GLENTOP

AMSTRAD CPCs

ADVANCED USERS-GUIDE BOOK 1

Daniel Martin

AMSTRAD

ADVANCED USERS GUIDE

by

Daniel Martin

Glentop Publishers Ltd

October 1986

ISBN 1 85181 122 2

Published by:
Glentop Publishers Ltd
Standfast House
Bath Place
High Street
Barnet
Herts EN5 5XE
Tel: 01–441–4130

About the Author

Daniel Martin, who wrote the original, French, version of this book, spent a brief period with the French National Ministry of Education before succumbing to the attractions of micro-computers – which have fascinated him since 1978 – and taking a job as a computer manager with the Tandy Corporation for eighteen months. He then worked for Apple in the Netherlands and is currently a systems engineer with Intertechnique, a major French manufacturer specialising in micro-computers based on the *PICK* system.

He wrote *Le livre du MSX (The MSX book)* in December 1984, *Les dessous du Spectravideo (Underneath the Spectravideo)* in February 1985. He is currently writing *L'assembleur du QL Sinclair (The Sinclair QL assembler)* and is preparing *Livre de l'Amstrad (The Amstrad book)*.

CONTENTS

INTERNAL ARCHITECTURE

GENERAL LAYOUT AND SPECIFICATIONS

The block diagram on the following page shows the main circuits making up the equipment.

The system is organised around a Z80 Central Processing Unit with a 4Mhz clock.

The most important circuit of the Amstrad, with the exception of the microprocessor itself, is the *gate array* which contains all the system control logic. In particular, it controls the colour, the screen mode and the Read Only Memory (ROM).

Together with the CRTC 6845 (Cathode Ray Tube Controller) the gate array controls all the video signals for the monitor (screen).

Another important circuit is the PSG AY3 8912 (PSG stands for Programmable Sound Generator). This circuit contains three separate channels, with a sound generator and envelope control for each channel. Programming is described in Chapter 5.

The system also has an Input/Output port which can be used to read the keyboard and joystick.

The PPI 8255 plays an important role in controlling the joystick, the parallel print port, the tape recorder and in the selection of keyboard columns.

The system has 64K of Random Access Memory (RAM) and 32K of Read Only Memory (ROM), the latter containing the operating system and BASIC.

The 32K ROM is part of the central circuitry and is divided into two blocks of 16K. The lower 16K block occupies addresses 0000 to 3FFF, the upper 16K block occupies addresses from C000 to FFFF.

These two memories can be handled separately, in or out of the circuit, under the control of the gate array.

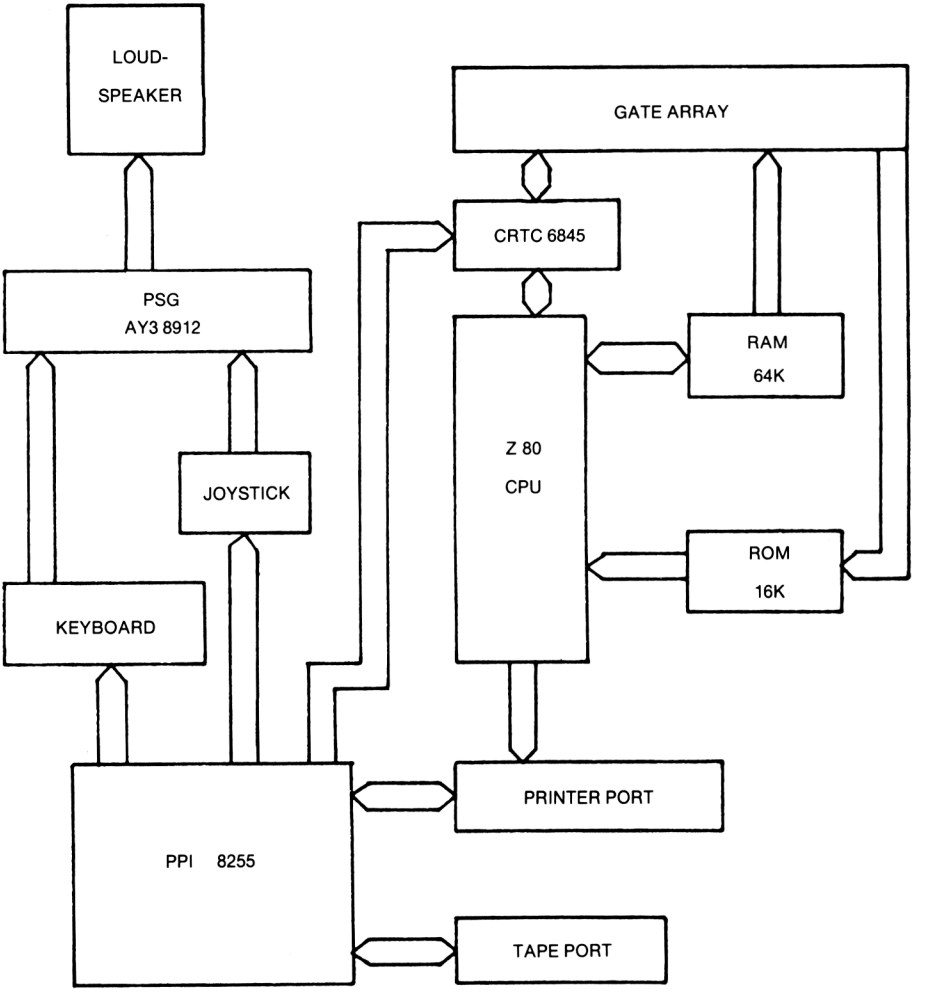
There is a signal on the port extension which can be used to disconnect the internal Read Only Memory and permit external memory access to the processor. This allows for example, for the use of floppy disks.

RAM consists of 64K bytes from address 0000 to FFFF. The lower and higher 16K blocks thus share addresses with the ROM.

Normally this will cause no problems since when writing, only RAM is capable of being affected and, when reading, it is possible to select between either ROM or RAM, depending on what you want to read.

The screen memory occupies 16K in the central memory area and can be found at addresses 0000, 4000, 8000 or C000. Generally, on startup, it will be located at address C000.

BLOCK DIAGRAM



BASIC

GENERAL FEATURES

Maximum memory space available: 43533 bytes

Variable names: 1 to 40 characters

Data

Integers: from -32768 to 32767

Single precision: from 293874 E − 39 to 170141 E30, to nine significant figures, or to six in exponential form.

String size: 0 to 255 characters

Length of program lines: 255 characters maximum

Program line numbers: from 1 to 65535

Memory requirements: a single line of BASIC occupies a minimum of 6 bytes, 2 for the line numbers, 2 for the length of the line, 1 for the separator and 1 for a minimum instruction (eg. REM, PAINT)

Allocation of Variables

Positive integers from 1 to 9: 1 byte

Negative integers from 1 to 9: 2 bytes

Positive integers from 10 to 255: 2 bytes

Negative integers from 10 to 255: 3 bytes

Positive single precision (255–65535): 3 bytes

Negative single precision (255–65535): 4 bytes

Positive integer above 65535 or positive non-integer: 6 bytes

Negative integer above 65535 or negative non-integer: 7 bytes

Note:
The words 'single precision' and 'real' are used synonymously in this book.

BASIC INSTRUCTIONS

AFTER

AFTER X, [Y] GOSUB N

Calls a program subroutine after waiting X 50ths of a second. Y (optional) indicates which clock to use. There are four clocks, numbered from 0 to 3, if no clock is specified, this defaults to 0.

AUTO

AUTO [N], [X]

Provides automatic line numbers, starting at line N and with line number intervals of X. N and X default to 10.

BORDER

BORDER X, [Y]

A and Y represent the numbers of the colours (0 to 26) to be used for the screen border. If Y is specified then the two colours alternate at a speed determined by the command SPEED INK.

CALL

CALL ADR [, list of parameters]

This command is used in BASIC to call a machine code subroutine located at address ADR. A list of parameters will, if included, be passed to the subroutine.

CAT

CAT

Reads the tape and lists the names of the files on it. Does not affect the currently loaded program.

CHAIN

CHAIN name [,N]

Loads a program from tape into central memory, replacing any previous program. It then runs the new program starting from line number N (if specified). If N is not specified then the program executes from the lowest line number.

CLEAR

CLEAR

Erases the contents of *all* variables.

CLG

CLG

Clears graphics.

CLOSEIN

CLOSEIN

Closes a tape file opened for input.

CLOSEOUT

CLOSEOUT

Closes a tape file opened for output.

CLS

CLS [#N]

Clears the screen or the screen window and leaves it coloured according to the last PAPER instruction. N is any channel number from 0 to 7 and corresponds to the screen as defined by the instruction WINDOW.

CONT

CONT

Resumes the running of a program after encountering STOP or

END or after pressing the BREAK key – as long as the program has not been edited in the meantime.

DATA

DATA A,B,C

where A, B and C are data items.
Uses a program line to store a list of values, it is interpreted by the READ function.

DEF FN

DEF FNf [(X,...)] = expr

Used to define a user function; f represents the name of the function, [X,...] represents its formal parameters and expr represents its general expression.

DEFINT

DEFINT X-Y *or* DEFINT X,Y...

Defines a set of variables in the range X-Y or in the list X,Y,... as being permanently of integer type.

DEFREAL

DEFREAL X-Y *or* DEFREAL X,Y...

Defines a set of variables in the range X-Y or in the list X,Y... as being permanently of real (single precision) type.

DEFSTR

DEFSTR X-Y *or* DEFSTR X,Y,...

Defines a set of variables in the range X-Y or in the list X,Y... as being permanently of string (character sequence) type.

DEG

DEG

Sets calculation mode to degrees (trigonometric functions normally use radians). This mode can be reset to use radians by the commands CLEAR and RAD or by loading another program.

DELETE

DELETE (N1,N2...) *or* DELETE N1-N2

Deletes lines N1, N2..., or all lines numbered between N1 and N2 (in the second example) from the currently loaded program.

DI

DI

Disables interrupts.
All commands which generate interrupts, with the exception of BREAK, cease to work.

DIM

DIM var(n) *or* DIM var(N1,N2...) var (n1,n2...)

Dimensions an array (var) from 1 to n. By default a variable is automatically dimensioned to 10 (var (10)).

DRAW

DRAW X,Y,A

Draws a line on the screen, starting at the position of the graphic cursor and moving to the position of the co-ordinates (X,Y) using colour number A.

DRAWR

DRAWR X,Y,A

Draws a line on the screen starting at the position of the graphic cursor and moving to the relative position +X,+Y using colour number A.

EDIT EDIT N

Invokes the editing mode on line number N.

EI EI

Enables interrupts. Cancels the effect of DI.

END END

Instruction to end execution of the program.

ENT ENT NE[,SE]

This defines a tone envelope permitting the addition of vibrato. NE represents the envelope number (0 to 15). SE comprises three quantities for each section (the number of steps, frequency value and time interval value for each); five sections can be described.

ENV ENV NE[,SE]

This defines the volume envelope allowing the definition of sound type. NE represents the envelope number (0 to 15). SE contains three quantities per section (count value, volume level and time for each); five sections can be described.

ERASE ERASE list of names of variables

Frees the memory space reserved by DIM commands.

ERROR ERROR N

N represents an integer. Enables a specific error trap and defines the course of action to be taken on encountering that error.

EVERY EVERY N,M GOSUB LN

The subroutine at line number LN will be executed every N 100ths of a second, counted on clock M. Four clocks are available numbered from 0 to 3. This command allows you to call a subroutine at regular intervals.

FOR FOR var=D to F [STEP p]

Introduces a loop. All the instructions lying in lines between FOR var=D to F [STEP P] and the corresponding NEXT, will be repeated once for each value of var from D to F in steps of P (if P is not specified then in steps of 1).

```
10 FOR I=1 TO 20 STEP 2
20 PRINT I, " ",I*I
30 NEXT I
```

GOSUB GOSUB LN

Calls (executes) the subroutine starting at line number LN.

GOTO GOTO LN

Jumps to line number LN.

IF IF condition THEN instruction

Carries out the instruction which follows the THEN provided that the condition following IF evaluates to true.

```
IF A=3 THEN GOSUB 1000
```

INK

```
INK, colour [,colour]
```

A varying number of inks are available according to the screen mode currently in use. The INK command determines the INK colour and the background colour. If *two* background colours are specified, then they will alternate every 50th of a second.

INPUT

```
INPUT [#channel number][;][prompt string;]
                                list of variables
```

Reads data coming from the specified channel and assigns it to the named variables. The first [;] cancels the carriage return after the prompt. A ; after the string causes a ? prompt to appear, while , causes the ? prompt to be supressed. When a tape channel is specified, there is no screen prompt call, instead a data item from the relevant file (channel number) will be assigned to each variable of the list.

KEY

```
KEY integer number, string of characters
```

Allows definition of a new function key. The number (128–140) defines the key to which the string of characters will be assigned. Key 0 of the keyboard is designated as number 128, key 1 as 129, key 9 as 137, the space key as 138, the combination CTRL and ENTER together as 140.

```
KEY 132, "RUN"+CHR$(13)
```

places the sequence RUN followed by an ENTER onto number key 4.

KEY DEF

```
KEY DEF, Key number, repetition, num character
```

Changes the value produced by a key.
KEY DEF 45,1,65 puts A on the J key with an auto-repeat facility. KEY DEF 46,0,63 puts ? on the N key and disables the auto-repeat.

LET

```
LET variable=expressions
```

Assigns the result of the expression on the right of the equals sign to the variable on the left.

```
LET A = 500*3
```

In AMSTRAD BASIC it is possible to write A=500*3. LET is only used to maintain compatibility with earlier programs.

LINE INPUT

```
LINE INPUT [#channel number,] [;] [string;] variable
```

or

```
LINE INPUT "NAME";A$
```

Reads in an entire line from the specified channel (defaults to channel 0). If a comma is found in the input it *will* be put into the variable, whereas the use of a simple INPUT command would have split the variable at this point.

LIST

```
LIST [line numbers] [#channel number]
```

Lists the program on the desired channel (0 corresponds to the screen and 8 to the printer). Screen scrolling can be stopped by pressing ESC and resumed by pressing any other key. Pressing ESC twice returns you to the command input (direct) mode.

LOAD

LOAD [name of file] [,address]

Loads a BASIC program from cassette into central memory, replacing anything that was there before. In the case of a binary program the loading address can be specified.

LOCATE

LOCATE [#No of channel,] X,Y

Places the text cursor at co-ordinate position (X,Y) relative to the origin of the screen window. The co-ordinate point (1,1) is at the top left hand corner of the window.

MEMORY

MEMORY address

Allows you to redefine the address of the highest memory address used by BASIC. This is normally address AB7F.

MERGE

MERGE ["filename"]

Identical to LOAD, but without the implied NEW command before loading. Where two line numbers are identical, the line contents become that of the new (LOADing) program. If the name of the file is not specified, then the first program encountered on the tape will be used. A tape program whose name is preceded by the sign ! is protected and will not be read.

MODE

MODE N

Allows changing of the screen mode (N=0, 1 or 2). Clears the screen and sets INK0 regardless of the PAPER INK value in use at the time. When this command is used the full screen is displayed and the cursors return to their points of origin.

MOVE

MOVE X,Y

Positions the graphic cursor at the absolute position of co-ordinates (X,Y).

MOVER

MOVER X,Y

Moves the graphic cursor to co-ordinate position (X,Y) relative to the current position.

NEW

NEW

Clears the memory. The current program and all variables disappear but key definitions and display modes remain unchanged.

NEXT

FOR I=1 TO 10 : ... : NEXT [I]

Determines the end point of a loop started by FOR.

ON...GOTO
ON...GOSUB

ON n GOTO list of line numbers
ON n GOSUB list of line numbers

Branches to the routine or subroutine at the nth position in the list of line numbers.

```
ON A GOTO 100,110,130,132,170,300,320,1000
```

if A = 1, a jump will be made to line 100
if A = 2, a jump will be made to line 110
if A = 7, a jump will be made to line 320
and so on.

ON BREAK GOSUB ON BREAK GOSUB line number

Calls a subroutine to be executed whenever a break (ESC ESC) is detected during the course of program execution.

ON BREAK STOP

Cancels the effect of the command ON BREAK GOSUB.

ON ERROR GOTO ON ERROR GOTO line number

Branches execution to the specified line when an error occurs.

ON SQ GOSUB ON SQ(n) GOSUB LN

Executes the sub-routine at line LN when the queue corresponding to sound channel n is no longer full. n can only have the values 1, 2 or 4, corresponding to channels A, B and C respectively.

OPENIN OPENIN "filename"

Opens a tape file, thus allowing the program running in central memory to read data directly from the cassette. If the name of the file is preceded by a !, the normal messages associated with use of the tape will not appear and the program will read in the first tape file block directly.

OPENOUT OPENOUT "filename"

Opens a tape file so that a program can write data to it. If the name of the file is preceded with !, the usual tape start-up messages will not appear. The program then creates its first 2K data buffer, but nothing is written onto the tape until the buffer is full or until the command CLOSEOUT is used to close the file.

ORIGIN ORIGIN X,Y,[,L,R,T,B]

Determines the co-ordinates (X,Y) for the point of origin of the graphic cursor. The optional elements L, R, T and B allow you to define a new window.

OUT OUT port number, integer

Sends an integer value to the specified port. The integer can take any value from 0 to 255 and the port number takes any value between 0 and 65535.

PAPER PAPER [#no of channel,] ink no

Defines the background colour for the next characters to be written to the screen.

PEN PEN [#no of channel,] ink no

Defines the colour of the next characters to be written to the screen.

PLOT PLOT X,Y [ink no]

Places the cursor at co-ordinate position (X,Y) in the colour specified by INK, defaults to the colour last used.

PLOTR

PLOTR X,Y [,ink no]

Places the cursor at co-ordinate position (X,Y) relative to the current position, in the colour specified by INK. Defaults to the colour last used.

POKE

POKE adr, data

Places the data at the specified address.

PRINT

PRINT [#channel No,] data

Prints the data on the selected channel (defaults to channel 0, the screen). PRINT USING allows you to specify different print formats.

RAD

RAD

Sets the trigonometric calculation mode to work in radians (*see* DEG).

RANDOMIZE

RANDOMIZE [N]

Sets a new sequence of pseudo-random numbers starting from N, N being an integer between 0 and 65535. By default, N is equal to 0.

READ

READ list of variables

Reads the data contained in DATA program lines and assigns it to the specified variables (*see* DATA and RESTORE).

RELEASE

RELEASE sound channels

Releases a sound channel from the waiting state.

REM

REM

Introduces a line of comments which will be ignored by the BASIC interpreter.

RENUM

RENUM [NN], [SN,] [ST]

Renumbers the current program lines. New line numbers start with NN (default 10); SN is the old line number from which renumbering is to start (default is the first line of the program), ST is the step between lines (default 10).

RESTORE

RESTORE (line number)

Defines the line number of the next DATA statements to be used by a READ. If no number is specified, READ begins at the first program line containing a DATA statement.

RESUME

RESUME [line number]

Allows program execution to continue from the given line number after an error has been trapped and corrected by use of ON ERROR GOTO.

RETURN

RETURN

Returns to the main program after completing execution of a subroutine called by GOSUB.

RUN

```
RUN [line number]
```

Runs the currently loaded program starting from the specified line number or, by default, from the lowest numbered line.

```
RUN "name of program"
```

Loads a specified program from tape and RUNs it. If you do not include the name of the program (as in RUN ""), BASIC loads and runs the first program encountered on the tape.

SOUND

```
SOUND channel, tone period [,duration[,volume[,
      volume envelope[,tone envelope]]]]
```

Produces a specified sound. For a more detailed explanation of how this is done, see the section on *Chips – AY3 8912.*

Channels: The three channels A,B and C can be selected together, in pairs or individually.

Tone period: This determines the pitch of the tone and can take any value between 0 and 4095. The frequency of the sound is obtained by dividing 125000 by the selected time value.

Duration: Can take values between −32768 and +32767 (default 20). If the duration has a positive value, it represents so many 100ths of a second; if it has a negative value, it represents the number of repetitions to be made of the complete volume envelope.

Volume: Takes a value between 0 and 15 (default 12 if the command ENT has been given, otherwise defaults to 4).

Volume envelope: Can take any value between 0 and 15 (default 0) and indicates the type of the envelope defined by the instruction ENV.

Tone envelope: Can take any value between 0 and 15 (default 0) and indicates the type of the tone envelope defined by the command ENT.

SPEED INK

```
SPEED INK, integer, integer
```

Allows modification of the alternating rate of background colours where two colours have been defined with the INK command.

```
10 INK 0,1,9
20 SPEED INK 100,20
```

SPEED KEY

```
SPEED KEY wait, repetition time
```

Sets the delay time (wait) before a key-stroke will auto-repeat, together with the speed of the repetition (repetition time). These adjustments are made in 100ths of a second and both values default to 10.

SPEED WRITE

```
SPEED WRITE n
```

Changes the speed at which a program is recorded onto tape. When LOADing, the CPC 464 automatically establishes the correct speed for reading. With n = 0 the WRITE speed will be 1000 baud; with n = 1 it will be 2000 baud. n defaults to 0 and may only take one of the values 0 and 1.

STOP

STOP

Stops program execution while maintaining the option of continuing with the CONT command.

SYMBOL

SYMBOL, character number; list of characters

Allows redefinition of the character whose number is specified. All characters numbered between 240 and 255 can be re-defined; to redefine others, see the command SYMBOL AFTER.

SYMBOL AFTER

SYMBOL AFTER integer

Sets the number of characters that may be redefined using SYMBOL. Normally set to 240.

TAG

TAG [#channel number]

Allows characters to be placed at the position of the graphic cursor. Written text can thus be mixed with graphics.

```
10 MOVE 200, 300
20 PRINT "YOOHOO"
30 TAG
40 PRINT "HELLO"
```

YOOHOO will be written at the position of the text cursor while HELLO will be written at the position of the graphic cursor (200,300).

TAGOFF

TAGOFF [channel number]

Cancels the effect of the command TAG on the appropriate channel (defaults to channel 0) and returns the text to where the text cursor was before the use of the command TAG.

TRON

TRON

Turns on TRACE mode.
During the execution of a program in TRACE mode, all the line numbers executed are displayed in order on the screen. This mode is really useful during the writing and debugging of a program.

TROFF

TROFF

Turns off TRACE mode.

WAIT

WAIT n PORT, mask byte, selection byte

Waits for a specified bit pattern to appear at the specified input port. This instruction reads the pattern at port n, ANDs the contents with the mask byte and then performs and an EXCLUSIVE OR function with the selection byte – program execution is resumed only when the result is non-zero. The mask function permits the isolation of one or more bits for testing. The selection function allows the test state to be inverted.

WEND

WEND

Ends execution of a loop begun by the WHILE command.

WHILE

WHILE logical expression

While the logical expression is true, the program will execute

the program lines between WHILE and WEND (in this case until X=Y).
The following program:

```
10 X=4: Y=0
20 WHILE X<>Y
30 INPUT "HOW MUCH IS 2 AND 2 "; Y
40 WEND
50 PRINT "BRAVO":END
```

will do exactly the same as the program:

```
10 X=4
20 INPUT "HOW MUCH IS 2 AND 2"; Y
30 IF X<>Y THEN GOTO 20
40 PRINT "BRAVO":END
```

The usefulness of the instructions WHILE and WEND only becomes particularly clear when using structured programming. One of the principles of this type of programming is a considerable reduction in the use of the jump instruction (GOTO) so as to make programs more readable.

WIDTH WIDTH integer

Sets the number of characters which may be printed on a single (logical) line.

WINDOW WINDOW [#channel no] left, right, top, bottom

Allows definition of a text window for a given channel of the screen, channels 0 to 7 can be used to define screen text windows.

WINDOW SWAP WINDOW SWAP, channel number, channel number

All attributes of the two channels are exchanged.

WRITE WRITE [#channel no,] [list to be written]

Writes the list to the specified channel (defaults to 0) without any changes in punctuation.

```
WRITE "YOOHOO", 23,5
```

will write "YOOHOO", 23, 5 on the screen.

XPOS XPOS

Returns the current horizontal position of the graphic cursor.

YPOS YPOS

Returns the current vertical position of the graphic cursor.

ZONE ZONE integer

With the PRINT command, a comma may be used to divide the printout into columns (defaulting to 13 character columns); ZONE allows this column width to be redefined.

```
10 ZONE 4:PRINT "*",1,2,3
```

prints

```
*   1   2   3
```

BASIC FUNCTIONS

ABS

```
ABS (numeric expression)
```

Returns the absolute value of the numeric expression shown in brackets.

ASC

```
ASC (String of characters)
```

Returns the ASCII code of the first character in the character string named in the brackets.

```
ASC("ABC")
```

returns 65.

ATN

```
ATN (numeric expression)
```

Returns the value in radians or in degrees (*see* DEG and RAD) of the angle whose tangent equals the numeric expression (arc-tangent).

BIN$

```
BIN$ (decimal integer [,N])
```

Converts an integer (in base 10) into a binary number (base 2) of length N characters (leading zeros ommited by default).

CHR$

```
CHR$ (N)
```

Returns the character with the ASCII code N. N is an integer between 0 and 255.

```
CHR$(65)
```

returns an A.

CINT

```
CINT (numeric expression)
```

Rounds a real number *up* to the next whole number if the fractional part of the expression (to the right of the decimal point) is higher than or equal to 0.5; otherwise rounding *down*. The new number will be stored as an integer.

```
PRINT CINT (1.4),CINT (1.6)
```

prints:

```
1             2
```

COS

```
COS (angle value)
```

Returns the value of the cosine of an angle in radians (by default) or in degrees if the DEG command has been used.

CREAL

```
(numeric expression)
```

Converts an integer number to real form. This is the inverse of the CINT function.

EOF

```
PRINT EOF
```

Indicates that the end of a tape file has been detected. Returns the value -1 when the cassette is at the end of a file, otherwise returns 0.

ERR

```
PRINT ERR
```

ERR is a variable containing the number of the last error message encountered.

ERL

PRINT ERL

ERL is a variable containing the number of the line which produced the most recent error message.

EXP

PRINT EXP (n)

Returns *e* to the power n.

FIX

FIX (n)

Like INT, this returns the decimal part of a number n, but this time it *truncates* (removes the decimal point) rather than *rounds*.

FRE

FRE (x) *or* FRE (" ")

Returns the number of bytes remaining free in memory. The argument (in brackets) is a dummy whose actual value is not used by the routine and so may take any (legal) form.

HEX$

HEX$ (n)

Converts an integer number to a hexadecimal number of N characters (leading zeros ommited by default).

HIMEM

HIMEM

Returns the highest address available for use by BASIC.

INKEY

INKEY (N)

Examines the keyboard to see what key has been pressed.
If key number N has been pressed, INKEY (N) returns 0.
If key number N has been pressed as the same time as the SHIFT key, INKEY (N) returns 32.
If no key or any key other than key (N) has been pressed, then INKEY (N) returns −1.

```
5  CLS
10 IF INKEY(54)=32 THEN 30
   ELSE IF INKEY (54)=0 THEN 40
20 GOTO 10
30 PRINT "You have typed SHIFT and B": GOTO 50
40 PRINT "You have typed B"
50 GOTO 10
```

INKEY$

A$=INKEY$

Loads the string variable A$ with the value of the key that has just been pressed on the keyboard. This function is particularly useful when waiting for a single key input.

```
10 CLS
20 PRINT "Do you take sugar in your coffee?";
30 A$=INKEY$: IF A=" THEN 30
40 IF A$ <> "Y" AND A$ <> "N" THEN 30
50 PRINT A$
```

INP

PRINT INP (Number of I/O port)

Reads the contents of a specified Input/Output port.

INSTR

INSTR ([N,] A$, B$)

If the string B$ is a part of the string A$, INSTR (A$, B$) takes a numeric value equal to the start position of B$ within

A$. If N is specified, the count will begin from the Nth character in the string A$.

```
PRINT INSTR ("BANANA", "AN")
```

prints 2.

INT

INT (numeric expression)

Ignores the fractional part of a number and rounds it *down* to the nearest whole number. Similar to FIX for positive numbers, but will give 1 less than FIX for negative numbers which are not integers.

JOY

JOY (N)

Reads the value at joystick number N (0 or 1). The value returned is expressed in the 6 least significant bits of the returned number. If the joystick is not being used, all 6 bits equal 0. Bits change to 1 with a change in position of the joystick or by pressing the trigger as follows:

bit 0 = 1	joystick UP (adds 1 to returned value)
bit 1 = 1	joystick DOWN (adds 2 to returned value)
bit 2 = 1	joystick LEFT (adds 4 to returned value)
bit 3 = 1	joystick RIGHT (adds 8 to returned value)
bit 4 = 1	trigger 1 fired (adds 16 to returned value)
bit 5 = 1	trigger 2 fired (adds 32 to returned value)

It is possible to deduce combinations. For example, if the joystick is in a downwards right position and trigger number 1 is being pressed, the joy function will return a value equal to the sum of all values that would be returned for each separate action:

Down = 2
Right = 8
Trigger 1 fired = 16
Returned value: 2 + 8 + 16 = 26.

LEFT$

LEFT$ (string, N)

Returns the N characters at the left of the specified string, N being an integer.

```
PRINT LEFT$ ("AMSTRAD",4)
```

will print:
AMST.

LEN

LEN (string)

Returns the number of characters in the string.

LOG

LOG (X)

Calculates the logarithm of X to base *e*.

LOG10

LOG10 (X)

Calculates the logarithm of X to base 10.

LOWER$

LOWER$ (string)

Transforms all capital letters in an alphanumeric string to lower case letters.

MAX

MAX (list of numeric expressions)

Returns the highest value found in a list of numbers or numeric expressions.

```
PRINT MAX (2, 67, 34, 987, 12, 9, 876, 0)
```

prints 987.

MID$

MID$ (string, N [,M])

Extracts M characters from the string, beginning at the Nth character. M defaults to 1.

MIN

MIN (list of numeric expressions)

Returns the smallest value contained in the list of numeric expressions.

PEEK

PEEK (N)

Returns the value contained at memory address N.

PI

PRINT PI

Returns the numeric value of PI.

```
PRINT PI
```

prints
3.14159265

POS

POS (#Channel number)

Indicates the current horizontal position of the text cursor for a given channel (the X co-ordinate). If the printer is specified, POS gives the horizontal position of the print-head, position 1 being at the left hand margin.

REMAIN (N)

Disables the specified clock (N = 0, 1, 2 or 3) and reads the time remaining. Returns 0 if the clock has not been set.

RIGHT$

RIGHT$ (string, N)

Returns N characters from the right-hand end of the specified string.

RND

RND (N)

Returns a psuedo-random number from the sequence determined by the RANDOMIZE command. If N is negative, then each N will give a repeatable pseudo-random value, until another RANDOMIZE command.

ROUND

ROUND (numeric expression [,N])

Rounds up the numeric expression to N decimal places. N, an integer, defaults to 0.

SGN

SGN (numeric expression)

Determines the sign of the numeric expression. Returns −1 if negative, 0 if 0, and 1 if positive.

SIN

SIN (angle value)

Returns the value of the sine of the angle in either radians or degrees (*see* RAD and DEG). Defaults to radians.

SPACE$

SPACE$ (N)

Creates a string of N spaces, N being an integer up to 255.

SQ

SQ (sound channel)

Returns the number of free places in the queue of a given channel.

SQR

SQR (N)

Calculates the square root of the number N.

STR$

STR$ ([&]n)

Converts the numeric expression N into a string of characters. If the numeric expression is preceeded by an ampersand (&), it is assumed to be a hexadecimal number and will first be converted into decimal before being converted into a string.

```
PRINT STR$ (&10)
```

would return the string 16.

STRING$

STRING$ (N, character)

Creates a string of characters made up of the specified character repeated N times. N can be expressed in hexadecimal if it is preceeded by a &.

```
PRINT STRING$ (4,"*")
```

and

```
PRINT STRING$ (4,42)
```

both print
****

TAN

TAN (angle)

Returns the value of the tangent of the angle, expressed in radians (default) or degrees (*see* RAD and DEG).

TEST

TEST (x,y)

Returns the value of INK used at the absolute co-ordinate position (x,y) on the screen.

TESTR

TESTR (x,y)

Returns the value of INK used at the co-ordinate position (x,y) on the screen, relative to the graphics cursor.

TIME

PRINT TIME

Returns the amount of time spent so far (in units of 1/300th of a second) since start-up. Tape read and write time is not included.

UNT

UNT (number)

Converts an unsigned integer to a signed integer between −32767 and +32768.

PRINT UNT (&7FFF) and PRINT UNT (32767) print 32767
PRINT UNT (&0010) and PRINT UNT (16) print 16
PRINT UNT (&0001) and PRINT UNT (1) print 1
PRINT UNT (&FFFF) and PRINT UNT (65535) print -1

```
PRINT UNT (&FFF6) and PRINT UNT (65526) print -10
PRINT UNT (&8000) and PRINT UNT (32768) print -
32768
```

UPPER$

```
UPPER$ (string)
```

Transforms the lower case letters of string to capitals.

VAL

```
VAL (string)
```

Transforms a string into a numeric expression. Will return 0 if the string starts with a letter.

```
PRINT VAL ("34E"), VAL ("123"), VAL ("A34)
```

prints

```
34          123          0
```

VPOS

```
VPOS (#channel number)
```

Returns the vertical position (the Y co-ordinate)of the text cursor for the specified channel.

XPOS

```
XPOS
```

Returns the horizontal position of the graphic cursor.

YPOS

```
YPOS
```

Returns the vertical position of the graphic cursor.

KEYWORDS AND ASSOCIATED CODES

All codes below 127 are preceded by a byte containing the value 255.

Decimal code	Hex code	Keyword	Decimal code	Hex code	Keyword
255+0	FF+0	ABS	255+27	FF+1B	UNT
255+1	FF+1	ASC	255+28	FF+1C	UPPER$
255+2	FF+2	ATN	255+29	FF+1D	VAL
255+3	FF+3	CHR$	255+64	FF+40	EOF
255+4	FF+4	CINT	255+65	FF+41	ERR
255+5	FF+5	COS	255+66	FF+42	HIMEM
255+6	FF+6	CREAL	255+67	FF+43	INKEY$
255+7	FF+7	EXP	255+68	FF+44	PI
255+8	FF+8	FIX	255+69	FF+45	RND
255+9	FF+9	FRE	255+70	FF+46	TIE
255+10	FF+A	INKEY	255+71	FF+47	XPOS
255+11	FF+B	INP	255+72	FF+48	YPOS
255+12	FF+C	INT	255+113	FF+71	BIN$
255+13	FF+D	JOY	255+114	FF+72	DEC$
255+14	FF+E	LEN	255+115	FF+73	HEX$
255+15	FF+F	LOG	255+116	FF+74	INSTR
255+16	FF+10	LOG10	255+117	FF+75	LEFT$
255+17	FF+11	LOWER$	255+118	FF+76	MAX
255+18	FF+12	PEEK	255+119	FF+77	MIN
255+19	FF+13	REMAIN	255+120	FF+78	POS
255+20	FF+14	SGN	255+121	FF+79	RIGHT$
255+21	FF+15	SIN	255+122	FF+7A	ROUND
255+22	FF+16	SPACES$	255+123	FF+7B	STRING$
255+23	FF+17	SQ	255+124	FF+7C	TEST
255+24	FF+18	SQR	255+125	FF+7D	TESTR
255+25	FF+19	STR$	255+127	FF+7F	VPOS
255+26	FF+1A	TAN			

The following codes are not preceded by 255.

Decimal code	Hex code	Keyword	Decimal code	Hex code	Keyword
128	80	AFTER	139	8B	CONT
129	81	AUTO	140	8C	DATA
130	82	BORDER	141	8D	DEF
131	83	CALL	142	8E	DEFINT
132	84	CAT	143	8F	DEFREAL
133	85	CHAIN	144	90	DEFSTR
134	86	CLEAR	145	91	DEG
135	87	CLG	146	92	DELETE
136	88	CLOSEIN	147	93	DIM
137	89	CLOSEOUT	148	94	DRAW
138	8A	CLS	149	95	DRAWR

Decimal code	Hex code	Keyword	Decimal code	Hex code	Keyword
150	96	EDIT	196	C4	RELEASE
151	97	ELSE	197	C5	REM
152	98	END	198	C6	RENUM
153	99	ENT	199	C7	RESTORE
154	9A	ENV	200	C8	RESUME
155	9B	ERASE	201	C9	RETURN
156	9C	ERROR	202	CA	RUN
157	9D	EVERY	203	CB	SAVE
158	9E	FOR	204	CC	SOUND
159	9F	GOSUB	205	CD	SPEED
160	A0	GOTO	206	CE	STOP
161	A1	IF	207	CF	SYMBOL
162	A2	INK	208	D0	TAG
163	A3	INPUT	209	D1	TAGOFF
164	A4	KEY	210	D2	TROFF
165	A5	LET	211	D3	TRON
166	A6	LINE	212	D4	WAIT
167	A7	LIST	213	D5	WEND
168	A8	LOAD	214	D6	WHILE
169	A9	LOCATE	215	D7	WIDTH
170	AA	MEMORY	216	D8	WINDOW
171	AB	MERGE	217	D9	WRITE
172	AC	MID$	218	DA	ZONE
173	AD	MODE	219	DB	DI
174	AE	MOVE	220	DC	EI
175	AF	MOVER	234	EA	TAB
176	B0	NEXT	235	EB	THEN
177	B1	NEW	236	EC	TO
178	B2	ON	237	ED	USING
179	B3	ON BREAK	238	EE	>
180	B4	ON ERROR GOTO	239	EF	=
181	B5	ON SQ	240	F0	>=
182	B6	OPENIN	241	F1	<
183	B7	OPENOUT	242	F2	<>
184	B8	ORIGIN	243	F3	<=
185	B9	OUT	244	F4	+
186	BA	PAPER	245	F5	-
187	BB	PEN	246	F6	*
188	BC	PLOT	247	F7	/
189	BD	PLOTR	248	F8	
190	BE	POKE	249	F9	\
191	BF	PRINT	250	FA	AND
192	C0		251	FB	MOD
193	C1	RAD	252	FC	OR
194	C2	RANDOMIZE	253	FD	XOR
195	C3	READ	254	FE	NOT

ASCII CODES – CHARACTERS

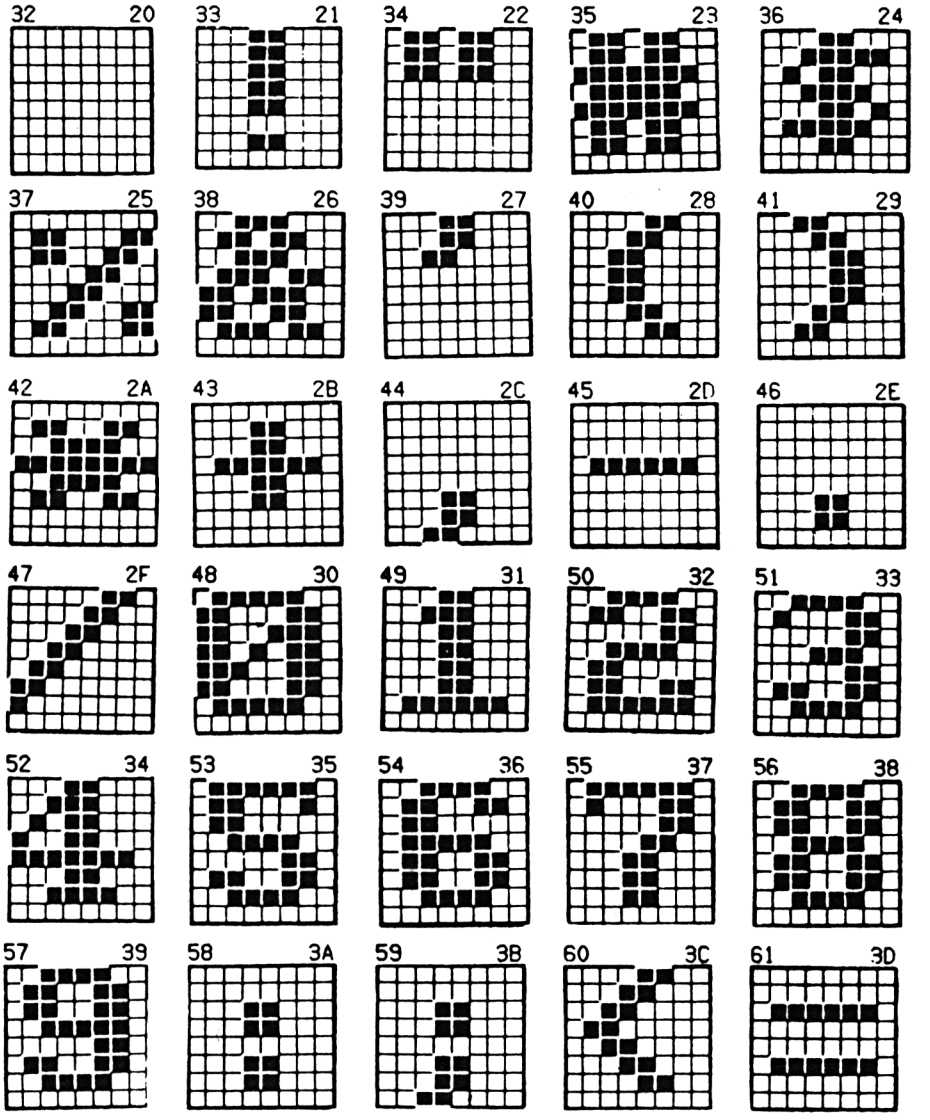
Character	ASCII codes		
	Hexadecimal	Decimal	Octal
NUL (CTRL @)	0	0	0
SOH (CTRL A)	1	1	1
STX (CTRL B)	2	2	2
ETX (CTRL C)	3	3	3
EOT (CTRL D)	4	4	4
ENQ (CTRL E)	5	5	5
ACK (CTRL F)	6	6	6
BEL (CTRL G)	7	7	7
BS (CTRL H)	8	8	10
HT (CTRL I)	9	9	11
LF (CTRL J)	A	10	12
VT (CTRL K)	B	11	13
FF (CTRL L)	C	12	14
CR (CTRL M)	D	13	15
SO (CTRL N)	E	14	16
SI (CTRL O)	F	15	17
DLE (CTRL P)	10	16	20
DC1 (CTRL Q)	11	17	21
DC2 (CTRL R)	12	18	22
DC3 (CTRL S)	13	19	23
DC4 (CTRL T)	14	20	24
NAK (CTRL U)	15	21	25
SYN (CTRL V)	16	22	26
ETB (CTRL W)	17	23	27
CAN (CTRL X)	18	24	30
EM (CTRL Y)	19	25	31
SUB (CTRL Z)	1A	26	32
ESC	1B	27	33
FS	1C	28	34
GS	1D	29	35
RS	1E	30	36
US	1F	31	37
(space)	20	32	40
!	21	33	41
"	22	34	42
#	23	35	43
$	24	36	44
%	25	37	45
&	26	38	46
'	27	39	47
(	28	40	50
)	29	41	51
*	2A	42	52
+	2B	43	53
,	2C	44	54
- *(dash)*	2D	45	55
.	2E	46	56
/	2F	47	57

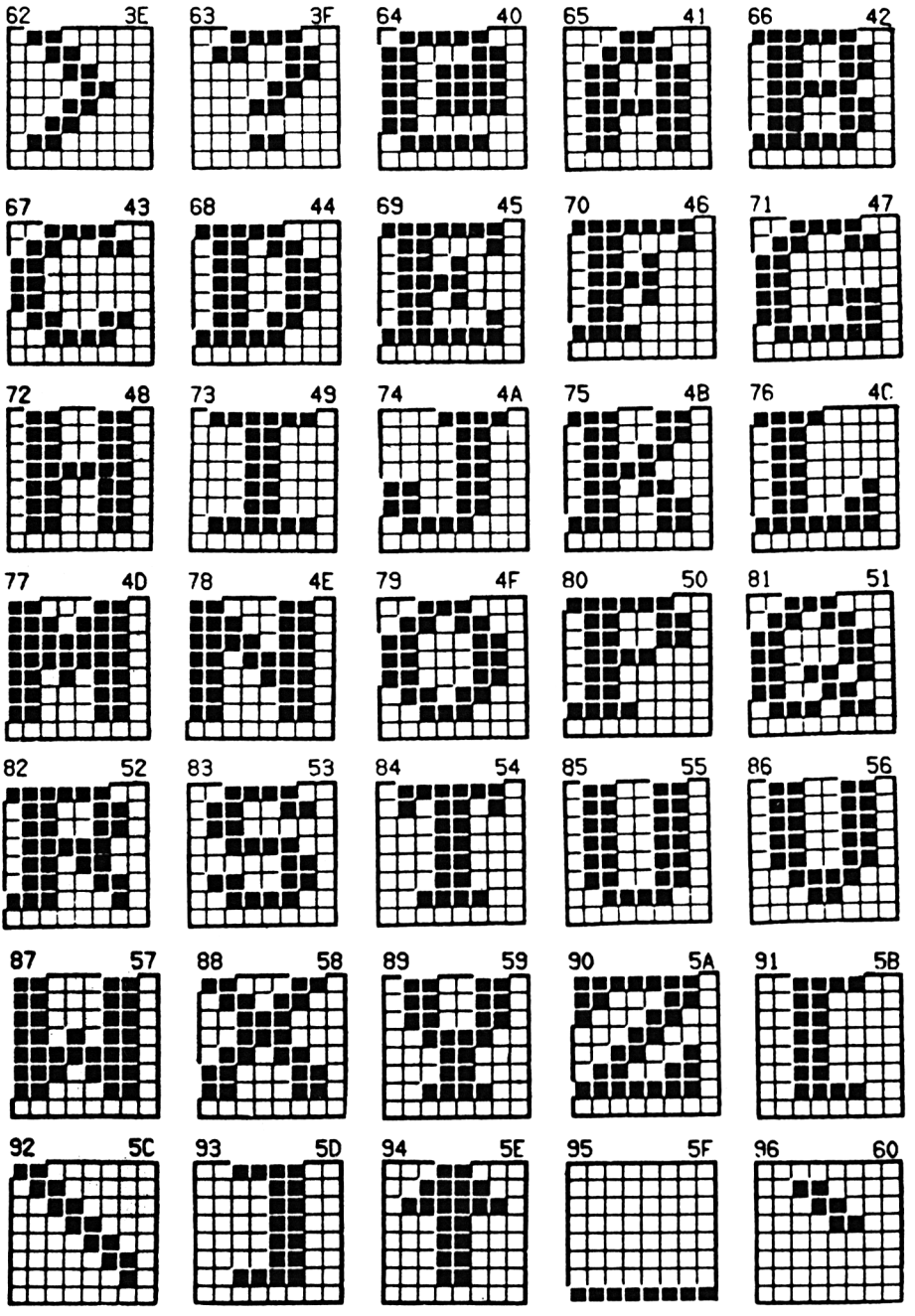
Character	ASCII codes		
	Hexadecimal	Decimal	Octal
0	30	48	60
1	31	49	61
2	32	50	62
3	33	51	63
4	34	52	64
5	35	53	65
6	36	54	66
7	37	55	67
8	38	56	70
9	39	57	71
:	3A	58	72
;	3B	59	73
<	3C	60	74
=	3D	61	75
>	3E	62	76
?	3F	63	77
@	40	64	100
A	41	65	101
B	42	66	102
C	43	67	103
D	44	68	104
E	45	69	105
F	46	70	106
G	47	71	107
H	48	72	110
I	49	73	111
J	4A	74	112
K	4B	75	113
L	4C	76	114
M	4D	77	115
N	4E	78	116
O	4F	79	117
P	50	80	120
Q	51	81	121
R	52	82	122
S	53	83	123
T	54	84	124
U	55	85	125
V	56	86	126
W	57	87	127
X	58	88	130
Y	59	89	131
Z	5A	90	132
[	5B	91	133
\	5C	92	134
]	5D	93	135
^	5E	94	136
_ *(underscore)*	5F	95	137
`	60	96	140
a	61	97	141
b	62	98	142
c	63	99	143
d	64	100	144

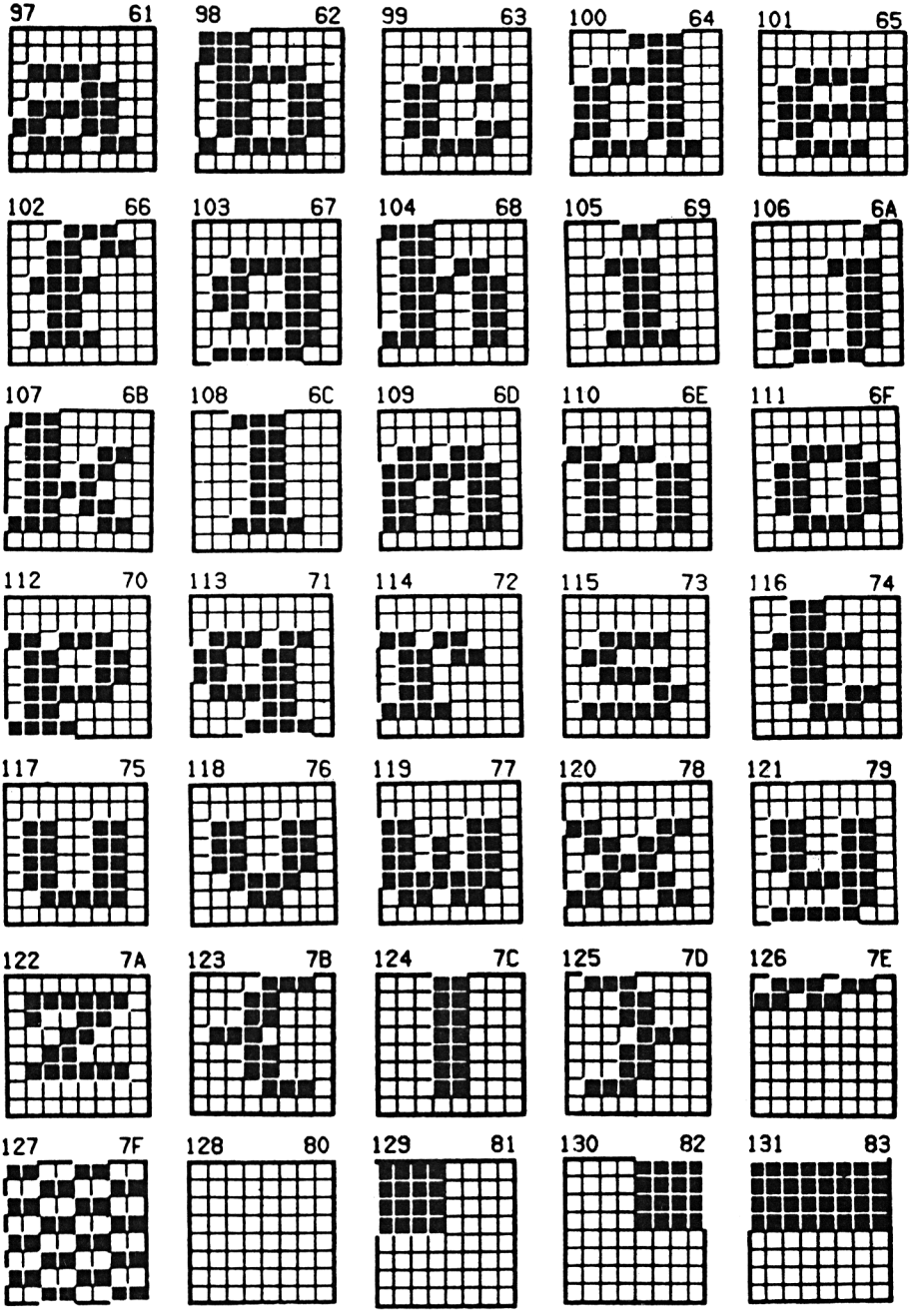
Character	ASCII codes		
	Hexadecimal	Decimal	Octal
e	65	101	145
f	66	102	146
g	67	103	147
h	68	104	150
i	69	105	151
j	6A	106	152
k	6B	107	153
l	6C	108	154
m	6D	109	155
n	6E	110	156
o	6F	111	157
p	70	112	160
q	71	113	161
r	72	114	162
s	73	115	163
t	74	116	164
u	75	117	165
v	76	118	166
w	77	119	167
x	78	120	170
y	79	121	171
z	7A	122	172
{	7B	123	173
|	7C	124	174
}	7D	125	175
~	7E	126	176
DEL	7F	127	177

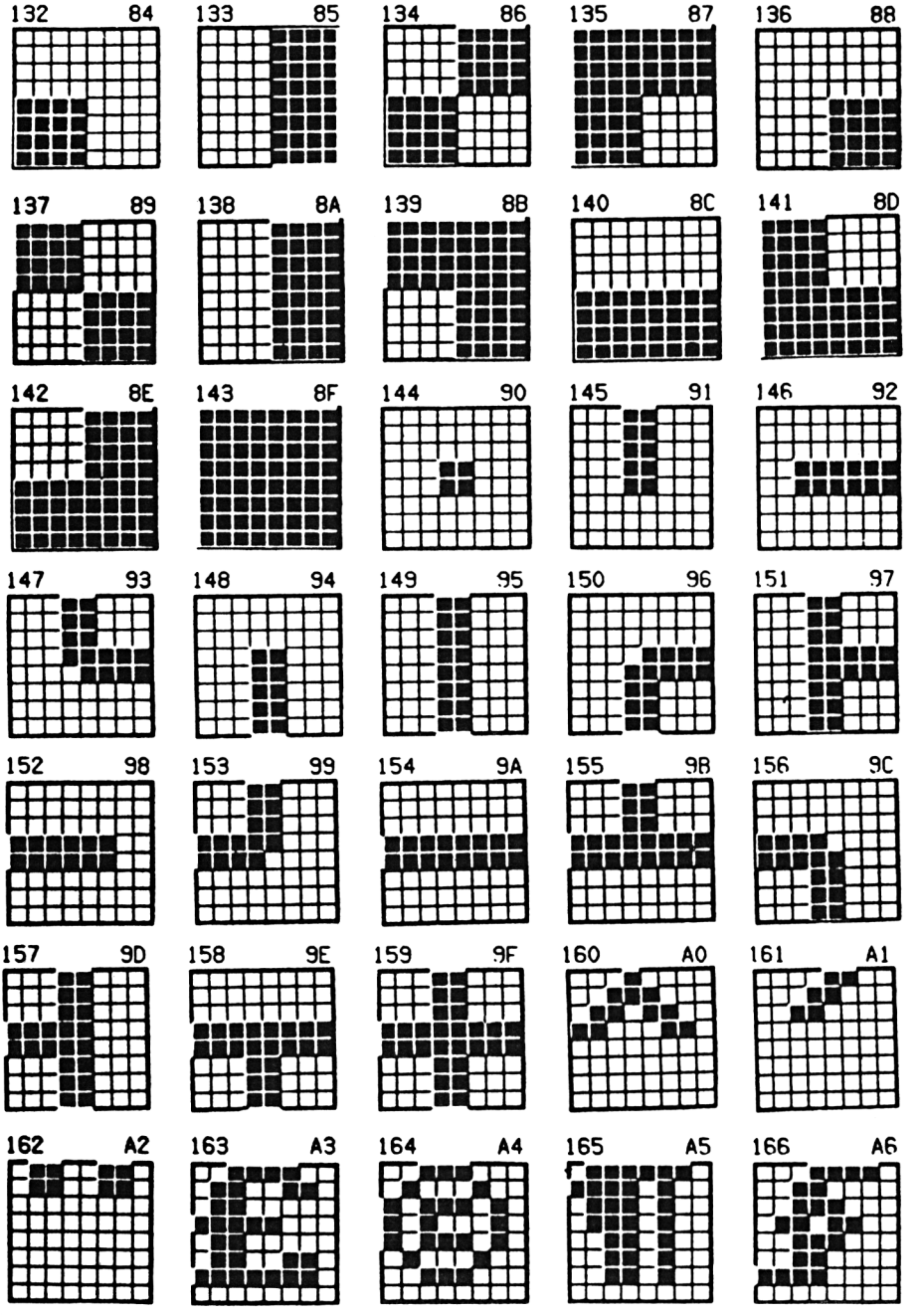
ASCII CODES – GRAPHICS

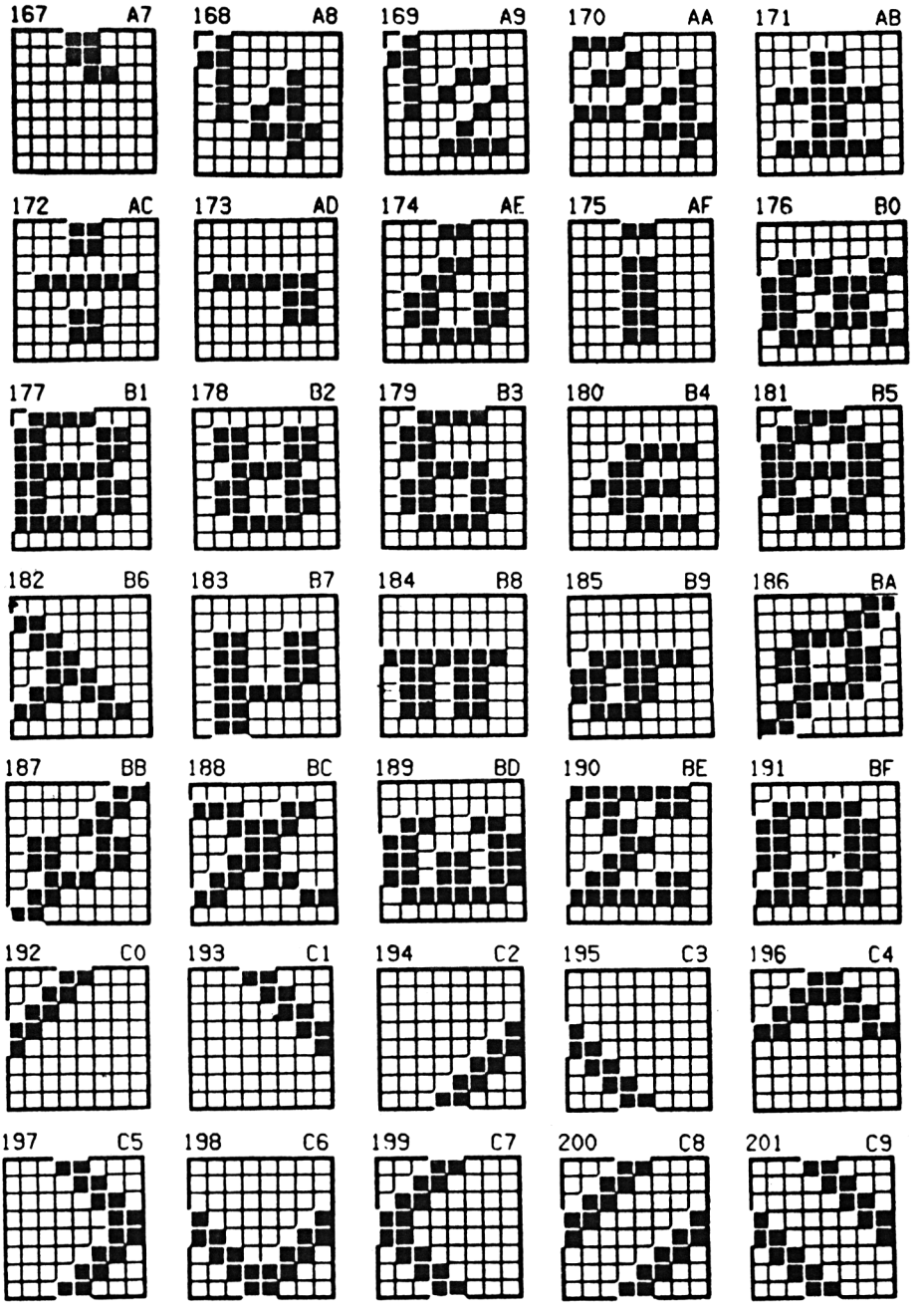
The character set is represented on the screen in an 8 by 8 matrix. Characters may be redefined using the SYMBOL command.

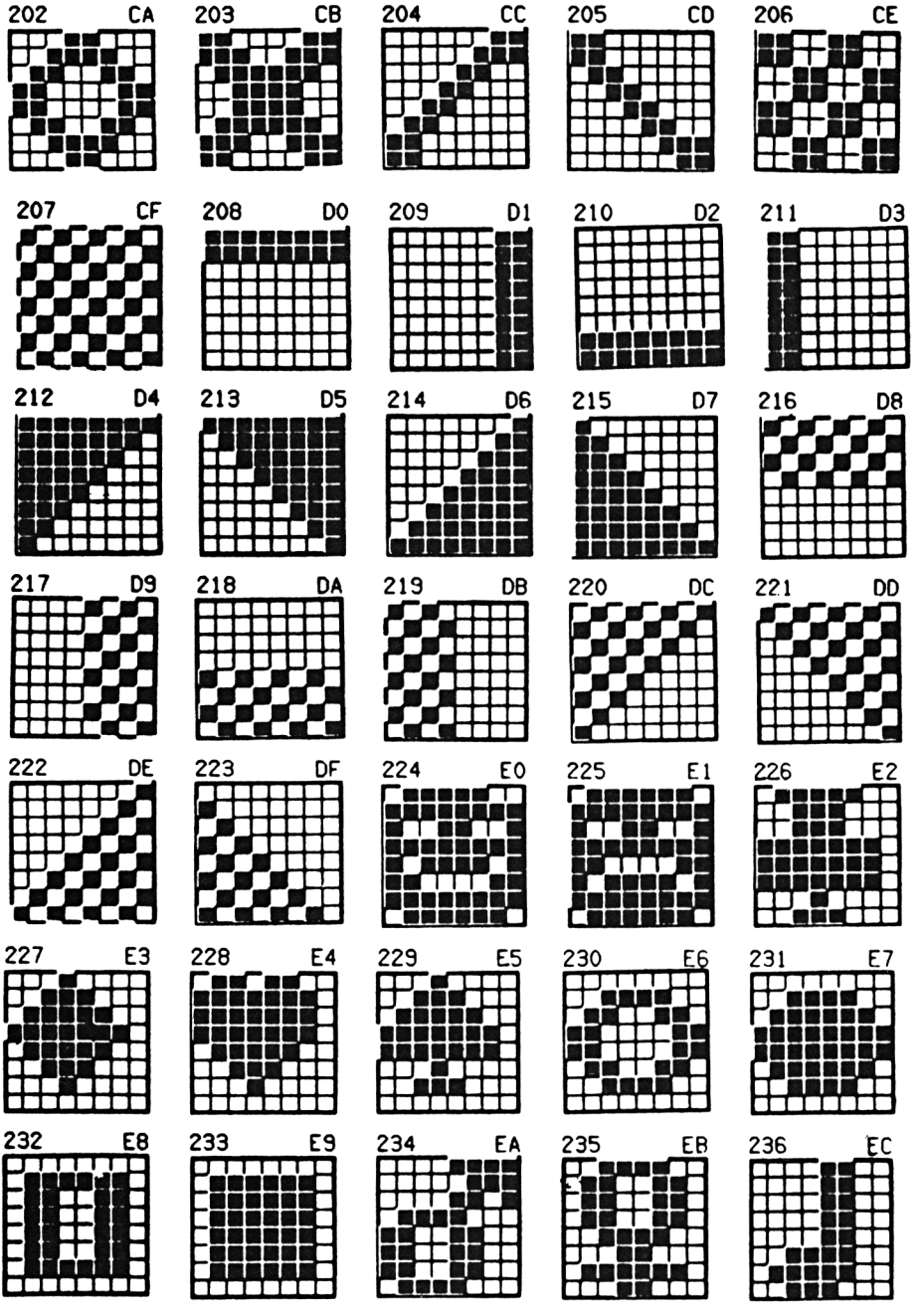


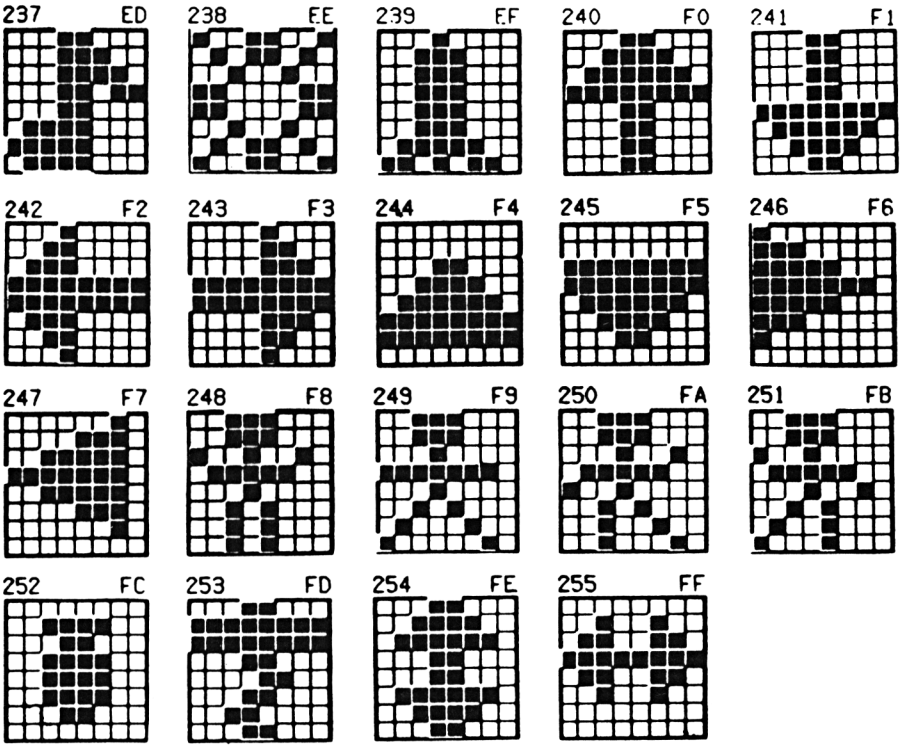

62 3E
63 3F
64 40
65 41
66 42
67 43
68 44
69 45
70 46
71 47
72 48
73 49
74 4A
75 4B
76 4C
77 4D
78 4E
79 4F
80 50
81 51
82 52
83 53
84 54
85 55
86 56
87 57
88 58
89 59
90 5A
91 5B
92 5C
93 5D
94 5E
95 5F
96 60


97 61
98 62
99 63
100 64
101 65
102 66
103 67
104 68
105 69
106 6A
107 6B
108 6C
109 6D
110 6E
111 6F
112 70
113 71
114 72
115 73
116 74
117 75
118 76
119 77
120 78
121 79
122 7A
123 7B
124 7C
125 7D
126 7E
127 7F
128 80
129 81
130 82
131 83


132 84
133 85
134 86
135 87
136 88
137 89
138 8A
139 8B
140 8C
141 8D
142 8E
143 8F
144 90
145 91
146 92
147 93
148 94
149 95
150 96
151 97
152 98
153 99
154 9A
155 9B
156 9C
157 9D
158 9E
159 9F
160 A0
161 A1
162 A2
163 A3
164 A4
165 A5
166 A6


167 A7
168 A8
169 A9
170 AA
171 AB
172 AC
173 AD
174 AE
175 AF
176 B0
177 B1
178 B2
179 B3
180 B4
181 B5
182 B6
183 B7
184 B8
185 B9
186 BA
187 BB
188 BC
189 BD
190 BE
191 BF
192 C0
193 C1
194 C2
195 C3
196 C4
197 C5
198 C6
199 C7
200 C8
201 C9


202 CA
203 CB
204 CC
205 CD
206 CE
207 CF
208 D0
209 D1
210 D2
211 D3
212 D4
213 D5
214 D6
215 D7
216 D8
217 D9
218 DA
219 DB
220 DC
221 DD
222 DE
223 DF
224 E0
225 E1
226 E2
227 E3
228 E4
229 E5
230 E6
231 E7
232 E8
233 E9
234 EA
235 EB
236 EC


237 ED
238 EE
239 EF
240 F0
241 F1
242 F2
243 F3
244 F4
245 F5
246 F6
247 F7
248 F8
249 F9
250 FA
251 FB
252 FC
253 FD
254 FE
255 FF

ERROR CODES AND ERROR MESSAGES

1. Unexpected NEXT
A NEXT command has been encountered without a corresponding FOR command having been executed.

2. Syntax error.
BASIC cannot understand the structure of a line or command.

3. Unexpected RETURN
A RETURN command has been encountered for which there is no matched GOSUB command.

4. Data exhausted
A READ command has tried to read more items of data than are available from a line (or series of lines) of DATA statements.

5. Improper argument
The parameters of a command, or the value of a function, have not been expressed correctly.

6. Overflow
A value introduced or calculated is too big or too small to be represented by the computer.

7. Memory full
All available memory space has been used or reserved. This can occur in case of DIMming oversized arrays, out of control FOR . . NEXT loops, or nested GOSUB calls.

8. Line does not exist
The line number referred to does not exist in memory.

9. Subscript out of range
An array index is outside the DIMensioned value of the array (either too big or too small).

10. Array already dimensioned
You have tried to redefine an array already defined by DIM.

11. Division by zero
Numbers cannot be divided by zero.

12. Invalid direct command
The command typed in is not acceptable in direct mode.

13. Type mismatch
An alphanumeric value has been assigned to a numeric variable or *vice versa*.

14. String space full
The space reserved for strings is full.

15. String too long
A string contains more than 255 characters.

16. String expression too complex
An string expression is too complex to be handled by the computer.

17. Cannot continue
Program execution cannot be resumed with the CONT command. This occurs if, after a break (ESC ESC), any program line has been modified.

18. Unknown user function
An FN function has been called without previously defining it with the command DEF FN.

19. RESUME missing
An ON ERROR GOTO error trapping routine has been encountered but it contains no RESUME statement.

20. Unexpected Resume
A RESUME statement has been encountered before an ON ERROR GOTO error trapping routine has been executed.

21. DIRECT command found
While loading a tape program BASIC has found data without a line number.

22. Operand missing
An expression without an operand has been encountered.

23. Line too long.
BASIC cannot accept lines longer than 255 characters.

24. EOF met
The program has reached the end of the file on the tape.

25. File type error
The file on the tape is not of the required type.

26. NEXT missing
A FOR statement has been found without a corresponding, matched NEXT statement.

27. File already open
You have tried to open a file which is already open.

28. Unknown command
The command is unknown.

29. WEND missing
The WEND corresponding to a WHILE command is missing from the program.

30. Unexpected WEND
A WEND has been encountered without a preceding WHILE.

31. Unknown error
This message is produced by all errors having an ERR value equal to or greater than 31.

BASIC AND MEMORY STORAGE

Your computer only understands binary code (ie. it deals with everything in bit patterns of 1s and 0s) – interpretation through the BASIC interpreter first of all involves translation of your programs into binary terms (assuming all has been written correctly so that it *can* be translated into this form).

Storage of BASIC keywords

The interpreter assigns a code called a *token* for each keyword encountered in BASIC. This system allows the saving of a considerable amount of space in central memory, since the one byte token takes up much less space than a complete word. (This, incidentally, is why BASIC keywords should never be used to define variables – the interpreter insists on replacing *any* keyword encountered, whether as a keyword or simply as part of a variable name, with the token for the corresponding instruction – with the exception of those included in text strings held between double quotes).

Storage of a BASIC line:

BASIC stores program lines starting from address 368. Let us see with the help of an example how it stores a line. Try writing this short program:

```
1990 PRINT "YOOHOO"
```

then type in the following instruction (command) line:

```
FOR I=368 TO 390: PRINT I; " "; PEEK(I):NEXT I
```

The following list of numbers will appear on the screen, the table details their meaning:

Address	Contents	Meaning
368	15	line length low byte
369	0	line length high byte
370	198	line number low byte
371	7	line number high byte
372	191	token for the keyword PRINT
373	32	ASCII code for SPACE
374	34	ASCII code for "
375	89	ASCII code for Y
376	79	ASCII code for O
377	79	ASCII code for O
378	72	ASCII code for H
379	79	ASCII code for O
380	79	ASCII code for O
381	34	ASCII code for "
382	0	code indicating the end of a BASIC line

The length of the line is expressed in two bytes, to turn it into a straightforward decimal number use the following formula:

low byte + (256 × high byte)

So the length of the above line equals = 15 + (0 × 256) = 15, in other words this line occupies 15 memory locations.

The line number is also expressed in two bytes and can be obtained with the same formula as used for the length. So it equals:

198 + (256 × 7) = 1990

To replace the PRINT with a REM, you can POKE the location of the PRINT token directly with the value of the REM token:

```
POKE 372, 197
```

Now list your program and you will see:

```
1990 REM "YOOHOO"
```

From now on you can modify your programs at will, or even get them to modify themselves by means of POKE lines within the program.... Have a good time!

Now let's look at how variables are stored. Write this little program:

```
10 ABC=20
```

Then, as before, type:

```
FOR I=368 TO 390: PRINT I; " "; PEEK(I):NEXT I
```

You will see the following list of numbers displayed:

Address	Contents	Meaning
368	14	line length low byte
369	0	line length high byte
370	10	line number low byte
371	0	line number high byte
372	13	indicates a numeric variable
373	7	length of the variable name + 4
374	0	separator
375	65	ASCII code of the variable name's first character
376	66	ASCII code of the variable name's second character
377	195	128 + ASCII code of the variable name's last character
378	239	token for = sign
379	25	variable size
380	20	variable value
381	0	separator

The value 13 at 372 is a code indicating that the variable is numeric in type. For a string variable the code would be 3.

Addresses 375 to 377 contain the codes for the variable name. All characters are coded in ASCII, except the last one which is represented as its ASCII value *plus* 128. At address 378, the value 239 represents the token for an = sign. The *token* for = is different from its ASCII code so that the computer knows that the = is not part of the variable name.

The value 25 at address 379 indicates the size of the variable. Here are the different values that can be found at this location:

Value	Variable length
15	variable value = 1, not coded
16	variable value = 2, not coded
23	variable value = 9, not coded
25	variable value between 10 and 255, coded in one byte
26	variable value between 255 and 65535, coded in two bytes
31	variable value above 65535 or non integer, coded in five bytes using the following formula:

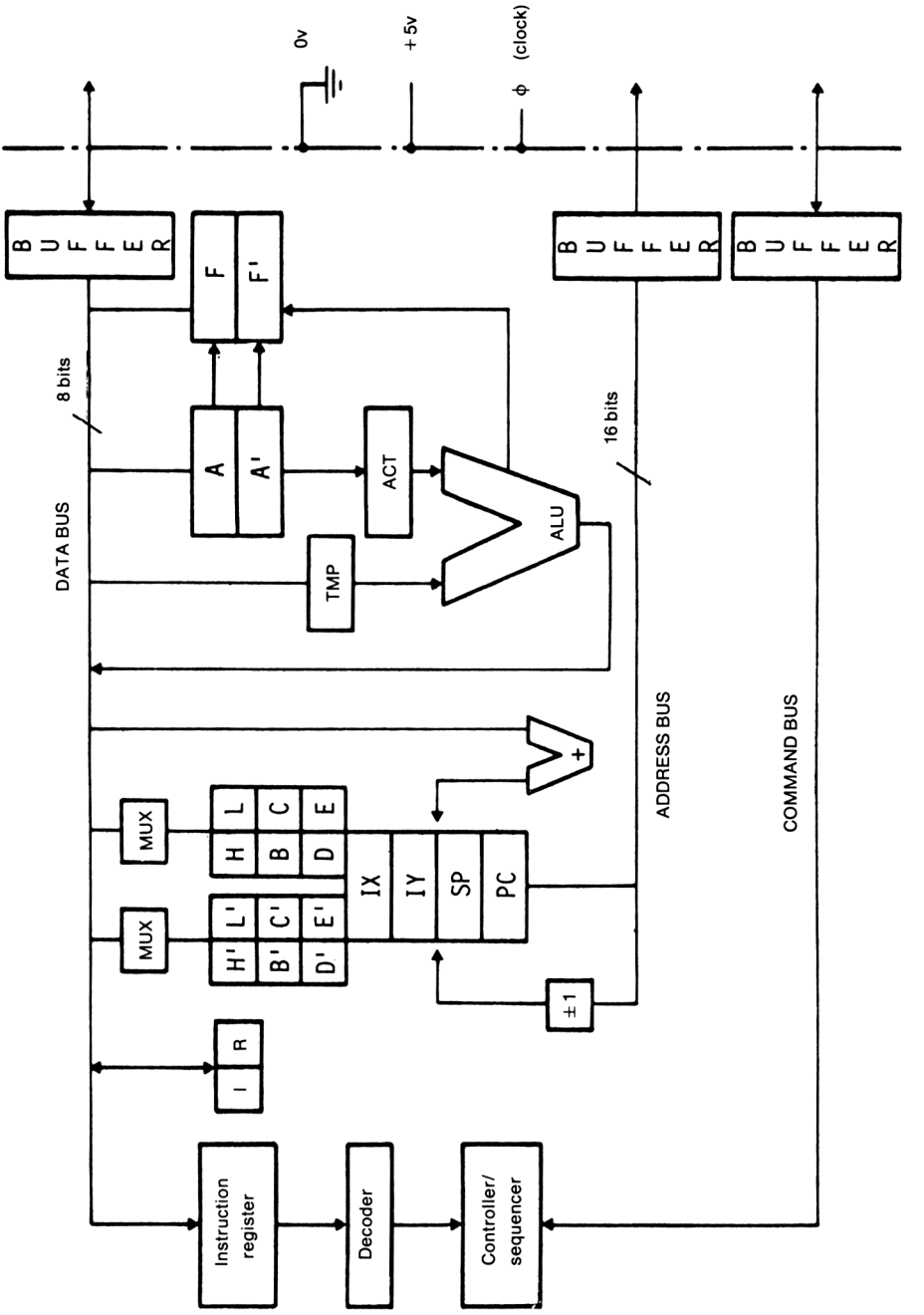
value = (2(b5 − 145)* (65536 + (b2/128) + (b3* 2) + (b4* 512) + (b1/32800)

where b1 to b5 represent the values held in the five addresses used to code the variable.

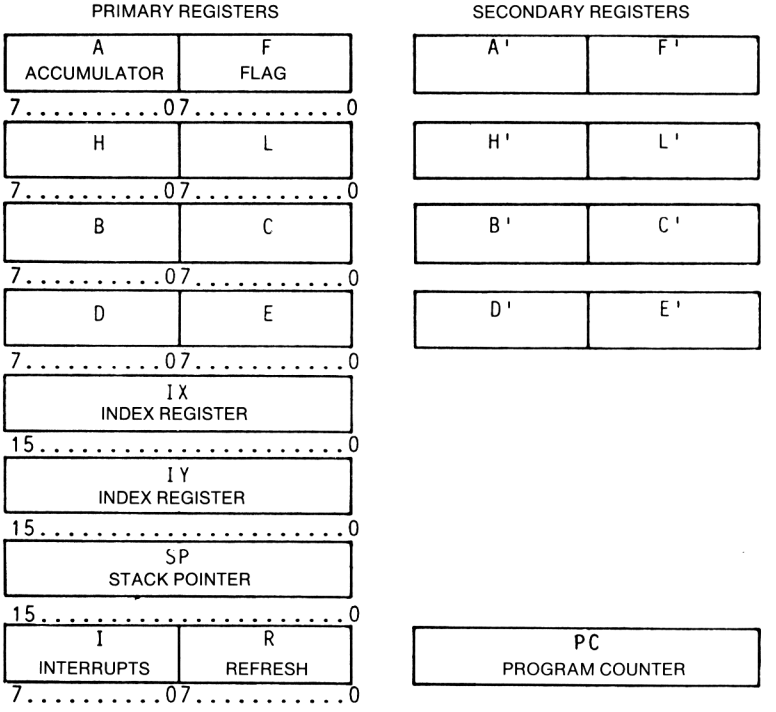
Where the variable is a negative number, the token for = (239) is followed by the token for the - (minus) sign, ie. 245.

MACHINE LANGUAGE

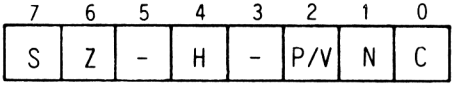
INTERNAL LAYOUT OF THE Z80



Z80 REGISTERS



Details of the flag register



S = sign: set to 1 if the most significant bit of the result of an operation is 1.

Z = zero: set to 1 if the result of an operation is zero.

H = half carry: identical to C (carry flag), but for four-bit (rather than eight-bit) operations.

P/V = Parity/overflow: set to 1 if there is an even number of bits set to 1 in the accumulator, or if there is an overflow after an operation using signed numbers.

C = carry: set to 1 if the result leads to a borrow (subtraction) or a carry (addition)

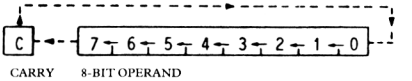
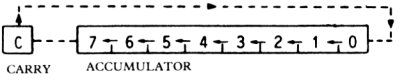
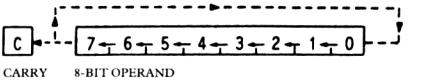
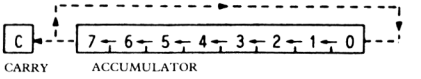
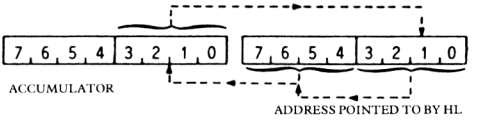
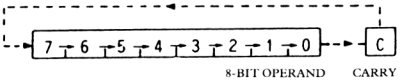
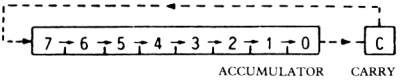
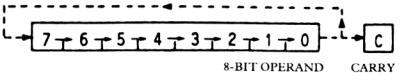
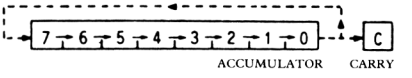
N = add/subtract: used to ensure that the DAA operation will be correct after either addition or subtraction.

Note:
Flags H and N cannot be tested.

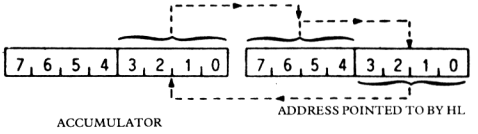
Z80 INSTRUCTION SET

Mnemonic	Operation carried out
ADC	Add with carry.
ADD	Add without carry.
BIT	Test a specified bit of a specified byte.
CALL	cc,mm Conditional call of a sub-routine.
CALL	Unconditional call of a sub-routine.
CCF	Complement the carry flag.
CP	Compare the operand with the accumulator.
CPD	Compare the accumulator with the contents of the address pointed to by HL and decrement HL and BC.
CPDR	Compare the accumulator with the contents of the address pointed to by HL. Decrements HL and BC. Repeats the sequence until BC = 0 or A=(HL).
CPI	Compare the accumulator with the contents of the address pointed to by HL. Increments HL and decrements BC.
CPIR	Compares the accumulator with the contents of the address pointed to by HL. Increments HL and decrements BC. Repeats the sequence until BC = 0 or A=(HL).
CPL	Complement accumulator.
DAA	Decimal adjustment of the accumulator.
DEC	Decrement a register, a register pair or the contents of an address pointed to by HL.
DI	Disable interrupts.
DJNZ	Decrement B and make a relative jump if B *is not* 0.
EI	Enable interrupts.
EX	Exchanges the contents of registers or address pointed to by Stack Pointer
EXX	Exchanges the contents of the registers BC, DE and HL with the registers BC', DE' and HL'.
HALT	Halts the CPU and places it in a waiting state for an interrupt or a reset.
IM	Set one of three interrupt modes (from 0 to 2).
IN	Load the accumulator or a register with the contents of an input/output port.
INC	Increment a register, a register pair or the contents of the address pointed to by HL.
IND	Load the address pointed to by HL with the contents of the input/output port pointed to by register C and decrement HL and B.
INDR	Loads the address pointed to by HL with the contents of the input/output port pointed to by C and decrements HL and B. Repeats the sequence until B is 0.

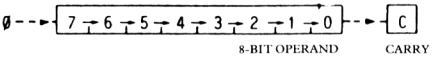
Mnemonic	Operation carried out
INI	Loads the address pointed to by HL with the contents of an input/output port defined in C, increments HL and decrements B.
INIR	Loads the address pointed to by HL with the contents of an input/output port defined in C, increments HL and decrements B. Repeats the sequence until B is 0.
JP	Unconditional jump to an address.
JP cc,aa	Conditional jump to address aa.
JR e	Unconditional jump relative to program counter plus offset e.
JR cc,e	Conditional jump relative to program counter plus offset e.
LD	Loads the accumulator, a register or an address with the contents of the accumulator, of a register or of an address.
LDD	Loads the address pointed to by HL with the contents of the address pointed to by DE, and then decrements DE, HL and BC.
LDDR	Loads the address pointed to by HL with the contents of the address pointed to by DE, and then decrements HL and BC. Repeats the sequence until BC = 0.
LDI	Loads the address pointed to by HL with the contents of the address pointed to by DE, and then increments DE and HL and decrements BC.
LDIR	Loads the address pointed to by HL with the contents of the address pointed to by DE, and then increments DE and HL and decrements BC. Repeats the sequence until BC = 0.
NEG	Negates the accumulator. The accumulator contents are subtracted from 0 using two's complement arithmetic.
NOP	No operation. The Z80 does not do anything.
OR	Perform a logical OR operation between operand and accumulator.
OTDR	Loads the input/output port pointed to by C with the contents of the location pointed to by HL, then decrements HL and B. Repeats the sequence until B = 0.
OTIR	Loads the input/output port pointed to by C with the contents of the location pointed to by HL, then increments B. Repeats the sequence until B = 0.
OUT	Loads the input/output port specified with the contents of the accumulator.
OUTD	Loads the input/output port pointed to by C with the contents of the location pointed to by HL, then decrements HL and B.
OUTI	Loads the input/output port pointed to by C with the contents of the location pointed to by HL, then increments HL and decrements B.
POP	Pops (removes) a register pair from the top of the stack (pointed to by SP).
PUSH	Places the contents of a register pair onto the top of the stack (pointed to by SP).
RES	Set a specified bit of the operand to zero.

Mnemonic	Operation carried out
RET	Return (at end of subroutine).
RETI	Return at end of an interrupt subroutine.
RETN	Return at end of a non-maskable interrupt subroutine.
RL	Rotation of the operand leftwards through the accumulator and carry flag. 
RLA	Rotation of the accumulator contents leftwards through the carry flag. 
RLC	Rotate register or operand left with branch carry. 
RLCA	Rotate accumulator left with branch carry. 
RLD	Rotate left decimal. Bits 0 to 3 of the accumulator are rotated to the left between the accumulator and the location pointed to by HL. 
RR	Rotate operand to the right through the carry flag. 
RRA	Rotate accumulator to the right through the carry flag. 
RRC	Rotate operand right with branch carry. 
RRCA	Rotate accumulator right with branch carry. 

Mnemonic	Operation carried out
RRD	Rotate right decimal. Bits 0 to 3 of the accumulator are rotated to the right between the accumulator and the location pointed to by HL.



RST	Restart at given address.
SBC	Subtraction with carry between either the accumulator and the operand or HL and a register pair.
SCF	Set the carry flag to 1.
SET	Set a specified bit to 1 either in a register, or at an address pointed to by HL or by IX and IY plus offset.
SLA	Arithmetic shift left. (add diagram)
SRA	Arithmetic shift right. (add diagram) *Note:* Bit 7 is unaffected
SRL	Logical shift to right of operand.



SUB	Subtract operand from accumulator.
XOR	Exclusive OR between the operand and the accumulator.

ALPHABETIC LIST OF Z80 INSTRUCTION CODES

d = 8-bit data
dd = 16-bit data
aa = 16-bit address
• = flag is modified
0 = flag set to 0
1 = flag set to 1

Object code	Instruction	S	Z	P/V	C
8E	ADC A,(HL)	•	•	•	•
DD8Ed	ADC A,(IX+d)	•	•	•	•
FD8Ed	ADC A,(IY+d)	•	•	•	•
8F	ADC A,A	•	•	•	•
88	ADC A,B	•	•	•	•
89	ADC A,C	•	•	•	•
8A	ADC A,D	•	•	•	•
8B	ADC A,E	•	•	•	•
8C	ADC A,H	•	•	•	•
8D	ADC A,L	•	•	•	•
CEd	ADC A,d	•	•	•	•
ED4A	ADC HL,BC	•	•	•	•
ED5A	ADC HL,DE	•	•	•	•
ED6A	ADC HL,HL	•	•	•	•
ED7A	ADC HL,SP	•	•	•	•
86	ADD A,(HL)	•	•	•	•
DD86d	ADD A,(IX+d)	•	•	•	•
FD86d	ADD A,(IY+d)	•	•	•	•
87	ADD A,A	•	•	•	•
80	ADD A,B	•	•	•	•
81	ADD A,C	•	•	•	•
82	ADD A,D	•	•	•	•
83	ADD A,E	•	•	•	•
84	ADD A,H	•	•	•	•
85	ADD A,L	•	•	•	•
C6d	ADD A,d	•	•	•	•
09	ADD HL,BC				•
19	ADD HL,DE				•
29	ADD HL,HL				•
39	ADD HL,SP				•
DD09	ADD IX,BC				•
DD19	ADD IX,DE				•
DD29	ADD IX,IX				•
DD39	ADD IX,SP				•
FD09	ADD IY,BC				•
FD19	ADD IY,DE				•
FD29	ADD IY,IY				•
FD39	ADD IY,SP				•
A6	AND (HL)	•	•	•	0
DDA6d	AND (IX+d)	•	•	•	0
FDA6d	AND (IY+d)	•	•	•	0
A7	AND A	•	•	•	0

Object code	Instruction	S	Z	P/V	C
A0	AND B	●	●	●	0
A1	AND C	●	●	●	0
A2	AND D	●	●	●	0
A3	AND E	●	●	●	0
A4	AND H	●	●	●	0
A5	AND L	●	●	●	0
E6d	AND d	●	●	●	0
CB46	BIT 0,(HL)	●	●	●	
DDCBd46	BIT 0,(IX+d)	●	●	●	
FDCBd46	BIT 0,(IY+d)	●	●	●	
CB47	BIT 0,A	●	●	●	
CB40	BIT 0,B	●	●	●	
CB41	BIT 0,C	●	●	●	
CB42	BIT 0,D	●	●	●	
CB43	BIT 0,E	●	●	●	
CB44	BIT 0,H	●	●	●	
CB45	BIT 0,L	●	●	●	
CB4E	BIT 1 (HL)	●	●	●	
DDCBd4E	BIT 1,(IX+d)	●	●	●	
FDCBd4E	BIT 1,(IY+d)	●	●	●	
CB4F	BIT 1,A	●	●	●	
CB48	BIT 1,B	●	●	●	
CB49	BIT 1,C	●	●	●	
CB4A	BIT 1,D	●	●	●	
CB4B	BIT 1,E	●	●	●	
CB4C	BIT 1,H	●	●	●	
CB4D	BIT 1,L	●	●	●	
CB56	BIT 2,(HL)	●	●	●	
DDCBd56	BIT 2,(IX+d)	●	●	●	
FDCBd56	BIT 2,(IY+d)	●	●	●	
CB57	BIT 2,A	●	●	●	
CB50	BIT 2,B	●	●	●	
CB51	BIT 2,C	●	●	●	
CB52	BIT 2,D	●	●	●	
CB53	BIT 2,E	●	●	●	
CB54	BIT 2,H	●	●	●	
CB55	BIT 2,L	●	●	●	
CB5E	BIT 3,(HL)	●	●	●	
DDCBd5E	BIT 3,(IX+d)	●	●	●	
FDCBd5E	BIT 3,(IY+d)	●	●	●	
CB5F	BIT 3,A	●	●	●	
CB58	BIT 3,B	●	●	●	
CB59	BIT 3,C	●	●	●	
CB5A	BIT 3,D	●	●	●	
CB5B	BIT 3,E	●	●	●	
CB5C	BIT 3,H	●	●	●	
CB5D	BIT 3,L	●	●	●	

Object code	Instruction	S	Z	P/V	C
CB66	BIT 4,(HL)	●	●	●	
DDCBd66	BIT 4,(IX+d)	●	●	●	
FDCBd66	BIT 4,(IY+d)	●	●	●	
CB67	BIT 4,A	●	●	●	
CB60	BIT 4,B	●	●	●	
CB61	BIT 4,C	●	●	●	
CB62	BIT 4,D	●	●	●	
CB63	BIT 4,E	●	●	●	
CB64	BIT 4,H	●	●	●	
CB65	BIT 4,L	●	●	●	
CB6E	BIT 5,(HL)	●	●	●	
DDCBd6E	BIT 5,(IX+d)	●	●	●	
FDCBd6E	BIT 5,(IY+d)	●	●	●	
CB6F	BIT 5,A	●	●	●	
CB68	BIT 5,B	●	●	●	
CB69	BIT 5,C	●	●	●	
CB6A	BIT 5,D	●	●	●	
CB6B	BIT 5,E	●	●	●	
CB6C	BIT 5,H	●	●	●	
CB6D	BIT 5,L	●	●	●	
DB76	BIT 6,(HL)	●	●	●	
DDCBd76	BIT 6,(IX+d)	●	●	●	
FDCBd76	BIT 6,(IY+d)	●	●	●	
CB77	BIT 6,A	●	●	●	
CB70	BIT 6,B	●	●	●	
CB71	BIT 6,C	●	●	●	
CB72	BIT 6,D	●	●	●	
CB73	BIT 6,E	●	●	●	
CB74	BIT 6,H	●	●	●	
CB75	BIT 6,L	●	●	●	
CB7E	BIT 7,(HL)	●	●	●	
DDCBd7E	BIT 7,(IX+d)	●	●	●	
FDCBd7E	BIT 7,(IY+d)	●	●	●	
CB7F	BIT 7,A	●	●	●	
CB78	BIT 7,B	●	●	●	
CB79	BIT 7,C	●	●	●	
CB7A	BIT 7,D	●	●	●	
CB7B	BIT 7,E	●	●	●	
CB7C	BIT 7,H	●	●	●	
CB7D	BIT 7,L	●	●	●	
DCaa	CALL C,aa				
FCaa	CALL M,aa				
D4aa	CALL NC,aa				
C4aa	CALL NZ,aa				
F4aa	CALL P,aa				
ECaa	CALL PE,aa				

Object code	Instruction	S	Z	P/V	C
E4aa	CALL PO,aa				
CCaa	CALL Z,aa				
CDaa	CALL aa				
3F	CCF				●
BE	CP (HL)	●	●	●	●
DDBEd	CP (IX+d)	●	●	●	●
FDBEd	CP (IY+d)	●	●	●	●
BF	CP A	●	●	●	●
B8	CP B	●	●	●	●
B9	CP C	●	●	●	●
BA	CP D	●	●	●	●
BB	CP E	●	●	●	●
BC	CP H	●	●	●	●
BD	CP L	●	●	●	●
FEd	CP d	●	●	●	●
EDA9	CPD	●	●	●	
EDB9	CPDR	●	●	●	
EDB1	CPIR	●	●	●	
EDA1	CPI	●	●	●	
2F	CPL				
27	DAA	●	●	●	
35	DEC (HL)	●	●	●	
DD35d	DEC (IX+d)	●	●	●	
FD35d	DEC (IY+d)	●	●	●	
3D	DEC A	●	●	●	
05	DEC B	●	●	●	
0B	DEC BC				
0D	DEC C	●	●	●	
15	DEC D	●	●	●	
1B	DEC DE				
1D	DEC E	●	●	●	
25	DEC H	●	●	●	
2B	DEC HL				
DD2B	DEC IX				
FD2B	DEC IY				
2D	DEC L	●	●	●	
3B	DEC SP				
F3	DI				
10d	DJNZ d				
FB	EI				
E3	EX (SP),HL				
DDE3	EX (SP),IX				
FDE3	EX (SP),IY				
08	EX AF,AF'				
EB	EX DE,HL				
D9	EXX				

Object code	Instruction	S	Z	P/V	C
76	HALT				
ED46	IM 0				
ED56	IM 1				
ED5E	IM 2				
ED78	IN A,(C)	●	●	●	
ED40	IN B,(C)	●	●	●	
ED48	IN C,(C)	●	●	●	
ED50	IN D,(C)	●	●	●	
ED58	IN E,(C)	●	●	●	
ED60	IN H,(C)	●	●	●	
ED68	IN L,(C)	●	●	●	
DBd	IN A,(d)				
34	INC (HL)	●	●	●	
DD34d	INC (IX+d)	●	●	●	
FD34d	INC (IY+d)	●	●	●	
3C	INC A	●	●	●	
04	INC B	●	●	●	
03	INC BC				
0C	INC C	●	●	●	
14	INC D	●	●	●	
13	INC DE				
1C	INC E	●	●	●	
24	INC H	●	●	●	
23	INC HL				
DD23	INC IX				
FD23	INC IY				
2C	INC L	●	●	●	
33	INC SP				
EDAA	IND	●	●	●	
EDBA	INDR	●	●	●	
EDA2	INI	●	●	●	
EDB2	INIR	●	●	●	
C3aa	JP aa				
E9	JP (HL)				
DDE9	JP (IX)				
FDE9	JP (IY)				
DAdd	JP C,aa				
FAaa	JP M,aa				
D2aa	JP NC,aa				
C2aa	JP NZ,aa				
F2aa	JP P,aa				
EAaa	JP PE,aa				
E2aa	JP PO,aa				
CAaa	JP Z,aa				
38d	JR C,d				
30d	JR NC,d				
20d	JR NZ,d				

Object code	Instruction		S	Z	P/V	C
28d	JR	Z,d				
18d	JR	d				
02	LD	(BC),A				
12	LD	(DE),A				
77	LD	(HL),A				
70	LD	(HL),B				
71	LD	(HL),C				
72	LD	(HL),D				
73	LD	(HL),E				
74	LD	(HL),H				
75	LD	(HL),L				
36d	LD	(HL),d				
DD77d	LD	(IX+d),A				
DD70d	LD	(IX+d),B				
DD71d	LD	(IX+d),C				
DD72d	LD	(IX+d),D				
DD73d	LD	(IX+d),E				
DD74d	LD	(IX+d),H				
DD75d	LD	(IX+d),L				
DD36d20	LD	(IX+d),d				
FD77d	LD	(IY+d),A				
FD70d	LD	(IY+d),B				
FD71d	LD	(IY+d),C				
FD72d	LD	(IY+d),D				
FD73d	LD	(IY+d),E				
FD74d	LD	(IY+d),H				
FD75d	LD	(IY+d),L				
FD36d20	LD	(IY+d),d				
32dd	LD	(dd),A				
ED43dd	LD	(dd),BC				
ED53dd	LD	(dd),DE				
22dd	LD	(dd),HL				
DD22dd	LD	(dd),IX				
FD22dd	LD	(dd),IY				
ED73dd	LD	(dd),SP				
0A	LD	A,(BC)				
1A	LD	A,(DE)				
7E	LD	A,(HL)				
DD7Ed	LD	A,(IX+d)				
FD7Ed	LD	A,(IY+d)				
3Add	LD	A,(dd)				
7F	LD	A,A				
78	LD	A,B				
79	LD	A,C				
7A	LD	A,D				
7B	LD	A,E				
7C	LD	A,H				

Object code	Instruction		S	Z	P/V	C
ED57	LD	A,I	•	•	•	
7D	LD	A,L				
3E	LD	A,d				
ED5F	LD	A,R	•	•	•	
46	LD	B,(HL)				
DD46d	LD	B,(IX+d)				
FD46d	LD	B,(IY+d)				
47	LD	B,A				
40	LD	B,B				
41	LD	B,C				
42	LD	B,D				
43	LD	B,E				
44	LD	B,H				
45	LD	B,L				
06d	LD	B,d				
ED4Bdd	LD	BC,(dd)				
01dd	LD	BC,dd				
4E	LD	C,(HL)				
DD4Ed	LD	C,(IX+d)				
FD4Ed	LD	C,(IY+d)				
4F	LD	C,A				
48	LD	C,B				
49	LD	C,C				
4A	LD	C,D				
4B	LD	C,E				
4C	LD	C,H				
4D	LD	C,L				
0Ed	LD	C,d				
56	LD	D,(HL)				
DD56d	LD	D,(IX+d)				
FD56d	LD	D,(IY+d)				
57	LD	D,A				
50	LD	D,B				
51	LD	D,C				
52	LD	D,D				
53	LD	D,E				
54	LD	D,H				
55	LD	D,L				
16d	LD	D,d				
ED5Bdd	LD	DE,(dd)				
11dd	LD	DE,dd				
5E	LD	E,(HL)				
DD5Ed	LD	E,(IX+d)				
FD5Ed	LD	E,(IY+d)				
5F	LD	E,A				
58	LD	E,B				
59	LD	E,C				

Object code	Instruction		S	Z	P/V	C
5A	LD	E,D				
5B	LD	E,E				
5C	LD	E,H				
5D	LD	E,L				
1E20	LD	E,n				
66	LD	H,(HL)				
DD66d	LD	H,(IX+d)				
FD66d	LD	H,(IY+d)				
67	LD	H,A				
60	LD	H,B				
61	LD	H,C				
62	LD	H,D				
63	LD	H,E				
64	LD	H,H				
65	LD	H,L				
26d	LD	H,d				
2Add	LD	HL,(dd)				
21dd	LD	HL,dd				
ED47	LD	I,A				
DD2Add	LD	IX,(dd)				
DD21dd	LD	IX,dd				
FD2Add	LD	IY,(dd)				
FD21dd	LD	IY,dd				
6E	LD	L,(HL)				
DD6Ed	LD	L,(IX+d)				
FD6Ed	LD	L,(IY+d)				
6F	LD	L,A				
68	LD	L,B				
69	LD	L,C				
6A	LD	L,D				
6B	LD	L,E				
6C	LD	L,H				
6D	LD	L,L				
2Ed	LD	L,d				
ED4F	LD	R,A				
ED7Bdd	LD	SP,(dd)				
F9	LD	SP,HL				
DDF9	LD	SP,IX				
FDF9	LD	SP,IY				
31dd	LD	SP,dd				
EDA8	LDD				●	
EDB8	LDDR				0	
EDA0	LDI				●	
EDB0	LDIR				0	
ED44	NEG					
00	NOP					
B6	OR	(HL)	●	●	●	0

Object code	Instruction		S	Z	P/V	C
DDB6d	OR	(IX+d)	●	●	●	0
FDB6d	OR	(IY+d)	●	●	●	0
B7	OR	A	●	●	●	0
B0	OR	B	●	●	●	0
B1	OR	C	●	●	●	0
B2	OR	D	●	●	●	0
B3	OR	E	●	●	●	0
B4	OR	H	●	●	●	0
B5	OR	L	●	●	●	0
F6d	OR	d	●	●	●	0
EDBB	OTDR		●	●	●	
EDB3	OTIR		●	●	●	
ED79	OUT	(C),A				
ED41	OUT	(C),B				
ED49	OUT	(C),C				
ED51	OUT	(C),D				
ED59	OUT	(C),E				
ED61	OUT	(C),H				
ED69	OUT	(C),L				
D3d	OUT	(d),A				
EDAB	OUTD		●	●	●	
EDA3	OUTI		●	●	●	
F1	POP	AF				
C1	POP	BC				
D1	POP	DE				
E1	POP	HL				
DDE1	POP	IX				
FDE1	POP	IY				
F5	PUSH	AF				
C5	PUSH	BC				
D5	PUSH	DE				
E5	PUSH	HL				
DDE5	PUSH	IX				
FDE5	PUSH	IY				
CB86	RES	0,(HL)				
DDCBd86	RES	0,(IX+d)				
FDCBd86	RES	0,(IY+d)				
CB87	RES	0,A				
CB80	RES	0,B				
CB81	RES	0,C				
CB82	RES	0,D				
CB83	RES	0,E				
CB84	RES	0,H				
CB85	RES	0,L				
CB8E	RES	1,(HL)				
DDCBd8E	RES	1,(IX+d)				

Object code	Instruction	S	Z	P/V	C
FDCBd8E	RES 1,(IY+d)				
CB8F	RES 1,A				
CB88	RES 1,B				
CB89	RES 1,C				
CB8A	RES 1,D				
CB8B	RES 1,E				
CB8C	RES 1,H				
CB8D	RES 1,L				
CB96	RES 2,(HL)				
DDCBd96	RES 2,(IX+d)				
FDCBd96	RES 2,(IY+d)				
CB97	RES 2,A				
CB90	RES 2,B				
CB91	RES 2,C				
CB92	RES 2,D				
CB93	RES 2,E				
CB94	RES 2,H				
CB95	RES 2,L				
CB9E	RES 3,(HL)				
DDCBd9E	RES 3,(IX+d)				
FDCBd9E	RES 3,(IY+d)				
CB9F	RES 3,A				
CB98	RES 3,B				
CB99	RES 3,C				
CB9A	RES 3,D				
CB9B	RES 3,E				
CB9C	RES 3,H				
CB9D	RES 3,L				
CBA6	RES 4,(HL)				
DDCBdA6	RES 4,(IX+d)				
FDCBdA7	RES 4,(IY+d)				
CBA7	RES 4,A				
CBA0	RES 4,B				
CBA1	RES 4,C				
CBA2	RES 4,D				
DBA3	RES 4,E				
CBA4	RES 4,H				
CBA5	RES 4,L				
CBAE	RES 5,(HL)				
DDCBdAE	RES 5,(IX+d)				
FDCBdAE	RES 5,(IY+d)				
CBAF	RES 5,A				
CBA8	RES 5,B				
CBA9	RES 5,C				
CBAA	RES 5,D				
CBAB	RES 5,E				
CBAC	RES 5,H				

Object code	Instruction	S	Z	P/V	C
CBAD	RES 5,L				
CBB6	RES 6,(HL)				
DDCBdB6	RES 6,(IX+d)				
FDCBdB6	RES 6,(IY+d)				
CBB7	RES 6,A				
CBB0	RES 6,B				
CBB1	RES 6,C				
CBB2	RES 6,D				
CBB3	RES 6,E				
CBB4	RES 6,H				
CBB5	RES 6,L				
CBBE	RES 7,(HL)				
DDCBdBE	RES 7,(IX+d)				
FDCBdBE	RES 7,(IY+d)				
CBBF	RES 7,A				
CBB8	RES 7,B				
CBB9	RES 7,C				
CBBA	RES 7,D				
CBBB	RES 7,E				
CBBC	RES 7,H				
CBBD	RES 7,L				
C9	RET				
D8	RET C				
F8	RET M				
D0	RET NC				
C0	RET NZ				
F0	RET P				
E8	RET PE				
E0	RET PO				
C8	RET Z				
ED4D	RETI				
ED45	RETN				
CB16	RL (HL)	●	●	●	●
DDCBd16	RL (IX+d)	●	●	●	●
FDCBd16	RL (IY+d)	●	●	●	●
CB17	RL A	●	●	●	●
CB10	RL B	●	●	●	●
CB11	RL C	●	●	●	●
CB12	RL D	●	●	●	●
CB13	RL E	●	●	●	●
CB14	RL H	●	●	●	●
CB15	RL L	●	●	●	●
17	RLA				●
CB06	RLC (HL)	●	●	●	●
DDCBd06	RLC (IX+d)	●	●	●	●
FDCBd06	RLC (IY+d)	●	●	●	●

Object code	Instruction	S	Z	P/V	C
CBDB	SET 3,E				
CBDC	SET 3,H				
CBDD	SET 3,L				
CBE6	SET 4,(HL)				
DDCBdE6	SET 4,(IX+d)				
FDCBdE6	SET 4,(IY+d)				
CBE7	SET 4,A				
CBE0	SET 4,B				
CBE1	SET 4,C				
CBE2	SET 4,D				
CBE3	SET 4,E				
CBE4	SET 4,H				
CBE5	SET 4,L				
CBEE	SET 5,(HL)				
DDCBdEE	SET 5,(IX+d)				
FDCBdEE	SET 5,(IY+d)				
CBEF	SET 5,A				
CBE8	SET 5,B				
CBE9	SET 5,C				
CBEA	SET 5,D				
CBEB	SET 5,E				
CBEC	SET 5,H				
CBED	SET 5,L				
CBF6	SET 6,(HL)				
DDCBdF6	SET 6,(IX+d)				
FDCBdF6	SET 6,(IY+d)				
CBF7	SET 6,A				
CBF0	SET 6,B				
CBF1	SET 6,C				
CBF2	SET 6,D				
CBF3	SET 6,E				
CBF4	SET 6,H				
CBF5	SET 6,L				
CBFE	SET 7,(HL)				
DDCBdFE	SET 7,(IX+d)				
FDCBdFE	SET 7,(IY+d)				
CBFF	SET 7,A				
CBF8	SET 7,B				
CBF9	SET 7,C				
CBFA	SET 7,D				
CBFB	SET 7,E				
CBFC	SET 7,H				
CBFD	SET 7,L				
CB26	SLA (HL)	●	●	●	●
DDCBd26	SLA (IX+d)	●	●	●	●
FDCBd26	SLA (IY+d)	●	●	●	●
CB27	SLA A	●	●	●	●
CB20	SLA B	●	●	●	●

Object code	Instruction	S	Z	P/V	C
CB21	SLA C	●	●	●	●
CB22	SLA D	●	●	●	●
CB23	SLA E	●	●	●	●
CB24	SLA H	●	●	●	●
CB25	SLA L	●	●	●	●
CB2E	SRA (HL)	●	●	●	●
DDCBd2E	SRA (IX+d)	●	●	●	●
FDCBd2E	SRA (IY+d)	●	●	●	●
CB2F	SRA A	●	●	●	●
CB28	SRA B	●	●	●	●
CB29	SRA C	●	●	●	●
CB2A	SRA D	●	●	●	●
CB2B	SRA E	●	●	●	●
CB2C	SRA H	●	●	●	●
CB2D	SRA L	●	●	●	●
CB3E	SRL (HL)	●	●	●	●
DDCBd3E	SRL (IX+d)	●	●	●	●
FDCBd3E	SRL (IY+d)	●	●	●	●
CB3F	SRL A	●	●	●	●
CB38	SRL B	●	●	●	●
CB39	SRL C	●	●	●	●
CB3A	SRL D	●	●	●	●
CB3B	SRL E	●	●	●	●
CB3C	SRL H	●	●	●	●
CB3D	SRL L	●	●	●	●
96	SUB (HL)	●	●	●	●
DD96d	SUB (IX+d)	●	●	●	●
FD96d	SUB (IY+d)	●	●	●	●
97	SUB A	●	●	●	●
90	SUB B	●	●	●	●
91	SUB C	●	●	●	●
92	SUB D	●	●	●	●
93	SUB E	●	●	●	●
94	SUB H	●	●	●	●
95	SUB L	●	●	●	●
D6d	SUB d	●	●	●	●
AE	XOR (HL)	●	●	●	0
DDAEd	XOR (IX+d)	●	●	●	0
FDAEd	XOR (IY+d)	●	●	●	0
AF	XOR A	●	●	●	0
A8	XOR B	●	●	●	0
A9	XOR C	●	●	●	0
AA	XOR D	●	●	●	0
AB	XOR E	●	●	●	0
AC	XOR H	●	●	●	0
AD	XOR L	●	●	●	0
EEd	XOR d	●	●	●	0

DISASSEMBLY TABLES

Single byte instructions

n = bytes (8 bits, from 0 to 255)
nn = Double bytes (16 bits, from 0 to 65535)
d = relative address offset (8 bits)

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	DJNZ d	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR d	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ,d	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA	JR Z,d	ADD HL,HL	LD HL,(nn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,d	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF C,d	JR C,d	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST 0	RET Z	RET	JP Z,nn		CALL Z,nn	CALL nn	ADC A,n	RST 8
D	RET NC	POP DE	JP NC,nn	OUT (n),A	CALL NC,nn	PUSH DE	SUB n	RST 16	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn		SBC A,n	RST 24
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	CALL PO,nn	PUSH HL	AND n	RST 32	RET DE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn		XOR n	RST 40
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR n	RST 48	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn		CP n	RST 56

Two-byte instructions prefixed with CB

All the instructions in this table must be preceded by the prefix CB.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3									SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

Two-byte instructions prefixed with ED

All the instructions in this table must be preceded by the prefix ED.

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø																
1																
2																
3																
4	IN B,(C)	OUT (C),B	SBC HL,BC	ID (nn),BC	NEG	RETN	IM Ø	LD I,A	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(nn)		RETI		LD R,A
5	IN D,(C)	OUT (C),D	SBC HL,DE	LD (nn),DE			IM 1	LD A,I	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(nn)			IM 2	LD A,R
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (nn),HL				RRD	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(nn)				RLD
7	IN F,(C)		SBC HL,SP	LD (nn),SP					IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(nn)				
8																
9																
A	LDI	CPI	INI	OUTI					LDD	CPD	IND	OUTD				
B	LDIR	CPIR	INIR	OTIR					LDDR	CPDR	INDR	OTDR				
C																
D																
E																
F																

Two-byte indexed instructions prefixed with DD

All instructions in this table must be preceded by a prefix; DD in the case of the index register IX, and FD in the case of index register IY.

Code	Mnemonic	Code	Mnemonic
09	ADD IX,BC	CB d 0E	RRC (IX + d)
19	ADD IX,DE	CB d 16	RL (IX + d)
21	LD IX,nn	CB d 1E	RR (IX + d)
22	LD (nn),IX	CB d 26	SLA (IX + d)
23	INC IX	CB d 2E	SRA (IX + d)
29	ADD IX,IX	CB d 3E	SRL
2A	LD IX,(nn)	CB d 46	BIT 0,(IX + d)
2B	DEC IX	CB d 4E	BIT 1,(IX + d)
34	INC (IX + d)	CB d 56	BIT 2,(IX + d)
35	DEC (IX + d)	CB d 5E	BIT 3,(IX + d)
36	LD (IX + d),nn	CB d 66	BIT 4,(IX + d)
39	ADD IX,SP	CB d 6E	BIT 5,(IX + d)
46	LD B,(IX + d)	CB d 76	BIT 6,(IX + d)
4E	LD C,(IX + d)	CB d 7E	BIT 7,(IX + d)
56	LD D,(IX + d)	CB d 86	RES 0,(IX + d)
5E	LD E,(IX + d)	CB d 8E	RES 1,(IX + d)
66	LD H,(IX + d)	CB d 96	RES 2,(IX + d)
6E	LD L,(IX + d)	CB d 9E	RES 3,(IX + d)
70	LD (IX + d),B	CB d A6	RES 4,(IX + d)
71	LD (IX + d),C	CB d AE	RES 5,(IX + d)
72	LD (IX + d),D	CB d B6	RES 6,(IX + d)
73	LD (IX + d),E	CB d BE	RES 7,(IX + d)
74	LD (IX + d),H	CB d C6	SET 0,(IX + d)
75	LD (IX + d),L	CB d CE	SET 1,(IX + d)
77	LD (IX + d),A	CB d D6	SET 2,(IX + d)
7E	LD A,(IX + d)	CB d DE	SET 3,(IX + d)
86	ADD A,(IX + d)	CB d E6	SET 4,(IX + d)
8E	ADC A,(IX + d)	CB d EE	SET 5,(IX + d)
96	SUB (IX + d)	CB d F6	SET 6,(IX + d)
9E	SBC A,(IX + d)	CB d FE	SET 7,(IX + d)
A6	AND (IX + d)	E1	POP IX
AE	XOR (IX + d)	E3	EX (SP),IX
B6	OR (IX + d)	E5	PUSH IX
BE	CP (IX + d)	E9	JP (IX)
CB d 06	RLC (IX + d)	F9	LD SP,IX

INTERNAL SOFTWARE

INTRODUCTION

The internal software of the Amstrad can be divided into three main areas:

- The lower ROM which contains the various control routines described below, the maths routines, and character generation.
- The upper ROM contains the BASIC interpreter.
- The workspace in memory contains system variables, call vectors for the routines in the lower ROM and the various buffers used by controllers and BASIC.

The control routines can be divided into nine main groups:

The keyboard controller

Controls the keyboard, generates the characters associated with key functions, tests for BREAK and monitors the joysticks.

The text mode controller

This looks after the management of the cursor, interpretation of control codes and the screen display of characters.

The graphic controller

This draws pixels (points) and lines on the screen.

The screen controller

This interfaces text and graphics with the specialised screen management routines and circuits.

The tape controller

This handles reading from and writing to the tape, together with control of the tape motor.

The sound controller

Deals with sound queues, envelopes, mixing and so on.

The Kernel

This is the heart of the operating system which deals with interrupts, execution of programs and ROM memory management.

Low-level management system

This deals with the management of the printer interface and with low-level routines.

The jump block

Controls vectoring.

For ease of understanding, the software system will be presented as follows:

- RAM memory entry points for system subroutines.
- Indirect vectors.
- Kernel vectors and restarts.
- Vectors to the maths routines.
- The main system variables in RAM.
- Principal addresses in the lower ROM.
- Principal addresses in the upper ROM.
- A table showing the relationship between vectors and addresses in the lower ROM.
- A table of BASIC keyword routine addresses.
- The principal operating system tables.

OPERATING SYSTEM ENTRY POINTS

For each numbered subroutine, the entry point is shown (in hex) followed by an explanation.

Keyboard management routines

Note:
Throughout these descriptions, flag status is referred to as *true* if the flag is set to 1, and *false* if the flag is set to 0.

00 BB00 **Initialise the keyboard manager**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified. All other registers are preserved.

01 BB03 **Reset keyboard manager**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified. All other registers are preserved.

02 BB06 **Waits for a character to be typed into the keyboard**
Entry conditions: none
Exit conditions: If the carry flag is true, the accumulator will contain the ASCII code of the character that has been typed. All registers are preserved. Expansion tokens are expanded.

03 BB09 **This routine tests whether a character is available from the keyboard and reads it if it is**
Entry conditions: none
Exit conditions: If a character is available, the carry flag is true and A contains the ASCII code of the character. If there is not an available character, the carry flag is false and A is modified. All other registers are preserved. Expansion tokens are expanded.

04 BB0C **Saves a character for next call of the previous routine**
Entry conditions: A contains the character to be saved.
Exit conditions: All registers preserved.

05 BB0F **Associates a character string with a key-code**
Entry conditions: B contains the key-code to be associated with the string.
C contains the length of the string.
HL contains the address of the string.
Exit conditions: If the operation has been successful, the carry flag is set to true. If the string is too long or the key-code is invalid, the carry flag is set false. A, BC, DE and HL are all modified.

06 BB12 **Reads a character from an expanded string of characters**
Characters in the string are numbered from 0.
Entry conditions: A contains the expansion code. L contains the character number.

Exit conditions: If the character is found, A contains the character and the carry flag is set true. If the instruction is invalid or if the string is too long then carry flag is set false and A is modified. DE is modified.

07 **BB15** **Allocation of a buffer to an expanded character string**
Entry conditions: DE contains the address of the buffer and HL its length.
Exit conditions: if everything is correct, the carry flag is set to true; otherwise it is set false. Registers A, BC, DE and HL are modified.

08 **BB18** **Waits for a character from the keyboard**
Entry conditions: None
Exit conditions: The carry flag is true and A contains the character typed. All registers are preserved. Expansion tokens are not expanded.

09 **BB1B** **Tests whether a character is available at the keyboard**
Entry conditions: None
Exit conditions: If a character is available, the carry flag is true and A contains the character; otherwise, the carry flag is false. Expansion tokens are not expanded.

10 **BB1E** **Tests whether a key has been pressed**
Also allows testing of the joystick.
Entry conditions: A contains the key number or joystick position number to be tested.
Exit conditions: If the key is not pressed, the zero flag is true; if the key is pressed, the zero flag is false. The carry flag is always false, A and HL are modified, C will contain the status of the SHIFT and Control keys.

11 **BB21** **Checks whether the SHIFT or CAPS/LOCK key is pressed**
Entry conditions: None
Exit conditions: L contains the status of SHIFT LOCK key and H contains the status of CAPS LOCK key for the upper case mode. H contains 00 if the lock is off and FF if the lock is on . The AF register is modified.

12 **BB24** **Reads the status of the joystick**
Entry conditions: None
Exit conditions: H contains the status of joystick number 0.
L contains the status of joystick number 1.
A contains the status of joystick number 0.
The use of bits is the same as in the JOY function described earlier under BASIC functions.

13 **BB27** **Sets the code to be returned when pressing a key without** accompanying CTRL or SHIFT
Entry conditions: A contains the key number, B contains the ASCII code that this key is to return.
Exit conditions: AF and HL are modified.

14 **BB2A** **Returns a code corresponding to the number of a pressed key**
Entry conditions: A contains the key number
Exit conditions: A contains the ASCII code corresponding to the key. HL and F are modified.

15 **BB2D** **Sets the code that will be returned when pressing a SHIFTed key***Entry conditions:* A contains the key number, B contains the ASCII code that this key is to return.
Exit conditions: AF and HL are modified.

16 **BB30** **Returns the ASCII code of a SHIFTed key**
Entry conditions: A contains the key number.
Exit conditions: A contains the ASCII code corresponding to the key. HL and F are modified.

17 **BB33** **Sets the code that will be returned when pressing CTRL and a key**
Entry conditions: A contains the number of the key. B contains the ASCII code to be returned by this key.
Exit conditions: AF and HL are modified.

18 **BB36** **Returns the ASCII code corresponding to a key pressed at the same time as CTRL key**
Entry conditions: A contains the key number.
Exit conditions: A contains the ASCII code corresponding to the key. HL and F are modified.

19 **BB39** **Sets whether a key auto-repeats**
Entry conditions: A contains the key number. If the key is to repeat then B should contain FF; otherwise B should contain 00.
Exit conditions: AF, BC and HL are modified.

20 **BB3C** **Tests whether a specified key has been set to auto-repeat**
Entry conditions: A contains the key number.
Exit conditions: If the key can auto-repeat, the zero flag is set false, if it cannot be repeated, the zero flag is set true. In both cases the carry flag is set false and AF and HL are modified.

21 **BB3F** **Sets the duration of the delay before auto-repeating and sets the repeat delay**
Entry conditions: H contains the delay before the first repeat. L contains the speed of repetition. Both delays are expressed in 50ths of a second.
Exit conditions: AF is modified.

22 **BB42** **Returns auto-repeat delay and repeat interval**
Entry conditions: none
Exit conditions: H contains the delay before the first repetition and L contains the repeat delay, both expressed in 50ths of a second. AF is modified.

23 **BB45** **Arm the BREAK routine**
Entry conditions: DE contains the address of the BREAK handling routine, C contains the ROM address selected for this routine.
Exit conditions: AF, BC, DE and HL are modified.
Note:
This routine can be disabled by calling the next routine.

24 **BB48** **Disable the BREAK routine**
Entry conditions: none
Exit conditions: AF and HL are modified.

25 **BB4B** **Generates a BREAK interrupt if a BREAK routine has been specified by routine 23**
Entry conditions: none
Exit conditions: AF and HL are modified

Text management routines

26 **BB4E** **Initialise text mode**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

27 **BB51** **Reset text mode**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

28 **BB54** **Allows characters to be printed to the current stream**
Entry conditions: none
Exit conditions: AF is modified.

29 **BB57** **Prevents characters from being displayed on the screen**
Entry conditions: none
Exit conditions: AF is modified.

30 **BB5A** **Sends a character or control code (ASCII 0 to 1F) to the screen in text mode**
Entry conditions: A contains the character to be sent.
Exit conditions: All registers unchanged.

31 **BB5D** **Sends a character or a control code instruction to the screen in text mode**
Entry conditions: A contains the character to be printed.
Exit conditions: AF, BC, DE and HL are changed.

32 **BB60** **Reads a character from the screen at the current cursor position**
Entry conditions: none
Exit conditions: If a character has been found then the carry flag is set true and A contains the character. Otherwise the carry flag is false and A contains 0.

33 **BB63** **Turns the graphic character processor on or off**
Entry conditions: A set to 0 to turn graphics generator off, if A is not zero then graphic processor is turned on.
Exit conditions: AF is modified.

34 **BB66** **Sets the size of the current text window**
Entry conditions: H contains the column number of the left edge.
D contains the column number of the right edge.
L contains the row number of the top edge.
E contains the row number of the bottom edge.
Exit conditions: AF, BC, DE and HL are modified

35 **BB69** **Returns the size of the current window**
Entry conditions: none
Exit conditions: If the window covers the complete screen, the carry flag is set false, otherwise it is true. In both cases, H contains the number of the left column, D the number of the right column, L the number of the top line and E the number of the bottom line. A is modified.

36 **BB6C** **Clear the current window (CLS)**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

37 **BB6F** **Sets the horizontal position of the cursor**
Entry conditions: A contains the column number of the cursor.
*Exit conditions:*AF and HL are modified.

38 **BB72** **Sets the vertical position of the cursor**
Entry conditions: A contains the row number of the cursor.
Exit conditions: AF and HL are modified.

39 **BB75** **Sets the position of the cursor**
Entry conditions: H contains the column number and L contains the row number of the cursor.
Exit conditions: AF and HL are modified.

40 **BB78** **Returns the current cursor position**
Entry conditions: none
Exit conditions: H contains the column number of the cursor.
L contains the row number of the cursor.
A contains the scroll count.

41 **BB7B** **Enables the text mode cursor**
Entry conditions: none
Exit conditions: AF is modified.

42 **BB7E** **Disables the text mode cursor**
Entry conditions: none
Exit conditions: AF is modified.

43 **BB81** **Enables the operating system cursor**
Entry conditions: none
Exit conditions: none

44 **BB84** **Disables the operating system cursor**
Entry conditions: none
Exit conditions: none

45 **BB87** **Tests if a cursor position occurs within a window**
Entry conditions: H contains the column number of the position to test.
L contains the row number of the position to test.
Exit conditions: H contains the column number where the character will be printed.
L contains the row number where the character will be printed.
A and F are modified. If printing will not cause scrolling then

the carry flag is true and B is modified. If printing will cause scrolling then the carry flag is false and B contains FF. If it will cause reverse scrolling then the carry flag is false and B contains 00.

46 **BB8A** **Positions a cursor on the screen**
Entry conditions: none
Exit conditions: AF is modified.

47 **BB8D** **Removes the cursor from the screen**
Entry conditions: none
Exit conditions: AF is modified.

48 **BB90** **Sets the foreground (PEN) colour**
Entry conditions: A contains the INK number.
Exit conditions: AF and HL are modified.

49 **BB93** **Returns the foreground (PEN) colour**
Entry conditions: none
Exit conditions: A contains the INK number, F is modified.

50 **BB96** **Sets the background (PAPER) colour**
Entry conditions: A contains the INK number.
Exit conditions: AF and HL are modified.

51 **BB99** **Returns the background (PAPER) colour**
Entry conditions: none
Exit conditions: A contains the INK number of the background colour, A and F are modified.

52 **BB9C** **Swaps text and background colours**
Entry conditions: none
Exit conditions: AF and HL are modified.

53 **BB9F** **Enables/Disables background display**
Entry conditions: A = 0 if the background is to be displayed (opaque mode); if the background is not to be displayed (transparent mode) then A must contain a non-zero value.
Exit conditions: AF and HL are modified.

54 **BBA2** **Returns backgroud display mode** (*see* 53)
Entry conditions: none
Exit conditions: A will be 0 if the background can be displayed, otherwise A will contain some other value. DE, HL and F are modified.

55 **BBA5** **Returns the address of a character matrix**
Entry conditions: A contains the character to look for in the table.
Exit conditions: A and F are modified. If the table is user-defined then the carry flag is true. If the table is held in ROM, the carry is false and HL contains the address of the table.

56 **BBA8** **Creates a matrix for a user-defined character**
Entry conditions: A contains the character representing the matrix and HL contains the address of the table.

Exit conditions: If the character is user-defined then the carry flag is true, otherwise it is false. AF, BC, DE and HL are modified.

57 **BBAB** **Sets the address of a user-defined matrix table**
Entry conditions: DE contains the first character of the table and HL contains the first address of the new table.
Exit conditions: If there is no existing table then the carry flag is false and A and HL are modified. If a table has already been defined by the user, the carry flag is true, A contains the first character of the old table, HL contains the address of the old table and BC and DE are modified.

58 **BBAE** **Reads the table address of a user-defined matrix**
Entry conditions: none
Exit conditions: If there are no matrix tables defined by the user, the carry flag is false, A and HL are modified. If there is a table, the carry flag is true, A contains the first character of the table and HL contains the address of the table.

59 **BBB1** **Returns the address of the control code table.**
Entry conditions: none
Exit conditions: HL contains the address of the control codes. All the other registers are preserved.

60 **BBB4** **Sets a new VDU stream (attribute) table**
Entry conditions: A contains the number of stream required.
Exit conditions: A contains the number of the old stream, HL and F are modified.

61 **BBB7** **Swaps the states of the two stream (attribute) tables**
Entry conditions: B contains the number of stream 1.
C contains the number of stream 2.
Exit conditions: AF, BC, DE and HL are modified.

Note:
A stream table consists of an INK number, a PAPER number, a cursor position and the WINDOW parameters.

Graphics management routines

62 **BBBA** **Initialise graphic mode**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

63 **BBBD** **Reset graphic management system**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

64 **BBC0** **Jump to absolute screen co-ordinate position**
Entry conditions: DE contains the absolute X co-ordinate.
HL contains the absolute Y co-ordinate.
Exit conditions: AF, BC, DE and HL are modified.

65 **BBC3** **Jump to a screen co-ordinate position relative to the current cursor position**
Entry conditions: DE contains the relative X co-ordinate.
HL contains the relative Y co-ordinate.
Exit conditions: AF, BD, DE and HL are modified.

66 **BBC6** **Returns current position of the graphic cursor**
Entry conditions: none
Exit conditions: DE contains the X co-ordinate, HL contains the Y co-ordinate. AF is modified.

67 **BBC9** **Set cursor origin (home) position**
Entry conditions: DE contains the X co-ordinate of the origin.
HL contains the Y co-ordinate of the origin.
Exit conditions: AF, BC, DE and HL are modified.

68 **BBCC** **Returns the co-ordinates of the current origin**
Entry conditions: none
Exit conditions: DE contains the X co-ordinate of the origin.
HL contains the Y co-ordinate of the origin.

69 **BBCF** **Set left and right edges of a graphic window**
Entry conditions: DE contains the horizontal co-ordinate of one edge.
HL contains the horizontal co-ordinate of the other edge.
Exit conditions: AF, BC, DE and HL are modified.

70 **BBD2** **Set top and bottom edges of a graphic window**
Entry conditions: DE contains the Y co-ordinate of one of the edges.
HL contains the Y co-ordinate of the other edge.
Exit conditions: AF, BC, DE and HL are modified.

71 **BBD5** **Returns left and right edge values of a graphic window**
Entry conditions: none
Exit conditions: DE contains the X co-ordinate of the left edge.
HL contains the X co-ordinate of the right edge.
AF is modified.

72 **BBD8** **Returns top and bottom edge values of a graphic window**
Entry conditions: none
Exit conditions: DE contains the Y co-ordinate of the top edge of the window.
HL contains the Y co-ordinate of the bottom edge of the window.
AF is modified.

73 **BBDB** **Clears a graphic window**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

74 **BBDE** **Sets graphics INK colour**
Entry conditions: A contains the colour number.
Exit conditions: AF is modified.

75 **BBE1** **Returns the graphic INK colour**
Entry conditions: None
Exit conditions: A contains the colour number.

76 **BBE4** **Sets background (PAPER) colour**
Entry conditions: A contains the colour number.
Exit conditions: AF is modified.

77 **BBE7** **Returns current background (PAPER) colour**
Entry conditions: None
Exit conditions: A contains the colour number.

78 **BBEA** **Displays a pixel at an absolute co-ordinate**
Entry conditions: DE contains the absolute X co-ordinate.
HL contains the absolute Y co-ordinate.
Exit conditions: AF, BC, DE and HL are cleared.

79 **BBED** **Displays a pixel at a relative co-ordinate**
Entry conditions: DE contains the relative X co-ordinate.
HL contains the relative Y co-ordinate.
Exit conditions: AF, BC, DE and HL are modified.

80 **BBFO** **Tests a pixel at an absolute co-ordinate**
Entry conditions: DE contains the absolute X co-ordinate.
HL contains the absolute Y co-ordinate.
Exit conditions: A contains the INK colour number of the tested pixel, BC, DE and HL are modified.

81 **BBF3** **Tests a pixel at a relative co-ordinate**
Entry conditions: DE contains the relative X co-ordinate.
HL contains the relative Y co-ordinate.
Exit conditions: A contains the INK colour of the tested pixel, BC, DE and HL are modified.

82 **BBF6** **Draws a line from the current cursor position to an absolute co-ordinate position**
Entry conditions: DE contains the absolute X co-ordinate of the end pixel.
HL contains the absolute Y co-ordinate of the end pixel.
The line will be drawn from the current cursor position to the absolute position.
Exit conditions: AF, BC, DE and HL are modified.

83 **BBF9** **Draws a line from the current cursor position to a relative co-ordinate position**
Entry conditions: DE contains the relative X co-ordinate of the end pixel.
HL contains the relative Y co-ordinate of the end pixel.
The line will be drawn from the current cursor position to the relative position.
Exit conditions: AF, BC, DE and HL are modified.

84 **BBFC** **Writes a character at the graphic cursor position**
Entry conditions: A contains the character to be written.
Exit conditions: AF, BC, DE and HL are modified.

Screen management routines

85 BBFF **Initialisation of the screen management system**
Modes, INK and PAPER values use the default values.
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

86 BC02 **Re-initialisation of screen management system**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

87 BC05 **Sets the initial screen OFFSET value**
Modifying this value can cause the screen to scroll.
Entry conditions: HL contains the desired OFFSET value.
Exit conditions: AF and HL are modified.

89 BCOB **Returns the screen memory address and the OFFSET value**
Entry conditions: none
Exit conditions: A contains the high byte of the screen memory address and HL contains the current OFFSET value. F is modified.

90 BCOE **Sets a screen mode**
Entry conditions: A contains the mode number.
Exit conditions: AF, BC, DE and HL are modified.

91 BC11 **Returns the current screen mode**
Entry conditions: none
Exit conditions: A contains the mode number, the carry and zero flags are set according to the mode:

Mode 0:	Carry = 1,	Zero = 0
Mode 1:	Carry = 0,	Zero = 1
Mode 2:	Carry = 0,	Zero = 0

92 BC14 **Clears the screen**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

93 BC17 **Returns the size**
Entry conditions: none
Exit conditions: B contains the last physical column number of the screen, C contains the last row number and AF is modified.

94 BC1A **Returns the memory address of a character whose screen position has been provided**
Entry conditions: H contains the column and L contains the row.
Exit conditions: HL contains the real memory address, B contains the width in bytes of the character in memory and AF is modified.

95 BC1D **Returns the memory address of a pixel whose screen position has been provided**
Entry conditions: DE contains the X co-ordinate of the pixel and HL contains the Y co-ordinate.
Exit conditions: HL contains the memory address of the pixel, B contains the number of pixels in a byte − 1, C contains the pixel mask. AF and DE are modified.

96 **BC20** **Calculation of the real address of the byte to the right of the real current address**
Entry conditions: HL contains the current address.
Exit conditions: HL contains the new address and AF is modified.

97 **BC23** **As 96 (BC20), but for the byte to the left**

98 **BC26** **As 96 (BC20), but for the next line down**

99 **BC29** **As 96 (BC20), but for the preceding line**

100 **BC2C** **Conversion of an INK number to provide a mask**
This mask, if applied to a pixel storage byte will set all the pixels in the appropriate INK colour
Entry conditions: A contains the INK colour.
Exit conditions: A contains the mask, F is modified.

101 **BC2F** **Extraction of an INK colour from a mask** *(see above)*
Entry conditions: A contains the mask
Exit conditions: A contains the INK number, F is modified.

102 **BC32** **Sets INK colours**
Entry conditions: A contains the INK number.
B contains the first colour.
C contains the second colour.
Exit conditions: AF, BC, DE and HL are modified.

103 **BC35** **Returns current INK colour values**
Entry conditions: A contains the INK number
Exit conditions: B contains the first colour.
C contains the second colour.
AF, DE and HL are modified.

104 **BC38** **Sets the colours of the screen border**
Entry conditions: B contains the first colour.
C contains the second colour.
Exit conditions: AF, BC, DE and HL are modified.

105 **BC3B** **Returns the border colours**
Entry conditions: none
Exit conditions: B contains the first colour.
C contains the second colour.
AF, DE and HL are modified.

106 **BC3E** **Sets the flash rate of the border colours**
Entry conditions: H contains the duration of the first colour.
L contains the duration of the second colour.
*Exit conditions:*AF and HL are modified.

107 **BC41** **Returns the flash rates of the border colours**
Entry conditions: none
Exit conditions: H contains the duration of the first colour.
L contains the duration of the second.
AF is modified.

108 BC44 **Fills a rectangle with INK**
Entry conditions: A contains the mask corresponding to the INK to be used.
H contains the left-hand column number.
D contains the right-hand column number.
L contains the top line number.
E contains the bottom line number.
Exit conditions: AF, BC, DE and HL are modified.

109 BC47 **Masks a series of bytes in screen memory with INK values**
Entry conditions: A contains the INK mask.
HL contains the memory address corresponding to the top left corner.
D contains the width, in bytes, to be set.
E contains the height in screen lines.
Exit conditions: AF, BC, DE and HL are modified.

110 BC4A **Swaps the two colour values associated with a character**
Entry conditions: B contains the mask for the first colour.
C contains the mask for the second colour.
H contains the column number.
L contains the row number.
Exit conditions: AF, BC, DE and HL are modified.

111 BC4D **Moves the entire screen eight pixels up or down**
Entry conditions: B must be 0 for a downwards movement.
B must be non-zero to move upwards.
Exit conditions: AF, BC, DE and HL are modified.

112 BC50 **Moves a part of the screen eight pixels up or down**
Entry conditions: B must be 0 for a downwards movement.
B must be non-zero for an upwards movement.
A contains the INK mask to clear the new line.
H contains the left column number.
D contains the right column number.
L contains the upper line number.
E contains the lower line number.

113 BC53 **Conversion of a character matrix from its standard form into a series of pixel masks in the current mode**
Entry conditions: HL contains the address of the matrix.
DE contains the address where the masks are to be stored.
Exit conditions: AF, BC, DE and HL are modified

114 BC56 **Conversion of a series of current mode pixel masks into a standard character matrix (inverse of 113)**
Entry conditions: A contains the INK mask to be matched.
H contains the character column.
L contains the character row.
DE contains the address where the matrix will be built.
Exit conditions: AF, BC, DE and HL are modified.

115 BC59 **Sets the screen write mode for graphics**
Entry conditions: A contains the mode (0 = Fill, 1 = exclusive OR, 2 = AND, 3 = OR).
Exit conditions: AF, BC, DE and HL are modified.

116 BC5C **Writes a pixel on the screen regardless of the mode defined by the preceding routine (115)**
Entry conditions: B contains the INK mask.
C contains the pixel mask.
HL contains the memory address of the pixel.
Exit conditions: AF is modified.

117 BC5F **Draws a horizontal line**
Entry conditions: A contains the INK mask.
DE contains the start X co-ordinate.
BC contains the end X co-ordinate.
HL contains the Y co-ordinate.
Exit conditions: AF, BC, DE and HL are modified.

118 BC62 **Draws a vertical line**
Entry conditions: A contains the INK mask
DE contains the X co-ordinate of the line
HL contains the start Y co-ordinate.
BC contains the end Y co-ordinate.
Exit conditions: AF, BC, DE and HL are modified.

Tape management routines

119 BC65 **Initialises the tape management system**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

120 BC68 **Set tape write speed**
Entry conditions: HL contains the length of half a zero bit.
A contains the pre-equalisation value required.
Exit conditions: AF and HL are modified.

121 BC6B **Enables/Disables display of tape prompt messages**
Entry conditions: A set to 0 to enable, to non-zero to disable message display.
Exit conditions: AF is modified.

122 BC6E **Turns tape motor ON**
Entry conditions: none
Exit conditions: If the motor responds as expected, the carry flag is true; if ESC has been pressed, the carry flag is false. A reflects the previous state of the motor.

123 BC71 **Turns tape motor OFF**
Entry conditions: none
Exit conditions: as above (122).

124 BC74 **Resets tape motor to previous state**
Entry conditions: A contains the previous state of the motor.
Exit conditions: as above (122).

125 BC77 **Opens a read buffer and reads in the first block**
Entry conditions: B contains the length of the file name.
HL contains the address of the file name.
DE contains the address of the 2K data buffer.
Exit conditions: If the operation is successful, the carry flag will be true and the zero flag false.
HL contains the address of the buffer carrying the header data.
DE contains the address of the file data.
BC contains the length of the file.
A contains the file-type.
If the channel has already been used, the carry flag is false and A, BC, DE and HL will all have been modified.
If ESC has been pressed, the carry flag will be false and the zero flag will be true. AF, BC, DE and HL will all be modified.
In all cases IX is modified.

126 BC7A **Closes a file**
Entry conditions: none
Exit conditions: If successful, the carry flag is true, otherwise the carry flag is false. In both cases registers AF, BC, DE and HL will be modified.

127 BC7D **Abandons reading of a tape and closes the file**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

128 BC80 **Reads a single byte**
Entry conditions: none
Exit conditions: If successful, the carry flag is true, the zero flag is false and A contains the character read.
If the end of file (EOF) has been encountered then the carry and zero flags will both be false and A will be changed.
If ESC has been pressed, the carry flag will be false, the zero flag will be true and A will have been modified.
In both cases IX is modified.

129 BC83 **Reads file data into memory**
Entry conditions: HL contains the address in memory to store the file.
Exit conditions: As 128 for the carry and zero flags. HL contains the entry point if the read is successful. In both cases AF, BC, DE and HL and IX are modified.

130 BC86 **Places the last character read by routine 128 back into the read buffer**
Entry conditions: none
Exit conditions: none

131 BC89 **Tests whether the end of file (EOF) has been reached**
Entry conditions: none
Exit conditions: If the end of the file has been reached, the carry and

zero flags are false. If the end of file has not been reached, the carry flag is true and the zero flag false. If the user has pressed ESC (break), the carry flag will be false and the zero flag true. In both cases AF and IX are modified.

132 BC8C **Opens a file for output**
Entry conditions: B contains the length of the file name.
HL contains the address of the file name.
DE contains the address of the next 2K file buffer.
Exit conditions: If the file has been correctly opened, the carry flag is true, the zero flag is false and HL contains the address of the header buffer to be written at the start of each data block. If the user has pressed ESC, the carry flag is false and the zero flag is true. If the buffer has already been used, the carry and zero flags will both be false. In both cases AF, BC, DE, HL and IX are modified.

133 BC8F **Normal close of an output file**
Entry conditions: none
Exit conditions: If the close has been successful, the carry flag is true and the zero flag is false. If the file was not open in the first place then the carry and zero flags will both be false. If ESC has been pressed, the carry flag will be false and the zero flag true. In both cases, AF, BC, DE, HL and IX will be modified.

134 BC92 **Immediate close of an output file**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are all modified.

135 BC95 **Write a single character to an output file**
Entry conditions: A contains the character to write
Exit conditions: If the operation is successful the carry flag is true and the zero flag is false. If the file was not open, the carry and zero flags are both false. If ESC has been pressed then the carry flag will be false and the zero flag true. In both cases, AF and IX will be modified.

136 BC98 **Direct write of memory contents to an output file**
Entry conditions: HL contains the memory address.
DE contains the number of bytes to be written.
BC contains the entry point.
A contains the type of file.
Exit conditions: As routine 135, but AF, BC, DE, HL and IX are modified.

137 BC9B **Records a tape directory**
Entry conditions: DE contains the address of data to write.
Exit conditions: If the recording went correctly, the carry flag will be true. Otherwise, the carry flag will be false. In both cases, AF, BC, DE, HL and IX are modified.

138 BC9E **Writes data to tape**
Entry conditions: HL contains the address of the data to be written.
DE contains the number of bytes to write.

A contains the synchronisation character.
Exit conditions: If the write went correctly, the carry flag will be true, otherwise the carry flag will be false and A will contain an error code. In both cases AF, BC, DE, HL and IX are modified.

139 BCA1 Reads data from tape
Entry conditions: HL contains the address to which the data will be written.
DE contains the number of bytes to read.
A contains the synchronisation character.
Exit conditions: If the read went correctly, the carry flag will be true, otherwise the carry flag will be false and A will contain an error code. In both cases AF, BC, DE, HL and IX are modified.

140 BCA4 Compares a tape recording with the contents of memory
Entry conditions: HL contains the address of data to be compared.
DE contains the number of bytes to compare.
A contains the synchronisation character.
Exit conditions: If the comparison produces a perfect match then the carry flag is set to true, otherwise the carry flag is set false and A contains an error code. In both cases, AF, BC, DE, HL and IX are modified.

Sound management routines

141 BCA7 Initialises the sound management system
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

142 BCAA Adds a sound to a sound queue
Entry conditions: HL contains the address of the sound program which must be within the 32K central RAM memory.
Exit conditions: If the sound has been correctly added to the queue, the carry flag is true and HL is modified. If all sound queues are full and the required sound has not been added to one of them, the carry flag is false and HL will be unchanged. In both cases AF, BC, DE and IX are modified. all other registers are preserved.

143 BCAD Checks whether there is space available in a sound queue
Entry conditions: A contains the number of the sound channel to be tested:
0 tests channel A
1 tests channel B
2 tests channel C
Exit conditions: A contains the status of the channel tested and F, BC, DE and HL are all modified.

144 BCB0 Sets up an interrupt for use when a sound queue is empty
Entry conditions: A contains the number of the sound channel to be monitored:
0 tests channel A
1 tests channel B
2 tests channel C
HL contains the address of the interrupt routine.
Exit conditions: AF, BC, DE and HL are modified.

145 BCB3 **Resumes sound output through a specified channel after inhibition by routine 146**
Entry conditions: A contains the channel number to release:
0 tests channel A
1 tests channel B
2 tests channel C
Exit conditions: AF, BC, DE and HL are modified.

146 BCB6 **Stops *all* sound output**
Entry conditions: none
Exit conditions: If a sound channel was active, the carry flag will be true. If no sound was active, the carry flag will be false. In both cases, AF, BC and HL are modified.

147 BCB9 **Restarts *all* sounds stopped by routine 146**
Entry conditions: none
Exit conditions: AF, BC, DE and IX are modified.

148 BCBC **Sets up of one of the 15 programmable amplitude envelopes**
Entry conditions: A contains the envelope number.
HL contains the address of the amplitude data.
Exit conditions: If an envelope has been correctly set up, the carry flag is true, HL contains the block address of data + 16, A and BC are modified.
If the envelope number was invalid then carry flag is false and A, B and HL are all modified.
In both cases F and DE will be modified.

149 BCBF **Sets up of one of the 15 programmable frequency envelopes**
Entry conditions: A contains an envelope number.
HL contains the address of the frequency data.
Exit conditions: If the frequency envelope has been correctly set up, the carry flag is true, HL contains the block address of data + 16, A and BC are modified.
If the envelope number was invalid then carry flag is false and A, B and HL are all preserved.
In both cases F and DE will be modified.

150 BCC2 **Returns the address of an amplitude envelope**
Entry conditions: A contains the envelope number.
Exit conditions: If the envelope is valid then the carry is true, HL contains the address of the envelope and BC contains its length.
If the envelope number is invalid, the carry flag will be wrong, HL will be modified and BC will be preserved.
In both cases AF will be modified.

151 BCC5 **Returns the address of a tone envelope**
Entry conditions: A contains an envelope number.
Exit conditions: If the envelope is valid then the carry flag will be set to true, HL will contain the address of the envelope and BC the length of the envelope.

If the envelope number is invalid, the carry flag will be false, HL will have been modified and BC will be unchanged.
In both cases AF will have been modified.

The Kernel

152 BCC8 **Clears all interrupts and clocks**
Entry conditions: none
Exit conditions: B contains the ROM select address (if relevant).
DE contains the ROM entry point.
C contains the ROM select address of a program in RAM.
AF and HL are both modified.

153 BCCB **Locate and initialise all background ROMs**
Entry conditions: DE contains the address of the first usable byte of memory.
HL contains the address of the last usable byte of memory.
Exit conditions: DE contains the address of the new first usable byte of memory.
HL contains the address of the new last usable byte of memory.
AF and BC are modified.

154 BCCE **Initialise a background ROM**
Entry conditions: C contains the selection address of the ROM to be initialised.
DE contains the address of the first usable byte of memory.
HL contains the address of the last usable byte of memory.
Exit conditions: DE contains the address of the first new usable byte of memory.
HL contains the address of the last new usable byte of memory.
AF and B are modified.

155 BCD1 **Introduces an RSX (Resident System eXtension) to the firmware**
Entry conditions: BC contains the address of the RSX command table.
HL contains the address of four RAM bytes for the kernel to use.
Exit conditions: DE is modified.

156 BCD4 **Searches for an RSX, background or foreground ROM to execute a command**
Entry conditions: HL contains the address of the command name to be found.
Exit conditions: If an RSX or background ROM is found, the carry flag is true, C contains the ROM selection address and HL contains the routine address.
If the command has not been found, the carry flag is false.
In both cases AF, BC and DE are modified.

157 BCD7 **Initialises an event block and adds it to the list of blocks to be activated during a CRT interrupt**
Entry conditions: HL contains the address of the event block.
B contains the class of event.

C contains the ROM selection address.
DE contains the address of event routine.
Exit conditions: AF, DE and HL are modified.

158 **BCDA** **Adds an event block to the list of blocks to be activated during a CRT interrupt**
Entry conditions: HL contains the address of the event block.
Exit conditions: AF, DE and HL are modified.

159 **BCDD** **Deletes an event block from the list of blocks to be activated during a CRT interrupt**
Entry conditions: HL contains the address of the event block.
Exit conditions: AF, DE and HL are modified.

160 **BCE0** **Initialises an event block and adds it to the list of blocks to activate during a rapid (1/300th of a second) interrupt**
Entry conditions: HL contains the address of the block.
B contains the event class.
C contains the ROM selection address.
DE contains the address of the event routine.
Exit conditions: AF, DE and HL are modified.

161 **BCE3** **Adds an event block to the list of blocks to be activated during a rapid interrupt**
Entry conditions: HL contains the address of the event block.
Exit conditions: AF, DE and HL are modified.

162 **BCE6** **Deletes an event block from the list of blocks to be activated during a rapid interrupt**
Entry conditions: HL contains the event block address.
Exit conditions: AF, DE and HL are modified.

163 **BCE9** **Adds an event block to the list of blocks to be activated during a normal (1/50th of a second) interrupt**
Entry conditions: HL contains the address of the event block.
DE contains the initial value of the counter.
BC contains the reload value for the counter when it reaches 0.
Exit conditions: AF, BC, DE and HL are modified.

164 **BCEC** **Removes an event block from the list of blocks to be activated during a normal interrupt**
Entry conditions: HL contains the address of the event block.
Exit conditions: If the block has been found in the list, the carry flag is true and DE contains the counter, otherwise the carry flag is false. In both cases, AF, DE and HL are modified.

165 **BCEF** **Initialises an event block**
Entry conditions: HL contains the address of the event block.
B contains the class of event.
C contains the ROM selection address.
DE contains the address of the event routine.
Exit conditions: HL contains the address of the event block + 7.

166 **BCF2** **Activates an event block**
Entry conditions: HL contains the address of the event block.
Exit conditions: AF, BC, DE and HL are modified.

167 **BCF5** **Clears synchronous time event queue**
Entry conditions: none
Exit conditions: AF and HL are modified.

168 **BCF8** **Removes synchronous event from the queue**
Entry conditions: HL contains the event block address.
Exit conditions: AF, BC, DE and HL are modified.

169 **BCFB** **Processes the next event in the queue**
Entry conditions: none
Exit conditions: If there is an event to process, the carry flag is true and HL contains the address of the event block. A contains the priority code of the previous event.
If there is no event to process, the carry flag is false.
In both cases AF, DE and HL are modified.

170 **BCFE** **Processes an event routine**
Entry conditions: HL contains the address of an event block.
Exit conditions: AF, BC, DE and HL are modified.

171 **BD01** **Ends the processing of an event**
Entry conditions: HL contains the address of an event block.
A contains the priority code of the preceding event.
Exit conditions: AF, BC, DE and HL are modified.

172 **BD04** **Disables normal synchronous events**
Entry conditions: none
Exit conditions: HL is modified.

173 **BD07** **Enables normal synchronous events**
Entry conditions: none
Exit conditions: HL is modified.

174 **BD0A** **Inhibits a specified event**
Entry conditions: HL contains the address of the event block.
Exit conditions: AF is modified.

175 **BD0D** **Returns elapsed time in 300ths of a second**
Entry conditions: none
Exit conditions: DEHL contains the elapsed time as a 4-byte value.

General and peripheral interface routines

176 **BD10** **Sets the elapsed time counter**
Entry conditions: DEHL contains the 4-byte value in 300ths of a second. *Exit conditions:* AF is modified.

177 **BD13** **Loads a program into RAM and runs it**
Entry conditions: HL contains the address of the routine to call to load the program.
Exit conditions: program dependent.

178 **BD16** **Runs a program in a foreground ROM**
Entry conditions: HL contains the entry point.
C contains the ROM selection.
Exit conditions: indeterminate.

179 **BD19** **Waits for the CRT to generate a frame sync signal**
Entry conditions: none
Exit conditions: none

180 **BD1C** **Sets the screen mode**
Entry conditions: A contains the mode (0, 1 or 2).
Exit conditions: AF is modified.

181 **BD1F** **Sets the screen memory offset**
Entry conditions: A contains the base address of the new screen.
HL contains the offset.
Exit conditions: AF is modified.

182 **BD22** **Sets all INKs to the same colour to give the impression of clearing the screen**
Entry conditions: DE contains the address of an ink vector.
Exit conditions: AF is modified.

183 **BD25** **Sets the INK and BORDER colours**
Entry conditions: DE contains the address of an ink vector.
Exit conditions: AF is modified.

184 **BD28** **Reinitialises printer output**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

185 **BD2B** **Sends a character to the printer (and detects unusually long printer BUSY signals)**
Entry conditions: A contains the character to send.
Exit conditions: If the character has been sent, the carry flag is true.
If the printer has been busy for too long, the carry flag goes false.
In either case AF is modified.

186 **BD2E** **Tests whether the printer is busy**
Entry conditions: none
Exit conditions: If the printer is busy, the carry flag is set true, otherwise it is false.

187 **BD31** **Sends a character to the printer (which must not be busy)**
Entry conditions: A contains the character to be sent.
Exit conditions: carry flag true, AF modified.

188 **BD34** **Sends data to a PSG register**
Entry conditions: A contains the register number.
C contains the data.
Exit conditions: AF and BC are modified.

The Jump Block

189 **BD37** **Resets standard jump blocks**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

INDIRECTION VECTORS

Indirection vectors allow the user to intercept and alter a certain number of actions of the software system without having to rewrite the entire system.

Note:
The following addresses are not entry points but internal calls which can be trapped.

1 **BDCD** **Enables screen cursor**
Entry conditions: none
Exit conditions: AF is modified.

2 **BDDO** **Disables screen cursor**
Entry conditions: none
Exit conditions: AF is modified.

3 **BDD3** **Writes a character to the screen**
Entry conditions: A contains the character to be written.
H contains the column number.
L contains the row number.

4 **BDD6** **Reads a screen character**
Entry conditions: H contains the column number.
L contains the row number.
Exit conditions: if the character is found,then carry flag is true and A contains the character. Otherwise the carry flag is false and A contains 0. In both cases AF, BC, DE and HL are modified.

5 **BDD9** **Writes a character or interprets a control code**
Entry conditions: A contains the character or control code number.
Exit conditions: AF, BC, DE and HL are modified.

6 **BDDC** **Draws a pixel**
Entry conditions: DE contains the X co-ordinate of the pixel.
HL contains the Y co-ordinate.
Exit conditions: AF, BC, DE and HL are modified.

7 **BDDF** **Tests a pixel**
Entry conditions: DE contains the X co-ordinate of the pixel.
HL contains the Y co-ordinate.
Exit conditions: A contains the INK value of the specified pixel. A, BC, DE and HL are modified.

8 **BDE2** **Draws a line from the current position**
Entry conditions: DE contains the X co-ordinate of the end pixel.
HL contains the Y co-ordinate of the end pixel.
Exit conditions: AF, BC, DE and HL are modified.

9 **BDE5** **Reads a pixel in screen memory and decodes its INK colour**
Entry conditions: HL contains the screen address of the pixel.
C contains the pixel mask.
Exit conditions: A contains the decoded INK value of the specified pixel. AF is modified.

10 **BDE8** **Writes one or more pixels in the current graphic mode.**
Entry conditions: HL contains the screen address of the pixel or pixels.
C contains the mask for the pixel or pixels.
B contains the INK code.
Exit conditions: AF is modified.

11 **BDEB** **Clears the screen with INK 0**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

12 **BDEE** **Tests the ESC key (BREAK)**
Entry conditions: interrupts disbaled. C contains the state of the CTRL and SHIFT keys.
Exit conditions: AF and HL are modified.

13 **BDF1** **Writes a character to the printer**
Entry conditions: A contains the character.
Exit conditions: if the character has been correctly written, then the carry flag is true.
If the printer has been busy too long the carry flag goes false.
In either case, AF and BC are modified.

KERNEL VECTORS AND RESTARTS

A series of routines are used to control the selection and state of the ROM, these lie outside the principal entry points of the system software and should not be modified by the user.

Upper memory vectors

1 **B900** **Selects the upper ROM**
Entry conditions: none
Exit conditions: A contains the previous state of ROM.
AF is modified.

2 **B903** **Disables the upper ROM to reselect RAM**
Entry conditions: none
Exit conditions: A contains the previous state of ROM.
AF is modified.

3 **B906** **Selects the lower ROM**
Entry conditions: none
Exit conditions: A contains the previous state of ROM.
AF is modified.

4 **B909** **Disables the lower ROM to reselect RAM**
Entry conditions: none
Exit conditions: A contains the previous state of ROM.
AF is modified.

5 **B90C** **Restores the former state of a ROM**
Entry conditions: A contains the former state of the ROM.
Exit conditions: AF is modified.

6 **B90F** **Selects a specified upper ROM**
Entry conditions: C contains the select address of the required ROM.
Exit conditions: C contains the select address of the previous ROM.
B contains the state of the previous ROM.
AF is modified.

7 **B912** **Determines a ROM select address**
Entry conditions: none
Exit conditions: Contains the select address of the current ROM.

8 **B915** **Determines the type and the version number of a ROM**
Entry conditions: C contains the select address of the ROM to be examined.
Exit conditions: A contains a ROM class.
H contains a version number.
L contains a type number.
B and F are modified.

9 **B918** **Reselects a previously selected upper ROM**
Entry conditions: C contains the select address of the ROM to be selected.
B contains its state.
Exit conditions: BC is modified.

10 **B91B** **Executes a block memory transfer with increment (LDIR) with both upper and lower ROMs disabled.**
Entry conditions: BC, DE and HL are programmed as for a normal LDIR.
Exit conditions: BC, DE, HL and F are in the same state as after a normal LDIR.

11 **B91E** **As above, but with decrement (LDDR).**

12 B921 **Tests for the existence of a higher priority event than the current event**
Entry conditions: none
Exit conditions: if an event with a higher priority is pending, then the carry flag will be true, otherwise it will be false. AF is modified.

Low memory vectors

1 0000 **RST 0**
Cold boot (as at power-up)
Entry conditions: none
Exit conditions: not relevant

2 0008 **RST 1**
Jump to of a routine in ROM or in lower RAM.
The two bytes following the RST contain the execution address.
If set, bits 15 and 14 disable upper and lower ROMs respectively.
Entry conditions: all registers are passed on to the routine without alteration.
Exit conditions: depends on the routine.

3 000B **Jump to a routine in ROM or low RAM**
Entry conditions: HL contains the lower address of the routine.
Exit conditions: depends on the routine.

4 000E **Jumps to the address contained in BC**
Entry conditions: BC contains the address.
Exit conditions: depends on the routine.

5 0010 **RST 2**
Sub-routine call to a secondary ROM.
The two bytes following the RST contain the execution address to which &C000 is automatically added and the selection address of the ROM. See p.101.
Entry conditions: all registers except IY are passed unaltered to the routine.
Exit conditions: depends on the routine.

6 0013 **Sub-routine call to a secondary ROM**
The address is contained in HL.
Entry conditions: HL contains the address and all registers except IY are passed unaltered to the routine.
Exit conditions: depends on the routine.

7 0016 **Jumps to the address contained in DE**
Entry conditions: DE contains the address.
Exit conditions: depends on the routine.

8 0018 **RST 3**
Call to a sub-routine in RAM or ROM.
The two bytes immediately following the call contain the address of the far-address of the sub-routine. See p.101.
Entry conditions: all registers except IY are passed on to the sub-routine.
Exit conditions: depends on sub-routine.

9 001B **Call to a sub-routine in RAM or ROM with the address in HL**
Entry conditions: HL contains the address.
C contains the selection byte of the ROM or RAM.
All the registers are passed on to the routine, except IY.
Exit conditions: depends on the routine.

10 001E **Jumps to the address contained in HL**
Entry conditions: HL contains the address.
Exit conditions: depends on the routine.

11 0020 **RST 4**
Loads the byte in RAM pointed to by HL into the accumulator, regardless of the state of the ROM.
Entry conditions: HL contains the address.
Exit conditions: A contains the value read.

12 0023 **Calls a sub-routine in RAM or ROM**
Entry conditions: HL contains the address where the far-address of the sub-routine is held. All the registers are passed on to the sub-routine except IY. See p.101.
Exit conditions: depends on the sub-routine.

13 0028 **RST 5**
Jumps to an address in the lower ROM. The two bytes following the RST contain the address.
Entry conditions: all the registers are preserved.
Exit conditions: depends on the sub-routine.

14 0030 **RST 6**
User-definable reset jump.
Bytes 30 to 37 inclusive are available to the user for any purpose.

15 0038 **RST 7**
Entry point for system generated interrupts.
Entry conditions: none
Exit conditions: all registers are preserved.

16 003B **External interrupt handling routine**
Entry conditions: none
Exit conditions: AF, BC, DE and HL are modified.

VECTORS FOR MATHS ROUTINES

The maths routines are contained in the lower ROM and are regularly called by the BASIC ROM in order to carry out the BASIC calculation functions (+, *, /, sine, cosine and so on).

A series of vectors has been created to facilitate use of these calls.

The BASIC maths functions use a virtual accumulator of six bytes located at B0C1 to B0C6. B0C1 contains 2 if the variable is an integer, 3 if it is a string, 5 if it is a real.

An integer variable is stored in two bytes in signed binary format.

A real variable is more complex. It is stored in five bytes according to the following formula:

Step 1
Express the number in binary.

Step 2
Count the number of significant bits before the decimal point and add 128 (80 hex) to it to get the fifth byte.

Step 3
Delete the left-most bit and convert the seven remaining bits into decimal. If the number is negative, add 128 (80 hex). This gives the fourth byte.

Step 4
To obtain bytes 3, 2 and 1, take the remaining bits in groups of 8 and convert them into decimals.

Example:
Coding the real variable −2527

2527 in binary is 1001 1101 1111 (12 digits)

byte 5: 128 + 12 = 140 = 8C

byte 4: take the next seven bits: 0011101 = 29 = 1D
Since the number is negative, add 128: 29 + 128 = 157 = 9D

byte 3: the eight following bits are 1111 0000 = 240 = F0

bytes 2 and byte 1: = 00 since there are no further bits.

−2527 is therefore stored as 00 00 F0 9D 8C in hexadecimal.

Vector address	Absolute address	Purpose
BD3D	2E18	Copies the five bytes pointed to by DE into the area pointed to by HL and transfers the content of the byte located at address HL − 1 (the variable type) into A.
BD40	2E29	Copies the contents of A into the five bytes pointed to by DE.
BD43	2E55	Conversion of the binary number pointed to by HL into a format suitable for use in the 5 byte virtual accumulator.
BD46	2E66	Transforms the value contained in the 5 bytes pointed to by HL into an integer in HL.
BD49	2E8E	Transforms the value contained in the 5 bytes pointed to by HL into an integer, then places this in the first two bytes pointed to by HL.
BD4C	2EA1	Performs the FIX function.
BD4F	2EAC	Performs the INT function.
BD52	2EB6	Routine used by STR$ and PRINT.
BD55	2F1D	Transformation routine.
BD58	333F	Addition of two reals. HL points to 5 bytes representing a number in real format (called ACCUM1), DE points to another five bytes (called ACCUM2). On completion of the routine, HL points to ACCUM1 which contains the sum of ACCUM1 + ACCUM2.
BD5B	3337	Subtraction of two reals. HL points to 5 bytes representing a number in real format (called ACCUM1). DE points to another five bytes (called ACCUM2). On completion of the routine, HL points to ACCUM1 which contains the value of ACCUM1 − ACCUM2.
BD5E	333B	Subtraction of two reals. As above, but ACCUM1 contains the value of ACCUM2 − ACCUM1.
BD61	4315	Multiplication of two reals. As above, but ACCUM1 contains the value of ACCUM1 * ACCUM2.
BD64	349E	Division of two reals. As above, but ACCUM1 contains the value of ACCUM1/ACCUM2.
BD67	3578	Adds A to the last byte of the number pointed to by HL.
BD6A	359A	Comparison of two reals. If ACCUM1>ACCUM2, then A = 1 If ACCUM1<ACCUM2, then A = 255 If ACCUM1 = ACCUM2, then A = 0

Vector address	Absolute address	Purpose
BD6D	359A	Negation of a real number. HL points to ACCUM1 which contains the value of −ACCUM1.
BD70	35E8	Tests the real contained in ACCUM1. HL points to ACCUM1. If ACCUM1>0, then A = 1 If ACCUM1<0, then A = 255 If ACCUM1 = 0, then A = 0
BD73	31AE	Sets trig-calculation mode to degrees or radians. If A = 0, mode is RADIAN, if A does not equal 0 then DEGREE mode is selected.
BD76	31A3	Places the constant value PI in the area pointed to by HL on entry.
BD79	310A	Extraction of the square root of a real number. On entry, HL points to the 5 bytes containing the number. On exit, the same bytes will contain the square root of the number.
BD7C	310D	Raise a real number to a power. HL points to ACCUM1 which contains the number and DE points to ACCUM2 which contains the power. On exit, ACCUM1 contains the value of ACCUM1 to the power ACCUM2.
BD7F	3014	Calculation of the napierian logarithm (to base *e*) of a real number. HL points to ACCUM1 which contains the entry number. On exit, ACCUM1 contains the value of the logarithm.
BD82	300F	Calculation of the common logarithm (to base 10) of a real number. HL points to ACCUM1 which contains the number. On exit, ACCUM1 contains the value of the logarithm.
BD85	3090	Calculation of the exponent of a number. HL points to ACCUM1 which, on exit, contains the value of the number's exponent.
BD88	31BC	Calculation of the sine of an angle.
BD8B	31B2	Calculation of the cosine of an angle.
BD8E	3231	Calculation of the tangent of an angle.
BD91	3241	Calculation of the arc-tangent of an angle.
BD94	2E5E	Evaluation routine.
BD97	2F94	Routine to load B8E4 and B8E6 on initialisation.
BD9A	2FA1	Routine used during random number generation.

Vector address	Absolute address	Purpose
BD9D	2FB7	Routine used during random number generation.
BDA0	2FE6	Routine used during random number generation.
BDA3	3708	Manipulation using HL.
BDA6	370E	Loads B and E with 0, loads C with 2.
BDA9	3715	Manipulation using HL.
BDAC	3728	Addition of two integer numbers. HL = HL + DE. A = FF in the case of an overflow.
BDAF	3731	Subtraction of two integer numbers. HL = HL − DE. A = FF in the case of an overflow.
BDB2	3730	Subtraction of two integer numbers. HL = DE − HL. A = FF in the case of an overflow.
BDB5	3739	Multiplication of two integer numbers. HL = HL*DE. A = FF in the case of an overflow.
BDB8	377A	Division of two integer numbers. HL = HL/DE. DE contains the remainder of the division on exit.
BDBB	3781	Remainder of the division of two integers. HL = remainder of HL/DE.
BDBE	3750	A particularly obscure operation using HL and DE.
BDC1	378C	Routine used during the PRINT instruction.
BDC4	37E9	Comparison of two integer numbers. If HL>DE then A = 1 If HL<DE then A = FF If HL = DE then A = 0
BDC7	37D4	Negation of an integer number. On exit, HL = − (HL).
BDCA	37E0	Tests HL. If HL>0 then A = 1 If HL<0 then A = 255 If HL = 0 then A = 0

MAIN SYSTEM VARIABLES

Address	Length	Contents
AC00	26	Code C9 (RET) repeated 26 times.
AC1C	1	AUTO flag: 0 = auto enabled, 1 = auto disabled.
AC1D	2	Number of the current line (used by AUTO).
AC1F	2	Value of the increment between line numbers (AUTO).
AC24	1	Used by WIDTH instruction.
AC26	2	Used by NEXT instruction.
AC2C	2	Used by FOR instruction.
AC2E	2	Used by WHILE..WEND instruction pairs.
AC30	11	Used by ON..GOTO instruction.
ACA4	1	Used by EVERY instruction.
ACA5	256	Keyboard input buffer.
AD81	2	Line number for ON ERROR instruction.
ADA6	2	Pointer for RESUME instruction.
ADA8	2	Used by error-handling routine.
ADAA	1	Error number.
ADAB	2	Address of last byte executed.
ADAD	2	Address for END, STOP and CONT.
ADB1	1	Error number for ON ERROR GOTO function.
ADB2	9	Parameters used by SOUND instruction.
AE0C	26	Variable type declaration table. Consists of 26 bytes (1 for each letter of the alphabet). Each byte contains a code determining the default variable type of each variable beginning with the letter.
AE2E	2	Address of current line for READ DATA.
AE30	2	Address at which READing of DATA starts, used with RESTORE.
AE34	2	Used by ON ERROR GOTO.
AE38	1	TRACE flag: 0 = TROFF, 1 = TRON.
AE72	2	Temporary store of DE for use by CALL instruction.
AE74	1	Temporary store of accumulator for use by CALL instruction.
AE75	2	Temporary store of HL for use by CALL instruction.
AE77	2	Temporary store of SP for use by CALL instruction.
AE79	2	Used by ZONE instruction (address).
AE7B	2	HIMEM (upper address limit for BASIC).
AE7D	2	Used by SYMBOL instruction (address).
AE81	2	Address of start of BASIC program (defaults to 016F).
AE83	2	Address of end of BASIC program.
AE85	2	Address of start of variable table.
AE87	2	Address of simple variables table.
AE89	2	Address of array variables table.
B0BA	1	Key pressed flag (used by INKEY).
B0C1	1	State of virtual accumulator.

Address	Length	Contents
B0C2	5	Five bytes used by the accumulator.
B1C7	1	INK mask byte.
B1C8	1	Screen mode (0, 1 or 2).
B1C9	2	Screen offset (values from 0 to 7FF).
B1CB	1	High byte of start of real screen memory.
B1CC	1	Sometimes contains a C3 (jump).
B1CD	2	Contains a jump address.
B1D7	1	Length of first period of flashing of border.
B1D8	1	Length of second period of flashing of border.
B1DA	32	INK colours (two bytes per colour).
B1FC	1	Used by border.
B20C	1	STREAM number.
B285	1	Current cursor row.
B286	1	Current cursor column.
B287	1	Window flag.
B288	1	Start row of current window.
B289	1	Start column of current window.
B28A	1	Last row of current window.
B28B	1	Last column of current window.
B28D	1	Cursor flag: 0 = cursor enabled, 255 = cursor disabled.
B28E	1	Display flag: 0 = display disabled, 255 = display enabled.
B28F	1	Current foreground INK value.
B290	1	Current background (PAPER) INK value.
B291	1	Background display flag: 0 = background display enabled, 255 = background display disabled.
B294	2	First character in, and state of, user-defined character matrix table.
B296	2	Address of user-defined character matrix table.
B2C3	96	Table of control codes.
B328	2	Coordinate of origin of X axis.
B32A	2	Co-ordinate of Y axis.
B32C	2	Graphic X co-ordinate.
B32E	2	Graphic Y co-ordinate.
B330	2	X co-ordinate of one edge of graphic window.
B332	2	X co-ordinate of the other edge of graphic window.
B334	2	Y co-ordinate of one edge of graphic window.
B336	2	Y co-ordinate of the other edge of graphic window.
B338	1	Graphic foreground INK colour.
B339	1	Graphic background INK colour.
B33A	8	Four sets of two bytes used as temporary store during line drawing.
B342	2	X co-ordinate of end-point for line drawing.
B344	2	Y co-ordinate of end-point for line drawing.
B34C	80	Table of key values when used without SHIFT or CTRL.

Address	Length	Contents
B39C	80	Table of SHIFTed key values.
B3EC	80	Table of key values when used with with CTRL.
B43C	80	Table of repeat values for each key.
B4DE	2	Used for scanning (address).
B4E0	1	Temporary store of scanned character (BB0C).
B4E9	1	Value of auto-repeat speed for all keys.
B4EA	1	Value of delay before a key repeat.
B4F1	1	State of joystick 1.
B4F4	1	State of joystick 2.
B50C	1	Used for BREAK control.
B541	2	Address of key table when used without SHIFT or CTRL.
B543	2	Address of SHIFTed key table.
B545	2	Address of key table when used with CTRL.
B547	2	Address of key repeat data table.
B551		Start of sound control variables area.
B60A	240	15 groups of 16 bytes containing values for amplitude envelopes.
B6FA	240	15 groups of 16 bytes containing values for tone envelopes.
B800	1	Tape prompt flag: prompt message enabled if 0, disabled if not 0.
B802	1	File open flag.
B803	2	Address of 2K directory buffer.
B805	2	Address of read buffer.
B819	1	File status.
B81A	2	Current address of read buffer.
B81C	2	Address of data memory area.
B81F	2	Logical length of file.
B847	1	Status of write stream.
B84A	2	Address of write buffer.
B85F	2	Current address of write buffer.
B8CD	1	Synchronisation character.
B8D1	2	Read and write speed.
B8F7	1	Radian/Degree flag: 0 = RADIANS, 255 = DEGREES mode.

PRINCIPAL LOWER ROM ADDRESSES

The lower ROM contains the system routines, the maths routines and the character generator.

Note:
Addresses marked with a * are described in detail in other sections of this book.

005C	BCC8 *
0099	BD0D *
00A3	BD10 *
0163	BCD7 *
016A	BCDA *
0170	BCDD *
0176	BCE0 *
017D	BCE3 *
0183	BCE6 *
01B3	BCE9 *
01C5	BCEC *
01D2	BCEF *
01E2	BCF2 *
021A	BCFE *
0228	BCF5 *
0256	BCFB *
0277	BD01 *
0285	BCF8 *
028E	BD0A *
0295	BD04 *
029B	BD07 *
02A1	BCD1 *
02B2	BCD4 *
0329	BCCB *
0332	BCCE *
05DC	BD13 *
060B	BD16 *
066D	64K MICROCOMPUTER (V1) *or* 128K...*(6128)*, *(message)*.
068A	Copyright 1984 Amstrad Consumer Electronics PLC and Locomotive Software Ltd. *(message)*
06F4	*** program load failed *** *(message)*.
0727	List of compatibles Arnold, Amstrad, Orion, Schneider, Awa, Solavox, Saisho, Triumph, Isp.
0776	BD1C *
0786	BD22 *
0799	BD25 *
07BA	BD19 *
07C6	BD1F *
07E6	BD28 *
0782	BD2B *
07F8	BDF1 *
0807	BD31 *
081B	BD2E *
0826	BD34 *
0888	BD37 *
0AA0	BBFF *
0AB1	BC02 *
0ACA	BC0E *
0AEC	BC11 *
0AF7	BC14 *
0AF7	BDEB *
0B3C	BC05 *
0B45	BC08 *
0B50	BC0B *
0B57	BC17 *
0B64	BC1A *
0BA9	BC1D *
0BF9	BC20 *
0C05	BC23 *
0C13	BC26 *
0C2D	BC29 *
0C49	BC59 *

0C68 BDE8 *
0C6B BC5C *
0C82 BDE5 *
0C86 BC2C *
0CA0 BC2F *
0CE4 BC3E *
0CE8 BC41 *
0CEC BC32 *
0CF1 BC38 *
0D14 BC35 *
0D19 BC3B *
0DB3 BC44 *
0DB7 BC47 *
0DDF BC4A *
0DFA BC4D *
0E3E BC50 *
0EF3 BC53 *
0F49 BC56 *
0FC4 BC5F *
102F BC62 *
1078 BB4E *
1088 BB51 *
10E8 BBB4 *
1107 BBB7 *
115E BB6F *
1169 BB72 *
1174 BB75 *
1180 BB78 *
11CE BB87 *
120C BB66 *
1256 BB69 *
1263 BDCD *
1263 BDD0 *
1268 BB8A *
1268 BB8D *
1279 BB81 *
1281 BB84 *
1289 BB7B *
129A BB7E *
12A9 BB90 *
12AE BB96 *
12BD BB93 *
12C3 BB99 *
12C9 BB9C *
12D3 BBA5 *
12F1 BBA8 *
12FD BBAB *
132A BBAE *
1334 BB5D *
134A BDD3 *
137A BB9F *
1387 BBA2 *
13A7 BB63 *
13AB BB60 *
13C0 BDD6 *
1400 BB5A *
140C BDD9 *
144B BB57 *
1451 BB54 *
146B Table of terminal control codes (96 bytes).
14CB BBB1 *
1540 BB6C *
15B0 BBBA *
15DF BBBD *
15F1 BBC3 *
15F4 BBC0 *
15FC BBC6 *
1604 BBC9 *
1612 BBCC *
1734 BBCF *
1779 BBD2 *
17A6 BBD5 *
17BC BBD8 *
17C5 BBDB *
17F6 BBDE *
17FD BBE4 *
1804 BBE1 *
180A BBE7 *
1810 BBED *
1813 BBEA *
1816 BDDC *
1824 BBF3 *
1827 BBF0 *
182A BDD0 *
182A BDDF *
1836 BBF9 *
1839 BBF6 *
183C BDE2 *
1945 BBFC *
19E0 BB00 *
1A1E BB03 *

1A3C BB06 *
1A42 BB09 *
1A77 BB0C *
1A7B BB15 *
1AB3 Default values of extended keys (RUN for CTRL CR).
1ABD BB0F *
1B2E BB12 *
1B56 BB18 *
1B5C BB1B *
1BB3 BB21 *
1C2F BDEE *
1C5C BB24 *
1C6D BB3F *
1C69 BB42 *
1C71 BB45 *
1C82 BB48 *
1C90 BB4B *
1CA6 BB3C *
1CAB BB39 *
1CBD BB1E *
1D52 BB27 *
1D3E BB2A *
1D57 BB2D *
1D43 BB30 *
1D5C BB33 *
1D48 BB36 *
1D69 Table of default values of keyboard keys.
1E68 BCA7 *
1ECB BCB6 *
1EE6 BCB9 *
1F9F BCAA *
204A BCB3 *
206C BCAD *
2089 BCB0 *
2338 BCBC *
233D BCBF *
2349 BCC2 *
234E BCC5 *
2370 BC65 *
237F BC68 *
238E BC6B *
2392 BC77 *
23AB BC8C *
23FC BC7A *
2401 BC7D *
2415 BC8F *
242E BC92 *
2435 BC80 *
245B BC95 *
2496 BC89 *
249A BC86 *
24AB BC83 *
24EA BC98 *
2528 BC9B *
27C5 Press play then any key (*message*).
27DB Error (*message*).
27E5 REC (*message*).
27E8 And (*message*).
27ED Read (*message*).
27F3 Write (*message*).
27FA Rewind (*message*).
2800 Tape (*message*).
2805 Found (*message*).
280D Loading (*message*).
2815 Saving (*message*).
281D OK (*message*).
2820 Block (*message*).
2826 Unnamed (*message*).
282D File (*message*).
2836 BCA1 *
283F BC9E *
2851 BCA4 *
2A4B BC6E *
2A4F BC71 *
2A51 BC74 *
2E18 BD3D *
2E29 BD40 *
2E55 BD43 *
2E5E BD94 *
2E66 BD46 *
2E8E BD49 *
2EA1 BD4C *
2EAC BD4F *
2EB6 BD52 *
2F10 BD55 *
2F53 Table of powers of 10 (13 sets of 5 bytes for values 10 to 1013).
2F94 BD97 *

2FA1	BD9A *
2FB7	BD9D *
2FE6	BDA0 *
300F	BD82 * LOG10
3014	BD7F * LOG
3086	Coded value of LOG(2) (0.693147181)
308C	Coded value of LOG10(2) (0.301029996)
3090	BD85 * EXP
30CC	Coded 0.5 (*constant*)
30FB	1.44269504 (*constant*).
3100	88.0296919 (*constant*).
3105	−88.7228391 (*constant*).
310A	BD79 * SQR
310D	BD7C *
31A3	BD76 * PI
31A9	PI (3.14159265) (*constant*).
31AE	BD73 * DEG−RAD
31B2	BD8B * COS
31BC	BD88 * SIN
31EC	Table of 6 numbers, each of 5 bytes, for calculating sines and cosines.
321D	Table of 4 numbers, each of 5 bytes, for calculating sines and cosines.
3231	BD8E * TAN
3241	BD91 * ATN
3258	Table of 11 numbers, each of 5 bytes, for calculating arc-tangent.
3337	BD5B *
333B	BD5E *
333F	BD58 *
3415	BD61 *
349E	BD64 *
3578	BD67 *
359A	BD6A *
35E8	BD70 *
35F8	BD6D *
3708	BDA3 *
370E	BDA6 *
3715	BDA9 *
3728	BDAC *
3730	BDB2 *
3731	BDAF *
3739	BDB5 *
3750	BDBE *
377A	BDB8 *
3781	BDBB *
378C	BDC1 *
37D4	BDC7 *
37E0	BDCA *
37E9	BDC4 *
3800	Start of character generator table (256 groups of 8 bytes).
3FFF	End of table.

PRINCIPAL UPPER ROM ADDRESSES

The upper ROM contains the BASIC keyword handling routines.

C002 Initialisation and output of BASIC 1.0 (*message*)
C03F BASIC 1.0 (*message*)
C053 EDIT function
C090 Main entry (READY display)
C0CC READY (*message*)
C0DF AUTO
C12B NEW
C132 CLEAR
C20A PAPER
C212 PEN
C221 BORDER
C22A INK
C24F MODE
C25A CLS
C262 VPOS
C276 POS
C2D2 LOCATE
C2E1 WINDOW
C319 TAG
C320 TAGOFF
C337 Displays message pointed to by HL
C3E3 WIDTH
C417 EOF
C48C ORIGIN
C4B5 CLG
C4C6 DRAW
C4CB DRAWR
C4D0 PLOT
C4D5 PLOTR
C4E9 TEST
C4EE TESTR
C505 MOVE
C50A MOVER
C529 FOR
C5FB NEXT
C6C7 IF
C6E8 GOTO
C6ED GOSUB
C70F RETURN
C747 WHILE
C776 WEND
C7C3 ON
C8CB ON BREAK
C8E1 DI
C8E7 EI
C940 ON SQ
C971 AFTER
C979 EVERY
C99F REMAIN
CA8F ERROR
CB23 UNDEFINED LINE (*message*)
CB33 Routine to send 'BREAK IN' message
CB4F BREAK (*message*)
CB55 IN (*message*)
CB5A STOP
CB65 END
CBC0 CONT
CBF8 ON ERROR
CC03 RESUME
CC5B Table of error messages
CE66 End of table of error messages
CF81 Table of entry points for arithmetic and logic operations
D0CA Table of entry points for the functions EOF, ERR, HIMEM, INKEY$, PI, RND, TIME, XPOS and YPOS
D0DC ERR
D0F4 HIMEM
D107 XPOS
D10E YPOS
D190 Table of entry points for functions
D219 ROUND
D1EA MIN
D1EE MAX

D256 OPENOUT
D25F OPENIN
D298 CLOSEIN
D2A1 CLOSEOUT
D2C0 SOUND
D31E RELEASE
D329 SQ
D34E ENV
D385 ENT
D409 INKEY
D423 JOY
D439 KEY DEF
D494 SPEED
D4DB PI
D4E7 DEG
D4EB RAD
D4EF SQR
D4F4 Routine for raising to a power
D520 EXP
D525 LOG10
D52A LOG
D52F SIN
D534 COS
D539 TAN
D53E ATN
D543 RANDOM NUMBER SEED ? (*message*)
D559 RANDOMIZE
D584 RND
D614 DEFSTR
D618 DEFINT
D61C DEFREAL
D654 LET
D67D DIM
D9C0 ERASE
DAF8 LINE
DB28 INPUT
DB77 ? redo from start (*message*)
DCD9 RESTORE
DCEB READ
DDEZ TRON
DDEG TROFF
DE01 Table of entry points for BASIC keywords
DEBA End of table
DFDC Table of keywords which need to be followed by a line number (GOTO, RESTORE, AUTO, EDIT, etc.)
E0F7 LIST
E2DD Routine to search for keywords in table
E327 Routine to check whether a keyword is in the keyword table
E354 Table of addresses for each of the 26 letters of the alphabet
E388 Table of keywords and keyword codes
E64A End of table
E728 DELETE
E7DF RENUM
E8EF DATA
E8F3 REM
E9BD RUN
E9F6 LOAD
EA3C CHAIN
EAA6 MERGE
EC09 SAVE
F158 PEEK
F15F POKE
F16D INP
F177 OUT
F17D WAIT
F1BA CALL
F1F6 ZONE
F1FD PRINT
F2C4 PRINT USING
F47B WRITE
F4EF MEMORY
F69D SYMBOL
F834 LOWER$
F839 Routine for conversion into lower case
F842 UPPER$
F8BA BIN$
F8C4 HEX$
F8EA DEC$
F91E STR$
F93C LEFT$
F943 RIGHT$

F993 MID$
FA0A LEN
FA10 ASC
FA16 CHR$
FA24 INKEY$
FA36 STRING$
FA57 SPACE$
FA77 VAL
FAA1 INSTR
FC2D FRE
FCCC +
FCE1 −
FCF5 *
FD12 /
FD37 Integer division
FD49 MODULO (remainder after division)
FD58 logical AND function
FD63 logical OR function
FD6D logical XOR function (EXCLUSIVE OR)
FD85 ABS
FDE8 FIX
FDED INT
FE8D CINT
FEC2 UNT
FEEC CREAL
FEF3 Clear accumulator
FF02 SGN
FF0A Puts an integer into the accumulator
FF16 Conversion of an integer to a real
FF1D Puts variable type into C
FF23 Puts variable type into A
FF27 Tests whether the accumulator contains a string pointer
FF62 Copies the accumulator into the area pointed to by DE
FF71 Tests for a capital letter
FF7B Tests for a digit
FF8A Conversion into a capital letter
FFAA Compares A with the contents of HL
FFB8 Compares HL with DE
FFBE Compares HL with BC
FFC4 DE = HL − DE
FFCF HL = HL − DE
FFDA BC = HL − DE
FFE7 HL = HL − BC
FFF2 LDIR
FFF5 LDDR
FFF8 JP (HL)
FFF9 Return to address held in BC
FFFB Return to address held in DE

ROM ABSOLUTE ADDRESSES

Table of absolute jump addresses for vectors

Vector address	Absolute address	Vector Address	Absolute Address	Vector address	Absolute address
BB00	19E0	BB72	1169	BBE4	17FD
BB03	1A1E	BB75	1174	BBE7	180A
BB06	1A3C	BB78	1180	BBEA	1813
BB09	1A42	BB7B	1289	BBED	1810
BB0C	1A77	BB7E	129A	BBF0	1827
BB0F	1ABD	BB81	1279	BBF3	1824
BB12	1B2E	BB84	1281	BBF6	1839
BB15	1A7B	BB87	11CE	BBF9	1836
BB18	1B56	BB8A	1268	BBFC	1945
BB1B	1B5C	BB8D	1268	BBFF	0AA0
BB1E	1CBD	BB90	12A9	BC02	0AB1
BB21	1BB3	BB93	12BD	BC05	0B3C
BB24	1C5C	BB96	12AE	BC08	0B45
BB27	1D52	BB99	12C3	BC0B	0B50
BB2A	1D3E	BB9C	12C9	BC0E	0ACA
BB2D	1D57	BB9F	137A	BC11	0AEC
BB30	1D43	BBA2	1387	BC14	0AF7
BB33	1D5C	BBA5	12D3	BC17	0B57
BB36	1D48	BBA8	12F1	BC1A	0B64
BB39	1CAB	BBAB	12FD	BC1D	0BA9
BB3C	1CA6	BBAE	132A	BC20	0BF9
BB3F	1C6D	BBB1	14CB	BC23	0C05
BB42	1C69	BBB4	10E8	BC26	0C13
BB45	1C71	BBB7	1107	BC29	0C2D
BB48	1C82	BBBA	15B0	BC2C	0C86
BB4B	1C90	BBBD	15DF	BC2F	0CA0
BB4E	1078	BBC0	15F4	BC32	0CEC
BB51	1088	BBC3	15F1	BC35	0D14
BB54	1451	BBC6	15FC	BC38	0CF1
BB57	144B	BBC9	1604	BC3B	0D19
BB5A	1400	BBCC	1612	BC3E	0CE4
BB5D	1334	BBCF	1734	BC41	0CE8
BB60	13AB	BBD2	1779	BC44	0DB3
BB63	13A7	BBD5	17A6	BC47	0DB7
BB66	120C	BBD8	17BC	BC4A	0DDF
BB69	1256	BBDB	17C5	BC4D	0DFA
BB6C	1540	BBDE	17F6	BC50	0E3E
BB6F	115E	BBE1	1804	BC53	0EF3

Vector address	Absolute address	Vector Address	Absolute Address	Vector address	Absolute address
BC56	0F49	BCC5	234E	BD34	0826
BC59	0C49	BCC8	005C	BD37	0888
BC5C	0C6B	BCCB	0329	B900	BA5E
BC5F	0FC4	BCCE	0332	B903	BA68
BC62	102F	BCD1	02A1	B906	BA4A
BC65	2370	BCD4	02B2	B909	BA54
BC68	237F	BCD7	0163	B90C	BA72
BC6B	238E	BCDA	016A	B90F	BA7E
BC6E	2A4B	BCDD	0170	B912	BAA2
BC71	2A4F	BCE0	0176	B915	BA83
BC74	2A51	BCE3	017D	B918	BA8C
BC77	2392	BCE6	0183	B91B	BAA6
BC7A	23FC	BCE9	01B3	B91E	BAAC
BC7D	2401	BCEC	01C5	BDCD	1263
BC80	2435	BCEF	01D2	BDD0	1263
BC83	24AB	BCF2	01E2	BDD3	134A
BC86	249A	BCF5	0228	BDD6	13C0
BC89	2496	BCF8	0285	BDD9	140C
BC8C	23AB	BCFB	0256	BDDC	1816
BC8F	2415	BCFE	021A	BDDF	182A
BC92	242E	BD01	0277	BDE2	183C
BC95	245B	BD04	0295	BDE5	0C82
BC98	24EA	BD07	029B	BDE8	0C68
BC9B	2528	BD0A	028E	BDEB	0AF7
BC9E	283F	BD0D	0099	BDEE	1C2F
BCA1	2836	BD10	00A3	BDF1	07F8
BCA4	2851	BD13	05DC	0008	B982
BCA7	1E68	BD16	060B	000B	B97C
BCAA	1F9F	BD19	07BA	0010	BA16
BCAD	206C	BD1C	0776	0013	BA10
BCB0	2089	BD1F	07C6	0018	B9BF
BCB3	204A	BD22	0786	001B	B9B1
BCB6	1ECB	BD25	0799	0020	BACB
BCB9	1EE6	BD28	07E6	0023	B9B9
BCBC	2338	BD2B	07F2	0028	BA2E
BCBF	233D	BD2E	081B	0038	B939
BCC2	2349	BD31	0807		

EXECUTION ADDRESSES OF BASIC KEYWORDS

Keyword	Address	Keyword	Address
ABS	FD85	ERASE	D9C0
AFTER	C971	ERR	D0DC
ASC	FA10	ERROR	CA8F
ATN	D53E	EVERY	C979
AUTO	C0DF	EXP	D520
BIN$	F8BA	FIX	FDE8
BORDER	C221	FOR	C529
CALL	F1BA	FRE	FC2D
CAT	D246	GOSUB	C6ED
CHAIN	EA3C	GOTO	C6E8
CHR$	FA16	HEX$	F8C4
CINT	FE8D	HIMEM	D0F4
CLEAR	C132	IF	C6C7
CLG	C4B5	INSTR	FAA1
CLOSEIN	D298	INK	C22A
CLOSEOUT	D2A1	INKEY	D409
CLS	C25A	INKEY$	FA24
CONT	CBC0	INP	F16D
COS	D534	INPUT	DB2B
CREAL	FEEC	INT	FDED
DATA	E8EF	JOY	D423
DEC$	F8EA	KEY	D439
DEF	D417	LEFT$	F93C
DEFINT	D618	LEN	FA0A
DEFREAL	D61C	LET	D654
DEFSTR	D614	LINE	DAF8
DEG	D4E7	LIST	E0F7
DELETE	E728	LOAD	E9F6
DI	C8E1	LOCATE	C2D2
DIM	D67D	LOG	D52A
DRAW	C4C6	LOG10	D525
DRAWR	C4CB	LOWER$	F834
EDIT	C052	MAX	D1EE
EI	C8E7	MEMORY	F4EF
ELSE	E8F3	MERGE	EAA6
END	CB65	MID$	F993
ENT	D385	MIN	D1EA
ENV	D34E	MODE	C24F
EOF	C417	MOVE	C505

Keyword	Address
MOVER	C50A
NEXT	C5FB
NEW	C12B
ON	C7E3
ON BREAK	C8CB
ON ERROR	CBF8
ON SQ	C940
OPENIN	D25F
OPENOUT	D256
ORIGIN	C48C
OUT	F177
PAPER	C20A
PEEK	F158
PEN	C212
PI	D4DB
PLOT	C4D0
PLOTR	C4D5
POKE	F15F
POS	C276
PRINT	F1FD
'(REM)	E8F3
RAD	D4EB
RANDOMIZE	D559
READ	DCEB
RELEASE	D31E
REM	E8F3
REMAIN	C99F
RENUM	E7DF
RESTORE	DCD9
RESUME	CC03
RETURN	C70F
RIGHT$	F943
RND	D584
ROUND	D219
RUN	E9BD

Keyword	Address
SAVE	EC09
SGN	FF02
SIN	D52F
SOUND	D2C0
SPACE$	FA57
SPEED	D494
SQ	D329
SQR	D4EF
STOP	CB5A
STR$	F91E
STRING$	FA36
SYMBOL	F69D
TAG	C319
TAGOFF	C320
TAN	D539
TEST	C4E9
TESTR	C4EE
TIME	D0E5
TROFF	DDE6
TRON	DDE2
UNT	FEC2
UPPER$	F842
VAL	FA77
VPOS	C262
WAIT	F17D
WEND	C776
WHILE	C747
WIDTH	C3E3
WINDOW	C2E1
WRITE	F47B
XPOS	D107
YPOS	D10E
ZONE	F1F6

CONTROL BLOCKS

ROM expansion

byte	0	ROM TYPE
byte	1	MAKE
byte	2	VERSION
byte	3	LEVEL
byte	4	TABLE

Streams

byte	0	VIDEO
byte	1	CURSOR
byte	2	CURSOR POSITION
byte	3	WINDOW SIZE
byte	4	INK
byte	5	CHARACTER
byte	6	GRAPHIC

Sound queue

byte	0	LINK CHANNEL
byte	1	AMPLITUDE ENVELOPE
byte	2	TONE ENVELOPE
bytes	3 and 4	SOUND PERIOD
byte	5	NOISE PERIOD
byte	6	INITIAL AMPLITUDE
bytes	7 and 8	DURATION OF ENVELOPE

Amplitude and tone control block

byte	0	NUMBER OF SECTIONS
bytes	1, 2 and 3	FIRST SECTION
bytes	4, 5 and 6	SECOND SECTION
bytes	7, 8 and 9	THIRD SECTION
bytes	10, 11 and 12	FOURTH SECTION
bytes	13, 14 and 15	FIFTH SECTION

Ink vector

byte	0	BORDER COLOUR
byte	1	INK COLOUR 0
byte	2	INK COLOUR 1
and so on		
to...		
byte	16	INK COLOUR 15

Format of the two bytes following a RESTART

bit 15	X
bit 14	Y
bits 13 to 0	ADDRESS

Standard ROM

X = 0 UPPER ROM DESELECTED
X = 1 UPPER ROM SELECTED

Y = 1 LOWER ROM DESELECTED
Y = 0 LOWER ROM SELECTED

Additional (secondary) ROM

XY gives a value from 0 to 3, which, when added to the selection address of the main ROM, give the address of the secondary ROM.

Format of a Far-Address

Bytes 0,1 give the address of the routine to call.
Byte 2 as follows:

00-FB Select given ROM, enable upper, disable lower.
FC ROM unchanged, enable upper, enable lower.
FD ROM unchanged, enable upper, disable lower.
FE ROM unchanged, disable upper, enable lower.
FF ROM unchanged, disable upper, disable lower.

On return from the routine, ROM select and state are restored.

Format of cassette files

Complete block

MOTOR GAP	HEADER BLOCK	DATA

(the motor gap provides a period during which nothing is recorded, allowing time for the tape motor to come up to full operating speed)

The first and last blocks include an additional silent gap which provides separation between programs or files.

First block

MOTOR GAP	START GAP	HEADER	DATA

Second block

MOTOR GAP	HEADER	DATA	END GAP

Format of recording

LEADER	DATA BLOCK 1	DATA BLOCK 2	. .	DATA BLOCK *n*	TRAILER

1 data block = 256 bytes + 2 byte checksum (CRC)

Recording of header:	1 DATA BLOCK
Recording of data:	1 to 8 DATA BLOCKS (usually 8)
Leader:	2048 bits set to 1 followed by a bit set to 0 and a synchronising byte.
Trailer:	32 bits set to 1

Format of header

bytes 0 to 15	NAME OF FILE
byte 16	BLOCK NUMBER
byte 17	NOT ZERO IF LAST BLOCK
byte 18	FILE TYPE
bytes 19 and 20	LENGTH OF DATA
bytes 21 and 22	DESTINATION ADDRESS OF DATA
byte 23	NOT ZERO IF FIRST BLOCK
bytes 24 and 25	TOTAL LENGTH OF FILE IN BYTES
bytes 26 and 27	ENTRY POINT
bytes 28 to 63	NOT USED

Description of byte 18 (file type)

bit 0	1 if file is protected
bits 1 and 2	00 = BASIC 01 = BINARY 10 = SCREEN DUMP 11 = ASCII
bit 3	Not used
bits 4 to 7	Always set to 0 except in case of ASCII files when bit 4 is set to 1.

Event block

bytes 0 and 1	SYSTEM POINTER
byte 2	COUNTER
byte 3	CLASS
bytes 4 and 5	PROCESSING ROUTINE ADDRESS
byte 6	ROM SELECTION ADDRESS

Interrupt control block (normal)

bytes 0 and 1	SYSTEM POINTER
bytes 2 and 3	COUNTER. Interrupt takes place when zero.
bytes 4 and 5	RELOAD. Value of re-initialisation after reaching 0.
bytes 6...	EVENT BLOCK (*see above*).

Interrupt (rapid) and CRT control block

bytes 0 and 1	SYSTEM POINTER
bytes 2...	EVENT BLOCK (*see above*).

CHIPS AND CIRCUITS

THE AY3 8912 CHIP

Internal structure

The PSG (Programmable Sound Generator) is made up of the following elements:

Sound generators:
There are three independent generators, each producing a square wave whose frequency can be programmed. They are called CHANNELS A, B and C. They have no inherent priority.

White noise generator:
Produces a wide spectrum white noise.

Mixer:
Allows the mixing (combining) of outputs from the three sound generators and the white noise generator.

Amplitude control:
Selection of the output amplitude can be controlled in two ways. The first is by controlling the amplitude with the microprocessor itself (called the *fixed amplitude* mode). The second is by controlling the amplitude using the envelope generator (called the *variable amplitude* mode).

Envelope generator:
Produces an amplitude modulation envelope. It possesses eight envelope forms.

Digital to analog converters:
The three D/A converters produce signals at 16 possible levels determined by the amplitude control.

Input/Output port:
Not used in sound production (*see later*).

The PSG registers

There are 15 registers numbered R0 to R14.

In order to produce a sound, a combination of registers R0 to R13 must be loaded with data. Each parameter includes the noise component, the sound component, the frequency, the shape and the duration of the envelope.

Registers R0 to R5

The first three pairs of registers (R0–R1, R2–R3, R4–R5) are the frequency control registers for the three channels A, B and C.

Registers R1, R3 and R5 are the registers for coarse adjustment and only the least signficant (left-most) four bits are used. Registers R0, R2 and R4 are the registers for fine adjustment and all eight bits are used.

Thus the values loaded into R0, R2 and R4 take values between 0 and 255, while the values loaded into R1, R3 and R5 take values between 0 and 15.

The value used is determined by dividing 125000 by the required frequency in Hertz (cycles per second).

Register R6

Register R6 determines the frequency of the white noise generator; only the five least significant bits are used. The value of R6 therefore lies between 0 and 31. The same formula is used as for R0–R5 to determine the value to be used for a specific frequency.

Register R7

Register R7 controls the mixing of the three sound generators and the noise generator. R7 is also used in the control of I/O port (*see later*).

The next table summarises the effects of register R7.

BIT	set to 0	set to 1
7	not used	not used
6	Input port	Output port (unused)
5	White noise on channel C ON	White noise on channel C OFF
4	White noise on channel B ON	White noise on channel B OFF
3	White noise on channel A ON	White noise on channel A OFF
2	Sound on channel C ON	Sound on channel C OFF
1	Sound on channel B ON	Sound on channel B OFF
0	Sound on channel A ON	Sound on channel A OFF

Note:
Switching a channel OFF is not enough to stop its output, you must also place a 0 in the relevant amplitude control register (*see below*).

Example:
Imagine that you want sound on channel A without noise, sound on channel B with noise and noise only on channel C.

bit:	7	6	5	4	3	2	1	0	
value:	x	x	0	0	1	1	0	0	= 12

x = *state not important*

Thus you would write the value 12 into register 7 to obtain the required combination.

Registers R8 to R10

Registers R8 to R10 control the amplitudes of channels A, B and C. Only the four least significant bits are used, and therefore the possible values will all lie in the range 0–15.

A value of 0 sets the amplitude to its minimum value (no amplitude) and 15 corresponds to the maximum amplitude. The fifth bit (bit 4) selects the amplitude control mode. If set to 0, the amplitude will not vary. If set to 1, the amplitude is controlled by the envelope generator (*see below*).

Registers R11 and R12

These two registers control the period of the envelope. A calculation with a formula similar to that used for R0–R5 determines the value of R11 and R12:

value = 125000 * P/16
where P is the period of the envelope.

Register R13

Register R13 controls the form of the modulation used. If bit 4 in registers R8 to R10, is set to 1, then modulation takes place. Otherwise, the contents of register 13 are ignored.

Only the four least significant bits are used.

Bit 3	2	1	0	Envelope form		Possible values
0	0	x	x	A	A single cycle starts at maximum amplitude and decays to zero	0,1,2,3
0	1	x	x	B	A single cycle starts with zero amplitude and increases to its maximum value before dropping sharply back to zero,	4, 5, 6, 7
1	0	0	0	C	As A, but continually repeating	8
1	0	1	0	D	As C, but climbing more steeply to its maximum (steeper attack)	10
1	0	1	1	E	As A, but resets to maximum value at end	11
1	1	0	0	F	As B, but continually repeating	12
1	1	0	1	G	As B, but resets to maximum value at end	13
1	1	1	0	H	As F, but with steeper attack	14

Register 14

This register has nothing to do with sound production. It is an input/output port which deals with reading the keyboard and the joystick.

Bit 6 of register R7 controls the direction of transmission, but as the port is used exclusively for input, you need only set bit 6 of R7 to 0.

Programming the AY3 8912

The PSG is accessible through ports A and C of the PPI 8255 (*see next section*).

To simplify matters, routine 188 (at address BD34) can be used to write into the PSG registers. Reading the state of the keyboard and joysticks is, however, more difficult to perform directly and it is best to do this through the normal entry points described earlier.

If you want to program the PSG directly, the two command signals BDIR and BC1 are available at port C of the PPI 8255.

Function of BDIR and BC1

BDIR	BC1	Function
0	0	Inactive: no function
0	1	Reading: the contents of the current register are placed on the data bus D0–D7.
1	0	Writing: the data bus D0–D7 contains data to be written into the current register.
1	1	Writing: the data bus D0–D7 contains the number of the register which is to be used.

THE PPI 8255 CHIP

General

The PPI is an interface circuit designed for 8080 series microprocessors and is manufactured by INTEL under the name 8255A. It contains 24 input/output bits which can be programmed in two groups of 12 bits and which can be used in three principal modes.

In the first mode (mode 0), each 12-bit port can be programmed as 3 4-bit ports, allowing both input and output.

In the second mode (mode 1), each 12-bit port can be programmed with 8 bits used for input and output, and the remaining four bits for *handshaking* (transmission control).

The third mode (mode 2) allows 8 bits to be used as a bidirectional port with the remaining 5 bits used for handshaking.

The PPI also allows bits to be set directly to 0 or 1.

For the sake of simplicity, the PPI is considered to be divided into three 8-bit ports called port A, port B and port C.

Port C is divided into two 4-bit ports to form the 12-bit groups with A and B.

Allocation of ports

Port A – Input and Output

B0 to B7 Correspond to D0 up to D7 on AY3 8912

Port B – Input only

Bit 7	Cassette data read (INPUT)
Bit 6	Printer BUSY signal (INPUT)
Bit 5	
Bit 4	
Bit 3	Not available
Bit 2	
Bit 1	
Bit 0	CRT generated interrupt

Port C – Output only

Bit 7	Controls BDIR on AY3 8912 OUT
Bit 6	Controls BC1 on AY3 8912 OUT
Bit 5	Cassette data write
Bit 4	Cassette motor ON/OFF
Bits 3 to 0	Keyboard scan row selection

Programming

The PPI is interfaced at the following addresses:

Address F4xx	Read and write at port A
Address F5xx	Read and write at port B
Address F6xx	Read and write at port C
Address F7xx	Write to the control register

Notes:
xx *signifies any value.*

A is used for reading (input) *and* writing (output), B is used for reading (input) only and C for writing (output) only.

Of the three modes described above, only mode 0 will be covered here since it covers all likely operations.

The PPI is programmed through a write-only control register. It is not possible to read this register.

Writing to the control register

Writing to the control register is carried out using a simple OUT command (in BASIC or Z80 machine code) to port F7xx.

The control word is an 8 bit word, made up as follows:

Bit 7	Always 1 in a control word.
Bit 6	Port A mode selection, first bit. Together with bit 5 this sets the port A mode. To select mode 0, this bit must be set to 0. When set to 1, it selects mode 2 (*but see bit 5, below*).
Bit 5	Port A mode selection, second bit. To select mode 0, this bit must be 0. When set to 1, it selects mode 1 (*but see bit 6, above*).
Bit 4	Sets direction of port A; 0 for output and 1 for input. Will normally be 1.
Bit 3	Sets direction of upper part of port C. 0 for output and 1 for input.

Bit 2 Sets port B mode. 0 signifies mode 0 and 1 signifies mode 1. Will always be 0.

Bit 1 Sets working direction of port B. 0 for output and 1 for input. Will always be 1.

Bit 0 Sets working direction of lower part of port C. 0 for output and 1 for input. Will always be 0.

If bit 7 is set to 0, the register is not used as a port control, but instead allows port C bits to be set to 0 or 1.

Bit 7 = 0 use register to set bits.
Bits 6, 5 and 4 not used.
Bits 3, 2 and 1 set the number of the bit to be positioned.
Bit 0 determines whether the bit is to be set to 0 or 1. 0 here means set the required bit to 0, 1 means set it to 1.

Programming is thus effected by sending the appropriate status word to the control register and then performing either a read or a write to the relevant port.

THE CRTC 6845 CHIP

General

The 6845 CRTC (Cathode Ray Tube Controller) controls the generation of video signals. It consists of an 8-bit bidirectional port and can be set up using its 19 internal registers. One of the registers serves as a buffer for programming the other 18.

The 6845 registers

R0 to R3

These determine the horizontal format and the timing. They are loaded with specific values according to the mode. For example, in mode 1:

R0 = 63
R1 = 40
R2 = 46
R3 = 142

R4 to R9

These determine the vertical format. They are loaded with specific values:

R4 = 38
R5 = 0
R6 = 25
R7 = 30

R10 to R15

These control the cursor and are constantly modified by the software.

R16 to R17

These deal with control of the light pen (not implemented).

R0	Total number of character spaces available horizontally (0–255)
R1	Number of characters displayed horizontally (0–255)
R2	Horizontal sync (position. 0–255)
R3	Length of synchronisation (0–15)
R4	Total number of rows available (0–127)
R5	Vertical sync (0–31)
R6	Number of characters displayed vertically (0–127)
R7	Vertical synch (position. 0–127)
R8	Interlace mode (0–3)
R9	Scanning (0–31)
R10	Start line of cursor scan (0–31)
R11	End line of cursor scan (0–31)
R12	Most significant byte of starting address of video RAM from 16383 (0–16383)
R13	Least significant byte of video RAM from 16383 (0–16383)
R14	Cursor position (MSB)
R15	Cursor position (LSB)

Programming

Two port addresses are used to program the CRTC.

Port BCxx is used to set register addresses and port BDxx is used to write data to the current register.

These registers are write-only, with the exception of registers 14 and 15 which can be read to give the current cursor position.

THE VIDEO GATE ARRAY

General

The Amstrad is equipped with a special circuit which looks after ROM switching and the CRTC chip. This is a custom circuit known as a *gate array*, designed specifically for the Amstrad.

Programming

The gate array may be looked upon as an 8-bit output port controlled using an OUT 7Fxx instruction.

The two top bits control the application:

Bit 7	Bit 6	Function
0	0	Loading of palette register
0	1	Loading of palette memory
1	0	ROM switching and video control
1	1	Reserved

ROM switching and video control

BIT	7	1	
BIT	6	0	
BIT	5	0	
BIT	4	1	Resets interrupting device to 0
BIT	3	0	Selects upper ROM. 1 deselects upper ROM
BIT	2	0	Selects lower ROM. 1 deselects lower ROM
BIT	1		video control MC1 (*see below*)
BIT	0		video control MC0 (*see below*)

MC1 and MC0

0	0	Mode 0 (24 rows of 20 columns)
0	1	Mode 1 (24 rows of 40 columns)
1	0	Mode 2 (24 rows of 80 columns)
1	1	Illegal combination

Palette register

BIT	7	0	
BIT	6	0	
BIT	5	0	
BIT	4	0	Load ink colour number according to bits 0–3
BIT	4	1	Load border colour number (bits 0–3 ignored)
BITS	3 to 0		Set the ink number (15 colours available)

Palette memory

BIT	7	0
BIT	6	1
BIT	5	0
BITS	4 to 0	31 values for decoding the colour of the palette register. The number of possible colours varies according to the mode in use.

HINTS AND TIPS

DUMPING HEX MEMORY FROM ROMS TO PRINTER

These programs will write ROM contents to the printer in hex.

Lower ROM hex dump

```
10 MEMORY &6000
15 CLS
20 FOR I=&A000 to &A010
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA F3,CD,06,B9,21,00,00,11,00,60,01,FF,3F,ED,B0,C9
70 DATA 00
80 CALL &A000
100 FOR I=&6000 TO 40960
120 IF INT(I/16)*16=I THEN PRINT #8,""
    :PRINT #8,HEX$(I-&6000);" ";
130 A=PEEK (I)
135 A$=RIGHT$("00"+HEX$(A),2)
140 PRINT #8,A$;" ";
150 NEXT I
```

Upper ROM hex dump

```
10 MEMORY &6000
15 CLS
20 FOR I=$A000 TO $A010
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA F3,CD,00,B9,21,00,C0,11,00,60,01,FF,3F,ED,B0,C9
70 DATA 00
80 CALL &A000
100 FOR I=&6000 TO 40960
120 IF INT(I/16)*16=I THEN PRINT #8,""
    :PRINT #8,HEX$(I+&6000);" ";
130 A=PEEK(I)
135 A$=RIGHT$("00"+HEX$(A),2)
140 PRINT #8,A$;" ";
150 NEXT I
```

ASCII DUMP OF UPPER AND LOWER ROMS TO PRINTER

Lower ROM ASCII DUMP

```
10 MEMORY &6000
15 CLS
20 FOR I=&A000 TO &A010
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA F3,CD,06,B9,21,00,00,11,00,60,01,FF,3F,ED,B0,C9
70 DATA 00
80 CALL &A000
100 FOR I=&6000 TO 40960
120 IF INT(I/64)*64=I THEN PRINT #8,""
    :PRINT #8,HEX$(I-&6000);" ";
130 A=PEEK(I)
140 IF (A>31 AND A<127) OR A>159 THEN PRINT #8,CHR$(A);
                                 ELSE PRINT #8,".";
150 NEXT I
```

Upper ROM ASCII dump

```
10 MEMORY &6000
15 CLS
20 FOR I=&A000 TO &A010
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA F3,CD,00,B9,21,00,C0,11,00,60,01,FF,3F,ED,B0,C9
70 DATA 00
80 CALL &A000
100 FOR I=&6000 TO 40960
120 IF INT(I/64)*64=I THEN PRINT #8,""
    :PRINT #8,HEX$(I+&6000);" ";
130 A=PEEK(I)
140 IF (A>31 AND A<127) OR A>159 THEN PRINT #8,CHR$(A);
                                 ELSE  PRINT #8,".";
150 NEXT I
```

STARTING AND STOPPING THE CASSETTE MOTOR

To start the motor: `OUT &HF600,16`

To stop the motor: `OUT &HF600,0`

PROTECTING A PROGRAM

Type this at the start of the program:

```
10 REM
20 PRINT "START"
```

followed by the program to be protected.

When the program has been entered, type:

```
POKE 372,225
```

From now on it becomes impossible to list the program, and it can only be executed by typing RUN 20.

The POKE instruction (*above*) has the effect of replacing the REM instruction in memory with an invalid token number (225) so that when the computer attempts to list the program, it encounters a token which it cannot translate and displays the message SYNTAX ERROR.

Similarly, when it tries to execute the program (with RUN), it encounters the same invalid token and freezes. RUN 20 allows execution of the program since it avoids ever having to try to interpret line 10.

ORIGINAL NOISES

```
5 REM STARSKY AND HUTCH SIREN
10 FOR I=80 TO 220 STEP 12
20 SOUND 1,I,2
30 NEXT I
40 FOR I=220 TO 80 STEP -12
50 SOUND 1,I,2
60 NEXT I
70 GOTO 10
```

```
5 REM PHASER SOUND
10 FOR I=90 TO 125
20 SOUND 1,I,2,15
30 NEXT I
50 GOTO 10
```

```
5 REM DEATH WHINE
10 FOR I=15 TO 8 STEP -1
20 SOUND 1,500,20,I,,,1
30 NEXT I
```

CIRCLE AND ELLIPSE PLOTTING PROGRAM

This program simulates the Microsoft BASIC CIRCLE instruction which is not available in Amstrad BASIC.

X and Y are the horizontal and vertical co-ordinates of the centre of the circle.

R is the radius of the circle.

SA represents the start angle and EA the end angle. These are both expressed in degrees and allow arcs of a circle to be plotted.

FF represents a flattening factor which allows ellipses to be plotted.

```
10 CLS
20 X=320:Y=200:R=100
30 SA=0
40 EA=360
50 FF=2
60 DEG
70 PLOT X+R*COS(SA),Y+R*SIN(SA)
80 FOR A=SA TO EA
90 X1=X+R*COS(A):Y1=Y+R*SIN(A)/FF
100 DRAW X1,Y1
110 PLOT X1,Y1
120 NEXT A
```

SCANNING THE KEYBOARD

Try entering the following program, RUN it and then press different keys. Note the values thus obtained, and you will be able to use them in your programs by PEEKing the relevant byte and testing its value. This routine can replace INKEY$ to some advantage. Use BREAK to end execution.

```
10 FOR I=&B4EB TO &B4F4
20 PRINT PEEK(I);
30 NEXT I
40 PRINT
50 GOTO 10
```

The following POKE modifies the background (PAPER) colour, producing narrow bands. Try using it with different values of N.

```
POKE &B290,N
```

Note:
N must have a value between 0 and 225.

PUTTING A MACHINE CODE ROUTINE INTO A COMMENT LINE

Short routines can be set up in a REM line (max. 255 characters, including the REM itself and any spaces) as long as they do not contain two bytes set to 0 in succession at any point.

Type:

```
10 REM ****************************
```

Use one asterisk for each byte of your routine.

As BASIC stores programs starting at address 368, the first asterisk will be found at address 374.

The following program sets up the routine and then erases itself.

```
20 FOR I=374 TO 379: REM for a 6-byte routine
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DATA 3E,19,21,88,CD,C9
70 DELETE 20-70
```

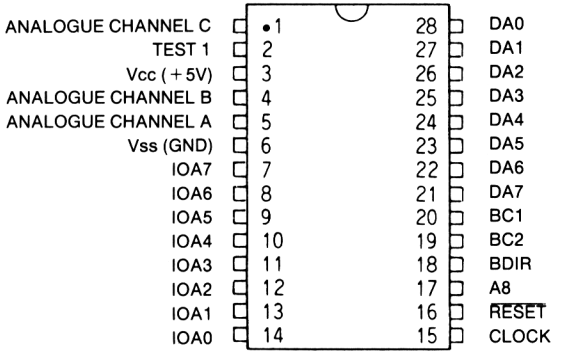
Note:
The above code is an example and serves no particular purpose.

CONNECTORS AND CHIP PINOUTS

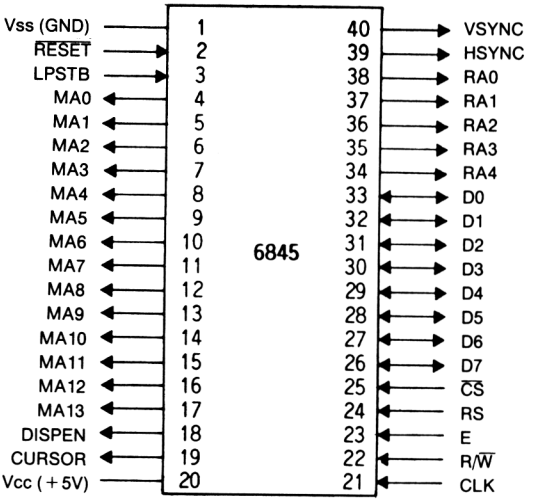
PINOUTS ON THE AY3 8912

28 LEAD DUAL IN LINE
AY3 8912

Top View

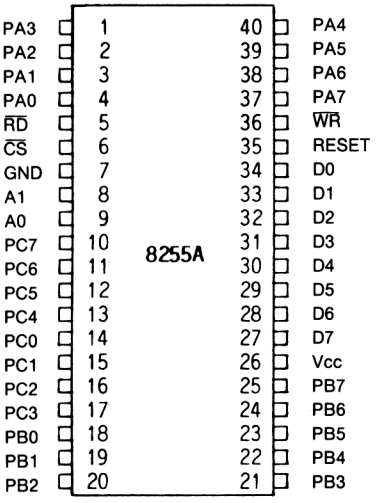


PINOUTS ON THE CRTC 6845



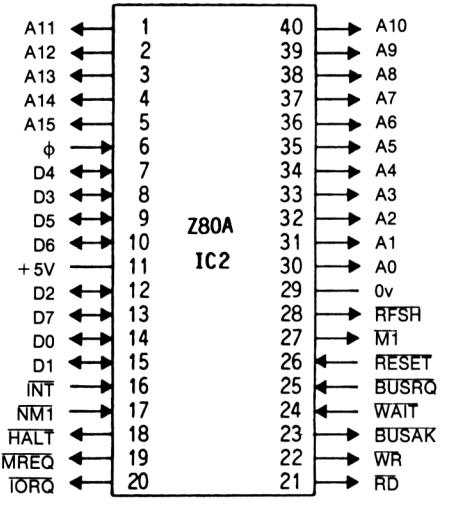
Pin name	Description Direction
D0–D7	Data bus – bidirectional tristate
CS	Circuit selection – input
RS	Register selection – input
R/W	Read/Write – input/output
E	Synchronisation signal – input
CLK	Clock – input
RESET	Initialisation – input
Vcc	Power supply (+5V) – input
MA0–MA13	Memory address (16K) – output
RA0–RA4	Line address (scanning) – output
HSYNC	Horizontal synchronisation – output
VSYNC	Vertical synchronisation – output
DISPEN	Enable/Disable display – output
CURSOR	Enable/Disable cursor – output
LPSTB	Light pen flag – input

PINOUTS ON THE PPI 8255



Name of pin	Function
D7–D0	Data bus (bidirectional)
RESET	Initialisation
CS	Chip select
RD	Read input
WR	Write input
A0–A1	Port address
PA7–PA0	Port A (bit)
PB7–PB0	Port B (bit)
PC7–PC0	Port C (bit)
Vcc	Power supply (+5 volts)
GND	0 volts (ground)

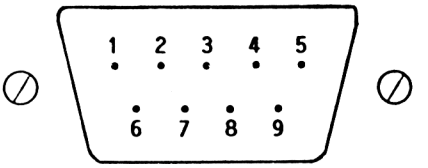
PINOUTS ON THE Z80



Pin	Function
1	Address bit 11
2	Address bit 12
3	Address bit 13
4	Address bit 14
5	Address bit 15
6	Clock input
7	Data bit 4
8	Data bit 3
9	Data bit 5
10	Data bit 6
11	+5 volt supply
12	Data bit 2
13	Data bit 7
14	Data bit 0
15	Data bit 1
16	Maskable interrupt request
17	Non-maskable interrupt request
18	HALT signal to microprocessor
19	Memory request
20	Input/Output request

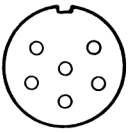
Pin	Function
21	Memory read command
22	Memory write command
23	Bus acknowledge
24	CPU wait request
25	Bus request
26	Initialise CPU
27	Start of machine cycle signal
28	Dynamic memory refresh signal
29	0 volts (ground)
30	Address bit 0
31	Address bit 1
32	Address bit 2
33	Address bit 3
34	Address bit 4
35	Address bit 5
36	Address bit 6
37	Address bit 7
38	Address bit 8
39	Address bit 9
40	Address bit 10

JOYSTICK CONNECTOR



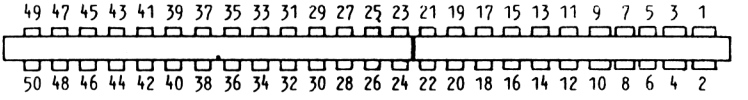
Pin 1	Top
Pin 2	Bottom
Pin 3	Left
Pin 4	Right
Pin 5	Unused
Pin 6	Fire button 2
Pin 7	Fire button 1
Pin 8	Common earth
Pin 9	Common earth 2

VIDEO OUTPUT CONNECTOR



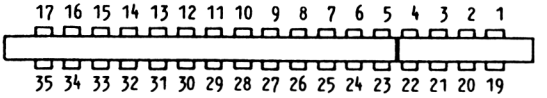
Pin 1	Red
Pin 2	Green
Pin 3	Blue
Pin 4	Sync
Pin 5	Earth
Pin 6	Brightness

EXPANSION CONNECTOR OUTPUT



Pin 1	Sound	Pin 26	D0
Pin 2	Earth	Pin 27	+5 volts
Pin 3	A15	Pin 28	MREQ
Pin 4	A14	Pin 29	M1
Pin 5	A13	Pin 30	RFSH
Pin 6	A12	Pin 31	IORQ
Pin 7	A11	Pin 32	RD
Pin 8	A10	Pin 33	WR
Pin 9	A9	Pin 34	HALT
Pin 10	A8	Pin 35	INT
Pin 11	A7	Pin 36	NMI
Pin 12	A6	Pin 37	BUSRD
Pin 13	A5	Pin 38	BUSAK
Pin 14	A4	Pin 39	READY
Pin 15	A3	Pin 40	BUS RESET
Pin 16	A2	Pin 41	RESET
Pin 17	A1	Pin 42	ROMEN
Pin 18	A0	Pin 43	ROMDIS
Pin 19	D7	Pin 44	RAMRD
Pin 20	D6	Pin 45	RAMDIS
Pin 21	D5	Pin 46	CURSOR
Pin 22	D4	Pin 47	LIGHT PEN
Pin 23	D3	Pin 48	EXP
Pin 24	D2	Pin 49	EARTH
Pin 25	D1	Pin 50	0

PRINTER OUTPUT CONNECTOR



Pin 1	STROBE
Pin 2	D0
Pin 3	D1
Pin 4	D2
Pin 5	D3
Pin 6	D4
Pin 7	D5
Pin 8	D6
Pin 9	D7
Pin 11	BUSY
Pin 14	GROUND (earth)
Pin 16	GROUND
Pin 19	GROUND
Pin 20	GROUND
Pin 21	GROUND
Pin 22	GROUND
Pin 23	GROUND
Pin 24	GROUND
Pin 25	GROUND
Pin 26	GROUND
Pin 28	GROUND
Pin 33	GROUND

Unused pins are not listed

APPENDIX A

TABLE OF VALUES FOR CHROMATIC SCALE

C	3822
C#	3608
D	3405
D#	3214
E	3034
F	2863
F#	2703
G	2551
G#	2408
A	2273
A#	2145
B	2025

These values correspond to an octave based on middle C, for each octave above this, divide the value by 2.

TERMINAL CONTROL CODES

Code	Action	Number of parameters
0	n/a	n/a
1	Prints the next character	1
2	Disables cursor display	0
3	Enables cursor display	0
4	Sets screen mode to 0, 1, 2	1
5	Prints the next character in graphic mode	1
6	Enables video display	0
7	Rings the bell	0
8	Destructive backspace	0
9	Moves the cursor one character right	0
10	Moves the cursor down one line	0
11	Moves the cursor up one line	0
12	Clears the current window, cursor home	0
13	Carriage return	0
14	Sets paper ink	1
15	Sets pen ink	1
16	Deletes the current character	0
17	Deletes the to start of (window) line	0
18	Deletes the to end of (window) line	0
19	Deletes from top left of window to cursor	0
20	Deletes to end of window	0
21	Disables (inhibits) the display	0
22	Sets opaque (0) or transparent (1) mode	1
23	Sets graphic mode	1
24	Swaps PAPER and PEN INK values	0
25	Sets up a character matrix	9
26	Sets the boundaries of a window	4
27	n/a	n/a
28	Sets INK colours	3
29	Sets border colours	2
30	Positions the cursor at top left-hand of window (home)	0
31	Absolute positioning of cursor in a window	2

TABLE OF PORT ADDRESSES

Address	Function	Direction
7Fxx	VIDEO GATE ARRAY	OUT
BCxx	6845 (ADDRESS)	OUT
BDxx	6845 DATA	OUT
BExx	6845 STATUS	IN
BFxx	6845 DATA	IN
DFxx	NON-EXTERNAL SELECTION	OUT
EFxx	PRINTER PORT	OUT
F4xx	8255 PORT A	I/O
F5xx	8255 PORT B	I/O
F6xx	8255 PORT C	I/O
	8255 CONTROL PORT	–
FFxx	RESERVED FOR USER	

SCREEN MEMORY FORMAT

Size: 16K

Normal start address: C000 *(but can begin at 0000, 4000 or 8000)*

Whatever the mode, screen memory can be considered as consisting of 8000 16-bit words, each defining 4, 8 or 16 pixels in modes 0, 1 and 2 respectively.

Mode 0:	4 pixels of 16 bits:	4 bits per pixel:	16 colours
Mode 1:	8 pixels of 16 bits:	2 bits per pixel:	4 colours
Mode 2:	16 pixels of 16 bits:	1 bit per pixel:	1 colour

Lines 0, 8, 16, 24...192 are stored in the first 2K.

Lines 1, 9, 17, 25...193 are stored in the next 2K.

...

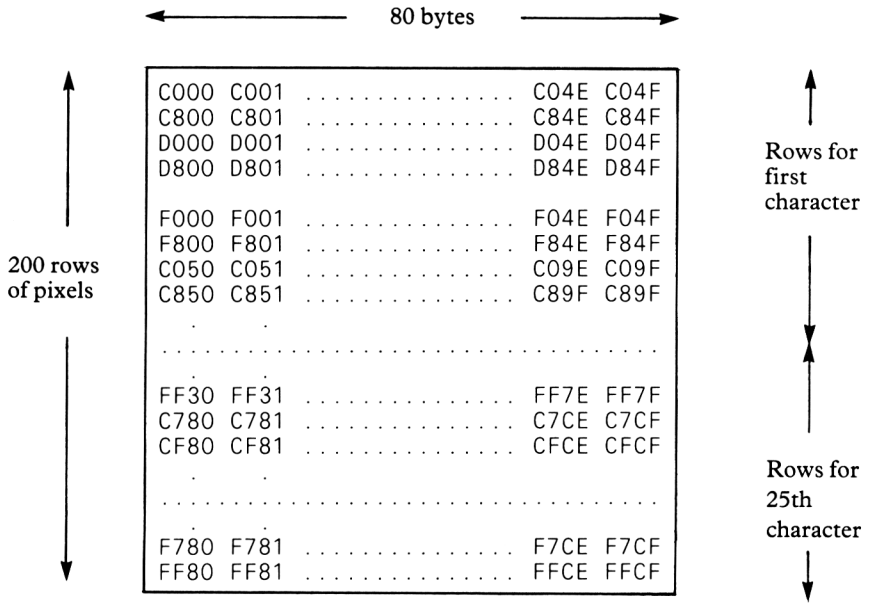
Lines 7, 15, 23, 31...199 are stored in the last 2K.

The 6845 address register determines the starting address of a 2K block (stored as a 10-bit value).

Each line uses 80 consecutive bytes in memory.

Example:
If the starting address is C000, then
line 0 occupies the first 80 bytes from C000 to C04F
line 1 occupies 80 bytes, from C800 to C84F
line 8 occupies the bytes from C050 to C09F

	Mode 0	Mode 1	Mode 2
Left-most pixel	bits 1, 5, 3, 7	bits 3, 7	bit 7
			bit 6
		bits 2, 6	bit 5
			bit 4
	bits 0, 4, 2, 6	bits 1, 5	bit 3
			bit 2
		bits 0, 4	bit 1
Right-most pixel			bit 0



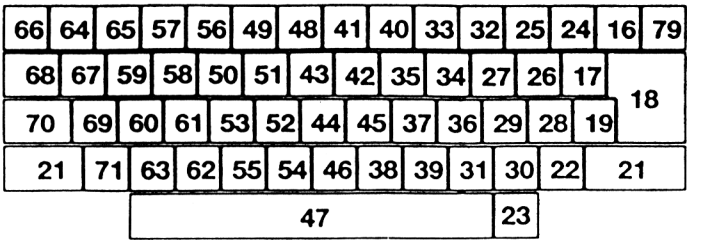
C7D0 to C7FF, CFD0 to CFFF, and so on. FFD0 to FFFF are not used.

TABLE OF COLOURS

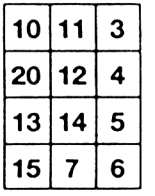
Number	Colour	6845 reg value
0	Black	20
1	Blue	4
2	Bright blue	21
3	Red	28
4	Magenta	24
5	Mauve	29
6	Bright red	12
7	Violet	5
8	Bright magenta	13
9	Green	22
10	Cyan (blue)	6
11	Sky blue	23
12	Yellow	30
13	White	0
14	Pastel blue	31
15	Orange	14
16	Pink	7
17	Pastel magenta	15
18	Bright green	18
19	Sea green	2
20	Bright cyan	19
21	Lemon yellow	26
22	Pastel green	25
23	Pastel cyan	27
24	Bright yellow	10
25	Pastel yellow	3
26	Bright white	11

TABLE OF KEYBOARD CODES

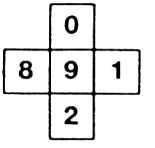
Keyboard



Numeric keypad

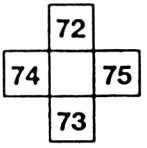


Cursor keys



Joysticks

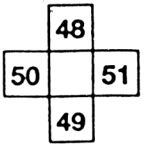
Joystick 0



Fire button 1

Fire button 2

Joystick 1



Fire button 1

Fire button 2

APPENDIX B

CPC 664 – MACHINE SPECIFIC INSTRUCTIONS

COMMANDS AND FUNCTIONS UNIQUE TO THE CPC 664

Functions

**COPYCHR$** COPYCHR$ (#channel number)
Copies the character at the current cursor position in the specified channel into a string variable.

**DEC$** DEC$ (numeric expression, format)
Formats a number for output (this format is identical with that of the PRINT USING instruction). This function makes it possible to put the result of USING into a string variable.

DERR DERR
Prints the last error number returned.

SPC SPC(N)
Generates N spaces for use with PRINT.

Commands

CLEAR INPUT CLEAR INPUT
This instruction clears the input buffer removing all characters currently in it.

CURSOR CURSOR operating system cursor flag, user cursor flag
This instruction enables or disables the cursor. The flag takes the value 1 if the cursor is to be enabled, 0 if it is to be disabled.

FILL FILL ink
Fills an area in the specified ink colour.

FRAME FRAME
Synchronises the writing of graphics with the CRT scan pulses to reduce flickering.

GRAPHICS GRAPHICS PAPER ink *and* GRAPHICS PEN ink
Sets the graphic PAPER INK or the graphic PEN INK values without otherwise affecting the pen or the paper.

MASK

```
MASK integer 0 to 255, integer 0 to 7
```

A very useful instruction which allows the structure of a line to be specified so as to be able to draw with a dotted or a composite line. The first byte specifies the structure of the line over 8 pixels (values 0 to 255), the second specifies the starting point within the 8 pixels.

Example:
To draw a dotted line using every other pixel:

```
MASK &X10101010,0
```

or

```
MASK 170,0
```

MID$

```
MID$(string1,position,length)=string2
```

Inserts string 2 into string 1, starting from the character defined by position and for number of characters length.

ON BREAK CONT

```
ON BREAK CONT
```

Disables the BREAK key. This function must be used with caution in finished programs. Once set, the only way to interrupt the program is by means of the RESET.

MATHS ROUTINE VECTORS IN THE CPC 664

The maths routines in the lower ROM are frequently called from the BASIC ROM in order to carry out all the BASIC calculating functions (+, *, /, sine, cosine, etc).

A series of vectors has been created to facilitate use of these calls.

The BASIC maths functions operate in a virtual accumulator of six bytes exactly as previously described earlier in this book.

Vector address	Absolute address	Purpose
BD5E	2F91	Copies the 5 bytes pointed to by DE into the area pointed to by HL and transfers the contents of the byte located in address HL – 1 (variable type) into A.
BD61	2F9F	Integer to floating point conversion in the 5 bytes pointed to by DE.
BD64	2FC8	Conversion of the binary number pointed to by HL into a number suitable for use in the 5 bytes of the virtual accumulator.
BD67	2FD9	Transforms the value contained in the 5 bytes pointed to by HL into an integer which will be held in HL.
BD6A	3001	Transforms the value contained in the 5 bytes pointed to by HL into an integer which will be held in the first 2 bytes pointed to by HL.
BD6D	3014	Performs the FIX function.
BD70	3055	Performs the INT function.
BD73	305F	SGN function (used by STR$ and PRINT).
BD76	30C6	Transformation routine (multiplies by 10^A).
BD79	34A2	Addition of two reals. HL points to an area of 5 bytes representing a number in real format (called ACCUM1). DE points to another area of 5 bytes (called ACCUM2). On completion of the routine, HL still points to ACCUM1 which contains the sum of ACCUM1 + ACCUM2.
BD7C	3159	RND function.
BD7F	349E	Subtraction of two reals. HL points to an area of 5 bytes representing a real number (called ACCUM1). DE points to another area of 5 bytes (called ACCUM2). On completion of the routine, HL still points to ACCUM1 which contains the value of ACCUM1 – ACCUM2.

Vector address	Absolute address	Purpose
BD82	3577	Multiplication of two reals. As above, but ACCUM1 ends up containing the value of ACCUM1*ACCUM2.
BD85	3604	Division of two reals. As above, but ACCUM1 contains the value of ACCUM1/ACCUM2.
BD88	3188	Returns the last RND value.
BD8B	36DF	Comparison of two reals: If ACCUM1>ACCUM2, then A = 1 If ACCUM1<ACCUM2, then A = 255 If ACCUM1 = ACCUM2, then A = 0.
BD8E	3731	Negation of a real. HL points to ACCUM1 which contains the value −ACCUM1.
BD91	3727	Tests the real contained in ACCUM1: HL points to ACCUM1. If ACCUM1>0, then A = 1 If ACCUM1<0, then A = 255 If ACCUM1 = 0, then A = 0.
BD94	3345	Sets angle-calculating mode to degrees or radians. If A = 0, selects RADIANS mode. If A<>0, selects DEGREES mode.
BD97	2F73	On exit, the area pointed to by HL on entry contains the constant PI.
BD9A	32AC	Extraction of the square root of a real number. On entry, HL points to an area of 5 bytes containing a real number. On exit, this area contains the square root of that number.
BD9D	32AF	Raising to a power of a real number. HL points to ACCUM1 which contains the number and DE points to ACCUM2 which contains the power. On exit, ACCUM1 contains the value of ACCUM1 raised to the power ACCUM2.
BDA0	31B6	Calculation of the napierian logarithm (to base *e*) of a real number, HL points to ACCUM1 which contains the entry number. On exit, ACCUM1 contains the value of the number's logarithm.
BDA3	31B1	Calculation of the common logarithm (to base 10) of a real number. HL points to ACCUM1 which contains the entry number. On exit, ACCUM1 contains the value of the number's common log.

Vector address	Absolute address	Purpose
BDA6	322F	Calculation of the exponent of a number. HL points to ACCUM1 which, on completion, contains the value of the number's exponent.
BDA9	3353	Calculation of the sine of an angle.
BDAC	3349	Calculation of the cosine of an angle.
BDAF	33C8	Calculation of the tangent of an angle.
BDB2	33D8	Calculation of the arc-tangent of an angle.
BDB5	2FD1	Evaluation routine.
BDB8	3136	RND routine (B8E4 and B8E6) initialisation.
BDBB	3143	Random number generator.

MAIN SYSTEM VARIABLES IN THE CPC 664

Address	Length	Contents
AC01	1	AUTO flag: 0 = AUTO enabled, 1 = AUTO disabled.
AC02	2	Number of the current line (used by AUTO).
AC04	2	Value of increment between lines (AUTO).
AC09	1	Used by WIDTH instruction.
AC0C	1	Used by NEXT instruction.
AC12	2	Used by FOR instruction.
AC14	2	Used by WHILE..WEND instructions.
AC16	11	Used by ON..GOTO instruction.
AC8A	256	Keyboard input buffer.
AD8C	2	Pointer for RESUME instruction.
AD8E	2	Used for error correction.
AD90	1	Error number.
AD91	2	Address of last byte executed.
AD93	2	Address for END, STOP and CONT.
AD98	1	Error number for ON ERROR GOTO function.
AD99	9	Parameters used by SOUND instruction.
ADF3	26	Variable declaration table. Consists of 26 bytes (1 for each letter of the alphabet). Each byte contains a code describing the default status of each variable beginning with the relevant letter.
AE15	2	Address of current line for READ DATA.
AE17	2	Address of start DATA statements for use with RESTORE and READ.
AE1B	2	Used for ON ERROR GOTO.
AE1F	1	TRACE flag: 0 = TROFF, 1 = TRON.
AE55	2	Temporary store of DE for use by CALL instruction.
AE577	1	Temporary store of accumulator for use with CALL instruction.
AE58	2	Temporary store of HL for use by CALL instruction.
AE5A	2	Temporary store of SP for use by CALL instruction.
AE5C	2	Used by ZONE instruction (address).
AE5E	2	HIMEM (upper address of BASIC).
AE60	2	Used by SYMBOL instruction (address).
AE64	2	Address of start of BASIC program (default 016F).
AE66	2	Address of end of BASIC program.
AE68	2	Address of start of variable table.
AE6A	2	Address of simple variables table.
AE6C	2	Address of array variables table (DIM).
B06F	2	Address of start of BASIC stack.
B09F	1	Status of virtual accumulator.
B0A0	5	5 bytes used by the virtual accumulator.

Address	Length	Contents
B113	1	Radian/degree mode flag.
B118	1	Prompt message flag: 0 = prompt enabled, not 0 = disabled.
B11A	1	File open indicator.
B11B	2	Address of 2K directory buffer.
B11D	2	Address of read buffer.
B131	1	Status of file.
B132	2	Current address of read buffer.
B134	2	Address of data memory.
B136	2	Logical length of file.
B15F	1	Status of write stream.
B162	2	Address of write buffer.
B176	2	Current address of write buffer.
B1E5	1	Synchronisation character.
B1E9	2	Read/Write speed.
B1ED		Start of sound control variables.
B2A6	240	15 groups of 16 bytes containing values for amplitude envelopes.
B396	240	15 groups of 16 bytes containing values for tone envelopes.
B496	80	Table of key values when used without SHIFT or CTRL.
B4E6	80	Table of SHIFTed key values.
B536	80	Table of key values when used with CTRL.
B586	80	Table of repeat data for each key.
B628	2	Used during keyboard scanning (address).
B62A	1	Temporary store for a scanned character (BB0C).
B633	1	Key repeat speed value.
B634	1	Key pre-repeat delay value.
B635	10	Key-scan table.
B63B	1	State of joystick 1.
B63E	1	State of joystick 2.
B68B	2	Address of key table for keys used without SHIFT or CTRL.
B68D	2	Address of SHIFTed key table.
B68F	2	Address of key table for keys used with CTRL.
B691	2	Address of key repeat details table.
B693	2	X co-ordinate of origin.
B695	2	Y co-ordinate of origin.
B697	2	Graphic X co-ordinate.
B699	2	Graphic Y co-ordinate.
B69B	2	X co-ordinate of one edge of graphic window.
B69D	2	X co-ordinate of the other edge of graphic window.
B69F	2	Y co-ordinate of one edge of graphic window.
B6A1	2	Y co-ordinate of the other edge of graphic window.
B6A3	1	Graphic PEN INK value.
B6A4	1	Graphic PAPER INK value.
B6A5	8	4 2-byte areas used as temporary stores during line drawing.

Address	Length	Contents
B6AD	2	X co-ordinate of end-point for line drawing.
B6AF	2	Y co-ordinate of end-point for line drawing.
B6B5	1	STREAM number.
B726	1	Current cursor row position.
B727	1	Current cursor column position.
B728	1	Window flag.
B729	1	Start row of current window.
B72A	1	Start column of current window.
B72B	1	End row of current window.
B72C	1	End column of current window.
B72E	1	Cursor flag: 0 = enabled, 1 = disabled.
B72F	1	Current INK for PEN.
B730	1	Current INK for PAPER.
B731	1	Background flag: 0 = enabled, 255 = disabled.
B734	2	First character and state of user-defined matrix table.
B736	2	Address of user-defined matrix table.
B763	96	Control code table.
B7C2	1	byte for INK mask.
B7C3	1	Screen mode (0, 1 or 2).
B7C4	2	Screen offset (0 to 7FF).
B7C6	1	High byte byte of start of screen storage area.
B7C7	1	Sometimes contains a C3 (jump).
B7C8		Contains jump address.
B7D2	1	Duration of first period of border flashing.
B7D3	1	Duration of second period of border flashing.
B7D4	32	INK colours (2 bytes per colour).
B7F7	1	Used by BORDER.

PRINCIPAL ADDRESSES OF CPC 664 LOWER ROM

The lower ROM contains the system routines (communication with hardware), the maths routines and the character generator.

Notes:
Addresses corresponding to routines already described in detail are marked with a * sign.

Routines located at identical addresses to those described for the CPC 461 are marked with an = sign.

005C = BCC8 *
0099 = BD0D *
003A = BD10 *
0163 = BCD7 *
016A = BCDA *
0170 = BCDD *
0176 = BCE0 *
017D = BCE3 *
0183 = BCE6 *
01B3 = BCE9 *
01C5 = BCEC *
01D2 = BCEF *
01E2 = BCF2 *
0219 BCFE *
0227 BCF5 *
0255 BCFB *
0276 BD01 *
0284 BCF8 *
028D BD0A *
0294 BD04 *
029A BD07 *
02A0 BCD1 *
02B1 BCD4 *
0326 BCCB *
0330 BCCE *
05D7 BD13 *
0606 BD16 *
066F 64K MICROCOMPUTER (V2) (*message*)
068B Copyright 1984 Amstrad Electronics PLC and Locomotive Software Ltd. (*message*)
07F5 *** program load failed *** (*message*)
0728 List of compatibles Arnold, Amstrad, Orion, Schneider, Awa, Solavox, Saisho, Triumph, Isp.
0766 BD1C *
0776 BD22 *
077C BD25 *
07A4 BD19 *
07B0 BD1F *
07D0 BD28 *
080B BD2B *
0825 BDF1 *
0834 BD31 *
0848 BD2E *
0853 BD34 *
08BB BD37 *
0ABB BBFF *
0ACC BC02 *
0AE5 BC0E *
0B08 BC11 *
0B13 BC14 *
0B13 BDEB *
0B33 BC05 *
0B38 BC08 *
0B52 BC0B *
0B59 BC17 *
0B66 BC1A *
0BAB BC1D *
0C01 BC20 *
0C0D BC23 *

0C1B	BC26 *
0C35	BC29 *
0C51	BC59 *
0C6D	BDE8 *
0C70	BC5C *
0C86	BDE5 *
0C8A	BC2C *
0CA3	BC2F *
0CE6	BC3E *
0CEA	BC41 *
0CEE	BC32 *
0CF3	BC38 *
0D16	BC35 *
0D1B	BC3B *
0DB5	BC44 *
0DB9	BC47 *
0DE1	BC4A *
0DFC	BC4D *
0E40	BC50 *
0EF5	BC53 *
0F26	BC56 *
0F8F	BC5F *
0F97	BC62 *
1070	BB4E *
1080	BB51 *
10E0	BBB4 *
10FF	BBB7 *
1156	BB6F *
1161	BB72 *
116C	BB75 *
1178	BB78 *
11C6	BB87 *
1204	BB66 *
124E	BB69 *
125B	BDCD *
125B	BDD0 *
1261	BB8A *
1261	BB8D *
1272	BB81 *
127A	BB84 *
1282	BB7B *
1293	BB7E *
12A2	BB90 *
12A7	BB96 *
12B6	BB93 *
12BC	BB99 *
12C2	BB9C *
12D0	BBA5 *
12EE	BBA8 *
12FA	BBAB *
1327	BBAE *
1331	BB5D *
1347	BDD3 *
1377	BB9F *
1384	BBA2 *
13A4	BB63 *
13A8	BB60 *
13BA	BDD6 *
13FA	BB5A *
1406	BDD9 *
144E	BB57 *
1455	BB54 *
14D0	BBB1 *
154B	BB6C *
15A4	BBBA *
15D3	BBBD *
15F7	BBC3 *
15FA	BBC0 *
1602	BBC6 *
160A	BBC9 *
1618	BBCC *
16A1	BBCF *
16E6	BBD2 *
1713	BBD5 *
1729	BBD8 *
1732	BBDB *
1763	BBDE *
176A	BBE4 *
1771	BBE1 *
1776	BBE7 *
177C	BBED *
177F	BBEA *
1782	BDDC *
1790	BBF3 *
1793	BBF0 *
1796	BDDF *
17A2	BBF9 *
17A5	BBF6 *
17B0	BDE2 *
193C	BBFC *
1B5C	BB00 *
1B98	BB03 *

1BBF	BB06 *
1BC5	BB09 *
1BFA	BB0C *
1C04	BB15 *
1C3C	Default value of extended keys (RUN for CTRL CR)
1C46	BB0F *
1CB3	BB12 *
1CDB	BB18 *
1CE1	BB1B *
1D38	BB21 *
1DB8	BDEE *
1DE5	BB24 *
1DF2	BB42 *
1DF6	BB3F *
1DFA	BB45 *
1E0B	BB48 *
1E19	BB4B *
1E2F	BB3C *
1E34	BB39 *
1E45	BB1E *
1EC4	BB2A *
1EC9	BB30 *
1ECE	BB36 *
1ED8	BB27 *
1EDD	BB2D *
1EE2	BB33 *
1EEF	Table of key default values
1FE9	BCA7 *
2050	BCB6 *
206B	BCB9 *
2114	BCAA *
21AC	BCB3 *
21CE	BCAD *
21EB	BCB0 *
2495	BCBC *
249A	BCBF *
24A6	BCC2 *
24AB	BCC5 *
24BC	BC65 *
24CE	BC68 *
24E1	BC6B *
288B	BC77, BC7A, BC7D, BC80, BC83, BC86, BC89, BC8C, BC8F, BC92, BC95, BC98, BC9B * (Cassette and disk routines)
2935	Press play then any key (*message*)
294B	Error (*message*)
2955	REC (*message*)
2958	And (*message*)
295D	Read (*message*)
2963	Write (*message*)
296A	Rewind (*message*)
2970	Tape (*message*)
2975	Found (*message*)
297D	Loading (*message*)
2985	Saving (*message*)
298D	OK (*message*)
2990	Block (*message*)
2996	Unnamed (*message*)
299D	File (*message*)
29A6	BCA1 *
29AF	BC9E *
29C1	BCA4 *
2BBB	BC6E *
2BBF	BC71 *
2BC1	BC74 *
2F73	BD97 * PI
2F78	CONSTANT PI
2F91	BD5E *
2F9F	BD61 *
2FC8	BD64 *
2FD1	BDB5 *
2FD9	BD67 *
3001	BD6A *
3014	BD6D *
3055	BD70 *
305F	BD73 *
30C6	BD76 *
30F5	Table of powers of 10. 13 sets of 5 bytes for values 10 to 10^{13}
3136	BDB8 * RND INT
3143	BDBB * RND SEED
3159	BD7C * RND
3188	BD88 * RND
31B1	BDA3 * LOG10
31B6	BDA0 * LOG
31EE	Constant for calculating LOG (4 groups of 5 bytes)
3220	Stored value of 1/SQR(2)

3225	Stored value of LOG(2) (0.693147181)
322A	Stored value of LOG10(2) (0.301029996)
322F	BDA6 * EXP
329D	Constant 1.44269504
32A2	Constant 88.0296919
32A7	Constant −88.7228391
32AC	BD9A * SQR
32AF	BD9C * POWER
3345	BD94 * DEG–RAD
3349	BDAC * COS
3353	BDA9 * SIN
3382	Table of 6 coded numbers, each of 5 bytes, for calculating sines and cosines
33B4	Table of 4 coded numbers, each of 5 bytes, for calculating sines andcosines
33C8	BDAF * TAN
33D8	BDB2 * ATN
33EE	Table of 11 coded numbers, each of 5 bytes, for calculating arc-tangents
349E	BD7F * −
34A2	BD79 * +
3577	BD82 * * (multiply)
3604	BD85 */
36DF	BD8B * COMPARISON
3727	BD91 * SGN
3731	BD8E * SIGN CHANGE
3800	Start of character generator table (256 groups of 8 bytes)
3FFF	End of table

PRINCIPAL ADDRESSES OF CPC 664 UPPER ROM

The upper ROM contains all the BASIC keyword processing routines.

C006	Initialisation and output of BASIC 1.1 (*message*)
C033	BASIC 1.1 (*message*)
C046	EDIT function
C058	Main input (READY display)
C0D7	READY (*message*)
C0EA	AUTO
C128	NEW
C12F	CLEAR
C23C	PAPER
C227	PEN
C24B	BORDER
C254	INK
C278	MODE
C283	CLS
C29B	COPYCHR$
C2A4	VPOS
C2A8	POS
C302	LOCATE
C311	WINDOW
C346	TAG
C34D	TAGOFF
C363	CURSOR
C42D	WIDTH
C452	EOF
C4E1	ORIGIN
C509	CLG
C515	FILL
C532	MOVE
C537	MOVER
C53C	DRAW
C541	DRAWR
C546	PLOT
C54B	PLOTR
C574	TEST
C579	TESTR
C59D	GRAPHICS
C5C3	MASK
C5D7	FOR
C6A5	NEXT
C76A	IF
C789	GOTO
C78F	GOSUB
C7B3	RETURN
C7EA	WHILE
C81D	WEND
C885	ON
C979	ON BREAK
C99A	DI
C9A0	EI
C9F8	ON SQ
CA25	AFTER
CA2D	EVERY
CA53	REMAIN
CB54	ERROR
CB74	UNDEFINED LINE (*message*)
CC04	Send 'BREAK IN' message
CC1F	BREAK (*message*)
CC25	IN (*message*)
CC29	STOP
CC34	END
CC96	CONT
CCCD	ON ERROR
CCD8	RESUME
CD17	Table of error messages (part of word)
CFF0	Table of of arithmetic and logic operation entry points
D11A	Table of entry points for the functions EOF, ERR, HIMEM, INKEY$, PI, RND, TIME, XPOS and YPOS.
D12E	DERR
D133	ERR
D14B	HIMEM
D164	XPOS
D16B	YPOS
D1E8	Table of entry points for functions
D242	MIN

D246	MAX
D26D	ROUND
D2AB	OPENOUT
D2B7	OPENIN
D2F0	CLOSEIN
D2F8	CLOSEOUT
D316	SOUND
D373	RELEASE
D37E	SQ
D3A1	ENV
D3D7	ENT
D459	INKEY
D473	JOY
D489	KEY DEF
D4DE	SPEED
D520	PI
D52C	DEG
D530	RAD
D534	SQR
D539	Routine for raising to a power
D563	EXP
D568	LOG10
D56D	LOG
D572	SIN
D577	COS
D57C	TAN
D581	ATN
D587	RANDOM NUMBER SEED ? (*message*)
D59C	RANDOMIZE
D5C4	RND
D653	DEFSTR
D657	DEFINT
D65B	DEFREAL
D691	LET
D6B9	DIM
D9F4	ERASE
DB18	LINE
DB48	INPUT
DB7F	? redo from start (*message*)
DCCD	RESTORE
DCDF	READ
DEC6	TRON
DECA	TROFF
DEE5	Table of entry points for BASIC keywords
DFA8	End of table
E0C8	Table of keywords which may be followed by a line number (GOTO, RESTORE, AUTO, EDIT, etc)
E1D2	LIST
E3AD	Routine for positioning character table during keyword search
E3F0	Test for keyword in table
E41D	Table of addresses for each of the 26 letters of the alphabet
E451	Table of keywords with their code
E73A	End of table
E7F3	DELETE
E8A3	RENUM
E9A8	DATA
E9AC	REM
EA7D	RUN
EABA	LOAD
EB02	CHAIN
EB59	MERGE
ECE1	SAVE
F20D	PEEK
F214	POKE
F21E	INP
F228	OUT
F232	WAIT
F261	CALL
F2A2	ZONE
F2A9	PRINT
F383	PRINT USING
F50D	WRITE
F570	MEMORY
F784	SYMBOL
F8EC	LOWER$
F8F1	Routine for conversion to lower case
F8FA	UPPER$
F964	BIN$
F969	HEX$
F98F	DEC$

F9BC	STR$
F9D3	LEFT$
F9D8	RIGHT$
FA07	MID$
FA69	LEN
FA6E	ASC
FA74	CHR$
FA7E	INKEY$
FA8D	STRING$
FAAD	SPACE$
FABE	VAL
FAE5	INSTR
FC53	FRE
FD0C	+
FD21	–
FD35	* (multiply)
FD52	/
FD67	Integer division
FD79	MODULO (remainder after division)
FD87	AND function (LOGICAL AND)
FD92	OR function (LOGICAL OR)
FD9C	XOR function (EXCLUSIVE OR)
FDB0	ABS
FE0E	FIX
FE13	INT
FEB6	CINT
FEEB	UNT
FF14	CREAL
FF1B	Clear accumulator
FF2A	SGN
FF32	Places an integer in the accumulator
FF3E	Conversion of an integer into a real
FF45	Places variable type in C
FF4B	Places variable type in A
FF83	Copies the accumulator to the area pointed to by DE
FF92	Tests for capitals
FF9C	Tests for number
FFAB	Conversion into capitals
FFCA	Compares A with contents of HL
FFD8	Compares HL with DE
FFDE	Compares HL with BC
FFE4	DE = HL – DE
FFF2	LDIR
FFF8	LDDR
FFFB	JP (HL)
FFFC	Return to address pointed to by BC
FFFE	Return to address pointed to by DE

CPC 664 ROM ABSOLUTE ADDRESSES

Vector address	Absolute address	Vector address	Absolute address	Vector address	Absolute address
BB00	1B5C	BB03	1B98	BB06	1BBF
BB09	1BC5	BB0C	1BFA	BB0F	1C46
BB12	1CB3	BB15	1C04	BB18	1CDB
BB1B	1CE1	BB1E	1E45	BB21	1D38
BB24	1DE5	BB27	1ED8	BB2A	1EC4
BB2D	1EDD	BB30	1EC9	BB33	1EE2
BB36	1ECE	BB39	1E34	BB3C	1E2F
BB3F	1DF6	BB42	1DF2	BB45	1DFA
BB48	1E0B	BB4B	1E19	BB4E	1070
BB51	1080	BB54	1455	BB57	144E
BB5A	13FA	BB5D	1331	BB60	13A8
BB63	13A4	BB66	1204	BB69	124E
BB6C	154B	BB6F	1156	BB72	1161
BB75	116C	BB78	1178	BB7B	1282
BB7E	1293	BB81	1272	BB84	127A
BB87	11C6	BB8A	1261	BB8D	1261
BB90	12A2	BB93	12B6	BB96	12A7
BB99	12BC	BB9C	12C2	BB9F	1377
BBA2	1384	BBA5	12D0	BBA8	12EE
BBAB	12FA	BBAE	1327	BBB1	14D0
BBB4	10E0	BBB7	10FF	BBBA	15A4
BBBD	15D3	BBC0	15FA	BBC3	15F7
BBC6	1602	BBC9	160A	BBCC	1618
BBCF	16A1	BBD2	16E6	BBD5	1713
BBD8	1729	BBDB	1732	BBDE	1763
BBE1	1771	BBE4	176A	BBE7	1776
BBEA	177F	BBED	177C	BBF0	1793
BBF3	1790	BBF6	17A5	BBF9	17A2
BBFC	193C	BBFF	0ABB	BC02	0ACC
BC05	0B33	BC0B	0B38	BC0B	0B52
BC0E	0AE5	BC11	0B0B	BC14	0B13
BC17	0B59	BC1A	0B66	BC1D	0BAB
BC20	0C01	BC23	0C0D	BC26	0C1B
BC29	0C35	BC2C	0C8A	BC2F	0CA3
BC32	0CEE	BC35	0D16	BC38	0CF3
BC3B	0D1B	BC3E	0CE6	BC41	0CEA
BC44	0DB5	BC47	0DB9	BC4A	0DE1
BC4D	0DFC	BC50	0E40	BC53	0EF5
BC56	0F26	BC59	0C51	BC5C	0C70

Vector address	Absolute address	Vector address	Absolute address	Vector address	Absolute address
BC5F	0F8F	BC62	0F97	BC65	24BC
BC68	24CE	BC6B	24E1	BC6E	2BBB
BC71	2BBF	BC74	2BC1	BC77	288B
BC7A	288B	BC7D	288B	BC80	288B
BC83	288B	BC86	288B	BC89	288B
BC8C	288B	BC8F	288B	BC92	288B
BC95	288B	BC98	288B	BC9B	288B
BC9E	29AF	BCA1	29A6	BCA4	29C1
BCA7	1FE9	BCAA	2114	BCAD	21CE
BCB0	21EB	BCB3	21AC	BCB6	2050
BCB9	206B	BCBC	2495	BCBF	249A
BCC2	24A6	BCC5	24AB	BCC8	005C
BCCB	0326	BCCE	0330	BCD1	02A0
BCD4	02B1	BCD7	0163	BCDA	016A
BCDD	0170	BCE0	0176	BCE3	017D
BCE6	0183	BCE9	01B3	BCEC	01C5
BCEF	01D2	BCF2	01E2	BCF5	0227
BCF8	0284	BCFB	0255	BCFE	0219
BD01	0276	BD04	0294	BD07	029A
BD0A	028D	BD0D	0099	BD10	00A3
BD13	05D7	BD16	0606	BD19	07A4
BD1C	0766	BD1F	07B0	BD22	0776
BD25	077C	BD28	07D0	BD2B	080B
BD2E	0848	BD31	0834	BD34	0853
BD37	08BB	BD3A	1D3C	BD3D	1BFE
BD40	145C	BD43	15E8	BD46	19D1
BD49	17AC	BD4C	17A8	BD4F	1626
BD52	19D5	BD55	0B41	BD58	07FC
BD5B	2C02	BD5E	2F91	BD61	2F9F
BD64	2FC8	BD67	2FD9	BD6A	3001
BD6D	3014	BD70	3055	BD73	305F
BD76	30C6	BD79	34A2	BD7C	3159
BD7F	349E	BD82	3577	BD85	3604
BD88	3188	BD8B	36DF	BD8E	3731
BD91	3727	BD94	3345	BD97	2F73
BD9A	32AC	BD9D	32AF	BDA0	31B6
BDA3	31B1	BDA6	322F	BDA9	3353
BDAC	3349	BDAF	33C8	BDB2	33D8
BDB5	2FD1	BDB8	3136	BDBB	3143

EXECUTION ADDRESSES OF BASIC KEYWORDS IN THE CPC 664

Address	Keyword	Address	Keyword
ABS	FDB0	ERR	D133
AFTER	CA25	ERROR	CB54
ASC	FA6E	EVERY	CA2D
ATN	D581	EXP	D563
AUTO	C0EA	FIX	FE0E
BIN$	F964	FOR	C5D7
BORDER	C24B	FRE	FC53
CALL	F261	GOSUB	C78F
CAT	D299	GOTO	C789
CHAIN	EB02	HEX$	F969
CHR$	FA74	HIMEM	D14B
CINT	FEB6	IF	C76A
CLEAR	C12F	INSTR	FAE5
CLG	C509	INK	C254
CLOSEIN	D2F0	INKEY	D459
CLOSEOUT	D2F8	INKEY$	FA7E
CLS	C283	INP	F21E
CONT	CC96	INPUT	DB48
COS	D577	INT	FE13
CREAL	FF14	JOY	D473
DATA	E9A8	KEY	D489
DEC$	F9F8	LEFT$	F9D3
DEF	D174	LEN	FA69
DEFINT	D657	LET	D691
DEFREAL	D65B	LINE	DB18
DEFSTR	D653	LIST	E1D2
DEG	D52C	LOAD	EABA
DELETE	E7F3	LOCATE	C302
DI	C99A	LOG	D56D
DIM	D6B9	LOG10	D568
DRAW	C53C	LOWER$	F8EC
DRAWR	C541	MAX	D246
EDIT	C046	MEMORY	F570
EI	C9A0	MERGE	EB59
ELSE	E9B2	MID$	FA07
END	CC34	MIN	D242
ENT	D3D7	MODE	C278
ENV	D3A1	MOVE	C532
EOF	C452	MOVER	C537
ERASE	D9F4	NEXT	C6A5

Address	Keyword	Address	Keyword
NEW	C128	SAVE	ECE1
ON	C885	SGN	FF2A
ON BREAK	C979	SIN	D572
ON ERROR	CCCD	SOUND	D316
ON SQ	C9F8	SPACE$	FAAD
OPENIN	D2B7	SPEED	D4DE
OPENOUT	D2AB	SQ	D37E
ORIGIN	C4E1	SQR	D534
OUT	F228	STOP	CC29
PAPER	C23C	STR$	F9CB
PEEK	F20D	STRING$	FA8D
PEN	C227	SYMBOL	F784
PI	D520	TAG	C346
PLOT	C546	TAGOFF	C34D
PLOTR	C54B	TAN	D57C
POKE	F214	TEST	C574
POS	C2AD	TESTR	C579
PRINT	F2A9	TIME	D13C
'(REM)	E9AC	TROFF	DEC6
RAD	D530	TRON	DECA
RANDOMIZE	D59C	UNT	FEEB
READ	DCDF	UPPER$	F8FA
RELEASE	D373	VAL	FABE
REM	E9AC	VPOS	C2A4
REMAIN	CA53	WAIT	F2E2
RENUM	E8A3	WEND	C81D
RESTORE	DCCD	WHILE	C7EA
RESUME	CCD8	WIDTH	C42D
RETURN	C7B3	WINDOW	C311
RIGHT$	F9D8	WRITE	F50D
RND	D5C4	XPOS	D164
ROUND	D26D	YPOS	D16B
RUN	EA7D	ZONE	F2A2

New keywords

Address	Keyword	Address	Keyword
COPYCHR$	C29B	FRAME	BD19
CURSOR	C363	GRAPHICS	C59D
DERR	D12E	MASK	C5C3
FILL	C515		

APPENDIX C

CPC 6128 – MACHINE SPECIFIC INSTRUCTIONS

The CPC 6128 is slightly diferent to the CPC 664.

The lower ROM contains most of the differences, the upper ROM is practically identical to that of the 664 apart from a slight offset in the actual addresses. The system vectors, system variables and maths routines are identical to those of the CPC 664 in both their functions and their addresses.

The following pages only describe the relevant differences in the BIOS and BASIC.

MAIN ADDRESSES OF THE CPC 6128 LOWER ROM

The lower ROM contains the system routines (communication with hardware), the maths routines and the character generator.

Note:
Addresses corresponding to routines already described in detail are followed by a * sign.

Routines located at identical addresses to those described for the CPC 664 are labelled here with a = sign.

005C =	BCC8 *	0227	BCF5 *
0099 =	BD0D *	0255	BCFB *
003A =	BD10 *	0276	BD01 *
0163 =	BCD7 *	0284	BCF8 *
016A =	BCDA *	028D	BD0A *
0170 =	BCDD *	0294	BD04 *
0176 =	BCE0 *	029A	BD07 *
017D =	BCE3 *	02A0	BCD1 *
0183 =	BCE6 *	02B1	BCD4 *
01B3 =	BCE9 *	0326	BCCB *
01C5 =	BCEC *	0330	BCCE *
01D2 =	BCEF *	05ED	BD13 *
01E2 =	BCF2 *	061C	BD16 *
0219	BCFE *	0688	64K MICROCOMPUTER (V2) (*message*)

068B	Copyright 1984 Amstrad Electronics PLC and Loco-motive Software Ltd. (*message*)
07F5	*** program load failed *** (*message*)
0728	List of compatibles Arnold, Amstrad, Orion, Schneider, Awa, Solavox, Saisho, Triumph, Isp.
0766	BD1C *
0786	BD22 *
078C	BD25 *
07B4	BD19 *
07C0	BD1F *
07E0	BD28 *
081B	BD2B *
0835	BDF1 *
0844	BD31 *
0858	BD2E *
0863	BD34 *
08BD	BD37 *
0ABF	BBFF *
0AD0	BC02 *
0AE9	BC0E *
0B0C	BC11 *
0B17	BC14 *
0B17	BDEB *
0B37	BC05 *
0B3C	BC08 *
0B56	BC0B *
0B5D	BC17 *
0B6A	BC1A *
0BAF	BC1D *
0C05	BC20 *
0C11	BC23 *
0C1F	BC26 *
0C39	BC29 *
0C55	BC59 *
0C71	BDE8 *
0C74	BC5C *
0C8A	BDE5 *
0C8E	BC2C *
0CA7	BC2F *
0CEA	BC3E *
0CEE	BC41 *
0CF2	BC32 *
0CF7	BC38 *
0D1A	BC35 *
0D1F	BC3B *
0DB9	BC44 *
0DBD	BC47 *
0DE5	BC4A *
0E00	BC4D *
0E44	BC50 *
0EF9	BC53 *
0F2A	BC56 *
0F93	BC5F *
0F9B	BC62 *
1074	BB4E *
1084	BB51 *
10E4	BBB4 *
1103	BBB7 *
115A	BB6F *
1165	BB72 *
1170	BB75 *
117C	BB78 *
11CA	BB87 *
1208	BB66 *
1252	BB69 *
125F	BDCD *
125F	BDD0 *
1265	BB8A *
1265	BB8D *
1276	BB81 *
127E	BB84 *
1286	BB7B *
1297	BB7E *
12A6	BB90 *
12AB	BB96 *
12BA	BB93 *
12C0	BB99 *
12C6	BB9C *
12D4	BBA5 *
12E2	BBA8 *
12FE	BBAB *
132B	BBAE *
1335	BB5D *
134B	BDD3 *

137B	BB9F *
1388	BBA2 *
13A8	BB63 *
13AC	BB60 *
13BE	BDD6 *
13FE	BB5A *
140A	BDD9 *
1452	BB57 *
1459	BB54 *
14D4	BBB1 *
154F	BB6C *
15A8	BBBA *
15D7	BBBD *
15FB	BBC3 *
15FE	BBC0 *
1606	BBC6 *
160E	BBC9 *
161C	BBCC *
16A5	BBCF *
16EA	BBD2 *
1717	BBD5 *
172D	BBD8 *
1736	BBDB *
1767	BBDE *
176E	BBE4 *
1775	BBE1 *
177A	BBE7 *
1780	BBED *
1783	BBEA *
1786	BDDC *
1794	BBF3 *
1797	BBF0 *
179A	BDDF *
17A6	BBF9 *
17A9	BBF6 *
17B4	BDE2 *
1940	BBFC *

1B5C= From this address onwards, the lower ROM routines use the same entry points in the CPC 6128 and the CPC 664.

MAIN ADDRESSES OF THE CPC 6128 UPPER ROM

The upper ROM contains all the BASIC keyword processing routines.

C006	Initialisation and output of BASIC 1.1 (*message*)
C033	BASIC 1.1 (*message*)
C046	EDIT function
C058	Main input (READY display)
C0D7	READY (*message*)
C0EA	AUTO
C128	NEW
C12F	CLEAR
C224	PEN
C239	PAPER
C248	BORDER
C251	INK
C275	MODE
C280	CLS
C298	COPYCHR$
C2A1	VPOS
C2A5	POS
C2FF	LOCATE
C30E	WINDOW
C343	TAG
C34A	TAGOFF
C360	CURSOR
C42A	WIDTH
C44F	EOF
C4DE	ORIGIN
C506	CLG
C512	FILL
C52F	MOVE
C534	MOVER
C539	DRAW
C53E	DRAWR
C543	PLOT
C548	PLOTR
C571	TEST
C576	TESTR
C59A	GRAPHICS
C5C0	MASK
C5D4	FOR
C6A2	NEXT
C767	IF
C786	GOTO
C78C	GOSUB
C7B0	RETURN
C7E7	WHILE
C81A	WEND
C882	ON
C976	ON BREAK
C997	DI
C99D	EI
C9F5	ON SQ
CA22	AFTER
CA2A	EVERY
CA50	REMAIN
CB51	ERROR
CBF1	UNDEFINED LINE (*message*)
CC01	Send 'BREAK IN' message routine
CC1C	BREAK (*message*)
CC22	IN (*message*)
CC26	STOP
CC31	END
CC93	CONT
CCCA	ON ERROR
CCD5	RESUME
CD14	Table of error messages (part of word)
CFED	Table of of arithmetic and logic operation entry points
D01D	–
D028	NOT
D036	+
D117	Table of entry points for the functions EOF, ERR, HIMEM, INKEY$, PI, RND, TIME, XPOS and YPOS
D12B	DERR
D130	ERR
D139	TIME
D142	ERL
D148	HIMEM

D14E	@
D161	XPOS
D168	YPOS
D1E5	Table of entry points for functions
D23F	MIN
D243	MAX
DE6A	ROUND
D296	CAT
D2A8	OPENOUT
D2B4	OPENIN
D2ED	CLOSEIN
D2F5	CLOSEOUT
D313	SOUND
D370	RELEASE
D37B	SQ
D39E	ENV
D3D4	ENT
D456	INKEY
D470	JOY
D486	KEY DEF
D4DB	SPEED
D51D	PI
D529	DEG
D52D	RAD
D531	SQR
D536	Routine for raising to a power
D560	EXP
D565	LOG10
D56A	LOG
D57F	SIN
D574	COS
D579	TAN
D57E	ATN
D584	RANDOM NUMBER SEED ? (*message*)
D599	RANDOMIZE
D5C1	RND
D650	DEFSTR
D654	DEFINT
D658	DEFREAL
D68E	LET
D6B6	DIM
D9F0	ERASE
DB13	LINE
DB43	INPUT
DB7A	? redo from start (*message*)
DCC8	RESTORE
DCDA	READ
DEC1	TRON
DEC5	TROFF
DEE0	Table of entry points for BASIC keywords
DFA3	End of table
E0C3	Table of keywords which may be followed by a line number (GOTO, RESTORE, AUTO, EDIT, etc)
E1CD	LIST
E3A8	Routine for positioning character table during keyword search
E3EB	Test for keyword in table
E418	Table of addresses for each of the 26 letters of the alphabet
E44C	Table of keywords and their tokens
E735	End of keyword token table
E7EE	DELETE
E89E	RENUM
E9A3	DATA
E9A7	REM
E9AD	ELSE
EA78	RUN
EAB5	LOAD
EAFD	CHAIN
EB54	MERGE
ECDC	SAVE
F208	PEEK
F20F	POKE
F219	INP
F223	OUT
F229	WAIT
F25C	CALL
F29D	ZONE
F2A9	PRINT
F383	PRINT USING
F508	WRITE
F56B	MEMORY

F784	SYMBOL
F8EC	LOWER$
F8F1	Routine for conversion to lower case
F8FA	UPPER$
F964	BIN$
F969	HEX$
F98F	DEC$
F9BC	STR$
F9D3	LEFT$
F9D8	RIGHT$
FA07	MID$
FA69	LEN
FA6E	ASC
FA74	CHR$
FA7E	INKEY$
FA8D	STRING$
FAAD	SPACE$
FABE	VAL
FAE5	INSTR
FC53	FRE
FD0C	+
FD21	–
FD35	* (multiply)
FD52	/
FD67	Integer division
FD79	MODULO (remainder after division)
FD87	AND function (LOGICAL AND)
FD92	OR function (LOGICAL OR)
FD9C	XOR function (EXCLUSIVE OR)
FDB0	ABS
FE0E	FIX
FE13	INT
FEB6	CINT
FEEB	UNT
FF14	CREAL
FF1B	Clear accumulator
FF2A	SGN
FF32	Puts an integer into the accumulator
FF3E	Conversion into real
FF45	Puts variable type in C
FF4B	Puts variable type in A
FF83	Copies the accumulator to the address pointed to by DE
FF92	Tests for capitals
FF9C	Tests for number
FFAB	Conversion into capitals
FFCA	Compares A with contents of HL
FFD8	Compares HL with DE
FFDE	Compares HL with BC
FFE4	DE = HL – DE
FFF2	LDIR
FFF8	LDDR
FFFB	JP (HL)
FFFC	Return to address pointed to by BC
FFFE	Return to address pointed to by DE

INDEX

Amstrad Advanced Users Guide BOOK 1

The Amstrad Advanced Users Guide is a handbook allowing instant access to all the information you need: BASIC programming, the Z80 instruction set, entry points for system routines, control blocks, internal architecture, connectors, pinouts and the registers of principal circuits.

The Amstrad Advanced Users Guide is a mine of information: protecting programs, producing original sounds, scanning the keyboard or installing a machine-code routine and much more besides to help you discover the power of your CPC 464, 664 or 6128.

GLENTOP

£8.50

The Glentop Press Ltd
Standfast House
Bath Place
High Street Barnet
Herts EN5 5XE

ISBN 1-85181-122-2

9 781851 811229

AMSTRAD ADVANCED USERS-GUIDE BOOK 1 Martin GLENTOP

Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/