6.0

RESIDENT SYSTEM SOFTWARE

6.1

Introduction

The resident software is comprised of major modules, Basic Interpreter, the Machine Language Utility, and the General Housekeeping Module. Under normal system operation they work together to allow use of BASIC programs from cassette. For machine code programs major functions available as subroutines.

6.2

Resident DAI BASIC

6.2.1

Alphabetic Index of DAI BASIC Statements

6.2.1.1

BASIC Commands

6.2.1.2

BASIC Functions

ABS	6.2.14.1	LOG	6.2.14.15
ACOS	6.2.14.2	LOGT	6.2.14.16
ALOG	6.2.14.3	MID$	6.2.14.17
ASC	6.2.14.4	PDL	6.2.7.4
ASIN	6.2.14.5	PEEK	6.2.7.5
ATN	6.2.14.6	PI	6.2.14.18
CHR$	6.2.14.7	RIGHT$	6.2.14.19
COS	6.2.14.8	RND	6.2.14.20
CURX	6.2.12.10	SCRN	6.2.12.8
CURY	6.2.12.10	SGN	6.2.14.21
EXP	6.2.14.9	SIN	6.2.14.22
FRAC	6.2.14.10	SPC	6.2.14.23
FRE	6.2.11.3	SQR	6.2.14.24
FREQ	6.2.13.6	STR$	6.2.14.25
GETC	6.2.8.2	TAB	6.2.14.26
HEX$	6.2.14.11	TAN	6.2.14.27
INP	6.2.7.12	VAL	6.2.14.28
INT	6.2.14.12	VARPTR	6.2.11.5
LEFT$	6.2.14.13	XMAX	6.2.12.6
LEN	6.2.14.14	YMAX	6.2.12.7

6.2.1.3

Arithmetic and Logical Operators

+, -, *, /, MOD, ↑, =, <, >, <>, <=, >=, IOR, IAND, IXOR, INOT, SHL, SHR, AND, OR.

6.2.2

Format rules and constraints

6.2.2.1

Variables and Numbers

DAI BASIC recognises 2 types of numeric value, integer, and floating point. Integers are whole numbers only, and of restricted range. $\pm 2 \uparrow 32 - 1$ (e.g. about 9 digits). However, integer arithmetic is exact and gives no rounding errors. Floating point numbers include non-integer values, and allow numbers whose size is in range 10^{-18} to 10^{18}, with 6 digit printout resolution. (32 bit floating point format).

Various DAI BASIC commands expect either an integer or a floating point value. For example:

a) DRAW A,B C,D X. All of parameters A,B,C,D and X are expected to be integers.

b) LET A = SQRT (B). The parameter B is expected to be a positive floating point number.

DAI BASIC obeys the following rules regarding numerical values:

1) When a floating point value is found where an integer value is required, it is truncated (e.g. 2.3 → 2, -1.7 → -1).

2) When an integer value is found where a floating point value is required, it is converted automatically.

3) Where an integer representation (e.g. "3" not "3.0") is typed in, it will be encoded as a floating point or integer value as the context demands, or if neither is defined, e.g. in "PRINT", as the type set by the "IMP" command.

Variable names have from 1 to 14 characters, of which the first must be alphabetic, and the rest either alphabetic or numeric. Alphanumeric characters after the 14th are ignored. If no type letter ($, %, !) is appended then the type depends on the IMP command. Initially all such variables are floating point.

Numeric variables in DAI BASIC may be either floating point or integer type. Integer variable names are terminated by the character "%", and floating point by " ! ". String variables have "$" as a terminator. But see examples for influence of IMP command.

Examples:
Initially

I,A,S are floating point, because they are abbreviations of I!, A!, S!

I%,A%,S% are integer and distinct from I, A, S.

I!,A!,S! are floating point, and are the same variables as I, A,S.

I$,A$,S$ are string variables.

So if the IMP command is never used, floating point variables can be indicated by leaving off the "type" letter, integer variables by using %, and string by using $.

After IMP INT I-N
IMP STR S-S

I is an abbreviation for I%, or integer variable

A is an abbreviation for A! or floating point variable

S is an abbreviation for S$ or string variable

However any variable with a type letter (I$, A%, S!) is totally unaffected by the IMP command. When the Personal Computer is LISTING a program, it uses the shortest form for a name. In other words after the example above, the variable I% would be printed as just I, S$ as just S, and A! as just A. If the IMP command is used in the form "IMP INT" or "IMP FPT", without a range of letters, then all variable names are defaulted to that type. In addition integer number representations e.g. "3", are interpreted as the required type.

Command	Means same as	"3" is interpreted as	and A as
IMP INT	IMP INT A - Z	Integer 3	A%
IMP FPT	IMP FPT A - Z	Floating point 3.0	A!
IMP STR	Not allowed	-	-

At power on the system does an initial "IMP FPT".

6.2.2.2
Strings

1) A string may be from 0 to 255 characters in length.
2) String arrays may be dimensioned exactly like numeric arrays. For instance, DIM A$(10,10) creates a string array of 121 elements, eleven rows by eleven columns (rows 0 to 10 and columns 0 to 10). Each string array element is a complete string, which can be up to 255 characters in length.
3) The total number of characters in use in strings and associated control bytes at any time during program execution cannot exceed the amount of string space requested, or an error message will result.
4) Strings cannot contain the character double quote (Hex 22). It can be printed using CHR$(#22).

Examples of String Usage (Do not forget to make first a CLEAR).

DIM A$(10,10)

Allocates space for a pointer in string space for each element of a string matrix. No furhter string space is used at this time.

A$ = "F00" + V$

Assigns the value of a string expression to a string variable, requiring string space equal to the number of characters plus one.

IF A$ = B$ THEN STOP

String comparison operators. Comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If during the comparison of two strings, the end of one string is r reached, the shorter string is considered smaller. Note that "A " is greater than "A" since trailing spaces are significant.

INPUT X$

Reads a string from the keyboard. String does not have to be in quotes, but if not leading blanks will be ignored and the string will be terminated on a "," character.

READ X$

Reads a string from DATA statements within the program. Strings do not have to be in quotes, but if they are not they are terminated on a "," character or end of line, and leading spaces are ignored.

PRINT X$

PRINT "F00"+A$

Prints the result of the string expression.

6.2.2.3
Operators

It is obvious that the result of adding I% + J% when I% contains 3 and J% contains 4 should be the integer 7. It is also reasonable to expect I + J where I contains 3.0 and J contains 4.0 to give the floating point result 7.0. Thus some BASIC operators do different things depending on the types of their operands. It is always permitted to give operands of either type to any operator. However the operator may convert either or both operands to another type before use.
Relational operators and the operators "AND" and "OR" produce results of type "logical". These results cannot be assigned to any variables and are only used in "IF" statements.

6.2.2.4
Statements

In the description of statements, an argument of V or W denotes a numeric variable, X denotes a numeric expression and an I, J or K denotes an expression that is truncated to an integer before the statement

is executed. A, B indicate array names without any parameters.
An expression is a series of variables, operators, function calls and constants which after the operations and function calls are performed using the precedence rules, evaluates to a numeric or string value.

A constant is either a number (3.14) or a string literal ("F00").

6.2.2.5
Expressions

The cardinal principle behind the evaluation of expressions by DAI BASIC is that if an expression contains only integer values or variables and operators which work on integers, then at no time is floating point arithmetic used. This gives fast integer arithmetic where it is needed for industrial control and graphics applications.

Order of Evaluation

Expressions in Brackets
↑
* / MOD
+ -
SHL SHR
IOR IAND IXOR
> < = < > <= >=
AND OR

Operators on the same level are evaluated from left to right.
E.g. 3 * 5 MOD 2 = 1

6.2.3

Error Reporting

6.2.3.1

Error Report Format

When an error is encountered a message is printed giving details. Under certain circumstances, other information will be given.

(i) If an immediate command has just been input, than no other information is given.

(ii) If a stored program line has just been input, then a reflection of the line with a "?" near the error will be printed.

(iii) If an immediate command is being run, no other information is given.

(iv) If a stored program line is being run, the words "IN LINE NUMBER" and the line number are given.

In case (ii), the line goes into the program with a "***" on the front.
(Internally coded as an ERROR LINE)

6.2.3.2

Error Messages Dictionary

CAN'T CONT
There is no suspended program to be "CONTinued".

COLOUR NOT AVAILABLE
A colour has been used in 4 colour mode when it has not been set up by a COLORG command.

COMMAND INVALID
This command cannot be used in a non-stored program line, or in a stored program line, whichever was attempted.

DIVISION BY 0
Integer or floating point divide by 0.

ERROR LINE RUN
A line which gave an error message when it was input has been run without first correcting it.

INVALID NUMBER
The parameter given to a VAL function was not a valid floating point number.

LINE NUMBER OUT OF RANGE
A line number greater than 65535 or zero has been used. (or negative)

LINE TOO COMPLEX
Line typed in would generate more than 128 bytes of encoded program.

LOADING ERROR 0 , 1 , 2 or 3
The program or data requested could not be loaded.
For cassette:
0 means Checksum error on program name.
1 means Insufficient memory
2 means Checksum error on program.
3 means Data dropout error.

NEXT WITHOUT FOR
A "NEXT" statement has been executed without a corresponding "FOR" statement.

NUMBER OUT OF RANGE
Some number has been used in context where it is too large or small.

OFF SCREEN
A point has been referred to which does not exist in this mode.

OUT OF DATA
A "READ" statement has tried to use more DATA than exists.

OUT OF MEMORY
Some attempt has been made to use too much space for the program, symbol table, screen, heap (strings + arrays storage) or edit buffer.

OUT OF SPACE FOR MODE
This message occurs if a program is running in modes 1 or 2, with insufficient free space to run mode 0, 1A or 2A, and attempts to print a message. The system deletes the program by a NEW and prints this message.

OUT OF STRING SPACE
More string space has been used than was allowed for.

OVERFLOW
Integer or floating point overflow.

RETURN WITHOUT GOSUB
A "RETURN" statement has been executed with no corresponding "GOSUB"

STACK OVERFLOW
A line too complex has been typed in, or, too much stack space has been used by a running program.

STRING TOO LONG
A string of over 255 characters has been created.

SUBSCRIPT ERROR
A subscript has been evaluated which is outside the declared range for the array, an array name has been used with the wrong number of parameters, or a dimension of 0 has been requested.

SYNTAX ERROR
Some error in the line just typed in, or the line of data read by an INPUT or READ.

TYPE MISMATCH
Some expression gives a result of an incorrect type for its position. Can occur on input or while a program is running.

UNDEFINED ARRAY
A reference has been made to an array which has not yet been "DIMensioned".

UNDEFINED LINE NUMBER
A reference has been made to a non-existent program line.

6.2.4

Interacting with DAI BASIC

6.2.4.1

Facilities of the Character Screen

When the Personal Computer first prints the message "BASIC" and the prompt, the screen is in what is known as mode 0. That is 24 lines of 60 characters. At any time the screen can be returned to this mode with the command "MODE 0".
The next position where a character will be displayed is indicated by a flashing underline cursor.
Lines on the screen are obviously physically 60 characters long. But when characters are being output the line can be extended with up to 3 "continuation" lines. These have the letter C in column 0 and the first character of those coninuation lines are indented 7 spaces to the right.
The cursor is moved forward when a character is output, and backwards for a backspace (# 8) character. Carriage return (# D) ends a line.
The form feed character (# C) has the special effect of entirely clearing the character area (in any mode) and placing the cursor at the top left position.
The tab (# 9) character has no special function.

When the third continuation line is used up, further characters output to the screen are ignored, until a carriage return, backspace or form feed.
When BASIC is expecting input it only notices characters in positions after the prompt character. If the prompt is deleted with backspaces, then any character put in that position will be ignored, probably causing a syntax error. The colours used for characters are initially set at power on. They can be changed using the COLORT Command.

6.2.4.2

Input of programs and data

When the Personal Computer expects input, it always types a "prompt" character, normally a "* ", but during INPUT commands a "?".
The user can then type in characters at will. To delete the last entered character, the "CHAR DEL" key is used. If more information is input than fits across the screen, then it is continued on the following line, indented and with a "C" (for continuation) in column 0. Up to 3 continuation lines may be used, giving a line length of 59 + 53 + 53 + 53 = 218 characters.

Pressing BREAK while typing in commands causes a " " to be printed, and the line is ignored. However during input for an INPUT command, it causes suspension of the program.

6.2.4.3

Amending and running of programs.

When the Personal Computer is ready to accept instructions, it prints a prompt character.
The user can then type in a line of one or more commands, separated by the character ":", and terminated by a "RETURN". The commands will be encoded immediately, and if they have the right syntax, will be run.
If the line has a number on the front, it will be encoded as before and placed into the stored program in the machine, according to its line number. It replaces any previous line with that number. If the line is not syntactically correct, an error message will be printed. If there was no line number, no other action is taken. If there was, then a is is inserted as a dummy first command on the line, and the first 121 characters of the line are encoded as if the line were a REM statement. Attempted execution of the line yields the message "ERROR LINE RUN". A question mark is inserted near the point where the error was detected. The line is then inserted into the program as before.
When the user wishes to run a stored program, he types "RUN", to start at the first line or "RUN 22" to start at line 22.

(for example). The program will then run until some error, or one of the following, occurs:

(i) If an END statement is executed, the program stops. It prints the message: END PROGRAM. The program can only be restarted using RUN.

(ii) If a STOP statement is executed, the program stops. It prints the message: STOPPED IN LINE X with X the appropriate line number. The program is then said to be "suspended".

(iii) If the BREAK key is held down, one of two results will occur:
a) In most circumstances the message BREAK IN LINE X will be printed immediately. The program is then suspended.
b) Under some circumstances, after a pause the system will print: ***BREAK. The program cannot now be restarted.

When a program is suspended, it can be restarted by use of the CONT command. This restarts the program just as if it had never stopped. However any variables etc. changed by the user during the suspension are not restored to their old values.
If the system has cause to report any run-time error to the user, or if the user RUNs any other program or does a SAVE, LOAD, EDIT, CLEAR or NEW, then the suspended program is no longer valid and cannot be CONTinued. If the user tries to do so a message will be printed: CAN'T CONT. When a RUN, SAVE, CLEAR, LOAD, EDIT or NEW command is executed, all variables are reset to 0 (if arithmetic) or a null string (if string). All space assigned to arrays is returned, and any subsequent reference to an array before running a DIM statement for it will give an error.
To delete the stored program the command NEW is used. After this there are no stored lines in the machine and no variables are set to any values.

When a program is suspended the STEP command may be used to continue the program one line at a time. Before each line is executed it is listed to the screen and the machine waits for a space to be typed in on the keyboard.

At power on DAI BASIC defaults into the floating point variable mode where integer variable names must be concluded by the (%) character. A facility to allow this to be switched is provided by the IMP statement. The operator must type in any IMP switches that he desires before he enters his program.

6.2.4.4
Merging of BASIC Programs

CLEAR 10000
LOAD SEGMENT 1 OF PROGRAMS TO BE MERGED
EDIT + BREAK + BREAK
LOAD SEGMENT 2 OF PROGRAMS TO BE MERGED
(THE LINE NUMBERS CANNOT BE THE SAME IN SEGMENTS 1 AND 2)
POKE #135,2

6.2.4.5
Merging of BASIC and machine Language Programs (or routine)(MLP/R)

a) Prepare of the MLP/R and save it after the BASIC program you intend to use with this MLP/R.

EXAMPLE SAVE FIRST YOUR BASIC PROGRAM (see example under of program)

```
MLP/R   10 CLEAR 2000
        20 DIM A (20,20)
        30 FOR I% = Ø TO 9
        40 READ B% : POKE (#2F1 + I%), B% : NEXT
        50 SAVEA A "TEST" : STOP
        60 DATA #F5,#3E,#FF,#32,#50,#BE,#F1,#C9,Ø,Ø
```

N. B. The size of a one dimension array is (256 x 4) bytes maximum. In this example the size is (20 x 20 x 4) = 1764 bytes.
The basic program you intend to use must have:

- a CLEAR - a DIM (of the same name and the same array size as the MLP/R - a LOADA (of same name than the MLP/R)
EXAMPLE of BASIC program that you have on cassette before the MLP/R

```
10 CLEAR 2ØØØ
20 DIM A (2Ø,2Ø)
30 LOADA A
40 CALLM #-2F1
50 STOP
```

This program will load the MLP/R after you make a RUN and execute the MLP/R by the CALLM of line 40. You should now RUN 40 each time for calling the MLP/R. You can also delete the first 3 lines by typing 10, RETURN, 30, RETURN.

Important: When the MLP/R has been loaded by the BASIC program do not use the EDIT mode, nor RUN the lines containing the CLEAR, DIM and LOADA commands (in this example you must RUN 40), nor use somewhere in the BASIC program a CLEAR command or a DIM statement with the same array name used for the MLP/R.

When using an MLP/R with a BASIC program (if you have not been locating this MLP/R at any location of your choice) you will find the # location of the begin of the MLP/R by
PRINT HEX$ (VARPTR (A(Ø,Ø))). This location is usually 2FØ for the first MLP/R for a one dimension array and #-2F1 for a 2 dimension array (when the discs are not used, as the DOS moves the Heap).

6.2.5.
User Control Statements

6.2.5.1
EDIT

EXAMPLE(s)

(i) EDIT
Moves entire BASIC program into edit Buffer for possible modification and display

(ii) EDIT 100
Moves only the BASIC program line number 100 into the edit buffer for possible modification and display.

(iii) EDIT 100 -
Moves the BASIC program line numbers 100 until the end of the BASIC program into the edit buffer for possible modification and display.

(iv) EDIT 100-130
Moves the BASIC program line numbers 100 to 130 into the edit buffer for possible modification and display.

(v) EDIT - 130
Moves the BASIC programs from the first line to line number 130 into the edit buffer for possible modification and display.

Functional Explanation
The Edit statement provides a simple means to modify or type-in aprogram into the DAI Personal Computer. A number of program lines are placed into an internal edit buffer. The first 24 BASIC program lines in the edit buffer are displayed on the screen. The cursor is positioned at the first character of the first line on the display.

The cursor can be moved around the screen by use of the cursor control keys. (↑ ↓ → ←). If the operator attempts to move the cursor off the screen

the part of the document which can be seen on the screen is moved to keep the cursor visible. The visible area of the document is known as the "window". The window can also be changed by using the cursor control keys plus the "shift" key. The cursor stays in the same place in the document, unless moving the window would take it off the screen.
The CHAR DEL key deletes the character at the cursor. It has no effect to the right of a carriage return. Any other character typed in is inserted before the cursor position, if the cursor is left of the carriage return on the line.
When all editing is finished, the BREAK key should be pressed. If it is followed by a second BREAK, then the whole effect of the editing is ignored. If followed by a space, then the original version of the edited text is deleted, just as if it were typed in from the keyboard.
Any necessary error messages will be put on the screen, andfollowed by a prompt. The Edit command is also used to achieve Program merges from different cassettes.

Special note:

Avoid pressing BREAK or any other key after typing the end of the EDIT command and before the program has been displayed on the screen.
See "Edit Buffer Program" in appendix.

6.2.5.2

IMP

EXAMPLES

See examples given in paragraph 6.2.2

6.2.5.3

LIST

EXAMPLE(S)

(i) LIST

Displays the entire BASIC program. During display the output can be made to pause by pressing any character key. Then pressing of the space bar will continue the listing display output.

(ii) LIST 100

Displays BASIC program line number 100 only.

(iii) LIST 100 -

Displays BASIC program starting at line number 100 until the end of the program.

(iv) LIST 100-130

Displays BASIC program line numbers 100 to 130.

(v) LIST - 100

Displays BASIC program starting at first line of program and until line number 130.

6.2.5.4

NEW

EXAMPLE(S)

(i) NEW

Deletes current BASIC program that is stored in memory and resets all variables to the undefined state. The HEAP reservation is is not changed. (See 6.2.11).

6.2.5.5

RUN

EXAMPLE(S)

(i) RUN

Starts execution of the BASIC program currently in memory at the lowest line number.

(ii) RUN 100

Starts execution of ten BASIC program currently in memory at line number 100. If line 100 does not exist, an error message occurs.

6.2.6

Program control Statements

6.2.6.1

END

EXAMPLE(S)

(i) END

Terminates the execution of a BASIC program. The program cannot be further continued without a RUN command. An "END PROGRAM" message is displayed.

6.2.6.2

FOR.....NEXT

EXAMPLE(S)

(i) FOR V = 1 TO 9.3 STEP .6

(ii) FOR V = 1 TO 9.3

(iii) FOR V = 10*N TO 3.4/Q STEP SQR(R)

(iv) FOR V = 9 TO 1 STEP - 1

(v) FOR W = 1 TO 10 : FOR W = 0 TO 3 : NEXT : NEXT

The variable in the FOR statement is set to the first expression given. Statements are executed until a NEXT statement is encountered. Action at this point depends on the rest ot the FOR statement. When the FOR statement is executed the "TO" and "STEP" expressions are also calculated. The step defaults to 1 if it is not explicitly given. Then the range is divided by the step to calculate a repeat count for the loop. This must be within the ranges 0 to 2↑23-1 for a floating point loop and 0 to 2↑31-1 for an integer one. The loop is run this number of times irrespective of anything else, and is always run at least once.
If the STEP is not explicitely given then the NEXT statement uses a special fast routine to increment the variable value. If it is explicitely given it is added to the variable. Loops using integer variables run faster than those using floating point ones.

Special cases:

a) The interpreter will terminate an unfinished loop if a NEXT statement for an outer one is encountered. E.g.
FOR A = 1 TO 10 : FOR B = 0 TO 3 : NEXT A
is allowable.

b) The interpreter will terminate all loops up to the correct level if a loop is restarted. E.g.
10 FOR A = 1 TO 10
20 FOR B = 0 TO 3
30 GOTO 10
is allowable.

c) FOR loops inside a subroutine are separate from those outside for purpose of special cases (a) and (b)

d) A FOR loop may be abandoned by a RETURN statement. E.g.
10 GOSUB 10
20 STOP
30 FOR A = 1 TO 10
40 RETURN
is allowable.

e) after a FOR loop finishes, the variable has the value it would next have taken.
E.g. 10 FOR I = 0 TO 10 : NEXT
20 PRINT I
Will print 11.Ø.

6.2.6.3

GOSUB

EXAMPLE

(i) GOSUB 910

Branches to the specified statement, i.e. (910). When a Return statement is encountered the next statement executed is the statement following the GOSUB. GOSUB nesting is limited only by the available stack memory. A program can have 10 levels of GOSUB or 15 levels of FOR loops without difficulty.

6.2.6.4

GOTO

EXAMPLE

GOTO 100

Branches to the statement specified.

6.2.6.5

IF.....GOTO

EXAMPLES

(i) IF X = Y + 23.4 GOTO 92

Equivalent to IF ... THEN, except that IF ... GOTO must be followed by a line number, while IF ... THEN is followed by another statement, or a line number.

(ii) IF X = 5 GOTO 50:Z = A

Warning: Z = A will never be executed.

6.2.6.6

IF ... THEN

EXAMPLE

(i) IF X < 0 THEN PRINT "X LESS THAN 0" : GOTO 350

In this example, if X is less than 0, the PRINT statement will be executed and then the GOTO statement will branch to line 350. If the X was 0 or positive, BASIC will proceed to execute the lines after this one.

(ii) IF X = Y + 23.4 THEN 92

IF ... THEN statement in this form is exactly equivalent to IF ... GOTO example (1).

6.2.6.7

ON ... GOSUB

EXAMPLE(S)

(i) ON I GOSUB 50, 60

Identical to "ON ... GOTO", except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the GOSUB branches to the statement after the ON ... GOSUB.

6.2.6.8

ON ... GOTO

(i) ON I GOTO 10, 20, 30, 40

Branches to the line indicated by the I'th number after the GOTO.

That is:

IF I=1 THEN GOTO LINE 10
IF I=2 THEN GOTO LINE 20
IF I=3 THEN GOTO LINE 30
IF I=4 THEN GOTO LINE 40

If I is < = 0 or > (number of line numbers) then the following statement is executed.

If I attempts to select a non-existent line, an error message will result. As many line numbers as will fit on a line can follow an ON ... GOTO.

(ii) ON SGN(X)+2 GOTO 40, 50, 60.

This statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero, and to line 60 if it is greater than zero.

6.2.6.9

RETURN

EXAMPLE(S)

(i) RETURN

Causes a subroutine to return to the statement that follows the most recently executed GOSUB.

6.2.6.10

STOP

EXAMPLE(S)

(i) 100 STOP

BASIC suspends execution of programs and enters the command mode. "STOPPED IN LINE 100" is displayed. To continue program with next sequential statement type in "CONT".

6.2.6.11

WAIT

EXAMPLE(S)

(i) WAIT I, J, K

This statement reads the status of REAL WORLD INPUT port I, exclusive OR's K with the status, and then AND's the result with J until a result equal to J is obtained. Execution of the program continues at the statement following the WAIT statement. If the WAIT statement only has two arguments, K is assumed to be zero. If waiting for a bit to become zero, there should be a one in the corresponding position for K. I, J and K must be ≥ 0 and ≤ 255.

(ii) WAIT MEM I,J,K

WAIT MEM I,J

As example (i), but I is a memory location, which of course may be a memory-mapped I/O port.

(iii) WAIT TIME I

Delays program execution for a time given by the expression I. The result should be in the range 0 to 65535.

Time is measured in units of 20 milliseconds.

6.2.7

Physical Machine Access Statements

6.2.7.1

CALLM

EXAMPLES

(i) CALLM 1234

Calls a machine language routine located at the memory locations specified.

(ii) CALLM I,V

Calls a machine language routine located at the memory locations specified by I. Upon entry to the machine language program the register pair H, L contains the address of the variable specified by V. The machine language subroutine must preserve all of the 8080 registers and flags and restore them on return.

If V is a variable, the pointer is to V. If V is a string, the pointer is to a pointer to the string. The string consists of a length byte followed by characters. If V is a matrix, pointer is as though V is a normal variable.

6.2.7.2

INP (I)

EXAMPLE

A = INP (31)

Reads the byte present in the DCE-BUS CARD 3 PORT 1 and assigns it to a variable A. The port-number should be = 0 and = 255.

6.2.7.3

OUT I, J

EXAMPLE

OUT 91,A

Sends the number in variable A to the DCE-BUS card 9 PORT 1. Both I and J must be = 0 and = 255.

6.2.7.4

PDL (I)

EXAMPLE

A = PDL (I)

Sets the variable A to a number between 0 and 255 which represents the position of one of the paddle potentiometers. I must be or = Ø and or = 5.

6.2.7.5

PEEK (I)

EXAMPLES

(i) A = PEEK (#13C2)

The contents of memory address Hex 13C2 will be assigned to the variable A. If I is 65536 or 0 an error will be flagged. An attempt to read a memory location non-existent in a particular configuration will return an unpredictable value.

Displays the value in the decimal memory address 258.

6.2.7.6

POKE

EXAMPLE(S)

(i) POKE I, J

The POKE statement stores the byte specified by its second argument (J) into the memory location given by its first argument (I). The byte to be stored must be > = 0 and < = 255, or an error will occur. If address I is not > = 0 and < 64K, an error results. Careless use of the POKE statement will probably cause BASIC to stop, that is, the machine will hang, and any program already typed in will be lost. A POKE to a non-existing memory location is usually harmless.

Example of POKEs (see also the ASSEMBLY section of the book)

POKE#131,Ø	OUTPUT TO SCREEN AND RS 232
#131,1	OUTPUT TO SCREEN ONLY
#131,2	OUTPUT TO EDIT BUFFER
#135,2	READ (INPUT) FROM EDIT BUFFER
#13D,#10	SELECT CASSETTE 1,#20 FOR CASSETTE2
#40,#28	CASSETTE MOTOR CONTROL 1 ON
#40,#28	CASSETTE MOTOR CONTROL 2 ON
#40,#30	CASSETTE MOTOR CONTROL 1 AND 2 OFF
#730,#30	FLOPPY DRIVE Ø ACTIVATED
#730,#31	FLOPPY DRIVE 1 ACTIVATED

See also useful POKES in paragraph (5.9.1 + 2 + 3)

6.2.7.7

UT

EXAMPLE

UT

Calls the Machine Language Monitor.

6.2.8

BASIC System Data & I/O Statements

6.2.8.1

DATA

EXAMPLES

(i) DATA 1, 3, -1E3, -0.4.

Specifies data, read from left to right. Information appears in data statements in the same order as it will be read in by the program.

(ii) DATA "F00", "Z00"

Strings may be read from DATA statements. If the string contains leading spaces (blanks), or commas (,), it must be enclosed in double quotes.

6.2.8.2

GETC

EXAMPLE(S)

(i) A = GETC

The ASCII value of the last character typed on the keyboard. If no character has been typed in since the last GETC statement zero value is returned. Note that GETC forces a scan of the keyboard. Scanning the keyboard too often will cause "key bounce" and keys may appear to be pressed twice when they were only pressed once.

6.2.8.3

INPUT

EXAMPLE(S)

(i) INPUT V, W, W2

Requests data from the terminal (to be typed in). Each value must be separated from the previous value by a comma (,). The last value typed should be followed by a carriage return. A "?" is typed as a prompt character. Only constants may be typed in as a response to an INPUT statement, such as 4.5E-3 or "CAT". If more data was requested in an INPUT statement than was typed in, another "?" is printed and the rest of the data should be typed in.

If more data was typed in than was requested, the extra data will be ignored. The program will print a warning when this happens. Strings must be input in the same format as they are specified in DATA statements.

(ii) INPUT "VALUE";V

Optionally types a prompt string ("VALUE") before requesting data from the terminal.

Typing CONT after an INPUT command has been interrupted due to the BREAK key will cause execution to resume at the INPUT statement. If any error occurs, the INPUT statement will restart completely.

6.2.8.4

PRINT (can be replaced by "?")

EXAMPLES

(i) PRINT X, Y, Z

(ii) PRINT

(iii) PRINT X, Y

(iv) PRINT "VALUE IS", A

(v) ? A2, B

Prints the numeric or string expressions on the terminal. If the list of values to be printed out does not end with a comma, (,) or a semicolon (;), then a new a new line is output after all the values have been printed. If a semicolon separates two expressions in the list, their values are printed next to each other. If a comma appears after an expression in the list, the cursor is positioned at the beginning of the next column field. If there is no list of expressions to be printed, as in example (ii), then the cursor goes to a new line.

There are 5 fields on the line in positions Ø, 12, 24, 36, 48.

6.2.8.5

READ

EXAMPLE

READ V,W

Reads data into a specified variables from a DATA statement. The first piece of data read will be the first not read by any previous data statement. A RUN or RESTORE statement restarts the process from the first item of data in the lowest numbered DATA statement in the program. The next item of data to be read will be the first item in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an error message.

6.2.8.6

RESTORE

EXAMPLE

(i) RESTORE

Allows the re-reading of DATA statements. After a RESTORE, the next item of data read will be the first item listed in the first DATA statement of the program, and so on as in a normal READ operation.

6.2.9

Cassette and Disc I/O Statements

Additional Cassette and Disc commands are available using the Resident Machine Utility Program (See Section 6.3).

6.2.9.1

CHECK

The CHECK command scans a cassette tape or disc and examines all the files. The type and name of each is printed followed by the word "OK" or "BAD" depending upon the file checksumming correctly. For cassettes the command does not stop of its own accord, but will stop if the BREAK key is held down.

6.2.9.2

LOAD

EXAMPLES

(i) LOAD "FRED"

Loads the program named "FRED" from the cassette tape or disc. When done, the LOAD will type a prompt as usual. The file name may be any string of printable characters.

(ii) LOAD

Loads the first program that is encountered on the tape. If

the recorder motor is under automatic control it will be started. Otherwise the recorder should be started manually.

If a LOAD command is executed directly, not as part of a program, then as each data block or file is passed on the tape, its type (0 for a BASIC program) and its name will be printed. When the load is finished succesfully, a prompt is printed. If the LOAD is unsuccessful, then a message "LOADING ERROR" is printed. It is followed by a number giving details of the problem. The flashing of the cursor will cease while the data is being read from the tape.

6.2.9.3

LOADA

Loads ARRAY or Machine Language programs stored as arrays.

Example LOADA A$ "FRED" or LOADA F$ + "J"

FRED or J are the array names.

```
10 DIM A$ (Ø,Ø)          100 DIM A$ (Ø,Ø)
20 INPUT A$              110 LOADA A$
30 SAVEA A$ "INFO"       120 GOTO 100
40 GOTO 10
```

6.2.9.4

SAVE

EXAMPLE

(i) SAVE "GEORGE"

(ii) SAVE A$

Saves on cassette tape or disc the current program in the memory. The program in memory is left unchanged. More than one program may be stored on one cassette/disc using this command. The program is written on the cassette under the name given.

(iii) SAVE

The program is written on the cassette under a null name.

The system replies to the command with the message "SET RECORD, START TAPE, TYPE SPACE". Place the tape recorder into the right state for recording (note that if the motor control is connected to the Personal Computer, the motor will not yet start). Then press the space key. When the motor will stop (if automatically controlled) a prompt character will appear on the screen. If the cassette is working manually, then it should now be stopped.

6.2.9.5
SAVEA

EXAMPLE
(i) SAVEAG "GEORGES"
(ii) SAVEA A$
Saves an array on cassette or disk.
(iii) SAVEA A
EXAMPLE
```
20 INPUT A$
30 SAVE A$
40 GOTO 10
```

After typing RUN and pressing RETURN key the tape recorder will start automatically to record the input you enter in line 20 (the tape recorder must have a remote control and must be in recording mode).

COPY OF A PROGRAM FOLLOWED BY AN ARRAY (OR MACHINE LANGUAGE ROUTINE) WITH 2 TAPE RECORDERS (1 BEING ON PLAY, 2 ON RECORD).

```
POKE #40,#28 : LOAD : POKE #40,#18 : SAVE : POKE #40,#28 :
PRINT "SAVE ENDED" : CLEAR 2000 : DIM A (20, 20) : LOAD A :
POKE  40,  18
SAVEA A POKE  40,  28
```

PRESS RETURN: the array is named A.

6.2.10
Program Debug and Comment Statements

6.2.10.1
CONT

EXAMPLE
CONT
Continues BASIC program execution with the next statement following the "STOP" Statement or "BREAK" position.

6.2.10.2
REM

EXAMPLES
(i) REM NOW SET V=0
Allows comments inside BASIC programs. REM statements are not executed, but they can be branched to. A REM statement is terminated by end of line, but not by a (:) character.
(ii) REM SET V=0:V=0
The V=0 statement will not be executed.
(iii) The V=0 statement will be executed.

6.2.10.3
STEP

Command to allow single step execution of BASIC programs. After "BREAK" or "STOP" the operator types in STEP and then each depression of the space bar allows execution of the next sequential BASIC line. The line to be executed is displayed before execution of that line.

6.2.10.4
TRON

EXAMPLE

(i)
```
100 A = 0
105 TRON
106 A = 1
107 A = 2
108 TROFF
```

When you RUN, and after the TRON (TRACE ON) is executed the lines 106 and 107 will be executed and displayed at the same time until the TROFF (TRACE OFF) is reached and executed.

6.2.10.5
TROFF

EXAMPLE SEE 6.2.10.4

6.2.11
Array and Variable Statements

6.2.11.1
CLEAR

EXAMPLE

(i) CLEAR 999

Resets all variables to Ø or the null string, and returns all space assigned to arrays. The size of the HEAP (array and string storage) is than set to the number specified by the CLEAR statement. The minimum size is 4 (no space would be available) and the maximum is 32767

6.2.11.2
DIM

EXAMPLE

(i) DIM A(3), B(10)

(ii) DIM R3(5,5), D$(2,2,2)

Allocates space for arrays. Arrays can have more than one dimension. All subscripts start at zero (0), which means that DIM X (100) really allocates 101 matrix elements. The maximum size for a dimension is 254 Dimensions may be specified as variables or expressions.
DIM statements may be re-executed to vary the size of an array. The space used for arrays is in the same part of RAM as that for strings, the size of which is set by the CLEAR command.

6.2.11.3
FRE

EXAMPLE(S)

(i) A = FRE

The variable A is set to the number of memory bytes currently unused by the BASIC program. Memory allocated for string and arrays is not included in this count.

(ii) PRINT FRE

The amount of remaining memory space will be displayed.

6.2.11.4
LET

EXAMPLE(S)

(i) LET W = X

(ii) V = 5.1

Assigns a value to a variable. The word "LET" is optional.

6.2.11.5
VARPTR (V)

EXAMPLE(S)

(i) A = VARPTR (B)

Variable named (A) is set to the memory address of the variable named (B).

(ii) A = VARPTR (B(3,4))
Variable named (A) is set to the memory address of the array element B(3,4).

6.2.12
GRAPHICS AND DISPLAY STATEMENTS (See Example program "TOWER OF HANOI")

6.2.12.1
MODE

EXAMPLE(S)
(i) MODE 0
Places display in character only mode.
(ii) MODE 1A
Places display in split mode. Low resolution graphics with 16 colours and a four line character display at the bottom.

The Personal Computer has 3 different graphic definitions available for the graphics display and at each definition there are 4 possible configurations of the screen. Two of these have only graphics on the screen, and the others are exactly the same except that the graphics area is moved up the screen to make room for four lines of characters. The graphics hardware has 2 different ways in which it can be used. That is why at each definition there are 2 different types of display. The display types are known as 16-colour, and 4-colour modes. In the 16 colour modes each point on the screen can be set to any of the 16 colours. However each field of 8 dots horizontally (positions 0 to 7, 8 to 15 etc.) can only have 2 or sometimes 3 separate colours in it. For exact details of the restrictions on what can be drawn. (See 3.2.2.1) At any time the 4 selected colours can be altered, and the existing picture changes colour immediately. This allows interesting effects. (see for instance "ANIMATE").

MODE DEFINITION TABLE

Number	Graphics size	Text size	Type of graphics
0	-	24 X 60 CHAR	-
1	72,65	-	16 colour
1A	72,65	4 X 60	16 colour
2	72,65	-	4 colour
2A	72,65	4 X 60	4 colour
3	160,130	-	16 colour
3A	160,130	4 X 60	16 colour
4	160,130	-	4 colour
4A	160,130	4 X 60	4 colour
5	336,256	-	16 colour
5A	336,256	4 X 60	16 colour
6	336,256	-	4 colour
6A	336,256	4 X 60	4 colour

6.2.12.2
COLORG

EXAMPLE
COLORG 1 2 3 4
Sets the colours available in any four colour graphics mode to 1,2,3 and 4.
If the screen is already in a 4 colour mode, then the colour change will be immediate. Any area which was in the first-named colour of the previous COLORG statement, is now displayed in colour 1, and so on.
If the screen is in a 16 colour mode, no immediate effect is visible.
In any event, the next time a new graphics mode is entered, the initial colour of the graphics area will be the first colour given in the COLORG command. This applies both for 4 and 16 colour modes.
If COLORG has not been used, then after a 4 colour mode command (i.e. mode 2) the colours available will be Ø, 5, 1Ø, 15.

6.2.12.3
COLORT

EXAMPLE
COLORT 8 15 0 0
Sets up colour number 8 as the background colour for the text screen and colour 15 as the colour of the characters. The other two colour numbers are not normally used. However they define an alternative set of colours which can be used by POKE access, or machine code routines.

6.2.12.4
Drawing Facilities

Points on the graphic screen are specified by an X,Y co-ordinate with 0, 0 located at the bottom left corner of the display screen. An attempt to draw out of the maximum area for a particular graphics mode will result in an error.

It is possible, however, to draw in the invisible top section of the graphics area in split screen modes. The drawing facilities provide statements to draw dots, lines and rectangles on the graphic display screen. The DOT statement places a single dot of a specified colour at any allowable X, Y coordinate on the display statement allow the drawing of a line and the colouring of a rectangular area specified by two X, Y coordinates. See color codes paragraph 3.2.12.

6.2.12.4.1
DOT

EXAMPLE(S)
(i) DOT 10, 20 15
Places a dot of colour 15 at the position X = 10 and Y = 20. The size of the dot will depend upon which graphic resolution was selected.

6.2.12.4.2
DRAW

EXAMPLE
DRAW 91,73 42,77 15
Draws a line in colour 15 between 91, 73 and 42, 77. There is no restriction on the order of the coordinates. Line width will depend upon which resolution was selected.

6.2.12.4.3
FILL

EXAMPLE
FILL 91,73 42,77 15
Fills the rectangle with opposite corners at 91, 73 and 42, 77 with the colour 15. There is no restriction on the order of the points. The physical size of the rectangle depends upon the resolution selected.

6. 2. 12. 5

Animated Drawing Facility.

With the screen in a 4 colour mode each point is described by 2 bits. The binary value of these 2 bits selects which of the four available colours should be displayed. Normally a DOT, DRAW or FILL sets both of these bits to their new value. However, a facility is available to set or clear only one of the two. This is accomplished by specifying colour numbers 16, 17, 18 or 19. It is emphasized that these are not real colours, but an extra facility.

For example:

```
MODE 2A
COLORG 6 9 12 15
```

These commands set all points on the screen to colour 6. The two bits for each point on the screen are both Ø. (Binary Ø Ø).

```
DOT 10, 10 17
```

This sets the lower bit only for point 10, 10. Thus the point changes to colour 9 (Binary 0 1).

```
DOT 10,10 19
```

This sets up the upper bit only. The point changes to colour 15 (binary 11 = 3)

```
DOT 10,10 16
```

This clears the lower bit, and gives colour 12 (binary 10 = 2).

```
DOT 10, 10 18
```

This clears the upper bit, and gives colour 6 (binary 00). The usefulness of this system is that by the COLORT command two pictures can be independently maintained and altered on the screen. This allows one pattern to be changed invisibly while the other is displayed. The pictures can be swapped instantaneously and the invisible one changed.

Example program:

```
 5  MODE 2
10  COLORG Ø Ø Ø Ø
20  FOR Q = 1 TO XMAX
30  DRAW Ø,Ø Q,YMAX 17+2 * A:REM COLOR = 17 OR 19.
40  COLORG Ø 15 - 15 * A   15 * A   15:REM COLOR = 18 OR
                                                    16.
```



```
50 DRAW Ø,Ø Q - 1, YMAX 18-2  A : A = 1 - A : NEXT
```

"ANIMATE"

When the screen is in a 4 colour mode, each point on the screen is described by 2 bits. A facility is provided for drawing using only one bit from each pair, without affecting the other.

Drawing using the number	has effect of
17	set lower bit
19	set upper bit
16	clear lower bit
18	clear upper bit

This allows two totally independent pictures to be maintained and separately updated. They simply appear to overlap. If the SCOLG entrypoint is used to make only 1 visible at a time, then animation effects can be achieved.

If the colours set by the SCOLG command are numbered 0, 1, 2, 3 in order as given, then the colour seen on the screen is selected by the two bits for each point in the natural way.

E. g.

If SCOLG sets up red, yellow, green and blue, in that order

Upper Bit	Lower Bit	Visible Colour
0	0	Red
0	1	Yellow
1	0	Green
1	1	Blue

"Colours 20 to 23"

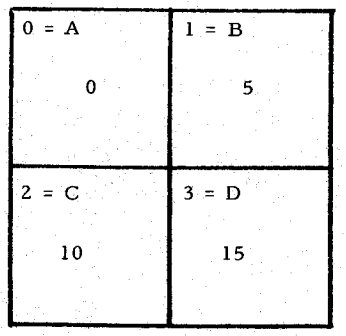
In 4 colour mode only, the colour numbers 20 to 23 may be used to request the 4 colours set up by the last SCOLG call. Colour 20 always refers to the first colour given irrespective of what it is. Similarly 21 is the second colour, and so on.

The "animate" facility using colours 16 to 19 can be explained as a 4 boxes square where a colour is assigned to a box.

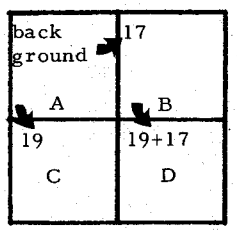
Number 0 1 2 3 of the

COLORG A B C D command assigning a color to each box.

A DOT, DRAW or FILL Command with a 16 to 19 colour definition will move the background and foreground colours as indicated by the arrows.



16 ←
17 →
18 ↑
19 ↓



COLORG 0 0 15 15

COLORG 0 15 0 15

6.2.12.6

XMAX

EXAMPLE

A = XMAX

Sets the variable A to the maximum allowable X value for the current graphics mode.

6.2.12.7

YMAX

EXAMPLE

A = YMAX

Sets the variable A to the maximum allowable Y value for the current graphics mode.

6.2.12.8

SCRN (X,Y)

EXAMPLE

(i) A = SCRN (31,20)

Sets the variable to a number corresponding to the colour of the screen at coordinate 31,20.

6.2.12.9

CURSOR

EXAMPLE

(i) CURSOR 40,20

Moves the cursor to the fourtieth character position of the twentieth line from the bottom of the screen.

The cursor can be moved to any position on the screen by using the CURSOR command. The positions are given by X, Y coordinates where the bottom left corner of the screen is 0,0.

6.2.12.10
CURX

EXAMPLE
A = CURX
Sets the variable A to the X position of the cursor (character position). Value returned will be < = 60.

6.2.12.11
CURY

EXAMPLE
A = CURY
Sets the variable A to the Y position of the cursor (line position). Value returned will be < = 24.

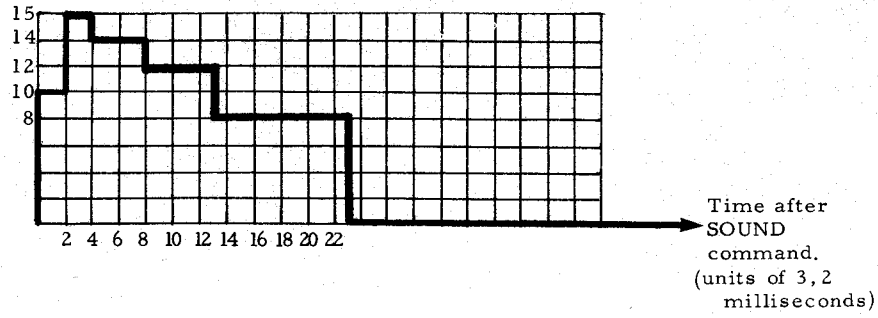
6.2.13
Graphical Sound Statement.

6.2.13.1
Programmable Sound Facility

The Graphical Sound Generator of the DAI Personal Computer is supported by the BASIC to give a set of commands that allow program control of the sound system, 3 oscillator channels plus a white noise channel. The SOUND command is the primary method of control. The SOUND command specifies a channel to which it applies, an envelope to be used, the required volume and requency. A simple sound command would be:

SOUND 0 1 15 0 FREQ (1000)

This would set channel 0, using envelope number 1, at a volume of 15 and frequency 1000 Hz. The ENVELOPE statement allows the volume of a note to be rapidly changed, in the same way as that of a musical instrument. Thus the rise and fall in volume for a note can be specified. The command specifies a set of pairs of volume and time. The volume constants are in the range 0 to 15 and the time is in units of 3.2 milli-seconds. For example the command:
ENVELOPE 0 10,2;15,2;14,4;12,5;8,10;0
This sets a volume envelope like this:



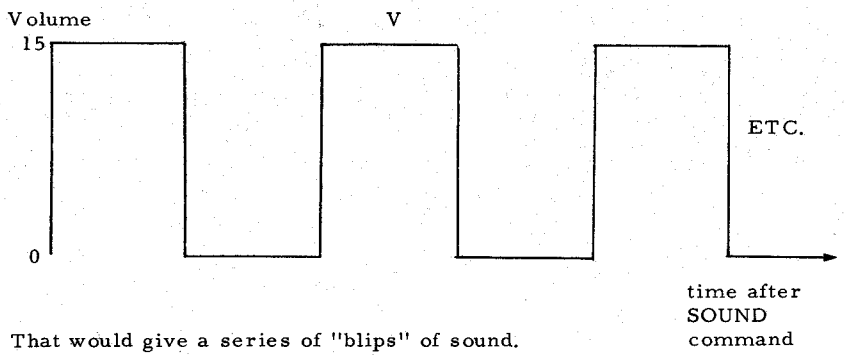
So every time a SOUND command is given it produces a short burst of sound whose volume is as shown above. Varying the envelope varies the quality of the sound heard.

The volume given in a SOUND command is effectively multiplied by that in the envelope. So if the SOUND command requests a volume of 8 units, which is 8/15 of full volume, and the envelope requests 4 units, which is 1/4 of the maximum figure, then the volume used is 2/15 of the maximum. (as 1/4 x 8/15 = 8/60 = 2/15.)

The envelope command can end, as above, in a single volume, in which case that volume continues for ever, or in a pair of volume and time, in which case the envelope is repeated indefinitely. For example:

ENVELOPE 0 15,10;0,10;

Sets an envelope like this:



That would give a series of "blips" of sound.

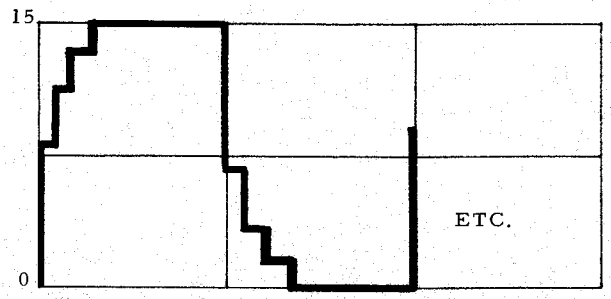
The simplest envelope is obviously:

ENVELOPE 0 15

Which then has no audible effect on SOUND commands, as all volumes are multiplied by 15/15.

Special note:

The BASIC Interpreter limits the rapidity with which the volume on any channel is allowed to change. The maximum change is d/2 + 1, where d is the difference between the requested and current volumes. Thus the actual volume output for the envelope above would be:



This helps reduce spurious sound caused by volume changes.

The noise generator is controlled by a NOISE command that controls the audible output of the white noise generator. Only its volume and envelope can be set. e.g. :

NOISE 0 15

Turns on the noise channel using envelope 0 and overall volume 15.

In addition to the facilities already described, the SOUND command controls 2 others. They are TREMOLO and GLISSANDO.

Tremolo is simply a rapid variation of volume by $\pm$ 2 units. This gives a "warbling" effect to the sound. Glissando is an effect where the new note on a channel does not start immediately at the requested frequency, but "slides" there from the previous frequency. The effect resembles a Hawaiian Guitar or Stylophone. Glissando + Tremolo are controlled by one parameter in the SOUND command. Setting the bottom bit requests Tremolo and the next bit Glissando. E.g. :

(i) SOUND 0 0 13 1 FREQ (1000)

(ii) SOUND 0 0 15 2 FREQ (5000).

The first example sets channel 0, using envelope 0, at volume 13 and with tremolo. The volume put will vary rapidly from 11 to 15.

The second example increases the volume to 15, and slides the frequency "GLISSANDO" up to 5000 Hz. The flexibility and facilities of the Graphical Sound Generator have been illustrated fully and their capabilities exploited with the three commands previously discussed.

Due to the flexibility of change in volume and frequency it is quite feasible to explore the possibilities of vocal sound generation. The BASIC of the DAI Personal Computer gives full control to the programmer who wishes to develop experimentally a burst of sound and frequencies that result in audible words.

6.2.13.2

SYNTAX : SOUND

(i) SOUND <CHAN><ENV> <VOL><TG> FREQ <PERIOD>
(ii) SOUND <CHAN> OFF
(iii) SOUND OFF

< CHAN> is an expression in the range 0 to 2. It selects programmable oscillator 0, 1 or 2.
< ENV> is an expression in the range 0, 1. It selects which of the 2 previously defined envelopes should be used.
< VOL> is an expression in the range 0 to 16. It selects the volume for this particular sound. It is multiplied by the volumes in the ENVELOPE specified.
< TG> is an expression in the range 0 to 3.

0 selects no tremolo + no glissando
1 selects tremolo + no glissando
2 selects no tremolo + glissando
3 selects tremolo + glissando

< PERIOD> is an expression in the range 2 to 65535. It sets the period of the required sound in units of 1/2 microseconds.

6.2.13.3

SYNTAX: ENVELOPE

(i) ENVELOPE <ENV> {<V> , <T> ;} <V> , <T> ;
(ii) ENVELOPE <ENV> {<V> , <T> ;} <V>

ENV is an expression in the range 0 to 1. It selects which of 2 envelopes is being defined.

V is an expression in the range 0 to 15. It selects a volume level by which that in a SOUND command is to be multiplied.

T is an expression in the range 1 to 254. It selects the time for which the volume V applies. It is in units of 3.2 milliseconds.

Note: The parts of the command in curly brackets are optional and may be absent or repeated as many times as required.

6.2.13.4

SYNTAX: NOISE

(i) NOISE ENV VOL
(ii) NOISE OFF

ENV is an expression in the range Ø to 1.
VOL is an expression in the range 0 to 15.

This represents a 4 bit binary number. The top 2 bits of this number (when modified by the ENVELOPE specified) control the volume of the noise. The bottom 2 bits control the frequency.

6.2.13.5

FREQ

EXAMPLE

A = FREQ (1000)

Sets the variable A to a number that can be sent to a Graphical Sound Generator channel to result in a 1000 hertz rate.

6.2.13.6

Synthesing Vocal Sound.

6.2.13.6.1

TALK

TALK ADDRESS

CODE	DATA
0	2 BYTES FREQ. CODE CHANNEL 0
2	" " 1
4	" " 2
8	1 BYTE VOLUME CHANNEL 0
9	" " 1
A	" VOLUME W. NOISE GENERATOR
C	2 BYTES DELAY IN UNITS OF MSEC
D	CALL MACHINE CODE
FF	END

DATA BLOCK

	location	content	
#2000	20 00	09C4	set channel 0 freq. 800
	20 02	1AØA	set channel 1 freq. 300
	20 08	0F	set maximum volume ch Ø
	20 09	0F	set maximum volume ch 1
	20 0C	FEFE	set + listen to it for ---- msec
	20 08	00	turns volume down
	20 09	00	
	20 0D	0050	machine codes at 5000
	20 FF		End.

#5000	ØØ	[LXI H, VARPTR (Q(Ø))]	21 ØØ 20
5004		RETURN	C9

```
Ex.    3  CLEAR 1000
       4  DIM Q (100)
       5  B% = VARPTR (Q(Ø))
      1Ø  READ A%
      2Ø  POKE B%, A% : B% = B% + 1
      3Ø  IF A% <> #FF GOTO 1Ø
      4Ø  TALK VARPTR (Q(Ø))
     (5Ø  WAIT TIME 1Ø)
      6Ø  GOTO 4Ø
      8Ø  DATA Ø, 9, #C4, 2, #1A, #A, 8, #F, 9, #F
      9Ø  DATA #C, #FE, #FE, 8, Ø, 9, Ø, #FF
```

6.2.14

Arithmetic and String Functions

The following is a list of the mathematical + character handling functions provided by BASIC. Each takes a number of expressions (arguments) in brackets and works on them to return a result. This result may be used in just the same way as a variable or constant in expressions.

EXAMPLES

(i) A = 3.0 + 2.1

(ii) A = SIN (3.0) + 2.1

6.2.14.1
ABS(X)

Gives the floating point absolute value of the expression X. ABS returns X if X > = 0, -X otherwise. For example ABS(-253.7) = 253.7.

6.2.14.2
ACOS(X)

Returns arc cosine of X. Result is between -PI/2 and PI/2.

6.2.14.3
ALOG(X)

Returns antilog base 10 of X.

6.2.14.4
ASC(X$)

Returns the integer ASCII value of the first character of the string X$.
E.g.: ASC("ABC") returns 65 since A has code 41 Hex or 65 decimal.

6.2.14.5
ASIN(X)

Returns the arcsine of X in radians. Result is between -PI/2 and +PI/2.
X may be any value between + 1 and - 1 inclusive.

6.2.14.6
ATN(X)

Returns the arctangent of X in radians.

6.2.14.7
CHR$(I)

Inverse of ASC. Returns a 1 character string whose ASCII value is I.
I must be between 0 and 255.
E.g. : CHR$ (65) returns the character "A".

6.2.14.8
COS(X)

Gives the cosine of the expression X, measured in radians. (X) may be any value between 0 and 2π inclusive.

6.2.14.9
EXP(X)

Returns the value "e" (2.71828) to the power X, (e ↑ X). "e" is the base for natural logarithms. The maximum argument that can be passed to EXP without overflow occurring depends on whether the software or hardware maths option is being used. For hardware
- 32 < X < 32 exactly.
For software -43 ≤ X ≤ 43 approximately.

6.2.14.10
FRAC(X)

Returns the floating point fractional part of the argument.
e.g.: FRAC (2.7) = 0.7, FRAC (-1.2) = -0.2

6.2.14.11
HEX$ (I)

EXAMPLE(S)
Returns a string of characters representing the hexadecimal value of the number I. I must be between 0 and 65535.

6.2.14.12
INT(X)

Returns the largest integral floating point value less than or equal to its argument X. For example:
INT(.23) = 0, INT(7) = 7.0, INT(-2.7) = -3.0, INT(1.1) = 1.0
INT (43.999) = 43,0
Note: INT(-1) = -2.0.

6.2.14.13
LEFT$(X$,I)

Returns a string which is the leftmost I characters of the string X$.
E.g.: LEFT$("DOGFISH",3) equals "DOG"

6.2.14.14
LEN(X$)

Returns an integer giving the length in characters of the string X$.
E.g.: LEN("HELLO") equals 5.

6.2.14.15
LOG(X)

Calculates the natural logarithm (base e) of the argument (X).

6.2.14.16
LOGT(X)

Calculates the logarithm base 10 of X.

6.2.14.17
MID$(X$,I,J)

Returns (J) characters starting at position I in the string (X$). The first character is position 0.
E.g. : MID$ ("SCOWL", 1,3) returns "COW".

6.2.14.18
PI

Returns the floating point value 3.14159

6.2.14.19
RIGHT$(X$,I)

Returns the rightmost (I) characters of string (X$).
E.g. : RIGHT$("SCOWL", 3) returns "OWL".

6.2.14.20
RND(X)

Generates a hardware or software generated random number.
E.g.

If X < 0 Starts a new sequence of software numbers with X as seed. The same negative X produces the same sequence of numbers. The number returned is between 0 and X

If X > 0 Returns the next pseudo-random number from the current sequence. The number is in the range 0 to X

If X = 0 Returns a hardware generated random number in the range 0 to 1.

Ex.

```
5     CLEAR 1000
10    DIM B% (100)
20    INPUT C%
30    FOR A% = 1 TO 20
```

```
40      B% (A%) = RND (C%)
50      PRINT B% ( A%)
60      NEXT A%
```

6.2.14.21
SGN(X)

Returns 1.0 if X > 0, 0 if X = 0, and -1.0 if X < 0.

6.2.14.22
SIN(X)

Calculates the sine of the variable X. X is in radians.
Note: 1 Radian = 180/PI degrees = 57.2958 degrees; so that the sine of X degrees = SIN(X/57.2958).

6.2.14.23
SPC(I)

Returns a string of the number of spaces given by I. I ≤ 255.

6.2.14.24
SQR(X)

Gives the square root of the argument X. An error will occur if X is less than zero.

6.2.14.25
STR$(X)

Returns a string which is the ASCII representation of the number X.
E.g.: STR$ (9.2) returns the string "9.2".

6.2.14.26
TAB(I)

Returns a string of the number of spaces necessary to move the screen cursor right to the column given by I. The cursor can only be moved to the right.

6.2.14.27
TAN(X)

Gives the tangent of the expression X, X must be expressed in radians.

6.2.14.28
VAL(X$)

Returns the floating point value of the number represented by the string variable X$.
E.g. : VAL ("9.2") returns 9.2
X$ must represent a valid floating point number.

6.2.15

Arithmetic and Logical Operators

Operator	Usage	Type of Result
+ (addition)	int + int	int
	fpt + int, int + fpt, fpt + fpt (Note 1)	fpt
	str + str	str
-/* (subtract, divide, multiply)	as +, except no string version	
↑ (power (^ on keyb.)	as	always fpt
IAND, IOR, IXOR, MOD, SHL, SHR	int ... int, int ... fpt, fpt ... int, int ... int	integer (Note 2)
INOT	int	integer
= equal	str ... str, fpt ... fpt, fpt ... int, int ... fpt (Note 1), int ... int	logical
greater than		
smaller than		
different from		
= greater than or equal to		
= smaller than or equal to		
AND OR	logical ... logical	logical

Note 1: The integer values are converted to fpt before use.
Note 2: The fpt values are truncated to integer before use.

EXAMPLE(S)

(Numbers without decimal parts represent integers)

(i) Operation	Result	Type of Result	
1 + 2	3	integer	
1.0 + 2.0	3.0	fpt	
1.0 + 2	3.0	fpt	
3 * 4	12	integer	
3 ↑ 4	81.0	fpt	NB
12.0/4.0	3.0	fpt	
12.0/4	3.0	fpt	
12/4	3	integer	
11/4	2	integer	NB
3 IAND 2	2	integer	
3.0 IAND 6.0	2	integer	
3.14 IAND 6.72	2	integer	
3 SHL 2	12	integer	
3.2 SHL2.1	12	integer	
7 = 4	FALSE	logical	
3.0 > 2.1	TRUE	logical	
"FRED" < "FREDA"	TRUE	logical	
"A" = "A"	TRUE	logical	
7.1 = 7	FALSE	logical	
7.0 = 7	TRUE	logical	NB
3 < 4 OR 7 = 8	TRUE	logical	
3 = 7 AND 9 < 10	FALSE	logical	

(i) (In all of the cases below, leading zeroes on binary numbers are not shown).

63 IAND 16 = 16	Since 63 equals binary 111111 and 16 equals binary 1000 , the result of the IAND is binary 1000 or 16.
15 IAND 14 = 14	15 equals binary 1111 and 14 equals binary 1110, so 15 IAND 14 equals binary 1110 or 14.
-1 IAND 8 = 8	-1 equals binary 11 .... 11 and 8 equals binary 1000, so the result is binary 1000 or 8 decimal.
4 IAND 2 = 0	4 equals binary 100 and 2 equals binary 10, so the result is binary 0 because none of the bits in either argument match to give a 1 bit in the result.
4 IOR 2 = 6	Binary 100 IOR'd with binary 10 equals binary 110 or 6 decimal.
10 IOR 10 = 10	Binary 1010 IOR'd with binary 1010 equals binary 1010, or 10 decimal.
- 1 IOR -2 = -1	Binary 11.... 11 (-1) OR'd with binary 11.... 10 (-2) equals binary 11.... 11 or -1.

The following truth table shows the logical operations on bits:

Operator	Arg. 1	Arg. 2	Result
IAND	1	1	1
	0	1	0
	1	0	0
	0	0	0
IOR	1	1	1
	1	0	1
	0	1	1
	0	0	0
INOT	1	-	0
	0	-	1

A typical use of the bitwise operators is to test bits set in the REAL WORLD input ports which reflect the state of some REAL WORLD device.

Bit position 7 is the most significant bit of a byte, while position 0 is the least significant.

For instance, suppose bit 1 of REAL WORLD port 5 is 0 when the door to Room X is closed, and 1 if the door is open. The following program will print "Intruder Alert" if the door is opened:

```
10 IF (INP(5)IAND 2) = 2 THEN 10
```

This alert will execute over and over until bit 1 (masked or selected by the 2) becomes a 1. When that happens, we go to line 20.

```
20 PRINT "INTRUDER ALERT"
```

Line 20 will output "INTRUDER ALERT".

However, we can replace statement 10 with a "WAIT" statement, which has exactly the same effect.

```
10 WAIT 5,2
```

This line delays the execution of the next statement in the program until bit 1 of REAL WORLD port 5 becomes 1. The WAIT is much faster than the equivalent IF statement and also takes less bytes of program storage.