PRINT #/PRINT # USING

Format

```
PRINT # <file number>,[<list of expressions>]
PRINT # <file number>,USING<"format string">;<list of expressions>
```

Purpose

These statements are used to write data to a sequential disk file.

Remarks

<file number> is the number under which the file was OPENed for output; <"format string"> is specified as described in PRINT USING, and the expressions included in <list of expressions> are the numeric and/or string expressions which are to be written to the file. Both of the formats above write values to the disk in display image format; i.e., data is written to the disk in exactly the same format as it would be displayed on the screen. Therefore, care should be taken to ensure that data is delimited so that it will be properly input when the file is later read (see the descriptions of the INPUT # and LINE INPUT # statements).

Numeric expressions included in <list of expressions> should be delimited with semicolons; if commas are used, the extra blanks that would be inserted between display fields by a PRINT statement will be written to the disk. String expressions included in <list of expressions> must be separated from each other by semicolons; further, a string expression consisting of an explicit delimiter (a comma or carriage return) should be included between each expression which is to be read back into a separate variable by the INPUT # statement (the reason for this is that the INPUT # statement regards all characters preceding a comma or carriage return as one item). This can be done using one of the following formats.

```
PRINT #1, <string expression>;",";<string expression>..
PRINT #1<string expression>;CHR$(13);
<string expression>...
```

If a string which is to be read back into a variable with the INPUT # statement includes commas, significant leading blanks, or carriage returns, surround the corresponding expression in the PRINT # statement with explicit quotation marks (by specifying CHR$(34)). This can be done as follows.

```
PRINT #1,CHR$(34);"SMITH, JOHN";CHR$(34);CHR$(34);
"SMITH, ROBERT";CHR$(34);...
```

When the LINE INPUT# statement is to be used to read data items back into variables, delimit string expressions in <list of expressions> with CHR$(13) (the carriage return code) as shown in the example above.

See also INPUT#, LINE INPUT#, WRITE#, Chapter 5

PSET

Format PSET [STEP](X,Y)[,<color code>]

Purpose This statement turns on the dot at the specified graphic coordinates on the display screen.

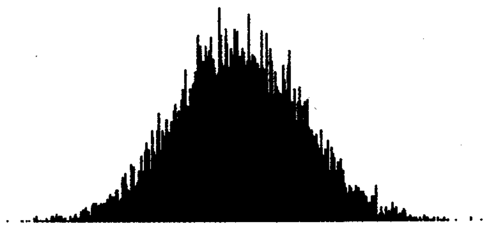
Remarks This statement sets the dot whose graphic coordinates are specified by (X,Y). When STEP is specified, (X,Y) specify a position relative to the last coordinates used by the preceding CIRCLE, COLOR, CONNECT, LINE, or PRESET statement. When <color code> is specified, the dot is set to the specified color; if it is omitted, the dot is set to the current foreground color.

See also CIRCLE, COLOR, CONNECT, LINE, PRESET

Example

```
10 CLS
15 DIM A(600)
16 LOCATE 20,1,0:PRINT "START:"TIME$
17 FOR I=0 TO 10000
18 LOCATE 1,1,0:PRINT I
20 X1=RND:X2=RND
30 X=(-2*LOG(X1))^.5*COS(2*3.1416*X2)
40 X=50*X+200
50 IF X>=0 AND X<=600 THEN A(X)=(A(X)+1):PSET (X,200-A(X))
60 NEXT
70 LOCATE 40,1,0:PRINT "END:"TIME$
80 END
```

```
10000          START:11:58:01     END:12:17:24
```



NOTE:
After execution of the PSET statement, the last reference pointer (LRP) is updated to the values specified for (X, Y).

PUT

Format	PUT[#]<file number>[,<record number>]
Purpose	This statement is used to write a data record from a random file buffer to a random access disk file.
Remarks	Before this statement can be executed, the disk file must have been opened in the R mode with the OPEN statement and the data to be written to the file must have been set in the random file buffer with the LSET and/or RSET statements. <file number> is the number under which the file was OPENed. If <record number> is omitted, the number used will be the one following that used by the previous PUT statement; this must be a value in the range from 1 to 32767.
See also	LSET/RSET, OPEN
Example	See Chapter 5.

PUT@

Format	PUT[@](X,Y),<variable array name>[,<function>]
Purpose	This statement is used to display a graphic pattern on the display screen.
Remarks	Graphic pattern data to be displayed by the PUT@ statement must have been previously stored in the variable array specified with the GET@ statement. (X,Y) are graphic coordinates specifying the location of the dot at which the upper left corner of the pattern is to be displayed. <variable array name> is the name of the variable array containing the graphic pattern data, and must be the same as that specified in the GET@ statement. <function> specifies the function to be used for display of the graphic pattern; the following functions may be specified.
PSET	Causes the graphic pattern stored in the array to be displayed as is.
PRESET	Displays the color whose code is the complement of the pattern color stored in the array (7-<color code>). In the black and white mode, reverses the setting of each dot in the graphic pattern.
OR	ORs each bit of the graphic pattern in the array and one on the screen, and displays the result.
AND	ANDs each bit of the graphic pattern in the array and one on the screen, and displays the result.
XOR	XORs each bit of the graphic pattern in the array and one on the screen, and displays the result. XOR is assumed for <function> unless otherwise specified.
See also	GET@

NOTE:
After execution of the PUT@ statement, the last reference pointer (LRP) is updated to the values specified for (X, Y).

RANDOMIZE

Format RANDOMIZE [<expression>]

Purpose This statement uses the seed number specified in <expression> to reinitialize the sequence of random numbers returned by the RND function.

Remarks The value specified in <expression> must be a number in the range from -32768 to 32767. If <expression> is omitted, MFBASIC suspends program execution and displays the following message to prompt the operator to enter a value from the keyboard before reinitializing the random number sequence.

```
Random number seed (-32768 to 32767)?
```

If the random number sequence is not reinitialized, the RND function will return the same sequence of random numbers each time the program is run. To change the sequence in which random numbers are generated, place a RANDOMIZE statement at the beginning of the program and change the argument each time the program is executed.

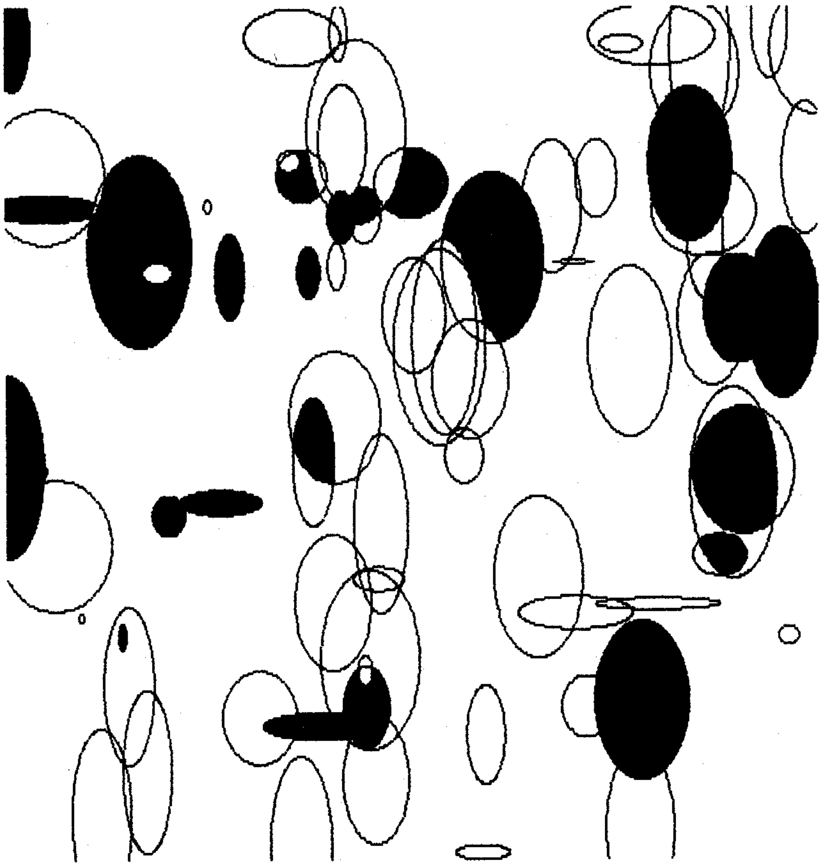
See also RND

Example

```
10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RND;
40 NEXT I
50 END
```

```
RUN
Random number seed (-32768 to 32767)? 1
 .58041  .128928  .928324  .901162  .532818
Ok
RUN
Random number seed (-32768 to 32767)? 2
 .0438479  .891465  .801263  .656113  .16401
Ok
RUN
Random number seed (-32768 to 32767)? 1
 .58041  .128928  .928324  .901162  .532818
Ok
```

```
10 RANDOMIZE VAL(RIGHT$(TIME$,2))
20 CLS
30 FOR Q=1 TO 1000
40 FOR I=1 TO 3
50 X=RND*639:Y=RND*399
60 CIRCLE (X,Y),RND*50,,,,RND*2
70 NEXT I
80 PAINT (X,Y)
90 NEXT Q
```



READ

Format READ <list of variables>

Purpose This statement reads values from DATA statements and substitutes those values into the variables specified in <list of variables>.

Remarks Values which are substituted into variables by the READ statement must be specified elsewhere in the program in a DATA statement. These values are substituted into the variables specified in <list of variables> on a one-to-one basis; i.e., an internal pointer keeps track of the location of the next value to be read in the DATA statement's <list of constants>, and this pointer is advanced for each value read into one of the variables in the READ statement's <list of variables>. Note that the type of each value read must be the same as that of its corresponding variable in <list of variables>; otherwise, a "Syntax error" will occur.

A single READ statement may access one or more DATA statements, or several READ statements may access the same DATA statement.

If the number of variables specified in <list of variables> is greater than the number of constants specified in the DATA statement(s), an "Out of data" error will occur. If the number of variables specified in <list of variables> is smaller than the number of constants specified in the DATA statement(s), subsequent READ statements will begin reading data at the first unread item. If there are no subsequent READ statements, the extra items are ignored.

The pointer used to keep track of the items read from DATA statements can be reset by means of the RESTORE statement.

See also DATA, RESTORE

Example

```
10 FOR I=1 TO 5
20 READ A(I),B$(I),C(I)
30 NEXT I
40 FOR I=1 TO 5
50 PRINT A(I),B$(I),C(I)
60 NEXT I
70 END
80 DATA 1,aaaa,11
90 DATA 2,bbbb,22
100 DATA 3,cccc,33
110 DATA 4,dddd,44
120 DATA 5,eeee,55
```

```
Ok
RUN
 1         aaaa        11
 2         bbbb        22
 3         cccc        33
 4         dddd        44
 5         eeee        55
Ok
```

REM

Format

```
REM <remark>
' <remark>
```

Purpose

The REM statement is used to insert explanatory remarks into a program.

Remarks

Either of the above formats may be used to insert explanatory remarks into a program. REM statements are ignored by MFBASIC during program execution, but are output exactly as entered when the program is listed.

If program execution is branched to a line which begins with a REM statement, execution resumes with the first subsequent line which begins with an executable statement.

If a remark statement is to be appended to a line which includes an executable statement, be sure to precede it with a colon (:). Note that any executable statements following a REM statement on a given program line will be ignored.

Example

```
10 '*** INTEREST COMPUTATION PROGRAM ***
20 'This program uses the Newton-Raphson method
21 'to compute the effective annual rate
22 'of interest on a loan when given the
23 'amount borrowed, the number of payments,
24 'the amount of each payment, and the
25 'number of payments per year.
30 '
40 'The Newton-Raphson method states
41 'that, if c is an approximate value of
42 'a root of an equation, then a better
43 'approximation is the number c-[f(c)/f'(c)].
50 '
60 CLS
70 PRINT"ENTER AMOUNT OF LOAN":INPUT;PV:PRINT
80 PRINT "ENTER NO. OF PAYMENTS":INPUT;NO:PRINT
90 PRINT "ENTER AMOUNT OF EACH PAYMENT":INPUT;AMT:
PRINT
100 PRINT "ENTER NO. OF PAYMENTS/YEAR":INPUT;YR:PR
INT
110 I=1:'Initial estimate of c
120 'Lines 130 to 160 calculate f(c)
130 A=PV*I^NO
140 FOR AA=NO-1 TO 0 STEP -1:LOCATE 37,10,0:PRINT"
RUNNING":LOCATE 37,10,0:PRINT"       "
150 A=A-AMT*I^AA
160 NEXT
170 'Lines 180 to 210 calculate f'(c)
180 B=NO*PV*I^(NO-1)
190 FOR AA=NO-2 TO 0 STEP -1:LOCATE 37,10,0:PRINT"
RUNNING":LOCATE 37,10,0:PRINT"       "
200 B=B-(AA+1)*AMT*I^AA
210 NEXT
220 I=I-A/B:'Calculation of c-[f(c)/f'(c)]
230 IF ABS(A/B)>.000001 THEN 130
231 'Line 230 checks to see whether the
232 'difference between f(c) and f'(c) is
233 'less than .000001; if not, calculations
234 'are repeated using the new estimate
235 'obtained for c on line 220.
240 I$=STR$((I^YR-1)*100)
250 PRINT USING "\    \";I$;:PRINT"% ANNUALLY"
```

```
ENTER AMOUNT OF LOAN
? 10000
ENTER NO. OF PAYMENTS
? 30
ENTER AMOUNT OF EACH PAYMENT
? 400
ENTER NO. OF PAYMENTS/YEAR
? 12


 15.65% ANNUALLY
Ok
```

RENUM

Format RENUM [[<new line number>][,[<old line number>][,<increment>]]]

Purpose This command is used to renumber the lines of programs.

Remarks This command renumbers the lines of a program according to the values specified in its arguments. <new line number> is the first line number to be used in the new sequence of program lines, and <old line number> is the line number in the current program with which renumbering is to begin. <increment> is the value by which each successive line number in the new sequence is to be increased over the number of the preceding line.

The default value for <new line number> is 10, that for <old line number> is the first line of the current program, and that for <increment> is 10.

The RENUM command also changes all line number references included in GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB, and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, the message "Undefined line xxxxx in yyyyy" is displayed. The incorrect line number indicated by xxxxx is not changed, but that indicated by yyyyy may be changed.

NOTE:
The RENUM command cannot be used to change the order of program lines (for example, RENUM 15,30 when the program has three lines numbered 10, 20, and 30) or to create line numbers greater than 65529.

An "Illegal function call" error will result if this rule is not observed.

Examples RENUM
Renumbers the entire program. The first line number of the new sequence will be 10, and the numbers of subsequent lines will be increased in increments of 10.

RENUM 300,50
Renumbers program lines starting with line 50. The number of line number 50 in the current program will be changed to 300, and all subsequent lines will be increased in increments of 10.

RENUM 1000,900,20
Renumbers the lines beginning with 900 so that they start with line number 1000 and are increased in increments of 20.

RESET

Format RESET

Purpose This command is used to reset the READ ONLY condition after the flexible disk in one of the drives has been replaced.

Remarks When the flexible disk in one of the disk drives is replaced with a different one after that drive has been accessed, writes to that drive will be inhibited if the disk is replaced with another one. This is done automatically by the QX-10's basic disk operating system to protect directory information on the disk. Executing the RESET command resets the read-only state and makes it possible to write data to the new disk.

Example

```
100 CLOSE
110 PRINT "REPLACE DISK IN DRIVE A AND PRESS ENTER
 WHEN READY"
120 A$=INPUT$(1)
130 RESET
140 OPEN"O",#1,"A:FILE1"
```

RESTORE

Format RESTORE [<line number>]

Purpose This statement is used to reset the pointer which keeps track of the last item read from DATA statements, making it possible to reread DATA statements from a specific line.

Remarks If <line number> is specified in the RESTORE statement, the next READ statement will access the first item in the DATA statement on the specified line. If <line number> is not specified when RESTORE is executed, the next READ statement will access the first item in the first DATA statement in the program.

See also DATA

Example

```
10 FOR I=1 TO 5
20 READ A$(I),B$(I),C$(I)
30 NEXT I
40 FOR I=1 TO 5
50 PRINT A$(I),B$(I),C$(I)
60 NEXT I
65 PRINT
70 RESTORE 160
80 FOR I=1 TO 3
90 READ D$(I),E$(I),F$(I)
100 NEXT I
110 FOR I=1 TO 3
120 PRINT D$(I),E$(I),F$(I)
130 NEXT I
140 END
150 DATA AAAA,BBBB,CCCC,DDDD,EEEE,FFFF,GGGG,HHHH,I
III,JJJJ,KKKK,LLLL
160 DATA MMMM,NNNN,OOOO,PPPP,QQQQ,RRRR,SSSS,TTTT,U
UUU

Ok
RUN
AAAA          BBBB          CCCC
DDDD          EEEE          FFFF
GGGG          HHHH          IIII
JJJJ          KKKK          LLLL
MMMM          NNNN          OOOO

MMMM          NNNN          OOOO
PPPP          QQQQ          RRRR
SSSS          TTTT          UUUU
Ok
```

RESUME

Format

RESUME
RESUME 0
RESUME NEXT
RESUME <line number>

Purpose

The RESUME statement is used to continue program execution after execution has branched to an error processing routine.

Remarks

The RESUME statement makes it possible to resume program execution at a specific line or statement after error recovery processing has been completed. The point at which execution is resumed is determined by the format in which the statement is executed as follows.

RESUME or RESUME 0	Resumes program execution at or the statement which caused the error.
RESUME NEXT	Resumes program execution at the statement immediately following that which caused the error.
RESUME <line number>	Resumes program execution at the program line specified in <line number>.

A "RESUME without error" error will occur if a RESUME statement is encountered anywhere in a program except in an error processing routine.

See also

ERROR, ON ERROR GOTO, ERR/ERL

Example

```
10 ON ERROR GOTO 70
20 CLS
30 INPUT "Input a number from 1 to 9";A
40 IF A<1 OR A>9 THEN ERROR 200
50 PRINT A
60 END
70 IF ERR=200 THEN PRINT "Error":RESUME 30
```

```
Input a number from 1 to 9? 0
Error
Input a number from 1 to 9? 10
Error
Input a number from 1 to 9? 5
 5
Ok
```

RUN

Format

```
RUN [<line number>]
RUN <file descriptor>[,R]
```

Purpose

The RUN command is used to start execution of a program.

Remarks

The first format is used to start execution of the program currently in memory. Execution begins at the first line of the program unless <line number> is specified. If <line number> is specified, program execution begins at that line.

The second format is used to load and execute a program from a disk drive (including disk image RAM or the CMOS RAM file). Specify the name under which the file was saved in <file descriptor>; if the extension is omitted, ".BAS" is assumed. (For the CMOS RAM file, specify "CMOS:" as the file descriptor.)

The RUN command normally closes all files which are open and deletes the current contents of memory before loading the specified program. However, all data files will remain open if the R option is specified.

See also

LOAD, MERGE

Examples

```
RUN 300
RUN "ADDRESS.BAS"
RUN "B:SAMPLE",R
```

SAVE

Format

SAVE <file descriptor>[,A|,P]

Purpose

The SAVE command is used to save programs on disk files or to the CMOS RAM file.

Remarks

This command saves BASIC programs to a disk file or the CMOS RAM file. In the former case, specify the drive name, primary file name, and extension in <file descriptor>. The currently active drive is assumed if the drive name is omitted, and ".BAS" is assumed if the extension is omitted. In the latter case, specify "CMOS:" as the file descriptor.

If the A option is specified, the program will be saved in ASCII format; otherwise, it will be saved in compressed binary format. The ASCII format requires more disk space for storage than binary format, but some file access operations require that the file be in ASCII format (for example, the file must be in ASCII format if it is to be loaded using the MERGE command).

If the P (protect) option is specified, the program will be saved in an encoded binary format. When a file is saved using this option, it cannot be edited or listed when it is subsequently loaded. Once a program has been saved with the P option, the protected condition cannot be cancelled.

See also

LOAD, MERGE

Examples

```
SAVE"ADDRESS"
SAVE"B:ADDRESS.ASC",A
SAVE"CMOS:"
SAVE"SECRET",P
```

SET

Format SET <file descriptor>[,P]

Purpose This statement is used to set or reset the write protect attribute of disk files.

Remarks This statement sets or resets the write protect attribute of the disk file specified by <file descriptor>. (The file specified in <file descriptor> must be a disk file.) The write protect attribute is set if the P option is specified, and is reset if the P option is omitted.

When the write protect attribute is set for a file, any attempt to write data to that file will result in a ''Disk write protect'' error.

Examples

```
SET "B:CLIENTS.DAT",P
SET "B:CLIENTS.DAT"
```

SOUND

Format

SOUND <pitch>,<duration>

Purpose

The SOUND statement outputs a tone of the specified pitch and duration from the speaker. The frequency of the tone generated is specified in <pitch> as a value from 31 to 32767; no sound is generated by specifying values from 0 to 30. The duration of the tone is specified in <duration> as a value from 0 to 5461; when the statement is then executed, the duration of the tone will be equal to <duration> × 10 msec.

MFBASIC continues execution of programs statements even while a sound is being generated by the SOUND statement; however, subsequent SOUND statements encountered will not be executed until the tone resulting from the previous one has been output for the specified amount of time.

See also

BEEP

Example

```
10 FOR X=1 TO 8
20 READ A:SOUND A,50
30 NEXT
40 FOR X=1 TO 8
50 READ A:SOUND A,50
60 NEXT
70 DATA 256,288,320,341,384,426,480,512
80 DATA 512,480,426,384,341,320,288,256
```

STOP

Format STOP

Purpose The STOP statement terminates program execution and returns MFBASIC to the command level.

Remarks STOP statements are generally used to interrupt program execution during debugging to allow intermediate values to be examined or changed in the direct mode; program execution can then be resumed by executing a CONT command.
The following message is displayed when a STOP statement is encountered.

```
BREAK IN line nnnnn
```

Unlike the END statement, no files are closed when a STOP statement is executed.

See also CONT

Example

```
10 OPEN"O",#1,"A:FILE1"
20 FOR I=1 TO 30
30 PRINT #1,STR$(I)
40 NEXT
50 CLOSE
60 OPEN"I",#1,"A:FILE1"
70 IF EOF(1) THEN 130
80 INPUT#1,A$
90 PRINT A$
100 STOP
110 GOTO 70
120 STOP
130 CLOSE
```

```
RUN
1
Break in 100
Ok
CONT
2
Break in 100
Ok
CONT
3
Break in 100
Ok
CONT
4
Break in 100
Ok
.
.
.
```

STOP KEY

Format

```
STOP KEY |ON |
         |OFF|
```

Purpose

The STOP KEY statement is used to disable or reenable the BREAK key.

Remarks

Executing STOP KEY OFF inhibits the function of the BREAK key (or CTRL and C). This prevents processing from being interrupted by accidentally pressing the BREAK key during program execution. The BREAK key function is reenabled by executing STOP KEY ON.

If no STOP KEY statement is executed, STOP KEY ON is assumed.

Since STOP KEY OFF completely disables operation of the BREAK key (or CTRL and C), it should be used with caution.

SWAP

Format SWAP <variable 1>,<variable 2>

Purpose The SWAP statement exchanges the values of the variables specified in <variable 1> and <variable 2>.

Remarks The SWAP statement may be used to exchange the values of any type of variable, but the same variable types must be specified in both <variable 1> and <variable 2>; otherwise, a "Type mismatch" error will occur.

Further, an "Illegal function call" error will occur if the name specified in <variable 2> is that of a non-array variable to which no value has been previously assigned.

Example

```
10 FOR I=1 TO 5
20 INPUT "ENTER NAME";A$(I)
30 NEXT I
40 FOR I=2 TO 5
50 IF A$(I-1)>A$(I) THEN SWAP A$(I-1),A$(I):I=1
60 NEXT I
70 FOR I=1 TO 5
80 PRINT A$(I)
90 NEXT
```

```
Ok
RUN
ENTER NAME? JACKSON
ENTER NAME? SMITH
ENTER NAME? CLARK
ENTER NAME? JONES
ENTER NAME? FONDA
CLARK
FONDA
JACKSON
JONES
SMITH
Ok
```

SYSTEM

Format SYSTEM

Purpose The SYSTEM command clears program memory and returns control from MFBASIC to CP/M.

Remarks This command may be executed either in the MFBASIC direct mode or the indirect mode.

Example SYSTEM

TIME$

Format

As a statement
TIME$="<HH>:<MM>:<SS>"
As a variable
X$=TIME$

Purpose

TIME$ is a system variable which contains the time of the QX-10's built-in clock.

Remarks

As a statement, TIME$ is used to set the date of the QX-10's built-in clock. <HH> is a 2-digit number from 00 to 23 which indicates the hour, <MM> is a 2-digit number from 00 to 59 which indicates the minute, and <SS> is a 2-digit number from 00 to 59 which indicates the second.

As a variable, TIME$ returns the time of the built-in clock in "HH:MM:SS" format.

See also

TIME

Example

```
10 PRINT"CURRENT TIME IS ";TIME$
20 INPUT "ENTER NEW TIME (HH:MM:SS)";A$
30 TIME$=A$
40 PRINT TIME$
```

```
Ok
RUN
CURRENT TIME IS 08:03:38
ENTER NEW TIME (HH:MM:SS)? 16:44:40
16:44:40
Ok
```

TRON/TROFF

Format TRON
TROFF

Purpose These commands are used to trace program execution.

Remarks Executing TRON (either in the direct mode or indirect mode) causes the number of each program line to be displayed on the screen in brackets as it is executed. Tracing is discontinued by executing TROFF or a NEW command.
These commands can be used in conjunction with the STOP and CONT commands during program debugging.

See also CONT, STOP

Example

```
10 TRON
20 GOSUB 70
30 TROFF
40 PRINT
50 GOSUB 70
60 END
70 PRINT"EPSON ";
80 PRINT"QX-10 ";
90 PRINT"MFBASIC"
100 RETURN
```

```
Ok
RUN
[20][70]EPSON [80]QX-10 [90]MFBASIC
[100][30]
EPSON QX-10 MFBASIC
Ok
```

WAIT

Format WAIT <port number>,I[,J]

Purpose This statement suspends program execution while monitoring the status of a machine input port.

Remarks The WAIT statement causes execution to be suspended until a specified machine input port develops a specified bit pattern. The data read at the port is exclusive ORed with the value of integer expression J, then ANDed with I. If the result is zero, MFBASIC loops back and reads the port again. If the result is not 0, execution continues with the next statement. If J is omitted, 0 is assumed.

NOTE:
Use of this statement requires in depth knowledge of the QX-10 firmware, and using it incautiously can result in loss of system control and other problems. See the QX-10 Programmer's Guide for detailed information on the QX-10 firmware.

WHILE...WEND

Format

```
WHILE <expression>
   .
   .
[<loop statements>]
   .
   .
WEND
```

Purpose

The WHILE...WEND statement is used to repeat the series of instructions included between WHILE and WEND as long as a specified condition is satisfied.

Remarks

This statement is used in the same manner as the FOR...NEXT statement, except that the loop is repeated until the condition specified in <expression> is no longer satisfied.
As with FOR/NEXT loops, WHILE/WEND loops may be nested to any level (they may also be included within FOR/NEXT loops, or vice versa); when loops are nested, the first WEND corresponds to WHILE of the innermost loop, the second WEND corresponds to WHILE of the next innermost loop, and so forth.

A "WHILE without WEND" error will occur if WHILE without a corresponding WEND is encountered, and a "WEND without WHILE" error will occur if WEND without a corresponding WHILE is encountered.

See also

FOR...NEXT

Example

```
10 INPUT "ENTER ARBITRARY NUMBER";X
20 WHILE X^A<1E+06
30 PRINT STR$(X);"^";MID$(STR$(A),2,5);"=";MID$(ST
R$(X^A),2,6)
40 A=A+1
50 WEND
```

```
Ok
RUN
ENTER ARBITRARY NUMBER? 5
 5^0=1
 5^1=5
 5^2=25
 5^3=125
 5^4=625
 5^5=3125
 5^6=15625
 5^7=78125
 5^8=390625
Ok
```

WIDTH

Format

WIDTH <no. of columns>
WIDTH <file descriptor>, <no. of columns>
WIDTH# <file no.>, <no. of columns>
WIDTH LPRINT <no. of columns>

Purpose

This statement is used to set the column width of the specified device or file.

Remarks

The functions of the four specification formats are as follows.

WIDTH <no. of columns> specifies the number of characters which can be displayed on each display line; either 40 or 80 can be specified in <no. of columns>. Execution of the WIDTH command in this format clears the screen and places the display in either the double width (40 column) or normal (80 column) display mode. In the 40 column mode, the 1-byte character set can be changed with the OPTION STYLE statement.

WIDTH <file descriptor>, <no. of columns> specifies the number of characters which can be output in each line to the device specified in <file descriptor>. The devices which can be specified in <file descriptor> are "CMOS:", "LPT0:", and "COMn:". An "Illegal function call" error will result if any other device is specified. When "LPT0:" is specified, this format performs the same function as WIDTH LPRINT. When the WIDTH statement is executed in this format and the specified device is already open, the output width specified does not become effective until the device has been closed and then reopened.

WIDTH# <file no.>, <no. of columns> specifies the output width for the file (channel) specified by <file no.>. With this format, the file specified by <file no.> must be open at the time the WIDTH statement is executed. Further, the file device must be either "SCRN:", "CMOS:", "LPT0:" or "COMn:"; an "Illegal function call" error will result if these conditions are not satisfied.

The width specified by executing the WIDTH statement in this format becomes effective immediately for the specified file (channel); this makes it possible to change the width of applicable open device files at any time.

WIDTH LPRINT <no. of columns> specifies the maximum number of characters which can be printed on each line by the printer.

For devices other than the display screen any value from 1 to 255 can be specified in <no. of columns>. When a value from 1 to 254 is specified, MFBASIC monitors the number of characters output and, when the number exceeds that specified, automatically outputs a carriage return/line feed code. When 255 is specified, the output line width is assumed to be infinite and MFBASIC does not automatically output carriage returns and line feed codes.

Example 1

```
10 WIDTH 40
20 PRINT "ABCDEFGH"
30 END
Ok
```

```
ABCDEFGH
Ok
```

Example 2

```
10 OPEN"O",#1,"LPT0:"
20 WIDTH #1,10
30 PRINT#1,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
40 PRINT#1,"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
50 PRINT#1,"CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
60 CLOSE
AAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAAAA
AAA
BBBBBBBBBB
BBBBBBBBBBB
BBBBBBBBBBB
BBB
CCCCCCCCCC
CCCCCCCCCCC
CCCCCCCCCCC
CCC
```

Example 3

```
10 WIDTH "LPT0:",15
20 LPRINT"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
30 LPRINT"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
40 LPRINT"CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
AAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA
AAAAA
BBBBBBBBBBBBBBB
BBBBBBBBBBBBBBB
BBBBB
CCCCCCCCCCCCCCC
CCCCCCCCCCCCCCC
CCCCC
```

Example 4

```
10 WIDTH LPRINT 20
20 LPRINT"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
30 LPRINT"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
40 LPRINT"CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCC
```

NOTE:
The initial line width values are as follows.

LPT0:	*80*
COM0: -COM4:	*255*
CMOS:	*255*

WRITE

Format WRITE[<list of expressions>]

Purpose This statement is used to display data on the CRT screen.

Remarks If <list of expressions> is omitted, a blank line is output to the display screen. If <list of expressions> is included, the values of the expressions are displayed on the screen.

Numeric and string expressions can both be included in <list of expressions>, but each expression must be separated from the one following it with a comma. Commas are displayed between each item included, and strings displayed will be enclosed in quotation marks. After the last item has been output, the cursor is automatically advanced to the next line.

The WRITE statement displays numeric values using the same format as the PRINT statement; however, no spaces are output to the left or right of numbers displayed.

See also PRINT

Example

```
10 A=10:B=90:C$="EPSON QX-10"
20 WRITE A,B,C$
30 END

Ok
RUN
10,90,"EPSON QX-10"
Ok
```

WRITE

Format WRITE # <file number>,<list of expressions>

Purpose This statement is used to write data to a sequential disk file.

Remarks <file number> is the number under which the file was OPENed for output, and the expressions included in <list of expressions> are the numeric and/or string expressions which are to be written to the file. Data is written to the file in the same format as for display on the screen by the WRITE statement; i.e., commas are inserted between items and strings are delimited with quotation marks. Therefore, it is not necessary to specify explicit delimiters in <list of expressions>, as is the case with the PRINT # statement). A carriage return/line feed sequence is written to the disk following the last item in <list of expressions.>

See also PRINT # and PRINT # USING

Example

```
100 A$="EPSON":B$="QX-10"
110 OPEN"O",#1,"DATA"
120 WRITE#1,A$,B$
```

The above sequence writes the following image to the disk:

```
  "EPSON","QX-10"
```

A subsequent INPUT # statement (such as INPUT #1,A$,B$) would input each item to separate variables ("EPSON" to A$ and "QX-10" to B$). Compare this with the example given in the description of the PRINT # statement.