A NUMBER    SET UP FOR CVBTD
12AA F6 1C04 R          LDA B NUMBER+1
12AD CE 1289 R          LDX #FCLN3+12
12B0 BD 2487 R          JSR CVBTD
12B3 CE 127D R          LDX #FCLN3
12B6 BD 13AB R          JSR STR1        OUTPUT LINE 1
12B9 7D 12E2 R          TST UBFLAG
12BC 27 06              BEQ UNOP3
                 *
12BE CE 12EE R          LDX #UNOPL2
12C1 7E 13AB R          JMP STR1        OUTPUT "STX .ACCUM"
                 *
12C4 F6 20E5 R UNOP3    LDA B TMPVAL
12C7 BD 139C R          JSR RTBL        OUTPUT UNOPERATION
12CA 7D 368E R          TST COMPL
12CD 27 01              BEQ *+3
12CF 39                 RTS
                 *
12D0 B6 1C03 R UNOP4    LDA A NUMBER    SET UP FOR CVBTD
12D3 F6 1C04 R          LDA B NUMBER+1
12D6 CE 129B R          LDX #FCLN4+12
12D9 BD 2487 R          JSR CVBTD
12DC CE 128F R          LDX #FCLN4
12DF 7E 13AB R          JMP STR1        OUTPUT LINE 3
                 *
12E2 0001        UBFLAG RMB 1
12E3 20          UNOPL1 FCC ' LDX '
12E8 0005               RMB 5
12ED 0D                 FCB $0D
12EE 20          UNOPL2 FCC ' STX  ACCUM'
12F9 0D                 FCB $0D
                 *
                 * BINARY OPERATION CODE GENERATOR
                 * GENERATES FOLLOWING CODE:
                 *      LDX ARGUMENT 1
                 *      STX ACCUM1
                 *      LDX ARGUMENT 2
```

```
                    *        JSR BINOPERATION
                    *        STX ARGUMENT 1
                    *
12FA 7F 368E R BINOP   CLR COMPL
12FD BD 1C28 R BINOP2  JSR STAKDN       ROLL OPSTACK DOWN
1300 7F 12E2 R         CLR UBFLAG
1303 73 12E2 R         COM UBFLAG
1306 8D 9F             BSR UNOP2        USE UNOP FOR LINES 1 AND 2
1308 B6 1C05 R         LDA A AX         SET UP FOR CVBTD
130B F6 1C06 R         LDA B AX+1
130E CE 1289 R         LDX #FCLN3+12
1311 BD 2487 R         JSR CVBTD
1314 CE 127D R         LDX #FCLN3
1317 BD 13AB R         JSR STR1         OUTPUT LINE 3
131A 20 A8             BRA UNOP3        USE UNOP FOR LINES 4 AND 5
                    *
                    * CHECK BINARY OPERATIONS
                    *
131C 7D 1BD3 R CHKBIN  TST STKCNT       2 OPERANDS ON STACK?
131F 2C 0C             BGE *+14         O.K.
                    *
1321 FF 0E03 R         STX XSAV
1324 CE 0301           LDX #0301        OTHERWISE ERROR
1327 BD 24E8 R         JSR PRINTE
132A FE 0E03 R         LDX XSAV
                    *
132D 39                RTS
                    *
                    * ARITHMETIC AND BINARY FUNCTION ROUTINES
                    * ADD, SUB, MULT, DIV, AND, OR, EXOR, EXP, MOD
                    * FLOAT, XTOY
                    *
132E C6 12          MULT   LDA B #18
1330 F7 20E5 R EVAL19  STA B TMPVAL
1333 8D E7             BSR CHKBIN       2 OPERANDS?
1335 20 C3             BRA BINOP        GENERATE CODE
                    *
1337 C6 0C          ADDZ   LDA B #12
1339 20 F5             BRA EVAL19
                    *
133B C6 0F          SUBTR  LDA B #15
133D 20 F1             BRA EVAL19
                    *
133F C6 15          DIV    LDA B #21
1341 20 ED             BRA EVAL19
                    *
1343 C6 09          ORPR   LDA B #9
1345 20 E9             BRA EVAL19
                    *
1347 C6 03          ANDPR  LDA B #3
1349 20 E5             BRA EVAL19
                    *
134B C6 06          EXOPR  LDA B #6
134D 20 E1             BRA EVAL19
                    *
134F C6 24          EXPR   LDA B #36
1351 20 DD             BRA EVAL19
                    *
1353 C6 21          MODPR  LDA B #33
1355 20 D9             BRA EVAL19
                    *
1357 86 FF          FLOTPR LDA A #$FF
1359 B7 0578 R         STA A FLPFLG     SET FLAG
135C B7 12E2 R         STA A UBFLAG
135F B7 368E R         STA A COMPL
1362 BD 12A7 R         JSR UNOP2        VALUE INTO ACCUM
1365 FE 1BCD R         LDX RUNCNT
1368 08                INX
1369 08                INX              MAKE ROOM FOR FLOATING VALUE
136A 08                INX
136B 08                INX
136C 08                INX
136D 08                INX
136E 08                INX
136F FF 1BCD R         STX RUNCNT       ADDRESS OF VALUE
1372 C6 60             LDA B #96
1374 7E 1224 R         JMP FUNOP
                    *
1377 8D A3          XTOYPR BSR CHKBIN   2 OPERANDS?
1379 BD 123F R         JSR FUNOP2       FIRST ARGUMENT
137C BD 1C28 R         JSR STAKDN       ROLL STACK DOWN
137F C6 A2             LDA B #162
1381 7E 1224 R         JMP FUNOP
                    *
                    * GET-ADDRESS COMPILER OPERATION
                    * GETAD A=B  (ADDRESS OF B INTO A)
                    *
1384 73 139B R GETAD   COM GAFLG2       SET "GETAD" FLAG
1387 7E 0E05 R         JMP LETPR        FINISH UP LIKE "ASSIGNMENT"
                    *
                    * SUBROUTINE TO GET ADDRESSES FOR OTHER ROUTINES
138A 7F 139A R GETADR  CLR GAFLG
138D 73 139A R         COM GAFLG        SET "GETAD" FLAG
1390 BD 172F R         JSR NXTOK        GET NEXT TOKEN
1393 BD 3DED R         JSR EVADIM+3     GET-ADDRESS CODE GENERATION
1396 7D 24CC R         TST ERNUM        ERROR CHECK
1399 39                RTS
                    *
139A 0001           GAFLG  RMB 1
139B 0001           GAFLG2 RMB 1
                    *
                    * OUTPUT RUNTIME CALL
                    * ROUTINE NUMBER IN B-REGISTER
                    *
139C 4F             RTBL   CLR A
139D CE 13C3 R         LDX #RTBL2+5
13A0 BD 2487 R         JSR CVBTD
13A3 CE 13B3 R         LDX #RTBL1
13A6 8D 03             BSR STR1
13A8 CE 13BE R         LDX #RTBL2
13AB 86 64          STR1   LDA A #100
13AD B7 06F0 R         STA A CMPCNT
13B0 7E 0DAF R         JMP OUTCMP       CALL TO RUN-TIME
                    *
13B3 20             RTBL1  FCC ' JSR  RTBL'
13B0 0D                FCB #0D
13BE 20             RTBL2  FCC ' FCB '
13C3 0005              RMB 5
13C8 0D                FCB #0D
                    *
                    *
13C9 86 78          FADD   LDA A #120    FLOATING ADD
```

```
13CB B7 12E2 R          STA A UBFLAG
13CE 86 6F              LDA A #111
13D0 B7 20E5 R          STA A TMPVAL
13D3 86 72              LDA A #114
13D5 B7 20E6 R          STA A TMPVAL+1
13D8 20 17              BRA FBINOP
                *
13DA 86 7B      FSUB    LDA A #123      FLOATING SUBTRACT
13DC 20 ED              BRA FADD+2
                *
13DE 86 7E      FMUL    LDA A #126      FLOATING MULTIPLY
13E0 20 E9              BRA FADD+2
                *
13E2 86 81      FDIV    LDA A #129      FLOATING DIVIDE
13E4 B7 12E2 R          STA A UBFLAG
13E7 86 72              LDA A #114
13E9 B7 20E5 R          STA A TMPVAL    NOTE REVERSED ARGS
13EC 86 6F              LDA A #111
13EE B7 20E6 R          STA A TMPVAL+1
                *
13F1 BD 131C R FBINOP   JSR CHKBIN      CHECK FOR TWO OPS
13F4 BD 1C26 R          JSR STAKDN      ROLL STACK DOWNWARD
                *
                * GENERATE FLOATING OP CODE:
                *       LDX #ADDRESS 1
                *       JSR FTAREG
                *       LDX #ADDRESS 2
                *       JSR FTCREG
                *       JSR FUNCTION
                *       LDX #ADDRESS 1
                *       JSR FEDREG
                *
13F7 B6 1C03 R          LDA A NUMBER
13FA F6 1C04 R          LDA B NUMBER+1
13FD CE 1264 R          LDX #FCLN1+13
1400 BD 2487 R          JSR CVBTD
1403 CE 1257 R          LDX #FCLN1
1406 BD 13AB R          JSR STR1        OUTPUT LINE 1
1409 F6 20E5 R          LDA B TMPVAL
140C BD 139C R          JSR RTBL        OUTPUT LINE 2
140F B6 1C05 R          LDA A AX
1412 F6 1C06 R          LDA B AX+1
1415 CE 1264 R          LDX #FCLN1+13
1418 BD 2487 R          JSR CVBTD
141B CE 1257 R          LDX #FCLN1
141E BD 13AB R          JSR STR1        OUTPUT LINE 3
1421 F6 20E6 R          LDA B TMPVAL+1
1424 BD 139C R          JSR RTBL        OUTPUT LINE 4
1427 F6 12E2 R          LDA B UBFLAG
142A BD 139C R          JSR RTBL        OUTPUT LINE 5
142D B6 1C03 R          LDA A NUMBER
1430 F6 1C04 R          LDA B NUMBER+1
1433 CE 1264 R          LDX #FCLN1+13
1436 BD 2487 R          JSR CVBTD
1439 CE 1257 R          LDX #FCLN1
143C BD 13AB R          JSR STR1        OUTPUT LINE 6
143F C6 75              LDA B #117
1441 7E 139C R          JMP RTBL        OUTPUT LINE 7
                *
                *
                * "PROCESS BYTE" COMPILER OPCODE
                * BYTE A=B (LOW-ORDER BYTE OF 'B' STORED AT 'A')
                *
1444 8D 57      BYTEPR  BSR SSETUP      GET ADDRESS OF 'A'
1446 CE 15F3 R          LDX #STRP1
1449 BD 13AB R          JSR STR1        "STX ACCUM4"
144C 7F 0578 R          CLR FLPFLG      FORCE INTEGER MODE
144F BD 1C47 R          JSR EVALS       EVALUATE 'B'
1452 CE 0F86 R          LDX #STRL1
1455 BD 13AB R          JSR STR1        "STX ACCUM3"
1458 CE 1601 R          LDX #STRP2
145B BD 13AB R          JSR STR1        "LDX ACCUM4"
145E C6 54              LDA B #84
1460 BD 139C R          JSR RTBL        "POKE"
1463 7E 0937 R          JMP MAIN1       DONE!
                *
                *
                * PROCESS "STRING" COMPILER OPCODE
                * FOUR POSSIBLE FORMS
                *
                * FORMS 1,2  ----ASSIGN A STRING
                * FORM  3    ----COMPARE TWO STRINGS
                * FORM  4    ----CONCATENATE STRINGS
                *
                * 1   STRING A='------'
                * 2   STRING A=B     (B IS A STRING)
                * 3   STRING IF B  COP. C  (B,C STRINGS OR '------')
                *                   (COP IS COMPARISON OPERATOR)
                *                   ( EQ   NE.   GE   GT  .LE   LT )
                *                   ( FORMS LIKE "IF" STATEMENT)
                *                   ( IF()THEN, IF()THEN-ELSE, IF()STMT )
                * 4   STRING A=B,C...  (A NUMBER OF STRINGS ARE CONCATENATE
                *                      ("B" AND "C" MAY BE EITHER NAMES OR
                *                      QUOTED CHARACTERS LIKE FORMS 1 AND 2
                *
1466 FE 06F1 R STRPR    LDX CUCHAR      SAVE TOKEN POINTER
1469 FF 1C05 R          STX AX
146C BD 172F R          JSR NXTOK       GET NEXT TOKEN
146F BD 1B52 R          JSR MNLKP       CHECK FOR "IF"
1472 8C 36CC R          CPX #IFPR
1475 26 03              BNE *+5         NOT "IF"
1477 7E 1655 R          JMP STRCMP      STRING-IF
                *
147A FE 1C05 R          LDX AX
147D FF 06F1 R          STX CUCHAR      RESTORE TOKEN POINTER
1480 8D 1B              BSR SSETUP      GET ADDRESS OF 'A'
1482 27 09              BEQ STR2        ERROR?
                *
1484 CE 0312    STRBAD  LDX #$0312      ERROR MESSAGE
1487 BD 24E8 R          JSR PRINTE
148A 7E 0937 R          JMP MAIN1       QUIT
                *
148D BD 172F R STR2     JSR NXTOK       GET NEXT TOKEN
1490 FE 0642 R          LDX DESCRA
1493 FF 06F1 R          STX CUCHAR      BACK UP TOKEN POINTERS
1496 C1 27              CMP B #$27      TOKEN A QUOTE?
1498 26 64              BNE STRP        IF NOT, STRING FORM 2
                *
149A 7E 1541 R          JMP STRQ        IF SO, STRING FORM 1
                *
                * SET-UP ROUTINE FOR STRING FUNCTIONS
                *
```

```
149D BD 3812 R SSETUP  JSR PRESCN    GET FLOAT TYPE
14A0 BD 138A R         JSR GETADR    GET ADDRESS OF LEFT VALUE
14A3 26 11             BNE SSTERR    ERROR?
                 *
14A5 FE 0642 R         LDX DESCRA
14A8 A6 00             LDA A 0,X     CHECK LAST CHARACTER
14AA 81 29             CMP A #')     RIGHT PAREN?
14AC 26 04             BNE *+6       NO
                 *
14AE BD 172F R         JSR NXTOK     GET NEW TOKEN
14B1 17                TBA
                 *
14B2 81 3D             CMP A #'=     EQUALS SIGN?
14B4 27 04             BEQ SSTUP     YES, RETURN
                 *
14B6 31          SSTERR INS          SKIP RETURN ON STACK
14B7 31                INS
14B8 20 CA             BRA STRBAD    ERROR MESSAGE
                 *
14BA 39          SSTUP  RTS
                 *
                 *
                 * "CONVERT" COMPILER OPCODE
                 * CONVERT STRING=EXPRESSION
                 * CHANGES A VALUE INTO A STRING
                 *
14BB 8D E0       CONVPR BSR SSETUP      SET UP FOR STRINGS
                 *
                 * ADDRESS OF 'A' IN X-REGISTER
                 *       STX ACCUM4
                 * GET VALUE OF 'B' IN X-REGISTER
                 *       LDX #RUNSTK     (FLOATING MODE ONLY)
                 *       STX ACCUM3
                 *       LDX ACCUM4
                 *       JSR ASCII
                 *
14BD CE 15F3 R         LDX #STRP1
14C0 BD 13AB R         JSR STR1        LINE 1
14C3 BD 3812 R         JSR PRESCN      GET FLOAT MODE
14C6 BD 1C47 R         JSR EVALS       GET VALUE
14C9 7D 24CC R         TST ERNUM       ERROR?
14CC 26 B6             BNE STRBAD      IF SO, QUIT
                 *
14CE 7D 0578 R         TST FLPFLG      FLOATING?
14D1 27 0E             BEQ STRN        NO
                 *
14D3 4F                CLR A           TOP OF STACK
14D4 5F                CLR B
14D5 CE 1264 R         LDX #FCLN1+13
14D8 BD 2487 R         JSR CVBTD
14DB CE 1257 R         LDX #FCLN1
14DE BD 13AB R         JSR STR1        "LDX #RUNSTK"
                 *
14E1 CE 0F86 R STRN    LDX #STRL1
14E4 BD 13AB R         JSR STR1        LINE 2
14E7 CE 1601 R         LDX #STRP2
14EA BD 0DAF R         JSR OUTCMP      LINE 3
14ED 7D 0578 R         TST FLPFLG      CHECK MODE OF 'B'
14F0 27 04             BEQ STRO        INTEGER
                 *
                 * FLOATING POINT CODE ("CONVERT")
                 *       JSR FASCII
                 *
14F2 C6 AB             LDA B #171
14F4 20 02             BRA *+4
                 *
14F6 C6 42       STRO  LDA B #66       LINE 4
14F8 BD 139C R         JSR RTBL
14FB 7E 0937 R         JMP MAIN1       DONE!
                 *
                 * PROCESS FORM 2.  .  GENERATE CODE AS FOLLOWS:
                 *       STX ACCUM4
                 * GET ADDRESS OF 'B' IN X-REGISTER
                 *       STX ACCUM3
                 *       LDX ACCUM4
                 *       STX ACCUM1
                 *       LDX ACCUM3
                 *       JSR XFER
                 *
14FE CE 15F3 R STRP    LDX #STRP1
1501 BD 13AB R         JSR STR1        LINE 1
1504 BD 138A R         JSR GETADR      GET ADDRESS
1507 27 03             BEQ *+5
1509 7E 1484 R         JMP STRBAD      FOUND ERROR
                 *
150C CE 0F86 R         LDX #STRL1
150F BD 13AB R         JSR STR1        LINE 2
1512 CE 1601 R         LDX #STRP2
1515 BD 13AB R         JSR STR1        LINE 3
1518 CE 12EE R         LDX #UNOPL2
151B BD 13AB R         JSR STR1        LINE 4
151E CE 161D R         LDX #STRP4
1521 BD 13AB R         JSR STR1        LINE 5
1524 C6 48             LDA B #72
1526 BD 139C R         JSR RTBL        LINE 6
1529 FE 0642 R MERGE   LDX DESCRA      CHECK LAST TOKEN
152C A6 00             LDA A 0,X
152E 81 2C             CMP A #',       COMMA?
1530 26 03             BNE *+5         NO
                 *
1532 7E 14BD R         JMP STR2        YES, APPEND NEXT STRING
                 *
1535 81 0D             CMP A #$0D      END-OF-LINE?
1537 26 03             BNE *+5         NO
                 *
1539 7E 0937 R         JMP MAIN1       DONE!
                 *
153C BD 172F R         JSR NXTOK       GET NEXT TOKEN
153F 20 E8             BRA MERGE       KEEP LOOKING
                 *
                 *
                 * PROCESS FORM 1  .  GENERATE CODE AS FOLLOWS
                 * ADDRESS OF 'A' IN X-REGISTER
                 *       STX ACCUM1
                 *       LDX #*+9
                 *       JSR XFER
                 *       BRA  LABEL
                 *       FCC '-------'
                 *       FCB $04
                 * LABEL EQU *
                 *
1541 BD 172F R STRO    JSR NXTOK       GET NEXT TOKEN
```

```
1544 CE 12EF R          LDX #UNOPL2
1547 BD 13AB R          JSR STR1        LINE 1
154A CE 2960 R          LDX #QL1
154D BD 0DAF R          JSR OUTCMP      LINE 2
1550 C6 48              LDA B #72
1552 BD 139C R          JSR RTBL        LINE 3
1555 BD 298D R          JSR QUOT2       HANDLE QUOTED STRING
1558 20 CF              BRA MERGE       FINISH UP
                *
                *
                * "VALUE" COMPILER OPCODE
                * VALUE A=STRING
                * CHANGES A STRING INTO A VALUE
                *
155A BD 149D R VALUPR   JSR SSETUP      SET UP FOR STRINGS
                *
                * (ADDRESS OF A IN X-REGISTER)
                *       STX ACCUM4
                * GET ADDRESS OF B IN X-REGISTER
                *       JSR VAL         (INTEGER MODE)
                *       STX ACCUM3      "
                *       LDX ACCUM4      "
                *       LDA A ACCUM3    "
                *       LDA B ACCUM3+1  "
                *       STA A 0,X       "
                *       STA B 1,X       "
                *
155D CE 15F3 R          LDX #STRP1
1560 BD 13AB R          JSR STR1        OUTPUT LINE 1
1563 BD 138A R          JSR GETADR      GET ADDRESS
1566 27 03              BEQ *+5         NO ERROR FOUND
1568 7E 1484 R          JMP STRBAD      ERROR, QUIT
                *
156B 7D 0578 R          TST FLPFLG      CHECK MODE OF 'A'
156F 27 08              BEQ STRV1       INTEGER
                *
                * FLOATING MODE ("VALUE")
                *       JSR FVAL
                *
1570 C6 A8              LDA B #168
1572 BD 139C R          JSR RTBL
1575 7E 0937 R          JMP MAIN1       DONE!
                *
                *
1578 C6 3F      STRV1   LDA B #63       RUNTIME CALL
157A BD 139C R          JSR RTBL
157D CE 0F86 R STRV2    LDX #STRL1
1580 BD 0DAF R          JSR OUTCMP      OUTPUT LINE 3
1583 CE 1601 R          LDX #STRP2
1586 BD 0DAF R          JSR OUTCMP      OUTPUT LINE 4
1589 BD 0F6E R          JSR STRD2       FINISH UP
158C 7E 0937 R          JMP MAIN1       DONE!
                *
                *
                * "LENGTH" COMPILER OPCODE
                * LENGTH A=STRING
                * GETS THE NUMBER OF CHARACTERS IN A STRING
                * "A" MUST BE INTEGER TYPE
                *
158F BD 149D R LENGPR   JSR SSETUP      SET UP FOR STRINGS
1592 7D 0578 R          TST FLPFLG      CHECK FLOAT MODE
1595 27 09              BEQ STRLN2      INTEGER
                *
1597 CE 0333    STRLN1  LDX #$0333      MISMATCH ERROR
159A BD 24E8 R          JSR PRINTE
159D 7E 0937 R          JMP MAIN1
                *
15A0 CE 15F3 R STRLN2   LDX #STRP1
15A3 BD 13AB R          JSR STR1        OUTPUT LINE 1
15A6 BD 138A R          JSR GETADR      GET ADDRESS
15A9 27 03              BEQ *+5         NO ERROR FOUND
15AB 7E 1484 R          JMP STRBAD      ERROR, QUIT
                *
15AE C6 3C              LDA B #60
15B0 BD 139C R          JSR RTBL        RUN-TIME CALL
15B3 20 C8              BRA STRV2       FINISH UP LIKE "VALUE"
                *
                *
                * "EXTRACT" COMPILER OPCODE
                * XTRACT STRING=EXPRESSION,STRING
                * GETS A SUBSTRING FROM A GIVEN STRING
                * "EXPRESSION" MUST BE INTEGER TYPE
                *
15B5 BD 149D R XTRPR    JSR SSETUP      SET UP FOR STRINGS
                *
                * (ADDRESS OF A IN X-REGISTER)
                *       STX ACCUM4
                * GET VALUE OF LENGTH IN X-REGISTER
                *       STX ACCUM3
                * GET ADDRESS OF STRING (B) INTO X-REGISTER
                *       JSR MID
                *
15B8 7D 0578 R          TST FLPFLG      TEST FLOAT MODE
15BB 26 DA              BNE STRLN1      MUST BE INTEGER
                *
15BD CE 15F3 R          LDX #STRP1
15C0 BD 13AB R          JSR STR1        OUTPUT LINE 1
15C3 7F 0578 R          CLR FLPFLG      FORCE INTEGER
15C6 BD 1C47 R          JSR EVALS       GET LENGTH VALUE
15C9 7D 24CC R          TST ERNUM       ERROR?
15CC 27 03              BEQ *+5         NO
15CF 7E 1484 R          JMP STRBAD      ERROR, QUIT
                *
15D1 CE 0F86 R          LDX #STRL1
15D4 BD 13AB R          JSR STR1        OUTPUT LINE 2
15D7 FE 0642 R          LDX DESCRA
15DA A6 00              LDA A 0,X
15DC 81 0D              CMP A #$0D      AT END OF LINE?
15DE 26 03              BNE *+5         NO, O.K
15E0 7E 1484 R          JMP STRBAD      IF SO, ERROR
                *
15E3 BD 138A R          JSR GETADR      GET ADDRESS
15E6 27 03              BEQ *+5         NO ERROR FOUND
15E8 7E 1484 R          JMP STRBAD      ERROR, QUIT
                *
15EB C6 45              LDA B #69       RUN-TIME CALL
15ED BD 139C R          JSR RTBL
15F0 7E 0937 R          JMP MAIN1       DONE!
                *
15F3 20         STRP1   FCC ' STX ,ACCUM+6'
1600 0D                 FCB $0D
1601 20         STRP2   FCC ' LDX  ACCUM+6'
```

```
160E 0D                 FCB $0D
160F 20          STRP3  FCC ' LDX . ACCUM+8'
161C 0D                 FCB $0D
161D 20          STRP4  FCC ' LDX . ACCUM+4'
162A 0D                 FCB $0D
162B 20          STRP5  FCC ' STX . ACCUM+8'
1638 0D                 FCB $0D
1639 20          STRP6  FCC ' STX  ACCUM+2'
1646 0D                 FCB $0D
1647 20          STRP7  FCC ' LDX . ACCUM+2'
1654 0D                 FCB $0D
                 *
                 * PROCESS FORM 3.    STRING COMPARES
                 * STRINGS MAY BE VARIABLES OR QUOTED CHARACTERS
                 * ALL SIX COMPARISON OPERATORS LEGAL
                 * HANDLES ALL "IF" FORMS
                 *
1655 FE 06F1 R STRCMP   LDX CUCHAR     SAVE POINTER
1658 FF 1C05 R          STX AX
165D BD 172F R          JSR NXTOK      GET NEXT TOKEN
165E C1 27              CMP B #$27     A QUOTE?
1660 27 26              BEQ STRC3      YES
                 *
1662 C1 01              CMP B #1       A NAME?
1664 27 03              BEQ *+5        YES
                 *
1666 7E 1484 R STRCER   JMP STRBAD     ERROR
                 *
1669 FE 1C05 R          LDX AX
166C FF 06F1 R          STX CUCHAR     RESTORE POINTERS
166F BD 138A R          JSR GETADR     GET ADDRESS OF VARIABLE
1672 26 F2              BNE STRCER     ERROR?
                 *
1674 CE 15F3 R          LDX #STRP1     " STX ACCUM4"
1677 BD 13AB R          JSR STR1
167A FE 0642 R          LDX DESCRA     BACK UP TOKEN POINTER?
167D A6 00              LDA A 0,X
167F 81 29              CMP A #')      AT A RIGHT PAREN?
1681 27 14              BEQ STRC4      IF SO, SKIP BACKUP
                 *
1683 FF 06F1 R          STX CUCHAR     BACK UP A TOKEN
1686 20 0F              BRA STRC4
                 *
1688 CE 16E2 R STRC3    LDX #STRCL2
168B BD 13AB R          JSR STR1       LINE 1
                 *
                 * CODE:  LDX #*+8
                 *        STX ACCUM4
                 *        BRA .LABEL
                 *        FCC '-----'
                 *        FCB $04
                 * LABEL  EQU *
                 *
168E CE 15F3 R          LDX #STRP1
1691 BD 13AB R          JSR STR1       LINE 2
1694 BD 298D R          JSR QUOT2      LINES 3-6
1697 BD 172F R STRC4    JSR NXTOK      GET NEXT TOKEN
169A C1 01              CMP B #1       NAME?
169C 26 C8              BNE STRCER     IF NOT, ERROR
                 *
169E BD 1B52 R          JSR MNLKP      MNEMONIC?
16A1 4D                 TST A
16A2 26 C2              BNE STRCER     IF NOT, ERROR
                 *
16A4 C5 40              BIT B #$40     COMPARISON OP?
16A6 27 BE              BEQ STRCER     IF NOT, ERROR
                 *
16A8 C0 41              SUB B #$41     REMOVE BIAS
16AA 2B BA              BMI STRCER     ERROR?
                 *
16AC 37                 PSH B          SAVE OPERATOR
16AD FE 06F1 R          LDX CUCHAR
16B0 FF 1C05 R          STX AX
16B3 BD 172F R          JSR NXTOK      GET NEXT TOKEN
16B6 C1 27              CMP B #$27     A QUOTE?
16B8 27 13              BEQ STRC5      YES
                 *
16BA C1 01              CMP B #1       NAME?
16BC 26 A8              BNE STRCER     IF NOT, ERROR
                 *
16BE FE 1C05 R          LDX AX
16C1 FF 06F1 R          STX CUCHAR     RESTORE POINTER
16C4 BD 138A R          JSR GETADR     GET ADDRESS OF VARIABLE
16C7 27 23              BEQ STRC6      NO ERROR
                 *
16C9 33                 PUL B
16CA 7E 1484 R          JMP STRBAD     ERROR
                 *
16CD CE 16D8 R STRC5    LDX #STRCL1
16D0 BD 13AB R          JSR STR1       LINE 1
                 *
                 * CODE:  LDX #*+5
                 *        BRA .LABEL
                 *        FCC '-----'
                 *        FCB $04
                 * LABEL  EQU *
                 *
16D3 BD 298D R          JSR QUOT2      LINES 2-5
16D6 20 1D              BRA STRC6A
                 *
16D8 20          STRCL1 FCC ' LDX #*+5'
16E1 0D                 FCB $0D
16E2 20          STRCL2 FCC ' LDX #*+8'
16EB 0D                 FCB $0D
                 *
16EC FE 0642 R STRC6    LDX DESCRA
16EF A6 00              LDA A 0,X
16F1 81 29              CMP A #')      AT RIGHT PAREN?
16F3 26 03              BNE *+5        IF NOT, O.K
                 *
16F5 BD 172F R STRC6A   JSR NXTOK      MOVE PAST TOKEN
                 *
16F8 C6 4E              LDA B #78
16FA BD 139C R          JSR RTBI       RUNTIME CALL
16FD CE 1709 R          LDX #CTBI
1700 33                 PUL B          GET OPERATOR BACK
1701 5D                 TST B
1702 27 11       STRC6B BEQ STRC7
1704 08                 INX
1705 08                 INX
1706 5A                 DEC B
1707 20 F9              BRA STRC6B     LOOP UNTIL FOUND
```

```
                        *
1709 3604    R CTBL     FDB EQLN        BRANCH CODE TABLE
170B 3612    R          FDB GELN
170D 3620    R          FDB GTLN
170F 362E    R          FDB LELN
1711 363C    R          FDB LTLN
1713 364A    R          FDB NELN
                        *
1715 EE 00              STRC7   LDX 0,X         INDIRECT ADDRESS
1717 BD 13AB R                  JSR STR1        OUTPUT BRANCH
171A CE 3691 R                  LDX #COMPL2
171D BD 13AB R                  JSR STR1        "LDX #0"
1720 CE 3699 R                  LDX #COMPL3
1723 BD 13AB R                  JSR STR1        "BRA *+5"
1726 CE 36AA R                  LDX #COMPL5
1729 BD 13AB R                  JSR STR1        "LDX #$FFFF"
172C 7E 36D7 R                  JMP IFERO       FINISH UP LIKE "IF"
                        *
                        *
                        *
                        * NEXT TOKEN ROUTINE *
                        * SCANS A LINE OF SOURCE CODE AND RETURNS
                        * THE NEXT TOKEN,CLASS & RC IN REGS A,B
                        * THE ADDRESS OF THE TOKEN IS RETURNED IN
                        * DESCRA AND THE # OF BYTES IN THE TOKEN IS
                        * RETURNED IN DESCRC
                        * THE RC AND CLASS ARE.
                        *
                        * TYPE    RC [B]    CLASS [A]
                        *
                        * NAME    01        02  SUBSTRINGS
                        * HEX     03        02
                        * DECIMAL 09        02
                        * FLOAT   0F        02
                        *
                        * #       23        04  DELIMITERS
                        * ,       2C        04
                        * '       27        04
                        * %       25        04
                        * $       24        04  (HEX INDICATOR)
                        * >       3E        04  (SPACE FUNCTION)
                        * [       5B        04  (FORMAT SPECIFIER)
                        * ]       5D        04
                        *
                        * (       28        08  PARENTHESES
                        * )       29        08
                        *
                        * =       3D        40  ASSIGNMENT
                        *
                        *
                        * *       2A        24  ARITHMETIC
                        * /       2F        24
                        * +       2B        24
                        * -       2D        24
                        *
                        * A       41        01  A,B,X REGS
                        * B       42        01
                        * X       58        01
                        *
                        * CR      0D        0D  EOL
                        *
```

```
                        * ERROR   00        00  ERRORS
                        *
                        *
172F 7F 0644 R NXTOP    CLR   DESCRC
1732 7C 0644 R          INC   DESCRC    DESCRC =1
1735 FE 06F1 R NXTO     LDX   CUCHAR    POINT TO CURRENT CHAR
1738 FF 0642 R          STX   DESCRA    INIT DESCRA
173B A6 00              LDA A 0,X       GET CHAR
173D 08                 INX
173E FF 06F1 R          STX   CUCHAR    POINT TO NEXT CHAR
1741 81 20              CMP A #$20      LESS THAN 20 HEX?
1743 27 F0              BEQ   NXTO      BLANK,SKIP OVER
1745 22 09              BHI   NXT1      >20
                        *
1747 81 0D              CMP A #$0D      CR?
1749 27 03              BEQ   *+5
174B 7E 17DB R          JMP   NXTER     NO,UNRECOG  CHAR
                        *
174E 16                 TAB             YES, SET RC
174F 39                 RTS
                        *
1750 81 5F      NXT1    CMP A #$5F      >5F?
1752 23 03              BLS   NXT3      NO
1754 7E 17DB R          JMP   NXTER     YES,UNRECOG  CHAR
                        *
1757 BD 19C5 R NXT3     JSR   GCHRTB    GET BYTE FROM CHARTAB
175A 85 01              BIT A #$01      A,B,X REGS?
175C 27 1C              BEQ   NXT4      NO
                        *
175E 7D 06F3 R          TST   CFLAG     COMPILING?
1761 26 17              BNE   NXT4      YES,NO REGS
                        *
1763 FE 06F1 R          LDX   CUCHAR    YES,CHECK NEXT CHAR
1766 86 01              LDA A #$01      SET CLASS
1768 E6 00              LDA B 0,X
176A C1 20              CMP B #$20      BLANK?
176C 27 06              BEQ   NXT3A     YES
                        *
176E 86 0D              LDA A #$0D      SET CLASS
1770 C1 0D              CMP B #$0D      EOL?
1772 26 29              BNE   NXT4A     NO,GO TO NSCAN
                        *
1774 FF 0642 R NXT3A    LDX   DESCRA    GET RC
1777 E6 00              LDA B 0,X
1779 39                 RTS             IF DONE, RETURN RC
                        *
177A 85 80      NXT4    BIT A #$80      NAME?
177C 27 21              BEQ   NXT5      NO
                        *
177E 7D 06F3 R          TST   CFLAG     COMPILING?
1781 27 1A              BEQ   NXT4A     NO, SKIP FLOATING POINT
                        *
1783 FE 0642 R          LDX   DESCRA
1786 A6 00              LDA A 0,X
1788 81 2E              CMP A #'.       DECIMAL?
178A 26 11              BNE   NXT4A     NO
                        *
178C A6 01              LDA A 1,X
178E BD 19C5 R          JSR   GCHRTB    NEXT CHAR  DECIMAL?
1791 85 40              BIT A #$40
1793 27 08              BEQ   NXT4A     NO
```

```
                    *
1795 7D 0578 R          TST   FLPFLG     IN FLOATING MODE?
1798 27 41              BEQ   NXTER      IF NOT, ERROR
                    *
179A 7E 185D R          JMP   FSCAN      SCAN FLOATING NUMBER
                    *
179D 20 56          NXT4A BRA   NSCAN      YES SCAN NAME STRING
                    *
179F 85 40          NXT5  BIT A #$40       DECIMAL?
17A1 27 0A              BEQ   NXT6        NO
                    *
17A3 7D 0578 R          TST   FLPFLG     IN FLOATING MODE?
17A6 27 03              BEQ   *+5         NO
                    *
17A8 7E 185D R          JMP   FSCAN      SCAN FLOATING NUMBER
                    *
17AB 20 31              BRA   DSCAN      YES, SCAN DECIMAL STRING
                    *
17AD 85 20          NXT6  BIT A #$20       ARITHMETIC?
17AF 26 C3              BNE   NXT3A       YES GET RC AND RTN
                    *
17B1 85 04              BIT A #$04       DELIMITERS?
17B3 27 0B              BEQ   NXT7        NO
17B5 FE 0642 R          LDX   DESCRA     GET CHAR
17B8 E6 00              LDA B 0,X
17BA C1 24              CMP B #'$        $? (HEX)
17BC 26 B6              BNE   NXT3A       NO, GET RC AND RTN
                    *
17BE 20 72              BRA   HSCAN      YES, SCAN HEX STRING
                    *
17C0 85 10          NXT7  BIT A #$10       PAREN OR ASSIGN?
17C2 27 17              BEQ   NXTER       NEITHER-ERROR
                    *
17C4 FE 0642 R          LDX   DESCRA
17C7 E6 00              LDA B 0,X        GET CHARACTER
17C9 C1 28              CMP B #'(        LEFT PAREN?
17CB 26 03              BNE   NXT9
                    *
17CD 86 08          NXT8  LDA A #8         SET CLASS
17CF 39                 RTS
                    *
17D0 C1 29          NXT9  CMP B #')        RIGHT PAREN?
17D2 27 F9              BEQ   NXT8
                    *
17D4 C1 3D          NXT10 CMP B #'=        ASSIGNMENT?
17D6 26 03              BNE   NXTER       UNDEFINED CHAR, ERROR
                    *
17D8 86 40              LDA A #$40       SET CLASS
17DA 39                 RTS
                    *
17DB 4F             NXTER CLR A            TROUBLE, SET RC, CLASS=00
17DC 5F                 CLR B
17DD 39                 RTS
                    * DSCAN SCAN DECIMAL STRING STOP AT
                    * FIRST NON-DECIMAL CHAR
                    *
17DE FE 06F1 R      DSCAN LDX   CUCHAR     POINT TO NEXT CHAR
17E1 A6 00              LDA A 0,X        GET CHAR
17E3 7C 0644 R          INC   DESCRC     BUMP COUNT
17E6 08                 INX
17E7 FF 06F1 R          STX   CUCHAR     POINT TO NEXT CHAR
17EA BD 19C5 R          JSR   GCHRTB     GET BYTE IN CHARTAB
                    *
17ED 85 40              BIT A #$40       DECIMAL?
17EF 26 ED              BNE   DSCAN      YES CONTINUE SCAN
                    *
17F1 C6 09              LDA B #$09
17F3 20 5B              BRA   ENDSCN     RETURN
                    *
                    *
                    * NSCAN SCAN NAME STRING STOP AT
                    * FIRST NON-ALPHANUMERIC CHAR
                    *
17F5 FE 06F1 R      NSCAN LDX   CUCHAR     POINT TO NEXT CHAR
17F8 A6 00              LDA A 0,X        GET CHAR
17FA 81 2E              CMP A #'.        PERIOD?
17FC 26 18              BNE   NSCAN3      NO
                    *
17FE FE 0642 R          LDX   DESCRA
1801 A6 00              LDA A 0,X
1803 81 2E              CMP A #'.        WAS FIRST CHAR  POINT?
1805 26 0A              BNE   NSCAN2      NO
                    *
1807 FE 06F1 R          LDX   CUCHAR     INCLUDE PERIOD IN TOKEN
180A 08                 INX
180B FF 06F1 R          STX   CUCHAR
180E 7C 0644 R          INC   DESCRC
1811 C6 01          NSCAN2 LDA B #1        SET RC
1813 86 02              LDA A #2         SET CLASS
1815 39                 RTS
                    *
1816 7C 0644 R      NSCAN3 INC   DESCRC    BUMP COUNT
1819 08                 INX
181A FF 06F1 R          STX   CUCHAR     POINT TO NEXT CHAR
181D BD 19C5 R          JSR   GCHRTB     GET BYTE IN CHARTAB
1820 85 C0              BIT A #$C0       ALPHANUMERIC?
1822 26 D1              BNE   NSCAN       YES, CONTINUE SCAN
                    *
1824 C6 07              LDA B #$07       NAME TOO LONG ?
1826 F1 0644 R          CMP B DESCRC
1829 24 03              BCC   NSCANA      NO
                    *
182B F7 0644 R          STA B DESCRC     YES, TRUNCATE
182E C6 01          NSCANA LDA B #$01    LOAD RC
1830 20 1E              BRA   ENDSCN     RETURN
                    *
                    *
                    * HSCAN SCAN HEX STRING STOP AT
                    * FIRST NON-HEX CHAR
                    *
1832 7F 0644 R      HSCAN CLR   DESCRC     DESCRC,=0
1835 FE 06F1 R          LDX   CUCHAR     POINT TO NEXT CHAR
1838 FF 0642 R          STX   DESCRA     INIT DESCRA
183B FF 06F1 R      HSCAN1 LDX  CUCHAR     POINT TO NEXT CHAR
183E A6 00              LDA A 0,X        GET CHAR
1840 7C 0644 R          INC   DESCRC     BUMP COUNT
1843 08                 INX
1844 FF 06F1 R          STX   CUCHAR     POINT TO NEXT CHAR
1847 BD 19C5 R          JSR   GCHRTB     GET BYTE IN CHRTAB
184A 85 02              BIT A #$02       HEX?
184C 26 ED              BNE   HSCAN1     YES CONTINUE SCAN
                    *
```

```
184F C6 03                LDA B #$03
                *
1850 7A 0644 R ENDSCN DEC   DESCRC    DESCRC = CORRECT COUNT
1853 FE 06F1 R        LDX   CUCHAR
1856 09               DEX
1857 FF 06F1 R        STX   CUCHAR    CUCHAR = CORRECT VALUE
185A 86 02            LDA A #2        LOAD CLASS RC
185C 39               RTS             ALL DONE
                *
                * FLOATING-POINT SCANNER
                * BCD-FORMAT LEFT IN FTEMP
                * FORMAT HAS 9 BYTES AS FOLLOWS:
                * FE-EE-MM-MM-MM-MM-MM-MM-MM
                * WHERE 'E' IS AN EXPONENT DIGIT
                *       'M' IS A  MANTISSA DIGIT
                *       'F' IS SIGNS/FLAGS
                * BASE 10 EXPONENT
                *
                *
185D CE 1988 R FSCAN     LDX #FTEMP
1860 C6 09               LDA B #9
1862 6F 00     FR1       CLR 0,X       CLEAR FTEMP
1864 08                  INX
1865 5A                  DEC B
1866 26 FA               BNE FR1
                *
1868 7F 1982 R           CLR DECFLG    CLEAR FLAGS
186B 7F 1984 R           CLR SIGDIG
186E 7F 1985 R           CLR EXPTMP
1871 CE 1988 R           LDX #FTEMP
1874 C6 0E               LDA B #14     14 DIGITS OF MANTISSA
1876 F7 1987 R           STA B TEMP
1879 C6 20               LDA B #$20
187B E7 00               STA B 0,X     SET ZERO BIT
187D C6 FF               LDA B #$FF    BYTE TOGGLE
187F F7 1983 R           STA B NIBBLE
1882 BD 196F R           JSR SCNCHR    GET A CHARACTER
1885 81 2E               CMP A #'      DECIMAL POINT?
1887 26 06               BNE CRCHK     NO, CHECK FOR CR
                *
1889 73 1982 R           COM DECFLG    FOUND DECIMAL
                *
188C BD 196F R GETIN     JSR SCNCHR    GET A CHARACTER
188F 81 0D     CRCHK     CMP A #$0D    CR?
1891 26 03               BNE *+5
1893 7E 192A R           JMP SETEXP    YES
                *
1896 80 30               SUB A #$30    REMOVE ASCII BIAS
1898 27 10               BEQ GOTZER    ZERO?
                *
189A 2B 42               BMI NOTDON    <0?
                *
189C 81 09               CMP A #9      >9?
189E 22 3E               BHI NOTDON
                *
18A0 7D 1984 R           TST SIGDIG    SIGNIF  DIGIT?
18A3 26 05               BNE GOTZER    YES
                *
18A5 6F 00               CLR 0,X       TURN OFF ZERO BIT
18A7 7C 1984 R           INC SIGDIG    HAVE SIG  DIGIT
                *
18AA 7D 1987 R GOTZER    TST TEMP      FULL BUFFER?
18AD 27 0D               BEQ GETIN     YES, SKIP DIGIT
                *
18AF 7D 1984 R           TST SIGDIG    SIG  DIGIT YET?
18B2 26 0A               BNE TSTNXT    YES
                *
18B4 7D 1982 R           TST DECFLG    DEC  POINT YET?
18B7 27 03               BEQ GETIN     NO
                *
18B9 7A 1985 R           DEC EXPTMP    BACK UP EXPONENT
18BC 20 CE               BRA GETIN
                *
18BE 7D 1982 R TSTNXT    TST DECFLG    DEC  POINT YET?
18C1 26 03               BNE *+5       YES, EXP. O.K
                *
18C3 7C 1985 R           INC EXPTMP    INCREASE EXPONENT
                *
18C6 7D 1983 R           TST NIBBLE    WHICH HALF?
18C9 27 04               BEQ LEFTH
                *
18CB 48                  ASL A
18CC 48                  ASL A
18CD 48                  ASL A
18CE 48                  ASL A         SHIFT OVER
18CF AA 02     LEFTH     ORA A 2,X     MERGE IN
18D1 A7 02               STA A 2,X
18D3 73 1983 R           COM NIBBLE
18D6 27 01               BEQ *+3       CHECK FOR NEW BYTE
                *
18D8 08                  INX           MOVE POINTER
                *
18D9 7A 1987 R           DEC TEMP      CHECK FOR BUFFER END
18DC 20 AE               BRA GETIN
                *
18DE 8B 30     NOTDON    ADD A #$30    RESTORE BIAS
18E0 81 45               CMP A #'E     EXPONENT?
18E2 27 0E               BEQ EXPIN     YES
                *
18E4 81 2E               CMP A #'      DECIMAL POINT?
18E6 26 42               BNE SETEXP    NO
                *
18E8 7D 1982 R           TST DECFLG    HAVE DEC.  POINT YET?
18EB 26 3D               BNE SETEXP    IF SO, DONE
                *
18ED 73 1982 R           COM DECFLG    SET FLAG
18F0 20 9A               BRA GETIN
                *
18F2 8D 7B     EXPIN     BSR SCNCHR    GET CHARACTER
18F4 CE 1988 R           LDX #FTEMP
18F7 7F 1986 R           CLR CNTR
18FA 7F 1984 R           CLR SIGDIG
18FD 81 2B               CMP A #'+     PLUS SIGN?
18FF 27 0A               BEQ EXPIN2    SKIP IT
                *
1901 81 2D               CMP A #'-     MINUS SIGN?
1903 26 08               BNE EXPIN2+2  NO
                *
1905 C6 40               LDA B #$40    SET MINUS BIT
1907 EA 00               ORA B 0,X
1909 E7 00               STA B 0,X
                *
```

```
190B 8D 62      EXPIN2   BSR SCNCHR    GET A CHARACTER
190D 80 30               SUB A #$30    REMOVE BIAS
190F 2B 19               BMI SETEXP    <0?
                 *
1911 81 09               CMP A #9      >9?
1913 23 02               BLS EXPOK     NO
                 *
1915 20 13               BRA SETEXP    YES, FINISH EXPONENT
                 *
1917 E6 01      EXPOK    LDA B 1,X     STORE EXP  DIGIT
1919 58                  ASL B
191A 58                  ASL B
191B 58                  ASL B
191C 58                  ASL B
191D 1B                  ABA
191E A7 01               STA A 1,X
1920 7D 1986 R           TST CNTR      TWO DIGIT EXPONENT?
1923 26 05               BNE SETEXP    IF SO, FINISH EXPONENT
                 *
1925 73 1986 R           COM CNTR      SET FLAG FOR FIRST DIGIT
1928 20 E1               BRA EXPIN2
                 *
                 * FIX UP BCD EXPONENT
                 *
192A CE 1988 R  SETEXP   LDX #FTEMP
192D A6 01               LDA A 1,X
192F 84 0F               AND A #$0F    GET LOW-ORDER DIGIT
1931 36                  PSH A         SAVE IT
1932 A6 01               LDA A 1,X
1934 84 F0               AND A #$F0    GET HIGH-ORDER DIGIT
1936 44                  LSR A
1937 16                  TAB           MULT. HIGH-DIGIT BY 10
1938 44                  LSR A
1939 44                  LSR A
193A 1B                  ABA
193B 33                  PUL B         GET LOW-ORDER DIGIT
193C 1B                  ABA           BINARY EXP =LOW + 10*HIGH
193D E6 00               LDA B 0,X     CHECK EXP  SIGN
193F C5 40               BIT B #$40
1941 27 01               BEQ *+3       PLUS
                 *
1943 40                  NEG A         NEGATE BINARY EXPONENT
                 *
1944 BB 1985 R           ADD A EXPTMP  ADJUST EXPONENT
1947 2A 07               BPL SETX2
                 *
1949 40                  NEG A         MAKE POSITIVE
194A C6 40               LDA B #$40
194C EA 00               ORA B 0,X
194E E7 00               STA B 0,X
                 *
1950 5F         SETX2    CLR B
1951 81 0A      SETX3    CMP A #10     BCD(B)=BINARY(A)
1953 25 06               BCS SETX4
                 *
1955 80 0A               SUB A #10
1957 CB 10               ADD B #$10
1959 20 F6               BRA SETX3
                 *
195B 1B         SETX4    ABA           SET "H" FLAG FOR DAA
195C 19                  DAA           BINARY TO BCD
195D 24 02               BCC *+4       CARRY TO HIGH DIGIT?
                 *
195F 6C 00               INC 0,X       YES
                 *
1961 A7 01               STA A 1,X
1963 C6 0F               LDA B #$0F    SET RC
1965 86 02               LDA A #$02    SET CLASS
1967 FE 06F1 R           LDX CUCHAR    BACK UP POINTER
196A 09                  DEX
196B FF 06F1 R           STX CUCHAR
196E 39                  RTS
                 *
                 * GET A CHARACTER FROM LINE
196F FF 2265 R  SCNCHR   STX DVAL      SAVE X-REGISTER
1972 FE 0642 R           LDX DESCRA    GET POINTER
1975 A6 00               LDA A 0,X     GET CHARACTER
1977 08                  INX
1978 FF 0642 R           STX DESCRA    MOVE POINTERS
197B FF 06F1 R           STX CUCHAR
197E FE 2265 R           LDX DVAL      RESTORE X
1981 39                  RTS
                 *
1982 0001       DECFLG   RMB 1         DECIMAL-FLAG
1983 0001       NIBBLE   RMB 1         BYTE TOGGLE
1984 0001       SIGDIG   RMB 1         SIG.-DIGIT FLAG
1985 0001       EXPTMP   RMB 1         TEMP. EXPONENT
1986 0001       CNTR     RMB 1
1987 0001       TEMP     RMB 1
1988 0009       FTEMP    RMB 9         FLOATING VALUE
1991 00                  FCB 0
                 *
                 * PUT OUT CROSS-REFERENCE BLOCK TO OBJECT FILE
                 *
1992 B6 06F8 R  XREF     LDA A OPTNS   WANT XREF?
1995 85 08               BIT A #$08
1997 26 2B               BNE XREF2     NO
                 *
1999 7D 0575 R           TST PASS      PASS 1?
199C 27 26               BEQ XREF2     NO XREF IN PASS 1
                 *
199E 36                  PSH A
199F 84 BF               AND A #$BF    TURN ON "OBJECT"
19A1 B7 06F8 R           STA A OPTNS
19A4 F6 056F R           LDA B LNUM
19A7 BD 3E55 R           JSR OUTBIN    OUTPUT LINE NUMBER
19AA F6 0570 R           LDA B LNUM+1
19AD BD 3E55 R           JSR OUTBIN
19B0 BD 1ABF R           JSR HASH      FORMAT NAME
19B3 86 06               LDA A #6
19B5 CE 19EC R           LDX #HSMBL
19B8 BD 3124 R           JSR PBLK2     OUTPUT SYMBOL NAME
19BB C6 0D               LDA B #$0D
19BD BD 3E69 R           JSR OUTBNR    OUTPUT "C R "
19C0 32                  PUL A
19C1 B7 06F8 R           STA A OPTNS   RESTORE OPTIONS
19C4 39         XREF2    RTS           DONE!
                 *
                 *
                 * GET BYTE IN CHRTAB INDEXED BY VALUE OF
                 * CHAR IN REG A
                 *
```

```
19C5 81 20      GCHRTB CMP A #$20      VALID CHARACTER?
19C7 25 16             BCS   GCHRTR     NO < 20
19C9 81 5F             CMP A #$5F      VALID CHAR ?
19CB 22 12             BHI   GCHRTR     NO, > 5F
                *
19CD 7F 19E1 R         CLR   CHPTR     INIT PARM
19D0 B7 19E2 R         STA A CHPTR+1   SAVE CHAR
19D3 CE 19E1 R         LDX   #CHPTR    POINT TO PARM
19D6 BD 234B R         JSR   ADD16     ADD IN BASE OF CHARTAB
19D9 FE 19E1 R         LDX   CHPTR     GET BYTE IN CHARTAB
19DC A6 00             LDA A 0,X
19DE 39                RTS
                *
19DF 4F         GCHRTR CLR A
19E0 39                RTS
                *
19E1 0002       CHPTR  RMB   2         PARM LIST
19E3 050F     R        FDB   CHRTAB-$20
                *
                *
                *
                * TABLE MANIPULATION ROUTINES FOR TABLES
                * SYMTAB AND MNTAB
                *
                * STORAGE LOCATIONS USED BY THE ROUTINES:
                *
19E5 0002       PSTNG1 RMB   2         ADDRESS OF MNEMONIC
19E7 0002       PSTNG2 RMB   2         ADDRESS IN THE TABLE
19E9 0001       PCOUNT RMB   1         LENGTH OF MNEMONIC
19EA 0002       TBADD  RMB   2         TABLE POINTER
19EC 0006       HSMBL  RMB   6         SYMBOL TEMP LOC
19F2 0002       HKEYA  RMB   2         HASHED CODE
19F4 0002       HKEYB  RMB   2         TEMP LOC FOR HASHED CODE
19F6 0002       HSAV1  RMB   2         TEMP LOC FOR PTR
19F8 0002       HSAV2  RMB   2
                *
                *
                * STORE A SYMBOL IN SYMTAB
                * ON ENTRY DESCRA CONTAINS ADDRESS OF
                * THE SYMBOL, AND DESCRC CONTAINS THE LENGTH
                * A STANDARD HASH CODED METHOD IS USED
                *
                *
19FA BD 1ABF R STOSYM JSR  HASH        GET HASHED KEY
19FD FF 06F6 R         STX  SYMPTR     SAVE
                *
                * SEE IF LOC(HKEYA) IS EMPTY
                *
1A00 A6 00      SYMA   LDA A 0,X       GET FIRST CHAR
1A02 81 20             CMP A #$20      BLANK ?
1A04 26 35             BNE   SYMB      NO
                *
                * STORE SYMBOL IN SYMTAB
                *
1A06 FF 19F8 R         STX   HSAV2     SAVE TABLE PTR
1A09 CE 19EC R         LDX   #HSMBL    POINT TO HSMBL
1A0C FF 19F6 R         STX   HSAV1     SAVE
1A0F C6 06             LDA B #6        LOAD SYMBOL LENGTH
                *
                * DO TRANSFER
                *
1A11 FE 19F6 R SYM1    LDX   HSAV1     POINT TO HSYMBL
1A14 A6 00             LDA A 0,X       GET CHAR
1A16 08                INX
1A17 FF 19F6 R         STX   HSAV1     POINT TO NEXT CHAR
1A1A FE 19F8 R         LDX   HSAV2     POINT TO TABLE ENTRY
1A1D A7 00             STA A 0,X       STORE CHAR IN SYMTAB
1A1F 08                INX
1A20 FF 19F8 R         STX   HSAV2     POINT TO NEXT POSITION
1A23 5A                DEC B           ALL DONE ?
1A24 26 EB             BNE   SYM1      NO
                *
                * STORE LC, AND SET INFO BYTE
                *
1A26 B6 0573 R         LDA A LC        GET LC
1A29 A7 00             STA A 0,X       STORE
1A2B B6 0574 R         LDA A LC+1      GET LS BYTE OF LC
1A2E A7 01             STA A 1,X       STORE
1A30 4F                CLR A           INFO BYTE CLEAR
1A31 7D 06F3 R         TST   CFLAG     COMPILING?
1A34 26 02             BNE   *+4       YES
                *
1A36 86 40             LDA A #$40      INFO BYTE:=RELOC,DEFINED
                *
1A38 A7 02             STA A 2,X
1A3A 39                RTS             RETURN
                *
                * COMPARE HSMBL WITH ENTRY IN SYMTAB
                *
1A3B 8D 48      SYMB   BSR   SYMCMP    COMPARE
1A3D 26 0F             BNE   SYMC      NO MATCH
                *
                * ERROR, SYMBOL ALREADY IN TABLE
                *
1A3F FE 06F6 R         LDX   SYMPTR    GET ADDRESS OF ENTRY
1A42 86 80             LDA A #$80
1A44 AA 08             ORA A 8,X       SET REDEFINED BIT
1A46 A7 08             STA A 8,X
1A48 CE 0206           LDX   #$0206    LOAD ERROR#
1A4B 7E 24E8 R SYMB1   JMP   PRINTE    PRINT IT
                *
                * FIND ANOTHER SLOT IN SYMTAB FOR SYMBOL
                *
1A4E 8D 49      SYMC   BSR   SYMMOD    GET A(NEXT SLOT)
1A50 BC 19F2 R         CPX   HKEYA     CHECKED ALREADY ?
1A53 27 02             BEQ   *+4       YES,TABLE IS FULL
                *
1A55 20 A9             BRA   SYMA      TRY AGAIN
                *
1A57 CE 0221           LDX   #$0221    LOAD ERROR#
1A5A 20 EF             BRA   SYMB1     PRINT IT & RETURN
                *
                *
                * LOOK UP SYMBOL IN SYMTAB
                * ON ENTRY DESCRA CONTAINS ADDRESS OF SYMBOL
                * AND DESCRC CONTAINS THE LENGTH OF THE
                * SYMBOL.
                * ON RETURN
                *  B=VALUE OF INFO BYTE
                *  B=FF SYMBOL NOT FOUND
                *  X=VALUE OF SYMBOL.
                *
```

```
1A5C 8D 61      LKPSYM BSR   HASH       GET KEY
1A5E FF 06F6 R         STX   SYMPTR     SAVE
1A61 BD 1992 R         JSR   XREF       GENERATE X-REF. BLOCK
1A64 FE 06F6 R         LDX   SYMPTR     RESTORE
                *
1A67 A6 00      LKPSM1 LDA A 0,X
1A69 81 20             CMP A #$20       BLANK?
1A6B 26 03             BNE   LKPSM3     NO
                *
                * ENTRY NOT IN SYMTAB
                *
1A6D C6 FF      LKPSM2 LDA B #$FF       LOAD RC
1A6F 39                RTS              RETURN
                *
                * COMPARE SYMBOL WITH ENTRY IN SYMTAB
                *
1A70 8D 13      LKPSM3 BSR   SYMCMP     COMPARE
1A72 26 08             BNE   LKPSM4     NO MATCH
                *
                * FOUND. EXTRACT INFO. AND VALUE
                *
1A74 FE 06F6 R         LDX   SYMPTR     POINT TO ENTRY
1A77 E6 08             LDA B 8,X        GET INFO BYTE
1A79 EE 06             LDX   6,X        GET VALUE
1A7B 39                RTS
                *
                * PROBE AGAIN FOR SYMBOL IN SYMTAB
                *
1A7C 8D 1B      LKPSM4 BSR   SYMMOD     GET NEXT KEY
1A7E BC 19F2 R         CPX   HKEYA      ALREADY CHECKED?
1A81 26 E4             BNE   LKPSM1     NO,TRY AGAIN
                *
1A83 20 E8             BRA   LKPSM2     ENTRY NOT FOUND
                *
                * ROUTINE TO COMPARE SYMBOL WITH ENTRY
                *
1A85 FF 19E5 R SYMCMP STX    PSTNG1     SAVE PTR TO ENTRY
1A88 86 06             LDA A #6
1A8A B7 19E9 R         STA A PCOUNT     PCOUNT:=L(SYMBOL)
1A8D CE 19EC R         LDX   #HSMBL
1A90 FF 19E7 R         STX   PSTNG2     POINT TO HSMBL
1A93 CE 19E5 R         LDX   #PSTNG1    POINT TO PARMS
1A96 7E 0DD7 R         JMP   COMPAR     COMPARE
                *
                *
                * FIND NEXT SLOT IN SYMTAB
                * SYMPTR:=SYMPTR+9 (MODULO NSYM)
                *
1A99 FF 06F6 R SYMMOD LDX    SYMPTR     GET ALCURRENT SLOT]
1A9C 08                INX              SYMPTR:=SYMPTR+9
1A9D 08                INX
1A9E 08                INX
1A9F 08                INX
1AA0 08                INX
1AA1 08                INX
1AA2 08                INX
1AA3 08                INX
1AA4 08                INX
                *
                * BEYOND SYMTAB ?
                *
1AA5 BC 008C R         CPX   SYMEND
1AA8 26 03             BNE   *+5        NO
1AAA FE 008A R         LDX   SYMTAB     POINT TO FIRST ENTRY
1AAD FF 06F6 R         STX   SYMPTR     SAVE PTR TO ENTRY
1AB0 39                RTS
                *
                * DELETE LAST SYMBOL ENTERED
                *
1AB1 FE 06F6 R DELSYM LDX    SYMPTR
1AB4 86 20             LDA A #$20       LOAD BLANK
1AB6 C6 09             LDA B #9         LOAD ENTRY LENGTH
                *
1AB8 A7 00      DEL1   STA A 0,X        BLANK BYTE
1ABA 08                INX              POINT TO NEXT BYTE
1ABB 5A                DEC B            ALL DONE ?
1ABC 26 FA             BNE   DEL1       NO
                *
1ABE 39                RTS              YES, RETURN
                *
                * HASH SYMBOL TO PRODUCE A KEY
                *
                *
1ABF CE 2020    HASH   LDX   #$2020     BLANK HSMBL
1AC2 FF 19EC R         STX   HSMBL
1AC5 FF 19EE R         STX   HSMBL+2
1AC8 FF 19F0 R         STX   HSMBL+4
                *
                * MOVE SYMBOL TO HSMBL
                *
1ACB CE 19EC R         LDX   #HSMBL     POINT TO HSMBL
1ACE FF 19F8 R         STX   HSAV2      SAVE
1AD1 FE 0642 R         LDX   DESCRA     POINT TO SYMBOL
1AD4 FF 19F6 R         STX   HSAV1      SAVE
1AD7 F6 0644 R         LDA B DESCRC     GET L(SYMBOL)
1ADA C1 07             CMP B #7         TRUNCATE IF NECESSARY
1ADC 2D 02             BLT   HASH1
1ADE C6 06             LDA B #6
                *
1AE0 FF 19F6 R HASH1  LDX   HSAV1       POINT TO SYMBOL
1AE3 A6 00             LDA A 0,X        GET CHAR
1AE5 08                INX
1AE6 FF 19F6 R         STX   HSAV1      POINT TO NEXT CHAR
1AE9 FE 19F8 R         LDX   HSAV2      POINT TO HSYMBL
1AEC A7 00             STA A 0,X        STORE CHAR.
1AEE 08                INX
1AEF FF 19F8 R         STX   HSAV2      POINT TO NEXT CHAR
1AF2 5A                DEC B            ALL DONE?
1AF3 26 EB             BNE   HASH1      NO
                *
                * FOLD OVER HSMBL CREATING KEYA
                *
1AF5 73 22FF R         COM   OVFLG      ALLOW OVERFLOWS
1AF8 FE 19EC R         LDX   HSMBL      HKEYA =HSMBL(2)
1AFB FF 19F2 R         STX   HKEYA
1AFE FE 19EE R         LDX   HSMBL+2
1B01 FF 19F4 R         STX   HKEYB
1B04 CE 19F2 R         LDX   #HKEYA
1B07 BD 234B R         JSR   ADD16            +HSMBL+2(2)
1B0A FE 19F0 R         LDX   HSMBL+4
1B0D FF 19F4 R         STX   HKEYB
1B10 CE 19F2 R         LDX   #HKEYA
```

```
1B13 BD 234B R        JSR   ADD16          +HSMBL+4(2)
                *
                * HKEYA:=REMAINDER OF HKEYA/NSYM
                *
1B16 B6 19F2 R        LDA A HKEYA     LOAD VALUES
1B19 F6 19F3 R        LDA B HKEYA+1
1B1C FE 008E R        LDX   NSYM
1B1F FF 19F4 R        STX   HKEYB
1B22 CE 19F4 R        LDX   #HKEYB    POINT TO NSYM
1B25 BD 2300 R        JSR   DIV16
1B28 FF 19F2 R        STX   HKEYA     SAVE REMAINDER
                *
                * HKEYA:=HKEYA*9
                *
1B2B 4F               CLR A
1B2C C6 09            LDA B #9
1B2E CE 19F2 R        LDX   #HKEYA
1B31 BD 22C3 R        JSR   MPY16
1B34 B7 19F2 R        STA A HKEYA
1B37 F7 19F3 R        STA B HKEYA+1
                *
                * ADD IN BASE ADDRESS OF SYMTAB
                *
1B3A FE 008A R        LDX   SYMTAB
1B3D FF 19F4 R        STX   HKEYB
1B40 CE 19F2 R        LDX   #HKEYA
1B43 BD 234B R        JSR   ADD16
1B46 FE 19F2 R        LDX   HKEYA
1B49 7F 22FF R        CLR   OVFLG
1B4C 39               RTS
                *
                *
                *
                * LOOK UP MNEMONIC IN MNTAB
                * ON ENTRY DESCRA POINTS TO MNEMONIC, AND
                * DESCRC CONTAINS THE LENGTH
                * ON RETURN
                *  REG A = 00  FOUND
                *  REG A = FF  NOT IN TABLE
                *  REG X = ADDRESS OF ROUTINE TO PROCESS
                *          THE OPCODE/PSEUDOP
                *  REG B = MACHINE CODE FOR OPCODES
                *        = FF FOR PSEUDOPS
                *        = F0 FOR COMPILER OPERATIONS
                *
                * THE ALGORITHM IS A BINARY SEARCH
                *
                * TEMPORARY LOCATIONS
                *
1B4D 0001       LP    RMB   1   ONE BELOW LOWEST ENTRY
1B4E 0001       MP    RMB   1   ONE HIGHER THAN HIGHEST ENTRY
1B4F 0001       IP    RMB   1   CALCULATED PROBE VALUE
1B50 0007       ENSIZ EQU   7   LENGTH OF ENTRY IN MNTAB
                *
1B52 BD 1ABF R MNLKP  JSR   HASH      FORMAT NAME
1B55 73 22FF R        COM   OVFLG     ALLOW OVERFLOW
1B58 86 04            LDA A #4
1B5A B7 19E9 R        STA A PCOUNT    INIT PCOUNT
1B5D 86 AA            LDA A #CHRTAB-MNTAB/7+1  (# ENTRIES+1)
1B5F B7 1B4E R        STA A MP        INIT MP
1B62 4F               CLR A
1B63 B7 1B4D R        STA A LP        INIT LP
                *
1B66 B6 1B4D R MNLKPA LDA A LP
1B69 4C               INC A           A:=LP+1
1B6A B1 1B4E R        CMP A MP        MP=LP+1  ?
1B6D 26 06            BNE   MNLKPB    NO
                *
1B6F 7F 22FF R        CLR   OVFLG
1B72 86 FF            LDA A #$FF      YES,ENTRY NOT IN TABLE
1B74 39               RTS
                *
                * IP:=( LP+MP )/2 TRUNCATED
                *
1B75 F6 1B4D R MNLKPB LDA B LP
1B78 FB 1B4E R        ADD B MP        B:=LP+MP
1B7B 56               ROR B           B:=B/2
1B7C F7 1B4F R        STA B IP        SAVE IP
                *
                * GET 16-BIT ADDRESS OF ENTRY
                *
1B7F 4F               CLR A
1B80 CE 1B50 R        LDX   #ENSIZ    GET ENTRY LENGTH
1B83 5A               DEC B           B:=IP-1
1B84 BD 22C3 R        JSR   MPY16     GET (IP-1)*7
1B87 B7 19E5 R        STA A PSTNG1    SAVE
1B8A F7 19E6 R        STA B PSTNG1+1
1B8D CE 0090 R        LDX   #MNTAB
1B90 FF 19E7 R        STX   PSTNG2    PSTNG2:=BASE OF MNTAB
1B93 CE 19E5 R        LDX   #PSTNG1   POINT TO PARMS
1B96 BD 234B R        JSR   ADD16     PSTNG1:=(IP-1)*7+MNTAB
1B99 FE 19E5 R        LDX   PSTNG1
1B9C FF 19EA R        STX   TBADD     SAVE
                *
                * COMPARE MNEMONIC WITH ENTRY IN MNTAB
                *
1B9F CE 19EC R        LDX   #HSMBL    GET MNEMONIC ADDRESS
1BA2 FF 19E7 R        STX   PSTNG2    INIT PARM FOR COMPARE
1BA5 CE 19E5 R        LDX   #PSTNG1   POINT TO PARMS
1BA8 BD 0DD7 R        JSR   COMPAR    COMPARE
1BAB 25 0E            BCS   MNLI      ENTRY<MNEMONIC
1BAD 26 15            BNE   MNMI      ENTRY>MNEMONIC
                *
1BAF 4F               CLR A           ENTRY FOUND
1BB0 FE 19EA R        LDX   TBADD     POINT TO ENTRY
1BB3 E6 06            LDA B 6,X       GET MC
1BB5 EE 04            LDX   4,X       GET BRANCH ADDRESS
1BB7 7F 22FF R        CLR   OVFLG
1BBA 39               RTS
                *
                * ENTRY<MNEMONIC LP:=IP
                *
1BBB B6 1B4F R MNLI   LDA A IP
1BBE B7 1B4D R        STA A LP
1BC1 7E 1B66 R        JMP   MNLKPA    TRY AGAIN
                *
                * ENTRY>MNEMONIC MP:=IP
                *
1BC4 B6 1B4F R MNMI   LDA A IP
1BC7 B7 1B4E R        STA A MP
1BCA 7E 1B66 R        JMP   MNLKPA    TRY AGAIN
                *
```

```
                        *
                        * EVALUATE NUMBERS,SYMBOLS AND EXPRESSIONS
                        * "EVALS" FOR COMPILER
                        * "NSEVL" FOR ASSEMBLER
                        *
1BCD 0002               RUNCNT  RMB 2           POINTER TO RUN-TIME STACK
1BCF 0002               DMSAV   RMB 2           SAVE DIMENSION DATA
1BD1 0001               DMDATA  RMB 1           FLAG DATA
1BD2 0001               DIMFLG  RMB 1           DIMENSION FLAG
1BD3 0001               STKCNT  RMB 1           COUNT OF OPSTACK
1BD4 0002               STKTOP  RMB 2           OPSTACK POINTER
1BD6 0002               STPSTK  RMB 2           TOP-OF-STACK POINTER
1BD8 0001               UNFLG   RMB 1           UNARY OPERATOR FLAG
1BD9 0028               OPSTAK  RMB 40          OPERATOR STACK (20 LOCATIONS)
1C01 0002               AC      RMB 2
1C03 0002               NUMBER  RMB 2           BOTTOM OF STACK
1C05 0002               AX      RMB 2
                        *
                        * ROUTINE TO CLEAR "NUMBER" (2 BYTES)
1C07 8D 07              UPSCLR  BSR STAKUP      ROLL STACK UP
1C09 7F 1C03 R                  CLR NUMBER
1C0C 7F 1C04 R                  CLR NUMBER+1
1C0F 39                         RTS
                        *
                        * ROUTINE TO ROLL OPERAND STACK UP (2 BYTES)
1C10 CE 1BD9 R          STAKUP  LDX #OPSTAK
1C13 E6 02              STAKU1  LDA B 2,X
1C15 E7 00                      STA B 0,X
1C17 08                         INX
1C18 8C 1C03 R                  CPX #NUMBER     DONE?
1C1B 26 F6                      BNE STAKU1
                        *
1C1D B6 1BD3 R                  LDA A STKCNT    CHECK STACK OVERFLOW
1C20 81 14                      CMP A #20
1C22 2E 1A                      BGT EXPERR
                        *
1C24 7C 1BD3 R                  INC STKCNT      ONE MORE ELEMENT
1C27 39                         RTS
                        *
                        * ROUTINE TO ROLL OPERAND STACK DOWN (2 BYTES)
1C28 CE 1C04 R          STAKDN  LDX #AX-1
1C2B E6 00              STAKD1  LDA B 0,X
1C2D E7 02                      STA B 2,X
1C2F 09                         DEX
1C30 8C 1BD8 R                  CPX #OPSTAK-1
1C33 26 F6                      BNE STAKD1
                        *
1C35 7A 1BD3 R                  DEC STKCNT      ONE LESS ELEMENT
1C38 39                         RTS
                        *
                        *
                        * BEGIN COMPILER EXPRESSION EVALUATOR
                        *
                        *
                        *ERROR RETURNS HERE. PRINT MESSAGE AND QUIT
1C39 CE 0333            MSMTCH  LDX #$0333      MISMATCH ERROR
1C3C 20 03                      BRA *+5
                        *
1C3E CE 0300            EXPERR  LDX #$0300      EVAL  ERROR
1C41 BE 1BD4 R                  LDS STKTOP      RESTORE STACK
1C44 7E 24E6 R                  JMP PRINTE
                        *
1C47 BF 1BD4 R          EVALS   STS STKTOP      SAVE STACK POINTER
1C4A 8E 20D4 R                  LDS #FXSTK
1C4D BF 1BD6 R                  STS STPSTK      SAVE POINTER
1C50 7F 1BD3 R                  CLR STKCNT      CLEAR RUN-TIME STACK
1C53 7F 1BCD R                  CLR RUNCNT
1C56 7F 1BCF R                  CLR RUNCNT+1
1C59 7F 139A R                  CLR GAFLG       NOT A GET-ADDRESS
1C5C 7F 0702 R                  CLR PARFLG      NOT A PARAMETER
1C5F 7F 1BD2 R                  CLR DIMFLG      NOT AN INDEX
                        *
1C62 7D 0707 R          EXPR1   TST NXTFLG      GET NEXT TOKEN?
1C65 26 03                      BNE *+5         NO
                        *
1C67 BD 172F R                  JSR NXTOK       YES
                        *
1C6A 7F 24CC R                  CLR FRNUM
1C6D 7F 0707 R                  CLR NXTFLG
1C70 7F 1BD8 R                  CLR UNFLG
1C73 C1 2B                      CMP B #'+       UNARY PLUS?
1C75 27 1E                      BEQ EXPR2       YES
                        *
1C77 C1 2D                      CMP B #'-       UNARY MINUS?
1C79 26 20                      BNE EXPR2A      NO
                        *
1C7B 7D 0578 R          UNEXP   TST FLPFLG      IN FLOATING MODE?
1C7E 27 05                      BEQ UNEXP2      NO
                        *
1C80 C6 47                      LDA B #$47      STACK "FNEG" CALL
1C82 37                         PSH B
1C83 20 10                      BRA EXPR2
                        *
1C85 37                 UNEXP2  PSH B           STACK OPERATOR
1C86 CE 3691 R                  LDX #COMPL2     OUTPUT EXTRA "ZERO"
1C89 BD 13AB R                  JSR STR1        TREAT LIKE CONSTANT
1C8C BD 1C07 R                  JSR UPSCLR      TAKE CARE OF STACK
1C8F 73 1BD8 R                  COM UNFLG       SET UNARY FLAG
1C92 7E 1D34 R                  JMP EVAL7D      FINISH UP
                        *
1C95 BD 172F R          EXPR2   JSR NXTOK       GET NEXT TOKEN
1C98 7F 1BD8 R                  CLR UNFLG
1C9B 5D                 EXPR2A  TST B           BAD TOKEN?
1C9C 27 A0                      BEQ EXPERR      YES
                        *
1C9E 7F 0702 R                  CLR PARFLG      NOT PARAMETER
1CA1 C1 2C                      CMP B #',       TERMINATOR (,)?
1CA3 27 43                      BEQ EVAL4       YES
1CA5 C1 3D                      CMP B #'=       TERMINATOR (=)?
1CA7 27 3F                      BEQ EVAL4       YES
1CA9 C1 0D                      CMP B #$0D      TERMINATOR (CR)?
1CAB 27 3B                      BEQ EVAL4       YES
1CAD C1 5D                      CMP B #']       TERMINATOR (])?
1CAF 27 37                      BEQ EVAL4       YES
                        *
1CB1 C1 01                      CMP B #1        NAME?
1CB3 26 03                      BNE *+5
1CB5 7E 1E0B R                  JMP EVAL8       YES
                        *
1CB8 81 02                      CMP A #2        NUMBER?
1CBA 27 42                      BEQ EVAL7       YES (HEX OR DECIMAL)
                        *
```

```
1CBC C1 29                CMP B #')       RIGHT PAREN?
1CBE 27 12                BEQ EVAL3       YES
                  *
1CC0 81 24                CMP A #$24      ARITHMETIC?
1CC2 27 0B                BEQ EVAL2A      YES
                  *
1CC4 C1 28                CMP B #'(       LEFT PAREN?
1CC6 26 04                BNE EVAL2       NO, THEN ERROR
                  *
1CC8 37                   PSH B           STACK IT
1CC9 7E 1C62 R            JMP EXPR1       CONTINUE
                  *
1CCC 7E 1C3E R    EVAL2   JMP EXPERR      NONE OF THESE, ERROR
                  *
1CCF 37           EVAL2A  PSH B           PUT ON STACK
1CD0 20 C3                BRA EXPR2       CONTINUE
                  *
1CD2 30           EVAL3   TSX             GET STACK POINTER
1CD3 09                   DEX             ADJUST
1CD4 F6 1BD2 R            LDA B DIMFLG    DIMENSION CHECK
1CD7 BC 1BD6 R            CPX STPSTK      STACK EMPTY?
1CDA 27 09                BEQ EVAL3A      YES
                  *
1CDC 32                   PUL A
1CDD 5F                   CLR B
1CDE 81 28                CMP A #'(       LEFT PAREN ON STACK?
1CE0 26 03                BNE EVAL3A      NO
                  *
1CE2 7E 1FAB R            JMP EVAL12      SKIP LEFT PAREN
                  *
1CE5 5D           EVAL3A  TST B           DIMENSIONED VARIABLE?
1CE6 27 E4                BEQ EVAL2       IF NOT, ERROR
                  *
1CE8 F6 1BD3 R    EVAL4   LDA B STKCNT
1CEB 5A                   DEC B           GET STACK COUNT
1CEC 26 DE                BNE EVAL2       NOT EMPTY, ERROR
                  *
1CEE 30           EVAL5   TSX             GET STACK POINTER
1CEF 09                   DEX             ADJUST
1CF0 BC 1BD6 R            CPX STPSTK      OPSTAK EMPTY?
1CF3 26 D7                BNE EVAL2       NOT EMPTY, ERROR
                  *
1CF5 7D 1BD2 R            TST DIMFLG      IN AN ARRAY INDEX?
1CF8 26 03                BNE *+5         IF SO, RETURN
                  *
1CFA BE 1BD4 R            LDS STKTOP      IF NOT,RESTORE STACK
1CFD 39                   RTS             DONE!
                  *
                  * HANDLE CONSTANT HERE, HEX OR DECIMAL
1CFE C1 03        EVAL7   CMP B #3        HEX STRING?
1D00 26 0C                BNE EVAL7A      NO
                  *
1D02 F6 0644 R            LDA B DESCRC    GET TOKEN COUNT
1D05 C1 04                CMP B #4        TOO LONG?
1D07 2E C3                BGT EVAL2       YES, THEN ERROR
                  *
1D09 BD 2214 R            JSR CVHB        CONVERT TO BINARY
1D0C 20 0E                BRA EVAL7B      FINISH UP
                  *
1D0E C1 09        EVAL7A  CMP B #9        DECIMAL STRING?
1D10 26 6F                BNE EVAL7F      NO
```

```
                  *
1D12 F6 0644 R            LDA B DESCRC    GET TOKEN COUNT
1D15 C1 05                CMP B #5        TOO LONG?
1D17 2E B3                BGT EVAL2       YES, THEN ERROR
                  *
1D19 BD 226C R            JSR CVDB        CONVERT TO BINARY
                  *
1D1C FF 20E5 R    EVAL7B  STX TMPVAL      SAVE CONVERTED VALUE
1D1F BD 1C07 R            JSR UPSCLR      ROLL OPSTACK UP
                  *
                  * GENERATE CODE FOR CONSTANT
                  *       LDX #TMPVAL
                  *       STX ADDRESS
                  *
1D22 B6 20E5 R            LDA A TMPVAL    SET UP FOR CVBTD
1D25 F6 20E6 R            LDA B TMPVAL+1
1D28 CE 1D70 R            LDX #CLN1+6
1D2B BD 2487 R            JSR CVBTD
1D2E CE 1D6A R            LDX #CLN1       OUTPUT LINE 1
1D31 BD 13AB R            JSR STR1
                  *
1D34 B6 1BCD R    EVAL7D  LDA A RUNCNT    SET UP FOR CVBTD
1D37 F6 1BCE R            LDA B RUNCNT+1
1D3A CE 129B R            LDX #FCLN4+12
1D3D BD 2487 R            JSR CVBTD
1D40 CE 128F R            LDX #FCLN4      OUTPUT LINE 2
1D43 BD 0DAF R            JSR OUTCMP
1D46 FE 1BCD R    EVAL7E  LDX RUNCNT      BUMP POINTER
1D49 FF 1C03 R            STX NUMBER      SAVE ADDRESS IN COMP  STACK
1D4C 08                   INX
1D4D 08                   INX
1D4E 7D 0578 R            TST FLPFLG      IN FLOATING MODE?
1D51 27 07                BEQ *+9         NO
                  *
1D53 08                   INX
1D54 08                   INX
1D55 08                   INX
1D56 08                   INX
1D57 08                   INX
1D58 08                   INX             9 BYTES IN FLOAT-POINT
1D59 08                   INX
                  *
1D5A FF 1BCD R            STX RUNCNT
1D5D F6 1BCE R            LDA B RUNCNT+1
1D60 C1 50                CMP B #80       CHECK FOR STACK OVERFLOW
1D62 2F 03                BLE *+5
1D64 7F 1C3E R            JMP EXPERR      RUNTIME OVERFLOW ERROR
                  *
1D67 7E 1FAB R            JMP EVAL12      CONTINUE
                  *
1D6A 20           CLN1    FCC ' LDX #'
1D70 0005                 RMB 5
1D75 0D                   FCB $0D
1D76 20           CLN2    FCC ' STX '
1D78 0005                 RMB 5
1D80 0D                   FCB $0D
                  *
                  *
                  * HANDLE FLOATING CONSTANT HERE
                  *
1D81 C1 0F        EVAL7F  CMP B #$0F      FLOATING CONSTANT?
```

```
1D83 27 03                BEQ *+5
1D85 7E 1CCC R            JMP EVAL2       IF NOT, ERROR
                *
                * FORM CODE AS FOLLOWS:
                *         LDX #ADDRESS
                *         STX ACCUM1
                *         LDX #*+9
                *         JSR MOVER
                *         BRA *+12
                *         FDB $----
                *         FDB $----
                *         FDB $----
                *         FDB $----
                *         FDB $----
                *
1D88 B6 1BCD R            LDA A RUNCNT
1D8B F6 1BCE R            LDA B RUNCNT+1
1D8E CE 1264 R            LDX #FCLN1+13
1D91 BD 2487 R            JSR CVBTD
1D94 CE 1257 R            LDX #FCLN1
1D97 BD 13AB R            JSR STR1        OUTPUT LINE 1
1D9A CE 12EE R            LDX #UNOPL2
1D9D BD 13AB R            JSR STR1        OUTPUT LINE 2
1DA0 CE 2960 R            LDX #@L1
1DA3 BD 13AB R            JSR STR1        OUTPUT LINE 3
1DA6 C6 A5                LDA B #165
1DA8 BD 139C R            JSR RTBL        OUTPUT LINE 4
1DAB CE 1DD0 R            LDX #FCL2
1DAE BD 13AB R            JSR STR1        OUTPUT LINE 5
1DB1 CE 1988 R            LDX #FTEMP
1DB4 8D 2F                BSR FCOUT       OUTPUT LINE 6
1DB6 CE 198A R            LDX #FTEMP+2
1DB9 8D 2A                BSR FCOUT       OUTPUT LINE 7
1DBB CE 198C R            LDX #FTEMP+4
1DBE 8D 25                BSR FCOUT       OUTPUT LINE 8
1DC0 CE 198E R            LDX #FTEMP+6
1DC3 8D 20                BSR FCOUT       OUTPUT LINE 9
1DC5 CE 1990 R            LDX #FTEMP+8
1DC8 8D 1B                BSR FCOUT       OUTPUT LINE 10
1DCA BD 1C07 R            JSR UPSCLR      INSERT STACK ENTRY
1DCD 7E 1D46 R            JMP EVAL7E      FINISH UP
                *
1DD0 20         FCL2      FCC ' BRA *+12'
1DD9 0D                   FCB $0D
1DDA 20         FCL3      FCC ' FDB $'
1DE0 0004                 RMB 4
1DE4 0D                   FCB $0D
                *
1DE5 A6 00      FCOUT     LDA A 0,X       FIRST BYTE
1DE7 BD 3E75 R            JSR OUTHL       CONVERT LEFT NIBBLE
1DEA B7 1DE0 R            STA A FCL3+6
1DED A6 00                LDA A 0,X
1DEF BD 3E79 R            JSR OUTHR       CONVERT RIGHT NIBBLE
1DF2 B7 1DE1 R            STA A FCL3+7
1DF5 A6 01                LDA A 1,X       SECOND BYTE
1DF7 BD 3E75 R            JSR OUTHL       CONVERT LEFT NIBBLE
1DFA B7 1DE2 R            STA A FCL3+8
1DFD A6 01                LDA A 1,X
1DFF BD 3E79 R            JSR OUTHR       CONVERT RIGHT NIBBLE
1E02 B7 1DE3 R            STA A FCL3+9
1E05 CE 1DDA R            LDX #FCL3
```

```
1E08 7E 13AB R            JMP STR1        WRITE LINE
                *
                *
                * HANDLE NAMES HERE, VARIABLE OR FUNCTION
                *
1E0B BD 1B52 R EVAL8      JSR MNLKP       CHECK MNEMONIC TABLE
1E0E 4D                   TST A           FOUND?
1E0F 26 5E                BNE EVAL9       NO
                *
1E11 C1 F0                CMP B #$F0      A COMPILER OPCODE?
1E13 26 03                BNE *+5
1E15 7E 1CF8 R            JMP EVAL4       IF SO, QUIT
                *
1E18 FE 0642 R            LDX DESCRA      CHECK FOR FUNCTION
1E1B A6 00                LDA A 0,X       FUNCTION HAS '.' AS FIRST CHAR
1E1D 81 2E                CMP A #'.       PERIOD?
1E1F 26 4E                BNE EVAL9       NO
                *
                *
                * CHECK FOR APPROPRIATE FLOAT MODE
1E21 7D 0578 R            TST FLPFLG      IN FLOAT MODE?
1E24 26 17                BNE EVAL8A      YES
                * IN INTEGER MODE
1E26 C5 80                BIT B #$80
1E28 27 03                BEQ *+5         INTEGER FUNCTION
1E2A 7E 1C39 R            JMP MSMTCH      MISMATCH ERROR
                *
1E2D C1 39      EOFPR     CMP B #$39      "EOF" OR "ERR" OP?
1E2F 26 03                BNE *+5         NO
                *
1E31 7E 1F89 R            JMP EVAL9E      YES, TREAT LIKE NAME
                *
1E34 C1 36                CMP B #$36      "FIX" OP?
1E36 26 33                BNE EVAL8C      NO
                *
1E38 F7 0578 R            STA B FLPFLG    SETUP FOR FIX
1E3B 20 2E                BRA EVAL8C
                *
1E3D C5 80      EVAL8A    BIT B #$80      FLOAT OP?
1E3F 27 0B                BEQ EVAL8B      IF NOT, CHECK SPEC  CASES
                *
1E41 C4 7F                AND B #$7F      STRIP HIGH BIT
                * FLOATING MODE
1E43 C1 37                CMP B #$37      "FLOAT" OP?
1E45 26 24                BNE EVAL8C      NO
                *
1E47 7F 0578 R            CLR FLPFLG      SETUP FOR FLOAT
1E4A 20 1F                BRA EVAL8C
                *
1E4C C5 40      EVAL8B    BIT B #$40      COMPARE OP?
1E4E 26 1B                BNE EVAL8C      YES, O.K
                *
1E50 C1 2C                CMP B #$2C      "EXP"?
1E52 27 17                BEQ EVAL8C
                *
1E54 C1 3A                CMP B #$3A      "NOT"?
1E56 27 13                BEQ EVAL8C
                *
1E58 C1 3B                CMP B #$3B      "ABS"?
1E5A 27 0F                BEQ EVAL8C
                *
```

```
1E5C C1 3D              CMP B #$3D      "SIGN"?
1E5E 27 0B              BEQ EVAL8C
                  *
1E60 C1 3E              CMP B #$3E      "OR"?
1E62 27 07              BEQ EVAL8C
                  *
1E64 C1 3F              CMP B #$3F      "AND"?
1E66 27 03              BEQ EVAL8C
                  *
1E68 7E 1C39 R          JMP MSMTCH      OTHERWISE, MISMATCH ERROR
                  *
1E6B 37           EVAL8C PSH B          STACK FUNCTION CODE
1E6C 7E 1C62 R          JMP EXPR1
                  *
1E6F BD 2A4E R    EVAL9  JSR DEFS       DEFINE SYMBOL
1E72 FF 1BCF R          STX DMSAV       SAVE DIM. VALUES
1E75 F7 1BD1 R          STA B DMDATA
1E78 C5 80              BIT B #$80      PARAMETER?
1E7A 27 03              BEQ *+5         NO
                  *
1E7C 7C 0702 R          INC PARFLG      YES, SET FLAG
                  *
1E7F C5 03              BIT B #3        DIMENSIONED?
1E81 26 03              BNE *+5
1E83 7E 1F33 R          JMP EVAL9C      NO
                  *
1E86 C4 40              AND B #$40      TEST STRING BIT
1E88 F7 25C6 R          STA B DSTFLG    SET STRING FLAG
1E8B B6 1BD2 R          LDA A DIMFLG    SAVE PARAMETERS
1E8E F6 1BD3 R          LDA B STKCNT
1E91 36                 PSH A
1E92 37                 PSH B
1E93 B6 1BCD R          LDA A RUNCNT
1E96 F6 1BCE R          LDA B RUNCNT+1
1E99 36                 PSH A
1E9A 37                 PSH B
1E9B B6 1BD6 R          LDA A STPSTK
1E9E F6 1BD7 R          LDA B STPSTK+1
1EA1 36                 PSH A
1EA2 37                 PSH B
1EA3 7F 1BD3 R          CLR STKCNT
                  *
                  * CHECK FOR FLOATING VARIABLE NAME
1EA6 F6 1BD1 R          LDA B DMDATA
1EA9 7D 0578 R          TST FLPFLG      IN FLOATING MODE?
1EAC 27 4E              BEQ EVAL9A      INTEGER
                  *
1EAE C5 08              BIT B #$08      TEST VARIABLE
1EB0 26 03              BNE *+5         FLOATING NAME
1EB2 7E 1C39 R          JMP MSMTCH      MISMATCH ERROR
                  *
                  * GENERATE CODE FOR FLOATING ARRAY
                  * GET ADDRESS OF ENTRY IN X-REGISTER
                  *       STX  ACCUM2
                  *       LDX  #ADDRESS
                  *       STX  ACCUM1
                  *       LDX  ACCUM2
                  *       JSR  MOVER
                  *
1EB5 BD 3C7C R          JSR FNDDIM      GET ADDRESS
1EB8 CE 1639 R          LDX #STRP6
1EBB BD 13AB R          JSR STR1        OUTPUT LINE 1
1EBE FE 1C03 R          LDX NUMBER      SAVE VALUE
1EC1 FF 0E03 R          STX XSAV
1EC4 FE 20E5 R          LDX TMPVAL      GET STACK LEVEL
1EC7 FF 1C03 R          STX NUMBER
1ECA BD 123F R          JSR FUNOP2      OUTPUT LINES 2,3
1ECD CE 1647 R          LDX #STRP7
1ED0 BD 13AB R          JSR STR1        OUTPUT LINE 4
1ED3 C6 A5              LDA B #165
1ED5 BD 139C R          JSR RTBL        OUTPUT LINE 5
1ED8 33                 PUL B
1ED9 32                 PUL A
1EDA B7 1BD6 R          STA A STPSTK    RESTORE VALUES
1EDD F7 1BD7 R          STA B STPSTK+1
1EE0 33                 PUL B
1EE1 32                 PUL A
1EE2 B7 1BCD R          STA A RUNCNT
1EE5 F7 1BCE R          STA B RUNCNT+1
1EE8 33                 PUL B
1EE9 32                 PUL A
1EEA B7 1BD2 R          STA A DIMFLG
1EED F7 1BD3 R          STA B STKCNT
1EF0 FE 0E03 R          LDX XSAV        RESTORE VALUE
1EF3 FF 1C03 R          STX NUMBER
1EF6 BD 1C07 R          JSR UPSCLR      ROLL OPSTACK UP
1EF9 7E 1D46 R          JMP EVAL7E
                  *
1EFC C5 08        EVAL9A BIT B #$08     TEST VARIABLE
1EFE 27 03              BEQ *+5         INTEGER NAME
1F00 7E 1C39 R          JMP MSMTCH      MISMATCH ERROR
                  *
1F03 BD 3C7C R          JSR FNDDIM
                  *
                  * ADDRESS OF ENTRY NOW IN X-REGISTER
                  * GENERATE CODE FOR INTEGERS
                  *       LDX 0,X
                  *       STX ADDRESS
                  *
1F06 CE 1F2A R          LDX #DIML1
1F09 BD 13AB R          JSR STR1        OUTPUT LINE 1
1F0C 33           EVAL9B PUL B          RESTORE PARAMETERS
1F0D 32                 PUL A
1F0E F7 1BD7 R          STA B STPSTK+1
1F11 B7 1BD6 R          STA A STPSTK
1F14 33                 PUL B
1F15 32                 PUL A
1F16 F7 1BCE R          STA B RUNCNT+1
1F19 B7 1BCD R          STA A RUNCNT
1F1C 33                 PUL B
1F1D 32                 PUL A
1F1E F7 1BD3 R          STA B STKCNT
1F21 B7 1BD2 R          STA A DIMFLG
1F24 BD 1C07 R          JSR UPSCLR      ROLL OPSTACK UP
1F27 7E 1D34 R          JMP EVAL7D      FINISH UP
                  *
1F2A 20           DIML1 FCC ' LDX 0,X'
1F32 0D                 FCB $0D
                  *
                  *GENERATE CODE FOR INTEGER VARIABLE NAME
                  *       LDX NAME
                  *       STX ADDRESS
```

```
                        *
                        *
                        * CHECK FOR FLOATING VARIABLE NAME
1F33 7D 0578 R EVAL9C   TST FLPFLG      IN FLOATING MODE?
1F36 26 07              BNE EVAL9D      YES
                        *
1F38 C5 08              BIT B #$08      TEST NAME
1F3A 27 4D              BEQ EVAL9E      INTEGER NAME
1F3C 7E 1C39 R          JMP MSMTCH      MISMATCH ERROR
                        *
                        *
1F3F C5 08      EVAL9D  BIT B #$08      TEST NAME
1F41 26 03              BNE *+5         FLOATING NAME
1F43 7E 1C39 R          JMP MSMTCH      MISMATCH ERROR
                        *
                        *
                        * GENERATE CODE FOR FLOATING VARIABLE NAME
                        *       LDX #ADDRESS
                        *       STX ACCUM1
                        *       LDX #NAME
                        *       JSR MOVER
                        *
1F46 B6 1BCD R          LDA A RUNCNT
1F49 F6 1BCE R          LDA B RUNCNT+1
1F4C CE 1264 R          LDX #FCLN1+13
1F4F BD 2487 R          JSR CVBTD
1F52 CE 1257 R          LDX #FCLN1
1F55 BD 13AB R          JSR STR1        OUTPUT LINE 1
1F58 CE 12EE R          LDX #UNOPL2
1F5B BD 13AB R          JSR STR1        OUTPUT LINE 2
1F5E CE 1D6A R          LDX #CLN1
1F61 86 05              LDA A #5
1F63 B7 06F0 R          STA A CMPCNT
1F66 BD 0DAF R          JSR OUTCMP      "LDX"
1F69 7D 0702 R          TST PARFLG      PARAMETER?
1F6C 26 0A              BNE EVAL10      YES, SKIP "#"
                        *
1F6E 86 23              LDA A #'#
1F70 BD 0D98 R          JSR WRTCHR      "#"
1F73 C6 23              LDA B #'#
1F75 BD 0DA3 R          JSR TPCHAR
                        *
1F78 BD 0DA6 R EVAL10   JSR OUTNAM      "NAME"
1F7B BD 0DD3 R          JSR OUTCR
1F7E C6 A5              LDA B #165
1F80 BD 139C R          JSR RTBL        OUTPUT LINE 4
1F83 BD 1C07 R          JSR UPSCLR      ROLL OPSTACK UP
1F86 7E 1D46 R          JMP EVAL7E      FINISH UP
                        *
1F89 CE 1D6A R EVAL9E   LDX #CLN1
1F8C 86 05              LDA A #5
1F8E B7 06F0 R          STA A CMPCNT
1F91 BD 0DAF R          JSR OUTCMP      OUTPUT "LDX"
1F94 BD 0DA6 R          JSR OUTNAM      OUTPUT NAME
1F97 BD 0DD3 R          JSR OUTCR
1F9A 7D 0702 R          TST PARFLG      PARAMETER?
1F9D 27 06              BEQ EVAL11      NO
                        *
1F9F CE 1F2A R          LDX #DIMLI      OUTPUT 'LDX 0,X'
1FA2 BD 13AB R          JSR STR1
                        *
1FA5 BD 1C07 R EVAL11   JSR UPSCLR      ROLL OPSTACK UP
1FA8 7E 1D34 R          JMP EVAL7D      FINISH UP LIKE NUMBER
                        *
1FAB 30         EVAL12  TSX             GET STACK POINTER
1FAC 09                 DEX             ADJUST
1FAD BC 1BD6 R          CPX STPSTK      EMPTY?
1FB0 26 13              BNE EVAL13      NO
                        *
1FB2 FE 0642 R          LDX DESCRA
1FB5 A6 00              LDA A 0,X
1FB7 81 0D              CMP A #$0D      AT A CR?
1FB9 27 07              BEQ *+9         YES
                        *
1FBB 81 2C              CMP A #',       AT A COMMA?
1FBD 27 03              BEQ *+5         YES
                        *
1FBF 7E 1C95 R          JMP EXPR2       CONTINUE
                        *
1FC2 7E 1CE8 R          JMP EVAL4       FINISH UP
                        *
1FC5 32         EVAL13  PUL A           GET TOP OF STACK
1FC6 36                 PSH A
1FC7 7D 1BD8 R          TST UNFLG       UNARY OPERATION?
1FCA 27 03              BEQ *+5
1FCC 7E 1C95 R          JMP EXPR2       YES, CONTINUE
                        *
1FCF 81 28              CMP A #'(       LEFT PAREN ON STACK?
1FD1 26 03              BNE *+5         NO
                        *
1FD3 7E 1C95 R          JMP EXPR2       YES, CONTINUE
                        *
1FD6 32                 PUL A           REMOVE OP FROM STACK
1FD7 80 2A              SUB A #$2A      REMOVE BIAS
1FD9 2C 03              BGE *+5         O.K.
                        *
1FDB 7E 1C3E R          JMP EXPERR      ERROR
                        *
1FDE 81 1D              CMP A #29
1FE0 2E F9              BGT *-5         NOT FUNCTION
                        *
1FE2 CE 1FFA R          LDX #OPTBL
1FE5 7D 0578 R          TST FLPFLG      FLOATING MODE?
1FE8 27 03              BEQ *+5
                        *
1FEA CE 2034 R          LDX #OPTBL2     USE FLOATING TABLE
                        *
1FED 48                 ASL A           2 BYTES IN TABLE ENTRY
1FEE 27 04              BEQ EVAL15
                        *
1FF0 08         EVAL14  INX
1FF1 4A                 DEC A           ADD "A" TO "OPTBL"
1FF2 26 FC              BNE EVAL14
                        *
1FF4 EE 00      EVAL15  LDX 0,X         INDIRECT ADDRESS
1FF6 AD 00              JSR 0,X         PROCESS OPERATION
1FF8 20 B1              BRA EVAL12      CONTINUE
                        *
                        *
                        * INTEGER FUNCTION TABLE
1FFA 132E       R OPTBL FDB MULT
1FFC 1337       R       FDB ADDZ
```

```
1FFE 134F   R          FDB EXPR
2000 133B   R          FDB SUBTR
2002 1353   R          FDB MODPR
2004 133F   R          FDB DIV
2006 1C3E   R          FDB EXPERR        UNIMPLEMENTED OPERATIONS
2008 1C3E   R          FDB EXPERR
200A 1C3E   R          FDB EXPERR
200C 1C3E   R          FDB EXPERR
200E 1C3E   R          FDB EXPERR
2010 1C3E   R          FDB EXPERR
2012 1C3E   R          FDB EXPERR
2014 1357   R          FDB FLOTPR
2016 11E7   R          FDB PEEKPR
2018 1E2D   R          FDB EOFPR
201A 11D3   R          FDB NOTPR
201C 11DB   R          FDB ABSPR
201E 11DF   R          FDB RANPR
2020 11E3   R          FDB SIGPR
2022 1343   R          FDB ORPR
2024 1347   R          FDB ANDPR
2026 134B   R          FDB EXOPR
2028 35F9   R          FDB EQPR
202A 360D   R          FDB GEPR
202C 361B   R          FDB GTPR
202E 3629   R          FDB LEPR
2030 3637   R          FDB LTPR
2032 3645   R          FDB NEPR
                  *
                  *
                  * FLOATING FUNCTION TABLE
2034 13DE   R OPTBL2   FDB FMUL
2036 13C9   R          FDB FADD
2038 11FF   R          FDB FEXP
203A 13DA   R          FDB FSUB
203C 1222   R          FDB TENPR
203E 13E2   R          FDB FDIV
2040 11EF   R          FDB ACOSPR
2042 11F3   R          FDB ASINPR
2044 11F7   R          FDB ATANPR
2046 11FB   R          FDB COSPR
2048 121A   R          FDB SINPR
204A 121E   R          FDB TANPR
204C 1203   R          FDB FIXPR
204E 1C3E   R          FDB EXPERR
2050 120A   R          FDB LOGPR
2052 120E   R          FDB NATPR
2054 11D3   R          FDB NOTPR
2056 11E8   R          FDB FABS
2058 1377   R          FDB XTOYPR
205A 1216   R          FDB FSIGN
205C 1343   R          FDB ORPR
205E 1347   R          FDB ANDPR
2060 134B   R          FDB EXOPR
2062 35F9   R          FDB EQPR
2064 360D   R          FDB GEPR
2066 361B   R          FDB GTPR
2068 3629   R          FDB LEPR
206A 3637   R          FDB LTPR
206C 3645   R          FDB NEPR
206E 1212   R          FDB FNEG
                  *
```

```
                  *
2070 0064                 RMB 100
2004 0001         EXSTK   RMB 1          EVALUATION STACK SPACE
                  *
                  *
                  *
                  * 'DELAY' COMPILER OPERATION
                  * GENERATE "ARG" MILLISECONDS OF DELAY
                  * USING RUN-TIME ROUTINE
                  *
2005 7F 0578 R    DELAY   CLR FLPFLG     FORCE INTEGER MODE
20D8 BD 1C47 R            JSR EVALS      EVAL. ARGUMENT
20DB C6 4B                LDA B #75
20DD BD 139C R            JSR RTBL       CALL RUN-TIME
20F0 7E 0937 R            JMP MAIN1      DONE
                  *
                  *
                  * ASSEMBLER "EVALUATION" ROUTINE
                  *
20E3 0002         VALUE   RMB   2        TEMPORARY LOCS
20E5 0002         TMPVAL  RMB   2
20E7 0001         CLFLG   RMB   1        CLASS OF PREVIOUS TOKEN
20E8 0001         CLASS   RMB   1        CLASS OF CURRENT TOKEN
20E9 0001         OPERN   RMB   1        ARITH. OPERATOR
                  *
20EA 7F 20E3 R    NSEVL   CLR   VALUE
20ED 7F 20E4 R            CLR   VALUE+1  VALUE.=0
20F0 7F 20E7 R            CLR   CLFLG    CLFLG.=0
20F3 B7 20E8 R            STA A CLASS    SAVE CLASS OF CURRENT TOKEN
20F6 C1 2A                CMP B #'*      * ?
20F8 27 12                BEQ   NSEVL3   YES
                  *
20FA C1 2B                CMP B #'+      UNARY PLUS?
20FC 26 08                BNE   NSEVL2   NO
                  *
20FE B7 20E7 R    NSEVL1  STA A CLFLG    HANDLE UNARY OPERATOR
2101 F7 20E9 R            STA B OPERN
2104 20 14                BRA   NSVLA    CONTINUE SCAN
                  *
2106 C1 2D        NSEVL2  CMP B #'-      UNARY MINUS?
2108 27 F4                BEQ   NSEVL1   YES
                  *
210A 20 2D                BRA   NSVLC1   NO
                  *
210C FE 0573 R    NSEVL3  LDX   LC       YES
210F FF 20E3 R            STX   VALUE    VALUE =LC
2112 86 02                LDA A #2
2114 B7 20E7 R            STA A CLFLG    CLFLG:=2
2117 73 0579 R            COM   RELFLG   RELFLG.=RELOC
                  *
211A FE 06F1 R    NSVLA   LDX   CUCHAR
211D A6 00                LDA A 0,X      GET NEXT CHAR
211F 81 20                CMP A #$20     BLANK?
2121 27 08                BEQ   NSVLB    YES
                  *
2123 81 0D                CMP A #$0D     EOL?
2125 27 04                BEQ   NSVLB    YES
                  *
2127 81 2C                CMP A #',      "," ?
2129 26 08                BNE   NSVLC    NO
                  *
```

```
212B FE 20E3 R NSVLB   LDX    VALUE
212E FF 239A R         STX    ADR1       ADR1,2.=VALUE
2131 5F                CLR B             RC.=00
2132 39                RTS               ALL DONE
                 *
2133 BD 172F R NSVLC   JSR    NXTOK      GET NEXT TOKEN
2136 B7 20F8 R         STA A  CLASS      SAVE CLASS
2139 B1 20E7 R NSVLC1  CMP A  CLFLG      CLASS=CLFLG?
213C 26 06             BNE    NSVLF      NO
                 *
213E CE 0204   NSVLD   LDX    #$0204     ERROR
2141 5F        NSVLE   CLR B
2142 53                COM B             RC.=FF
2143 39                RTS               RETURN
                 *
2144 81 02     NSVLF   CMP A  #$02       STRING?
2146 27 15             BEQ    NSVLH      YES
                 *
2148 81 01             CMP A  #$01       A,B, OR X REGISTER?
214A 27 11             BEQ    NSVLH      TREAT LIKE STRING
                 *
214C 81 24             CMP A  #$24       ARITHMETIC OPERATOR?
214E 26 EE             BNE    NSVLD      IF NOT, ERROR
                 *
2150 7D 20E7 R NSVLG   TST    CLFLG      CLFLG=0?
2153 27 E9             BEQ    NSVLD      YES,ERROR
                 *
2155 F7 20E9 R         STA B  OPERN      SAVE OPERATOR
2158 B7 20E7 R         STA A  CLFLG      CLFLG:=CLASS
215B 20 BD             BRA    NSVLA      SCAN AGAIN
                 *
215D C1 03     NSVLH   CMP B  #$03       HEX STRING?
215F 26 11             BNE    NSVLJ      NO
                 *
2161 F6 0644 R         LDA B  DESCRC     YES
2164 C1 04             CMP B  #4         > 4 ?
2166 2F 05             BLE    NSVLH1     NO
                 *
2168 CE 0210           LDX    #$0210     YES,ERROR
216B 20 D4             BRA    NSVLE
                 *
216D BD 2214 R NSVLH1  JSR    CVHB       CONVERT
2170 20 3B             BRA    NSVLM
                 *
2172 C1 09     NSVLJ   CMP B  #9         DECIMAL?
2174 26 11             BNE    NSVLK      NO
                 *
2176 F6 0644 R         LDA B  DESCRC
2179 C1 05             CMP B  #5         > 5 ?
217B 2F 05             BLE    NSVLJ1     NO
                 *
217D CE 0210           LDX    #$0210     YES,ERROR
2180 20 BF             BRA    NSVLE
                 *
2182 BD 226C R NSVLJ1  JSR    CVDB       CONVERT
2185 20 26             BRA    NSVLM
                 *
2187 C1 01     NSVLK   CMP B  #$01       SYMBOL?
2189 27 04             BEQ    NSVLL      YES
                 *
218B 81 01             CMP A  #$01       A,B,OR X REGISTER?
218D 26 AF             BNE    NSVLD      IF NOT,ERROR
                 *
218F BD 1A5C R NSVLL   JSR    LKPSYM     LOOKUP SYMBOL
2192 C5 80             BIT B  #$80       REDEFINED ?
2194 26 12             BNE    NSVLLA     YES
                 *
2196 C5 40             BIT B  #$40       RELOC ?
2198 27 05             BEQ    *+7        NO
219A 73 0579 R         COM    RELFLG     YES RELFLG.=RELOC
219D 20 0E             BRA    NSVLM
                 *
219F C5 10             BIT B  #$10       COMMON?
21A1 27 03             BEQ    *+5        NO
                 *
21A3 73 057A R         COM    CMNFLG     SET COMMON FLAG
21A6 20 05             BRA    NSVLM
                 *
21A8 CE 0211   NSVLLA  LDX    #$0211     NO,ERROR
21AB 20 94             BRA    NSVLE
                 *
21AD FF 20E5 R NSVLM   STX    TMPVAL     SAVE,CONVERTED VALUE
21B0 7D 20E7 R         TST    CLFLG      CLFLG=0 ?
21B3 26 0F             BNE    NSVLP      NO
                 *
21B5 FE 20E5 R         LDX    TMPVAL     YES
21B8 FF 20E3 R         STX    VALUE      VALUE.=TMPVAL
                 *
21BB B6 20E8 R NSVLN   LDA A  CLASS
21BE B7 20E7 R         STA A  CLFLG      CLFLG:=CLASS
21C1 7E 211A R         JMP    NSVLA      SCAN AGAIN
                 *
21C4 B6 20E9 R NSVLP   LDA A  OPERN       GET LAST OPERATOR
21C7 81 2B             CMP A  #'+        + ?
21C9 26 08             BNE    NSVLP1     NO
                 *
21CB CE 20E3 R         LDX    #VALUE
21CE BD 234B R         JSR    ADD16      VALUE:=VALUE+TMPVAL
21D1 20 E8             BRA    NSVLN
                 *
21D3 81 2D     NSVLP1  CMP A  #'-        - ?
21D5 26 08             BNE    NSVLP2     NO
                 *
21D7 CE 20E3 R         LDX    #VALUE     YES
21DA BD 235E R         JSR    SUB16      VALUE:=VALUE-TMPVAL
21DD 20 DC             BRA    NSVLN
                 *
21DF 81 2A     NSVLP2  CMP A  #'*        * ?
21E1 26 14             BNE    NSVLP3     NO
                 *
21E3 B6 20E3 R         LDA A  VALUE
21E6 F6 20E4 R         LDA B  VALUE+1
21E9 CE 20E5 R         LDX    #TMPVAL
21EC BD 22C3 R         JSR    MPY16      VALUE.=VALUE*TMPVAL
21EF B7 20E3 R         STA A  VALUE
21F2 F7 20E4 R         STA B  VALUE+1
21F5 20 C4             BRA    NSVLN
                 *
21F7 81 2F     NSVLP3  CMP A  #'/        / ?
21F9 27 03             BEQ    NSVLP4     YES
                 *
21FB 7E 213E R         JMP    NSVLD      NO, ERROR
```

```
                        *
21FE B6 20E3 R NSVIP4   LDA A VALUE
2201 F6 20E4 R          LDA B VALUE+1
2204 CE 20F5 R          LDX   #TMPVAL
2207 BD 2300 R          JSR   DIV16     VALUE:=VALUE/TMPVAL
220A B7 20E3 R          STA A VALUE
220D F7 20E4 R          STA B VALUE+1
2210 20 A9              BRA   NSVIN
                        *
                        *
                        * CVHB CONVERT HEX TO BINARY
                        *
                        * ON ENTRY DESCRA = ADDRESS OF STRING
                        *          DESCRC = # OF BYTES IN STRING
                        * ON RETURN [X]=VALUE
                        *
2212 0002               HVAL    RMB   2         TEMP STORAGE
                        *
2214 FE 0642 R CVHB     LDX   DESCRA    GET ADDRESS OF STRING
2217 7F 2212 R          CLR   HVAL
221A 7F 2213 R          CLR   HVAL+1
221D F6 0644 R          LDA B DESCRC    GET COUNT
2220 09                 DEX             DECREMENT PTR TO STRING
2221 08                 CVHB1 INX       POINT TO RIGHT MOST
2222 5A                 DEC B           BYTE OF THE
2223 26 FC              BNE   CVHB1     STRING
                        *
2225 F6 0644 R          LDA B DESCRC    GET COUNT
2228 8D 30              BSR   CVHBS     CONVERT
222A B7 2213 R          STA A HVAL+1    SAVE
222D 5A                 DEC B           DECREMENT COUNT
222E 27 26              BEQ   CVHBD     (1 HEX DIGIT)
                        *
2230 09                 DEX             POINT TO NEXT LEFT BYTE
2231 8D 27              BSR   CVHBS     CONVERT
2233 48                 ASL A           SHIFT TO LEFT NIBBLE
2234 48                 ASL A
2235 48                 ASL A
2236 48                 ASL A
2237 BA 2213 R          ORA A HVAL+1    CONVERT TO BYTE
223A B7 2213 R          STA A HVAL+1    SAVE
223D 5A                 DEC B           DECREMENT COUNT
223E 27 16              BEQ   CVHBD     (2 HEX DIGITS)
                        *
2240 09                 DEX             POINT TO NEXT LEFT BYTE
2241 8D 17              BSR   CVHBS     CONVERT
2243 B7 2212 R          STA A HVAL      SAVE
2246 5A                 DEC B           DECREMENT COUNT
2247 27 0D              BEQ   CVHBD     (3 HEX DIGITS)
                        *
2249 09                 DEX             POINT TO NEXT LEFT BYTE
224A 8D 0E              BSR   CVHBS     CONVERT
224C 48                 ASL A           SHIFT TO LEFT NIBBLE
224D 48                 ASL A
224E 48                 ASL A
224F 48                 ASL A
2250 BA 2212 R          ORA A HVAL      CONVERT TO BYTE
2253 B7 2212 R          STA A HVAL      SAVE
2256 FE 2212 R CVHBD    LDX   HVAL      GET FINAL VALUE
2259 39                 RTS             RETURN
                        *
```

```
                        * ROUTINE TO CONVERT ASCII TO BINARY
                        *
225A A6 00              CVHBS LDA A 0,X         GET BYTE
225C 80 30              SUB A #$30      CONVERT
225E 81 09              CMP A #$09      0 - 9 ?
2260 2F 02              BLE   *+4       YES
2262 80 07              SUB A #$07      NO, 10 - 15
2264 39                 RTS
                        *
                        *
                        * CVDB: CONVERT DECIMAL TO BINARY
                        * ON ENTRY DESCRA = ADDRESS OF DECIMAL STRING
                        *          DESCRC = # BYTES IN DECIMAL STRING
                        * ON RETURN [X] = VALUE IN BINARY
                        *
2265 0002               DVAL  RMB   2           TEMP STORAGE FOR BINARY
2267 0001               DCOUNT RMB  1           DIGIT COUNT
2268 0002               TENVL RMB   2           POWER OF TEN
226A 0002               DXSAV RMB   2           TEMPORARY STORAGE FOR X
                        *
226C 7F 2265 R CVDB     CLR   DVAL      DVAL:=0
226F 7F 2266 R          CLR   DVAL+1
2272 7F 2268 R          CLR   TENVL
2275 7F 2269 R          CLR   TENVL+1
2278 7C 2269 R          INC   TENVL+1   TENVL:=1
227B FE 0642 R          LDX   DESCRA    POINT TO STRING
227E 09                 DEX
227F F6 0644 R          LDA B DESCRC
2282 F7 2267 R          STA B DCOUNT    INIT DCOUNT
                        *
2285 08                 CVDB1 INX       POINT TO
2286 5A                 DEC B           LEAST SIGNIFICANT
2287 26 FC              BNE   CVDB1     DIGIT
                        *
2289 FF 226A R CVDB2    STX   DXSAV     SAVE POINTER
228C E6 00              LDA B 0,X       GET DIGIT
228E C4 0F              AND B #$0F      CONVERT TO BCD
2290 4F                 CLR A           CLEAR ACCUMULATOR
2291 CE 2268 R          LDX   #TENVL    POINT TO POWER OF TEN
2294 BD 22C3 R          JSR   MPY16     (A,B) =TENVL*DIGIT
2297 FB 2266 R          ADD B DVAL+1    DVAL:=DVAL+TENVL*DIGIT
229A B9 2265 R          ADC A DVAL
229D 8D 4A              BSR   OVFLW     CHECK OVERFLOWS
229F B7 2265 R          STA A DVAL
22A2 F7 2266 R          STA B DVAL+1
22A5 7A 2267 R          DEC   DCOUNT    DONE?
22A8 27 15              BEQ   CVDB3     YES
                        *
22AA 4F                 CLR A
22AB C6 0A              LDA B #$0A      B =10
22AD CE 2268 R          LDX   #TENVL    POINT TO POWER OF TEN
22B0 BD 22C3 R          JSR   MPY16     TENVL:=TENVL*10
22B3 B7 2268 R          STA A TENVL
22B6 F7 2269 R          STA B TENVL+1
22B9 FE 226A R          LDX   DXSAV     RESTORE POINTER TO STRING
22BC 09                 DEX             POINT NEXT LEFT DIGIT
22BD 20 CA              BRA   CVDB2
                        *
22BF FE 2265 R CVDB3    LDX   DVAL      GET FINAL VALUE
22C2 39                 RTS             RETURN
                        *
```

```
                    *
                    * MPY16 16 BIT MULTIPLY ROUTINE
                    * (A,B):=(A,B)*(2 BYTES POINTED AT BY X REG)
                    * USES 7 BYTES ON THE STACK
                    *
22C3 37             MPY16  PSH B          PUT VALUES ON TO THE STACK
22C4 36                    PSH A
22C5 A6 01                 LDA A 1,X
22C7 36                    PSH A
22C8 A6 00                 LDA A 0,X
22CA 36                    PSH A
22CB 86 10                 LDA A #16
22CD 36                    PSH A
22CE 30                    TSX           POINT TO DATA
22CF A6 03                 LDA A 3,X
                    *
22D1 58             MPY163 ASL B
22D2 49                    ROL A          FORM ANSWER
22D3 68 02                 ASL   2,X      SHIFT MULTIPLICAND
22D5 69 01                 ROL   1,X
22D7 24 06                 BCC   MPY167
                    *
22D9 EB 04                 ADD B 4,X      ADD MULTIPLIER
22DB A9 03                 ADC A 3,X
22DD 8D 0A                 BSR   OVFLW    CHECK OVERFLOWS
22DF 6A 00          MPY167 DEC   0,X
22E1 26 FE                 BNE   MPY163   COUNT NOT ZERO
                    *
22E3 31                    INS
22E4 31                    INS
22E5 31                    INS
22E6 31                    INS
22E7 31                    INS
22E8 39                    RTS            ALL DONE
                    *
                    *
                    * SUBROUTINE TO FLAG ARITHMETIC OVERFLOWS
                    *
22E9 28 11          OVFLW  BVC OVFLW2     OVERFLOW DETECTED?
                    *
22EB 7D 22FF R             TST OVFLG      OVERFLOW ALLOWED HERE?
22EE 26 0C                 BNE OVFLW2     YES
                    *
22F0 FF 22FD R             STX XSAVR
22F3 CE 0210               LDX #$0210
22F6 BD 24E8 R             JSR PRINTE     PRINT ERROR MESSAGE
22F9 FE 22FD R             LDX XSAVR
22FC 39             OVFLW2 RTS
                    *
22FD 0002           XSAVR  RMB 2
22FF 0001           OVFLG  RMB 1
                    *
                    *
                    * DIV16 16 BIT DIVIDE (UNSIGNED)
                    * (A,B):=(A,B)/ (X),(X+1)
                    * [X]=REMAINDER
                    *
2300 37             DIV16  PSH B          DIVIDEND TO STACK
2301 36                    PSH A
2302 A6 00                 LDA A 0,X
2304 E6 01                 LDA B 1,X
2306 37                    PSH B          DIVISOR TO STACK
2307 36                    PSH A
2308 34                    DES            LEAVE ROOM FOR COUNT
2309 30                    TSX            (X) PNTR TO STACKED DATA
230A 86 01                 LDA A #1
230C 6D 01                 TST   1,X
230E 2B 0B                 BMI   DIV153
                    *
2310 4C             DIV151 INC A
2311 68 02                 ASL   2,X
2313 69 01                 ROL   1,X
2315 2B 04                 BMI   DIV153
                    *
2317 81 11                 CMP A #17
2319 26 F5                 BNE   DIV151
                    *
231B A7 00          DIV153 STA A 0,X      SAVE COUNT
231D A6 03                 LDA A 3,X
231F E6 04                 LDA B 4,X
2321 6F 03                 CLR   3,X
2323 6F 04                 CLR   4,X
2325 E0 02          DIV163 SUB B 2,X
2327 A2 01                 SBC A 1,X
2329 24 07                 BCC   DIV165   DIVISOR STILL OK
                    *
232B EB 02                 ADD B 2,X      DIVISOR TOO LARGE
232D A9 01                 ADC A 1,X      RESTORE
232F 0C                    CLC
2330 20 01                 BRA   DIV167
                    *
2332 0D             DIV165 SEC
2333 69 04          DIV167 ROL   4,X
2335 69 03                 ROL   3,X
2337 64 01                 LSR   1,X      ADJUST DIVISOR
2339 66 02                 ROR   2,X
233B 6A 00                 DEC   0,X
233D 26 E6                 BNE   DIV163
                    *
233F A7 00                 STA A 0,X      SAVE REMAINDER IN X
2341 E7 01                 STA B 1,X
2343 EE 00                 LDX   0,X
2345 31                    INS            CLEAN UP STACK
2346 31                    INS
2347 31                    INS
2348 32                    PUL A
2349 33                    PUL B
234A 39                    RTS
                    *
                    *
                    * ADD16 16 BIT ADDITION
                    * [X] POINTS:
                    *          LOC(2),TEMP(2)
                    * LOC(2).=LOC(2)+TEMP(2)
                    *
234B 36             ADD16  PSH A
234C 37                    PSH B
234D A6 01                 LDA A 1,X
234F E6 00                 LDA B 0,X
2351 AB 03                 ADD A 3,X
2353 E9 02                 ADC B 2,X
2355 8D 92                 BSR OVFLW      CHECK OVERFLOWS
```

```
2357 A7 01              STA A 1,X
2359 E7 00              STA B 0,X
235B 33                 PUL B
235C 32                 PUL A
235D 39                 RTS
                *
                *
                * SUB16 16 BIT SUBTRACTION
                * [X] POINTS:
                *             LOC(2),TEMP(2)
                * LOC(2) =LOC(2)-TEMP(2)
                *
235E 36         SUB16   PSH A
235F 37                 PSH B
2360 A6 01              LDA A 1,X
2362 E6 00              LDA B 0,X
2364 A0 03              SUB A 3,X
2366 E2 02              SBC B 2,X
2368 BD 22E9 R          JSR OVFLW       CHECK OVERFLOWS
236B A7 01              STA A 1,X
236D E7 00              STA B 0,X
236F 33                 PUL B
2370 32                 PUL A
2371 39                 RTS
                *
                *
                * PRINTL PRINT A LINE ON THE TTY
                *
2372 B6 06F8 R  PRINTL  LDA A OPTNS     GET OPTIONS
2375 85 80              BIT A #$80      LIST?
2377 26 10              BNE   PLEND     NO
                *
2379 7D 0575 R          TST   PASS      PASS?
237C 27 0B              BEQ   PLEND     PASS1
                *
237E 7D 0625 R          TST   MACFLG    IN MACRO MODE?
2381 27 04              BEQ   PRINT1    NO
                *
2383 85 04              BIT A #$04      MACRO LIST OPTION?
2385 26 02              BNE   PLEND     NO
                *
2387 8D 1B      PRINT1  BSR   OUTL      PRINT A LINE
2389 39         PLEND   RTS             ALL DONE
                *
                * SPACE TO TOP OF PAGE
                *
238A B6 06F8 R  SPACER  LDA A OPTNS
238D 85 80              BIT A #$80      LISTING?
238F 26 05              BNE *+7         NO
                *
2391 86 0C              LDA A #$0C      "FORM-FEED"
2393 BD 3EAE R          JSR OUTCHR
                *
2396 39                 RTS
                *
                * PRINT A FORMATTED LINE OF LISTING ON THE TTY
                *
2397 0001       MCOUNT  RMB   1         # BYTES OF MACHINE CODE
2398 0001       POP     RMB   1         PSEUDOP;0=NO;1,2 BYTES
2399 0001       OPCD    RMB   1         OPCODE IN HEX
239A 0001       ADR1    RMB   1         INSTRUCTION ADDRESS
239B 0001       ADR2    RMB   1
239C 0005       LINEN   RMB   5         LINENUM IN ASCII
23A1 2E                 FCB   $2E       DECIMAL POINT
23A2 2004               FDB   $2004     SPACE,EOT
                *
                *
23A4 CE 239C R  OUTL    LDX   #LINEN    LOAD PARMS
23A7 B6 056F R          LDA A LNUM      LOAD LINE# (BINARY)
23AA F6 0570 R          LDA B LNUM+1
23AD BD 2487 R          JSR   CVBTD
23B0 CE 239D R          LDX   #LINEN+1  POINT TO DECIMAL LINE#
23B3 BD 3E8F R          JSR   PDATA1    PRINT LINE#,SPACE
                *
23B6 7D 0625 R          TST   MACFLG    IN MACRO MODE?
23B9 27 05              BEQ   *+7       NO
                *
23BB 86 2B              LDA A #'+       YES
23BD BD 3EAE R          JSR   OUTCHR
                *
23C0 7D 2397 R          TST   MCOUNT    PRINT LC ?
23C3 26 0D              BNE   OUTLA     YES
                *
23C5 7D 2398 R          TST   POP       PRINT LC?
23C8 26 08              BNE   OUTLA     YES
                *
23CA CE 2481 R          LDX   #BLANK3   NO,PRINT BLANKS (5)
23CD BD 3E8F R          JSR   PDATA1
23D0 20 2B              BRA   OUTL2
                *
23D2 CE 0573 R  OUTLA   LDX   #LC       POINT TO LC
23D5 BD 3EA0 R          JSR   OUT4HS    PRINT IN HEX,SPACE
23D8 F6 2398 R          LDA B POP       PSEUDOP?
23DB 27 20              BEQ   OUTL2     NO
                *
23DD C1 01              CMP B #$01      1 BYTE?
23DF 27 0E              BEQ   OUTL1     YES
                *
23E1 CE 239A R          LDX   #ADR1     POINT TO ADR1,ADR2
23E4 BD 3EA0 R          JSR   OUT4HS    PRINT 2 BYTES 4 HEX,SPACE
23E7 CE 2483 R          LDX   #BLANK5   POINT TO BLANKS
23EA BD 3E8F R          JSR   PDATA1    PRINT BLANKS
23ED 20 45              BRA   OUTL6
                *
23EF CE 239B R  OUTL1   LDX   #ADR2     POINT TO ADR2
23F2 BD 3EA2 R          JSR   OUT2HS    PRINT 1 BYTE 2 HEX,SPACE
23F5 CE 2481 R          LDX   #BLANK3   POINT TO BLANKS
23F8 BD 3E8F R          JSR   PDATA1    PRINT BLANKS
23FB 20 37              BRA   OUTL6
                *
23FD F6 2397 R  OUTL2   LDA B MCOUNT    PRINT NOTHING?
2400 26 08              BNE   OUTL3     NO
                *
2402 CE 247E R          LDX   #BLANK    PRINT JUST 8 BLANKS
2405 BD 3E8F R          JSR   PDATA1
2408 20 2A              BRA   OUTL6
                *
240A CE 2399 R  OUTL3   LDX   #OPCD
240D BD 3EA2 R          JSR   OUT2HS    PRINT OPCODE(HEX),SPACE
2410 C1 01              CMP B #$01      ONLY OPCODE?
2412 26 08              BNE   OUTL4     NO
                *
```

```
2414 CE 2481 R          LDX   #BLANK3    PRINT BLANKS
2417 BD 3E6F R          JSR   PDATA1
241A 20 18              BRA   OUTL6
                  *
241C C1 02        OUTL4 CMP B #$02       1 BYTE ADDRESS?
241E 26 0E              BNE   OUTL5      NO, 2 BYTES
                  *
2420 CE 239B R          LDX   #ADR2      POINT TO ADR2
2423 BD 3EA2 R          JSR   OUT2HS     PRINT 1 BYTE ADDRESS,SPACE
2426 CE 2484 R          LDX   #BLANK6    PRINT BLANKS
2429 BD 3E6F R          JSR   PDATA1
242C 20 06              BRA   OUTL6
                  *
242E CE 239A R OUTL5    LDX   #ADR1      POINT TO ADR1,ADR2
2431 BD 3EA0 R          JSR   OUT4HS     PRINT 2 BYTE ADDRESS,SPACE
2434 7D 057A R OUTL6    TST   CMNFLG     COMMON?
2437 27 04              BEQ   *+6        NO
                  *
2439 86 43              LDA A #'C
243B 20 1D              BRA   OUTL6B
                  *
243D 7D 057B R          TST   EXTFLG     EXTERNAL?
2440 27 04              BEQ   *+6        NO
                  *
2442 86 58              LDA A #'X
2444 20 14              BRA   OUTL6B
                  *
2446 7D 057C R          TST   ENTFLG     ENTRY?
2449 27 04              BEQ   *+6        NO
                  *
244B 86 4E              LDA A #'N
244D 20 0B              BRA   OUTL6B
                  *
244F 7D 0579 R          TST   RELFLG     RELOCATABLE?
2452 27 04              BEQ   *+6        NO
                  *
2454 86 52              LDA A #'R
2456 20 02              BRA   OUTL6B
                  *
2458 86 20              LDA A #$20       ABSOLUTE (SPACE)
                  *
245A BD 3EAE R OUTL6B   JSR   OUTCHR     PRINT TYPE
245D BD 3EA4 R          JSR   OUTS       PRINT SPACE
2460 FE 06F4 R          LDX   CULINE     POINT TO LINE
2463 A6 00        OUTL7 LDA A 0,X        GET CHAR
2465 36                 PSH A            SAVE A
2466 BD 3EAF R          JSR   OUTCHR     PRINT CHAR
2469 08                 INX              BUMP POINTER
246A 32                 PUL A            RESTORE A
246B 81 0D              CMP A #$0D       CR?
246D 26 F4              BNE   OUTL7      NO
                  *
246F 86 0A              LDA A #$0A       YES
2471 BD 3FAE R          JSR   OUTCHR     PRINT LF
2474 7F 2398 R          CLR   POF
2477 7F 2397 R          CLR   MCOUNT
247A 7F 0579 R          CLR   RELFLG
247D 39                 RTS
                  *
247E 20           BLANK FCB   $20        BLANKS.
247F 20                 FCB   $20
2480 20                 FCB   $20
2481 20           BLANK3 FCB  $20
2482 20                 FCB   $20
2483 20           BLANK5 FCB  $20
2484 20           BLANK6 FCB  $20
2485 20                 FCB   $20
2486 04                 FCB   $04        EOT
                  *
                  *
                  * CONVERT BINARY 16 BITS TO 5 DECIMAL CHARS
                  * ON ENTRY (A,B) = 16 BIT BINARY VALUE
                  * [X] = ADDRESS OF 5 BYTE STRING FOR DECIMAL
                  * (ASCII) CONVERTED VALUE.
                  *
2487 FF 24C8 R CVBTD    STX   SAVEX      SAVE DATA PTR
248A CE 24BD R          LDX   #KTOK      LOAD PTR TO CONSTANTS
248D 7F 24C7 R CVDEC1   CLR   SAVEA      INIT DEC CHAR
2490 E0 01     CVDEC2   SUB B 1,X
2492 A2 00              SBC A 0,X
2494 25 05              BCS   CVDEC5     OVERFLOW
2496 7C 24C7 R          INC   SAVEA      BUMP CHAR BEING BUILT
2499 20 F5              BRA   CVDEC2
                  *
249B EB 01     CVDEC5   ADD B 1,X        RESTORE PARTIAL RESULT
249D A9 00              ADC A 0,X
249F 36                 PSH A
24A0 FF 24CA R          STX   SAVEX1
24A3 FE 24C8 R          LDX   SAVEX      LOAD STORE CHAR PTR
24A6 B6 24C7 R          LDA A SAVEA
24A9 8B 30              ADD A #$30       MAKE ASCII CHAR
24AB A7 00              STA A 0,X
24AD 32                 PUL A
24AE 08                 INX
24AF FF 24C8 R          STX   SAVEX
24B2 FE 24CA R          LDX   SAVEX1     LOAD PTR TO CONSTANTS
24B5 08                 INX
24B6 08                 INX
24B7 8C 24C7 R          CPX   #KTOK+10
24BA 26 D1              BNE   CVDEC1
24BC 39                 RTS
                  *
                  * CONSTANTS
24BD 2710      KTOK     FDB   10000
24BF 03E8               FDB   1000
24C1 0064               FDB   100
24C3 000A               FDB   10
24C5 0001               FDB   1
                  *
                  * TEMPORARY STORAGE
                  *
24C7 0001      SAVEA    RMB   1
24C8 0002      SAVEX    RMB   2          STORE DATA PTR
24CA 0002      SAVEX1   RMB   2          PTR TO CONSTANTS
                  *
                  *
                  *
                  * PRINT ERROR MESSAGES ROUTINE *
                  * ON ENTRY [X] = ERROR# IN BCD *
                  *
24CC 0002      ERNUM    RMB   2          ERROR # IN BCD
24CE 0D0A      ERMSA    FDB   $0D0A
```

```
24D0 2A                FCC   '**** ERROR# '
24DC 0003       ERMSB  RMB   3          ERROR # IN ASCII
24DF 20                FCB   $20        BLANK
24E0 0005       ERMSC  RMB   5          LINE# IN ASCII
24E5 20                FCB   $20        BLANK
24E6 3A                FCC   ':'
24F7 04                FCB   $04        EOT
                *
24E8 36         PRINTE PSH   A
24E9 37                PSH   B
24FA FF 24CC R         STX   ERNUM      SAVE ERROR #
24ED B6 0654 R         LDA A LPTSW
24F0 26 03             BNE *+5          LINEPRINTER LISTS ERROR?
                *
24F2 7C 0653 R         INC CONSW        CONSOLE LISTS ERROR
                *
24F5 B6 24CC R         LDA A ERNUM      GET ERROR #
24F8 8B 30             ADD A #$30       CONVERT TO ASCII
24FA B7 24DC R         STA A ERMSB      SAVE
24FD B6 24CD R         LDA A ERNUM+1    GET ERROR #
2500 BD 3E75 R         JSR   OUTHL      CONVERT LEFT NIBBLE TO ASCII
2503 B7 24DD R         STA A ERMSB+1    SAVE
2506 B6 24CD R         LDA A ERNUM+1    GET ERROR#
2509 BD 3E79 R         JSR   OUTHR      CONVERT RIGHT NIBBLE TO ASCII
250C B7 24DE R         STA A ERMSB+2    SAVE
250F CE 24E0 R         LDX   #ERMSC     POINT TO LNUM AREA
2512 B6 056F R         LDA A LNUM
2515 F6 0570 R         LDA B LNUM+1
2518 BD 2487 R         JSR   CVBTD      CONVERT LNUM TO DECIMAL
251B CE 24CE R         LDX   #ERMSA     PRINT MESSAGE
251E BD 3E8F R         JSR   PDATA1
2521 FF 06F4 R         LDX   CULINE
2524 A6 00      PE2    LDA A 0,X        PRINT ERRONEOUS LINE
2526 36                PSH A            SAVE CHARACTER
2527 BD 3EAF R         JSR   OUTCHR
252A 32                PUL A            RECOVER CHARACTER
252B 81 0D             CMP A #$0D       CR?
252D 27 06             BEQ PE3          IF SO, STOP
                *
252F 08                INX
2530 BC 06F1 R         CPX   CUCHAR     STOP AT ERROR LOCATION
2533 26 EF             BNE   PE2
                *
2535 BD 3E84 R PE3     JSR   CRLF
2538 33                PUL B
2539 32                PUL A
253A FE 0703 R         LDX   ECOUNT     BUMP ECOUNT
253D 08                INX
253E FF 0703 R         STX   ECOUNT
                *
2541 7F 0653 R         CLR CONSW
2544 FE 24CC R         LDX   ERNUM
2547 39                RTS
                *
                *
                * PROCESS 'DIGSET' COMPILER OPCODE
                * CALL EXTERNAL DIGSET IN RUNTIME
                * NO  OF DIGITS IN INDEX REGISTER
                *
2548 BD 1A5C R DIGSPR JSR LKPSYM     CHECK FOR DIGSET
254B C1 FF             CMP B #$FF     FOUND?
254D 26 15             BNE DGSPR2     YES
                *
254F FF 056F R         LDX LNUM
2552 FF 0573 R         STX LC         SET LINE NUMBER
2555 BD 19FA R         JSR STOSYM     STORE DIGSET
2558 CE 3699 R         LDX #COMPL3
255B BD 13AB R         JSR STR1       OUTPUT ' BRA *+5'
255E CE 2573 R         LDX #DIGL1
2561 BD 13AB R         JSR STR1       OUTPUT ' EXT DIGSET'
                *
2564 7F 0578 R DGSPR2 CLR FLPFLG      FORCE INTEGER MODE
2567 BD 1C47 R         JSR EVALS      EVALUATE EXPRESSION
256A CE 257F R         LDX #DIGL2
256D BD 13AB R         JSR STR1       OUTPUT ' JSR DIGSET'
2570 7E 0937 R         JMP MAIN1      DONE!
                *
2573 20         DIGL1  FCC ' EXT DIGSET'
257E 0D                FCB $0D
257F 20         DIGL2  FCC ' JSR DIGSET'
258A 0D                FCB $0D
                *
                *
                * PROCESS "DSECTION" COMPILER OPCODE
258B CE 3B9D R DSECPR LDX #LABL+6
258E BD 13AB R         JSR STR1       OUTPUT DUMMY LINE
2591 B6 06F8 R         LDA A OPTNS
2594 B7 25A8 R         STA A DSCTMP   SAVE OPTION BYTE
2597 8A 50             ORA A #$50     TURN OFF I AND O BITS
2599 B7 06F8 R         STA A OPTNS    GENERATE NO CODE
259C 7E 0937 R         JMP MAIN1      DONE!
                *
                * PROCESS "DEND" COMPILER OPCODE
259F B6 25A8 R DENDPR LDA A DSCTMP    RESTORE OPTIONS
25A2 B7 06F8 R         STA A OPTNS
25A5 7E 0937 R         JMP MAIN1      DONE!
                *
25A8 0001       DSCTMP RMB 1          TEMP  OPTIONS
                *
                * PROCESS "CSECTION" COMPILER OPCODE
25A9 7C 0576 R CSECPR INC COMFLG      SET FLAG
25AC CE 3B9D R         LDX #LABL+6    OUTPUT DUMMY LINE
25AF BD 13AB R         JSR STR1
25B2 7E 0937 R         JMP MAIN1
                *
                * PROCESS "CEND" COMPILER OPCODE
25B5 7F 0576 R CENDPR CLR COMFLG      CLEAR FLAG
25B8 CE 3B9D R         LDX #LABL+6    OUTPUT DUMMY LINE
25BB BD 13AB R         JSR STR1
25BE 7E 0937 R         JMP MAIN1
                *
                *
                * PROCESS "DECLARE-STRING" COMPILER OPCODE
                * WORKS LIKE "INTEGER" OR "DEFINE"
                *
25C1 7C 25C6 R DSTPR   INC DSTFLG      SET D-STRING FLAG
25C4 20 06             BRA DIMPR       USE DIMENSION CODE
                *
25C6 0001       DSTFLG RMB 1           "DECLARE-STRING" FLAG
                *
                *
                * PROCESS "INTEGER" COMPILER OPCODE
```

```
                        * USES "DIMENSION" CODE
                        *
25C7 20 03              INTPR   BRA DIMPR
                        *
                        * PROCESS "DECLARE FLOATING" COMPILER OPCODE
                        * USES "DIMENSION" CODE
                        *
25C9 73 0578 R          DEFPR   COM FLPFLG      SET FLAG
                        *
                        * ONE OR TWO DIMENSIONS PERMITTED
                        *
25CC B6 06F8 R          DIMPR   LDA A OPTNS     SAVE "OPTIONS"
25CF B7 0701 R                  STA A PFLAG
25D2 7D 0576 R                  TST COMFLG      IN "CSECTION"?
25D5 26 19                      BNE DPR1        YES
                        *
25D7 FE 0708 R                  LDX LBLCNT      MAKE NEW LABEL
25DA 08                         INX
25DB FF 0708 R                  STX LBLCNT
25DE CE 3901 R                  LDX #BLOKL2     SET UP FOR CVBTD
25E1 B6 0708 R                  LDA A LBLCNT
25E4 F6 0709 R                  LDA B LBLCNT+1
25E7 BD 2487 R                  JSR CVBTD
25EA CE 38FB R                  LDX #BLOKL+6    JUMP AROUND DATA BLOCK
25ED BD 13AB R                  JSR STR1        OUTPUT CODE LINE
                        *
25F0 BD 172F R          DPR1    JSR NXTOK       GET NEXT TOKEN
25F3 C1 01                      CMP B #1        VALID NAME?
25F5 26 15                      BNE DPR2A       IF ERROR, QUIT
                        *
25F7 BD 1A5C R          DPR2    JSR LKPSYM      NAME ALREADY IN USE?
25FA C1 FF                      CMP B #FF
25FC 27 11                      BEQ DPR3        NOT FOUND
                        *
25FE C5 80                      BIT B #80       PARAMETER?
2600 27 0A                      BEQ DPR2A       NO, ERROR
                        *
2602 B6 06F8 R                  LDA A OPTNS     FORCE "DSECT"
2605 8A 50                      ORA A #50       NO-I, NO-O
2607 B7 06F8 R                  STA A OPTNS
260A 20 0C                      BRA DPR3A
                        *
260C 7E 2675 R          DPR2A   JMP DPR4
                        *
260F FE 056F R          DPR3    LDX LNUM
2612 FF 0573 R                  STX LC          PUT LINE NO  INTO SYMBOL
2615 BD 19FA R                  JSR STOSYM      STORE NAME IN SYM. TABLE
2618 7D 0578 R          DPR3A   TST FLPFLG      IN "DEF"?
261B 27 09                      BEQ DEFP1       NO
                        *
261D FE 06F6 R                  LDX SYMPTR
2620 A6 08                      LDA A 8,X
2622 8A 08                      ORA A #08       SET FLOAT-BIT
2624 A7 08                      STA A 8,X
                        *
2626 7D 0576 R          DEFP1   TST COMFLG      IN "CSECTION"?
2629 27 1E                      BEQ DEFP1A      NO
                        *
262B CE 2645 R                  LDX #CMNL1
262E 86 04                      LDA A #4
2630 B7 06F0 R                  STA A CMPCNT
2633 BD 0DAF R                  JSR OUTCMP      " CMN "
2636 BD 0DA6 R                  JSR OUTNAM      "NAME"
2639 86 2C                      LDA A #',
263B BD 0D98 R                  JSR WRTCHR      ","
263E C6 2C                      LDA B #',
2640 BD 0DA3 R                  JSR TPCHAR
2643 20 12                      BRA DEFP1B
                        *
2645 20                 CMNL1   FCC ' CMN'
                        *
2649 BD 0DA6 R          DEFP1A  JSR OUTNAM      PUT OUT LABEL NAME
264C CE 0448 R                  LDX #RMBZ
264F 86 03                      LDA A #3
2651 B7 06F0 R                  STA A CMPCNT
2654 BD 0DAF R                  JSR OUTCMP      OUTPUT 'RMB' CODE
2657 BD 172F R          DEFP1B  JSR NXTOK       GET NEXT TOKEN
265A C1 28                      CMP B #'(       LEFT PAREN?
265C 27 20                      BEQ DPR5        YES
                        *
265E 81 0D                      CMP A #$0D      CR?
2660 27 04                      BEQ DEFP2       YES
                        *
2662 C1 2C                      CMP B #',       COMMA?
2664 26 0F                      BNE DPR4        NO, ERROR
                        *
2666 CE 0649 R          DEFP2   LDX #OPTIN
2669 4F                         CLR A
266A C6 02                      LDA B #2        INTEGER HAS 2 BYTES
266C 7D 0578 R                  TST FLPFLG      IN INTEGER MODE?
266F 27 74                      BEQ DEFP3       YES
                        *
2671 C6 09                      LDA B #9        FLOAT HAS 9 BYTES
2673 20 70                      BRA DEFP3
                        *
                        *
2675 CE 0303            DPR4    LDX #$0303
2678 BD 24E8 R                  JSR PRINTE      PRINT ERROR MSG
267B 7E 0937 R                  JMP MAIN1       QUIT
                        *
267E B6 0578 R          DPR5    LDA A FLPFLG    SAVE FLAG
2681 36                         PSH A
2682 7F 0578 R                  CLR FLPFLG      FORCE INTEGER
2685 BD 172F R                  JSR NXTOK       GET NEXT TOKEN
2688 32                         PUL A
2689 B7 0578 R                  STA A FLPFLG    RESTORE FLAG
268C C1 09                      CMP B #9        DECIMAL NUMBER?
268E 26 E5                      BNE DPR4        IF NOT, ERROR
                        *
2690 FE 06F6 R                  LDX SYMPTR
2693 A6 08                      LDA A 8,X
2695 8A 02                      ORA A #$02      SET FIRST DIM. FLAG
2697 A7 08                      STA A 8,X
2699 BD 226C R                  JSR CVDB        CONVERT TO BINARY
269C 7D 25C6 R                  TST DSTFLG      IN STRING MODE?
269F 27 01                      BEQ *+3         NO
                        *
26A1 08                         INX             SPACE FOR END-STRING CHARS
                        *
26A2 FF 20E3 R                  STX VALUE
26A5 2B CE                      BMI DPR4        OVERFLOW ERROR
                        *
```

```
26A7 BD 172F R          JSR NXTOK       GET NEXT TOKEN
26AA FE 06F6 R DPR6     LDX SYMPTR
26AD B6 20E3 R          LDA A VALUE     SET DIM  VAL.  INTO TABLE
26B0 A7 06              STA A 6,X
26B2 B6 20E4 R          LDA A VALUE+1
26B5 A7 07              STA A 7,X
26B7 C1 29              CMP B #')       RIGHT PAREN NOW?
26B9 27 03              BEQ *+5         NO-TWO DIMS
26BB 7E 273B R          JMP DPR9
                *
26BE B6 20E3 R          LDA A VALUE     SET UP FOR CVBTD
26C1 F6 20E4 R          LDA B VALUE+1
26C4 CE 0649 R DPR7     LDX #OPTLN      TEMP  STORAGE
26C7 7D 25C6 R          TST DSTFLG      IN STRING MODE?
26CA 26 19              BNE DEFP3       ONE BYTE/CHARACTER
                *
26CC 7D 0578 R          TST FLPFLG      IN FLOAT MODE?
26CF 27 10              BEQ DPR7A       NO
                *
26D1 CE 26DF R          LDX #C9
26D4 BD 22C3 R          JSR MPY16       9 BYTES FOR FLOAT
26D7 CE 0649 R          LDX #OPTLN
26DA 4D                 TST A
26DB 2B 98              BMI DPR4        OVERFLOW ERROR?
                *
26DD 20 06              BRA DEFP3
                *
26DF 0009       C9      FDB 9
                *
26E1 58         DPR7A   ASL B           2 BYTES FOR INTEGER
26E2 49                 ROL A
26E3 2D 90              BMI DPR4        OVERFLOW ERROR?
                *
26E5 BD 2487 R DEFP3    JSR CVBTD
26E8 CE 0649 R          LDX #OPTLN
26EB 86 05              LDA A #5
26ED B7 06F0 R          STA A CMPCNT
26F0 BD 0DAF R          JSR OUTCMP      OUTPUT CODE
26F3 BD 0DD3 R          JSR OUTCR
26F6 B6 0701 R          LDA A PFLAG     RESTORE "OPTNS"
26F9 B7 06F8 R          STA A OPTNS
26FC 7D 25C6 R          TST DSTFLG      IN STRING MODE?
26FF 27 09              BEQ DEFP3A      NO
                *
2701 FE 06F6 R          LDX SYMPTR      YES
2704 A6 08              LDA A 8,X
2706 8A 40              ORA A #$40      SET STRING BIT
2708 A7 08              STA A 8,X
                *
270A FE 0642 R DEFP3A   LDX DESCRA      GET LAST TOKEN
270D A6 00              LDA A 0,X
270F 81 2C              CMP A #',       WAS IT COMMA?
2711 27 0B              BEQ DEFP4       IF SO, CONTINUE
                *
2713 81 0D              CMP A #$0D      WAS IT CR?
2715 27 0A              BEQ DPR8        IF SO, DONE
                *
2717 BD 172F R          JSR NXTOK       GET NEXT TOKEN
271A C1 2C              CMP B #',       A COMMA?
271C 26 03              BNE DPR8        NO MORE VARIABLES ON LINE
                *
271F 7E 25F0 R DEFP4    JMP DPR1        PROCESS ANOTHER VARIABLE
                *
2721 7D 0576 R DPR8     TST COMFLG      IN "CSECTION"?
2724 26 12              BNE DPR8A       YES, DONE!
                *
2726 CE 3898 R          LDX #LABL+1
2729 B6 0708 R          LDA A LBLCNT
272C F6 0709 R          LDA B LBLCNT+1  SET UP FOR CVBTD
272F BD 2487 R          JSR CVBTD
2732 CE 3897 R          LDX #LABL
2735 BD 13AB R          JSR STR1        ENDING LABEL
2738 7E 0937 R DPR8A    JMP MAIN1       DONE
                *
273B B6 0578 R DPR9     LDA A FLPFLG    SAVE FLAG
273E 36                 PSH A
273F 7F 0578 R          CLR FLPFLG      FORCE INTEGER
2742 BD 172F R          JSR NXTOK
2745 32                 PUL A
2746 B7 0578 R          STA A FLPFLG    RESTORE FLAG
2749 C1 09              CMP B #9        DECIMAL NUMBER?
274B 26 21              BNE DPR11       IF NOT, ERROR
                *
274D BD 226C R          JSR CVDB        CONVERT VALUE
2750 FF 20E3 R          STX VALUE
2753 2B 19              BMI DPR11       OVERFLOW ERROR
                *
2755 BD 172F R          JSR NXTOK
2758 FE 06F6 R DPR10    LDX SYMPTR
275B A6 08              LDA A 8,X       SET SECOND DIM  FLAG
275D 8A 01              ORA A #$01
275F A7 08              STA A 8,X
2761 A6 06              LDA A 6,X       PREPARE FOR MULTIPLY
2763 E6 07              LDA B 7,X
2765 CE 20E3 R          LDX #VALUE
2768 BD 22C3 R          JSR MPY16
276B 4D                 TST A
276C 2A 03              BPL *+5         OVERFLOW?
276E 7E 2675 R DPR11    JMP DPR4        ERROR
                *
2771 7E 26C4 R          JMP DPR7        NO ERROR
                *
                *
                * HANDLE "PRINT" COMPILER OPCODE
                *
2774 5F         PRNTPR  CLR B
2775 F7 27BD R          STA B WRFLG     CLEAR 'READ' FLAG
2778 F7 27BE R          STA B FIOFLG    CLEAR 'FILE' FLAG
277B 7E 286B R          JMP   WPRO      FINISH UP LIKE "WRITE"
                *
                *
                * HANDLE "INPUT" COMPILER OPCODE
                *
277E 7F 27BE R INPPR    CLR   FIOFLG    CLEAR 'FILE' FLAG
2781 C6 FF              LDA B #$FF
2783 F7 27BD R          STA B WRFLG     SET 'READ' FLAG
2786 7E 286B R          JMP   WPRO      FINISH UP LIKE "READ"
                *
                *
                * HANDLE "OPEN" COMPILER OPCODE
                *
2789 BD 2A00 R OPENPR   JSR IOCBPR      PROCESS (IOCB)
```

```
278C BD 172F R          JSR NXTOK        GET NEXT TOKEN
278F C1 27              CMP B #$27       QUOTE?
2791 26 0B              BNE OPE1         NO
                 *
                 * CODE:      LDX #*+5
                 *           BRA LABEL
                 *           FCC '------'
                 *           FCB $04
                 *     LABEL EQU *
                 *
2793 CE 16D8 R          LDX #STRCL1      FIRST LINE OF CODE
2796 BD 13AB R          JSR STR1
2799 BD 298D R          JSR QUOTZ        HANDLE QUOTED STRING
279C 20 09              BRA OPE1A
                 *
279E FE 0642 R OPE1     LDX DESCRA       BACK UP TOKEN POINTER
27A1 FF 06F1 R          STX CUCHAR
27A4 BD 198A R          JSR GETADR       POINT TO STRING
                 *
27A7 C6 AE       OPE1A  LDA B #174
                 *
27A9 BD 139C R OPE2     JSR   RTBL       OUTPUT RUNTIME CALL
27AC 7E 0937 R          JMP   MAIN1      DONE!
                 *
                 *
                 * HANDLE "CLOSE" COMPILER OPCODE
                 *
27AF BD 2A00 R CLOSPR   JSR IOCBPR       HANDLE (IOCB)
27B2 C6 B1              LDA B #177
27B4 20 F3              BRA   OPE2       FINISH UP LIKE "OPEN"
                 *
                 *
                 * HANDLE "REWIND" COMPILER OPCODE
                 *
27B6 BD 2A00 R REWPR    JSR IOCBPR       HANDLE (IOCB)
27B9 C6 B4              LDA B #180
27BB 20 EC              BRA   OPE2       FINISH UP LIKE "OPEN"
                 *
                 *
                 *
                 *
                 * PROCESS "READ" COMPILER OPCODE
                 * USES "WRITE" FOR SLASH (LF), AND QUOTED STRINGS
                 *
27BD 0001        WRFLG  RMB 1            READ/WRITE FLAG
27BE 0001        FIOFLG RMB 1            FILE I/O FLAG
                 *
27BF C6 FF       READPR LDA B #$FF
27C1 F7 27BD R          STA B WRFLG      SET 'READ' FLAG
27C4 F7 27BE R          STA B FIOFLG     SET 'FILE' FLAG
27C7 BD 2A00 R          JSR IOCBPR       HANDLE (IOCB)
27CA 7E 286B R          JMP WPR0         USE "WRITE" CODE
                 *
                 * GENERATE THE FOLLOWING CODE FOR INPUT:
                 *       JSR INPUT
                 *       STX NAME
                 *
27CD C1 01       READ2  CMP B #1         A NAME?
27CF 27 09              BEQ READ3        O K
                 *
27D1 CE 0310            LDX #$0310
27D4 BD 24E8 R          JSR PRINTE       OUTPUT ERROR MESSAGE
27D7 7E 286B R          JMP WPR0
                 *
27DA BD 1A5C R READ3    JSR LKPSYM       CHECK SYMBOL TABLE
27DD C1 FF              CMP B #$FF       FOUND?
27DF 27 5D              BEQ READ4        IF NOT, INTEGER
                 *
27E1 C5 08              BIT B #$08       FLOATING POINT?
27E3 27 20              BEQ READ3A       NO
                 *
                 * GENERATE FLOATING-POINT CODE
                 *
27E5 F7 139A R          STA B GAFLG      SET FLAGS
27E8 F7 0578 R          STA B FLPFLG
27EB BD 3DED R          JSR EVADIM+3
27EE C6 5A              LDA B #90
27F0 7D 27BE R          TST FIOFLG       FILE I/O?
27F3 27 02              BEQ *+4          NO
                 *
27F5 C6 BA              LDA B #186       YES
                 *
27F7 BD 139C R          JSR RTBL         OUTPUT CALL
27FA 7D 24CC R          TST ERNUM        ERROR?
27FD 27 03              BEQ *+5          NO
27FF 7E 0937 R          JMP MAIN1        IF SO, QUIT
                 *
2802 7E 2956 R          JMP WPR3B        CONTINUE
                 *
2805 5D          READ3A TST B            PARAMETER?
2806 2A 20              BPL READ3B       NO
                 *
                 * MAKE PARAMETER CODE
2808 C6 2D              LDA B #45
280A 7D 27BE R          TST FIOFLG       FILE I/O?
280D 27 02              BEQ *+4          NO
                 *
280F C6 B7              LDA B #183       YES
2811 BD 139C R          JSR RTBL         RUNTIME CALL
2814 CE 0F86 R          LDX #STRL1
2817 BD 13AB R          JSR STR1
281A 86 FF              LDA A #$FF
281C B7 139A R          STA A GAFLG
281F BD 3DED R          JSR EVADIM+3
2822 BD 0F6E R          JSR STRD2
2825 7E 2956 R          JMP WPR3B        DONE!
                 *
2828 C5 03       READ3B BIT B #3         CHECK FOR ARRAY
282A 27 12              BEQ READ4        NOT ARRAY
                 *
                 * GENERATE ARRAY ELEMENT CODE
282C C6 2D              LDA B #45        OUTPUT CALL
282E 7D 27BE R          TST FIOFLG       FILE I/O?
2831 27 02              BEQ *+4          NO
                 *
2833 C6 B7              LDA B #183       YES
                 *
2835 BD 139C R          JSR RTBL
2838 BD 0F65 R          JSR STRDIM       FORM INDEXING CODE
283B 7E 2956 R          JMP WPR3B        DONE
                 *
283F C6 2D       READ4  LDA B #45
```

```
2840 7D 27BE R          TST FIOFLG      FILE I/O?
2843 27 02              BEQ *+4         NO
                *
2845 C6 B7              LDA B #183      YES
                *
2847 BD 139C R          JSR RTBL        OUTPUT CALL
284A CE 15F3 R          LDX #STRP1
284D 86 04              LDA A #4
284F B7 06F0 R          STA A CMPCNT
2852 BD 0DAF R          JSR OUTCMP      OUTPUT LINE 2
2855 BD 0DA6 R          JSR OUTNAM      OUTPUT NAME
2858 BD 0DD3 R          JSR OUTCR
285B BD 2A4E R          JSR DEFS        DEFINE NAME (IF NOT ALREADY)
285E 20 0B              BRA WPR0        CONTINUE
                *
                * PROCESS "WRITE" COMPILER OPCODE
                * CAN HANDLE NAME, QUOTED STRING, OR SLASH (LF)
                * 'GREATER THAN' (>) FUNCTION OUTPUTS SPACES
                *
2860 7F 27BD R WRITPR   CLR WRFLG       CLEAR 'READ' FLAG
2863 C6 FF              LDA B #$FF
2865 F7 27BE R          STA B FIOFLG    SET 'FILE' FLAG
2868 BD 2A00 R          JSR IOCBPR      HANDLE (IOCB)
                *
286B BD 172F R WPR0     JSR NXTOK       GET NEXT TOKEN
286E C1 29              CMP B #')       RIGHT PAREN?
2870 27 F9              BEQ WPR0        SKIP IT
                *
2872 C1 27              CMP B #$27      A QUOTE?
2874 26 05              BNE WPR1        NO
                *
2876 BD 297B R          JSR QUOTPR      PROCESS QUOTED STRING
2879 20 F0              BRA WPR0
                *
287B C1 2F      WPR1    CMP B #'/       A SLASH?
287D 26 1C              BNE WPR2        NO
                *
287F C6 2A              LDA B #42       PROCESS A LINEFEED
2881 7D 27BE R          TST FIOFLG      FILE I/O?
2884 27 02              BEQ *+4         NO
                *
2886 C6 CF              LDA B #207      YES
2888 7D 27BD R          TST WRFLG       READ OR WRITE?
288B 27 09              BEQ *+11        WRITE
                *
288D C6 2A              LDA B #42       READ
288F 7D 27BE R          TST FIOFLG      FILE I/O?
2892 27 02              BEQ *+4         NO
                *
2894 C6 C9              LDA B #201      YES
                *
2896 BD 139C R          JSR RTBL        OUTPUT CODE
2899 20 D0              BRA WPR0
                *
289B C1 0D      WPR2    CMP B #$0D      A CR?
289D 26 03              BNE *+5         NO
289F 7E 0937 R          JMP MAIN1       DONE
                *
28A2 C1 3E              CMP B #'>       'GREATER THAN' SIGN (>)?
28A4 26 15              BNE WPR3        NO
                *
                * GET SPACE COUNT INTO X-REGISTER
                * AND CALL RUNTIME ROUTINE
28A6 7F 0578 R          CLR FLPFLG      FORCE INTEGER
28A9 BD 1C47 R          JSR EVALS       GET COUNT
28AC C6 39              LDA B #57
28AE 7D 27BE R          TST FIOFLG      FILE I/O?
28B1 27 02              BEQ *+4         NO
                *
28B3 C6 CC              LDA B #204      YES
                *
28B5 BD 139C R          JSR RTBL        CALL RUNTIME
28B8 7E 2956 R          JMP WPR3B       FINISH UP
                *
                *
28BB C1 2C      WPR3    CMP B #',       A COMMA?
28BD 27 AC              BEQ WPR0        YES, LOOK AGAIN
                *
28BF C1 5B              CMP B #'[       LEFT BRACKET?
28C1 26 30              BNE FMT2        NO
                *
                * HANDLE FORMAT SPECIFICATION
                * [INTEGER DIGITS,DECIMAL PLACES]
                *
28C3 7F 0578 R          CLR FLPFLG      FORCE INTEGER
28C6 BD 1C47 R          JSR EVALS       EVALUATE 'SIZE'
28C9 CE 15F3 R          LDX #STRP1
28CC BD 13AB R          JSR STR1        SAVE IN 'ACCUM4'
28CF FE 0642 R          LDX DESCRA
28D2 A6 00              LDA A 0,X       CHECK TERMINATOR
28D4 81 2C              CMP A #',       COMMA?
28D6 27 08              BEQ FMT1        YES
                *
28D8 CE 3691 R          LDX #COMPL2     DEFAULT 'DECPL'=0
28DB BD 13AB R          JSR STR1
28DE 20 03              BRA *+5
                *
28E0 BD 1C47 R FMT1     JSR EVALS       EVALUATE 'DECPL'
                *
28E3 C6 D2              LDA B #210      RUNTIME CALL TO DIGSET
28E5 BD 139C R          JSR RTBL
28E8 FE 0642 R          LDX DESCRA
28EB A6 00              LDA A 0,X       CHECK TERMINATOR
28ED 81 5D              CMP A #']       RIGHT BRACKET?
28EF 27 6C              BEQ WPR4        YES
                *
28F1 20 63              BRA WPR3B       NO, BACK UP
                *
                *
28F3 C1 25      FMT2    CMP B #'%       PERCENT SIGN?
28F5 26 24              BNE WPR3A       NO, THEN EXPRESSION
                *
                * HANDLE STRING I/O
                *
28F7 BD 138A R          JSR GETADR      GET ADDRESS
28FA 27 03              BEQ *+5         NO ERROR FOUND
28FC 7E 0937 R          JMP MAIN1       IF SO, QUIT
                *
28FF C6 F3              LDA B #243      OUTPUT (FORMATTED)
2901 7D 27BE R          TST FIOFLG      FILE I/O?
2904 27 02              BEQ *+4         NO
                *
```

```
2906 C6 F6                LDA B #246        YES
                   *
2908 7D 27BD R            TST WRFLG         READ/WRITE?
290B 27 09                BEQ WPR30         OUTPUT
                   *
290D C6 36                LDA B #54         INPUT
290F 7D 27BE R            TST FIOFLG        FILE I/O?
2912 27 02                BEQ *+4           NO
                   *
2914 C6 BD                LDA B #189        YES
                   *
2916 BD 139C R WPR30      JSR RTBL
2919 20 3B                BRA WPR3B         CONTINUE
                   *
                   * GENERATE THE FOLLOWING CODE:
                   *      LDX VALUE
                   *      JSR OUTPUT
                   *
291B 7D 27BD R WPR3A      TST WRFLG         READING OR WRITING?
291E 27 03                BEQ *+5
2920 7E 27CD R            JMP READ2
                   *
2923 FE 0642 R            LDX DESCRA
2926 FF 06F1 R            STX CUCHAR        BACK UP A TOKEN
2929 BD 3812 R            JSR PRESCN        PRESCAN FOR FLOAT MODE
292C BD 1C47 R            JSR EVALS         EVALUATE EXP.
292F 7D 24CC R            TST ERNUM         EVALUATION ERROR?
2932 27 03                BEQ *+5           NO
2934 7F 0937 R            JMP MAIN1         YES. QUIT
                   *
2937 C6 30                LDA B #48
2939 7D 27BE R            TST FIOFLG        FILE I/O?
293C 27 02                BEQ *+4           NO
                   *
293E C6 C0                LDA B #192        YES .
                   *
2940 7D 0578 R            TST FLPFLG        TEST FLOAT MODE
2943 26 05                BNE WPR3C         FLOATING POINT
                   *
2945 BD 139C R            JSR RTBL          OUTPUT CALL
2948 20 0C                BRA WPR3B
                   *
294A C6 57        WPR3C   LDA B #87         FLOAT OUTPUT
294C 7D 27BE R            TST FIOFLG        FILE I/O?
294F 27 02                BEQ *+4           NO
                   *
2951 C6 C3                LDA B #195        YES
                   *
2953 BD 1224 R            JSR FUNOP         OUTPUT CALL
2956 FE 06F1 R WPR3B      LDX CUCHAR
2959 09                   DEX
295A FF 06F1 R            STX CUCHAR
295D 7E 266B R WPR4       JMP WPR0
                   *
                   * HANDLE QUOTED MATERIAL IN I/O STMTS.
                   * GENERATE THE FOLLOWING CODE.
                   *      LDX #*+9
                   *      JSR PDATA1
                   *      BRA .LABEL
                   *      FCC '-------------'
                   *      FCB $04 'EOT' MARK
                   * LABEL EQU *
                   *
2960 20           QL1     FCC ' LDX #*+9'
2969 0D                   FCB $0D
296A 20           QL3     FCC ' FCC '
296F 27           QUOTE   FCB $27
2970 0D                   FCB $0D
2971 2F           PDOT    FCC '/'
2972 20           QL4     FCC ' FCB $04'
297A 0D                   FCB $0D
                   *
297B CE 2960 R QUOTPR     LDX #QL1
297E BD 13AB R            JSR STR1          OUTPUT CODE LINE 1
2981 C6 33                LDA B #51
2983 7D 27BE R            TST FIOFLG        FILE I/O?
2986 27 02                BEQ *+4           NO
                   *
2988 C6 C6                LDA B #198        YES
                   *
298A BD 139C R            JSR RTBL          OUTPUT CODE LINE 2
298D 86 04        QUOT2   LDA A #4
298F B7 06F0 R            STA A CMPCNT
2992 CE 1DD0 R            LDX #FCL2
2995 BD 0DAF R            JSR OUTCMP        OUTPUT CODE LINE 3
2998 FE 0708 R            LDX LBLCNT
299B 08                   INX               MAKE NEW COMPILER LABEL
299C FF 0708 R            STX LBLCNT
299F FF 3BAF R            STX BLKBOT
29A2 B6 2971 R            LDA A PDOT
29A5 B7 0649 R            STA A OPTLN
29A8 CE 064A R            LDX #OPTLN+1      SET UP FOR CVBTD
29AB B6 3BAF R            LDA A BLKBOT
29AE F6 3BB0 R            LDA B BLKBOT+1
29B1 BD 2487 R            JSR CVBTD
29B4 8D 3F                BSR QPR1          OUTPUT LABEL
29B6 BD 0DD3 R            JSR OUTCR
29B9 CE 296A R            LDX #QL3
29BC 86 04                LDA A #4
29BE B7 06F0 R            STA A CMPCNT
29C1 BD 0DAF R            JSR OUTCMP        OUTPUT CODE LINE 4
29C4 86 27                LDA A #$27
29C6 BD 0D98 R            JSR WRTCHR
29C9 C6 27                LDA B #$27
29CB BD 0DA3 R            JSR TPCHAR
29CE BD 323A R            JSR FCCB          OUTPUT QUOTED STRING
29D1 CE 296F R            LDX #QUOTE
29D4 BD 13AB R            JSR STR1          OUTPUT ' CR
29D7 CE 2972 R            LDX #QL4
29DA BD 0DAF R            JSR OUTCMP        OUTPUT 'EOT' CODE
29DD 8D 16                BSR QPR1          OUTPUT LABEL
29DF CE 3B9F R            LDX #LABL+7
29E2 86 05                LDA A #5
29E4 B7 06F0 R            STA A CMPCNT
29E7 BD 0DAF R            JSR OUTCMP
29EA BD 0DD3 R            JSR OUTCR         FINISH LABEL
29ED FE 06F1 R            LDX CUCHAR
29F0 08                   INX
29F1 FF 06F1 R            STX CUCHAR
29F4 39                   RTS
                   *
29F5 CE 0649 R QPR1       LDX #OPTLN
```

```
29F8 86 06              LDA A #6
29FA B7 06F0 R          STA A CMPCNT
29FD 7E 0DAF R          JMP OUTCMP
                 *
                 *
                 * PROCESS 'IOCB' STRUCTURE (IOCB)
                 *
2A00 BD 172F R IOCBPR   JSR NXTOK       GET A TOKEN
2A03 C1 28              CMP B #'(       LEFT PAREN?
2A05 26 18              BNE IOCERR      IF NOT, ERROR
                 *
2A07 BD 138A R          JSR GETADR      GET ADDRESS OF IOCB
2A0A 26 13              BNE IOCERR      ERROR?
                 *
2A0C C6 EA              LDA B #234
2A0F BD 139C R          JSR RTBL        RUNTIME CALL
2A11 BD 172F R          JSR NXTOK       GET A TOKEN
2A14 C1 29              CMP B #')       RIGHT PAREN?
2A16 27 06              BEQ IOCB2       YES
                 *
2A18 FE 0642 R          LDX DESCRA      NO, BACK UP A TOKEN
2A1B FF 06F1 R          STX CUCHAR
2A1E 39          IOCB2  RTS             DONE
                 *
2A1F CE 0334     IOCERR LDX #$0334      IOCB ERROR
2A22 BD 24E8 R          JSR PRINTE
2A25 31                 INS            SKIP SUBROUTINE RETURN
2A26 31                 INS
2A27 7E 0937 R          JMP MAIN1      QUIT
                 *
                 *
                 * "POKE" COMPILER OPCODE HANDLER
                 * BYTE OUTPUT TO ADDRESS IN X-REGISTER
                 * PUT DATA INTO ACCUM3
                 *
2A2A 7F 0578 R POKEPR   CLR FLPFLG      FORCE INTEGER
2A2D BD 1C47 R          JSR EVALS       GET DATA
2A30 7D 24CC R          TST ERNUM       ERROR?
2A33 26 16              BNE POKES       YES, QUIT
                 *
2A35 CE 0F86 R          LDX #STR1       STORE-DATA CODE
2A38 BD 13AB R          JSR STR1
2A3B 7F 0578 R          CLR FLPFLG      FORCE INTEGER
2A3F BD 1C47 R          JSR EVALS       GET ADDRESS
2A41 7D 24CC R          TST ERNUM
2A44 26 05              BNE POKES       IF ERROR, QUIT
                 *
2A46 C6 54              LDA B #84       RUNTIME-CALL CODE
2A48 BD 139C R          JSR RTBL
2A4B 7E 0937 R POKES    JMP MAIN1       DONE
                 *
                 *
                 *
                 *
                 * DEFINE STORAGE FOR VARIABLE
                 * IF NOT ALREADY DEFINED
                 *
2A4E BD 1A5C R DEFS     JSR LKPSYM      CHECK SYMBOL TABLE
2A51 C1 FF              CMP B #$FF      FOUND?
2A53 27 01              BEQ DEFS2       NO, MUST DEFINE IT.
                 *
2A55 39                 RTS             YES, QUIT
                 *
2A56 FE 056F R DEFS2    LDX LNUM        PUT LINE NUMBER INTO SYMBOL DEF
2A59 FF 0573 R          STX LC
2A5C BD 19FA R          JSR STOSYM      PUT INTO SYMBOL TABLE
                 *
                 * GENERATE THE FOLLOWING CODE:
                 *      BRA *+4
                 *NAME  RMB 2
                 *
2A5F CE 2A7C R          LDX #DEFSL1
2A62 BD 13AB R          JSR STR1        OUTPUT LINE 1
2A65 FE 0642 R          LDX DESCRA
2A68 B6 0644 R          LDA A DESCRC
2A6B B7 06F0 R          STA A CMPCNT
2A6E BD 0DAF R          JSR OUTCMP      OUTPUT NAME
2A71 CE 2A85 R          LDX #DEFSL2
2A74 BD 13AB R          JSR STR1        OUTPUT LINE 2
2A77 5F                 CLR B           RETURN NO-DIMENSIONS
2A78 CE 0000            LDX #$0000
2A7B 39                 RTS
                 *
2A7C 20          DEFSL1 FCC ' BRA *+4'
2A84 0D                 FCB $0D
2A85 52          DEFSL2 FCC 'RMB 2'
2A8A 0D                 FCB $0D
                 *
                 *
                 *
                 *          **ADDRESS TYPE 1**
                 *
                 * (ADC ADD AND BIT CMP EOR LDA ORA SBC SUB)
                 *
                 * IMMEDIATE (2 BYTES).
                 *   CCC A #NUMBER       CCC B #NUMBER
                 *   CCC A #SYMBOL       CCC B #SYMBOL
                 *   CCC A #EXPRESSION   CCC B #EXPRESSION
                 *   CCC A #'C           CCC B # C
                 *
                 * DIRECT(2 BYTES) OR EXTENDED(3 BYTES)
                 *   CCC A NUMBER        CCC B NUMBER
                 *   CCC A SYMBOL        CCC B SYMBOL
                 *   CCC A EXPRESSION    CCC B EXPRESSION
                 *
                 * INDEXED(2 BYTES).
                 *   CCC A NUMBER,X      CCC B NUMBER,X
                 *   CCC A SYMBOL,X      CCC B SYMBOL,X
                 *   CCC A EXPRESSION,X  CCC B EXPRESSION,X
                 *
                 *
2A8B BD 2D12 R ADDR1   JSR   ADRIN1     INIT ADDRESS FIELD VALUES
2A8E BD 172F R         JSR   NXTOK      GET NEXT TOKEN
2A91 C1 0D             CMP B #$0D       EOL?
2A93 26 08             BNE   ADDR1B     NO
                 *
2A95 CE 0204   ADDR1A  LDX   #$0204     ERROR
2A98 BD 24E8 R         JSR   PRINTE     PRINT
2A9B 20 5B             BRA   ADDR1E     RETURN
                 *
2A9D BD 2D40 R ADDR1B  JSR   ABRCK      CHECK FOR REGISTER A OR B
2AA0 F6 2D0D R         LDA B ABR        NEITHER?
```

```
2AA3 27 F0              BEQ    ADDR1A     YES ERROR
                  *
2AA5 BD 172F R          JSR    NXTOK      GET NEXT TOKEN
2AA8 C1 23              CMP B  #'#        IMMED  MODE?
2AAA 26 14              BNE    ADDR1C     NO
                  *
2AAC 73 2D0E R          COM    IMMED      SET IMMEDIATE FLAG
2AAF BD 172F R          JSR    NXTOK      GET NEXT TOKEN
2AB2 C1 27              CMP B  #$27       "'" ?
2AB4 26 0A              BNE    ADDR1C     NO
                  *
2AB6 FE 06F1 R          LDX    CUCHAR     GET NEXT CHAR
2AB9 A6 00              LDA A  0,X
2ABB B7 239B R          STA A  ADR2
2ABE 20 0B              BRA    ADDR1K
                  *
2AC0 BD 20EA R ADDR1C   JSR    NSEVL      EVALUATE OPERAND
2AC3 BD 2D60 R          JSR    P2ERR      PRINT PASS 2 ERRORS
2AC6 F6 2D0E R          LDA B  IMMED      IMMEDIATE MODE?
2AC9 27 0C              BEQ    ADDR1D     NO
                  *
2ACB C6 80      ADDR1K  LDA B  #$80       IMMEDIATE FORM A
2ACD F7 2D10 R          STA B  ORBYA      NIBBLE
2AD0 C6 C0              LDA B  #$C0       OF
2AD2 F7 2D11 R          STA B  ORBYB      MACHINE CODE
2AD5 20 3C              BRA    ADDR1H
                  *
2AD7 BD 172F R ADDR1D   JSR    NXTOK      GET NEXT TOKEN
2ADA BD 2D4D R          JSR    INXCK      INDEXED?
2ADD 26 2A              BNE    ADDR1G     YES
                  *
2ADF 7D 057A R          TST    CMNFLG     COMMON?
2AE2 26 0A              BNE    ADDR1L     YES
                  *
2AE4 7D 0579 R          TST    RELFLG     RELOC ?
2AE7 26 05              BNE    ADDR1L     YES
                  *
2AE9 F6 239A R          LDA B  ADR1       DIRECT?
2AEC 27 0F              BEQ    ADDR1F     YES
                  *
2AEE C6 B0      ADDR1L  LDA B  #$B0       EXTENDED,FORM A
2AF0 F7 2D10 R          STA B  ORBYA      NIBBLE
2AF3 C6 F0              LDA B  #$F0       OF
2AF5 F7 2D11 R          STA B  ORBYB      MACHINE CODE
                  *
2AF8 BD 2DE4 R ADDR1E   JSR    LCNAB3     FORM MACHINE CODE
2AFB 20 19              BRA    ADDR1J
                  *
2AFD C6 90      ADDR1F  LDA B  #$90       DIRECT,FORM A
2AFF F7 2D10 R          STA B  ORBYA      NIBBLE
2B02 C6 D0              LDA B  #$D0       OF
2B04 F7 2D11 R          STA B  ORBYB      MACHINE CODE
2B07 20 0A              BRA    ADDR1H
                  *
2B09 C6 A0      ADDR1G  LDA B  #$A0       INDEXED,FORM A
2B0B F7 2D10 R          STA B  ORBYA      NIBBLE OF
2B0E C6 E0              LDA B  #$E0       OF
2B10 F7 2D11 R          STA B  ORBYB      MACHINE CODE
                  *
2B13 BD 2D9C R ADDR1H   JSR    LCNAB2     FORM MACHINE CODE
2B16 BD 2E48 R ADDR1J   JSR    LCLCN      LC =LC+LCN
2B19 7E 0937 R          JMP    MAIN1      RETURN TO MAIN LOOP
                  *
                  *
                  *
                  *          **ADDRESS TYPE 2**
                  *
                  * [STA]
                  *
                  * DIRECT(2 BYTES) OR EXTENDED(3 BYTES)
                  *"   CCC A NUMBER        CCC B NUMBER
                  *    CCC A SYMBOL        CCC B SYMBOL
                  *    CCC A EXPRESSION    CCC B EXPRESSION
                  *
                  * INDEXED(2 BYTES):
                  *    CCC A NUMBER,X      CCC B NUMBER,X
                  *    CCC A SYMBOL,X      CCC B SYMBOL,X
                  *    CCC A EXPRESSION,X  CCC B EXPRESSION,X
                  *
                  *
2B1C BD 2D12 R ADDR2    JSR    ADRINT     INIT ADDRESS FIELD FLAGS
2B1F BD 172F R          JSR    NXTOK      GET NEXT TOKEN
2B22 C1 0D              CMP B  #$0D       EOL?
2B24 26 08              BNE    ADDR2B     NO
                  *
2B26 CE 0204    ADDR2A  LDX    #$0204     ERROR
2B29 BD 24E8 R          JSR    PRINTE     PRINT
2B2C 20 32              BRA    ADDR2E     RETURN
                  *
2B2E BD 2D40 R ADDR2B   JSR    ABRCK      CHECK FOR REGISTER A OR B
2B31 F6 2D0D R          LDA B  ABR        NEITHER?
2B34 27 F0              BEQ    ADDR2A     YES ERROR
                  *
2B36 BD 172F R          JSR    NXTOK      GET NEXT TOKEN
2B39 BD 20EA R          JSR    NSEVL      EVALUATE OPERAND
2B3C BD 2D60 R          JSR    P2ERR      PRINT PASS 2 ERRORS
2B3F BD 172F R          JSR    NXTOK      GET NEXT TOKEN
2B42 BD 2D4D R          JSR    INXCK      INDEXED?
2B45 26 2A              BNE    ADDR2G     YES
                  *
2B47 7D 057A R          TST    CMNFLG     COMMON?
2B4A 26 0A              BNE    ADDR2K     YES
                  *
2B4C 7D 0579 R          TST    RELFLG     RELOC ?
2B4F 26 05              BNE    ADDR2K     YES
                  *
2B51 F6 239A R          LDA B  ADR1       DIRECT?
2B54 27 0F              BEQ    ADDR2F     YES
                  *
2B56 C6 B0      ADDR2K  LDA B  #$B0       EXTENDED,FORM A
2B58 F7 2D10 R          STA B  ORBYA      NIBBLE
2B5B C6 F0              LDA B  #$F0       OF
2B5D F7 2D11 R          STA B  ORBYB      MACHINE CODE
                  *
2B60 BD 2DE4 R ADDR2E   JSR    LCNAB3     FORM MACHINE CODE
2B63 20 19              BRA    ADDR2J
                  *
2B65 C6 90      ADDR2F  LDA B  #$90       DIRECT,FORM A
2B67 F7 2D10 R          STA B  ORBYA      NIBBLE
2B6A C6 D0              LDA B  #$D0       OF
2B6C F7 2D11 R          STA B  ORBYB      MACHINE CODE
2B6F 20 0A              BRA    ADDR2H
```

```
                        *
2B71 C6 A0              ADDR2G LDA B #$A0      INDEXED,FORM A
2B73 F7 2D10 R                 STA B ORBYA     NIBBLE
2B76 C6 E0                     LDA B #$E0      OF
2B78 F7 2D11 R                 STA B ORBYB     MACHINE CODE
                        *
2B7B BD 2D9C R ADDR2H  JSR     LCNAB2          FORM MACHINE CODE
2B7E BD 2E48 R ADDR2J  JSR     LCLCN           LC.=LC+LCN
2B81 7E 0937 R                 JMP     MAIN1   RETURN TO MAIN LOOP
                        *
                        *
                        *
                        *        **ADDRESS TYPE 3**
                        *
                        * [ASL ASR CLR COM DEC INC LSR NEG ROL ROR TST]
                        *
                        * ACCUMULATOR(1 BYTE):
                        *   CCC A
                        *   CCC B
                        *
                        * EXTENDED(3 BYTES):
                        *   CCC NUMBER
                        *   CCC SYMBOL
                        *   CCC EXPRESSION
                        *
                        * INDEXED(2 BYTES)
                        *   CCC NUMBER,X
                        *   CCC SYMBOL,X
                        *   CCC EXPRESSION,X
                        *
                        *
2B84 BD 2D12 R ADDR3   JSR     ADRINI          INIT ADDRESS FIELD FLAGS
2B87 BD 172F R                 JSR     NXTOK   GET NEXT TOKEN
2B8A C1 0D                     CMP B #$0D      EOL?
2B8C 26 08                     BNE     ADDR3B  NO
                        *
2B8E CE 0204                   LDX     #$0204  ERROR
2B91 BD 24E8 R                 JSR     PRINTE  PRINT
2B94 20 2A                     BRA     ADDR3D  RETURN
                        *
2B96 BD 2D40 R ADDR3B  JSR     ABRCK           CHECK FOR REGISTER A OR B
2B99 7D 2D0D R                 TST     ABR     NEITHER?
2B9C 27 0F                     BEQ     ADDR3C  YES
                        *
2B9E C6 40                     LDA B #$40      ACCUMULATOR,FORM A
2BA0 F7 2D10 R                 STA B ORBYA     NIBBLE
2BA3 C6 50                     LDA B #$50      OF
2BA5 F7 2D11 R                 STA B ORBYB     MACHINE CODE
2BA8 BD 2D73 R                 JSR     LCNAB1  FORM MACHINE CODE
2BAB 20 20                     BRA     ADDR3F
                        *
2BAD BD 20EA R ADDR3C  JSR     NSEVL           EVALUATE OPERAND
2BB0 BD 2D60 R                 JSR     P2ERR   PRINT PASS 2 ERRORS
2BB3 BD 172F R                 JSR     NXTOK   GET NEXT TOKEN
2BB6 BD 2D4D R                 JSR     INXCK   INDEXED?
2BB9 26 0A                     BNE     ADDR3E  YES
                        *
2BBB C6 70                     LDA B #$70      EXTENDED,FORM A
2BBD F7 2D10 R                 STA B ORBYA     NIBBLE OF MACHINE CODE
                        *
2BC0 BD 2E02 R ADDR3D  JSR     LCN3            FORM MACHINE CODE
2BC3 20 08                     BRA     ADDR3F
                        *
2BC5 C6 60              ADDR3E LDA B #$60      INDEXED,FORM A
2BC7 F7 2D10 R                 STA B ORBYA     NIBBLE OF MACHINE CODE
2BCA BD 2DBA R                 JSR     LCN2    FORM MACHINE CODE
                        *
2BCD BD 2E48 R ADDR3F  JSR     LCLCN           LC:=LC+LCN
2BD0 7E 0937 R                 JMP     MAIN1   RETURN TO MAIN LOOP
                        *
                        *
                        *
                        *        **ADDRESS TYPE 4**
                        *
                        * [PSH PUL]
                        *
                        * ACCUMULATOR(1 BYTE)
                        *   PSH A
                        *   PSH B
                        *   PUL A
                        *   PUL B
                        *
                        *
2BD3 BD 2D12 R ADDR4   JSR     ADRINI          INIT ADDRESS FIELD FLAGS
2BD6 BD 172F R                 JSR     NXTOK   GET NEXT TOKEN
2BD9 BD 2D40 R                 JSR     ABRCK   CHECK FOR A,B REGS
2BDC 7D 2D0D R                 TST     ABR     NEITHER ?
2BDF 26 06                     BNE     ADDR4A  NO
                        *
2BE1 CE 0204                   LDX     #$0204  ERROR
2BE4 BD 24E8 R                 JSR     PRINTE
                        *
2BE7 7C 2D11 R ADDR4A  INC     ORBYB           ORBYB.=01
2BEA BD 2D73 R                 JSR     LCNAB1  FORM MC
2BED BD 2E48 R                 JSR     LCLCN   LC.=LC+LCN
2BF0 7E 0937 R                 JMP     MAIN1   RETURN TO MAIN LOOP
                        *
                        *
                        *
                        *        **ADDRESS TYPE 5**
                        *
                        * [CPX LDS LDX]
                        *
                        * IMMEDIATE(3 BYTES):
                        *   CCC  #NUMBER
                        *   CCC  #SYMBOL
                        *   CCC  #EXPRESSION
                        *   CCC  #'CC
                        *
                        * DIRECT(2 BYTES) OR EXTENDED(3 BYTES):
                        *   CCC  NUMBER
                        *   CCC  SYMBOL
                        *   CCC  EXPRESSION
                        *
                        * INDEXED(2 BYTES)
                        *   CCC  NUMBER,X
                        *   CCC  SYMBOL,X
                        *   CCC  EXPRESSION,X
                        *
                        *
2BF3 BD 2D12 R ADDR5   JSR     ADRINI          INIT ADDRESS FIELD FLAGS
2BF6 BD 172F R                 JSR     NXTOK   GET NEXT TOKEN
```

```
2BF9 C1 0D              CMP B #$0D      EOL ?
2BFB 26 08              BNE   ADDR5B    NO
                *
2BFD CE 0240    ADDR5A  LDX   #$0240    ERROR
2C00 BD 24E8 R          JSR   PRINTE    PRINT
2C03 20 44              BRA   ADDR5E    RETURN
                *
2C05 C1 23      ADDR5B  CMP B #'#       IMMEDIATE?
2C07 26 19              BNE   ADDR5C    NO
                *
2C09 73 2D0E R          COM   IMMED     SET IMMEDIATE FLAG
2C0C BD 172F R          JSR   NXTOK     GET NEXT TOKEN
2C0F C1 27              CMP B #$27      "/" ?
2C11 26 0F              BNE   ADDR5C    NO
                *
2C13 FE 06F1 R          LDX   CUCHAR    YES, GET NEXT TWO CHARS
2C16 A6 00              LDA A 0,X
2C18 B7 239A R          STA A ADR1
2C1B A6 01              LDA A 1,X
2C1D B7 239B R          STA A ADR2
2C20 20 27              BRA   ADDR5E
                *
2C22 BD 20EA R ADDR5C   JSR   NSEVL     EVALUATE OPERAND
2C25 BD 2D60 R          JSR   P2ERR     PRINT PASS 2 ERRORS
2C28 F6 2D0E R          LDA B IMMED     IMMEDIATE?
2C2B 26 1C              BNE   ADDR5E    YES
                *
2C2D BD 172F R ADDR5D   JSR   NXTOK     GET NEXT TOKEN
2C30 BD 2D4D R          JSR   INXCK     INDEXED?
2C33 26 20              BNE   ADDR5G    YES
                *
2C35 7D 057A R          TST   CMNFLG    COMMON?
2C38 26 0A              BNE   ADDR5K    YES
                *
2C3A 7D 0579 R          TST   RELFLG    RELOC ?
2C3D 26 05              BNE   ADDR5K    YES
                *
2C3F F6 239A R          LDA B ADR1      DIRECT?
2C42 27 0A              BEQ   ADDR5F    YES
                *
2C44 C6 30      ADDR5K  LDA B #$30      EXTENDED, FORM A
2C46 F7 2D10 R          STA B ORBYA     NIBBLE OF MACHINE CODE
2C49 BD 2E02 R ADDR5E   JSR   LCN3      FORM MACHINE CODE
2C4C 20 0F              BRA   ADDR5J
                *
2C4E C6 10      ADDR5F  LDA B #$10      DIRECT, FORM A
2C50 F7 2D10 R          STA B ORBYA     NIBBLE OF MACHINE CODE
2C53 20 05              BRA   ADDR5H
                *
2C55 C6 20      ADDR5G  LDA B #$20      INDEXED, FORM A
2C57 F7 2D10 R          STA B ORBYA     NIBBLE OF MC
                *
2C5A BD 2DBA R ADDR5H   JSR   LCN2      FORM MC
2C5D BD 2E48 R ADDR5J   JSR   LCLCN     LC =LC+LCN
2C60 7E 0937 R          JMP   MAIN1     RETURN TO MAIN LOOP
                *
                *
                *
                *        **ADDRESS TYPE 6**
                *
                * [STX,STS]
                *
                * DIRECT(2 BYTES) OR EXTENDED(3 BYTES)
                *   CCC  NUMBER
                *   CCC  SYMBOL
                *   CCC  EXPRESSION
                *
                * INDEXED(2 BYTES)
                *   CCC  NUMBER,X
                *   CCC  SYMBOL,X
                *   CCC  EXPRESSION,X
                *
2C63 BD 2D12 R ADDR6    JSR   ADRIN1    INIT ADDRESS FIELD FLAGS
2C66 BD 172F R          JSR   NXTOK     GET NEXT TOKEN
2C69 C1 0D              CMP B #$0D      EOL?
2C6B 26 B5              BNE   ADDR5C    NO
2C6D 20 8E              BRA   ADDR5A    YES, ERROR
                *
                *
                *
                *        **ADDRESS TYPE 7**
                * [JMP JSR]
                *
                * EXTENDED(3 BYTES).
                *   CCC  NUMBER
                *   CCC  SYMBOL
                *   CCC  EXPRESSION
                *
                * INDEXED(2 BYTES):
                *   CCC  NUMBER,X
                *   CCC  SYMBOL,X
                *   CCC  EXPRESION,X
                *
                *
2C6F BD 2D12 R ADDR7    JSR   ADRIN1    INIT ADDRESS FIELD FLAGS
2C72 BD 172F R          JSR   NXTOK     GET NEXT TOKEN
2C75 C1 0D              CMP B #$0D      EOL?
2C77 26 08              BNE   ADDR7A    NO
                *
2C79 CE 0204            LDX   #$0204    ERROR
2C7C BD 24E8 R          JSR   PRINTE    PRINT
2C7F 20 0E              BRA   ADDR7B
                *
2C81 BD 20EA R ADDR7A   JSR   NSEVL     EVALUATE OPERAND
2C84 BD 2D60 R          JSR   P2ERR     PRINT PASS 2 ERRORS
2C87 BD 172F R          JSR   NXTOK     GET NEXT TOKEN
2C8A BD 2D4D R          JSR   INXCK     INDEXED?
2C8D 26 0A              BNE   ADDR7C    YES
                *
2C8F C6 10      ADDR7B  LDA B #$10      EXTENDED, FORM A NIBBLE
2C91 F7 2D10 R          STA B ORBYA     OF MC
2C94 BD 2E02 R          JSR   LCN3      FORM MACHINE CODE
2C97 20 03              BRA   ADDR7D
                *
2C99 BD 2DBA R ADDR7C   JSR   LCN2      FORM MACHINE CODE
2C9C BD 2E48 R ADDR7D   JSR   LCLCN     LC =LC+LCN
2C9F 7E 0937 R          JMP   MAIN1     RETURN TO MAIN LOOP
                *
                *
                *
                *        **ADDRESS TYPE 8**
                *
```

```
                    * [BCC BCS BEQ BGE BGT BHI BLE BLS
                    *  BLT BMI BNE BPL BRA BSR BVC BVS]
                    *
                    * RELATIVE(2 BYTES):
                    *   CCC  NUMBER
                    *   CCC  SYMBOL
                    *   CCC  EXPRESSION
                    *
                    *
2CA2 BD 2D12 R ADDR8  JSR   ADRINT   INIT ADDRESS FIELD FLAGS
2CA5 BD 172F R        JSR   NXTOK    GET NEXT TOKEN
2CA8 C1 0D            CMP B #$0D     EOL?
2CAA 26 08            BNE   ADDR8A   NO
                    *
2CAC CE 0204          LDX   #$0204   ERROR
2CAF BD 24E8 R        JSR   PRINTE   PRINT
2CB2 20 36            BRA   ADDR8D
                    *
2CB4 7D 0575 R ADDR8A TST   PASS     PASS ?
2CB7 27 31            BEQ   ADDR8D   PASS1
                    *
2CB9 BD 20FA R        JSR   NSEVL    PASS 2 EVAL OPERAND
2CBC BD 2D60 R        JSR   P2ERR    PRINT PASS 2 ERRORS
2CBF FE 0573 R        LDX   LC       LSAVE.=LC+2
2CC2 08               INX
2CC3 08               INX
2CC4 FF 06FA R        STX   LSAVE
2CC7 B6 239B R        LDA A ADR2     CALCULATE OFFSET
2CCA F6 239A R        LDA B ADR1
2CCD B0 06FB R        SUB A LSAVE+1
2CD0 F2 06FA R        SBC B LSAVE
                    *
2CD3 C1 FF            CMP B #$FF     CHECK FOR OUT OF RANGE
2CD5 26 03            BNE   ADDR8E
                    *
2CD7 4D               TST A          NEGATIVE? (FF - 80)
2CD8 2B 0D            BMI   ADDR8C   OK
                    *
2CDA C1 00     ADDR8E CMP B #$00
2CDC 26 03            BNE   ADDR8F   OUT OF RANGE
                    *
2CDE 4D               TST A          POSITIVE? (00 - 7F)
2CDF 2A 06            BPL   ADDR8C   OK
                    *
2CE1 CE 0208   ADDR8F LDX   #$0208   ERROR
2CE4 BD 24E8 R        JSR   PRINTE   PRINT
                    *
2CE7 B7 239B R ADDR8C STA A ADR2     SAVE OFFSET
2CEA BD 2DDA R ADDR8D JSR   LCN2     FORM MC
2CED BD 2E48 R        JSR   LCLCN    LC.=LC+LCN
2CF0 7E 0937 R        JMP   MAIN1    RETURN TO MAIN LOOP
                    *
                    *
                    *
                    *         *ADDRESS TYPE 9**
                    *
                    * [ABA CBA CLC CLI CLV DES DEX INS
                    *  INX NOP RTI RTS SBA SEC SEI SEV
                    *  SWI TAB TAP TBA TPA TSX TXS WAI]
                    *
                    * INHERENT(1 BYTE):
                    * CCC
                    *
                    *
2CF3 BD 2D12 R ADDR9  JSR   ADRINT   INIT ADDRESS FIELD FLAGS
2CF6 7D 0575 R        TST   PASS     PASS ?
2CF9 27 03            BEQ   ADDR9A   PASS 1
                    *
2CFB BD 3E55 R        JSR   OUTBIN   OUTPUT MC
                    *
2CFE 7C 2397 R ADDR9A INC   MCOUNT   MCOUNT =1
2D01 7C 06F9 R        INC   LCN      LCN:=1
2D04 BD 2372 R        JSR   PRINTL
2D07 BD 2E48 R        JSR   LCLCN    LC.=LC+LCN
2D0A 7E 0937 R        JMP   MAIN1    RETURN TO MAIN LOOP
                    *
                    *
                    *
                    * ROUTINES USED TO INIT AND CHECK ADDRESS FIELD
                    * FLAGS,MC FORMS AND LISTING FLAGS
                    *
2D0D 0001      ABR    RMB   1        REG A OR B FLAG
2D0E 0001      IMMED  RMB   1        IMMEDIATE MODE FLAG
2D0F 0001      INDEX  RMB   1        INDEX MODE FLAG
2D10 0001      ORBYA  RMB   1        FORM FOR A NIBBLE OF MC
2D11 0001      ORBYB  RMB   1        FORM FOR A NIBBLE OF MC
                    *
2D12 7F 06F9 R ADRINT CLR   LCN
2D15 7F 0579 R        CLR   REIFLG
2D18 7F 057A R        CLR   CNNFLG
2D1B 7F 057B R        CLR   EXTFLG
2D1E 7F 057C R        CLR   ENTFLG
2D21 7F 2398 R        CLR   POP
2D24 F7 2399 R        STA B OPCD     SAVE OPCODE
2D27 7F 2397 R        CLR   MCOUNT
2D2A 7F 2D0D R        CLR   ABR
2D2D 7F 2D0E R        CLR   IMMED
2D30 7F 2D0F R        CLR   INDEX
2D33 7F 239A R        CLR   ADR1
2D36 7F 239B R        CLR   ADR2
2D39 7F 2D10 R        CLR   ORBYA
2D3C 7F 2D11 R        CLR   ORBYB
2D3F 39               RTS
                    *
                    *
                    * CHECK FOR PRESENCE OF A OR B REG
                    *
2D40 C1 41     ABRCK  CMP B #'A      "A" ?
2D42 27 05            BEQ   ABRCKA   YES
                    *
2D44 C1 42            CMP B #'B      "B" ?
2D46 27 01            BEQ   ABRCKA   YES
                    *
2D48 39               RTS            NEITHER, RETURN
                    *
2D49 F7 2D0D R ABRCKA STA B ABR      SAVE REG
2D4C 39               RTS
                    *
                    * CHECK FOR INDEXED MODE
                    *
2D4D C1 2C     INXCK  CMP B #',      "," ?
2D4F 26 0B            BNE   INXCKR   NO
```

```
                        *
2D51 BD 172F R          JSR   NXTOK      GET NEXT TOKEN
2D54 C1 58              CMP B #'X        "X" ?
2D56 26 04              BNE   INXCKR     NO
                        *
2D58 73 2D0F R          COM   INDEX      INDEX,=FF
2D5B 39                 RTS
                        *
2D5C 7F 2D0F R INXCKR   CLR   INDEX
2D5F 39                 RTS
                        *
                        *
                        * CHECK FOR PASS 2 ERRORS
                        *
2D60 C1 FF              P2ERR CMP B #$FF       ERROR (FROM NSEVL)?
2D62 26 0E              BNE   P2ERRB     NO
                        *
2D64 7D 0575 R          TST   PASS       YES,PASS?
2D67 27 03              BEQ   *+5        PASS1
                        *
2D69 BD 24E8 R          JSR   PRINTE     PASS 2,PRINT ERROR
                        *
2D6C 7F 239A R          CLR   ADR1
2D6F 73 239A R          COM   ADR1       ADR1:=FF (TO KILL DIRECT)
2D72 39                 P2ERRB RTS
                        *
                        *
                        *
                        * ROUTINES TO FINISH UP ADDRESS TYPE PROCESSING
                        * THESE ROUTINES DO THE FOLLOWING:
                        *   PASS 1
                        *      A. LCN:= # OF BYTES IN THE INSTRUCTION
                        *   PASS 2
                        *      A  FORM COMPLETE OPCODE
                        *      B  OUTPUT MACHINE CODE GENERATED
                        *      C. PRINT A LINE OF LISTING
                        *      D. LCN:= # OF BYTES IN THE INSTRUCTION
                        *
                        * LCNAB1 1 BYTE ACCUMULATOR INSTRUCTIONS
                        *
2D73 7D 0575 R LCNAB1   TST   PASS       PASS?
2D76 27 20              BEQ   LNAB1S     PASS 1
                        *
2D78 F6 2399 R          LDA B OPCD       PASS 2,LOAD PARTIAL OPCODE
2D7B B6 2D0D R          LDA A ABR        A OR B ?
2D7E 27 0F              BEQ   LNAB1O     NEITHER
                        *
2D80 81 42              CMP A #'B        "B" ?
2D82 27 05              BEQ   LNAB1B     YES
                        *
2D84 FA 2D10 R          ORA B ORBYA      A FORM COMPLETE OPCODE
2D87 20 03              BRA   LNAB1C
                        *
2D89 FA 2D11 R LNAB1B   ORA B ORBYB      B FORM COMPLETE OPCODE
2D8C F7 2399 R LNAB1C   STA B OPCD       SAVE
2D8F BD 3E55 R LNAB1O   JSR   OUTBIN     OUTPUT OPCODE
2D92 7C 2397 R          INC   MCOUNT     MCOUNT,=1
2D95 BD 2372 R          JSR   PRINTL     PRINT A LINE OF LISTING
2D98 7C 06F9 R LNAB1S   INC   LCN        LCN=1
2D9B 39                 RTS              RETURN
                        *
                        *
                        * LCNAB2 2 BYTE REGISTER(A,B),INDEXED,
                        *  DIRECT, AND IMMEDIATE TYPE INSTRUCTIONS
                        *
2D9C 7D 0575 R LCNAB2   TST   PASS       PASS ?
2D9F 27 3C              BEQ   LCN2B      PASS 1
                        *
2DA1 F6 2399 R          LDA B OPCD       PASS 2,GET PARTIAL OPCODE
2DA4 B6 2D0D R          LDA A ABR        A OR B ?
2DA7 27 22              BEQ   LCN2A      NEITHER
                        *
2DA9 81 42              CMP A #'B        "B" ?
2DAB 27 05              BEQ   LNB2       YES
                        *
2DAD FA 2D10 R          ORA B ORBYA      A, FORM COMPLETE OPCODE
2DB0 20 03              BRA   LNAB2S
                        *
2DB2 FA 2D11 R LNB2     ORA B ORBYB      B, FORM COMPLETE OPCODE
2DB5 F7 2399 R LNAB2S   STA B OPCD       SAVE
2DB8 20 11              BRA   LCN2A      FINISH UP
                        *
                        *
                        * LCN2 2 BYTE INDEXED,DIRECT,AND IMMEDIATE TYPE
                        *  INSTRUCTIONS
                        *
2DBA 7D 0575 R LCN2     TST   PASS       PASS ?
2DBD 27 1E              BEQ   LCN2B      PASS 1
                        *
2DBF 7F 0579 R          CLR   REFLG
2DC2 F6 2399 R          LDA B OPCD       PASS 2,GET PARTIAL OPCODE
2DC5 FA 2D10 R          ORA B ORBYA      FORM COMPLETE OPCODE
2DC8 F7 2399 R          STA B OPCD       SAVE
                        *
2DCB BD 3E55 R LCN2A    JSR   OUTBIN     OUTPUT OPCODE
2DCE F6 239B R          LDA B ADR2       GET ADDRESS PART OF MC
2DD1 BD 3E55 R          JSR   OUTBIN     OUTPUT IT
2DD4 7C 2397 R          INC   MCOUNT     MCOUNT:=2
2DD7 7C 2397 R          INC   MCOUNT
2DDA BD 2372 R          JSR   PRINTL     PRINT A LINE OF LISTING
                        *
2DDD 7C 06F9 R LCN2B    INC   LCN        LCN,=2
2DE0 7C 06F9 R          INC   LCN
2DE3 39                 RTS              RETURN
                        *
                        *
                        * LCNAB3 3 BYTE REGISTER(A,B),EXTENDED TYPE
                        *   INSTRUCTIONS
                        *
2DE4 7D 0575 R LCNAB3   TST   PASS       PASS ?
2DE7 27 55              BEQ   LCN3B      PASS 1
                        *
2DF9 F6 2399 R          LDA B OPCD       PASS 2 GET PARTIAL OPCODE
2DEC B6 2D0D R          LDA A ABR        A OR B ?
2DEF 27 1F              BEQ   LCN3A      NEITHER
                        *
2DF1 81 42              CMP A #'B        "B" ?
2DF3 27 05              BEQ   LNB3       YES
                        *
2DF5 FA 2D10 R          ORA B ORBYA      A, FORM COMPLETE OPCODE
2DF8 20 03              BRA   LNAB3S
                        *
```

```
2DFA FA 2D11 R LN03    ORA B ORBYB    B, FORM COMPLETE OPCODE
2DFD F7 2399 R LNAB3S  STA B OPCD     SAVE
2E00 20 0E             BRA   LCN3A    FINISH UP
                 *
                 *
                 * LCN3 3 BYTE EXTENDED AND IMMEDIATE TYPE
                 *  INSTRUCTIONS
                 *
2E02 7D 0575 R LCN3    TST   PASS     PASS ?
2E05 27 37             BEQ   LCN3B    PASS 1
                 *
2E07 F6 2399 R         LDA B OPCD     GET PARTIAL OPCODE
2E0A FA 2D10 R         ORA B ORBYA    FORM COMPLETE OPCODE
2E0D F7 2399 R         STA B OPCD     SAVE
                 *
2E10 BD 3E55 R LCN3A   JSR   OUTBIN   OUTPUT OPCODE
2E13 F6 239A R         LDA B ADR1     OUTPUT THE REST OF THE MC
2E16 BD 3E55 R         JSR   OUTBIN
2E19 F6 239B R         LDA B ADR2
2E1C BD 3E55 R         JSR   OUTBIN
2E1F 7D 057A R         TST   CMNFLG   COMMON?
2E22 27 04             BEQ   *+6      NO
                 *
2E24 C6 4D             LDA B #'M
2E26 20 07             BRA   *+9
                 *
2E28 7D 0579 R         TST   RELFLG   RELOC?
2E2B 27 05             BEQ   *+7      NO
                 *
2E2D C6 52             LDA B #'R
                 *
2E2F BD 3F69 R         JSR   OUTBNR
                 *
2E32 7C 2397 R         INC   MCOUNT   MCOUNT =3
2E35 7C 2397 R         INC   MCOUNT
2E38 7C 2397 R         INC   MCOUNT
2E3B BD 2372 R         JSR   PRINTL   PRINT A LINE OF LISTING
                 *
2E3E 7C 06F9 R LCN3B   INC   LCN      LCN =3
2E41 7C 06F9 R         INC   LCN
2E44 7C 06F9 R         INC   LCN
2E47 39                RTS            RETURN
                 *
                 *
                 * LCLCN LC =LC+LCN
                 *
2E48 B6 0574 R LCLCN   LDA A LC+1
2E4B F6 0573 R         LDA B LC
2E4E BB 06F9 R         ADD A LCN      ADD LCN
2E51 C9 00             ADC B #$00
2E53 B7 0574 R         STA A LC+1     SAVE LC
2E56 F7 0573 R         STA B LC
2E59 39                RTS            RETURN
                 *
                 *
                 *
                 * POCMN: ALLOCATE COMMON AREAS
                 *
2E5A BD 2D12 R POCMN   JSR ADRINT     INITIALIZE FLAGS
2E5D BD 3E41 R         JSR LBLCK      CHECK FOR LABEL
                 *
2E60 BD 172F R         JSR NXTOK      GET SYMBOL NAME
2E63 C1 01             CMP B #1       VALID NAME?
2E65 27 08             BEQ POCMN2     YES
                 *
2E67 CE 0216   POCMN0  LDX #$0216     ERROR
2E6A BD 24E8 R POCMN1  JSR PRINTE
2E6D 20 5D             BRA POCMN4
                 *
2E6F 7D 0575 R POCMN2  TST PASS       PASS 1?
2E72 26 41             BNE POCMN3     NO
                 *
2E74 BD 19FA R         JSR STOSYM     ENTER NAME INTO SYMTAB
2E77 FE 06F6 R         LDX SYMPTR
2E7A FF 2ED2 R         STX CMNXS      SAVE POINTER
2E7D BD 172F R         JSR NXTOK      GET DELIMITER
2E80 C1 2C             CMP B #',      COMMA?
2E82 26 E3             BNE POCMN0     NO, ERROR
                 *
2E84 BD 172F R         JSR NXTOK      GET LENGTH
2E87 BD 20EA R         JSR NSEVL      EVALUATE
2E8A C1 FF             CMP B #$FF     O.K ?
2E8C 27 DC             BEQ POCMN1     NO, ERROR
                 *
2E8E FE 2ED2 R         LDX CMNXS      RESTORE POINTER
2E91 86 BF             LDA A #$BF     TURN OFF REL. BIT
2E93 A4 08             AND A 8,X
2E95 8A 10             ORA A #$10     TURN ON COM  BIT
2E97 A7 08             STA A 8,X
2E99 B6 0705 R         LDA A CMNLC    GET COMMON LC
2E9C A7 06             STA A 6,X
2E9E B6 0706 R         LDA A CMNLC+1
2EA1 A7 07             STA A 7,X
                 *
                 * CMNLC =CMNLC+[ADR1,ADR2]
                 *
2EA3 B6 239B R         LDA A ADR2
2EA6 F6 239A R         LDA B ADR1
2EA9 BB 0706 R         ADD A CMNLC+1
2EAC F9 0705 R         ADC B CMNLC
2EAF B7 0706 R         STA A CMNLC+1
2EB2 F7 0705 R         STA B CMNLC
                 *
2EB5 BD 1A5C R POCMN3  JSR LKPSYM     GET SYMBOL
2EB8 FE 06F6 R         LDX SYMPTR
2EBB EE 06             LDX 6,X
2EBD FF 239A R         STX ADR1       STORE COMMON ADDRESS
2EC0 73 2398 R         COM POP
2EC3 73 057A R         COM CMNFLG
2EC6 7C 2397 R         INC MCOUNT
2EC9 7C 2397 R         INC MCOUNT
                 *
2ECC BD 2372 R POCMN4  JSR PRINTL     PRINT LINE
2ECF 7E 0937 R         JMP MAIN1      DONE!
                 *
2ED2 0002      CMNXS   RMB 2
                 *
                 *
                 *
                 * POEND  PROCESS END PSEUDOP
                 *
2ED4 BD 2D12 R POEND   JSR ADRINT     INIT FLAGS
```

```
2ED7 BD 3E41 R          JSR LBLCK       CHECK FOR A LABEL
2EDA 7D 0575 R POENDO   TST PASS        PASS ?
2EDD 26 0F              BNE POFND1      PASS2
                  *
2EDF FE 0573 R          LDX LC          PASS1
2EE2 FF 0571 R          STX TSTPH       TSTPH.=LC
2EE5 73 0575 R          COM PASS        PASS:=PASS2
                  *
2EE8 BD 3EDE R          JSR REWIND      REWIND INPUT FILE
2EEB 7E 0902 R          JMP PASS2       EXECUTE PASS2
                  *
2EEE 7D 06F3 R POENDI   TST CFLAG       COMPILING?
2EF1 27 1A              BEQ ENDP1       NO
                  *
2EF3 FF 06F4 R          LDX CULINE
2EF6 BD 130B R          JSR STR1        WRITE "END" TO TAPE
2EF9 FE 0573 R          LDX LC
2EFC FF 0571 R          STX TSTPH       NO PHASING
                  *
2EFF FE 389D R POENDZ   LDX BLKCNT      CHECK FOR UNCLOSED BLOCKS
2F02 8C 389C R          CPX #BLKEND
2F05 2C 06              BGE ENDP1
                  *
2F07 CE 0306            LDX #$0306      ERROR MESSAGE
2F0A BD 24E8 R          JSR PRINTE
                  *
2F0D FE 0571 R ENDP1    LDX TSTPH       PHASING ERRORS?
2F10 BC 0573 R          CPX LC
2F13 27 06              BEQ ENDP2       NO
                  *
2F15 CE 0220            LDX #$0220
2F18 BD 24E8 R          JSR PRINTE      PRINT ERROR
                  *
2F1B B6 06F8 R ENDP2    LDA A OPTNS
2F1E 85 80              BIT A #$80      LISTING?
2F20 26 0B              BNE ENDP3       NO
                  *
2F22 7D 06F3 R          TST CFLAG       COMPILING?
2F25 26 03              BNE *+5         YES
                  *
2F27 BD 2372 R          JSR PRINTL
2F2A BD 238A R          JSR SPACER      FINISH UP LISTING
                  *
2F2D B6 06F8 R ENDP3    LDA A OPTNS
2F30 85 20              BIT A #$20      LIST SYMTAB?
2F32 27 03              BEQ SORT1       YES
                  *
                  * LIST SYMBOL TABLE IN ALPHABETIC ORDER
2F34 7E 3049 R          JMP ENDP6       NO
                  *
2F37 CE 303B R SORT1    LDX #ZZZ        INITIALIZE SORT
2F3A FF 3044 R          STX CBLOCK
2F3D 7F 3048 R          CLR SORTF       CLEAR SORT FLAG
2F40 B6 06F8 R          LDA A OPTNS
2F43 84 7F              AND A #$7F      FORCE 'LIST' ON
2F45 B7 06F8 R          STA A OPTNS
2F48 FE 008A R          LDX SYMTAB      POINT TO TABLE
2F4B FF 06F6 R          STX SYMPTR
2F4E 20 1D              BRA SORT2A
                  *
2F50 BD 1A99 R SORT2    JSR SYNMOD      INCREMENT 9 TIMES
2F53 BC 008A R          CPX SYMTAB      AT TABLE END?
2F56 26 08              BNE SORT2A      NO, KEEP LOOKING
                  *
2F58 7D 3048 R          TST SORTF       FOUND ENTRY?
2F5B 26 3A              BNE SORT5       IF SO, PRINT ENTRY
                  *
2F5D 7E 3049 R          JMP ENDP6       IF NOT, DONE
                  *
2F60 E6 00      SORT2A  LDA B 0,X
2F62 C1 20              CMP B #$20      EMPTY ENTRY?
2F64 27 EA              BEQ SORT2       GET NEXT ENTRY
                  *
2F66 E6 08              LDA B 8,X       USED ENTRY?
2F68 C1 FF              CMP B #$FF
2F6A 27 E4              BEQ SORT2       GET NEXT ENTRY
                  *
2F6C 86 FF              LDA A #$FF
2F6E B7 3048 R          STA A SORTF     SET SORT FLAG
                  * COMPARE ENTRY AT CBLOCK WITH NEW ENTRY
2F71 FF 3046 R          STX CXS2        SET UP FOR COMPARISON
2F74 FF 19E7 R          STX PSTNG2
2F77 FE 3044 R          LDX CBLOCK
2F7A FF 19E5 R          STX PSTNG1
2F7D C6 06              LDA B #6
2F7F F7 19E9 R          STA B PCOUNT
2F82 CE 19E5 R          LDX #PSTNG1
2F85 BD 0DD7 R          JSR COMPAR
2F88 22 05              BHI SORT4       NEED SWITCH?
                  *
2F8A FE 3046 R          LDX CXS2
2F8D 20 C1              BRA SORT2
                  *
2F8F FF 3046 R SORT4    LDX CXS2        NEW CBLOCK POINTER
2F92 FF 3044 R          STX CBLOCK
2F95 20 B9              BRA SORT2
                  *
2F97 C6 06      SORT5   LDA B #6        SET UP FOR PRINT
2F99 FE 3044 R          LDX CBLOCK
                  *
2F9C A6 00      ENDP4   LDA A 0,X       GET CHAR
2F9E BD 3EAE R          JSR OUTCHR      PRINT
2FA1 08                 INX             POINT TO NEXT CHAR
2FA2 5A                 DEC B           DECREMENT COUNT
2FA3 26 F7              BNE ENDP4       NOT DONE
                  *
2FA5 BD 3EA4 R          JSR OUTS        PRINT BLANK
2FA8 FF 0E03 R          STX XSAV
2FAB A6 00              LDA A 0,X       GET VALUE
2FAD E6 01              LDA B 1,X
2FAF CE 239C R          LDX #LINEN
2FB2 BD 2487 R          JSR CVBTD       MAKE DECIMAL
2FB5 CE 239C R          LDX #LINEN
2FB8 BD 3E8F R          JSR PDATA1      PRINT FIVE DECIMAL DIGITS
2FBB FE 0E03 R          LDX XSAV        RESTORE X
2FBE BD 3EA0 R          JSR OUT4HS      PRINT 4 HEX DIGITS
2FC1 FF 30BE R          STX ENDXS       SAVE INDEX
2FC4 E6 00              LDA B 0,X
2FC6 C5 80              BIT B #$80      PARAMETER?
2FC8 27 05              BEQ *+7         NO
                  *
2FCA 86 50              LDA A #'P       PRINT "P"
```

```
2FCC BD 3EAE R        JSR OUTCHR
                *
2FCF C5 40            BIT B #$40      RELOC ?
2FD1 27 0C            BEQ *+14        NO
                *
2FD3 86 52            LDA A #'R       LOAD "R"
2FD5 7D 06F3 R        TST CFLAG       COMPILING?
2FD8 27 02            BEQ *+4         NO
                *
2FDA 86 53            LDA A #'S       STRING MODE
2FDC BD 3EAE R        JSR OUTCHR      PRINT IT
                *
2FDF C5 20            BIT B #$20      MACRO NAME?
2FE1 27 05            BEQ *+7         NO
                *
2FE3 86 4D            LDA A #'M       PRINT "M"
2FE5 BD 3EAE R        JSR OUTCHR
                *
2FE8 C5 10            BIT B #$10      COMMON?
2FEA 27 05            BEQ *+7         NO
                *
2FEC 86 43            LDA A #'C       PRINT "C"
2FEE BD 3EAE R        JSR OUTCHR
                *
2FF1 C5 08            BIT B #$08      EXTERNAL?
2FF3 27 0A            BEQ *+12        NO
                *
2FF5 7D 06F3 R        TST CFLAG       COMPILING?
2FF8 26 05            BNE *+7         YES
                *
2FFA 86 58            LDA A #'X       PRINT "X"
2FFC BD 3EAE R        JSR OUTCHR
                *
2FFF C5 04            BIT B #$04      ENTRY?
3001 27 05            BEQ *+7         NO
                *
3003 86 4E            LDA A #'N       PRINT "N"
3005 BD 3EAE R        JSR OUTCHR
                *
3008 7D 06F3 R        TST CFLAG       COMPILING?
300B 27 12            BEQ ENDP4B      NO, SKIP ARRAYS
                *
300D C5 03            BIT B #$03      AN ARRAY?
300F 27 05            BEQ *+7         NO
                *
3011 86 44            LDA A #'D       LOAD "D"
3013 BD 3EAE R        JSR OUTCHR      PRINT IT
3016 C5 08            BIT B #$08      FLOATING?
3018 27 05            BEQ ENDP4B      NO
                *
301A 86 46            LDA A #'F       LOAD "F"
301C BD 3EAE R        JSR OUTCHR
                *
301F A6 00      ENDP4B LDA A 0,X      REDEFINED?
3021 2A 0B            BPL ENDP5       NO
                *
3023 7D 06F3 R        TST CFLAG       COMPILING?
3026 26 06            BNE ENDP5       IF SO, SKIP ERROR
                *
3028 CE 307B R        LDX #REDEF      PRINT ERROR MESSAGE,
302B BD 3F8F R        JSR PDATA1
                *
302E FE 30BE R  ENDP5 LDX ENDXS
3031 C6 FF            LDA B #$FF      MARK ENTRY "USED"
3033 E7 00            STA B 0,X
3035 BD 3E84 R        JSR CRLF        PRINT CR,LF
3038 7E 2F37 R        JMP SORT1       LOOP AGAIN
                *
                *
303B 5B         ZZZ   FCC 'TTTTTT'    DUMMY SYMBOL ENTRY
3041 0000             FDB 0000
3043 00               FCB 0
3044 0002       CBLOCK RMB 2          POINTER STORAGE
3046 0002       CXS2  RMB 2
3048 0001       SORTF RMB 1           SORT FLAG
                *
3049 CE 3096 R  ENDP6 LDX   #ENDMB    PRINT FINAL MESSAGE
304C B6 0703 R        LDA A ECOUNT
304F F6 0704 R        LDA B ECOUNT+1
3052 BD 2487 R        JSR   CVBTD     CONVERT # OF ERRORS TO ASCII
3055 B6 0705 R        LDA A CMNLC     CONVERT COMMON LENGTH
3058 F6 0706 R        LDA B CMNLC+1
305B CE 30B5 R        LDX   #CMNSG
305E BD 2487 R        JSR   CVBTD
3061 BD 238A R        JSR SPACER
3064 CE 3086 R        LDX   #ENDMA
3067 BD 3E8F R        JSR   PDATA1    PRINT IT
                *
306A B6 06F8 R        LDA A OPTNS     GET OPTIONS
306D 43               COM A
306E 85 48            BIT A #$48      OBJECT OR XREF?
3070 27 03            BEQ   ENDP7     NO
                *
3072 BD 3ECF R        JSR   CLOSO     CLOSE OUTPUT FILE
                *
3075 BD 3ECC R  ENDP7 JSR   CLOSI     CLOSE INPUT FILE
3078 7E 0749 R        JMP   OPC2A     RETURN TO OPTIONS
                *
307B 20         REDEF FCC ' REDEFINED'
3085 04               FCB   $04       EOT
                *
3086 0D0A       ENDMA FDB $0D0A       CR,LF
3088 0D0A             FDB $0D0A       CR,LF
308A 54               FCC 'THERE WERE  '
3096 0005       ENDMB RMB   5
3098 20               FCC ' ERRORS'
30A2 0D0A             FDB $0D0A       CR,LF
30A4 0D0A             FDB $0D0A       CR,LF
30A6 43               FCC 'COMMON LENGTH= '
30B5 0005       CMNSG RMB 5
30BA 2E               FCC '.'
30BB 0D0A             FDB   $0D0A     CR,LF
30BD 04               FCB   $04       EOT
                *
30BE 0002       ENDXS RMB   2         SAVE AREA (SYMTAB POINTER)
                *
                *
                * POENT  PROCESS "ENTRY" PSEUDOP
                *        DEFINES AN ENTRY POINT FOR
                *        REFERENCE BY OTHER MODULES
                *
30C0 BD 2D12 R  POENT  JSR ADRIN1
```

```
30C3 BD 3E41 R          JSR LBLCK       CHECK FOR A LABEL
                *
30C6 BD 172F R          JSR NXTOK
30C9 C1 01              CMP B #1        ENTRY NAME OK?
30CB 27 08              BEQ POENT1      YES
                *
30CD CE 0216            LDX #$0216      NO
30D0 BD 24E8 R          JSR PRINTE
30D3 20 38              BRA POENT3
                *
30D5 7D 0575 R POENT1   TST PASS         PASS?
30D8 27 42              BEQ POENT4      PASS 1
                *
30DA BD 1A5C R          JSR LKPSYM      GET ENTRY ADDRESS
30DD C1 FF              CMP B #$FF      IN SYMTAB?
30DF 26 08              BNE POENT2      YES
                *
30E1 CE 0211            LDX #$0211      NO, ERROR
30E4 BD 24E8 R          JSR PRINTE
30E7 20 24              BRA POENT3
                *
30E9 FF 239A R POENT2   STX ADR1        SAVE ENTRY ADDRESS
30EC FE 06F6 R          LDX SYMPTR      POINT TO ENTRY
30EF A6 08              LDA A 8,X       GET INFO BYTE
30F1 8A 04              ORA A #$04      TURN ON ENT BIT
30F3 A7 08              STA A 8,X
                *
30F5 8D 28              BSR PBLOCK      OUTPUT LABEL
30F7 F6 239A R          LDA B ADR1      OUTPUT ENTRY ADDRESS
30FA BD 3E55 R          JSR OUTBIN
30FD F6 239B R          LDA B ADR2
3100 BD 3E55 R          JSR OUTBIN
3103 C6 52              LDA B #'R       "RELOCATABLE"
3105 BD 3E69 R          JSR OUTBNR
3108 C6 4E              LDA B #'N       "ENT"
310A BD 3E69 R          JSR OUTBNR
                *
310D 73 2398 R POENT3   COM POP
3110 73 057C R          COM ENTFLG
3113 7C 2397 R          INC MCOUNT
3116 7C 2397 R          INC MCOUNT
3119 BD 2372 R          JSR PRINTL
                *
311C 7E 0937 R POENT4   JMP MAIN1       ALL DONE
                *
                * PBLOCK: ROUTINE TO WRITE LABEL ENTRY SYMBOL
                *
311F FE 06F6 R PBLOCK   LDX SYMPTR      POINT TO ENTRY SYMBOL
3122 86 06              LDA A #6
                *
3124 E6 00     PBLK2    LDA B 0,X       GET A CHAR
3126 36                 PSH A
3127 FF 3136 R          STX PBXS
312A BD 3E55 R          JSR OUTBIN
312D 32                 PUL A
312E FE 3136 R          LDX PBXS
3131 08                 INX
3132 4A                 DEC A           ALL DONE?
3133 26 EF              BNE PBLK2       NO
                *
3135 39                 RTS
                *
3136 0002      PBXS     RMB 2
                *
                *
                * POEQU  PROCESS EQU PSEUDOP
                *
3138 BD 2D12 R POEQU    JSR   ADRINT    INIT ADDRESS FIELD FLAGS
313B FE 06F6 R          LDX   SYMPTR
313E FF 318F R          STX   EQUXS     SAVE SYMPTR
3141 7D 0577 R          TST   LBFLG     LABEL?
3144 27 43              BEQ   EQUE      NO
                *
3146 BD 172F R EQUB     JSR   NXTOK     GET NEXT TOKEN
3149 C1 0D              CMP B #$0D      EOL?
314B 26 08              BNE   EQUC      NO
                *
314D CE 0216            LDX   #$0216    ERROR
3150 BD 24E8 R EQUA     JSR   PRINTE    PRINT MESSAGE
3153 20 34              BRA   EQUE
                *
3155 BD 20EA R EQUC     JSR   NSEVL     EVALUATE OPERAND
3158 C1 FF              CMP B #$FF      ERRORS?
315A 27 F4              BEQ   EQUA      YES
                *
315C 7D 0575 R          TST   PASS      PASS?
315F 26 1F              BNE   EQUD      PASS2
                *
3161 FE 318F R          LDX   EQUXS
3164 A6 08              LDA A 8,X       GET INFO BYTE
3166 7D 0579 R          TST   RELFLG    RELOC ?
3169 26 09              BNE   EQUF      YES
                *
316B 84 BF              AND A #$BF      NO, TURN OFF IN INFO BYTE
                *
316D 7D 057A R          TST   CMNFLG    COMMON?
3170 27 02              BEQ   EQUF      NO
                *
3172 8A 10              ORA A #$10      YES TURN ON IN INFO BYTE
                *
3174 A7 08     EQUF     STA A 8,X
3176 B6 239B R          LDA A ADR2      STORE VALUE
3179 A7 07              STA A 7,X
317B B6 239A R          LDA A ADR1
317E A7 06              STA A 6,X
                *
3180 73 2398 R EQUD     COM   POP       SET PSEUDOP FLAG
3183 7C 2397 R          INC   MCOUNT    MCOUNT:=2
3186 7C 2397 R          INC   MCOUNT
                *
3189 BD 2372 R EQUE     JSR   PRINTL    PRINT A LINE OF LISTING
318C 7E 0937 R          JMP   MAIN1     RETURN TO MAIN LOOP
                *
318F 0002      EQUXS    RMB   2         SAVE AREA
                *
                *
                * POEXT: PROCESS "EXTERNAL" PSEUDO-OP
                *        MAKE EXTERNALLY-DEFINED VALUE
                *        AVAILABLE TO MODULE
                *
3191 BD 2D12 R POEXT    JSR ADRINT          INIT. ADDRESS FLAGS
3194 BD 3E41 R          JSR LBLCK
```

```
                        *
3197 BD 172F R          JSR NXTOK       GET EXT ENTRY NAME
319A C1 01              CMP B #1        OK?
319C 27 0B              BEQ POEXT1      YES
                        *
319E CE 0216            LDX #$0216      NO, ERROR
31A1 BD 24E8 R          JSR PRINTE
31A4 BD 2372 R          JSR PRINTL
31A7 20 47              BRA POEXT4
                        *
31A9 7C 06F9 R POEXT1   INC LCN
31AC 7C 06F9 R          INC LCN         3 BYTES OF CODE
31AF 7C 06F9 R          INC LCN
                        *
31B2 7D 0575 R          TST PASS
31B5 26 0E              BNE POEXT2      PASS 2
                        *
31B7 BD 19FA R          JSR STOSYM      PUT NAME IN SYMTAB
31BA FE 06F6 R          LDX SYMPTR
31BD A6 08              LDA A 8,X
31BF 8A 08              ORA A #$08      SET EXT BIT
31C1 A7 08              STA A 8,X
31C3 20 28              BRA POEXT3
                        *
31C5 C6 7E      POEXT2  LDA B #$7E      "JMP"
31C7 F7 2399 R          STA B OPCD
31CA BD 3E55 R          JSR OUTBIN
31CD BD 1A5C R          JSR LKPSYM      LOOK UP ENTRY NAME
31D0 BD 311F R          JSR PBLOCK      OUTPUT NAME
31D3 C6 58              LDA B #'X       "EXT"
31D5 BD 3E69 R          JSR OUTBNR
31D8 7F 239A R          CLR ADR1
31DB 7F 2398 R          CLR ADR2        CODE=JMP 0000
31DE 7C 2397 R          INC MCOUNT
31E1 7C 2397 R          INC MCOUNT
31E4 7C 2397 R          INC MCOUNT
31E7 73 057B R          COM EXTFLG
31EA BD 2372 R          JSR PRINTL
                        *
31ED BD 2E48 R POEXT3   JSR LCLCN
31F0 7E 0937 R POEXT4   JMP MAIN1       ALL DONE
                        *
                        *
                        * POFCB: PROCESS FCB PSEUDOP
                        *
31F3 BD 2D12 R POFCB    JSR   ADRINT    INIT ADDRESS FIELD FLAGS
31F6 BD 172F R          JSR   NXTOK     GET NEXT TOKEN
31F9 C1 0D              CMP B #$0D      EOL?
31FB 26 08              BNE   FCBB      NO
                        *
31FD CE 0216    FCBA    LDX   #$0216    ERROR
3200 BD 24E8 R          JSR   PRINTE    PRINT
3203 20 1A              BRA   FCBC      FINISH UP
                        *
3205 BD 20EA R FCBB     JSR   NSEVL     EVALUATE OPERAND
3208 BD 2D60 R          JSR   P2ERR     PRINT PASS  ERRORS
320B 7C 06F9 R          INC   LCN       LCN:=1
320E 7D 0575 R          TST   PASS      PASS?
3211 27 0F              BEQ   FCBD      PASS1
                        *
3213 F6 2398 R          LDA B ADR2      PASS2, OUTPUT MC
3216 BD 3E55 R          JSR   OUTBIN
3219 7C 2397 R          INC   MCOUNT    MCOUNT.=1
321C 7C 2398 R          INC   POP       POP.=1
                        *
321F BD 2372 R FCBC     JSR   PRINTL    PRINT A LINE OF LISTING
3222 BD 2E48 R FCBD     JSR   LCLCN     LC.=LC+LCN
3225 7E 0937 R          JMP   MAIN1     RETURN TO MAIN LOOP
                        *
                        *
                        * POFCC: PROCESS FCC PSEUDOP
                        *
3228 BD 2D12 R POFCC    JSR   ADRINT    INIT ADDRESS FIELD FLAGS
322B BD 172F R          JSR   NXTOK     GET NEXT TOKEN
322E C1 27              CMP B #$27      QUOTE ?
3230 27 08              BEQ   FCCB      YES
                        *
3232 CE 0204    FCCA    LDX   #$0204    ERROR
3235 BD 24E8 R          JSR   PRINTE    PRINT
3238 20 60              BRA   FCCG      FINISH UP
                        *
323A FE 06F1 R FCCB     LDX   CUCHAR    GET CURRENT CHAR
323D E6 00              LDA B 0,X
323F C1 0D              CMP B #$0D      EOL ?
3241 27 EF              BEQ   FCCA      YES
                        *
3243 F7 239B R          STA B ADR2      NO,SAVE CHAR
                        *
3246 7D 0575 R FCCC     TST   PASS      PASS ?
3249 27 12              BEQ   FCCD      PASS1
                        *
324B 7D 06F3 R          TST   CFLAG     COMPILING?
324E 27 0A              BEQ   FCC1      NO
                        *
3250 37                 PSH B
3251 BD 0DA3 R          JSR   TPCHAR    YES, CHAR TO OBJECT AND LIST
3254 32                 PUL A
3255 BD 0D98 R          JSR   WRTCHR
3258 20 03              BRA   FCCD
                        *
325A BD 3E55 R FCC1     JSR   OUTBIN    OUTPUT MC
325D FE 06F1 R FCCD     LDX   CUCHAR    CUCHAR =CUCHAR+1
3260 08                 INX
3261 FF 06F1 R          STX   CUCHAR
3264 7C 06F9 R          INC   LCN       LCN =LCN+1
3267 E6 00              LDA B 0,X       GET CHAR
3269 C1 27              CMP B #$27      QUOTE?
326B 27 06              BEQ   FCCE      YES
                        *
326D C1 0D              CMP B #$0D      EOL?
326F 26 D5              BNE   FCCC      NO
3271 20 1B              BRA   FCCF      YES
                        *
3273 E6 01      FCCE    LDA B 1,X       GET NEXT CHAR
3275 C1 27              CMP B #$27      TWO QUOTES ?
3277 26 15              BNE   FCCF      NO
                        *
3279 7D 06F3 R          TST   CFLAG     COMPILING?
327C 27 0A              BEQ   FCC2      NO
                        *
327E 37                 PSH B
327F BD 0DA3 R          JSR   TPCHAR    PRINT QUOTE
```

```
3282 32                PUL A
3283 36                PSH A
3284 BD 0D98 R         JSR   WRTCHR
3287 33                PUL B
                 *
3288 08          FCC2  INX
3289 FF 06F1 R         STX   CUCHAR     CUCHAR:=CUCHAR+1
328C 20 B8             BRA   FCCC
                 *
328E 7D 06F3 R   FCCF  TST   CFLAG      COMPILING?
3291 27 01             BEQ   *+3        NO
                 *
3293 39                RTS
                 *
3294 7C 2398 R         INC   POP        POP:=1
3297 7C 2397 R         INC   MCOUNT     MCOUNT:=1
                 *
329A BD 2372 R   FCCG  JSR   PRINTL     PRINT LINE OF LISTING
329D BD 2E48 R         JSR   LCLCN      LC:=LC+LCN
32A0 7E 0937 R         JMP   MAIN1      RETURN TO MAIN LOOP
                 *
                 *
                 * POFDB: PROCESS FDB PSEUDOP
                 *
32A3 BD 2D12 R   POFDB JSR   ADRINT     INIT ADDRESS FIELD FLAGS
32A6 BD 172F R         JSR   NXTOK      GET NEXT TOKEN
32A9 C1 0D             CMP B #$0D       EOL?
32AB 26 08             BNE   FDBB       NO
                 *
32AD CE 0216     FDBA  LDX   #$0216     ERROR
32B0 BD 24E8 R         JSR   PRINTE     PRINT
32B3 20 39             BRA   FDBC       FINISH UP
                 *
32B5 BD 20EA R   FDBB  JSR   NSEVL      EVALUATE OPERAND
32B8 BD 2D60 R         JSR   P2ERR      PRINT PASS 2 ERRORS
                 *
32BB 7C 06F9 R         INC   LCN        LCN:=2
32BE 7C 06F9 R         INC   LCN
32C1 7D 0575 R         TST   PASS       PASS?
32C4 27 2B             BEQ   FDBD       PASS1
                 *
32C6 F6 239A R         LDA B ADR1       OUTPUT MC
32C9 BD 3E55 R         JSR   OUTBIN
32CC F6 239B R         LDA B ADR2
32CF BD 3E55 R         JSR   OUTBIN
32D2 7D 0579 R         TST   RELFLG     RELOC ?
32D5 27 04             BEQ   FDBF       NO
                 *
32D7 C6 52             LDA B #'R        YES
32D9 20 07             BRA   FDBG
                 *
32DB 7D 057A R   FDBF  TST   CMNFLG     COMMON?
32DE 27 05             BEQ   FDBE       NO
                 *
32E0 C6 4D             LDA B #'M        YES
                 *
32E2 BD 3E69 R   FDBG  JSR   OUTBNR     OUTPUT "R" OR "M"
                 *
32E5 7C 2397 R   FDBE  INC   MCOUNT     MCOUNT.=2
32E8 7C 2397 R         INC   MCOUNT
32EB 73 2398 R         COM   POP        POP.=FF
                 *
32EE BD 2372 R   FDBC  JSR   PRINTL     PRINT A LINE OF LISTING
32F1 BD 2E48 R   FDBD  JSR   LCLCN      LC =LC+LCN
32F4 7E 0937 R         JMP   MAIN1      RETURN TO MAIN LOOP
                 *
                 *
                 * POFFP: PROCESS FFP PSEUDOP (FORM FLOATING POINT)
                 *
32F7 BD 2D12 R   POFFP JSR   ADRINT     INIT ADDRESS FIELD FLAGS
32FA 7C 0578 R         INC   FLPFLG     SET FLOATING MODE
32FD BD 172F R         JSR   NXTOK      GET NEXT TOKEN
3300 C1 0F             CMP B #$0F       FLOATING?
3302 26 A9             BNE   FDBA       NO, ERROR AND QUIT
                 *
3304 86 09             LDA A #9         9 BYTES FOR FLOAT
3306 B7 06F9 R         STA A LCN
3309 7D 0575 R         TST   PASS       WHICH PASS?
330C 27 E3             BEQ   FDBD       FINISH PASS1
                 *
330E FE 1988 R         LDX   FTEMP      LOAD FLOATING VALUE
3311 FF 239A R         STX   ADR1       SAVE FOR PRINT
3314 CE 1988 R         LDX   #FTEMP     STARTING ADDRESS
3317 BD 3124 R         JSR   PBLK2      OUTPUT MC
331A 20 C9             BRA   FDBE       FINISH UP
                 *
                 * POFUNC: PROCESS A FUNCTION IN STRUBOL+
                 *
331C BD 2D12 R   POFUNC JSR ADRINT       INIT. FLAGS
331F 7F 3384 R         CLR CMFLG        NO COMMENTS
3322 7F 3385 R         CLR MACERR       NO ERRORS
3325 CE 3B9D R         LDX #LABL+6
3328 BD 13AB R         JSR STR1         OUTPUT DUMMY LINE
332B BD 172F R         JSR NXTOK        GET NEXT TOKEN
332E C1 01             CMP B #$01       VALID NAME?
3330 27 09             BEQ POFNC3       YES
                 *
3332 CE 0340     POFNC2 LDX #$0340       ERROR MESSAGE
3335 BD 24E8 R         JSR PRINTE
3338 7E 0937 R         JMP MAIN1        QUIT
                 *
333B BD 1A5C R   POFNC3 JSR LKPSYM       CHECK SYMBOL TABLE
333E C1 FF             CMP B #$FF       IN TABLE?
3340 26 F0             BNE POFNC2       IF SO, ERROR
                 *
3342 BD 19FA R         JSR STOSYM       PUT IT IN TABLE
3345 BD 1992 R         JSR XREF         HANDLE CROSS-REF
3348 7E 33A7 R         JMP POMAC1       FINISH LIKE MACRO
                 *
                 *
                 * POIFC: PROCESS THE IFC PSEUDOP
                 *
334B BD 2D12 R   POIFC  JSR ADRINT       INIT  FLAGS
334E BD 3E41 R         JSR LBLCK        CHECK FOR LABEL
3351 BD 172F R         JSR NXTOK        GET NEXT TOKEN
3354 C1 0D             CMP B #$0D       EOL?
3356 26 08             BNE POIFB        NO
                 *
3358 CE 0216     POIFA LDX #$0216       YES, ERROR
335B BD 24E8 R         JSR PRINTE
335E 20 1E             BRA POIFE        QUIT
                 *
```

```
3360 BD 20EA R POIFB  JSR NSEVL      EVALUATE OPERAND
3363 C1 FF            CMP B #$FF     ERRORS?
3365 27 F1            BEQ POIFA      YES
                 *
3367 BD 34D8 R        JSR PSHIF      STACK PRESENT CIFFLG
336A 7D 0641 R        TST CIFFLG     ASSEMBLING?
336D 27 0F            BEQ POIFE      NO
                 *
336F FE 239A R        LDX ADR1       OPERAND=0?
3372 26 05            BNE POIFD      NO
                 *
3374 7F 0641 R        CLR CIFFLG     TURN OFF ASSEMBLY
3377 20 05            BRA POIFE
                 *
3379 86 FF       POIFD LDA A #$FF    TURN ON ASSEMBLY
337B B7 0641 R        STA A CIFFLG
                 *
337E BD 2372 R POIFE  JSR PRINTL     LIST LINE
3381 7E 0937 R        JMP MAIN1      DONE!
                 *
                 *
                 * POMAC:  PROCESS MACRO PSEUDOP (ALSO USED BY POFUNC)
                 *
3384 0001        CMFLG  RMB 1        COMMENT FLAG
3385 0001        MACERR RMB 1        ERROR FLAG
                 *
3386 BD 2D12 R POMAC  JSR ADRINI     INIT  FLAGS
3389 BD 2372 R        JSR PRINTL     LIST LINE
338C 7F 3384 R        CLR CMFLG      NO COMMENTS
338F 7F 3385 R        CLR MACERR     NO ERRORS
3392 7D 0575 R        TST PASS       PASS TWO?
3395 26 34            BNE POMAC2     YES
                 *
3397 7D 0577 R        TST LBFLG      LABELED?
339A 26 08            BNE POMAC1     YES, O.K
                 *
339C 73 3385 R        COM MACERR     NO. ERROR
339F CE 0226          LDX #$0226     ERROR MESSAGE
33A2 BD 24E8 R        JSR PRINTE
33A5 20 24            BRA POMAC2
                 *
                 * POFUNC ENTERS HERE
                 *
33A7 FE 06F6 R POMAC1 LDX SYMPTR     POINT TO LABEL  IN SYMTAB
33AA 86 20            LDA A #$20
33AC A7 08            STA A 8,X      SET MACRO FLAG BIT
33AE B6 0626 R        LDA A MACPTR
33B1 A7 06            STA A 6,X      SET MACRO LOCATION
33B3 B6 0627 R        LDA A MACPTR+1
33B6 A7 07            STA A 7,X
33B8 BD 172F R POMAC3 JSR NXTOK      GET NEXT TOKEN
33BB 81 04            CMP A #$04     DELIMITER?
33BD 27 F9            BEQ POMAC3     YES, SCAN AGAIN
                 *
33BF FE 0642 R        LDX DESCRA
33C2 A6 00            LDA A 0,X      GET FIRST CHAR  OF TOKEN
33C4 81 43            CMP A #'C      "C"?
33C6 26 03            BNE *+5        NO
                 *
33C8 73 3384 R        COM CMFLG      YES, INCLUDE COMMENTS
                 *
33CB BD 0B3C R POMAC2 JSR RDLINO     GET NEXT LINE
33CE FE 056F R        LDX LNUM
33D1 08               INX            COUNT LINE
33D2 FF 056F R        STX LNUM
33D5 BD 2372 R        JSR PRINTL     LIST LINE
33D8 FE 06F4 R        LDX CULINE     POINT TO LINE
33DB A6 00            LDA A 0,X      GET FIRST CHAR
33DD 81 2A            CMP A #'*      COMMENT?
33DF 26 0A            BNE POMAC5     NO
                 *
33F1 7D 3384 R        TST CMFLG      WANT COMMENTS?
33E4 27 E5            BEQ POMAC2     NO
                 *
33E6 BD 344C R        JSR MACMOV     YES, SAVE LINE
33E9 20 E0            BRA POMAC2
                 *
33EB 7F 0577 R POMAC5 CLR LBFLG      NO LABEL
33EE FE 06F4 R        LDX CULINE     POINT TO LINE
33F1 A6 00            LDA A 0,X      GET FIRST CHAR
33F3 81 20            CMP A #$20     BLANK?
33F5 27 06            BEQ POMAC6     YES
                 *
33F7 BD 172F R        JSR NXTOK      GET LABEL
33FA 73 0577 R        COM LBFLG      SET LABEL FLAG
                 *
33FD BD 172F R POMAC6 JSR NXTOK      GET MNEMONIC
3400 86 04            LDA A #4
3402 B7 19E9 R        STA A FCOUNT   4 CHAR  COMPARISON
3405 FE 0642 R        LDX DESCRA
3408 FF 19E5 R        STX PSTNG1     COMPARE MNEMONIC WITH "MEND"/"FEND"
340B CE 348D R        LDX #MEND
340E 7D 06F3 R        TST CFLAG      COMPILING?
3411 27 03            BEQ *+5        NO
                 *
3413 CE 3491 R        LDX #FEND
                 *
3416 FF 19E7 R        STX PSTNG2
3419 CE 19E5 R        LDX #PSTNG1
341C BD 0DD7 R        JSR COMPAR     COMPARE
341F 26 0D            BNE POMAC8     NOT "MEND"/"FEND"
                 *
3421 7D 0577 R        TST LBFLG      LABELED?
3424 27 0E            BEQ POMAC7     NO
                 *
3426 CE 0227          LDX #$0227     YES, ERROR
3429 BD 24E8 R        JSR PRINTE
342C 20 06            BRA POMAC7
                 *
342E BD 344C R POMAC8 JSR MACMOV     PUT INTO MACRO TABLE
3431 7E 33CB R        JMP POMAC2
                 *
3434 7D 06F3 R POMAC7 TST CFLAG      COMPILING?
3437 26 05            BNE *+7        YES
                 *
3439 7D 0575 R        TST PASS       PASS TWO?
343C 26 0B            BNE POMACA     YES
                 *
343E 86 17            LDA A #$17
3440 FF 0626 R        LDX MACPTR
3443 A7 00            STA A 0,X      ETB ONTO MACRO DEF
3445 08               INX
```

```
3446 FF 0626 R          STX MACPTR
                 *
3449 7E 0937 R POMACA JMP MAIN1        DONE!
                 *
                 * MOVE MACRO TO MACRO TABLE
                 *
344C 7D 06F3 R MACMOV TST CFLAG        COMPILING?
344F 26 05             BNE *+7         YES
                 *
3451 7D 0575 R         TST PASS        PASS TWO?
3454 26 36             BNE MACMVE      YES
                 *
3456 7D 3385 R         TST MACERR      ERROR?
3459 26 31             BNE MACMVE      YES
                 *
345B FE 06F4 R         LDX CULINE
345E FF 06F1 R         STX CUCHAR      POINT TO LINE
3461 FE 06F1 R MACLOP LDX CUCHAR
3464 A6 00             LDA A 0,X       GET CHAR FROM LINE
3466 08                INX
3467 FF 06F1 R         STX CUCHAR      MOVE POINTER
346A FE 0626 R         LDX MACPTR      POINT TO MACRO TABLE
346D BC 0086 R         CPX MACEND      TABLE FULL?
3470 26 10             BNE MACMV1      NO
                 *
3472 86 0D             LDA A #$0D      FORCE EOL
3474 A7 00             STA A 0,X
3476 08                INX
3477 FF 0626 R         STX MACPTR
347A CE 0228           LDX #$0228      ERROR MESSAGE
347D BD 24E8 R         JSR PRINTE
3480 20 0A             BRA MACMVE
                 *
3482 A7 00      MACMV1 STA A 0,X       STORE CHAR  IN TABLE
3484 08                INX
3485 FF 0626 R         STX MACPTR
3488 81 0D             CMP A #$0D      EOL?
348A 26 D5             BNE MACLOP      NO, KEEP GOING
                 *
348C 39         MACMVE RTS             DONE!
                 *
348D 4D         MEND   FCC 'MEND'
3491 46         FEND   FCC 'FEND'
                 *
                 *
                 * PONAM. PROCESS NAM PSEUDOP
                 *
3495 BD 2D12 R PONAM  JSR   ADRINT     INIT FLAGS
3498 BD 3E41 R        JSR   LBLCK      CHECK FOR A LABEL
349B BD 172F R        JSR   NXTOK      GET NAME
349E C1 01            CMP B #1         VALID NAME?
34A0 27 09            BEQ   PONAM1     YES
                 *
34A2 CE 0216          LDX   #$0216     ERROR
34A5 BD 24E8 R        JSR   PRINTE
34A8 7E 3100 R        JMP   POENT3     FINISH UP
                 *
34AB 7D 0575 R PONAM1 TST   PASS       PASS 1?
34AE 26 06            BNE   PONAM2     NO
                 *
34B0 BD 19FA R        JSR   STOSYM     PUT NAME INTO SYMTAB
34B3 7E 30D5 R        JMP   POENT1     FINISH UP
                 *
34B6 F6 0705 R PONAM2 LDA B CMNLC      PUT OUT COMMON LENGTH
34B9 BD 3E55 R        JSR   OUTBIN
34BC F6 0706 R        LDA B CMNLC+1
34BF BD 3E55 R        JSR   OUTBIN
34C2 C6 50            LDA B #'P        "PROGRAM"
34C4 BD 3E69 R        JSR   OUTBNR
34C7 7E 30D5 R        JMP   POENT1     FINISH UP
                 *
                 *
                 * PONIF: PROCESS NIFC (END-IFC) PSEUDOP
                 *
34CA BD 2D12 R PONIF  JSR ADRINT       INIT. FLAGS
34CD BD 3E41 R        JSR LBLCK        CHECK LABEL
34D0 8D 1D            BSR PULIF        GET LATEST CIFFLG
34D2 BD 2372 R        JSR PRINTL       LIST LINE
34D5 7E 0937 R        JMP MAIN1        DONE!
                 *
                 *
                 * PSHIF: PUSH THE CURRENT CIFFLG ONTO THE STACK
                 *
34D8 07         PSHIF TPA
34D9 36               PSH A            SAVE COND. CODES
34DA 0F               SEI              DISABLE INTERRUPTS
34DB BF 057F R        STS STKSAV       SAVE STACK POINTER
34DE FE 063F R        LDX @IFSTK       POINT TO IFSTACK
34E1 8C 0637 R        CPX #IFSTK-8     STACK FULL?
34E4 27 20            BEQ PSPLER       IF SO, ERROR
                 *
34E6 BE 063F R        LDS @IFSTK
34E9 B6 0641 R        LDA A CIFFLG
34EC 36               PSH A            STACK CIFFLG
34ED 20 1F            BRA PSPLCM
                 *
                 * PULIF. PULL LATEST CIFFLG OFF THE STACK
                 *
34EF 07         PULIF TPA
34F0 36               PSH A            SAVE COND  CODES
34F1 0F               SEI              DISABLE INTERRUPTS
34F2 BF 057F R        STS STKSAV       SAVE STACK POINTER
34F5 FE 063F R        LDX @IFSTK       POINT TO IFSTACK
34F8 8C 063F R        CPX #IFSTK       UNDERFLOW?
34FB 27 09            BEQ PSPLER       IF SO, ERROR
                 *
34FD BE 063F R        LDS @IFSTK
3500 32               PUL A
3501 B7 0641 R        STA A CIFFLG     UNSTACK CIFFLG
3504 20 08            BRA PSPLCM
                 *
3506 CE 0254    PSPLER LDX #$0254      ERROR MESSAGE
3509 BD 24E8 R        JSR PRINTE
350C 20 03            BRA *+5
                 *
350E BF 063F R PSPLCM STS @IFSTK       SAVE IFSTACK POINTER
                 *
3511 BE 057F R        LDS STKSAV       RESTORE STACK POINTER
3514 32               PUL A
3515 06               TAP              RESTORE COND. CODES
3516 39               RTS
                 *
```

```
                        *
                        * POPAG. PROCESS PAG PSEUDOP
                        *
3517 BD 2D12 R POPAG    JSR   ADRINT    INIT FLAGS
351A BD 3E41 R          JSR   LBLCK     CHECK FOR LABEL
351D 7D 0575 R          TST   PASS      PASS?
3520 27 0B              BEQ   PAGEND    PASS 1
                        *
3522 F6 06FC R          LDA B LCOUNT    LCOUNT=0?
3525 27 06              BEQ   PAGEND    YES
                        *
3527 BD 238A R          JSR   SPACER    TOP OF PAGE
352A 7F 06FC R          CLR   LCOUNT    LCOUNT.=0
                        *
352D 7E 0937 R PAGEND   JMP   MAIN1     RETURN TO MAIN LOOP
                        *
                        *
                        * PORMB: PROCESS RMB PSEUDOP
                        *
3530 BD 2D12 R PORMB    JSR   ADRINT    INIT ADDRESS FIELD FLAGS
3533 BD 172F R          JSR   NXTOK     GET NEXT TOKEN
3536 C1 0D              CMP B #$0D      EOL?
3538 26 08              BNE   RMBB      NO
                        *
353A CE 0216            LDX   #$0216    ERROR
353D BD 24E8 R RMBA     JSR   PRINTE    PRINT
3540 20 18              BRA   RMBC      FINISH UP
                        *
3542 BD 20EA R RMBB     JSR   NSEVL     EVALUATE OPERAND
3545 C1 FF              CMP B #$FF      ERRORS?
3547 27 F4              BEQ   RMBA      YES
                        *
3549 7D 0575 R          TST   PASS      PASS?
354C 27 0F              BEQ   RMBD      PASS1
                        *
354E BD 3572 R          JSR   RMBOUT    OUTPUT MC
3551 7C 2397 R          INC   MCOUNT    MCOUNT =2
3554 7C 2397 R          INC   MCOUNT
3557 73 2398 R          COM   POP       SET PSEUDOP FLAG
                        *
355A BD 2372 R RMBC     JSR   PRINTL    PRINT A LINE OF LISTING
355D B6 0574 R RMBD     LDA A LC+1      LC.=LC+ADR1,ADR2
3560 F6 0573 R          LDA B LC
3563 BB 239B R          ADD A ADR2
3566 F9 239A R          ADC B ADR1
3569 B7 0574 R          STA A LC+1
356C F7 0573 R          STA B LC
356F 7E 0937 R          JMP   MAIN1     RETURN TO MAIN LOOP
                        *
                        *
3572 5F                 RMBOUT CLR B    LOAD 00
3573 FE 239A R          LDX   ADR1      LSAVE.=#BYTES FROM RMB
3576 FF 06FA R          STX   LSAVE
                        *
3579 BD 3F55 R RMBOTA   JSR   OUTBIN    OUTPUT MC
357C FE 06FA R          LDX   LSAVE
357F 09                 DEX
3580 27 06              BEQ   RMBOTB    DONE
                        *
3582 FF 06FA R          STX   LSAVE
3585 5F                 CLR B
3586 20 F1              BRA   RMBOTA    DO AGAIN
                        *
3588 39                 RMBOTB RTS      RETURN
                        *
                        *
                        * POSET  PROCESS SET PSEUDOP
                        *
3589 0002               SETXS  RMB 2    TEMP  STORE
                        *
358B BD 2D12 R POSET    JSR ADRINT      INIT. FLAGS
358E FE 06F4 R          LDX CULINE
3591 FF 06F1 R          STX CUCHAR      INIT. POINTER
3594 A6 00              LDA A 0,X       GET FIRST CHAR
3596 81 20              CMP A #$20      BLANK?
3598 26 09              BNE SETC        NO, LABELED
                        *
359A CE 0213            SETA   LDX #$0213
359D BD 24E8 R SETB     JSR PRINTE      ERROR MESSAGE
35A0 7E 35F3 R          JMP SETF
                        *
35A3 BD 172F R SETC     JSR NXTOK       GET LABEL
35A6 BD 1A5C R          JSR LKPSYM      CHECK SYMBOL TABLE
35A9 C1 FF              CMP B #$FF      IN TABLE?
35AB 26 03              BNE SETD        YES
                        *
35AD BD 19FA R          JSR STOSYM      STORE IT IN SYMTAB
                        *
35B0 BD 1992 R SETD     JSR XREF        HANDLE CROSS-REF
35B3 FE 06F6 R          LDX SYMPTR
35B6 FF 3589 R          STX SETXS       SAVE INDEX
35B9 BD 172F R          JSR NXTOK       SKIP "SET"
35BC BD 172F R          JSR NXTOK
35BF BD 20EA R          JSR NSEVL       EVALUATE OPERAND
35C2 C1 FF              CMP B #$FF      ERRORS?
35C4 26 05              BNE SETD1       NO
                        *
35C6 CE 0216            LDX #$0216      YES, ERROR
35C9 20 D2              BRA SETB
                        *
35CB FE 3589 R SETD1    LDX SETXS       POINT TO SYMBOL
35CE A6 08              LDA A 8,X       GET INFO. BYTE
35D0 7D 0579 R          TST RELFLG      OPERAND RELOC?
35D3 26 09              BNE SETE        YES
                        *
35D5 84 BE              AND A #$BE      NO, TURN OFF REL  BIT
                        *
35D7 7D 057A R          TST CNNFLG      OPERAND COMMON?
35DA 27 02              BEQ SETE        NO
                        *
35DC 8A 10              ORA A #$10      YES, TURN ON COMMON BIT
                        *
35DE A7 08              SETE   STA A 8,X STORE NEW INFO. BYTE
35E0 B6 239A R          LDA A ADR1
35E3 A7 06              STA A 6,X       SAVE OPERAND VALUE
35E5 B6 239B R          LDA A ADR2
35E8 A7 07              STA A 7,X
35EA 73 2398 R          COM POP
35ED 7C 2397 R          INC MCOUNT      SET PRINT FLAGS
35F0 7C 2397 R          INC MCOUNT
35F3 BD 2372 R SETF     JSR PRINTL      LIST LINE
35F6 7E 0937 R          JMP MAIN1       DONE!
```

```
                        *
                        *
                        * HANDLE COMPARISON OPERATORS
                        * .EQ.  .GE   GT  .LE  .LT.  .NE
                        *
35F9 CE 3604 R EQPR     LDX #EQLN
35FC BD 131C R EVAL21   JSR CHKBIN       2 OPERANDS?
35FF FF 368F R          STX COMPL+1
3602 20 4F              BRA CMPR
                        *
3604 20        EQLN     FCC ' BEQ *+7'
3600 0D                 FCB $0D
                        *
360D CE 3612 R GEPR     LDX #GELN
3610 20 EA              BRA EVAL21
3612 20        GELN     FCC ' BGE *+7'
361A 0D                 FCB $0D
                        *
361B CE 3620 R GTPR     LDX #GTLN
361E 20 DC              BRA EVAL21
3620 20        GTLN     FCC ' BGT *+7'
3628 0D                 FCB $0D
                        *
3629 CE 362E R LEPR     LDX #LELN
362C 20 CE              BRA EVAL21
362E 20        LELN     FCC ' BLE *+7'
3636 0D                 FCB $0D
                        *
3637 CE 363C R LTPR     LDX #LTLN
363A 20 C0              BRA EVAL21
363C 20        LTLN     FCC ' BLT *+7'
3644 0D                 FCB $0D
                        *
3645 CE 364A R NEPR     LDX #NELN
3648 20 B2              BRA EVAL21
364A 20        NELN     FCC ' BNE *+7'
3652 0D                 FCB $0D
                        *
                        *
                        * GENERATE COMPARISON CODE.
                        *        LDX ADDRESS1
                        *        STX ACCUM1
                        *        LDX ADDRESS2
                        *        JSR COMPAR
                        *        B-- *+7
                        *        LDX #0          FALSE
                        *        BRA *+5
                        *        LDX #$FFFF      TRUE
                        *        STX ADDRESS1
                        *
3653 7D 0578 R CMPR     TST FLPFLG       IN FLOAT MODE?
3656 27 0D              BEQ CMPR1        NO
                        *
365B BD 123F R          JSR FUNOP2       FLOATING COMPARE
365B BD 1C28 R          JSR STAKDN
365E C6 6C              LDA B #108
3663 20 0E              BRA CMPR2
                        *
3665 7F 368E R CMPR1    CLR COMPL        GENERATE CODE
3668 73 368E R          COM COMPL        SET FLAG
```

```
366B C6 18              LDA B #24
366D F7 20E5 R          STA B TMPVAI
3670 BD 12FD R          JSR BINOP2
3673 FE 368F R CMPR2    LDX COMPL+1
3676 BD 13AB R          JSR STR1         OUTPUT LINE 5
3679 CE 3691 R          LDX #COMPL2
367C BD 13AB R          JSR STR1         OUTPUT LINE 6
367F CE 3699 R          LDX #COMPL3
3682 BD 13AB R          JSR STR1         OUTPUT LINE 7
3685 CE 36AA R          LDX #COMPL5
3688 BD 13AB R          JSR STR1         OUTPUT LINE 8
368B 7E 12D0 R          JMP UNOP4        FINISH UP
                        *
368E 0003      COMPL    RMB 3
3691 20        COMPL2   FCC ' LDX #0'
3698 0D                 FCB $0D
3699 20        COMPL3   FCC ' BRA *+5'
36A1 0D                 FCB $0D
36A2 20        COMPL4   FCC ' LDX #1'
36A9 0D                 FCB $0D
36AA 20        COMPL5   FCC ' LDX #$FFFF'
36B5 0D                 FCB $0D
                        *
                        *
                        * PROCESS "RETURN" COMPILER OPCODE
                        *
36B6 CE 36BF R RETPR    LDX #ZRTS        GET 'RTS'
36B9 BD 13AB R          JSR STR1         OUTPUT CODE
36BC 7E 0937 R          JMP MAIN1        DONE
                        *
36BF 20        ZRTS     FCC ' RTS'
36C3 0D                 FCB $0D
                        *
                        * PROCESS "STOP" COMPILER OPCODE
                        *
36C4 C6 ED     STOPPR   LDA B #237       GET STOP CALL
36C6 BD 139C R          JSR RTBL
36C9 7E 0937 R          JMP MAIN1
                        *
                        *
                        * HANDLE "IF-THEN-ELSE" COMPILER OPCODE
                        * ALSO HANDLE "IF-STMT" FORM
                        * ANY STMT EXCEPT FOR, IF, WHILE
                        *
36CC BD 3812 R IFPR     JSR PRESCN       PRESCAN FOR FLOAT MODE
36CF BD 1C47 R          JSR EVALS        EVALUATE EXPRESSION
36D2 7D 24CC R          TST ERNUM        EVALUATION ERROR?
36D5 26 1B              BNE IFPR4        IF SO, QUIT
                        *
36D7 BD 1B52 R IFPR0    JSR MNLKP        AT A MNEMONIC?
36DA 4D                 TST A            FOUND?
36DB 26 0F              BNE IFPR3        IF NOT, ERROR
                        *
36DD 8C 36E5 R          CPX #IFPR2       TOKEN="THEN"?
36E0 27 03              BEQ IFPR2        IF-THEN FORM
                        *
36E2 7E 3748 R          JMP IFSTM1       IF-STMT FORM
                        *
36E5 BD 172F R IFPR2    JSR NXTOK        GET NEXT TOKEN
36F8 C1 01              CMP B #1         A VALID BRANCH LABEL?
36EA 27 09              BEQ IFPR2A       O K
```

```
                        *
36EC CE 0308    IFPR3   LDX #$0308        ERROR MESSAGE
36EF BD 24E8 R          JSR PRINTE
36F2 7E 0937 R  IFPR4   JMP MAIN1         DONE
                        *
                        * OUTPUT CODE (LAST LINE WAS A 'STX')
                        *       BEQ *+5
                        *       JMP LABEL
                        *
36F5 CE 373F R  IFPR2A  LDX #IFL1
36F8 BD 13AB R          JSR STR1          OUTPUT LINE 1
36FB CE 3BA4 R          LDX #JMPZ
36FE 86 04              LDA A #4
3700 B7 06F0 R          STA A CMPCNT
3703 BD 0DAF R          JSR OUTCMP
3706 BD 0DA6 R          JSR OUTNAM        OUTPUT LINE 2
3709 BD 0DD3 R          JSR OUTCR
370C BD 1992 R          JSR XREF          HANDLE X-REFERENCE
370F BD 172F R          JSR NXTOK         GET NEXT TOKEN
3712 C1 0D              CMP B #$0D        END OF LINE?
3714 27 DC              BEQ IFPR4         IF SO, DONE
                        *
3716 BD 1852 R          JSR MNLKP         AT A MNEMONIC?
3719 4D                 TST A             FOUND?
371A 26 D0              BNE IFPR3         IF NOT, ERROR
                        *
371C 8C 36EC R          CPX #IFPR3        TOKEN="ELSE"?
371F 26 CB              BNE IFPR3         IF NOT, ERROR
                        *
                        * OUTPUT CODE FOR "ELSE"
                        *       JMP LABEL
                        *
3721 BD 172F R          JSR NXTOK         GET NEXT TOKEN
3724 C1 01              CMP B #1          A VALID LABEL?
3726 26 C4              BNE IFPR3         IF NOT, ERROR
                        *
3728 CE 3BA4 R          LDX #JMPZ
372B 86 04              LDA A #4
372D B7 06F0 R          STA A CMPCNT
3730 BD 0DAF R          JSR OUTCMP
3733 BD 0DA6 R          JSR OUTNAM        OUTPUT LINE
3736 BD 0DD3 R          JSR OUTCR
3739 BD 1992 R          JSR XREF          HANDLE X-REFERENCE
373C 7E 0937 R          JMP MAIN1         DONE
                        *
373F 20         IFL1    FCC ' BEQ *+5'
3747 0D                 FCB $0D
                        *
3748 73 06FD R  IFSTMT  COM IFFLG         SET IF-FLAG
374B FE 0708 R          LDX LBLCNT
374E 08                 INX
374F FF 0708 R          STX LBLCNT        GENERATE NEW COMP. LABEL
3752 FF 06FE R          STX IFLBL
                        *
                        * GENERATE CODE (LAST LINE WAS 'STX')
                        *       BNE *+5
                        *       JMP LABEL
                        *
3755 CE 3809 R          LDX #WHPR12
3758 BD 13AB R          JSR STR1          LINE 1
375B CE 3901 R          LDX #BLOKL2       SET UP FOR CVBTD
```

```
375E B6 06FF R          LDA A IFLBL
3761 F6 06FF R          LDA B IFLBL+1
3764 BD 2487 R          JSR CVBTD
3767 CE 38FB R          LDX #BLOKL+6
376A BD 13AB R          JSR STR1          OUTPUT LINE 2
376D C6 20              LDA B #$20        BLANK TO LIST/OBJECT
376F BD 0DA3 R          JSR TPCHAR
3772 86 20              LDA A #$20
3774 BD 0D98 R          JSR WRTCHR
3777 7E 09C7 R          JMP COMP2         FINISH UP
                        *
377A 20         IFL2    FCC ' BEQ '
3780 0005               RMB 5
3785 0D                 FCB $0D
                        *
                        *
                        * HANDLE "WHILE" COMPILER OPCODE
                        * END WITH "BLOCK" COMPILER OPCODE TO FINISH UP
                        *
3786 CE 3B9D R  WHILPR  LDX #LABL+6       PUT OUT DUMMY LINE
3789 BD 13AB R          JSR STR1
378C FE 0708 R          LDX LBLCNT        GENERATE NEW LABEL
378F 08                 INX
3790 FF 0708 R          STX LBLCNT
3793 B6 0708 R          LDA A LBLCNT      SET UP FOR CVBTD
3796 F6 0709 R          LDA B LBLCNT+1
3799 CE 3B98 R          LDX #LABL+1
379C BD 2487 R          JSR CVBTD
379F CE 3B97 R          LDX #LABL
37A2 BD 13AB R          JSR STR1          OUTPUT TOP LABEL
37A5 FE 0708 R          LDX LBLCNT
37A8 08                 INX               GENERATE NEW LABEL
37A9 FF 0708 R          STX LBLCNT
37AC 07                 TPA               SAVE COND  CODES
37AD 36                 PSH A
37AE 0F                 SEI               MASK OFF INTERRUPTS
37AF BF 1BD4 R          STS STKTOP        SAVE STACK POINTER
37B2 BF 389D R          LDS BLKCNT
37B5 B6 0708 R          LDA A LBLCNT
37B8 36                 PSH A             SAVE LABEL FOR "BLOCK"
37B9 B6 0709 R          LDA A LBLCNT+1
37BC 36                 PSH A
37BD 30                 TSX
37BF 09                 DEX               ADJUST
37BF BE 1BD4 R          LDS STKTOP        RESTORE STACK POINTER
37C2 32                 PUL A             RESTORE COND  CODES
37C3 06                 TAP
37C4 8C 3889 R          CPX #BLKSTR       OUT OF BLOCK-STACK SPACE?
37C7 26 09              BNE WHPR2         NO, O.K.
                        *
37C9 CE 0305            LDX #$0305
37CC BD 24E8 R          JSR PRINTE        ERROR MESSAGE
37CF 7E 0937 R  WHPR1   JMP MAIN1         DONE
                        *
37D2 FF 389D R  WHPR2   STX BLKCNT
37D5 BD 3812 R          JSR PRESCN        PRESCAN FOR FLOAT MODE
37D8 BD 1C47 R          JSR EVALS         EVALUATE EXPRESSION
                        *
                        * GENERATE CODE (LAST LINE WAS A 'STX')
                        *       BNE *+5
                        *       JMP ENDLBL+3
```

```
                        *
37DB CE 3809 R          LDX #WHPRL2
37DE BD 13AB R          JSR STR1          OUTPUT LINE 1
37E1 CE 3804 R          LDX #JMP7
37E4 86 04              LDA A #4
37E6 B7 06F0 R          STA A CMPCNT
37E9 BD 0DAF R          JSR OUTCMP
37EC CE 3801 R          LDX #WHPRL+1
37EF B6 0708 R          LDA A LBLCNT      SET UP FOR CVBTD
37F2 F6 0709 R          LDA B LBLCNT+1
37F5 BD 2487 R          JSR CVBTD
37F8 CE 3800 R          LDX #WHPRL
37FB BD 13AB R          JSR STR1          OUTPUT LINE 2
37FE 20 CF              BRA WHPR1         DONE
                        *
3800 2E         WHPRL   FCC '.'
3801 0005               RMB 5
3806 2B                 FCC '+3'
3808 0D                 FCB $0D
3809 20         WHPRL2  FCC ' BNE *+5'
3811 0D                 FCB $0D
                        *
                        *
                        * PRESCAN ROUTINE TO FIND EXPRESSION TYPE
                        * FLOATING OR INTEGER?
                        *
3812 FE 06F1 R PRESCN   LDX CUCHAR        SAVE POINTER
3815 FF 1C05 R          STX AX
3818 86 FF              LDA A #$FF
381A B7 0578 R          STA A FLPFLG      FLOATING FOR NOW
                        *
381D BD 172F R PRES2    JSR NXTOK         GET A TOKEN
3820 C1 0D              CMP B #$0D        CR?
3822 27 36              BEQ PRERTN        IF SO, DONE
                        *
3824 C1 01              CMP B #1          NAME?
3826 26 46              BNE PRES3         NO
                        *
3828 BD 1B52 R          JSR MNLKP         A MNEMONIC?
382B 4D                 TST A             FOUND?
382C 26 33              BNE PRES2A        NO
                        *
382E FE 0642 R          LDX DESCRA
3831 A6 00              LDA A 0,X
3833 81 2E              CMP A #'.         FUNCTION?
3835 26 23              BNE PRERTN        NO, QUIT
                        *
3837 C5 80              BIT B #$80        FLOAT FUNCTION?
3839 26 1F              BNE PRERTN        YES, DONE
                        *
383B C5 40              BIT B #$40        COMPARISON FUNCT?
383D 26 DE              BNE PRES2         YES, CONTINUE SEARCH
                        *
383F C1 2C              CMP B #$2C        "EXP"?
3841 27 DA              BEQ PRES2
                        *
3843 C1 3A              CMP B #$3A        "NOT"?
3845 27 D6              BEQ PRES2
                        *
3847 C1 3B              CMP B #$3B        "ABS"?
3849 27 D2              BEQ PRES2
                        *
384B C1 3D              CMP B #$3D        "SIGN"?
384D 27 CE              BEQ PRES2
                        *
384F C1 3E              CMP B #$3E        "OR"?
3851 27 CA              BEQ PRES2
                        *
3853 C1 3F              CMP B #$3F        "AND"?
3855 27 C6              BEQ PRES2
                        *
3857 7F 0578 R RTNINT   CLR FLPFLG        RETURN INTEGER MODE
                        *
385A FE 1C05 R PRERTN   LDX AX            RESTORE POINTER
385D FF 06F1 R          STX CUCHAR
3860 39                 RTS
                        *
3861 BD 1A5C R PRES2A   JSR LKPSYM        CHECK SYMBOL TABLE
3864 C1 FF              CMP B #$FF        FOUND?
3866 27 EF              BEQ RTNINT        NO, THEN INTEGER
                        *
3868 C5 08              BIT B #$08
386A 26 EE              BNE PRERTN        FLOATING NAME
                        *
386C 20 E9              BRA RTNINT        INTEGER NAME
                        *
386E 81 02      PRES3   CMP A #2          NUMBER?
3870 26 11              BNE PRES4
                        *
3872 C1 03              CMP B #3          HEX NUMBER?
3874 27 E1              BEQ RTNINT        YES, THEN INTEGER
                        *
3876 7D 1982 R          TST DECFLG        DECIMAL NUMBER?
3879 26 DF              BNE PRERTN        YES
                        *
387B 7D 1984 R          TST SIGDIG        FLOATING NUMBER?
387E 27 DA              BEQ PRERTN        YES
                        *
3880 7E 381D R          JMP PRES2         OTHERWISE, CONTINUE SEARCH
                        *
3883 5D         PRES4   TST B             BAD TOKEN?
3884 27 D4              BEQ PRERTN        YES, QUIT
                        *
3886 7E 381D R          JMP PRES2
                        *
                        *
                        *HANDLE "BLOCK" COMPILER OPCODE
                        *
3889 0014       BLKSTR  RMB 20  SPACE FOR 10 NESTED BLOCKS
389D 389C     R BLKEND  EQU *-1
389D 0002       BLKCNT  RMB 2   POINTER TO BLKSTR
                        *
389F CE 3B9D R BLOCKFR  LDX #LABL+6       OUTPUT DUMMY LABEL
38A2 BD 13AB R          JSR STR1
38A5 07                 TPA               SAVE COND  CODES
38A6 36                 PSH A
38A7 0F                 SEI               MASK OFF INTERRUPTS
38A8 BF 16D4 R          STS STKTOP        SAVE STACK POINTER
38AB BE 389D R          LDS BLKCNT
38AE 32                 PUL A
38AF B7 20E6 R          STA A TMPVAL+1
38B2 32                 PUL A
```

```
38B3 B7 20E5 R          STA A TMPVAL.    UNSTACK "BLOCK" LABEL
38B6 30                 TSX
38B7 09                 DEX              ADJUST
38B8 BE 1BD4 R          LDS STKTOP       RESTORE STACK POINTER
38BB 32                 PUL A            RESTORE COND. CODES
38BC 06                 TAP
38BD 8C 389C R          CPX #BLKEND      GOOD BLOCK LABEL?
38C0 2C 09              BGE BLKPR2       O.K
                *
38C2 CE 0311            LDX #$0311
38C5 BD 24E8 R          JSR PRINTE       PRINT ERROR MESSAGE
38C8 7E 0937 R BLKPR1   JMP MAIN1        DONE
                *
38CB FF 389D R BLKPR2   STX BLKCNT
38CE B6 20E5 R          LDA A TMPVAL.    SET UP FOR CVBTD
38D1 F6 20E6 R          LDA B TMPVAL+1
38D4 CE 38F6 R          LDX #BLOKL+1
38D7 BD 2487 R          JSR CVBTD
38DA FE 20E5 R          LDX TMPVAL
38DD 09                 DEX              JMP LBL=LABEL-1
38DE FF 20E5 R          STX TMPVAL
38E1 B6 20E5 R          LDA A TMPVAL.    SET UP FOR CVBTD
38E4 F6 20E6 R          LDA B TMPVAL+1
38E7 CE 3901 R          LDX #BLOKL2
38EA BD 2487 R          JSR CVBTD
38ED CE 38F5 R          LDX #BLOKL
38F0 BD 13AB R          JSR STR1         OUTPUT CODE
38F3 20 D3              BRA BLKPR1       DONE
                *
38F5 2E         BLOKL   FCC ' '
38F6 0005               RMB 5
38FB 20                 FCC ' JMP  '
3901 0005       BLOKL2  RMB 5
3906 0D                 FCB $0D
                *
                *
                * "ON" COMPILER OPCODE PROCESSOR
                * GENERATES CODE TO WORK WITH BRANCH TABLE
                * TABLE GENERATED BY "GOTO" OR "CALL" OPCODES
                *
                *       LDX #TOP       LABEL AT TOP OF BRANCH TABLE
                *       LDA A VALUE    VALUE OF BRANCH INDEX (COMPUTED)
                *       DEC A
                *       BGT *+4
                *       JMP 0,X
                *       INX
                *       INX
                *       INX
                *       INX
                *       INX
                *       CPX #BOTTOM
                *       BEQ *-10
                *       BRA *-15
                *
3907 20         ONL1    FCC ' LDX # '
390E 0005               RMB 5
3913 0D                 FCB $0D
3914 20         ONL2    FCC ' LDA A  ACCUM+1'
3923 0D                 FCB $0D
3924 20         ONL3    FCC ' DEC A'
392A 0D                 FCB $0D
392B 20         ONL4    FCC ' BGT *+4'
3933 0D                 FCB $0D
3934 20         ONL5    FCC ' JMP 0,X'
393C 0D                 FCB $0D
393D 20         ONL6    FCC ' INX'
3941 0D                 FCB $0D
3942 20         ONL7    FCC ' CPX # '
3949 0005               RMB 5
394F 2D                 FCC '-3'
3950 0D                 FCB $0D
3951 20         ONL8    FCC ' BEQ *-10'
395A 0D                 FCB $0D
395B 20         ONL9    FCC ' BRA *-15'
3964 0D                 FCB $0D
                *
3965 7F 0578 R ONPR     CLR FLPFLG       FORCE INTEGER MODE
3968 BD 1C47 R          JSR EVALS        EVALUATE EXPRESSION
396B 7F 12E2 R          CLR UBFLAG       PUT ANSWER INTO ACCUM1
396E 73 12E2 R          COM UBFLAG
3971 BD 12A7 R          JSR UNOP2
3974 FE 0708 R          LDX LBLCNT       GET LABEL VALUES
3977 08                 INX
3978 FF 3BAD R          STX BLKTOP
397B 08                 INX
397C FF 3BAF R          STX BLKBOT
397F FF 0708 R          STX LBLCNT
3982 B6 3BAD R          LDA A BLKTOP
3985 F6 3BAF R          LDA B BLKTOP+1 SET UP FOR CVBTD
3988 CE 390E R          LDX #ONL1+7
398B BD 2487 R          JSR CVBTD
398E B6 3BAF R          LDA A BLKBOT
3991 F6 3BB0 R          LDA B BLKBOT+1 SET UP FOR CVBTD
3994 CE 3949 R          LDX #ONL7+7
3997 BD 2487 R          JSR CVBTD
399A CE 3907 R          LDX #ONL1        PREPARE TO OUTPUT CODE
399D BD 13AB R          JSR STR1         LINE 1
39A0 CE 3914 R          LDX #ONL2
39A3 BD 13AB R          JSR STR1         LINE 2
39A6 CE 3924 R          LDX #ONL3
39A9 BD 13AB R          JSR STR1         LINE 3
39AC CE 392B R          LDX #ONL4
39AF BD 13AB R          JSR STR1         LINE 4
39B2 CE 3934 R          LDX #ONL5
39B5 BD 13AB R          JSR STR1         LINE 5
39B8 CE 393D R          LDX #ONL6
39BB BD 13AB R          JSR STR1         LINE 6 (5 TIMES)
39BE CE 393D R          LDX #ONL6
39C1 BD 13AB R          JSR STR1
39C4 CE 393D R          LDX #ONL6
39C7 BD 13AB R          JSR STR1
39CA CE 393D R          LDX #ONL6
39CD BD 13AB R          JSR STR1
39D0 CE 393D R          LDX #ONL6
39D3 BD 13AB R          JSR STR1
39D6 CE 3942 R          LDX #ONL7
39D9 BD 13AB R          JSR STR1         LINE 7
39DC CE 3951 R          LDX #ONL8
39DF BD 13AB R          JSR STR1         LINE 8
39E2 CE 395B R          LDX #ONL9
39E5 BD 13AB R          JSR STR1         LINE 9
39E8 7E 0937 R          JMP MAIN1        DONE
```

```
                        *
                        * PROCEDURE "CALL" COMPILER OPCODE
                        * CALL EXTERNAL PROCEDURE
                        * PASS ADDRESS ARGUMENTS
                        * EXAMPLE. CALL SUB1(I,J,K)
                        *
                        * CODE.  JSR SUB1
                        *        BRA .LABEL
                        *        FDB I
                        *        FDB J
                        *        FDB K
                        *        FDB $FFFF        MARK END OF PARAMETERS
                        *        (EXT SUB1)       FIRST TIME "SUB1" CALLED
                        *.LABEL  EQU *
                        *
39EB CE 3BA8 R  CALLPR   LDX #JSR2
39EE 86 04               LDA A #4
39F0 B7 06F0 R           STA A CMPCNT
39F3 BD 0DAF R           JSR OUTCMP       OUTPUT 'JSR'
39F6 BD 172F R           JSR NXTOK        GET NEXT TOKEN
39F9 C1 01               CMP B #1         NAME?
39FB 26 6D               BNE CALL4        NO, ERROR MESSAGE AND QUIT
                        *
39FD FE 0642 R           LDX DESCRA
3A00 FF 3BAD R           STX BLKTOP       SAVE TOKEN PARAMETERS
3A03 B6 0644 R           LDA A DESCRC
3A06 B7 3BAF R           STA A BLKBOT
3A09 B7 06F0 R           STA A CMPCNT
3A0C BD 0DAF R           JSR OUTCMP       OUTPUT 'SUB-NAME'
3A0F BD 0DD3 R           JSR OUTCR
3A12 BD 172F R           JSR NXTOK        GET NEXT TOKEN
3A15 C1 0D               CMP B #$0D       CR?
3A17 27 04               BEQ CALL2        YES
                        *
3A19 C1 28               CMP B #'(        LEFT PAREN?
3A1B 26 4D               BNE CALL4        NO, ERROR MESSAGE AND QUIT
                        *
3A1D 86 04      CALL2    LDA A #4
3A1F B7 06F0 R           STA A CMPCNT
3A22 CE 1DD0 R           LDX #FCL2
3A25 BD 0DAF R           JSR OUTCMP       OUTPUT 'BRA'
3A28 FF 0708 R           LDX LBLCNT       MAKE NEW COMPILER LABEL
3A2B 08                  INX
3A2C FF 0708 R           STX LBLCNT
3A2F B6 2971 R           LDA A PDOT
3A32 B7 0649 R           STA A OPTLN
3A35 CE 064A R           LDX #OPTLN+1     SET UP FOR CVBTD
3A38 B6 0708 R           LDA A LBLCNT
3A3B F6 0709 R           LDA B LBLCNT+1
3A3E BD 2487 R           JSR CVBTD
3A41 BD 29F5 R           JSR QPR1         OUTPUT LABEL
3A44 BD 0DD3 R           JSR OUTCR
3A47 FE 0642 R           LDX DESCRA
3A4A A6 00               LDA A 0,X        CHECK LAST TOKEN
3A4C 81 0D               CMP A #$0D       WAS IT CR?
3A4E 27 3C               BEQ CALL5        YES, SKIP ARGUMENTS
                        *
3A50 BD 172F R  CALL3    JSR NXTOK        GET NEXT TOKEN
                        *
3A53 C1 01      CALL3A   CMP B #1         NAME?
3A55 27 16               BEQ CALL4A       YES
                        *
3A57 C1 2C               CMP B #$2C       COMMA?
3A59 26 0F               BNE CALL4        NO
                        *
3A5B CE 3AD7 R           LDX #NULLP       OUTPUT DUMMY PARAMETER
3A5E BD 13AB R           JSR STR1
3A61 BD 172F R           JSR NXTOK        GET NEXT TOKEN
3A64 C1 29               CMP B #$29       RIGHT PAREN?
3A66 27 24               BEQ CALL5        YES
                        *
3A68 20 E9               BRA CALL3A       NO
                        *
3A6A 7E 3BE1 R  CALL4    JMP GTPR1        ERROR MESSAGE AND QUIT
                        *
3A6D 86 04      CALL4A   LDA A #4
3A6F B7 06F0 R           STA A CMPCNT
3A72 CE 1DDA R           LDX #FCL3
3A75 BD 0DAF R           JSR OUTCMP       OUTPUT 'FDB'
3A78 BD 0DA6 R           JSR OUTNAM       OUTPUT ARG  NAME
3A7B BD 0DD3 R           JSR OUTCR
3A7E BD 1992 R           JSR XREF         HANDLE X-REFERENCE
3A81 BD 172F R           JSR NXTOK        GET NEXT TOKEN
3A84 C1 2C               CMP B #',        COMMA?
3A86 27 C8               BEQ CALL3        YES, GET NEXT ARG
                        *
3A88 C1 29               CMP B #')        RIGHT PAREN?
3A8A 26 DE               BNE CALL4        NO, ERROR MESSAGE AND QUIT
                        *
3A8C CE 3ACC R  CALL5    LDX #MINUS1      OUTPUT MARKER
3A8F BD 13AB R           JSR STR1         'FDB $FFFF'
3A92 B6 3BAF R           LDA A BLKBOT
3A95 B7 0644 R           STA A DESCRC     RESTORE TOKEN PARAMETERS
3A98 FE 3BAD R           LDX BLKTOP
3A9B FF 0642 R           STX DESCRA
3A9E BD 1A5C R           JSR LKPSYM       CHECK SYMBOL TABLE
3AA1 C1 FF               CMP B #$FF       FOUND?
3AA3 26 1A               BNE CALL6        YES
                        *
3AA5 FE 056F R           LDX LNUM         INSERT LINE NUMBER
3AA8 FF 0573 R           STX LC
3AAB BD 19FA R           JSR STOSYM       PUT IN TABLE NOW
3AAE 86 04               LDA A #4
3AB0 B7 06F0 R           STA A CMPCNT
3AB3 CE 3AC8 R           LDX #EXT2
3AB6 BD 0DAF R           JSR OUTCMP       OUTPUT 'EXT'
3AB9 BD 0DA6 R           JSR OUTNAM       OUTPUT 'SUB-NAME'
3ABC BD 0DD3 R           JSR OUTCR
3ABF FE 0708 R  CALL6    LDX LBLCNT
3AC2 FF 3BAF R           STX BLKBOT
3AC5 7E 3C67 R           JMP GTPR4        MAKE END LABEL LINE
                        *
3AC8 20         EXT2     FCC ' EXT'
3ACC 20         MINUS1   FCC ' FDB $FFFF'
3AD6 0D                  FCB $0D
3AD7 20         NULLP    FCC ' FDB 0'
3ADD 0D                  FCB $0D
                        *
                        *
                        * "PROCEDURE" COMPILER OPCODE
                        * SET UP PARAMETER LIST FOR EXTERNAL ROUTINES
                        *
```

```
                        * PROCEDURE PNAME(PAR1,PAR2,PAR3,    )
                        *
                        * GENERATE CODE AS FOLLOWS:
                        *
                        *       ENT PNAME       PROCEDURE NAME
                        *PNAME  JSR PROUTN      RUNTIME CALL TO MOVE ADDRESSES
                        *       BRA .LABL
                        *PAR1   RMB 2           PAR  ADDRESSES FORMED HERE
                        *PAR2   RMB 2
                        *PAR3   RMB 2
                        *       FDB $FFFF       MARK LAST SPACE
                        *.LABL  EQU *
                        *
3ADE CE 3B83 R          PROCPR  LDX #ENT
3AE1 86 04                      LDA A #4
3AE3 B7 06F0 R                  STA A CMPCNT
3AE6 BD 0DAF R                  JSR OUTCMP      OUTPUT "ENTRY"
3AE9 BD 172F R                  JSR NXTOK       GET TOKEN
3AEC C1 01                      CMP B #1        VALID NAME?
3AEE 27 09                      BEQ PROC2       YES
                        *
3AF0 CE 0335            PROCER  LDX #$0335      NO, ERROR MESSAGE
3AF3 BD 24E8 R                  JSR PRINTE
3AF6 7E 0937 R                  JMP MAIN1       QUIT
                        *
3AF9 BD 0DA6 R          PROC2   JSR OUTNAM      OUTPUT "PNAME"
3AFC BD 0DD3 R                  JSR OUTCR
3AFF FE 056F R                  LDX LNUM
3B02 FF 0573 R                  STX LC          INSERT LINE NUMBER
3B05 BD 19FA R                  JSR STOSYM      SAVE PNAME IN SYMBOL TABLE
3B08 BD 1992 R                  JSR XREF        HANDLE X-REFERENCE
3B0B BD 0DA6 R                  JSR OUTNAM      OUTPUT "PNAME"
3B0E C6 F0                      LDA B #240
3B10 BD 139C R                  JSR RTBL        RUNTIME CALL
3B13 86 04                      LDA A #4
3B15 B7 06F0 R                  STA A CMPCNT
3B18 CE 1DD0 R                  LDX #FCL2
3B1B BD 0DAF R                  JSR OUTCMP      OUTPUT "BRA"
3B1E FE 0708 R                  LDX LBLCNT
3B21 08                         INX
3B22 FF 0708 R                  STX LBLCNT      NEW COMPILER LABEL
3B25 B6 2971 R                  LDA A PDOT
3B28 B7 0649 R                  STA A OPTLN
3B2B CE 064A R                  LDX #OPTLN+1
3B2E B6 0708 R                  LDA A LBLCNT    SET UP FOR CVBTD
3B31 F6 0709 R                  LDA B LBLCNT+1
3B34 BD 2487 R                  JSR CVBTD
3B37 BD 29F5 R                  JSR GPR1        OUTPUT LABEL
3B3A BD 0DD3 R                  JSR OUTCR
3B3D BD 172F R                  JSR NXTOK       GET NEXT TOKEN
3B40 C1 0D                      CMP B #$0D      CR ?
3B42 27 36                      BEQ PROC5       YES
                        *
3B44 C1 28                      CMP B #$28      LEFT PAREN?
3B46 26 A8                      BNE PROCER      IF NOT, ERROR
                        *
3B48 BD 172F R          PROC3   JSR NXTOK       GET NEXT TOKEN
3B4B C1 01                      CMP B #1        VALID NAME?
3B4D 27 0A                      BEQ PROC4       YES
                        *
3B4F C1 2C                      CMP B #',       COMMA?
3B51 27 F5                      BEQ PROC3       YES, KEEP LOOKING
                        *
3B53 C1 29                      CMP B #')       RIGHT PAREN?
3B55 27 23                      BEQ PROC5       YES, FINISH UP
                        *
3B57 20 97              PROC3A  BRA PROCER      OTHERWISE, ERROR
                        *
3B59 BD 1A5C R          PROC4   JSR LKPSYM      CHECK SYMBOL TABLE
3B5C C1 FF                      CMP B #$FF      FOUND?
3B5E 26 F7                      BNE PROC3A      YES, ERROR
                        *
3B60 BD 19FA R                  JSR STOSYM      PUT IN TABLE
3B63 BD 1992 R                  JSR XREF        HANDLE X-REFERENCE
3B66 FE 06F6 R                  LDX SYMPTR
3B69 A6 08                      LDA A 8,X
3B6B 8A 80                      ORA A #$80      SET PARAMETER BIT
3B6D A7 08                      STA A 8,X
3B6F BD 0DA6 R                  JSR OUTNAM      OUTPUT "PAR"
3B72 CE 2A85 R                  LDX #DEFSL2
3B75 BD 13AB R                  JSR STR1        OUTPUT "RMB 2"
3B78 20 CE                      BRA PROC3       KEEP LOOKING
                        *
3B7A CE 3ACC R          PROC5   LDX #MINUS1     MARK LAST PARAM
3B7D BD 13AB R                  JSR STR1        OUTPUT "FDB $FFFF"
3B80 7E 3ABF R                  JMP CALL6       FINISH LIKE "CALL"
                        *
3B83 20                 ENT     FCC ' ENT'
                        *
                        *
                        * PROCESS "GOTO" COMPILER OPCODE
                        * ALSO HANDLES "GOSUB" COMPILER PROCESSING
3B87 CE 3BA4 R          GOTOPR  LDX #JMPZ       GOTO OPCODE
3B8A FF 3BB1 R                  STX OPPD
3B8D 20 24                      BRA GTPRO       PROCESS
                        *
3B8F CE 3BA8 R          GOSBPR  LDX #JSRZ       GOSUB OPCODE
3B92 FF 3BB1 R                  STX OPPD
3B95 20 1C                      BRA GTPRO       PROCESS
                        *
3B97 2E                 LABL    FCC '.'
3B98 0005                       RMB 5
3B9D 20                         FCC ' EQU *'
3BA3 0D                         FCB $0D
3BA4 20                 JMPZ    FCC ' JMP'
3BA8 20                 JSRZ    FCC ' JSR'
3BAC 0001               GFLAG   RMB 1           SIMPLE OR COMPUTED?
3BAD 0002               BLKTOP  RMB 2           ADDRESS OF BLOCK TOP
3BAF 0002               BLKBOT  RMB 2           ADDRESS OF BLOCK BOTTOM
3BB1 0002               OPPD    RMB 2           ADDRESS OF OPCODE CHARS
                        *
3BB3 BD 172F R          GTPRO   JSR  NXTOK      GET NEXT TOKEN
3BB6 7F 3BAC R                  CLR GFLAG
3BB9 C1 01                      CMP B #1        A LABEL?
3BBB 27 5C                      BEQ GTPR3A      YES
                        *
3BBD C1 28                      CMP B #'(       LEFT PAREN?
3BBF 27 29                      BEQ GTPR2       YES
                        *
3BC1 FE 0642 R                  LDX DESCRA      CHECK TOKEN
3BC4 A6 00                      LDA A 0,X
3BC6 81 40                      CMP A #'@       'AT' SIGN?
```

```
3BC8 26 17              BNE GTPR1       IF NOT, SYNTAX ERROR
                   *
3BCA BD 1C47 R          JSR EVALS       EVALUATE ADDRESS EXPRESSION
3BCD FE 3BB1 R          LDX OPPD
3BD0 86 04              LDA A #4
3BD2 B7 06F0 R          STA A CMPCNT
3BD5 BD 0DAF R          JSR OUTCMP      OUTPUT 'JMP' OR 'JSR'
3BD8 CE 3939 R          LDX #ONL5+5
3BDB BD 13AB R          JSR STR1        OUTPUT ',X'
3BDE 7E 0937 R          JMP MAIN1       DONE!
                   *
3BE1 CE 0309    GTPR1   LDX #$0309      ERROR
3BE4 BD 24E8 R          JSR PRINTE
3BE7 7E 0937 R          JMP MAIN1       QUIT
                   *
3BEA CE 3B9E R  GTPR2   LDX #LABL+7     PREPARE DUMMY STMT
3BED 7C 3BAC R          INC GFLAG       SET COMPUTE FLAG
3BF0 86 05              LDA A #5
3BF2 B7 06F0 R          STA A CMPCNT
3BF5 BD 0DAF R          JSR OUTCMP      OUTPUT CODE
3BF8 BD 0DD3 R          JSR OUTCR
3BFB B6 3BAD R          LDA A BLKTOP
3BFE F6 3BAE R          LDA B BLKTOP+1  SET UP FOR CVBTD
3C01 CE 3B98 R          LDX #LABL+1
3C04 BD 2487 R          JSR CVBTD
3C07 86 06              LDA A #6
3C09 B7 06F0 R          STA A CMPCNT
3C0C CE 3B97 R          LDX #LABL
3C0F BD 0DAF R          JSR OUTCMP      OUTPUT CODE
3C12 BD 172F R  GTPR3   JSR NXTOK       GET NEXT TOKEN
3C15 C1 01              CMP B #1        VALID LABEL?
3C17 26 C8              BNE GTPR1       NO, ERROR
                   *
3C19 FE 3BB1 R  GTPR3A  LDX OPPD        GET CHAR ADDRESS
3C1C 86 04              LDA A #4
3C1E B7 06F0 R          STA A CMPCNT
3C21 BD 0DAF R          JSR OUTCMP      OUTPUT CODE
3C24 BD 0DA6 R          JSR OUTNAM      OUTPUT BRANCH LABEL
3C27 BD 0DD3 R          JSR OUTCR
3C2A BD 1992 R          JSR XREF        HANDLE X-REFERENCE
3C2D 7D 3BAC R          TST GFLAG       SIMPLE OR COMPUTED?
3C30 27 47              BEQ GTPR5       SIMPLE, DONE
                   *
3C32 BD 172F R          JSR NXTOK       GET NEXT TOKEN
3C35 C1 2C              CMP B #,        A COMMA?
3C37 26 27              BNE GTPR3B      NO
                   *
3C39 86 04              LDA A #4
3C3B B7 06F0 R          STA A CMPCNT
3C3E CE 395B R          LDX #ONL9
3C41 BD 0DAF R          JSR OUTCMP      OUTPUT 'BRA BOTTOM'
3C44 B6 3BAF R          LDA A BLKBOT
3C47 F6 3BB0 R          LDA B BLKBOT+1
3C4A CE 3B98 R          LDX #LABL+1
3C4D BD 2487 R          JSR CVBTD
3C50 CE 3B97 R          LDX #LABL
3C53 86 06              LDA A #6
3C55 B7 06F0 R          STA A CMPCNT
3C58 BD 0DAF R          JSR OUTCMP
3C5B BD 0DD3 R          JSR OUTCR
3C5E 20 B2              BRA GTPR3       GET NEXT LABEL
                   *
3C60 C1 29      GTPR3B  CMP B #)        RIGHT PAREN?
3C62 27 03              BEQ GTPR4       END OF BLOCK
                   *
3C64 7E 3BE1 R          JMP GTPR1       ERROR FOUND
                   *
3C67 B6 3BAF R  GTPR4   LDA A BLKBOT    SET UP FOR CVBTD
3C6A F6 3BB0 R          LDA B BLKBOT+1
3C6D CE 3B98 R          LDX #LABL+1
3C70 BD 2487 R          JSR CVBTD
3C73 CE 3B97 R          LDX #LABL
3C76 BD 13AB R          JSR STR1        OUTPUT END LABEL
3C79 7E 0937 R  GTPR5   JMP MAIN1       DONE
                   *
                   *
                   *
                   * SUBROUTINE TO FIND VALUE OF ARRAY ELEMENT
3C7C 7F 139A R  FNDDIM  CLR GAFLG
3C7F CE 1D6A R          LDX #CLN1
3C82 86 04              LDA A #4
3C84 B7 06F0 R          STA A CMPCNT
3C87 BD 0DAF R          JSR OUTCMP      OUTPUT " LDX #"
3C8A 7D 0702 R          TST PARFLG      PARAMETER?
3C8D 26 0A              BNE FND1        IF SO, SKIP "#"
                   *
3C8F 86 23              LDA A #'#
3C91 BD 0D98 R          JSR WRTCHR
3C94 C6 23              LDA B #'#
3C96 BD 0DA3 R          JSR TPCHAR
                   *
3C99 BD 0DA6 R  FND1    JSR OUTNAM      OUTPUT ARRAY NAME
3C9C BD 0DD3 R          JSR OUTCR
3C9F B6 1BCD R          LDA A RUNCNT    SET UP FOR CVBTD
3CA2 F6 1BCE R          LDA B RUNCNT+1
3CA5 CE 129B R          LDX #FCLN4+12
3CA8 BD 2487 R          JSR CVBTD
3CAB CE 128F R          LDX #FCLN4
3CAE BD 13AB R          JSR STR1        OUTPUT " STX ADDRESS"
3CB1 FE 1BCD R          LDX RUNCNT
3CB4 FF 20E5 R          STX TMPVAL      SAVE ADDRESS
3CB7 08                 INX
3CB8 08                 INX
3CB9 FF 1BCD R          STX RUNCNT
3CBC BD 172F R          JSR NXTOK       GET NEXT TOKEN
3CBF C1 28              CMP B #(        LEFT PAREN?
3CC1 27 17              BEQ FND2
                   *
3CC3 7D 139A R          TST GAFLG       A "GETAD" IN PROGRESS?
3CC6 27 01              BEQ DIMERR      IF SO, QUIT
3CC8 39                 RTS             OTHERWISE ERROR
                   *
3CC9 BE 1BD4 R  DIMERR  LDS STKTOP      IF NOT, ERROR
3CCC CE 0304            LDX #$0304
3CCF 7D 139A R          TST GAFLG       A "GETAD" IN PROGRESS?
3CD2 27 03              BEQ *+5
3CD4 CE 0313            LDX #$0313      IF SO, ERROR #313
                   *
3CD7 7E 24E8 R          JMP PRINTE
                   *
3CDA B6 20E5 R  FND2    LDA A TMPVAL
3CDD F6 20E6 R          LDA B TMPVAL+1  STACK ADDRESS
```

```
3CE0 36                PSH A
3CE1 37                PSH B
3CE2 B6 1BCF R         LDA A DMSAV      STACK DIMENSION
3CE5 F6 1BD0 R         LDA B DMSAV+1
3CE8 36                PSH A
3CE9 37                PSH B
3CEA B6 1BD1 R         LDA A DMDATA
3CED 36                PSH A
3CEE B6 0578 R         LDA A FLPFLG     STACK FLOATING FLAG
3CF1 36                PSH A
3CF2 7F 0578 R         CLR FLPFLG       ARRAY INDEX MUST BE INTEGER
3CF5 30                TSX
3CF6 09                DEX
3CF7 09                DEX
3CF8 09                DEX
3CF9 FF 1BD6 R         STX STPSTK
3CFC 7F 1BD2 R         CLR DIMFLG
3CFF 7C 1BD2 R         INC DIMFLG       SET DIMENSION FLAG
3D02 BD 1C62 R         JSR EXPR1        EVALUATE INDEX
3D05 32                PUL A
3D06 B7 0578 R         STA A FLPFLG     RESTORE FLAG
3D09 32                PUL A
3D0A B7 1BD1 R         STA A DMDATA
3D0D 33                PUL B
3D0E 32                PUL A
3D0F B7 1BCF R         STA A DMSAV      RESTORE DIMENSION
3D12 F7 1BD0 R         STA B DMSAV+1
3D15 BD 1C28 R         JSR STAKUN       POP VALUE STACK
3D18 FE 0642 R         LDX DESCRA
3D1B A6 00             LDA A 0,X        TERMINATION CHAR?
3D1D 81 2C             CMP A #',        COMMA?
3D1F 27 06             BEQ FND3         IF SO, TWO DIM
                 *
3D21 81 29             CMP A #')        RIGHT PAREN ?
3D23 27 69             BEQ FNDONE       IF NOT, ERROR
                 *
3D25 20 A2             BRA DIMERR
                 * PROCESS SECOND DIMENSION
                 *
3D27 B6 1BD1 R FND3    LDA A DMDATA     ARRAY HAS SECOND DIM?
3D2A 85 01             BIT A #$01
3D2C 27 9B             BEQ DIMERR       IF NOT, ERROR
                 *
3D2E B6 1C05 R         LDA A AX         GET ADDRESS OF INDEX VALUE
3D31 F6 1C06 R         LDA B AX+1
3D34 36                PSH A
3D35 37                PSH B            STACK ADDRESS
3D36 B6 1BCF R         LDA A DMSAV      STACK DIMENSION
3D39 F6 1BD0 R         LDA B DMSAV+1
3D3C 36                PSH A
3D3D 37                PSH B
3D3E F6 0578 R         LDA B FLPFLG     STACK FLOATING FLAG
3D41 37                PSH B
3D42 7F 0578 R         CLR FLPFLG       ARRAY INDEX MUST BE INTEGER
3D45 30                TSX
3D46 09                DEX
3D47 09                DEX
3D48 09                DEX
3D49 FF 1BD6 R         STX STPSTK       STACK POINTER SET
3D4C BD 1C62 R         JSR EXPR1        EVALUATE SECOND INDEX
3D4F 33                PUL B
3D50 F7 0578 R         STA B FLPFLG     RESTORE FLAG
3D53 BD 1C28 R         JSR STAKUN       POP VALUE STACK
                 *
                 * OUTPUT CODE AS FOLLOWS:
                 *       DEX              ADJUST ORIGIN
                 *       STX ACCUM1
                 *       LDX DIMENSION
                 *       JSR MULT
                 *       STX ACCUM1
                 *       LDX ADDRESS
                 *       JSR ADD
                 *
3D56 CE 3DD5 R         LDX #DIML2
3D59 BD 13AB R         JSR STR1         LINE 1
3D5C CE 12EE R         LDX #UNOPL2
3D5F BD 13AB R         JSR STR1         LINE 2
3D62 CE 1D70 R         LDX #CLN1+6
3D65 33                PUL B            RESTORE DIMENSION
3D66 32                PUL A
3D67 BD 2487 R         JSR CVBTD
3D6A CE 1D6A R         LDX #CLN1
3D6D BD 13AB R         JSR STR1         LINE 3
3D70 C6 12             LDA B #18
3D72 BD 139C R         JSR RTBL         LINE 4
3D75 CE 12EE R         LDX #UNOPL2
3D78 BD 13AB R         JSR STR1         LINE 5
3D7B 33                PUL B            GET ADDRESS
3D7C 32                PUL A
3D7D CE 1289 R         LDX #FCLN3+12    SET UP FOR CVBTD
3D80 BD 2487 R         JSR CVBTD
3D83 CE 127D R         LDX #FCLN3
3D86 BD 13AB R         JSR STR1         LINE 6
3D89 C6 0C             LDA B #12
3D8B BD 139C R         JSR RTBL         LINE 7
                 *
3D8E CE 3DD5 R FNDONE  LDX #DIML2
                 *
                 * GENERATE CODE AS FOLLOWS:
                 *       DEX              ADJUST ORIGIN
                 *       STX ACCUM1
                 * NOTE: SKIP MULTIPLY FOR STRING ARRAY
                 *       LDX #2           2 BYTES/VALUE (FOR INTEGER)
                 *       LDX #9           9 BYTES/VALUE (FOR FLOAT)
                 *       JSR MULT
                 *       STX ACCUM1
                 *       LDX ADDRESS
                 *       JSR ADD
                 *
3D91 BD 13AB R         JSR STR1         OUTPUT LINE 1
3D94 CE 12FE R         LDX #UNOPL2
3D97 BD 13AB R         JSR STR1         OUTPUT LINE 2
3D9A 7D 25C6 R         TST DSTFLG       IN STRING MODE?
3D9D 26 19             BNE FND4         YES, SKIP MULTIPLY CODE
                 *
3D9F CE 3DDA R         LDX #DIML3
3DA2 7D 0578 R         TST FLPFLG       IN FLOATING MODE?
3DA5 27 03             BEQ *+5          NO
                 *
3DA7 CE 3DE2 R         LDX #DIML4       YES
                 *
3DAA BD 13AB R         JSR STR1         OUTPUT LINE 3
```

```
3DAD C6 12                LDA B #18
3DAF BD 139C R            JSR RTBL        OUTPUT LINE 4
3DB2 CE 12EF R            LDX #UNOPL2
3DB5 BD 13AB R            JSR STK1        OUTPUT LINE 5
3DB8 33          FND4     PUL B           RESTORE ADDRESS
3DB9 32                   PUL A
3DBA B7 20E5 R            STA A TMPVAL
3DBD F7 20E6 R            STA B TMPVAL+1
3DC0 CE 1289 R            LDX #FCLN3+12   SET UP FOR CVBTD
3DC3 BD 2487 R            JSR CVBTD
3DC6 CE 127D R            LDX #FCLN3
3DC9 BD 13AB R            JSR STK1        OUTPUT LINE 6
3DCC C6 0C                LDA B #12
3DCE BD 139C R            JSR RTBL        OUTPUT LINE 7
3DD1 7F 1BD2 R            CLR DIMFLG
3DD4 39                   RTS
                 *
3DD5 20          DIML2    FCC ' DEX'
3DD9 0D                   FCB $0D
3DDA 20          DIML3    FCC ' LDX #2'
3DE1 0D                   FCB $0D
3DE2 20          DIML4    FCC ' LDX #9'
3DE9 0D                   FCB $0D
                 *
                 *
                 *
                 * SUBROUTINE TO SETUP STACKS FOR FNDDIM
                 * IF NOT CALLED FROM "EVALS"
                 *
3DEA 7F 139A R EVADIM     CLR GAFLG       INIT. EVALS POINTERS AND FLAGS
3DED CE 0000              LDX #$0000
3DF0 FF 1BCD R            STX RUNCNT
3DF3 7F 1BD3 R            CLR STKCNT
3DF6 7F 1BD2 R            CLR DIMFLG
3DF9 7F 0707 R            CLR NXTFLG
3DFC BF 1BD4 R            STS STKTOP
3DFF FE 0642 R            LDX DFSCRA
3E02 A6 00                LDA A 0,X
3E04 BD 19C5 R            JSR GCHRTB      CHECK TOKEN
3E07 85 80                BIT A #$80      ALPHABETIC?
3E09 27 06                BEQ EVAD1       NO, ERROR
                 *
3E0B BD 1B52 R            JSR MNEMP       A MNEMONIC?
3E0E 4D                   TST A
3E0F 26 06                BNE EVAD2       IF SO, ERROR
                 *
3E11 CE 0313     EVAD1    LDX #$0313      ERROR MESSAGE
3E14 7E 24E8 R            JMP PRINTE
                 *
3E17 BD 2A4E R EVAD2      JSR DEFS        DEFINE SYMBOL DATA
3E1A FF 1BCF R            STX DMSAV
3E1D F7 1BD1 R            STA B DMDATA
3E20 B6 0578 R            LDA A FLPFLG
3E23 36                   PSH A           STACK FLOAT FLAG
3E24 C4 08                AND B #$08      GET FLOAT BIT
3E26 F7 0578 R            STA B FLPFLG    SET FLAGS TO TYPE OF ARRAY
3E29 F6 1BD1 R            LDA B DMDATA
3E2C C4 40                AND B #$40      GET STRING BIT
3E2E F7 25C6 R            STA B DSTFLG
3E31 F6 1BD1 R            LDA B DMDATA
3E34 C4 80                AND B #$80      GET PARAMETER BIT
3E36 F7 0702 R            STA B PARFLG
3E39 BD 3C7F R            JSR FNDDIM+3
3E3C 32                   PUL A
3E3D B7 0578 R            STA A FLPFLG    RESTORE FLOAT FLAG
3E40 39                   RTS
                 *
                 *
                 *
                 * LBLCK. CHECK FOR A AN ILLEGAL LABEL ON A
                 *        PSEUDOP  IF THERE IS ONE DELETE IT,
                 *        AND PRINT AN ERROR MESSAGE.
                 *
3E41 7D 0577 R LBLCK      TST     LBFLG     LABEL?
3E44 27 0F               BEQ     LBLCK2    NO
                 *
3E46 7D 0575 R           TST     PASS      PASS?
3E49 26 03               BNE     LBLCK1    PASS2
                 *
3E4B BD 1AB1 R           JSR     DELSYM    PASS1 DELETE LAST SYMBOL
                 *
3E4E CE 0223   LBLCK1    LDX     #$0223    ERROR
3E51 BD 24E8 R           JSR     PRINTE    PRINT
                 *
3E54 39        LBLCK2    RTS               RETURN
                 *
                 *
                 * OUTBIN  FILL OTBUF WITH OBJECT MC
                 * OUTBNK. OUTPUT A BYTE TO OTBUF
                 *
3E55 B6 06F8 R OUTBIN     LDA A OPTNS       CHECK OPTIONS
3E58 85 40               BIT A #$40        OBJECT?
3E5A 27 01               BEQ     *+3       YES
3E5C 39                  RTS               OTHERWISE SKIP
                 *
3E5D 17                  TBA
3E5E 8D 15               BSR     OUTHL     CONVERT LEFT NIBBLE
3E60 BD 3ED8 R           JSR     OUTB
3E63 17                  TBA
3E64 8D 13               BSR     OUTHR     CONVERT RIGHT NIBBLE
3E66 7E 3ED8 R           JMP     OUTB
                 *
                 * OUTPUT A BYTE
                 *
3E69 B6 06F8 R OUTBNK     LDA A OPTNS       CHECK OPTIONS
3E6C 85 40               BIT A #$40        OBJECT?
3E6E 27 01               BEQ     *+3       YES
3E70 39                  RTS               OTHERWISE SKIP
                 *
3E71 17                  TBA
3E72 7E 3ED8 R           JMP OUTB
                 *
3E75 44        OUTHL     LSR A             LEFT NIBBLE CONVERSION TO ASCII
3E76 44                  LSR A
3E77 44                  LSR A
3E78 44                  LSR A
3E79 84 0F     OUTHR     AND A #$0F        RIGHT NIBBLE CONVERSION
3E7B 8B 30               ADD A #$30
3E7D 81 39               CMP A #$39
3E7F 23 02               BLS *+4
3E81 8B 07               ADD A #$07
3E83 39                  RTS
```

```
                        *
                        *
                        * ASSORTED I/O ROUTINES
                        *
                        *
3F84 CE 3F89 R CRLF     LDX #CRLF1        PRINT CR/LF
3F87 20 06              BRA PDATA1
                        *
3F89 0D0A       CRLF1   FDB $0D0A
3F8B 04                 FCB $04
                        *
3F8C 8D 20      PDATA2  BSR OUTCHR        PRINT STRING
3F8E 08                 INX
3F8F A6 00      PDATA1  LDA A 0,X
3F91 81 04              CMP A #4
3F93 26 F7              BNE PDATA2
                        *
3F95 20 17              BRA OUTCHR
                        *
3F97 A6 00      OUT2H   LDA A 0,X         PRINT HEX BYTE
3F99 8D 0D      OUT2HA  BSR OUTHL
3F9B A6 00              LDA A 0,X
3F9D 08                 INX
3F9F 20 0C              BRA OUTHRR
                        *
3FA0 8D F5      OUT4HS  BSR OUT2H         PRINT HEX DOUBLE-BYTE + SPACE
3FA2 8D F3      OUT2HS  BSR OUT2H         PRINT HEX BYTE + SPACE
3FA4 86 20      OUTS    LDA A #$20        PRINT SPACE
3FA6 20 06              BRA OUTCHR
                        *
3FA8 8D CB      OUTHL   BSR OUTHR         PRINT LEFT NIBBLE
3FAA 20 02              BRA OUTCHR
                        *
3FAC 8D CB      OUTHRR  BSR OUTHR         PRINT RIGHT NIBBLE
                        *
3FAE 36         OUTCHR  PSH A             PRINT A CHARACTER
3FAF 7D 0653 R          TST CONSW         CONSOLE OUTPUT?
3FB2 27 03              BEQ *+5
                        *
3FB4 BD 070D R          JSR OUTEFE
                        *
3FB7 32                 PUL A
3FB8 36                 PSH A
3FB9 7D 0654 R          TST LPTSW         LINEPRINTER OUTPUT?
3FBC 27 03              BEQ *+5
                        *
3FBE BD 0710 R          JSR OUTLPT
                        *
3FC1 32                 PUL A
3FC2 39         OUTCHF  RTS
                        *
                        *
                        * LINKAGES TO MASS-STORAGE DRIVERS
                        *
3FC3 7F 0000 X          EXT OPEN1         OPEN INPUT FILE
3FC6 7F 0000 X          EXT OPENO         OPEN OUTPUT FILE
3FC9 7E 0000 X          EXT TABLFS        SYMBOL TABLE START
3FCC 7F 0000 X          EXT CLOSI         CLOSE INPUT FILE
3FCF 7E 0000 X          EXT CLOSO         CLOSE OUTPUT FILE
3FD2 7E 0000 X          EXT MONTOR        RETURN TO MONITOR
3FD5 7E 0000 X          EXT GETB          GET A BYTE FROM INPUT FILE
3FD8 7F 0000 X          EXT OUTB          PUT A BYTE INTO OUTPUT FILE
3FDB 7E 0000 X          EXT INITIO        INITIALIZE FILES
3FDE 7F 0000 X          EXT RFWIND        REWIND INPUT FILE
                                *
3FE1 3E8F    N          ENT PDATA1        ENTRY POINTS FOR DRIVERS
3FE1 3F84    N          ENT CRLF
                                *
                        END
```

@TEST1K	063F	R
ABK	2B0D	R
ABKCK	2D40	R
ABKCKA	2D49	R
ABSFR	11DB	R
AC	1C01	R
ACOSPR	11EF	R
ADD16	234B	R
ADDR1	2A8B	R
ADDR1A	2A95	R
ADDR1B	2A9D	R
ADDR1C	2AC0	R
ADDR1D	2AD7	R
ADDR1E	2AF8	R
ADDR1F	2AFD	R
ADDR1G	2B09	R
ADDR1H	2B13	R
ADDR1J	2B16	R
ADDR1K	2AC8	R
ADDR1L	2ADE	R
ADDR2	2B1C	R
ADDR2A	2B26	R
ADDR2B	2B2E	R
ADDR2E	2B60	R
ADDR2F	2B65	R
ADDR2G	2B71	R
ADDR2H	2B7B	R
ADDR2J	2B7F	R
ADDR2K	2B56	R
ADDR3	2B84	R
ADDR3B	2B96	R
ADDR3C	2BAD	R
ADDR3D	2BC0	R
ADDR3F	2BC5	R
ADDR3F	2BCD	R
ADDR4	2BD3	R
ADDR4A	2BE7	R
ADDR5	2BF3	R
ADDR5A	2BFD	R
ADDR5B	2C05	R
ADDR5C	2C22	R
ADDR5D	2C2D	R
ADDR5F	2C49	R
ADDR5F	2C4F	R
ADDR5G	2C55	R
ADDR5H	2C5A	R
ADDR5J	2C5D	R
ADDR5K	2C44	R
ADDR6	2C63	R
ADDR7	2C6F	R
ADDR7A	2C81	R
ADDR7B	2C8F	R
ADDR7C	2C99	R
ADDR7D	2C9C	R
ADDR8	2CA2	R
ADDR8A	2CB4	R
ADDR8C	2CE7	R
ADDR8D	2CFA	R
ADDR8E	2CDA	R
ADDR8F	2CE1	R

ADDR9	2CF3	R
ADDR9A	2CFE	R
ADDZ	1337	R
ADR1	239A	R
ADR2	239B	R
ADRCINT	2D12	R
ANDFR	1347	R
ASINFR	11F3	R
ATANFR	11F7	R
AX	1C05	R
BINOP	12FA	R
BINOP2	12FD	R
BLANK	247E	R
BLANK3	2481	R
BLANK5	2483	R
BLANK6	2484	R
BLKBOT	3BAF	R
BLKCNT	389D	R
BLKFND	389C	R
BLKPR1	38C8	R
BLKPR2	38CB	R
BLKSTR	3889	R
BLKTOP	3BAD	R
BLOKL	38F5	R
BLOKL2	3901	R
BLOKPR	389F	R
BYTEFR	1444	R
C9	26DF	R
CALL2	3A1D	R
CALL3	3A50	R
CALL3A	3A53	R
CALL4	3A6A	R
CALL4A	3A6D	R
CALL5	3A8C	R
CALL6	3ABF	R
CALLPR	39EB	R
CBLOCK	3044	R
CBUNF	0E00	R
CENDPR	25B5	R
CFLAG	06F3	R
CHKBIN	131C	R
CHPTR	19E1	R
CHRTAB	052F	R
CIFFLG	0641	R
CLASS	20F8	R
CLIFLG	20E7	R
CLN1	1D6A	R
CLN2	1D76	R
CLOST	3FCC	RX
CLOSO	3ECF	RX
CLOSPR	27AF	R
CMFLG	3384	R
CMNFLG	057A	R
CMNL1	2645	R
CMNLC	0705	R
CMNSG	30B5	R
CMNXS	2ED2	R
CMP1	0DDF	R
CMP2	0DFD	R
CMPCNT	06E0	R
CMPLNG	0655	R

CMPR	3653	R
CMPR1	3665	R
CMPR2	3673	R
CNTR	1986	R
COMFLG	0576	R
COMPO	099A	R
COMP1	09B6	R
COMP2	09C7	R
COMP2A	09F4	R
COMP2B	0A14	R
COMP2C	0A1A	R
COMP2D	0A1D	R
COMP2E	0A2C	R
COMP2F	0A35	R
COMP2G	0A5E	R
COMP2H	0A4B	R
COMP3	0A67	R
COMPAR	0DD7	R
COMPI	368E	R
COMPL2	3691	R
COMPL3	3699	R
COMPL4	36A2	R
COMPL5	36AA	R
CONSW	0653	R
CONVPR	14BB	R
CORASM	07E2	R
COSPR	11FB	R
CRCHK	188F	R
CRLF	3E84	RN
CRLF1	3E89	R
CSECPR	25A9	R
CTBL	1709	R
CUCHAR	06F1	R
CULINE	06F4	R
CVDID	2487	R
CVDB	226C	R
CVDB1	2285	R
CVDB2	2289	R
CVDB3	22BF	R
CVDEC1	248D	R
CVDEC2	2490	R
CVDEC5	249B	R
CVHB	2214	R
CVHB1	2221	R
CVHID	2256	R
CVHBS	225A	R
CXS2	3046	R
DCOUNT	2267	R
DECFLG	1982	R
DELP1	2626	R
DELP1A	2649	R
DELP1B	2657	R
DELP2	2666	R
DELP3	26B5	R
DELP3A	270A	R
DELP4	271F	R
DELPR	25C9	R
DELS	2A4E	R
DELS2	2A56	R
DELSL1	2A7C	R
DELSL2	2A85	R

DEL1	1A88	R
DELAY	2D05	R
DELSYM	1AB1	R
DENDPR	259F	R
DESCRA	0642	R
DESCRC	0644	R
DOSPR2	2564	R
DTOL1	2573	R
DTOL2	257F	R
DTOSPR	2548	R
DTMERR	3C09	R
DTMFLG	1BD2	R
DTMI1	1F2A	R
DTMI2	3BB5	R
DTMI3	3DDA	R
DTMI4	3DE2	R
DTMPR	25CC	R
DIV	133F	R
DIV151	2310	R
DIV153	231B	R
DIV16	2300	R
DIV163	2325	R
DIV165	2332	R
DIV167	2333	R
DMDATA	1BD1	R
DMSAV	1BCF	R
DPR1	25F0	R
DPR10	2758	R
DPR11	276E	R
DPR2	25F7	R
DPR2A	260C	R
DPR3	260F	R
DPR3A	2618	R
DPR4	2675	R
DPR5	267E	R
DPR6	26AA	R
DPR7	26C4	R
DPR7A	26E1	R
DPR8	2721	R
DPR8A	2738	R
DPR9	273B	R
DSCAN	17DE	R
DSCTNP	25A8	R
DSCPR	258B	R
DSTFLG	25C6	R
DSTPR	25C1	R
DVDI	2265	R
DXSAV	226A	R
ECOUNT	0703	R
ENDMA	3086	R
ENDMB	3096	R
ENDP1	2F0D	R
ENDP2	2F1B	R
ENDP3	2F2D	R
ENDP4	2F9C	R
ENDP4B	301F	R
ENDP5	302F	R
ENDP6	3049	R
ENDP7	3075	R
ENDSCN	1850	R
ENDXS	30DE	R

ENST7	1B50	R
ENT	3B83	R
ENTFLG	057C	R
EOFPR	1E2D	R
EOLN	3604	R
EQPR	35F9	R
EQUA	3150	R
EQUB	3146	R
EQUC	3155	R
EQUD	3160	R
EQUE	3189	R
EQUF	3174	R
EQUXS	318F	R
ERMSA	24CE	R
ERMSB	24DF	R
ERMSC	24E0	R
ERNUM	24CC	R
ES	0042	
EVAD1	3E11	R
EVAD2	3E17	R
EVADTM	3BFA	R
EVAL10	1F78	R
EVAL11	1FA5	R
EVAL12	1FAB	R
EVAL13	1FC5	R
EVAL14	1FF0	R
EVAL15	1FF4	R
EVAL19	1350	R
EVAL2	1CCC	R
EVAL20	11D5	R
EVAL21	35FC	R
EVAL2A	1CCF	R
EVAL3	1CD2	R
EVAL3A	1CF5	R
EVAL4	1CE8	R
EVAL5	1CEE	R
EVAL7	1CFE	R
EVAL7A	1D0E	R
EVAL7B	1D1C	R
EVAL7D	1D34	R
EVAL7E	1D46	R
EVAL7F	1D81	R
EVAL8	1E0B	R
EVAL8A	1E3D	R
EVAL8B	1E4C	R
EVAL8C	1D6B	R
EVAL9	1E6F	R
EVAL9A	1EFC	R
EVAL9B	1F0C	R
EVAL9C	1F33	R
EVAL9D	1F3F	R
EVAL9E	1F89	R
EVALS	1C47	R
EXOPR	134B	R
EXPERR	1C3E	R
EXPIN	18F2	R
EXPINZ	190B	R
EXPOR	1917	R
EXPR	134F	R
EXPR1	1C42	R
EXPR2	1C95	R

EXPR3A	1C9B	R
EXPIMP	1985	R
EXSTK	2D04	R
EXTFLG	057B	R
EXTZ	3AC8	R
FABS	11EB	R
FADD	1309	R
FDINOP	13F1	R
FDBA	31FD	R
FDBB	3205	R
FDBC	321F	R
FDBD	3222	R
FDC1	325A	R
FDC2	3268	R
FDCA	3232	R
FDCB	323A	R
FDCC	3246	R
FDCD	325D	R
FDCE	3273	R
FDCF	328F	R
FDCG	329A	R
FDL2	1DD0	R
FDL3	1DDA	R
FDLN1	1257	R
FDLN2	126A	R
FDLN3	127D	R
FDLN4	128F	R
FDOUT	1DE5	R
FDBA	32AD	R
FDBB	32B5	R
FDBC	32EE	R
FDBD	32F1	R
FDDE	32E5	R
FDBF	32DB	R
FDBG	32E2	R
FDIV	13F2	R
FEND	3491	R
FEXP	11FF	R
FIOFLG	27BE	R
FIYPR	1203	R
FLOTPR	1357	R
FLPFLG	0578	R
FMI1	28E0	R
FMI2	28F5	R
FMUL	1D0E	R
FND1	3C99	R
FND2	3CDA	R
FND3	3D27	R
FND4	3DB6	R
FNDDIM	3C7C	R
FNDONE	3D8F	R
FNEG	1212	R
FORFLG	0700	R
FORI1	1075	R
FORI3	1083	R
FORI4	108F	R
FORI5	1098	R
FORPR	0FD9	R
FORPR2	101A	R
FORPR3	1020	R
FORPR5	1055	R

FR1	1862	R
FSCAN	185D	R
FSTON	1216	R
FSUB	13DA	R
FTEMP	1988	R
FUNOP	1224	R
FUNOP2	123F	R
GALLG	139A	R
GOALG2	139B	R
GCHRTB	19C5	R
GCHRTR	19DF	R
GELN	3612	R
GEPR	360D	R
GLIAD	1384	R
GLIADR	138A	R
GPTB	3ED5	RX
GLIIN	188C	R
GLIAG	30AC	R
GOSBPR	3B8F	R
GOTOPR	3B87	R
GOTZPR	18AA	R
GTLN	3620	R
GTPR	361B	R
GTPR0	3DB3	R
GTPR1	3BE1	R
GTPR2	3BEA	R
GTPR3	3C12	R
GTPR3A	3C19	R
GTPR3B	3C20	R
GTPR4	3C67	R
GTPR5	3C79	R
HASH	1ABF	R
HASH1	1AE0	R
HKEYA	19F2	R
HKEYB	19F4	R
HSAV1	19F6	R
HSAV2	19F8	R
HSCAN	1892	R
HSCAN1	189B	R
HSMBI	19EC	R
HVAL	2212	R
IFNLG	06FD	R
IFT1	373F	R
IFT2	377A	R
IFTBL	06FF	R
IFPR	36CC	R
IFPR0	36D7	R
IFPR2	36E5	R
IFPR2A	36F5	R
IFPR3	36EC	R
IFPR4	36F2	R
IFSTK	063F	R
IFSTNT	3748	R
IMMED	2D0F	R
INBUF	0B77	R
INDEX	2D0F	R
INFEE	070A	RX
INITIO	3EDB	RX
INIFR	277F	R
INIPR	25C7	R
INXCK	2D4D	R

INXCKR	2D5C	R
IOCB2	2A1F	R
IOLBPR	2A00	R
IOCERR	2A1F	R
IP	1B4F	R
JMP7	3DA4	R
JSR7	3DA8	R
KTOR	24DD	R
LABEL	062C	R
LABL	3B97	R
LBFLG	0577	R
LBLCK	3E41	R
LBLCK1	3F4F	R
LBLCK2	3E54	R
LBLCNT	0708	R
LC	0573	R
LCLCN	2D48	R
LCN	06F9	R
LCN2	2DBA	R
LCN2A	2DCB	R
LCN2B	2DDD	R
LCN3	2E07	R
LCN3A	2E10	R
LCN3B	2E3F	R
LCNAB1	2D73	R
LCNAB2	2D9C	R
LCNAB3	2DF4	R
LCOUNT	06FC	R
LEFTH	18CF	R
LELN	362F	R
LENGPR	158F	R
LFPR	3629	R
LET1	0E2F	R
LET2	0E3B	R
LET3	0E5F	R
LET3A	0E73	R
LET3B	0E80	R
LET3C	0E84	R
LET3D	0E8F	R
LET3E	0ED5	R
LET3F	0F1C	R
LET3G	0F1B	R
LET4	0F18	R
LET5	0F42	R
LETERR	0F0C	R
LETER	0E05	R
LETER2	0E1D	R
LINEN	239C	R
LKPSM1	1A67	R
LKPSM2	1A6D	R
LKPSM3	1A70	R
LKPSM4	1A7C	R
LKSYM	1A5C	R
LNAB10	2D8F	R
LNAB1B	2D69	R
LNAB1C	2D8C	R
LNAB1S	2D98	R
LNAB2S	2DC5	R
LNAB3S	2DED	R
LNB2	2DB2	R
LNB3	24FA	R

LNUM	056F	R
LOGPR	120A	R
LP	1B4D	R
LPISW	0654	R
LSAVE	06FA	R
LTLN	363C	R
LTMR	3637	R
MACEND	0086	R
MACERR	0385	R
MACFLG	0625	R
MACLIN	0585	R
MACLOP	3461	R
MACNOV	344C	R
MACNV1	3482	R
MACNVE	348C	R
MACPAR	0505	R
MACPSH	0CD8	R
MACPTR	0626	R
MACPUL	0D40	R
MACSAV	0628	R
MACSTK	0088	R
MACTBI	0084	R
MAIN1	0937	R
MAIN2	0955	R
MAIN3	0975	R
MAIN3A	0A69	R
MAIN3B	0A75	R
MAIN3C	0A8E	R
MAIN3D	0A95	R
MAIN3E	0A87	R
MAIN4	0ABC	R
MAIN4A	0AC7	R
MAIN5	0ACA	R
MAIN6	0AD1	R
MAIN6A	0AD6	R
MAIN6B	0AF7	R
MAIN6C	0B1A	R
MAIN6D	0B20	R
MAIN7	0B22	R
MAIN8	0B25	R
MCLPTR	062A	R
MCOUNT	2397	R
MEND	348D	R
MERGE	1529	R
MINUS1	3ACC	R
MNLI	1BBB	R
MNLKP	1B52	R
MNLKPA	1B66	R
MNLKPB	1B75	R
MNMI	1BC4	R
MNTAB	0090	R
MODPR	1353	R
MONTOR	3FD2	RX
MP	1B4E	R
MPSH1	0CE9	R
MPSH2	0D00	R
MPSH3	0D09	R
MPSH5	0D2B	R
MPUL1	0D4F	R
MPUL2	0D5C	R
MPY16	22C3	R

MPY163	22D1	R
MPY167	22DF	R
MSMTCH	1C39	R
MSTKPT	057D	R
MULT	132F	R
MXSAV1	0581	R
MXSAV2	0583	R
MYSAV	0645	R
MYSAV2	0647	R
NATPR	120E	R
NELN	364A	R
NEPR	3645	R
NEXTPR	10A5	R
NIBBLE	1983	R
NOTDON	18DE	R
NOTPR	11D3	R
NSCAN	17F5	R
NSCAN2	1811	R
NSCAN3	1816	R
NSCANA	182E	R
NSEVL	20EA	R
NSEVL1	20FE	R
NSEVL2	2106	R
NSEVL3	210C	R
NSVLA	211A	R
NSVLB	212B	R
NSVLC	2133	R
NSVLC1	2139	R
NSVLD	213E	R
NSVLE	2141	R
NSVLF	2144	R
NSVLG	2150	R
NSVLH	215D	R
NSVLH1	216D	R
NSVLJ	2172	R
NSVLJ1	2182	R
NSVLK	2187	R
NSVLL	218F	R
NSVLLA	21A8	R
NSVLM	21AD	R
NSVLN	21BB	R
NSVLP	21C4	R
NSVLP1	21D3	R
NSVLP2	21DF	R
NSVLP3	21F7	R
NSVLP4	21FE	R
NSYM	008E	R
NULLP	3AD7	R
NUMBER	1C03	R
NX10	1735	R
NX11	1750	R
NX110	17D4	R
NX13	1757	R
NX13A	1774	R
NXT4	177A	R
NX14A	179D	R
NXT5	179F	R
NXT6	17AD	R
NX17	17C0	R
NX18	17CD	R
NXT9	17D0	R

NX1ER	17DB	R
NX1ERR	10C2	R
NX1FLG	0707	R
NXIL1	1157	R
NXTL2	1163	R
NXIL3	1171	R
NXTL4	117A	R
NXIL5	1183	R
NXIL6	1191	R
NXTOK	172F	R
NX1PR2	10C8	R
ONL1	3907	R
ONL2	3914	R
ONL3	3924	R
ONL4	392B	R
ONL5	3934	R
ONL6	393D	R
ONL7	3942	R
ONL8	3951	R
ONL9	395B	R
ONPR	3965	R
OPC1	0715	R
OPC1A	0736	R
OPC2	073C	R
OPC2A	0749	R
OPC2B	075A	R
OPC3	079B	R
OPC3A	07B4	R
OPC3B	07BD	R
OPC3C	07CD	R
OPC4	07FE	R
OPC5	0807	R
OPC6	081A	R
OPCD	2399	R
OPE1	279F	R
OPE1A	27A7	R
OPE2	27A9	R
OPENT	3EC3	RX
OPEND	3EC6	RX
OPENPR	2789	R
OPERN	20E9	R
OPFD	3BB1	R
OPSTAK	1BD9	R
OPTBL	1FFA	R
OPTBL2	2034	R
OPTIN	0649	R
OPTNS	06F8	R
OPTPMT	06DE	R
OPYES	0793	R
ORBYA	2D10	R
ORBYB	2D11	R
ORPR	1343	R
OTCMP1	0DB1	R
OTCMP2	0DC5	R
OUT2H	3E97	R
OUT2HA	3E99	R
OUT2HS	3EA2	R
OUT4HS	3EA0	R
OUTB	3EB8	RX
OUTBIN	3F55	R
OUTBNR	3E69	R

OUTCHE	3EC2	R
OUTCHR	3EAE	R
OUTCMP	0DAF	R
OUTCR	0DD3	R
OUTEFE	070D	RX
OUTHI	3E75	R
OUTHLL	3EA8	R
OUTHR	3E79	R
OUTHRR	3EAC	R
OUTL	23A4	R
OUTL1	23EF	R
OUTL2	23FD	R
OUTL3	240A	R
OUTL4	241C	R
OUTL5	242E	R
OUTL6	2434	R
OUTL6B	245A	R
OUTL7	2469	R
OUTLA	23D2	R
OUTLPT	0710	RX
OUTNAM	0DA6	R
OUTS	3EA4	R
OVFLG	22FF	R
OVFLW	22E9	R
OVFLW2	22FC	R
P2ERR	2D60	R
P2ERRB	2D72	R
PAGEND	352D	R
PARFLG	0702	R
PARMV2	0D78	R
PARMV3	0D97	R
PARMVE	0D6F	R
PARNUM	062E	R
PASS	0575	R
PASS1	08D7	R
PASS1A	08DD	R
PASS2	0902	R
PBLK2	3124	R
PBLOCK	311F	R
PBXS	3136	R
PCOUNT	19E9	R
PDATA1	3E8F	RN
PDATA2	3E8C	R
PDOT	2971	R
PL2	2524	R
PL3	2535	R
PEEKPR	11E7	R
PFLAG	0701	R
PLEND	2389	R
POCMN	2E5A	R
POCMN0	2E67	R
POCMN1	2E6A	R
POCMN2	2E6F	R
POCMN3	2E85	R
POCMN4	2ECC	R
POEND	2ED4	R
POLNDO	2EDA	R
POEND1	2EEE	R
POEND2	2EFF	R
POENT	3000	R
POENT1	3005	R

POENT2	30E9	R
POENT3	310D	R
POENT4	311C	R
POEQU	3138	R
POEXT	3191	R
POEXT1	31A9	R
POEXT2	31C5	R
POEXT3	31ED	R
POEXT4	31F0	R
POFCB	31F3	R
POFCC	3228	R
POFDB	32A3	R
POFFP	32F7	R
POFNC2	3332	R
POFNC3	333B	R
POFUNC	331C	R
POIFA	3358	R
POIFB	3360	R
POIFC	334B	R
POIFD	3379	R
POIFF	337E	R
PONFPR	2A2A	R
PORES	2A4B	R
POMAC	3386	R
POMAC1	33A7	R
POMAC2	33CB	R
POMAC3	33B8	R
POMAC5	33EB	R
POMAC6	33FD	R
POMAC7	3434	R
POMAC8	342E	R
POMACA	3449	R
PONAM	3495	R
PONAM1	34AB	R
PONAM2	34B6	R
PONIF	34CA	R
POP	2398	R
POPAG	3517	R
POPPR	11B7	R
PORMB	3530	R
POSET	358B	R
PREFRTN	385A	R
PRES2	381D	R
PRESZA	3861	R
PRES3	384E	R
PRES4	3883	R
PRESCN	3812	R
PRINT1	2387	R
PRINTE	24E8	R
PRINTL	2372	R
PRNTPR	2774	R
PROC2	3AF9	R
PROC3	3D48	R
PROC3A	3B57	R
PROC4	3B59	R
PROC5	3B7A	R
PROCER	3AF0	R
PROCPR	3ADE	R
PSHIF	34D8	R
PSHCM	350E	R
PSPLER	3506	R

PSTNG1	19F5	R
PSTNG2	19E7	R
PULIF	34EF	R
PUSHPR	119B	R
QL1	2960	R
QL3	296A	R
QL4	2972	R
QPR1	29F5	R
QUOT2	298D	R
QUOTE	296F	R
QUOTPR	297B	R
RANPR	11DF	R
RDL1	0B45	R
RDLIA	0B5C	R
RDL2	0B6F	R
RDL3	0B72	R
RDLINA	0B3C	R
RDLINE	0B7E	R
RDM1	0C28	R
RDM2	0C39	R
RDM3	0C7D	R
RDMAC	0BC7	R
RDMAC0	0BDA	R
RDMAC1	0BDB	R
RDMAC2	0BF7	R
RDMAC3	0C4F	R
RDMAC4	0C83	R
RDMAC5	0C9A	R
RDMAC6	0C80	R
RDMAC7	0C8E	R
RDMERR	0CBF	R
READ2	27CD	R
READ3	27DA	R
READ3A	2805	R
READ3B	2828	R
READ4	283E	R
READPR	27BF	R
REDFF	307B	R
RELFLG	0579	R
RELPR	36D6	R
REWIND	3FDF	RX
REWPR	27B6	R
RMBA	353D	R
RMBB	3542	R
RMBC	355A	R
RMBD	355D	R
RMBOTA	3579	R
RMBOTB	3588	R
RMBOUT	3572	R
RMB7	0448	R
RTBL	139C	R
RTBL1	13B3	R
RTBL2	13BE	R
RTL1	0858	R
RTL2	0868	R
RTL3	0877	R
RTL4	0886	R
RTL5	0892	R
RTL6	089E	R
RTL7	08AC	R
RTL8	08B5	R

RTMSG	08C0	R
RTNINT	3657	R
RTNCNT	1BCD	R
SAVFA	24C7	R
SAVEL	0FCA	R
SAVFX	24C8	R
SAVEX1	24CA	R
SCNCHR	196F	R
SETA	359A	R
SETB	359D	R
SETC	35A3	R
SETD	35B0	R
SETD1	35CB	R
SETE	35DE	R
SETEXP	192A	R
SETF	35F3	R
SETX2	1950	R
SETX3	1951	R
SETX4	1958	R
SETXS	3589	R
STGDIG	1984	R
STGFR	11E3	R
STNPR	121A	R
SORT1	2F37	R
SORT2	2F50	R
SORT2A	2F60	R
SORT4	2F8F	R
SORT5	2F97	R
SORTF	3048	R
SPACER	238A	R
SSETUP	149D	R
SSTERR	14B6	R
SSTUP	14BA	R
STACK	0713	R
STAKD1	1C2B	R
STAKDN	1C28	R
STAKU1	1C13	R
STAKUP	1C10	R
STKCNT	1BD3	R
STKSAV	057F	R
STKTOP	1BD4	R
STOPPR	36C4	R
STOSYM	19FA	R
STPSTK	1BD6	R
STR1	13AB	R
STR2	148D	R
STRBAD	1484	R
STRC3	1688	R
STRC4	1697	R
STRC5	16CD	R
STRC6	16FC	R
STRC6A	16F5	R
STRC6B	1702	R
STRC7	1715	R
STRCER	1666	R
STRCL1	16D8	R
STRCL2	16E2	R
STRCNP	1655	R
STRD2	0F6F	R
STRDIM	0F65	R
STRFLG	0EDB	R

STRL1	0F86	R
STRL2	0F94	R
STRL3	0FA4	R
STRL4	0FB4	R
STRL5	0FBF	R
STRLN1	1597	R
STRLN2	15A0	R
STRN	14E1	R
STRO	14F6	R
STRP	14FF	R
STRP1	15F3	R
STRP2	1601	R
STRP3	160F	R
STRP4	161D	R
STRP5	162B	R
STRP6	1639	R
STRP7	1647	R
STRPR	1466	R
STRQ	1541	R
STRUBA	0000	RN
STRV1	1578	R
STRV2	157D	R
SUB16	235E	R
SUBTR	133B	R
SYM1	1A11	R
SYMA	1A00	R
SYMB	1A3B	R
SYMB1	1A4B	R
SYMC	1A4E	R
SYMCMP	1A85	R
SYMEND	008C	R
SYMMOD	1A99	R
SYMPTR	06F6	R
SYMTAB	00BA	R
TABLES	3EC9	RX
TANPR	121E	R
TBADD	19EA	R
TENP	1987	R
TENPR	1222	R
TENVL	2268	R
TNPVAL	20E5	R
TPCHAR	0DA3	R
TSTNXT	18BE	R
TSTPH	0571	R
UBFLAG	12E2	R
UNEXP	1C7B	R
UNEXP2	1C85	R
UNFLG	1BD8	R
UNOP	12A1	R
UNOP2	12A7	R
UNOP3	12C4	R
UNOP4	12D0	R
UNOPL1	12E3	R
UNOPL2	12FE	R
UPSCLR	1C07	R
VALUE	20E3	R
VALUPR	155A	R
WHILPR	3786	R
WHPR1	37CF	R
WHPR2	37D2	R
WHPR3	3800	R

WHPRL2	3809	R
WPR0	286B	R
WPR1	287B	R
WPR2	289B	R
WPR3	28BB	R
WPR3D	2916	R
WPR3A	291B	R
WPR3B	2956	R
WPR3C	294A	R
WPR4	295D	R
WRFLG	27BD	R
WRITPR	2860	R
WRTCHR	0D98	R
XREF	1992	R
XREF2	19C4	R
XSAV	0E03	R
XSAVR	22FD	R
XTUYPR	1377	R
XTNPR	15B5	R
ZRO5	36BF	R
ZZZ	303B	R

```
0000 0000   N        NAM CPDRVR
                  *
                  * DRIVER FOR CP/68 SOFTWARE PRODUCTS
                  * COPYRIGHT 1979 BY HEMENWAY ASSOCIATES, INC
                  *                ALL RIGHTS RESERVED
                  *
                  *
0000 0545   N        ENT OPENI
0000 0583   N        ENT OPENO
0000 05A1   N        ENT CLOSI
0000 05B6   N        ENT CLOSO
0000 05BB   N        ENT REWIND
0000 05CF   N        ENT GETB
0000 05FF   N        ENT OUTB
0000 0460   N        ENT INITIO
0000 045E   N        ENT MONTOR
0000 041B   N        ENT TABLES
0000 040F   N        ENT INEEE
0000 04BF   N        ENT OUTEEE
0000 048C   N        ENT OUTLPT
                  *
                  * FILE CONTROL BLOCKS FOR IN-FILE, OUTFILE
                  *
0000 0002         INFCB  RMB 2
0002 44              FCC 'DSK'
0005 0000            FDB $0000
0007 002A   R        FDB INBUF
0009 0021            RMB 33
                  *
002A 0100         INBUF  RMB 256
                  *
012A 0002         OUTFCB RMB 2
012C 44              FCC 'DSK'
012F 0080            FDB $0080
0131 0154   R        FDB OUTBUF
0133 0021            RMB 33
                  *
0154 0100         OUTBUF RMB 256
                  *
                  * FILE CONTROL BLOCK FOR LINEPRINTER
                  *
0254 0002         LPTFCB RMB 2
0256 4C              FCC 'LPT'
0259 0080            FDB $0080
025B 027E   R        FDB LPTBUF
025D 0021            RMB 33
                  *
027E 0084         LPTBUF RMB 132
                  *
                  * FILE CONTROL BLOCKS FOR CONSOLE I/O
                  *
0302 0002         CNFCBI RMB 2
0304 43              FCC 'CON'
0307 0000            FDB $0000
0309 032C   R        FDB CNBUF1
030B 0021            RMB 33
                  *
032C 0084         CNBUF1 RMB 132
                  *
03B0 0002         CNFCBO RMB 2
03B2 43              FCC 'CON'
03B5 0080            FDB $0080
03B7 030A   R        FDB CNBUF2
03B9 0021            RMB 33
                  *
030A 0084         CNBUF2 RMB 132
                  *
                  * SET UP ADDRESSING EQUATES
                  *
045E 0020         DFSCRA EQU $20
045E 0023         CUCHAR EQU $23
045E 0025         RC     EQU $25
045E 0026         CLASS  EQU $26
                  *
045E 0005         FCBSTA EQU 5
045E 0007         FCBDBA EQU 7
045E 0010         FCBNAM EQU 16
045E 001D         FCBTYP EQU 29
045E 001E         FCBACS EQU 30
045E 0027         FCBIND EQU 39
045E 0029         FCBSCF EQU 41
                  *
045E 3F           MONTOR SWI          CALL CP/68
045F 1F                  FCB 31       "WARM START"
                  *
0460 3F           INITIO SWI          CALL CP/68
0461 33                  FCB 51       "INITDK"
0462 CE 03B0 R           LDX #CNFCBO  INIT. CONSOLE OUTPUT
0465 6F 05               CLR FCBSTA,X
0467 A6 07               LDA A FCBDBA,X
0469 E6 08               LDA B FCBDBA+1,X
046B A7 27               STA A FCBIND,X
046D E7 28               STA B FCBIND+1,X
046F CE 0254 R           LDX #LPTFCB  INIT  LINEPRINTER OUTPUT
0472 6F 05               CLR FCBSTA,X
0474 EE 07               LDX FCBDBA,X
0476 86 0C               LDA A #$0C   INITIAL FORM-FEED
0478 A7 00               STA A 0,X
047A 08                  INX
047B 3F                  SWI          CALL CP/68
047C 02                  FCB 2        "TXAB"
047D CE 0254 R           LDX #LPTFCB
0480 A7 27               STA A FCBIND,X
0482 E7 28               STA B FCBIND+1,X
0484 CE 0302 R           LDX #CNFCBI  INIT. CONSOLE INPUT
0487 6F 05               CLR FCBSTA,X
0489 6F 27               CLR FCBIND,X FLAG BUFFER EMPTY
048B 39           OUTRET RTS
                  *
                  *
048C 81 04        OUTLPT CMP A #$04   CNTL-D?
048E 27 FB               BEQ OUTRET   IF SO, QUIT
                  *
0490 81 0A               CMP A #$0A   LINEFEED?
0492 27 F7               BEQ OUTRET   IF SO, QUIT
                  *
0494 3F                  SWI          CALL CP/68
0495 00                  FCB 0        "PUSH ALL"
0496 CE 0254 R           LDX #LPTFCB
0499 EE 27               LDX FCBIND,X POINT TO BUFFER
049B A7 00               STA A 0,X    STORE CHARACTER
```

```
049D 08                 INX
049E 36                 PSH A
049F 3F                 SWI             CALL CP/68
04A0 02                 FCB 2           "TXAB"
04A1 CE 0254 R          LDX #IPTFCB
04A4 A7 27      OUTL1   STA A FCBIND,X
04A6 E7 28              STA B FCBIND+1,X
04A8 32                 PUL A
04A9 81 0D              CMP A #$0D      C.R ?
04AB 27 03              BEQ OUTL2       YES
                *
04AD 3F                 SWI             CALL CP/68
04AE 01                 FCB 1           "PULL ALL"
04AF 39                 RTS
                *
04B0 3F         OUTL2   SWI             CALL CP/68
04B1 13                 FCB 19          "IOHDR"
04B2 A6 07              LDA A FCBDBA,X
04B4 E6 08              LDA B FCBDBA+1,X
04B6 A7 27              STA A FCBIND,X
04B8 E7 28              STA B FCBIND+1,X
04BA 6F 05              CLR FCBSTA,X
04BC 3F                 SWI             CALL CP/68
04BD 01                 FCB 1           "PULL ALL"
04BF 39                 RTS
                *
                *
                *
04BF 3F         OUTEEE  SWI             CALL CP/68
04C0 00                 FCB 0           "PUSH ALL"
04C1 CE 03B0 R          LDX #CNFCBO
04C4 EE 27              LDX FCBIND,X    POINT TO BUFFER
04C6 A7 00              STA A 0,X       STORE CHARACTER
04C8 08                 INX
04C9 36                 PSH A
04CA 3F                 SWI             CALL CP/68
04CB 02                 FCB 2           "TXAB"
04CC CE 03B0 R          LDX #CNFCBO
04CF A7 27              STA A FCBIND,X
04D1 E7 28              STA B FCBIND+1,X
04D3 32                 PUL A
04D4 81 0D              CMP A #$0D      C.R ?
04D6 27 D8              BEQ OUTL2       YES
                *
04D8 81 04              CMP A #$04      CNIL-D?
04DA 27 D4              BEQ OUTL2       YES
                *
04DC 3F                 SWI             CALL CP/68
04DD 01                 FCB 1           "PULL ALL"
04DE 39                 RTS
                *
                *
                *
04DF 37         INEEE   PSH B
04E0 3F                 SWI             CALL CP/68
04E1 05                 FCB 5           "PSHX"
04E2 CE 0302 R          LDX #CNFCBI
04E5 6D 27              TST FCBIND,X    BUFFER INIT. YET?
04E7 26 0C              BNE IN2         YES
                *
04E9 3F         IN1     SWI             CALL CP/68
04EA 13                 FCB 19          "IOHDR"
04EB A6 07              LDA A FCBDBA,X  INIT  NEW BUFFER
04ED E6 08              LDA B FCBDBA+1,X
04EF A7 27              STA A FCBIND,X
04F1 E7 28              STA B FCBIND+1,X
04F3 6F 05              CLR FCBSTA,X
                *
04F5 FF 27      IN2     LDX FCBIND,X    POINT TO BUFFER
04F7 A6 00              LDA A 0,X       GET CHARACTER
04F9 08                 INX
04FA 36                 PSH A
04FB 3F                 SWI             CALL CP/68
04FC 02                 FCB 2           "TXAB"
04FD CE 0302 R          LDX #CNFCBI
0500 A7 27              STA A FCBIND,X
0502 E7 28              STA B FCBIND+1,X
0504 32                 PUL A
0505 81 0D              CMP A #$0D      C.R ?
0507 27 04              BEQ IN3         YES
                *
0509 3F                 SWI             CALL CP/68
050A 06                 FCB 6           "PULX"
050B 33                 PUL B
050C 39                 RTS
                *
050D 6F 27      IN3     CLR FCBIND,X    RESET FOR RE-INIT.
050F 3F                 SWI             CALL CP/68
0510 06                 FCB 6           "PULX"
0511 33                 PUL B
0512 39                 RTS
0513 49         INMSG   FCC 'INPUT FILE. '
051F 04                 FCB 4
                *
0520 4F         OUTMSG  FCC 'OUTPUT FILE  '
052D 04                 FCB 4
                *
052E 52         RDERR   FCC 'READ ERROR'
0538 0D                 FCB $0D
                *
0539 57         WRTERR  FCC 'WRITE ERROR'
0544 0D                 FCB $0D
                *
                *
0545 CE 0000 R  OPENI   LDX #INFCB
0548 6F 10              CLR FCBNAM,X    NO FILE YET
054A CE 0513 R          LDX #INMSG      PROMPT FOR FILENAME
054D 3F                 SWI             CALL CP/68
054F 31                 FCB 49          "PRTMSG"
054F 3F                 SWI             CALL CP/68
0550 30                 FCB 48          "GTCMD"
0551 D6 25              LDA B RC        CHECK TOKEN
0553 C1 0D              CMP B #$0D      C.R ?
0555 26 03              BNE OPNI2       NO
                *
0557 0A                 CLV             NO-FILE RETURN
0558 0D                 SEC
0559 39                 RTS
                *
055A DF 20      OPNI2   LDX RESCRA
055C DF 23              STX CUCHAR      BACK UP ONE TOKEN
055E CE 0000 R          LDX #INFCB
```

```
0246   0561 6F 05      OPEN3  CLR FCBSTA,X
0247   0563 6F 27             CLR FCBIND,X   NO FILE YET FLAG
0248   0565 6F 1F             CLR FCBACS,X   ACCESS CODE=0
0249   0567 86 FF             LDA A #$FF
0250   0569 A7 29             STA A FCISCF,X BLANK COMPRESSION ON
0251   056B 86 03             LDA A #3
0252   056D A7 1D             STA A FCBTYP,X FILE TYPE "TEXT"
0253   056F 3F                SWI            CALL CP/68
0254   0570 2C                FCB 44         "FMTFCB"
0255   0571 6D 05             TST FCBSTA,X   ERROR?
0256   0573 26 09             BNE FILERR     YES
0257                   *
0258   0575 3F                SWI            CALL CP/68
0259   0576 14                FCB 20         "OPEN"
0260   0577 6D 05             TST FCBSTA,X   ERROR?
0261   0579 26 03             BNE FILERR     YES
0262                   *
0263   057B 0C                CLC            GOOD RETURN
0264   057C 0A                CLV
0265   057D 39                RTS
0266                   *
0267   057F 3F         FILERR SWI            CALL CP/68
0268   0580 1F                FCB 30         "PRTERR"
0269   0581 0C                CLC
0270   0582 0B                SEV            ERROR RETURN
0271   0583 39                RTS
0272                   *
0273                   *
0274                   *
0275   0584 CE 012A R  OPEN0  LDX #OUTFCB
0276   0587 6F 10             CLR FCBNAM,X   NO FILE YET
0277   0589 CE 0520 R         LDX #OUTMSG    PROMPT FOR FILE NAME
0278   058C 3F                SWI            CALL CP/68
0279   058D 31                FCB 49         "PRTMSG"
0280   058E 3F                SWI            CALL CP/68
0281   058F 30                FCB 48         "GTCMD"
0282   0590 D6 25             LDA B NC       CHECK TOKEN
0283   0592 C1 0D             CMP B #$0D     C R ?
0284   0594 26 05             BNE OPN02      NO
0285                   *
0286   0596 0A                CLV
0287   0597 0D                SEC            NO-FILE RETURN
0288   0598 39                RTS
0289                   *
0290   0599 DE 20      OPN02  LDX DESCRA
0291   059B DF 23             STX CUCHAR     BACK UP ONE TOKEN
0292   059D CE 012A R         LDX #OUTFCB
0293   05A0 20 C0             BRA OPEN3      FINISH LIKE OPEN1
0294                   *
0295   05A2 CE 0000 R  CLOS1  LDX #INFCB
0296   05A5 6F 05      CLOSE1 CLR FCBSTA,X   INIT  STATUS
0297   05A7 6D 10             TST FCBNAM,X   WAS THERE A FILE?
0298   05A9 26 03             BNE CLOS12     YES
0299                   *
0300   05AB 0A                CLV
0301   05AC 0D                SEC            NO-FILE RETURN
0302   05AD 39                RTS
0303                   *
0304   05AE 3F         CLOS12 SWI            CALL CP/68
0305   05AF 15                FCB 21         "CLOSE"
0306   05B0 6D 05             TST FCBSTA,X   ERRORS?
0307   05B2 26 CB             BNE FILERR     YES
0308                   *
0309   05B4 0C                CLC            GOOD RETURN
0310   05B5 0A                CLV
0311   05B6 39                RTS
0312                   *
0313                   *
0314   05B7 CE 012A R  CLOS0  LDX #OUTFCB
0315   05BA 20 F9             BRA CLOSE1     FINISH LIKE CLOS1
0316                   *
0317                   *
0318                   *
0319   05BC CE 0000 R  REWIND LDX #INFCB     POINT TO INPUT FCB
0320   05BF 6F 05             CLR FCBSTA,X
0321   05C1 3F                SWI            CALL CP/68
0322   05C2 15                FCB 21         "CLOSE"
0323   05C4 6D 05             TST FCBSTA,X   ERROR?
0324   05C6 26 B8             BNE FILERR     YES
0325                   *
0326   05C8 3F                SWI            CALL CP/68
0327   05C9 14                FCB 20         "OPEN"
0328   05CB 6D 05             TST FCBSTA,X
0329   05CD 26 B2             BNE FILERR
0330                   *
0331   05CF 0C                CLC
0332   05D0 0A                CLV
0333   05D1 39                RTS
0334   05D2 37         GETB   PSH B
0335   05D3 3F                SWI            CALL CP/68
0336   05D4 05                FCB 5          "PSHX"
0337   05D5 CE 0000 R         LDX #INFCB
0338   05D8 6D 10             TST FCBNAM,X   IS THERE AN OPEN FILE ?
0339   05DA 26 06             BNE GETB0      YES
0340                   *
0341   05DC C6 08             LDA B #8       NO, FORCE END-FILE
0342   05DE E7 05             STA B FCBSTA,X
0343   05E0 20 0A             BRA GETEOF
0344                   *
0345   05E2 3F         GETB0  SWI            CALL CP/68
0346   05E3 18                FCB 24         "READ"
0347   05E4 E6 05             LDA B FCBSTA,X CHECK STATUS
0348   05E6 27 0A             BEQ GETB1      GOOD?
0349                   *
0350   05E8 C1 06             CMP B #6       EOF?
0351   05EA 26 0C             BNE GETB2      ERROR?
0352                   *
0353   05EC 3F         GETEOF SWI            CALL CP/68
0354   05ED 06                FCB 6          "PULX"
0355   05EE 33                PUL B
0356   05EF 0A                CLV
0357   05F0 0D                SEC            EOF RETURN
0358   05F1 39                RTS
0359                   *
0360   05F2 3F         GETB1  SWI            CALL CP/68
0361   05F3 06                FCB 6          "PULX"
0362   05F4 33                PUL B
0363   05F5 0C                CLC
0364   05F6 0A                CLV            GOOD RETURN
0365   05F7 39                RTS
0366                   *
0367   05F8 BD 057E R  GETB2  JSR FILERR     ISSUE SYSTEM MESSAGE
```

```
0370   05F8 CE 052E R          LDX #RDENR
0371   05FB 3F                 SWI            CALL CP/68
0372   05FC 1E                 FCB 30         "PRTMSG"
0373   05FD 3F                 SWI            CALL CP/68
0374   05FE 1F                 FCB 31         "WARM START"
0375                   *
0376                   *
0377                   *
0378   0600 37         OUTB    PSH B
0379   0601 3F                 SWI            CALL CP/68
0380   0602 05                 FCB 5          "PSHX"
0381   0603 CE 012A R          LDX #OUTFCB
0382   0606 3F                 SWI            CALL CP/68
0383   0607 19                 FCB 25         "WRITE"
0384   0608 6D 05              TST FCBSTA,X   ERRORS?
0385   060A 27 0A              BEQ OUTB1
0386                   *
0387   060C BD 057E R          JSR FILERR     ISSUE SYSTEM MESSAGE
0388   060F CE 0539 R          LDX #WRTERR
0389   0612 3F                 SWI            CALL CP/68
0390   0613 1F                 FCB 30         "PRTMSG"
0391   0614 3F                 SWI            CALL CP/68
0392   0615 1F                 FCB 31         "WARM START"
0393                   *
0394   0616 3F         OUTB1   SWI            CALL CP/68
0395   0617 06                 FCB 6          "PUL X"
0396   0618 33                 PUL B
0397   0619 0C                 CLC
0398   061A 0A                 CLV            GOOD RETURN
0399   061B 39                 RTS
0400                   *
0401                   *
0402                   *
0403   061C 061D       R TABLES FDB *+2
0404                   *
0405                   *
0406                           END
```

```
CLASS   0076        FILERR 057E R      OPLNG   0583 RN
CLOFF1  05A4 R      GETB   05FF RN     OPNO2   055A R
CLOSI   05A1 RN     GETBO  05DF R      OPNO2   0598 R
CLOSI2  05AD R      GETB1  05FF R      OUTB    05FF RN
CLOSO   0586 RN     GETB2  05F5 R      OUTB1   0615 R
CNBUF1  03AC R      GETFOF 05E9 R      OUTBUF  0154 R
CNBUF2  0DDA R      IN1    04E9 R      OUTFFE  04BF RN
CNFCB1  0303 R      IN2    04F5 R      OUTFCB  012A R
CNFCB2  0380 R      IN3    050B R      OUTH1   04A4 R
CPMRVR  0000 RN     INBUF  002A R      OUTH2   04B0 R
CRCHAR  0024        INEFF  041BF RN    OUTLFT  046C RN
DESCRA  0020        INFCB  0000 R      OUTMSG  0520 R
FCBAES  001F        INITIO 0460 RN     OUTRFT  048D R
FCBDA   0007        INMSG  0513 R      RD      0075
FCBIND  0027        INPTOF 027F R      RDERR   057E R
FCBNAM  0010        INTFCB 0254 R      REWIND  0DDB RN
FCBSFF  0029        MONITOR 045E RN    TOBFS   0418 RN
FCBSTA  0005        OPEN3  0561 R      WRTERR  0539 R
FCBTYP  001D        OPLNI  0545 RN
```

```
S0032000D8
S1212000BF2713FE5DFADE00FF2084FF39F5CE1000FF39E7CE39E5D0434BFE39E5FFA2
S1212010205A7E0100FF39E7CE39E5D0434BFE39E5FF208808FF208AFF39E7CE7FFFF6
S1212050FF39E5A FF39E5BD435EB639E5F659E6E0009FF39E7CE39E7BD4300B7208EE9
S1212060F7208FCE0009FF39E5CE39E5BD42C3B739E5F739E6FE208AFF39E7CE39E57E
S1212078BD4341FE39E5FF20A7F2715000000000000000000000E4142533 1DB6A
S1212096B2F41434F31FFB02E414E443473F2F41534931F3B12E41544131F7B72E76
S1212 0B4434F5341FBB32E454F463F2D392E45512E55F912E455252 3E2D392E455893
S1212 0D2 0354F2C2E46495320336 2E4A4C4F3357D72E47452E560D422E47542E5644
S1212 0E0 01D432E4C452E562 9442E4C4F47320A83
S1212 0FFB82E4F542E56 37452F4D4F44353 32E2E4E4154320EB92E4E452E5645462E6F
S1212 11D4F4F431D33A2E4F522E33433E2F5045453 1E7382E52414E311F3C2E534745
S1212 13B4E31F33D2F53494E321AB42E54414E321EB52F54454F3222AE2E584F523344
S1212 159AB402F5544F337/BC414241204CF31B414443204A8B0941444420 4A8B0B7D
S1212 177414E44204A8B0441534C204B840C415352204B8407424343204CA224424348
S1212 195 3204CA2254245512 04CA22742474520 4CA22C424754204CA22E4248492081
S1212 1B34CA2224242454204A8B05424C45204CA22F424C4F43587FF0424C53204CA21B
S1212 1D1342 4C54204CA22D424D49204CA22B424E4520 4CA22642504C204CA22A423A
S1212 1FB241204CA2242 5352204CA28D425602
S1212 1FE432 04CA22842565 3204CA22942595445344 4F0434 14C4C59EBF0434242 1052
S1212 221C4CF31143454L4445B5F0434C43204CF30F434C49204CF30E434C4F5347AFA3
S1212 230F0434C52204B840F434C56 204CF30A434D4E204E50FF434D50204A8B0143A2
S1212 2584F4D204B8403434F4D4D4E50FF434F4F5634BBF0435058204BF38C435 45 3E
S1212 2764345A9F0444141204CF31944454 5204B840A4445464945C9F0444454C4140B6
S1212 2941B5F044454F44459FF0444553204CF3344445582 04CF30944494749474F0F4
S1212 2B2444947534546F0445345343458BF0445 3545245C1F0454C534556ECF0454F66
S1212 2D0442 04ED4FF454F542050C0FF454E545250 0FF454F52204A8B0845515 201B
S1212 2EF5138FF45515541513 8FF4558542051 3F
S1212 2FD91FF4558544551 91FF4643422051F3FF4643432052 28FF4644422052A3FF A5
S1212 31B4646502052B7FF464F82202FD9F0465 4E435310F0474554413384F0474F74
S1212 339535558BFF0474F544F5B7F0494637 0205A0FF0494632 0534BFF494E4320 8B
S1212 3574B840C494E505477FF0494E54204CF3314 94F544545C7F0494E58204CF3A4
S1212 375084A4D50204C6F6E4A5 52204C6FAD4C4441204A8B064C445 4204BF38E4CB0
S1212 3993445 4204BF3CE4C454E47358FF04C 4554202F05F04C535 2204B84044D4143A9
S1212 3B15 7538 6FF4E414D205495FF4E414D455495FF4E4547204B84004E455 85 30BB
S1212 3DFA5F04E4946435 4CAFF4E4F50204C F5014F4E20205965F04F50454E4789F050
S1212 3ED4F5241204A8B0A504147205517F5049
S1212 3FC4147455517F504F4B454A2AF0504F50203 1B7F05052494E4774F0505 24F58
S1212 41A435ADFF0505346204B033 6505540204B0 3 2505 5348 819BF0524541444716
S1212 438BFF0524554555 6B6F0524557494 7B6F0524D422055 30FF524F4C204B840709
S1212 456524F5 204B840652544920 4CF33B5254 5 3204CF339534 241204FF310534288
S1212 4744 204A8B0253454 3204CF 30D534549204CF30F534554205 58BFF534556207D
S1212 4920CF30B535441204B1C075354455 01FFF053544F5056C4F053545247346612
S1212 4B01 0535453204C63 6F535455 3204C63CF535542204A8B0053574920 4CF33F5461
S1212 4CE4142204CF316544150204CF306544241204CF3175448454E56E5F0544F2040
S1212 4EC20FFFFF0545041204CF307545354206 9
S1212 4FB4B840D545358204CF3 0054585 3204CF335564140 55355AF0574149204CF3CB
S1212 519B5748494F5786F0575249544860F0585452413585F0000000040400045A
S1212 537101024240424802442424242424242424242000000100400048383828216
S1212 555828080808080808080808080808080808060808180800400040000000000D9
S1212 573000000000000000000000000000000000000000000000000000000000000046
S1212 591000000000000000000000000000000000000000000000000000000000000028
S1212 5AF000000000000000000000000000000000000000000000000000000000000A
S1212 5C D000000000000000000000000000000000000000000000000000000000000EC
S1212 5EB000000000000000000000000000000DD
S1212 5F A000000000000000000000000000000000000000000000000000000000000BF
S1212 613000000000000000000000000000000000000000000000000000000000000A0
S1212 6360000000000000000000000000000000000000000000000000000000000052
S1212 654000D0A5354525542414C2B2043 4F4D504 94C45522E2E2E2E5645525 494FFA
S1212 672 4L2031 2E 310D434F5059524 4746542031 3937382042590D524F42455254F3
```

```
S1212690204752415050454C2C204C4558494E47544F4E2C204D4153532F0D414E4468
S1212A6E204A41434B2048454D454E5741592C20424F53544F4E204D4153532E0D4162
S1212C0C4C205249474854532052455345525454400D0A454E44552204F505449F2
S1122E4A4F4E533A20400000000000000000008F
S1212F900000000000000000000000000000000007E638C7E636C7E63390000BD5E D0
S1212717D87F26537F26547C2653CE2655BD5E8FBD4D127F25767F26FDCE589CFF58FB
S1212759D7F26F87326F8BD270A8540270284DF91422611CE26DE7F26547F26537C2F
S1212753265368D5E8F20DC810D273D8151260376E5ED2814C2604867F2026814D260470
S121277186FB201E814F260486BF201681532604 86DF200E8158260486F7200681491 1
S121276F26A686EFB426F8B726F820A1BD5E84BD5EC325F829F6B626F84385482707AF
S1212 7AUBD5EC625F829F9CE27E2BD5E8F7F2649BD270A914227F0810D2705B72649C0
S1212 7CB20F0BD5E84B6264984DF814126037E28D7814326D4201C0D0A2843294F4D0C
S1122 7E950494C45204F5220284129535345 4D08
S1212 7F8424C453F2004CE28C0BD5E8FCE285D8D270AA700810D270A914227EA088C0A
S1212834BD33ABCE2868BD33ABCE2868BD33ABCE2868BD33ABCE26776A
S1212852BD33AB7E26BD204E414D200000000000000000000D20434D4E202E524E31
S1212670535544B2C38300D20434D4E202E414343554D2C32310D20434D4E202E454F82
S1212 88F462C320D20434D4E202E455252 2C320D20434D4E202E435552494F2C320D9E
S1212 8AC204252412 02A2B350D204558542 02E5254424C0D0D0A454E54455 20505267
S1212 8CA4F4752414D204E414D453A20047F25757F26F3BD438A8620FE208AA700089F
S1122 8E8RC208C26F8CE00C0FF2571FF2703FFCC
S1212 8F7270 5FF3BCDFF 2708FF339A7F26FCCE0000FF2573FF256FFE2084FF26267F8D
S1212 9152625FE2088FF257D86FFB72641CE263FFF263FB626F8849488942706 7F26FA
S1212 933537C2654BD2B2E7F2707F45C67F25777F25787F42FF7F339B7F44CC7F2773
S1212 95100BD4D12FE256F08FF256FFE26F4A600812A2610BD43727D26F327CAFE265F
S1212 94F4BD33AB20C27D26F326037E2A69BD43727D26FD2675F/8BD3C47CE3089B65BADF65BAEBD448735
S1212 98DFFCE589BD4487CE5B97BD33ABFE26F4A6008120271 3BD372FC10127037E5F
S1212 9AB2AB77C2577BD2DA6BD3992C620BD2DA38620BD2D98BD372FC1012665BD3BF6
S1212 9C9524D2750BD3A5CC1FF2746C5202742FF44C8CE5B9DBD33ABFE44C87D2625CA
S1122 9E7270BBD2CD87D262526037E2937FF26F6
S1212 9F626FF270808FF2708FF262C7F262E7C2625BD372FC10D2606F725D57E29375F
S1212A14BD2D6F7E29377E2E1D8CFFF270A6C56E527058C56EC2609CE0202BD44E83A
S1212A327E2937C1F0272E8C5517260DF626FC271B7F26FCBD438A20138C4ED42717CF
S1212A508C5495260 9CE0201BD44E87E2937FE2642BD33AB7E29376E007D2641262 0B1
S1212A6E81202703BD372FBD372FBD3B528C534B27E78C54CA27E2BD4D12BD4372 7EF4
S1212A8C29378120270 3BD372FBD372FBD3B528C558B27C7FE26F4FF26F1A60081209E
S1212AA271EBD372FC1012709810127 05CE020520697C25777D25752603BD39FABD9F
S1212AC83992BD372FC1012705CE0202204FBD3B524D2744BD3A5CC1FF273FC5202748
S1122AE6307D2625270BBD2CD87D262526037E78
S1212AF52937FF2626FE270808FF2708FF262C7F262E7C2625BD4372BD372FC10D263D
S1212B1306F725D57E2937BD2D6F7E29376E00CE0207BD44E8BD43727E2937/D26254E
S1212B312709BD2BC77D2625270139CE2B77FF26F1FF26F4BD5ED52412BD5ECCBD5EB3
S1212B4F0329FB250220E6BE27137E4EDA810A27E5810027E18C2BC62705A70008201A
S1212B6D04C60DE700810D26CF390000000000000000000000000000000000000000CC
S1212B8B000000000000000000000000000000000000000000000000000000000028
S1212BA900000000000000000000000000000000000000000000000000000000000A
S1212BC7FE2626A60081 7260B7A26252705BD2D4020ED39CE2585FF26F4FF26F1FF2C
S1122BE5262AFE2626A60008FF26268126275B21
S1212BF481212641862EFE262AA70008B6262CF6262DBD4487FE2626A600BD39C585FC
S1212C128026037E2CBFFE2626A60008FF2626FE262AA701C60608FF262A8C25CF268C
S1212C30037E3CBF5A26F120AEFE262AA70008FF262A8C25CF26037E2CBF810D26992C
S1212C4E39BD425AB1262E2F03B6262E1608FF26265D261B8624FE262AA700B6262EE1
S1212C6CBD5E75A701B6262EBD5E79A702C60320ABCE25D5FF2628A60008812C270498
S1212C8A810D26F55A26EFFE2628A60008FF2628FE262AA70008FF262A8C25CF2717BF
S1212CA8FE2628A60008FF2628812C26037E2BE7810D26DE7E2BE7860DFE262AFF2635
S1212CC6F1A700CE0230BD44E8FE26F4FF26F17E2BE7FF25810736 0FBF257FBE257DFY
S1122CE4CE262FC609A60009FF25833009BC2081
S1212CF3862735FE2583365A26ECCE25D5A600810D27030820F7A600FF25833009BC0E
S1212D112086271 6FE2583 3609 8C25D426EABF257DBE257F3206FE258139BE2088BF46
```



```
S1212D2F257DBE257F3206CE0251BD44E8FF258139FF2581073 60FBF257FBE257DCEDD
S1212D4D25D532A70008810D26F8CE2626C60932A700085A26F9BF257DBE257F32069F
S1212D6BFE258139CF25D57C262EFF2628FF2642A60008FF2642FE2628A700D8FF26E9
S1212D892881 0D2709812C26E67C262E20E139F626F8C51 0260 3BD5EAE397E5E69FE28
S1212DA72642B62644B726F0A60036 8DE133378DEB32810D2608860A8DD7C60A20DED6
S1212DC5087A26F026F486208DC962020D0A60D20DA3637E604FF2E03FE2E03EE0047
S1122DE3A600FE2E036C0126026C00FE02A10076
S1212DF2260CFE2E036C0326026C025A26DE33239000 0BD372FC1012711CE03027DF0
S1212F10339B2703CE0313BD44E87F2937BD3A5C7F27017F2EDBC1FF2605BD4A562013
S1212E2E0CC5032708732EDB7D270026D1FE2642FF2645B62644B72649C58027037370
S1212E4C2701C408F725/8270A7D270026D2B339B26ADBD372FFE2642FF2647C13D1E
S1212E6A2718C10D279CFE264208A600813D2706810D278E20F308FF26F17D339B2791
S1212E8805BD338A2003BD3C47/D2FDB2746FE2645FF2642FE2647FF26F1B62649B726
S1212EA626447D25782728FE2FCABD33ABCE32EEBD33ABC6A5BD3224BD5DEACE32EEDC
S1212EC4BD33ABCE2FCABD33ABC6A5BD339C7E2937BD2F657E2937007D25782737 86ED
S1122EE204B726F0CE32E3BD2DAF7D2701260ABB
S1212EF18623BD2D98C623BD2DA3FE2645B62649B726F0BD2DAFBD2DD3CE32EEBD338F
S1212F0FABC6A5BD32247E2937/D27012725CE2F86BD33AB86FFB7339AFE2645FF26F3
S1212F2D42FE2647FF26F1B62649B72644BD5DED8D2F7E2937CE32EE8604B726F0BDD6
S1212F4B2DAFFE2645B62649B726F0BD2DAFBD2DD37D270027013 97E2937CE2F86BDB4
S1212F6933ABBD5DEACE2F94BD33ABCE2FA4BD33ABCE2FB4BD33ABCE2FBF7E33AB207B
S1212F8753545820 2E414343554D2B340D204C4441204 2O2E41434355 4D2B340D2071
S1212FA54C4441204220 2E414343554D2B350D2053544120412032 0C580D205354 4161
S1212FC3204220312C580D204C445820232E414343554D2B380D7327007F2578BD2EB5
S1122FE105FE270808FF2708FF5BAD7F2578BD95
S1212FF03C477D44CC265EB65BADF65BAECE307BBD4487CE3075BD33ABFE2642A60053
S1213000F810D2606CE56A2BD33AB20067F25/8BD3C47CE3089B65BADF65BAEBD448735
S1213302CCE3089BD33AB07360FBF3BD4BF589DB65BAD36B65BAE363009BE3BD432067 2
S1213404A8C58892609CF0305BD44E87E2937FF589DCE308FBD33ABCE3099B65BADF6C4
S1213065BAEBD4487CE3098BD33AB20E0205354582 02E00000000002B320D205354E6
S1213086582 02E00000000000D20425241202A2B360D2E00000000020524D42203445
S1213 0A40D07360FBF3BD4BF589D32B75BAE32B75BAD3009BE3BD432068C589C2C095A
S1213 0C2CE0307BD44E87E2937FF589DBD372FC10126EDBD3A5CC1FF27E6C58B26E2E9
S1122 0EOCF32E38604B726F0BD2DAFBD2DA6BDBD
S1213 0EF2DD3CE3189B658ADF65BAEBD4487CE3183BD33ABCE3191BD33ABCE32EEBDFF
S1213 10D33ABCE315DB65BADF65BAEBD4487CE315/BD33ABC60CBD339CCE32EE86045A
S1213 12BB726F0BD2DAFBD2DA6BD2DD3CE5BA43604B726F0BD2DAFCE317B65BADF6E4
S1213 1495BAEBD4487CE317ABD33AB7E2937204C4458202E00000000000D204C445876
S1213 1672 02E00000000002B320D20424754202A2B350D2E00000000002B340D2043BD
S1213 185058202F0000000002B320D204245512 02A2B31380DBD5B12BD938AC6D8A6
S1213 1A37D25782702C6DB7D45C62702C6DEBD339C7E2937BD5812BD338AC6E17D257D
S1213 1C1782702C6E47D45C62702C6E/BD339C7E2937C600F740E57E32A1C61B20F6AF
S1122 1DFC62720F2C61E20EEC65120FAC6632082
S1213 1FE35C69 02031C68D202DC693 2029C68 7202 5C69F20217F2578C65D201AC69694
S1213 20C2016C699201 2C669200EC666200C6842006C68A2002C69CF740E5B63C03D1
S1213 22AF63C04CE3264BD4487CE3257BD33ABF640E57E339CB63C03F63C04CE32647/
S1213 248BD4487CE3257BD33ABCE32EE7E33AB204C4458202E524E53544B2B00006A
S1213 266000000D205354582 02E524E53544B2B00000000000D204C4458202E5237
S1213 2844E53544B2B00000000000D205354582 02E524E53544B2B00000000000D7FFA
S1213 2A232E27F568EB63C03F63C04CE3289BD4487CE327DBD33AB7D32E22706CE3281
S1213 2C0EE7E33ABF640E5BD339C7D568E/70139B63C03F63C04CE329BBD4487FE32EB
S1122 2DE8F7E33AB00204C4458 200000000000CA
S1213 2EB0D205354582 02E414343554D0D7F568EBD3C287F32E27332E28D9FB63C050E
S1213 30BF63C066E3289BD4487CE327DBD33AD20A87D3BD32C0CFF2E03CE0301BD44B1
S1213 329E8FE2E0339C612F740E58DE/20C3C60C20F5C60F20F1C61520ED0 60920E25A
S1213 347C60320E5C60620E1C62420DDC6212 0D986FFB72578B732E2B7568EBD32A72D
S1213 365FF3BCD8080808080808FF3BCD0C607E3724 8DA3BD323FBD3C28C6A27E3270
S1213 3832473339B7E2E057F339A75339ABD372FBD5DED7D44CC3900004FCF33C3BDC6
S1213 3A14487CE33B38D03CE33BE8664B726F07E2DAF204A5352202E5254424C0D206D
S1213 3BF464342200000000000D8678B732E2866FB740E58672B740E62017867B202D
```

S11233DDED8467E20E98681B732E28672B740E53D
S12133FC866FB740F6BD331CBD3C28B63C03F63C04CE3264BD4487CE3257BD33ABF6C1
S121340A40E5BD339CB63C05F63C06CE3264BD4487CF3257BD33ABF640E6BD339CF644
S121342832F2BD339CB63C03F63C04CE3264BD4487CE3257BD33ABC6757E339C8D576D
S121344A4E35F3BD33AB7F2578BD3C47CE2F86BD33ABCF3601BD33ABC654BD339C7E95
S121346429F7FE26F1FF3C05BD372FBD3B528C56CC26037E3655FE3C05FF26F18D1B47
S121348227D9CE0312BD44E87E2937BD372FFE2642FF26F1C12726647E3541BD581222
S12134A0BD338A2611FE2642A600812926048D372F1781BD2704313120CA398DE0CE96
S12134BE35F3BD33ABBD5812BD3C477D44CC26B67D278270E4F5FCE3264BD4487CEA2
S12134DC3257BD33ABCE2F86BD33ABCE3601BDD9
S12134FB2DAF7D25782704C6AB20D2C642BD339C7E2937CE35F3BD33ABBD338A270364
S121350976E3484CE2F86BD33ABCE3601BD33ABCE32EFBD33ABCE361DBD33ABC648BD9C
S121352739D9CFE2642A600812C26037E348D810D26037E2937BD372F20E8BD372FCEF1
S121354532FFBD33ABCE4960BD2DAFC648BD339CBD498D20CFBD349DCE35F3BD33AB5E
S121356BBD338A27037E34847D25782708C6A8BD339C7E2937C63FBD339CCE2F86BD7A
S121358122DAFCE3601BD2DAFBD2F6E7E2937BD349D7D25782709CE0333BD44E87E290A
S121359F37CF35F3BD33ABBD338A27037E3484C63CBD339C20C8BD349D7D257826DA4A
S12135BDCF35F3BD33AB7F2578BD3C477D44CC27037E3484CE2F86BD33ABFE2642A6EB
S12135DB00810D26037E3484BD338A27037E349A
S12135FA84C645BD339C7E2937205345820 2E414343554D2B360D204C4458202E41FB
S121360843435 54D2B360D204C4458202F414343554D2B380D204C4458202E414343BE
S121362645 4D2B340D205345820 2E414343554D2B380D205345820 2E414343554D58
S121364420 20D204C4458202E414343554D2B320DFE26F1FF3C05BD372FC127272624
S121366201012703 7E3484FE3C05FF26F1BD338A26F2CE35F3BD33ABFE2642A600811F
S121368029271 4FF26F1200FCE36E2BD33ABCE35F3BD33ABBD498DBD372FC10126C807
S121369EBD3B524D26C2C54027BEC0412BBA37FE26F1FF3C05BD372FC1272713C10128
S12136C26A8FF3C05FF26F1BD338A2723337E3484CE36D8BD33ABBD498D201D204CE9
S12136D44582023 2A2B350D204C44582023 2AF2
S12136F92D380DFF2642A600812926 03BD372FC64EBD339CCE37093 5D271108085A6D
S121370720F9560456125620562E563C564AEE00BD33ABCE5691BD33ABCE5699BD3313
S121372 5A0FE56AABD33AB7F56D77F26447C2644FE26F1FF2642A60008FF26F1812013
S121374327F0220981 0D27037E371B1639815F23037E37DBBD39C58501271C7D26F3DB
S121376 12617FE26F18601E600C12027068 60DC10D2629FF2642E600398 58027217D7A
S121377F26F3271AFE2642A600812E2611A601BD39C5854027087D257827417E385DF1
S121379D2056854027 0A7D25/827037E385D203185202 6C38504270BFE2642E600C1A0
S12137BB2426B6207285102717FE2642E600C1282603860839C12927F9C13D260386AB
S12137D940394F5F39FE26F1A6007C264408FFD5
S12137F826F1BD39C5854026EDC609205BFE26F1A600812E2618FE2642A600812E2642
S121380600NFF26F17C2644C601B602397C264408FF26F1BD39C585C026D1FA
S121382406 07F12644240 3F72644C601201E7F2644FE26F1FF2642FE26F1A6007C260B
S121384240 8FF26F1BD39C5850226EDC6037A2644FE26F109FF26F1860239CE39887C
S121386006096F00085A26FA7F39827F39847F3985CE3988C60EF73987C620E700C626
S121387EFFF/3983BD396F812E2606733982BD396F810D26037E392A80302710 2B42B7
S121389C8109223F7D3984 26056F007C39847D398727DD7D3984260A7D398227D37ABC
S12138BA398520CE7D3982 26037C3985/D3983270448484848AA02A70273398327017 5
S12138D8087A398730AE8B30814527 0E812E2642
S12138E7427D3982263D733982209A8D7BCE39887F39867F3984812B270A812D260830
S121390 5640FA00F7008D6280302B198109230220 13F601585858581BA7017D3986BE
S121392326057339 6620F1CE3988A601840F36A60184F0441644441B331BE600C540D4
S121394127014 0BB39852A0740C640EA00E/005F810A2506800ACB1020F61B19240251
S121395F6000A701C60F8602FE26F109FF26F139FF4265FE2642A60008FF2642FF2622
S121397DFFE4265390000000000000000000000000000000000000B626F8850826 2B7D2505
S121399B7527263684 8FB726F8F6256FBD5E55F62570BD5E55BD3ABF8606CE39ECBD13
S12139B9512 4C60DBD5F6932B726F83981202516815F22127F39E1B739E2CE39E1BIEO
S12139D7434BFE39E1A600394F3900000250F009C
S12139E600000000000000000000000000000000000000BD3ABFFF26F6A6008120A7
S12139A017635FF39F8CE39ECFF39F6C606FE39F6A60008FF39F6FE39F8A70008FF3909
S12139A22F85A26FBE62573A700B62574A7014F7D26F32602864 0A702398D48260FFE76
S12139A4026F68A80A08A708CE02067E44F88D49BC39F2270220A9CE022120EF8D61C4
S12139A5FFF26F6BD3992FE26F6A6008120260 3C6FF398D132608FE26F6E608EE063918
S12139A7C8D1BBC39F226F420F8FF39F56606B739E9CE39FCFF39E7CE39E57E2BD7FE87
S12139A9A26F6080808080808080808080BC208C2603FE208AFF26F639FE26F68620C609A
S12139AB8A700085A26FA39CE2020FF39ECFF39FEFF39FCCE39ECFF39F8FE2642FF3949
S12139D6F4E2644C1072D02C606FF39F6A600F1
S12139E508FF39F6FF39F8A70008FF39F85A26EB7342FFFE39ECFF39F2FE39EEFF39B2
S12139F03FC0FF39F2BD434BFE39F0FF39F4CE39F2BD434BB639F2F639F3FE208EFF39EA
S1213A21FC0FF39F4BD4300FF39F24FC609CE39F2BD42C3B739F2F739F3FE208AFF39AB
S1213A3FFC0CE39F2BD434BFF39F27F42FF39000000007BD3ABF7342FF8604B739E936
S1213A5D086AAB7CB4F4FB73B4DB63B4D4CB13B4E26067F42FF86FF39F63B4DFB3B4E18
S1213A7D56F73B4F4FCE3B505ABD4C3D739F5F739E6CE2090FF39E7CE39E5BD434BFE
S1213A99FE39E5FF39FACE39ECFF39E7CE39E5BD2DD7250E26154FFE39EAE606EE0451
S1213AB7/F42FF39B63B4FB73B4D7E3B66B63B4FB73B4E/E3B660000000000000000B6
S1123ABD5000000000000000000000000000000BD
S1213BE4000000000000000000000000000000000000000000000000000000000000BF
S1213C0200000000008D077F3C037F3C039CE3BD9F602E700088C3C0326F6B63BD3F2
S1213C2081142E1A7C3BD339CE3C04E600E702098C3BD826F67A3BD339CE033 39200359
S1213C3FCE0300BF3BD47E44E8BF3BD48E40D4BF3BD67F3BD37F3BCD7F3BCE7F339AF4
S1213C5C7F27027F3BD27D2707260 3BD372F7F44CC7F2707/F3BD3C12B271EC12D2608
S1213C7A207D25/82705C6473720103 7CE5691BD33ABBD3C07733BD87E3D34BD372F2F
S1213C987F3BD85D27A07F2702C12C2743C13D273FC10D273BC15D2737C10126037EDC
S1213CB63E0B8102274 2C1292712 8124270BC128 2604377F3C627E3C3E3720C3300900
S1123CD4F63BD2BC3BD62709325F8128260 37EFC
S1213CE39FAB5D27F4F63BD35A261F3009BC3BD626D77D3BD22603BE3BD439C10326 65
S1213D010CF62644C1042EC3BD4214200EC109266FF62644C1052EBSDD426FFF40E548
S1213D1FBD3C07B640E5F640E6CE3D70BD4487CE3D6ABD3AEB63BCDF63BCECE329B20
S1213D3DBD4487CE328FBD2DAFF3BCDFF3C0308087D257827070808080808080808FFF1
S1213D5B3BCDF63BCEC1502F037E3C3E7E3FAB204C4458202300000000000D2053547D
S1213D79582000000000000DC10F27037E3CCCB63BCDF63BCECE3264BD4487CE325/23
S1213D97BD33ABCE32EFBD33ABCE4960BD33ABC6A5BD339CCE3DDOBD33ABCE39868D4B
S1213DB52FCE398A8D2ACE398C8D25CE398E8D20CE39908D1BBD3C077E3D46204252FA
S1123DD341202A2B31320D204644422024000087
S1213DE200000DA600BD5E75B73DE0A600BD5E79B73DE1A601BD5E75B73DE2A601BD28
S1213E005E79B73DE3CE3DDA7E33ABBD3B524D265EC1F026037E3CE8FE2642A600818D
S1213E1F2E264E/D257826170580270 37E3C39C13926037E3F89C1362633F7257820BA
S1213E3C2EC580270BC47FC13726247F2578201FC54026 1BC12C2717C13A2713C13B3D
S1213E5A270FC13D270BC13E2707C13F27037E3C39377E3C62BD4A4FFF3BCFF73BD1E2
S1213E78C580270 37C2702C50326037E3F33C440F745C6B63BD2F63BD33637B63BCD3B
S1213E96F63BCE3637B63BD6F63BD736377F3BD3F63BD17D2578274EC5082603 7E3C94
S1213FB439BD5C7CCE3639BD3ABFE3CC3FF2E03FE40F5FF3C03BD323FCE3647BD330F
S1123ED2ABC6A5BD339C3332B73BD6F73BD7330D2
S1213FE132B73BCDF73BCE3332B73BD2F73BD3FE2E03FF3C03BD3C077E3D46C508273E
S1213FFF037E3C39BD5C7CCE3F2ABD33AB3332F73BD7B73BD63332F73BCEB73BCD33B7
S1213F13B2F73BD3B73BD2BD3C077F3D34204C4458203 02C580D7D257826 07C5082774
S1213F3B4D7F3C39C50826037E3C39B63BCDF63BCECE3264BD4487CE325/BD33ABCED2
S1213F59322FFBD33ABCE3D6A8605B726F0BD2DAF/D2702260A8623BD2D98C623BD2D51
S1213F77A9BD2DA6BD2DD3C6A5BD339CBD3C07/E3D46CE3D6A8605B726F0BD2DAFBD1D
S1213F952DA6BD2DD37D27022706CE3F2ABD33ABBD3C077E3D343009BC3BD62613FEA9
S1213FB32642A600810D2707812C27037E3C957F3CE832367D3BD827037E3C95812840
S1123FD126037E3C9532802A2C037F3C3E811DC4
S1213FF02EF9CE3FFA7D257827 03CE403448270408 4A26FCEE00AD0020B1332E3337FD
S1214FFE334F333B3353333F3C3E3C3E3C3E3C3E3C3F3C3E3C3E3357 31E73E2D31D352
S1214401C31DB31DF31E3334333473 34B55F9560D561B5629563756453 3DF33C931FFDE
S1214403A33DA322233E231FF31F331F/31FB321A321E32033C3E320A320E31D331EB6F
S1214058 3J7732163343333473 34B55F9560D561B5629563756453212000000000000D9
S1214076000000000000000000000000000000000000000000000000000000000000028
S1214094000000000000000000000000000000000000000000000000000000000000A
S12140B2000000000000000000000000000000000000000000000000000000000000EC
S1124 0D000000000007F2578BD3C47C64BBD3380
S12140DF9C7E2937000000000000007F40E37F40F47F40E7B740E8C12A2712C12B2645
S12140F108B740E7F740E92014C12D27F4202DFE25/3FF40E38602B740E773257 9FEE4

```
S121411B26F1A600812027088100270481202608FE40F3FF439A5F39BD372FB740F8FA
S1214139B140F72606CE02045F5939810227158101271181242 6EE7D40E727E9F74084
S121415F9B740F7208DC1032611F62644C1042F05CE021020D4BD4214203BC109261C
S121417511F62644C1052F05CE021020BFBD426C2026C1012704810126AFBD3A5CC5F1
S121419380261205402705732 5/9200EC51027037325 7A2005CE02112094FF40E57D76
S121418140F7260FF40E5FF40E3B640F8B740E77E411AB640F9812B2608CE40E3BD4F
S11241CF434B20F8812D2608CE40E3BD435E20FC
S12141DE1C812A2614B640E3F640E4CE40E5BD42C3B740E3F740E420C4812F27037F25
S12141FC413EB640E3F640E4CE40E5BD4300B740E3F740E420A90000FE26427F421245
S121421A7F4213F62644109085A26FCFF62644BD30B742135A2726098D27484848BA5A
S1214238421 3B742135A271 6098D17B742125A270D098D0E4848484 8BA4212B7421249
S1214256FF421239A6080308109 2F0280073900000000000000 7F42657F42667F42DC
S121427468 7F42697E4269FE264209F62644F/4267085A26FCFF426AE600C40F4FCEFB
S121429242 68BD42C3FB4266B942658D4AB74265F742667A426727154FC6 0ACE4268D1
S12142B0BD42C3B74268F74269FE426A0920CAFF426539373 6A60136A60036861036BA
S11242CF30A6 0358496802690124 06EB04A903CA
S12142DD8D0A6A0026EF3131313131392811 7D42FF260CFF42FDCE0210BD44E8FE420C
S12142FBFD39000000373 6A600E601373634308601 6D012B0B4C680269012B04811194
S121431926F5A700A603E6046F036F04E002A2012407EB02A9010C20010D690469 03FE
S121433760160260A0076E6A700E701EE003131313233393637A601E600AB03E902E0
S1214358D92A701E700333239393637A601E600A003E202BD42F9A701E700333239B6AE
S121437326F88580261070257527 0B7D2625270485042 6028D1B39B626F88580260 5FD
S1214391860CBD5EAF390000000000000000 03E2004CE439CB6256FF62570BD44A1
S12143AF87CE439DBD5E8F7D2625270586 2BBD5EAE7D4397260D7D43982608CE4481FC
S1124 3DBD5E8F202BCE2573BD5EA0F6439827CF
S121438D20C101270FCE439ABD5EA0CE4483BD5E8F2045CE439BBD5EA2CE4481BD5E87
S12143FA8F2037F6439726 08CF447EBD5E8F202ACE4399BD5EA2C1012608CE4481BD92
S1144185F8F2018C102260FCE439BBD5EA2CE4484BD5E8F2006CE439ABD5EA07D258F
S1214436702704864320 1D7D257B2704865820147D257C2704864E200B7D25792704D1
S121445486522002862 0BD5EAEBD5EA4FE26F4A60036BD5EAE0832810D26F4860ABD32
S1214725EAF7F43987F43977F2579392020202020202004FF44C8CE44BD7F44C7AB
S1214490E001A200250 57C44C720F5EB01A90036FF44CAFE44C8B644C78B30A700328A
S12144A408FF44C8FE44CA08088C44C726D139271003E80064000A0001000000000065
S1124 4CC00000D0A2A2A2A2A204552524F522351
S12144DB2000000020000000000000203A043637FF44CCB6265426037C2653B644CC8B06
S12144F930B744DCB644CDBD5E75B744DDB644CDBD5E79B744DECE44E0B6256FF625E0
S1214517 70BD4487CF44CEBD5E8FFE26F4A60036BD5EAE3281 0D270608BC26F126EF66
S1214535BD5F843332FF270308FF27037F2653FE44CC39BD3A5CC1FF2615FE256FFFE9
S1214553253BD39FACE5699BD33ABCE4573BD33ABCE457FBD33AB7E3E
S1214571293720455854204449475345540D204A535204449475345540DCE5B9DBD4B
S12145BF33ABB626F8B745A88A50B726F87E2937B645A8B726F87E2937007C2576CEEC
S12145AD5B9DBD33AB7E29377F2576CE5B9DBD33AB7E29377C45C6200600200 37325BA
S1124 5CB708626F8B727017D25/62619FE27082E
S12145D008FF2708CE5901B62708F62709BD4487CE58FBBD33ABBD372FC1012615BD35
S12145F63A5CC1FF2711C580270AB626F88A50B726F8200C7E4675FE256FFF2573BDCF
S1214616399FA7D25782709FE26F6A6088A08A7087D2576271ECE4645860 4B726F0BD32
S1214634 2BFBD2DA6862CBD2B98C62CBD2DA3201220434D4E6D2DA6CE2448860 3B70B
S1214652 26F0BD2DAFBD372FC1282720810D2704C12C260FCE26494FC6027D25/827D4
S1214670 74C6092070CE0303BD44E87E2937B625/8367F2576BD372F32B72576C109A2
S1214688E26E5FE26F6A6088A02A708BD426C7D45C6270108FF40E32BCEBD372FFE2677
S12146AC F6B640E3A706B640E4A707C1292703 7E473BB640E3F640E4CE26497D45C61C
S1124 6CA76197D25782710CF46DFBD42C3CE2684
S12146D9494D2B98 2006000958492B90BD4487CE2649860 5B726F0BD2DAFBD2DD3B6B2
S1214EF72701B726F87D45C62709FE26F6A6088A40A708FE2642A600812C270B810D32
S1214715270ABD372FC12C26037E45F07D25762612CE5H98B62708F62709BD4487CEF3
S1214733LB97BD33AB7E2937B6257836 7F25/8BD372F32B72578C109262 1BD426CFF30
S1214751 40F32B19BD372FFE26F6A6088A01A708A606E607CE40E3BD42C34D2A037E76
S1214769 46757E46C45FF747BDF747BE7E486B7F47BEC6FFF747BD7E486BBD4A00BD25
S1214788 372FC12726 0BCE36D8BD33ABBD498D2009FE2642FF26F1BD338AC6AFBD33FE
S12147AB9C7E2937BD4A00C6B120F3BD4A00C6B420EC0000C6FFF747BDF747BEBD4A91
S1124 7C9007F486DC101/709CE0310BD44F87E72
S12147D8486BBD3A5CC1FF2751C5082720F/339AF72578BD5DEDC65A7D47BE2702C671
S12147F6BABD339C7D44CC27037E2937E49565D2A20C62D7D47BE2702C6B7BD339C5B
S1214814CE2F86BD33AB86FFB7339ABD5DEDBD2F6E7E4956C5032712C62D7D47BE2740
S1214832 02C6B7BD339CBD2F657E4956C62D7D47BE2702C6B7BD339CCE35F38604B70D
S1214850 26F0BD2DAFBD2DA6BD2DD3BD4A4E200B7F47BEBD4A00BD372F5C
S1214486EC12927F9C1272605BD497B20F0C12F261CC62A7D47BE2702C6CF7D47BD2770
S12148 8C09C62A7D47BE2702C6C9BD339C20D0C10D26037E2937C13E26157F2578BD73
S12148A3C47C6397D47BE2702C6CCBD339C7E4956C12C27ACC15B26307F2578BD3C3D
S1124 8B847CE35F3BD33ABFE2642A6008120 2725
S12148D708CE5691BD33AB2003BD3C47C6D2BD339CFE2642A600815D276C2063C125FA
S12148F52624BD338A27037E2937C6F37D47BE2702C6F67D47BD2709C6367D47BE275F
S1214913 02C6BDBD339C203B7D47BD27037F47CDFE2642FF26F1BD5812BD3C477D4435
S1214931CE27037E2937C6307D47BE2702C6C07D2578260 5BD339C200CC6577D47BECD
S1214 4F2702C6C3BD3224FE26F109FF26F17E486B204C445820232A2B390D2046438D
S1214 96D4320270D2F2046434220243034 0DCE4960BD33ABC6337D47BE2702C6C6BDC4
S1214 98B339C8604B726F0CE3DD0BD2DAFFE270808FF2708FF5BAFB64971B72649CEA0
S1214 9A9264AB65BAFF65B6BD44878D3FBD2DD3CE496A8604B726F0BD2DAFB627BDCF
S1124 9C72D98C627BD2DA3BD523ACE496FBD33DF
S1214 9D6ABCE4972BD2DAFBD16CE5B9E8605B726F0BD2DAFBD2DD3FE26F108FF26F1A7
S1214 9F439CE2649860 6B726F07E2DAFBD372FC1282618BD338A2613C6EABD339CBD82
S1214A1273 2FC1292706FE2642FF26F139CE0334BD44E8313 17E29377F2578BD3C47C6
S1214A307D44CC2616CE2F86BD33AB7F2578BD3C477D44CC2605C654BD339C7E2937E5
S1214A4EBD3A5CC1FF270139FE256FFF2573BD39FACE4A7CBD33ABFE2642B62644B74D
S1214A6C26F0BD2DAFCE4A85BD33AB5FCE0000392042524120 2A2B340D524D422032FD
S1214A8000BD4D12BD372FC10D2608CE0204BD44E8205BBD4D40F64D0D27F0BD372FB6
S1214AA8C12826147 34D0EBD372FC127260AFE26F1A600B7439B200BBD40FABD4D60F9
S1124 0C6F64D0E270CC680F/4D10C6C0F/4D11E4
S1214AD5203CBD372FBD4D4D262A7D257A260A7D2579260 5F6439A270FC6B0F74D1034
S1214AF3C6F0F74D11BD4DE42019C690F74D10C6D0F74D11200AC6A0F74D10C6E0F/59
S1214B114D11BD4B9CBD4E487E2937BD4D12BD372FC10D2608CE0204BD44E82032BD46
S1214B2F4D40F64D0D27F0BD372FBD40FABD4D60BD372FBD4D4D262A7D257A260A7B69
S1214B4D2579260 5F6439A270FC6B0F/4D10C6F0F/4D11BD4DE42019C690F74D10C608
S1214B6BD0F74D11200AC6A0F/4D10C6E0F74D11BD4D9CBD4E487E2937BD4D12BD3730
S1214B892FC10D2608CE0204BD44F8202ABD4D407D4D0D270FC640F74D10C650F74DCD
S1214BA711BD4D732020BD40EABD4D60BD372FBD4D4D260AC670F74D10BD4FC02200864
S1124 BC5C660F/4D10BD4DBABD4E487E2937BBB1
S1214BD44D12BD372FBD4D407D4D0D2606CF0204BD44E87C4D11BD4D73BD4E487E29E2
S1214BF237BD4D12BD372FC10D2608CE0240BD44E8204C123261973 4D0EBD372FC1FD
S1214C102 7260FFE26F1A600B7439AA601B7439B2027BD40EABD4D60F64D0F261CBDB3
S1214C2F372FBD4D4D26207D257A260A7D2579260 5F6439A270AC630F74D10BD4F0274
S1214C4C200FC610F74D102005C620F74D10BD4DBABD4E487E2937BD4D12BD372FC199
S1214C6A0B26B5208EBD4D12BD372FC10D2608CE0204BD44E8200EBD40EABD4D60BD5E
S1214C88372FBD4D4D260AC610F74D10BD4E022003BD4DBABD4E487E2937BD4D12BDF5
S1214CA6372FC10D2608CE0204BD44E8 0367D25752731BD40EABD4D60FE 2573080811
S1124 CC4FF26FAB6439BF6439AB026FBF226FA74
S1214CD3C1FF26034D2B0DC100260 34D2A06CE0208BD44E8B7439BBD4DBABD4E487EFF
S1214CF12937BD4D127D2575270 3BD5E557C4397C26F9BD4372BD4E487E2937000E0
S1214D0F0000007F26F97F25797F257A7F257B7F2570 7F4398F743997F43977F4D0D0B
S1214D3D7F4D0F/F4D0F7F439A7F439B7F4D107F4D1139C1412705C1422703 9F74D2F
S1214D4B0D39C12C260BBD372FC1582604734D0F397F4D0F39C1FF260E7D2575270 32B
S1214D69BD44F87F439A73439A397D2575272 0F64399B4D0D270F81422705FA4D109E
S1214D872003FA4D11F74399BD5E557C4397BD43727C26F9397D2575273CF64399B6AF
S1214DA54D0D27228142270 5FA4D102003FA4D11F743992011 7D2575271E7F2579F615
S1124 DC34399FA4D10F74399BD5E557F6439BBDD6
S1214DD25E557C43977C4397BD43727C26F97C26F9397D2575275 5F64399B4D0D27E3
S1214DF01F8142270 5FA4D102003FA4D11F74399200E7D2575273 7F64399FA4D10F72B
S1214E0F4399BD5E557F6439ABD5E557F6439BBD5E557D257A2704C64D2007D2579 27EC
S1214E2C05C652BD5F697C43977C43977C4397BD43727C26F97C26F97C26F939B6256A
S1214E4A74F62573BB26F9C900B7257 4F7257339BD4D12BD5E41BD372FC1012708CE2F
```

```
$1214F680216BD44F8205D7D2575264 1BD39FAFE26F6FF4ED2BD372FC12C26E3BD37F6
$1214E867FBD40FAC1FF27DCFF4ED286BFA4088A10A708B6270CA706B6270CA707B603
$1214EA4439BF6439AB82706F927C5B72706F72705BD0A5CFE26F6EE06FF439A7343.34
$1124FC29873257A7C43977C4397BD43727F296E
$1214ED1370000BD4D12BD5F417D2575260FFE2573FF257173257 5BD5EHE7E29027D6D
$1214EFF26F3271AFF26F4BD33ABFE2573FF2571FE589D8C589C2C06CE0306BD44E804
$1214FDNFE2571BC2573270 6CE0220BD44E8B626F86580260B7D26F3260 3BD4377BD9C
$1214F2B438AB626F8852077037F5049CE503BFF50447F5048B626F8847FB726F8FL30
$1214F49208AFF26F62010BD3A99BC208A26087D504826 3A7E5049E600C12027EAE6F3
$1214F6708C1FF27F484FFB75048FF5046FF39E7FE5044FF39E5C606F739F9CE39E54D
$1214F85BD2BD72205FF504620C1FE5046FF504420B9C606FE5044A600BD5FAE085A7E
$1214FA324F7BD5EA4FF2F03A600E601CE439CBD4487CE439CBD5F8FFE2E03BD5EA0DD
$1124FC1FF50FF600C580270586 50BD5EAEC515
$1214FD040270C86527D26F3270286 53BD5EAEC5202705864DBD5EAEC5102705864 39C
$1214FEFBD5EAEC508270A7D26F3260 58658BD5FAEC5042705864EBD5EAE7D26F3272 3
$121500C12C50327058644BD5EAEC5082705864 6BD5EAEA6002A0B7D26F3260 6CE50A0
$121502A7BBD5F8FFF50FFC4FFE700BD5E847E4F375B5B5B5B5B5B0000000000000C2
$121504800CF5096B62703F62704BD4487B62705F62706CE50B5BD4487BD438ACE5001
$121506686BD5F8FB626F843854827 03BD5ECFBD5ECC7E2749205245444546494E45C9
$121508444040D0A0D0A5448455245205745 2453A2000000000002045 52524F525372
$12150A20D0A0D0A434F4D4D4F4E204C454E475448 3D2000000000002E0D0A0400006D
$1125 0C0BD4D12BD5E41BD372FC1012708CE0281
$121500CF16BD44E820387D2575274 2BD3A5CCFF2608CE0211BD44E82024FF439AFEBF
$121500D26F6A6088A04A7088D28F6439ABD5E55F6439BBD5E55C652BD5E69C64EBDF1
$121510B5E697343987 3257C7C43977C4397BD43727E2937FE26F68606E60036FF514B
$121512936BD5F5532FF5136084A26EF390000BD4D12FE26F6FF518F7D25772743BD12
$121514727FC10D2608CE0216BD44E8201FF27F47D2575261FFE518FA61F
$121516508 7D2579260984BF7D257A27028A10A708B6439BA707B6439AA7067343983 5
$121518 37C43977C4397BD43727E2937 0000BD4D12BD5E41BD372FC101270BCE021699
$12151A1BD44E8BD4372204 77C26F97C26F97C26F97D2575260EBD39FAFE26F6A60856
$11251BF8A08A7082028C67EF74399BD5E55BD10
$121510 305CBD511FC658BD5F697F439A7F439B7C43977C43977C43977 3257BBD438C
$121517FC72BD4F487F2937BD4D12BD372FC10D2608CE0216BD44E8201ABD40EABD4DC9
$121520A607C26F97D2575270FF6439BBD5E557C43977C4398BD4372BD4E487E2937A6
$121522 8BD4D12BD372FC1272708CE0204BD44E8206 0FE26F1E600C10D27EFF7439BD
$121524 67D2575271 27D26F3270A37BD2DA332BD2D98200 3BD5E55FE26F108FF26F1F1
$121526 47C26F9F600C1272706C10D26D5201BE601C1272 6157D26F3270A37BD2DA3F9
$121528 2323 6BD2D983308FF26F120BB7D26F3270139 7C43987C4397BD4372BD4E488E
$121520A7E2937BD4D12BD372FC10D2608CE0216BD44E8 039BD40EABD4D607C26F9BF
$112520F7C26F97D2575272BF6439ABD5E55F6A0
$121530D439BBD5F857D25792704C65220077D257A2705C64DBD5E697C43977C439761
$121532EB7 34398BD4372BD4F487E2937BD4D127C2578BD372FC10F26A98609B726F954
$121530 97D25/527F3FE398BFF439ACF3988BD5124 20C9BD4D127F53847F5385CE562A
$121537 9BD33ABBD372FC1012709CE0340BD44E87F2937BD3A5CC1FF26F0BD39FA26
$121534 5BD39927E53A7BD4D12BD5F41BD372FC10D2608CE0216BD44E8201EBD40EABB
$121536 3C1FF27F1BD54D87D2641270FFE439A2605 7F2641200586FFB72641BD437227
$121538 17E2937 0000BD4D12BD43727F53847F5385 7D2575263 47D2577260B735385EB
$121539FE0226BD44E82024FF26F6 6620A708B62626A706B62627A707BD372F810452
$11253BD77F9FF2642A6008143260 3735384B1BD
$121530C2B3CFF256F08FF256FBD4372FF26F4A600812A260A7D538427E5BD544C2043
$121538FA607F2577FF26F4A600812 02706BD372F732577BD372F8604B739E9FF2642FC
$121540 8FF39E5CF548D7D26F3270 3CF5491FF39E7CF39E5BD2DD7260D7D257 27 0EF6
$121542 76CE0237BD44F82006BD544C7F53CB7D26F326057D2575260B8617FE2626A7CE
$121544 40008FF26267F29377D26F326057D2575263 67D5385263 1FE26F4FF26F1FED9
$121546 226F1A60008FF26F1FF2626BC2086261 0860DA70008FF2626CE0228BD44E8FD
$121548 0200AA70008FF2626810D26B5394D454E4446454E44BD4D12BD5E41BD372F48
$121549FC1012709CE0216BD44E87E510D7D2575260 6BD39FA7F50D5F62705BD5E55E7
$11254BCF62706BD5E55C650BD5F697E50D5BD50
$121540B4D12BD5F418D1DBD43727E2937 07360FBF257FFE263F8C263727 20BE263FA5
$121548F9B6261136201F07360FBF257FFE263F8C263F2709BE263F32B726412008CF73
$121550 70254BD44F82003BF263FBF257F3 20639BD4D12BD5E417D2575270BF626FC50
$121552 5270 4BD448A7E26FC7F2937BD4D12BD372FC10D2608CE0216BD44E82018BD2F
$121554 340FAC1FF27F47D2575270F1B055727C43977C43977 34398BD4372B62574F6BF
$121556 12573BD499BF9439AB72574F7257 37F29375FFF439AFF26FABD5E55CE26FA7D
$121557F092706FF26FA5F20F1390000BD4D12FE26F4FF26F1A60081202609CE021369
$121559NBD44E87F55F3BD372FBD3A5CC1FF2603BD39FABD3992FE26F6FF5589BD377B
$11255BD3FBD372FBD40FAC1FF2605CE021620B3
$121550AD0FF5589A6087D2579260984BF7D257A27028A10A708B6439AA706B6439B75
$121555F8A707734398 7C43977C4397BD43727E2937CE5604BD331CFF568F204F2042C0
$121560645512 02A2B370DCF5612 20EA2042474520 2A2B370DCE562020DC204247540F
$121562 4202A2B370DCE562E20CE20424C45202A2B370DCE563C20C0204 24C54202A33
$121564 2B370DCF564A20B220424E45202A2B370D7D2578270DBD323FBD3C28C66C1A
$121566BD3224200F7F568E73568EC618F740F5BD32FDFF568FBD33ABCE5691BD3324
$121567LABCE5699BD33ABCE56AABD33AB7F32D0000000204C445820233 00D204252E2
$121569C41202A2B350D204C4458202331 0D204C4458202324464646460DCE56BFBD37
$11256BA33AB7F2937 2052545 30DC6FBBD339CBC
$121560 97E2937BD5812BD3C477D44CC261BBD3B524D260F8C56E527037E5748BD37DE
$121562 7 4C1012709CE0308BD44F87E2937CE573FBD33ABCE5BA48604B726F0BD2DDG
$121570 5AFBD2DA6BD2DD3BD3992BD372FC10D27DCBD3B524D26D08C56EC26CBBD37CA
$121572 32FC10126C4CE5BA48604B726F0BD2DAFBD2DA6BD2DD3BD39927F29372042B7
$121574 4451202A2B350D7326FDFF270808FF2708FF26FECE5809BD33ABCE5901B635
$121575F26FFF626FFBD4487CE58FBBD33ABCA70BD2DA38620BD2D987E29C7204245F5
$121577 4512 02FC00000000000CE5B9DBD33ABFF270808FF2708B62708F62709CE5B66
$121579B9BBD4487CE5B97BD33ABFE270808FF2708073 60FBF3BD4BE589DB627083626
$11257B9B627093630 09BF3BD432068C588926F0
$121570 609CE0305BD44E87E2937FF589DBD5812BD3C47CE5809BD33ABCE5BA4860 49D
$121577 6F726F0BD2DAFCE5801BE6270 8FC2709BD4487CE5800BD33AB20CF2E000000A3
$121580 40000 2B330D20424E45202A2B350DFF26F1FF3C0586FFB72578BD372FC10D4C
$121582 2227 36C1012646BD3B524D263 3FF2642A60081 2E2623C580261FC54026DEC191
$121584 02C27DAC13A27D6C13B27D2C13D27CFC13E27CAC13F27C67F2578FF3C05FF02
$121585F26F139BD3A5CC1FF27FC50826EE20E9810226 11C10327E17D398226DF7D85
$121587C398427DA7F581D5D27D47E581D0000000000000000000000000000000000E
$121589A0000000000CE5B9DBD33AB07360FBF3BD4BE589D32B740F632B740F5300968
$11258B8BF3BD432068C589C2C09CE0311BD4440
$121580 7E87E2937FF589DB640F5F640E6CE58F6BD4487FE40E509FF40F5B640E5F649
$121588E540F6CE5901BD4487CE58F5BD33AB20D37E0000000000204A4D50202E00009F
$121590 300000000D204C4458202 32F0000000000D204C4441204 1202E414343554DF6
$121592 12B310D204445432041 0D204247542 02A2B340D204A4D502D302C580D20499D
$121593F4F580D204350582023 2E0000000000 2D330D204245 1202A2D31300D20426B
$121595 5241203A2D31350D7F2578BD3C477F32F27332F2BD32A7FE270808FF5BAD63
$121597B08FF5BAFF2708B65BADF65BAFCE590FBD4487B65BAFF65BB0CE5949BD441F
$121599987CF5907BD33ABCE5914BD33ABCE5924BD33ABCE592BBD33ABCE5934BD33A8
$11759B7ABCE593DBD33ABCE593DBD33ABCE590D
$121590 3BD33ABCE593DBD33ABCF593DBD33ABCE5942BD33ABCE5951BD33ABCE59AB
$121590F45BD33AB7F2937CE5BA8860 4B726F0BD2DAFBD372FC1012 66DFF2642FF5BD4
$121500A2DB6264 4B75BAFB726F0BD2DAFBD2DD3BD372FC10D2704C12824D8604B718
$121500A2026F0CF3BD0BD2DAFFE270808FF2708B64971B7264 9CE264AB62708F62709 38
$121500A3EBD4487BD49F5BD2DD3FE2642A600810D273CBD372FC1012716C12C260FCEF 7
$121500A5C5AD7BD33ABBD372FC1292724 20E97E5BE18604B726F0CE3DDABD2DAFBD2DB2
$121500A70A6BD2DD3BD3992BD372FC12C27C8C12926DFCE5ACCBD33ABB65BAFB72644C 2
$121500A98FF5BADFF2642BD3A5CC1FF261AFE256FFF2573BD39FA8604B726F0CE5ACBCC
$11250AB6BD2DAFBD2DA6BD2DD3FE2708FF5BAFC1
$121500AC57F5C67204558542046444 2204246464640D204644422 0300DCE5B83860499
$121500AE3B726F0BD2DAFBD372FC1012709CE0335BD44F87F2937BD2DA6BD2DD3FF25F9
$121500B016FFF257 3BD39FABD3992BD2DA6C6F0BD339C8604B726F0CF3DDOBD2DAFFE69
$121500B1F270808FF2708B64971B7264 9CE264AB62708F62709BD4487BD49F5BD2DD3FO
$121500B3BBD372FC10D2736C12C27F5C12 2726A6BD372FC1012 70A0C12C27F5C12 2723209 7BD3A40
$121500B5D5C1FF26F7DD39FABD3992FF26F6A6088A80A708BD2DA6CE4A85BD33AB200F
$121500B79CFCE5ACCBD33AB7F5ABF20454F54CF5BA4FF56B12024CE5BA8FF56B1201CDB
$121500B972FD0000000000204555 5202A0D204A4D50204A535200000000000000BD3752
```

```
S1125BB52F7F5BACC101275CC1282729FE264244
S1215BC4A60081402617BD3C47FE5BB18604B726F0BD2DAFCE5939BD33AB7F2937CE3A
S1215BE20309BD44E87E2937CE5B9E7C5BAC8665B726F0BD2DAFBD2DD3B65BADF65DC7
S1215C00AFCE5B98BD4487860 6B726F0CE5B97BD2DAFBD372FC10126C8FE5BB1860472
S1215C1B726F0BD2DAFBD2DA6BD2DD3BD39927D5BAC2747BD372FC12C26278604B799
S1215C3C26F0CE595BD2DAF65BAFF65BB0CE5B98BD4487CE5B978606B726F0BD2D08
S1215C5AAFBD2DD32OB2C12927037F5BE1B65BAFF65BB0CF5B98BD4487CE5B97BD3362
S1215C78AB7E29377F339ACE3D6A8604B726F0BD2DAF7D270226DA8623BD2D98C623E6
S1215C96BD2DA3BD2DA6BD2DD3B63BCDF63BCECE329BBD4487CE328FBD33ABFF3BCD02
S1125CB4FF40F50808FF3BCDBD372FC128271758
S1215CC37D339A270139BE3BD4CE03047D339A2703CE03137E44E8B640E5F640E63643
S1215CE137D63BCFF63BDO3637B63BD136B6257B367F257B30090909FF3BD67F3BD2B8
S1215CFF/C3BD2BD3C6232B7257832B73BD13332B73BCFF73BD0BD3C28FL2642A600CF
S1215D1B12C270681292769 20A2B63BD18501279BB63C05F63C06363 7B63BCFF63BF7
S1215D3BD03637F62578377F2578 30090909FF3BD6BD3C6233F72578BD3C28CE5DD585
S1215D59BD33ABCE32EEBD33ABCE3D703332BD4487CE3D6ABD33ABC612BD339CCE3228
S1215D7/EEBD33AB3332CE3289BD4487CE327DBD33ABC60CBD339CCE5DD5BD33ABCE2C
S1215D9533FEBD33AB7D45C62619CE5DDA7D2578270 3CE5DE2BD33ABC612BD339CCE47
S1125DB332FEBD33AB3332B740F5F740E6CE3204
S1215DC289BD4437CE327DBD33ABC60CBD339C7F3BD239204445580D204C445820231A
S1215DE0320D204C445820239 90D7F339ACE0000FF3BCD7F3BD37F3BD27F2707BF3BF0
S1215DFD4FF2642A600BD39C585802706BD3B524D2606CE03137E44E8BD4A4EFF3BD6
S1215F1CCFF73BD1B6257836C403F/2578F63BD1C440F745C6F63BD1C480F72702BD7E
S1215E3A5C7F32B72578397D2577270E7D2575260 3BD3AB1CE0223BD44E839B626F88D
S1215E588540270139178D15BD5EB6178D137E5EB8B626F88540270139177E5FD844E2
S1215E76444444840F8B30813923028B0739CE5E8920060D0A048D2008A6008104264A
S1215E94F/7017A6008D0DA60008200C8DF5BDF386202006 8DCB20028DCB367D2653D8
S1125EB22703BD270D32367D26542703BD271045
S1215EC132397F63FD7E643B7F64D37E64597F646E7E630B7E64877E64B77E630D7EBF
S1215FDF647300004453 4B00005F0B0000000000000000000000000000000000000007E
S1215EFD0000000000000000000000000000000000000000000000000000000000000083
S1215F1B0000000000000000000000000000000000000000000000000000000000000064
S1215F390000000000000000000000000000000000000000000000000000000000000046
S1215F570000000000000000000000000000000000000000000000000000000000000028
S1215F750000000000000000000000000000000000000000000000000000000000000A
S1215F930000000000000000000000000000000000000000000000000000000000000EC
S1125FB10000000000000000000000000000DD
S1215FC0000000000000000000000000000000000000000000000000000000000000BF
S1215FDF0000000000000000000000000000000000000000000000000000000000000A1
S1215FFC0000000000000000000000000000044534B0080603500000000000008C
S1216 01A0000000000000000000000000000000000000000000000000000000000000064
S1216038000000000000000000000000000000000000000000000000000000000000046
S1216056000000000000000000000000000000000000000000000000000000000000028
S1216074000000000000000000000000000000000000000000000000000000000000A
S1216092000000000000000000000000000000000000000000000000000000000000EC
S1126 0B0000000000000000000000000000000DD
S12160BF0000000000000000000000000000000000000000000000000000000000000BF
S12160DD0000000000000000000000000000000000000000000000000000000000000A1
S12160FB0000000000000000000000000000000000000000000000000000000000000083
S1216119000000000000000000000000000000000000000000000000000000000000064
S1216137 4C5054008061 5F0000000000000000000000000000000000000000000000016
S1216155000000000000000000000000000000000000000000000000000000000000028
S1216173000000000000000000000000000000000000000000000000000000000000A
S1216191000000000000000000000000000000000000000000000000000000000000FC
S11261AF000000000000000000000000000DD
S12161BF0000000000000000000000000000000000000000000000000000000000000BF
S12161DC0000000000000000000434F4E000062 0D000000000000000000000000000052
S12161FA0000000000000000000000000000000000000000000000000000000000000083
S1216218000000000000000000000000000000000000000000000000000000000000064
S1216236000000000000000000000000000000000000000000000000000000000000046
S1216254000000000000000000000434F4E0080 6287000000000000000000000000000F
S1216272000000000000000000000000000000000000000000000000000000000000000A
S1216290000000000000000000000000000000000000000000000000000000000000000EC
S11262AF000000000000000000000000000DD
S12162BD0000000000000000000000000000000000000000000000000000000000000000BF
S12162DB0000000000000000000000000000000000000000000000000000000000000000A1
S12162F9000000000000000000000000000000000000000000000000000000000000000083
S1216317 08A727D728CE61356F05FF07860CA700083F02CE6135A727E728CE61E36F6F
S1216335056F273981047 7F8810A77F73F00CF6135EF27A7000836 3F02CE6135A72712
S1216353F7283281 0D27033F01393F13A607F608A727E7286F053F01393F00CE625D33
S1216371F27A7000836 3F02CF625DA727F7283281 0D27D8810427D43F01393 73F0532
S1216 38FCF61F36D2726173F13A607F608CB018900A727E7286F053F0633B6620D39A0
S11263DDFF27A6000836 3F02CE61F3A727E726B4
S12163D0332810D27043F0633396F273F0633394F505554204649 4C453A20044F550A
S12163DA5450555420464 94C453A2004524541442045524F520D57524954452045F3
S12163F852524F520DCF5FE16F10CE63CB3F313F30D625C10D26030A0D39DF20DF2388
S1216416CF5FE16F056F276F1F86FFA7298603A71D3F2C6D052A093F146D0526030C18
S12164340A393F1F0C0B39CE600B6F10CE63D83F313F30D625C10D26030A0D39DF2076
S12164 52DF23CF600B20C0CE5EE16F056D107 6030A0D393F156D0526CB0C0A39CF6062
S12164700B20E9CE5EF16F053F156D0526B83F146D0526B20C0A39373F05CF5EE16DF0
S121648E102606C608E705200A3F18F605770AC108260C3F06330A0D393F06330C0A08
S11264AC39BD6436CF63E63F1F3F1F373F05CF32
S12164BB600B3F196D05270ABD6436CF63F13F1F3F1F3F06330C0A3964D5000000025
S10A64D900000000000000B8
S9
```

```
LOAD LIMITS:
2000 64FD

LOAD MAP
STRUBA 2000
UNLF   5E84
PDATA1 5E8F
CPDRVR 5EE1
MONITOR 633F
INITIO 6341
OUTLPT 636D
OUTEFE 63A0
INPHE  63C0
OPENI  6426
OPENO  6464
CLOSI  6482
CLOSO  6497
REWIND 649C
GETB   64B0
PUTB   64E0
TABLES 64FC
```

```
0001  0000 0000   N        NAM RUNTIM  (STRUBAL +)
0002                 *
0003                 * MODIFIED FOR CP/68
0004                 *
0005                 * STRUBAL + COMPILER RUN TIME PACKAGE
0006                 * COPYRIGHT 1978 BY ROBERT GRAPPEL, LEXINGTON, MASS
0007                 *                 AND JACK HEMENWAY, BOSTON, MASS
0008                 *                 ALL RIGHTS RESERVED
0009                 *
0010                 * 16-BIT SIGNED ARITHMETIC FUNCTIONS
0011                 * ALL UNARY FUNCTIONS USE X-REGISTER FOR ARGUMENT
0012                 * BINARY FUNCTIONS USE X-REGISTER FOR 2ND ARGUMENT
0013                 * AND USE MEMORY LOCATION "ACCUM1" FOR FIRST ARGUMENT
0014                 * EXCEPTIONS: ASCII,MID, AND POKE FUNCTIONS USE "ACCUM3"
0015                 *              FVAL,STRCMP,MID, AND DIGSEI USE "ACCUM4"
0016                 *
0017                 * RETURN IN X-REGISTER FOR ALL FUNCTIONS
0018                 *
0019                 *
0020                 * 9 BYTE FLOATING-POINT ARITHMETIC FUNCTIONS
0021                 * 7 BYTE MANTISSA (14 BCD DIGITS)
0022                 * 3 DIGIT BCD EXPONENT
0023                 * LINKAGE LIKE 16-BIT INTEGER FUNCTIONS
0024                 * EXCEPT RETURNS IN MEMORY
0025                 *
0026                 *
0027  0000 0000   C        CMN RUNSTK,80
0028  0000 0050   C        CMN ACCUM1,2
0029  0000 0052   C        CMN ACCUM2,2
0030  0000 0054   C        CMN ACCUM3,2
0031  0000 0056   C        CMN ACCUM4,2
0032  0000 0058   C        CMN ACCUM5,9
0033  0000 0061   C        CMN ADRTO,2
0034  0000 0063   C        CMN ADRFM,2
0035  0000 0065   C        CMN  EOF,2
0036  0000 0067   C        CMN  ERR,2
0037  0000 0069   C        CMN .CURIO,2
0038                 *
0039  0000 0651   N        ENT DIVERR
0040  0000 0C62   N        ENT FIX
0041  0000 0DAD   N        ENT FABS
0042  0000 0DD8   N        ENT FNEG
0043  0000 0DB4   N        ENT FSIGN
0044  0000 0706   N        ENT ILLERR
0045  0000 080B   N        ENT ASCII
0046  0000 0D87   N        ENT FCOMPR
0047  0000 0CDD   N        ENT FLOAT
0048  0000 0FE9   N        ENT INITIO
0049  0000 10A0   N        ENT LPON
0050  0000 10A7   N        ENT LPOFF
0051  0000 0409   N        ENT PDATA1
0052  0000 03A1   N        ENT INEEE
0053  0000 0411   N        ENT CRLF
0054  0000 0006   N        ENT .RTBL
0055                 *
0056                 *
0057  0000 7E 106C R SYSIN  JMP PUTIN     GET A CHARACTER
0058  0003 7E 1048 R SYSOUT JMP PUTOUT    PUT A CHARACTER
0059                 *
0060                 *
0061                 * "DISPATCH" ROUTINE
0062                 *
0063                 * DISPATCH TABLE AT "BRTABL"
0064                 * OFFSET BYTE FOLLOWS CALL
0065                 *
0066  0006 FF 002A R  RTBL  STX RTEMP     SAVE X-REGISTER
0067  0009 30               TSX          GET STACK POINTER
0068  000A A6 00            LDA A 0,X
0069  000C E6 01            LDA B 1,X    GET RETURN ADDRESS
0070  000E CB 01            ADD B #1
0071  0010 89 00            ADC A #0     SKIP OFFSET BYTE
0072  0012 A7 00            STA A 0,X
0073  0014 E7 01            STA B 1,X
0074  0016 FE 00            LDX 0,X      INDIRECT ADDRESS
0075  0018 09               DEX
0076  0019 E6 00            LDA B 0,X    GET OFFSET BYTE
0077  001B 4F               CLR A
0078  001C FB 0029 R        ADD B TBL+1
0079  001F B9 0028 R        ADC A TBL    ADD OFFSET TO TABLE ADDRESS
0080  0022 37               PSH B
0081  0023 36               PSH A
0082  0024 FF 002A R        LDX RTEMP    RESTORE X-REGISTER
0083  0027 39               RTS          RETURN TO NEW ADDRESS ON STACK
0084                 *
0085  0028 002C     R TBL   FDB BRTABL
0086  002A 0002       RTEMP RMB 2
0087                 *
0088                 *
0089  002C 002C     R BRTABL EQU *
0090                 *
0091  002C 7E 012B R        JMP NOT      LOGIC OPERATIONS
0092  002F 7E 0143 R        JMP ANDZ
0093  0032 7E 014D R        JMP EXOR
0094  0035 7E 0157 R        JMP OR
0095  0038 7E 0161 R        JMP ADDZ     ARITHMETIC OPERATIONS
0096  003B 7E 0170 R        JMP SUBZ
0097  003E 7E 0195 R        JMP MULT
0098  0041 7E 0214 R        JMP DIV
0099  0044 7E 017F R        JMP COMPAR
0100  0047 7E 0205 R        JMP ABS
0101  004A 7E 0276 R        JMP SIGN
0102  004D 7E 0286 R        JMP MOD
0103  0050 7E 028D R        JMP EXP
0104  0053 7E 02DD R        JMP RAND
0105  0056 7E 0416 R        JMP LFEED    I/O OPERATIONS
0106  0059 7E 04C5 R        JMP INPUT
0107  005C 7E 0617 R        JMP OUTPUT
0108  005F 7E 0892 R        JMP QOUT
0109  0062 7E 08B9 R        JMP STRIN
0110  0065 7E 08CC R        JMP SPACES
0111  0068 7E 07C7 R        JMP LEN      STRING OPERATIONS
0112  006B 7E 07EF R        JMP VAL
0113  006E 7E 080B R        JMP ASCII
0114  0071 7E 083B R        JMP MID
0115  0074 7E 087B R        JMP XFER
0116  0077 7E 0958 R        JMP DELAY    DELAY OPERATION
0117  007A 7E 02BC R        JMP STRCMP   STRING COMPARE OPERATION
0118  007D 7F 0138 R        JMP PEEK     BYTE I/O OPERATIONS
0119  0080 7E 013D R        JMP POKE
0120  0083 7E 09ED R        JMP FPRINT   FLOATING-POINT OPERATIONS
0121  0086 7E 0B02 R        JMP FREAD
```

```
0122  0089 7E 0C62 R        JMP FIX
0123  008C 7E 0CDD R        JMP FLOAT
0124  008F 7E 0DA0 R        JMP FABS
0125  0092 7E 0DB4 R        JMP FSIGN
0126  0095 7E 0DD8 R        JMP FNEG
0127  0098 7E 0D87 R        JMP FCOMPR
0128                  *
0129                  * FLOATING-POINT ROUTINES
0130                  *
0131  009B 7E 0000 X        EXT FTAREG
0132  009E 7E 0000 X        EXT FTCREG
0133  00A1 7E 0000 X        EXT FFDREG
0134  00A4 7E 0000 X        EXT FADD
0135  00A7 7E 0000 X        EXT FSUB
0136  00AA 7E 0000 X        EXT FMUL
0137  00AD 7E 0000 X        EXT FDIV
0138                  *
0139                  * SCIENTIFIC FUNCTIONS
0140                  *
0141  00B0 7E 0000 X        EXT SIN
0142  00B3 7E 0000 X        EXT COS
0143  00B6 7E 0000 X        EXT TAN
0144  00B9 7E 0000 X        EXT ASIN
0145  00BC 7E 0000 X        EXT ACOS
0146  00BF 7E 0000 X        EXT ATAN
0147  00C2 7E 0000 X        EXT LOG10
0148  00C5 7E 0000 X        EXT NATLOG
0149  00C8 7E 0000 X        EXT TENEXP
0150  00CB 7E 0000 X        EXT FEXP
0151  00CE 7E 0000 X        EXT XTOY
0152                  *
0153                  * FLOATING-POINT MOVE UTILITY
0154  00D1 7E 0421 R        JMP ZMOVER
0155                  * FLOATING-POINT "VALUE" CONVERSION
0156  00D4 7E 0AF2 R        JMP FVAL
0157                  * FLOATING-POINT "ASCII" CONVERSION
0158  00D7 7E 082A R        JMP FASCII
0159                  *
0160                  * FILE I/O ROUTINES
0161                  *
0162  00DA 7E 0000 X        EXT OPEN
0163  00DD 7E 0000 X        EXT CLOSE
0164  00E0 7E 0000 X        EXT REWIND
0165  00E3 7E 0728 R        JMP IREAD
0166  00E6 7E 0734 R        JMP FLREAD
0167  00E9 7E 0740 R        JMP SREAD
0168  00EC 7E 0780 R        JMP IWRITE
0169  00EF 7E 0791 R        JMP FWRITE
0170  00F2 7E 07A5 R        JMP GWRITE
0171  00F5 7E 0767 R        JMP SRECRD
0172  00F8 7E 07BB R        JMP FSPACE
0173  00FB 7E 07AF R        JMP FLFEED
0174                  *
0175                  * SET UP FORMATTING FOR PRINT
0176  00FE 7E 0963 R        JMP DIGSET
0177                  *
0178  0101 7E 0000 X        EXT MOVER
0179                  *
0180                  * PUSH AND POP ROUTINES
0181  0104 7E 0313 R        JMP PUSHI
0182  0107 7E 0326 R        JMP PUSHF
0183  010A 7E 0339 R        JMP PUSHS
0184  010D 7E 0358 R        JMP POPI
0185  0110 7E 0368 R        JMP POPF
0186  0113 7E 037B R        JMP POPS
0187                  *
0188                  * FILE I/O DRIVERS
0189                  *
0190  0116 7E 075D R        JMP CURIOB      SET UP IOB ADDRESSING
0191  0119 7E 042F R        JMP STOP        RETURN TO MONITOR
0192                  *
0193                  * PARAMETER TRANSFER FOR 'PROCEDURE'
0194  011C 7E 08E6 R        JMP PARAM
0195                  *
0196                  * FORMATTED STRING OUTPUT
0197  011F 7E 089B R        JMP STROUT      (CONSOLE)
0198  0122 7E 079B R        JMP SWRITE      (FILE)
0199                  *
0200                  *
0201  0125 7E 0000 X        EXT GETB        GET A BYTE
0202  0128 7E 0000 X        EXT OUTB        PUT A BYTE
0203                  *
0204                  *
0205                  * LOGICAL "NOT" FUNCTION
0206                  *
0207  012B FF 0052 C NOT    STX ACCUM2
0208  012E 73 0052 C        COM ACCUM2
0209  0131 73 0053 C        COM ACCUM2+1
0210  0134 FE 0052 C        LDX ACCUM2
0211  0137 39               RTS
0212                  *
0213                  *
0214                  * BYTE INPUT FUNCTION
0215                  * ADDRESS OF BYTE IN X-REGISTER
0216                  *
0217  0138 4F         PEEK  CLR A
0218  0139 E6 00             LDA B 0,X
0219  013B 20 4E             BRA FINISH
0220                  *
0221                  * BYTE OUTPUT FUNCTION
0222                  * ADDRESS OF BYTE IN X-REGISTER
0223                  * DATA BYTE IN LOW ORDER OF ACCUM3
0224                  *
0225  013D B6 0055 C POKE   LDA A ACCUM3+1
0226  0140 A7 00             STA A 0,X
0227  0142 39               RTS
0228                  *
0229                  * LOGICAL "AND" FUNCTION
0230                  *
0231  0143 8D 3C      AND7  BSR SETUP
0232  0145 B4 0052 C        AND A ACCUM2
0233  0148 F4 0053 C        AND B ACCUM2+1
0234  014B 20 3E             BRA FINISH
0235                  *
0236                  * LOGICAL "EXCLUSIVE-OR" FUNCTION
0237                  *
0238  014D 8D 32      EXOR  BSR SETUP
0239  014F B8 0052 C        EOR A ACCUM2
0240  0152 F8 0053 C        EOR B ACCUM2+1
0241  0155 20 34             BRA FINISH
0242                  *
0243                  * LOGICAL "OR" FUNCTION
```

```
                        *
0157 8D 28      OR      BSR SETUP
0159 BA 0052 C          ORA A ACCUM2
015C FA 0053 C          ORA B ACCUM2+1
015F 20 2A              BRA FINISH
                        *
                        * ARITHMETIC "ADD" FUNCTION
                        *
0161 8D 1E      ADDZ    BSR SETUP
0163 FB 0053 C          ADD B ACCUM2+1
0166 B9 0052 C          ADC A ACCUM2
0169 28 03              BVC *+5         OVERFLOW?
                        *
016B 8D 06A8 R          JSR OVERR       YES
                        *
016E 20 1B              BRA FINISH
                        *
                        * ARITHMETIC "SUB" FUNCTION
                        *
0170 8D 0F      SUBZ    BSR SETUP
0172 F0 0053 C          SUB B ACCUM2+1
0175 B2 0052 C          SBC A ACCUM2
0178 28 03              BVC *+5         OVERFLOW?
                        *
017A 8D 06A8 R          JSR OVERR       YES
                        *
017D 20 0C              BRA FINISH
                        *
                        * "COMPARE" FUNCTION - LIKE SUBTRACTION
                        *
017F 20 EF      COMPAR  BRA SUBZ
                        *
                        * BINARY SETUP
                        *
0181 B6 0050 C  SETUP   LDA A ACCUM1
0184 F6 0051 C          LDA B ACCUM1+1
0187 FF 0052 C          STX ACCUM2
018A 39                 RTS
                        *
                        * BINARY CLEANUP
                        *
018B B7 0052 C  FINISH  STA A ACCUM2
018E F7 0053 C          STA B ACCUM2+1
0191 FE 0052 C          LDX ACCUM2
0194 39                 RTS
                        *
                        * ARITHMETIC "MULTIPLY" FUNCTION
                        *
0195 8D 1D      MULT    BSR SETUPS
0197 CE 0010            LDX #16
019A 58         MPY1    ASL B
019B 49                 ROL A
019C 78 0053 C          ASL ACCUM2+1
019F 79 0052 C          ROL ACCUM2
01A2 24 0B              BCC MPY2
                        *
01A4 FB 0051 C          ADD B ACCUM1+1
01A7 B9 0050 C          ADC A ACCUM1
01AA 28 03              BVC *+5         OVERFLOW?
                        *
01AC 8D 06A8 R          JSR OVERR       OVERFLOW ERROR
                        *
01AF 09         MPY2    DEX
01B0 26 E8              BNE MPY1
01B2 20 27              BRA FINSHS
                        *
                        * BINARY SETUP WITH SIGN
                        *
01B4 7F 01F5 R  SETUPS  CLR SIG         CLEAR SIGN
01B7 FF 0052 C          STX ACCUM2
01BA 7D 0050 C          TST ACCUM1      CHECK SIGN OF FIRST ARG
01BD 2C 08              BGE SET2        BRANCH IF PLUS
                        *
01BF 73 01F5 R          COM SIG         SET SIGN
01C2 CE 0050 C          LDX #ACCUM1
01C5 8D 2F              BSR NEGATE      MAKE POSITIVE
                        *
01C7 7D 0052 C  SET2    TST ACCUM2      CHECK SIGN OF 2ND ARG
01CA 2C 08              BGE SET3        PLUS
                        *
01CC 73 01F5 R          COM SIG         SET SIGN
01CF CE 0052 C          LDX #ACCUM2
01D2 8D 22              BSR NEGATE      MAKE POSITIVE
                        *
01D4 B6 0050 C  SET3    LDA A ACCUM1
01D7 F6 0051 C          LDA B ACCUM1+1
01DA 39                 RTS
                        *
                        * BINARY FINISH WITH SIGNS
                        *
01DB 4D         FINSHS  TST A           OVERFLOW?
01DC 2C 03              BGE F2          NO
                        *
01DE 8D 06A8 R          JSR OVERR       YES
                        *
01E1 B7 0052 C  F2      STA A ACCUM2
01E4 F7 0053 C          STA B ACCUM2+1
01E7 7D 01F5 R          TST SIG         NEED TO INVERT?
01EA 27 05              BEQ F3          NO
                        *
01EC CE 0052 C          LDX #ACCUM2
01EF 8D 05              BSR NEGATE      CHANGE SIGN
                        *
01F1 FE 0052 C  F3      LDX ACCUM2
01F4 39                 RTS
                        *
01F5 0001       SIG     RMB 1           SIGN STORE
                        *
                        * "NEGATE" FUNCTION (CHANGE SIGN OF (X),(X+1))
                        *
01F6 A6 00      NEGATE  LDA A 0,X
01F8 E6 01              LDA B 1,X
01FA 43                 COM A
01FB 53                 COM B
01FC CB 01              ADD B #1
01FE 89 00              ADC A #0
0200 A7 00              STA A 0,X
0202 E7 01              STA B 1,X
0204 39                 RTS
                        *
                        * ARITHMETIC "ABS" FUNCTION
                        *
```

```
0205 FF 0050 C ABS     STX ACCUM1
0208 2B 01             BMI ABS2      IF PLUS, DONE
                *
020A 39                RTS
                *
020B CE 0050 C ABS2    LDX #ACCUM1
020E 8D E6             BSR NEGATE    SWITCH SIGN
0210 FE 0050 C         LDX ACCUM1
0213 39                RTS
                *
                * ARITHMETIC "DIVIDE" FUNCTION
                *
0214 BD 01B4 R DIV     JSR SETUPS    BINARY SETUP WITH SIGN
0217 FE 0052 C         LDX ACCUM2    TEST FOR DIV/0
021A 26 03             BNE DIV2      OK
                *
021C BD 0651 R         JSR DIVERR    DIV/0 ERROR
                *
021F 86 01     DIV2    LDA A #1
0221 7D 0052 C         TST ACCUM2
0224 2B 0D             BMI DIV4
                *
0226 4C        DIV3    INC A
0227 78 0053 C         ASL ACCUM2+1
022A 79 0052 C         ROL ACCUM2
022D 2B 04             BMI DIV4
                *
022F 81 11             CMP A #17
0231 26 F3             BNE DIV3
                *
0233 B7 0274 R DIV4    STA A COUNT      SAVE BIT COUNT
0236 B6 0050 C         LDA A ACCUM1
0239 F6 0051 C         LDA B ACCUM1+1
023C 7F 0050 C         CLR ACCUM1
023F 7F 0051 C         CLR ACCUM1+1
                *
0242 F0 0053 C DIV5    SUB B ACCUM2+1
0245 B2 0052 C         SBC A ACCUM2
0248 24 09             BCC DIV6         DIVISOR OK
                *
024A FB 0053 C         ADD B ACCUM2+1   DIVISOR TOO LARGE
024D B9 0052 C         ADC A ACCUM2     RESTORE
0250 0C                CLC
0251 20 01             BRA DIV7
                *
0253 0D        DIV6    SEC
0254 79 0051 C DIV7    ROL ACCUM1+1
0257 79 0050 C         ROL ACCUM1
025A 74 0052 C         LSR ACCUM2       ADJUST DIVISOR
025D 76 0053 C         ROR ACCUM2+1
0260 7A 0274 R         DEC COUNT
0263 26 DD             BNE DIV5
                *
0265 B7 0274 R         STA A COUNT      SAVE REMAINDER
0268 F7 0275 R         STA B COUNT+1
026B B6 0050 C         LDA A ACCUM1
026E F6 0051 C         LDA B ACCUM1+1
0271 7E 01DB R         JMP FINSHB
                *
0274 0001      COUNT   RMB 1        BIT COUNT
0275 0001              RMB 1        ALSO REMAINDER
                *
                * ARITHMETIC "SIGN" FUNCTION
                *
0276 FF 0052 C SIGN    STX ACCUM2
0279 26 01             BNE SIGN2     IF=0, QUIT
                *
027B 39                RTS
                *
027C 2B 04     SIGN2   BMI SIGN3     IF NEGATIVE, SKIP
027E CE 0001           LDX #1        SET=1 FOR PLUS
0281 39                RTS
                *
0282 CE FFFF   SIGN3   LDX #$FFFF    SET=-1 FOR MINUS
0285 39                RTS
                *
                * ARITHMETIC "MODULO" FUNCTION
                *
0286 BD 0214 R MOD     JSR DIV       PERFORM DIVISION
0289 FE 0274 R         LDX COUNT     GET REMAINDER
028C 39                RTS
                *
                * ARITHMETIC "EXPONENTIAL" FUNCTION
                *
028D FF 02B8 R EXP     STX POWER     SAVE EXPONENT
0290 26 04             BNE EXP2      EXP  NE.  0
                *
0292 CE 0001           LDX #1        EXP=0,  RETURN=1
                *
0295 39        EXP1    RTS
                *
0296 2B 19     EXP2    BMI EXP4      EXP .LE 0
                *
0298 FE 0050 C         LDX ACCUM1    EXP > 0
029B FF 02BA R         STX TEMP
029E 7A 02B9 R EXP3    DEC POWER+1
02A1 27 F2             BEQ EXP1      DONE
                *
02A3 BD 0195 R         JSR MULT      ARG1*ARG2
02A6 FE 02BA R         LDX TEMP
02A9 FF 0050 C         STX ACCUM1    RESTORE ARG1
02AC FE 0052 C         LDX ACCUM2
02AF 20 ED             BRA EXP3
                *
02B1 BD 0670 R EXP4    JSR NEGERR    NEGATIVE EXPONENT ERROR
02B4 FE 0050 C         LDX ACCUM1    RETURN ARG1
02B7 39                RTS
                *
02B8 0002      POWER   RMB 2
02BA 0002      TEMP    RMB 2
                *
                *
                *
                * STRING COMPARISON ROUTINE
                * ADDRESS OF FIRST STRING IN ACCUM4
                * ADDRESS OF SECOND STRING IN X-REGISTER
                * SETS PROPER CONDITION CODES
                *
02BC FF 0052 C STRCMP  STX ACCUM2
02BF FE 0056 C         LDX ACCUM4
02C2 A6 00     STRIP   LDA A 0,X       GET CHAR  FROM STRING 1
02C4 FE 0052 C         LDX ACCUM2
```

```
02C7 A1 00              CMP A 0,X       COMPARE WITH STRING 2
02C9 27 01              BEQ *+3         IF NO MATCH, DONE
                *
02CB 39         STREND  RTS             DONE!
                *
02CC 81 04              CMP A #$04      END-STRING?
02CE 27 FB              BEQ STREND      YES, DONE
                *
02D0 08                 INX
02D1 FF 0052 C          STX ACCUM2      MOVE POINTERS
02D4 FE 0056 C          LDX ACCUM4
02D7 08                 INX
02D8 FF 0056 C          STX ACCUM4
02DB 20 E5              BRA STRLP       CONTINUE
                *
                *
                *
                * "RANDOM" GENERATOR
                * IGNORES ARGUMENT UNLESS ARG=0
                * IF ARG=0, RESTART GENERATOR
                *
02DD FF 0052 C  RAND    STX ACCUM2      ARG=0?
02E0 26 06              BNE RAND2       NO
                *
02E2 CE 1357    RESTRT  LDX #$1357      YES
02E5 FF 0311 R          STX SEED
                *
02E8 FE 0311 R  RAND2   LDX SEED
02EB 27 F5              BEQ RESTRT      SEED MUST BE NONZERO
                *
02ED B6 0311 R          LDA A SEED
02F0 84 60              AND A #$60      MASK FEEDBACK BITS
02F2 27 08              BEQ RAND3       NO FEEDBACK
                *
02F4 81 40              CMP A #$40      ONE BIT?
02F6 27 0C              BEQ RAND4       YES
                *
02F8 81 20              CMP A #$20      ONE BIT?
02FA 27 08              BEQ RAND4       YES
                *
02FC 78 0312 R  RAND3   ASL SEED+1      NO FEEDBACK
02FF 79 0311 R          ROL SEED
0302 20 09              BRA RAND5       FINISH UP
                *
0304 78 0312 R  RAND4   ASL SEED+1      FEEDBACK
0307 79 0311 R          ROL SEED
030A 7C 0312 R          INC SEED+1      SET BIT
                *
030D FE 0311 R  RAND5   LDX SEED
0310 39                 RTS
                *
0311 0002       SEED    RMB 2
                *
                *
                * PUSH AND POP ROUTINES FOR INTEGER,FLOAT,AND STRING
                * ADDRESS OF VALUE IN INDEX REGISTER
                *
0313 32         PUSHI   PUL A           INTEGER PUSH
0314 B7 0050 C          STA A ACCUM1    REMOVE AND SAVE RETURN
0317 32                 PUL A
0318 B7 0051 C          STA A ACCUM1+1
031B A6 00              LDA A 0,X
031D 36                 PSH A
031E A6 01              LDA A 1,X
0320 36                 PSH A
0321 FE 0050 C  RETRN   LDX ACCUM1
0324 6E 00              JMP 0,X         RETURN
                *
0326 32         PUSHF   PUL A           PUSH FLOATING-POINT
0327 B7 0050 C          STA A ACCUM1    REMOVE AND SAVE RETURN
032A 32                 PUL A
032B B7 0051 C          STA A ACCUM1+1
032E C6 09              LDA B #9        NINE BYTES IN FLOAT
0330 A6 00      PUSHF2  LDA A 0,X
0332 36                 PSH A
0333 08                 INX
0334 5A                 DEC B
0335 26 F9              BNE PUSHF2      DONE?
                *
0337 20 E8              BRA RETRN       RETURN
                *
0339 32         PUSHS   PUL A           PUSH STRING
033A B7 0050 C          STA A ACCUM1    REMOVE AND SAVE RETURN
033D 32                 PUL A
033E B7 0051 C          STA A ACCUM1+1
0341 FF 0052 C          STX ACCUM2      SAVE POINTER
0344 A6 00      PUSHS2  LDA A 0,X
0346 81 04              CMP A #$04      END-STRING?
0348 27 03              BEQ PUSHS3      YES
                *
034A 08                 INX
034B 20 F7              BRA PUSHS2      NO
                *
034D A6 00      PUSHS3  LDA A 0,X
034F 36                 PSH A           PUSH DATA IN REVERSE ORDER
0350 BC 0052 C          CPX ACCUM2      DONE?
0353 27 CC              BEQ RETRN       YES, RETURN
                *
0355 09                 DEX
0356 20 F5              BRA PUSHS3      NO, CONTINUE
                *
0358 32         POPI    PUL A           POP INTEGER
0359 B7 0050 C          STA A ACCUM1    REMOVE AND SAVE RETURN
035C 32                 PUL A
035D B7 0051 C          STA A ACCUM1+1
0360 32                 PUL A
0361 A7 01              STA A 1,X
0363 32                 PUL A
0364 A7 00              STA A 0,X
0366 20 B9              BRA RETRN       RETURN
                *
0368 32         POPF    PUL A           POP FLOATING-POINT
0369 B7 0050 C          STA A ACCUM1    REMOVE AND SAVE RETURN
036C 32                 PUL A
036D B7 0051 C          STA A ACCUM1+1
0370 C6 09              LDA B #9        9 BYTES FOR FLOAT
0372 32         POPF2   PUL A
0373 A7 06              STA A 6,X       STORE IN REVERSE ORDER
0375 09                 DEX
0376 5A                 DEC B
0377 26 F9              BNE POPF2       DONE?
                *
```

```
0379 20 A6                BRA RETRN       YES, RETURN
                *
037B 32         POPS      PUL A           POP STRING
037C B7 0050 C            STA A ACCUM1    REMOVE AND SAVE RETURN
037F 32                   PUL A
0380 B7 0051 C            STA A ACCUM1+1
0383 32         POPS2     PUL A
0384 A7 00                STA A 0,X
0386 08                   INX
0387 81 04                CMP A #$04      END-STRING?
0389 26 F8                BNE POPS2       NO
                *
038B 20 94                BRA RETRN       YES, RETURN
                *
                *
                * I/O ROUTINES
                *
038D 44         OUTHL     LSR A           LEFT NIBBLE TO ASCII
038E 44                   LSR A
038F 44                   LSR A
0390 44                   LSR A
                *
0391 84 0F      OUTHR     AND A #$F       RIGHT NIBBLE TO ASCII
0393 8B 30                ADD A #$30
0395 81 39                CMP A #$39
0397 23 2A                BLS OUTEEE
                *
0399 8B 07                ADD A #$7
039B 20 26                BRA OUTEEE
                *
039D 86 20      OUTS      LDA A #$20      OUTPUT SPACE
039F 20 22                BRA OUTEEE
                *
                *
03A1 B6 077F R  INEEE     LDA A FRDFLG    FILE READ?
03A4 81 FF                CMP A #$FF
03A6 26 18                BNE INE3        NO
                *
03A8 7F 0065 C  INE1      CLR .EOF        TURN OFF END-OF-FILE
03AB 7F 0066 C            CLR  EOF+1
                *
03AE BD 0125 R            JSR GETB        GET A BYTE FROM FILE
03B1 24 09                BCC INE2        END-FILE?
                *
03B3 73 0065 C            COM .EOF        YES, SET END-OF-FILE
03B6 73 0066 C            COM  EOF+1
03B9 86 0D                LDA A #$0D       FORCE END-OF-LINE
03BB 39                   RTS
                *
03BC 4D         INE2      TST A           NULL?
03BD 27 E9                BEQ INE1        SKIP NULLS
                *
03BF 39                   RTS
                *
03C0 7E 0000 R  INE3      JMP SYSIN       "INEEE"
                *
03C3 36         OUTEEE    PSH A
03C4 37                   PSH B
03C5 7D 03F3 R            TST FLAG
03C8 27 12                BEQ OUTE2       NOT STRING MODE
                *
03CA FF 03EF R            STX XTEMP       SAVE X-REGISTER
03CD FE 03F1 R            LDX XPOINT      GET POINTER
03D0 33                   PUL B
03D1 32                   PUL A
03D2 A7 00                STA A 0,X       STORE CHARACTER
03D4 08                   INX
03D5 FF 03F1 R            STX XPOINT      MOVE POINTER
03D8 FE 03EF R            LDX XTEMP       RESTORE X-REGISTER
03DB 39                   RTS
                *
03DC F6 077F R  OUTE2     LDA B FWRFLG    FILE WRITE?
03DF C1 FF                CMP B #$FF
03E1 26 06                BNE OUTE3       NO
                *
03E3 BD 0128 R            JSR OUTB        WRITE A BYTE TO FILE
03E6 33                   PUL B
03E7 32                   PUL A
03E8 39                   RTS
                *
03E9 BD 0003 R  OUTE3     JSR SYSOUT      "OUTEEE"
03EC 33                   PUL B
03ED 32                   PUL A
03EE 39         OUTEND    RTS
                *
03EF 0002       XTEMP     RMB 2
03F1 0002       XPOINT    RMB 2
03F3 00         FLAG      FCB 0
                *
                * OUTPUT A STRING OF CHARS POINTED TO BY [X]
                * ENTRY POINT IS PDATA
                *
03F4 8D CD      PDATA2    BSR OUTEEE
03F6 08                   INX
03F7 A6 00      PDATA     LDA A 0,X
03F9 81 04                CMP A #4
03FB 26 F7                BNE PDATA2
                *
03FD 7D 03F3 R            TST FLAG
0400 26 EC                BNE OUTEND
                *
0402 7D 077F R            TST FWRFLG
0405 26 E7                BNE OUTEND
                *
0407 20 BA                BRA OUTEEE
                *
0409 7F 077F R  PDATA1    CLR FWRFLG      ENTRY POINTS FOR FILE I/O
040C 7F 03F3 R            CLR FLAG
040F 20 E6                BRA PDATA
                *
0411 7F 03F3 R  CRLF      CLR FLAG
0414 20 03                BRA LFEED+3
                *
                * OUTPUT A CRLF
                *
0416 7F 077F R  LFEED     CLR FWRFLG
0419 CE 041E R            LDX #CRL
041C 20 D9                BRA PDATA
                *
041E 0D0A       CRL       FDB $0D0A
0420 04                   FCB $4
                *
```

```
                        *
                        * MOVE-FLOAT UTILITY
                        * "TO" ADDRESS IN ACCUM1
                        * "FROM" ADDRESS IN X-REGISTER
                        *
0421 FF 0063 C ZMOVER STX ADRFM
0424 FE 0050 C        LDX ACCUM1
0427 FF 0061 C        STX ADRTO
042A C6 09            LDA B #9
042C 7E 0101 R        JMP MOVER
                        *
                        *
                        * HANDLE 'STOP' COMMAND
                        * RETURN TO SYSTEM MONITOR
                        *
042F 3F               STOP  SWI            CALL CP/68
0430 1F                     FCB 31         "WARMSTART"
0431 39                     RTS
                        *
                        *
                        * INPUT HANDLER
                        *
                        * PROMPT CHARACTER  ">"      $3E
                        * RETURN TO MONITOR "CTL C"  $03
                        *
0432 20               PRMPT FCC ' >'
0434 04                     FCB 4
0435 03               RETM  FCB $03
                        *
0436 0052             BUFFER RMB 82        INPUT BUFFER
                        *
                        *
                        * INPUT EDITOR ROUTINES
                        *
0488 7F 03F3 R INEDIT CLR FLAG
048B B6 077E R        LDA A FRDFLG FILE READ?
048E 81 FF            CMP A #$FF
0490 27 06            BEQ INP1     YES
                        *
0492 CE 0432 R        LDX #PRMPT   PRINT PROMPT
0495 BD 03F7 R        JSR PDATA
0498 CE 0436 R INP1   LDX #BUFFER
049B 7F 0274 R        CLR COUNT
                        *
049E BD 03A1 R INP2   JSR INEEE    INPUT A CHAR
04A1 81 0D            CMP A #$0D   END OF INPUT?
04A3 27 1A            BEQ INP6     YES,DONE
                        *
04A5 B1 0435 R INP3   CMP A RETM   RETURN TO MONITOR?
04A8 26 03            BNE INP4     NO
                        *
04AA 7E 042F R        JMP STOP     YES
                        *
04AD A7 00     INP4   STA A 0,X    STORE CHAR
04AF 08               INX
04B0 8C 0487 R        CPX #BUFFER+81
04B3 26 05            BNE INP5     OK
                        *
04B5 BD 0690 R        JSR BADERR   TOO MANY CHARS. ERROR
04B8 20 06            BRA RETRY
                        *
04BA 7C 0274 R INP5   INC COUNT
04BD 20 DF            BRA INP2     GET NEW CHAR
                        *
04BF 39        INP6   RTS
                        *
04C0 BD 0416 R RETRY  JSR LFEED
04C3 20 C3            BRA INEDIT
                        *
                        * END INPUT EDITING - NOW PROCESS BUFFER
                        *
                        *
04C5 7F 077E R INPUT  CLR FRDFLG   NOT FILE READ
04C8 BD BE            BSR INEDIT   GET INPUT LINE
04CA 7F 0616 R        CLR RTYFLG   RETRY AFTER ERROR O.K.
04CD CE 0436 R        LDX #BUFFER
04D0 A6 00     CNVRT2 LDA A 0,X    CHECK HEX/DECIMAL
04D2 81 20            CMP A #$20   SKIP BLANKS
04D4 26 17            BNE CNVRT3   NON-BLANK
                        *
04D6 08               INX
04D7 7A 0274 R        DEC COUNT
04DA 2E F4            BGT CNVRT2   KEEP SCANNING
                        *
04DC BD 0690 R        JSR BADERR   NO INPUT LEFT
04DF 7D 0616 R        TST RTYFLG   ERROR RETRY O.K.?
04E2 27 04            BEQ *+6      YES
                        *
04E4 CE FFFF          LDX #$FFFF   NO, RETURN -1 AND QUIT
04E7 39               RTS
                        *
04E8 8D D6            BSR RETRY    TRY AGAIN
04EA 7E 04CA R        JMP INPUT+5
                        *
04ED 81 24     CNVRT3 CMP A #'$    "$" FOR HEX
04EF 27 03            BEQ *+5      HEX
04F1 7E 056D R        JMP DECIM    DECIMAL
                        *
                        * HEX TO BINARY CONVERSION
                        *
04F4 7F 0050 C        CLR ACCUM1
04F7 7F 0051 C        CLR ACCUM1+1
04FA F6 0274 R        LDA B COUNT
                        *
04FD 08        CVHB1  INX          SHIFT TO RIGHTMOST CHAR
04FE 5A               DEC B
04FF 26 FC            BNE CVHB1
                        *
0501 09               DEX
0502 7A 0274 R        DEC COUNT
0505 27 B9            BEQ RETRY    NO ENTRY
                        *
0507 8D 3E            BSR CVHBS    ASCII TO BINARY
0509 B7 0051 C        STA A ACCUM1+1
050C 7A 0274 R        DEC COUNT
050F 27 32            BEQ CVHBD    (1 DIGIT)
                        *
0511 09               DEX
0512 8D 33            BSR CVHBS    ASCII TO BINARY
0514 48               ASL A        SHIFT TO LEFT NIBBLE
0515 48               ASL A
0516 48               ASL A
```

```
0517 48                ASL A
0518 BA 0051 C         ORA A ACCUM1+1
051B B7 0051 C         STA A ACCUM1+1
051E 7A 0274 R         DEC COUNT
0521 27 20             BEQ CVHBD     (2 DIGITS)
                *
0523 09                DEX
0524 8D 21             BSR CVHBS     ASCII TO BINARY
0526 B7 0050 C         STA A ACCUM1
0529 7A 0274 R         DEC COUNT
052C 27 15             BEQ CVHBD     (3 DIGITS)
                *
052E 09                DEX
052F 8D 16             BSR CVHBS     ASCII TO BINARY
0531 48                ASL A         SHIFT TO LEFT NIBBLE
0532 48                ASL A
0533 48                ASL A
0534 48                ASL A
0535 BA 0050 C         ORA A ACCUM1
0538 B7 0050 C         STA A ACCUM1
053B 7A 0274 R         DEC COUNT
053E 27 03             BEQ CVHBD     (4 DIGITS)
                *
0540 BD 06C4 R         JSR TRUNER    TRUNCATION ERROR
                *
0543 FE 0050 C CVHBD   LDX ACCUM1    RETURN VALUE
0546 39                RTS
                *
                * ASCII TO BINARY CONVERSION
                *
0547 A6 00     CVHBS   LDA A 0,X     GET BYTE
0549 80 30             SUB A #$30    CONVERT
054B 2B 0F             BMI ERROR     BAD CHARACTER
                *
054D 81 09             CMP A #9      DIGIT?
054F 2F 0A             BLE CVHBSD    YES
                *
0551 80 07             SUB A #7      10-15?
0553 81 0A             CMP A #10
0555 2D 05             BLT ERROR     BAD CHAR
                *
0557 81 0F             CMP A #15
0559 2F 01             BGT ERROR     BAD CHAR
                *
055B 39        CVHBSD  RTS
                *
055C BD 0690 R ERROR   JSR BADERR    ERROR COMMENT
055F 31                INS           SKIP RETURN ON THE STACK
0560 31                INS
0561 7D 0616 R         TST RTYFLG    ERROR RETRY O K ?
0564 27 04             BEQ *+6       YES
                *
0566 CE FFFF           LDX #$FFFF    NO, RETURN -1 AND QUIT
0569 39                RTS
                *
056A 7E 04C5 R         JMP INPUT
                *
                *
                *
                * DECIMAL TO BINARY CONVERSION - 16 BITS, SIGNED
                *
056D 7F 0615 R DECIM   CLR SIG2
0570 81 2D             CMP A #'-     CHECK FOR MINUS SIGN
0572 26 0F             BNE DECIM2    NO
                *
0574 73 0615 R         COM SIG2      YES, SET SIGN
0577 08                INX
0578 7A 0274 R         DEC COUNT
057B 26 06             BNE DECIM2    OK
                *
057D BD 04C0 R         JSR RETRY     NO ENTRY
0580 7E 04CA R         JMP INPUT+5
                *
0583 7F 0611 R DECIM2  CLR DVAL      INITIALIZE
0586 7F 0612 R         CLR DVAL+1
0589 7F 0613 R         CLR TENVAL
058C 7F 0614 R         CLR TENVAL+1
058F 7C 0614 R         INC TENVAL+1
0592 F6 0274 R         LDA B COUNT
                *
0595 08        DECIM3  INX           SHIFT TO RIGHTMOST CHAR
0596 5A                DEC B
0597 26 FC             BNE DECIM3
                *
0599 09                DEX
059A FF 060F R         STX DXSAV     SAVE POINTER
                *
059D E6 00     CVDB1   LDA B 0,X     GET DIGIT
059F C1 20             CMP B #$20    SWALLOW BLANKS
05A1 26 0E             BNE CBLNK     NOT A BLANK
                *
05A3 FE 060F R         LDX DXSAV
05A6 09                DEX
05A7 FF 060F R         STX DXSAV     MOVE OVER A BLANK
05AA 7A 0274 R         DEC COUNT
05AD 27 06             BEQ ERROR2    NULL DATA, ERROR
                *
05AF 20 EC             BRA CVDB1     GET NEXT DIGIT
                *
05B1 C0 30     CBLNK   SUB B #$30    ASCII TO BINARY
05B3 2A 0F             BPL CVDB2     OK
                *
05B5 BD 0690 R ERROR2  JSR BADERR    ERROR COMMENT
05B8 7D 0616 R         TST RTYFLG    RETRY AFTER ERROR?
05BB 27 04             BEQ *+6       YES
                *
05BD CE FFFF           LDX #$FFFF    NO, RETURN -1 AND QUIT
05C0 39                RTS
                *
05C1 7E 04C5 R         JMP INPUT
                *
05C4 C1 09     CVDB2   CMP B #9      OK?
05C6 2E ED             BGT ERROR2    NO
                *
05C8 F7 0051 C         STA B ACCUM1+1
05CB 7F 0050 C         CLR ACCUM1
05CE FE 0613 R         LDX TENVAL
05D1 BD 0195 R         JSR MULT      X,=TENVAL*DIGIT
05D4 FF 0050 C         STX ACCUM1
05D7 FE 0611 R         LDX DVAL
05DA BD 0161 R         JSR ADD2      X =DVAL+TENVAL*DIGIT
05DD FF 0611 R         STX DVAL
```

```
05E0 FE 060F R        LDX DXSAV     RESTORE POINTER
05E3 09               DEX
05E4 FF 060F R        STX DXSAV
05E7 7A 0274 R        DEC COUNT     DONE?
05EA 27 14            BEQ CVDB3     YES
                *
05EC FE 0613 R        LDX TENVAL
05EF FF 0050 C        STX ACCUM1
05F2 FE 0083 R        LDX TEN10
05F5 BD 0195 R        JSR MULT      X=TENVAL*10
05F8 FF 0613 R        STX TENVAL
05FB FE 060F R        LDX DXSAV     RESTORE POINTER
05FE 20 9D            BRA CVDB1     NO
                *
0600 7D 0615 R CVDB3  TST SIG2      NUMBER MINUS?
0603 27 06            BEQ DECIM4    NO
                *
0605 CE 0611 R        LDX #DVAL
0608 BD 01F6 R        JSR NEGATE    CHANGE SIGN
                *
060B FE 0611 R DECIM4 LDX DVAL      RETURN VALUE
060E 39               RTS
                *
060F 0002      DXSAV  RMB 2         STORE POINTER
0611 0002      DVAL   RMB 2         STORE BINARY
0613 0002      TENVAL RMB 2         POWER OF TEN
0615 0001      SIG2   RMB 1         SIGN FLAG
0616 0001      RTYFLG RMB 1         ERROR RETRY FLAG
                *
                *
                * OUTPUT FUNCTION - SIGNED DECIMAL INTEGER
                * CONVERT TO FLOATING-POINT AND USE FPRINT
                *
0617 7F 077F R OUTPUT CLR FWRFLG     NOT FILE WRITE
061A FF 0050 C        STX ACCUM1
061D 7F 03F3 R        CLR FLAG
0620 CE 062C R OUTSS  LDX #FTEMP    POINT TO TEMP BUFFER
0623 BD 0CDD R        JSR FLOAT     CONVERT TO FLOATING-POINT
0626 CE 062C R        LDX #FTEMP
0629 7E 09F3 R        JMP FPRINT+6  NOW PRINT IT
                *
062C 0009      FTEMP  RMB 9
0635 0002      XSAV   RMB 2
                *
                *
                * ERROR REPORTING ROUTINES
                *
                * ERROR PRINT ROUTINE - GIVES CALL ADDRESS
                * ADDRESS OF ERROR TEXT IN BUFFER
                *
0637 7F 03F3 R PERR   CLR FLAG
063A 7F 077F R        CLR FWRFLG    NOT FILE WRITE
063D BD 0416 R        JSR LFEED
0640 FF 043E R        LDX BUFFER+8  OUTPUT ERROR MESSAGE
0643 BD 03F7 R        JSR PDATA
0646 30               TSX           GET CALL ADDRESS
0647 08               INX
0648 08               INX
0649 C6 04            LDA B #4      4 HEX CHARACTERS
064B BD 0905 R        JSR MPRINT    OUTPUT ADDRESS
064E 7E 0416 R        JMP LFEED     CRLF
                *
                * DIVIDE BY ZERO - RETURNS TO MONITOR
                *
0651 CE 065C R DIVERR LDX #DIVMSG
0654 FF 043E R        STX BUFFER+8
0657 8D DE            BSR PERR
0659 7E 042F R        JMP STOP      RETURN TO MONITOR
                *
065C 20        DIVMSG FCC ' DIVIDE BY ZERO AT '
065F 04               FCB 4
                *
                * NEGATIVE EXPONENT ERROR - DO NOTHING
                *
0670 CE 0679 R NEGERR LDX #NEGMSG
0673 FF 043E R        STX BUFFER+8
0676 8D BF            BSR PERR
0678 39               RTS
                *
0679 20        NEGMSG FCC ' NEGATIVE EXPONENT AT '
068F 04               FCB 4
                *
                * BAD INPUT ERROR - RETRYS INPUT
                *
0690 CE 0699 R BADERR LDX #BADMSG
0693 FF 043E R        STX BUFFER+8
0696 8D 9F            BSR PERR
0698 39               RTS
                *
0699 20        BADMSG FCC ' BAD INPUT AT '
06A7 04               FCB 4
                *
                * OVERFLOW ERROR
                * FIXUP  MAKE LARGEST POSITIVE VALUE
                *
06A8 CE 06B6 R OVERR  LDX #OVMSG
06AB FF 043E R        STX BUFFER+8
06AE BD 0637 R        JSR PERR
06B1 86 7F            LDA A #$7F    FIXUP
06B3 C6 FF            LDA B #$FF
06B5 39               RTS
                *
06B6 20        OVMSG  FCC ' OVERFLOW AT '
06C3 04               FCB 4
                *
                * TRUNCATION ERROR ON INPUT
                *
06C4 7F 03F3 R TRUNER CLR FLAG
06C7 7F 077F R        CLR FWRFLG
06CA BD 0416 R        JSR LFEED
06CD CE 06D6 R        LDX #TERR
06D0 BD 03F7 R        JSR PDATA
06D3 7E 0416 R        JMP LFEED
                *
06D6 20        TERR   FCC ' TRUNCATED'
06E0 04               FCB 4
                *
                * STRING LENGTH ERROR
                * FIXUP  MAKE LENGTH=ZERO
                *
06E1 CE 06F1 R LENERR LDX #LENMSG    PRINT ERROR MESSAGE
06E4 FF 043E R        STX BUFFER+8
```

```
06E7 BD 0637 R          JSR PERR
06EA 7F 0052 C          CLR ACCUM2
06ED 7F 0053 C          CLR ACCUM2+1     RETURN LENGTH=0
06F0 39                 RTS
                 *
06F1 20          LENMSG FCC ' STRING TOO LONG AT '
0705 04                 FCB 4
                 *
                 * ILLEGAL ARGUMENT TO SCIENTIFIC FUNCTION
                 *
0706 CE 0712 R   ILLERR LDX #ILLMSG
0709 FF 043E R          STX BUFFER+8
070C BD 0637 R          JSR PERR
070F 7E 042F R          JMP STOP         TRAP TO MONITOR
                 *
0712 20          ILLMSG FCC ' ILLEGAL ARGUMENT AT '
0727 04                 FCB 4
                 *
                 *
                 * FILE I/O HANDLER ROUTINES
                 *
0728 86 FF       IREAD  LDA A #$FF
072A B7 077E R          STA A FRDFLG     SET FILE-READ FLAG
072D BD 04C8 R          JSR INPUT+3      INTEGER READ FROM FILE
0730 7F 077E R          CLR FRDFLG
0733 39                 RTS
                 *
0734 86 FF       FLREAD LDA A #$FF
0736 B7 077E R          STA A FRDFLG     SET FILE-READ FLAG
0739 BD 0805 R          JSR FREAD+3      FLOATING-POINT READ FROM FILE
073C 7F 077E R          CLR FRDFLG
073F 39                 RTS
                 *
0740 86 FF       SREAD  LDA A #$FF
0742 B7 077F R          STA A FRDFLG     SET FILE-READ FLAG
0745 BD 03A1 R   SRD1   JSR INEEE        STRING READ FROM FILE
0748 81 0D              CMP A #$0D       END-STRING (CR)?
074A 27 09              BEQ SRD2         YES
                 *
074C 81 0A              CMP A #$0A       LINE FEED?
074E 27 F5              BEQ SRD1         SWALLOW LF
                 *
0750 A7 00              STA A 0,X        STORE IT
0752 08                 INX              MOVE POINTER
0753 20 F0              BRA SRD1
                 *
0755 86 04       SRD2   LDA A #$04       APPEND END-STRING
0757 A7 00              STA A 0,X
0759 7F 077E R          CLR FRDFLG
075C 39                 RTS
                 *
                 * SET CURRENT IOB ADDRESS
                 *
075D FF 0069 C   CURIOB STX .CURIO
0760 7F 0067 C          CLR .ERR         RESET ERROR FLAG
0763 7F 0068 C          CLR .ERR+1
0766 39                 RTS
                 *
0767 CE 0000     SRFCRD LDX #0            RESET END-FILE FLAG
076A FF 0065 C          STX  EOF
076D BD 0125 R   SREC1  JSR GETB          GET A BYTE FROM FILE
0770 24 07              BCC SREC2        END-FILE?
                 *
0772 CE FFFF            LDX #$FFFF
0775 FF 0065 C          STX .EOF         SET END-FILE INDICATOR
0778 39                 RTS
                 *
0779 81 0A       SREC2  CMP A #$0A       END-OF-RECORD?
077B 26 F0              BNE SREC1        SKIP TO END OF RECORD
                 *
077D 39                 RTS
                 *
077E 0001        FRDFLG RMB 1            FILE-READ FLAG
077F 0001        FWRFLG RMB 1            FILE-WRITE FLAG
                 *
0780 86 FF       IWRITE LDA A #$FF
0782 B7 077F R          STA A FWRFLG     SET FILE-WRITE FLAG
0785 BD 061A R          JSR OUTPUT+3     INTEGER WRITE TO FILE
0788 86 0D       IWRIT2 LDA A #$0D       APPEND CRET.
078A BD 0128 R          JSR OUTB
078D 7F 077F R          CLR FWRFLG
0790 39                 RTS
                 *
0791 86 FF       FWRITE LDA A #$FF
0793 B7 077F R          STA A FWRFLG     SET FILE-WRITE FLAG
0796 BD 09F0 R          JSR FPRINT+3     FLOATING WRITE TO FILE
0799 20 ED              BRA IWRIT2       FINISH UP
                 *
079B 86 FF       SWRITE LDA A #$FF       (FORMATTED)
079D B7 077F R          STA A FWRFLG     SET FILE-WRITE FLAG
07A0 BD 089E R          JSR STROUT+3     STRING WRITE TO FILE
07A3 20 E3              BRA IWRIT2       FINISH UP
                 *
07A5 86 FF       QWRITE LDA A #$FF       (UNFORMATTED)
07A7 B7 077F R          STA A FWRFLG     SET FILE-WRITE FLAG
07AA BD 0695 R          JSR QOUT+3       STRING WRITE TO FILE
07AD 20 D9              BRA IWRIT2       FINISH UP
                 *
07AF 86 FF       FLFEED LDA A #$FF
07B1 B7 077F R          STA A FWRFLG     SET FILE-WRITE FLAG
07B4 BD 0419 R          JSR LFEED+3      WRITE CR,LF TO FILE
07B7 7F 077F R          CLR FWRFLG
07BA 39                 RTS
                 *
07BB 86 FF       FSPACE LDA A #$FF
07BD B7 077F R          STA A FWRFLG     SET FILE-WRITE FLAG
07C0 BD 08CF R          JSR SPACES+3     WRITE SPACES TO FILE
07C3 7F 077F R          CLR FWRFLG
07C6 39                 RTS
                 *
                 *
                 * STRING ROUTINES
                 * USE 04 CODE FOR END-STRING
                 * ASSUMES ASCII CODING
                 *
                 * "LENGTH" FUNCTION
                 *
07C7 7F 0052 C   LEN    CLR ACCUM2       LENGTH=0
07CA 7F 0053 C          CLR ACCUM2+1
07CD A6 00       LEN2   LDA A 0,X        GET STRING CHAR
07CF 81 04              CMP A #$04       END-STRING?
07D1 27 18              BEQ LEN3         YES
```

```
                    *
07D3 FF 0635 R          STX XSAV        SAVE X-REGISTER
07D6 FE 0052 C          LDX ACCUM2
07D9 08                 INX             COUNT CHARACTER
07DA FF 0052 C          STX ACCUM2
07DD B6 0052 C          LDA A ACCUM2
07E0 2B 06              BMI LENR        STRING TOO LONG
                    *
07E2 FE 0635 R          LDX XSAV        RESTORE X-REGISTER
07E5 08                 INX
07E6 20 E5              BRA LEN2        KEEP COUNTING
                    *
07E8 BD 06E1 R LENR     JSR LENERR      ERROR MESSAGE
                    * RETURN LENGTH OF ZERO
                    *
07EB FE 0052 C LEN3     LDX ACCUM2      RETURN LENGTH
07EE 39                 RTS
                    *
                    * STRING "VALUE" FUNCTION
                    * CONVERT STRING TO NUMBER
                    * ADDRESS OF STRING IN X REGISTER
                    *
07EF FF 0050 C VAL      STX ACCUM1      SAVE X REGISTER
07F2 8D D3              BSR LEN         GET STRING LENGTH
07F4 FF 0635 R          STX XSAV
07F7 B6 0636 R          LDA A XSAV+1
07FA B7 0274 R          STA A COUNT
07FD 26 01              BNE VAL2        STRING LENGTH > 0
07FF 39                 RTS             IF NOT, RETURN 0
                    *
0800 FE 0050 C VAL2     LDX ACCUM1      SET UP FOR CONVERSION
0803 86 FF              LDA A #$FF
0805 B7 0616 R          STA A RTYFLG    NO ERROR RETRY
0808 7E 04D0 R          JMP CNVRT2      CONVERT
                    *
                    *
                    * STRING "ASCII" FUNCTION
                    * CONVERTS NUMBER TO CHARACTER STRING
                    * ADDRESS OF STRING IN X REGISTER
                    * NUMBER IN "ACCUM3"
                    *
080B 7F 03F3 R ASCII    CLR FLAG
080E 73 03F3 R          COM FLAG        SET "STRING" FLAG
0811 FF 03F1 R          STX XPOINT      SAVE ADDRESS
0814 FE 0054 C          LDX ACCUM3      MOVE NUMBER
0817 FF 0050 C          STX ACCUM1
081A BD 0620 R          JSR OUTSS       CONVERT
081D 86 04      ASCII2  LDA A #$04      END-STRING
081F BD 03C3 R          JSR OUTEFE
0822 7F 03F3 R          CLR FLAG
0825 FE 03F1 R          LDX XPOINT      POINT TO LAST CHARACTER
0828 09                 DEX             BACK PAST END-STRING
0829 39                 RTS
                    *
                    *
                    * FLOATING-POINT TO STRING CONVERSION
                    * ADDRESS OF NUMBER IN "ACCUM3"
                    * ADDRESS OF STRING IN X-REGISTER
                    *
082A 7F 03F3 R FASCII   CLR FLAG
082D 73 03F3 R          COM FLAG
```

```
0830 FF 03F1 R          STX XPOINT
0833 FE 0054 C          LDX ACCUM3
0836 BD 09F3 R          JSR FPRINT16    CONVERT
0839 20 E2              BRA ASCII2      FINISH UP
                    *
                    *
                    * STRING "MID" FUNCTION
                    * RETURNS SUBSTRING FROM GIVEN STRING
                    * ADDRESS OF SOURCE STRING IN INDEX REGISTER
                    * ADDRESS OF DESTINATION IN 'ACCUM4'
                    * CHARACTER COUNT IN 'ACCUM3'
                    *
083B FF 0063 C MID      STX ADRFM       SET SOURCE ADDRESS
083E FE 0056 C          LDX ACCUM4
0841 FF 0061 C          STX ADRTO       SET DEST  ADDRESS
0844 FE 0054 C          LDX ACCUM3      GET CHAR  COUNT
0847 2A 09              BPL MID2        IF <32768, O.K.
                    *
0849 BD 06E1 R          JSR LENERR      IF NOT, LENGTH ERROR
084C CE 0001            LDX #1          DEFAULT CHAR  COUNT=1
084F FF 0054 C          STX ACCUM3
                    *
0852 FE 0063 C MID2     LDX ADRFM
0855 A6 00              LDA A 0,X       GET CHAR. FROM SOURCE
0857 08                 INX
0858 FF 0063 C          STX ADRFM       MOVE POINTER
085B FE 0061 C          LDX ADRTO
085F 81 04              CMP A #$04      END-STRING IN SOURCE?
0860 27 11              BEQ MID3        YES
                    *
0862 A7 00              STA A 0,X       NO, STORE CHAR  IN DEST
0864 08                 INX
0865 FF 0061 C          STX ADRTO       MOVE POINTER
0868 FE 0054 C          LDX ACCUM3
086B 09                 DEX             COUNT DOWN CHARACTER
086C 27 05              BEQ MID3        DONE?
                    *
086E FF 0054 C          STX ACCUM3      NOT DONE
0871 20 DF              BRA MID2
                    *
0873 FE 0061 C MID3     LDX ADRTO       APPEND END-STRING
0876 86 04              LDA A #$04
0878 A7 00              STA A 0,X
087A 39                 RTS
                    *
                    *
                    * TRANSFER STRING CHARACTERS
                    * FROM (X) TO (ACCUM1)
                    *
087B FF 0052 C XFER     STX ACCUM2      SAVE X
087E FE 0050 C          LDX ACCUM1
0881 FF 03F1 R          STX XPOINT      SET "TO" ADDRESS
0884 FE 0052 C          LDX ACCUM2      SET "FROM" ADDRESS
0887 7F 03F3 R          CLR FLAG
088A 73 03F3 R          COM FLAG        SET STRING-MODE FLAG
088D BD 03F7 R          JSR PDATA       MOVE CHARACTERS
0890 20 8B              BRA ASCII2      APPEND END-STRING
                    *
                    *
                    * STRING OUTPUT ROUTINES
                    * INDEX REGISTER POINTS TO CHARACTER STRING
```

```
                    * 'STROUT' IS FORMATTED
                    * 'QOUT' IS UNFORMATTED
                    *
0892 7F 077F R QOUT   CLR FWKFLG
0895 7F 03F3 R        CLR FLAG
0898 7E 03F7 R        JMP PDATA
                    *
089B 7F 077F R STROUT CLR FWKFLG
089E 7F 03F3 R        CLR FLAG
08A1 F6 09D4 R        LDA B SIZE     GET FIELD FORMAT WIDTH
08A4 A6 00      STRUT LDA A 0,X      GET A CHARACTER
08A6 81 04            CMP A #$04     END-STRING?
08A8 27 08            BEQ STRUT2     YES
                    *
08AA BD 03C3 R        JSR OUTEEE     NO, OUTPUT CHARACTER
08AD 08               INX
08AE 5A               DEC B          MOVE POINTERS
08AF 26 F3            BNE STRUT      LOOP UNTIL DONE
                    *
08B1 39               RTS
                    *
08B2 BD 039D R STRUT2 JSR OUTS       OUTPUT PADDING SPACES
08B5 5A               DEC B          DONE?
08B6 26 FA            BNE STRUT2     LOOP UNTIL DONE
                    *
08B8 39               RTS
                    *
                    *
                    * STRING INPUT ROUTINE
                    * ADDRESS OF STRING IN X REGISTER
                    * WILL INPUT UP TO 80 CHARACTERS
                    *
08B9 7F 077E R STRIN  CLR FRDFLG
08BC FF 0050 C        STX ACCUM1
08BF BD 0488 R        JSR INEDIT     GET STRING
08C2 86 04            LDA A #$04
08C4 A7 00            STA A 0,X      APPEND END-STRING
08C6 CE 0436 R        LDX #BUFFER
08C9 7E 087B R        JMP XFER       MOVE STRING
                    *
                    * "SPACES" FUNCTION (OUTPUT BLANKS)
                    * ARGUMENT < 0 , NO SPACES
                    *
08CC 7F 077F R SPACES CLR FWKFLG
08CF FF 0052 C        STX ACCUM2
08D2 2B 11            BMI SPCRTN     IF <0, NO SPACES
08D4 27 0F            BEQ SPCRTN     ARG  ZERO, DONE
                    *
08D6 7F 03F3 R SPC1   CLR FLAG
08D9 BD 039D R        JSR OUTS       PRINT BLANK
08DC FE 0052 C        LDX ACCUM2     COUNT DOWN
08DF 09               DEX
08E0 FF 0052 C        STX ACCUM2
08E3 26 F1            BNE SPC1
                    *
08E5 39         SPCRTN RTS
                    *
                    * TRANSFER PARAMETER-LIST ADDRESS FUNCTION
                    * FROM 'CALL' LIST TO 'PROCEDURE' LIST
                    * LAST ADDRESS MARKER=FFHF (HEX)
                    * CHECK FOR EQUAL LENGTH LISTS
                    *
08E6 30         PARAM  TSX
08E7 EF 02             LDX 2,X        GET 'CALL' ADDRESS
08E9 FF 0063 C         STX ADRFM      SAVE IT
08EC 30                TSX
08ED EF 00             LDX 0,X        GET 'PROCEDURE' ADDRESS
08EF FF 0061 C         STX ADRTO      SAVE IT
                    *
08F2 A6 02      PARAM2 LDA A 2,X      GET ENTRY @ADRTO
08F4 E6 03             LDA B 3,X
08F6 81 FF             CMP A #$FF     END-MARKER?
08F8 26 39             BNE PARAM3     NO
                    *
08FA C1 FF             CMP B #$FF     END-MARKER?
08FC 26 35             BNE PARAM3     NO
                    *
08FE FE 0063 C         LDX ADRFM      CHECK 'CALL' LIST
0901 A6 02             LDA A 2,X      GET ENTRY @ADRFM
0903 E6 03             LDA B 3,X
0905 81 FF             CMP A #$FF     END-MARKER?
0907 26 05             BNE PARERR     NO, ERROR
                    *
0909 C1 FF             CMP B #$FF     END-MARKER?
090B 26 01             BNE PARERR     NO, ERROR
                    *
090D 39                RTS            END OF BOTH LISTS, DONE!
                    *
090E CE 0917 R PARERR LDX #PARLN      LIST ERROR MESSAGE
0911 FF 043E R         STX BUFFER+8
0914 7E 0637 R         JMP PERR       GIVE UP!
                    *
0917 50         PARLN  FCC 'PARAMETER LIST MISMATCH AT
0932 04                FCB $04
                    *
0933 FF 0063 C PARAM3 LDX ADRFM
0936 A6 02             LDA A 2,X      GET PARAMETER @ADRFM
0938 E6 03             LDA B 3,X
093A 81 FF             CMP A #$FF     END-MARKER?
093C 26 04             BNE PARAM4     NO
                    *
093E C1 FF             CMP B #$FF     END-MARKER?
0940 27 CC             BEQ PARERR     IF SO, ERROR
                    *
0942 FE 0061 C PARAM4 LDX ADRTO
0945 A7 02             STA A 2,X      PUT PARAMETER @ADRTO
0947 E7 03             STA B 3,X
0949 08                INX
094A 08                INX
094B FF 0061 C         STX ADRTO      MOVE POINTERS
094E FE 0063 C         LDX ADRFM
0951 08                INX
0952 08                INX
0953 FF 0063 C         STX ADRFM
0956 20 9A             BRA PARAM2     CONTINUE
                    *
                    *
                    * DELAY FUNCTION
                    * ARGUMENT IN X-REGISTER
                    * DELAY FOR "ARG" MILLISECONDS
                    * CALIBRATE "DTIME" TO PROCESSOR
                    * DTIME=165 FOR 1 USEC PROCESSOR
```

```
                    *       = 88 FOR 2 USEC PROCESSOR
                    *
0958 F6 0962 R DELAY    LDA B DTIME
095B 5A                 DEC B
095C 26 FD              BNE *-1
                    *
095E 09                 DEX
095F 26 F7              BNE DELAY
                    *
0961 39                 RTS
                    *
0962 96             DTIME   FCB 150         TIMER ADJUSTMENT
                    *
                    *
                    * SUBROUTINE TO SET DIGIT PARAMETERS FOR PRINT
                    * SET SIZE OF INTEGER PART 'SIZE'
                    *       0 < SIZE < 128
                    * SET NUMBER OF DECIMAL PLACES 'DECPL'
                    *       -1 < DECPL < 13
                    * 'SIZE' PASSED IN ACCUM4
                    * 'DECPL' PASSED IN INDEX REGISTER
                    *
0963 FF 0050 C DIGSET   STX ACCUM1      SAVE 'DECPL'
0966 B6 0050 C          LDA A ACCUM1    CHECK FOR VALID RANGE
0969 26 1A              BNE NODIG1      BAD
                    *
096B B6 0051 C          LDA A ACCUM1+1
096E 81 0C              CMP A #12
0970 2E 13              BGT NODIG1      BAD
                    *
0972 B7 09D3 R DIGS2    STA A DECPL     GOOD VALUE
0975 B6 0056 C          LDA A ACCUM4    CHECK FOR VALID RANGE
0978 26 13              BNE NODIG2      BAD
                    *
097A B6 0057 C          LDA A ACCUM4+1
097D 27 0E              BEQ NODIG2      BAD
                    *
097F 2B 0C              BMI NODIG2      BAD
                    *
0981 B7 09D4 R DIGS3    STA A SIZE      GOOD VALUE
0984 39                 RTS
                    *
0985 CE 099D R NODIG1   LDX #DERR1      'DECPL' ERROR MESSAGE
0988 8D 0C              BSR DIGERR
098A 4F                 CLR A           DEFAULT 'DECPL'=0
098B 20 E5              BRA DIGS2
                    *
098D CE 09B6 R NODIG2   LDX #DERR2      'SIZE' ERROR MESSAGE
0990 8D 04              BSR DIGERR
0992 86 06              LDA A #6        DEFAULT 'SIZE =6
0994 20 EB              BRA DIGS3
                    *
0996 FF 043E R DIGERR   STX BUFFER+8
0999 BD 0637 R          JSR PERR        PRINT ERROR MESSAGE
099C 39                 RTS
                    *
099D 20             DERR1   FCC ' DECIMAL-PLACE ERROR AT '
09B5 04                 FCB 4
09B6 20             DERR2   FCC ' INTEGER-PART SIZE ERROR AT '
09D2 04                 FCB 4
                    *
09D3 00             DECPL   FCB 0           NUMBER OF DECIMAL PLACES
09D4 06             SIZE    FCB 6           SIZE OF INTEGER PART
                    *
                    * SUBROUTINE TO PRINT DIGIT STRINGS
                    * COUNT OF BCD DIGITS IN "B" REGISTER
                    * INDEX REGISTER POINTS TO DIGITS
                    * (ENTER AT 'MPRNT2' TO START WITH RIGHT DIGIT)
                    *
09D5 A6 02          MPRINT  LDA A 2,X       GET BYTE
09D7 BD 038D R          JSR OUTHL       PRINT LEFT DIGIT
09DA 73 0AEE R          COM NIBBLE      SWITCH SIDES
09DD 5A                 DEC B           COUNT DIGIT
09DE 27 0C              BEQ MQUIT       DONE?
                    *
09E0 A6 02          MPRNT2  LDA A 2,X       GET BYTE
09E2 BD 0391 R          JSR OUTHR       PRINT RIGHT DIGIT
09E5 73 0AEE R          COM NIBBLE      SWITCH SIDES
09E8 08                 INX             MOVE POINTER
09E9 5A                 DEC B           COUNT DIGIT
09EA 26 E9              BNE MPRINT      DONE?
                    *
09EC 39             MQUIT   RTS             DONE!
                    *
                    * FLOATING PRINT ROUTINE
                    * VALUE IS 9 BYTES (14 DIGIT BCD)
                    * POINTER IN X-REGISTER
                    * USE SCIENTIFIC NOTATION IF NECESSARY
                    * USE FLOATING-DECIMAL NOTATION OTHERWISE
                    * SIZE OF INTEGER PART SET AT "SIZE"
                    * NUMBER OF DECIMAL PLACES SET AT "DECPL"
                    *
09ED 7F 077F R FPRINT   CLR FWRFLG
09F0 7F 03F3 R          CLR FLAG
09F3 E6 00              LDA B 0,X       CHECK FOR ZERO
09F5 C5 20              BIT B #$20
09F7 27 13              BEQ NOTZ
                    *
09F9 BD 039D R          JSR OUTS
09FC CE 0A03 R          LDX #ZERO       PRINT ZERO (EXACT)
09FF C6 01              LDA B #1        ONE SIG  DIGIT
0A01 20 3D              BRA ZDOT2A
                    *
0A03 2001           ZERO    FDB $2001
0A05 0000               FDB $0000
0A07 0000               FDB $0000
0A09 0000               FDB $0000
0A0B 00                 FCB $00
                    *
0A0C C5 80          NOTZ    BIT B #$80      CHECK VALUE SIGN
0A0E 27 07              BEQ PLUS
                    *
0A10 86 2D              LDA A #'-
0A12 BD 03C3 R          JSR OUTLEE      PRINT MINUS
0A15 20 03              BRA ZDOT
                    *
0A17 BD 039D R PLUS     JSR OUTS        PRINT BLANK (PLUS)
                    *
0A1A 7F 0AEE R ZDOT     CLR NIBBLE      SET BYTE TOGGLE
0A1D E6 00              LDA B 0,X
0A1F C4 0F              AND B #$0F      CHECK FIRST EXP  DIGIT
```

```
0A21 26 14                BNE ZDOT2       TOO BIG?
                    *
0A23 E6 01                LDA B 1,X       CHECK EXP  DIGIT TWO,THREE
0A25 C1 14                CMP B #$14      TOO BIG?
0A27 22 0E                BHI ZDOT2
                    *
0A29 C1 0A                CMP B #10       GREATER THAN 10?
0A2B 2D 02                BLT ZDOT1       NO
                    *
0A2D C0 06                SUB B #6        CONVERT BCD TO BINARY
                    *
0A2F 5D             ZDOT1 TST B           EXP =0?
0A30 27 41                BEQ EXPM        YES
                    *
0A32 F1 09D4 R            CMP B SIZE      TOO BIG?
0A35 2F 03                BLE *+5
                    *
0A37 7E 0AA3 R      ZDOT2 JMP ZDOT3       MUST USE SCI  NOTATION
                    *
0A3A A6 00                LDA A 0,X
0A3C 85 40                BIT A #$40      CHECK EXP.  SIGN
0A3E 26 33                BNE EXPM        EXP<0
                    *
0A40 F7 0AEF R      ZDOT2A STA B SIGDIG   SAVE DIGIT COUNT
0A43 50                   NEG B
0A44 FB 09D4 R            ADD B SIZE      SIZE-SIGDIG
0A47 27 06                BEQ PLOOP       SIZE=SIGDIG?
                    *
0A49 BD 039D R      BLOOP JSR OUTS        ALIGN OUTPUT
0A4C 5A                   DEC B
0A4D 26 FA                BNE BLOOP
                    *
0A4F F6 0AEF R      PLOOP LDA B SIGDIG
0A52 BD 09D5 R            JSR MPRINT      PRINT INTEGER PART
0A55 F6 09D3 R            LDA B DECPL     INTEGER FORMAT?
0A58 27 10                BEQ PLOOP1      YES, THEN QUIT!
                    *
0A5A 86 2E                LDA A #'.
0A5C BD 03C3 R            JSR OUTEEE      PRINT DECIMAL POINT
0A5F F6 09D3 R            LDA B DECPL
0A62 7D 0AEE R            TST NIBBLE      WHICH SIDE?
0A65 26 06                BNE PLOOP2      RIGHT SIDE
                    *
0A67 BD 09D5 R            JSR MPRINT      PRINT FRACTION
0A6A 7E 039D R      PLOOP1 JMP OUTS       SPACE AND RETURN
                    *
0A6D BD 09E0 R      PLOOP2 JSR MPRNT2     PRINT FRACTION
0A70 7E 039D R            JMP OUTS        SPACE AND RETURN
                    *
0A73 F1 09D3 R      EXPM  CMP B DECPL     NEED SCI  NOTATION?
0A76 2C 2B                BGE ZDOT3       YES
                    *
0A78 F6 09D4 R            LDA B SIZE
0A7B 5A                   DEC B           INTEGER PART=0
                    *
0A7C BD 039D R      BLOOP2 JSR OUTS       ALIGN OUTPUT
0A7F 5A                   DEC B
0A80 26 FA                BNE BLOOP2
                    *
0A82 86 30                LDA A #'0
0A84 BD 03C3 R            JSR OUTEEE      PRINT 0 INTEGER PART
0A87 86 2E                LDA A #'        PRINT DECIMAL POINT
0A89 BD 03C3 R            JSR OUTEEE
0A8C E6 01                LDA B 1,X       GET EXPONENT
0A8E 27 08          PLOOP3 BEQ PLOOP4     NEED A LEADING ZERO?
                    *
0A90 86 30                LDA A #'0
0A92 BD 03C3 R            JSR OUTEEE      PRINT ZERO
0A95 5A                   DEC B
0A96 26 F6                BNE PLOOP3
                    *
0A98 F6 09D3 R      PLOOP4 LDA B DECPL
0A9B E0 01                SUB B 1,X       NO  OF FRACTION DIGITS
0A9D BD 09D5 R            JSR MPRINT      PRINT FRACTION
0AA0 7E 039D R            JMP OUTS        SPACE AND RETURN
                    *
0AA3 86 30          ZDOT3 LDA A #'0
0AA5 BD 03C3 R            JSR OUTEEE      PRINT 0.
0AA8 86 2E                LDA A #'
0AAA BD 03C3 R            JSR OUTEEE
0AAD E6 00                LDA B 0,X
0AAF F7 0050 C            STA B ACCUM1    SAVE EXPONENT
0AB2 E6 01                LDA B 1,X
0AB4 F7 0051 C            STA B ACCUM1+1
0AB7 C6 0E                LDA B #14       SET DIGIT COUNTER
0AB9 BD 09D5 R            JSR MPRINT      PRINT MANTISSA
0ABC BD 039D R            JSR OUTS        SPACE
0ABF 86 45                LDA A #'E
0AC1 BD 03C3 R            JSR OUTEEE      PRINT E
0AC4 F6 0050 C            LDA B ACCUM1
0AC7 C5 40                BIT B #$40
0AC9 27 04                BEQ EPLUS       CHECK EXPONENT SIGN
                    *
0ACB 86 2D                LDA A #'-       MINUS
0ACD 20 02                BRA ESGN
                    *
0ACF 86 2B          EPLUS LDA A #'+       PLUS
0AD1 BD 03C3 R      ESGN  JSR OUTEEE
0AD4 B6 0050 C            LDA A ACCUM1    PRINT EXPONENT
0AD7 84 0F                AND A #$0F
0AD9 27 03                BEQ *+5         SUPPRESS LEADING ZERO
                    *
0ADB BD 0391 R            JSR OUTHR       FIRST DIGIT
0ADE B6 0051 C            LDA A ACCUM1+1
0AE1 BD 038D R            JSR OUTHL       SECOND DIGIT
0AE4 B6 0051 C            LDA A ACCUM1+1
0AE7 BD 0391 R            JSR OUTHR       THIRD DIGIT
0AEA 7E 039D R            JMP OUTS
                    *
0AED 0001           CNTR  RMB 1
0AEE 0001           NIBBLE RMB 1          BYTE TOGGLE
0AEF 0001           SIGDIG RMB 1          SIG  DIGIT FLAG
0AF0 0001           DECFLG RMB 1          DEC  POINT FLAG
0AF1 0001           EXPTMP RMB 1          TEMPORARY EXPONENT
                    *
                    *
                    * STRING TO FLOATING-POINT CONVERSION
                    * STRING ADDRESS IN X-REGISTER
                    * VALUE ADDRESS IN "ACCUM4"
                    *
0AF2 FF 060F R      FVAL.  STX DXSAV      SAVE X-REGISTER
0AF5 FE 0056 C             LDX ACCUM4     POINT TO VALUE
```

```
0AF8 FF 0050 C        STX ACCUM1
0AFB 86 FF            LDA A #$FF
0AFD B7 0616 R        STA A RTYFLG   NO ERROR RETRY
0B00 20 16            BRA FRO        CONVERT
                *
                *
                * FLOATING INPUT ROUTINE
                * ADDRESS OF RETURN VALUE IN X-REGISTER
                *
0B02 7F 077E R FREAD  CLR FRDFLG
0B05 FF 0050 C        STX ACCUM1     SAVE POINTER
0B08 7F 0616 R        CLR RTYFLG     RETRY AFTER ERRORS
0B0B BD 0488 R        JSR INEDIT     GET DATA STRING
0B0E 86 04            LDA A #$04
0B10 A7 00            STA A 0,X      APPEND END-STRING
0B12 CE 0436 R        LDX #BUFFER
0B15 FF 060F R        STX DXSAV
0B18 FE 0050 C FRO    LDX ACCUM1     RESTORE X-REGISTER
0B1B C6 09            LDA B #9       CLEAR VALUE SPACE
0B1D 6F 00      FR1   CLR 0,X
0B1F 08               INX
0B20 5A               DEC B
0B21 26 FA            BNE FR1
                *
0B23 7F 0AF0 R        CLR DECFLG
0B26 7F 0AEF R        CLR SIGDIG
0B29 7F 0AF1 R        CLR EXPTMP
0B2C FE 0050 C        LDX ACCUM1
0B2F C6 0E            LDA B #14
0B31 F7 02BA R        STA B TEMP     14 DIGITS OF MANTISSA
0B34 86 20            LDA A #$20
0B36 A7 00            STA A 0,X      SET ZERO BIT
                *
0B38 C6 FF            LDA B #$FF     BYTE TOGGLE
0B3A F7 0AEE R        STA B NIBBLE
0B3D BD 0C4E R        JSR INCHAR     GET CHARACTER
0B40 81 2B            CMP A #'+      PLUS?
0B42 27 13            BEQ GETIN
                *
0B44 81 2D            CMP A #'-      MINUS?
0B46 26 08            BNE NOTMIN
                *
0B48 86 80            LDA A #$80     SET NEGATIVE
0B4A AA 00            ORA A 0,X
0B4C A7 00            STA A 0,X
0B4E 20 07            BRA GETIN
                *
0B50 81 2E      NOTMIN CMP A #'.     DECIMAL POINT?
0B52 26 06            BNE CRCHK      NO, CHECK FOR CR
                *
0B54 73 0AF0 R POINT  COM DECFLG     HAVE DECIMAL
                *
0B57 BD 0C4E R GETIN  JSR INCHAR     GET CHARACTER
                *
0B5A 81 0D      CRCHK CMP A #$0D     CR?
0B5C 26 03            BNE *+5        NO
0B5E 7E 0C14 R        JMP SETEXP
                *
0B61 81 04            CMP A #$04     EOM?
0B63 26 03            BNE *+5        NO
0B65 7E 0C14 R        JMP SETEXP     YES
                *
0B68 80 30            SUB A #$30     REMOVE ASCII BIAS
0B6A 27 14            BEQ GOTZER     ZERO
                *
0B6C 2B 46            BMI NOTDON     <0?
                *
0B6F 81 09            CMP A #9       >9?
0B70 22 42            BHI NOTDON
                *
0B72 7D 0AEF R        TST SIGDIG
0B75 26 09            BNE GOTZER
                *
0B77 E6 00            LDA B 0,X
0B79 C4 DF            AND B #$DF     TURN OFF ZERO BIT
0B7B E7 00            STA B 0,X
0B7D 7C 0AEF R        INC SIGDIG     HAVE SIG  DIGIT
                *
0B80 7D 02BA R GOTZER TST TEMP       FULL BUFFER?
0B83 27 02            BEQ GETIN      YES, SKIP DIGIT
                *
0B85 7D 0AEF R        TST SIGDIG     SIG  DIGIT YET?
0B88 26 0A            BNE TSTNXT     YES
                *
0B8A 7D 0AF0 R        TST DECFLG     DECIMAL POINT YET?
0B8D 27 C8            BEQ GETIN      NO
                *
0B8F 7A 0AF1 R        DEC EXPTMP     BACK UP EXPONENT
0B92 20 C3            BRA GETIN
                *
0B94 7D 0AF0 R TSTNXT TST DECFLG     DECIMAL POINT YET?
0B97 26 03            BNE STORE      YES, EXP  O.K.
                *
0B99 7C 0AF1 R        INC EXPTMP     INCREASE EXPONENT
0B9C 7D 0AEE R STORE  TST NIBBLE     WHICH HALF?
0B9F 27 04            BEQ LEFTH
                *
0BA1 48               ASL A
0BA2 48               ASL A          SHIFT OVER
0BA3 48               ASL A
0BA4 48               ASL A
0BA5 AA 02      LEFTH ORA A 2,X      MERGE IN
0BA7 A7 02            STA A 2,X
0BA9 73 0AEE R        COM NIBBLE
0BAC 27 01            BEQ NOTNXT     CHECK FOR NEW BYTE
                *
0BAE 08               INX
0BAF 7A 02BA R NOTNXT DEC TEMP       CHECK FOR BUFFER END
0BB2 20 A3            BRA GETIN
                *
0BB4 8B 30      NOTDON ADD A #$30    RESTORE BIAS
0BB6 81 45            CMP A #'E      EXPONENT?
0BB8 27 1D            BEQ EXPIN      YES
                *
0BBA 81 2E            CMP A #'       DECIMAL POINT?
0BBC 26 0A            BNE SYNERR     NO, ERROR
                *
0BBE 7D 0AF0 R        TST DECFLG     HAVE DECIMAL POINT
0BC1 26 05            BNE SYNERR     SYNTAX ERROR
                *
0BC3 73 0AF0 R        COM DECFLG     SET FLAG
0BC6 20 8F            BRA GETIN
```

```
                    *
0BC8 BD 0690 R SYNERR JSR BADERR
0BCB 7D 0616 R        TST RTYFLG    RETRY O K ?
0BCE 27 01            BEQ *+3       YES
                    *
0BD0 39               RTS           NO. QUIT
                    *
0BD1 FE 0050 C        LDX ACCUM1    ERROR RETRY
0BD4 7E 0B08 R        JMP FREAD+6
                    *
0BD7 8D 75     EXPIN  BSR INCHAR    GET CHARACTER
0BD9 FE 0050 C        LDX ACCUM1
0BDC 7F 0AED R        CLR CNTR
0BDF 81 2B            CMP A #'+     PLUS?
0BE1 27 0A            BEQ EXPIN2    SKIP IT
                    *
0BE3 81 2D            CMP A #'-     MINUS?
0BE5 26 10            BNE CHKNXT    NO
                    *
0BE7 C6 40            LDA B #$40    SET EXPONENT SIGN
0BE9 EA 00            ORA B 0,X
0BEB E7 00            STA B 0,X
0BED 8D 5F     EXPIN2 BSR INCHAR    GET CHARACTER
0BEF 81 0D            CMP A #$0D    CR?
0BF1 27 21            BEQ SETEXP
                    *
0BF3 81 04            CMP A #$04    EOM?
0BF5 27 1D            BEQ SETEXP
                    *
0BF7 80 30     CHKNXT SUB A #$30    REMOVE BIAS
0BF9 2B CD            BMI SYNERR    <0?
                    *
0BFB 81 09            CMP A #9      >9?
0BFD 23 02            BLS EXPOK
                    *
0BFF 20 C7            BRA SYNERR
                    *
0C01 E6 01     EXPOK  LDA B 1,X
0C03 58               ASL B
0C04 58               ASL B
0C05 58               ASL B
0C06 58               ASL B
0C07 1B               ABA
0C08 A7 01            STA A 1,X
0C0A 7D 0AED R        TST CNTR      TWO-DIGIT EXPONENT
0C0D 26 05            BNE SETEXP
                    *
0C0F 73 0AED R        COM CNTR
0C12 20 D9            BRA EXPIN2    CONTINUE
                    *
                    *
                    * ROUTINE TO FIX UP BCD EXPONENT
                    *
0C14 FE 0050 C SETEXP LDX ACCUM1
0C17 A6 01            LDA A 1,X     FIX EXPONENT
0C19 84 0F            AND A #$0F    CONVERT BCD TO BINARY
0C1B 36               PSH A
0C1C A6 01            LDA A 1,X
0C1E 84 F0            AND A #$F0    MULTIPLY BY 10
0C20 44               LSR A
0C21 16               TAB
0C22 44               LSR A
0C23 44               LSR A
0C24 1B               ABA
0C25 33               PUL B
0C26 1B               ABA
0C27 E6 00            LDA B 0,X     CHECK EXPONENT SIGN
0C29 C5 40            BIT B #$40
0C2B 27 01            BEQ NOSIGN
                    *
0C2D 40               NEG A         NEGATE BINARY EXPONENT
                    *
0C2E BB 0AF1 R NOSIGN ADD A EXPTMP  ADJUST BINARY EXPONENT
0C31 2A 07            BPL SETX2     EXPONENT NEGATIVE?
                    *
0C33 40               NEG A         MAKE POSITIVE
0C34 C6 40            LDA B #$40    SET EXP. SIGN
0C36 EA 00            ORA B 0,X
0C38 E7 00            STA B 0,X
                    *
0C3A 5F        SETX2  CLR B
                    *
0C3B 81 0A     SETX3  CMP A #10
0C3D 25 06            BCS SETX4
                    *
0C3F 80 0A            SUB A #10
0C41 CB 10            ADD B #$10
0C43 20 F6            BRA SETX3
                    *
0C45 1B        SETX4  ABA           SET 'H' FOR DAA
0C46 19               DAA           CONVERT BACK TO BCD
0C47 24 02            BCC NOINC
                    *
0C49 6C 00            INC 0,X       CARRY TO HIGH DIGIT
0C4B A7 01     NOINC  STA A 1,X
                    *
0C4D 39               RTS
                    *
                    *
0C4E FF 0611 R INCHAR STX DVAL      SAVE X-REGISTER
0C51 FE 060F R INCH2  LDX DXSAV
0C54 A6 00            LDA A 0,X     GET CHARACTER FROM STRING
0C56 08               INX
0C57 FF 060F R        STX DXSAV
0C5A 81 20            CMP A #$20    BLANK?
0C5C 27 F3            BEQ INCH2     SKIP BLANKS
                    *
0C5E FE 0611 R        LDX DVAL      RESTORE X-REGISTER
0C61 39               RTS
                    *
                    * FLOATING TO INTEGER CONVERSION
                    * X-REGISTER POINTS TO FLOATING NUMBER
                    * RETURNS BINARY IN X-REGISTER
                    *
0C62 A6 00     FIX    LDA A 0,X     GET EXP  FLAG BYTE
0C64 B7 0AED R        STA A CNTR    TEMP. SAVE
0C67 85 60            BIT A #$60    CHECK FOR <1
0C69 27 04            BEQ FIX2      >0?
                    *
0C6B CE 0000   FIX0   LDX #0         RETURN ZERO
0C6E 39               RTS
                    *
```

```
0C6F 84 0F      FIX2   AND A #$0F     CHECK FIRST EXP. DIGIT
0C71 26 08             BNE FIXERR     NOT ZERO, TOO BIG
                *
0C73 A6 01             LDA A 1,X      CHECK DIGITS 2 AND 3
0C75 27 F4             BEQ FIX0       EXP. =0?
                *
0C77 81 06             CMP A #6       MAX  EXP =5
0C79 25 0B             BCS FIX3       O K ?
                *
0C7B BD 06A8 R  FIXERR JSR OVERR      OVERFLOW ERROR
0C7E B7 0050 C         STA A ACCUM1   RETURN MAX  INTEGER
0C81 F7 0051 C         STA B ACCUM1+1
0C84 20 46             BRA FIX7       SET SIGN AND RETURN
                *
0C86 B7 0AEF R  FIX3   STA A SIGDIG   TEMP  SAVE
0C89 7F 0050 C         CLR ACCUM1
0C8C 7F 0051 C         CLR ACCUM1+1   BINARY SPACE
0C8F 7F 0AEE R         CLR NIBBLE     SET BYTE TOGGLE
                *
0C92 A6 02      FIX4   LDA A 2,X      GET DIGITS
0C94 7D 0AEE R         TST NIBBLE     WHICH HALF?
0C97 27 04             BEQ FIX5
                *
0C99 84 0F             AND A #$0F     RIGHT HALF
0C9B 20 04             BRA FIX6
                *
0C9D 44         FIX5   LSR A          LEFT HALF
0C9E 44                LSR A
0C9F 44                LSR A
0CA0 44                LSR A
0CA1 5F         FIX6   CLR B
0CA2 BB 0051 C         ADD A ACCUM1+1 ADD IN VALUE
0CA5 F9 0050 C         ADC B ACCUM1
0CA8 25 D1             BCS FIXERR     OVERFLOW ERROR?
                *
0CAA B7 0051 C         STA A ACCUM1+1
0CAD F7 0050 C         STA B ACCUM1
0CB0 7A 0AEF R         DEC SIGDIG     DONE YET?
0CB3 27 17             BEQ FIX7
                *
0CB5 FF 060F R         STX DXSAV
0CB8 CE 000A           LDX #10
0CBB BD 0195 R         JSR MULT       MULTIPLY BY 10
0CBE FF 0050 C         STX ACCUM1
0CC1 FE 060F R         LDX DXSAV
0CC4 73 0AEE R         COM NIBBLE     SWITCH HALVES
0CC7 26 C9             BNE FIX4
                *
0CC9 08                INX
0CCA 20 C6             BRA FIX4       NEED NEW BYTE
                *
0CCC B6 0AED R  FIX7   LDA A CNTR     TEST SIGN
0CCF 85 80             BIT A #$80
0CD1 27 06             BEQ FIX8       PLUS?
                *
0CD3 CE 0050 C         LDX #ACCUM1    NEGATE VALUE
0CD6 BD 01F6 R         JSR NEGATE
0CD9 FE 0050 C  FIX8   LDX ACCUM1     RETURN VALUE
0CDC 39                RTS
                *
                *
                * INTEGER TO FLOATING CONVERSION
                * ACCUM1 CONTAINS INTEGER VALUE
                * X-REGISTER CONTAINS ADDRESS FOR RESULT
                *
0CDD FF 0611 R  FLOAT  STX DVAL
0CE0 C6 09             LDA B #9       CLEAR FLOAT SPACE
0CE2 6F 00      FLOT2  CLR 0,X
0CE4 08                INX
0CE5 5A                DEC B          DONE?
0CE6 26 FA             BNE FLOT2
                *
0CE8 7F 0AEE R         CLR NIBBLE
0CEB 7F 0AEF R         CLR SIGDIG
0CEE FE 0050 C         LDX ACCUM1     CHECK FOR ZERO
0CF1 26 08             BNE FLOT3
                *
0CF3 FE 0611 R         LDX DVAL
0CF6 86 20             LDA A #$20     SET ZERO BIT
0CF8 A7 00             STA A 0,X
0CFA 39                RTS
                *
0CFB 2A 0D      FLOT3  BPL FLOT4      CHECK FOR MINUS
                *
0CFD 86 80             LDA A #$80     SET MINUS BIT
0CFF FE 0611 R         LDX DVAL
0D02 A7 00             STA A 0,X
0D04 CE 0050 C         LDX #ACCUM1    NEGATE VALUE
0D07 BD 01F6 R         JSR NEGATE
                *
0D0A FE 0611 R  FLOT4  LDX DVAL
0D0D FF 0052 C         STX ACCUM2
0D10 CE 0D7D R         LDX #KTOK      POINT TO CONSTANTS
0D13 FF 060F R         STX DXSAV
0D16 86 05             LDA A #5
0D18 B7 0AF1 R         STA A EXPTMP   UP TO FIVE DIGITS
0D1B 7F 0AED R  FLOT5  CLR CNTR       CONVERT TO BCD
0D1E B6 0050 C         LDA A ACCUM1
0D21 F6 0051 C         LDA B ACCUM1+1
0D24 E0 01      FLOT6  SUB B 1,X
0D26 A2 00             SBC A 0,X
0D28 25 05             BCS FLOT7
                *
0D2A 7C 0AED R         INC CNTR
0D2D 20 F5             BRA FLOT6
                *
0D2F EB 01      FLOT7  ADD B 1,X      RESTORE PARTIAL SUM
0D31 A9 00             ADC A 0,X
0D33 B7 0050 C         STA A ACCUM1
0D36 F7 0051 C         STA B ACCUM1+1
0D39 7D 0AEF R         TST SIGDIG     LEADING DIGIT?
0D3C 26 0D             BNE FLOT9
                *
0D3E 7D 0AED R         TST CNTR       DIGIT=0?
0D41 26 05             BNE FLOT8
                *
0D43 7A 0AF1 R         DEC EXPTMP     ONE LESS SIG. DIGIT
0D46 20 1F             BRA FLOT11
                *
0D48 7C 0AEF R  FLOT8  INC SIGDIG     HAVE SIG  DIGIT
0D4B FE 0052 C  FLOT9  LDX ACCUM2     POINT TO MANTISSA
0D4E B6 0AED R         LDA A CNTR     GET DIGIT
```

```
0D51 7D 0AEE R        TST NIBBLE     WHICH HALF?
0D54 26 04            BNE FLOT10
                *
0D56 48               ASL A          SHIFT TO LEFT HALF
0D57 48               ASL A
0D58 48               ASL A
0D59 48               ASL A
0D5A AA 02      FLOT10 ORA A 2,X     MERGE IN
0D5C A7 02            STA A 2,X
0D5E 73 0AEE R        COM NIBBLE     SWITCH HALVES
0D61 26 04            BNE FLOT11
                *
0D63 08               INX
0D64 FF 0052 C        STX ACCUM2     NEED NEW BYTE
0D67 FE 060F R FLOT11 LDX DXSAV      MOVE CONSTANT POINTER
0D6A 08               INX
0D6B 08               INX
0D6C FF 060F R        STX DXSAV
0D6F 8C 0D87 R        CPX #KTOK+10   DONE?
0D72 26 A7            BNE FLOT5
                *
0D74 B6 0AF1 R        LDA A EXPTMP   FINISH UP
0D77 FE 0611 R        LDX DVAL
0D7A A7 01            STA A 1,X      SET EXPONENT
0D7C 39               RTS            DONE!
                *
0D7D 2710       KTOK  FDB 10000      POWERS OF 10 TABLE
0D7F 03E8             FDB 1000
0D81 0064             FDB 100
0D83 000A       TEN10 FDB 10
0D85 0001             FDB 1
                *
                *
                * FLOATING-POINT COMPARISON ROUTINE
                * ADDRESS OF SECOND ARG. IN "ACCUM1"
                * ADDRESS OF FIRST ARG. IN X-REGISTER
                * RESULT IN CONDITION CODES
                *
0D87 BD 009B R FCOMPR JSR FTAREG     FIRST ARGUMENT
0D8A FE 0050 C        LDX ACCUM1
0D8D BD 009E R        JSR FTCREG     SECOND ARGUMENT
0D90 BD 00A7 R        JSR FSUB       SUBTRACT ARGS.
0D93 CE 062C R        LDX #FTEMP
0D96 BD 00A1 R        JSR FFDREG     RESULT IN FTEMP
0D99 B6 062C R        LDA A FTEMP    GET SIGNS
0D9C 85 20            BIT A #$20     CHECK ZERO BIT
0D9E 26 0A            BNE FEQUAL     IF SET, EQUALS
                *
0DA0 85 80            BIT A #$80     CHECK SIGN BIT
0DA2 26 03            BNE FLTHAN     IF SET, LESS THAN
                *
0DA4 86 01      FGTHAN LDA A #1      RETURN +1
0DA6 39               RTS
                *
0DA7 86 FF      FLTHAN LDA A #$FF    RETURN -1
0DA9 39               RTS
                *
0DAA 86 00      FEQUAL LDA A #0      RETURN 0
0DAC 39               RTS
                *
                * FLOATING-POINT ABSOLUTE VALUE ROUTINE
                * ADDRESS OF VALUE IN X-REGISTER
                *
0DAD A6 00      FABS  LDA A 0,X      GET FLAG BYTE
0DAF 84 7F            AND A #$7F     MASK OUT SIGN BIT
0DB1 A7 00            STA A 0,X      RESTORE FLAG BYTE
0DB3 39               RTS
                *
                *
                * FLOATING-POINT SIGN ROUTINE
                * ADDRESS OF VALUE IN X-REGISTER
                *
0DB4 A6 00      FSIGN LDA A 0,X
0DB6 85 20            BIT A #$20     CHECK FOR ZERO
0DB8 26 1D            BNE FSIGN2     IF ZERO, DONE!
                *
0DBA C6 10            LDA B #$10
0DBC E7 02            STA B 2,X      MAKE MANTISSA=1
0DBE 6F 03            CLR 3,X
0DC0 6F 04            CLR 4,X
0DC2 6F 05            CLR 5,X
0DC4 6F 06            CLR 6,X
0DC6 6F 07            CLR 7,X
0DC8 6F 08            CLR 8,X
0DCA 6F 01            CLR 1,X        MAKE EXPONENT=1
0DCC 6C 01            INC 1,X
0DCE 6F 00            CLR 0,X
0DD0 4D               TST A          CHECK SIGN
0DD1 2A 04            BPL FSIGN2     IF PLUS, DONE!
                *
0DD3 86 80            LDA A #$80     SET MINUS SIGN
0DD5 A7 00            STA A 0,X
0DD7 39         FSIGN2 RTS
                *
                *
                * FLOATING-POINT NEGATE ROUTINE
                * ADDRESS OF VALUE IN X-REGISTER
                *
0DD8 A6 00      FNEG  LDA A 0,X      GET FLAG BYTE
0DDA 88 80            EOR A #$80     SWITCH SIGN
0DDC A7 00            STA A 0,X      RESTORE FLAG BYTE
0DDE 39               RTS
                *
                * SPECIAL CODE FOR CP/68 INTERFACE
                *
                *
                * LINEPRINTER FCB
                *
0DDF 0002       LPTFCB RMB 2
0DE1 4C               FCC 'LPT'
0DE4 0080             FDB $0080
0DE6 0E09     R       FDB LPTBUF
0DE8 0021             RMB 33
                *
0E09 0084       LPTBUF RMB 132
                *
                * CONSOLE INPUT FCB
                *
0E8D 0002       CNFCBI RMB 2
0E8F 43               FCC 'CON'
0E92 0000             FDB $0000
0E94 0EB7     R       FDB CNBUF1
```

```
0F96 0021                RMB 33
                 *
0E87 0084        CNBUF1  RMB 132
                 *
                 * CONSOLE OUTPUT FCB
                 *
0F3B 0002        CNFCBO  RMB 2
0F3D 43                  FCC 'CON'
0F40 0080                FDB $0080
0F42 0F65     R          FDB CNBUF2
0F44 0021                RMB 33
                 *
0F65 0084        CNBUF2  RMB 132
                 *
                 * FCB EQUATES
                 *
0FE9 0005        FCBSTA  EQU 5
0FE9 0007        FCBDBA  EQU 7
0FE9 0027        FCBIND  EQU 39
                 *
                 *
0FE9 3F          INITIO  SWI             CALL CP/68
0FEA 33                  FCB 51          "INITDK"
0FEB CE 0F3B R           LDX #CNFCBO     INIT. CONSOLE OUTPUT
0FEE 6F 05               CLR FCBSTA,X
0FF0 A6 07               LDA A FCBDBA,X
0FF2 E6 08               LDA B FCBDBA+1,X
0FF4 A7 27               STA A FCBIND,X
0FF6 E7 28               STA B FCBIND+1,X
0FF8 CE 0DDF R           LDX #LPTFCB     INIT. LINEPRINTER OUTPUT
0FFB 6F 05               CLR FCBSTA,X
0FFD EE 07               LDX FCBDBA,X
0FFF 86 0C               LDA A #$0C      INITIAL FORMFEED
1001 A7 00               STA A 0,X
1003 08                  INX
1004 3F                  SWI             CALL CP/68
1005 02                  FCB 2           "TXAB"
1006 CE 0DDF R           LDX #LPTFCB
1009 A7 27               STA A FCBIND,X
100B E7 28               STA B FCBIND+1,X
100D CE 0E8D R           LDX #CNFCBI     INIT. CONSOLE INPUT
1010 6F 05               CLR FCBSTA,X
1012 6F 27               CLR FCBIND,X    FLAG BUFFER EMPTY
1014 39          OUTRET  RTS
                 *
                 *
1015 81 04       OUTLPT  CMP A #$04      CNTL-D?
1017 27 FB               BEQ OUTRET      IF SO, QUIT
                 *
1019 81 0A               CMP A #$0A      LINEFEED?
101B 27 F7               BEQ OUTRET      IF SO, QUIT
                 *
101D 3F                  SWI             CALL CP/68
101E 00                  FCB 0           "PSHALL"
101F CE 0DDF R           LDX #LPTFCB
1022 EE 27               LDX FCBIND,X    POINT TO BUFFER
1024 A7 00               STA A 0,X       STORE CHARACTER
1026 08                  INX
1027 36                  PSH A
1028 3F                  SWI             CALL CP/68
1029 02                  FCB 2           "TXAB"
102A CE 0DDF R           LDX #LPTFCB
102D A7 27       OUTL1   STA A FCBIND,X
102F E7 28               STA B FCBIND+1,X
1031 32                  PUL A
1032 81 0D               CMP A #$0D      C R ?
1034 27 03               BEQ OUTL2       YES
                 *
1036 3F                  SWI             CALL CP/68
1037 01                  FCB 1           "PULALL"
1038 39                  RTS
                 *
1039 3F          OUTL2   SWI             CALL CP/68
103A 13                  FCB 19          "IOHDR"
103B A6 07               LDA A FCBDBA,X
103D E6 08               LDA B FCBDBA+1,X
103F A7 27               STA A FCBIND,X
1041 E7 28               STA B FCBIND+1,X
1043 6F 05               CLR FCBSTA,X
1045 3F                  SWI             CALL CP/68
1046 01                  FCB 1           "PULALL"
1047 39                  RTS
                 *
                 *
1048 81 0A       PUTOUT  CMP A #$0A      LINEFEED?
104A 27 C8               BEQ OUTRET      IF SO, QUIT
                 *
104C 3F                  SWI             CALL CP/68
104D 00                  FCB 0           "PSHALL"
104E CE 0F3B R           LDX #CNFCBO
1051 EE 27               LDX FCBIND,X    POINT TO BUFFER
1053 A7 00               STA A 0,X       STORE CHARACTER
1055 08                  INX
1056 36                  PSH A
1057 3F                  SWI             CALL CP/68
1058 02                  FCB 2           "TXAB"
1059 CE 0F3B R           LDX #CNFCBO
105C A7 27               STA A FCBIND,X
105E E7 28               STA B FCBIND+1,X
1060 32                  PUL A
1061 81 0D               CMP A #$0D      C R ?
1063 27 D4               BEQ OUTL2       YES
                 *
1065 81 04               CMP A #$04      CNTL-D?
1067 27 D0               BEQ OUTL2       YES
                 *
1069 3F                  SWI             CALL CP/68
106A 01                  FCB 1           "PULALL"
106B 39                  RTS
                 *
                 *
106C 37          PUTIN   PSH B
106D 3F                  SWI             CALL CP/68
106E 05                  FCB 5           "PSHX"
106F CE 0E8D R           LDX #CNFCBI
1072 6D 27               TST FCBIND,X    BUFFER INIT  YET?
1074 26 0C               BNE IN2         YES
                 *
1076 3F          IN1     SWI             CALL CP/68
1077 13                  FCB 19          "IOHDR"
1078 A6 07               LDA A FCBDBA,X
107A E6 08               LDA B FCBDBA+1,X
```

```
107C A7 27              STA A FCBIND,X
107E E7 28              STA B FCBIND+1,X
1080 6F 05              CLR FCBSTA,X
                  *
1082 EE 27        IN2   LDX FCBIND,X    POINT TO BUFFER
1084 A6 00              LDA A 0,X       GET CHARACTER
1086 08                 INX
1087 36                 PSH A
1088 3F                 SWI             CALL CP/68
1089 02                 FCB 2           "TXAB"
108A CE 0E8D R          LDX #CNFCBI
108D A7 27              STA A FCBIND,X
108F E7 28              STA B FCBIND+1,X
1091 32                 PUL A
1092 81 0D              CMP A #$0D      C.R.?
1094 27 04              BEQ IN3         YES
                  *
1096 3F                 SWI             CALL CP/68
1097 06                 FCB 6           "PULX"
1098 33                 PUL B
1099 39                 RTS
                  *
109A 6F 27        IN3   CLR FCBIND,X    RESET FOR RE-INIT.
109C 3F                 SWI             CALL CP/68
109D 06                 FCB 6           "PULX"
109E 33                 PUL B
109F 39                 RTS
                  *
                  *
10A0 CE 1015 R    LPON  LDX #OUTIPT
10A3 FF 0004 R          STX SYSOUT+1
10A6 39                 RTS
                  *
10A7 CE 1048 R    LPOFF LDX #PUTOUT
10AA FF 0004 R          STX SYSOUT+1
10AD 39                 RTS
                  *
                        END
```

CURTO	0069	C
EOF	0065	C
ERR	0067	C
.KTBL	0006	RN
ABS	0205	R
ABS2	020B	R
ACCUM1	0050	C
ACCUM2	0052	C
ACCUM3	0054	C
ACCUM4	0056	C
ACCUM5	0058	C
ACOS	00BC	RX
ADD7	0161	R
ADRFM	0063	C
ADRTO	0061	C
AND7	0143	R
ASCII	080B	RN
ASCII2	081D	R
ASIN	00B9	RX
ATAN	00BF	RX
BADERR	0690	R
BADMSG	0699	R
BLOOP	0A49	R
BLOOP2	0A7C	R
BRTABL	002C	R
BUFFER	0436	R
CBLNK	05B1	R
CHKNXT	0BF7	R
CLOSE	00DD	RX
CNBUF1	0EB7	R
CNBUF2	0F65	R
CNFCBI	0E8D	R
CNFCBO	0F3B	R
CNIR	0AED	R
CNVRT2	04D0	R
CNVRT3	04ED	R
COMPAR	017F	R
COS	00B3	RX
COUNT	0274	R
CRCHK	0B5A	R
CRI	041E	R
CRLF	0411	RN
CURIOB	075D	R
CVDB1	059D	R
CVDB2	05C4	R
CVDB3	0600	R
CVHB1	04FD	R
CVHBD	0543	R
CVHBS	0547	R
CVHBSD	055B	R
DECFLG	0AF0	R
DECIM	056D	R
DECIM2	0583	R
DECIM3	0595	R
DECIM4	060B	R
DELFL	09D3	R
DELAY	0958	R
DERR1	099D	R
DERR2	09B6	R
DIGERR	0996	R

DIGS2	0972	R
DIGS3	0981	R
DIGSFT	0963	R
DIV	0214	R
DIV2	021F	R
DIV3	0226	R
DIV4	0235	R
DIV5	0242	R
DIV6	0259	R
DIV7	0254	R
DIVERR	0651	RN
DIVMSG	065C	R
DTIME	0962	R
DVAL	0611	R
DXSAV	060F	R
EPLUS	0ACF	R
ERROR	055C	R
ERROR2	05B5	R
ESGN	0AD1	R
EXOR	014D	R
EXP	028D	R
EXP1	0295	R
EXP2	0296	R
EXP3	029E	R
EXP4	02B1	R
EXPIN	0BD7	R
EXPIN2	0BED	R
EXPM	0A73	R
EXPOK	0C01	R
EXPTMP	0AF1	R
F2	01E1	R
F3	01F1	R
FABS	0DAD	RN
FADD	00A4	RX
FASCII	082A	R
FCBDBA	0007	
FCBIND	0027	
FCBSTA	0005	
FCOMPR	0D87	RN
FDIV	00AD	RX
FEQUAL	0DAA	R
FEXP	00CB	RX
FFDREG	00A1	RX
FGTHAN	0DA4	R
FINISH	018B	R
FINSHS	01DB	R
FIX	0C62	RN
FIX0	0C6B	R
FIX2	0C6F	R
FIX3	0C86	R
FIX4	0C92	R
FIX5	0C9D	R
FIX6	0CA1	R
FIX7	0CCC	R
FIX8	0CD9	R
FIXERR	0C7B	R
FLAG	03F3	R
FLFEED	07AF	R
FLOAT	0CDD	RN
FLOT10	0D5A	R
FLOT11	0D67	R

FLOT2	0CE2	R
FLOT3	0CFB	R
FLOT4	0D0A	R
FLOT5	0D1B	R
FLOT6	0D24	R
FLOT7	0D2F	R
FLOT8	0D48	R
FLOT9	0D4B	R
FLREAD	0734	R
FLTHAN	0DA7	R
FMUL	00AA	RX
FNEG	0DD8	RN
FPRINT	09FD	R
FR0	0B18	R
FR1	0B1D	R
FRDFLG	077E	R
FREAD	0B02	R
FSIGN	0DB4	RN
FSIGN2	0DD7	R
FSPACE	07BB	R
FSUB	00A7	RX
FTAREG	009B	RX
FTCREG	009E	RX
FTEMP	062C	R
FVAL	0AF2	R
FWRFLG	077F	R
FWRITE	0791	R
GETB	0125	RX
GETIN	0B57	R
GOTZER	0B80	R
ILLERR	0706	RN
ILLMSG	0712	R
IN1	1076	R
IN2	1082	R
IN3	109A	R
INCH2	0C51	R
INCHAR	0C4F	R
INE1	03A8	R
INE2	03BC	R
INE3	03C0	R
INEDIT	0488	R
INEFE	03A1	RN
INITIO	0FE9	RN
INP1	0498	R
INP2	049F	R
INP3	04A5	R
INP4	04AD	R
INP5	04BA	R
INP6	04BF	R
INPUT	04C5	R
IREAD	0728	R
IWRITZ	0788	R
IWRITE	0780	R
KIOK	0D7D	R
LEFTH	0BA5	R
LEN	07C7	R
LEN2	07CD	R
LEN3	07EB	R
LENERR	06F1	R
LENMSG	06F1	R
LENR	07E8	R

```
LFEED   0416 R
LOG10   00C2 RX
LPOFF   10A7 RN
LPON    10A0 RN
LPTBUF  0E09 R
LPTFCB  0DDF R
MID     083B R
MID2    0852 R
MID3    0873 R
MOD     0286 R
MOVER   0101 RX
MPRINT  09D5 R
MPRNT2  09E0 R
MPY1    019A R
MPY2    01AF R
MQUIT   09EC R
MULT    0195 R
NATLOG  00C5 RX
NEGATE  01F6 R
NEGERR  0670 R
NEGMSG  0679 R
NIBBLE  0AFE R
NODIG1  0985 R
NODIG2  098D R
NOINC   0C4B R
NOSIGN  0C2E R
NOT     012B R
NOTMON  0BB4 R
NOTMIN  0B50 R
NOTNXT  0BAF R
NOTZ    0A0C R
OPEN    00DA RX
OR      0157 R
OUTB    012B RX
OUTE2   03DC R
OUTE3   03E9 R
OUTEEE  03C3 R
OUTFND  03FE R
OUTHL   038D R
OUTHR   0391 R
OUTL1   102D R
OUTL2   1039 R
OUTPT   1015 R
OUTPUT  0617 R
OUTRET  1014 R
OUTS    039D R
OUTSS   0620 R
OVERR   06A8 R
OVMSG   06B6 R
PARAM   08F6 R
PARAM2  08F2 R
PARAM3  0933 R
PARAM4  0942 R
PARERR  090F R
PARLN   0917 R
PDATA   03E7 R
PDATA1  0409 RN
PDATA2  03F4 R
PEEK    0138 R
PERR    0637 R
PLOOP   0A4F R
PLOOP1  0A6A R
PLOOP2  0A6D R
PLOOP3  0A8F R
PLOOP4  0A98 R
PLUS    0A17 R
POINT   0B54 R
POKE    013D R
POPF    0368 R
POPF2   0372 R
POPI    0358 R
POPS    037B R
POPS2   0383 R
POWER   02B8 R
PRMPT   0432 R
PUSHF   0326 R
PUSHF2  0330 R
PUSHI   0313 R
PUSHS   0339 R
PUSHS2  0344 R
PUSHS3  034D R
PUTIN   106C R
PUTOUT  1048 R
QOUT    0892 R
QWRITE  07A5 R
RAND    02DD R
RAND2   02E8 R
RAND3   02FC R
RAND4   0304 R
RAND5   030D R
RESTRT  02F2 R
RETM    0435 R
RETRN   0321 R
RETRY   0400 R
REWIND  00E0 RX
RTEMP   002A R
RTYFLG  0616 R
RUNSTK  0000 C
RUNTIM  0000 RN
SEED    0311 R
SET2    01C7 R
SET3    01D4 R
SETEXP  0C14 R
SETUP   0181 R
SETUPS  01B4 R
SETX2   0C3A R
SETX3   0C3B R
SETX4   0C45 R
S16     01F5 R
S16Z    0615 R
S16DIG  0AEF R
S16N    0276 R
S16N2   027C R
S16N3   0282 R
S1N     0080 RX
S17E    09D4 R
SPACES  08CC R
SPC1    08D6 R
SPCRTN  08E5 R
SRD1    0745 R
SRD2    0755 R
SREAD   0740 R
SREC1   076D R
SREC2   0779 R
SRECRD  0767 R
STOP    042F R
STORE   0B9C R
STRCMP  02BC R
STREND  02CB R
STRIN   0BB9 R
STRLP   02C2 R
STROUT  089B R
STRU1   08A4 R
STRU12  08B2 R
SUBZ    0170 R
SWRITF  079B R
SYNERR  0BC8 R
SYSIN   0000 R
SYSOUT  0003 R
TAN     0086 RX
TBL     0028 R
TEMP    02BA R
TEN10   0B83 R
TLNEXP  00C8 RX
TENVAL  0613 R
TERR    06D6 R
TRUNER  06C4 R
TSTNXT  0B94 R
VAL     07EF R
VAL2    0800 R
XFER    087B R
XPOINT  03F1 R
XSAV    0635 R
XTEMP   03EF R
XTOY    00CE RX
ZDOT    0A1A R
ZDOT1   0A2F R
ZDOT2   0A37 R
ZDOT2A  0A40 R
ZDOT3   0A43 R
ZERO    0003 R
ZMOVER  0421 R
```

```
0001   0000 0000     N          NAM     FLOTPT    (STROBAL +)
0002                            *
0003                            * COURTESY OF MOTOROLA M6800 USER GROUP LIBRARY
0004                            *          WRITTEN BY RAYMOND H  NAUGLE
0005                            *          PROGRAM #32
0006                            * MODIFIED BY R  GRAPPEL AND J.  HEMENWAY
0007                            * ADJUSTABLE PRECISION FUNCTION
0008                            *
0009   0000 0000     C          CMN     RUNSTK,80
0010   0000 0050     C          CMN     ACCUM1,2
0011   0000 0052     C          CMN     ACCUM2,2
0012   0000 0054     C          CMN     ACCUM3,2
0013   0000 0056     C          CMN     ACCUM4,2
0014   0000 0058     C          CMN     ACCUM5,9
0015   0000 0061     C          CMN     ADRTO,2
0016   0000 0063     C          CMN     ADRFM,2
0017                            *
0018   0000 009F     N          ENT     F1AREG
0019   0000 00AF     N          ENT     F1CREG
0020   0000 00BF     N          ENT     FFDREG
0021   0000 01D3     N          ENT     FADD
0022   0000 0405     N          ENT     FSUB
0023   0000 0410     N          ENT     FMUL
0024   0000 065E     N          ENT     FDIV
0025   0000 00CD     N          ENT     MOVER
0026   0000 0003     N          ENT     DIGSET
0027                            *
0028   0000 7F 0000  X          EXT     DIVERR    DIVIDE BY ZERO TRAP
0029                            *
0030                            * SET THE PRECISION OF FLOATING ARITHMETIC
0031                            * NO  OF DIGITS PASSED IN INDEX REGISTER
0032                            * MUST BE MULTIPLE OF TWO, GREATER THAN THREE,
0033                            * AND LESS THAN 15
0034                            * IF NOT, DEFAULT TO 14 DIGITS
0035                            *
0036   0003 FF 0050 C DIGSET STX ACCUM1    SAVE VALUE
0037   0006 B6 0051 C        LDA A ACCUM1+1
0038   0009 2B 43           BMI DFAULT    BAD VALUE?
0039                            *
0040   000B 81 04           CMP A #4
0041   000D 2D 3F           BLT DFAULT    BAD VALUE?
0042                            *
0043   000F 81 0E           CMP A #14
0044   0011 2E 3B           BGT DFAULT    BAD VALUE?
0045                            *
0046   0013 47              ASR A         DIVIDE BY 2
0047   0014 B7 0065 R       STA A NB      BYTE COUNT
0048   0017 48              ASL A         GET DIGIT COUNT
0049   0018 5F              CLR B
0050   0019 81 0A     DIGST2 CMP A #10     CONVERT TO BCD
0051   001B 25 06           BCS DIGST3
0052                            *
0053   001D 80 0A           SUB A #10
0054   001F CB 10           ADD B #10
0055   0021 20 F6           BRA DIGST2
0056                            *
0057   0023 1B        DIGST3 ABA
0058   0024 19              DAA
0059   0025 B7 0066 R       STA A NB      BCD DIGIT COUNT
0060   0028 CE 0075 R       LDX #AREG     LOAD POINTER
```

```
002B 86 07              LDA A #7
002D B0 0065 R          SUB A NB        DIFF. FROM 14 DIGITS
0030 27 04              BEQ DIGST5      NO OFFSET
                  *
0032 09           DIGST4 DEX            MOVE POINTER
0033 4A                 DEC A
0034 26 FC              BNE DIGST4      DONE YET?
                  *
0036 FF 0067 R    DIGST5 STX AMOD       SET AREG POINTER
0039 8D 22              BSR XINC7       ADD 7 TO INDEX
003B FF 0069 R          STX BMOD        SET BREG POINTER
003E 8D 1D              BSR XINC7
0040 FF 006B R          STX CMOD        SET CREG POINTER
0043 8D 18              BSR XINC7
0045 FF 006D R          STX DMOD        SET DREG POINTER
0048 8D 13              BSR XINC7
004A FF 006F R          STX EMOD        SET EREG POINTER
004D 39                 RTS
                  *
004E 86 07        DFAULT LDA A #7       SET DEFAULT VALUES
0050 B7 0065 R          STA A NB
0053 86 14              LDA A #$14
0055 B7 0066 R          STA A ND
0058 CE 0075 R          LDX #AREG
005B 20 D9              BRA DIGST5
                  *
005D 08           XINC7 INX             7 INCREMENTS
005E 08                 INX
005F 08                 INX
0060 08                 INX
0061 08                 INX
0062 08                 INX
0063 08                 INX
0064 39                 RTS
                  *
0065 07           NB    FCB 7           NUMBER OF BYTES IN FRACTION
0066 14           ND    FCB $14         NUMBER OF DIGITS (BCD)
0067 0075      R  AMOD  FDB AREG
0069 007C      R  BMOD  FDB BREG
006B 0083      R  CMOD  FDB CREG
006D 008A      R  DMOD  FDB DREG
006F 0091      R  EMOD  FDB EREG
                  *
                  *
                  *   THE FOLLOWING DEFINES THE RAM LOCATIONS
                  *   NEEDED TO OPERATE THE SUBROUTINES
                  *   THE ORDER SHOULD NOT BE CHANGED
                  *
0071 0002         EXP2  RMB   2         STORAGE FOR FLOATING EXP  #2
0073 0002         EXP1  RMB   2         STORAGE FOR FLOATING EXP  #1
0075 0007         AREG  RMB   7         ENTER FIRST NUMBER HERE
007C 0007         BREG  RMB   7         SCRATCH
0083 0007         CREG  RMB   7         ENTER SECOND NUMBER HERE
008A 0007         DREG  RMB   7         FLOATING POINT ANSWER
0091 0007         EREG  RMB   7         INTEGER ANSWER
0098 0002               RMB   2         BUFFER FOR FLOATING POINT
009A 0002         ADRSAV RMB  2         ADDRESS POINTER
009C 0002         EXPSAV RMB  2         EXPONENT SCRATCH
009E 0001         COUNTR RMB  1         COUNTER
                  *
009F F6 0065 R    FTAREG LDA B  NB      NUMBER OF BYTES TO MOVE
00A2 5C                 INC B
00A3 5C                 INC B           ROOM FOR EXPONENT
00A4 FF 0063 C          STX     ADRFM   SAVE ADDRESS FROM
00A7 CE 0075 R          LDX     #AREG
00AA FF 0061 C          STX     ADRTO   SAVE ADDRESS TO
00AD 20 1E              BRA     MOVER   GO TO MOVER
                  *
00AF F6 0065 R    FTCREG LDA B  NB      NUMBER OF BYTES TO MOVE
00B2 5C                 INC B
00B3 5C                 INC B           ROOM FOR EXPONENT
00B4 FF 0063 C          STX     ADRFM   SAVE ADDRESS FROM
00B7 CE 0083 R          LDX     #CREG
00BA FF 0061 C          STX     ADRTO   SAVE ADDRESS TO
00BD 20 0E              BRA     MOVER   GO TO MOVER
                  *
00BF F6 0065 R    FFDREG LDA B  NB      NUMBER OF BYTES TO MOVE
00C2 5C                 INC B
00C3 5C                 INC B           ROOM FOR EXPONENT
00C4 FF 0061 C          STX     ADRTO   SAVE ADDRESS TO
00C7 CE 008A R          LDX     #DREG
00CA FF 0063 C          STX     ADRFM   SAVE ADDRESS FROM
                  *
00CD FE 0063 C    MOVER LDX     ADRFM   MOVE DATA FROM LOCATION
00D0 A6 00              LDA A   0,X     POINTED AT BY ADRFM
00D2 08                 INX             TO LOCATION POINTED AT
00D3 FF 0063 C          STX     ADRFM   BY ADRTO
00D6 FE 0061 C          LDX     ADRTO
00D9 A7 00              STA A   0,X
00DB 08                 INX
00DC FF 0061 C          STX     ADRTO
00DF 5A                 DEC B
00E0 26 EB              BNE     MOVER   MOVER FINISHED?
                  *
00E2 39                 RTS
                  *   THIS SUBROUTINE ADDS TWO UNSIGNED
                  *   INTEGER NUMBERS
                  *     1ST NUMBER IN THE A REGISTER
                  *     2ND NUMBER IN THE C REGISTER
                  *   THE ANSWER IS IN THE E REGISTER
                  *   IF ANY OVERFLOW OCCURED WHILE ADDING THE NUMBERS
                  *   IT WILL BE PLACED IN LOCATION EREG-1
                  *
00E3 FE 0069 R    ADD   LDX     BMOD
00E6 F6 0065 R          LDA B   NB
00E9 0C                 CLC
00EA 09           ADD10 DEX               ADDITION LOOP
00EB A6 00              LDA A   0,X
00ED A9 0E              ADC A   14,X
00EF 19                 DAA
00F0 A7 1C              STA A   28,X
00F2 5A                 DEC B
00F3 26 F5              BNE     ADD10
                  *
00F5 F7 0090 R          STA B   EREG-1
00F8 24 03              BCC     ADD20   ADDITION OVERFLOW ?
                  *
00FA 7C 0090 R          INC     EREG-1  YES, ADD TO NUMBER STRING
00FD CE 0090 R    ADD20 LDX     #EREG-1
0100 09           ADD30 DEX               CLEAR D REGISTER
0101 E7 00              STA B   0,X
0103 8C 008A R          CPX     #DREG
```

```
0184  0106 26 F8              BNE     ADD30
0185                   *
0186  0108 39                 RTS
0188                   *    THIS SUBROUTINE SUBTRACTS TWO UNSIGNED
0189                   *    INTEGER NUMBERS
0190                   *      1ST NUMBER IN A REGISTER
0191                   *      2ND NUMBER IN C REGISTER
0192                   *    THE ANSWER IS IN THE E REGISTER IN MAGNITUDE
0193                   *    FORM   IF THE NUMBER IN THE C REGISTER IS
0194                   *    LARGER THAN THE NUMBER IN THE A REGISTER
0195                   *    A FLAG WILL BE SET IN THE MSB OF
0196                   *    LOCATION DREG INDICATING A NEGATIVE ANSWER
0197                   *
0198  0109 FE 0068 R SUB      LDX     CMOD
0199  010C F6 0065 R          LDA B   NB
0200  010F 09         SUB10   DEX                     SETUP CREG FOR SUBTRACT
0201  0110 86 99              LDA A   #$99            9'S COMPLEMENT FORM
0202  0112 A0 07              SUB A   7,X
0203  0114 A7 07              STA A   7,X
0204  0116 5A                 DEC B
0205  0117 26 F6              BNE     SUB10
0206                   *
0207  0119 FE 0069 R          LDX     BMOD
0208  011C F6 0065 R          LDA B   NB
0209  011F 0D                 SEC
0210  0120 09         SUB20   DEX                     ADD 10'S COMPLEMENT
0211  0121 A6 00              LDA A   0,X             OF CREG TO AREG
0212  0123 A9 0E              ADC A   14,X
0213  0125 19                 DAA
0214  0126 A7 1C              STA A   28,X
0215  0128 5A                 DEC B
0216  0129 26 F5              BNE     SUB20
0217                   *
0218  012B 25 2D              BCS     SUB40           WAS A LARGER THAN C ?
0219                   *
0220  012D CE 0090 R          LDX     #EREG-1         NO, C WAS LARGER THAN A
0221  0130 F6 0065 R          LDA B   NB
0222  0133 08         SUB30   INX                     FIND 9'S COMPLEMENT
0223  0134 86 99              LDA A   #$99            OF ANSWER
0224  0136 A0 00              SUB A   0,X
0225  0138 A7 00              STA A   0,X
0226  013A 5A                 DEC B
0227  013B 26 F6              BNE     SUB30
0228                   *
0229  013D FE 006F R          LDX     EMOD
0230  0140 F6 0065 R          LDA B   NB
0231  0143 0D                 SEC
0232  0144 09         SUB35   DEX                     ADD 1 TO 9'S COMPLEMENT
0233  0145 86 00              LDA A   #0              TO FORM 10'S COMPLEMENT
0234  0147 A9 07              ADC A   7,X             ANSWER IN MAGNITUDE FORM
0235  0149 19                 DAA
0236  014A A7 07              STA A   7,X
0237  014C 5A                 DEC B
0238  014D 26 F5              BNE     SUB35
0239                   *
0240  014F 86 80              LDA A   #$80            SET SIGN FLAG
0241  0151 A7 00              STA A   0,X
0242  0153 4F                 CLR A
0243  0154 F6 0065 R          LDA B   NB
0244  0157 5A                 DEC B
0245  0158 20 0A              BRA     SUB50
```

```
0246                   *
0247  015A CE 008A R SUB40      LDX     #DREG           YES, A WAS LARGER THAN C
0248  015D F6 0065 R          LDA B   NB
0249  0160 5A                 DEC B
0250  0161 4F                 CLR A
0251  0162 A7 00              STA A   0,X
0252  0164 08         SUB50   INX                     CLEAR D REGISTER
0253  0165 A7 00              STA A   0,X
0254  0167 5A                 DEC B
0255  0168 26 FA              BNE     SUB50
0256                   *
0257  016A 39                 RTS
0259                   * THIS SUBROUTINE MULTIPLIES TWO UNSIGNED
0260                   * INTEGER NUMBERS
0261                   *   1ST NUMBER IN A REGISTER
0262                   *   2ND NUMBER IN C REGISTER
0263                   * THE ANSWER IS IN THE E REGISTER
0264                   * IF ANY OVERFLOW OCCURS WHEN MULTIPLYING
0265                   * IT WILL BE LOCATED IN THE D REGISTER
0266                   *
0267  016B CE 0091 R MUL      LDX     #EREG           CLEAR ANSWER LOCATION---
0268  016E C6 0E              LDA B   #14             D REGISTER AND E REGISTER
0269  0170 4F                 CLR A
0270  0171 09         MUL05   DEX
0271  0172 A7 07              STA A   7,X
0272  0174 5A                 DEC B
0273  0175 26 FA              BNE     MUL05           CLEAR FINISHED?
0274                   *
0275  0177 C6 07              LDA B   #7              YES, CLEAR B REGISTER
0276  0179 09         MUL10   DEX
0277  017A A7 00              STA A   0,X
0278  017C 5A                 DEC B
0279  017D 26 FA              BNE     MUL10           CLEAR FINISHED?
0280                   *
0281  017F CE 007C R          LDX     #BREG
0282                   *
0283  0182 09         MUL15   DEX                     YES, MULTIPLICATION LOOP
0284  0183 8C 0074 R          CPX     #AREG-1
0285  0186 26 01              BNE     MUL20           LOOP FINISHED?
0286                   *
0287  0188 39                 RTS
0288                   *
0289  0189 FF 009A R MUL20     STX     ADRSAV          CHECK RIGHT DIGIT
0290  018C E6 00              LDA B   0,X
0291  018E C4 0F              AND B   #$0F
0292  0190 27 05              BEQ     MUL30           RIGHT DIGIT CLEAR?
0293                   *
0294  0192 8D 18      MUL25   BSR     MUL45           NO, ADD B,C, REG TO D, E REG
0295                   *
0296  0194 5A                 DEC B
0297  0195 26 FB              BNE     MUL25           IS ADDITION FINISHED?
0298                   *
0299  0197 8D 25      MUL30   BSR     MUL55           YES, MOVE B,C REG 4 BITS LEFT
0300                   *
0301  0199 E6 00              LDA B   0,X             CHECK LEFT DIGIT
0302  019B C4 F0              AND B   #$F0
0303  019D 27 09              BEQ     MUL40           LEFT DIGIT CLEAR?
0304                   *
0305  019F 54                 LSR B                   NO, SHIFT COUNTER 4 BITS
0306  01A0 54                 LSR B
0307  01A1 54                 LSR B
```

```
01A2 54                 LSR B
01A3 8D 07      MUL35   BSR     MUL45       ADD B,C REG TO D,E REG
                *
01A5 5A                 DEC B
01A6 26 FB              BNE     MUL35       IS ADDITION FINISHED
                *
01A8 8D 14      MUL40   BSR     MUL55       YES, SHIFT B,C REG 4 BITS LEFT
                *
01AA 20 D6              BRA     MUL15       GO TO TOP OF SUBROUTINE
                *
01AC 0C         MUL45   CLC                 ADDITION SUBROUTINE
01AD A6 0E      MUL50   LDA A   14,X        ADD B,C REG TO D,E REG
01AF A9 1C              ADC A   28,X
01B1 19                 DAA
01B2 A7 1C              STA A   28,X
01B4 09                 DEX
01B5 8C 006D R          CPX     #AREG-8
01B8 26 F3              BNE     MUL50       ADDITION FINISHED?
                *
01BA FE 009A R          LDX     ADRSAV      YES
01BD 39                 RTS
                *
01BE 86 04      MUL55   LDA A   #$04        4 BIT SHIFT SUBROUTINE
01C0 C6 0E      MUL60   LDA B   #14         SHIFT B,C REG ONE BIT
01C2 CE 008A R          LDX     #DREG       TO THE LEFT
01C5 0C                 CLC
01C6 09         MUL65   DEX
01C7 69 00              ROL     0,X
01C9 5A                 DEC B
01CA 26 FA              BNE     MUL65       BIT SHIFTING FINISHED?
                *
01CC 4A                 DEC A               YES
01CD 26 F1              BNE     MUL60       4 BIT SHIFTING FINISHED?
                *
01CF FE 009A R          LDX     ADRSAV      YES
01D2 39                 RTS
                * THIS SUBROUTINE ADDS TWO SIGNED FLOATING
                * POINT NUMBERS
                *   1ST NUMBER IN THE A REGISTER
                *   2ND NUMBER IN THE C REGISTER
                * (THE EXPONENT IS PLACED IN THE FIRST TWO
                * BYTES OF EACH REGISTER FOLLOWED BY THE
                * DEFINED NUMBER OF BYTES OF THE NUMBER)
                * THE ANSWER IS IN THE D REGISTER
                * IF THE TWO NUMBERS HAVE THE SAME SIGN THEY
                * WILL BE ADDED.  IF THE TWO NUMBERS HAVE OPPOSITE
                * SIGNS THE NUMBERS WILL BE SUBTRACTED.
                *
01D3 CE 0073 R  FADD    LDX     #AREG-2     POSITION A REGISTER
01D6 F6 0065 R          LDA B   NB
01D9 5C                 INC B
01DA 5C                 INC B
01DB A6 02      FADD01  LDA A   2,X
01DD A7 00              STA A   0,X
01DF 08                 INX
01E0 5A                 DEC B
01E1 26 F8              BNE     FADD01
                *
01E3 FE 0083 R          LDX     CREG        SAVE EXPONENT OF C REGISTER
01E6 FF 0071 R          STX     EXP2
01E9 CE 0083 R          LDX     #CREG       POSITION C REGISTER
01EC F6 0065 R          LDA B   NB
01EF A6 02      FADD02  LDA A   2,X
01F1 A7 00              STA A   0,X
01F3 08                 INX
01F4 5A                 DEC B
01F5 26 F8              BNE     FADD02
                *
01F7 CE 0075 R          LDX     #AREG       SAVE A REGISTER
01FA F6 0065 R          LDA B   NB
01FD A6 00      FADD03  LDA A   0,X
01FF A7 07              STA A   7,X
0201 08                 INX
0202 5A                 DEC B
0203 26 F8              BNE     FADD03
                *
0205 B6 0073 R          LDA A   EXP1
0208 84 20              AND A   #$20
020A 27 03              BEQ     FADD04      IS AREG = 0
                *
020C 7E 02BC R          JMP     FADD20      YES, MOVE CREG TO ANS  LOC
                *
020F B6 0071 R  FADD04  LDA A   EXP2        NO
0212 84 20              AND A   #$20
0214 27 03              BEQ     FADD06      IS CREG = 0
                *
0216 7E 02AA R          JMP     FADD16      YES, MOVE AREG TO ANS  LOC
                *
0219 B6 0073 R  FADD06  LDA A   EXP1        NO, FIND SIGNS OF EXPONENTS
021C 84 40              AND A   #$40        DETERMINE THE OFFSET
021E B7 009E R          STA A   COUNTR      OF THE NUMBERS.
0221 F6 0071 R          LDA B   EXP2
0224 C4 40              AND B   #$40
0226 11                 CBA
0227 27 21              BEQ     FADD08      EXPONENT SIGNS ?
                *
0229 B6 0071 R          LDA A   EXP2        --DIFFERENT--
022C 84 1F              AND A   #$1F        ADD EXPONENTS
022E B7 009C R          STA A   EXPSAV      TO DETERMINE OFFSET
0231 F6 0073 R          LDA B   EXP1
0234 C4 1F              AND B   #$1F
0236 B6 0072 R          LDA A   EXP2+1
0239 BB 0074 R          ADD A   EXP1+1
023C 19                 DAA
023D B7 009D R          STA A   EXPSAV+1
0240 17                 TBA
0241 B9 009C R          ADC A   EXPSAV
0244 19                 DAA
0245 B7 009C R          STA A   EXPSAV
0248 20 4E              BRA     FADD12      EXPSAV&+1 CONTAIN OFFSET
                *
024A 7F 009E R  FADD08  CLR     COUNTR      --SAME EXPONENT SIGNS--
024D B6 0073 R          LDA A   EXP1        FIND DIFFERENCE OF EXPONENTS
0250 84 1F              AND A   #$1F        SUBTRACT EXP OF CREG
0252 B7 009C R          STA A   EXPSAV      FROM EXP OF AREG
0255 86 99              LDA A   #$99        (ADD 10'S COMPLEMENT OF CREG)
0257 F6 0071 R          LDA B   EXP2
025A C4 1F              AND B   #$1F
025C 10                 SBA
025D 16                 TAB
025E 86 99              LDA A   #$99
```

```
0260 B0 0072 R        SUB A  EXP2+1
0263 0D               SEC
0264 B9 0071 R        ADC A  EXP1+1
0267 19               DAA
0268 B7 009D R        STA A  EXPSAV+1
026B 17               TBA
026C B9 009C R        ADC A  EXPSAV
026F 19               DAA
0270 B7 009C R        STA A  EXPSAV
0273 25 19            BCS    FADD10    WAS AREG EXPONENT LARGER 2
                *
0275 86 99            LDA A  #$99      NO, SUBTRACT EXP OF AREG
0277 7C 009E R        INC    COUNTR    FROM EXP OF CREG
027A 16               TAB              (10'S COMPLEMENT OF EXP)
027B F0 009C R        SUB B  EXPSAV
027E B0 009D R        SUB A  EXPSAV+1
0281 8B 01            ADD A  #$01
0283 19               DAA
0284 B7 009D R        STA A  EXPSAV+1
0287 17               TBA
0288 89 00            ADC A  #$00
028A 19               DAA
028B B7 009C R        STA A  EXPSAV    OFFSET IN EXPSAV&+1
                *
028E B6 0073 R FADD10 LDA A  EXP1      CHANGE LARGER NUMBER
0291 84 40            AND A  #$40      POINTER IS NEGATIVE
0293 27 03            BEQ    FADD12    EXPONENTS
                *
0295 7A 009E R        DEC    COUNTR
0298 B6 009C R FADD12 LDA A  EXPSAV    CHECK FOR ADJUSTMENT RANGE
029B 26 08            BNE    FADD14
                *
029D B6 009D R        LDA A  EXPSAV+1
02A0 B1 0066 R        CMP A  ND        ARE NUMBERS IN
02A3 2D 2E            BLT    FADD26    RANGE OF ADJUSTMENT?
                *
02A5 B6 009E R FADD14 LDA A  COUNTR    NO. MOVE NUMBER TO ANS. LOC
02A8 26 12            BNE    FADD20    WHICH NUMBER IS LARGER ?
                *
02AA CE 0073 R FADD16 LDX    #AREG-2   A REGISTER
02AD F6 0065 R        LDA B  NB
02B0 5C               INC B
02B1 5C               INC B
02B2 A6 00     FADD18 LDA A  0,X       MOVE TO ANSWER LOCATION
02B4 A7 17            STA A  23,X
02B6 08               INX
02B7 5A               DEC B
02B8 26 F8            BNE    FADD18
                *
02BA 20 14            BRA    FADD24    GO TO END
                *
02BC FE 0071 R FADD20 LDX    EXP2      C REGISTER
02BF FF 008A R        STX    DREG
02C2 CE 0083 R        LDX    #CREG
02C5 F6 0065 R        LDA B  NB
02C8 A6 00     FADD22 LDA A  0,X       MOVE TO ANSWER LOCATION
02CA A7 09            STA A  9,X
02CC 08               INX
02CD 5A               DEC B
02CE 26 F8            BNE    FADD22
                *
02D0 7E 03EA R FADD24 JMP    FADD72    GO TO END
                *
02D3 CE 0075 R FADD26 LDX    #AREG     ADJUST SMALLER NUMBER
02D6 B6 009E R        LDA A  COUNTR
02D9 26 03            BNE    FADD30    A REGISTER LARGER ?
                *
02DB CE 0083 R        LDX    #CREG     YES, ADJUST C REGISTER
02DE FF 009A R FADD30 STX    ADRSAV    ADJUSTING INDEX STORED
02E1 B6 009D R        LDA A  EXPSAV+1
02E4 44               LSR A
02E5 24 15            BCC    FADD36    ODD NUMBER OF SHIFTS?
                *
02E7 86 04            LDA A  #$04      YES
02E9 F6 0065 R FADD32 LDA B  NB        SHIFT NUMBER RIGHT 4 BITS
02EC FE 009A R        LDX    ADRSAV
02EF 0C               CLC
02F0 66 00     FADD34 ROR    0,X
02F2 08               INX
02F3 5A               DEC B
02F4 26 FA            BNE    FADD34
                *
02F6 4A               DEC A
02F7 26 F0            BNE    FADD32
                *
02F9 7A 009D R        DEC    EXPSAV+1
02FC B6 009D R FADD36 LDA A  EXPSAV+1
02FF 27 1A            BEQ    FADD42
                *
0301 FE 009A R FADD38 LDX    ADRSAV    NO. EVEN NUMBER OF DIGIT
0304 C6 06            LDA B  #6        SHIFTING (BYTE SHIFTING)
0306 09        FADD40 DEX
0307 A6 06            LDA A  6,X
0309 A7 07            STA A  7,X
030B 5A               DEC B
030C 26 F8            BNE    FADD40
                *
030E 6F 06            CLR    6,X
0310 86 98            LDA A  #$98      SUBTRACT 2 FROM SHIFT COUNT
0312 BB 009D R        ADD A  EXPSAV+1
0315 19               DAA
0316 B7 009D R        STA A  EXPSAV+1
0319 26 E6            BNE    FADD38    FINISHED SHIFTING?
                *
031B CE 0071 R FADD42 LDX    #EXP2     YES
031E B6 009E R        LDA A  COUNTR    USE EXPONENT OF LARGER
0321 26 03            BNE    FADD44    NUMBER AND SIGN OF NUMBER
                *
0323 CE 0073 R        LDX    #EXP1     IN A THE REGISTER
0326 A6 00     FADD44 LDA A  0,X       IF THE NUMBERS HAVE THE SAME
0328 84 5F            AND A  #$5F      SIGN THE TWO NUMBERS WILL BE
032A F6 0073 R        LDA B  EXP1      ADDED  IF THE TWO NUMBERS
032D C4 80            AND B  #$80      HAVE OPPOSITE SIGNS THE
032F 1B               ABA              NUMBERS WILL BE SUBTRACTED
0330 B7 009C R        STA A  EXPSAV    IF CREG IS LARGER A FLAG
0333 A6 01            LDA A  1,X       AT LOCATION DREG
0335 B7 009D R        STA A  EXPSAV+1  WILL BE SET (SUBTRACTION)
                *
0338 B6 0073 R        LDA A  EXP1      IF THE SIGNS OF THE
033B B8 0071 R        EOR A  EXP2      NUMBERS ARE SAME--ADD
033E 84 80            AND A  #$80      DIFFERENT--SUBTRACT
0340 27 29            BEQ    FADD54    GO TO SUBROUTINES
```

```
0553                    *
0554   0342 BD 0109 R          JSR    SUB       *** SUBTRACTION ***
0555                    *
0556   0345 B6 008A R          LDA A  DREG      GET EXPONENT
0557   0348 B8 009C R          EOR A  EXPSAV    INCLUDE SIGN CHANGE (IF ANY)
0558   034B B7 008A R          STA A  DREG
0559   034E B6 009D R          LDA A  EXPSAV+1
0560   0351 B7 008B R          STA A  DREG+1
0561   0354 CE 008A R          LDX    #DREG
0562   0357 F6 0065 R          LDA B  NB
0563   035A A6 07      FADD50  LDA A  7,X       MOVE TO ANSWER LOCATION
0564   035C A7 02              STA A  2,X
0565   035E 08                 INX
0566   035F 5A                 DEC B
0567   0360 26 F8              BNE    FADD50
0568                    *
0569   0362 CE 008A R          LDX    #DREG
0570   0365 BD 04CF R          JSR    PAK       PACK ANSWER
0571                    *
0572   0368 7E 03EA R          JMP    FADD72    GO TO END
0573                    *
0574   036B BD 00E3 R FADD54   JSR    ADD       *** ADDITION ***
0575                    *
0576   036E B6 0090 R          LDA A  EREG-1
0577   0371 26 0E              BNE    FADD60    DID OPERATION OVERFLOW?
0578                    *
0579   0373 FE 009C R          LDX    EXPSAV    NO OVERFLOW
0580   0376 FF 008A R          STX    DREG
0581   0379 CE 008A R          LDX    #DREG
0582   037C F6 0065 R          LDA B  NB
0583   037F 20 D9              BRA    FADD50
0584                    *
0585   0381 B6 009C R FADD60   LDA A  EXPSAV    YES, OVERFLOW
0586   0384 84 40              AND A  #$40
0587   0386 26 29              BNE    FADD64    SIGN OF EXPONENT?
0588                    *
0589   0388 B6 009C R          LDA A  EXPSAV    POSITIVE
0590   038B 16                 TAB
0591   038C 84 1F              AND A  #$1F
0592   038E B7 008A R          STA A  DREG
0593   0391 C4 C0              AND B  #$C0
0594   0393 F7 009C R          STA B  EXPSAV
0595   0396 B6 009D R          LDA A  EXPSAV+1
0596   0399 8B 02              ADD A  #$02      ADD 2 TO THE EXPONENT
0597   039B 19                 DAA
0598   039C B7 008B R          STA A  DREG+1
0599   039F 86 00              LDA A  #$00
0600   03A1 B9 008A R          ADC A  DREG
0601   03A4 19                 DAA
0602   03A5 16                 TAB
0603   03A6 84 1F              AND A  #$1F      KEEP OVERFLOW OF EXPONENT
0604   03A8 C4 20              AND B  #$20      IF PRESENT
0605   03AA 54                 LSR B
0606   03AB 1B                 ABA
0607   03AC F6 009C R          LDA B  EXPSAV
0608   03AF 20 27              BRA    FADD68
0609                    *
0610   03B1 B6 009C R FADD64   LDA A  EXPSAV    NEGATIVE EXPONENT
0611   03B4 16                 TAB
0612   03B5 84 1F              AND A  #$1F
0613   03B7 B7 008A R          STA A  DREG
0614   03BA C4 C0              AND B  #$C0
0615   03BC B6 009D R          LDA A  EXPSAV+1
0616   03BF 8B 98              ADD A  #$98      SUBTRACT 2 FROM EXP
0617   03C1 19                 DAA
0618   03C2 B7 008B R          STA A  DREG+1
0619   03C5 86 99              LDA A  #$99
0620   03C7 B9 008A R          ADC A  DREG
0621   03CA 19                 DAA
0622   03CB 25 0B              BCS    FADD68    WAS EXP = -1 OR -0?
0623                    *
0624   03CD 86 02              LDA A  #$02      YES SUBTRACT ORIGINAL
0625   03CF B0 009D R          SUB A  EXPSAV+1  NUMBER FROM 2 TO OBTAIN
0626   03D2 B7 008B R          STA A  DREG+1    CORRECT NUMBER
0627   03D5 4F                 CLR A
0628   03D6 C4 80              AND B  #$80      CLEAR SIGN BIT
0629                    *
0630   03D8 1B         FADD68  ABA
0631   03D9 B7 008A R          STA A  DREG
0632   03DC CE 008A R          LDX    #DREG
0633   03DF F6 0065 R          LDA B  NB
0634   03E2 A6 06      FADD70  LDA A  6,X
0635   03E4 A7 02              STA A  2,X
0636   03E6 08                 INX
0637   03E7 5A                 DEC B
0638   03E8 26 F8              BNE    FADD70
0639                    *
0640   03EA FE 0073 R FADD72   LDX    EXP1      RESTORE A REGISTER
0641   03ED FF 0075 R          STX    AREG
0642   03F0 CE 0075 R          LDX    #AREG
0643   03F3 F6 0065 R          LDA B  NB
0644   03F6 A6 07      FADD74  LDA A  7,X
0645   03F8 A7 02              STA A  2,X
0646   03FA 08                 INX
0647   03FB 5A                 DEC B
0648   03FC 26 F8              BNE    FADD74
0649                    *
0650   03FE CE 008A R          LDX    #DREG
0651   0401 BD 04CF R          JSR    PAK       PACK ANSWER
0652                    *
0653   0404 39                 RTS
0655                    *
0656                    *  THIS SUBROUTINE SUBTRACTS TWO SIGNED FLOATING
0657                    *  POINT NUMBERS.
0658                    *    1ST NUMBER IN THE A REGISTER
0659                    *    2ND NUMBER IN THE C REGISTER
0660                    *  (THE EXPONENT IS PLACED IN THE FIRST TWO
0661                    *  BYTES OF EACH REGISTER FOLLOWED BY THE
0662                    *  DEFINED NUMBER OF BYTES OF THE NUMBER)
0663                    *  THE ANSWER IS IN THE D REGISTER
0664                    *  IF THE TWO NUMBERS HAVE THE SAME SIGN THEY
0665                    *  WILL BE SUBTRACTED   IF THE TWO NUMBERS HAVE
0666                    *  OPPOSITE SIGNS THE NUMBERS WILL BE ADDED
0667                    *
0668   0405 86 80      FSUB    LDA A  #$80      CHANGE SIGN OF CREG
0669   0407 B8 0083 R          FOR A  CREG      AND JUMP TO ADDITION
0670   040A B7 0083 R          STA A  CREG      SUBROUTINE
0671   040D 7E 01D3 R          JMP    FADD
0673                    *
0674                    *  THIS SUBROUTINE MULTIPLIES TWO SIGNED
0675                    *  FLOATING POINT NUMBERS
0676                    *    1ST NUMBER IN THE A REGISTER
```

```
                        *    2ND NUMBER IN THE C REGISTER
                        *  (THE EXPONENT IS PLACED IN THE FIRST TWO
                        *  BYTES OF EACH REGISTER FOLLOWED BY THE
                        *  DEFINED NUMBER OF BYTES OF THE NUMBER)
                        *  THE ANSWER IS IN THE D REGISTER
                        *
0410 FE 0083 R  FMUL    LDX     CREG
0413 FF 0071 R          STX     EXP2        SAVE EXP OF REGISTER
0416 CE 0073 R          LDX     #AREG-2     MOVE A REGISTER INTO POSITION
0419 F6 0065 R          LDA B   NB
041C 5C                 INC B
041D 5C                 INC B
041E A6 02      FMUL10  LDA A   2,X
0420 A7 00              STA A   0,X
0422 08                 INX
0423 5A                 DEC B
0424 26 F8              BNE     FMUL10
                        *
0426 CE 0083 R          LDX     #CREG
0429 F6 0065 R          LDA B   NB          MOVE C REGISTER INTO POSITION
042C A6 02      FMUL20  LDA A   2,X
042E A7 00              STA A   0,X
0430 08                 INX
0431 5A                 DEC B
0432 26 F8              BNE     FMUL20
                        *
0434 BD 016B R          JSR     MUL         *** MULTIPLICATION ***
                        *
0437 CE 008A R          LDX     #DREG       MOVE ANSWER (BOTH BYTES)
043A C6 0E              LDA B   #14         RIGHT TWO BYTES
043C 09         FMUL30  DEX
043D A6 0E              LDA A   14,X
043F A7 10              STA A   16,X
0441 5A                 DEC B
0442 26 F8              BNE     FMUL30
                        *
0444 FE 0069 R          LDX     BMOD
0447 F6 0065 R          LDA B   NB          RESTORE A REGISTER
044A 09         FMUL40  DEX
044B A6 00              LDA A   0,X
044D A7 02              STA A   2,X
044F 5A                 DEC B
0450 26 F8              BNE     FMUL40
                        *
0452 B6 0073 R          LDA A   EXP1
0455 B8 0071 R          EOR A   EXP2
0458 84 40              AND A   #$40
045A 27 3E              BEQ     FMUL50      SIGNS OF EXPONENTS?
                        *
045C F6 0071 R          LDA B   EXP2        OPPOSITE SIGNS--
045F C4 1F              AND B   #$1F        SUBTRACT CREG EXP
0461 86 99              LDA A   #$99        FROM AREG EXP
0463 10                 SBA
0464 B7 008A R          STA A   DREG
0467 86 99              LDA A   #$99
0469 B0 0072 R          SUB A   EXP2+1
046C 0D                 SEC
046D B9 0074 R          ADC A   EXP1+1
0470 19                 DAA
0471 B7 008B R          STA A   DREG+1
0474 B6 0073 R          LDA A   EXP1
0477 84 1F              AND A   #$1F
0479 B9 008A R          ADC A   DREG
047C 19                 DAA
047D 25 37              BCS     FMUL60      WAS AREG EXP LARGER?
                        *
047F 86 99              LDA A   #$99        NO, CREG EXP LARGER
0481 16                 TAB                 FIND 10'S COMPLEMENT OF EXP
0482 F0 008A R          SUB B   DREG
0485 B0 008B R          SUB A   DREG+1
0488 8B 01              ADD A   #$01
048A 19                 DAA
048B B7 008B R          STA A   DREG+1
048E 17                 TBA
048F B9 00              ADC A   #$00
0491 19                 DAA
0492 F6 0071 R          LDA B   EXP2
0495 C4 40              AND B   #$40        SIGN OF EXP2(CREG)
0497 1B                 ABA
0498 20 22              BRA     FMUL70      BRANCH TO FINISH
                        *
049A F6 0071 R  FMUL50  LDA B   EXP2        SAME SIGNS
049D C4 1F              AND B   #$1F        ADD EXPONENTS
049F B7 009C R          STA A   EXPSAV
04A2 B6 0072 R          LDA A   EXP2+1
04A5 0C                 CLC
04A6 B9 0074 R          ADC A   EXP1+1
04A9 19                 DAA
04AA B7 008B R          STA A   DREG+1
04AD B6 0073 R          LDA A   EXP1
04B0 84 1F              AND A   #$1F
04B2 B9 009C R          ADC A   EXPSAV
04B5 19                 DAA
04B6 F6 0073 R  FMUL60  LDA B   EXP1        SIGN OF EXP1(AREG)
04B9 C4 40              AND B   #$40
04BB 1B                 ABA
04BC F6 0073 R  FMUL70  LDA B   EXP1        GET SIGN OF ANSWER
04BF F8 0071 R          EOR B   EXP2        MULTIPLY SIGNS
04C2 C4 80              AND B   #$80        SIGN OF ANSWER INSERTED
04C4 1B                 ABA
04C5 B7 008A R          STA A   DREG
04C8 CE 008A R          LDX     #DREG
04CB BD 04CF R          JSR     PAK         PACK ANSWER
                        *
04CE 39                 RTS
                        *
                        *  THIS SUBROUTINE PACKS THE FLOATING POINTING
                        *  NUMBER TO THE LEFT DELETING THE LEADING ZEROS
                        *  AND ADJUSTING THE EXPONENT ACCORDINGLY
                        *  ENTRY INTO THIS SUBROUTINE REQUIRES ONLY THE
                        *  INDEX REGISTER BE INITIALIZED   THE INDEX
                        *  REGISTER IS TO CONTAIN THE ADDRESS OF THE FIRST
                        *  BYTE OF THE FLOATING POINT NUMBER (THE A
                        *  PORTION OF THE EXPONENT AS DESCRIBED IN
                        *  NUMBER DEFINITION)
                        *
04CF FF 009A R  PAK     STX     ADRSAV      SAVE ADDRESS POINTER
04D2 A6 00              LDA A   0,X         SAVE M  S   DIGIT AND
04D4 84 1F              AND A   #$1F        OVERFLOW OF EXP
04D6 B7 009C R          STA A   EXPSAV
04D9 7F 009E R          CLR     COUNTR
                        *
```

```
04DC FE 009A R PAK05  LDX    ADRSAV
04DF A6 02            LDA A  2,X
04E1 26 26            BNE    PAK15      DOUBLE DIGIT
                 *
04E3 F6 0065 R        LDA B  NB         -YES
04E6 5A               DEC B
04E7 A6 03      PAK10 LDA A  3,X        MOVE NUMBER LEFT TWO DIGITS
04E9 A7 02            STA A  2,X        (1 BYTE)
04EB 08               INX
04EC 5A               DEC B
04ED 26 F8            BNE    PAK10
                 *
04EF 6F 02            CLR    2,X        CLEAR LAST BYTE
04F1 B6 009E R        LDA A  COUNTR     ADD TWO TO COUNTER
04F4 8B 02            ADD A  #$02
04F6 19               DAA
04F7 B7 009E R        STA A  COUNTR
04FA B1 0066 R        CMP A  ND
04FD 2D DD            BLT    PAK05      NUMBER OF DIGITS EXCEEDED?
                 *
04FF 86 20            LDA A  #$20       -YES, SET ERROR FLAG
0501 FE 009A R        LDX    ADRSAV
0504 A7 00            STA A  0,X
0506 6F 01            CLR    1,X
0508 39               RTS
                 *
0509 F6 009E R PAK15  LDA B  COUNTR
050C 27 56            BEQ    PAK30      ADJUST EXPONENT?
                 *
050E A6 00            LDA A  0,X        -YES
0510 84 40            AND A  #$40
0512 27 18            BEQ    PAK20      SIGN OF EXPONENT--
                 *
0514 A6 01            LDA A  1,X        NEGATIVE
0516 1B               ABA               ADD COUNTER TO EXPONENT
0517 19               DAA
0518 A7 01            STA A  1,X
051A 86 00            LDA A #0
051C B9 009C R        ADC A  EXPSAV
051F 19               DAA
0520 16               TAB
0521 84 1F            AND A  #$1F
0523 C4 20            AND B  #$20       OVERFLOW MARKER
0525 54               LSR B
0526 1B               ABA
0527 B7 009C R        STA A  EXPSAV
052A 20 38            BRA    PAK30
                 *
052C 86 99      PAK20 LDA A #$99        POSITIVE EXPONENT
052E 10               SBA               SUBTRACT COUNTER FROM EXP
052F 0D               SEC
0530 A9 01            ADC A  1,X
0532 19               DAA
0533 16               TAB
0534 86 99            LDA A  #$99
0536 B9 009C R        ADC A  EXPSAV
0539 19               DAA
053A 24 0A            BCC    PAK25      NUMBER ERROR?
                 *
053C B7 009C R        STA A  EXPSAV     NO,  SAVE EXPONENT
053F E7 01            STA B  1,X
0541 F6 009E R        LDA B  COUNTR
0544 20 1E            BRA    PAK30
                 *
0546 86 99      PAK25 LDA A  #$99       SUBTRACT EXPONENT FROM
0548 B0 009C R        SUB A  EXPSAV     COUNTER
054B 16               TAB
054C 86 99            LDA A #$99
054E A0 01            SUB A 1,X
0550 0D               SEC
0551 B9 009E R        ADC A COUNTR
0554 19               DAA
0555 A7 01            STA A  1,X
0557 17               TBA
0558 89 00            ADC A  #$00
055A 19               DAA
055B B7 009C R        STA A  EXPSAV     SAVE
055E A6 00            LDA A  0,X        CHANGE EXPONENTS'S SIGN
0560 8A 40            ORA A  #$40
0562 A7 00            STA A  0,X
                 *
0564 A6 02      PAK30 LDA A  2,X
0566 84 F0            AND A  #$F0
0568 26 28            BNE    PAK45      FIRST DIGIT ZERO?
                 *
056A 86 01            LDA A  #$01       -YES
056C B7 009E R        STA A  COUNTR
056F 86 04            LDA A  #$04       ROTATE NUMBER 1 DIGIT
0571 C6 07      PAK35 LDA B  #7
0573 FE 009A R        LDX    ADRSAV
0576 F0 0065 R        SUB B  NB         FIND OFFSET
0579 27 04            BEQ    PAK37      NO OFFSET?
                 *
057B 09         PAK36 DEX               MOVE POINTER
057C 5A               DEC B
057D 26 FC            BNE    PAK36      LOOP UNTIL DONE
                 *
057F F6 0065 R PAK37  LDA B  NB          GET BYTE COUNT
0582 0C               CLC
0583 09         PAK40 DEX
0584 69 09            ROL    9,X
0586 5A               DEC B
0587 26 FA            BNE    PAK40
                 *
0589 4A               DEC A
058A 26 F5            BNE    PAK35
                 *
058C FE 009A R        LDX    ADRSAV
058F 7E 0509 R        JMP    PAK15      ADJUST EXPONENT
                 *
0592 A6 00      PAK45 LDA A  0,X        GET SIGN OF NUMBER AND EXPONENT
0594 84 C0            AND A  #$C0
0596 BA 009C R        ORA A  EXPSAV     ADD EXPONENT
0599 A7 00            STA A  0,X        SAVE EXPONENT
059B 39               RTS
                 *  THIS SUBROUTINE DIVIDES TWO UNSIGNED
                 *  INTEGER NUMBERS
                 *    1ST NUMBER IN THE C REGISTER
                 *    2ND NUMBER IN THE A REGISTER
                 *  THE ANSWER (C/A) IS IN THE E REGISTER.
                 *  THE NUMBER IN THE A REGISTER (DIVISOR)
                 *  IS UNCHANGED
```

```
                        *
059C CE 0091 R  DIV     LDX     #EREG
059F C6 0E              LDA B   #14
05A1 4F                 CLR A
05A2 09         DIV04   DEX                     CLEAR D, E REGISTERS
05A3 A7 07              STA A   7,X
05A5 5A                 DEC B
05A6 26 FA              BNE     DIV04
                        *
05A8 C6 07              LDA B   #7
05AA 09         DIV08   DEX                     CLEAR B REGISTER
05AB A7 00              STA A   0,X
05AD 5A                 DEC B
05AE 26 FA              BNE     DIV08
                        *
05B0 7F 0074 R          CLR     AREG-1
05B3 86 04              LDA A   #$04
05B5 F6 0065 R  DIV12   LDA B   NB              SHIFT DIVISOR LEFT
05B8 5C                 INC B
05B9 FE 0069 R          LDX     BMOD            4 BITS (1 DIGIT)
05BC 0C                 CLC
05BD 09         DIV16   DEX
05BE 69 00              ROL     0,X
05C0 5A                 DEC B
05C1 26 FA              BNE     DIV16
                        *
05C3 4A                 DEC A
05C4 26 EF              BNE     DIV12
                        *
05C6 C6 07              LDA B   #7              INITIALIZE SHIFT
05C8 F0 0065 R          SUB B   NB              COUNT=2*(14-(7-NB))
05CB 86 0E              LDA A   #14
05CD 10                 SBA
05CF 48                 ASL A                   MULTIPLY BY 2
05CF B7 009E R          STA A   COUNTR          COUNTER
05D2 8D 51      DIV20   BSR     DIV60           10'S COMP OF A
                        *
05D4 8D 74      DIV24   BSR     DIV72           ADD A & B REGISTERS
                        *
05D6 24 07              BCC     DIV28           TOO MUCH?
                        *
05D8 FE 006F R          LDX     EMOD
05DB 6C 06              INC     6,X             NO, ADD 1 TO NUMBER
05DD 20 F5              BRA     DIV24
                        *
05DF 8D 44      DIV28   BSR     DIV60           10'S COMP OF A
                        *
05E1 7A 009E R          DEC     COUNTR
05E4 27 15              BEQ     DIV40           OPERATION FINISHED?
                        *
05E6 8D 62              BSR     DIV72           NO, ADD A & B REGISTERS
                        *
05E8 86 04              LDA A   #$04            SHIFT B,C,D,E
05EA C6 1C      DIV32   LDA B   #28             REGISTERS LEFT
05EC CE 0091 R          LDX     #EREG           4 BITS (1 DIGIT)
05EF 0C                 CLC
05F0 09         DIV36   DEX
05F1 69 07              ROL     7,X
05F3 5A                 DEC B
05F4 26 FA              BNE     DIV36
                        *
05F6 4A                 DEC A
05F7 26 F1              BNE     DIV32
                        *
05F9 20 D7              BRA     DIV20
                        *
05FB 86 04      DIV40   LDA A   #$04            RESTORE A REGISTER
05FD F6 0065 R  DIV44   LDA B   NB
0600 5C                 INC B
0601 CE 0074 R          LDX     #AREG-1
0604 0C                 CLC
0605 66 00      DIV48   ROR     0,X
0607 08                 INX
0608 5A                 DEC B
0609 26 FA              BNE     DIV48
                        *
060B 4A                 DEC A
060C 26 EF              BNE     DIV44
                        *
060E C6 07      DIV52   LDA B   #7              SET UP ANSWER
0610 F7 009E R          STA B   COUNTR          FOR INTEGER OUTPUT
0613 CE 0091 R          LDX     #EREG           SHIFT D & E REGISTER
0616 09         DIV56   DEX
0617 A6 00              LDA A   0,X
0619 E6 07              LDA B   7,X
061B A7 07              STA A   7,X
061D E7 00              STA B   0,X
061F 7A 009E R          DEC     COUNTR
0622 26 F2              BNE     DIV56
                        *
0624 39                 RTS
                        *
0625 F6 0065 R  DIV60   LDA B   NB              10'S COMPLEMENT OF
0628 5C                 INC B
0629 CE 0074 R          LDX     #AREG-1         A REGISTER
062C 86 99      DIV64   LDA A   #$99            9'S COMP
062E A0 00              SUB A   0,X
0630 A7 00              STA A   0,X
0632 08                 INX
0633 5A                 DEC B
0634 26 F6              BNE     DIV64
                        *
0636 0D                 SEC                     10'S COMPLEMENT(9'S +1)
0637 F6 0065 R          LDA B   NB
063A 5C                 INC B
063B FE 0069 R          LDX     BMOD
063E 09         DIV68   DEX
063F 86 00              LDA A   #$00
0641 A9 00              ADC A   0,X
0643 19                 DAA
0644 A7 00              STA A   0,X
0646 5A                 DEC B
0647 26 F5              BNE     DIV68
                        *
0649 39                 RTS
                        *
064A F6 0065 R  DIV72   LDA B   NB              ADD A & B REGISTERS
064D 5C                 INC B
064E FE 0069 R          LDX     BMOD
0651 0C                 CLC
0652 09         DIV76   DEX
0653 A6 00              LDA A   0,X
```

```
0655 A9 08              ADC A  8,X
0657 19                 DAA
0658 A7 08              STA A  8,X
065A 5A                 DEC B
065B 26 F5              BNE    DIV76
                 *
065D 39                 RTS
                 *
                 *  THIS SUBROUTINE DIVIDES TWO SIGNED
                 *  FLOATING POINT NUMBERS
                 *    1ST NUMBER IS IN THE C REGISTER
                 *    2ND NUMBER IS IN THE A REGISTER
                 *  THE ANSWER (C/A) IS IN THE D REGISTER  THE
                 *  NUMBER IN THE A REGISTER IS RETAINED
                 *
065F B6 0075 R FDIV     LDA A  AREG
0661 84 20              AND A  #$20      CHECK A REGISTER
0663 27 03              BEQ    FDIV04    A REGISTER EQUAL TO ZERO?
                 *
0665 BD 0000 R          JSR    DIVERR    YES, ERROR  DIVISION BY 0
                 *
0668 FE 0075 R FDIV04   LDX    AREG
066B FF 009C R          STX    EXPSAV    SAVE A EXPONENT
066E FE 0083 R          LDX    CREG
0671 FF 0071 R          STX    EXP2      SAVE C EXPONENT
0674 CE 0075 R          LDX    #AREG     MOVE A REGISTER
0677 F6 0065 R          LDA B  NB        INTO POSITION
067A A6 02       FDIV09 LDA A  2,X
067C A7 00              STA A  0,X
067E 08                 INX
067F 5A                 DEC B
0680 26 F8              BNE    FDIV09
                 *
0682 CE 0083 R          LDX    #CREG
0685 F6 0065 R          LDA B  NB        MOVE C REGISTER
0688 A6 02       FDIV12 LDA A  2,X       INTO POSITION
068A A7 00              STA A  0,X
068C 08                 INX
068D 5A                 DEC B
068E 26 F8              BNE    FDIV12
                 *
0690 BD 059C R          JSR    DIV       DIVIDE NUMBERS!
                 *
0693 FE 0069 R          LDX    BMOD      RESTORE A REGISTER
0696 F6 0065 R          LDA B  NB
0699 09          FDIV16 DEX
069A A6 00              LDA A  0,X
069C A7 02              STA A  2,X
069E 5A                 DEC B
069F 26 F8              BNE    FDIV16
                 *
06A1 FE 009C R          LDX    EXPSAV    AND THE A EXPONENT
06A4 FF 0075 R          STX    AREG
06A7 FF 0073 R          STX    EXP1
06AA FE 006F R          LDX    EMOD      MOVE ANSWER INTO POSITION
06AD 09                 DEX
06AE F6 0065 R          LDA B  NB
06B1 A6 00       FDIV20 LDA A  0,X
06B3 A7 02              STA A  2,X
06B5 09                 DEX
06B6 5A                 DEC B
06B7 26 F8              BNE    FDIV20
                 *
06B9 A6 0E              LDA A  14,X
06BB 27 71              BEQ    FDIV32    ANSWER LESS THAN ONE?
                 *
06BD A7 02              STA A  2,X       NO, ADD 2 TO C EXPONENT
06BF FE 006F R          LDX    EMOD      MOVE ANSWER 1 BYTE RIGHT
06C2 F6 0065 R          LDA B  NB
06C5 A6 00       FDIV22 LDA A  0,X
06C7 A7 01              STA A  1,X
06C9 09                 DEX
06CA 5A                 DEC B
06CB 26 F8              BNE    FDIV22
                 *
06CD B6 0071 R          LDA A  EXP2
06D0 84 40              AND A  #$40
06D2 26 29              BNE    FDIV24    SIGN OF C EXPONENT?
                 *
06D4 B6 0071 R          LDA A  EXP2      POSITIVE
06D7 16                 TAB
06D8 84 1F              AND A  #$1F      (A HAS NUMBER)
06DA B7 008A R          STA A  DREG
06DD C4 C0              AND B  #$C0      (B HAS SIGNS)
06DF F7 0071 R          STA B  EXP2
06E2 B6 0072 R          LDA A  EXP2+1
06E5 8B 02              ADD A  #$02      ADD 2 TO THE C EXPONENT
06E7 19                 DAA
06E8 B7 008B R          STA A  DREG+1
06EB 86 00              LDA A  #$00
06ED B9 008A R          ADC A  DREG
06F0 19                 DAA
06F1 16                 TAB
06F2 84 1F              AND A  #$1F      KEEP OVERFLOW OF EXPONENT
06F4 C4 20              AND B  #$20      IF PRESENT
06F6 54                 LSR B
06F7 1B                 ABA
06F8 F6 0071 R          LDA B  EXP2
06FB 20 27              BRA    FDIV28
                 *
06FD B6 0071 R FDIV24   LDA A  EXP2      NEGATIVE EXPONENT
0700 16                 TAB              (SUBTRACT 2 FROM EXPONENT)
0701 84 1F              AND A  #$1F
0703 B7 008A R          STA A  DREG
0706 C4 C0              AND B  #$C0
0708 B6 0072 R          LDA A  EXP2+1
070B 8B 98              ADD A  #$98      ADD -2 TO EXPONENT
070D 19                 DAA
070E B7 008B R          STA A  DREG+1
0711 86 99              LDA A  #$99
0713 B9 008A R          ADC A  DREG
0716 19                 DAA
0717 25 0B              BCS    FDIV28    WAS EXP * -1 OR -0?
                 *
0719 86 02              LDA A  #$02      YES, SUBTRACT ORIGINAL
071B B0 0072 R          SUB A  EXP2+1    NUMBER FROM 2 TO OBTAIN
071E B7 008B R          STA A  DREG+1    CORRECT NUMBER
0721 4F                 CLR A
0722 C4 80              AND B  #$80      CLEAR SIGN BIT
0724 1B          FDIV28 ABA
0725 B7 0071 R          STA A  EXP2
0728 B6 008B R          LDA A  DREG+1
```

```
072B B7 0072 R          STA A  EXP2+1
072E B6 0073 R FDIV32   LDA A  EXP1
0731 88 40              EOR A  #$40
0733 B7 0073 R          STA A  EXP1       CHANGE SIGN OF A EXPONENT
0736 B8 0071 R          EOR A  EXP2
0739 84 40              AND A  #$40
073B 27 3E              BEQ    FDIV36     SIGNS OF EXPONENTS?
                  *
073D F6 0071 R          LDA B  EXP2       OPPOSITE SIGNS
0740 C4 1F              AND B  #$1F       SUBTRACT CREG EXP FROM
0742 86 99              LDA A  #$99       AREG EXP
0744 10                 SBA
0745 B7 008A R          STA A  DREG
0748 86 99              LDA A  #$99
074A B0 0072 R          SUB A  EXP2+1
074D 0D                 SEC
074E B9 0074 R          ADC A  EXP1+1
0751 19                 DAA
0752 B7 008B R          STA A  DREG+1
0755 B6 0073 R          LDA A  EXP1
0758 84 1F              AND A  #$1F
075A B9 008A R          ADC A  DREG
075D 19                 DAA
075E 25 3F              BCS    FDIV40     WAS AREG EXP LARGER?
                  *
0760 86 99              LDA A  #$99       NO. CREG EXP WAS LARGER
0762 16                 TAB               FIND 10'S COMPLEMENT OF EXP
0763 F0 008A R          SUB B  DREG
0766 B0 008B R          SUB A  DREG+1
0769 8B 01              ADD A  #$01
076B 19                 DAA
076C B7 008B R          STA A  DREG+1
076F 17                 TBA
0770 89 00              ADC A  #$00
0772 19                 DAA
0773 F6 0071 R          LDA B  EXP2
0776 C4 40              AND B  #$40       SIGN OF C REGISTER
0778 1B                 ABA
0779 20 29              BRA    FDIV44
                  *
077B F6 0071 R FDIV36   LDA B  EXP2       SAME SIGNS
077E C4 1F              AND B  #$1F       ADD EXPONENTS
0780 B7 009C R          STA A  EXPSAV
0783 B6 0072 R          LDA A  EXP2+1
0786 0C                 CLC
0787 B9 0074 R          ADC A  EXP1+1
078A 19                 DAA
078B B7 008B R          STA A  DREG+1
078E B6 0073 R          LDA A  EXP1
0791 84 1F              AND A  #$1F
0793 B9 009C R          ADC A  EXPSAV
0796 19                 DAA
0797 16                 TAB               RETAIN OVERFLOW, IF ANY
0798 84 1F              AND A  #$1F
079A C4 20              AND B  #$20
079C 54                 LSR B
079D 1B                 ABA
079E F6 0073 R FDIV40   LDA B  EXP1       SIGN OF A REGISTER
07A1 C4 40              AND B  #$40
07A3 1B                 ABA
07A4 F6 0073 R FDIV44   LDA B  EXP1       GET SIGN OF ANSWER
07A7 F8 0071 R          EOR B  EXP2       (MULTIPLY SIGNS)
07AA C4 80              AND B  #$80       SIGN OF ANSWER INSERTED
07AC 1B                 ABA
07AD B7 008A R          STA A  DREG
07B0 CE 008A R          LDX    #DREG
07B3 BD 04CF R          JSR    PAK        PACK ANSWER
                  *
07B6 39                 RTS
                        END
```

```
ACCUM1  0050 C
ACCUM2  0052 C
ACCUM3  0054 C
ACCUM4  0056 C
ACCUM5  0058 C
ADD     00E3 R
ADD10   00EA R
ADD20   00FD R
ADD30   0100 R
ADRFM   0063 C
ADRSAV  009A R
ADRTO   0061 C
AMOD    0067 R
AREG    0075 R
BMOD    0069 R
BREG    007C R
CMOD    006B R
COUNTR  009E R
CREG    0083 R
DFAULT  004E R
D16SET  0003 RN
D16ST2  0019 R
D16ST3  0023 R
D16ST4  0032 R
D16ST5  0036 R
DIV     059C R
DIV04   05A2 R
DIV08   05AA R
DIV12   05B5 R
DIV16   05BD R
DIV20   05D2 R
DIV24   05D4 R
DIV28   05DF R
DIV32   05EA R
DIV36   05F0 R
DIV40   05FB R
DIV44   05FD R
DIV48   0605 R
DIV52   060E R
DIV56   0616 R
DIV60   0625 R
DIV64   062C R
DIV68   063E R
DIV72   064A R
DIV76   0652 R
DIVERR  0000 RX
DMOD    006D R
DREG    008A R
EMOD    006F R
EREG    0091 R
EXP1    0073 R
EXP2    0071 R
EXPSAV  009C R
FADD    01D3 RN
FADD01  01DB R
FADD02  01EF R
FADD03  01FD R
FADD04  020F R
FADD06  0219 R
FADD08  024A R
FADD10  028E R
FADD12  0298 R
FADD14  02A5 R
FADD16  02AA R
FADD18  02B2 R
FADD20  02BC R
FADD22  02C8 R
FADD24  02D0 R
FADD26  02D3 R
FADD30  02DE R
FADD32  02E9 R
FADD34  02F0 R
FADD36  02FC R
FADD38  0301 R
FADD40  0306 R
FADD42  0318 R
FADD44  0326 R
FADD50  035A R
FADD54  0368 R
FADD60  03B1 R
FADD64  03B1 R
FADD68  03D8 R
FADD70  03E2 R
FADD72  03EA R
FADD74  03F6 R
FDIV    065E RN
FDIV04  0668 R
FDIV09  067A R
FDIV12  0688 R
FDIV16  0699 R
FDIV20  06B1 R
FDIV22  06C5 R
FDIV24  06FD R
FDIV28  0724 R
FDIV32  072F R
FDIV36  077B R
FDIV40  079E R
FDIV44  07A4 R
FFDREG  00BF RN
FLOIPT  0000 RN
FMUL    0410 RN
FMUL10  041E R
FMUL20  042C R
FMUL30  043C R
FMUL40  044A R
FMUL50  049A R
FMUL60  04B6 R
FMUL70  04BC R
FSUB    0405 RN
FTAREG  009F RN
FTCREG  00AF RN
MOVER   00CD RN
MUL     016B R
MUL05   0171 R
MUL10   0179 R
MUL15   0182 R
MUL20   0189 R
MUL25   0192 R
MUL30   0197 R
MUL35   01A3 R
MUL40   01A8 R
MUL45   01AC R
MUL50   01AD R
MUL55   01BE R
MUL60   01C0 R
MUL65   01C6 R
NB      0065 R
ND      0066 R
PAK     04FF R
PAK05   041C R
PAK10   04E7 R
PAK15   0509 R
PAK20   052C R
PAK25   0546 R
PAK30   0564 R
PAK35   0571 R
PAK36   057B R
PAK37   057F R
PAK40   0583 R
PAK45   0592 R
RUNSTK  0000 C
SUB     0109 R
SUB10   010F R
SUB20   0120 R
SUB30   0133 R
SUB35   0144 R
SUB40   015A R
SUB50   0164 R
XINC7   005D R
```

```
0001   0000 0000   N        NAM SCIFUN  (STRUBAL +)
0002                  *
0003                  * SCIENTIFIC FUNCTION PACKAGE
0004                  *
0005                  *
0006                  * COPYRIGHT 1978 BY ROBERT GRAPPEL, LEXINGTON, MASS
0007                  *                AND JACK HEMENWAY, BOSTON, MASS
0008                  *                ALL RIGHTS RESERVED
0009                  *
0010                  * SIN, COS, TAN TAKE ARGUMENTS IN RADIANS
0011                  * ASIN, ACOS, ATAN GIVE RESULT IN RADIANS
0012                  * CALL WITH ADDRESS OF ARGUMENT IN X-REGISTER
0013                  *
0014                  * LINKED TO RUNTIME PACKAGE AND FLOATING-POINT
0015                  *
0016   0000 0000   C        CMN RUNSTK,80
0017   0000 0050   C        CMN ACCUM1,2
0018   0000 0052   C        CMN ACCUM2,2
0019   0000 0054   C        CMN ACCUM3,2
0020   0000 0056   C        CMN ACCUM4,2
0021   0000 0058   C        CMN ACCUM5,9
0022   0000 0061   C        CMN ADRTO,2
0023   0000 0063   C        CMN ADRFM,2
0024                  *
0025   0000 03FE   N        ENT SIN
0026   0000 036A   N        ENT COS
0027   0000 0327   N        ENT TAN
0028   0000 04E1   N        ENT ASIN
0029   0000 04AB   N        ENT ACOS
0030   0000 0419   N        ENT ATAN
0031   0000 054A   N        ENT LOG10
0032   0000 0536   N        ENT NATLOG
0033   0000 06AF   N        ENT TENEXP
0034   0000 0697   N        ENT FEXP
0035   0000 080E   N        ENT XTOY
0036                  *
0037   0000 7E 0000 X       EXT FTAREG
0038   0003 7E 0000 X       EXT FTCREG
0039   0006 7E 0000 X       EXT FFDREG
0040   0009 7E 0000 X       EXT FADD
0041   000C 7E 0000 X       EXT FSUB
0042   000F 7E 0000 X       EXT FMUL
0043   0012 7E 0000 X       EXT FDIV
0044   0015 7E 0000 X       EXT FIX
0045   0018 7E 0000 X       EXT FLOAT
0046   001B 7E 0000 X       EXT FABS
0047   001E 7E 0000 X       EXT FNEG
0048   0021 7E 0000 X       EXT FSIGN
0049   0024 7E 0000 X       EXT ASCII
0050   0027 7E 0000 X       EXT TLIENR
0051   002A 7E 0000 X       EXT FCOMPR
0052   002D 7E 0000 X       EXT MOVER
0053                  *
0054                  *
0055                  * DATA TABLES
0056                  *
0057   0030 8000      COSCON  FDB $8000
0058   0032 4999              FDB $4999
0059   0034 9999              FDB $9999
0060   0036 6300              FDB $6300
```

```
0061  0038 00          FCB $00
0062  0039 4001        FDB $4001
0063  003B 4166        FDB $4166
0064  003D 6641        FDB $6641
0065  003F 8000        FDB $8000
0066  0041 00          FCB $00
0067  0042 C002        FDB $C002
0068  0044 1388        FDB $1388
0069  0046 8397        FDB $8397
0070  0048 0000        FDB $0000
0071  004A 00          FCB $00
0072  004B 4004        FDB $4004
0073  004D 2476        FDB $2476
0074  004F 0900        FDB $0900
0075  0051 0000        FDB $0000
0076  0053 00          FCB $00
0077  0054 C006        FDB $C006
0078  0056 2605        FDB $2605
0079  0058 0000        FDB $0000
0080  005A 0000        FDB $0000
0081  005C 00          FCB $00
0082                 *
0083  005D 0001  TWOPI FDB $0001
0084  005F 6283        FDB $6283
0085  0061 1853        FDB $1853
0086  0063 0600        FDB $0600
0087  0065 00          FCB $00
0088                 *
0089  0066 0001  PIOV2 FDB $0001
0090  0068 1570        FDB $1570
0091  006A 7963        FDB $7963
0092  006C 2700        FDB $2700
0093  006E 00          FCB $00
0094                 *
0095  006F 8000  ATNCON FDB $8000
0096  0071 3333        FDB $3333
0097  0073 3145        FDB $3145
0098  0075 2800        FDB $2800
0099  0077 00          FCB $00
0100  0078 0000        FDB $0000
0101  007A 1999        FDB $1999
0102  007C 3550        FDB $3550
0103  007E 8500        FDB $8500
0104  0080 00          FCB $00
0105  0081 8000        FDB $8000
0106  0083 1420        FDB $1420
0107  0085 8899        FDB $8899
0108  0087 4400        FDB $4400
0109  0089 00          FCB $00
0110  008A 0000        FDB $0000
0111  008C 1065        FDB $1065
0112  008E 6263        FDB $6263
0113  0090 9300        FDB $9300
0114  0092 00          FCB $00
0115  0093 C001        FDB $C001
0116  0095 7528        FDB $7528
0117  0097 9640        FDB $9640
0118  0099 0000        FDB $0000
0119  009B 00          FCB $00
0120  009C 4001        FDB $4001
0121  009E 4290        FDB $4290
0122  00A0 9613        FDB $9613
0123  00A2 8000        FDB $8000
0124  00A4 00          FCB $00
0125  00A5 C001        FDB $C001
0126  00A7 1616        FDB $1616
0127  00A9 5736        FDB $5736
0128  00AB 7000        FDB $7000
0129  00AD 00          FCB $00
0130  00AE 4002        FDB $4002
0131  00B0 2866        FDB $2866
0132  00B2 2257        FDB $2257
0133  00B4 0000        FDB $0000
0134  00B6 00          FCB $00
0135                 *
0136  00B7 0000  ASNCON FDB $0000
0137  00B9 1666        FDB $1666
0138  00BB 6666        FDB $6666
0139  00BD 6666        FDB $6666
0140  00BF 67          FCB $67
0141  00C0 4001        FDB $4001
0142  00C2 7500        FDB $7500
0143  00C4 0000        FDB $0000
0144  00C6 0000        FDB $0000
0145  00C8 00          FCB $00
0146  00C9 4001        FDB $4001
0147  00CB 4464        FDB $4464
0148  00CD 7857        FDB $7857
0149  00CF 0000        FDB $0000
0150  00D1 00          FCB $00
0151  00D2 4001        FDB $4001
0152  00D4 3038        FDB $3038
0153  00D6 1944        FDB $1944
0154  00D8 0000        FDB $0000
0155  00DA 00          FCB $00
0156  00DB 4001        FDB $4001
0157  00DD 2237        FDB $2237
0158  00DF 2159        FDB $2159
0159  00E1 0000        FDB $0000
0160  00E3 00          FCB $00
0161  00E4 4001        FDB $4001
0162  00E6 1735        FDB $1735
0163  00E8 2764        FDB $2764
0164  00EA 0000        FDB $0000
0165  00EC 00          FCB $00
0166  00ED 4001        FDB $4001
0167  00EF 1396        FDB $1396
0168  00F1 4844        FDB $4844
0169  00F3 0000        FDB $0000
0170  00F5 00          FCB $00
0171  00F6 4001        FDB $4001
0172  00F8 1155        FDB $1155
0173  00FA 1801        FDB $1801
0174  00FC 0000        FDB $0000
0175  00FE 00          FCB $00
0176  00FF 4002        FDB $4002
0177  0101 9761        FDB $9761
0178  0103 6100        FDB $6100
0179  0105 0000        FDB $0000
0180  0107 00          FCB $00
0181  0108 4002        FDB $4002
0182  010A 8390        FDB $8390
```

```
0183   010C 3360          FDB $3360
0184   010E 0000          FDB $0000
0185   0110 00            FCB $00
0186                 *
0187   0111 0001    ONE   FDB $0001
0188   0113 1000          FDB $1000
0189   0115 0000          FDB $0000
0190   0117 0000          FDB $0000
0191   0119 00            FCB $00
0192                 *
0193   011A 0255    INFIN FDB $0255
0194   011C 9999          FDB $9999
0195   011E 9999          FDB $9999
0196   0120 9999          FDB $9999
0197   0122 99            FCB $99
0198                 *
0199   0123 0001    LNCON FDB $0001
0200   0125 2302          FDB $2302
0201   0127 5850          FDB $5850
0202   0129 9300          FDB $9300
0203   012B 00            FCB $00
0204                 *
0205   012C 0000    LOGE  FDB $0000
0206   012E 4342          FDB $4342
0207   0130 9448          FDB $9448
0208   0132 2000          FDB $2000
0209   0134 00            FCB $00
0210                 *
0211   0135 0003    C255  FDB $0003
0212   0137 2550          FDB $2550
0213   0139 0000          FDB $0000
0214   013B 0000          FDB $0000
0215   013D 00            FCB $00
0216                 *
0217   013E 0000    TCON1 FDB $0000
0218   0140 5000          FDB $5000
0219   0142 0000          FDB $0000
0220   0144 0000          FDB $0000
0221   0146 00            FCB $00
0222   0147 4001          FDB $4001
0223   0149 4166          FDB $4166
0224   014B 6666          FDB $6666
0225   014D 6666          FDB $6666
0226   014F 67            FCB $67
0227   0150 4002          FDB $4002
0228   0152 1388          FDB $1388
0229   0154 8890          FDB $8890
0230   0156 0000          FDB $0000
0231   0158 00            FCB $00
0232   0159 4004          FDB $4004
0233   015B 2480          FDB $2480
0234   015D 2000          FDB $2000
0235   015F 0000          FDB $0000
0236   0161 00            FCB $00
0237   0162 4006          FDB $4006
0238   0164 2760          FDB $2760
0239   0166 0000          FDB $0000
0240   0168 0000          FDB $0000
0241   016A 00            FCB $00
0242                 *
0243   016B 0000    TCON2 FDB $0000
0244   016D 1666          FDB $1666
0245   016F 6666          FDB $6666
0246   0171 6666          FDB $6666
0247   0173 67            FCB $67
0248   0174 4002          FDB $4002
0249   0176 8333          FDB $8333
0250   0178 3333          FDB $3333
0251   017A 3333          FDB $3333
0252   017C 33            FCB $33
0253   017D 4003          FDB $4003
0254   017F 1984          FDB $1984
0255   0181 1300          FDB $1300
0256   0183 0000          FDB $0000
0257   0185 00            FCB $00
0258   0186 4005          FDB $4005
0259   0188 2756          FDB $2756
0260   018A 0000          FDB $0000
0261   018C 0000          FDB $0000
0262   018E 00            FCB $00
0263   018F 4007          FDB $4007
0264   0191 2500          FDB $2500
0265   0193 0000          FDB $0000
0266   0195 0000          FDB $0000
0267   0197 00            FCB $00
0268                 *
0269   0198 0000    LOGCON FDB $0000
0270   019A 8685          FDB $8685
0271   019C 8896          FDB $8896
0272   019E 3800          FDB $3800
0273   01A0 00            FCB $00
0274   01A1 0000          FDB $0000
0275   01A3 2895          FDB $2895
0276   01A5 2965          FDB $2965
0277   01A7 4600          FDB $4600
0278   01A9 00            FCB $00
0279   01AA 0000          FDB $0000
0280   01AC 1737          FDB $1737
0281   01AE 1779          FDB $1779
0282   01B0 2800          FDB $2800
0283   01B2 00            FCB $00
0284   01B3 0000          FDB $0000
0285   01B5 1240          FDB $1240
0286   01B7 8413          FDB $8413
0287   01B9 7700          FDB $7700
0288   01BB 00            FCB $00
0289   01BC 4001          FDB $4001
0290   01BE 9650          FDB $9650
0291   01C0 9884          FDB $9884
0292   01C2 8800          FDB $8800
0293   01C4 00            FCB $00
0294   01C5 4001          FDB $4001
0295   01C7 7896          FDB $7896
0296   01C9 2633          FDB $2633
0297   01CB 0700          FDB $0700
0298   01CD 00            FCB $00
0299   01CE 4001          FDB $4001
0300   01D0 6681          FDB $6681
0301   01D2 4535          FDB $4535
0302   01D4 6800          FDB $6800
0303   01D6 00            FCB $00
0304   01D7 4001          FDB $4001
```

```
01D9 5790                FDB $5790
01DB 5930                FDB $5930
01DD 9200                FDB $9200
01DF 00                  FCB $00
01E0 4001                FDB $4001
01E2 5109                FDB $5109
01E4 3468                FDB $3468
01E6 4600                FDB $4600
01E8 00                  FCB $00
01E9 4001                FDB $4001
01EB 4571                FDB $4571
01ED 5208                FDB $5208
01EF 6200                FDB $6200
01F1 00                  FCB $00
01F2 4001                FDB $4001
01F4 4136                FDB $4136
01F6 1379                FDB $1379
01F8 2300                FDB $2300
01FA 00                  FCB $00
01FB 4001                FDB $4001
01FD 3776                FDB $3776
01FF 4737                FDB $4737
0201 5600                FDB $5600
0203 00                  FCB $00
                 *
0204 0001        THREE   FDB $0001
0206 3000                FDB $3000
0208 0000                FDB $0000
020A 0000                FDB $0000
020C 00                  FCB $00
                 *
020D 0000        LOGOF3  FDB $0000
020F 4771                FDB $4771
0211 2175                FDB $2125
0213 4900                FDB $4900
0215 00                  FCB $00
                 * FACTOR EXTRACTION SUBROUTINE
                 * ARGUMENT POINTER IN 'ARGPNT'
                 * CONSTANT POINTER IN 'CONPNT'
                 * INTEGER NUM. OF FACTORS IN IFACT
                 *
0216 FE 0262 R   EXTRCT  LDX ARGPNT
0219 BD 0003 R           JSR FTCREG
021C FE 0264 R           LDX CONPNT
021F BD 0000 R           JSR FTAREG
0222 BD 0012 R           JSR FDIV        ARGUMENT/CONSTANT
0225 CE 0266 R           LDX #FTEMP
0228 BD 0006 R           JSR FFDREG
022B CE 0266 R           LDX #FTEMP
022E BD 0015 R           JSR FIX         INTEGERIZE
0231 FF 026F R           STX IFACT
0234 FF 0050 C           STX ACCUM1
0237 CE 0266 R           LDX #FTEMP
023A BD 0018 R           JSR FLOAT       FLOAT BACK
023D CE 0266 R           LDX #FTEMP
0240 BD 0000 R           JSR FTAREG
0243 FE 0264 R           LDX CONPNT
0246 BD 0003 R           JSR FTCREG
0249 BD 000F R           JSR FMUL        MULTIPLY BACK
024C CE 0266 R           LDX #FTEMP
024F BD 0006 R           JSR FFDREG
0252 CE 0266 R           LDX #FTEMP
0255 BD 0003 R           JSR FTCREG
0258 FF 0262 R           LDX ARGPNT
025B BD 0000 R           JSR FTAREG
025E BD 000C R           JSR FSUB        GET DIFFERENCE
0261 39                  RTS
                 *
0262 0002        ARGPNT  RMB  2
0264 0002        CONPNT  RMB  2
0266 0009        FTEMP   RMB  9
026F 0002        IFACT   RMB  2
                 * SERIES EXPANSION SUBROUTINE
                 * X-REGISTER POINTS TO ARGUMENT
                 * 'ANGLE' CONTAINS ADDRESS OF COEFFICENTS
                 * 'TERMS' CONTAINS NUMBER OF TERMS
                 * 'SUM' IS INITIALIZED TO FIRST TERM
                 *
0271 FF 02FE R   SERIES  STX ARGSAV
0274 BD 0003 R           JSR FTCREG      FIRST POWER=ARG*ARG
0277 FE 02FE R           LDX ARGSAV
027A BD 0000 R           JSR FTAREG
027D BD 000F R           JSR FMUL
0280 CE 0266 R           LDX #FTEMP
0283 BD 0006 R           JSR FFDREG      EACH NEW TERM=OLD*ARG*ARG
0286 CE 02E3 R           LDX #POWER
0289 6F 00               CLR 0,X
028B BD 0021 R           JSR FSIGN       MAKE FIRST POWER=1
                 *
028E CE 02E3 R   SER2    LDX #POWER
0291 BD 0000 R           JSR FTAREG
0294 CE 0266 R           LDX #FTEMP
0297 BD 0003 R           JSR FTCREG
029A BD 000F R           JSR FMUL        MAKE NEW POWER
029D CE 02E3 R           LDX #POWER
02A0 BD 0006 R           JSR FFDREG
02A3 FE 0301 R           LDX ANGLE
02A6 B6 0301 R           LDA A ANGLE
02A9 F6 0302 R           LDA B ANGLE+1
02AC CB 09               ADD B #9        MOVE COEFFICIENT POINTER
02AE 89 00               ADC A #0
02B0 B7 0301 R           STA A ANGLE
02B3 F7 0302 R           STA B ANGLE+1
02B6 BD 0000 R           JSR FTAREG
02B9 CE 02E3 R           LDX #POWER
02BC BD 0003 R           JSR FTCREG
02BF BD 000F R           JSR FMUL        MULTIPLY TIMES COEFFICENT
02C2 CE 02EC R           LDX #STEMP
02C5 BD 0006 R           JSR FFDREG
02C8 CE 02F5 R           LDX #SUM
02CB BD 0000 R           JSR FTAREG
02CE CE 02EC R           LDX #STEMP
02D1 BD 0003 R           JSR FTCREG
02D4 BD 0009 R           JSR FADD        ADD NEW TERM
02D7 CE 02F5 R           LDX #SUM
02DA BD 0006 R           JSR FFDREG
02DD 7A 0300 R           DEC TERMS       DONE?
02E0 26 AC               BNE SER2        LOOP UNTIL DONE
                 *
02E2 39                  RTS
                 *
02E3 0009        POWER   RMB  9
```

```
02EC 0009        STEMP   RMB  9
02F5 0009        SUM     RMB  9
02FE 0002        ARGSAV  RMB  2
0300 0001        TERMS   RMB  1
0301 0002        ANGLE   RMB  2
0303 0002        RETURN  RMB  2
0305 0009        SINVAL  RMB  9
030E 0009        COSVAL  RMB  9
0317 0002        CRET    RMB  2
0319 0002        CCRET   RMB  2
031B 0009        COSTMP  RMB  9
0324 0001        SIGN    RMB  1
0325 0001        ASIGN   RMB  1
0326 0001        FLAG    RMB  1
                 * TANGENT FUNCTION
                 * TAN(ARG)=SIN(ARG)/COS(ARG)
                 *
0327 FF 0303 R TAN       STX RETURN      SAVE RETURN ADDRESS
032A FF 0063 C           STX ADRFM
032D CE 0305 R           LDX #SINVAL
0330 FF 0061 C           STX ADRTO
0333 C6 09               LDA B #9        MAKE COPY OF ARG AT SINVAL
0335 BD 002D R           JSR MOVER
0338 FE 0303 R           LDX RETURN
033B FF 0063 C           STX ADRFM
033E CE 030F R           LDX #COSVAL
0341 FF 0061 C           STX ADRTO
0344 C6 09               LDA B #9        MAKE COPY OF ARG AT COSVAL
0346 BD 002D R           JSR MOVER
0349 CE 030E R           LDX #COSVAL
034C BD 036A R           JSR COS         COMPUTE COS(ARG)
034F CE 0305 R           LDX #SINVAL
0352 BD 03FE R           JSR SIN         COMPUTE SIN(ARG)
0355 CE 0305 R           LDX #SINVAL
0358 BD 0003 R           JSR FTCREG
035B CE 030E R           LDX #COSVAL
035E BD 0000 R           JSR FTAREG
0361 BD 0012 R           JSR FDIV        COMPUTE SIN/COS
0364 FE 0303 R           LDX RETURN
0367 7E 0006 R           JMP FFDREG      RETURN VALUE
                 * COSINE FUNCTION
                 * USES SERIES EXPANSION ALGORITHM
                 * 8-DIGIT ACCURACY
                 *
036A FF 0317 R COS       STX CRET        SAVE RETURN ADDRESS
036D BD 001B R           JSR FABS        MAKE ARG POSITIVE
0370 FF 0262 R           STX ARGPNT
0373 CE 005D R           LDX #TWOPI
0376 FF 0264 R           STX CONPNT
0379 BD 0216 R           JSR EXTRCT      REMOVE FACTORS OF 360 DEG
037C CE 031B R           LDX #COSTMP
037F BD 0006 R           JSR FFDREG      SAVE ARG
0382 B6 031B R           LDA A COSTMP
0385 85 20               BIT A #$20      CHECK FOR ARG=0
0387 27 05               BEQ COS2        NONZERO
                 *
0389 CE 0111 R           LDX #ONE        COS(0)=1
038C 20 62               BRA COS5
                 *
038E CE 031B R COS2      LDX #COSTMP
0391 FF 0262 R           STX ARGPNT
0394 CE 0066 R           LDX #PIOV2
0397 FF 0264 R           STX CONPNT
039A BD 0216 R           JSR EXTRCT      REMOVE FACTORS OF 90 DEG
039D CE 031B R           LDX #COSTMP
03A0 BD 0006 R           JSR FFDREG
03A3 B6 0270 R           LDA A IFACT+1   CHECK QUADRANT
03A6 85 01               BIT A #$01      90 OR 270?
03A8 27 15               BEQ COS3
                 *
03AA CE 031B R           LDX #COSTMP
03AD BD 0003 R           JSR FTCREG      QUADRANTS 1,3 USE 90-ARG
03B0 CE 0066 R           LDX #PIOV2
03B3 BD 0000 R           JSR FTAREG
03B6 BD 000C R           JSR FSUB
03B9 CE 031B R           LDX #COSTMP
03BC BD 0006 R           JSR FFDREG
                 *
                 *
03BF 86 05       COS3    LDA A #5        SET UP FOR SERIES
03C1 B7 0300 R           STA A TERMS     5 TERMS IN COSINE SERIES
03C4 CE 0030 R           LDX #COSCON
03C7 FF 0301 R           STX ANGLE
03CA CE 02F5 R           LDX #SUM
03CD 6F 00               CLR 0,X
03CF BD 0021 R           JSR FSIGN       FIRST TERM=1
03D2 CE 031B R           LDX #COSTMP
03D5 BD 0271 R           JSR SERIES      EXPAND SERIES
03D8 CE 031B R           LDX #COSTMP
03DB BD 0006 R           JSR FFDREG
03DE B6 0270 R           LDA A IFACT+1   CHECK QUADRANT
03E1 27 0A               BEQ COS4        0-90 DEG
                 *
03E3 81 03               CMP A #3
03E5 27 06               BEQ COS4        270-360 DEG
                 *
03E7 CE 031B R           LDX #COSTMP
03EA BD 001E R           JSR FNEG        FOR 90-270 DEG., NEGATE
03ED CE 031B R COS4      LDX #COSTMP
                 *
03F0 FF 0063 C COS5      STX ADRFM       MOVE RESULT TO RETURN ADDRESS
03F3 FE 0317 R           LDX CRET
03F6 FF 0061 C           STX ADRTO
03F9 C6 09               LDA B #9
03FB 7E 002D R           JMP MOVER
                 * SINE FUNCTION
                 * USES COSINE(90-ARG)
                 * 8-DIGIT ACCURACY
                 *
03FE FF 0317 R SIN       STX CRET        SAVE RETURN ADDRESS
0401 BD 0003 R           JSR FTCREG
0404 CE 0066 R           LDX #PIOV2
0407 BD 0000 R           JSR FTAREG
040A BD 000C R           JSR FSUB        COMPUTE 90-ARG
040D FE 0317 R           LDX CRET
0410 BD 0006 R           JSR FFDREG
0413 FE 0317 R           LDX CRET
0416 7E 036A R           JMP COS         NOW USE COSINE
                 * INVERSE-TANGENT FUNCTION
                 * USES SERIES EXPANSION ALGORITHM
                 * 8-DIGIT ACCURACY
                 *
```

```
0419 FF 0317 R ATAN     STX CRET        SAVE RETURN ADDRESS
041C A6 00              LDA A 0,X
041E 7F 0326 R          CLR FLAG
0421 85 20              BIT A #$20      CHECK FOR ARG=0
0423 27 01              BEQ ATAN2       NONZERO
                      *
0425 39                 RTS             ATAN(0)=0
                      *
0426 B7 0324 R ATAN2    STA A SIGN      SAVE SIGN
0429 BD 001B R          JSR FABS        MAKE POSITIVE
042C FF 0050 C          STX ACCUM1
042F CE 0111 R          LDX #ONE
0432 BD 002A R          JSR FCOMPR      CHECK FOR ARG>1
0435 2C 18              BGE ATAN3       NO
                      *
0437 73 0326 R          COM FLAG
                      * FOR ARGS>1 INVERT ARG AND COMPUTE SERIES
                      * THEN ATAN()=90-ATAN()
043A CE 0111 R          LDX #ONE
043D BD 0003 R          JSR FTCREG
0440 FE 0317 R          LDX CRET
0443 BD 0000 R          JSR FTAREG
0446 BD 0012 R          JSR FDIV        ARG=1/ARG
0449 FE 0317 R          LDX CRET
044C BD 0006 R          JSR FFDREG
044F 86 08      ATAN3   LDA A #8        SET UP FOR SERIES
0451 B7 0300 R          STA A TERMS     8 TERMS IN ATAN SERIES
0454 CE 006F R          LDX #ATNCON
0457 FF 0301 R          STX ANGLE
045A CE 02F5 R          LDX #SUM
045D 6F 00              CLR 0,X
045F BD 0021 R          JSR FSIGN       FIRST TERM=1
0462 FE 0317 R          LDX CRET
0465 BD 0271 R          JSR SERIES      EXPAND SERIES
0468 CE 0305 R          LDX #SINVAL
046B BD 0006 R          JSR FFDREG
046E CE 0305 R          LDX #SINVAL
0471 BD 0003 R          JSR FTCREG
0474 FE 0317 R          LDX CRET
0477 BD 0000 R          JSR FTAREG
047A BD 000F R          JSR FMUL
047D CE 0305 R          LDX #SINVAL
0480 BD 0006 R          JSR FFDREG
0483 7D 0326 R          TST FLAG        WAS ARG>1?
0486 27 15              BEQ ATAN4       NO
                      *
0488 CE 0066 R          LDX #PIOV2
048B BD 0000 R          JSR FTAREG
048E CE 0305 R          LDX #SINVAL
0491 BD 0003 R          JSR FTCREG
0494 BD 000C R          JSR FSUB        ATAN(>1)=90-ATAN(<1)
0497 CE 0305 R          LDX #SINVAL
049A BD 0006 R          JSR FFDREG
049D CE 0305 R ATAN4    LDX #SINVAL
04A0 7D 0324 R          TST SIGN        FIX UP SIGN
04A3 2A 03              BPL ATAN5       POSITIVE SIGN, O.K
                      *
04A5 BD 001E R          JSR FNEG        ATAN(-X)=-ATAN(X)
04A8 7E 03F0 R ATAN5    JMP COS5        RETURN VALUE
                      * INVERSE-COSINE FUNCTION
                      * USES INVERSE-SINE FUNCTION
                      * ACOS(X)=90-ASIN(X)
                      *
04AB FF 0319 R ACOS     STX CCRET       SAVE RETURN ARGUMENT
04AE A6 00              LDA A 0,X       GET SIGNS
04B0 B7 0325 R          STA A ASIGN
04B3 BD 04F1 R          JSR ASIN        COMPUTE ASIN(ARG)
04B6 B6 0325 R          LDA A ASIGN     RECOVER SIGN
04B9 2B 11              BMI ACOS2       NEGATIVE
                      *
04BB CE 0066 R          LDX #PIOV2      POSITIVE ANGLE
04BE BD 0000 R          JSR FTAREG
04C1 FE 0319 R          LDX CCRET
04C4 BD 0003 R          JSR FTCREG
04C7 BD 000C R          JSR FSUB        ACOS(X)=90-ASIN(X)
04CA 20 0F              BRA ACOS3
                      *
04CC FE 0319 R ACOS2    LDX CCRET       NEGATIVE ANGLE
04CF BD 0000 R          JSR FTAREG
04D2 CE 0066 R          LDX #PIOV2
04D5 BD 0003 R          JSR FTCREG
04D8 BD 0009 R          JSR FADD        ACOS(-X)=90+ASIN(-X)
04DB FE 0319 R ACOS3    LDX CCRET
04DE 7E 0006 R          JMP FFDREG      RETURN VALUE
                      * INVERSE-SINE FUNCTION
                      * USES SERIES EXPANSION ALGORITHM
                      * 8-DIGIT ACCURACY  (ABS(X)<0.5)
                      * 5-DIGIT           (0.5<ABS(X)<0.75)
                      * 3-DIGIT           (0.75<ABS(X)<0.9)
                      *
04E1 FF 0317 R ASIN     STX CRET        SAVE RETURN ADDRESS
04E4 A6 00              LDA A 0,X
04E6 85 20              BIT A #$20      CHECK FOR ZERO
04E8 27 01              BEQ ASIN2       NONZERO
                      *
04EA 39                 RTS             ASIN(0)=0
                      *
04EB B7 0324 R ASIN2    STA A SIGN      SAVE SIGN
04EE FF 0050 C          STX ACCUM1
04F1 BD 001B R          JSR FABS        MAKE POSITIVE
04F4 CE 0111 R          LDX #ONE
04F7 BD 002A R          JSR FCOMPR      CHECK THAT ARG<1
04FA 2A 03              BPL ASIN3       O.K.
                      *
04FC BD 0027 R          JSR ILLERR      ERROR!!!
                      *
04FF 86 0A      ASIN3   LDA A #10       SET UP FOR SERIES
0501 B7 0300 R          STA A TERMS     10 TERMS IN ASIN SERIES
0504 CE 0067 R          LDX #ASNCON
0507 FF 0301 R          STX ANGLE
050A CE 02F5 R          LDX #SUM
050D 6F 00              CLR 0,X
050F BD 0021 R          JSR FSIGN       FIRST TERM=1
0512 FE 0317 R          LDX CRET
0515 BD 0271 R          JSR SERIES      EXPAND SERIES
0518 CE 0305 R          LDX #SINVAL
051B BD 0006 R          JSR FFDREG
051E CE 0305 R          LDX #SINVAL
0521 BD 0003 R          JSR FTCREG
0524 FE 0317 R          LDX CRET
0527 BD 0000 R          JSR FTAREG
052A BD 000F R          JSR FMUL
```

```
052D CE 0305 R          LDX #SINVAL
0530 BD 0006 R          JSR FFDREG
0533 7E 049D R          JMP ATAN4        FINISH UP LIKE ATAN
                      * NATURAL LOGARITHM FUNCTION
                      * USES BASE-10 LOG FUNCTION
                      * 8-DIGIT ACCURACY
                      *
0536 8D 12            NATLOG  BSR LOG10        GET BASE-10 LOG
0538 FE 0317 R          LDX CRET
053B BD 0000 R          JSR FTAREG
053E CE 0123 R          LDX #LNCON
0541 BD 0003 R          JSR FTCREG
0544 BD 000F R          JSR FMUL         CONVERT TO BASE-E
0547 7E 0691 R          JMP LOG15
                      *
                      * BASE-10 LOGARITHM FUNCTION
                      * USES SERIES-EXPANSION ALGORITHM
                      * 8-DIGIT ACCURACY
                      *
054A FF 0317 R        LOG10   STX CRET         SAVE RETURN ADDRESS
054D A6 00              LDA A 0,X        GET SIGNS
054F 2A 03              BPL LOG2         CHECK FOR NEGATIVE
                      *
0551 BD 0027 R          JSR ILLERR       ERROR!!!!
                      *
0554 85 20            LOG2    BIT A #$20       CHECK FOR ZERO
0556 27 09              BEQ LOG3         NONZERO
                      *
0558 CE 011A R          LDX #INFIN       LOG(0)=-INFINITY
055B BD 001E R          JSR FNEG
055E 7E 03F0 R          JMP COS5
                      *
0561 CE 0000          LOG3    LDX #0
0564 FF 031B R          STX COSTMP       CLEAR SPACE
0567 FF 031D R          STX COSTMP+2
056A FF 031F R          STX COSTMP+4
056D FF 0321 R          STX COSTMP+6
0570 FF 0323 R          STX COSTMP+8
0573 85 40              BIT A #$40       CHECK EXPONENT SIGN
0575 27 05              BEQ LOG4
                      *
0577 C6 80              LDA B #$80       SET MINUS SIGN
0579 F7 031B R          STA B COSTMP
057C C6 03            LOG4    LDA B #3         3 DIGITS OF EXPONENT
057E 84 0F              AND A #$0F       GET LEFT DIGIT
0580 26 03              BNE LOG5         NONZERO
                      *
0582 5A                 DEC B            SUPPRESS ZERO
0583 20 07              BRA LOG6
                      *
0585 48               LOG5    ASL A
0586 48                 ASL A            MOVE DIGIT
0587 48                 ASL A
0588 48                 ASL A
0589 B7 031D R          STA A COSTMP+2
058C FE 0317 R        LOG6    LDX CRET
058F A6 01              LDA A 1,X        GET TWO LOW DIGITS
0591 84 F0              AND A #$F0       HIGH DIGIT
0593 C1 03              CMP B #3         LEADING ZERO?
0595 26 0C              BNE LOG7         YES
                      *
0597 44                 LSR A
0598 44                 LSR A            MOVE DIGIT OVER
0599 44                 LSR A
059A 44                 LSR A
059B BA 031D R          ORA A COSTMP+2
059E B7 031D R        LOG61   STA A COSTMP+2
05A1 20 04              BRA LOG8
                      *
05A3 4D               LOG7    TST A            DIGIT ZERO?
05A4 26 F8              BNE LOG61        NO
                      *
05A6 5A                 DEC B            SUPPRESS ZERO
05A7 A6 01            LOG8    LDA A 1,X
05A9 84 0F              AND A #$0F       GET LOW DIGIT
05AB C1 03              CMP B #3
05AD 26 09              BNE LOG9
                      *
05AF 48                 ASL A
05B0 48                 ASL A            MOVE DIGIT OVER
05B1 48                 ASL A
05B2 48                 ASL A
05B3 B7 031E R          STA A COSTMP+3
05B6 20 1D              BRA LOG12
                      *
05B8 C1 02            LOG9    CMP B #2
05BA 26 08              BNE LOG101
                      *
05BC BA 031D R          ORA A COSTMP+2
05BF B7 031D R          STA A COSTMP+2
05C2 20 11              BRA LOG12
                      *
05C4 4D               LOG101  TST A            LAST DIGIT ZERO
05C5 26 07              BNE LOG11
                      *
05C7 86 20              LDA A #$20       SET ZERO BIT
05C9 B7 031B R          STA A COSTMP
05CC 20 0A              BRA LOG13
                      *
05CE 48               LOG11   ASL A
05CF 48                 ASL A            MOVE DIGIT OVER
05D0 48                 ASL A
05D1 48                 ASL A
05D2 B7 031D R          STA A COSTMP+2
05D5 F7 031C R        LOG12   STA B COSTMP+1   STORE EXPONENT
05D8 FE 0317 R        LOG13   LDX CRET
05DB 6F 00              CLR 0,X          REMOVE EXPONENT
05DD 6F 01              CLR 1,X
05DF BD 0000 R          JSR FTAREG
05E2 CE 0204 R          LDX #THREE
05E5 BD 0003 R          JSR FTCREG
05E8 BD 000F R          JSR FMUL         MULTIPLY ARG  BY 3.00
05EB FE 0317 R          LDX CRET
05EE BD 0006 R          JSR FFDREG
05F1 FE 0317 R          LDX CRET
05F4 BD 0000 R          JSR FTAREG
05F7 CE 0111 R          LDX #ONE
05FA BD 0003 R          JSR FTCREG
05FD BD 000C R          JSR FSUB         X-1
0600 CE 0305 R          LDX #SINVAL
0603 BD 0006 R          JSR FFDREG
0606 FE 0317 R        LOG14   LDX CRET
```

```
0802  0609 BD 0000 R          JSR FTAREG
0803  060C CE 0111 R          LDX #ONE
0804  060F BD 0003 R          JSR FTCREG
0805  0612 BD 0009 R          JSR FADD          X+1
0806  0615 CE 030E R          LDX #COSVAL
0807  0618 BD 0006 R          JSR FFDREG
0808  061B CE 0305 R          LDX #SINVAL
0809  061E BD 0003 R          JSR FTCREG
0810  0621 CE 030E R          LDX #COSVAL
0811  0624 BD 0000 R          JSR FTAREG
0812  0627 BD 0012 R          JSR FDIV          (X-1)/(X+1)
0813  062A CE 0305 R          LDX #SINVAL
0814  062D BD 0006 R          JSR FFDREG
0815  0630 CE 0198 R          LDX #LOGCON       SET UP FOR SERIES
0816  0633 FF 0063 C          STX ADRFM
0817  0636 CE 02F5 R          LDX #SUM
0818  0639 FF 0061 C          STX ADRTO
0819  063C C6 09              LDA B #9
0820  063E BD 002D R          JSR MOVER         FIRST TERM SET
0821  0641 CE 01A1 R          LDX #LOGCON+9
0822  0644 FF 0301 R          STX ANGLE
0823  0647 C6 0B              LDA B #11         ELEVEN TERMS
0824  0649 F7 0300 R          STA B TERMS
0825  064C CE 0305 R          LDX #SINVAL
0826  064F BD 0271 R          JSR SERIES        EXPAND SERIES
0827  0652 CE 030E R          LDX #COSVAL
0828  0655 BD 0006 R          JSR FFDREG
0829  0658 CE 030E R          LDX #COSVAL
0830  065B BD 0003 R          JSR FTCREG
0831  065E CE 0305 R          LDX #SINVAL
0832  0661 BD 0000 R          JSR FTAREG
0833  0664 BD 000F R          JSR FMUL
0834  0667 CE 0305 R          LDX #SINVAL
0835  066A BD 0006 R          JSR FFDREG
0836  066D CE 0305 R          LDX #SINVAL
0837  0670 BD 0000 R          JSR FTAREG        ADD IN POWERS-OF-10
0838  0673 CE 031B R          LDX #COSTMP
0839  0676 BD 0003 R          JSR FTCREG
0840  0679 BD 0009 R          JSR FADD
0841  067C CE 031B R          LDX #COSTMP
0842  067F BD 0006 R          JSR FFDREG
0843  0682 CE 031B R          LDX #COSTMP
0844  0685 BD 0000 R          JSR FTAREG
0845  0688 CE 020D R          LDX #LOGOF3       SUBTRACT LOG10(3.00)
0846  068B BD 0003 R          JSR FTCREG
0847  068E BD 000C R          JSR FSUB
0848  0691 FE 0317 R LOG15    LDX CRET          RETURN VALUE
0849  0694 7E 0006 R          JMP FFDREG
0851                  * EXPONENTIAL FUNCTIONS
0852                  * BASE-10 SERIES EXPANSION ALGORITHM
0853                  * 8-DIGIT ACCURACY
0854                  *
0855  0697 FF 0317 R FEXP     STX CRET          SAVE RETURN ADDRESS
0856  069A BD 0000 R          JSR FTAREG
0857  069D CE 012C R          LDX #LOGE
0858  06A0 BD 0003 R          JSR FTCREG
0859  06A3 BD 000F R          JSR FMUL          TIMES LOG10(E)
0860  06A6 FE 0317 R          LDX CRET
0861  06A9 BD 0006 R          JSR FFDREG
0862  06AC FE 0317 R          LDX CRET          NOW USE BASE-10 EXP,
0863                  *
0864                  *
0865  06AF FF 0317 R TENEXP   STX CRET          SAVE RETURN ADDRESS
0866  06B2 A6 00              LDA A 0,X         GET SIGNS
0867  06B4 85 20              BIT A #$20        CHECK FOR ZERO
0868  06B6 27 06              BEQ TEN2          NONZERO
0869                  *
0870  06B8 CE 0111 R          LDX #ONE          EXP(0)=1
0871  06BB 7E 03F0 R          JMP COS5
0872                  *
0873  06BE B7 0324 R TEN2     STA A SIGN        SAVE SIGNS
0874  06C1 BD 001B R          JSR FABS          MAKE POSITIVE
0875  06C4 FF 0050 C          STX ACCUM1
0876  06C7 CE 031B R          LDX #COSTMP
0877  06CA 6F 00              CLR 0,X
0878  06CC BD 0021 R          JSR FSIGN         COSTMP=1
0879  06CF CE 0111 R          LDX #ONE
0880  06D2 BD 002A R          JSR FCOMPR        CHECK ARG<1
0881  06D5 2F 03              BLE *+5
0882  06D7 7E 075C R          JMP TEN4          YES
0883                  *
0884  06DA FE 0317 R          LDX CRET
0885  06DD FF 0050 C          STX ACCUM1
0886  06E0 CE 0135 R          LDX #C255
0887  06E3 BD 002A R          JSR FCOMPR        CHECK ARG <255
0888  06E6 2C 18              BGE TEN3          YES
0889                  *
0890  06E8 7D 0324 R          TST SIGN          CHECK ARGUMENT SIGN
0891  06EB 2A 0A              BPL TEN21
0892                  *
0893  06ED FE 0317 R          LDX CRET
0894  06F0 86 20              LDA A #$20
0895  06F2 A7 00              STA A 0,X
0896  06F4 7E 0021 R          JMP FSIGN         TENEXP(-255)=0
0897                  *
0898  06F7 CE 011A R TEN21    LDX #INFIN        TENEXP(255)=INFINITY
0899  06FA BD 001B R          JSR FABS
0900  06FD 7E 03F0 R          JMP COS5
0901                  *
0902  0700 FE 0317 R TEN3     LDX CRET
0903  0703 BD 0015 R          JSR FIX           MAKE INTEGER
0904  0706 FF 026F R          STX IFACT
0905  0709 08                 INX               FORM EXPONENT
0906  070A FF 0054 C          STX ACCUM3
0907  070D 4F                 CLR A
0908  070E F6 0055 C          LDA B ACCUM3+1    MAKE LOW BYTE BCD
0909  0711 C0 64      TEN31   SUB B #100        FIND HUNDREDS DIGIT
0910  0713 25 03              BCS *+5
0911                  *
0912  0715 4C                 INC A
0913  0716 20 F9              BRA TEN31
0914                  *
0915  0718 B7 031B R          STA A COSTMP      STORE HUNDREDS DIGIT
0916  071B 4F                 CLR A
0917  071C CB 64              ADD B #100
0918  071E C0 0A      TEN32   SUB B #10         FIND TENS DIGIT
0919  0720 25 03              BCS *+5
0920                  *
0921  0722 4C                 INC A
0922  0723 20 F9              BRA TEN32
0923                  *
0924  0725 CB 0A              ADD B #10
```

```
0925  0727 48                ASL A          SHIFT TENS OVER
0926  0728 48                ASL A
0927  0729 48                ASL A
0928  072A 48                ASL A
0929  072B 1B                ABA            COMBINE TENS AND ONES
0930  072C B7 031C R         STA A COSTMP+1 STORE TENS AND ONES DIGITS
0931                *
0932  072F FE 026F R TEN33   LDX IFACT
0933  0732 FF 0050 C         STX ACCUM1
0934  0735 CE 0305 R         LDX #SINVAL
0935  0738 BD 0018 R         JSR FLOAT      FLOAT BACK
0936  073B FE 0317 R         LDX CRET
0937  073E BD 0000 R         JSR FTAREG
0938  0741 CE 0305 R         LDX #SINVAL
0939  0744 BD 0003 R         JSR FTCREG
0940  0747 BD 000C R         JSR FSUB       GET FRACTIONAL PART
0941  074A FE 0317 R         LDX CRET
0942  074D BD 0006 R         JSR FFDREG
0943  0750 FE 0317 R         LDX CRET
0944  0753 A6 00             LDA A 0,X      CHECK FOR ZERO
0945  0755 85 20             BIT A #$20
0946  0757 27 03             BEQ TEN4       ZERO. SKIP SERIES
0947                *
0948  0759 7E 07EE R         JMP TEN41
0949                *
0950  075C CE 02F5 R TEN4    LDX #SUM       SET UP FOR EVEN SERIES
0951  075F 6F 00             CLR 0,X
0952  0761 BD 0021 R         JSR FSIGN      FIRST TERM=1
0953  0764 FE 0317 R         LDX CRET
0954  0767 BD 0000 R         JSR FTAREG
0955  076A CE 0123 R         LDX #LNCON
0956  076D BD 0003 R         JSR FTCREG
0957  0770 BD 000F R         JSR FMUL       SCALE ARGUMENT
0958  0773 FE 0317 R         LDX CRET
0959  0776 BD 0006 R         JSR FFDREG
0960  0779 C6 05             LDA B #5
0961  077B F7 0300 R         STA B TERMS    5 TERMS
0962  077E CE 013E R         LDX #TCON1
0963  0781 FF 0301 R         STX ANGLE
0964  0784 FE 0317 R         LDX CRET
0965  0787 BD 0271 R         JSR SERIES     EXPAND EVEN SERIES
0966  078A CE 0305 R         LDX #SINVAL
0967  078D BD 0006 R         JSR FFDREG     TEMP. STORE
0968  0790 CE 02F5 R         LDX #SUM       SET UP FOR ODD SERIES
0969  0793 6F 00             CLR 0,X
0970  0795 BD 0021 R         JSR FSIGN      FIRST TERM=1
0971  0798 C6 05             LDA B #5
0972  079A F7 0300 R         STA B TERMS    5 TERMS
0973  079D CE 016B R         LDX #TCON2
0974  07A0 FF 0301 R         STX ANGLE
0975  07A3 FE 0317 R         LDX CRET
0976  07A6 BD 0271 R         JSR SERIES     EXPAND ODD SERIES
0977  07A9 CE 030E R         LDX #COSVAL
0978  07AC BD 0006 R         JSR FFDREG
0979  07AF CE 030E R         LDX #COSVAL
0980  07B2 BD 0000 R         JSR FTAREG
0981  07B5 FE 0317 R         LDX CRET
0982  07B8 BD 0003 R         JSR FTCREG
0983  07BB BD 000F R         JSR FMUL
0984  07BE CE 030E R         LDX #COSVAL
0985  07C1 BD 0006 R         JSR FFDREG
0986  07C4 CE 0305 R         LDX #SINVAL
0987  07C7 BD 0000 R         JSR FTAREG
0988  07CA CE 030E R         LDX #COSVAL
0989  07CD BD 0003 R         JSR FTCREG
0990  07D0 BD 0009 R         JSR FADD       COMBINE SERIES
0991  07D3 CE 0305 R         LDX #SINVAL
0992  07D6 BD 0006 R         JSR FFDREG
0993  07D9 CE 0305 R         LDX #SINVAL
0994  07DC BD 0000 R         JSR FTAREG
0995  07DF CE 031B R         LDX #COSTMP
0996  07E2 BD 0003 R         JSR FTCREG
0997  07E5 BD 000F R         JSR FMUL       ADD IN POWERS-OF-10
0998  07E8 CE 031B R         LDX #COSTMP
0999  07EB BD 0006 R         JSR FFDREG
1000  07EE 7D 0324 R TEN41   TST SIGN       CHECK ARG  SIGN
1001  07F1 2A 15             BPL TEN5
1002                *
1003  07F3 CE 031B R         LDX #COSTMP    ARG  WAS NEGATIVE
1004  07F6 BD 0000 R         JSR FTAREG
1005  07F9 CE 0111 R         LDX #ONE
1006  07FC BD 0003 R         JSR FTCREG     EXP(-X)=1/EXP(X)
1007  07FF BD 0012 R         JSR FDIV
1008  0802 CE 031B R         LDX #COSTMP
1009  0805 BD 0006 R         JSR FFDREG
1010  0808 CE 031B R TEN5    LDX #COSTMP
1011  080B 7E 03F0 R         JMP COS5       RETURN VALUE
1012                *
1014                * "POWER" FUNCTION  (X**Y)
1015                * USES TENEXP(Y*LOG10(X))
1016                * 'X' MUST BE POSITIVE
1017                * ADDRESS OF 'Y' IN ACCUM1
1018                * ADDRESS OF 'X' IN X-REGISTER
1019                *
1020  080E FF 0319 R XTOY     STX CCRET      SAVE RETURN ADDRESS
1021  0811 BD 054A R         JSR LOG10      LOG10(X)
1022  0814 FE 0050 C         LDX ACCUM1
1023  0817 BD 0000 R         JSR FTAREG
1024  081A FE 0319 R         LDX CCRET
1025  081D BD 0003 R         JSR FTCREG
1026  0820 BD 000F R         JSR FMUL       Y*LOG10(X)
1027  0823 FE 0319 R         LDX CCRET
1028  0826 BD 0006 R         JSR FFDREG
1029  0829 FE 0319 R         LDX CCRET
1030  082C 7E 06AF R         JMP TENEXP     NOW USE TENEXP
1031                *
1032                         END
```

```
ACCUM1  0050 C
ACCUM2  0052 C
ACCUM3  0054 C
ACCUM4  0056 C
ACCUM5  0058 C
ACOS    04AB RN
ACOS2   04CC R
ACOS3   04DB R
ADRFM   0063 C
ADRTO   0061 C
ANGLE   0301 R
ARGPNT  0262 R
ARGSAV  02FE R
ASCII   0024 RX
ASIGN   0325 R
ASIN    04E1 RN
ASIN2   04EB R
ASIN3   04FF R
ASNCON  00B7 R
ATAN    0419 RN
ATAN2   0426 R
ATAN3   044F R
ATAN4   049D R
ATAN5   04A8 R
ATNCON  006F R
C255    0135 R
CCRET   0319 R
CONPNT  0264 R
COS     036A RN
COS2    038F R
COS3    03BF R
COS4    03ED R
COS5    03F0 R
COSCON  0030 R
COSTMP  031B R
COSVAL  030F R
CRET    0317 R
EXTRCT  0216 R
FABS    0018 RX
FADD    0009 RX
FCOMPR  002A RX
FDIV    0012 RX
FEXP    0697 RN
FFLOREG 0006 RX
FIX     0015 RX
FLAG    0326 R
FLOAT   0018 RX
FMUL    000F RX
FNEG    001E RX
FSIGN   0021 RX
FSUB    000C RX
FTARGEG 0000 RX
FTCREG  0003 RX
FTEMP   0266 R
IFACT   026F R
ILLERR  0027 RX
INFIN   011A R
LNCON   0123 R
LOG10   054A RN
LOG101  05C4 R
LOG11   05CE R
LOG12   05D5 R
LOG13   05D8 R
LOG14   0606 R
LOG15   0691 R
LOG2    0554 R
LOG3    0561 R
LOG4    057C R
LOG5    0583 R
LOG6    058C R
LOG61   059E R
LOG7    05A3 R
LOG8    05A7 R
LOG9    05B8 R
LOGCON  0198 R
LOGE    012C R
LOGOF3  020D R
MOVER   002D RX
NATLOG  0536 RN
ONE     0111 R
P10V2   0066 R
POWER   02E3 R
RETURN  0303 R
RUNSTK  0000 C
SCIFUN  0000 RN
SER2    026E R
SERIES  0271 R
SIGN    0324 R
SIN     03FE RN
SINVAL  0305 R
STLMP   02EC R
SUM     02F5 R
TAN     0327 RN
TCON1   013E R
TCON2   016B R
TEN2    06BE R
TEN21   06F7 R
TEN3    0700 R
TEN31   0711 R
TEN32   071E R
TEN33   072F R
TEN4    075C R
TEN41   076E R
TEN5    0808 R
TENEXP  040F RN
TERMS   0300 R
THREE   0204 R
TWOPI   005D R
XTOY    080E RN
```

```
0001   0000 0000   N        NAM CP/68DR
0002                       *
0003                       * STRUBAL + DISK DRIVERS FOR CP/68 SYSTEM
0004                       * CONTAINS RANDOM-ACCESS FILE ROUTINE CALLS
0005                       *
0006                       * COPYRIGHT: 1979 BY HEMENWAY ASSOCIATES INC
0007                       *            ALL RIGHTS RESERVED
0008                       *
0009   0000 0000   C        CMN BLANK,101
0010   0000 0065   C        CMN .EOF,2
0011   0000 0067   C        CMN  ERR,2
0012   0000 0069   C        CMN .CURID,2
0013                       *
0014   0000 0018   N        ENT OPEN
0015   0000 00A4   N        ENT CLOSE
0016   0000 00BB   N        ENT REWIND
0017   0000 00D5   N        ENT GETB
0018   0000 0108   N        ENT OUTB
0019   0000 00D3   N        ENT MONTOR
0020   0000 0197   N        ENT BUILD
0021   0000 01FE   N        ENT ENLARGE
0022   0000 0123   N        ENT LOCATE
0023   0000 0161   N        ENT PLACE
0024   0000 0234   N        ENT DELETE
0025                       *
0026   0000 7E 0000 X       EXT  RTBL
0027   0003 7E 0000 X       EXT CREATE
0028   0006 7E 0000 X       EXT ROPEN
0029   0009 7E 0000 X       EXT RCLOSE
0030   000C 7E 0000 X       EXT RREAD
0031   000F 7E 0000 X       EXT RWRITE
0032   0012 7E 0000 X       EXT POSITION
0033   0015 7E 0000 X       EXT EXPAND
0034                       *
0035                       * BASE-PAGE EQUATES
0036                       *
0037   0018 0020           DESCRA EQU $20
0038   0018 0023           CUCHAR EQU $23
0039   0018 0025           RC     EQU $25
0040   0018 0026           CLASS  EQU $26
0041                       *
0042                       * FCB EQUATES
0043                       *
0044   0018 0002           FCBGDT EQU 2
0045   0018 0005           FCBSTA EQU 5
0046   0018 0006           FCBDTT EQU 6
0047   0018 0007           FCBDBA EQU 7
0048   0018 001D           FCBTYP EQU 29
0049   0018 001E           FCBACS EQU 30
0050   0018 0029           FCBSCF EQU 41
0051   0018 002A           FCBRNM EQU 42
0052   0018 002C           FCBRS7 EQU 44
0053   0018 002E           FCBRCD EQU 46
0054   0018 0030           FCBPOS EQU 48
0055                       *
0056                       *
0057                       *
0058   0018 DF 23          OPEN   STX CUCHAR     POINT TO FILENAME STRING
0059   001A FE 0069 C             LDX  CURID     POINT TO FCB
0060   001D 6F 05                 CLR FCBSTA,X   INIT. STATUS
```

```
001F 3F                SWI            CALL CP/68
0020 2C                FCB 44         "FMTFCB"
0021 6D 05             TST FCBSTA,X   ERROR?
0023 26 76             BNE SETER2     YES
                 *
0025 86 FF             LDA A #$FF
0027 A7 29             STA A FCBSCF,X COMPRESSION ON
0029 6F 06             CLR FCBDTT,X   INPUT (TEMP.)
002B 6F 1E             CLR FCBACS,X   ACCESS CODE=0
002D 6F 1D             CLR FCBTYP,X   FILETYPE=0 (BINARY)
002F 7F 0065 C         CLR .EOF
0032 7F 0066 C         CLR  EOF+1     CLEAR FLAG FOR EOF
0035 86 44             LDA A #'D      "DSK"
0037 A7 02             STA A FCBGDT,X
0039 86 53             LDA A #'S
003B A7 03             STA A FCBGDT+1,X
003D 86 4B             LDA A #'K
003F A7 04             STA A FCBGDT+2,X
0041 C6 2A             LDA B #42      42 BYTES IN FCB
0043 3F                SWI            CALL CP/68
0044 0A                FCB 10         "ADDBX"
0045 3F                SWI            CALL CP/68
0046 02                FCB 2          "TXAB"
0047 FE 0069 C         LDX  CURIO     POINT TO FCB
004A A7 07             STA A FCBDBA,X  INIT  BUFFER POINTER
004C E7 08             STA B FCBDBA+1,X
004E 3F                SWI            CALL CP/68
004F 2F                FCB 47         "NXTOK"
0050 96 26             LDA A CLASS    CHECK TOKEN CLASS
0052 81 04             CMP A #4       DELIMITER? (,)
0054 26 3F             BNE SETERR     NO, ERROR
                 *
0056 3F                SWI            CALL CP/68
0057 2F                FCB 47         "NXTOK"
0058 D6 25             LDA B RC       CHECK RC
005A C1 01             CMP B #1       NAME?
005C 26 36             BNE SETERR     NO, ERROR
                 *
005E DF 20             LDX DESCRA
0060 A6 00             LDA A 0,X      GET FIRST CHAR
0062 81 49             CMP A #'I      'INPUT'?
0064 26 0A             BNE OPEN3      NO
                 *
0066 FE 0069 C OPEN1   LDX .CURIO     POINT TO FCB
0069 3F                SWI            CALL CP/68
006A 14                FCB 20         "OPEN"
006B 6D 05     OPEN2   TST FCBSTA,X   ERROR?
006D 26 2C             BNE SETER2     YES
                 *
006F 39                RTS            NO, DONE!
                 *
0070 81 4F     OPEN3   CMP A #'O      'OUTPUT'?
0072 26 07             BNE OPEN4      NO
                 *
0074 FE 0069 C         LDX  CURIO     POINT TO FCB
0077 63 06             COM FCBDTT,X   MAKE OUTPUT
0079 20 EB             BRA OPEN1
                 *
007B 81 52     OPEN4   CMP A #'R      RANDOM?
007D 26 15             BNE SETERR     NO
                 *
007F FE 0069 C         LDX .CURIO     POINT TO FCB
0082 C6 AA             LDA B #170     RESET BUFFER POINTER FOR 170-BYTE FCI
0084 3F                SWI            CALL CP/68
0085 0A                FCB 10         "ADDBX"
0086 3F                SWI            CALL CP/68
0087 02                FCB 2          "TXAB"
0088 FE 0069 C         LDX .CURIO     POINT TO FCB
008B A7 07             STA A FCBDBA,X
008D E7 08             STA B FCBDBA+1,X
008F BD 0006 R         JSR ROPEN      OPEN A RANDOM-ACCESS FILE
0092 20 D7             BRA OPEN2      CHECK ERRORS
                 *
                 *
0094 86 15     SETERR  LDA A #21      FORCE "21" (BAD FILENAME) ERROR
0096 FE 0069 C         LDX  CURIO
0099 A7 05             STA A FCBSTA,X
009B 3F        SETER2  SWI            CALL CP/68
009C 1E                FCB 30         "PRTERR"
009D CE FFFF           LDX #$FFFF
00A0 FF 0067 C         STX  ERR       SET ERROR FLAG
00A3 39                RTS
                 *
                 *
                 *
00A4 FE 0069 C CLOSE   LDX .CURIO     POINT TO FCB
00A7 6F 05             CLR FCBSTA,X   INIT. STATUS
00A9 A6 1D             LDA A FCBTYP,X
00AB 81 02             CMP A #2       RANDOM?
00AD 26 05             BNE CLOSE2     NO
                 *
00AF BD 0009 R         JSR RCLOSE     CLOSE A RANDOM-ACCESS FILE
00B2 20 02             BRA CLOSE3     FINISH UP
                 *
00B4 3F        CLOSE2  SWI            CALL CP/68
00B5 15                FCB 21         "CLOSE"
                 *
00B6 6D 05     CLOSE3  TST FCBSTA,X   ERROR?
00B8 26 E1             BNE SETER2     YES
                 *
00BA 39                RTS            NO, DONE!
                 *
                 *
                 *
00BB FE 0069 C REWIND  LDX  CURIO     POINT TO FCB
00BE A6 1D             LDA A FCBTYP,X
00C0 81 02             CMP A #2       RANDOM?
00C2 26 06             BNE REW2       NO
                 *
00C4 86 12             LDA A #18      IF SO, ERROR 18
00C6 A7 05             STA A FCBSTA,X
00C8 20 D1             BRA SETER2
                 *
00CA 6F 05     REW2    CLR FCBSTA,X   INIT. STATUS
00CC 3F                SWI            CALL CP/68
00CD 16                FCB 22         "REWIND"
00CE 6D 05             TST FCBSTA,X   ERROR?
00D0 26 C9             BNE SETER2     YES
                 *
00D2 39                RTS            NO, DONE!
                 *
                 *
```

```
0183                    *
0184   00D3 3F          MONITOR SWI            CALL CP/68
0185   00D4 1F                  FCB 31        "WARM-START"
0186                    *
0187                    *
0188                    *
0189   00D5 37          GETB    PSH B         SAVE REGISTERS
0190   00D6 3F                  SWI           CALL CP/68
0191   00D7 05                  FCB 5         "PSHX"
0192   00D8 FE 0069 C           LDX .CURIO    POINT TO FCB
0193   00DB E6 1D               LDA B FCBTYP,X
0194   00DD C1 02               CMP B #2      RANDOM?
0195   00DF 26 10               BNE GETB00    NO
0196                    *
0197   00E1 BD 000C R           JSR RREAD     READ FROM RANDOM-ACCESS FILE
0198   00E4 E6 05               LDA B FCBSTA,X
0199   00E6 C1 0F               CMP B #$F     AT END OF FILE?
0200   00E8 27 04               BEQ GETB0     YES
0201                    *
0202   00EA C1 08               CMP B #$8     AT END OF FILE?
0203   00EC 26 10               BNE GETB2     NO
0204                    *
0205   00EE 0D          GETB0   SEC
0206   00EF 20 09               BRA GETB1     RETURN CARRY-SET (EOF)
0207                    *
0208   00F1 3F          GETB00  SWI           CALL CP/68
0209   00F2 18                  FCB 24        "READ"
0210   00F3 E6 05               LDA B FCBSTA,X  CHECK ERROR STATUS
0211   00F5 C1 08               CMP B #8      END-FILE?
0212   00F7 26 05               BNE GETB2     NO
0213                    *
0214   00F9 0D                  SEC           RETURN CARRY-BIT SET
0215   00FA 3F          GETB1   SWI           CALL CP/68
0216   00FB 06                  FCB 6         "PULX"
0217   00FC 33                  PUL B         RESTORE REGISTER
0218   00FD 39                  RTS
0219                    *
0220   00FE 5D          GETB2   TST B         ERROR STATUS?
0221   00FF 26 03               BNE GETB3     YES
0222                    *
0223   0101 0C                  CLC           CLEAR CARRY BIT
0224   0102 20 F6               BRA GETB1
0225                    *
0226   0104 BD 95       GETB3   BSR SETER2    ISSUE ERROR MESSAGE
0227   0106 20 F2               BRA GETB1
0228                    *
0229                    *
0230                    *
0231   0108 37          OUTB    PSH B         SAVE REGISTER
0232   0109 3F                  SWI           CALL CP/68
0233   010A 05                  FCB 5         "PSHX"
0234   010B FF 0069 C           LDX  CURIO    POINT TO FCB
0235   010F 6F 05               CLR FCBSTA,X  INIT.  STATUS
0236   0110 E6 1D               LDA B FCBTYP,X
0237   0112 C1 02               CMP B #2      RANDOM?
0238   0114 26 05               BNE OUTB2     NO
0239                    *
0240   0116 BD 000F R           JSR RWRITE    WRITE TO RANDOM-ACCESS FILE
0241   0119 20 02               BRA OUTB3     FINISH UP
0242                    *
0243   011B 3F          OUTB2   SWI           CALL CP/68
0244   011C 19                  FCB 25        "WRITE"
0245                    *
0246   011D 6D 05       OUTB3   TST FCBSTA,X  ERROR?
0247   011F 26 E3               BNE GETB3     YES
0248                    *
0249   0121 20 D7               BRA GETB1     NO
0250                    *
0252                    * RETURN PRESENT LOCATION OF RANDOM-ACCESS FILE
0253                    *
0254                    *       CALL LOCATE(LFCB,LRECNO,LBYTE)
0255                    *
0256                    * LFCB=RANDOM-ACCESS FCB
0257                    * LRECNO=INTEGER (WILL BE SET TO CURRENT RECORD NUMBER)
0258                    * LBYTE=INTEGER  (WILL BE SET TO CURRENT BYTE POINTER)
0259                    *
0260   0123 BD 0000 R LOCATE JSR  RTBL
0261   0126 F0                  FCB 240       HANDLE STRUBAL+ ARG  LINKAGE
0262   0127 20 08               BRA *+10
0263   0129 0002        LFCB    RMB 2         ADDRESS OF FCB
0264   012B 0002        LRECNO  RMB 2         ADDRESS OF REC  NUMBER
0265   012D 0002        LBYTE   RMB 2         ADDRESS OF BYTE POINTER
0266   012F FFFF                FDB $FFFF
0267   0131 7F 0067 C           CLR .ERR
0268   0134 7F 0068 C           CLR .ERR+1    NO ERRORS YET
0269   0137 FE 0129 R           LDX LFCB      POINT TO FCB
0270   013A A6 1D               LDA A FCBTYP,X
0271   013C 81 02               CMP A #2      RANDOM-FILE?
0272   013E 27 07               BEQ LOC2      YES
0273                    *
0274   0140 86 12               LDA A #18     NO, ERROR 18
0275   0142 A7 05               STA A FCBSTA,X
0276   0144 7E 009B R           JMP SETER2
0277                    *
0278   0147 A6 2E       LOC2    LDA A FCBRCD,X
0279   0149 E6 2F               LDA B FCBRCD+1,X
0280   014B FE 012B R           LDX LRECNO    PUT RECORD NO  INTO LRECNO
0281   014E A7 00               STA A 0,X
0282   0150 E7 01               STA B 1,X
0283   0152 FE 0129 R           LDX LFCB
0284   0155 A6 30               LDA A FCBPOS,X
0285   0157 E6 31               LDA B FCBPOS+1,X
0286   0159 FE 012D R           LDX LBYTE     PUT POINTER INTO LBYTE
0287   015C A7 00               STA A 0,X
0288   015E E7 01               STA B 1,X
0289   0160 39                  RTS
0291                    * PLACE A RANDOM-ACCESS FILE AT A GIVEN RECORD
0292                    *
0293                    *       CALL PLACE(PFCB,PREC)
0294                    *
0295                    * PFCB=ADDRESS OF FCB
0296                    * PREC=INTEGER NUMBER OF DESIRED RECORD
0297                    *
0298   0161 BD 0000 R PLACE  JSR .RTBL
0299   0164 F0                  FCB 240       HANDLE STRUBAL+ ARG  LINKAGE
0300   0165 20 06               BRA *+8
0301   0167 0002        PFCB    RMB 2         ADDRESS OF FCB
0302   0169 0002        PREC    RMB 2         ADDRESS OF DESIRED RECORD
0303   016B FFFF                FDB $FFFF
0304   016D 7F 0067 C           CLR  ERR
0305   0170 7F 0068 C           CLR  ERR+1    NO ERRORS YET
0306   0173 FE 0167 R           LDX PFCB      POINT TO FCB
```

```
0307  0176 A6 1D            LDA A FCBTYP,X
0308  0178 81 02            CMP A #2        RANDOM-ACCESS?
0309  017A 27 07            BEQ PLACE2      YES
0310                  *
0311  017C 86 12            LDA A #18       NO, ERROR 18
0312  017E A7 05            STA A FCBSTA,X
0313  0180 7E 009B R        JMP SETER2
0314                  *
0315  0183 FE 0169 R PLACE2 LDX PREC
0316  0186 A6 00            LDA A 0,X       GET RECORD NO
0317  0188 E6 01            LDA B 1,X
0318  018A FE 0167 R        LDX PFCB        POINT TO FCB
0319  018D A7 2E            STA A FCBRCD,X
0320  018F E7 2F            STA B FCBRCD+1,X
0321  0191 BD 0012 R        JSR POSITION    MOVE FILE TO THAT RECORD
0322  0194 7E 006B R        JMP OPEN2       FINISH UP ERROR CHECK
0324                  * BUILD A NEW RANDOM-ACCESS FILE
0325                  *
0326                  *       CALL BUILD(BFCB,BNAME,BRECNO,BRECSZ)
0327                  *
0328                  * BFCB=ADDRESS OF FCB
0329                  * BNAME=ADDRESS OF FILE SPECIFICATION
0330                  * BRECNO=NUMBER OF RECORDS IN FILE
0331                  * BRECSZ=NUMBER OF BYTES IN EACH RECORD
0332                  *
0333  0197 BD 0000 R BUILD  JSR .RTBL
0334  019A F0                 FCB 240         HANDLE STRUBAL+ ARG  LINKAGE
0335  019B 20 0A              BRA *+12
0336  019D 0002        BFCB   RMB 2           ADDRESS OF FCB
0337  019F 0002        BNAME  RMB 2           ADDRESS OF FILE SPEC.
0338  01A1 0002        BRECNO RMB 2           ADDRESS OF RECORD COUNT
0339  01A3 0002        BRECSZ RMB 2           ADDRESS OF RECORD SIZE
0340  01A5 FFFF               FDB $FFFF
0341  01A7 7F 0067 C          CLR .ERR
0342  01AA 7F 0068 C          CLR .ERR+1      NO ERRORS YET
0343  01AD FE 019D R          LDX BFCB        POINT TO FCB
0344  01B0 86 44              LDA A #'D
0345  01B2 A7 02              STA A FCBGDT,X  "DSK" IN FCB
0346  01B4 86 53              LDA A #'S
0347  01B6 A7 03              STA A FCBGDT+1,X
0348  01B8 86 4B              LDA A #'K
0349  01BA A7 04              STA A FCBGDT+2,X
0350  01BC C6 AA              LDA B #170
0351  01BE 3F                 SWI             CALL CP/68
0352  01BF 0A                 FCB 10          "ADDBX"
0353  01C0 3F                 SWI             CALL CP/68
0354  01C1 02                 FCB 2           "TXAB"
0355  01C2 FE 019D R          LDX BFCB        POINT TO FCB
0356  01C5 6F 05              CLR FCBSTA,X
0357  01C7 A7 07              STA A FCBDBA,X INIT  BUFFER POINTER
0358  01C9 E7 08              STA B FCBDBA+1,X
0359  01CB FE 019F R          LDX BNAME
0360  01CE DF 23              STX CUCHAR      INIT  NAME POINTER
0361  01D0 FE 019D R          LDX BFCB
0362  01D3 3F                 SWI             CALL CP/68
0363  01D4 2C                 FCB 44          "FMTFCB"
0364  01D5 6D 05              TST FCBSTA,X    ERROR?
0365  01D7 27 03              BEQ *+5         NO
0366                  *
0367  01D9 7E 0094 R          JMP SETERR      YES
0368                  *
0369  01DC FE 01A1 R          LDX BRECNO
0370  01DF A6 00              LDA A 0,X       PUT RECNO  INTO FCB
0371  01E1 E6 01              LDA B 1,X
0372  01E3 FE 019D R          LDX BFCB
0373  01E6 A7 2A              STA A FCBRNM,X
0374  01E8 E7 2B              STA B FCBRNM+1,X
0375  01EA FE 01A3 R          LDX BRECSZ
0376  01ED A6 00              LDA A 0,X       PUT RECSIZ.  INTO FCB
0377  01EF E6 01              LDA B 1,X
0378  01F1 FE 019D R          LDX BFCB
0379  01F4 A7 2C              STA A FCBRSZ,X
0380  01F6 E7 2D              STA B FCBRSZ+1,X
0381  01F8 BD 0003 R          JSR CREATE      CREATE NEW FILE
0382  01FB 7E 006B R          JMP OPEN2       FINISH UP
0384                  * ENLARGE A RANDOM-ACCESS FILE
0385                  *
0386                  *       CALL ENLARGE(EFCB,ERECNO)
0387                  *
0388                  * EFCB=ADDRESS OF FCB
0389                  * ERECNO=NUMBER OF NEW RECORDS TO ADD
0390                  *
0391  01FE BD 0000 R ENLARGE JSR .RTBL
0392  0201 F0                 FCB 240         HANDLE STRUBAL+ ARG  LINKAGE
0393  0202 20 06              BRA *+8
0394  0204 0002        EFCB   RMB 2           ADDRESS OF FCB
0395  0206 0002        ERECNO RMB 2           ADDRESS OF RECORD COUNT
0396  0208 FFFF               FDB $FFFF
0397  020A 7F 0067 C          CLR .ERR
0398  020D 7F 0068 C          CLR .ERR+1      NO ERRORS YET
0399  0210 FE 0204 R          LDX EFCB        POINT TO FCB
0400  0213 A6 1D              LDA A FCBTYP,X
0401  0215 81 02              CMP A #2        RANDOM-ACCESS?
0402  0217 27 07              BEQ ENLRG2      YES
0403                  *
0404  0219 86 12              LDA A #18       NO, ERROR 18
0405  021B A7 05              STA A FCBSTA,X
0406  021D 7E 009B R          JMP SETER2
0407                  *
0408  0220 FE 0206 R ENLRG2 LDX ERECNO       PUT RECNO  INTO FCB
0409  0223 A6 00              LDA A 0,X
0410  0225 E6 01              LDA B 1,X
0411  0227 FE 0204 R          LDX EFCB
0412  022A A7 2E              STA A FCBRCD,X
0413  022C E7 2F              STA B FCBRCD+1,X
0414  022E BD 0015 R          JSR EXPAND      NOW ENLARGE FILE
0415  0231 7E 006B R          JMP OPEN2       FINISH UP
0417                  * DELETE EXISTING FILE (RANDOM OR SEQUENTIAL)
0418                  *
0419                  * CALL DELETE(DFCB,DNAME)
0420                  *
0421                  * DFCB=ADDRESS OF FCB
0422                  * DNAME=ADDRESS OF FILE SPECIFICATION
0423                  *
0424  0234 BD 0000 R DELETE JSR  RTBL
0425  0237 F0                 FCB 240         HANDLE STRUBAL+ ARG. LINKAGE
0426  0238 20 06              BRA *+8
0427  023A 0002        DFCB   RMB 2           ADDRESS OF FCB
0428  023C 0002        DNAME  RMB 2           ADDRESS OF FILE SPEC.
0429  023E FFFF               FDB $FFFF
0430  0240 7F 0067 C          CLR  ERR
0431  0243 7F 0068 C          CLR  ERR+1      NO ERRORS YET
```

```
0432   0246 FE 023C R      LDX DNAME
0433   0249 DF 23          STX CUCHAR     INIT  SPEC  POINTER
0434   024B FE 023A R      LDX DFCB
0435   024E 86 44          LDA A #'D      "DSK" IN FCB
0436   0250 A7 02          STA A FCBGDT,X
0437   0252 86 53          LDA A #'S
0438   0254 A7 03          STA A FCBGDT+1,X
0439   0256 86 4B          LDA A #'K
0440   0258 A7 04          STA A FCBGDT+2,X
0441   025A C6 2A          LDA B #42
0442   025C 3F             SWI            CALL CP/68
0443   025D 0A             FCB 10         "ADDBX"
0444   025E 3F             SWI            CALL CP/68
0445   025F 02             FCB 2          "TXAB"
0446   0260 FE 023A R      LDX DFCB
0447   0263 A7 07          STA A FCBDBA,X INIT. BUFFER POINTER
0448   0265 E7 08          STA B FCBDBA+1,X
0449   0267 3F             SWI            CALL CP/68
0450   0268 2C             FCB 44         "FMTFCB"
0451   0269 6D 05          TST FCBSTA,X   ERROR?
0452   026B 27 03          BEQ *+5        NO
0453                 *
0454   026D 7E 0094 R      JMP SETERR     YES
0455                 *
0456   0270 3F             SWI            CALL CP/68
0457   0271 1C             FCB 28         "DELETE"
0458   0272 7E 006B R      JMP OPEN2      FINISH UP
0459                 *
0460                       END
```

```
 CURIO  0069 C
 EOF    0065 C
 ERR    0067 C
 RTBL   0000 RX
BFCB    019D R
BLANK   0000 C
BNAME   019F R
BRECNO  01A1 R
BRECS7  01A3 R
BUFLD   0197 RN
CLASS   0026
CLOSE   00A4 RN
CLOSE2  00B4 R
CLOSE3  00B6 R
CP68DR  0000 RN
CREATE  0003 RX
CUCHAR  0023
DELETE  0234 RN
DESCRA  0020
DFCB    023A R
DNAME   023C R
EFCB    0204 R
ENLARG  01FE RN
ENLRG2  0220 R
ERECNO  0206 R
EXPAND  0015 RX
FCBACS  001E
FCBDBA  0007
FCBDTT  0006
FCBGDT  0002
FCBPOS  0030
FCBRCD  002E
FCBRNM  002A
FCBRSZ  002C
FCBSCF  0029
FCBSTA  0005
FCBTYP  001D
GETB    00D5 RN
GETB0   00EE R
GETB00  00F1 R
GETB1   00FA R
GETB2   00FE R
GETB3   0104 R
LBYTE   012D R
LFCB    0129 R
LOC2    0147 R
LOCATE  0123 RN
LRECNO  012B R
MONITOR 00D3 RN
OPEN    0018 RN
OPEN1   0066 R
OPEN2   006B R
OPEN3   0070 R
OPEN4   007B R
OUTB    0108 RN
OUTB2   011B R
OUTB3   011D R
PFCB    0167 R
PLACE   0161 RN
PLACE2  0183 R
```

```
POSITI  0012 RX
PREC    0169 R
RC      0025
RCLOSE  0009 RX
REWZ    00CA R
REWIND  00BB RN
ROPEN   0006 RX
RREAD   000C RX
RWRITE  000F RX
SETER2  009B R
SETERR  0094 R
```

A

BRCK 117
BS 141
COS 164
DD 160
DD16 105
DDR1 114
DDR2 114
DDR3 114
DDR4 115
DDR5 115
DDR6 115
DDR7 115
DDR8 116
DDR9 116
DDZ 140
DRINT 116
NDZ 139
SCII 154
SIN 164
SSEMBLY 6
SSEMBLY-NSEVL 104
TAN 164

B

ADERR 148
INARY 88
INOP 88
LOCK 34
LOKPR 127
RTABL 138
YTE 28,35
YTEPR 90

C

ALL 17,35
ALLPR 128
END 36
ENDPR 108
HKBIN 88
HRTAB 79
LOSE 23,152
LOSE(FCBNAM) 36
LOSPR 110
MPR 125
OMMENTS 36
OMMON 26,134
OMPAR 83,141
OMPARISON 125
OMPILATION 5
CONVERT 26,36
CONVPR 91
COS 163
CRLF 145
CRUTCH-CODING 4
CSECPR 108
CSECT 37
CURIOB 150
CVBTD 107
CVDB 105
CVHB 105,146
CVHBS 105,147

D

DECIM 147
DEFINE 37
DEFPR 109
DEFS 113
DELAY 29,37,104
DELSYM 99
DEND 38
DENDPR 108
DIGITS 38
DIGSET 38,76,156,160
DIMPR 109
DIV 140,161
DIV16 106
DIVERR 148
DSCAN 95
DSECPR 108
DSECT 39
DSTPR 109
DSTRING 40
DUMMY 27

E

END 40
ENDSCN 96
ENTRY 135
EVADIM 132
EVALS 101
EXOR 139
EXP 141
EXTERNAL 135
EXTRCT 163

F

FABS 158
FADD 161
FASCII 154
FBINOP 90
FCOMPR 158
FDIV 162
FEND 41
FEXP 166
FFDREG 160
FINISH 139
FINSHS 140
FIX 158
FLFEED 151
FLOAT 158
FLOATING-POINT 90
FLOT 4
FLOTPNT 159
FLOTPR 88
FLREAD 150
FMUL 162
FNDDIM 131
FNEG 159
FOR 41
FOR/NEXT 18
FORPR 85
FPRINT 156
FREAD 157
FSCAN 96
FSIGN 158
FSPACE 152
FSUB 162
FTAREG 159
FTCREG 159
FUNCTION 41
FUNOP 87
FUNOP2 87
FVAL 157
FWRITE 151

G

GETAD 29,42,89
GETADR 89
GETB 76,153
GCHRTB 98
GOSBPR 129
GOSUB 16,42
GOTO 43
GOTOPR 129
GTPRO 130

H

HASH 99
HSCAN 96

I

IF 44
IFPR 110
ILLERR 149
INCHAR 157
INEDIT 146
INEEE 145
INITIO 152
INPR 126
INPUT 21,44,146
INTEGER 45
INTPR 109
INXCK 117
IOCBPR 113
IREAD 150
IWRITE 151

L

LBLCK 132
LCLCN 118
LCN2 117
LCN3 118
LCNAB1 117
LCNAB2 118
LCNAB3 118
LEN 153
LENERR 148
LENGPR 93
LENGTH 46
LET 46
LETPR 83
LFEED 146
LINCK 106
LKPSYM 98
LOG10 165
LPON 76

M

MACMOV 123
MACPSH 81
MACPUL 81
MAIN 79
MID 154
MNLKP 100
MNTAB 78
MOD 141
MONTOR 76
MOVER 160

PRINT 156
PY16 106
JL 161
JLT 140

N

ATLOG 165
EGATE 141
EGERR 148
EXT 46
EXTPR 85
OT 138
SCAN 96
XTOK 95

O

N 16,47
NPR 128
PEN 23,47,152
PENPR 110
R 139
JTB 76,153
JTBIN 132
JTBNR 132
JTCMP 82
JTCR 82
JTEEE 145
JT4H 146
JTHL 132,144
JTHR 132,144
JTL 107
JTNAM 82
JTPUT 147
JTS 145
VERR 148
VFLW 106

P

AK 161
ARAM 156
ARMVE 82
BLOCK 120
DATA 145
DATA1 145
EEK 142
2ERR 117
ERR 147
OCMN 118
POEND 119
POENT 120
POEQU 120
POEXT 121
POFCB 121
POFCC 121
POFDB 122
POFFP 122
POFUNC 122
POIFC 122
POKE 29,48,142
POKEPR 113
POMAC 123
PONAM 123
PONIFC 123
POP 30,48
POPAG 124
POPF 143
POPI 143
POPPR 86
POPS 144
PORMB 124
POSET 125
PRESCN 127
PRINT 21,48
PRINTE 107
PRINTL 106
PRNTPR 109
PROCEDURE 28,49
PROCPR 129
PSHIF 123
PULIF 124
PUSH 30,50
PUSHF 143
PUSHI 143
PUSHPR 86
PUSHS 143

Q

QPR1 112
QUOTE 155
QUOTPR 112
QWRITE 151

R

RAND 142
RDLINE 80
RDMAC 81
READ 20,50
READPR 110
RETPR 125
RETURN 17,50
REWIND 24,51,152
REWPR 110
RMBOUT 124
.RTBL 89,137
RUNTIM 137
RUNTIME 4
RUNTM 4

S

SCIF 4
SCIFUN 162
SCHCHR 97
SERIES 163
SETUP 139
SETUPS 140
SIGN 141
SIN 164
SPACER 107
SPACES 155
SREAD 150
SRECRD 150
SSETUP 91
STAKDN 101
STAKUP 100
STOP 51,144
STOPPR 126
STOSYM 98
STR1 90
STRCMP 94,142
STRDIM 84
STRIN 155
STRING 51
STRP 92
STRPR 91
STRQ 92
STROUT 155
SUB 161
SUB16 105
SUBZ 140
SWRITE 151
SYMCMP 99
SYMMOD 99
SYNTAX 3

T

TAN 163
TENEXP 166
TPCHAR 82
TRUNER 148

U

UNOP 87
UPSCLR 100

V

VAL 153
VALUE 52
VALUPR 92

W

WHILE 18,52
WHILPR 127
WRITE 22,53
WRITPR 111
WRTCHR 82

X

XFER 154
XREF 97
XTOY 166
XTOYPR 89
XTRACT 54
XTRPR 93

Z

ZMOVER 144