LIB-80 is the object time library manager for FORTRAN-80. To run LIB-80, type:

```
A>LIB <carriage return>
```

LIB-80 will respond with a prompt of "*". Each command in LIB-80 either lists information about a library or adds new modules to the library under construction.

A number of switches are used to control LIB-80 operation. These switches are always preceded by a slash:

- /O Octal - set Octal typeout mode for /L command
- /H Hex - set Hex typeout mode for /L command (default).
- /U List the symbols which would remain undefined on a search through the file specified.
- /L List the modules in the files specified and symbol definitions they contain.
- /C (Create) Throw away the library under construction and start over.
- /E Exit to CP/M. The library under construction (.LIB) is revised to .REL and any previous copy is deleted.
- /R Rename - same as /E but does not exit to CP/M on completion.

A module is typically a FORTRAN subprogram, main program or a MACRO-80 assembly that contains ENTRY statements.

The primary function of LIB-80 is to concatenate modules in .REL files to form a new library. In order to extract modules from previous libraries or .REL files, a powerful syntax has been devised to specify ranges of modules within a .REL file.

The simplest way to specify a module within a file is simply to use the name of the module:

```
SIN
```

But a relative quantity plus or minus 255 may also be used. For example,

```
SIN+1
```

specifies the module after SIN and

SIN-1

specifies the one before it.

Ranges of modules may also be specified by using two dots:

..SIN means all modules up to and including SIN.

SIN.. means all modules for SIN to the end of the file.

SIN..COS means SIN and COS and all the modules in between.

Ranges of modules and relative offsets may also be used in combination:

SIN+1..COS-1

To select a given module from a file use the name of the file followed by the module(s) specified enclosed in angle brackets and separated by commas:

FORLIB <SIN..COS>

or

MYLIB.REL <TEST>

or

BIGLIB.REL <FIRST,MIDDLE,LAST>

etc.

If no modules are selected from a file, then all the modules in the file are selected:

TESTLIB.REL

Each command to LIB-80 consists of an optional destination filename which sets the name of the library being created, followed by an equal sign and the names of the modules specified separated by commas. The default destination filename is FORLIB.LIB. Examples:

*NEWLIB=FILE1 <MOD2>, FILE3,TEST

*SIN,COS,TAN,ATAN

Any command specifying a set of modules concatenates the modules selected onto the end of the last destination filename given. Therefore,

*FILE1,FILE2 <BIGSUB>, TEST

is equivalent to

```
*FILE1
*FILE2 <BIGSUB>
*TEST
```

To list the contents of a file in cross reference format, use/L:

```
*FORLIB/L
```

It is important to place a module after another module that refers to a global symbol in the second module. Otherwise, LINK-80 will not satisfy the request to load the symbol on its single pass through the library.

Use /U to list the symbols which could be undefined in a single pass through a library. If a module in the library makes a backward reference to a symbol in another module, /U will list that symbol. Example:

```
*SYSLIB/U
```

NOTE: Since certain modules in the standard FORTRAN system are always force-loaded, they will be listed as undefined by /U but will not cause a problem when loading FORTRAN programs.

Listings are currently always sent to the terminal; use control-P to send the listing to the printer.

Sample LIB session:

```
A>LIB

*TRANLIB=SIN,COS,TAN,ATAN,ALOG
*EXP
*TRANLIB.LIB/U
*TRANLIB.LIB/L
  .
  .
  .
(List of symbols in TRANLIB.LIB)
  .
  .
  .
*/E
A>
```

Summary of switches and syntax:

/O Octal - set listing radix
/H Hex   - set listing radix
/U List undefineds
/L List cross reference
/C Create - start LIB over
/E Exit - Rename .LIB to .REL and exit
/R Rename - Rename .LIB to .REL

module::=module name {+ or - number}

module sequence ::=

module | ..module | module.. | module1..module2

file specification::=filename {<module sequence>{, module sequence}>}

command::= {library filename=} {list of file specifications} {list of switches}

Microsoft 8080 FORTRAN IV
FORTRAN-80 Reference Manual
Addendum to Appendix E, pages 100-101
April, 1978

RANDOM ACCESS DISK FILES

The CP/M and ISIS-II versions of FORTRAN-80 now provide random disk accessing, i.e., a record may be specified with a disk READ or WRITE.

The record number is specified by using the REC=n option in the READ or WRITE statement. For example:

```
I = 10
WRITE (6,20,REC=1,ERR=50) X,Y,Z
.
.
.
```

This program segment writes record 10 on LUN 6. If a previous record 10 exists, it is written over. If no record 10 exists, the file is extended to create one. Any attempt to read a non-existent record results in an I/O error.

The record length of any file accessed randomly is assumed to be 128 bytes (1 sector). Therefore, it is recommended that any file you wish to read randomly be created via FORTRAN (or Microsoft BASIC) random access statements.

Random access files may be created via FORTRAN programs, by using either binary or formatted WRITE statements. If the WRITE statement does not cause enough data to be transferred to fill the record (128 bytes), then the end of the record is filled with zeros (NULL characters).

ISIS-II DISK FILES

Disk files may now be created and accessed by FORTRAN-80 programs running under ISIS-II. Files are accessed either sequentially or randomly, as described in Appendix E of the FORTRAN-80 manual. The only programming difference under ISIS-II is that the parameters required by the OPEN subroutine have been altered slightly from the form described in

Appendix E. The form of an OPEN call under ISIS-II is

```
CALL OPEN (LUN, Filename)
```

where:

LUN = a Logical Unit Number to be associated with the file (must be an integer constant or variable with a value between 1 and 10).

Filename = an ASCII name which the operating system will associate with the file. The Filename should be a Hollerith or Literal constant, or a variable or array name where the variable or array contains the ASCII name. The filename should be in the form normally required by ISIS-II, i.e., a device name surrounded by colons, followed by a name of up to 6 characters, a period, an extension of up to 3 characters, and a space (or other non-alphanumeric character). The Filename must be terminated by a non-alphanumeric character.

The following are examples of valid OPEN calls:

```
CALL OPEN (6, ':F1:FOO.DAT ')

CALL OPEN (1, ':F5:TESTFF.TMP ')

CALL OPEN (10, ':F0:A.DAT ')

CALL OPEN (4, ':F3:A.B ')
```

Microsoft 8080 FORTRAN IV

FORTRAN-80 Reference Manual

Addendum

May, 1978

APPENDIX F

FORTRAN-80 Library Subroutines

The FORTRAN-80 library contains a number of subroutines that may be referenced by the user from FORTRAN or assembly programs. In the following descriptions, $AC refers to the floating accumulator; $AC is the address of the low byte of the mantissa. $AC+3 is the address of the exponent. $DAC refers to the DOUBLE PRECISION accumulator; $DAC is the address of the low byte of the mantissa. $DAC+7 is the address of the DOUBLE PRECISION exponent.

All arithmetic routines (addition, subtraction, multiplication, division, exponentiation) adhere to the following calling conventions.

1. Argument 1 is passed in the registers:
   Integer in [HL]
   Real in $AC
   Double in $DAC

2. Argument 2 is passed either in registers, or in memory depending upon the type:

   a) Integers are passed in [HL], or [DE] if [HL] contains Argument 1.

   b) Real and Double Precision values are passed in memory pointed to by [HL].
   ([HL] points to the low byte of the mantissa.)

The following arithmetic routines are contained in the Library:

Function	Name	Argument 1 Type	Argument 2 Type
Addition	$AA	Real	Integer
	$AB	Real	Real
	$AQ	Double	Integer
	$AR	Double	Real
	$AU	Double	Double
Division	$D9	Integer	Integer
	$DA	Real	Integer
	$DB	Real	Real
	$DQ	Double	Integer
	$DR	Double	Real
	$DU	Double	Double
Exponentiation	$E9	Integer	Integer
	$EA	Real	Integer
	$EB	Real	Real
	$EQ	Double	Integer
	$ER	Double	Real
	$EU	Double	Double
Multiplication	$M9	Integer	Integer
	$MA	Real	Integer
	$MB	Real	Real
	$MQ	Double	Integer
	$MR	Double	Real
	$MU	Double	Double
Subtraction	$SA	Real	Integer
	$SB	Real	Real
	$SQ	Double	Integer
	$SR	Double	Real
	$SU	Double	Double

Additional Library routines are provided for converting between value types. Arguments are always passed to and returned by these conversion routines in the appropriate registers:

Logical in [A]

Integer in [HL]

Real in $AC

Double in $DAC

Name	Function
$CA	Integer to Real
$CC	Integer to Double
$CH	Real to Integer
$CJ	Real to Logical
$CK	Real to Double
$CX	Double to Integer
$CY	Double to Real
$CZ	Double to Logical

Microsoft FORTRAN-80 User's Manual

Addendum to: SECTION 1
Compiling FORTRAN Programs

May, 1978

The following additions are to be made to Section 1 (Compiling FORTRAN Programs) of the Microsoft FORTRAN-80 User's Manual.

1. Page 6

Add to Section 1.1.2 FORTRAN-80 Compilation Switches

Switch	Action
M	Specifies to the compiler that the generated code should be in a form which can be loaded into ROMs. When a /M is specified, the generated code will differ from normal in the following ways: 1. FORMATs will be placed in the program area, with a "JMP" around them. 2. Parameter blocks (for subprogram calls with more than three parameters) will be initialized at runtime, rather than being initialized by the loader.

NOTE

If a FORTRAN program is intended for ROM, the programmer should be aware of the following ramifications:

1. DATA statements should not be used to initialize RAM. Such initialization is done by the loader, and will therefore not be present at execution. Variables and arrays may be initialized during execution via assignment statements, or by READing into them.

2. FORMATs should not be read into during execution.

3. If the standard library I/O routines are used, DISK files should not be OPENed on any LUNs other than 6, 7,8,9,10. If other LUNs are needed for Disk I/O, $LUNTB should be recompiled with the appropriate addresses pointing to the Disk driver routine.

A library routine, $INIT, sets the stack pointer at the top of available memory (as indicated by the operating system) before execution begins.

The calling convention is:

```
LXI     B,<return address>
JMP     $INIT
```

If the generated code is intended for some other machine, this routine should probably be rewritten.

Microsoft FORTRAN-80 User's Manual

Addenda to: SECTION 2
Linking FORTRAN Programs

May, 1978

The following additions are to be made to Section 2 (Linking FORTRAN Programs) of the Microsoft FORTRAN-80 User's Manual.

1. Page 11-12

Add to Section 2.1.2 LINK-80 Switches

/U and /M also print the origin and end of the program and data area in addition to selected globals. Example:

```
DATA      100      200
PROGRAM   1000     2000
```

The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

Switch	Action
N	If a <filename>/N is specified, the program will be saved on disk under the selected name (with a default extension of .COM for CP/M) when a /E or /G is done. A jump to the start of the program is inserted if needed so the program can run properly (at 100H for CP/M).
P and D	/P and /D allow the origin(s) to be set for the next program loaded. /P and /D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is /P:<address> or /D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix for non-MITS versions is hex. /O sets radix to octal; /H to hex.) LINK-80 does a default /P:<link origin>+3 (i.e., 103H for CP/M and 4003H for ISIS) to leave room for the jump to the start address. If no /D is given, programs load as usual, except the area base is

settable. If a /D is given, all Data and Common areas are loaded starting at the data origin and the program area at the program origin. Example:

```
*/P:200,FOO
DATA    200     300
*/R
*/P:200 /D:400,FOO
DATA    400     480
PROG    200     280
```

2. Page 13-14

Add to Section 2.3 Format of LINK Compatible Object Files

Loader type 9 is now in use; it is external + offset. Type 9 has only an A field, there is no B field as previously documented. The value for type 9 will be added to the two bytes starting at the current location counter. This addition is done after a /E or /G is given, so unless undefineds remain, the effect is external + offset.

This type can also be used to add program and data relatives or almost any other combination of relocation types. The assembler, however, only handles the case with externals.

3. Page 15

Add to Section 2.4 LINK-80 Error Messages

?Out of Memory	has replaced ?Fatal Table Collision
?<file> Not Found	has replaced ?File Not Found. The name of the file not found is printed.
%Overlaying [Program] [Data] Area	A /D or /P will cause already loaded data to be destroyed.

?Intersecting [Program] [Data] Area

The program and data area intersect and an address or external chain entry is in this intersection. The final value cannot be converted to a current value since it is in the area intersection.

?Start Symbol - <name> - Undefined

After a /E: or /G: is given, the symbol specified was not defined.

Origin [Above] [Below] Loader Memory, Move Anyway (Y or N)?

After a /E or /G was given, either the data or program area has an origin or top which lies outside loader memory (i.e., loader origin to top of memory). If a Y <cr> is given, LINK-80 will move the area and continue. If anything else is given, LINK-80 will exit. In either case, if a /N was given, the image will already have been saved.

?Can't Save Object File

A disk error occurred when the file was being saved.

Add Section 2.5 Program Break Information

LINK-80 stores the first free location in a symbol called $MEMRY if that symbol has been defined by a program loaded. $MEMRY is set to the top of the data area +1.

NOTE

If /D is given and the data origin is less than the program area, the user must be sure there is enough room to keep the program from being destroyed. This is particularly true with the disk driver for FORTRAN-80 which uses $MEMRY to allocate disk buffers and FCB's.

Microsoft FORTRAN-80 User's Manual

Addenda to: SECTION 3
The MACRO-80 Assembler

May, 1978

The following additions and corrections are to be made to Section 3 (The MACRO-80 Assembler) of the Microsoft FORTRAN-80 User's Manual.

1. Page 16

   Add to Section 3.1.2 MACRO-80 Switches

   /C is the Cross Reference Switch.

2. Page 17

   Add to Section 3.3.2 Constants

   e. Binary: Numbers consisting of a string of binary digits (0's and 1's) followed by a B. (e.g., 101011B)

3. Page 17

   Correction to Section 3.3.3 Labels

   Labels need not begin in column 1.

4. Page 17-18

   Replace Section 3.3.5 Address Expressions with the following:

   3.3.5 Address Expressions

   An address expression consists of a name or a constant or an address expression + or - an address expression. An address expression uses the current assigned address of a name or the 16-bit value of a constant to form a 16-bit value which, after the expression is evaluated, is truncated to the field size required by the operator.

5. Page 18

   Add to Section 3.3.7 Statement Form

   Statements may begin in column 1.

6. Page 18

Add Section 3.3.8 Expression Evaluation

Operator precedence during expression evaluation is as follows:

```
Parenthesized expressions
HIGH, LOW
*, /, MOD, SHL, SHR
+, - (unary and binary)
Relational Operators EQ, LT, LE, GT, GE, NE
Logical NOT
Logical AND
Logical OR, XOR
```

The Relational, Logical and HIGH/LOW operators must be separated from their operands by at least one space.

Byte Isolation Operators

The byte isolation operators are as follows:

HIGH	Isolate the high order 8 bits of a 16-bit value
LOW	Isolate the low order 8 bits of a 16-bit value

Example:

```
IF HIGH VALUE  EQ 0
```

The above IF pseudo-op determines whether the high order byte of VALUE is zero.

Relational Operators

The relational operators are as follows:

EQ	Equal
NE	Not equal
LT	Less than
LE	Less than or equal
GT	Greater than
GE	Greater than or equal

These operators yield a true or false result. They are commonly used in conditional IF pseudo-ops. They must be separated from their operands by spaces. Example:

```
IF LABEL1 EQ LABEL2
```

The above pseudo-op tests the values of LABEL1 and LABEL2 for equality. If the result of the comparison is true, the assembly language code following the IF pseudo-op is assembled, otherwise the code is ignored.

7. Page 18

Add Section 3.3.8 Opcodes as Operands

8080 opcodes are valid one-byte operands. Note that only the first byte is a valid operand. For example:

```
MVI     A,(JMP)
ADI     (CPI)
MVI     B,(RNZ)
CPI     (INX H)
ACI     (LXI B)
MVI     C,(MOV A,B)
```

Errors will be generated if more than one byte is included in the operand -- such as (CPI 5), (LXI B,LABEL1) or (JMP LABEL2).

Opcodes used as one-byte operands need not be enclosed in parentheses.

8. Page 19

Add to Section 3.3.4 Define Word

Example:

```
DW    'AB'
```

Two-byte values are stored in memory in low order byte/high order byte sequence. The ASCII code representation for character B is stored, then the character A is stored.

On the object code listing however, the printout for all two-byte values is in high order byte/low order byte sequence, for easier reading.

9. Page 22

Add Section 3.4.16 Memory Segment Specification

It is possible to specify that sections of a program be loaded in absolute, code relative or data relative segments of memory. The pseudo-ops are:

ASEG	For loading in an absolute segment of memory
DSEG	For loading in a data relative segment of memory
CSEG	For loading in a code relative segment of memory

One of the possible uses of these pseudo-ops is to specify RAM and ROM segments of memory. The data relative segment would be RAM, and the code relative segment would be ROM.

After an ASEG, CSEG or DSEG pseudo-op is encountered, all following code is loaded in that area until a subsequent ASEG, CSEG or DSEG pseudo-op is encountered.

If none of these three pseudo-ops is specified, the default condition is to load everything code relative.

Additional flexibility in relocating code is possible through use of the ORG pseudo-op, which sets the value of the appropriate program counter. For example:

```
DSEG        ; Sets the data relative program
ORG 50      ; counter to a value of 50
```

NOTES

1. The Intel operands PAGE and INPAGE will generate expression errors when used with CSEG or DSEG pseudo-ops. These errors are warnings; the assembler ignores the operands.

2. In version 3.0 of the MACRO-80 Assembler, references to a particular external symbol may not be made in more than one memory segment. For example, an external symbol EXT1 might be referenced in the code relative segment, external symbols EXT3, EXT4 might be referenced in the data relative segment, but none could be referenced in more than one memory segment.

Refer to Section 2, Linking FORTRAN Programs, to determine how these segments are placed in specific areas of memory.

10. Page 24

Add Section 3.8 Cross Reference Facility

The Cross Reference Facility is invoked by typing CREF80. In order to generate a cross reference listing, the assembler must output a special listing file with embedded control characters. The MACRO-80 command string tells the assembler to output this special listing file. An additional switch has been introduced, /C, the cross reference switch. When the /C switch is encountered in a MACRO-80 command string, the assembler opens a .CRF file instead of a .LST file.

Examples:

*=TEST/C	Assemble file TEST.MAC and create object file TEST.REL and cross reference file TEST.CRF
*T,U=TEST/C	Assemble file TEST.MAC and create object file T.REL and cross reference file U.CRF.

When the assembler is finished, it is necessary to call the cross reference facility by typing CREF80. The command string is:

```
*listing file=source file
```

The default extension for the source file is .CRF. The /L switch is ignored, and any other switch will cause an error message to be sent to the terminal. Possible command strings are:

*=TEST	Examine file TEST.CRF and generate a cross reference listing file TEST.LST.
*T=TEST	Examine file TEST.CRF and generate a cross reference listing file T.LST.

Cross reference listing files differ from ordinary listing files in that:

1. Each source statement is numbered.

2. At the end of the listing, variable names appear in alphabetic order along with the numbers of the lines on which they are referenced or defined.

FORTRAN-80

Overview

Microsoft's FORTRAN-80 package provides new capabilities for users of 8080 and Z-80 based microcomputer systems. FORTRAN-80 is comparable to FORTRAN compilers on large mainframes and minicomputers. All of ANSI Standard FORTRAN X3.9-1966 is included except the COMPLEX data type. -Therefore, users may take advantage of the many applications programs already written in FORTRAN.

Versions of FORTRAN-80 for the CP/M, ISIS-II, DTC Microfile and MITS DOS floppy disk operating systems are available off the shelf. Other versions will be prepared based upon user demand.

Relocatable Code and Library Features

FORTRAN-80 is unique in that it provides a microprocessor FORTRAN and assembly language development package that generates relocatable object modules. This means that only the subroutines and system routines required to run FORTRAN-80 programs are loaded before execution. Subroutines can be placed in a system library so that users develop a common set of subroutines that are used in their programs. Also, if only one module of a program is changed, it is necessary to re-compile only that module.

The standard library of subroutines supplied with FORTRAN-80 includes:

ABS	IABS	DABS	AINT
INT	IDINT	AMOD	MOD
AMAX0	AMAX1	MAX0	MAX1
DMAX1	AMIN0	AMIN1	MIN0
MIN1	DMIN1	FLOAT	IFIX
SIGN	ISIGN	DSIGN	DIM
IDIM	SNGL	DBLE	EXP
DEXP	ALOG	DLOG	ALOG10
DLOG10	SIN	DSIN	COS
DCOS	TANH	SQRT	DSQRT
ATAN	DATAN	ATAN2	DATAN2
DMOD	PEEK	POKE	INP
OUT			

The library also contains routines for 32-bit and 64-bit floating point addition, subtraction, multiplication, division, etc. These routines are among the fastest available for performing these functions on the 8080.

Enhancements

The FORTRAN-80 compiler has a number of enhancements of the ANSI Standard:

1. LOGICAL variables which can be used as integer quantities in the range +127 to -128.

2. LOGICAL DO loops for tighter, faster execution of small valued integer loops.

3. Mixed mode arithmetic.

4. Hexadecimal constants.

5. Literals and Holleriths allowed in expressions.

6. Logical operations on integer data. .AND., .OR., .NOT., .XOR. can be used for 16-bit or 8-bit Boolean operations.

7. READ/WRITE End of File or Error Condition transfer. END=n and ERR=n (where n is the statement number) can be included in READ or WRITE statements to transfer control to the specified statement on detection of an error or end of file condition.

8. ENCODE/DECODE for FORMAT operations to memory.

FORTRAN-80 Compiler Characteristics

The FORTRAN-80 compiler can compile several hundred statements per minute in a single pass and needs less than 24K bytes of memory to compile most programs. Any extra available memory will be used by the compiler for extended optimizations.

In spite of its small size, the FORTRAN-80 compiler optimizes the generated object code in several ways:

1. Common subexpression elimination. Common subexpressions are evaluated once, and the value is substituted in later occurrences of the subexpression.

2. Peephole Optimization. Small sections of code are replaced by more compact, faster code in special cases. Example: I=I+1 uses an INX H instruction instead of a DAD.

3. Constant folding. Integer constant expressions are evaluated at compile time.

4. Branch Optimizations. The number of conditional jumps in arithmetic and logical IFs is minimized.

Long descriptive error messages are another feature of the compiler. For instance:

? Statement unrecognizable

is printed if the compiler scans a statement that is not an assignment or other FORTRAN statement. The last twenty characters scanned before the error is dectected are also printed.

The compiler generates a fully symbolic listing of the machine language being generated. At the end of the listing, the compiler produces an error summary and tables showing the addresses assigned to labels, variables and constants.

Assembler, Linker and Library Manager

A relocating assembler (MACRO-80), relocating linking loader (LINK-80) and a library manager (LIB-80) are included in the FORTRAN-80 package.

The relocating assembler is compatibie with INTEL's assembler, except MACRO capability is not provided. The assembler uses approximately 7K bytes of memory.

LINK-80, the relocating loader, resolves internal and external references between the object modules loaded. LINK-80 also performs library searches for system subroutines and generates a load map of memory showing the locations of the main program, subroutines and COMMON areas. LINK-80 requires approximately 4K bytes of memory.

LIB-80, the library manager, allows the user to customize libraries of object modules. LIB-80 can be used to insert, replace or delete object modules within a library, or create a new library from scratch. LIB-80 commands can also list the modules in the library and the symbol definitions they contain. LIB-80 requires approximately 4K of memory and uses the rest of memory as a buffer for its editing operations.

Custom I/O Drivers

Users may write non-standard I/O drivers for each Logical Unit Number, making the task of interfacing non-standard devices to FORTRAN programs a straightforward one.

Future Extensions

During the first quarter of 1978 MACRO capability will be added to the assembler, and LINK-80 will be modified to handle overlays.

Support

FORTRAN-80 users will receive quick turnaround on bug fixes, and new versions of FORTRAN-80 will be documented and distributed in an expedient manner.

Other Products

Microsoft's complete product line includes FOCAL for the 6502 and 6800, BASIC for the 6502 and 6800, and Altair (8080) BASIC. In addition, Microsoft has development software that runs on the DEC-10 for all of these microprocessors.

Pricing

Single Copy Prices:

FORTRAN-80 system (including documentation)	$500.00
FORTRAN-80, MACRO-80, LINK-80, LIB-80 manuals and system users guide	$ 20.00

OEM and dealer agreements are available upon request.

For more information contact:

Steve Wood
General Manager
Microsoft
300 San Mateo NE, Suite 819
Albuquerque, NM 87108
505-262-1486

Microsoft FORTRAN for CP/M

Review by Alan R. Miller, Contributing Editor

INTRODUCTION

Digital computers consist of many binary memory cells. Each of these cells has only two possible states that can be expressed as: TRUE or FALSE; logic 1 or logic 0; ON or OFF; etc. Many different computer languages have been developed to help programmers convert their ideas into this fundamental binary code.

The programmer encodes concepts into a SOURCE PROGRAM and then uses another computer program to convert this source program into a binary OBJECT PROGRAM that the computer can use. FORTRAN, COBOL, PASCAL, and ALGOL are some of the common computer languages that do this translation.

Each type of computer language is especially suitable for a particular task. A line of a FORTRAN source program such as:

```
Z(I) = SQRT(X(I)**2 + Y(I)**2)
```

may be translated into many lines of computer instructions by a compiler or interpreter. The source program is generally machine independent, so that a sorting program written in BASIC will run on a 6800 microcomputer as well as on an 8080.

In contrast to these high-level computer languages, assembly language is a low-level computer language that is more difficult to use, but produces shorter programs that run faster. And, unlike the higher-level languages, each line of an assembly-language source program will generally produce one computer instruction. Besides being more difficult to use, assembly language has another disadvantage. The source program is usable only with a specific type of computer. This means that a sorting routine written in 8080 assembly language will not run on a 6800 computer.

FORTRAN and BASIC languages are especially suitable for mathematical calculations (compared to COBOL, e.g., which is useful for the handling of business records). These high-level programs utilize a separate processing program to convert the original, user-written source program into the ultimate binary code needed by the computer.

BASIC source programs are commonly processed by a BASIC interpreter that resides in the computer memory along with the user's original source program. Each line of the source program is interpreted as it is encountered. Thus if the instruction:

```
Y(I) = X(I)
```

occurs in a loop that is executed 500 times, the same instruction is interpreted 500 times. Exceptions to this are BASIC-E and CBASIC. For these programs, a preprocessor first converts the source program into an intermediate program, which is then used by a run-time monitor.

ADVANTAGES OF FORTRAN

FORTRAN works a little differently. Each source program is first compiled into a relocatable binary object program. Then a linking loader program places the needed relocatable modules into memory in such a way that they can be run by themselves. No run-time monitor or interpreter need be present. The advantages of FORTRAN compared to BASIC are that less memory is required at run-time and the programs run faster (once they have been compiled) since only the ultimate binary code resides in memory. BASIC requires an 8K to 20K-byte run-time interpreter, as well as the original source code, with all of its comments, to be present in memory. FORTRAN is faster since the source program instructions don't have to be converted each time they are encountered.

A third advantage of FORTRAN, the localization of variables, may be the most important of all. If a subroutine is written to sort an array X of length N, it can readily be used to sort the array Z of length M.

```
DIMENSION X(30),Z(50)
  • •
CALL SORT (Z,M)
  • •
CALL SORT (Z,M)
  • •
SUBROUTINE SORT(X,N)
CIMENSION X(1)
  •
RETURN
```

By contrast, all variables are global in BASIC. This means that the array Z would have to be copied into the array X and N would have to be changed to M before the sort routine could be called a second time:

```
10 DIM X(30),Z(50)
20 M = 50 : N = 30
60 GOSUB 1000 : REM SORT X
100 N = M
110 FOR I=1 TO N
120 X(I) = Z(I)
130 NEXT I
140 GOSUB 1000 : REM SORT Z
  • •
1000 REM SORTING ROUTINE
```

And if the array X were needed later, it would have to be saved by the first copying into another array. Of course, there could be two sort routines, one for X and the other for Z, but this solution seems to be even worse.

Yet another advantage of FORTRAN is that there is a wealth of software available in the mathematics and engineering fields. For example, the IBM Scientific Subroutine Package contains routines for statistical analysis, curve fitting, and simultaneous solution of linear equations.

One of the greatest disadvantages of FORTRAN is that a program cannot be debugged as easily as a BASIC program. Typing a Control-C will stop a BASIC program while it is running. The user can then print the current values of any of the variables and even change the values. The program can then be resumed with a CONT command. This potential problem can be greatly reduced in FORTRAN, however, by programming in modular fashion. Thus an input subroutine, and output subroutine, a sort subroutine, etc., can each be written, compiled, run, and debugged if necessary. These modules can then be called by a main program when needed.

Another possible problem with FORTRAN is that no check is made to see if array indexes are out of range. Consider the following example:

```
DIMENSION X(10),Y(10)
• • •
Y(1) = 5
X(11) = 8
WRITE (1,101) Y(1)
```

The value of Y(1) has been changed from 5 to 8. Y(1) was initially set to 5, but the expression X(11) actually evaluates to 11 locations past the start of X. In this case it is also the address of Y(1). This potential problem is present in almost all versions of FORTRAN.

MICROSOFT FORTRAN

Microsoft, the organization that produced the MITS BASICs, and the TRS Level II BASIC, now offers a disk-based FORTRAN for the 8080 and Z-80 microprocessors. Versions are available for CP/M, Tektronix, ISIS-II, DTC Microfile, and MITS disk operating systems. A net memory size of 24K bytes, in addition to the disk operating system (DOS), is needed for the compiler. The CP/M version is reviewed in this article, but the other versions appear to be similar. The Microsoft CP/M version of FORTRAN is easily implemented since it uses the CP/M DOS primitives for all peripheral operations such as disk, console, list output, etc.

THE MANUALS

Three extensive and well-written manuals are provided with FORTRAN-80:

1. FORTRAN Reference Manual
   Language, grammar, and syntax
2. FORTRAN User Manual
   a. Use of compiler
   b. Run-time error messages
3. Utility Software Manual
   a. Assembler
   b. Linking loader
   c. Library manager
   d. Differences for versions

The total documentation runs for 152 pages and comes in an attractive and useful ring binder.

CREATING A FORTRAN SOURCE PROGRAM

FORTRAN source programs are generated and edited with the regular CP/M context editor:

```
B>A:ED SORT.FOR
```

The default extension is FOR. ANSI Standard FORTRAN X3.1966 is utilized except that there are no complex functions. There are also some additional nice features that are discussed later in this article.

The standard FORTRAN line of 80 characters has the format:

```
Column 1 - 5    Statement label, a decimal number
Column 7 72-    Statement field
Column 72-80    Identification field
```

And if the statement is too long:

```
Column 6        Continuation field (next line)
Column 7 -72    Contiuation of the statement
```

Comments can be placed between statements:

```
Column 1        The letter C
Column 2 -72    Text of the comment
```

USE OF THE ASCII TAB CHARACTER

The ASCII tab (Control-I) can be used to speed up the typing and reduce the size of the source program. Enter the label (line number) first (if any) starting in column 1. Then type a tab followed by the FORTRAN statement. The compiler will interpret the tab as the equivalent number of spaces. Thus:

```
12 <tab> X = 4
```

has the same meaning as:

```
12 <6 blanks> X = 4
```

If you have existing FORTRAN source programs that use blanks instead of tabs, they can be converted by using the substitute command in the CP/M editor:

```
BMS^L        ^Z^L^I^Z
```

(The up-arrow means that the control key is pressed.)

THE ORDERING OF SOURCE STATEMENTS

For subprograms, the first line is a SUBROUTINE, FUNCTION, or BLOCK DATA statement. The next group of statements (and the first group for a main program) are the specification statements. They must appear before any executable statements, and must be in the following order:

```
EXTERNAL, DIMENSION, REAL, INTEGER, ETC.
COMMON
EQUIVALENCE
DATA
```

The executable statements appear next:

```
A = SQRT(X*X + Y*Y)
IF (I .LT. K) GOTO 28
STOP
```

It is good programming practice to group the format statements after the last executable statement (this will usually be a STOP or RETURN).

```
100  FORMAT(' PARABOLIC FIT')
101  FORMAT(1P6E13.2)
```

The final statement in each program is:

```
END
```

More than one program may be placed into the same file. This would normally be done if there are subroutines used only by one main program, or if one of the subroutines called the others. On the other hand, generaly subprograms such as a sort routine might be called by several different main programs. These then should either be placed into separate files, or combined with several similar routines into a utility library.

ADDITIONAL FORTRAN FEATURES

FORTRAN-80 adds some nice features to the standard ANSI FORTRAN:

1. Logical variables
2. Logical DO-loop indices
3. Mixed-mode arithmetic
4. ASCII strings in expressions
5. Hexadecimal constants
6. Logical operations
7. END= and ERR= in READ and WRITE
8. ENCODE and DECODE
9. PEEK and POKE
10. INP and OUT

FORTRAN considers variables starting with the letters I through N to be integers, and the others to be real, single-precision variables. But this default mode can be over-ridden with specific declarations. Variables can be explicitly declared as one of four types:

```
LOGICAL   1 byte, with a value of TRUE or FALSE or a
          number from -128 to 127
INTEGER   2 bytes, -32,768 to 32,767
REAL      4 bytes, 7+ decimal digits
                   10**-38 to 10**38
```

DOUBLE PRECISION 8 bytes, 16+ decimal digits; same dynamic range as REAL

There is effectively a fifth type of variable. Any of the above four variables can be used as a STRING variable, with a maximum of one ASCII character per byte.

MIXED MODE

Mixed-mode arithmetic means that an expression such as:

```
Y = 2 * A + 3
```

is allowed, i.e., the decimal points are not needed on the 2 and the 3. Hexadecimal constants can be defined with either an X or a Z:

```
I = Z'FF'    and
J = X'CO'
```

ASCII strings can be defined in three ways: in a data statement, a replacement statement, or a FORMAT statement.

```
      INTEGER TITLE(10)
      DATA TITLE/'NON-','LINE','AR C', 'URVE',' FIT'/
or
      NO = 'NO'
or
      WRITE (1,101)
        •  •  •
101   FORMAT('PRESSURE VS. TEMPERATURE')
```

The END= option makes it easy to read data without knowing how much there is. The statements:

```
      READ(6,102,END=20)(A(I),I=1,99)
20    N = I - 1
```

can be used to read values into the array A from logical device 6 (a disk for example) until the end-of-file (EOF) mark is encountered. Then the statement, labeled 20, sets the correct number of items read. (Since the EOF mark was also counted, the total must be reduced by one.)

ENCODE and DECODE operations allow the interconversion of ASCII and numeric values, much like the VAL and ASC functions of BASIC. PEEK and POKE allow memory locations to be read or changed. INP and OUT can be used to communicate with peripherals.

COMPILER THE FORTRAN SOURCE PROGRAM

At this point, the FORTRAN source programs have been generated with the CP/M context editor, or copied to a disk file from paper tape using the CP/M PIP program.

We also use a third method. IBM cards are read into our campus central computer and saved on disk files there. A telephone link is then established to our microcomputer using a modem. The FORTRAN files are transferred over the telephone line into our computer memory starting at 100 HEX. The programs are then saved on a floppy diskette by using the CP/M SAVE command.

It is possible, of course, to proceed this far without actually having a FORTRAN compiler, since only the CP/M editor has been used. You might want to do this in anticipation of receiving FORTRAN if you have a large library of programs.

THE ACTUAL COMPILING

Source programs are compiled with the command:

```
F80 =SORT
```

or

```
A:F80 =B:SORT
```

if the compiler is on drive A and the source program is on drive B. Several programs can be more easily compiled with the command:

```
F80
*=SORT
*=C:PPLOT
*=B:CURVFIT
*^C
```

In this case, the compiler prompts each new line with an asterisk. A Control-C is used to indicate the end of the compile session. If there are several subprograms within a single file, the compiler will list the name of each subprogram as it encounters it. The filename need not match any of the subprogram names. If the file contains a main program, the word $MAIN will also appear in the list during the compile procedure.

The compiler produces a relocatable, machine-language program with the same name as the source file, but with the file type of REL. During compilation, two types of error messages may be printed: warning and fatal errors. A warning might occur if a STOP statement were temporarily inserted into the middle of a program during a debugging session:

```
      WRITE (1,101) X
      STOP
      X = 4
```

The compiler will discover that there is no way to reach the statement X = 4 and so issues a warning message. Although this is not a serious problem, the warning message can be avoided by adding the dummy statement:

```
100   CONTINUE
```

after the STOP statement.

A fatal error can occur, for example, if there is an odd number of parentheses in a statement:

```
Y = A * (B + LOG(C)
```

In this case, it will be necessary to correct the error using the CP/M text editor, then recompile the program with F80.

A FORTRAN LISTING FILE

The FORTRAN compiler can be directed to generate a listing file during the compile process. The switch /L is used for this purpose.

```
F80 = SORT/L
```

This causes an additional file, with the extension PRN to be produced. It contains the original lines of the source program with the corresponding assembly listing of the generated code, interspersed throughout.

The PRN file is useful in debugging a program. It can also be used to increase the efficiency of a frequently used subprogram. In this case, the program is first written in FORTRAN, then compiled with the /L switch. Finally, the PRN file can be used as a guide for writing a more efficient assembly language program.

EXECUTING A FORTRAN PROGRAM

When all of the modules have been successfully compiled, they can be executed with the linking loader:

```
L80  MAIN,SUBR1,SUBR2/G
```

where MAIN, SUBR1, and SUBR2 are the file names of relocatable files. (Each may contain several subroutines.) The standard FORTRAN library routines such as ABS, ATAN, EXP, SIN, etc., are located in a file named FORLIB.REL. If FORLIB resides on the currently logged-in disk, it will be automatically searched for the necessary programs. If, however, the user-written FORTRAN programs are on a different drive from the FORTRAN processing programs, then the process is a little more complicated. The drive names must be included and FORLIB must be specifically listed if it is not on the default drive. For example, the execution command can be:

```
A>L80  B:MAIN,B:MATHLIB/G
```

in case L80 and FORLIB are on the currently logged-in drive A, or

```
B>A:L80 MAIN,MATHLIB,A:FORLIB/S/G
```

if B is the default drive. Notice that the filetype REL is not entered.

The FORTRAN linking loader will automatically find all necessary programs, relocate them in memory, then start execution if the /G switch has been given. The /S switch immediately following FORLIB instructs the loader to search that library for the necessary routines and then load them into memory. If the /S switch is not given, the entire FORLIB library will be loaded into memory.

The absolute memory image can be saved as a disk file of type COM if the /N switch is set. This will allow the program to be more quickly run. But the disadvantage is that the COM file requires relatively large amounts of disk space.

OUTPUTTING THE DATA

At some point in the process, the programmer will want to see at least some of the results of the calculations. This is accomplished in FORTRAN with a WRITE statement.

```
WRITE (LUN,101)  <list>
```

where LUN is the FORTRAN logical unit number specifying the particular peripheral, 101 is the line number of the format statement and <list> is a list of the variables to be written.

Logical unit numbers 1,3,4, and 5 are preassigned to the system console. An LUN of 2 is preassigned to the list device and LUN values of 5 through 10 are preassigned to disk operations. Units 11 through 255 can also be used by the programmer.

During the development of a new program, it would be advantageous to first view the results on the video screen of the system console. This is accomplished by defining the LUN in the WRITE statements to be 1. Then after the program is running satisfactorily, the output can be sent to the line printer so a permanent copy can be obtained. There are several ways in which this can be accomplished.

If the CP/M IOBYTE feature has been implemented, then the program called STAT can be used to reassign the console output to the list device:

```
A>STAT CON:=LST:
```

When the FORTRAN program is executed again, the results will appear at the line printer.

Another method would be to input the LUN from the console near the beginning of the program.

```
      LUN = 1
      WRITE (1,101)
      READ (1,102) NOYES
      IF (NOYES .EQ. 'Y' .OR. NOYES .EQ. 'y') LUN = 2
                  • • •
  101 FORMAT(' OUTPUT TO LINE PRINTER? ')
  102 FORMAT(A1)
```

This routine only looks at the first character that was entered, ignoring the rest. Thus, inputting a YES, a Y or a YUP will send the output to the line printer. Any other answer will send the output to the console.

ABORTING A FORTRAN PROGRAM

Suppose that you would like to generate a stream of random numbers so that the calculated values can be examined. Then at some point, you would like to stop. A Control-C can be used to abort a BASIC program in this case, but FORTRAN has no such option built in. The INP function provided by Microsoft, however, can be used for this purpose. The following routine could be executed after every 100 loops. It is written for a console status port of decimal 16, and a read-ready flat at bit 0, active high.

```
      LOGICAL DONE
                  • • •
      DONE = INP(16)
      DONE = DONE .AND. 1
      IF (DONE) STOP
```

DISK INPUT-OUTPUT

Both sequential and random-disk file access are available in the CP/M version. FORTRAN logical unit numbers 6 to 10 have been preassigned for this purpose. The FORTRAN statement:

```
WRITE (6,101) (A(I),I=1,N)
```

will place the data into a file named FORT06.DAT of the currently logged-in disk.

Alternatively, a more specific method is available. The command:

```
CALL OPEN (6,'NEWDATA.ASC',2)
```

will open a file named NEWDATA.ASC on drive B and associate it with logical unit number 6. The first argument defines the logical unit number and must evaluate to an integer. The second argument is the filename. Notice that it is not in the usual CP/M format. In this case, the filename must evaluate a string of exactly 11 ASCII characters and must not contain the usual decimal point between the primary name and the extention. The first eight characters are the primary name and the last three characters are the file type. If the primary filename is shorter than eight characters, as in the above example, the remainder must be filled with blanks.

The third argument of OPEN specifies the disk drive, and must evaluate to an integer. A zero value refers to the default drive and the numbers 1 through 4 explicitly specify drives A through D. Once a file has been opened, it can be read with the command:

```
READ (6,102) (B(I),I=1,N)
```

If data is written to the file with the statement:

```
WRITE (6,105) A,B,C
```

then a new file is created. If a file of the same name already exists, it is erased before the new data is written.

At the end of the disk access, the file should be closed with the command:

```
ENDFILE 6     or
REWIND 6
```

The latter command closes the file, then reopens it. This could be used to write data in one format, then read it back in a different format. (But see the ENCODE and DECODE commands.)

ASSEMBLY-LANGUAGE PROGRAMS FOR FORTRAN

The Microsoft FORTRAN compiler converts the user's source program into a relocatable machine-language program which is in turn converted into binary code. But the resulting binary code may not be as fast or occupy as small a memory space as if it had been originally written in assembly language. The tradeoff is that the FORTRAN source program can generally be written and debugged much more rapidly than if it had been written in assembly language. Nevertheless, for short, frequently used subroutines, it is often advantageous to use assembly language rather than FORTRAN.

The Microsoft FORTRAN package contains a macro assembler that produces compatible, relocatable modules that can be called from FORTRAN programs in the usual way. In fact, the programmer will not generally be concerned with whether the relocatable modules were originally written in FORTRAN or in assembly language.

An assembly language function to generate real random numbers can be written, since such a routine is not provided in the standard library. The algorithm, which appeared in the October 12, 1978, issue of Electronics is used to generate a 24-bit integer (Listing 1).

The low-order 23 bits are copied into the 3-byte mantissa of the FORTRAN floating-point accumulator (at $AC). The 24th (high order) bit is zeroed to make the resulting number positive. The 8-bit exponent is set to 80 HEX to give a resulting range of 0.5 to 1.0. The number is then converted to the usual range of 0 to 1 by using the FORTRAN arithmetic routines. The random number is first multiplied by 2 with the subroutine $MA, then 1 is subtracted with the subroutine $SA.

Notice that the subroutines $MA and $SA are declared as external, as is the location of the floating-point accumulator $AC. Also the subroutine name RND, used by the calling FORTRAN program, is declared to be an entry.

The assembly language random-number generator can be called from a FORTRAN program in the usual way:

```
Y = RND(NSKIP)
```

A real random number between 0 and 1 will be placed into the variable Y. The integer argument instructs the function to skip over NSKIP random numbers before choosing the next number. This argument can be retrieved with a MOV A,M instruction, since the H,L register pair points to the least-significant byte of argument.

THE LIBRARY MANAGER

The CP/M vesion of FORTRAN contains a program called LIB that can be used to build library files of relocatable programs. For example, the relocatable module of the above random-number generator can be incorporated into FORLIB by use of the program LIB. This makes it unnecessary to specifically list the module RND in the link command at execution time.

A SPEED COMPARISON

The Microsoft CP/M version of BASIC is much faster than earlier versions such as 4.1 EXTENDED, and also faster than many of the other 8080 or Z-80 BASICs. A speed comparison was made between Microsoft BASIC and FORTRAN by solving sets of linear equations. The same algorithm, a Householder technique, was coded in both BASIC and FORTRAN. the BASIC statement:

```
DEFINT I-N
```

was used to declare loop variables to be two-byte integers for faster operation. The FORTRAN program consistently produced the solution 8 times faster than the BASIC version (17 seconds vs. 135 seconds for 14 equations).□

PROGRAM LISTING

```
                        ; RND: A FORTRAN-CALLABLE FUNCTION
                        ;      TO GENERATE A RANDOM
                        ;      NUMBER FROM 0 TO 1
                        ;
                        ; METHOD: ELECTRONICS, OCT. 12, 1978
                        ; USAGE:  X = RND(NPASS)
                        ;  NPASS IS THE NUMBER OF TIMES TO SKIP
                        ;
                        ; PROGRAMMED BY ALAN R. MILLER
                        ; NEW MEXICO TECH, SOCORRO 87801
                        ;
0000'                           TITLE   RANDOM-NUMBER GENERATOR
                        ;
0000'                           EXT     $AC,$MA,$SA
0000'                           ENTRY   RND
                        ;
0000' 06 01             RND:    MVI     B,1     ;SET FOR ONE PASS
0002' 7E                        MOV     A,M     ;GET ARGUMENT
0003' E6 0F                     ANI     0FH     ;TAKE 4 BITS
0005' CA 0009 '                 JZ      NEXTN   ;CHANGE 0 TO 1
0008' 47                        MOV     B,A
0009' 2A 0049 '         NEXTN:  LHLD    WORD2   ;HIGH 2 BYTES
000C' EB                        XCHG            ;PUT IN D,E
000D' 2A 0047 '                 LHLD    WORD1   ;LOW 2 BYTES
0010' 29                        DAD     H       ;SHIFT LEFT
0011' 7B                        MOV     A,E
0012' 17                        RAL             ;SHIFT LEFT
0013' 5F                        MOV     E,A
0014' AD                        XRA     L       ;FEEDBACK
0015' F2 0019 '                 JP      SKIP
0018' 23                        INX     H
0019' 22 0047 '         SKIP:   SHLD    WORD1   ;HIGH 1 BYTES
001C' EB                        XCHG
001D' 22 0049 '                 SHLD    WORD2   ;HIGH 2 BYTES
0020' 05                        DCR     B       ;COUNT
0021' C2 0009 '                 JNZ     NEXTN   ;DO IT AGAIN
0024' 21 0000 *                 LXI     H,$AC   ;POINT TO FAC
0027' 3A 0047 '                 LDA     B1      ;LOW BYTE
002A' 77                        MOV     M,A     ;PUT IN FAC
002B' 23                        INX     H
002C' 3A 0048 '                 LDA     B2      ;SECOND BYTE
002F' 77                        MOV     M,A
0030' 23                        INX     H
0031' 3A 0049 '                 LDA     B3
0034' E6 7F                     ANI     7FH     ;BIT 7 PLUS
0036' 77                        MOV     M,A     ;PUT INTO FAC
0037' 23                        INX     H
0038' 36 80                     MVI     M,80H   ;SET EXPONENT
003A' 21 0002                   LXI     H,2
003D' CD 0000 *                 CALL    $MA     ;TIMES 2
0040' 21 0001                   LXI     H,1
0043' CD 0000 *                 CALL    $SA     ;SUBTR 1
0046' C9                        RET
                        ;
0047'                   WORD1:
0047' 0D                B1:     DB      0DH
0048' B1                B2:     DB      0B1H
0049'                   WORD2:
0049' 9B                B3:     DB      9BH
004A' 80                B4:     DB      80H
004B'                           END

$AC     0025*   $MA     003E*   $SA     0044*   RND     0000'
NEXTN   0009'   WORD2   0049'   WORD1   0047'   SKIP    0019'
B1      0047'   B2      0048'   B3      0049'   B4      004A'
```