MIRA SOFTWARE

FORTRAN COMPILER

Instruction Manual

The MIRA SOFTWARE Fortran Compiler for the Sinclair Spectrum is based on FORTRAN 77S, which is a subset of Standard FORTRAN 77. It compiles programs written in FORTRAN into machine code, which can then be run independently of the compiler itself.

This instruction manual is in two parts. The first concerns the loading and use of the compiler and the second gives details of the Fortran language commands as implemented on this compiler.

To begin programming you should start the tape at the beginning of side 1 and enter LOAD"". This will load a Basic LOADER program which will then load the first part of the compiler called FORTRAN1. When this is loaded the message 'Press required key' will appear at the bottom of the screen. You should then stop the tape. The line editor of the compiler will then allow you to enter a program. The editor will be in control key mode when pressing various keys will allow manipulation of the program.

ENTERING LINES

Pressing ENTER will put the editor into line entry mode allowing you to enter the lines of your Fortran program in sequence. The Fortran language specifies a certain layout for lines of a program. The line may not be more than 80 characters in length. The first character may be a C or a * specifying that the line is a comment line to be ignored by the compiler, or a D specifying a Debug line, which may either be a comment line or a program line. If the first character is none of these then the first five characters must either be left blank or be a statement label, that is a 1 to 5 digit number. The sixth character should be blank for a normal program line. The characters in positions 7 to 80 are used for the FORTRAN statement. Note that Fortran statements only use capital letters, lower case ones are not allowed. Only one statement is allowed on each line. If a statement is too long to fit into one line then you may continue it by putting a + character in the sixth character position of the next line and continuing the statement from there.

When a line is entered the compiler does an elementary syntax check before entering it into the program. If an error is detected then the line remains at the bottom of the screen with a '?' markng the probable position of the error, ready for you to correct the error. As each line is entered it will be added to the program listing, and the program cursor will mark its

position. In the listing each line has a line number to indicate its position in the program. These are not to be confused with statement labels which may occupy the first five character positions of a line.

In order to leave line entry mode and return to control key mode you should press SHIFTED 6. The messages 'STOP in INPUT' and 'Press required key' will then appear at the bottom of the screen.

CONTROL KEYS

When the program is listed on the screen a cursor (flashing '>') marks the position of the current line. This cursor may be moved up and down a line by pressing the 7 and 6 keys. If shifted 7 or shifted 6 are pressed then the cursor will move up or down eight lines at a time. The current line may be edited by pressing 1 or EDIT. Pressing E allows entry of a single line before the current line. If line entry mode is selected by pressing ENTER then the lines added will be positioned after the current line. Pressing V will list the program on a printer (i.e. on stream 3). Pressing H puts a list of the control keys and their functions on the screen.

Saving and Loading programs

Pressing S allows the Fortran Program to be saved to tape. You should enter the name of the program when this is requested, and then the program is saved on the tape in the normal way. Then the compiler will require the program to be verified. It is recommended that programs should always be saved before they are compiled as the compilation process erases the Fortran program from memory.

To load a program from tape you should press J and enter the name of the program required (pressing ENTER will load the next program on tape as usual). If this is done with a program already on the screen then the new program will be merged into the old at the position of the cursor.

Moving Blocks of lines

It is possible to move a block of program lines within the program. To do this the B key should be pressed to mark the start of the block at the current position of the cursor, then the cursor should be moved to the line following the end of the block. The block may then be stored by pressing K. It may later be retrieved by pressing R, when it will be put at the position of the cursor. A block may be deleted by pressing SHIFTED 0 (DELETE).

Compiling programs

When a program has been entered it may be compiled by pressing the X key. The compiler will then do further syntax checks on the program, and if an error is detected then it will indicate it by moving the cursor to the line in which it occurs, and indicating the probable position with a question mark, ready for the line to be edited and the error corrected. An error message may be printed at the bottom of the screen indicating the nature of the error.

While the compilation is in progress, the table of variable names etc. will be positioned in the screen memory. This will result in a random pattern appearing in the screen. If no errors are detected then a buzz will sound and the second part of the compiler needs to be loaded by starting the tape from where it was stopped on side 1. The second part is called 'CODER' but this will not appear on the screen as if it did it would corrupt the symbol table. If there are any errors during the loading of this then the buzz will sound again and it should be reloaded.

When the second part has been loaded it will proceed with the compilation. Occasionally an error will be detected at this stage, when the buzz will sound again, and an error number will appear on the screen. It is then necessary to rewind the tape and reload the first part of the compiler together with the Fortran program.

When the compilation is successfully completed some information will be printed on the screen and printer. The first number is the maximum address for RAMTOP when the compiled program is in use. Then come the start and length of the compiled program, and finally (after REM) the address of the blank common storage of the program. If you do not have a printer connected then you should make a note of these numbers. Then pressing any key will reset the computer, leaving the compiled program stored above RAMTOP. It may then be saved for future use as a block of code using the start and length specified. When you want to reload this code you should first set RAMTOP to the required address using CLEAR followed by the specified number.

The compiled code is run using RANDOMISE USR 63500. This may be used within a BASIC program if desired. If an error occurs during the running of a FORTRAN program then an error message will be displayed, with the line number of the BASIC program which called the Fortran code.

FORTRAN COMMANDS

The following section gives details of the Fortran commands, together with information on which aspects are different from the Fortran standard. It does not attempt to teach you Fortran if you are a novice programmer, since if this is the case then it is best for you to select a Fortran teaching guide which is to your personal taste.

Data types

The data types permitted by this compiler are INTEGER, REAL, LOGICAL and CHARACTER. The data types COMPLEX and DOUBLE PRECISION occuring in full Fortran77 are not permitted. Integer values must lie within the range -32767 to +32767. Real values are dealt with in the same way as in Spectrum Basic. A real constant must contain either a decimal point or an exponent or both. Thus 3000 is an integer constant, 3000., 3E3 are real constants, and 3000000 would be taken to be an integer constant and so an error. The two logical constants are .TRUE. and .FALSE. (The full stops at each end of such words must be remembered.) Character constants are written with a single quote (') at each end, quotes occuring within the constants are represented by two consecutive quotes.

Variables and type declarations

The names of variables and other identifiers in Fortran must not be longer than 6 characters, of which the first must be a letter and the rest letters or digits. Each variable has a type which is determined either by a declaration statement or by the implicit type conventions. The type declaration statements consist of the name of the data type (INTEGER, REAL, LOGICAL, or CHARACTER), followed by the names of the variables. In the case of CHARACTER variables the length may also be specified, e.g.

```
CHARACTER*6  A,B*3,C
```

would give length 6 to A and C and length 3 to B. In this implementation character variables may have a maximum length of 255 characters.

If a variable does not occur in a type declaration statement then if the name begins with I,J,K,L,M,N then it is of type INTEGER, otherwise it is of type REAL. This convention may be altered by an IMPLICIT statement, which specifies a type followed by the range of letters which will assume that type by default, e.g.

```
IMPLICIT CHARACTER*5 (A-D),LOGICAL(L), INTEGER(X-Z)
```

Expressions and assignment

An assignment statement in Fortran has the form

variable=expression

where the type of the expression must be the same as that of the variable, or else one must be integer and the other real. An arithmetic (i.e. REAL or INTEGER) expression is composed of terms combined together with parentheses (and), and the usual operators +,-,*,/ and ** where A**B represents A raised to the power B. The terms must be constants, variables or the result of function calls (intrinsic or user defined functions). If an integer value occurs where a real value is required or vice versa then it will be automatically converted to the appropriate type. The following is a list of intrinsic functions availiable. In this list X and Y represent real values and I and J represent integer values.

NAME	Definition
SQRT(X)	Square root
EXP(X)	exp
ALOG10(X)	logarithm to base 10
ALOG(X)	logarithm to base e
SIN(X)	sine
COS(X)	cosine
TAN(X)	tangent
ASIN(X)	arcsine
ACOS(X)	arccosine
ATAN(X)	arctangent
ATAN2(X,Y)	ATAN(Y/X) in correct quadrant
SINH(X)	Hyperbolic sine
COSH(X)	Hyperbolic cosine
TANH(X)	Hyperbolic tangent
AINT(X)	Truncates down to integer, but result is of REAL type
ANINT(X)	Rounds to nearest integer, REAL result
ABS(X)	Absolute value of X
AMOD(X,Y)	Remainder from X/Y
SIGN(X,Y)	ABS(X) with sign of Y
DIM(X,Y)	X-Y if X>Y otherwise 0
INT(X)	Truncates to integer type
FIX(X)	Same as INT
NINT(X)	Rounds to nearest integer
REAL(I)	Converts to real type
FLOAT(I)	Same as REAL

IABS(I)	Absolute value of I
MOD(I,J)	Remainder from I/J
ISIGN(I,J)	IABS(I) with sign of J
IDIM(I,J)	I-J if I > J, otherwise 0
AMAX1(X1,X2,...,Xn)	Takes maximum value
MAX1(X1,X2,...,Xn)	Takes integer part of maximum value
AMIN1(X1,X2,...,Xn)	Takes minimum value
MIN1(X1,X2,...,Xn)	Integer part of minimum value
AMAX0(I1,I2,...,In)	Max value converted to real type
MAX0(I1,I2,...,In)	Maximum value
AMIN0(I1,I2,...,In)	Minimum value converted to real type
MIN0(I1,I2,...,In)	Minimum value

Logical expressions consist of logical terms combined together with parentheses and the logical operators .NOT., .AND., .OR., .EQV., .NEQV. . Logical terms may be logical variables, functions or constants, and may also be relational expressions, that is comparisons of two arithmetical or character expressions using the relational operators .LT., .LE .EQ., .NE., .GT., .GE. (The usual symbols <,<=,=,<>,>,>= are not used in Fortran).

Character expressions may be character constants or variables. Character functions, substrings and concatenation are not permitted in this version of Fortran.

Program Structure and control statements

A Fortran program consists of a number of program units, one of which will be the main program, any others being user written subroutines and functions. The main program may optionally start with a PROGRAM statement, that is the keyword PROGRAM followed by a name. The last statement of each program unit must be an END statement, which will return to the calling program if in a subroutine, and return to Basic if in the main program. The STOP statment will return directly to Basic. The STOP keyword may be followed by a character string, or a sequence of digits to be printed out when it is executed. The PAUSE statement will halt the execution of the Fortran program until a key is pressed. This may also be followed by a string or digit sequence to be printed.

There are three different kinds of IF statement in Fortran, these are the block IF, the logical IF and the arithmetic IF. The block IF statement consists of the keyword IF followed by a logical expression in brackets followed by the keyword THEN. For each block IF statement there must be a corresponding END IF statement. ELSE and ELSE IF statements are also permitted.

The logical IF statement is a single statement consisting of IF followed by a logical expression in brackets, followed by the statement whose execution is conditional on the logical expression. The arithmetic IF statement consists of IF followed by an arithmetic expression in brackets, followed by three statment labels. If the expression is negative then control jumps to the first statement, if zero to the second and if it is positive then to the third.

Looping in Fortran is implemented using the DO loop. The DO statement has the form

```
DO lab IVAR = IE1,IE2,IE3
```

where lab represents a statement label determining the last statement of the loop, IVAR represents an integer variable, and IE1, IE2, IE3 are integer expressions determining the initial value of IVAR, the final value, and the step. The loop is thus executed INT((E2-E1+E3)/E3) times, and not at all if this is less than or equal to zero.

The statement CONTINUE is a dummy statement which does nothing. This is often used as the last statement of DO loops.

There are three types of GOTO statements in Fortran. The unconditional GOTO statement is simply the keyword GOTO followed by a statement label, where the execution of the program is to jump to. The computed GOTO statement is the keyword GOTO followed by a sequence of statement labels in brackets, followed by an integer expression. Control of the program jumps to the label whose number in the sequence is equal to the expression. An assigned GOTO statement consists of GOTO followed by an integer variable, followed by a sequence of labels in brackets. The variable must have been given a label value by an ASSIGN statement, which has the form

```
ASSIGN label TO IVAR
```

Then at the assigned GOTO statement, if the variable has been assigned the value of one of the labels following it, then control will jump to the statement which that label marks.

Storage of variables

As well as the type declaration statements described above, Fortran has a number of statements controlling the storage of variables. These are not executable statements, and occur in the program before any executable statements.

Arrays of any of the four types may be declared in Fortran by means of a DIMENSION statement. This has the form

```
DIMENSION AR(20,10),NAME(30)
```

which would make AR a two dimensional 20 by 10 array, and NAME a one dimensional array with 30 elements. An array

element is accessed in an expression by specifying the position in the array e.g. AR(2,3). Note that this implementation checks that the element lies within the total bounds of the array, but not the indiviual bounds for each dimension, so that a reference to AR(25,3) would be taken to refer to AR(5,4) rather than signalling an error. The declaration of an array may also be placed in a type declaration statement or a COMMON statement.

The values of variables and arrays may be initialised by means of DATA statements, which have the form:

```
DATA var1,var2,.../value1,value2,.../
```

where var1, etc. represent variable names, array names, or array elements, and value1 etc. are the values assigned to them. An array occuring in a data statement will have the corresponding values assigned to its elements in sequence. The notation number*value may be used in the list of values to represent a value repeated a number of times. Implied-DO lists in data statements are not permitted in this implementation.

The PARAMETER statement is not implemented by this compiler.

COMMON blocks are a way of indicating that variable storage is not only used by one subprogram. The COMMON statement typically has the form

```
COMMON /NAME/ VAR1,VAR2,...
```

indicating that the variables VAR1, VAR2, etc are to be stored in the common block called NAME, and so can be accessed by any other program unit containing a similar statement. If the name of the common block is omitted then storage in blank common occur. Varibles which are initialised in a DATA statement may not be stored in a common block.

Two or more variables may be made to share the same storage location by an EQUIVALENCE statement. This has the form

```
EQUIVALENCE (A,B,C(4)),...
```

indicating that the variables (or arrays) A and B are to start in the same position as the array element C(4).

Note that the Fortran language allows the same variable to occur in more than one EQUIVALENCE statement, as well as a COMMON statement. The compiler will then do its best to sort out the storage of the variables and arrays. Obviously this sort of mixing up of storage is not recommended. However, if you want to equivalence entities of different types you will need to know the amount of memory space taken by each type of variable. In this implementation real values take up 5 bytes, integer values take up two bytes, character values take up 1 byte and logical values take up 1 byte for a simple variable, while for a logical array, 8 values are stored per byte. This differs from the

Standard which specifies that REAL, INTEGER and LOGICAL values should each take up the same amount of storage.

The start of blank common in memory is given by the number following REM printed out after the compilation, allowing you to access Fortran variables from BASIC.

Functions and Subroutines

Apart from the main program, a Fortran program may contain other program units, known as subprograms. These may either be functions or subroutines. The first statement of a subprogram must be either a SUBROUTINE or a FUNCTION statement, with the keyword followed by its name, and then the list of arguments in brackets. (For a function the brackets must be included, even if there are no arguments). A function has a type, declared either by the word INTEGER, REAL, or LOGICAL preceeding the FUNCTION keyword, or within the function unit, or by the implicit type rules. It is not necessary in this implentation to declare the type of a function in each program unit in which it is called. A function unit must contain an assignment statement assigning a value to be returned by the function. A subroutine is called from another program unit by a CALL statement e.g.

```
CALL GRAPH(A,B,X+1)
```

A function unit is called by including its name and arguments to be passed to it in an expression, where it will be given the required value.

Note that if the argument passed is a variable, array, or array element then it will be passed by reference i.e. its value may be changed by the subroutine or function, whereas if the argument is an expression then it will be evaluated and its value passed. The type of the actual arguments passed must agree with that of the dummy arguments in the argument list of the subprogram. If one is integer and the other real then the required conversion will occur, but the argument will be passed by value not by reference. Note that it is possible to pass an array when a single variable is required, when the first element of the array will be taken. It is also possible to pass a single variable or an array element when an array is required. In the case of a single variable, it will be taken to be an array of size 1. In the case of an array element, the array passed will be the remainder of the array following the specified element.

When an array is one of the arguments of a subprogram it must be declared as an array in the subprogram. The dimensions of the array may be different to those of that passed as an argument, as long as the total size is no greater. It is also possible to have variable dimensions for the array (adjustable

arrays), and to specify the last dimension of the array as an asterisk, when the array will take the size of the array which is passed (assumed size arrays).

Subprograms may also be passed as arguments. In this case the actual argument must occur in an EXTERNAL statement (for user defined subprograms) or INTRINSIC statement (for intrinsic subroutines and functions) in the calling program, and the dummy argument it corresponds to must occur in an EXTERNAL statement in the subprogram in which it occurs. Note that there the types of the arguments of an actual subprogram passed as an argument and the types of the arguments occuring with the dummy subprogram must be exactly the same, i.e. INTEGER and REAL interconversion will not take place.

The following is a list of intrinsic subroutines which may be used on this compiler

```
CIRCLE(X,Y,R)   DRAW(X,Y)     PLOT(X,Y)
BEEP(X,Y)       ARC(X,Y,R)
```

These all have REAL arguments, and they have the same effect as the commands with the same names in BASIC. CALL ARC(X,Y,R) has the effect of the BASIC statement DRAW X,Y,R .

Variables in subprograms are preserved between calls so that the Fortran statement SAVE is unnecessary.

Input and Output in Fortran

The area of input and output is acknowledged to be the strong point of Fortran. This is because Fortran has a large number of ways of controlling input and output. Unfortunately this means that this is the area where differences between compilers are greatest, especially in this case, as the input and output devices for the Spectrum are totally different to those on a mainframe computer. The subset of Fortran defined in the standard (Fortran77S) is more suited to microcomputers, but this compiler still differs in some places from Fortran77S.

An output statement in Fortran might have the form

```
WRITE (2,10) X,Y,(IR(I),I=3,30,3)
```

Here the 2 is the unit number. This corresponds to a stream number in BASIC, so that 1 would normally refer to input from the keyboard and output to the lower screen, 2 to the main screen, 3 to the printer and other numbers to other channels which would have to be opened by a Basic command before calling the compiled Fortran program. The unit number may be replaced by an asterisk, specifying a default value of 1 for input and 2 for output. The replacement of a unit number by a character array is not permitted in this implemtation.

The 10 is the format identifier and refers to the label of a

FORMAT statement such as
10 FORMAT ('Values are ',2F8.2/3X,10I4)
The format identifier may be an integer variable given the value of a label in an ASSIGN statement.
The format may be included in the WRITE statement e.g.
WRITE (2,'(''Size ='',2X,F6.2)') SIZE
Also the format identifier may be an asterisk, implying list directed format, which for this compiler means that each value is printed out on a new line. If the format identifier is omitted then the output will be unformatted, that is it will be in the form in which the values are stored in memory.

The following format codes are allowed in a format specification
Iw,A,Aw,Fw.d,Ew.d,Lw,kP,nH...,'...',nX,/,comma
The following occur in full Fortran but not on this compiler:
Iw.m,Fw.dEe,Ew.dEe,Dw.d,Gw.d,Gw.dEe,Tc,TLc,TRc,S,SP,SS,:,BN,BZ
(Here w,e,n,c,d,m,k stand for integers)

This implementation also allows a format code C which is followed by an integer between 0 and 255 causing that ASCII code to be output. The format control CS will clear the screen (the unit number must be 2). Note that printer control codes at the start of format specifications are not implemented by this compiler.

The actual input/output list follows the control information list. This may contain variables, arrays, array elements, and i/o implied do lists, as above, but this implementation does not allow the occurence of expressions in the input/output list. (Thus character strings must not occur in the i/o list, they must either be assigned to variables, or be dealt with in the format specification)

An alternative output statement is the PRINT statement. Here the control information list is replaced by a format identifier (frequently an asterisk) and the unit is the default one e.g.
PRINT *,A,B,C

The input statement in Fortran is READ. This may have the form of either the PRINT statement, or the WRITE statement with the full control infomation list. In the second case an END specifier may be placed after the unit identifier, allowing a jump to another part of the program to occur if the end of file is detected for input from a microdrive or network channel. Note (a) This is not the position for the END specifier given in standard Fortran (b) The jump will only be made if the end of file condition occurs after an end of line character (CHR$ 13), otherwise the error End of File will be given in the normal way. The other control specifiers (REC, ERR, IOSTAT) are not implemented.

List directed formatting is recommended for input from the keyboard, as formatted input will require exactly the number of characters given in the FORMAT specification.

The nature of input and output for the Spectrum means that all files are treated as sequential access files, so that random access files do not occur and the statements CLOSE, OPEN, INQUIRE, BACKSPACE, ENDFILE and REWIND are not implemented.

Debugging your program

When you have written a program it may contain errors, and these may be detected at different stages of the process of compiling and running a program. Simple syntax errors such as misspelling a keyword or missing out a parenthesis will normally be detected either when you enter the line or when the compilation is started. This sort of error is usually simple to spot and correct. More complex errors will not be detected until slightly later in the compilation. When these are indicated they will have an error message printed. There follows a list of these error messages, and the possible errors they relate to:

27, Undefined Identifier
An identifier (variable or function name) appears in an expression when there is nowhere in the program where it could have been given a value.
28, Wrong data type
The type of an expression or variable is not that required, or else a subroutine or array name is used where a simple variable is required or vice versa.
29, Wrong statement order
The usual errors of this type are mixing up the declaration statements and actual commands, and omitting necessary statements such as the END statement.
30, Array syntax error
Typically this is due to an array being treated as a simple variable or vice versa.
31, Identifier declared twice
The same identifier occurs in more than one declaration statement, or occurs twice in the dummy argument list.

32, Invalid item in list
Typically this would be due to the name of a subprogram or a dummy argument occuring in the list of variables in a COMMON, EQUIVALENCE or DATA statement.
33, Invalid equivalencing
This means that the EQUIVALENCE and COMMON statements would require the same variable to be stored in two different places.
34, Argument Mismatch
The actual arguments passed to a subprogram do not agree in number or in type with the dummy arguments required.
35, Undeclared label
A statement label is refered to which does not occur in the program unit.
36, Array element out of bounds
The subscript of an array element lies outside the range specified in the dimensioning of the array
37, Label used twice
Two statements in the same program unit have the same label
38, Invalid nesting
The IF blocks and DO loops in a program unit are not nested properly
39, Syntax error
The syntax of the program is in error in some way not listed above.

The errors 4, Out of memory and 11, Integer out of range may also be reported.

The above all refer to errors detected at the time of compilation. The errors which may occur when the program is run are as follows.
27, Wrong code in Format statement
The type of an item to be input or output does not match that required by the corresponding FORMAT code
28, Invalid input
29, Insufficient space to write
The item to be output would take up more space than the amount specified in the FORMAT statement.
30, Dummy procedure error
The type of a duumy proceeduere or one of its arguments does not match that of the actual proceedure or actual arguments.
31, Dummy Variable length error
A dummy array or character variable has been declared to be longer than the corresponding actual array or variable.

The normal error messages will be printed for errors in evaluating expressions. If the subscript of an array element is out of bounds in the running of a program, then the error message 'Integer out of range' will be given.

It is particularly important to make sure that a character expression used as a format identifier is correct, as errors will not be detected at compile time, and may cause a program crash when the program is run.

If you have a problem with a Fortran program then Mira Software may be able to help on an informal basis . Please write with precise details of the problem and enclose a stamped, addressed envelope for the reply. We shall try to help if possible. If the problem is due to a bug in the compiler that needs correcting then we shall send you a corrected version when the correction has been made. Likewise if the tape seems to be faulty then please write with details of the problem so that we can send a replacement copy. Do not return the tape unless requested.

The aims of Mira Software is to widen the application of home computers by providing Educational/Utility programs which are not otherwise availiable, at an affordable price. We would be interested to hear your ideas on what programs you would like for your home computer which are not at present availible. The address of Mira Software is given below.

MIRA SOFTWARE,
24, Home Close,
Kibworth,
Leicestershire.
LE8 0JT.