BBG RealFortran V.1 for the Sharp MZ-700/800

BIALKE – BERENDSEN – GLISZCZYNSKI

---------- Software ----------

Loading the BBG Fortran Compiler

To use the BBG RealFortran compiler on the Sharp MZ-700, load the program using the monitor as follows:

```
** MONITOR 1Z-013A **
* L <CR>
▼ PLAY
LOADING REALFORTRAN
```

After a successful load, the screen clears and the following message is displayed:

```
BBG's REALFORTRAN: AXEL BIALKE
OK.
```

On a MZ-800 the MZ-700 compatibility mode must be activated before using BBG Fortran. The IPL (Key C) can then be used to load the program.

Compiler Mode

After the program has loaded, you are in "compiler mode".

The available commands (and their abbreviations) in "complier mode" are:

EDIT (E) - Select "Edit mode"

COMPIL (C) - Compile the program currently in memory. If the compilation succeeds, a message similar to the one below is displayed:

```
* ERROR TOTAL 00000
* MEMORY SIZE nnnnn (where nnnnn = size of compiled
* COMPILE OK *                    program in memory)
```

If compilation fails, then the error total indicates the number of errors found. A message similar to the one below is displayed:

```
* ERROR TOTAL mmmmm (where mmmmm = errors found)
* MEMORY SIZE nnnnn
* COMPILE NG *
```

EXEC - Load a compiled program directly from tape without first loading the code into the editor

BSAVE - Save the compiled program in memory to tape

RUN (R) - Run the compiled program. Note that if you enter EDIT mode, the currently compiled program held in memory is not retained. It will therefore need recompiling before it is run again.

LIST (L) - When the COMPIL command is executed, a source code listing is output during compilation

LISTN (LN) - When the COMPIL command is executed, a source code listing is not output during compilation

LISTE (LE) - When the COMPIL command is executed only source code lines that contain an error will be output during compilation

BYE - Return to the monitor

Edit Mode

Edit mode is accessed using the EDIT (E) command when in compiler mode. On entering edit mode, # is used as the prompt. The following commands are available in edit mode:

R - Load a source code program from tape. This is appended to any existing source code.

Wfname - Write a source code program called “fname” to tape

& - Deletes the current source code program from memory

! - Return to compiler mode

I - Append new lines of code to the current source code. Use SHIFT & BREAK together to return to the editor prompt, #.

Bn - Insert a line before line n

Dn - Delete line n. Beware! Note that when this command is given, all of the following lines (n+1) are renumbered automatically so that line numbers are always consecutive integers from 0. For example, issuing the command D0 followed by D0 again will delete the first two lines of the original program’s source code.

L - List all of the current program source code

L,n - Lists the current program source code from line number n

L,P - List all of the current program source code to the printer

Note: SHIFT & BREAK pressed together stops all list commands and returns you to the editor.

In edit mode a full-screen editor is available and operates in a similar manner to that seen in Sharp BASIC 1.0A. Existing program lines can be modified using the cursor keys to navigate after a program has been listed (L) to the screen.

Return to the # prompt and use the I command to add new lines to the end of the program.

Fortran commands and program structure

Fortran programs are structured in the following way:

n: <Optional Label> <Statement>

Where n: is an internal line number that the compiler generates automatically, starting from line 0. It is not entered as part of the program code.

Only one statement per program line is allowed.

Normally there must be 6 spaces between the internal line number and the start of a statement. However spaces 2 - 5 can be replaced by a numeric label between 1 and 9999 to facilitate control structures, such as DO loops and GOTO statements.

If the first character on a line is C then whatever follows it is treated by the compiler as a comment.

The compiler supports implicit typing only. Variables are of type INTEGER or REAL. If a variable name starts with I, J, K, L, M or N the variable is treated as an INTEGER. All other variables are treated as REAL.

Integer variables must be in the range from -32768 to +32767.

An example program to output all 256 primary characters by directly addressing the screen from memory location 53648 onwards (about halfway down the screen).

```
0:      DO 9999 I=0,255
1:      MEM(53648+I)=I
2: 9999 CONTINUE
3:      END
```

Real variables are in the range from 1.67E-19 to 1.67E+18 and can be negative, positive, or zero.

Variable names can be of any length, but the complier only takes into account the first 4 characters in determining if a variable is unique. For example, the variables ABCD, ABCDE and ABCDEF are all treated as referring to the same value.

The arithmetic operators (+,-,*,/) are the same as in Sharp BASIC. However, variables of different types cannot be combined directly using these operators (see the FLOAT function).

Assignments use '=' as in Sharp BASIC, however the LET keyword is not used.

Abbreviations:			
	exp	-	Integer expression
	fxp	-	Real (floating point) expression
	v	-	Variable name
	n	-	Integer constant
	hx	-	Hexadecimal number

DIMENSION: Dimensioning of variables, e.g. DIMENSION A(5), I(7) results in two arrays: A REAL array with the elements A(1)-A(5)and an INTEGER array I(1)-I(7). The DIMENSION statement can only be used at the start of a program before any other statements (except comments).

GOTO label: Jump to the specified label.

IF exp label 1, label 2, label 3: If the integer expression exp is negative the program branches to label 1. If the expression is zero the program branches to label 2. If the expression is positive the program branches to label 3.

Labels can be omitted if required. For example, IF (I-5),,1000 means that the program jumps to label 1000 if I is greater than 5.

DO label v = exp 1, exp 2, exp 3: This is a loop statement and counts from exp 1 to exp 2 in (optionally) steps of size exp 3.

When the program reaches the specified label, the loop is repeated until exp 2 is satisfied. The specified label must always be a CONTINUE statement.

CALL label: Call the subroutine defined at this label.

RETURN: Return from a subroutine to continue execution from the line after the originating CALL statement.

PAUSEn: The program will stop and report PAUSE n issued. The program will resume execution once a key is pressed.

STOPn: Program execution is interrupted and the message RESTART? is issued. Y = program execution resumes, N = control is returned to the compiler.

END: Program execution terminates and the message RESTART? is issued. Y = program execution restarts, N = control is returned to the compiler. While a label can be placed before an end statement, other statements (such as GOTO) cannot use this label.

BREAK: The program can only be resumed after a BREAK statement by pressing the SHIFT & BREAK keys together.

USR(exp): Execute a machine language subroutine (as Sharp BASIC).

$ML hx1,hx2, ..: Integrates machine language commands with Fortran.

For example, $ML 11,A3,11,CD,30,00 plays music at the frequency specified in memory location $11A3.

READ(v1.xx,v2.xx, ..): The READ command corresponds to the Sharp BASIC INPUT command. The formats allowed are:

- v.I: Decimal integer value
- v.B: Hexadecimal integer value
- v.E: Decimal real value
- v.A1: A single character is read and its Sharp character code is stored
- v.A2: Two characters are read in and the high and low bytes are stored

For example, READ ("Enter J:", J.I) prints the message “Enter J” to the screen and stores the input as an integer, J.

WRITE (exp1.xx, exp2.xx, ..): The write command is used for screen or printer output. The formats allowed are:

exp	Decimal output
exp.In	Integer expression I output to a field of length n-1
fxp.E	Real expression output (*)
exp.B2	Hexadecimal output to a field of length 2
exp.B4	Hexadecimal output to a field of length 4
exp.X	Output X spaces
exp.A1	Single character output
exp.A2	Two character output
exp.V	Vertical position of the output on screen
exp.H	Horizontal position of the output on screen

Examples:

```
WRITE (/,12.H,16.V, "BBG-SOFTWARE")
/ means line feed; output starts at screen position
row 12, column 16.
WRITE ($16.A1) corresponds to CLEAR+HOME
WRITE ($15.A1) corresponds to HOME
WRITE (1.A1)   Send output to printer
WRITE (0.A1)   Send (reset) output to screen
```

SETG (exp1, exp2): SET and RESET graphic points with the corresponding
RESG (exp1, exp2): x(exp1)and y (exp2)coordinates.

MEM (exp1) = exp2: Corresponds to Sharp BASIC POKE expl, exp2

v=MEM (exp): Corresponds to Sharp BASIC v=PEEK(exp)

I=GET: Corresponds to Sharp BASIC GET. I contains the character value of the key pressed. If no key is pressed, then I=0.

IOC (exp1) = exp2: Like MEM, but accesses the Z-80 ports

LOW (exp): Return the low order byte of expression exp

MOD (exp1, exp2): MOD returns the integer modulo division of exp1 by exp2

IABS (exp): Absolute value of exp

ISIGN (exp1, exp2): If exp1 is greater than exp2, exp1 is returned. In all other cases –exp1 is returned.

ABS (fxp): As IABS but for real numbers

SQRT (fxp): Returns the square root of a real number

SIN (fxp): Sine in radians
COS (fxp): Cosine in radians
TAN (fxp): Tangent in radians
ATAN (fxp): Arctangent (inverse tangent) in radians

ALOG (fxp): Natural logarithm
EXP (fxp): Exponent

FLOAT (exp): Convert an integer value to a real value
IFIX (fxp): Convert a real value to an integer value

```
IOR (exp1, exp2):      Logical OR
IAND (exp1, exp2):     Logical AND
IXOR (exp1, exp2):     Logical XOR (exclusive or)
```

Error Messages

```
ERROR 1:                        Unrecognised editor command
ERROR 2:                        Illegal variable name
ERROR 3:
ERROR 4:                        Dimension statement is not at the start of
                                the program
ERROR 5:                        Syntax error
ERROR 6:                        More than six levels of nested loops found
ERROR 7:                        Illegal array variable name
ERROR 8:                        Array dimension error (more than 2047
                                elements)
ERR DO LOOP:                    Loop failure
MEMORY SIZE OVER ABORT:         Program is too large
ST NO NOT FOUND:                Label for DO, GOTO, IF or CALL not found
FILE ERROR ABORT:               No program is available to compile
OBJECT PROGRAM NOTHING !:       No compiled program available for RUN to
                                execute
```

The following errors can occur during program execution:

```
ARRAY ERROR:                    A dimensioned variable is out of range
0 ERROR:                        REAL variable greater than 1.67E+18
U ERROR:                        REAL variable underflow
N ERROR:                        Illegal argument for maths function - for
                                Example, SQRT(-1.0)
D ERROR:                        Division by zero
```

Miscellaneous information

Warmstart location for RealFortran from the monitor (without deleting the program) is $122A

Coldstart location for RealFortran from the monitor is $1200

Sample Programs

Here is a program that divides two real numbers. Note that A and B are stored as real variables as they are not in the range I-N (which would define them as integers). In edit mode, L can be used to list the program:

```
#L

0:      A=11
1:      B=22
2:      C=B/A
3:      WRITE(C.E)
4:      KL=IFIX(C)
5:      WRITE(/,KL)
6:      END
```

To enter compiler mode, use !

```
#!
OK.
```

To compile the program, use C. By default, the program will be output to the screen during compilation (unless the LN or LE commands have been issued).

Zero errors should be reported, and the compiled program will consume 125 bytes.

Using R to execute the program will result in the following output:

```
0.200000E+1
2
```

A program to print out every character (codes 1-255) to fill the screen.

```
0:C THIS LINE IS A COMMENT
1:      DO 2 MM=1,255
2:      DO 4 KL=53248,54247
3:    4 MEM(KL)=MM
4:    2 CONTINUE
5:      END
```

All programs must finish with the END statement.

Values to be printed with the WRITE statement must be enclosed in brackets, as must values being input with the READ statement.

Remember not to confuse real and integer variables! Integer variables must always start with a letter in the range I-N inclusive, and real variables must always start with letters in the ranges A-H or O-Z inclusive.

A program to print out prime numbers to a specified limit.

```
 0:C     PRIME NUMBER GENERATOR
 1:C     TIM HOLYOAKE 12/06/2021
 2:C
 3:      DIMENSION IP(255)
 4:      WRITE(/,"Prime no. Limit")
 5:      READ("( < 32767 ) ? ",L.I)
 6:      WRITE(//,". 2 3")
 7:      IP(1)=3
 8:      N=1
 9:      IX=5
10:  100 M=IFIX(SQRT(FLOAT(IX))+0.5)
11:      DO 200 J=1,N
12:         IW=IP(J)
13:         IF (IW-M),,300
14:         IF (IX-IW*(IX/IW)),500,
15:  200 CONTINUE
16:  300 N+N+1
17:      IF (N-255),,400
18:      IP(N)=IX
19:  400 WRITE(" .",IX)
20:      GOTO 600
21:  500 WRITE(" . .")
22:  600 IX=IX+2
23:      IF (IX-L) 100,100,700
24:  700 WRITE(//,N+1," primes",/)
25:      END
```

Manual translated, corrected and clarified from the original German by Tim Holyoake, June 2021.