BASIC III Including ISAM, MIMER and Window Handler

Programming Card

ABC 1600®

- Conforms to the ANSI X3.60-78 standard with many extensions.
- Long variable names and labels.
- Multiline IF-THEN-ELSE control structure.
- Multiline recursive functions and procedures with local variables.
- Open pipe statement for effective fork handling.
- Request statement for access to all system calls.
- IEEE standard floating point arithmetic with 32 bit trigonometric functions.
- 32 or 16 bit integers with automatic conversion to/from float for maximum execution speed.
- Extended precision fix and floating point decimal string arithmetic.
- Advanced file handling with path/name translation tables.
- B-tree ISAM option for simple database handling.
- MIMER relational database management system.
- Window handler allowing up to 16 programs run simultaneously.

LUXOR DATORER

Instructions

BYE

Terminates utilization of the Basic III interpreter. Closes all files.

```
100 BYE
```

CHAIN ". . . . . ."

Loads the specified program from a disk to the memory and starts program execution. Provides the same effect as RUN . . . See also COMMON . . .

```
20 CHAIN "CAT.007"
60 CHAIN "PROGRAM2"
90 CHAIN A¤
```

CLOSE . . .

Closes the specified file (files), i.e. terminates reading from or writing into the file. If the file has been opened for writing, an end-of-file mark is written. If the file number is omitted, all open files are closed.

```
50 CLOSE 4
60 CLOSE 1, 4
70 CLOSE
```

COMMON . . .

The specified variables are not cleared when CHAIN ". . . . . ." is executed. Program lines containing COMMON . . . . . must appear first in the program.

```
10 COMMON X, Y, Z
20 COMMON X¤ (30) = 100, Y¤ (20) = 200
```

DATA . . .

Stores the variable values that are to be read by a READ-instruction. Strings which contain commas must be within quotation marks.

```
40 DATA 13, 2.8, EVE
50 DATA "Signed, SAM SMITH"
```

DEF FN . . . (. . .) = . . .

Defines a single-line user function.

```
70 DEF FNY(R) = PI * R * R
80 DEF FNZ(B, H) = (B * H)/2
90 DEF FNX¤(S¤) = LEFT¤(S¤, LEN(A¤)-2)
```

DEF FN . . . (. . .) LOCAL . . .

Defines a multi-line user function. LOCAL . . . defines local variables. LOCAL . . . can be omitted. The value of the function is returned by means of RETURN . . . The FNEND instruction terminates the definition.

```
100 DEF FNY(R) LOCAL S
200 S = R/180
300 RETURN S*R
400 FNEND
```

DIGITS . . .

Limits the number of digits that are printed out when variable values are printed out by means of PRINT . . . Has no effect on the accuracy of calculation.

```
10 DIGITS 4
70 DIGITS N
```

Instructions cont.

DIM . . .

Reserves spaces for field variables. Normally, the minimum index is 0, but this can be changed to 1 by means of OPTION BASE 1

```
10 DIM P(20)
```

Reserves spaces for 21 floating point variables, P(O) through P(20) inclusive.

```
20 DIM P¤ = 1000
```

Reserves space for a string containing a maximum of 1000 characters.

```
30 DIM P¤(20) = 10
```

Reserves space for 21 string variables, each containing a maximum of 10 characters.

```
40 DIM P%(10,20,5)
```

Reserves space for a matrix containing 10 by 21 by 6 integer variables.

```
50 DIM P¤(10,20) = 40
```

Reserves space for a string matrix having 11 rows and 21 columns in which each element can contain a maximum of 40 characters.

DOUBLE

Causes a floating point number to be presented by 16 digits rather than the normal seven. This instruction must appear first in the program. See also SINGLE.

END

Terminates the program execution and closes all files.

EXTEND

Permits variable names containing additional significant characters to be used. See also NO EXTEND.

```
10 EXTEND
20 Gross price = Net price * 1.25
```

FIELD . . . IN . . . AT . . . COUNT . . .

Defines a substring in a string.

```
10 FIELD Name¤ IN Posf¤ AT 1% COUNT 20%
```

FLOAT

Causes variables and constants to be handled as floating point numbers. Integer variables and integer constants are flagged with the % character. FLOAT must appear first in the program. See also INTEGER.

```
10 FLOAT
20 A = 2.567 : B% = 9%
```

FNEND

Terminates the definition of a multi-line user function. See DEF FN . . . (. . .)

FOR . . . TO . . . STEP . . .

NEXT . . .

Executes a section of a program the number of times specified by the initial and final values of the loop variable and the step value. If the step value is omitted, the default value is 1.

```
20 FOR K = 1 TO 20 STEP 2
 .
 .
40 NEXT K
60 FOR L% = A% * 3% TO B%
 .
 .
90 NEXT L%
```

GET . . .

Execution does not take place until a key is depressed. The character that is entered is stored in the specified string variable but is not displayed on the screen. All characters can be accepted.

```
20 GET S7¤
```

GET # . . ., . . . COUNT . . .

Transmits data from a specified direct-access file to the specified string variable. The number of characters that are to be transmitted must be specified after COUNT. If COUNT . . . is omitted, only one character is transmitted. The file pointer in question is updated automatically. See also POSIT (. . .).

```
90 GET #1, S¤ COUNT 6
```

GOSUB . . .

Causes a jump to the subroutine located at the specified line. See also RETURN.

```
80 GOSUB 2000
```

GOTO . . .

Causes a jump to the specified line.

```
60 GOTO 10
```

IF . . . THEN . . . ELSE . . .

If the expression after IF is true (< >0), whatever is specified between THEN and ELSE is executed. Otherwise, whatever is specified after ELSE is executed. ELSE . . . can be omitted.

```
40 IF A = 0 THEN GOTO 60 ELSE 10
   (GOTO can be omitted after THEN and ELSE)
```

IF . . . THEN . . . ELIF . . . ELSE . . . IFEND

Multiline IF statement, with optional ELSE IF construct.

```
50 IF A = 2 THEN :C = 9
60 B = 5
70 ELIF A = 7 THEN
80 B = 6
90 ELSE B = 8
95 IFEND
```

Instructions cont.

INPUT "..." ...

Writes the specified prompting text on the display unit screen and then awaits the entry of data via the keyboard. The prompting text can be omitted.

```
30 INPUT "Enter name here" S¤
90 INPUT P
```

INPUT # ..., ...

Transmits data from the specified file to the specified variable.

```
50 INPUT # 2, R
60 INPUT # F, R¤
```

INPUT LINE ...

Transmits a line entered via the keyboard to the specified string variable. All characters can be accepted. The last 2 characters in the string variable are always CR (carriage return) and LF (line feed). All entered characters except for CR and LF are written on the display unit.

```
90 INPUT LINE S¤
```

INPUT LINE # ..., ...

Transmits data from the specified file to the specified string variable. All characters including spaces and commas and quotation marks are transmitted. The last two characters in the string variable are CR (carriage return) and LF (line feed).

```
60 INPUT LINE # 2, S¤
70 INPUT LINE # F, S¤
```

INTEGER

Causes variables and constants to be handled as integers. Floating point variables and floating point constants are flagged with a period. INTEGER must appear first in the program. See also FLOAT.

```
10 INTEGER
20 A. = 2.567 : B = 9
```

KILL "....."

Erases the specified file from the disk.

```
30 KILL "TEST.bac"
90 KILL A¤ + "TEST.TXT"
```

LET ...

Assigns a value to a variable. LET can be omitted.

```
10 LET X = 5 * S
20 T¤ = "KATE"
```

LOCK # ...

Locks the specified direct access file. The file pointer is not updated.

```
30 LOCK # 1%
   (Locks 1% number of bytes)
```

NAME "....." AS "....."

Changes the name of a file.

```
90 NAME "OLD.bas" AS "NEW.bas"
```

NEXT ...

Terminates a loop. See also FOR ... TO ... STEP ...

Instructions cont.

NO EXTEND

Cancels the EXTEND instruction. When this occurs, only variable names consisting of a letter or a letter combined with one of digits 0-9 can be used.

NO TRACE

Cancels the TRACE instruction. See also TRACE.

ON ERROR GOTO . . .

Causes a jump to the error handling routine located at the specified program line if an error occurs during a run. See also RESUME. If no line number is specified, ON ERROR GOTO . . . cancels any previous ON ERROR GOTO . . .

```
10 ON ERROR GOTO 1000
```

ON . . . GOSUB . . .,. . .

Causes a jump to one of the specified subroutines. See also RETURN.

```
60 ON P GOSUB 100, 200, 300
```

(P = 1 causes a jump to line 100, P = 2 causes a jump to line 200, etc.)

ON . . . GOTO . . .,. . .

Causes a jump to one of the specified program lines.

```
70 ON R GOTO 100, 200, 300
```

(R = 1 causes a jump to line 100, R = 2 causes a jump to line 200, etc.)

ON . . . RESTORE . . .,. . .

Causes one of the specified RESTORE . . . instructions to be executed. See also RESTORE . . .

```
90 ON S RESTORE 100, 200, 300
```

(S = 1 causes RESTORE 100 to be executed, S = 2 causes RESTORE 200 to be executed, etc.)

ON . . . RESUME . . .

Causes the specified RESUME . . . instruction to be executed. See also RESUME.

```
80 ON T RESUME 100, 200, 300
```

(T = 1 causes RESUME 100 to be executed, T = 2 causes RESUME 200 to be executed, etc.)

OPEN ". . . . . " AS FILE . . .

Opens the specified file for reading and assigns the file a file number.

```
40 OPEN "ADDRESS.TXT" AS FILE 3%
60 OPEN "/src/lie/a" AS FILE 4%
80 OPEN A$ + ".TXT" AS FILE F%
85 OPEN "PIPEIN: echo*" AS FILE 1
90 OPEN "PIPEOUT: print" AS FILE 5
```

OPTION BASE . . .

Set the minimum index for field variables to 0 or 1. See DIM . . . OPTION BASE . . . must appear first on the program.

OPTION EUROPE

Specifies the format when using PRINT USING.

Instructions cont.

POSIT # . . .,. . .

Specifies the position in a direct access file where reading/writing is to take place. See also the POSIT(. . .) function and the GET # . . ., . . . and PUT # . . ., . . . instructions.

PREPARE ". . . . . " AS FILE . . .

Creates and opens a specified file for writing and assigns the file a file number.

```
30 PREPARE "dr1/address.txt" AS FILE 3
50 PREPARE A$ + ".TXT" AS FILE F
```

PRINT . . . (or ; . . .)

Writes on the display unit screen. Can be followed by an arithmetic expression and special functions TAB(K) and CUR(R,K).

```
20 PRINT "JANE"
30 PRINT A, B, C    (Printout in columns)
40 ; "AREA = " ; Y  (Continuous printout)
```

PRINT # . . .,. . . (or ; # . . .)

Writes in the specified file.

```
30 PRINT # 2, "RANDOM NUMBER"
```

PRINT USING ". . ." . . .

Causes the specified expression to be presented in the desired format. See BASIC III Manual.

```
20 PRINT USING " + # # # .# # " A;B
```

PUT # . . .,. . .

Transmits the specified string variable to the specified direct-access file. The file pointer in question is updated automatically. See also POSIT(. . .).

```
50 PUT # 1, S$
```

RANDOMIZE

Provides the RND function with a random initial value.

READ . . .

Assigns to variable the data obtained from DATA statements.

```
10 READ B, C$
```

REM . . . (or ! . . .)

Used for comments in a program. The colon before the ! is not required.

```
80 REM ***SUBROUTINE***
90 A = 5 ! initial value
```

REPEAT

UNTIL . . .

A section of a program is executed repeatedly as long as the specified condition is not true.

```
10 REPEAT
20 I = I + 1; I
30 UNTIL I = 10
```

RESTORE . . .

Causes the next READ statement to read data starting at and including a specified DATA statement.

```
50 RESTORE
```

(Reads data starting at and including the first DATA statement)

```
70 RESTORE 100
```

(Reads data from DATA statements starting at and including line 100.)

RESUME . . .

Causes a return jump from an error handling routine to the specified program line. If the line number is omitted, the return jump is made to the program line on which the error occurred.

```
80 RESUME 10
90 RESUME
```

RETURN

Causes a return jump from a subroutine to the statement following the corresponding GOSUB statement.

RETURN . . .

Returns a function value in a multi-line function. See also DEF FN . . . (. . .)

SINGLE

Causes floating point numbers to be represented with seven digits. Cancels the DOUBLE instruction.

STOP

Interrupts a program execution.

TRACE

The line numbers of executed program lines are written on the display unit screen or in a file (TRACE# . . .) during a program execution. See also NO TRACE.

UNLOCK# . . .

Unlocks the specified direct access file. The file pointer is not updated. (See also LOCK).

WHILE . . .

WEND

A section of a program is executed repeatedly as long as the specified condition remains true.

```
10 WHILEX < 10
20 PRINT X : X = X + 1
30 WEND
```

STRING FUNCTIONS

ASC(A¤) or ASCII(A¤)

Provides the ASCII value for the first character in the specified string.

```
20 LET P = ASC (B¤)
```

Instructions cont.

CHR¤ (. . .)

Provides a string containing the characters corresponding to the specified ASCII values.

```
10 PRINT CHR¤(12)
20 P¤ = CHR¤(P, R, S, T, U, V)
```

INSTR(S, A¤, B¤)

Searches for substring B¤ in string A¤ starting at position S. This function provides the location of the first occurrance after S of B¤ in A¤. If B¤ is not found in A¤, a value of 0 is obtained.

```
30 R = INSTR(S, A¤, B¤)
```

LEFT¤(A¤, S) or LEFT(A¤, S)

Provides the first S character in the specified string.

```
30 D¤ = LEFT¤(C¤, S)
```

LEN(A¤)

Provides an integer which corresponds to the length of the specified string.

```
20 X = LEN(A¤)
```

MID¤(A¤, S, T) or MID(A¤, S, T)

Provides T characters starting at and including character position S in the specified string.

```
30 B¤ = MID¤(A¤, S, T)
```

MID¤(A¤, S, T) = ". . ."

Replaces T characters starting at and including character position S in string A¤ by the string specified within quotation marks.

```
30 MID¤(A¤, 5, 4) = "NEW"
```

NUM¤(S)

Provides a numeric string containing the same characters as those that would be obtained if the specified numeric variable were written using a PRINT statement. The blank character that is reserved for the plus sign is not included in NUM¤(S).

RIGHT¤(A¤, S) or RIGHT(A¤, S)

Provides all characters in the specified string starting at and including character position S.

```
40 B¤ = RIGHT¤(A¤, S)
```

SPACE¤(S)

Provides a string consisting of the specified number of spaces.

```
20 B¤ = SPACE¤(253)
```

STRING¤(S, T)

Provides a string of S characters having ASCII value T.

```
30 C¤ = STRING¤(20, 45)
```

VAL(A¤)

Converts a specified numeric string to a numeric variable.

```
10 S = VAL(A¤)
20 S = VAL(".000000003")
```

+

Concatenates strings into a string expression.

```
30 S¤ = A¤ + B¤ + ".bas"
```

Expressions and Operators

Priority	Operator	Explaination	Example
1	** or ü	raising to a power	A**B
2	*	multiplication	A * B
2	/	division	A / B
3	+	addition	A + B
3	—	subtraction	A — B
4	>	greater than	A > B
4	<	less than	A < B
4	=	equal to	A = B
4	>=	greater than or equal to	A >= B
4	<=	less than or equal to	A <= B
4	<>	not equal to	A <> B

Logical Operators

The logical operators are presented below in priority sequence. NOT has the highest priority. OR and XOR have the same priority. All logical operators have a lower priority than other operators.

NOT

NOT is true if the operand is false.

```
10 IF NOT A < B THEN GOTO 20
```

AND

AND is true if both operands are true.

```
20 IF A > B AND C = D THEN 30
```

OR

OR is true if at least one of the operands are true.

```
30 IF A < B OR C = 10 THEN 40
```

XOR

EXCLUSIVE OR is true if either of the operands is true, but not both.

```
40 IF A = B XOR C = D THEN 50
```

IMP

IMPLICATION operates in such a way that A IMP B is false only if A is true and B is false.

```
50 IF A IMP B THEN 60
```

EQV

EQUIVALENCE is true if both operands are true and also if both operands are false.

```
60 IF A = B EQV C = D THEN 70
```

The logical operators can also be used on arbitrary integers. In such case, the operators are implemented bit by bit on the two binary numbers.

```
70 A% = B% AND 15%
```

ASCII Functions

ADD¤(A¤, B¤, T)

Adds the specified numeric strings. The result is rounded to the specified number (T) of decimals. The numeric strings can contain digits, the + and — characters and the decimal point.

```
20 C¤ = ADD¤(A¤, B¤, T)
```

DIV¤(A¤, B¤, T)

Divides one numeric string (A¤) by another (B¤). See ADD¤(A¤, B¤, T)

```
30 C¤ = DIV¤(A¤, B¤, 10)
```

MUL¤(A¤, B¤, T)

Multiplies one numeric string (A¤) by another (B¤). See ADD¤(A¤, B¤, T).

```
40 D¤ = MUL¤(A¤, B¤, T)
```

SUB(A¤, B¤, T)

Subtracts one numeric string (B¤) from another (A¤). See ADD¤(A¤, B¤, T).

```
50 C¤ = SUB¤(A¤, B¤, 2)
```

COMP%(A¤, B¤)

Compares two numeric strings. This function provides the following values as results:

```
-1 if A¤ < B¤
 0 if A¤ = B¤
+1 if A¤ > B¤
10 P = COMP%(A¤, B¤)
```

Mathematical Functions

ABS(X)	Absoulte value of x, $\|x\|$
ATN(X)	arctan x
COS(X)	cos x, x in radians
EXP(X)	e^x
FIX(X)	integer part of x, [x]
INT(X)	largest integer less than or equal to x
LOG(X)	logarithm of x to the base e
LOG10(X)	logarithm of x to the base 10
MOD(X, Y)	remainder of X/Y division
SGN(X)	-1 if $x < 0$, 0 if $x = 0$, +1 if $x > 0$
SIN(X)	sine x, x in radians
SQR(X)	square root of x, $\sqrt{x}$
TAN(X)	tan x, x in radians
PI	π, 3.14159 (or 3.14159265358979)
RND	represents a random number ranging from 0 to 0.999999 (or 0 to 0.9999999999999999)

Special Functions

ARGC%

Returns the number of parameters the basic interpreter was started with. ARGC% is used in combination with the function ARGV¤ to retrieve the startup parameters. See also ARGV¤.
If the basic was started:

basic — x format fill fil2

ARGC% will return 5, since there are five space separated strings in the start command.

ARGV¤

Returns the parameters the basic interpreter was started with. ARGV¤ is used in combination with the function ARGC% to retrieve the startup parameters.
If the basic was started:

basic -x format fill fil2

ARGC% will return 5, since there are five space separated strings in the start command. ARGV¤(1) will return 'basic' (the 5 character string: basic)

ARGV¤(2) will return '-x'
ARGV¤(3) 'format'
ARGV¤(4) 'fill'
ARGV¤(5) 'fil2'

CUR (R,K)

Moves the cursor to line R and column K on the screen. Can be used only in PRINT statements.

```
20 PRINT CUR (0, 26) "TOP RIGHT"
```

CVT . . . . . .(. . .)

Permits character conversion in connections with file reading/writing. The following variants can be used, where R% is an integer and T is a floating point number.

CVT%¤ (R%) R% is converted to a string (2 or 4 bytes)
CVT¤% (S¤) S¤ is converted to an integer
CVT F¤ (T) T is converted to a string
CVT¤ F (S¤) S¤ is converted to a floating point number

```
20 PUT #1, CVT F¤ (T)
```

ERRCODE

Provides the most recent error code.

```
30 E9 = ERRCODE
```

HEX¤(. . .)

OCT¤(. . .)

converts the specified decimal number to a hexadecimal or octal number in the form of a numeric string.

```
50X¤ = HEX¤ (128) : Y¤ = OCT¤ (83)
```

POSIT (F)

Provides the content of the file pointer associated with the file having number F.

```
50 PRINT POSIT (3).
```

Special Functions cont.

REQUEST(. . .,. . .,. . .)

Gives access to all ABCenix system calls.

SHORT INT

LONG INT

Specifies the integer size to use when using the string to integer and integer to string functions CVT%¤ and CVT¤%. The default is LONG corresponding to 4 bytes, SHORT is 2 bytes.

SYS(. . .)

Provides the value of the specified system variable. The system variables are listed below:

SYS(2)	Total user memory capacity (bytes)
SYS(3)	Size of program (bytes)
SYS(4)	Unused memory area (bytes)
SYS(11)	Starting address of program
SYS(12)	Address of first variable's name

TAB(T)

Moves the cursor to position T on the current line. Can only be used in PRINT statements.

```
40 PRINT TAB (14);"**Headline**"
```

TIME¤

Provides the date and time of day obtained from the system clock.

```
20 PRINT TIME¤
```

VAROOT(A¤)

Provides the address of a table that contains information about variable A¤.

```
30 PRINT VAROOT(A¤)
```

VARPTR(A¤)

Provides the starting address of the area where the value of variable A¤ is stored.

```
40 PRINT VARPTR(A¤)
```

Graphic Statements (ABC 1600 or ABC 806 with graphic PROM)

For all the statements the 'colnr' and 'pattern' parameters are optional, if not specified the last used values are maintained. For monochrome terminal (ABC 1600) 'colnr' = 1 and 'pattern' = 0 are used to draw complete lines, arcs, etc. 'colnr' = 0 and 'pattern = 0 are used to clear.

FGFILL x, y [,colnr] [,pattern]

Fill rectangular area with opposite corners in previous graphic cursor position and 'x', 'y'.

Ex.:

```
FGFILL 100,100,1,1
```

FGLINE x,y [,colnr] [,pattern]

Draw line from previous graphic cursor position to 'x','y'.

Ex.:

```
FGLINE 100,100,1,1
```

Special Functions cont.

FGPOINT x,y [,colnr] [,op]

Move graphic cursor to 'x' , 'y' and alter pixel according to 'op': 0 set pixel.
1 clear pixel
2 complement pixel

Ex.:

```
FGPOINT 100,100,1,2
```

FGPAINT x, y [,colnr] [,pattern]

Start paint from 'x', 'y'. After operation, graphic cursor is left in 'x', 'y'. If 'pattern' is 0 a complete "go around the corner" paint is done. Otherwise, it just starts at 'x, 'y' and goes outwards.

Ex.:

```
FGPAINT 100,100,1,1
```

FGCSEG x, y, len [,colnr] [, pattern]

Draws circle segments from graphic cursor position counter clockwise with origo in 'x', 'y'. The "length" of the segment is specified with 'len' in number of vertical and horizontal pixel steps. This means that a full circle is generated with len = 8* radius.

Ex.

```
FGCSEG 100, 100, 300, 1, 1
```

Memory Access and I/O Ports

CALL(A)

Call subroutines stored in the machine language at the specified address (A).

```
10 S = CALL(49800)
```

CALL(A, U)

Calls the subroutine stored in machine language at the specified address (A). Before the subroutine call the register of the microprocessor is loaded with the specified expression (U). After the subroutine has been executed, CALL (A, U) is equal to the number stored in the register of the microprocessor.

```
20 B% = CALL(49800%, U%)
```

INP(P)

Fetches a byte from the specified port (P).

```
30 C% = INP(34)
```

OUT P1, D1, P2, D2, . . .

Transmits data D1, D2, . . . to the specified ports P1, P2, . . .

```
40 OUT 58, 32
```

PEEK(A)

Fetches a byte from the specified memory address (A).

```
50 PRINT PEEK(49800)
```

PEEK2(A)

Fetches two bytes from the specified memory address.

```
40 B% = PEEK2(VARPTR(A¤)
```

Memory Access and I/O Ports cont.

PEEP4(A)

Fetches four bytes from the specified memory address.

```
40 B% = PEEK4(32000)
```

POKE A, D1, D2, . . .

Transmits data D1, D2, . . . to the specified memory cells starting at and including the specified address (A). Data and addresses are specified as decimal values.

```
60 POKE 65008, 10, 5, 2
```

SWAP%(D)

Provides the value of D in a 2-byte integer but with the first and second bytes swapped.

```
70 B% = SWAP%(D%)
```

SWAP2%(D)

Provides the value of D in a 2-word or 4-byte integer but with the first and second 16 bit's swapped.

```
70 B% = SWAP 2%(D%)
```

ISAM Options

ISAM DELETE #. . .,. . . (or ISDE #. . .,. . .)

Deletes a record from an ISAM index file.

```
30 ISAM DELETE # 1,A¤
```

ISAM OPEN ". . .,. . .,. . ." AS FILE . . . (or ISOP . . . AS FILE . . .)

Opens and ISAM index file and its associated data file.

```
10 ISAM OPEN "dfile" AS FILE 1
```

ISAM READ #. . .,. . . INDEX ". . ." KEY ". . ." (or ISRE # . . .)

Accesses an ISAM data file. See BASIC III Manual. KEY may be replaced by FIRST LAST NEXT PREVIOUS (or PREV).

```
20 ISRE # 1,A¤ INDEX "nfile" KEY "smith" ! smith
   record
30 ISRE # 1,A¤ FIRST ! Reads FIRST in INDEX file
40 ISRE # 1,A¤ NEXT ! Reads NEXT in INDEX file
50 ISRE # 1,A¤ ! sequential read
```

ISAM UPDATE #. . .,. . . TO . . . (or ISUP #. . .,. . . TO . . .)

Modifies an existing record in the data file associated with an ISAM index file. See Basic III Manual.

```
20 ISAM OPEN "ifile" AS FILE 1
30 ISAM READ # 1,A¤ INDEX "name" KEY "jones"
40 ISAM B¤ = "jones new york 727-2677"
50 ISAM UPDATE # 1,A¤ TO B¤
```

ISAM WRITE #. . .,. . . (or ISWE #. . .,. . .)

Enters a new record into the data file and updates all indices in the index file. See Basic III Manual.

```
50 ISAM WRITE # 1,A¤
```

MIMER DATABASE Options

MIMER BEGIN (string 1), (string 2) (or MIMBE . . ., . . .)

Start MIMER session. String 1 is the user name and string 2 is the password associated with the user name in MIMER.

```
10 MIMER BEGIN "USER1", Passwd¤
```

MIMER OPEN "(databank).(table)" AS FILE nr [ACCESS ac] column1, column2, . . . (or MIMOP" . . . . ." . . . . .)

Opens MIMER databank and table. Connects the specified MIMER columns with the corresponding variables in a MIMER GETFIRST, MIMER GETNEXT, MIMER WRITE or MIMER UPDATE statement.

The ACCESS values are:

ac	databank	table
RR	Read	Read
SR	Shared	Read
SS	Shared	Shared
XX (default)	Exclusive	Exclusive

```
10 Regnr¤ = ""
20 MIMER OPEN "DBL.car" AS FILE 7 ACCESS RR
30 Regnr¤, "model", "year", "color"
```

MIMER GETFIRST #nr,. . ., . . ., . . . (or MIMGF #nr,. . ., . . ., . . .)

Reads a row of data from the MIMER table associated with 'nr'.

```
30 MIMER GETFIRST #7,,Regnr¤,Mode¤,Col¤
   ,Year¤
```

MIMER GETNEXT #fnr, . . ., . . ., . . . (or MIMGN #fnr, . . ., . . ., . . .)

Reads the next row in a MIMER table.

```
50 MIMER GETNEXT #7 ,Regnr1¤,Model1¤,Col1¤
   ,Year1¤
60 MIMER GETNEXT #7 ,Regnr2¤,Model2¤,Col2¤
   ,Year2¤
```

MIMER WRITE #fnr , . . ., . . ., . . . (or MIMWR #fnr ,. . ., . . ., . . .)

Insert a new row in the MIMER table.
Enter a new row in table CAR

```
10 Regnr¤ = "ACC123"
20 Model¤ = "Volvo 760"
30 Col¤ = "Grey
40 Year¤ = "84"
50 MIMER WRITE #7 ,Regnr¤,Model¤,Col¤,Year¤
```

MIMER UPDATE #fnr ,. . .,. . . (or MIMUP #fnr ,. . .,. . .)

Change the current row in a MIMER table.

```
20 MIMER GETFIRST #7, "regnr" EQ "ABC123",
   Regnr¤,Model¤,Col¤,Year¤
30 Col¤ , = "Black"
40 MIMER UPDATE #7,Regnr¤,Model¤,Col¤
   ,Year¤
```

MIMER DATABASE Options cont.

MIMER DELETE #fnr (or MIMDE #fnr)

Delete a row in a MIMER table.
Find the car with registration number ABC123 and delete the row.

```
10 MIMER GETFIRST #7 , "regnr" EQ "ABC123"
   ,Regnr¤,Model¤,Col¤,Year¤
20 MIMER DELETE #7
```

MIMER TRANSACTION #fnr (or MIMTR #fnr)

Initiates transaction handling. The transaction handling works on databank level. See Basic III Manual.

```
30 MIMER TRANSACTION #1
```

MIMER COMMIT #fnr (or MIMCO #fnr)

All MIMER operations performed on the databank since MIMER TRANSACTION are made permanent.

```
40 MIMER COMMIT #2
```

MIMER ABORT #fnr (or MIMAB #fnr)

Do not make the changes. Transactions on intention are rolled back.

```
50 MIMER ABORT #2
```

MIMER END (or MIMEN)

Terminates a MIMER session, closing all tables and databanks.

```
10 MIMER BEGIN "USER1", Passwd¤
20 MIMER OPEN "DBL.car" AS FILE 7
30 MIMER GETFIRST #7 ,,Regnr¤,Model¤,Col¤,
   Year¤
40 PRINT Regnr¤; Model¤;Col¤;Year¤
50 CLOSE 7
60 MIMER END
```

Keyboard

CTL X — Erases the last entered line.

CTL C — Terminates the program execution that is in progress.
Can be followed by continue.

CTL Z — The program interrupts and one line is executed each time these keys are depressed.

[ ] — Program execution continues (depress any key).

The editor keys are described in the Basic III Manual.

!...

The remainder of the line after the '!' is sent to the shell to be interpreted as a command.

AUTO . . .,. . .

Cause lines to be numbered automatically as they are entered. Step: specifies the interval between line numbers.

AUTO (Line No : 10. Step : 10)
AUTO 100 (Line No : 100. Step : 10)
AUTO 500, 50 (Line No : 500. Step : 50)

CLEAR

Clears all variables and closes all files.

CONTINUE (or CON)

Causes the program to continue after the STOP instruction has been executed or after CTL C has been depressed.

ED . . . (or EDIT)

Makes it possible to change a program line without rewriting it.

ED 70

ERASE . . .

Erases the specified section of a program. This instruction has three variants:

ERASE 40-80 (Erases lines 40 through 80 incl.)
ERASE -80 (Erases lines upn to and incl. 80)
ERASE 40- (Erases all lines starting at and incl. 40)

LIST . . .

Causes the program stored in the memory to be written on the display screen. There are five variants:

LIST (All lines)
LIST 30 (Only line 30)
LIST 30-60 (Lines 30 through 60 incl.)
LIST -30 (Lines up to and incl. 30)
LIST 60- (Lines starting at and incl. 60)

LIST PR:

Provides printouts from the printer of programs stored in the memory.

LIST PR: (All lines)
LIST PR:, 10-90 (Lines 10 through 90 incl.)

LIST . . . . . .

Transmits the programs stored in the memory to a file on a disk in uncompiled form (text format). If file extension is omitted, the default extension is .bas

LIST subprg, 20-40
LIST sort.txt

Commands cont.

LOAD . . . . . .

Loads the program having the specified name from a disk to the memory. Erases the program already stored in the memory.

LOAD budget
LOAD estimate.001

MERGE . . . . . .

Loads a program in text format (stored by means of LIST . . .) from a disk to the memory without erasing the program already stored in the memory. (Erasure of the already stored program occurs only if the same line numbers are used in the loaded program.)

MERGE program 2

NEW

Erases the program stored in the memory and closes all files.

REN . . .,. . .,. . .-. . . (or RENUMBER . . .,. . .,. . .-. . .)

Assigns new line numbers to the specified program lines. There are a number of variants:

The following examples renumber all program lines:

REN (First line No. : 10. Step : 10)
REN 100 (First line No. : 100. Step : 10)
REN 100, 20 (First line No. : 100. Step : 20)

The following examples are for a specified program section in which the first line number is 600 and the step is 10.

REN 600, 10, 500-800 (lines 500 through 800 incl.)
REN 600, 10, -800 (lines up to and incl. 800)
REN 600, 10, 500- (lines starting with and incl. 500)

RUN

Runs the program stored in the memory after having cleared all variables.

RUN . . . . . .

Corresponds to the LOAD . . . . . command followed by RUN.
See also LOAD . . . . . The unit designation and file types .bac and .bas can be omitted.

RUN CAT.007

SAVE . . . . . .

Saves the program stored in the memory in a file on a disk in the form of semicompiled code. If the file extension is omitted, the default extension is .bac. See also UNSAVE. . . . . .

SAVE sort
SAVE sort.bac

STAT

Gives status of the Basic III i.e modes (INTEGER,EXTEND . . .) and program size.

UNSAVE . . . . . .

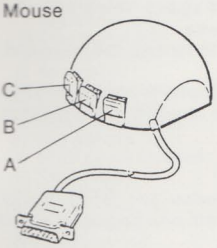
Erases the specified file from a disk. If the file type is omitted, the appropriate file type (.bac or .bas) is used.
See also SAVE . . . . . .

Window Handler

SHIFT CTRL PF15

Change cursor (▮) control to pointer (↑) control, and visa versa.

Mouse



Function Button A: Choose function when pointer points to an icon.

Function Button B: Copy information from one window to another.

Function Button C: Choose whick window is to be acitve (which window the user wishes to work in). Also used to page through all windows.

To control the pointer directly through the keyboard (without the mouse):

PF13 Serves as mouse button A.
PF14 Serves as mouse button B.
PF15 Serves as mouse button C.

Border Icons

Move the pointer to the desired icon and press mouse button A (or PF13). Note, some windows may not contain all or any border icons.

Close the window.

Move the entire window.

Move text down one line for each button press to see text which is above the displayed area.

Move entire text up or down instantly (vertically).

Move text up one line for each button press to see text which is below the displayed area.

Enlargen/reduce the size of a window.

Move text left one column for each button press to see text which is to the right of the displayed area.

Move entire text to the left/right instantly (horizontally).

Move text right one column for each button press to see text which is to the left of the displayed area.

Zoom: enlargens marked area for easier readability. Place pointer on the zoom icon, hold mouse button A (PF13) depressed and move the pointer to the area to be enlarged. When the button is released, the desired area is enlarged.

Restore the window to its original contents, position and size.

Window Handler cont.

To copy an area of text (a rectangle) to another window:
- Place pointer on the upper right character in the text and hold mouse button B depressed (or press PF14).
- Move pointer to the lower left character in the text and release mouse button B (or press PF14 again). The area to be copied is marked by a surrounding frame.
- Move pointer to the window that is to receive the copy and press mouse button B (PF14).
- The area marked with a surrounding frame is copied into the other window. Note that the text also remains in the original window.

To choose which window is to be active:
- Place the pointer on the desired window and press mouse button C (PF15). The chosen window will now be placed at the top of the pile and become active.
- If the chosen window is already on the top of the pile, it is then moved to the bottom of the pile and becomes inactive.

To page through all of the windows:
- Place the pointer in the field outside all windows and press mouse button C (PF15). This causes the window on the top of the pile to move to the bottom. Continue this procedure until the desired window is on the top of the pile (becomes active).

Error Messages

1 Not owner
2 No such file or directory
3 No such process
4 Interrupted system call
5 I/O error
6 No such device or address
7 Arg list too long
8 Exec format error
9 Bad file number
10 No children
11 No more processes
12 Not enough core
13 Permission denied
14 Bad address
15 Block device required
16 Device busy
17 File exists
18 Cross-device link
19 No such device
20 Not a directory
21 Is a directory
22 Invalid argument
23 File table overflow
24 Too many open files
25 Not a typewriter
26 Text file busy
27 File too large
28 No space left on device
29 Illegal seek
30 Read-only file system
31 Too many links
32 Broken pipe
33 Argument too large
34 Result too large
35 Structure needs cleaning
36 Would deadlock
37 Not a semaphore
38 Not available
39 File write protected
40 File delete protected
41 Disk full
42 Disk not ready
43 Disk write protected
44 Logic file not opened
45 Wrong logic file number
46 Wrong unit number
47 Wrong trap number
48 Error in library
49 Wrong physical file number
50 End of file
51 Too long line
58 Illegal character
64 Illegal NAME
68 Illegal time specification
100 MIMER: Relational operator expected
101 MIMER: Logical operator expected
103 MIMER: Access mode expected
110 MIMER: Too many columns
111 MIMER: Column name mismatch
112 MIMER: Wrong number of columns
113 MIMER: Not a MIMER file
114 MIMER: Projection problem
115 MIMER: Too many MIMER files
116 MIMER: Entry exists
117 MIMER: MIMER begin error
118 MIMER: Not logged in
120 ISAM: Can't find key
121 ISAM: Multiple key not allowed
122 ISAM: Wrong key
123 ISAM: Error on check-read
124 ISAM: Can't find index
125 ISAM: Wrong data record length
126 ISAM: Wrong version of ISAM file
128 ISAM: End of memory
130 Floating point overflow
131 Array index outside legal range
132 Integer overflow
133 Error in ASCII-arithmetic expression
134 String index neg. or too large
135 Negative TAB, SPACES, STRING$ arg
136 Overflow in string assign
137 Attempt to expand array or string
138 Expression out of range in ON
139 RETURN without GOSUB
140 Wrong return type
141 Out of DATA statements
142 Wrong argument to build-in function
143 Illegal SYS function
144 Illegal line
145 FNEND without previous RETURN
146 PRINT USING error
147 Illegal data
148 Too few data
149 RESTORE not to DATA line
150 Too many data
151 RESUME without error
152 Attempt to read from outpipe or write to inpipe
153 Open with untranslateable expression
176 Graphic point outside screen
180 Can't find line
181 Illegal GOTO into/out of function
182 NEXT, WEND, IFEND or UNTIL missing
183 FOR or WHILE missing
184 Wrong variable in NEXT
185 Illegal FOR, WHILE, REPEAT or mult in IF nesting
186 FOR-loop with local not allowed (use WHILE)
187 Function not defined
188 Several functions with same name
189 Illegal function
190 Wrong number of indices
191 Not assignable in function
192 REPEAT missing
193 IF missing (multiline type)
194 Multiple ELSE not allowed
200 Unit not connected
201 End of memory
202 Program LIST-protected
203 Illegal program format
204 Attempt to MERGE .bac file
205 COMMON must be first; CHAIN error
206 Use RUN
207 Can't continue
208 Not allowed as command
209 Wrong data to command
210 Wrong number
211 Precision can't be changed now
212 Compiler buffer overflow
220 Don't understand
221 Illegal character after statement
222 Must be first on a line
223 Wong number or types of argument
224 Illegal mix of number and strings
225 Not simple variable, index not allowed
226 Illegal statement after ON
227 "," missing
228 "=" missing
229 ")" missing
230 "AS FILE" missing
231 "AS" missing
232 "TO" missing
233 Missing line number
234 Illegal variable
235 # missing
236 IN, AT or COUNT missing

LUXOR
Datorer

Art.nr. 66 78400-91 Prod.: Tre-i-Reklam, Vingåker 1985