AMSTRAD

CPC 464

USER INSTRUCTIONS

CPC464 COLOUR PERSONAL COMPUTER 64K

Using this user guide

Computing has come a long way in a very short space of time. Of all the technological advances in the twentieth century, computing is easily the most startling.

Features of computer hardware and software have advanced far more rapidly than even existing users are able to follow, and to attempt to show owners of the CPC464 all the available power and subtlety of its BASIC, operating system and hardware attributes would require several thousands of pages.

Thus this guide is a concise introduction to the CPC464 and its software. It will be supplemented by many more specific and detailed instruction courses and publications.

Users familiar with other dialects of BASIC will become familiar with the framework of AMSTRAD BASIC very quickly - and newcomers will quickly appreciate the direct and unambiguous nature of the terminology used - it has been specifically devised and written to avoid the idiosyncracies found in many non-standard interpretations of BASIC, as well as introducing several fundamental 'real time' features not previously available in a low cost computer.

The user guide is divided into three sections.

The first is the beginners Foundation Course, written specifically to introduce computing concepts and terminology to the novice. If you have not previously owned or used a personal computer to the point at which you have written a small program of your own, then we advise that you work through the Foundation Course.

Those of you with previous experience should enter at Chapter 1. We have reiterated a number of essential items concerning setting up and familiarisation, but have concentrated on introducing the specific features of the CPC464 system, and made some assumptions about your familiarity with the terminology.

Each of the *'Primer'* chapters has been written to provide a broad guide to the many exciting features of the CPC464. Some fundamental points are repeated for emphasis - and because many users will want to dive straight into sound and graphics after the briefest possible introduction to the keyboard and more methodical aspects of learning BASIC.

AMSTRAD's 'Guide to BASIC' training course is intended to provide a thorough and extensively illustrated approach to understanding the many facets of your CPC464 and its boundless potential as a combination of tutor, games console and 'pure' computer, and we strongly advise that if you want to learn in a thorough manner, then you invest in a copy - *if you have not already done so!*

Finally, an extensive Appendix section provides a broad overview of computing concepts, as well as the machine-specific points of reference.

We wish you every success - you could have not have chosen finer value for money, nor a computer with a greater potential for developing your understanding of all aspects of the subject. There is no finer way of finding out about computing than by using a computer - and the CPC464 is particularly 'user friendly'.

AMSOFT

A division of

AMSTRAD

CONSUMER ELECTRONICS PLC

AMSOFT welcome comments and suggestions relating to the product or this manual.
All correspondance should be be addressed to:

AMSOFT
169 Kings Road
Brentwood
Essex CM14 4EF

First Published 1984
- Second Edition Autumn 1984
Compiled by W. Poel, R. Perry, I. Spital, R.J. Watkins
Published by AMSTRAD
Typeset by AMSOFT Computer Graphics

IMPORTANT

When reading this user guide, you should watch out for the different type styles that indicate the different ways in which references are made to programs, **[KEYS]** which are present on the computer, but which do not result in a printed character on the screen, and ‹general descriptions› which are associated with the programming words, but are not to be typed in as part of the instruction.

1. Always connect the Mains Lead to a 3 pin Plug following the instructions contained in the first section entitled 'Setting Up'.

2. Never connect the computer keyboard, monitor or power supply/modulator to any piece of equipment or source of power supply other than that described in this guide. Failure to comply with this will result in serious damage, and invalidate the guarantee.

3. Keep flower vases, drinks, etc. well away from the computer keyboard, monitor or power supply/ modulator. If liquid is spilt into any of these units, serious damage will result. Under such circumstances, consult qualified personnel.

4. Do not block or cover the ventilation slots on the top or back of the computer keyboard, monitor, or power supply/modulator.

5. Turning off the power will lose all that is stored in the CPC464 memory. If you wish to save a program, read Chapter 2 after having first completed the 'Foundation Course'.

6. It is recommended that you use cassettes specifically designed for use with computers. However, it is perfectly acceptable to use good quality audio-type cassettes made by leading manufacturers, providing that they are not Cr02 or 'metal' tape, and are no longer than 90 minutes (C-90).

To enable you to locate programs recorded on the tape more easily, we suggest that you use C-12 cassettes (6 minutes per side.)

7. Note that cassettes containing programs from other types of computer cannot be run or loaded on the CPC464.

8. If the cassette you are using has had the safety tabs removed to prevent accidental erasure of programs then the record button will not depress. Please do not use force on this button, otherwise the mechanism may be damaged. If you wish to re-record on a cassette where the safety tabs on the back of the cassette have been removed, this may be achieved by covering the holes on the back of the cassette with adhesive tape.

9. Remember to ensure that the tape in the cassette is wound beyond the first section (leader tape) before you start to save a program.

10. Take special care not to use or store any of the units in direct sunlight, in excessively hot, cold, damp or dusty areas, or places subject to any heavy vibration. Never store program cassettes near any magnetic fields, such as those that occur in loudspeakers or large electric motors.

11. General care of your cassettes and regular cleaning of your datacorder mechanism should lead to error free storage and retrieval of programs.

12. There are no user-serviceable parts inside the units. Do not attempt to gain access into the equipment. Refer all servicing to qualified service personnel.

13. Neither the whole or any part of the information contained herein, nor the programs or products described in this manual may be adapted or reproduced in any material form.

Contents

AMSTRAD CPC464
BEGINNERS' FOUNDATION COURSE

Foundations 1: SETTING UP

Initial instructions on the unpacking, interconnecting and switching on your CPC464 system.

The AMSTRAD CPC464 colour personal computer can be set up using either:

1.1 AMSTRAD GT64 Green Tube Monitor
1.2 AMSTRAD CTM640 Colour Monitor
1.3 AMSTRAD MP1 Modulator/Power supply and a domestic (UHF) TV receiver.

Please refer to the appropriate section to enable you to connect up your Computer system correctly and proceed onto the operating instructions.

1.1 AMSTRAD GT64 Green Tube Monitor

Unpack the monitor and connect a Mains Plug to the Mains Lead as follows:

IMPORTANT

The wires in this Mains Lead are coloured in accordance with the following code:

Blue: Neutral
Brown: Live

If a 13 Amp (BS1363) Plug is used, a 5 Amp Fuse must be fitted. The 13 Amp Fuse supplied in a new Plug must NOT be used. If any other Plug is used, a 5 Amp Fuse must be fitted - either in the Plug, or Adaptor or at the Distribution Board. As the colours of the wires in the Mains Lead of this apparatus may not correspond with the coloured markings identifying the terminals in your Plug, proceed as follows:

The wire which is coloured BLUE must be connected to the terminal which is marked with the letter 'N' or coloured BLACK.

The wire which is coloured BROWN must be connected to the terminal which is marked with the letter 'L' or coloured RED.

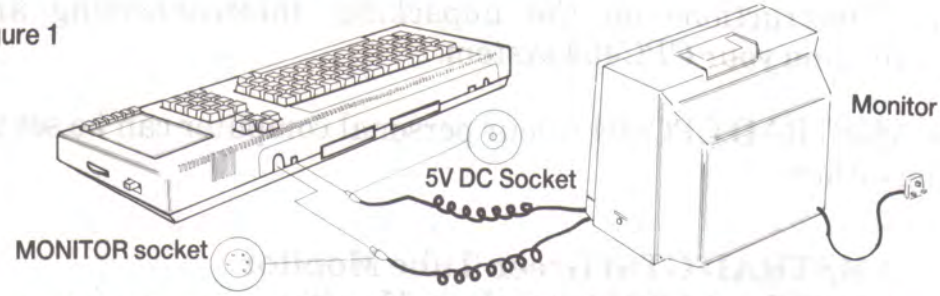
WARNING

Disconnect the Mains Plug from the Supply Socket when not in use.

No internal connections need to be made, therefore no attempt should be made to gain access to the inside of the equipment.

The computer should be positioned in front of the Monitor on a suitable table close to the Mains Supply Socket. As shown in figure 1, connect the lead with the larger (6 pin DIN) Plug from the monitor to the socket marked **MONITOR** on the back of the computer. Connect the lead with the smaller (DC power) Plug from the monitor to the socket marked **5V DC** on the back of the computer.

Figure 1



Ensure that the monitor **POWER** button is set to the **OFF** position (Out). Connect the Mains Plug from the monitor into the Mains Supply (240v AC) Socket.

Now switch on the monitor, and then switch on the computer using the slide switch marked **POWER** on the right hand end.

The red **ON** lamp at the top centre of the computer keyboard unit should be illuminated, and the monitor will display the following picture:

```
Amstrad 64K Microcomputer  (v1)
©1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

BASIC 1.0
Ready
```

Cursor

To avoid unnecessary eye-strain, adjust the control marked **BRIGHTNESS** until the display is adequately bright for comfortable viewing, without glare or blurring of the writing.

You should also adjust the **CONTRAST** control to the minimum setting consistent with comfortable viewing.

The vertical hold control on the GT64 is marked **V-HOLD**, and should be adjusted so that the picture is correctly positioned in the middle of the screen, without jitter or 'roll'.

1.2 AMSTRAD CTM640 Colour Monitor

Unpack the monitor and connect a Mains Power Plug to the Mains Lead as follows:

IMPORTANT

The wires in this Mains Lead are coloured in accordance with the following code:

Blue: Neutral
Brown: Live

If a 13 Amp (BS1363) Plug is used, a 5 Amp Fuse must be fitted. The 13 Amp Fuse supplied on a new Plug must NOT be used. If any other Plug is used, a 5 Amp Fuse must be fitted - either in the Plug, or Adaptor or at the Distribution Board. As the colours of the wires in the Mains Lead of this apparatus may not correspond with the coloured markings identifying the terminals in your Plug, proceed as follows:

The wire which is coloured BLUE must be connected to the terminal which is marked with the letter 'N' or coloured BLACK.

The wire which is coloured BROWN must be connected to the terminal which is marked with the letter 'L' or coloured RED.

WARNING

Disconnect the Mains Plug from the Supply Socket when not in use.

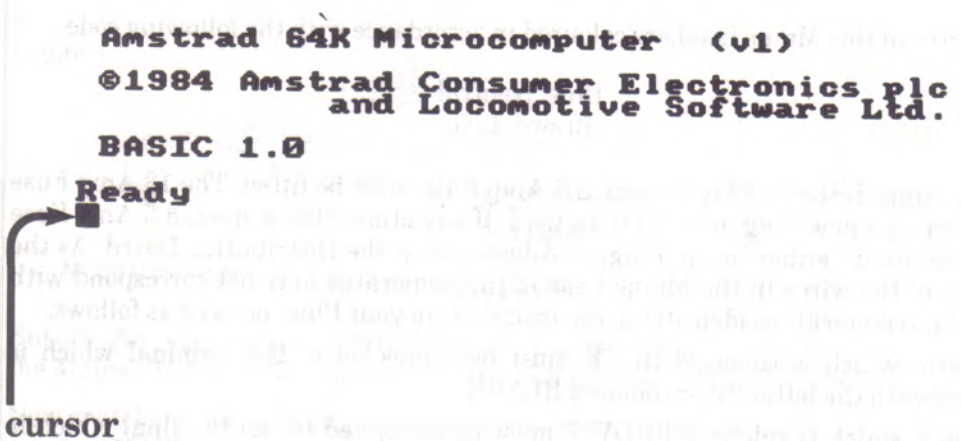
No internal connections need to be made, therefore no attempt should be made to gain access to the inside of the equipment.

The computer should be positioned in front of the Monitor on a suitable table close to the Mains Supply Socket. As shown in figure 1,on the previous page, connect the lead with the larger (6 pin DIN) Plug from the monitor to the socket marked **MONITOR** on the back of the computer. Connect the lead with the smaller (DC power) Plug from the monitor to the socket marked **5V DC** on the back of the computer.

Ensure that the monitor **POWER** button is set to the **OFF** position. Connect the Mains Plug from the monitor into the Mains Supply (240v AC) socket.

Now switch on the monitor, and then switch on the computer using the slide switch marked **POWER** on the right hand end.

The red **ON** lamp at the top centre of the computer keyboard unit should be illuminated, and the monitor will display the following picture:



To avoid unnecessary eye-strain, adjust the control at the side of the monitor marked **BRIGHTNESS** until the display is adequately bright for comfortable viewing, without glare or blurring of the writing.

1.3 AMSTRAD MP1 Modulator/Power supply and a domestic (UHF) Colour TV receiver.

The MP1 is an additional item that you may wish to purchase if you are currently using your CPC464 computer with the GT64 green tube monitor. The MP1 enables you to use the computer with your domestic colour TV and thereby enjoy the full colour facilities of your CPC464 computer.

Unpack the Modulator/Power Supply (MP1) and connect a Mains Plug to the Mains Lead of the MP1 as follows:

IMPORTANT

The wires in this Mains Lead are coloured in accordance with the following code:

Blue: Neutral
Brown: Live

If a 13 Amp (BS1363) Plug is used, a 3 Amp Fuse must be fitted. The 13 Amp Fuse supplied in a new Plug must NOT be used. If any other Plug is used, a 5 Amp Fuse must be fitted - either in the Plug, or Adaptor or at the Distribution Board. As the colours of the wires in the Mains Lead of this apparatus may not correspond with the coloured markings identifying the terminals in your Plug, proceed as follows:

The wire which is coloured BLUE must be connected to the terminal which is marked with the letter 'N' or coloured BLACK.

The wire which is coloured BROWN must be connected to the terminal which is marked with the letter 'L' or coloured RED.

WARNING

Disconnect the Mains Plug from the Supply Socket when not in use.

No internal connections need to be made, therefore no attempt should be made to gain access to the inside of the equipment.

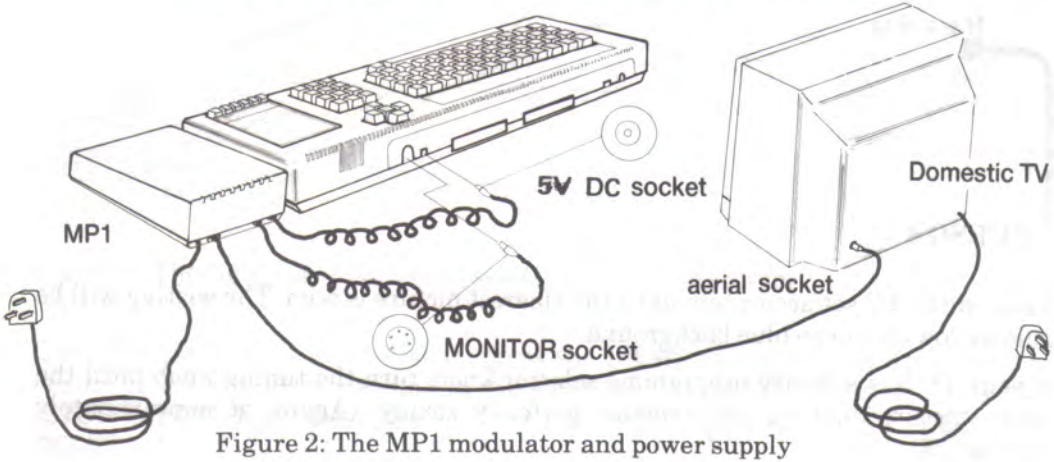


Figure 2: The MP1 modulator and power supply

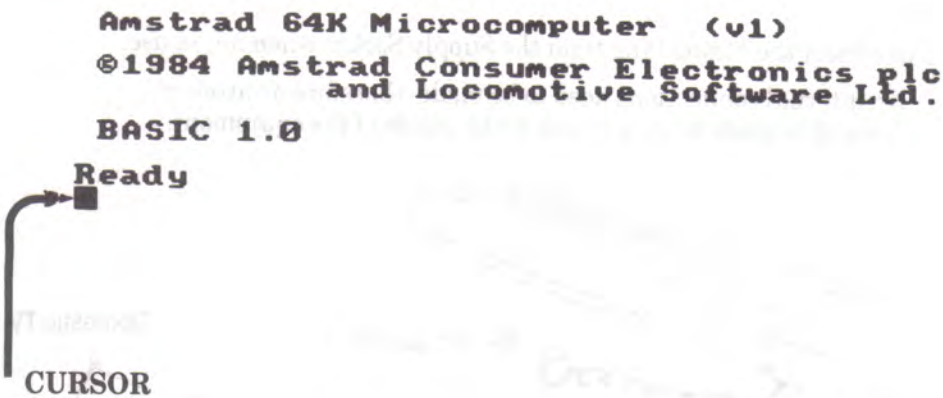
The modulator/power supply (MP1) should be positioned to the right of the computer on a suitable table close to the TV set and the Mains Supply socket. As shown in figure 2, connect the lead with the larger (6 pin DIN) plug from the MP1 to the socket marked **MONITOR** on the back of the computer. Connect the lead with the smaller (DC power) Plug from the MP1 to the socket marked **5V DC** on the back of the computer.

Connect the lead with the aerial plug from the MP1 to the aerial socket of your TV set.

Check that the computer **POWER** switch on the right hand end is set to the **OFF** position and then connect the Mains Plug from the MP1 into the Supply Socket.

Now reduce the volume control on your TV set to a minimum, switch on your TV, and then switch on the computer using the slide switch marked **POWER** on the right hand end.

The red **ON** lamp at the top centre of the computer keyboard unit should be illuminated, and you must now tune in your TV set to receive the signal from the computer.
If you have a TV with push-button channel selection, press a channel button to select a spare or unused channel. Adjust the corresponding tuning control in accordance with the TV set manufacturer's instructions (the signal will be approximately at channel 36 if your TV has a marked tuning scale), until you receive a picture that looks like:

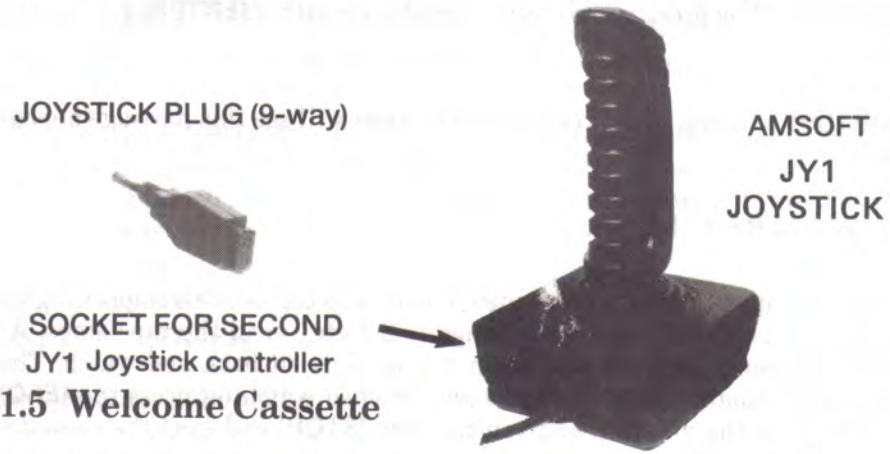


Tune in the TV set accurately until the clearest picture is seen. The writing will be gold/yellow on a deep blue background.

If your TV has a rotary programme selector knob, turn the tuning knob until the above picture appears and remains perfectly steady. (Again, at approximately channel 36).

1.4 JOYSTICK

The AMSOFT joystick model JY1 is an additional item that you may wish to purchase if you are using the CPC464 computer with software games which incorporate the facility for joystick control, and 'firing' within the game. The JY1 can be plugged into the back of your computer using the 9-way socket marked **USER PORTS (I/O)**. The Amstrad CPC 464 computer can be used with two joysticks. The second JY1joystick should be plugged into the socket on the first joystick.



1.5 Welcome Cassette

Packed into one of the polystyrene end caps in your computer packaging, you will have found the 'Welcome' cassette tape. Open the door on the Datacorder unit by pressing the key marked **[STOP/EJECT]**, and then insert the cassette into the Datacorder of the computer as shown in figure 3 - make sure that the **'SIDE 1'** printing is uppermost:

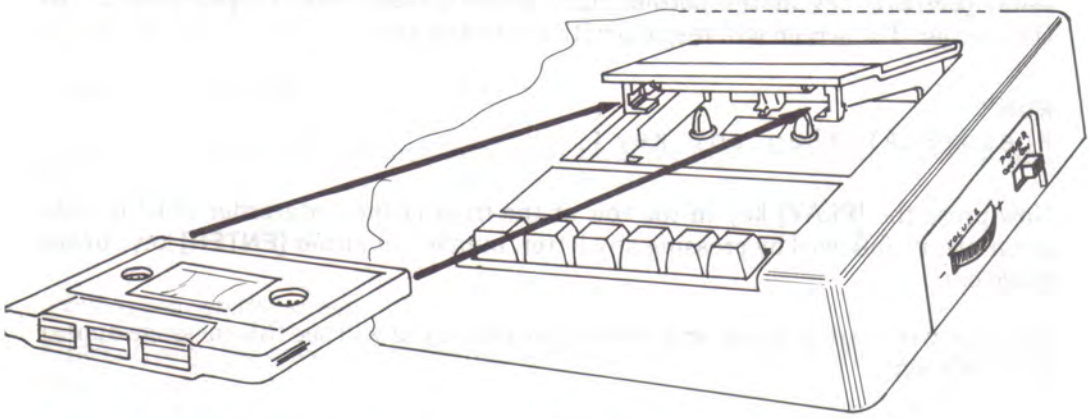


Figure 3 The right way to insert a cassette into the Datacorder

Close the lid until it latches shut, press the **[REW]**ind key on the datacorder to ensure that the tape is rewound to the start. As soon as the tape stops, press the **[STOP/EJECT]** key. Reset the tape counter to 000 by pressing the **COUNTER RESET** button.

Press down the key marked **[CTRL]** (Control), and *AT THE SAME TIME* press the small **[ENTER]** key at the bottom right of the number-only keypad next to the Datacorder. The screen will respond with the instructions:

```
RUN"
Press PLAY then any key:
```

Now press the **[PLAY]** key in the row at the front of the Datacorder until it locks down firmly, followed by pressing any letter, number, or either **[ENTER]** key - or the space bar.

The tape will start to move, and after a short time you will see this message appear on the screen:

```
Loading WELCOME 1 block 1
```

The tape will take approximately 5 minutes to load, you can see this happening on the screen since 'block' number will change to 2,3 etc., until the tape stops. At this time the 'Welcome' program will begin to run. Just sit back and watch. The program runs continuously, so when you have finished watching press the **[ESC]** key twice. This stops the program and you can now **[STOP]** and eject the cassette, then turn it over to run side two.

After turning the cassette over ready for Side 2, once again remember to press the **[REW]**ind button on the datacorder to ensure that the tape is rewound to the start.

Press down the key marked **[CTRL]** (Control), and *AT THE SAME TIME* press the small **[ENTER]** key at the bottom right of the number-only keypad next to the Datacorder. The screen will respond with the instructions:

```
RUN"
Press PLAY then any key:
```

Now press the **[PLAY]** key in the row at the front of the Datacorder until it locks down firmly, followed by pressing any letter, number, or either **[ENTER]** key - or the space bar.

The tape will start to move, and after a short time you will see this message appear on the screen:

```
Loading WELCOME 2 block 1
```

Follow the instructions given on the screen, and the program will invite you to participate by typing instructions as it progresses.

1.6 LOADING OTHER SOFTWARE CASSETTES

THE WELCOME TAPE CAN ONLY BE LOADED AND RUN as described in the previous section (1.5). Unprotected BASIC programs can be loaded by the following alternative methods. Rewind the tape that you have inserted by pressing the **[REW]** key on the datacorder until the spools stop turning, when you must immediately press the **[STOP/EJECT]** key.

Reset the computer to clear the memory by pressing the **[CTRL]**, **[SHIFT]** and **[ESC]** keys in order - but holding each key down until the **[ESC]** key is finally pressed - the screen will clear and the original message will reappear as if you had just switched on.

The expression **[ENTER]** in the following instructions indicates that you must press either of the two keys marked **[ENTER]** - do not type the word ENTER! The " symbol is obtained by pressing either **[SHIFT]** key together with the **2** key on the top row of the keyboard.

Type in:

```
load "" [ENTER]
```

The computer asks you to....

```
Press PLAY then any key:
```

Now press the **[PLAY]** key in the row at the front of the Datacorder until it locks down firmly, followed by pressing any letter, number, or either **[ENTER]** key - or the space bar.

The tape will start to move, and after a short time you will see this message appear on the screen:

```
Loading ‹program name› block 1
```

The block numbers will continue to increase until the tape has finished loading, and the message:

```
Ready
```

...will appear on the screen.

Alternatively, you may specify the name of the program that you wish to load. To do this, type in:

```
load "‹title›" [ENTER]
```

The computer asks you to....

```
Press PLAY then any key:
```

Now press the **[PLAY]** key in the row at the front of the Datacorder until it locks down firmly, followed by pressing any letter, number, or either **[ENTER]** key - or the space bar.

The tape will start to move. If the program you have asked the computer to load is not at the beginning of the cassette, the computer will search through the tape until it finds the exact title that you have asked it to load. Be careful to type in the program title correctly.

If, while searching for your program, the computer finds a different title to the one that you have typed in, you will see this message appear on the screen:

```
Found ‹other title› block 1
```

The computer will not load this program, but will continue to search through the tape until the exact program title that you have typed in is found, or until you press the **[ESC]** key to stop the computer searching the tape.

When the program has been found, you will see this message appear on the screen:

```
Loading ‹title› block 1
```

The block numbers will continue to increase until the tape has finished loading, and the message:

```
Ready
```

...will appear on the screen.

Then type:

```
run [ENTER]
```

...and the program you've just loaded will run. If there was already a program in the memory, this will be discarded and the newly loaded program will take its place.

To run a program *directly*, without first asking the computer to load it, simply type in:

```
RUN "" [ENTER]
```

..the computer will respond with the words:

```
Press PLAY then any key:
```

..after pressing **[PLAY]** followed by any letter, number, the space bar or either **[ENTER]** key, the computer will search for, then load the program and run it without further instructions from the keyboard. You can stop the sequence at any time by pressing **[ESC]** as usual.

1.7 Loading pre-recorded software cassettes

The instructions given so far will allow you to load any of the many titles of software available for the CPC 464 computer.

However, please also refer to the correct loading instructions printed in each software package.

1.8 SAVE

A program can be saved (recorded) for later use. Insert a cassette the correct way and close the cassette door (the record protection tabs in the rear of the cassette must not have been removed). Press **[REW]** to rewind tape to the start, remembering to press the **[STOP/EJECT]** button when the tape stops. Type in:

`save "`*‹program title›*`"` **[ENTER]**

The computer will respond with:

```
Press REC and PLAY then any key:
```

Now press the **[REC]** and **[PLAY]** keys on the Datacorder until they lock down firmly, followed by any key (letter, number, space or **[ENTER]** keys).

The computer will then respond with:

`Saving` *‹program title›* `block 1`

When the program has been saved, the cassette spools will stop and you will see the word: `Ready` appears on the screen. Now press the datacorder **[STOP/EJECT]** key and your program has been saved.

Note that you will not be able to save prerecorded software and games on to your own blank cassette.

Such programs are protected against unauthorised copying.

Foundations 2: KEYBOARD FAMILIARISATION

We will now explain the functions of some of the keys on the computer. Those who are experienced in using computers may skip this section.

[ENTER]

There are two **[ENTER]** keys. Either of these keys enter the information that you have typed into the computer. After the **[ENTER]** key is pressed, a new line is started on the screen. Each instruction that you type in to the computer should be followed by pressing the **[ENTER]** key.

From now on, we will show **[ENTER]** as meaning press the **[ENTER]** key after each instruction or program line..

[DEL]

This key is used to delete a character to the left of the cursor on the screen (for example a letter or a number) which is not required.

Type in `abcd` and you will see that the letter `d` is positioned to the left of the cursor. If you decide that you do not want the letter `d`, press **[DEL]** once and you will see the `d` removed. If you press **[DEL]** and continue to hold it down, the letters `abc` will also be removed.

[SHIFT]

There are two **[SHIFT]** keys. If you press either of these and hold it down whilst typing a character, a capital letter or upper case symbol will appear on the screen.

Type in the letter `a` then hold down the **[SHIFT]** key and type in the letter `a` again. On the screen you will see:

```
aA
```

Now type in a few spaces by holding down the space bar. Try the following using the number keys which are on the top line of the keyboard, above the letter keys. Type in the number `2`, then hold down the **[SHIFT]** key and type in the number `2` again. On the screen you will see:

```
2"
```

You can now see what happens when the **[SHIFT]** key is held down whilst pressing a character key. Experiment by typing any of the character keys, either on their own, or together with the **[SHIFT]** key.

[CAPS LOCK]

This has a similar operation to **[SHIFT]** except that you only have to press it once. From then on each letter that you type in will be in capitals, although the number keys will not be shifted. Press **[CAPS LOCK]** once, then type in:Type in:

```
abcdef123456
```

On the screen you will notice that although all the letters are shifted to capitals, the numbers have not been shifted to symbols. If you wish to type in a shifted symbol while **[CAPS LOCK]** is in operation, simply hold down the **[SHIFT]** key before pressing a character key. Press the following keys while holding down the **[SHIFT]** key:

```
ABCDEF123456
```

On the screen you will see:

```
ABCDEF!"#$%&
```

If you wish to return to small (lower case) characters again, press **[CAPS LOCK]** key once again.

If you wish to type in capital letters *and* shifted upper case symbols without having to constantly hold down the **[SHIFT]** key, this can be carried out by holding down the **[CTRL]** key, then pressing **[CAPS LOCK]** key once. Now type in:

```
abcdef123456
```

On the screen you will see:

```
ABCDEF!"#$%&
```

It is still possible to type in numbers while **[CTRL]** and **[CAPS LOCK]** are in operation, by using the number keys to the right of the main keyboard.

Holding down the **[CTRL]** key and pressing **[CAPS LOCK]** once will return you to the mode that you were previously in - CAPS LOCK or lower case. If you have returned to the CAPS LOCK mode, simply press **[CAPS LOCK]** once again to return to the lower case mode..

[CLR]

This key is used to clear a character within the cursor.

Type in `ABCDEFGH`. The cursor will be positioned to the right of the last letter typed (`H`). Now press the cursor left key **[←]** four times. The cursor will have moved four places to the left, superimposed over the top of the letter `E`.

Notice how the letter E is still visible within the cursor. Press the **[CLR]** key once and you will see that the letter E has been cleared and the letters FGH have each moved one space to the left with the letter F now appearing within the cursor. Now press the **[CLR]** key again and hold it down. You will see how the letter F is cleared followed by the letter G and H.

[ESC]

This key is used to **[ESC]**ape from a function that the computer is in the process of carrying out. Pressing the **[ESC]** key once will cause the computer to temporarily pause in its function, and will continue again if any other key is pressed.

Pressing the **[ESC]** key twice will cause the computer to completely **[ESC]**ape from the function which it is carrying out. The computer is then ready for you to type in some more instructions.

Now press the **[ESC]** key twice..

IMPORTANT

When you reach the right hand edge of the display by entering more than 40 characters on the screen, the next character will automatically appear on the next line at the left edge of the screen. This means that you should **NOT** press **[ENTER]** as those of you accustomed to typewriters might press a carriage return towards the right edge of a page.

The computer does this automatically for you, and will react to an unwanted **[ENTER]** by printing an error message - usually a `Syntax error`, either there and then, or when the program is run.

Syntax Error

If the message: `Syntax error` appears on the screen, the computer is telling you that it does not understand an instruction that you have entered. For example type in:

```
printt [ENTER]
```

On the screen you will see the message:

```
Syntax error
```

The message appears because the computer has not understood the instruction `printt`

If you type a mistake in the line of a program, such as:

```
10 printt "abc" [ENTER]
```

The `Syntax error` message will not appear until the instruction is processed by the computer when the program is run.

Type in:

`run` **[ENTER]**

On the screen you will see:

```
Syntax error in 10
10 printt "abc"
```

This message tells you in which line the error has occurred, and displays the program line together with the editing cursor so that you can correct the mistake.

Press the cursor right key **[→]** until the cursor is over the `t` in `printt`. Now press the **[CLR]** key to remove the unwanted `t` , then press the **[ENTER]** key to enter the corrected line into the computer.

Type in: `run` **[ENTER]** ...and you will see that the computer has accepted the instruction, and has printed `abc`

An introduction to AMSTRAD BASIC keywords

In Chapter 8 you will find an illustrated description of all the BASIC keywords found in the AMSTRAD BASIC. We will introduce some of the more commonly used BASIC keywords in this section.

CLS

Type in: `cls` (clear screen). You can use either upper case (capital) or lower case (small) letters. Then press **[ENTER]**. You will notice that the screen clears and the word `Ready` with the cursor ■ will appear at the top left of the screen.

PRINT

This is used whenever you want characters, words or figures in a program to be printed on the screen. Type in the following instruction line:

`print "hello"` **[ENTER]**

On the screen you will see:

```
hello
```

The quotation marks `""` are used to tell the computer what should be printed. `hello` appeared on the screen as soon as the **[ENTER]** key was pressed. Type in `cls` **[ENTER]** to clear the screen.

RUN

The previous example showed a single line program. Most programs have many lines. In front of each line, a number is first typed in. These numbers tell the computer the order in which to run the program. When **[ENTER]** is pressed, the line is stored in the memory until the program is run. Type:

```
10 print "hello" [ENTER]
```

Notice that when **[ENTER]** was pressed, `hello` was *not* printed on the screen. To do this, the word `run` has to be typed in. Type:

```
run [ENTER]
```

You will now see `hello` on the screen. Note that instead of continually typing in `print`, you can use the `?` symbol, for example:

```
10 ? "hello" [ENTER]
```

LIST

After a program has been stored in the memory, it is possible to check what has been typed in by listing the program. Type:

```
list [ENTER]
```

and on the screen you will see:

```
10 PRINT "hello"
```

which is the program stored in the memory.

Notice how the word `PRINT` is now in capitals. This means that the computer has accepted `PRINT` as a known BASIC keyword.
Type in `cls` **[ENTER]** to clear the screen. Note that although the screen is cleared when you type in `cls` **[ENTER]**, your program is not erased from the computer's memory.

GOTO

The `GOTO` keyword tells the computer to go from one line to another in order to either miss out a number of lines or to form a loop. Type this in:

```
10 print "hello" [ENTER]
20 goto 10 [ENTER]
```

Type:

```
run [ENTER]
```

and you will see `hello` printed continuously on the screen, one under another on the left side. To stop this program running, press **[ESC]** once. To start it again, press any other key. To stop it running so that other instructions can be typed in, press **[ESC]** twice.

Type in

```
cls [ENTER]
```

to clear the screen.

To see the word `hello` printed continuously on each line, one next to another filling the whole of the screen, type in the previous program but with a semi-colon `;` after the quotation marks " Type:

```
10 print "hello"; [ENTER]
20 goto 10 [ENTER]
run [ENTER]
```

Note that the semicolon `;` tells the computer to print the next character immediately following the previous one. Escape from this program by pressing **[ESC]** twice. Now type in line 10 again, but this time use a comma `,` instead of a semi-colon `;`

```
10 print "hello", [ENTER]
run [ENTER]
```

You will now see that the comma `,` tells the computer to print the next character (or group of characters) 13 columns away from the beginning of the first group of characters. This feature is useful for displaying information in separate columns. Note, however, that if the number of characters in a group exceeds 12, the next character will be displaced forward by another 13 columns.

Again, to escape from this program press **[ESC]** twice. To clear the computer's memory completely, hold down the **[SHIFT] [CTRL]** and **[ESC]** keys in that order, and the computer will reset.

INPUT

This command is used to let the computer know that it is expecting something to be typed in, for example, the answer to a question. Type the following:

```
10 input "what is your age"; age [ENTER]
20 print "you look younger than"; age "years
   old." [ENTER]
run [ENTER]
```

On the screen you will see:

```
what is your age?
```

Type in your age then **[ENTER]** If your age was 18, the screen would then show:

```
you look younger than 18 years old.
```

This example shows the use of the `input` command and a number variable. The word `age` was put into the memory at the end of line 10 so that the computer would associate the word `age` with whatever numbers were typed in and would `print` these numbers where the word `age` is on line 20. Although we used the expression `age` in the above for the `age` variable, we could have just as easily used a letter, for example `b`.

Reset the computer to clear the memory. (**[CTRL] [SHIFT]** and **[ESC]** keys). If you had wanted an `input` made up of any characters (letters or letters and numbers), the dollar sign `$` *must* be used at the end of the variable.

Type in the following program: (Note that in line 20 you must put a space after the `o` in `hello` and before the `m` in `my`).

```
10 input "what is your name"; name$ [ENTER]
20 print "hello "; name$" my name is Arnold" [ENTER]
run [ENTER]
```

On the screen you will see:

```
what is your name?
```

Type in your name then **[ENTER]**

If your name was Fred you will see on the screen:

```
hello Fred my name is Arnold
```

(As a point of interest, *Arnold* was the codename for the AMSTRAD CPC464 during its development.)

Although we used `name$` in the above for the name string variable, we could have just as easily used a letter, for example `a$`. Now we will combine the above 2 examples into one program.

Reset the computer by pressing **[CTRL] [SHIFT]** and **[ESC]**. Type in the following:

```
5 cls [ENTER]
10 input "what is your name"; a$ [ENTER]
20 input "what is your age"; b [ENTER]
30 print "I must say ";a$ " you dont look";
   b"years old" [ENTER]
run [ENTER]
```

In this program we have used 2 variables, `a$` for the name and `b` for the age. On the screen you will see:

```
what is your name?
```

Now type in your name (e.g. Fred) then **[ENTER]**. You will then be asked:

```
what is your age?
```

Now type in your age (e.g.18), then **[ENTER]**. If your name were Fred and your age 18, on the screen you will see:

```
I must say Fred you dont look 18 years old
```

EDITING A PROGRAM

If any of the lines in the program had been typed incorrectly, resulting in a `Syntax error` or other error message, it would be possible to edit that line rather than type it out again. To show this you can type the previous program out incorrectly.

Type in the following:

```
5 clss
10 input"what is you name"; a$ [ENTER]
20 input"what is your age"; b [ENTER]
30 print"I must say"; a$" you dont look";
b"years of age" [ENTER]
```

There are 3 mistakes in the above program:

In line 5 we typed in `clss` instead of `cls`.
In line 10 we typed in `you` instead of `your`.
In line 30 we forgot to put a space between `say` and the quotation marks ".

There are 3 methods of editing a program. The first is to simply type in the new line again. When a line is retyped and entered it replaces the same numbered line currently in the memory. Secondly, there is the *edit* method and finally there is the *copy cursor* method.

EDIT METHOD

To correct the mistake in line 5, type:

```
edit 5 [ENTER]
```

Line 5 will then be printed with the cursor superimposed over the `5`. To edit out the extra `s` in `clss`, press the cursor right key **[→]** until the cursor appears over the last `s`, then press the **[CLR]** key. You will see the `s` disappear. Now press;

[ENTER] and line 5 will now be corrected in the memory. Type in:

```
list [ENTER]
```

to check line 5 is now correct.

COPY CURSOR METHOD

To correct the mistakes in line 10 and 30, hold down **[SHIFT]** key then press the cursor up key **[↑]** until the *copy cursor* is positioned over the very beginning of line 10. You will notice the main cursor has not moved, so there are now two cursors on the screen. Now press the **[COPY]** key until the copy cursor is positioned over the space between `you` and `name`. You will notice that line 10 is being re-written on the last line and the main cursor stops at the same place as the copy cursor. Now type in the letter `r`. This appears on the bottom line only.

The main cursor has moved but the copy cursor stayed where it was. Now press the **[COPY]** key until the whole of line 10 is printed. Press **[ENTER]** and this new line 10 will be stored in the memory. The copy cursor disappears and the main cursor positions itself under the new line 10. To correct the second mistake, hold down **[SHIFT]** and press cursor up key **[↑]** until the copy cursor appears over the very beginning of line 30.

Press **[COPY]** until the copy cursor is positioned over the quotations next to `say`. Now press the space bar once. A space will be inserted on the *bottom* line. Hold down the **[COPY]** key until the whole of line 30 is printed. Then press **[ENTER]**. You can now list the program to check that it is corrected in the memory by typing: `list` **[ENTER]**.

Now reset the computer by pressing **[CTRL]** **[SHIFT]** and **[ESC]** keys.

IF THEN

We will now extend the previous program with the use of the `if` and `then` comands.

Type in the following : Notice that we have added two symbols. < means *less than* and is next to the **M** key, > means *greater than* and is next to the < (less than) key.

```
5 cls [ENTER]
10 input "what is your name";a$ [ENTER]
20 input "what is your age";age [ENTER]
30 if age < 13 then 60 [ENTER]
40 if age < 20 then 70 [ENTER]
50 if age > 19 then 80 [ENTER]
60 print"So "a$" You are not quite a teenager
   at"age"years old":end [ENTER]
70 print "So "a$" you are a teenager at"
   age"years old":end [ENTER]
80 print "Oh well"; a$" You are no longer
   a teenager at"age"years old" [ENTER]
```

To check this program is correct, type:

`list` **[ENTER]**

Now type in :

`run` **[ENTER]**

Now answer the questions prompted by the computer and see what happens.

You can now see what effect the `if` and `then` commands have in a program. We have also added the word `end` at the end of line 60 and 70. The keyword `end` is used literally to *end* the running of a program. If `end` wasn't there in line 60, the program would continue to run, and carry out lines 70 and 80.

Likewise, if `end` wasn't there in line 70, the program would continue to run, and carry out line 80. The colon `:` before the word `end` separates it from the previous instruction. Colons `:` can be used to enable you to put two or more instructions on one program line. We have also added line 5 to clear the screen at the start of the program. We will do this from now on in the following programs to make things look neater.

Other keywords associated with `IF` and `THEN` include `ELSE`, `OR` and `GOTO`. The use of these words in the `IF` command will become clearer as you work through this guide.

Reset the computer by pressing **[CTRL]** **[SHIFT]** and **[ESC]** keys.

FOR TO NEXT

We will now show the use of the `FOR`, `TO` and `NEXT` commands. For this example we will show a way the computer can print the 12 times table (1x12, 2x12, 3x12, etc)

Type the following, note that the * symbol means multiply.

```
5 cls [ENTER]
10 for a = 1 to 20 [ENTER]
20 print a"* 12 ="a*12 [ENTER]
30 next a [ENTER]
run [ENTER]
```

You will notice the columns look untidy, so type in:

```
5 cls [ENTER]
10 for a = 1 to 9 [ENTER]
20 print a" * 12 ="a*12 [ENTER]
30 next a [ENTER]
40 for a = 10 to 20 [ENTER]
50 print a"* 12 ="a*12 [ENTER]
60 next a [ENTER]
run [ENTER]
```

Try this program for multiplication tables using other numbers. For example, if you wanted to see the 17 times table, replace the number 12 in lines 20 and 50 with the number 17. Finally, reset the computer using the **[CTRL][SHIFT]** and **[ESC]** keys. Note that it is possible to specify the value of each step taken in the `FOR TO NEXT` command by use of the `STEP` command. For further information, study the description of `FOR` in Chapter 8.

SIMPLE ARITHMETIC

Your CPC464 can be used as a calculator quite easily.

To understand this carry out the following examples. We will use the ? symbol instead of typing print in this section. The answer will be printed as soon as the **[ENTER]** key is pressed.

ADDITION

(use **[SHIFT]** and **;** keys for plus)

Type:

```
?3+3 [ENTER]
 6
```

(note that you do not type in the equals sign =)

Type:

```
?8+4 [ENTER]
 12
```

SUBTRACTION

(use unshifted = key for minus)

Type:

```
?4-3 [ENTER]
 1
```

Type:

```
?8-4 [ENTER]
 4
```

MULTIPLICATION

(Use **[SHIFT]** and **:** keys for multiply (* means ×)

Type:

```
?3*3 [ENTER]
 9
```

Type:

```
?8*4 [ENTER]
 32
```

DIVISION

(use unshifted **?** key for divide (/ means ÷)

Type:

```
?3/3 [ENTER]
 1
```

Type:

```
?8/4
 2
```

SQUARE ROOT

To find the square root of a number type in `sqr ( )`. The number you want the square root of should be typed inside the brackets.
Type:

```
?sqr(16) [ENTER] (this means √16)
 4
```

Type:

```
?sqr (100) [ENTER]
 10
```

EXPONENTIATION

(use unshifted £ for exponentiation)
Exponentiation is when a number is raised to a power of another.
For example 3 squared (3^2),3 cubed (3^3) etc. Type:

```
?3↑3 [ENTER] (this means 3³)
 27
```

Type:

```
?8↑4 [ENTER] (this means 8⁴)
 4096
```

CUBE ROOT

You can quite easily calculate cube roots by using a similar method to the last example under square roots.
To find the cube root of 27 ($\sqrt[3]{27}$). Type:

```
?27↑(1/3) [ENTER]
 3
```

To find the cube root of 125
Type:

```
?125↑(1/3) [ENTER]
 5
```

MIXED CALCULATIONS (+,-,*,/,MOD,\)

Mixed calculations are understood by the computer, but they are calculated within certain priorities.

First priority is given to multiplication and division, then addition and subtraction. These priorities apply only to calculations containing only these four operations. The priorities will be further explained later and will include exponentiation etc.

If the calculation was:

```
3+7-2*7/4
```

You may think this would be calculated as

```
3+7-2 *7 /4
=    8 *7 /4
=      56/4
=      14
```

In fact it is calculated as

```
3+7-2*7/4
=3+7-14/4
=3+7-3.5
=10-3.5
=6.5
```

Prove this by typing in this calculation as it is written:

Type:

```
?3+7-2*7/4 [ENTER]
 6.5
```

You can change the way the computer calculates this by adding brackets. The computer will deal with the calculation inside brackets prior to the multiplication etc, outside the brackets. Prove this by typing in the calculation including brackets.

Type:

```
?(3+7-2)*7/4 [ENTER]
 14
```

FURTHER EXPONENTS

If you want to use very large or very small numbers in calculations, it is sometimes useful to use *scientific notation*. The letter E is used for the exponent of numbers to the base 10. You may use either lower case e or upper case E.

For example 300 is the same as $3x10^2$. In scientific notation, this is 3E2. Similarly, 0.03 is the same as $3x10^{-2}$). In scientific notation this is 3E-2. Try the following examples.

1. 30x10

You can type in:

```
?30*10 [ENTER]
 300
```

or you can type:

```
?3e1*1e1 [ENTER]
 300
```

2. 3000x1000

Type:

```
?3e3*1e3 [ENTER]
 3000000
```

3. 3000x0.001

Type:

```
?3e3*1e-3 [ENTER]
 3
```

Foundations 3: Graphics, modes and sound

The Amstrad CPC 464 Colour Personal Computer has three *modes* of screen display operation: Mode 0, Mode 1, and Mode 2.

When the computer is first switched on, it is automatically in Mode 1.

To understand the different modes, switch on the computer and press the number **1** key. Hold it down until two lines are full of number 1's. If you now count the number of 1's on a line, you will see that there are 40. This means that in mode 1, there are 40 columns. Press **[ENTER]** - you will get a `Syntax error` message, but don't worry, this is just a quick way of getting back to the `Ready` message that tells you the computer is waiting for your next instruction.

Now type in: `mode 0` **[ENTER]**

You will see the characters on the screen are now larger. Press the number **1** key again and hold it down until two lines are full of 1's. If you count the number of 1's on a line, you will see there are 20. This means that in mode 0, there are 20 columns. Press **[ENTER]** again.

Now type in: `mode 2` **[ENTER]**

You will see that this is the smallest mode, and if you type in a row of 1's, you will count 80. This means that in mode 2 there are 80 columns. To recap:

Mode 0 = 20 columns
Mode 1 = 40 columns
Mode 2 = 80 columns

Finally, press **[ENTER]** once more.

COLOURS

There is a choice of 27 colours. These are shown on a green monitor (GT 64) as various shades of green. If you purchased the GT 64 monitor, you can buy the AMSTRAD MP1 Modulator/Power supply in order to use the computer's colour facilities on your domestic colour T.V.

In Mode 0, up to 16 of the 27 available colours can be put on to the screen at any time.
In Mode 1, up to 4 of the 27 colours can be put on to the screen at any time.
In Mode 2, up to 2 of the 27 colours can be put on to the screen at any time.

You are able to change the colour of the `BORDER`, the `PAPER` (the area where the characters can appear) or the `PEN` (the character itself), all independently of each other.

The 27 colours available are listed in Table 1, each with their INK reference number.

MASTER COLOUR CHART

Ink Number	Colour/Ink	Ink Number	Colour/Ink
0	Black	14	Pastel Blue
1	Blue	15	Orange
2	Bright Blue	16	Pink
3	Red	17	Pastel Magenta
4	Magenta	18	Bright Green
5	Mauve	19	Sea Green
6	Bright Red	20	Bright Cyan
7	Purple	21	Lime Green
8	Bright Magenta	22	Pastel Green
9	Green	23	Pastel Cyan
10	Cyan	24	Bright Yellow
11	Sky Blue	25	Pastel Yellow
12	Yellow	26	Bright White
13	White		

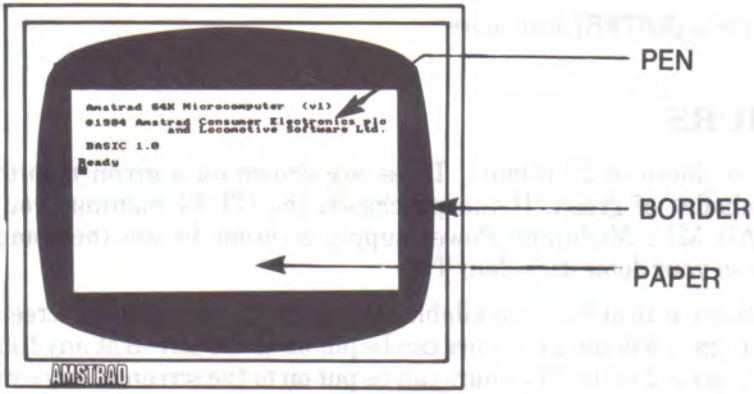
Table one: The INK numbers and colours

As explained earlier, when the computer is first switched on, it is in Mode 1. To return to Mode 1, type in:

```
mode 1 [ENTER]
```

The colours of BORDER, PAPER, and PEN at switch-on are:

Border: Colour number 1 (blue)
Paper (screen): Colour number 1 (blue)
Pen (characters): Colour number 24 (bright yellow)



The BORDER is the area surrounding the PAPER. (Note that when the computer is first switched on, the BORDER and PAPER are both blue). The PAPER is the area of the screen inside the BORDER where characters can appear. The PEN is the colour of the characters.

To explain this further, we can associate the PEN and PAPER on the monitor screen to an actual PEN and piece of notepaper. As the colour of the INK in a PEN can be changed, so the colour of the characters on the screen can be changed. As the colour of notepaper can be changed, so the colour of the PAPER on the screen can also be changed.

To change the colour of the BORDER, type in:

```
border 0 [ENTER]
```

You will see the BORDER colour change from blue to black. If you refer to Table 1, you will see that 0 is black. The BORDER can be changed to any of these colours by typing in: border then the colour number required.

Now type in:

```
cls [ENTER]
```

to clear the screen.

To see the PAPER colour change, type in:

```
paper 2 [ENTER]
```

You will see the background colour behind the word Ready change to bright cyan. Now type in:

```
cls [ENTER]
```

to clear the screen again to the new PAPER colour.

To see the PEN colour change, type in:

```
PEN 3 [ENTER]
```

You will see that the PEN colour has changed and the word Ready is printed in bright red,

Now type in:

```
cls [ENTER]
```

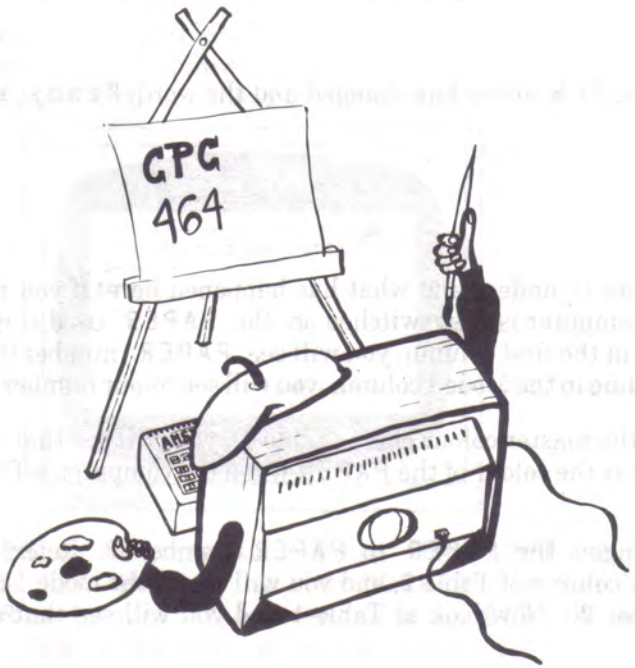
You will find it easier to understand what has happened here if you refer also to Table 2. When the computer is first switched on, the PAPER used is number 0. If you look at Table 2, in the first column, you will see PAPER number 0. If you now look along the same line in the Mode 1 column, you will see colour number 1.

If you now refer to the master colour chart (Table 1), you will see that number 1 is equal to blue, which is the colour of the PAPER when the computer is first switched on.

We have just changed the PAPER to PAPER number 2. Refer to PAPER number 2 in the left column of Table 2, and you will see in the mode 1 column that this is colour number 20. Now look at Table 1 and you will see that colour 20 is bright cyan.

Paper/Pen No.	Ink Colour Mode 0	Mode 1	Mode 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	Flashing 1,24	20	1
15	Flashing 16,11	6	24

Table two: `PAPER/ PEN/MODE/INK` reference



The PEN used when the computer is first switched on, is number 1. Look at Table 2 and you will see in the mode 1 column, that PEN number 1 is colour number 24. If you now refer to Table 1, you will see that colour number 24 is bright yellow, which is the colour of the characters (PEN) when the computer is first switched on.

We have just changed the PEN to number 3. Looking at Table 2, you will see that PEN number 3 is colour number 6 in mode 1. Now refer to Table 1, and you will see that colour 6 is bright red.

At the moment, we are using PAPER number 2 and PEN number 3. We can now go on to change the colour of these. We do this by using the INK command. The INK command has two numbers, the first is the number of the PEN or PAPER to be changed, the second is the colour that the PEN or PAPER is to be changed to. Look at Table 1 for the colour numbers. As an example, we will show how to change the colour of PAPER 2 to black and the colour of PEN 3 to bright white.

In Table 1, you will see that the colour number for black is 0, and the colour number for bright white is 26.

Now type in:

```
ink 2,0 [ENTER]
```

(As explained, 2 is the current PAPER number, and 0 is black.)

Now type in:

```
ink 3,26 [ENTER]
```

(3 is the current PEN number and 26 is bright white)

Now fully reset the computer by pressing **[CTRL] [SHIFT]** and **[ESC]** keys.

As explained earlier, when the machine is first switched on, or reset using **[CTRL] [SHIFT] [ESC]**, the PAPER number is 0 and the PEN number is 1. The colour of the PAPER is 1 (blue) and the colour of the PEN is 24, (bright yellow). These would be typed in as `ink 0,1` for the PAPER, and `ink 1, 24` for the PEN. To change these immediately to white characters (PEN) on a black background (PAPER), type in:

```
ink 0,0 [ENTER]
```

Then type in:

```
ink 1,26 [ENTER]
```

FLASHING COLOURS

It is possible to make the colour of the characters flash between one colour and another. This can be achieved by adding an extra colour number to the INK command of the PEN.

To see the characters on the screen flashing between bright white and bright red, type in:

```
ink 1,26,6 [ENTER]
```

In this case, 1 is the PEN number, while 26 is the colour bright white, and 6 is the alternate colour, bright red.

It is also possible to make the colour of the PAPER behind the characters flash between one colour and another. This can be achieved by adding an extra colour number to the INK command for the PAPER.

To see the PAPER flashing between green and bright yellow, behind the characters, type in:

```
ink 0,9,24 [ENTER]
```

In this case 0 is the PAPER number. while 9 is the colour green, and 24 is the alternate colour, bright yellow.

Now reset the computer, **[CTRL] [SHIFT] [ESC]**

Note that in mode 0, two of the PENs (numbers 14 and 15), together with two of the PAPERs (numbers 14 and 15) are default flashing colours. In other words, it is not necessary to add an extra number to the INK command.

type in the following:

```
mode 0 [ENTER]
pen 15 [ENTER]
```

on the screen you will see the word Ready flashing between sky blue and pink.

Now type in:

```
paper 14 [ENTER]
cls [ENTER]
```

You will now see that in addition to the word Ready flashing between sky blue and pink, the background PAPER is also flashing between yellow and blue.

It is possible to change these default flashing colours by typing in a new INK command for the PEN or PAPER. To change the colour of the PEN to flashing black and bright white, type in:

```
ink 15,0,26 [ENTER]
```

In this case 15 is the PEN number, while 0 is the colour black, and 26 is the alternate colour bright white.

Finally, it is possible to make the BORDER flash between two colours by adding an extra colour number to the BORDER command. Type in:

```
border 6,9 [ENTER]
```

You will now see that the BORDER is flashing between bright red and green.

Now reset the computer **[CTRL] [SHIFT] [ESC]**

DEMONSTRATION PROGRAM

For further demonstration of the colours available, type in the following program, then run it.

We have included some sound in the program. This will be explained in a later section.

```
 10 mode 0: ink 0,2:ink 1,24: paper 0 [ENTER]
 20 pen 1: for b=0 to 26: border b [ENTER]
 30 locate 3,12:print"BORDER COLOUR";B [ENTER]
 40 sound 4,(40-b) [ENTER]
 50 for t=1 to 600:next t:next b:cls [ENTER]
 60 for p=0 to 15:paper p:pen 5:print "paper";
    p:print [ENTER]
 70 for n=0 to 15:pen n:print "pen";n [ENTER]
 80 sound 1,(n*20+p) [ENTER]
 90 for t=1 to 100:next t:next n [ENTER]
100 for t=1 to 1000:next t:cls:next p [ENTER]
110 cls:paper 0: pen 1:locate 7,12:print
    "THE END": for t=1 to 2000: next t [ENTER]
120 mode 1: border 1:ink 0,1:ink 1,24:paper 0:
    pen 1 [ENTER]

run [ENTER]
```

GRAPHICS

From this point on, we will not ask you to press the **[ENTER]** key after each line. We will just assume that you will do it automatically.

There are a number of character symbols in the computer's memory. To print any one of these, use the key word `chr$( )`. Inside the brackets should be the symbol number, which is in the range from 32 to 255.

Press**[CTRL][SHIFT]** and **[ESC]** to reset the computer, then type in:

```
print chr$(250)
```

On the screen you will see character number 250, which is a man walking to the right.

To see all the characters and symbols appear on the screen with their associated number, type in the following program, remembering to press **[ENTER]** after each line.

```
10 for n=32 to 255: print n;chr$(n);
20 next n
run
```

For your reference, the range of characters together with their respective reference numbers, appear in Appendix III at the back of this book.

LOCATE

This command is used to reposition the character cursor to a specified part of the screen. Unless changed by the `locate` command, the character cursor starts at the top left corner of the screen, which corresponds to x, y co-ordinates 1,1 (x is the horizontal position and y is the vertical position). In mode 1 there are 40 columns and 25 lines. To position a character in the centre of the top line in mode 1, we would use 20,1 as the x,y co-ordinates.

To see this, type in: (remember to **[ENTER]** each line)

```
mode 1
```
.....screen clears, cursor moves to top left

```
10 locate 20,1
20 print chr$ (250)
run
```

Just to prove that this is on the top line, type in:

```
border 0
```

The BORDER will now be black and you will see the man at the middle of the top line of the screen.

In mode 0, there are only 20 columns, but the same 25 lines. If you now type in:

```
mode 0
run
```

You will see that the man now appears at the top right corner of the screen. This happens because the x co-ordinate 20, is the last column in mode 0.

In `mode 2`, there are 80 columns and 25 lines. Using the same program, you will probably be able to guess where the man will appear. Type in:

```
mode 2
run
```

return to `mode 1` by typing in:

```
mode 1
```

Now experiment for yourself, modifying the `locate` and `chr$ ( )` numbers to position various characters anywhere on the screen. Just for example, type in:

```
locate 20,12:print chr$ (240)
```

You will see an arrow in the centre of the screen. Note that in this instruction

20 was the horizontal (x) co-ordinate (in the range 1 to 40)
12 was the vertical (y) co-ordinate (in the range 1 to 25)
240 was the character symbol number (in the range 32 to 255)

To get the character symbol 250 to be repeated across the screen, type in the following program:

```
5 cls
10 for x = 1 to 39
20 locate x,20
40 print chr$ (250)
50 next x
60 goto 5
run
```

Press **[ESC]** key twice to break

In order to remove the previous character from the screen before printing the next character, type in:

```
40 print " "; chr$(250)
```

(This new line 40 automatically replaces the line previously typed as line 40.)

Now type in:

```
run
```

To improve the movement of the character across the screen, add the following line:

```
30 call &bd19
```

This program can be further enhanced to improve the movement by adding some delay loops and by using a different returning character symbol.

Type in:

```
list
```

Now add the following lines to the program:

```
60 for n = 1 to 300 : next n
65 for x = 39 to 1 step-1
70 locate x,20
75 call &bd19
80 print chr$ (251);" "
85 next x
90 for n = 1 to 300:next n
95 goto 10
run
```

Try this interesting small program. We have added some other commands that will be explained in later chapters. For now, just type in:

```
new
10 mode 1
20 locate 21,14:print chr$(244)
30 tag
40 for x=0 to 624 step 2
50 mover -16,0
60 if x<308 or x>340 then y=196:goto 90
70 if x<324 then y=x-104:goto 85
80 y=536-x
85 sound 1,0,20,7
90 move ox, oy:print " ";:ox=x:oy=y
100 move x,y
110 if (x mod 4) = 0 then print chr$(250);
    else print chr$(251);
120 for n=1 to 4: call &bd19:next n
130 next x
140 tagoff
150 goto 20
run
```

PLOT

Unlike the locate command, plot is used to determine the position of the graphics cursor, using pixel co-ordinates. (A pixel is an extremely small segment of the screen).

Note that the graphics cursor is not visible and is different to the character cursor.

There are 640 horizontal pixels by 400 vertical pixels. The x,y co-ordinates are positioned with respect to the bottom left corner of the screen, which has x,y co-ordinates of 0,0. Unlike the locate command used for characters, the co-ordinates do not differ in modes 0,1, or 2.

To see this, first reset the computer using **[CTRL] [SHIFT] [ESC]**, then type in (remembering to **[ENTER]**) :

```
plot 320,200
```

A small dot will appear in the centre of the screen.

Now change the mode by typing:

```
mode 0
plot 320,200
```

You will see the dot is still in the centre but is now larger. Change the mode again and type in the same command to see the effect in mode 2. Type in:

```
mode 2
plot 320,200
```

The dot is still in the centre, but it is now much smaller.

Plot several dots over the screen in various modes, in order to accustom yourself with this command. When you have finished, return to mode 1 and clear the screen by typing in:

```
mode 1
```

DRAW

First reset the computer using **[CTRL][SHIFT]** and **[ESC]**. The draw command draws a line from the current graphics cursor position. To see this in more detail, draw a rectangle on the screen by using the following program. We start by repositioning the graphics cursor with a plot command. Then drawing a line from the graphics cursor position, up towards the top left corner, then from here to the right corner etc.

Type in:

```
5 cls
10 plot 10,10
20 draw 10,390
30 draw 630,390
40 draw 630,10
50 draw 10,10
60 goto 10
run
```

Press **[ESC]** twice to break from this program.

Now add the following lines to the program, to draw a second rectangle inside the first. Type in:

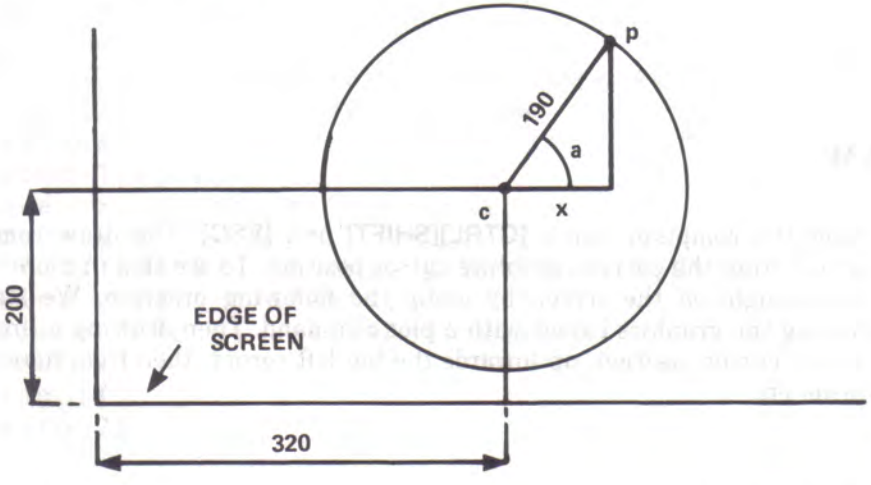
```
60 plot 20,20
70 draw 20,380
80 draw 620,380
90 draw 620,20
100 draw 20,20
200 goto 10
run
```

Press **[ESC]** twice to break from this program.

CIRCLES

Circles can either be plotted or drawn. One method of forming a circle is to plot the x,y, co-ordinates of each point on the circumference of a circle. Refer to the figure below and you will see that point p on the circumference can be plotted using x and y co-ordinates. These are:

x=190*cos(a) and y=190*sin(a)



NEW

We have now started to use the keyword `new` before typing in the program itself. This tells the computer to fully clear the memory in a similar way to reset (**[CTRL] [SHIFT]** and **[ESC]** keys). However, it differs in that the screen is not cleared; only the memory. This is useful if you want to keep the old program on the screen for reference when writing a new program.

Plotting the points of a circle.

In the previous program we plotted points with respect to the bottom left corner of the screen. If we now wanted to position a circle in the centre of the screen we would have to plot the centre of the circle at co-ordinates 320,200, then position all points of the circle relative to the centre position, by adding on the centre position co-ordinates.

A program to draw a circle would then be like this. Type in:

```
new
5 cls
10 for a=1 to 360
15 deg
20 plot 320,200
30 plot 320+190*cos(a),200+190*sin(a)
40 next
run
```

The radius of the circle can be reduced by lowering the `190` figure (190 refers to pixels).

To see the effect of the circle being plotted differently (in radians), delete line 15 from the program by typing in:

```
15
```

To see a solid circle drawn by lines from the centre, edit line 30, replacing the word `plot` with the word `draw`. Line 30 will then be:

```
draw 320+190*cos(a), 200+190*sin(a)
```

Try this with and without line 15 again.

You will note that line 40 of this program is `next` instead of `next a`

It is permissable to simply type in `next` in programs. The computer will work out which `for` expression the `next` is to be associated with. In programs where there are numerous `for` and `next` loops you may wish to add the appropriate name after the word `next` in order to identify the `next` statement when studying the program.

ORIGIN

In the previous program we used the plot command to plot the centre of a circle, then added the x,y co-ordinates to this centre position. Instead of adding these centre co-ordinates to the point plotted, we can use the origin command. This will position the centre of the circle, then position the x,y co-ordinates of all the points on the circumference (in 1 degree steps) from the origin. To see this type in:

```
new
5 cls
10 for a = 1 to 360
15 deg
20 origin 320,200
30 plot 190*cos(a),190*sin(a)
40 next
run
```

Again, you can alter lines 15 or 30 to remove `deg` or `draw` the solid circle from the centre.

To plot four smaller circles on the screen, type in the following program:

```
new
5 cls
10 for a=1 to 360
15 deg
20 origin 196,282
30 plot 50*cos(a),50*sin(a)
40 origin 442,282
50 plot 50*cos(a),50*sin(a)
60 origin 196,116
70 plot 50*cos(a),50*sin(a)
80 origin 442,116
90 plot 50*cos(a),50*sin(a)
100 next
run
```

Once again, you can remove line 15 and modify lines 30,50,70 and 90 to use the `DRAW` command.

GOSUB RETURN

If there are a set of instructions within a program which are to be carried out a number of times, these instructions can be typed as a sub-routine, and can be called into action by the command `gosub` followed by the line number.

The end of a `gosub` routine is marked by typing in the instruction `return`. At this point, the computer will return to the line that followed the `gosub` command which it had just obeyed.

In the previous program, the instruction plot 50*cos(a),50*sin(a) was repeated 4 times. This instruction can be typed in as a sub-routine; and called into action each time it is needed by using the word gosub. Type in the following:

```
new
5 cls
10 for a=1 to 360
15 deg
20 origin 196,282
30 gosub 120
40 origin 442,282
50 gosub 120
60 origin 196,116
70 gosub 120
80 origin 442,116
90 gosub 120
100 next
110 end
120 plot 50*cos(a),50*sin(a)
130 return
run
```

Note that the instruction end is used in line 110; otherwise the program would naturally continue after instruction 100, and carry out instruction 120, which is only required when called by gosub.

To conclude this section, try the following program which incorporates a lot of the programming commands and keywords that you should now understand. Type in:

```
new
10 mode 0:border 6:paper 0:ink 0,0
20 gosub 160:for x=1 to 19:locate x,3
30 pen 15:print" ";chr$(238)
40 for t=1 to 50:next t:sound 2,(x+100)
50 next x:gosub 160:for b=3 to 22
60 locate 20,b:pen 7:print chr$(252)
70 cls:gosub 160:next b
80 sound 2,0,100,15,0,0,1
90 gosub 160:border 16,24:locate 20,25
100 pen 14:print chr$(253);
110 for t = 1 to 1000:next t
120 border 6:gosub 160:for f=3 to 24
130 locate 10,(25-f):pen 2
140 print chr$(144):cls:gosub 160
150 sound 7,(100-f),5:next f:goto 10
160 locate 10,25: pen 12
170 print chr$(239):return
run
```

SOUND

Sound effects are generated by a loudspeaker within the computer itself. If you are using the MP1 modulator power supply and a domestic television, turn the TV's volume control to a minimum.

The level of sound can be adjusted by use of the **VOLUME** control on the right hand end of the computer. The sound can also be fed to the auxiliary input socket of your stereo system, using the **(I/O)** socket at the left hand end of the computer back panel. This will enable you to listen to the sound generated by the computer in stereo, through your hi-fi loudspeakers or headphones.

The `SOUND` command has seven parameters. The first two of these must be used, the rest are optional. The command is typed in as:

`SOUND` channel status, tone period, duration, volume, volume envelope, tone envelope, noise period.

In the following examples, we will type in `1` as the channel status - in other words, its reference number.

TONE PERIOD

Refer to Appendix VII and you will see that the note middle c, has a tone period of 478. Type in:

```
new
10 sound 1,478
run
```

You will hear a short note which is middle c lasting 0.2 second.

DURATION

When no duration of the `sound` is specified it will last 0.2 second. The unit of duration is in 0.01 second. To make a note last 1 second, 100 would be used; to last 2 seconds, 200 would be used. Type in:

```
10 sound 1,478,200
run
```

You will hear the note middle c lasting 2 seconds.

VOLUME

This number specifies the starting volume of a note. The number is in the range 0 to 7.If however, a volume envelope is specified, the range is extended from 0 to 15. If no number is used, 4 is assumed. Type in:

```
10 sound 1,478,200,3
run
```

Note the volume of this `sound`. Now type it in using a higher volume number:

```
10 sound 1,478,200,7
run
```

You will now hear that this is much louder.

VOLUME ENVELOPE

The volume envelope command is `env`. This normally has 4 parameters: The last 3 parameters may appear in any of up to 5 optional envelope sections available. We are only using one of these here. Further explanation will appear in chapter 6.

`env` envelope number, number of steps, amplitude (size) of step, step time.

ENVELOPE NUMBER

This is the number given to a particular envelope so that it can be specified in the sound command. The range of envelope numbers is 0 to 15.

NUMBER OF STEPS

This is used in conjunction with the step time. For example, you may wish to have 10 steps of 1 second each. In such a case the number of steps is 10. The range of step numbers is 0 to 127.

AMPLITUDE OF STEP

Each step can vary in amplitude from a level of 0 to 15 with respect to the last step. The 15 volume levels are the same as those in the sound command. However, the step can be adjusted from -128 to +127 so that you can not only vary the amplitude up or down in the obvious way, but can vary it by using numbers higher than 15 to give some strange effects. The range of amplitude of step numbers is -128 to +127

STEP TIME

This number specifies the time between steps in 0.01 second, (1/100th's of a second) units. The range of step time numbers is 0 to 256. The longest time between steps is therefore 2.56 seconds.

To experiment with the volume envelope, type in the following program:

```
5 env 1,10,1,100
10 sound 1,284,1000,1,1
run
```

Line 10 specifies a sound with a tone period of 284 (international a), lasting for 10 seconds with a start volume of 1, and using volume envelope number 1, shown in line 5 consisting of 10 steps, raising the volume of each step by 1, every 1 second (100 x 0.01 second).

Change line 5 in each of the following ways and then `run` each time to hear the effect of changing the envelope.

```
5 env 1,100,1,10
5 env 1,100,2,10
5 env 1,100,4,10
5 env 1,50,20,20
5 env 1,50,2,20
5 env 1,50,15,30
```

And finally try this:

```
5 env 1,50,2,10
```

You will notice that half way through the sound, the level remains constant. This is because the number of steps was 50 and the time between each step was 0.1 second. Therefore the length of time during which the amplitude varied was only 5 seconds, but the duration of the sound in the `sound` command in line 10 was 10 seconds (number 1000).

Try experimenting yourself, to see what type of sounds you can create.

TONE ENVELOPE

The tone envelope command is `ent`.

This normally has 4 parameters.

The last 3 parameters may appear in any of up to 5 optional envelope sections available. We are only using one of these here. Further explanation will appear in Chapter 6.

`ent` envelope number, number of steps, tone period of step, step time.

ENVELOPE NUMBER

This is the number given to a particular envelope so that it can be specified in the `sound` command. The range of envelope numbers is 1 to 15.

NUMBER OF STEPS

This is used in conjunction with the step time. For example, you may wish to have 10 steps of 1 second each. The range of step numbers is 0 to 239.

TONE PERIOD OF STEP

The tone period for each step can vary between -128 to +127. Negative steps increase the frequency of the notes (make the notes higher). The shortest tone period is 0. This must be remembered when calculating the tone envelope. The full range of tone periods is shown in Appendix VII. The range of tone period of step numbers is -128 to +127.

STEP TIME

This number specifies the time between steps in 0.01 second (1/100th's of a second). The range of a step time number is 0 to 255. The longest time between steps therefore, is 2.55 seconds.

To experiment with the tone envelope, type in the following program:

```
5 ent 1,100,2,2
10 sound 1,284,200,7,0,1
run
```

Line 10 specifies a sound with a tone period of 284 (interntional a) lasting for 2 seconds with a start volume of 7, without a volume envelope (represented by 0), and with tone envelope number 1.

Line 5 is tone envelope number 1 consisting of 100 steps, increasing the tone period (reducing the frequency) by 2 every 0.02 second (2/100th's of a second).

Now change line 5 in each of the following ways and then run each time to hear the effect of changing the tone envelope:

```
5 ent 1,100,-2,2
5 ent 1,10,4,20
5 ent 1,10,-4,20
```

Now replace the `sound` command and the tone envelope by typing in:

```
5 ent 1,2,17,70
10 sound 1,142,140,15,0,1
15 goto 5
run
```

Press the **[ESC]** key twice to break

Now you can put the volume envelope, tone envelope, and `sound` command together to create various sounds. Start by typing in:

```
new
5 env 1,100,1,3
10 ent 1,100,5,3
20 sound 1,284,300,1,1,1
run
```

Then replace line 10 by typing in:

```
10 ent 1,100,-2,3
```

Now replace all the lines by typing in:

```
5 env 1,100,2,2
10 ent 1,100,-2,2
20 sound 1,284,200,1,1,1
run
```

Try some more variations for yourself.

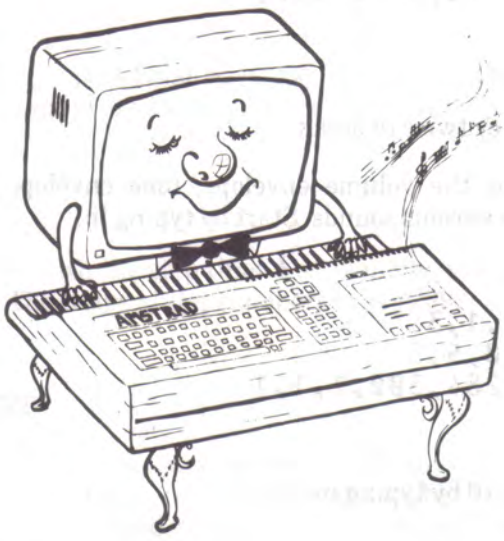
NOISE

Noise can be added to the end of the `sound` command. A range of noise is available in the range 1 to 31. Try this by adding the noise number at the end of the `sound` command still using the `env`elope command.

Replace lines 5 and 20 by typing in:

```
5 env 1,100,3,1
20 sound 1,200,100,1,1,1,5
run
```

Again, try to get some unusual sounds by modifying the volume envelope and the `sound` command, with and without noise.



1. Starters:

If those of you who skipped the beginners' introduction, (which included details of connecting up, switching on and keyboard familiarisation) find the terminology used here confusing then go back to the introduction section for the Foundation Course.

Subjects covered in this chapter:

* Conventions used in this user guide
* Switching on
* Keyboard familiarisation

No matter how familiar you are with programming and computers, please ensure that you follow these few setting up instructions. If you have just opened the box and cannot wait to get started, then this chapter will tell you all you need to know to satisfy your initial curiosity and so get into the application of the BASIC. This section is intended for users who have some familiarity with computers. Beginners should start at the Introductory chapter.

IMPORTANT - you MUST read this:

Terminology.

In order to clarify the references in the text to keys on the keyboard, and text that forms part of the program listing, the following conventions are used from here onwards in this user guide:

[ENTER] : Keys that do not have a corresponding printed character on the screen are shown in this form, surrounded by square brackets **[ ]**.

QWERTYUIOP : Keys that have a corresponding printed character are shown in this form, without square brackets.

`10 FOR N = 1 to 1000` : Text that either appears at the screen, or is to be typed in at the keyboard is shown in this form. Note difference between the zero `0` and the capital letter `O`

It is assumed that you will end each program or direct command line by hitting **[ENTER]**, and this will not be repeated in the listings given throughout the remainder of the guide.

Furthermore, it is assumed that you will type in `run` after each program is entered. BASIC converts all keywords entered in lower case letters into UPPER CASE when a program is `LIST`ed. Examples shown from here on use UPPER CASE, since this is how the program will appear when `LIST`ed. If you enter using lower case, you will be able to spot typing errors more readily since the mistyped keywords will still be displayed in lower case when `LIST`ed.

1.1.1 OPEN THE BOX!

The Colour Monitor

IMPORTANT

Please refer to the **Setting Up** instructions detailed at the start of this User Guide describing the wiring up of the Mains Plug to the Mains Lead of your equipment.

With the computer correctly connected as shown in *Foundation 1*, switch on the monitor, and then the computer, using the slide switch on the right hand end. After about 30 seconds of warm up time, the monitor will display:

```
Amstrad 64K Microcomputer  (v1)
©1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.0

Ready
```

This is known variously as the 'Reset', 'Early Morning' or 'Wake Up' message, that indicates the computer has been completely reset to its initial state - a condition which occurs on power up, and after a full computer reset via the keyboard, when the correct sequence of three keys have been pressed simultaneously.

(**[CTRL][SHIFT]** and **[ESC]**ape pressed in sequence and held down simultaneously - try it now before you enter anything else.)

Adjust the **BRIGHTNESS** control on the right hand side of the monitor. The colour is preset at the factory, so if you want to change the colour of the display (gold lettering on a blue background), this must done by program instructions and you will have to skip to the graphics primer and BASIC keywords in Chapter 8. If you don't mind not jumping ahead a little, here's a short program that will give you one of the best and most readable combinations of colours for text entry, using the high resolution 80 column display mode.

Type in:

```
10 MODE 2
20 INK 1,0
30 INK 0,13
40 BORDER 13
```

...or you could enter the above lines as single statements in the 'direct mode'.

If the above program means nothing to you, or you cannot seem to type it correctly, you will need to go back to the *BEGINNERS FOUNDATION COURSE* which you will find at the start of this guide.You'll find that the 80 column text display is easily the most useful display mode for developing programs - you might like to save the above short program (when you've read through chapter 2) on the start of a blank cassette, to save typing it in every time you switch on.

BEWARE! The high brightness of the colour monitor means that you must be careful to avoid eyestrain through sitting nearby, or using brightness levels that are greater than required by ambient lighting conditions. The moment you feel the onset of eyestrain, switch off and do something else, but to avoid it in the first place:

1. Always work with enough other light to enable you to read the print of this manual with ease. If you are reading this manual by the glare from the screen; this is definitely not to be recommended!
2. Use the minimum brightness required to see what you are doing with ease.
3. Sit as far away from the screen as you can.

A desk lamp positioned alongside the monitor will help to reduce eyestrain - as long as it is positioned behind the front of the screen to avoid reflections.

1.1.2 The Green Screen Monitor

The GT64 monitor has three controls beneath the display front. These are to provide the user with adjustment of the **BRIGHTNESS**, the **CONTRAST** (the difference between the brightest and dullest parts of the display), and the Vertical **HOLD** adjustment, that enables the user the lock the display, and prevent unwanted vertical 'rolling' of the display.

The Vertical **HOLD** will not require frequent adjustment - and once initially set, may be forgotton. The **BRIGHTNESS** and **CONTRAST** may be adjusted from time to time to suit the lighting conditions in the room where the CPC464 is in use.

Using a monochrome monitor (a single colour display that varies the intensity of the characters to provide contrast between the different elements of the display), the switch on message will be the same as with the colour monitor (see the previous page) - except that the text will be displayed in bright green on a duller green background.

Although too much use of your computer and GT64 monitor can lead to eyestrain, the generally 'softer' display from the mono monitor will be much easier to work with over an extended period. In particular, you will be able to make the most of the full width '80 column' text display mode (80 letters or numbers on a single line across the screen) since the resolution definition (the ability to display a number of small elements of the screen display close by one another without 'blurring') of a

monochrome screen is inherently superior to any but the most expensive colour displays.

Adjust the **BRIGHTNESS** control to provide an adequately lit display without the 'dots' that make up the lines in the characters of the display becoming excessively blurred.

To setup in the 80 column mode, type the following brief program listing into the CPC464. You get a choice of display:

```
10 REM set display format
20 FOR n=0 TO 26
30 MODE 2
40 INK 1,n
50 INK 0,(26-n)
55 BORDER n
60 LOCATE 15,12: PRINT "Hit any printing key
   to change the display format"
70 a$=INKEY$
80 IF a$="" GOTO 70
90 NEXT
100 GOTO 20
```

The above illustrates a further important point concerning the representation of style in this manual. Some program listing lines will wrap (over run the line end) and it is important to note that when we print the listings, the additional spaces occuring after a line break are not required in the program itself - they are inserted just to tidy up the listing.

This doesn't illustrate all the possible combinations of shades of grey (colours), but if will give an idea of what's available. When you find a combination you like, stop the program using the **[ESC]** key twice (a `*Break*` message will occur).

From here on, the guide will be making many references that are specific to the colour monitor option. Programs that display interesting colour and graphics effects may appear nearly invisible on the monochrome monitor, although great care has been taken to produce a range of colours with a progression of corresponding grey levels (described more fully in chapter 5).

The advantage of the monochrome monitor is the crisper and less fatiguing display when doing program development - but if and when you find that you are bitten by the computing bug, then there's little hope of you avoiding the inevitability of getting the colour option as well!

1.1.3 The MP1 TV Modulator/Power supply

The MP1 is an additional item that you may wish to purchase if you are currently using your CPC464 computer with the GT64 green tube monitor. The MP1 enables you to use the computer with your domestic colour TV and thereby enjoy the full colour facilities of your CPC464 computer.

IMPORTANT

Please refer to the **Setting Up** instructions detailed at the start of this User Guide describing the wiring up of the Mains Plug and Mains Lead of your equipment..

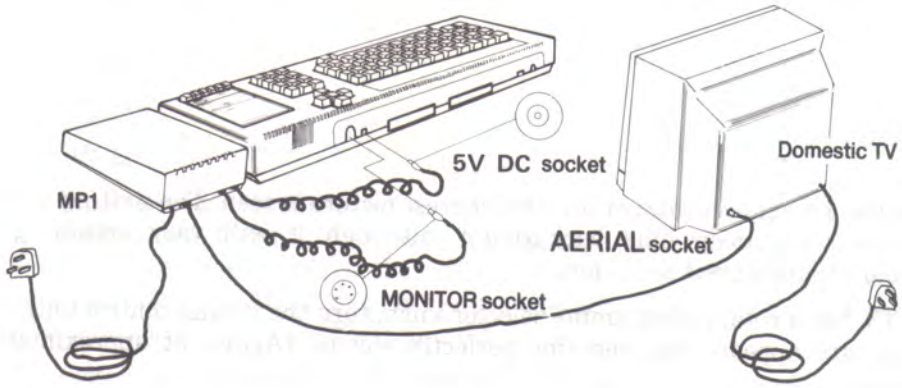


Figure 2: Connections for the MP1, computer and the aerial input of your TV

The modulator/power supply (MP1) should be positioned to the right of the computer on a suitable table close to the TV set and the Mains Supply socket. As shown in figure 2, above.

Now reduce the volume control on your TV set to a minimum - the CPC464 has its own internal loudspeaker, so the hiss from the TV with the volume turned up and the computer switched on is quite normal, Switch on your TV, and then switch on the computer using the slide switch marked **POWER** on the right hand end.

The red **ON** lamp at the top centre of the computer keyboard unit should be illuminated, and you must now tune in your TV set to receive the signal from the computer.

If you have a TV with push-button channel selection, press a channel button to select a spare or unused channel. Adjust the corresponding tuning control in accordance with the TV set manufacturer's instructions (the signal will be approximately at channel 36 if your TV has a marked tuning scale), until you receive a picture similiar to that shown on the next page:

```
Amstrad 64K Microcomputer  (v1)
©1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

BASIC 1.0

Ready
```

Tune in the TV set accurately until the clearest picture is seen. The writing will be gold/yellow on a deep blue background, although it will vary according to individual TV alignment conditions.

If your TV has a rotary programme selector knob, turn the tuning control until the above picture appears and remains perfectly steady. (Again, at approximately channel 36).

As the signal passes through all the various stages of the process of first being *modulated*, and then being *demodulated*, a certain amount of deterioration of the video signal will occur. The results cannot be as good as those obtainable by using a direct video interface monitor, and depending on the quality of your TV, you may find that the 80 column text mode (mode 2) produces results that are not perfectly clear, in which case you should use mode 1 for text entry whenever convenient.

1.2 First steps

You are now 'plugged in': the power supply should be connected, the video lead hooked up to either of the monitors or the modulator option, and you have switched on. Your computer is now waiting for an input:

The 'switch on' screen display, ie. the *'Wake Up'* message, is the only 'built-in' text that you can display without first entering additional instructions through the keyboard.

If you are familiar with the BASIC programming language then there's a good chance that you will already have entered a brief program to 'get aquainted'. AMSTRAD BASIC will be familiar in many respects, and just to get you going, we will show you a brief program that you can enter that will display all the built-in characters that are available from the computer. This is the character set, which is the term used to describe the complete range of numbers, figures and other printable elements of the display that can be called up by typing at the keyboard.

Some of the characters that you will see are not directly accessible by pressing the keys on the keyboard, but only available for display using the PRINT CHR$(‹number›) statement described later on in this guide.

This is because each element stored in the computer is stored in the unit of data known as the 'byte' - and as you will see if you work through Appendix II, a byte has 256 different possible combinations of value. But as the computer has to use at least one whole byte per character stored (whether we want it to or not, it's the smallest denomination that the CPC464 appreciates), we might as well use all 256 possible combinations, rather than simply be satisfied with the 96 or so standard characters that are printed on most typewriters - and throw away the spare 160 possibilities.

The 'standard' range of characters is known as a 'subset'. It is classified throughout the computer world as the 'ASCII' display system, a term derived from:

American
Standard
Code for
Information
Interchange

...it's primarily a system that ensures the data sent from one computer to another is in a recognisable form. It's about the only aspect of computing that is truly universal, so we strongly advise that you become familiar with all aspects of ASCII. Appendix III lists the ASCII display range, together with the additional characters available on the CPC464 and their numeric codes.

Some of the other 'unprintable' characters can be displayed by using a combination of the CONTROL (marked on the keyboard as **[CTRL]**) key and the other keys on the keyboard - but don't worry about that just yet, since until you understand the control key function, you can do more harm than good by testing it at random.

1.2.2

To see exactly what these characters look like on the screen for yourself, type in the following program, and we'll exercise your curiosity and the CPC464. This program will also help establish your confidence in both the simplicity of programming, and the fact that as long as you can get to the 'wake up' message on switching on, then you are not likely to encounter any further 'hardware' problems or misunderstanding, and the CPC464 is simply waiting to be programmed with the right information to process.

(If you make an error when you are typing this program, skip to 1.2.7 to see if you can correct the error without needing to restart entering the program from scratch.)

When typing the program on the next page into the computer, it doesn't matter if you use the lower case letters (a b c), or the **[SHIFT]**'ed uppercase letters (A B C), the computer will sort out the following program either way. You MUST delimit the words using spaces or other delimiters (commas, colons etc. as appropriate) at the positions shown, since AMSTRAD BASIC permits the use of the reserved words (fully listed in Appendix VIII) within variable names.

The 'physical' keyboard is shown in figure 4: it is referred to as the 'physical keyboard' here since many of the keys are available for user re-definition with a series of expansion tokens described in subsequent sections.

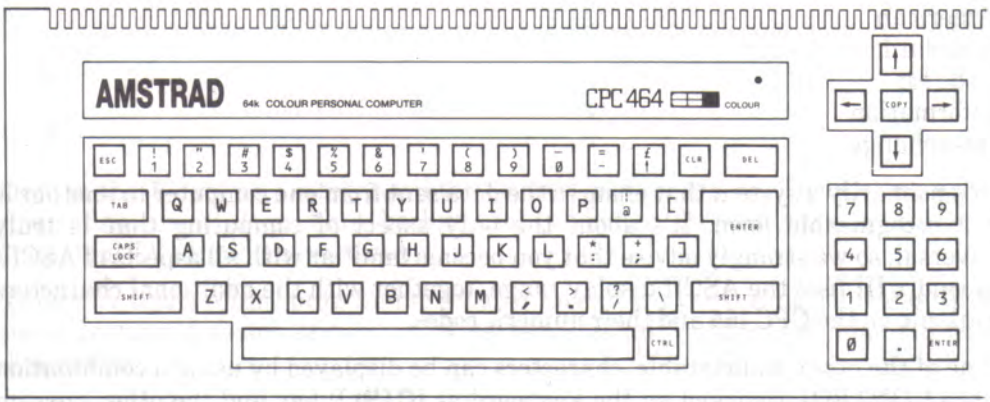


Figure 4: The keyboard of the CPC464

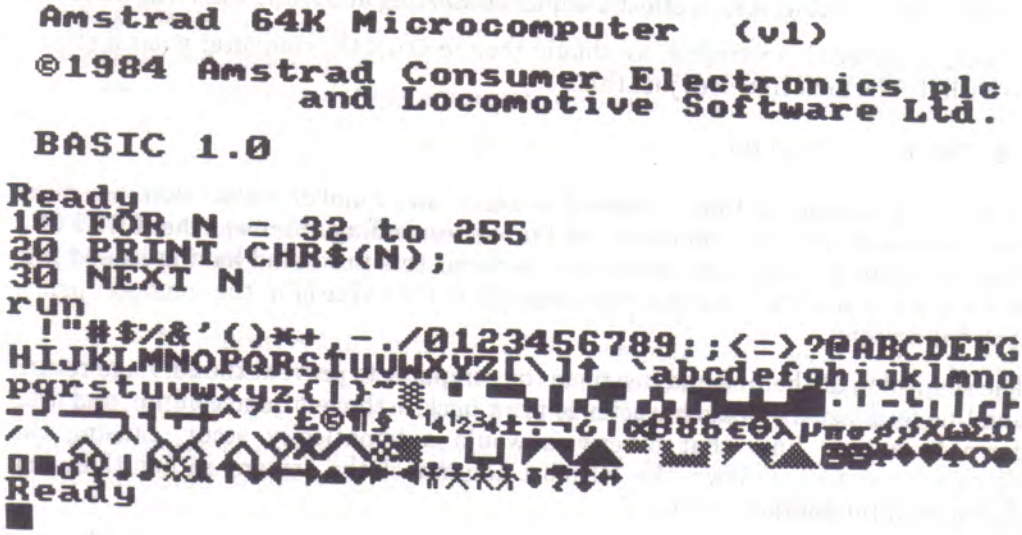
Pressing **[ENTER]** has the effect of **[ENTER]**ing the command or program line you have just typed into the computer, and then asking the computer to process the instruction contained on that line - or if the line began with a number, then to store it away as part of your program.

[ENTER] is also sometimes referred to as the 'carriage return' or simply 'return': which is a reference to the early forms of computer terminals which were based on mechanical typewriter principles. The term has remained, and is enshrined forever in the ASCII character set, where the code for **[ENTER]** is annotated by the letters 'CR'. Type in:

```
10 FOR N = 32 TO 255
20 PRINT CHR$(N);
30 NEXT N
RUN
```

1.2.3

Look and see what has happened on the screen:



The computer has been told to display its full character set using the brief program you have just written. If it hasn't, then you've made an error in typing in the program that you haven't noticed - skip to 1.2.7 and see how to resolve the problem.

Assuming that all is well, and you have the required result, we'll examine what lies behind the displayed characters - it will help you understand exactly how your CPC464 communicates through its range of characters.

The first point is to notice that the computer is not instructed to `PRINT "abcdefghijklmn.....etc"`, it is asked to:

```
PRINT CHR$(N)
```

N just happens to be a convenient shorthand note for a *variable*, the choice of the letter is arbitrary, it just happens to be the mathematicians' favourite in such applications, a variable is an item of computer information that 'varies' according to the instructions given in the program.

A number like 5 is fixed, it occurs between the numbers 4 and 6 - thus it is not a variable, a character `N` is also fixed - it's a letter from the alphabet.

So how did the computer know the difference ? If the letter `N` had been declared to be the alphabetical character, we would have typed `N` in quotation marks:

```
"N"
```

- and the computer would have responded with the message `Syntax error` -because it does not understand the command sequence `FOR "N"`.

Simply by using N in this way, we have told the computer that N is a variable. The definition of the FOR statement in BASIC requires that it should be followed by a variable - so the computer assumes that whatever follows FOR is just that.

We have also told the computer that N = 32 to 255. Thus we have declared the range of the variable, it is in effect a sequence starting at 32, and finishing at 255.

Having declared this variable, we should then instruct the computer what it should do with it - the next line does just this:

```
20 PRINT CHR$(N);
```

It tells the computer that it should convert this number value that has been assigned to N into the character of the corresponding number, the CHR$(N) function is the BASIC instruction that performs this task. And having looked into its memory to see what character corresponds to the value of N, the computer prints it on the screen.

The semicolon at the line end instructs the computer to prevent the carriage return and line feed (set the next character to print back to the left hand column, and drop down to a new line) that otherwise would automatically occur, causing the character set to list down the leftmost column of the screen, rather than one character after another in rows.

The next line tells the computer that when it has performed the task with the first number in the sequence (32), it should return to the line where the FOR is located, and do the same again with the NEXT value it assigns (allocates) to the variable N. This process is known as looping, and this is one of the most vital and fundamental aspects of computer programming and operation.

This FOR loop is one of the most fundamental features of computing, it occurs in all programming lanuages in one shape or form. It saves typing in long repetitious sequences manually, and you will quickly come to use it in your own programming.

When this FOR loop reaches the limit of its declared range (255), the operation ceases and the computer then looks for the next line after line 30 - but there isn't one, so it simply stops and returns to the command prompt by displaying Ready. This tells you that the computer is ready to accept further instructions - or you can RUN again and repeat the execution of the program. The program is safely stored away in the memory and will remain there until you tell the computer otherwise -or if you turn the power off - when all data (programs, variables etc) will be lost unless you save it using the cassette.

This program neatly illustrates a fundamental point about computing - that is everything the computer does is related to numbers. The computer has displayed the alphabet - and a whole range of other characters - using a number as its reference to the character required. When you type the key marked A, you don't ask the computer to type an A on the screen, but you tell the computer to look into the part of its memory that contains the numeric information to display a letter A on the screen. The actual location of this data is defined by the numeric code that is activated by the action of typing at the keyboard.

Each character has a corresponding number, and these are listed in Appendix III of this manual.

Similarly the displayed character has nothing to do with 'writing' the letter on the screen, it's all about numbers.

1.2.4

Please don't worry if you do not understand all the technicalities or jargon used on this page. It is important to fully explain how the computer deals with your instructions and comes up with the required results, but it is also likely that only the technically minded will fully appreciate the explanation. If you find this section heavy going, skip to section 1.2.5.

For example, the code for the letter A is 97. The computer doesn't understand 97 either, and this number has to be translated from the human decimal code into a code that the computer can relate to - it's generally referred to as machine code, and the principles underlying this aspect of the machine are covered in APPENDIX II.

At first, the translation from the decimal number notation we are used to in every day life to the hexadecimal notation of the computer will seem heavy going. Thinking of numbers that are based on the ten unit is so natural that to do otherwise is like trying to eat with your knife and fork in the opposite hands.

A similar degree of mental dexterity must be aquired to understand hex notation (as HEXadecimal is abbreviated), but once you do, many things about computing will fall into place and the elegant structure of the numbering system will become apparent.

Once the computer has translated the striking of the **A** key into the type of number it understands, it looks into that part of the memory indicated, and the result is another series of numbers that define the character. That is to say the character you see displayed on your screen is built up from a block of data, stored in memory as a numeric matrix:

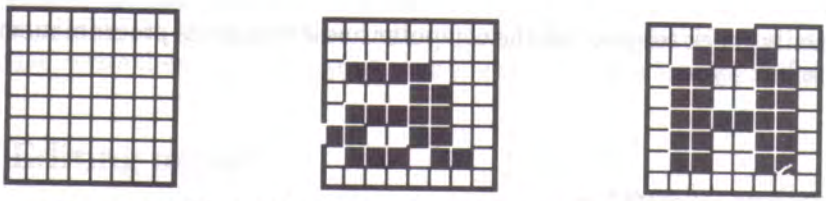


Figure 5: /. blank character grid Lower case a Upper case A

The elements of the matrix are rows and columns of dots. The character is displayed by turning the required sequence of dots on or off - each dot is determined by data stored in the computer's memory. There are 8 rows and 8 columns in each character cell on the CPC464 display and if you don't find a character you want in the set of 256 that are provided, then you can redefine your own characters using the instructions that are given using the keywords `SYMBOL` and `SYMBOL AFTER` in Chapter 8.

User defined characters can be made up using any combination of 0 to 64 dots, so the complete character set that uses all possible combinations of this matrix would comprise many more different elements (characters). Add to this the fact that you can group blocks of elements together to form larger block characters, and possibilities for user-defined characters are only limited by your time and ingenuity.

1.2.5 Back to the program !

The result of the first program you have typed looks rather untidy. There's still the remains of the 'wake up' message at the top of the screen. It looks tidier if we wipe the screen clean before we started the program running. We'll add one line to the program to fix this.

Type the following in on the line where the cursor rests (the cursor is the solid block immediately underneath and to the left of the `Ready` prompt message, and if you didn't know that, what are you doing reading this before the foundation section??):

```
5 CLS
RUN
```

See how the screen clears completely this time before writing the character set starting at the top left.

This also demonstrates one of the most understanding aspects of the BASIC programming language, namely that it does not matter in which order you enter the program line numbers - and you don't actually need to have the program displayed, to add to it once it's been entered into the memory.

The computer always sorts the line numbers into strict numerical sequence before it starts to execute the program. Check by using the `LIST` instruction.

1.2.6 `LIST`ing

You can easily check to see what the computer has stored in its program memory by asking it to list. Type:

```
LIST
```

and the result on the screen is:

```
5 CLS
10 FOR N = 32 TO 255
20 PRINT CHR$(N);
30 NEXT N
Ready
```

This program will stay in CPC464's memory until you either:

*Switch off

*RESET - by pressing **[CTRL][SHIFT][ESCAPE]** in sequence and holding each down after it has been pressed until the reset occurs.

*`LOAD` or `RUN` a program from the cassette unit

*TYPE `NEW` **[ENTER]** which resets all variables and clears the program memory without resetting things like the display mode and the colours.

Now set up one of the function keys to perform the task of typing **[ENTER]**CLS:LIST**[ENTER]** - a feature that speeds up program entry and development enormously. To do this, enter:

```
KEY 138, CHR$(13)+"CLS:LIST"+CHR$(13)
```

Now press the decimal point key on the numeric keypad. Up to 32 keys can be pre-programmed in this way and you can choose any of the keys on the keyboard to redefine in case you want to use the keypad for its original purpose, see the description of the KEY command in chapter 8.

If yours is a long program , then define the key as:

```
KEY 138,CHR$(13)+"CLS:LIST"
```

then the key will allow you to enter the line number range you require - or hitting it twice in succession will perform a full listing.

When you are experimenting with colour, it's possible to get lost in a combination of colours where it's not possible to read the display because the background and the writing have been set to the same values, so if you set:

```
KEY 139,CHR$(13)+"mode2:ink1,0:ink0,9"+chr$(13)
```

...you only have to press the smaller of the two **[ENTER]** keys (the one on the number keypad), and you can get back to base with a combination of colours that are visible. (You will not lose the program in memory.)

User defined key codes are reset with the rest of the machine since they have to be made available to program instructions, so once you have worked out your favourites, write them into a program, and save it onto tape for easy recall.

1.2.7 Editing primer

You will inevitably make mistakes when typing in the program. Welcome to this section all those of you who skipped here from 1.2.2 !

The CPC464 has tried to make correcting these errors as simple as possible, at the same time avoiding the problems of accidentally overwriting characters that you don't mean to change.

The clustered keys that control the motion of the cursor (the solid block that defines where you are going to type the text) provide the means of steering the computer's attention to the part of the display you wish to alter.

When entering an error in a numbered line eg:

```
10 FOR N = 332 TO 255
```

you have several options:

1. You may hit **[ENTER]** and retype the whole line. The incorrect line will be erased from memory and then be overwritten by the next line to be typed that starts with the same line number.

2. You can press the **[←]** key and move the cursor block back to the incorrect entry:

```
10 FOR N = 332 TO 255
```

Note that the character underneath the cursor is displayed in reverse video. In other words, the character which is normally the colour of the cursor has become the same colour as the background (the 'paper') so it will show through the cursor block that is presently placed over the top of it.

Now press the key marked **[CLR]** (short for *CLEAR*) and the character within the cursor will disappear - and the line closes up to fill the gap:

```
10 FOR N = 32 TO 255
```

Press **[ENTER]** again, and this corrected line will be the one that the computer remembers. The cursor doesn't have to be at the end of the line - the computer enters the whole line - *irrespective of the cursor position.*

3. You can also take the cursor back to the character immediately to the right of the one that you wish to delete:

```
10 FOR N = 332 TO 255
```

Now press **[DEL]** (short for *DELETE*), the character to the left of the cursor is deleted, and the cursor pulls the line along behind it without affecting the character within the cursor.

Press **[ENTER]** and the line will be stored as before.

```
10 FOR N = 32 TO 255
```

1.2.8 Afterthoughts

The preceding methods are fine if you spot the error before you reach the line end and type **[ENTER]**. Most errors, however, occur inadvertently and only come to light when you try to run the program - and the computer responds with an error message (APPENDIX VIII).

A number of errors will cause the computer to display the faulty line, with the editing cursor placed over the first column (left end). If this is the case, then you can use the procedures above as if you had spotted the error before entering the program line.

If the error does not prompt you with the faulty line, you will have to LIST the program, find the cause of the problem, and fix it.

1.2.9 Copy Cursor Editing

First list the program using `LIST`. (We'll continue to assume that you're working on the short trial program that fits onto a single screen)

```
5 CLS
10 FOR N = 32 TO 255
20 PRINT CHRS(N);
30 NEXT N
```

The error is in line 20 - there's an `S` instead of a string signifier `$` (`$` tells the computer to consider the characters immediately following to be text, and not numeric data). You can either retype 20 from the beginning, and then re-enter, or you can use the screen editor as follows:

Hold down **[SHIFT]** (either the one on the left or right of the keyboard will do) and then press the up cursor key **[↑]**.

You can either tap it a line at a time, or you can hold it down and wait for the auto repeat to move it for you. The instant you take your finger off the key, the cursor stops, and a little practise will soon familiarise you with the effects. If you overshoot the line, then return with the 'down' key **[↓]** again whilst holding down **[SHIFT]**.

This process causes the *'COPY CURSOR'* to separate from the main cursor (they both look the same), and the copy cursor is taken to the line you wish to modify. Start with the copy cursor placed over the first character in the line.

```
5 CLS
10 FOR N = 32 TO 255
20 PRINT CHRS(N);
30 NEXT N
Ready
■
```

If you were to take the main cursor to the line to modify using the cursor keys without the **[SHIFT]** held down, the computer would not recognise this as a valid action since only the characters entered directly after the main cursor will be recognised as valid instructions.

If you do try and overwrite the line in this way, you can easily get out of the tangle by pressing the **[ESC]** key *BEFORE* you hit any **[ENTER]** key or function key that contains a `chr$(13)`. If you accidentally type (and **[ENTER]**) the command word `NEW` - then your entire program will have been lost forever - so be careful.

Once you have started typing on a line, if you try and move out of it without pressing **[ENTER]**, the computer will bleep if you hit an illegal boundary. once you escape from the problem by hitting **[ENTER]** nothing will be lost in the program -unless you have entered a valid line number as the first typed characters, in which case the line whose number has been written will be 'overwritten' and will need to be retyped.

With the COPY cursor correctly in place, keep tapping the **[COPY]** key until the COPY cursor reaches the item on the line that you need to change. (When you get used to the speed at which the COPY cursor moves, you will be able to hold down the the **[COPY]** key, and move the cursor more rapidly).

```
5 CLS
10 FOR N = 32 TO 255
20 PRINT CHRS(N);
30 NEXT N
Ready
20 PRINT CHR■
```

now release **[COPY]** and type **$** - it will appear under the MAIN cursor, which then steps along one place to the right as usual.

```
20 PRINT CHR$■
```

You must step the COPY cursor past the erroneous S, and to do this hold down the **[SHIFT]** key, and then press the cursor right key **[→]** once. The COPY cursor then rests over the (. Release the **[SHIFT]** and hold down the **[COPY]** key until you reach past the end of the line. Hit **[ENTER]** and the corrected line at the bottom of the screen replaces the incorrect line shown in the listing.

You can combine all these techniques by simply **[COPY]**ing the entire faulty line, and before you hit **[ENTER]** at the end, edit it using the line editing features of the main cursor, using the cursor keys, and the **[CLR]** and **[DEL]** keys without the **[SHIFT]**. Holding down **[CTRL]** and either the cursor left **[←]**, or the cursor right **[→]** key will take the cursor to either the left or right hand end of the line being edited, in one step.

Practise, you'll find it gets easy in a short while.

Finally, you can edit by typing:

```
EDIT 20
```

The computer responds with:

```
20 PRINT CHRS(N);
```

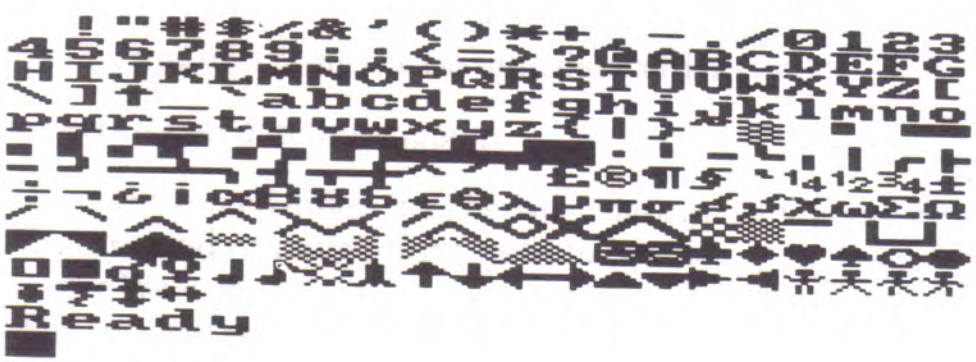
Simply use the main cursor keys together with the **[CLR]** and **[DEL]** keys, as instructed above, and when you are satisfied with the result, press **[ENTER]** as before. If you get in a tangle, press **[ESC]**, and `LIST` once again. The line where the **[ESC]**ape was pressed will not have been overwritten.

Now enter `LIST` once again, and you will see the corrected version of the program displayed. If it's not correct, try again!

Thus far we've only begun to explore the groundwork for CPC464. To take a look at the other two modes, type in:

```
MODE 0
RUN
```

Note that the display first clears, then your program displays the characters 20 wide across the screen:

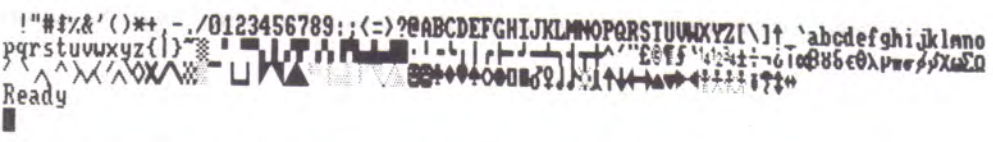


To return to the original display again, type:

```
MODE 1
RUN
```

And we're back where we started. To look at the 80 column display, type:

```
MODE 2
RUN
```



We've made a start - and you should by now have overcome some of your initial curiosity. Advanced users will already be setting about converting their favourite programs to run under this particular dialect of BASIC, less familiar users should proceed through the primer sections to get an overview of the machine-specific features in the BASIC.

2 Cassette Datacorder

Loading and operating the cassette datacorder system

Subjects covered in this chapter:

* Similarities and differences between data and audio cassettes
* Loading and running programs from cassette
  – the Welcome tape
* The optional cassette data speeds
* Saving programs onto cassette
* Read errors

The CPC464 memory is only able to store data as long as the power is connected to the computer - and the unit itself is switched on - in computer terminology it is a 'volatile' storage medium . If you want to store programs or data when the power is switched off - then this must be stored onto the cassette (or other 'non-volatile' storage medium - such as the optional disk system).

2.1 Cassette controls

At the right hand end of the keyboard unit you will find the cassette datacorder -figure 2.1. The mechanics of this cassette mechanism are essentially the same as those of an audio cassette system - except that the electronics that control the signals are specifically optimised for use with computer data storage systems.

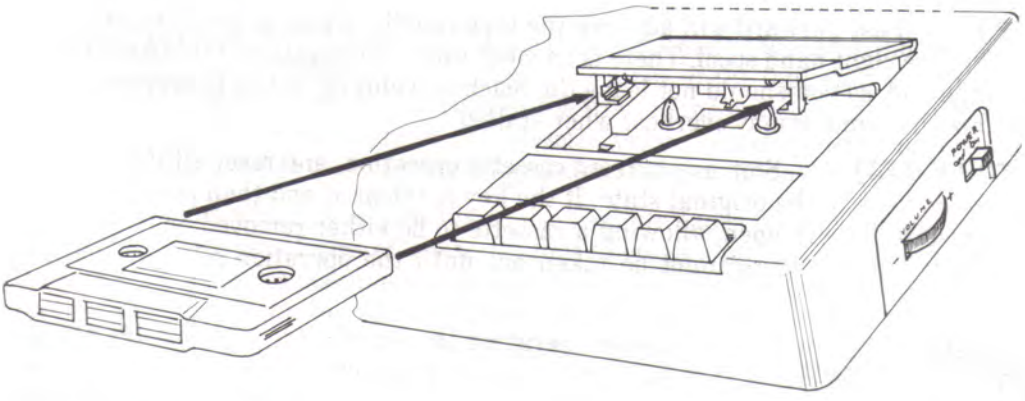


Figure 2.1 The right way to insert a cassette into the Datacorder

Similarly, operation of the auxillary keyboard for the cassette datacorder is the same as for most types of audio cassette recorder:

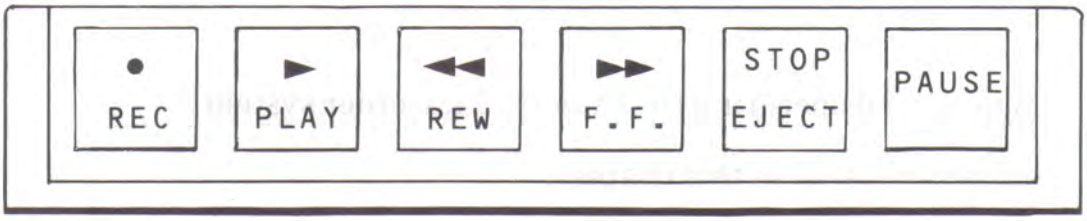


Figure 2.2 The CPC464 Datacorder controls

Note that these control keys require to be pressed considerably harder than the keys on the main 'typewriter-style' keyboard.

[REC] = **[REC]**ord (operates simultaneously with **[PLAY]** to record data when instructed by the program to do so). It can not be activated unless a cassette with the 'record protect' tab (see figure 2.2) is placed in the machine, and the door shut. To activate the function, you must press **[REC]** - and then whilst holding down the **[REC]** key, press the **[PLAY]** key. The computer will then write data onto the cassette when instructed to do so by the program currently in the memory -or by a direct `SAVE` command entered from the keyboard.

[PLAY] = **[PLAY]** the tape to either `LOAD` or `RUN` a program from the cassette system. The computer will then read data from the cassette when instructed to do so by the program currently in the memory - or by a direct command entered from the keyboard. Both the **[REC]** and **[PLAY]** keys are reset by a mechanical trip in the mechanism when the cassette tape reaches its end.

[REW] = **[REW]**ind the tape from the right hand spool to the left hand spool. There is no mechanical cancellation of this function at the tape end, so you should not leave the machine running in rewind mode or the tape drive motor may overheat when stalled.

[F.F.] = Fast Forward will advance the tape rapidly, winding from the left hand spool to the right hand spool. There is no mechanical cancellation of this function at the tape end, so you should not leave the machine running in fast forward mode or the tape drive motor may overheat when stalled.

[STOP/EJECT] = Stop any current cassette operation, and reset all the cassette operation keys to the original state. If the key is released and then pressed again, the cassette lid will open, allowing a cassette to be either removed or inserted as required. The cassette cannot be taken out until the operation of the drive has ceased.

[PAUSE] = A mechanical pause operating in conjunction with the **[PLAY]** or **[REC]** and **[PLAY]** keys. It should not be used to pause during either a data read or data write operation, or an error will occur. All pauses during play and record operations are handled by internal instructions in the CPC464 software, leaving the mechanical pause facility largely unused.

2.2 Write protect (figure 2.3)

In order to prevent accidental erasure of data that you wish to keep on cassette, a mechanical interlock tab is provided on all standard compact cassettes. This tab may be removed by snapping it off using a small pair of pliers etc. whereupon you will be unable to depress the **[REC]** key with the 'write protected' cassette in the drive.

The write protect feature applies individually to EACH SIDE of the cassette - so in order to protect *both sides* of the cassette, *both* of the tabs must be removed. If you subsequently wish to re-enable the facility to record on a cassette that has been protected in this way, then you can cover the depression in the cassette housing created by removal of the tab using a piece of self adhesive tape pulled taught across the opening.

2.3 Loading the cassette

Figure 2.1 illustrates the correct way to fit a cassette into the machine, using the tape supplied with your CPC464.

The cassette should be fully rewound (from the right hand to the left hand spool) - so if it is not, press the **[REW]** key until the motor stalls at the tape end. If the tape is accidentally looped out of the aperture on the front of the cassette, then you should rewind the tape into the body of the cassette before inserting into the cassette machine or the cassette tape and the information stored on it may be lost.

Please note that whilst it is possible to use cassette tapes on audio systems that have suffered various forms of abuse leading to creases and other damage to the tape surface - it is not possible to maltreat a computer data recording in the same way and still expect it to continue to work reliably.

For example, if you damage a tape by trapping a stray loop in the cassette door etc., but find that you are still able to load or run the program from it, you should immediately re-`SAVE` the program on a fresh undamaged tape (while the program is safe in the CPC464 memory), and throw the damaged tape away before you are tempted to try and re-use it.

2.4 Running the Welcome tape

The tape supplied with the CPC464 contains a number of demonstrations of the sound and graphics capabilities of the computer and its built-in software - the AMSTRAD BASIC, and the Machine Operating System. (MOS)

Part of the function of the machine operating system is the cassette operating software itself, which includes a series of brief commands that read and write data and programs on to the tape. The most frequently encountered are the `LOAD` and `RUN` commands.

To make operation of the computer as simple as possible, the CPC464 includes a number of special functions that simplify the keyboard entry until you've had some practice with the keyboard. If you have switched the computer on and are looking at the 'wake up' screen message, with the computer prompting you with the word:

```
Ready
```

Insert the tape as shown (figure 2.1), reset the tape counter to **000** by pressing the small button alongside the tape counter window, and press the control **[CTRL]** key and then whilst still holding this key down, press the small **[ENTER]** key at the bottom right-hand corner of the number-only keypad. You will be prompted by the message:

```
RUN"
Press PLAY then any key:
```

For your information, this is the an example of the expansion token, first encountered in the `KEY` command briefly introduced in Chapter 1, where the computer is instructed to perform a complete command sequence using a shorthand form of keyboard entry to speed up simple and frequently required instruction sequences.

You could have typed in the `RUN"` instruction - and then pressed the **[ENTER]** key, but it is easier to use just 2 keystrokes to achieve the same results.

For most purposes, the two **[ENTER]** keys perform the same task - but there are circumstances where the smaller of the two keys can be altered (redefined) to some other operation using instructions from programs. The subject of 'User Defined Keys' will be covered in more detail later on.

The **[PLAY]** key you are being prompted to press is the key on the cassette drive mechanism. Press it now until it 'latches' and stays down. The part of the prompt message that asks you 'press any key' is a commonly used phrase in computing that can mislead the unwary. It is used to simplify sequences that instruct a program to get on with the next operation by avoiding the need for further specific instruction keywords.

It would be more correct to say '*press any key - other than the* **[SHIFT][CAPS LOCK][CTRL][ESC]** *keys (or any of the keys on the cassette sub-panel)*' - but all computers inevitably have to make certain assumptions in the interests of simplicity. Wherever the term 'press any key' occurs in this user guide (or in

programs supplied by AMSOFT and other vendors), assume the exceptions outlined above still apply.

The key you press will not result in a character appearing on the screen, it will simply cause the tape motor to start playing the tape. If the tape does not start - press another key (it's a good habit to use the large **[ENTER]** key) and check that the **[PAUSE]** key has not been accidentally pressed down.

If you type more than one key, the computer will discard any other keystrokes once it has started to load the program.

Note that you have not specified any particular program name. As long as nothing else occurs on the same line immediately after the

```
RUN"
```

instruction, the computer will search for the first program it can find on the tape, and start to load it. When the computer has found the first correctly recorded program on the tape, the following message will appear:

```
Loading WELCOME 1 block 1
```

This advises you that the computer has located the first of a series of blocks of the program entitled *'Welcome 1'*. Each program is saved on the tape in the form of blocks of data (up to 2kbytes long) that are read into the computer - each data block is separately identified on the tape, and the message on the screen advises you of the current block being read. After each block of data, the tape pauses momentarily, and then starts again - shortly after restarting, the message 'updates' to include the block number presently being read.

If at any time the computer detects bad data, it will display an error message (listed in Appendix VIII) that advises the nature of the problem. It is not generally possible to do anything other than attempt to run the program again until it loads without an error.

Assuming that your tape is loading, read the instructions on the screen, and your *Welcome* tape will do the rest.....

2.5 Supersafe and Speedload

The CPC464 offers two speeds to the user: a *Supersafe* speed of 1000 baud (bits of data per second), and a *Speedload* rate of 2000 baud. The Speedload rate thus writes and reads data at twice the Supersafe rate, although it sacrifices the safety margin sometimes required to account for inconsistencies in low cost tapes and the errors that can arise from the different alignment of tape heads in different recorders.

For programs saved and loaded on the same machine, the Speedload rate should prove to be acceptably reliable using reasonable quality tapes. Speedload will also be capable of loading most commercial software tapes directly without errors - although AMSTRAD advise that all such software be recorded using the option of both speeds.

The computer automatically sets itself to read the rate at which the tape has been recorded. When SAVEing a program, you need to instruct the computer if you wish to use the Speedload rate, or it will default to the Supersafe speed.

To select the higher rate to save your programs and data, check you are in direct mode (the Ready prompt displayed), and type:

```
SPEED WRITE 1
```

to return to the default speed, you can either reset the computer (in which case all data will be lost, or you can type (at the Ready prompt):

```
SPEED WRITE Ø
```

2.6 Saving programs and data on the cassette

BASIC has several commands that are concerned with the way in which data is written onto the cassette. They are summarised briefly here, together with examples.

2.6.1 SAVE "‹file name›"

The most straightforward method of saving data onto the cassette is to use the command SAVE when the CPC464 is displaying the Ready prompt after executing or listing a program. Using the example of a brief program that lists the displayable characters discussed in Chapter 1 as the object of our SAVE command.

The ‹file name› can be any combination of 16 keyboard characters (including spaces). If you try to enter a longer name, the 17th and subsequent characters will be discarded. With the program in the computer's memory, type:

```
SAVE "CHARACTERS"
```

The computer responds with the prompt message:

```
Press REC and PLAY then any key:
```

Remember the qualification of the term 'any key', the tape will start and the computer will save the program under the filename CHARACTERS.

IMPORTANT

Note that the computer cannot detect whether or not you have actually pressed the correct cassette keys - so if you only press **[PLAY]**, the tape will start and the program will appear to be saved when it is not.

BEWARE: If you accidentally press **[REC]** and **[PLAY]** when you want to read/load in a program from tape, then you will erase whatever programs exist on the cassette. If uninterrupted by pressing the **[ESC]** key, the tape will wind through to the end and be completely erased since the tape will not have been halted by the computer finding the program it was searching for. If you are anxious to avoid losing data in this way, get in to the habit of using the snap-off write protect tabs on the cassette casing before you remind yourself the hard way.

There are four ways in which files may be SAVEd by the CPC464. You have just seen the most general method, but there are three alternative formats for more specialised purposes.

2.6.2 SAVE "‹filename›",A

The procedure is the same as above, except that the suffix ,A instructs the computer to save the program or data in the form of an ASCII text file, rather than the shorthand notation the computer uses by default.

This method of saving data applies to files created by wordprocessors and other applications programs, and its use will be further discussed as applications are encountered.

2.6.3 SAVE "‹filename›",P

The ,P tells the computer to protect (scramble) the data on the cassette so that the program cannot easily be read by anyone RUNning it from cassette and then stopping its execution using the **[ESC]** key function. Programs saved in this way can only be recalled using the RUN or CHAIN commands. If you anticipate wanting to edit or alter the program, you should also keep a copy for yourself in unprotected form.

2.6.4 SAVE "‹filename›",B,‹start address›, ‹length› [,‹entry point›]

This option allows you to perform a binary save where a complete block of the data stored in the computer's RAM is stored onto the cassette exactly as it occurs in the internal memory. It is necessary to instruct the computer where the section of memory you need to save starts, how long it is, and the memory address at which to start execution if the file is to RUN as a program.

This binary save feature allows data from a screen display to be stored directly onto the cassette in the form of a screen dump. One of it's main uses is to build 'title' sequences for long programs - breaking the monotony of a lengthy loading process.

2.6.5 Unnamed files and CAT

If you SAVE a file without giving it a file name:

```
SAVE ""
```

then BASIC will save it as an Unnamed file. The cassette can save as many files of the same name (including unnamed files) as can be fitted onto a tape one after another, unlike a disk system which requires each entry to have a unique name.

You will quickly lose track of your programs if you don't give them names that will remind you of their content - and you are advised to add some date code to the name so that you can tell which are the most recent updates of your programs and data files.

You can CATalogue the contents of a cassette by entering the command CAT, and following the instructions:

```
Press PLAY then any key:
```

BASIC will now list the contents of the tape, returning all ‹file names› as UPPER CASE, followed by the number of blocks present, and then a single character that tells you what sort of file it is:

$ is a standard BASIC program
% is a protected BASIC program
* is an ASCII text file
& is a binary file

An Ok at the end of the line indicates that file is readable, and would have loaded had the computer been requested to do so. Performing the CAT function will not affect the program currently in the computer's memory..

2.7 Read errors

If you get the message displayed that a Read error has occured while the CPC464 has been trying to load a program or data from the cassette, then the tape will continue to play, and the computer will continue to read the blocks that it finds after the error - except that it will not attempt to LOAD them unless they are identified as being block 1 of the program it first tried to load (unsuccessfully).

This means that after a read error, you can stop the tape using the cassette **[STOP/EJECT]** key, **[REW]**ind the tape to the beginning, and press **[PLAY]** again. The computer will then have another try at loading the program in which it found the read error - and with luck, this time you will be successful.

Read errors arise from a number of causes - the most common of which is accidental damage to the cassette tape through creasing, stretching or other afflictions of the recording surface. They can also arise from the apparently innocent practise of switching off the computer whilst the cassette **[PLAY]** key or the **[REC]** and **[PLAY]** keys are still pressed down.

This is because when the key is pressed down, the tape is held up against the tape head, and a brief pulse of electricity can pass through this record/replay head as the power supply discharges itself. Even though the tape itself has remained static -this 'switch off pulse' (and also the switch on pulse) can effectively scramble the information in that part of the tape, and make it unreadable. Furthermore, the stationary tape is held tightly between the capstan and pinch roller, which, if greatly prolonged, can lead to tape creasing.

Read errors can also occur if the tape was **[PAUSE]**d during the record or play process, or if it has been originally recorded on another CPC464 where the tape heads are incorrectly aligned.

Read errors will also arise from time to time for purely arbitrary reasons. The compact cassette was not originally designed as a medium for data storage, and as such it has several shortcomings that set it apart from the more complex and considerably more costly 'professional' tape storage systems.

Notwithstanding this, the cassette has performed an excellent job of providing a standard 'medium' for low cost computer data storage and retrieval. The limitations of the physical characteristsics of the size of the magnetic particles on the tape surface, coupled with the speed at which the tape passes the head places a finite limitation on the rate at which data will transfer between tape and computer. To attempt to push speeds beyond the upper tolerance of the system will cause unreliable operation - particularly with mass cassette duplication methods used for manufacturing low cost software.

NOTE that cassettes containing programs from other types of computer cannot be read or loaded on the CPC464. They may look the same - they may even make similar 'sounds' if played through an audio cassette playback system - but they will not load and run. If you do find any that appear to load and run - then we would be pleased to hear from you giving details of the computer type and program concerned.

2.8 Cassette considerations

Although the Datacorder will happily accept all types of cassette up to C90, you should only use C12 (6 minutes per side) and C30 at the most. Programs stored at the end of long cassettes are hard to locate unless you are prepared to wait for them to be found (and you can remember the filename you gave them), or you are meticulous about using the tape counter in your indexing system (ie in the file name). If you want to overwrite a program stored on a long cassette, then you must be careful to locate the start point, and take care that you do not overwrite any other program that you may wish to keep.

Overall, use each separate cassette for as few programs as possible. C12 cassettes are relatively cheap - and should you damage the tape for any reason, it's considerably less tempting to want to save a cassette which is 'good in parts'.

Finally, please remember that commercial software is nearly always supplied under strict conditions of copyright. You should not attempt to copy or otherwise duplicate software supplied on cassette other than in accordance with the terms of sale of that software (some programs actually encourage you to make backup copies) - even if it's 'only for a friend'. The laws of copyright are being revised to deal with all forms of unauthorised software duplication, and although there have been few prosecutions to date, this situation will change substantially over the next few years, and may be retrospectively enforcable.

3 BASIC primer

A brief introduction to programs written using the AMSTRAD BASIC language

Subjects covered in this chapter

* The rules of syntax and descriptions of syntax
* The PRINT commands, streams and display formatting
* ZONEs

3.1 BASIC basics

The fundamental relationship of the CPC464's built-in BASIC to the internal operation of the computer is introduced in Appendix II. If you have not previously programmed a computer, then we'll try and help you along gently - but it may be necessary to make some assumptions that might cause confusion to the novice. If so, we suggest that you browse through the many books available that are designed to introduce the fundamentals of programming to the newcomer.

You should be able to work through this chapter following the simple exercises herein without requiring a complete appreciation of what's happening - although the more of the rules you learn, the easier it will all become.

BASIC is the language that comes supplied 'built-in' with your CPC464. It's right there when you switch on, and makes it's presence felt with the prompt word:

```
Ready
```

BASIC is the simplest language to learn. It is organised with clearly defined words and 'grammar', and operates completely logically - as long as you understand the rules.

AMSTRAD BASIC is capable of executing the commands given in Chapter 8. Each command is identified by one or more leading keywords, and may have a number of *parameters* - some of which are optional. In general each parameter may be an expression, involving constants, variables and functions. Combinations of numbers and letters are known as strings, and various forms of numeric data types are supported including decimal, hexadecimal and binary types.

Files on the cassette are handled sequentially ie one after another - as opposed to randomly, where a file can be selected directly from the midst of many others without first passing through the unwanted files.

3.2 The structure of a BASIC program

Program instructions are presented to BASIC on lines. A line may comprise several commands, separated by colons, limited only by the maximum line length of 255 characters. A 'character' is a number, letter, space. In *Direct Mode* lines are typed in from the keyboard and must not commence with a number. In *Program Mode* lines are read from the current program in memory, and are executed in strict sequence according to the number that appears as the first entry on the line.

AMSTRAD BASIC allows the user to add and remove lines from the program whilst in direct mode, and to amend existing lines. Before RUNning or LISTing a program, BASIC will internally re-organise the order of the lines in ascending numerical sequence, regardless of the order in which they were entered.

3.3 Line Input.

BASIC accepts lines of up to 255 characters, terminated by **[ENTER]**. During line input, it is possible to edt the current line, and to use the COPY cursor facilities to insert characters from elsewhere on the visible screen display - see chapter one section 1.2.7.

All keywords must be delimited by a separator - which is either a space, mathematical *operator* (+ − etc.) or other recognised character. This is because it is permissable to use variables where a keyword (or *reserved* word) forms part of the variable - obviously a reserved word cannot be a variable unless 'surrounded' to prevent the computer from recognising the start and end of the instruction.

Keywords may be entered either as lower case letters (un**[SHIFT]**ed), or CAPITAL letters (**[SHIFT]**ed).

The command PRINT may be abbreviated to a question mark ? and when used in this form, the delimiter is not required. Mathematical operators (+−*/MOD\) also perform the function of delimiting the keyword, so the following is valid, although not encouraged, since it may encourage bad habits in program entry where spaces are necessary:

```
for n= 1 to 50:?n:next
```

Similarly, a single quote mark ' (**[SHIFT]** 7) can be used as a substitute for :REM in remark statements.

Additional spaces will be ignored, and may used to 'format' the program listing to indicate areas of loops etc.

3.4 Terminology

In order to describe BASIC commands and keywords, a formal but simple terminology must be used. Each command is described as it should appear when entered at the keyboard, with any variable or optional parts shown by various types of brackets or *parentheses* which refer to items detailed in the subsequent description.

These items are represented by the various terms, and enclosed in angle brackets ‹›. For example, in various places an expression which yields a numeric result is required, this is represented by :

‹numeric expression›

Anything not enclosed by angle brackets is required as given. For example the STOP command takes the form :

STOP

Where there is an optional part in a definition, the optional part is enclosed in square brackets. For example, if a numeric expression were to be optional then it would appear:

[‹numeric expression›]

If an optional part may be repeated (so may appear any number of times, including none at all) an asterisk is included after the closing square bracket. For example, a string of digits, requiring at least one digit, would appear :

‹digit›[‹digit›]*

Such an expression might be:

34
or 344
or 345678 etc

In many places a list of items separated by commas is used. A short form is used, which is best illustrated by example, thus :

‹list of: ‹expression› means: ‹expression›[,‹expression›]* or:
‹list of: [#]‹number› means: [#]‹number›[,[#]‹number›]*

an example of which is:

3,4
or 3,4,4
or 3,4,5,6,7,8 etc

The *list* may be a single object. If the list contains more than one object, then each additional object must be preceded by a comma delimiter, since it marks the boundary or limit between items that the computer must treat separately.

Numbers may be expressed in several forms:

a. ‹unscaled numbers›

.....are numbers without exponentiation - i.e. those with no exponent part.

b. ‹scaled numbers› are numbers that are 'raised to the power' or 'scaled' using the form:

2E4 (2 times 10 to the power of 4, or 2.10^4)

.....the exponent part may be either positive, or negative.

c. ‹based numbers› are numbers that are declared to be either binary or hexadecimal (see Appendix II):

Decimal form (the default condition)	`100`	
Hexadecimal form	`&64 or &H64`	..... (the 'H' is optional)
Binary form	`&X1100100`	..... (the 'X' is compulsory)

3.5 Practice makes perfect - introducing PRINT

In order to demonstrate the way this terminology works, here are some examples of BASIC at work.

One BASIC keyword that displays most of the features of the terminology is the simple command `PRINT` . A command is a BASIC keyword or statement that will operate with the computer in either direct entry mode or program mode. A function requires the presence of a command to 'invoke' the function eg:

```
PRINT FRE("")
```

To get the CPC464 to give you the answer to a question, you must tell it three things:

1. Where you want the answer to appear - ie the screen, the printer or 'elsewhere'
2. You must give the computer the 'data' to work with
3. You must tell the computer what to do with the data

`PRINT` is used to tell the computer to put the results of a command onto a specific output 'stream' - where a stream is identified by a number from 0 to 9, which is described in the analysis of BASIC as a ‹stream expression› - which is the number that defines the particular stream to be used:

0...7 are text streams to text 'windows' that have been previously set up with the `WINDOW` command.

8 is the parallel printer port, and can only be used if a Centronics compatible printer has been attached correctly.

9 is a cassette output file - which must have been properly opened earlier in the program.

A concise form of the `PRINT` command (not employing the '`PRINT USING`' format template) is thus:

`PRINT` [#‹stream expression›][‹print list›]

....so far. The square brackets mean that you don't have to declare the ‹stream expression›, nor do you actually have to give the `PRINT` command a list of anything to print at all (in which case the computer responds with an empty line -try it). If you don't direct the output to a particular stream the CPC464 assumes you mean stream `#0` , the default screen stream. Try this line (remember to press **[ENTER]** at the end to tell the computer to deal with the instructions you have given it):

```
PRINT "HELLO"
```

The computer responds:

```
HELLO
```

Note that the quotations marks "" have been omitted from the 'output stream'. Double quote marks are used by BASIC strictly to delimit (ie. mark the beginning and end of) constant strings.

Now type:

```
PRINT #Ø,"HELLO"
```

...and the result is the same.

But type:

```
PRINT #4,"HELLO"
```

....and the computer has put the result at the top left of the screen because this is the first entry on screen stream number 4 - which defaults to cover the entire text area unless previously defined by the **WINDOW** command. The starting position for all text on a screen stream is the top left - and stream 4 is as yet unused. The sign-on message used stream 0 (the default), so the text was sent to the stream that appeared after the characters already displayed there.

This feature of AMSTRAD BASIC is particularly powerful, since it allows complex screen displays to be built using simple commands and definitions, that contribute towards ease of producing a tidily formatted screen display.

BASIC will print anything you put inside the double quotes without 'taking any notice' of it. Reserved words may be used in the ‹print list›:

```
PRINT "4*4"
```

and the computer responds with:

```
4*4
```

In order to instruct BASIC to operate on the multiplication, 4 multiplied by 4, the numbers and the *operator* (the multiplication symbol *) must be left accessible to BASIC, and directed to a print stream:

```
PRINT 4*4
```

....and BASIC returns the answer

```
 16
```

Notice that the number is offset one column in from the left margin, since BASIC reserves this space for the *minus symbol* (-) that identifies a negative result.

The **PRINT** command has many other forms, using the full formatting facility of a thorough implementation of the standard series of *templates*.

The ‹print list› in the PRINT command refers to the list of items to be printed. This may be either a number, variable or a string expression - which means a previously defined string variable (eg HELLO$), or anything else contained within double quotes ""

PRINT USING tidies up numbers into a fixed format for printing, so that columns can be aligned, and unwanted remainders or fractional parts discarded.

3.6 The PRINT USING format and ZONES

At switch-on, BASIC sets the screen ZONE width into positions 13 columns wide. When the print instruction includes a comma , the next item is tabbed forward to the next ZONE position. If there are fewer columns available on a line than specified in the ZONE command, then BASIC will start the next item to be printed on a new line. It does not break the item across the edge of a line.

With no USING format specified, BASIC prints positive numbers preceded by a space, negative numbers preceded by a - sign. All numbers are followed by a space. The decimal point is omitted if there is no significant fractional part of the item to be printed.

AMSTRAD BASIC does not support the **[TAB]** key as a column tab, since there is considerable lack of unity on the meaning and function of this feature in various dialects of BASIC. Pressing the **[TAB]** key prints a right arrow character →(the same as **[CTRL]** and the I key together), but otherwise has no purpose in AMSTRAD BASIC.

3.7 PRINT TAB(‹integer expression›)(‹print list›)

The effect of this is best illustrated by example. Enter this, and see the result:

```
5 MODE 2: INK 1,0: INK 0,9
10 FOR N=1 TO 5
20 ZONE 40
30 PRINT TAB(N*4)"HI",N
40 NEXT
```

This program illustrates both the ZONE together with the comma , and the TAB() functions at work. Run it again with line 10 altered to:

```
10 FOR N=-5 to 5
```

The TAB instruction moves the start of the PRINT forward by the number of spaces specified in the ‹integer expression›. (ZONE may be set in the range 1....255 - see the definitions in Chapter 8.)

The PRINT USING form is used to format the results of calculations where *real number* results would produce an untidy array of decimal places. It is a complicated concept that is best appreciated through practical examples, since the technical form:

PRINT [#‹stream expression›,] [‹print list›][‹using clause›][‹separator›]

Could be justly described as *non-user-friendly*, especially since the ‹using clause› further divides down to:

USING ‹string expresion›;[‹using list›]

where the ‹using list› further subdivides to

‹expression›[‹separator›‹expression›]*

Try the following:

```
PRINT 123.456, USING "###.##";4567.896
```

the result..

```
 123.456      %4567.90
```

....illustrates several points. Firstly the item to be printed before the USING clause is not affected. Secondly, the USING clause allocates the number of 'output' positions available for the print item following (which may be a variable of course), and if this item exceeds the allocated space to the left of the decimal point, it still displays, but uses the % to indicate that an overflow has occured. Next, the comma after 123.456 caused the following number to be printed at the start of the next print zone. If it had been a semi-colon, the number would have been printed alongside the 123.456 one space to the right. Numbers are always separated by one space when printed on the same line - for obvious reasons!

Finally note that the expression is *rounded* and does not simply discard the digit(s) beyond the last place used in the format.

Try this:

```
PRINT 123.456, USING "#####.##+";4567.899
```

....and the sign appears at the end of the formatted number. A minus sign will precede a negative number by default.

PRINT USING is a very useful facility in producing any form of tabulated result output. It will always warn you if the format specified is too restrictive (using the %).

4 Variables, operators and data

Handling the information in a BASIC program

Subjects covered in this chapter:

* Getting familiar
* Variable types : real, integer and string
* Operators, logical expressions
* Arrays
* DATA

4.1 Spot the reserved word

Note (if you haven't already) that the commands and other reserved words in AMSTRAD BASIC are delimited, either by spaces or punctuation marks, numeric operators etc. Programs are easier to read and debug, because although you may enter the keywords either in UPPER CASE or lower case - when the program is LISTed, all keywords have been converted to UPPER CASE anyway. If they haven't then there is an error concerning the way they have been typed in, and the program will not run.

AMSTRAD BASIC allows you to imbed keywords into variable names. A variable in AMSTRAD BASIC is a name that you assign to a specific element. It can be as simple as a single letter (variables must always start with a letter - not a number), although its easier to read and understand a long program if you use variable names that reflect what's going on:

```
ANSWER=4*4: print ANSWER
```

Variables in AMSTRAD BASIC may contain up to 40 characters (the first must be a letter), and all 40 are significant. Variables must not contain any spaces - or BASIC will only read the letters (or letters and numbers) up to the first space and then respond with:

```
Syntax error
```

Which is indicating that an illegal sequence of characters has been entered (see Appendix VIII). If you want to include phrases with words separated, then use a full stop . where you would otherwise have left a space. All the usual forms of subscripted variables are possible, bearing in mind the need to DIMension arrays.

4.2 Short cuts

It's tedious typing PRINT every time, so you can use a ? instead, and BASIC will understand this means PRINT (so long as it is not placed inside any phrase within quotation marks ""). Note that the ? option does not require to be de-limited using a space like the word PRINT. If you write the line as a program rather than a direct command:

```
10?4*4
run
```

The answer is still the same, but now LIST this one-line program and as if by magic.....

```
list
10 PRINT 4*4
```

BASIC has also inserted a space before *expanding* the question mark. In PRINT statements that use quotation marks....

```
10?"HELLO"
```

You may also get lazy and drop the final quote marks " on a line, as evidenced by the small **[CTRL][ENTER]** shorthand cassette load and run routine which prints RUN" on the screen. The same works in program lines, but it's not a good habit to get into, since if you subsequently go back to the line and add to it, you will probably forget to close the quotes again.

4.3 Multi-statement lines and Mixed calculations

You can perform a number of operations on a single line of BASIC - in fact, as many as you like up to the maximum line length of 255 characters. As usual, statements should be separated by colons :

```
?2*8/5+5-4*777E9/3
```

returns.....

```
-1.036E+12
```

However, it is essential to understand the order in which the various mathematical operators (+-*/ MOD etc) are recognised by BASIC, or you will make fundamental mistakes. They are:

↑	Exponentiation - raising a number to a given power of 10.
MOD	Modulus - The remainder after Integer division.
–	Unary Minus (the minus sign used to declare a number as negative)
*	Multiply
/	Divide
\	Integer division: the result is shortened to give the whole number part of the result, the decimal part is discarded by BASIC.
+	Addition
–	Subtraction

Anything contained within brackets ( ) is dealt with first of all, and if the contents of the brackets are themselves in the form a mixed calculation, then these will be handled in the order outlined above - including any further brackets within the brackets. You must always end up with as many right hand brackets as left hand brackets in such an expression or a `Syntax error` will result.

4.4 Getting going

We've come some way since we introduced the `PRINT` statement back in 3.5. You should have picked up enough of the ground rules of AMSTRAD BASIC to enable a progression to be made into the more 'computer' like realms of BASIC as opposed to the simple pseudo-calculator features. The BASIC keywords will be introduced as necessary, refer to Chapter 8 for the alphabetical listing and descriptions if their use is not obvious from the context in which they occur.

Many BASIC keywords say what they mean - the `GOTO 50` command means go to the line numbered 50 and continue execution from there. `END` means just that, and BASIC returns to the direct command prompt `Ready` whenever it encounters an `END` - even if it's the first line of the program.

The direct (or immediate) mode allows you to enter a number of program steps by separating the commands with a colon `:` . However, once you execute the line (press the **[ENTER]** key) - the instructions are processed, and the line is discarded. You can always re-run it using the copy cursor feature, assuming it's still on the screen after it was executed.

4.5 Conditional and logical statements

BASIC makes extensive use of the computer's capacity to do simple repetitive tasks at speed - and without getting bored. A number of the programming commands are provided to assist in the generation of this process (looping): commands that initiate, continue and spot when to finish looping, if a predetermined set of conditions should be met.

The last of these elements of loop control concerns the 'relational expression'. In other words, how one piece of data 'relates' to another is determined by a 'relational operator'. You can compare one piece of variable data with another, or you can compare variable data with a predetermined reference. The relational operators are:

<	less than
<=	less than or equal
=	equal
>	greater than
>=	greater than or equal
<>	not equal

Here comes a brief program to demonstrate these operators, based on subject very dear to our hearts. If you are not already looking at the switch on screen message:

```
Amstrad 64k Microcomputer (v1)
c 1984 Amstrad Consumer Electronics plc
         and Locomotive Software Ltd

BASIC 1.0

Ready
```

....then press the **[CTRL]** **[SHIFT]** and **[ESC]** keys, holding them down simultaneously, and the computer will reset to the above message. Now proceed and type in the following (editing instructions are given in 1.2.7):

```
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE":END
40 PRINT "ASK FOR A BIGGER CAR"
```

Run this small program by typing:

```
RUN
```

and the obligatory **[ENTER]**. The computer asks you a question:

```
WHAT IS YOUR SALARY?
```

(Note how the computer automatically adds the question mark when it wants you to respond with an input of data). Answer the question using only numbers - not letters, currency signs or commas, and enter your answer with the **[ENTER]** key.

Add line 5 below by simply typing after the end of the above operation, when the `Ready` prompt reappears:

```
5 CLS
```

`RUN` again to wipe the screen. If your first answer was less than 10000, make this reply more than 10000, and you will see the difference. Now extend the original program by adding line 50 below:

```
5 CLS
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE": END
40 PRINT "ASK FOR A BIGGER CAR"
50 IF SALARY >30000 THEN PRINT "and
 get a good accountant"
run
```

The `END` at line 30 stops the program and returns to the prompt. Since there's no end on line 40, the program proceeds to see if the `SALARY` variable is greater than 30000 - in which case the program responds with yet more sound advice!

Keep going, and add another line at 60:

```
5 CLS
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE": END
40 PRINT "ASK FOR A BIGGER CAR"
50 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
60 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver"
run
```

Note the relational operators (> and <) act as delimiters to mark the boundaries of the number, and that you do not need to put a space before or after them - if you do, the space will be ignored by BASIC. If you answer `26000` when you run this program, then you will see that BASIC passes through line 50 as if it was not there, and gets trapped by the operation in line 60.

At this point, construct a program of your own using the topics covered so far. Note the way in which this program has been 'grown' - most programs evolve in this way, which introduces perhaps the most important concept in programming that utilises the ideal organising ability of BASIC....

4.6 Evolution : the origin of programs

The way in which BASIC allows you to build programs as you go along is its most convenient feature. Purists will argue that this convenience leads to sloppy and 'unstructured' programming techniques with programs being tacked together as ideas occur; realists may consider it's the best way to retain the interest of the student, who has the means of checking progress in easy stages.

Take our example program again, and now add line 70 which loops the program back to the beginning, after pausing long enough for you to read the screen response:

```
5 CLS
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE": END
40 PRINT "ASK FOR A BIGGER CAR"
50 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
60 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver"
70 for n=1 to 900: next n:goto 5
run
```

Note that the new line, and other lines that have been added, have been entered in lower case to remind you that AMSTRAD BASIC understands the difference between a variable name, and keyword. Press **[ESC]** twice to break from this program, then `LIST` the program and see how the computer converts the keywords to UPPER CASE letters, but leaves the variable `n` in lower case.

Line 70 introduces a delay loop as the computer counts 'n' from 1 to 900 before executing the next statement - `GOTO 5`. Thus the program re-cycles itself without ending. The only way out is to **[ESC]**ape using the **[ESC]** key provided. Pressing this once will stop execution of the program. Pressing it again will BREAK out into the direct command mode, without losing the contents of the program memory.

In fact, unless you press **[ESC]** while the computer is still looping through the delay on line 70, the program will break immediately, since the computer isn't executing anything when it is waiting for an input. The line it was waiting on will be listed after the Break. If you escape while the computer is waiting for input, you will get the message:

```
Break in 10
```

If you can catch it in the delay loop, the message after the second depression of the **[ESC]** key is:

```
Break in 70
```

If you escape during the loop on line 70, you will suspend operation, and this can be resumed where it left off simply by pressing 'any key'. If you break out of the program, you can also resume where you broke out by entering:

```
CONT
```

And the program will `CONT`inue from the line in which operation was suspended.

Whatever you do with the **[ESC]** key, you will not lose the program stored in current memory unless you specifically instruct the computer to clear it out using a `NEW` command - or you issue a full reset command by holding down the **[CTRL] [SHIFT]** and **[ESC]** keys simultaneously.

Thus there should be no need to provide a safety net for those who 'inadvertently' reset the machine. The action of clearing out the current program memory is very deliberate, and very permanent. Save anything on to the cassette if you are in any doubt whether you will want to use it again.

4.7 More variables, and strings

The essence of computing is the variable. If the computer is dealing strictly with fixed commodoties, then it is merely an electronic typing aid. Remember that if any part of a mathematical expression is variable, then the result must also be variable.

Variables have three attributes or characteristics: a name, a type and an 'organisation'. Names have been discussed earlier (4.1) - types are optional, so we could define a variable according to the rules of (3.4) as:

The type markers are: ‹name›[‹type marker›]

% for integer numbers where anything to the right of a decimal point is discarded. Integer variables occupy less space in memory, and so programs that do not require decimal manipulations can be made to run faster if the variables are `DEF`ined as integers. The command `DEFINT` indicates that variable names without an implicit type marker are to be taken as integer variables. Following the command DEFINT A therefore, the same variable could be called A% or A, both being integer. Integer values range from -32768....+32767.

! declares a variable to be *real* - which means the integer part, *and* the portion to the right of the decimal point. Variables default to being real upon switch on.Real variables can take any value in the range 2.9E–39 to 1.7E+38.

$ indicates a *string variable*, where the contents may be a mixture of numbers, letters etc. In other words an arbitrary collection of characters that are delimited by being enclosed within double quotes " ". For example:

```
NAME$="BOB SMITH"
```

Using this in our evolutionary example, add line 6, and adjust line 60:

```
5 CLS
6 INPUT "What is your name";NAME$
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE": END
40 PRINT "ASK FOR A BIGGER CAR"
50 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
60 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver ";NAME$
61 DAILY.RATE=SALARY/365:
 PRINT "that's £";DAILY.RATE;" a day"
70 FOR n=1 to 5000: NEXT n:GOTO 5
run
```

Note that a space has been added after `fiver` or the name would be butted hard up against `fiver`. Try it if you don't believe us! The semicolon `;` at the end of `PRINT` or `INPUT` statement supresses the computer's desire to start a new line at the end of each such statement - unless told otherwise.

We can also work in the subject of integers by adding line 61. Just type it in, and the computer will organise the program:

```
5 CLS
6 INPUT "What is your name";NAME$
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE": END
40 PRINT "ASK FOR A BIGGER CAR"
50 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
60 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver ";NAME$
61 DAILY.RATE=SALARY/365:
 PRINT "that's £";DAILY.RATE;" a day"
70 FOR n=1 to 5000: NEXT n:GOTO 5
run
```

Note that the delay in line 70 has been increased to 5000 now that there's more to read on the screen. The result of the daily rate calculation is untidy - you might as well round it to an integer value. Add line 62.....

```
5 CLS
6 INPUT "What is your name";NAME$
10 INPUT "WHAT IS YOUR SALARY";SALARY
20 IF SALARY < 10000 THEN GOTO 30 ELSE 40
30 PRINT "ASK FOR A PAY RISE": END
40 PRINT "ASK FOR A BIGGER CAR"
50 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
60 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver ";NAME$
61 DAILY.RATE=SALARY/365: PRINT
 "that's £";DAILY.RATE;" a day"
62 INTEGER.RATE%=DAILY.RATE: PRINT
 "or £";INTEGER.RATE%;" if you are not
worried about the odd pence"
70 FOR n=1 to 5000: NEXT n:GOTO 5
```

And run again.

Note that you must remember to continue to use the type marker % since it is possible to have a real variable with the same name as the integer variable, the difference being identified by the % symbol. Also see how the computer display 'wraps' lines that are too long to fit on a single line (it happens in all three modes). Use `MODE 2` for writing long programs, since it's much easier to read programs that are not for ever running over the ends of the lines.

To get to `MODE 2`, type

```
MODE 2
```

And to produce black on white display that is easier to read on the CTM640, enter the following three direct commands:

```
INK 1,0
INK 0,13
BORDER 13
```

Now `LIST` again.

4.8 Display formatting

Part of the evolution of your program is the process of tidying it up from time to time. The first thing we can do is renumber all the lines to multiples of 10 by using the `RENUM` command. At the `Ready` prompt, type:

```
RENUM
```

And then `LIST` again:

```
10 CLS
20 INPUT "What is your name";NAME$
30 INPUT "WHAT IS YOUR SALARY";SALARY
40 IF SALARY < 10000 THEN GOTO 50 ELSE 60
50 PRINT "ASK FOR A PAY RISE": END
60 PRINT "ASK FOR A BIGGER CAR"
70 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
80 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver ";NAME$
90 DAILY.RATE=SALARY/365: PRINT
 "that's £";DAILY.RATE;" a day"
100 INTEGER.RATE%=DAILY.RATE: PRINT
 "or £";INTEGER.RATE%;" if you are not
 worried about the odd pence"
110 FOR n=1 to 5000: NEXT n:GOTO 10
```

All the line numbers have been rounded up - including those referred to within the body of the program. It wouldn't be much to you use if BASIC didn't keep track of all the lines numbers and update them all simultaneously.

Now we'll tidy up the display on the screen - and to do this, first disable the loop on line 110, by turning the line from a command into a REMark:

```
10 CLS
20 INPUT "What is your name";NAME$
30 INPUT "WHAT IS YOUR SALARY";SALARY
40 IF SALARY < 10000 THEN GOTO 50 ELSE 60
50 PRINT "ASK FOR A PAY RISE": END
60 PRINT "ASK FOR A BIGGER CAR"
70 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
80 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver ";NAME$
90 DAILY.RATE=SALARY/365: PRINT
 "that's £";DAILY.RATE;" a day"
100 INTEGER.RATE%=DAILY.RATE: PRINT
 "or £";INTEGER.RATE%;" if you are not
worried about the odd pence"
110 REM FOR n=1 to 5000: NEXT n:GOTO 10
```

Inserting REM at the beginning of the line causes the remainder to be bypassed by BASIC, which then ends the program and returns to the Ready prompt - leaving the screen display in view. Now type in:

```
15 mode 1
```

Line 15 will firstly fix the mode of the display so that regardless of what mode was active when you asked the program to run, line 15 will set it to mode 1. The MODE command automatically performs a CLS - so line 10 is now redundant, but we'll leave it there anyway.

Now RUN the program and respond:

```
What is your name? Bob
WHAT IS YOUR SALARY? 40000
ASK FOR BIGGER CAR
and get a good accountant
...and lend me a fiver Bob
that's £ 109.589041  a day
or £110
if you are not worried about the odd pe
nce
Ready
```

It's not very elegant - especially with the break in the word `pence` . Add...

```
25 PRINT: PRINT
85 PRINT
```

and `EDIT 100` to read:

```
100 INTEGER.RATE%=DAILY.RATE: PRINT
 "or £";INTEGER.RATE%;" if you are ":PRINT
 "not worried about the odd pence"
```

`RUN` again, and you will see that the computer has placed `if you are` back on the line above, since it will now fit (in `MODE 1`). Add line120 to drive the `Ready` message further down the screen:

```
120 ?:?:?:?:?
```

And then `run` the program again; or suppress `Ready` altogether bv typing:

```
120 GOTO 120
```

Once you `run` this version, the only way out is to **[ESC]**ape. Remember that `?` is a quick means of typing `PRINT`. Now `LIST` to check the results so far:

```
10 CLS
15 MODE 1
20 INPUT "What is your name";NAME$
25 PRINT:PRINT
30 INPUT "WHAT IS YOUR SALARY";SALARY
40 IF SALARY < 10000 THEN GOTO 50 ELSE 60
50 PRINT "ASK FOR A PAY RISE": END
60 PRINT "ASK FOR A BIGGER CAR"
70 IF SALARY >30000 THEN PRINT
 "and get a good accountant"
80 IF SALARY >25000 THEN PRINT
 ".... and lend me a fiver ";NAME$
85 PRINT
90 DAILY.RATE=SALARY/365: PRINT
 "that's £";DAILY.RATE;" a day"
100 INTEGER.RATE%=DAILY.RATE: PRINT
 "or £";INTEGER.RATE%;" if you are ":PRINT
 "not worried about the odd pence"
110 REM:FOR n=1 to 5000: NEXT n:GOTO 10
120 GOTO 120
```

4.9 LOCATE

So far, most BASIC commands have been written using the universal BASIC syntax that can be understood by most machines that include any sort of BASIC interpreter. `LOCATE` is one of the dialect features of AMSTRAD BASIC (and several others) that allows you to position the *text* cursor anywhere on the screen:

```
LOCATE 10,4
```

...places the text cursor in the 10th column, 4 lines down from the top of the screen. If you attempt to do this as a direct statement, the cursor will `LOCATE` - but the `Ready` prompt will cause a new line to be started so that the cursor ends up on the left margin once again. Add line 16:

```
16 LOCATE 10,4
run
```

and see that the first input request is lowered and inset accordingly. The next line carries on as before - flush left - since line 25 inserts a couple of empty lines. Press **[ESC]** twice, then add line 26:

```
26 LOCATE 10,4
```

`RUN` the program, and the second question 'overwrites' the first. You can now carry on placing the text exactly where you want it to appear on the screen, using the co-ordinates on the screen planners listed in Appendix 6.

If you want all the questions and comments to appear on the same line, then you should precede each `LOCATE` by a `CLS:` to avoid confusion with the remains of the last entry.

Note that the co-ordinates of `LOCATE` may be variables themselves, generated from within the program. We've worn the 'salary' program out by now, so the next example will be based on a different topic. If you want to save what you have constructed this far, then do so now using the cassette commands of Chapter 2. You may even have grown to like the program by now - so you may wish to keep it to refer to, and expand it further with the commands about to be introduced.

4.10 IF.... THEN

Its use is straightforward and quite literal. `IF` ‹logical expression› `THEN GOTO` ‹line number› is one of several forms of the command.

`IF` checks to see if the result of the ‹logical expression› happens to be true - in which case the option is executed. The `IF` operation can be built into loops that repeat when a specified condition is, or is not satisfied. Reset the computer using **[CTRL][SHIFT][ESC]** and type in:

```
1 MODE 1
10 AMSTRAD=0
20 PRINT "AMSTRAD CPC464 colour personal
 computer"
30 AMSTRAD = AMSTRAD +1
40 IF AMSTRAD <10 GOTO 20
```

Run this and you will see that the print statement of line 20 is repeated until the condition in line 40 is fulfilled. Thus 40 loops back to 20. See the significance of the term 'variable' in the way that the value of AMSTRAD changes with every looping of the program.

If you want to see what's happening to the value of AMSTRAD during this program, then we can add in another line at 35.

```
35 LOCATE 1,20: PRINT AMSTRAD:LOCATE 1,AMSTRAD
run
```

If that was too fast, slow it down with a delay loop:

```
36 for n=1 to 500:next
```

Now add a touch of colour (if you have the CTM640 option) by including:

```
34 BORDER AMSTRAD
```

This line switches the colour of the border so that it is set to the value of AMSTRAD by line 30. List the program again:

```
1 MODE 1
10 AMSTRAD=0
20 PRINT "AMSTRAD CPC464 colour personal
 computer"
30 AMSTRAD = AMSTRAD +1
34 BORDER AMSTRAD
35 LOCATE 1,20:PRINT AMSTRAD:LOCATE 1,AMSTRAD
36 FOR n=1 TO 500:NEXT
40 IF AMSTRAD <10 GOTO 20
run
```

And no doubt you all want to see all the colours available on the CPC464, so alter line 40...

```
40 IF AMSTRAD <26 GOTO 20
```

RUN the program, and you will see all the available colours, starting with the darkest, and ending up with bright white. You can make the border colour number message more useful by adding the word Border to line 35:

```
35 LOCATE 1,20:PRINT "Border ";AMSTRAD:
  LOCATE 1,AMSTRAD
run
```

Whilst we're on the subject of borders, when the program has finished and returned to the Ready prompt, enter

```
BORDER 14,6
```

And you will see that the border alternates between colours 14 and 6. You will have to wait until the next chapter to get more explanation of the way graphics and colours work. To round off this sub-section, enter:

```
ink 1,18,16
```

then...

```
speed ink 1,5
```

Now find the aspirin, and reset the computer. Remember to `SAVE` the program on the cassette if you want to impress your friends.

4.11 ARRAYS

Some programs require a lot of variables to store data. This is all very well but it is often difficult to keep track of which variable is being used to store which piece of data. Fortunately, BASIC provides for this situation in the form of data arrays.

What is an array ? An array is basically a group of variables all referenced by the same name.

The best means of explanation is to take an example. Consider a program that simulates a card game, and involves the dealing of randomly selected cards. Obviously the same card cannot be dealt twice and so a record must be kept of each individual card. A simple way of doing this would be to assign a variable to each card and then set the variable according to the card's location - say 1 to signify that it has been dealt and 0 to signify that it is still in the deck.

Evidently, this will involve 52 separate variables (unless you're playing with a fixed deck!) and you have to remember which variable refers to which card - this is where the array comes in useful.

Firstly, we need to give the array a name, say `PACK`. Now, in order to access a particular variable or element in the array we simply give it a number. So if the first thirteen elements are used to define the suite of hearts, then the six is represented by `PACK(6)` the ten by `PACK(10)` and the king by `PACK(13)`. Getting clearer ?

You cannot, unfortunately, simply go on and on referencing more elements ad infinitum without giving the computer fair warning to reserve space in memory. The `DIM` command is used to achieve this.

This command `DIM`ensions the array i.e. sets its size and so for our example of a pack of cards fifty two elements need to be reserved.

The required BASIC is :

```
DIM PACK (52)
```

This tells the computer to reserve space in memory for fifty two variables (fifty three actually since element zero can also be accessed).

We can now write the framework for a subroutine to deal a card:

```
10 DIM PACK(52)
20 FOR X = 1 TO 52
30 LET PACK(X)=0
40 NEXT X
......
1000 CARD=INT(RND*52)+1
1010 IF PACK(CARD) = 1 THEN GOTO 1000
1020 PACK(CARD) = 1
1030 RETURN
```

Note that the DIM statement is the first line in the program . This is because an array can only be dimensioned once - it cannot be redimensioned further on in the program.

Lines 20 to 40 set the elements of the array to zero. The subroutine that starts at line 1000 then chooses a card at random and checks that it has not been already dealt. If it has, then another is chosen until one is found that is still in the deck . The routine then changes the value of the appropriate element in the array to signify that the card has now been dealt and returns from the subroutine.

Array handling is not restricted to single dimensions but can be extended to any number of dimensions desired. This is achieved by simply adding further reference numbers to the variable. For example a 10x10x10 array (or matrix) can be set up by the command:

```
DIM ARRAY(10,10,10)
```

This technique is useful for dividing data into smaller subsets within a large group. In our example we could split the pack into four suits of thirteen cards which can then be accessed separately by using the format:

```
DIM PACK (4,13)
```

Now if we wish to find the four of clubs,which might have been element 43 in our original array, we simply have to examine element (2,4) - assuming that clubs are the second 'row' in our new array.

Arrays do not have to be used to store numerical data but can be used to handle strings as well . An application for this might be to record the names of people booked into seats at a theatre or on an aeroplane flight.

4.12 DATA

This command, in conjunction with the command READ, can be used to input data into a program. The required data is listed in a line with each item separated by a comma and the whole list preceeded by the DATA command. The data can now be accessed sequentially using the READ command.

An example program is:

```
10 READ X,Y,Z
20 PRINT X;"+";Y;"+";Z;"  = ";X+Y+Z
30 DATA 1,3,5
```

The data can be either numerical or text or a mixture of both. Don't worry if all your data doesn't fit onto one line you just start a new line with the DATA command at the beginning. When the computer comes across a READ command it searches through the program for the next sequential piece of data regardless of where it may occur. Make sure you have sufficient data for all your READs or an error will result.

The only way to interrupt this sequential input of data is to use the RESTORE command. This sets the data pointer to the beginning of the program again allowing the same data to be read several times if required. This program illustrates the use of the DATA, READ and RESTORE commands:

```
10 FOR C =1 TO 5
20 READ X$
30 PRINT X$;" ";
40 NEXT C
50 RESTORE
60 GOTO 10
70 DATA HELLO,HOW,ARE,YOU,TODAY
```

Press **[ESC]** to break from this program.

Note that although the data line has been placed at the end of the program in the example, it can be located anywhere that is convenient.

The DATA command need not be used just to contain information which is to be PRINTed when read; numeric values in a DATA statement can, for example, be read into a SOUND command. Type in:

```
10 FOR n=1 TO 30
20 READ s
30 SOUND 1,s,40,5
40 NEXT n
50 DATA 100,90,100,110,120,110,100,0
60 DATA 130,120,110,0,120,110,100,0
70 DATA 100,90,100,110,120,110,100,0
80 DATA 130,0,100,0,120,150
```

If you cannot hear anything, adjust the volume control at the right hand end of the computer.

Concluding this brief introduction to BASIC, here's a program that enables you to play pontoon with the CPC464 (blackjack and 21 are the other well known names for variations). It demonstrates the use of many features of BASIC, and should be readily understood thanks to the use of representative variable names. You can embellish it with graphics, add tension with sound - and generally develop the theme in the way that all the best BASIC programs evolve from a humble core.

The object of the game is get as close to a total of 21 by adding the face values of the cards in your hand, and then for the house to attempt to match this, or get closer still - without exceeding 21 and so going 'bust'. After typing in line 1, use the command `AUTO` to automatically generate the line numbers..

```
1 REM PONTOON
10 REM INITIALISE
20 YC=2:CC=2
30 ACES=0
40 CACES=0
50 S=0
60 T=0
70 DIM SUIT$(4)
80 SUIT$(1)="CLUBS"
90 SUIT$(2)="HEARTS"
100 SUIT$(3)="SPADES"
110 SUIT$(4)="DIAMONDS"
120 CLS
130 DIM PACK (52)
140 FOR X=L TO 52
150 PACK (X)=0
160 NEXT X
170 REM DEAL TWO CARDS TO EACH PLAYER
180 LOCATE 10,3
190 PRINT"YOU";SPC(15)"HOUSE"
200 LOCATE 3,5
210 GOSUB 740
220 S=S+F
230 IF F=11 THEN ACES=ACES+1
240 LOCATE 3,6
250 GOSUB 740
260 S=S+F
270 IF F=11 THEN ACES=ACES+1
280 LOCATE 24,5
290 GOSUB 740
300 T=T+F
310 IF F=11 THEN CACES=CACES+1
320 LOCATE 24,6
330 GOSUB 740
340 T=T+F
350 IF F=11 THEN CACES=CACES+1
360 REM INPUT OPTION-TWIST (T) OR STICK(S)
370 X$=INKEY$:IF X$<>"S" AND X$<>"T" THEN 370
380 IF X$="S" THEN 560
390 LOCATE 3,YC+5
400 YC=YC+1
```

```
410 GOSUB 740
420 S=S+F
430 IF F=11 THEN ACES=ACES+1
440 REM CHECK SCORE AND ACES
450 IF S<22 THEN 370
460 IF ACES = 0 THEN 500
470 ACES = ACES-1
480 S=S-10
490 GOTO 450
500 LOCATE 12,19
510 PRINT "YOU'RE BUST"
520 PRINT:PRINT"ANOTHER GAME (Y/N)"
530 X$=INKEY$:IF X$<>"Y" AND X$<>"N" THEN 530
540 IF X$="Y" THEN RUN
550 END
560 IF T>16 THEN GOTO 700
570 CC=CC+1
580 LOCATE 24,CC+4
590 GOSUB 740
600 T=T+F
610 IF F=11 THEN CACES=CACES-1
620 IF T<21 THEN 560
630 IF CACES = 0 THEN 670
640 CACES = CACES-1
650 T=T-10
660 GOTO 620
670 LOCATE 12,19
680 PRINT "YOU WIN"
690 GOTO 520
700 LOCATE 12,19
710 IF T<S THEN 680
720 PRINT"THE HOUSE WINS"
730 GOTO 520
740 REM DEAL CARD
750 LET CARD=INT ( RND(1)*52+1)
760 IF PACK(CARD)=1 THEN GOTO 750
770 PACK(CARD)=1
780 F=CARD-13*INT(CARD/13)
790 IF F=0 THEN F=13
800 IF F=1 OR F>10 THEN GOTO 850
810 PRINT F;" OF ";
820 IF F>10 THEN F=10
830 PRINT SUIT$(INT((CARD-1)/13)+1)
840 RETURN
850 IF F=11 THEN PRINT "JACK OF  ";
860 IF F=12 THEN PRINT "QUEEN OF ";
870 IF F=13 THEN PRINT "KING OF  ";
880 IF F<>1 THEN GOTO 820
890 F=11
900 PRINT "ACE OF  ";
910 GOTO 830
```

Enter `T` for a 'twist' (your next card to add to those originally dealt to you), or `S` to stick and play the house. We don't claim that this is the last word in card games for your CPC464, but it will provide you with a substantial bone upon which to add the meat of graphics and sound.
(Note that you must enter `T`(twist) or `S`(stick) in upper case capital letters).

4.13 Logical Expressions.

A major difference between a calculator and computer is the computer's ability to handle logical operations in applications like the conditional `IF....THEN` sequence. To do this, the logical operators treat the values to which they are applied as bit patterns (bitwise), and operate on the individual bits. The description and use is entirely, well ....er *logical* - but it is notoriously difficult to describe logic in simple terms without the precision of the concise definitions.

The two halves of the logical expression are known as the *arguments*. A logical expression comprises:

‹argument›[‹logical operator›‹argument›]

where:

‹argument› is: `NOT` ‹argument›
or: ‹numeric expression›
or: ‹relational expression›
or: (‹logical expression›)

Both the arguments for a logical operator are forced to integer representation, and `Error 6` results if an argument will not fit into the integer range.

The logical operators, in order of precedence, and their effect on each bit are :

`AND` Result is 0 unless both argument bits are 1
`OR` Result is 1 unless both argument bits are 0
`XOR` Result is 1 unless both argument bits are the same

`AND` is the most commonly employed logical operator, and *does not* mean 'add'.

```
PRINT 10 AND 10
```

Results in the answer 10.

```
PRINT 10 AND 12
```

Results in 8.

```
PRINT 10 AND 1000
```

Results in 8 again.

This is because the numbers 10 and 1000 have been converted to *binary* representation:

```
      1010
1111101000
```

The AND operation checks each corresponding bit at a time, and where the bit in the top *and* the bottom row is 1, the answer is 1:

```
0000001000
```

...which when converted back to decimal notation is our result of 8. All this means that the logical operator **AND** is used to detect when two conditions are present simultaneously. Here's a self explanatory application:

```
10 CLS:INPUT "The number of the day";day
20 INPUT "The number of the month";month
30 IF day=25 AND month=12 GOTO 50
40 GOTO 10
50 PRINT "Merry Christmas!"
```

OR works on bits as well, where the result is 1 unless both bits from the arguments are 0, in which case the result is 0. Using the same numbers as for the AND example

```
PRINT 1000 OR 10
 1002
```

Bitwise:

```
      1010
1111101000
```

Resulting in the answer:

```
1111101010
```

And in the program example:

```
10 CLS:
20 INPUT "The number of the month";month
30 IF month=12 OR month=1 OR month=2 GOTO 50
40 CLS:GOTO 10
50 PRINT "It must be winter"
```

```
10 CLS
20 INPUT "The number of the month";month
30 IF NOT(month=6 OR month=7 OR month=8) goto 50
40 CLS:GOTO 10
50 PRINT "It can't be summer !"
```

The final major feature to consider is the fact that you can add together a number of logical conditions to distill the facts yet further (up to the maximum line length) in fact:

```
10 CLS:INPUT "The number of the day";day
20 INPUT "The number of the month";month
30 IF NOT(month=12 OR month=1) AND day=29 GOTO 50
40 CLS:GOTO 10
50 PRINT "This is neither December nor
 January,but this might be a leap year"
```

The result of a relational expression is either -1 or 0. The bit representation for -1 is all bits of the integer = 1; for 0 all bits of the integer are 0. The result of a logical operation on two such arguments will yield either -1 for True, or 0 for False.

Check this by adding line 60 to the above program:

```
60 PRINT NOT(month=12 OR month=1)
70 PRINT (month=12 OR month=1)
```

And when the program is run, entering 29 for the day and, say, 5 for the month will produce the answer in line 50, and the actual values returned by the logical expressions in lines 60 and 70 will be displayed beneath..

Finally, `XOR` (*eXclusive OR*) produces a true result as long as both arguments are different. The following summarises all these features in what is known as a *truth table*; which is a convenient way of illustrating what happens in a bitwise logical operation.

Argument A Argument B	1 0	0 1	1 1	0 0
AND result	0	0	1	0
OR result	1	1	1	0
XOR result	1	1	0	0

5 Graphics Primer

Finding your way around the colour and graphics of the CPC464

Subjects covered in this chapter:

* Screen modes and pixels
* The colours
* INK, PAPER and PEN
* Drawing lines
* WINDOWS

5.1 Machine-specific features

The description and applications of AMSTRAD BASIC have so far largely been based on an industry-standard specification. Most of the purely arithmetical operations will run with only slight adjustments from one 'dialect' of BASIC to another however, the BASIC commands that control the graphics (and the text cursor locations) are more specifically dedicated to the way in which the CPC464 hardware controls the screen display and must be carefully understood to get the most from your micro.

As usual BASIC KEYWORDS are shown displayed in the special computer display style typeface, further examples and a concise description of the keyword and its use can be found by reference to the appropriate entry in chapter 8.

5.1.1 Colour Selection

'Black' ie no colour or illumination is considered as a colour for the purposes of all following descriptions that describe the attributes of the colours and the various commands associated with them.

The BORDER can be set to a ANY pair of colours regardless of the screen mode, and is not reset when a MODE command is issued. It may be set to flash, or to a single steady colour.

The number of available INKs that may be displayed simultaneously depends on the screen MODE selected. Each INK can be set to a pair of colours ie flashing; or a single colour ie steady. The number of usable inks at any time depends on the screen mode as previously defined. The text PAPER, text PEN, and graphics PEN can then be set to an available ink.

5.1.2 Transparent option and the relationship of PEN, INK and PAPER.

Except for the conditions where the flashing alternate colours are specified, two INKs are used when writing to the screen; one INK determines the colour of the PEN, while the other determines the colour of the PAPER.
NB The number associated with PAPER command is the INK declared for that number - and **NOT** the colour number as listed in Appendix VI. Similarly the number associated with the PEN command is the INK declared for that PEN number, and **NOT** the colour number as listed in Appendix VI.

The PAPER number defaults to 0 if not specified, while the PEN number defaults to 1 if not specified. To set the INK for PAPER number 0 to green, which is colour number 9, you would type in:

```
INK 0,9
```

Similarly to set the INK for PEN number 1 to black, which is colour number 0, you would type in:

```
INK 1,0
```

Setting the PAPER to the same INK as the PEN ie: INK 0,0 will black out the entire display.

The text 'writing' can be set to be transparent or opaque using one of a series of control characters that provide useful extensions to the major BASIC graphics commands. With the transparent option, you can either ignore the paper colour and overwrite the graphics, or completely overwrite the background. This brief program illustrates the effect available:

[CTRL][SHIFT][ESC]

```
10 MODE 1
20 INK 2,19
30 DRAW 200,200,2
40 LOCATE 1,21
50 PRINT "1 NORMAL"
60 PRINT CHR$(22)+CHR$(1)
70 ORIGIN 0,0
80 DRAW 500,200,2
90 LOCATE 12,18
100 PRINT"2 TRANSPARENT"
110 PRINT CHR$(22)+CHR$(0)
120 LOCATE 22,15
130 PRINT"3 NORMAL AGAIN"
```

The first DRAW command in line 30 was performed before transparent mode was set, but the second was drawn after the CHR$(22)+CHR$(1) command to set transparent mode was issued in line 60. Note how the overlap points have changed colour, and how (in transparent mode) the INK filling the character cell has been completely overwritten.

Swap the location of the transparency-on (line 60) and transparency-off (line 110) commands in the program and see how this affects the displayed results. A full list of these additional commands is given in Appendix VI.

5.2 Screen modes

There are three modes in which the screen (the text and graphics operation) functions:

a) Normal

Mode 1: 40 columns x 25 lines, 4 INK text
320x200 pixels, individually addressable in 4 colours

b) Multicolour mode

Mode 0: 20 columns x 25 lines, 16 INK text
160x200 pixels, individually addressable in 16 colours

c) High Resolution mode

Mode 2: 80 columns x 25 lines, 2 INK text
640x200 pixels, individually addressable in 2 colours

As you can see, the difference is in the number of individual horizontal 'elements' of the display. Not to be confused with the small stripes on the face of the TV tube, which are a separate feature of the TV monitor hardware.

Each of the three different displays is referred to as the screen or display MODE, and only one mode may be displayed at any given time using BASIC. Changing mode causes the screen to clear completely - including all text and graphics windows (the same effect as a CLS and CLG command), but does not affect the contents of the program memory.

Mode changes may be invoked from BASIC programs, or they may be entered using the direct commands.

5.2.1 MODE Ø is the multicolour graphics display.

16 of the 27 colours available can be displayed simultaneously. The display consists of 160 pixels in each horizontal row, and 200 in each vertical column. A plan of this grid appears in Appendix VI.

In MODE Ø, there are 20 characters on each of 25 lines.

5.2.2 MODE 1 is the 'standard' or default mode.

MODE 1 is pre-set when the CPC464 is turned on. 4 of the 27 colours may be displayed simultaneously - although you can switch rapidly through all 27 if you want. The display is 320 pixels wide, by 200 high. A plan of this grid appears in Appendix VI.

In MODE 1, there are 40 characters on each of 25 lines.

5.2.3 Mode 2 is the high resolution mode.

MODE 2 allows two colours to be used simlutaneously, and is used primarily for its ability to produce 80 text characters per line - which makes a program much easier to write, since you can see so much more of the program at a glance.

MODE 2 provides 640 pixels per horizontal row, again with 200 in each vertical column.

5.2.4 Try this....

With the CPC464 fully reset using **[CTRL][SHIFT][ESC]**, type in this program:

```
5 REM GRAPHICS EXAMPLE DEMONSTRATION
10 MODE 1
15 INK 2,0
16 INK 3,6: REM SETS THE COLOUR USED IN LINE 90
17 BORDER 1: REM DARK BLUE
20 CLG: REM CLEAN UP THE DISPLAY
30 b%=RND*5+1 :REM SET UP PSEUDO
 RANDOM INTEGER VARIABLES
40 c%=RND*5+1
50 ORIGIN 320,200 :REM FIX THE GRAPHICS ORIGIN
60 FOR a = 0 TO 1000 STEP PI/30
70 x%=100*COS(a)
80 MOVE x%,x% :REM MOVE THE GRAPHICS CURSOR
90 DRAW 200*COS(a/b%),200*SIN(a/c%),3
 :REM DRAW THE LINE
91 IF INKEY$<>"" THEN 20
100 NEXT :REM BACK TO 60 UNLESS INTERRUPTED AT 91
110 GOTO 20
```

Now RUN the program. Hit 'any key' on the keyboard, to get another pattern. This demonstrates several important features of the CPC464's hardware and software: the CPC464 'writes the screen' very smoothly without judder or 'tearing', and the software includes commands that permit very sophisticated effects to be achieved with the minimum of effort. The REM statements (REMarks) are simply there for your convenience, you don't need to include them for the program to work, it just helps you (and particularly people who did not write the program in the first place) to understand what's going on.

Note that several of the line numbers indicate 'afterthought' entries - and whilst we could tidy the listing by simply issuing a RENUM command, it will help you to follow the way in which programs evolve and develop from their initial structures if we leave the original numbering.

SAVE this program on the cassette - eg:

```
SAVE "GRAPHICS 5.5.84"
```

This graphics demonstration plots a different coloured interference pattern:

```
new
10 a$=INKEY$: REM PRESS ANY KEY TO
 INITIATE A NEW PATTERN SEQUENCE
20 IF a$="" THEN 10
30 CLS
40 m=INT(RND*3):REM SELECT A RANDOM
 NUMBER BETWEEN 0 AND 3
50 IF m>2 THEN 40:REM TRY AGAIN
 IF THE VALUE EXCEEDS 2
60 MODE m
70 i1=RND*26:REM SELECT RANDOM INK VALUES
80 i2=RND*26
90 IF ABS(i1-i2)<5 THEN 70
100 INK 0,i1:INK 1,i2
110 s=RND*5+3
120 ORIGIN 320,-100
130 FOR x= -1000 TO 0 STEP s
140 MOVE 0,0
150 DRAW x,300:DRAW 0,600
160 MOVE 0,0
170 DRAW -x,300: DRAW 0,600
180 a$=INKEY$
190 IF a$<>"" THEN 30:REM INTERRUPT
 THE LOOP BY PRESSING ANY KEY
200 NEXT x
210 GOTO 10
```

This and the preceding program illustrate simple mathematical concepts in a colourful and very visual way. Both are basically doing some sums on randomly generated 'seed' numbers to ensure that each pattern is different in some way, and displaying the results as random lines.

Your CPC464 is excellent electronic graph paper, and one of the most classic geometrical patterns is a sine wave:

```
10 REM DRAW SINE WAVE
20 MODE 2
30 INK 1,21
40 INK 0,0
50 CLS
60 DEG
70 ORIGIN 0,200
80 FOR n=0 TO 720
90 y=SIN(n)
100 PLOT n*640/720,198*y,1
110 NEXT
```

The PLOT statement in line 100 is the part of the program that draws the line. It produces one dot (pixel) on the screen for each calculation it makes in the FOR NEXT loop (lines 80-110) - and the result is displayed on your screen.

The CPC464 has many simple and powerful commands - you can add to the effect of the above progam by simply adding:

```
15 BORDER 6,9
```

RUN again. The border is now alternating between the colour numbers 6 and 9. The flashing rate is set by the 'default' values. To make the program loop continuously until you press **[ESC]**ape (twice to break out of the program, once to suspend operation), add:

```
120 GOTO 50
```

See that the flashing border did not stop when the program did - this is because the border is controlled indepedently of the rest of the program. To stop the flashing and set the border to bright blue, press **[ESC]** twice, then change line 15 to

```
15 BORDER 2
```

RUN the program, and the flashing stops.

To change the colour of the curve and the background, you must change the colour of the INK in lines 30 and 40. When you LIST the program, it should then look like:

```
10 REM DRAW SINE WAVE
15 BORDER 2
20 MODE 2
30 INK 1,2
40 INK 0,20
50 CLS
60 DEG
70 ORIGIN 0,200
80 FOR n=0 TO 720
90 y=SIN(n)
100 PLOT n*640/720,198*y,1
110 NEXT
120 GOTO 50
```

The number 1 at the end of the PLOT statement in line 100 tells the computer to plot the curve in the colour specified by the INK 1 command in line 30. Check the definition of the PLOT statement in the keyword listing of Chapter 8 and you will see exactly how the various parts of this statement work.

If you look closely at the curve being plotted on the screen, you will see that it is not a continuous line, but is broken in many fine segments. The smallest individual segment is an example of a 'pixel' described earlier.

5.2.5 The graphics cursor and drawing lines

You have now tried some of the ways you can translate programs into graphic displays - and several of the program commands and concepts have been given a chance to perform. When drawing lines at the screen, there are some important considerations to watch out for to avoid confusion.

The first point to watch for is the current state of the program memory. The computer remembers the current colour settings even after a `NEW` instruction to clear the program memory. To reset everything to the starting point, you should use the simultaneous **[CTRL][SHIFT][ESCAPE]** sequence to get back to the switch-on conditon. (`SAVE` anything you need to before doing this!)

You can prove this by simply typing:

```
NEW: CLS
```

....after you have broken out of the previous program. Now type:

```
DRAW 100,100
```

The `DRAW` instruction draws a straight line from the last location of the *GRAPHICS* cursor to the x,y coordinate point specified (100,100). The GRAPHICS CURSOR is an invisible concept that indicates the point at which the next graphics operation will occur.

To find out where it is, you must use the functions `XPOS` and `YPOS`. Type in:

```
PRINT XPOS
```

The answer is

```
 100
```

(which is the same for `YPOS` at this point)

Note that if the text goes down to the bottom of the screen and causes the display to be moved up ('scrolled' up), the graphics display will move up as well but the graphics cursor position remains the same as before. Try it - hold down the cursor down key **[↓]** until the screen clears away at the top, then ask for `XPOS` and `YPOS` again. The graphics cursor value is still there in the memory.

To specify a colour for a line drawn, add the instruction at the end of the `DRAW` command (see the description of the `PLOT` command after the program on the previous page - it works the same way).

You must first have specified the `INK` - and remember that you can only use the number of `INK`s and colours that are permissible in the screen mode you are using. To see this, type in:

```
10 MODE 1
20 INK 0,10
30 ORIGIN 0,0
40 INK 1,26
50 INK 2,0
60 DRAW 320,400,1
70 DRAW 640,0,2
```

Here's an example of a program that uses all the items mentioned so far, and introduces a couple more useful concepts. See how the first line (10) sets up the colour and ink conditions to make sure that whatever was in the memory of the CPC464 at the time is reset to produce the expected results:

```
10 INK 0,0:INK 1,26:INK 2,6:INK 3,18: BORDER 0
20 REM this programs draws patterns
30 mode 1:DEG
40 PRINT "3,4 or 6 sided pattern ? ";
50 LINE INPUT p$
60 IF p$="3" THEN sa=120:GOTO 100
70 IF p$="4" THEN sa=135:GOTO 100
80 IF p$="6" THEN sa=150:GOTO 100
90 GOTO 50
100 PRINT:PRINT "Calculating";
105 IF p$="3" THEN ORIGIN 0,-50,0,640,0,400
 ELSE ORIGIN 0,0,0,640,0,400
110 DIM cx(5),cy(5),r(5),lc(5)
120 DIM np(5)
130 DIM px%(5,81),py%(5,81)
140 st=1
150 cx(1)=320:cy(1)=200:r(1)=80
160 FOR st=1 TO 4
170 r(st+1)=r(st)/2
180 NEXT st
190 FOR st=1 TO 5
200  lc(st)=0:np(st)=0
210  np(st)=np(st)+1
220  px%(st,np(st))=r(st)*SIN(lc(st))
230  py%(st,np(st))=r(st)*COS(lc(st))
240  lc(st)=lc(st)+360/r(st)
245  IF (lc(st) MOD 60)=0 THEN PRINT ".";
250  IF lc(st) < 360 THEN 210
252  px%(st,np(st)+1)=px%(st,1)
254  py%(st,np(st)+1)=py%(st,1)
260 NEXT st
265 CLS:ink 1,2
270 st=1
280 GOSUB 340
290 LOCATE 1,1
300 EVERY 25,1 GOSUB 510
310 EVERY 15,2 GOSUB 550
320 EVERY 5,3 GOSUB 590
330 GOTO 330
340 REM draw circle plus 3,4 or 6 others
 around it
350 cx%=cx(st):cy%=cy(st):lc(st)=0
360 FOR x%=1 TO np(st)
370  MOVE cx%,cy%
```

/more.....

```
380  DRAW cx%+px%(st,x%),cy%+py%(st,x%),
1+(st MOD 3)
390  DRAW cx%+px%(st,x%+1),cy%+py%(st,
x%+1),1+(st MOD 3)
400 NEXT x%
410 IF st=5 THEN RETURN
420 lc(st)=0
430 cx(st+1)=cx(st)+1.5*r(st)*SIN(sa+lc(st))
440 cy(st+1)=cy(st)+1.5*r(st)*COS(sa+lc(st))
450 st=st+1
460 GOSUB 340
470 st=st-1
480 lc(st)=lc(st)+2*sa
490 IF (lc(st) MOD 360)<>0 THEN 430
500 RETURN
510 ik(1)=1+RND*25
520 IF ik(1)=ik(2) OR ik(1)=ik(3) THEN 510
530 INK 1,ik(1)
540 RETURN
550 ik(2)=1+RND*25
560 IF ik(2)=ik(1) OR ik(2)=ik(3) THEN 550
570 INK 2,ik(2)
580 RETURN
590 ik(3)=1+RND*25
600 IF ik(3)=ik(1) OR ik(3)=ik(2) THEN 590
610 INK 3,ik(3)
620 RETURN
```

When you RUN this program, it will ask you a question (line 40) - answer 3 for the speediest results. The program will then display the message Calculating, and display a dot . every few seconds (line 245) to indicate that it is still 'thinking' to itself and confirm that the program is still running.

The subroutines called by lines 300-320 flash the different coloured inks at the rates determined by the EVERY command. If you want to slow down the flashing, EDIT lines 300-320 to read:

```
300 EVERY 250,1 GOSUB 510
310 EVERY 150,2 GOSUB 550
320 EVERY 50,3 GOSUB 590
```

To see what you have done, look up the EVERY command in Chapter 8, it's one of the most useful features in AMSTRAD BASIC. One interesting effect of the EVERY command is the way it stacks up requests to do something if the program is interrupted by pressing the **[ESC]** key - only *ONCE*.

Pause the operation of the program for a few seconds by doing this, then restart by pressing 'any key'. The display will flash frantically as the 'queued' timing instructions rush through to catch up. There's only a finite amount of space in the queue, so after a while, the new EVERY command gets discarded until space is made by allowing those in the queue to work their way through.

5.3 Windows

The user can select up to eight text windows into which characters are written, and also a graphics window into which plotting may be performed. Windows are reset to defaults when the screen mode is set. See the keyword description in Chapter 8.

NB: If the text window is equivalent to the entire screen (default), then rapid rolling is achieved by hardware. If the text window is less than the available screen, then rolling is achieved by software, which is correspondingly slower.

The `WINDOW` command specifies the left/right/top/bottom character cells of the specified screen stream - windows may overlap one another, and provide a rapid means of drawing filled boxes. Before starting to explore them, type:

```
KEY 139, "mode 2:paper 0:ink 1,0:ink 0,9:
          list"+chr$(13)
```

This sets the smaller **[ENTER]** key to clear and restore the text to visible colours should you get lost in some invisible combinations of `PEN` and `PAPER`. The following program draws a series of windows across the screen, and illustrates two major points:

```
5 MODE 0
10 FOR n=0 TO 7
20 WINDOW #n,n+1,n+6,n+1,n+6
30 PAPER #n,n+4
40 CLS #n
50 FOR c=1 TO 200:NEXT
60 NEXT
```

The first point is that each new screen overwrites the one before, and the second is to emphasise that the messages appear in stream `#0` at all times (unless redirected). Before doing anything else, type:

```
LIST
```

And the program will be squeezed through stream 0. Try:

```
LIST #5
```

Then:

```
CLS #6
```

..illustrating the point that the most recently addressed screen stream will overwrite all else - and that the system message `Ready` appears in stream 0, even when the listing was sent to stream 5.

Using the `WINDOW SWAP` command, add in line 55:

```
55 IF n=3 THEN WINDOW SWAP 7,0
```

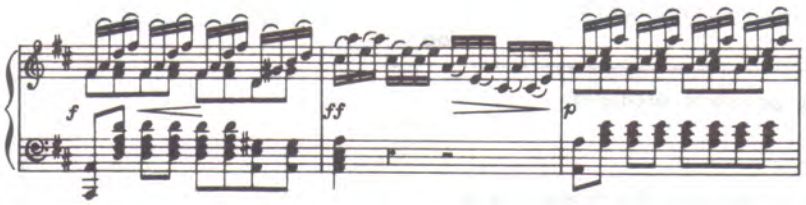
You may imagine that this will direct the `Ready` message at the end of the program execution to stream 7. Run it and see. By developing this simple program, you will get an appreciation of the way `WINDOW`s operate and interact.

6 Sound primer

Sound effects are generated by a loudspeaker within the computer itself. If you are using the MP1 modulator power supply and a domestic television, turn the TV's volume control to a minimum.

The level of sound can be adjusted by use of the **VOLUME** control on the right hand end of the computer. The sound can also be fed to the auxiliary input socket of your stereo system, using the **(I/O)** socket at the left hand end of the computer back panel. This will enable you to listen to the sound generated by the computer in stereo, through your hi-fi loudspeakers or headphones.

The CPC464 sounds as good as it looks. To get the most from the imaginative SOUND processing software, you need to understand the philosophy behind the timing structures.



Subjects covered in this chapter:

* Tone periods
* Sound command
* Enveloping
* Queuing and synchronisation

If all you want to do is to make the computer 'bleep' - then type

```
PRINT CHR$(7)
```

And go no further....but you will be missing some of the most interesting and rewarding features of the CPC464, so this section is a primer that ranges across an overall view of the subject followed by a detailed look at the keywords and how to specify them. It will furnish the programmer with the basics on how to build a scale of notes, invent various types of musical instruments, and structure tunes using them.

6.1 Fundamentals of SOUND

When you hear the sound created by a standard note of a tune there are several characteristics to be considered:

1) Pitch, and variations of pitch during the time of the note.
2) Volume, and variations of volume during the time of the note.
3) Length of the note.

6.2 Pitch

In a musical note the pitch is by far the single most important factor. A musical note can be described as a regular oscillation, all oscillations have frequency, period and amplitude. (see fig 1).

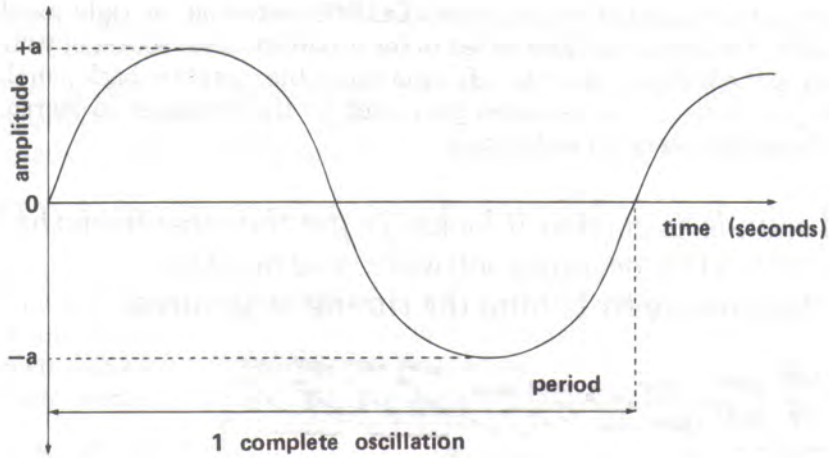


Figure 1: The characteristics of a musical note

The frequency is the number of oscillations per second, and the period is length of time each oscillation lasts. Amplitude is an attribute of volume and is not relevant here in defining the pitch of a note.

Frequency and period are related to each other by the simple formula:

Frequency *(expressed in Hertz)* = 1/Period *(expressed in seconds)*

The relationship between frequency and the ‹tone period› in the SOUND command is:

Frequency *(expressed in Hz)* = 125000 / ‹tone period›

Thus a ‹tone period› of 1000 would result in a 125 Hz tone, and a ‹tone period› of 125 would result in a 1000Hz (1kHz) note.

Either of these could be used to set the pitch, but Amstrad BASIC chooses to use the ‹tone period›. Don't be mislead by the fact that as the period goes up in value, the pitch goes down. The pitch appears in the keyword description of SOUND as ‹tone period›. The range of tone periods available is extensive (0....4095) and must be expressed as an integer. This may lead to some rounding errors when constructing a scale of musical notes, but nothing that would offend any but the most critical ear -see Appendix VII.

When a note is played on a real musical instrument the pitch may vary , sometimes deliberately by vibrato. Using the ENT command (ENvelope Tone) we can design a specific format for the structure of period changes during each note - and this may be invoked in the SOUND command. This feature is known here as the ‹tone envelope›.

6.3 Volume

Volume is simply the measure of how loud the the sound is. To set the initial volume level of each SOUND on the CPC464, there is a straightfoward scale :
increase in value = increase in volume level, set by an integer number
in the range 0....15 (0...7 if no volume envelope is specified).

This value is also established in the SOUND command and is refered to as ‹volume›.

If you have already tried some simple notes, you will have noticed the relatively unexciting nature of the way in which a regular note with no 'processing' is reproduced. This is because in conventional musical instruments, as each note is played there is an increase in volume at the start of each note - called the attack -and a decrease in volume at the end of the note - called the decay.

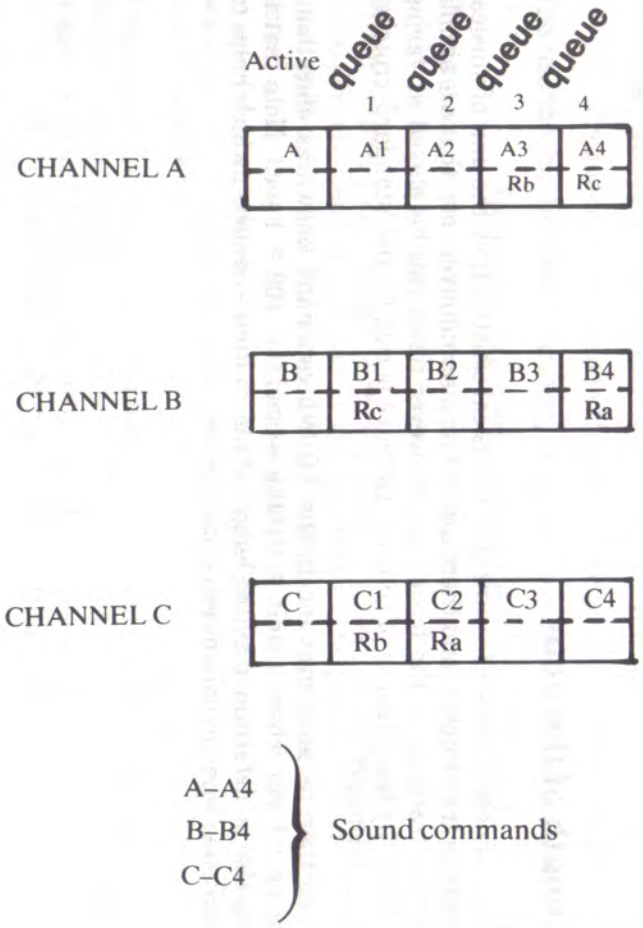
The different forms of attack and decay for each type of musical instrument can be simulated on the computer by adjusting the volume characteristics within a ‹volume envelope›

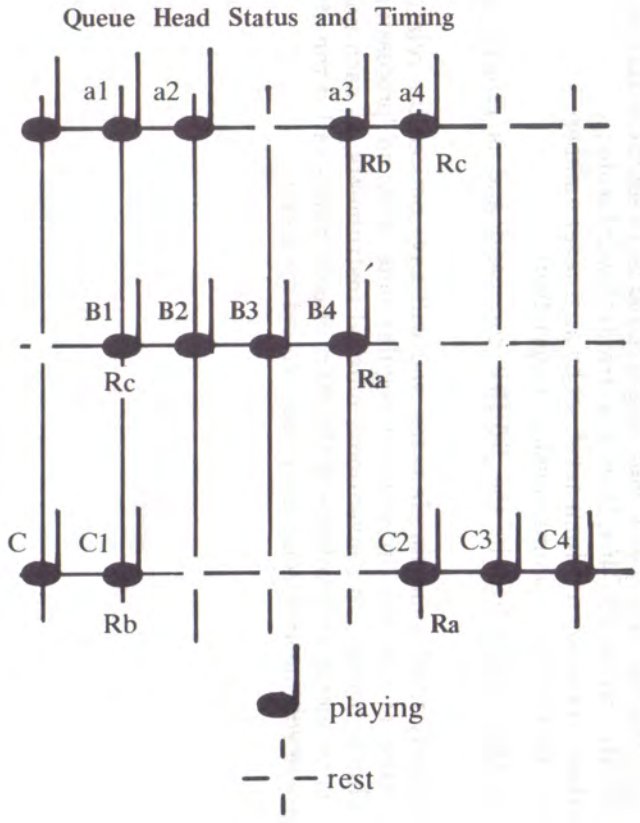
6.4 Length of the note

The length of note is a basic feature of any musical construction, the unit of musical note length is the crotchet, and there are minims, semiquavers etc. that are simple multiples or fractions of the basic unit. However tunes can be played at varing speeds - and the basic length of time or reference period for the 'standard' crotchet has to be determined.

To do this there is an expression in the SOUND command known as ‹duration›, which is an integer where 1 unit = 1/100th second (i.e. 100 = 1 sec.). Note that it may also also be determined by the length of the ‹volume envelope›, which helps to avoid confusion when a ‹volume envelope› is used.

Rendezvous for sound channels





Ra – rendezvous with channel A
Rb – rendezvous with channel B
Rc – rendezvous with channel C

6.5 Other SOUNDs

Random noise ('white noise') is a SOUND and can be generated on the computer, but it is not a musical note. It can be added to the background of tunes to create interesting variations, or simply used on its own for special effects - and explosion is basically white noise with ‹volume envelope› control. This noise varies the frequency randomly around a period that is set in the SOUND command parameter ‹noise period›. An important feature is that although there are three separate channels available with the ability to set three ‹tone periods›, only one ‹noise period› can be set to appear on the required combination of channels.

6.6 Multiple SOUND and channels

Most pieces of music are written with at least two 'clefs', bass and treble. To enable this approach on the CPC464 there have been three SOUND channels provided -A,B and C. These can all play independently, or be timed to coincide when required. Selection of channels is made in the SOUND command parameter ‹channel status›.

6.7 Channel queues

Each SOUND channel has a queue of SOUNDs to play. There is space in this queue for five separate SOUND commands: one active and four waiting. The operating system of the CPC464 can continue with other tasks while playing out the sound queue, only returning when necessary to pick up more SOUND commands.

6.8 Channel status

The keyword SQ is used to determine the state of the channel that you wish to interrogate, and it returns information about free places in the queue, 'rendezvous' and holds. ON SQ GOSUB is an interrupt command used to bring the attention of the computer back to the SOUND generation section of the program.

6.9 Rendezvous and holds

To force channels to synchronize there is a *'RENDEZVOUS'* facility. This is where a marker is set on two or more channels that force the simultaneous action of those SOUND commands. These are very useful to reset the timing of continious notes where a tune contains breaks or 'rests', not occurring simultaneously on both channels.

HOLD (see the SOUND ‹channel status›) is important where an overall synchronization of channels is required (i.e. at the start of tune), maybe with a delay, and it is necessary to have the queues primed ready. Then a HOLD and RELEASE sequence is used.

6.10 Sound generation command sequence

Form of the command: SOUND G,H,I,J,K,L,M

Where....
G: Channel Status
H: Tone Period
I: Duration
J: Volume
K: Volume Envelope
L: Tone Envelope
M: Noise Period

SOUND is a command where the parameters G to M are all integers and only the first two are mandatory. The remaining parameters are optional, but have default values that will be discussed under the parameter descriptions.

6.10.1 Parameter descriptions:

G: Channel Status

Value range 1.....255

default if omitted: none (mandatory entry)

In the CPC464 it is possible to play up to three different SOUNDs at once. This is achieved by having three SOUND channels (or queues), referred to as A, B and C. The input integer range is given in decimal form, but when converted to an 8 bit value (thus yielding a bit significant result), the active bits specify the following commands:

DECIMAL	BIT	COMMAND
1	0 LSB	send SOUND to channel A
2	1	send SOUND to channel B
4	2	send SOUND to channel C
8	3	rendezvous with channel A
16	4	rendezvous with channel B
32	5	rendezvous with channel C
64	6	hold
128	7 MSB	flush

(LSB and MSB signify Least Significant Bit, and Most Significant Bit in the table above.)

integer entry examples:

2 = send the following SOUND to output channel B
5 = send the following SOUND to output channels A and C
98 = 64 + 32 + 2
= send the following SOUND to output channel B, rendezvous with channel C and hold.

It is important to remember that if a rendezvous is to be arranged between two (or more) channels, then it is necessary to flag on the ‹channel status› of each channel at the appropriate time.

The hold on any combination of channels, stops the processing of that command and freezes the queue behind it, until it is freed by a RELEASE command (or 'flushed' by a later SOUND command).

When the flush bit is activated with a channel(s) the SOUND put through with the flush is excecuted immediately leaving the queue empty and the head of the channel inactive. Any SOUNDs currently being played are terminated.

H: Tone Period

Value range 0....4095

default if omitted : none (mandatory entry)

Enters a period that sets the frequency of the SOUND to be played (pitch of the note). The frequency can be evaluated from the formula :

frequency = 125000 / period

By using 0, no frequency is set, this is most useful where only a 'noise' is required.

I: Duration

Any integer value in the permissable range -32768....+32767

default if omitted : 20

For values greater than zero the value entered represents the duration in 1/100ths second. When equal to zero, the duration is governed by the length of the volume envelope specified.

Where the duration is less than zero, the positive value of this number gives the number of times the volume envelope specified should be repeated.

J: Volume

An integer value in the range 0....15 (0....7 where no volume envelope is specified)

default if omitted : 12 (4 where no volume envelope is specified)

This is the starting volume, it can be altered by the volume envelope if one is specified. Zero is no volume.

K: Volume Envelope

An integer value in the range 0....15

default if omitted : 0

The value used specifies a pre-defined envelope. To define an envelope use the ENV command. A permanent definition is volume ‹envelope number› 0, this cannot be changed by the ENV command and is set to 2 seconds constantly at the 'volume level'.

L: Tone Envelope

An integer value in the range 0....15

default if omitted : 0

The value used specifies a pre-defined envelope. To define an envelope use the ENT command. A permanent definition is tone ‹envelope number› 0, this cannot be changed by the ENT command and is set to the 'tone period' constantly.

M: Noise Period

An integer value in the range 0....15

default if omitted : 0

Specifies the noise to add to the SOUND. If default or zero is used, then no noise is added. Remember that only one noise period can be set at one time. This means that all channels specified for noise will receive the same noise.

6.11 The ENV command and volume envelopes

Attack and decay are required to add dimension to the note and give it life. This command (ENvelope Volume) enables the shape of a note to be formed. It is a good idea before attempting to describe the shape to the computer using the numeric instruction sequence to sketch out your requirement on paper. See example below :

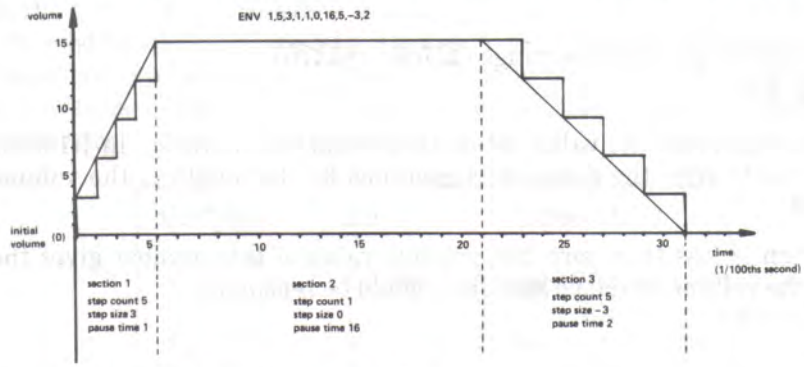


Figure 3: Volume envelope example

The shape must be plotted in numbers so that it can be formatted into a command statement. To do this , subdivide the shape into sections , you may have up to 5. But in each section the shape must be a straight line. Now divide each section into steps, the number of steps you choose will be known as the ‹step count›, these steps will be given a length of time called ‹pause time›. 1 = 1/100th sec.].

The steps will also have an increase or decrease in volume known as the ‹step size›. If no steps are specified in a section then there is just a volume setting carried foward to the next section.

N.B. When attempting to simulate a musical instrument it is often the case that the initial and final volume of each note will be zero, so the initial volume setting in the SOUND command should be zero and all the volume be added by the envelope.

Form of the command:

ENV N,P1,Q1,R1,P2,Q2,R2,P3,Q3,R3,P4,Q4,R4,P5,Q5,R5

N: Envelope Number	value range 1....15
P1....5 : Step Count (sections 1....5)	value range 0....127
Q1....5 : Step Size (sections 1....5)	value range -128....+127
R1....5 : Pause Time (sections 1....5)	value range 0....255

The ‹envelope number› is a compulsory label. A complete section is mandatory, but the number of sections is optional. For example, omitting sections 4 and 5 assumes only 3 sections.

6.12 The ENT command and tone envelopes

These have exactly the same type of construction as volume envelopes, only this feature creates small variations in frequency of a note, in other words, a form of vibrato. After deciding on the shapes of the note's frequency, sketch it as before and divide into sections and steps. see example below :

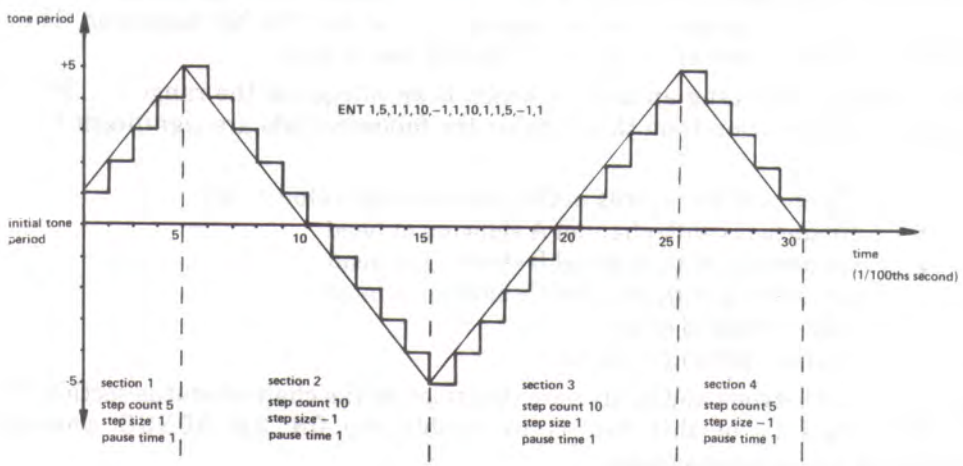


Figure 4: Characteristics of a tone envelope

The major difference between volume envelopes and tone envelopes is that tone envelopes have no effect on the duration of a note previously defined - either in the SOUND command or in a volume envelope. If the tone envelope finishes before the note then the final tone period continues. However, if it overruns the time scale of the note then the remaining steps' sections are abandoned.

To repeat a tone envelope for the duration of the sound, use a negative envelope number. (Note It is not called up as negative in the SOUND command).

Form of the command:

ENT S,T1,V1,W1,T2,V2,W2,T3,V3,W3,T4,V4,W4,T5,V5,W5

S: Envelope Number	value range 1....15 (-ve for repeat)
T1....5 : Step Count (sec. 1....5)	value range 0....239
V1....5 : Step Size (sec. 1....5)	value range -128....+127
W1....5 : Pause Time (sec. 1....5)	value range 0....255 (1/100ths second)

S is mandatory , and completion of each individual section is also mandatory.

When an ‹envelope number› is defined, all previous settings are overruled. Specifying an envelope of either type without any sections will reset the settings of that ‹envelope number› to zero.

6.13 Other associated functions and commands

`SQ(x)`

x is the channel number : 1,2 or 4

These values represent the channels A, B, and C respectively. As shown in the 'bit significant' table of ‹channel status› a parameter of the SOUND command. This is an inquiry into the state of the SOUND channel nominated.

It is a function returning an answer which is an integer in the range 0.....255. To extract the information from this number the following bits are *significant* for the data:

Bit	0....2	number of free spaces in the queue tested (value 0....4)
Bit	3	rendezvous with channel A showing at head
Bit	4	rendezvous with channel B showing at head
Bit	5	rendezvous with channel C showing at head
Bit	6	hold at head of queue
Bit	7	channel currently playing

Bits 3 to 6 (rendezvous and hold) were described in the channel status section. The only other facility of this test is to disable the `ON SQ GOSUB` command described in the next paragraph.

`ON SQ GOSUB`

`ON SQ(y) GOSUB` ‹line number›

y is the channel number : 1,2 or 4

This is an interrupt command that is armed and detects when there is a gap in the `SOUND` queue nominated, then fires into the `SOUND` subroutine. The `SQ` and `SOUND` keywords both disarm the interrupt. The subroutine ends with a `RETURN` in the normal way. All channels have the same priority and when there is a space in the nominated channel the command will 'fire', thus disarming itself at the same time. Hence the subroutine will need to reset the interrupt, if it is required again.

RELEASE

RELEASE z

z indicates the channel number(s), an integer value in the range 1.....7, which is bit significant.

As described in the **SOUND** command parameter ‹channel status› it is possible to flag a hold on a channel with a particular **SOUND** command. The **RELEASE** command is simply to remove these holds. The channel input is bit significant, and when decoded gives the combination of channels to be released. Release of a channel not held has no effect.

Bit 0 : Channel A
Bit 1 : Channel B
Bit 2 : Channel C

7 Printers and joysticks

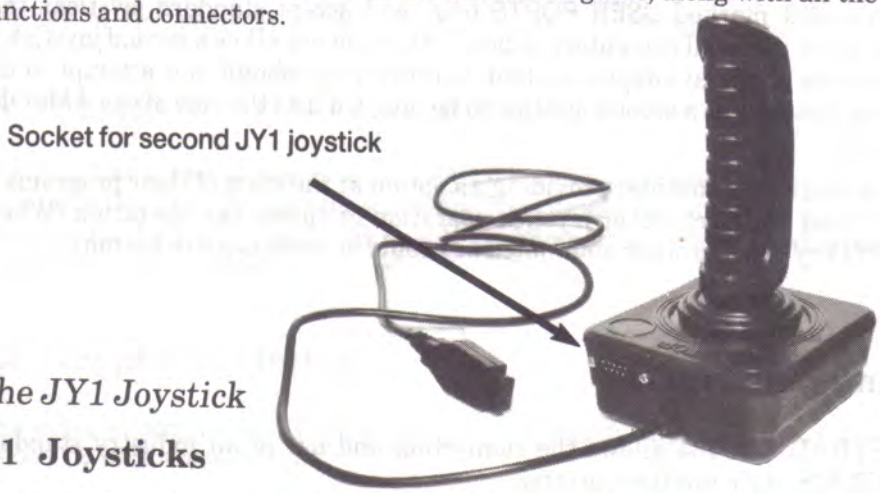
The CPC464 requires no additional interfaces to operate with either one or two joystick controllers, and a Centronics compatible parallel printer.

Subjects covered in this chapter:

* Joysticks
* Parallel printers
* Interfacing

The AMSOFT joystick model *JY1* is an additional item that you may wish to purchase if you are using the CPC464 computer with software games which incorporate the facility for joystick control, and 'firing' within the game. The *JY1* can be plugged into the back of your computer using the 9-way socket marked **USER PORTS (I/O)**. The Amstrad CPC 464 computer can be used with two joysticks. The second joystick should be plugged into the socket on the first joystick.

There are no special points to consider when connecting a joystick to the CPC464: the AMSOFT *JY1* will plug directly into the 9 pin socket provided at the rear of the computer, marked **USER PORTS (I/O)**. A second joystick may be fitted if required into the socket provided in the base of the first *JY1* joystick. The connections of this device are listed in Appendix V at the end of this guide, along with all the other I/O functions and connectors.



The JY1 Joystick

7.1 Joysticks

The built-in software in the AMSTRAD CPC464 supports either one or two joysticks, and these are treated as part of the keyboard and may be interrogated by `INKEY$` and `INKEY` commands as if they were keyboard keys. Note that if there is only one fire button available on your joystick it is likely to be the 'Fire 2' in AMSTRAD CPC464 terminology.

A special function is available to inspect the joysticks directly. This is `JOY(Ø)`, for the first joystick, and `JOY(1)` for the second. The function returns a bit-significant result which indicates the state of the joystick switches at the last keyboard scan. As there are 50 keyboard scans per second the result is virtually the instantaneous state of the joystick switches.

The joysticks return values as follows where `KEY` is the value to use in an `INKEY` function and `MIRROR` is the equivalent keyboard key:

First Joystick	JOY(0)	KEY	Second Joystick	JOY(1)	KEY	MIRROR
Up	Bit 0	72	Up	Bit 0	48	6
Down	Bit 1	73	Down	Bit 1	49	5
Left	Bit 2	74	Left	Bit 2	50	R
Right	Bit 3	75	Right	Bit 3	51	T
Fire 2	Bit 4	76	Fire 2	Bit 4	52	G
Fire 1	Bit 5	77	Fire 1	Bit 5	53	F

Note that when the second joystick is interrogated the CPC464 cannot tell the difference between the joystick and the indicated keyboard keys. In practice it is most unlikely that a conflict of interpretation will exist. Indeed the keyboard could be used as a substitute for the second joystick.

When using the *AMSOFT JY1*, the second joystick is identical to the first, and plugs into the socket on the side of the first joystick. No special wiring is required to allow use of the second joystick.

The 9 pin socket marked **USER PORTS (I/O)** will accept standard joysticks that work with other personal computers, athough these do not allow a second joystick to be fitted unless a special adaptor is used. However, you should *not* attempt to use one of these joysticks as a second joystick to be plugged into the side of the *AMSOFT JY1* joystick.

Software writers may consider providing an option at the start of their programs to enable the user to select either joystick operation or cursor key operation (Where the **[COPY]** key or some other nominated key could be used as a fire button).

7.2 Printer interfacing

The AMSTRAD CPC464 allows the connection and use of an industry standard 'CENTRONICS' style interface printer.

The printer cable is simply constructed as a one-to-one connection between the **PRINTER** port and parallel printer connector. Note that there are two less 'fingers' on the computer's printed circuit board than on the printer connector; permitting use of a standard printed circuit board edge connector.

The actual interface details are illustrated in Appendix V.

The cable should be constructed so that pin 1 on the computer connects to pin 1 on the printer, pin 19 on the computer to pin 19 on the printer etc., with pins 18 and 36 of the printer not connected to the computer.

In particular the lower row of fingers on the computer is numbered 19 onwards (rather than 18 onwards as one might expect given that there are 17 fingers in the upper row) in order that every wire that is used, is connected to exactly the same numbered computer edge connector finger and printer connector pin.

The computer uses the *BUSY* signal (pin 11) to synchronise with the printer and will wait if the printer is *OFF LINE*.

There are no user setup commands required, and the output is directed to the printer by specifying stream 8:

```
LIST #8
```

Will cause the BASIC program in the memory to be listed - as long as it is of a type suitable for listing - ie unprotected.

Within programs, the printer may be addressed by using the simple form:

```
PRINT #8,"This is sent to the printer"
```

Many printers will automatically 'wrap around' line endings if the output reaches the end of a line - check the printer manual. AMSTRAD BASIC will also wrap the output as directed by a `WIDTH` command. The default value of printer width is 132 and may be set to a new value as required: eg `WIDTH 80`.

If set to the special value of 255 then AMSTRAD BASIC will not 'wrap' the output and relies completely on the printer to check for line endings. BASIC maintains a counter of the position of the printer which can be interrogated by the `POS` function.

```
IF POS(#8) > 50 THEN GOTO 100
```

The CPC464 issues a line feed `CHR$(10)` and a carriage return `CHR$(13)` at the end of the line. The printer will usually contain a preset switch for selecting the appropriate form of input, and it will be immediately obvious what the default standard is once you attempt to print.

7.3 Graphics printing

The manual supplied with your printer will specify the control codes, which are generally in the form of the :

```
PRINT CHR$(n)
```

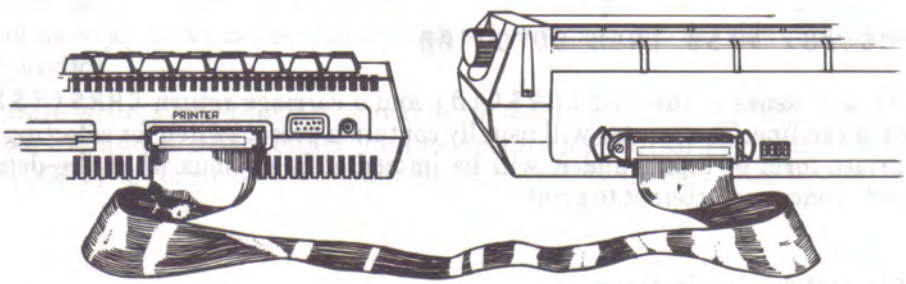
Some printers may have characters similar to many of the AMSTRAD graphics characters listed in Appendix III, but it is unlikely that the character numbers will correspond exactly, so you will have to devise your own conversion table to suit the printer in question.

Although the printer interface is envisaged for use with low cost dot matrix printers, it will support daisywheel printers with a suitable interface, and also graphics plotters, and multi colour ink jet printers. The key to compatibility is the standard parallel interface.

Amstrad DMP-1

The Amstrad DMP-1 printer is a low cost printer, supplied complete with the necessary cable to connect directly to the printer port on the CPC464.

*** 50 CPS print speed**
*** Plain paper operation**
*** Cable and ribbon supplied**
*** Dot graphics**
*** Tractor fed paper**



The DMP-1 is an impact dot-matrix printer using plain paper, with a maximum print speed of 50 characters per second. The customised software in the DMP-1 includes dot graphics ability, with the capability to print complete screen 'dumps'.

The low cost, versatility and features specifically customised for the CPC464 make the DMP-1 an ideal printer for all 'hard copy' requirements.

For further details, contact *AMSOFT* at:
Brentwood House, 169 Kings Road, Brentwood, Essex CM14 4EF,
telephone 0277-230222

8 Concise reference guide to AMSTRAD BASIC

A listing of BASIC keywords illustrated by examples, giving the precise form of use and associated keywords.

Subjects covered in this chapter

* The form of the notation
* Special characters and their significance
* All AMSTRAD BASIC keywords in alphabetical order

This chapter contains a concise summary of the functions in the BASIC supplied in *ROM* with the CPC464. The range of features available represents an industry-standard implementation of the language with specific extensions to complement the hardware features of the CPC464.

8.1 Notation.

Special Characters

& or &H	Prefix for hexadecimal constant
&X	Prefix for binary constant
:	Separates statements typed on the same line
#	Prefix for stream director

Data types

Strings may be from 0 to 255 characters long, and are expressed as ‹string expression›. Strings may be appended to one another using the + operator, as long as the resulting ‹string expression› is less than 255 characters long.

Numeric data can be either Integer or real. Integer data is held in the range -32768....32767 and real data is held to a little over nine digits of precision in the range ±1.7E+38 with the smallest value above zero approximately 2.9E-39.

Type markers are % Integer, ! real, $ string.

A ‹numeric expression› is any expression that results in a numeric value: it may simply be numbers, or it may be a numeric variable, or it may be numbers operated on by variables. Just about anything that is not a ‹string expression›.

A ‹stream expression› refers to a ‹numeric expression› which identifies the screen, printer or cassette where the text is required to 'stream'.

An 'improper argument' means that a ‹numeric expression› has returned a value that is outside the range defined as permissable in that situation or that a command parameter is invalid in some way.and that BASIC has not accepted this as valid input.

KEYWORDS

Please note that AMSTRAD BASIC keywords are listed here using the form:

KEYWORD
Syntax/Function
Example
Description
Associated KEYWORDS

IMPORTANT:

Keywords are either

COMMANDS : operations that are executed directly.
FUNCTIONS : operations that are invoked as arguments in an expression

Brackets

`()` are required as part of the command or function. Other types of brackets used in the KEYWORD description are for the purposes of the description, and should not be typed as part of the line:
[] enclose optional items.
‹› enclose various expressions which are described in the subsequent description.

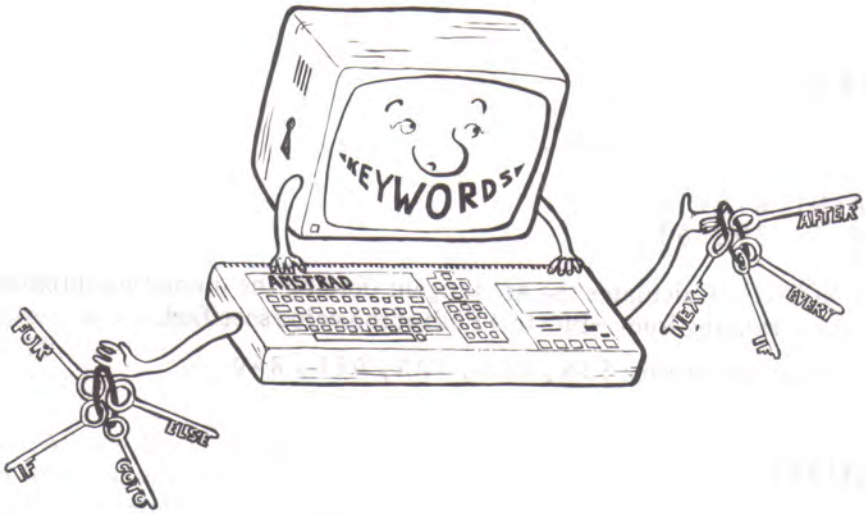
Quotes

Only " " form part of the actual BASIC program structure. " are used to emphasise or highlight certain aspects of the description. " should not appear in any part of the syntax/Function or Example entries other than within ‹string expression›s.

Entering

BASIC converts all keywords entered in `lower case` letters into `UPPER CASE` when a program is `LIST`ed. The examples shown here use `UPPER CASE`, since this is how the program will appear when `LIST`ed - if you enter using `lower case`, you will be able to spot typing errors more readily since the mistyped keyword will still be displayed in `lower case` when `LIST`ed.

Keywords are delimited by separators, since AMSTRAD BASIC allows you to 'bury' keywords into variable names: eg end2 and `LISTCODE` are acceptable as variables in AMSTRAD BASIC.



ABS

ABS(‹numeric expression›)

```
PRINT ABS(-67.98)
 67.98
```

FUNCTION: Returns the absolute value of the given expression -which primarily means that negative numbers are returned as positive.

Associated keywords: SGN

AFTER

AFTER ‹integer expression›[,‹integer expression›] GOSUB ‹line number›.

```
AFTER 200,2 GOSUB 320
```

COMMAND: Invoke a subroutine after a given time period has elapsed. The first ‹integer expression› indicates the period of the delay, in units of 1/50 second, and the second ‹integer expression› (in range 0....3), indicates which of the four available delay timers should be used. See also Chapter 10.

Associated keywords: EVERY, REMAIN

ASC

ASC(‹string expression›)

```
PRINT ASC("X")
 88
```

FUNCTION: Gets the numeric value of the first character of a string as long as ASCII characters are used.

Associated keywords: CHR$

ATN

ATN(‹numeric expression›)

```
PRINT ATN(1)
 0.785398163
```

FUNCTION: Calculates the arc-tangent (forcing the ‹numeric expression› to a real number ranging from −PI/2 to +PI/2) of the value specified.

Associated keywords: SIN, COS, TAN, DEG, RAD

AUTO

AUTO [‹line number›][,‹increment›]

```
AUTO 100,50
```

COMMAND: Generate line numbers automatically. The ‹line number› sets the first line to be generated, in case you want to add to the end of an existing program. The value of the ‹increment› between line numbers, and the first line number to be generated, both default to 10 if not specified.

Where an existing program line is in danger of being overwritten, BASIC inserts a star * after the line number generated as a warning.

BIN$

BIN$ (‹unsigned integer expression›[,‹integer expression›])

```
PRINT BIN$(64,8)
01000000
```

FUNCTION: Produces a string of binary digits that represents the value of the ‹unsigned integer expression›, filling with leading zeros to the number of digits instructed by the second ‹integer expression›.

Associated keywords: HEX$, STR$

BORDER

BORDER ‹colour›[,‹colour›]

```
BORDER 3,2
```

COMMAND: To change the colour of the border on the screen. If two colours are specified, the border alternates between the two at the rate determined in the SPEED INK command, if given. The range of border colours is 0....26.

Associated keywords: SPEED INK

CALL

```
CALL ‹address expression›[ ,‹list of: ‹parameter›]
CALL &BD19
```

COMMAND: Allows an externally developed sub-routine to be invoked from BASIC. Use with caution, not a function for the inexperienced to experiment with. The above `CALL` is relatively harmless, since it waits for the next frame flyback, which is particularly useful for tidying up the movement of characters around the screen when producing animation effects.

Associated keywords: `UNT`

CAT

```
CAT
```

```
CAT
```

COMMAND: Causes BASIC to start reading the cassette and to display the names of all files found. This does not affect the program currently in memory, and so may be used to verify a program that has just been saved before altering the program memory. The Command asks you to play the cassette, and on finding a program responds:

```
FILENAME   Block Number   Flag   Ok
```

Flags indicate the type of recording made:

$	a BASIC program file
%	a Protected BASIC file
*	an ASCII text file
&	a Binary file

Other characters may occur in this column if the file was not produced by BASIC.

Associated keywords: `LOAD, RUN, SAVE`

CHAIN
CHAIN MERGE

CHAIN ‹file name›[,‹line number expression›]
CHAIN MERGE ‹file name›[,‹line number expression›]
[,DELETE ‹line number range›]

```
CHAIN "TEST", 350
```

COMMAND: CHAIN loads a program from cassette into the memory; replacing the existing program. CHAIN MERGE *merges* a program from cassette into the current program memory. It adds the contents of a file to the current program in memory. The ‹line number expression› indicates the line number from which execution is to begin once the new program is chain merged. In the absence of ‹line number expression›, BASIC will default to the lowest line number available.

If no file name is stated, then BASIC will attempt to merge the first valid file encountered on tape. If the first character of the filename is a ! , then it is removed from the filename, and suppresses the usual messages generated by the cassette reading process.

CHAIN MERGE retains all current variables although User Functions and open files are discarded. ON ERROR GOTO is turned off, a RESTORE is implemented and the DEFINT, DEFREAL & DEFSTR settings are reset, and all active FOR WHILE and GOSUB commands are forgotten. Protected files will not merge.

Associated keywords: LOAD, MERGE

CHR$

CHR$(‹integer expression›)

```
PRINT CHR$(100)
 d
```

FUNCTION: Converts a numeric value to its character equivalent, (using the AMSTRAD CPC464 character set in Appendix III)

Associated keywords: ASC, LEFT$, RIGHT$, MID$, STR$

CINT

CINT(‹numeric expression›)

```
10 n=578.76543
20 PRINT CINT(n)
RUN
 579
```

FUNCTION: Converts the given value to a rounded integer in the range -32768...32767.

Associated keywords: CREAL, INT, FIX, ROUND, UNT

CLEAR

CLEAR

CLEAR

COMMAND: Clears all variables and files.

CLG

CLG[‹masked ink›]

CLG

COMMAND: To clear the graphics screen.

Associated keywords: CLS, ORIGIN

CLOSEIN

CLOSEIN

CLOSEIN

COMMAND: Close the cassette input file. Commands such as NEW and CHAIN MERGE will abandon any open files.

Associated keywords: OPENIN, CLOSEOUT

CLOSEOUT

CLOSEOUT

CLOSEOUT

COMMAND: Close the output cassette file.

Associated keywords: OPENOUT, CLOSEIN

CLS

CLS[#‹stream expression›]

CLS

COMMAND: To clear the given screen window to its Paper Ink.

CONT

CONT

CONT

COMMAND: Continue program execution after a *Break*, STOP or END, as long as the program has not been altered. Direct commands may be entered.

COS

COS(‹numeric expression›)

```
?COS(34)
-Ø.848570274
```

and

```
deg:?cos(34)
 Ø.829037573
```

FUNCTION: Calculates the COSINE of a given value. The function defaults to radian measure unless specifically instructed otherwise by the DEG command. Note in the above example the use of ? shortform for PRINT, and the use of lowercase entry for keywords - a feature fully compatible with AMSTRAD BASIC.

Associated keywords: SIN, TAN, ATN, DEG, RAD

CREAL

CREAL(‹numeric expression›)

```
5 DEFINT n
10 n=75.765
20 d=n/34.6
30 PRINT d
40 PRINT CREAL(n)
50 PRINT n/55.4
run
2.19653179
76
1.37184116
Ready
```

FUNCTION: Converts a value to a real number. (As opposed to integer).

Associated keywords: CINT

DATA

DATA ‹list of:‹constant›

```
10 REM Proofreader list
20 DIM Proofer$(5)
30 DIM ProoferSurname$(5)
40 FOR n=1 to 5
50 READ Proofer$(n)
60 READ ProoferSurname$(n)
65 PRINT Proofer$(n);" "ProoferSurname$(n)
70 DATA Bob,Smith,Dicky,Jones,
   Malcolm,Green,Alan,Brown,Ivor,Curry
90 NEXT
```

COMMAND: Declares constant data for use within a program. One of the most widely used features of BASIC that lumps constant data in DATA statements for retrieval as required. The data type must be consistent with the variable invoking it. A DATA statement may appear anywhere in a program.

Associated keywords: READ, RESTORE

DEF FN

DEF FN‹name›[(‹formal parameters›)]=‹general expression›

```
10 DEF FNinterest(principle)=1.14*principle
20 INPUT "What is the principle sum";principle
30 PRINT "The amount owing plus interest
   after one year is";FNinterest(principle)
```

COMMAND: BASIC allows the program to define and use simple value returning functions. DEF FuNction is the definition part of this mechanism and creates program-specific function which works within the program in the same way as a function such as COS operates as a built-in function of BASIC.

It may be invoked throughout the program. Variable types must be consistent and the DEF FuNction command should be written in part of the program outside the execution loop.

DEFINT
DEFSTR
DEFREAL

DEFtype ‹range(s) of letters›

```
DEFINT I-N
DEFSTR A,W-Z
DEFREAL
```

COMMAND: Define default variable types where 'type' is integer, real or string. The variable will be set according to the first letter of the variable's name -which may be either upper or lower case.

Associated keywords: LOAD, RUN, CHAIN, NEW, CLEAR

DEG

```
DEG
```

```
DEG
```

COMMAND: Set degrees mode. The default condition is for functions such as SIN and COS is to assume radian measure for numeric data. The command sets to degree mode until instructed otherwise by a CLEAR or RAD - or if any new program is loaded.

Associated keywords: RAD

DELETE

DELETE ‹line number range›

```
DELETE 100-200
```

COMMAND: A command that removes part of the current program as defined in the ‹line number range› expression. Not recoverable if issued in error, so use with care, and check for mistyping before entering.

Associated keywords: NEW

DI

```
DI

10 CLS
20 TAG
30 EVERY 10 GOSUB 100
40 X1=RND*320:X2=RND*320
50 Y=200+RND*200
60 FOR X=320-X1 TO 320+X2 STEP 2
70 DI:PLOT 320,0,1:MOVE X-2,
   Y:PRINT " ";:MOVE X,Y:PRINT "#";:EI
80 NEXT
90 GOTO 40
100 MOVE 320,0
110 DRAW X+8,Y-16,0
120 RETURN
```

COMMAND: Disable interrupts (other than the *Break* interrupt) until re-enabled explicitly by EI or implicitly by the RETURN at the end of an interrupt GOSUB routine.

Used when the program wishes to get on literally without interruption - for example when two routines within a program are competing for use of resources. In the example above, the main program and the interrupt subroutine are competing for the use of the graphics display.

Associated keywords: EI

DIM

DIM ‹list of: ‹subscripted variable›

```
10 CLS:PRINT "Enter 5 names....":PRINT
20 DIM B$(5)
30 FOR N=1 TO 5
40 PRINT "Name"N"please";
50 INPUT B$(N)
60 NEXT
70 FOR N=1 TO 5
80 PRINT "How wise of you ";B$(N);
  " to buy a CPC464"
90 NEXT
```

COMMAND: Allocate space for arrays and specify maximum subscript values. Basic must be advised of the space to be reserved for an array, or it will default to 10. Once set either implicitly or explicitly, the size of the array may not be changed, or an error will result.

A ‹subscripted variable› is one where the same variable name can take a series of values as set out in the list of integer numbers that comprise the 'dimension list'. The first number in the dimension list may be thought of as levels in a multistorey car park, and the subsequent numbers, the number of parking bays etc. A full understanding of arrays is a major element in advanced BASIC programming. The size of an array is limited only by available memory, and the programmers ability to keep track of the entries in the dimension list.

Associated keywords: ERASE

DRAW

DRAW ‹x co-ordinate›,‹y co-ordinate›[,‹masked ink›]

```
DRAW 200,200,13
```

COMMAND: Draws a line on the screen from the current graphics cursor position to an absolute position. The co-ordinate positions remain unchanged between the three different screen modes. Further examples in Chapter 5.

Associated keywords: DRAWR, PLOT, PLOTR, MOVE, MOVER, TEST, TESTR, XPOS, YPOS, ORIGIN

DRAWR

DRAWR ‹x offset›,‹y offset›[,‹masked ink›]

```
DRAWR 200,200,13
```

COMMAND: To draw a line on the screen from the current graphics cursor position to a position relative to it.

Associated keywords: DRAW, PLOT, PLOTR, MOVE, MOVER, TEST, TESTR, XPOS, YPOS, ORIGIN

EDIT

`EDIT` ‹line number›

```
EDIT 110
```

COMMAND: Edit a program line by calling for a specific line number. See Chapter 1.4.

Associated keywords: `LIST`

EI

`EI`

```
EI
```

COMMAND: To Enable Interrupts disabled by a `DI` command.

Associated keywords: `DI`

END

`END`

```
END
```

COMMAND: End of program. An `END` is implicit in AMSTRAD BASIC as the program passes the last line of instruction. `END` closes all cassette files and returns to the direct mode. Sound queues will continue until empty.

Associated keywords: `STOP`

ENT

`ENT` ‹envelope number›[`,`‹envelope sections›]

```
10 ENT 1,100,2,20
20 SOUND 1,100,1000,4,0,1
```

COMMAND: While a sound is being generated, it is possible to vary its tone. A tone envelope defines how the tone is to be varied. See Chapter 6 and Appendix VII for a fuller description of sound and its operation.

The ‹envelope number› is an ‹integer expression› whose absolute value must be in the range 1..15, which specifies the tone envelope to set. If the ‹envelope number› is negative, then the envelope is repeated.

Up to five ‹envelope sections› may be supplied, and each may take one of the forms :
‹step count›,‹step size›,‹pause time›
or: =‹tone period›,‹pause time›

The first form specifies an incremental change relative to the current tone period setting. The second form specifies an absolute setting for the tone period. Where :
‹step count› gives the number of steps in the section - an ‹integer expression› in the range 0..239.

‹step size› gives the amount by which to vary the tone period at each step in the envelope - an ‹integer expression› yielding a value in the range -128..+127.

‹pause time› gives the time to wait between steps - an ‹integer expression› specifying the time in 0.01 second units. The expression must yield a value in the range 0..255 (where 0 is treated as 256).

‹tone period› gives the new setting for the period - an ‹integer expression› yielding a value in the range 0..4095.

The `SOUND` command sets the initial tone period, and may specify one of the fifteen tone envelopes. If no envelope, or an envelope which has not been set up is specified, then the tone remains constant throughout the sound.

A tone envelope has no effect on the duration of the sound. If there are steps remaining in the tone envelope when the sound finishes, they are simply abandoned.

A repeating tone envelope will be restarted each time it finishes until the sound terminates.

The expressions in the tone envelope are evaluated when the command is executed and the results stored away for future use. Using the tone envelope does not cause the command to be re-executed.

Each time a given tone envelope is set, its previous value is lost. Changing an envelope while a sound using it is active or pending will produce indeterminate (but possibly interesting) effects.

Specifying an envelope with no sections cancels any previous setting. Any further use of the envelope will be ignored and the default used instead.

Associated keywords: `ENV`, `SOUND`

ENV

`ENV` ‹envelope number›[`,`‹envelope sections›]

```
10 ENV 1,100,2,20
20 SOUND 1,100,1000,4,1
```

COMMAND: While a sound is being generated, it is possible to vary its volume using this command. A volume envelope consists of:

‹step count›`,`‹step size›`,`‹pause time›

and defines how the volume is to be varied, by specifying a number of steps in the range 0..127, the size of the step in the range -128....+127, and the pause time in 0.01 second intervals in the range 1....256.

The ‹envelope number› is an ‹integer expression› yielding a value in the range 1..15 specifying the volume envelope to set.

Up to five ‹envelope sections› may be given. Each may take one of the forms:
‹step count›,‹step size›,‹pause time›

or: =‹hardware envelope›,‹envelope period›

The first form specifies an envelope section under software control, where the parameters are :

‹step count› gives the number of steps in the section - an ‹integer expression› in the range 0..127.

‹step size› gives the amount by which to vary the amplitude at each step in the envelope - an ‹integer expression› in the range -128....+127.
or : if the step count is zero, then the value to set the amplitude to - that is an absolute setting.

‹pause time› gives the time to wait between steps - an ‹integer expression› specifying the time in 1/100th's of a second. The expression must yield a value in the range 0....255 (where 0 is treated as 256).

The second form specifies an envelope section to be executed directly by the sound hardware, where :

‹hardware envelope› is the value to be set into the envelope shape register (register 15, octal).

‹envelope period› is the value to be set into the envelope period registers (registers 13 & 14, octal).

Hardware envelope settings do not have an associated pause time, so the next section of the envelope is immediately executed. It is advisable therefore that the next step should have a pause of a suitable length. If there is no next step, then a pause of 2 seconds is taken.

The SOUND command sets the initial volume, and may specify one of the fifteen volume envelopes. If no envelope, or an envelope which has not been set up, is specified, then the volume remains constant throughout the sound.

Setting a ‹step size› of zero with a non-zero step count causes the current volume setting to be maintained.

The expressions in the amplitude envelope are evaluated when the command is executed and the results stored away for future use. Using the amplitude envelope does not cause the command to be re-executed.

Each time a given amplitude envelope is set, its previous value is lost. Changing an envelope while a sound using it is active or pending will produce indeterminate (but possibly interesting) effects.
Specifying an envelope with no sections cancels any previous setting. Any further use of the envelope will be ignored and the default used instead.

Associated keywords: ENT, SOUND

EOF

```
EOF
```

```
PRINT EOF
-1
```

FUNCTION: Tests to see if the cassette input is at the end of the file. Returns -1 (true) at the end, otherwise 0 (false).

Associated keywords: `OPENIN`

ERASE

`ERASE` ‹list of; ‹variable name››

```
ERASE A,B$
```

COMMAND: When an array is no longer required, it may be `ERASE`d and the memory used be reclaimed ready for other use.

Associated keywords: `DIM`

ERR
ERL

```
ERR
ERL
```

```
10 CLS
20 ON ERROR GOTO 1000
30 DATA SALLY,EMMA,JOANNE,HELEN,GEMMA
40 READ A$
50 PRINT A$
60 GOTO 40
70 REM error detection starts here
1000 IF ERR=4 THEN PRINT "I have spotted
 a DATA EXHAUSTED error!"
1010 IF ERL<70 AND ERL>20 THEN PRINT
 ".....in lines 30-60"
1020 END
```

VARIABLES: These variables are used in error handling subroutines to discover the error number and line where the error occured. See the error message listing in Appendix VIII.

Associated keywords: `ON ERROR, ERROR`

ERROR

ERROR ‹integer expression›

```
ERROR 17
```

COMMAND: Take Error action with a given error number. The error may be one already used and recognised by BASIC (Appendix VIII), in which case the action taken is the same as would be taken if such an error had been detected by BASIC. Error numbers beyond those recognised by BASIC may be used by the program to signal its own errors.

Associated keywords: ON ERROR, ERR, ERL

EVERY

EVERY ‹integer expression›[,‹integer expression›] GOSUB ‹line number›

```
EVERY 500,2 GOSUB 50
```

COMMAND: The CPC464 maintains a real time clock. The EVERY command allows a BASIC program to arrange for subroutines to be called at regular intervals. Four delay timers are available, specified by the 2nd ‹integer expression› in the range 0....3 each of which may have a subroutine associated with it. See also Chapter 10.

Associated keywords: AFTER, REMAIN

EXP

EXP(‹numeric expression›)

```
PRINT EXP(6.876)
 968.743625
```

FUNCTION: Calculates 'E' to the power given in ‹numeric expression› - where 'E' is approximately 2.7182818 - the number whose natural logarithm is 1.

Associated keywords: LOG

FIX

FIX(‹numeric expression›)

```
PRINT FIX(9.99999)
 9
```

FUNCTION: Unlike CINT, FIX merely removes the part of the ‹numeric expression› to the right of the decimal point, and leaves an integer result, rounding towards zero.

Associated keywords: CINT, INT, ROUND

FOR

FOR ‹simple variable› = ‹start› TO ‹end› [STEP ‹size›]

```
FOR DAY=1 to 5 STEP 2
```

COMMAND: Execute a body of program a given number of times, stepping a control variable between a start and an end value. If not specified, STEP defaults to 1.

Associated keywords: NEXT, WHILE

FRE

FRE(‹numeric expression›)

FRE(‹string expression›)

```
PRINT FRE(Ø)
PRINT FRE("")
```

FUNCTION: Establishes how much memory remains unused by BASIC. The form FRE("") forces a 'garbage collection' before returning a value for available space.

GOSUB

GOSUB ‹line number›

```
GOSUB 21Ø
```

COMMAND: Call a BASIC subroutine by branching to the specified line number. See RETURN.

Associated keywords: RETURN

GOTO

GOTO ‹line number›

```
GOTO 9Ø
```

Branch to specified line number.

HEX$

HEX$(‹unsigned integer expression›[,‹integer expression›])

```
PRINT HEX$(65534)
FFFE
```

FUNCTION: Converts the number given into Hexadecimal form. (See Appendix II). The second ‹integer expression› can be used to specify the minimum length of the result.

Associated keywords: BIN$, STR$

HIMEM

HIMEM

```
?HIMEM
 43903
```

VARIABLE: Gives the address of the highest byte of memory used by BASIC, and can be used in numeric expressions in the usual way.

Before resetting the highest byte of available memory using the MEMORY command, it is advisable to issue the command mm=HIMEM. You will thereafter be able to return to the previous memory capacity by issuing the command: MEMORY mm.

Associated keywords: FRE, MEMORY

IF

IF

IF ‹logical expression› THEN ‹option part› [ELSE ‹option part›]

IF ‹logical expression› GOTO ‹line number› [ELSE ‹option part›]

```
IF A>B THEN A=C ELSE A=D

IF A>B GOTO 1000 ELSE 300
```

COMMAND: A very widely used command that is used to conditionally determine branch points in a program. The logical part is evaluated, and if true the THEN or GOTO part is executed, if false, the program skips to the ELSE part, or merely passes onto the next line. The statements may be nested to any depth, limited by line length. The IF is terminated by the line end. It is not permissable to have further statements that are independent of the IF on the same line.

Associated keywords: THEN, ELSE, GOTO, OR, AND, WHILE

INK

```
INK ‹ink›,‹colour›[,‹colour›]

INK 0,1
```

COMMAND: Depending on the current Screen Mode (Chapter 5), a number of INKs are available. The colour, or colours, used for an INK may be changed by an INK command, according to the table of colour values in Appendix IV.

Associated keywords: PEN, PAPER

INKEY

```
INKEY(‹integer expression›)
10 CLS:IF INKEY(55)=32 THEN 30 ELSE 20
20 CLS:GOTO 10
30 PRINT "You're pressing [SHIFT] and V"
40 FOR t=1 TO 1000:NEXT:GOTO 10
```

FUNCTION: This function interrogates the keyboard to report which keys are being pressed. The keyboard is scanned at 1/50 sec. The function is particularly useful for spotting Y/N responses, since the state of shift is not required according to one of the interpretation options. The above example detects when **[SHIFT]** and **V** are pressed together. Shift and control are identified according to

Value returned	[SHIFT]	[CTRL]	KEY
-1	?	?	UP
0	UP	UP	DOWN
32	DOWN	UP	DOWN
128	UP	DOWN	DOWN
160	DOWN	DOWN	DOWN

Associated keywords: INPUT, INKEY$

INKEY$

```
INKEY$

10 CLS
20 PRINT "Are you clever (y or n) ?"
30 a$=INKEY$: IF a$="" GOTO 30
40 IF a$="N" OR a$="n" THEN
   PRINT "You must have been to buy me!":END
50 IF a$="Y" OR a$="y" THEN
   PRINT "You're too modest!!!":END
60 GOTO 20
```

FUNCTION: Reads a key from the keyboard to provide operator interaction without hitting **[ENTER]** after every answer. If there is a key pressed, then the function responds - if no key is pressed, it continues to return an empty string which is used to loop until a valid input is detected for processing.

Associated keywords: INPUT, INKEY

INP

INP(‹port number›)

```
PRINT INP(&FF77)
```

FUNCTION: A function that returns the input value from the I/O port specified in the address.

Associated keywords: OUT, WAIT

INPUT

INPUT [#‹stream expression›,][;][‹quoted string›;] ‹list of: [variable]›
or INPUT [#‹stream expression›,][;][‹quoted string›,] ‹list of: [variable]›

```
10 CLS
20 INPUT "Give me two numbers, separated by
 a comma ";A,B
30 IF A=B THEN PRINT "The two numbers
 are the same"
40 IF A>B THEN PRINT A "is greater than" B
50 IF A<B THEN PRINT A "is less than" B
60 CLEAR:GOTO 20
```

COMMAND: Reads data from the stated stream. A semicolon after INPUT suppresses the carriage return typed at the end of the line being entered. A semicolon after the ‹quoted string› causes a question mark to be displayed. A comma suppresses the question mark. If an entry is made that is of the wrong type (eg a letter O was typed instead of a 0 in a ‹numeric expression›, then BASIC will prompt with:

```
?Redo from start
```

..and the original prompt text that you entered.

All responses must be terminated with an **[ENTER]**. The semicolon immediately after the ‹stream expression› has the effect of suppressing the carriage return typed at the end of the line being entered, leaving the cursor at the end of the text just entered. Where a cassette stream is indicated, no prompt is generated. If one is specified, it will be ignored by the cassette software, so the same program may read from either stream.

One item will be read from the stream for each variable in the list given. It must be compatible with the type specified in the INPUT command, which is: a numeric variable, terminated either by comma, carriage return, white space or end of file. Commas or **[ENTER]**s sent after trailing space will be ignored. Quoted strings will be read verbatim until terminated by double quotes, subsequent entries are ignored as for numeric values. Unquoted string items are terminated as in the case of numeric values.

Associated keywords: LINE INPUT, READ, INKEY$

INSTR

```
INSTR([‹integer expression›,]‹string expression›, ‹string expression›)
```

```
PRINT INSTR(2,"BANANA","AN")
```

FUNCTION: Searches the first ‹string expression› for the first occurance of the second ‹string expression›, where the optional number at the start indicates where to start the search - otherwise the search begins at the first character of the first ‹string expression›.

Associated keywords: `MID$, LEFT$, RIGHT$`

INT

```
INT(‹numeric expression›)
```

```
PRINT INT(-1.995)
-2
```

FUNCTION: Rounds the number to the nearest lower integer, removing any fractional part. The same as `FIX` for positive numbers, but returns one less than `FIX` for negative numbers not already integers.

Associated keywords: `CINT, FIX, ROUND`

JOY

```
JOY(‹integer expression›)
```

```
10 IF JOY(0) AND 8 THEN GOTO 100
```

FUNCTION: The `JOY` function reads a bit-significant result from the joystick specified in the ‹integer expression› (either 0 or 1).

Bit	Decimal
0: Up	1
1: Down	2
2: Left	4
3: Right	8
4: Fire 2	16
5: Fire 1	32

Associated keywords: `INKEY`

KEY

KEY ‹integer expression›,‹string expression›

```
KEY 140,"RUN"+CHR$(13)
```

COMMAND: Fixes a new function key definition. There are thirty two keyboard 'expansion' characters in the range 128-159 listed in Appendix III. When one of these characters is read it is expanded into the string associated with it - although the total number of expansion characters cannot exceed 100. The KEY command associates a string with a given expansion character.

Associated keywords: KEY DEF

KEY DEF

KEY DEF‹key number›,‹repeat›[,‹normal›[,‹shifted›[,‹control›]]]

```
KEY DEF 46,1,63
```

COMMAND: Associates the value as defined in Appendix III to a key on the keyboard. The above example converts the **N** key to print the question mark character **?**. (Character number 63).

To return the key to its normal function:

```
KEY DEF 46,1,110
```

where the character number 110 is the lower case n.

Associated keywords: KEY

LEFT$

LEFT$(‹string expression›,‹integer expression›)

```
10 CLS
20 A$ = "AMSTRAD"
30 B$ = LEFT$(A$,3)
40 PRINT B$
RUN
   [SCREEN CLEARS]
AMS
Ready
```

FUNCTION: Extracts characters to the left of, and including the position specified in the ‹integer expression› from the the given ‹string expression›. If the ‹string expression› is shorter than the required length, the whole ‹string expression› is returned.

Associated keywords: MID$, RIGHT$

LEN

LEN(‹string expression›)

```
A$="AMSTRAD":PRINT LEN(A$)
7
```

FUNCTION: Returns a number corresponding to the number of all types of characters, including spaces, in the ‹string expression›.

LET

LET ‹variable›=‹expression›

```
LET x=100
```

COMMAND: A remnant from early BASICs where variable assignments had to be 'seen coming'. No use apart from providing compatibility with the programs supplied in early BASIC training manuals. The above example need only be typed:

```
x=100
```

using AMSTRAD BASIC.

LINE INPUT

LINE INPUT [‹#‹stream expression›,][;][quoted string;]‹string variable›
LINE INPUT [‹#‹stream expression›,][;][quoted string,]‹string variable›

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

COMMAND: Reads an entire line from the stream indicated. The first optional semicolon suppresses the echo of carriage return / line feed. The default ‹stream expression› is, as always, #0 :screen.

Associated keywords: READ, INPUT, INKEY$, INPUT$

LIST

LIST [‹line number range›][,#‹stream expression›]

```
LIST 100-1000,#1
```

COMMAND: List program lines to the given stream. 0 is the default screen, 8 is the printer. LISTing may be suspended by pressing **[ESC]** once, and restarted by pressing the space bar. A double **[ESC]** will return BASIC to the direct mode. Programs may be listed to previously defined windows to assist in debugging programs without overwriting the entire screen area. Listing may be performed from the start of a program to a given point, or from a specified line number to the program end, by omitting the first or last numbers in the ‹line number range›. eg.

```
LIST -200     or     LIST 30-
```

LOAD

LOAD ‹file name›[,‹address expression›]

```
LOAD "INVENT"
```

COMMAND: To read a BASIC program from cassette into memory, replacing any existing program, or if using the optional address expression, to load a binary file into memory. See Chapter 2.

LOCATE

LOCATE [#‹stream expression›,]‹x coord›,‹y coord›

```
10 MODE 1
20 LOCATE 20,12
30 PRINT CHR$(249)
```

COMMAND: Moves the text cursor at the stream indicated, to the position specified by the x and y co-ordinates, which are relative to the origin of the stream (WINDOW). Stream 0 is the default stream.

Associated keywords: WINDOW

LOG

LOG(‹numeric expression›)

```
?LOG(9999)
 9.21024037
```

FUNCTION: Calculates the natural logarithm of ‹numeric expression›.

Associated keywords: EXP, LOG10

LOG10

LOG10(‹numeric expression›)

```
?LOG10(9999)
 3.99995657
```

FUNCTION: Calculates the base 10 logarithm of ‹numeric expression›.

Associated keywords: EXP, LOG

LOWER$

LOWER$(‹string expression›)

```
A$="AMSTRAD":PRINT LOWER$(A$)
amstrad
```

FUNCTION: Returns a new string expression the same as the input ‹string expression› but in which all upper case characters are converted to lower case. Useful for processing input where the answers may come in mixed upper/lower case.

Associated keywords: UPPER$

MAX

MAX(‹list of: ‹numeric expression›)

```
10 n=66
20 PRINT MAX(1,n,3,6,4,3)
```

FUNCTION: Extracts the largest value from the list of ‹numeric expression›s

Associated keywords: MIN

MEMORY

MEMORY ‹address expression›

```
MEMORY &20AA
```

COMMAND: Reset BASIC memory parameters to change the amount of memory available by setting the address of the highest byte. See the description of the keyword HIMEM. To examine the amount of memory, use the FRE command.

Associated keywords: HIMEM, FRE

MERGE

```
MERGE [‹file name›]

MERGE "PLAN"
```

COMMAND: Merge a program from cassette into the current program memory. It adds the contents of a file to the current program in memory. If no file name is stated, then BASIC will attempt to merge the first valid file encountered on tape. If the first character of the filename is a '!', then it is removed from the filename, and has the effect of suppressesing the usual messages generated by the cassette reading process.

If you wish to merge a program into memory without overwriting the current program, then the current program lines should be RENUMbered to a range different from those in the incoming program.

All variables, User Functions and open files are discarded. ON ERROR GOTO is turned off, a RESTORE is implemented and the DEFINT, DEFREAL & DEFSTR settings are reset. Protected files will not merge.

Associated keywords: LOAD, CHAIN, CHAIN MERGE

MID$

```
MID$(‹string›,‹integer expression›[,‹integer expression›])

A$="AMSTRAD":PRINT MID$(A$,2,4)
MSTR
A$="AMSTRAD":b$=MID$(A$,2,4):PRINT b$
MSTR
```

COMMAND and FUNCTION: MID$ specifies part of a string (a sub-string) which can be used either as the destination of an assignment (MID$ as a command) or as an argument in a string expression (MID$ as a Function). The first ‹integer expression› specifies the position of the first character of the sub-string.

The second ‹integer expression› specifies the length of the sub-string to be returned. If omitted, this extends to the end of the original string.

Associated keywords: LEFT$, RIGHT$

MIN

```
MIN(‹list of:‹numeric expression›)

PRINT MIN(3,6,2.999,8,9)
 2.999
```

FUNCTION: Extracts the smallest value from the list of ‹numeric expression›s

Associated keywords: MAX

MODE

MODE ‹integer expression›

MODE 1

COMMAND: To change the screen mode (0,1 or 2), and clear the screen to INK Ø, which may not be the current PAPER INK. All text and graphics WINDOWS are reset to the whole screen, and the text and graphics cursors homed to their respective origins.

Associated keywords: WINDOW, ORIGIN

MOVE

MOVE ‹x coord›,‹y coord›

MOVE 34,34

COMMAND: To move the graphics cursor to a position specified by the absolute co-ordinates. YPOS and XPOS are the corresponding functions to establish the current graphics cursor position.

Associated keywords: MOVER, PLOT, PLOTR, DRAW, DRAWR, ORIGIN, TEST, TESTR, XPOS, YPOS

MOVER

MOVER ‹x offset›,‹y offset›

MOVER 34,34

COMMAND: To move the graphics cursor to a position relative to the current co-ordinates. YPOS and XPOS are the corresponding functions to establish the current graphics cursor position.

Associated keywords: MOVE, PLOT, PLOTR, DRAW, DRAWR, TEST, TESTR, XPOS, YPOS

NEW

NEW

NEW

COMMAND: Delete current program and variables. KEY definitions are not lost, and the display is not cleared.

NEXT

NEXT [‹list of:‹variable›]

```
FOR n=1 TO 1000:NEXT
```

COMMAND: Delimits the end of a FOR loop. The NEXT command may be anonymous, or may refer to its matching FOR, which in the above example would be:

NEXT n

Associated keywords: FOR

ON GOSUB
ON GOTO

ON ‹integer expression› GOSUB ‹list of:‹line number›

ON ‹integer expression› GOTO ‹list of:‹line number›

```
10 ON DAY GOSUB 100,200,300,400,500
10 ON RATE GOTO 1000,2000,3000,4000
```

COMMAND: GOSUB to the subroutine, or GOTO the statement as directed by the result of the ‹integer expression›. If the result is 1, then the first line number in the list is chosen, if 2 then the second etc. In the above line 10, when DAY = 1, the subroutine at line 100 would be visited. DAY=2 would cause a branch to line 200, and so on.

Associated keywords: GOTO, GOSUB

ON BREAK GOSUB

ON BREAK GOSUB ‹line number›

```
10 ON BREAK GOSUB 40
20 PRINT "program running"
30 GOTO 20
40 CLS
50 PRINT "pressing [ESC] twice calls GOSUB
 routine"
60 FOR t=1 TO 2000:NEXT
70 RETURN
```

COMMAND: Calls a subroutine on breaking from program execution by pressing **[ESC]** twice.

Associated keywords: ON BREAK STOP, RETURN

ON BREAK STOP

```
ON BREAK STOP
```

```
10 ON BREAK GOSUB 40
20 PRINT "program running"
30 GOTO 20
40 CLS
50 PRINT "pressing [ESC] twice calls GOSUB
 routine"
60 FOR t=1 TO 2000:NEXT
65 ON BREAK STOP
70 RETURN
```

COMMAND: When issued in an ON BREAK interrupt subroutine, ON BREAK STOP disables the trap, but has no other immediate effect. In the above program, which includes ON BREAK STOP, the ON BREAK GOSUB trap will only operate once.

Associated keywords: ON BREAK GOSUB

ON ERROR GOTO

```
ON ERROR GOTO ‹line number›
```

```
10 ON ERROR GOTO 80
20 CLS
30 PRINT"if there is an error, I would"
40 PRINT"like the program listed, so that"
50 PRINT"I can see where I went wrong"
60 FOR t=1 TO 4000:NEXT
70 GOTO 200
80 CLS:PRINT"THERE IS AN ERROR IN LINE";ERL:PRINT
90 LIST
```

COMMAND: Go to a specified line number in the program on detecting an error. In this example, an error will be found in line 70.

Associated keywords: ERR, ERL, RESUME

ON SQ GOSUB

```
ON SQ (‹channel›) GOSUB ‹line number›
```

```
ON SQ (2) GOSUB 2000
```

COMMAND: Enable an interrupt for when there is a free slot in the given sound queue. The ‹channel› is an integer expression yielding one of the values:

1: for channel A
2: for channel B
4: for channel C

Associated keywords: SOUND, SQ

OPENIN

```
OPENIN ‹filename›

100 OPENIN "!INFORMATION"
```

COMMAND: Opens an input file from cassette which contains information for use in the current program in the computer's memory.

If the first character in the ‹file name› is ! then the displayed cassette processing messages are suppressed. The program reads in the first block from the cassette, ready for processing.

Associated keywords: `CLOSEIN, OPENOUT`

OPENOUT

```
OPENOUT ‹filename›

OPENOUT "!FACTS"
```

COMMAND: Opens an output file onto cassette for use with the current program in the computer's memory. If the first character in the ‹file name› is ! then the displayed cassette processing messages are suppressed. The program creates the first block of data, in the file with the given name. Each block consists of up to 2048 bytes of data.

NB A `NEW` command will abandon any open file buffered, and data will be lost.

Associated keywords: `CLOSEOUT, OPENIN`

ORIGIN

`ORIGIN` ‹x›,‹y›[,‹left›,‹right›,‹top›,‹bottom›]

```
10 CLS:BORDER 13
20 ORIGIN 0,0,50,590,350,50
30 DRAW 540,350
40 GOTO 20
```

COMMAND: Determines the start point for the graphics cursor. The [optional part] of the command contains the instructions to set a new graphics window, which will be operational in all screen modes due the pixel addressing technique employed.

The `ORIGIN` is the point with co-ordinates 0,0 (co-ordinates grow up and right).

If any of the window edges are specified to a position that is off the screen, they are assumed to represent the furthest 'visible' position in the given direction.

Associated keywords: `WINDOW`

OUT

`OUT` ‹port number›,‹integer expression›

```
OUT &F8F4,10
```

COMMAND: Sends the value in the ‹integer expression› (which must lie in the range 0....255) to the port specified in the ‹port number› by its address.

Associated keywords: `INP`, `WAIT`

PAPER

PAPER [#‹stream expression›,]‹masked ink›

```
10 MODE 0
20 FOR p=0 TO 15
30 PAPER p:CLS
40 PEN 15-p
50 LOCATE 6,12:PRINT "PAPER"p
60 FOR t=1 TO 500: NEXT t
70 NEXT p
```

COMMAND: Sets the background ink for characters. When characters are written to the text screen, the character cell is filled with the PAPER ink before the character is written - unless the transparent mode has been selected.

For PAPER/MODE/colour correlation, see Table 2, page F3.4.

Associated keywords: INK, WINDOW, PEN

PEEK

PEEK(‹address expression›)

```
10 MODE 2
20 INK 1,0: INK 0,12 : BORDER 12
30 INPUT "Start address for examination";first
40 INPUT "End address for examination";last
50 FOR n= first TO last
60 VALUE$=HEX$(PEEK(n),2)
70 PRINT VALUE$;
80 PRINT" at ";HEX$(n,4),
90 NEXT
```

FUNCTION: Examine the contents of a memory location. The above 'utility' program allows you to browse through the RAM of the CPC464. It reads the RAM under the lower (&0000-&3FFF) and upper (&C000-&FFFF) ROM - not the ROM.

Associated keywords: POKE

PEN

```
PEN [#‹stream expression›,]‹masked ink›

PEN 1,2
```

COMMAND: PEN sets the ink to be used when drawing at the given screen stream, defaulting to screen #0.

Associated keywords: INK, PAPER

PI

```
PI

PRINT PI
3.14159265

10 REM Perspective drawing
20 MODE 2
30 RAD
40 INK 1,0
50 INK 0,12
60 BORDER 9
70 FOR N= 1 TO 200
80 ORIGIN 420,0
90 DRAW 0,200
100 REM draw angles from vanishing point
110 DRAW 30*N*SIN(N*PI/4),(SIN(PI/2))*N*SIN(N)
120 NEXT
130 MOVE 0,200
140 DRAWR 0,50
150 DRAWR 40,0
160 WINDOW 1,40,1,10
170 PRINT"Now you can finish the
 Hangman program!"
```

FUNCTION: The value of the ratio between the circumference and the diameter of a circle. It is used extensively in graphics routines such as the one listed above.

Associated keywords: DEG, RAD

PLOT

PLOT ‹x co-ordinate›,‹y co-ordinate›[,‹masked ink›]

```
10 MODE 2:PRINT "Enter 4 numbers,
 separated by commas":PRINT
20 PRINT "Enter X origin (0-639),
 Y origin (0-399), radius and
  angle to step":INPUT x,y,r,s
30 ORIGIN x,y
40 FOR angle = 1 to 360 STEP s
50 XPOINT = r*COS(angle)
60 YPOINT = r*SIN(angle)
70 PLOT XPOINT,YPOINT
74 REM MOVE 0,0
75 REM DRAW XPOINT,YPOINT
80 NEXT
```

COMMAND: Try 320,200,20,1 as your first response. **PLOT** is the same as **MOVE**, except that the pixel at the destination is written. If you un-**REM** line 75 above and **REM** line 70 to make it inoperative, you will see the difference. (Un-**REM** line 74 to fill the circle).

Note that the process fills in the outline of the circle by repeated running around the perimeter. Remember that this program has not reset the **RAD**ian mode of angular calculation, so the 'angle' in each step is considerably more than one degree. Enter the command `25 DEG` and run again.

Associated keywords: **DRAW, DRAWR, PLOT, PLOTR, MOVE, MOVER, ORIGIN, TEST, TESTR, XPOS, YPOS**

PLOTR

PLOTR ‹x co-ordinate›,‹y co-ordinate›[,‹masked ink›]

```
5 DEG
10 MODE 2:PRINT "Enter 4 numbers,
 separated by commas":PRINT
20 PRINT "Enter X origin (0-639),
 Y origin (0-399), radius and
  angle to step":INPUT x,y,r,s
30 ORIGIN x,y
40 FOR angle = 1 to 360 STEP s
50 XPOINT = r*COS(angle)
60 YPOINT = r*SIN(angle)
70 PLOTR XPOINT, YPOINT
80 NEXT:GOTO 40
```

COMMAND: Try 320,0,2,1 in reponse. **PLOTR** is the same as **DRAWR**, except that only the pixel at the destination is written.

Associated keywords: **DRAW, DRAWR, PLOT, PLOTR, MOVE, MOVER, ORIGIN, TEST, TESTR, XPOS, YPOS**

POKE

POKE ‹address expression›,‹integer expression›

```
POKE &00FF,10
```

COMMAND: Provides direct access to the machine memory, and loads the ‹integer expression› in the range 0....255 directly at the address specified. Not to be used by the unwary.

Associated keywords: PEEK

POS

POS(#‹stream expression›)

```
PRINT POS(#0)
 1
```

FUNCTION: Establishes the current position for a given stream. In this instance there is no default for ‹stream expression›, and omitting it will result in a Syntax error message.

Screen: Returns the current 'X' co-ordinate of the text cursor, relative to the current window origin. The top left corner is represented as 1,1.

Printer: Returns the carriage position, where 1 is the left margin. All characters with ASCII reference numbers greater than &1F (31) are included.

Cassette output stream: As for the printer.

Associated keywords: VPOS

PRINT

PRINT [#‹stream expression›,][‹print list›][‹USING clause›][‹separator›]

```
PRINT #0,"abc"
```

COMMAND: For a full explanation of PRINT, see page 54 of this chapter.

Associated keywords: USING, TAB, SPC

RAD

```
RAD
```

```
RAD
```

COMMAND: Set Radians Mode

Associated keywords: DEG, SIN, COS, TAN, ATN

RANDOMIZE

```
RANDOMIZE [‹numeric expression›]
```

```
10 RANDOMIZE 23
20 PRINT RND(6)
```

COMMAND: BASIC's random number generator produces a pseudo random sequence in which each number depends on the previous number - starting from a given number, the sequence is always the same. RANDOMIZE sets a new initial value for the random number generator, either to a given value, or to a value entered by the operator. RANDOMIZE TIME will produce a sequence that will be difficult to repeat.

Associated keywords: RND

READ

```
READ list of:‹variable›
```

```
10 FOR X=1 TO 4
20 READ N$
30 PRINT N$
40 DATA ADAM,DANNY,JAMIE,RICHARD
50 NEXT
```

COMMAND: READ fetches data from the list of constants supplied in the corresponding DATA statements and assigns it to variables, automatically stepping to the next item in the data statement. RESTORE will return the pointer to the beginning of the DATA statement. See the DATA keyword.

Associated keywords: DATA, RESTORE

RELEASE

RELEASE ‹sound channels›

```
RELEASE 4
```

COMMAND: When a sound is placed on a sound queue it may include a 'hold' state. If any of the channels specified in this channel are in 'hold' state, then they are released. The expression to identify the sound channel is 'bit significant': A= bit 0, B= bit1, C= bit2. Thus 4 (binary 0100) releases Queue C.

Associated keywords: SOUND

REM

REM ‹rest of line›

```
10 REM Intergalatic Hyperspace Monster
   Invaders   Deathchase by AMSOFT
20 REM Copyright AMSOFT 1984
```

COMMAND: Used to place REMarks or REMinders in programs without affecting the program operation in any way. The : line separator is also ignored, everything from the REM to the line end is ignored. A single quote character ' in a line (not part of a ‹string expression›) is equivalent to :REM, other than in a DATA command, where it is treated as part of an unquoted string.

REMAIN

REMAIN (‹integer expression›)

```
REMAIN (3)
PRINT #6,REMAIN(0);
```

FUNCTION: Disables the specified delay timer (in the range 0....3).
Reads the remaining count from the delay timer. Zero is returned if the delay timer was not enabled.

Associated keywords: AFTER, EVERY

RENUM

RENUM [‹new line number›][,[‹old line number›][,‹increment›]]

```
RENUM
RENUM 100,,100
```

COMMAND: Renumber program lines from the line specified, using the increment specified. The ‹new line number› gives the first number for the new sequence, defaulting to 10. The ‹old line number›identifies where RENUM is to commence, and assumes the first program line if omitted. The ‹increment› sets the increment to use between the line numbers, again defaulting to 10.

RENUM takes care of all GOSUB, GOTO and other line calls. If all the specifiers are omitted from the command, the program is renumbered as if RENUM 10,,10 were issued.

Line numbers are valid in the range 1....65535.

RESTORE

RESTORE [‹line number›]

```
RESTORE 300
```

```
10 FOR N=1 TO 6
20 READ A$
30 PRINT A$;" ";
40 DATA restored,data,can,be,read,again
50 NEXT
60 PRINT
70 RESTORE
80 GOTO 10
```

COMMAND: Restores the position of the reading pointer back to the beginning of the DATA statement specified in the optional ‹line number›. Omitting ‹line number› restores the position of the pointer back to the beginning of the first DATA statement.

Associated keywords: READ, DATA

RESUME

RESUME [‹line number›]
or RESUME NEXT

```
RESUME 300
```

COMMAND: When an error has been trapped by an ON ERROR GOTO command, and has been processed, RESUME allows normal program execution to continue, the resuming line number being optionally specifiable. If not specified, the line in which the error has occurred is returned to. RESUME NEXT returns to the line immediately following the statement in which the error was detected.

Associated keywords: ON ERROR GOTO

RETURN

```
RETURN
```

```
RETURN
```

COMMAND: Signals the end of a subroutine. BASIC returns to continue processing at the point after the `GOSUB` which invoked it.

Associated keywords: `GOSUB`, `ON x GOSUB`, `ON SQ GOSUB`, `AFTER n GOSUB`, `EVERY n GOSUB`, `ON BREAK GOSUB`

RIGHT$

`RIGHT$(`‹string expression›`,`‹integer expression›`)`

```
10 CLS
20 A$ = "AMSTRAD"
30 B$ = RIGHT$(A$,3)
40 PRINT B$
RUN
[SCREEN CLEARS]
RAD
Ready
```

FUNCTION: Extracts the number of characters specified by the ‹integer expression› from the right of the ‹string expression›. If the ‹string expression› is shorter than the required length, the whole ‹string expression› is returned.

Associated keywords: `MID$`, `LEFT$`

RND

`RND[(`‹numeric expression›`)]`

```
10 RANDOMIZE 23
20 PRINT RND
```

FUNCTION: Fetches a random number, which may be the next in sequence, a repeat of the last one, or the first in a new sequence. The `RANDOMIZE` command in the above program ensures that `RND` returns the same number each time

`RND(0)` returns a copy of the last random number generated. Where ‹numeric expression› is negative, the number sequence generated is predictable.

Associated keywords: `RANDOMIZE`

ROUND

ROUND(‹numeric expression›[,‹integer expression›])

```
10 x=0.123456789
20 FOR r=9 TO 0 STEP -1:PRINT r,ROUND(x,r):NEXT
25 x=123456789
30 FOR r=0 TO -9 STEP -1
40 PRINT r,ROUND (x,r)
50 NEXT
```

FUNCTION: Rounds ‹numeric expression› to a number of decimal places or power of ten specified in ‹integer expression›. If the ‹integer expression› is less than zero, then value is rounded to give an absolute integer followed by a number of zeros determined by the ‹integer expression› before the decimal point.

Associated keywords: INT, FIX, CINT, ABS

RUN

RUN ‹string expression›

```
RUN "WELCOME"
```

COMMAND: Load a program from cassette and start executing it. If the ‹string expression› is empty "" BASIC attempts to load and execute the first file it encounters on the tape. If the first character of the string expression is ! then the displayed cassette processing messages are suppressed.

NB: BASIC effectively executes an implied NEW immediately a ‹filename› is read on the tape.

Associated keywords: LOAD

RUN

RUN [‹line number›]

```
RUN 100
```

COMMAND: Starts executing the current program at the line specified, or from the beginning if no line is specified. All current program, user functions and variables are deleted from memory. DEFINT, DEFREAL and DEFSTR settings are reset. All cassette files are abandoned, and any buffered output is lost.

Associated keywords: LOAD

SAVE

SAVE ‹filename›[,‹file type›][,‹binary parameters›]

```
SAVE "PROG",P
SAVE "BINARY",B,10000,16000,10003
```

COMMAND: Save the program in memory with name ‹filename›. Binary parameters comprise ‹start address›,‹file length›[,‹entry point›]

,A saves program in ASCII.
,P protects file.
,B saves an area of memory as a binary file - e.g. the screen.

Associated keywords: LOAD, RUN ‹filename›,MERGE,CHAIN, CHAIN MERGE

SGN

SGN(‹numeric expression›)

```
10 INPUT "What's your current Bank Balance";CASH
20 IF SGN(CASH) <1 GOTO 30 ELSE 40
30 PRINT "Oh dear, oh dear":END
40 PRINT"You've got more money than me"
```

FUNCTION: Determines the sign of the ‹numeric expression›. Returns -1 if ‹numeric expression› is less than 0, returns 0 if ‹numeric expression› = 0, and returns 1 if ‹numeric expression› is greater than zero.

Associated keywords: ABS

SIN

SIN(‹numeric expression›)

```
PRINT SIN(PI/2)
 1
```

FUNCTION: Calculates the Real value for the Sine of ‹numeric expression›, defaulting to the Radian measure mode unless otherwise declared by a DEG command.

Associated keywords: COS, TAN, ATN, DEG, RAD

SOUND

SOUND ‹channel status›,‹tone period›[,‹duration›[,‹volume›[,‹volume envelope›[,‹tone envelope›[,‹noise period›]]]]]

```
SOUND 1,200,1000,7,0,0,1
```

COMMAND: The SOUND features of the CPC464 is one of the most complex extensions to BASIC and is introduced in chapter 6.

Associated keywords: ENV, ENT

SPACE$

SPACE$(‹integer expression›)

```
SPACE$(190)
```

FUNCTION: Creates a string of spaces of the given length.

Associated keywords: PRINT, SPC, TAB.

SPEED INK

SPEED INK ‹integer expression›,‹integer expression›

```
5 INK 0,9,12:INK 1,0,26
10 BORDER 12,9
20 SPEED INK 50,20
```

COMMAND: The INK and BORDER commands allow two colours to be associated with each Ink, in which case the INK alternates between the two colours. The first ‹integer expression› specifies the time for the first INK specified to be used, and the second ‹integer expression› sets the time for the second INK. Times between colour changes are measured in units of 1/50 second. (50 Hz mains versions)
You must exercise careful judgement to avoid mesmeric effects when selecting colours and repeat rates!

Associated keywords: INK, BORDER

SPEED KEY

SPEED KEY ‹start delay›,‹repeat period›

```
SPEED KEY 20,3
```

COMMAND: If held down continuously, the keys on the CPC464 auto repeat at the ‹repeat period› after the given ‹start delay› period. The setting is made in 1/50 sec units, in the range 1....255. The default rate is set to 30,2

Very small start delays will interact with keyboard de-bounce routines. The actual speed at which the keyboard is 'read' by the software is not affected by this command.

Not all keys repeat, the KEY DEF commnd will allow the user to redefine the particular attributes of a given key.

Associated keywords: KEY DEF

SPEED WRITE

SPEED WRITE ‹integer expression›

```
SPEED WRITE 1
```

COMMAND: The cassette can be witten at either 2000 baud (where ‹integer expression› is 1), or the default of 1000 baud (where the ‹integer expression› is 0). When loading a file from tape, the CPC464 automatically establishes the correct reading speed from information in the file software, thus it is not necessary for the user to specify.

When using cassette tape of uncertain data recording ability, the 1000 baud rate is recommended for maximum reliability, see the notes in Chapter 2 for further information.

Associated keywords: SAVE

SQ

SQ(‹channel›)

```
10 MODE 1
20 FOR n=20 TO 0 STEP-1
30 PRINT n;
40 SOUND 1,10+n,100,7
50 WHILE SQ(1)>127:WEND
60 NEXT
```

FUNCTION: The SQ function is used to check the number of free entries in the queue for a given channel, where channel A is 1, B is 2, and C is 3. The function determines whether the channel is active - and if not - why the entry at the head of the queue (if any) is waiting. The result is bit significant:

0,1,2 indicate the number of free entries in the queue
3,4,5 indicate the Rendezvous state at the head of the queue (if any)
6 is set if the head of the queue is held
7 is set if the channel is currently active

Associated keywords: SOUND, ON SQ GOSUB

SQR

SQR(‹numeric expression›)

```
PRINT SQR(9)
 3
```

FUNCTION: Returns the square root of ‹numeric expression›.

Associated keywords: PRINT

STOP

STOP

```
300 IF n<0 THEN STOP
```

COMMAND: To stop execution of a program, but leave BASIC in a state where the program can be restarted after the STOP command by using the CONT command. This may be used to interrupt the program at a particular point when debugging.

Associated keywords: CONT, END

STR$

STR$(‹numeric expression›)

```
PRINT STR$(&766)
 1894

PRINT STR$(&X1010100)
 84
```

FUNCTION: Converts the ‹numeric expression› to a decimal string representation in the same form as used in the PRINT command.

Associated keywords: VAL, PRINT, HEX$, BIN$

STRING$

STRING$(‹integer expression›,‹character specificier›)

```
PRINT STRING$(&16,"*")
**********************
```

FUNCTION: Delivers a ‹string expression› consisting of the specified character repeated a number of times.

Associated keywords: SPACE$

SYMBOL

SYMBOL ‹character number›,‹list of:‹row›

```
5 MODE 2
10 SYMBOL AFTER 90
20 SYMBOL 93,&80,&40,&20,&10,&8,&4,&2,&1
30 FOR n=1 TO 2000
40 PRINT CHR$(93);
50 NEXT
60 GOTO 60
```

COMMAND: The SYMBOL command redefines the representation of a given character that has first been specified in the SYMBOL AFTER command. The ‹character number› is chosen from the available ASCII or other characters from the CPC464's standard character set, and the following entries define the new character on an 8x8 pixel matrix. A 0 in the row indicates the paper colour to be used, and a 1 indicates that the pixel is to be set to the current ink colour. Also see Appendices II and III. The example above produces a backslash that goes diagonally across the character cell, accessible by pressing the] key.

Associated keywords: SYMBOL AFTER

SYMBOL AFTER

SYMBOL AFTER ‹integer expression›

```
SYMBOL AFTER 90
```

COMMAND: The number of user definable characters is set by the SYMBOL AFTER command. The default setting is 240, giving 16 user defined characters. If the ‹integer expression› is 32, then all characters from 32 to 255 are redefinable.

Whenever a SYMBOL AFTER command is used, all user defined characters are reset to the default condition.

Associated keywords: SYMBOL

TAG

TAG [#‹stream expression›]

```
10 MODE 2
11 BORDER 9
14 INK 0,12
15 INK 1,0
20 FOR n=1 TO 100
30 MOVE 200+n,320+n
40 TAG
50 IF n<70 GOTO 60 ELSE 70
60 PRINT"Hello";:GOTO 80
70 PRINT" Farewell";
80 NEXT
90 GOTO 20
```

COMMAND: Text sent to a given stream may be redirected to be written at the graphics cursor position. This allows text and symbols to be mixed with graphics. The ‹stream expression› defaults to 0 if omitted. The top left of the character cell is tagged to the graphics cursor, and non-printing control characters will display. In particular, 'new line' characters will display if the PRINT statement is not followed by a semi-colon ;

Associated keywords: TAGOFF

TAGOFF

TAGOFF [#‹stream expression›]

```
TAGOFF #0
```

COMMAND: Cancels the TAG for a given stream, and sends the text to the previous text cursor position at the point at which TAG was invoked.

Associated keywords: TAG

TAN

```
TAN(‹numeric expression›)
```

```
PRINT TAN(45)
```

FUNCTION: Calculates the tangent for the angles given in ‹numeric expression›, which must be in the range -200,000....+200,000, defaulting to radian measure unless declared otherwise by a DEG command.

Associated keywords: COS, SIN, ATN, DEG, RAD

TEST

```
TEST(‹x co-ordinate›,‹y co-ordinate›)
```

```
PRINT TEST(300,300)
```

FUNCTION: Reports the value of the ink currently at the specified graphics screen location.

Associated keywords: TESTR, MOVE, MOVER, PLOT, PLOTR, DRAW, DRAWR

TESTR

```
TESTR(‹x offset›,‹y offset›)
```

```
TESTR(5,5)
```

FUNCTION:Moves the graphics cursor to the specified location and reports the value of the ink there.

Associated keywords:TEST, MOVE, MOVER, PLOT, PLOTR, DRAW, DRAWR

TIME

```
TIME
```

```
10 DATUM = INT(TIME/300)
20 TICKER=((TIME/300)-DATUM)
30 PRINT TICKER;
40 GOTO 20
```

FUNCTION: Reports the elapsed time since switch-on, excluding periods when reading or writing the cassette. The units of time are 1/300th of a second.

TRON
TROFF

```
TRON
TROFF

TRON
```

COMMAND: BASIC includes the facility to trace the execution of a program, by reporting the number of each line in square brackets `[]`, just before it is executed. `TRON` enables the feature, `TROFF` turns it off.

Associated keywords: `RUN`

UNT

`UNT(`‹address expression›`)`

```
PRINT UNT(65535)
-1
```

FUNCTION: Converts an unsigned 16-bit integer in the range 0....65535. Returns an integer value in the range -32768....+32767.

Associated keywords: `INT, FIX, CINT, ROUND`

UPPER$

`UPPER$(`‹string expression›`)`

```
PRINT UPPER$("amstrad")
AMSTRAD
```

FUNCTION: Returns a new string expression the same as the input ‹string expression› but in which all lower case characters are converted to upper case.

Associated keywords: `LOWER$`

VAL

`VAL(`‹string expression›`)`

```
10 A$="7 is my lucky number"
20 PRINT VAL(A$)
```

FUNCTION: Extracts a ‹numeric expression› from the beginning of the ‹string expression›. The opposite of `STR$`.

Associated keywords: `STR$`

VPOS

`VPOS(#‹stream expression›)`

```
PRINT VPOS(#0)
```

FUNCTION: Returns the vertical position of the text cursor for the ‹stream expression›

Associated keywords: `POS`

WAIT

`WAIT ‹port number›,‹mask›[,‹inversion›]`

```
WAIT &FF34,20,25
```

COMMAND: Suspends operation until a given I/O port returns a particular value in the range 0...255. BASIC loops whilst reading the I/O port. The value read is EXclusive ORed with the ‹inversion› and then `AND`ed with the ‹mask› until a non-zero result occurs. BASIC will get stuck in a `WAIT` loop if the required condition does not occur. If you type in the above example, you will have to fully reset the computer to escape.

Associated keywords: `INP, OUT`

WEND

```
WEND
```

```
10 MODE 1:REM BASIC CLOCK TIME ROUTINE
20 INPUT "Enter the current hour, minute, and
   second (h,m,s)";hour,minute,second
30 CLS:datum = INT(TIME/300)
40 WHILE hour<13
50 WHILE minute<60
60 WHILE tick<60
70 tick=(INT(TIME/300)-datum)+second
80 LOCATE 70,4
90 PRINT #0,USING "## ";hour,minute,tick
100 WEND
110 tick=0
115 second=0
120 minute=minute+1
130 GOTO 30
140 WEND
150 minute=0
160 hour=hour+1
170 WEND
180 hour=1
190 GOTO 40
```

COMMAND: The `WHILE/WEND` loop repeatedly executes a body of program until a given condition is true. The illustration here uses the `WHILE/WEND` loop to demonstrate the elegance of programs constructed using this approach. The body of the features of a variety of different clocks can now be added. The `WEND` command terminates the `WHILE` loop..

Associated keywords: `WHILE`

WHILE

`WHILE` ‹logical expression›

```
WHILE DAY <0
```

see example above

COMMAND: A `WHILE` loop repeatedly executes a body of program until a given condition is true. The `WHILE` command defines the head of the loop, and gives the condition. The `WEND` command terminates the `WHILE` loop.

Associated keywords: `WEND`

WIDTH

`WIDTH` ‹integer expression›

```
WIDTH 86
```

COMMAND: Tells BASIC how wide the printer is in characters, this information allows BASIC to insert carriage returns as required when printing.

Associated keywords: `PRINT, POS`

WINDOW

`WINDOW` [#‹stream expression›,]‹left›,‹right›,‹top›,‹bottom›

```
10 MODE 1
20 BORDER 6
30 WINDOW 10,30,7,18
40 PAPER 2:PEN 3
50 CLS
60 PRINT CHR$(143);CHR$(242);"THIS IS LOCATION"
70 PRINT "1,1 IN TEXT WINDOW"
80 GOTO 80
```

COMMAND: Sets a text window for a given screen stream.

Associated keywords: `ORIGIN`

WINDOW SWAP

`WINDOW SWAP` ‹stream expression›,‹stream expression›

```
WINDOW SWAP 0,2
```

COMMAND: Exchanges the text windows. For example, BASIC messages sent to stream #0 may be swapped with another window to highlight aspects of program development and operation.

Associated keywords: `WINDOW, PEN, PAPER, TAG`

WRITE

`WRITE` [#‹stream expression›,][‹write list›]

```
WRITE #2,"HELLO",4,5
"HELLO",4,5
```

COMMAND: Prints the values of a number of expressions to the given stream, separating them by commas and enclosing strings in double quotes. Used mainly for outputting data to cassette files.

XPOS

```
XPOS
```

```
PRINT XPOS
```

FUNCTION: Establishes the horizontal position of the graphics cursor.

Associated keywords: `YPOS`, `MOVE`, `MOVER`, `ORIGIN`

YPOS

```
YPOS
```

```
PRINT YPOS
```

FUNCTION: Establishes the vertical position of the graphics cursor.

Associated keywords: `XPOS`, `MOVE`, `MOVER`, `ORIGIN`

ZONE

`ZONE` ‹integer expression›

```
10 PRINT 1,2,3
20 ZONE 19
30 PRINT 4,5,6
```

COMMAND: Changes the width of the Print Zone used in `PRINT`, from the default value of 13 to a new value in the range 1....255. Reset by `NEW`, `LOAD`, `CHAIN` and `RUN` "‹file name›" commands.

Associated keywords: `PRINT`, `WIDTH`

PRINT

PRINT [#‹stream expression›,][‹print list›][‹USING clause›][‹separator›]

‹print list› is ‹print item›[‹separator›‹print list›]
‹print item› is ‹expression›
or SPC(‹integer expression›)
TAB(‹integer expression›)

Write data to a cassette file.

```
10 OPENOUT "DATA"
20 PRINT #9, "Hello"
30 CLOSEOUT
```

Write data to a line printer

```
10 BOBSSALARY =23000*PI
20 PRINT #8, USING "######.##";BOBSSALARY
30 PRINT #0, BOBSSALARY
RUN
 72256.6311
Ready
```

[Meanwhile, on the PRINTER.....]

```
 72256.63
```

COMMAND: Print data at ‹stream expression› using the format specified. See table for format characters.

Where no formats are specified, BASIC prints in 'free format', where a comma following the ‹print item› will send the printed item to start at the next print ZONE (default to 13). A semi-colon simply separates the expressions.

SPC(‹integer expression›) prints the given number of spaces, defaulting to zero if the ‹numeric expression› is less than 1. SPC does not require to be terminated by a comma or semi-colon - a semi-colon is assumed at all times.

TAB(‹integer expression›) prints spaces to move to the given print position, if less than 1, then 1 is assumed.

If the required position is equal to or exceeds the current position, spaces are printed to reach the required position - if less, then a carriage return is sent, followed by spaces to reach the required position. TAB does not require termination by a comma or semi-colon.

(Continued.....)

PRINT *(continued...)*

PRINT USING "<Format Field Specifiers>"

NUMERIC

Specifier	Possible Digits	Field Characters	Definition	Example
#	1	1	Numeric field	####
.	0	1	Decimal point	#.#
+	0	1	Print leading or trailing sign Positive number will have +	+## ###+
-	0	1	Trailing sign. Prints – if negative, otherwise blank	##.##-
**	2	2	Leading asterisk	**###.##
$$	1	2	Floating dollar sign $ is placed in front of the leading digit	$$##.##
**$	2	3	Asterisk fill and floating dollar sign	**$#.##
,	1	1	Use comma every three digits (left of decimal point only.)	##,###,#
↑↑↑↑	0	4	Exponential format. Number is aligned so leading digit is non-zero	#,##↑↑↑↑

STRING

!			First character only	!
\‹spaces›\			Number of spaces specified, plus a leading and trailing space character field	\ \
&			Variable length field	&

9 Further programming information

Subjects covered in this chapter

*** Text Legal Position**
*** Control Characters**
*** The Machine Operating System**
*** Interrupt structures**

9.1 Cursor locations and control code extensions

In a variety of applications programs, the text cursor may be positioned outside the current window. Various operations force the cursor to a legal position before they are performed, these are :

- writing a character
- drawing the cursor 'block'
- obeying the control codes marked with an asterisk in the list on the following pages.

The procedure for forcing the cursor to a legal position is as follows :

a. If the cursor is to the right of the right hand edge, then it is moved to the leftmost column of the next line down.

b. If the cursor is to the left of the left hand edge, then it is moved to the rightmost column of the next line up.

c. If the cursor is above the top edge, then the window is rolled down a line and the cursor is set to the top line of the window.

d. If the cursor is below the bottom edge, then the window is rolled up a line and the cursor is set to the bottom line of the window.

The tests and operations are done in the order given. The illegal cursor positions may be zero or negative, which are off to the left or above the window.

Character values (see appendix III) in the range 0....31 sent to the text screen do not produce a character on the screen (and should not be injudiciously applied) but are interpreted as CONTROL CODES. Some of the codes alter the meaning of one or more of the following characters, which are the code's parameters.

The codes marked * force the cursor to a legal position in the current window before they are obeyed - but may leave the cursor in an illegal position. The codes and their meanings are described with first their HEX value (&XX), then the decimal equivalent.

Additional control character commands: not generally accessible via the keyboard CTRL key

Value	Name	Parameter	Meaning
&00 0	NUL		No effect. Ignored.
&01 1	SOH	0..255	Print the symbol given by the parameter value. This allows the symbols in the range 0....31 to be displayed.
&02 2	STX		Turn off text cursor.
&03 3	ETX		Turn on text cursor. Note that BASIC uses an overriding cursor disable, which is released only when it is awaiting keyboard input.
&04 4	EOT	0..2	Set screen mode. Parameter taken MOD 4. Equivalent to a MODE command.
&05 5	ENQ	0..255	Send the parameter character to the graphics cursor.
&06 6	ACK		Enable Text Screen. (See &15, NAK, next page.)
&07 7	BEL		Sound Bleeper. Note that this flushes the sound queues.
&08 8 *	BS		Move cursor back one character.
&09 9 *	TAB		Move cursor forward one character.
&0A 10 *	LF		Move cursor down one line.
&0B 11 *	VT		Move cursor up one line.
&0C 12	FF		Clear text window and move cursor to top left corner. Equivalent to a CLS command.
&0D 13 *	CR		Move cursor to left edge of window on current line.
&0E 14	SO	0..15	Set Paper Ink. Parameter taken MOD 16. Equivalent to PAPER command.
&0F 15	SI	0..15	Set Pen Ink. Parameter taken MOD 16. Equivalent to PEN command.
&10 16 *	DLE		Delete current character. Fills character cell with current Paper Ink.
&11 17 *	DC1		Clear from left edge of window to, and including, the current character position. Fills affected cells with the current Paper Ink.

Value	Name	Parameter	Meaning
&12 18 *	DC2		Clear from, and including, the current character position to the right edge of window. Fills affected cells with the current Paper Ink.
&13 19 *	DC3		Clear from start of window to, and including, the current character position. Fills affected cells with the current Paper Ink.
&14 20 *	DC4		Clear from, and including, the current character position to the end of window. Fills affected cells with the current Paper Ink.
&15 21	NAK		Turn off text screen. The screen will not react to anything sent to it until after an ACK (&06 6) is sent.
&16 22	SYN	0..1	Parameter MOD 2 0 disables the transparent option 1 enables
&17 23	ETB	0..3	Parameter MOD 4 0 Sets Normal Graphics Ink Mode 1 " XOR " " " 2 " AND " " " 3 " OR " " "
&18 24	CAN		Exchange Pen and Paper Inks.
&19 25	EM	0..255 0..255 0..255 0..255 0..255 0..255 0..255 0..255 0..255	Set Matrix for User Definable Character. Equivalent to a `SYMBOL` command. Takes nine parameters. The first parameter specifies which character's matrix to set. The next eight specify the matrix : the most significant bit of the first byte corresponds to the top left hand pixel of the character cell, the least significant bit of the last byte corresponds to the bottom right hand pixel of the character cell.
&1A 26	SUB	1..80 1..80 1..25 1..25	Set Window. Equivalent to a `WINDOW` command. The first two parameters specify the left and right hand edges of the window - the smaller value is taken as the left edge, the larger the right. The second two parameters specify the top and bottom edges of the window - the smaller value is taken as the top edge, the larger the bottom edge.

Value	Name	Parameter	Meaning
&1B 27	ESC		No effect. Ignored.
&1C 28	FS	0..15 0..31 0..31	Set Ink to a pair of colours. Equivalent to an Ink command. The first parameter (MOD 16) specifies the Ink, the next two (MOD 32) the required colours.
&1D 29	GS	0..31 0..31	Set Border to a pair of colours. Equivalent to a BORDER command. The two parameters (MOD 32) specify the two colours.
&1E 30	RS		Move cursor to top left hand corner of window.
&1F 31	US	1..80 1..25	Move cursor to the given position in the current window. Equivalent to a LOCATE command. The first parameter gives the column to move to, the second the line.

9.2 Machine Operating System

The housekeeping of the CPC464 is provided by a sophisticated real time operating system. The operating system 'directs the traffic' through the computer from the input to the output.

It primarily interfaces the hardware with the BASIC interpreter - for example the ink flashing function, where the BASIC passes the parameters - and the OS gets on with the task, with one part determining what is required - and the other part determining the timing of these events.

The machine operating system is generally referred to as the 'firmware', and comprises the machine code routines that are called by the high level commands in the BASIC.

If you are tempted to POKE around in the machine memory addresses or CALL the subroutines - save your program and listing before doing so, or you may regret it! The extensive operating system firmware of the CPC464 is described in the advanced user guide, and is beyond the scope of this introductory user guide.

9.3 Interrupts

The CPC464 makes extensive use of the Z80 interrupt structure to provide an operating system that includes several multitasking features as exemplified by the AFTER and EVERY structure described in chapter 8. The precedence of the interrupts is:

Break **[ESC][ESC]**
Timer 3
Timer 2 (and the three sound channel queues).
Timer 1
Timer 0

Interrupts should be included after considering the consequences of all the possible intermediate variable states at the point of interruption. The interrupt subroutine itself should avoid unwanted interaction with the state of variable in the main program.

The sound queues have indepedant interrupts of equal priority. Once a sound interrupt has started, it is not interrupted by any other - enabling sound commands to share variables with immunity from the effects mentioned above for the timer interrupt's structure.

When a sound queue's interrupt is enabled, it will immediately interrupt if the sound queue for that channel is not full, otherwise it will interrupt when the next sound starts and there is room for more in the queue. The action of interrupting disables the event, so the subroutine must re-enable it if further interrupts are required. Attempting to issue a sound or testing the queue status will also disable any event.

The priority of the **[ESC] [ESC]** sequence above all other interrupts ensures that BASIC program operation can be halted without loss of the program - provided no ancillary action has been taken to ensure the integrity of the program through one of the various protection techniques.

9.3 AMSOFT Assembler

In order to program extensively using machine code, it will be necessary to use an assembler. The AMSOFT assembler comprises a relocatable Z80 assembler, with editor, disassembler and monitor.

10 Interrupt features

Subjects covered in this chapter:

* AFTER
* EVERY
* REMAIN
* The master clock

If you haven't already noticed, a major innovation in the software of the CPC464 is its unique ability to handle interrupts from BASIC - which means that AMSTRAD BASIC is capable of performing a number of simultaneous but separate operations within a program. Such a facility is sometimes referred to as *multitasking*, and it is implemented by the application of the new commands AFTER and EVERY.

This facility is also clearly demonstrated in the way in which sound may be handled through facilities such as queues and rendezvous.

Every aspect of timing is referred to the master system clock, which is a quartz controlled timing system within the main computer that looks after the timing and synchronisation of events that happen in the computer - things like the scanning of the display and clocking the processor. Where a function in the hardware is related to time, this can be traced back to the quartz master clock.

The software implementation is the AFTER and EVERY command, which in keeping with the user-friendly approach of AMSTRAD BASIC do precisely what they say: eg. AFTER the time you have preset in the command line, the program will divert to the designated subroutine and perform the task defined therein.

10.1 AFTER

The CPC464 maintains a real time clock. The AFTER command allows a BASIC program to arrange for subroutines to be called at some time in the future. Four delay timers are available, each of which may have a subroutine associated with it.

AFTER ‹integer expression›[,‹integer expression›] GOSUB ‹line number›

The first ‹integer expression› specifies how long before the subroutine should be called. This time is measured in 1/50ths of a second.

The second ‹integer expresssion› specifies which of the four possible delay timers is to be used. The expression must yield a value in the range 0..3. If the expression is omitted, 0 is assumed.

When the time specified has passed, the subroutine is called automatically, just as if a GOSUB had been issued at the current position in the program. When the subroutine finishes, using a normal RETURN command, the main program continues running where it was interrupted.

The timers have different interrupt priorities. Timer 3 has the highest priority and timer 0 the lowest.

AFTER commands may be issued at any time, resetting the subroutine and time associated with the given delay timer. The delay timers are the same as those used in the EVERY command, so an AFTER overrides any previous EVERY for the given timer, and vice versa.

```
10 MODE 1:X=0
20 AFTER 45 GOSUB 100
30 AFTER 100,1 GOSUB 200
40 PRINT "AMSOFT"
50 WHILE X<100
60 LOCATE #1,30,1:PRINT #1,X:X=X+1
70 WEND
80 END
100 PRINT "peripherals"
110 RETURN
200 PRINT "and software"
210 RETURN
```

Note the use of two streams (windows) to allow printing by the 'main program' (Lines 50-80) using independant cursor positioning from that used by the interrupt subroutines.

AFTER the time you have preset in the command line, the program diverts to the designated subroutine and performs the task defined therein:

10.2 EVERY

The EVERY command allows a BASIC program to arrange for subroutines to be called at regular intervals. Four delay timers are available, each of which may have a subroutine associated with it. Its form is:

EVERY ‹integer expression›[,‹integer expression›] GOSUB ‹line number›

The first ‹integer expression› specifies how long to wait between each call of the subroutine. This time is measured in 1/50ths of a second.

The second ‹integer expresssion› specifies which of the four possible delay timers is to be used. The expression must yield a value in the range 0..3. If the expression is omitted, timer 0 is assumed.

When the time specified has passed, the subroutine is called automatically, just as if a GOSUB had been issued at the current position in the program. When the subroutine finishes, using a normal RETURN command, the main program continues running where it was interrupted.

The timers have different interrupt priorities. Timer 3 has the highest priority and timer 0 the lowest. Immediately the timer expires, the count is reset and the count down to the next call of the subroutine begins.

EVERY commands may be issued at any time, resetting the subroutine and time associated with the given delay timer. The delay timers are the same as those used in the AFTER command, so an EVERY overrides any previous AFTER for the given timer, and vice versa.

```
10 MODE 1:X=0
20 P100=0:EVERY 10 GOSUB 100
30 P200=0:EVERY 12,1 GOSUB 200
40 PRINT "AMSOFT"
50 WHILE X<200
60 LOCATE #1,30,1:PRINT #1,X:X=X+1
70 WEND
80 LOCATE 1,20:END
100 DI:PEN P100:LOCATE 1,2:PRINT "peripherals":EI
105 IF P100=0 THEN P100=1 ELSE P100=0
110 RETURN
200 PEN P200:LOCATE 1,3:PRINT "and software"
205 IF P200=2 THEN P200=3 ELSE P200=2
210 RETURN
```

NOTE the use of DI and EI commands which disable and enable timer and sound interrupts whilst the commands between them are executed. This has the effect of delaying the (higher priority) interrupt of timer 1 from ever occurring during the processing of the interrupt from timer 0 (lines 100-110).Therefore, the PEN or LOCATE settings are not upset before the PRINT command.

10.3 REMAIN

This function returns the remaining count for one of the four system delay timers. It disables the timer, returning zero if the timer is already disabled. It is used in the form:

REMAIN (‹integer expression›)

Appendix I

The Art of the possible

The newcomer to computing frequently starts by establishing some points of reference that will help give an impression of the computer's capabilities and limitations. This section sets out to give a broad guide as to what happens and why.

Zap the wotsit!

Even if the only reason you bought your CPC464 was to take advantage of the sophisticated computer games available to run on the 'hardware', you may still probably be wondering about several aspects of the computer that come under the heading of 'hardware'.

The hardware is what you can pick up and carry around: the main computer keyboard, the monitor, the connecting leads etc. In fact, it's about everything that isn't specifically the software - programs, manuals, and cassette based information.

Certain features of the way the computer behaves are produced by courtesy of the hardware - things like the coloured display on the TV set (or monitor) - and then it's up to the software to make use of this hardware capability to produce specifically designed characters and shapes on the screen.

The hardware actually directs the beam of electrons at the electro luminescent surface on the inside of the screen of the TV tube to make it 'light up' - the software adds order and intelligence by telling the hardware when and how to perform. It adds timing, control and sequencing to produce the effect of a spaceship taking off, or something more mundane like a letter appearing when you type at the keyboard.

Q So what makes one computer better than another ?

Hardware without software is worthless. Software without hardware is equally worthless - the value of the computer begins when the two come together to perform various tasks. There are some very basic considerations that can be used to grade performance of both hardware and software.

The generally accepted reference points for personal computers are now:

1. The screen resolution - the smallest discernible item on the display.

This is a combination of factors, including the number of different colours available to the programmer, the number of different areas that can be resolved on the display - the pixels,and the number of text characters that can be displayed on a single screen area.

You will find that your CPC464 compares very favourably with any similarly priced machine in all these respects.

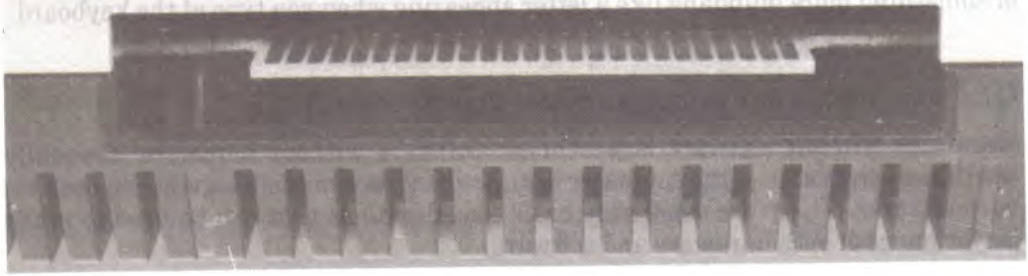
2. The BASIC interpreter

Virtually every home computer includes with it a BASIC interpreter that allows the user to start creating programs to use the hardware features. The built in programming language (BASIC) that comes supplied with your machine is itself a program - an immensely complicated and intricate program that has been evolved over a million man-years of experience since BASIC was 'invented' in the USA. The 'Beginners All-purpose Symbolic Instruction Code' is easily the most widely used computer language in the world, and like any language, it comes in a variety of local 'dialects'.

The version in the CPC464 is one of the most widely compatible dialects of BASIC, and will run many of the common BASIC programmes written for operation under the CP/M disk operating system. It is a very fast implementation of BASIC - in other words it performs its calculations quickly - and whilst you may not be too concerned that one computer may take 0.05 of a second to multiply 3 by 5, and display the answer whereas another may take 0.075 second to do the same - where a program that draws graphics patterns on the screen may call for many thousands of simple repetitive calculations, the difference between 0.05 second and 0.075 of a second adds up to a considerable difference in performance.

You will frequently hear the term 'machine code' being used. Machine code is the raw form of instruction code that can be passed to the processor. It takes less time to work out what it's been asked to, and gets on with producing the result some 5 to 15 times faster than an equivalent operation being passed along through the BASIC interpreter. But it can take 5 to 50 times longer to write an equivalent program in machine code when compared to performing the same overall task using BASIC.

The BASIC in your Amstrad computer is amongst the fastest and most fully featured to be found in any home computer system, and incorporates many features that help the experienced BASIC programmer overcome some of the inherent sluggishness of a 'high level language' interpreter to perform surprisingly dynamic visual and musical effects.



3. Expandability

Most computers pay attention to the need to 'add on' additional items of hardware: printers, joysticks, disk drives. Paradoxically, some of the most successful home computers require the addition of add-on units known as 'expansion interfaces' before even a simple printer or joystick controller can be installed.

The purchaser does not always think ahead to his needs in the future, otherwise a machine that incorporates a properly supported parallel printer (Centronics compatible) and a games joystick may actually be cheaper in the long run.

The CPC464 computer features a built-in Centronics printer port, facilities for up to two joysticks, a stereo sound output - and a comprehensive expansion bus that can be used to attach disk drive controllers, additional expansion ROMs, serial (RS232) interfaces etc.

A ROM (Read Only Memory) is an integrated circuit memory device that contains stored program information. The BASIC that is supplied with your computer is stored in one such ROM, and it is possible for other programs to be supplied either to supplement the built-in ROM, or replace its function completely.

TV games consoles use 'cartridge' software. The cartridge is basically an ROM (Read Only Memory - a form of integrated circuit that stores program information) supplied in a neat plastic housing with a suitable form of reusable connector that allows it to be taken in and out very easily. Thus a ROM provides the same function as a cassette for supplying program information. However, it loads the information into the computer virtually instantaneously as compared to the several minutes that it takes to load a large program from cassette, and so its major advantage is one of sheer convenience.

A ROM cannot be used to store information to be taken out and stored or transported to another computer in the same way that the cassette Datacorder does.

Expansion is the means to ensure that your computer can make the most of future developments in software and peripherals. The CPC464 system has a very complete and fully documented expansion capability.

4. Sound

The sound features of a computer determine whether or not it sounds like a bluebottle in a empty cocoa tin - or if it can produce an acceptable representation of an electronic musical instrument.

The CPC464 computer uses a 3 channel 8 octave sound generator, which can produce a very acceptable musical quality, with full control of the amplitude and tone envelopes. Furthermore, the sound is divided into a stereo configuration, where one channel provides the left output and one channel provides the right output- and the third channel sits in the middle.

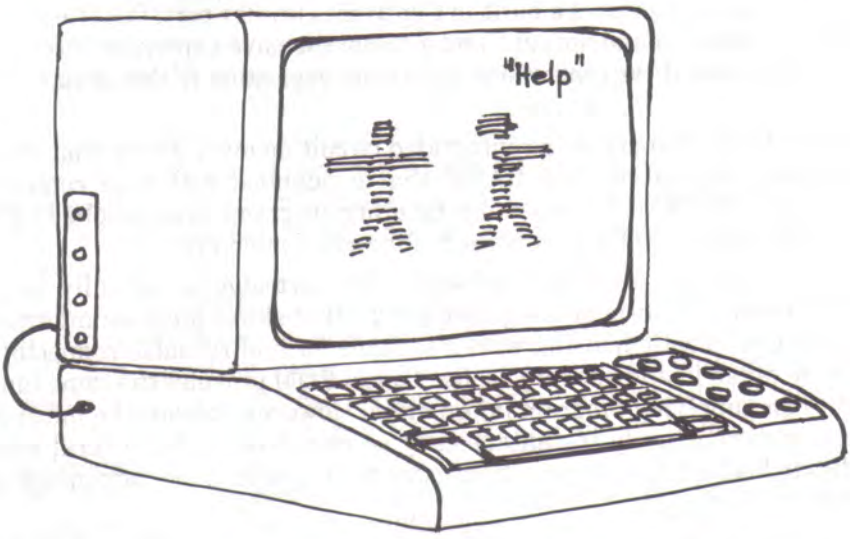
This provides considerable scope for writing programs that track the sound effects across the screen to follow the motion of an arcade-style game.

Ultimately, you will make your own mind up about which of these features is most important to you. We hope that you will try them all to make the most of your computer.

Why can't ?

With all the power of modern technology, users frequently wonder why even a machine as advanced as the CPC464 is apparently unable to display the sort the type of pictures seen on any TV set.

Q Why, for instance, can't a computer animate a picture of someone walking across the screen in a natural fashion - why do all computers represent movement with 'matchstick' figures ?



The answer is simple yet complex. The simple answer is that you must not beguiled into believing that the screen of your computer has anything of the subtlety of the screen of a TV set. A television set operates using linear information that can describe a virtually infinite range of resolution between the extremes of light and dark across all the colours of the spectrum. This process means that in computer terms, the display 'memory' of a full TV picture is some 20 times more than the converted equivalent of a home computer video display.

That's only part of the problem, since to animate this picture requires that this enormous amount of memory must be processed at high speed (around 50 times each second). It can be done - but only by machines that cost a few thousand times more than a home computer at least, for the time being!

Until the cost of high speed memory falls dramatically (it will eventually), small computers have to make do with a relatively small amount of memory available to control the screen display - which results in lower resolution, and jerkier movements. Thoughtful hardware design and good programming can go a long way to making the best of this situation, but we are still a way from cheap computers that can reproduce flowing motion and lifelike pictures in the same way that even a moderate animated cartoon can produce.

Q Why can't you simply walk up to the computer and type a page of simple text into the machine ?

Don't be mislead by the fact that the computer looks like a typewriter with an electronic display. The screen is not a piece of electronic paper - it's a command console - jargon which means that it simply provides you with the means of communicating with the programming language (and the programs) in the machine memory.

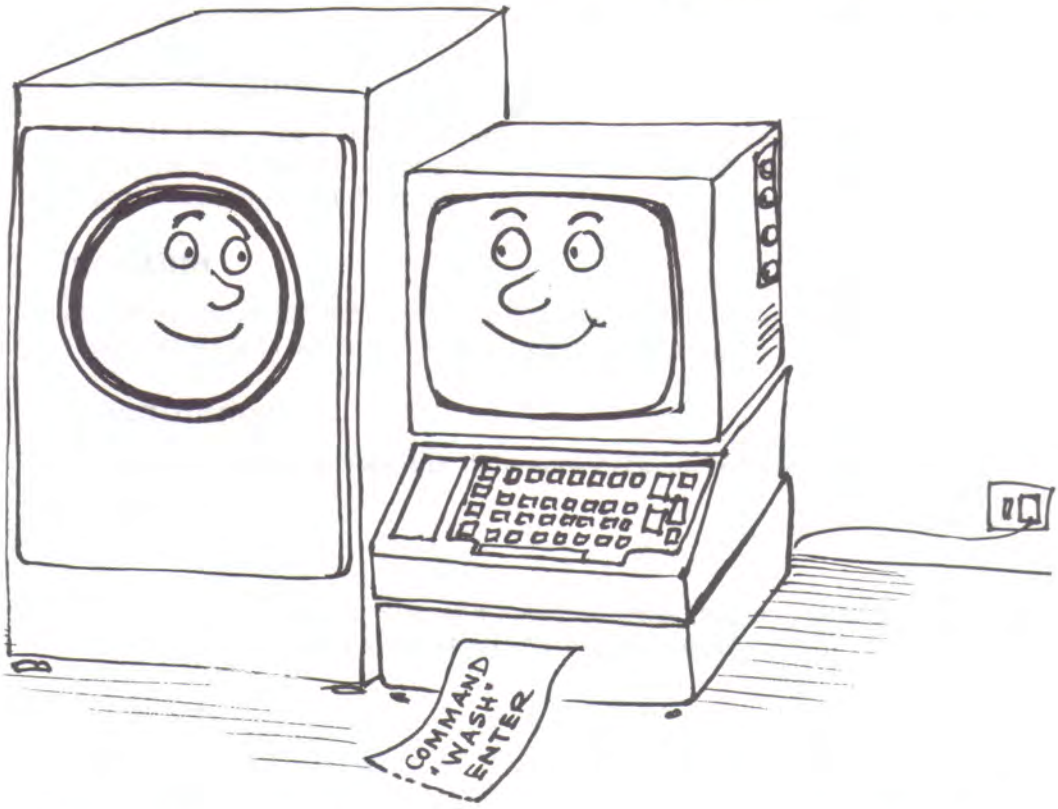
Until you tell it to the contrary, the computer will try and interpret all the characters that you type at the keyboard as being program instructions. When you press the **[ENTER]** key, the computer will look through what has been typed and if it doesn't make sense to the built in BASIC, it will reject the input with the comment:

```
Syntax error
```

However, it may just happen that the program presently residing in your computer is a Word Processor system, in which case you will be able to type random words, press **[ENTER]** and carry on typing as if the system were operating an electronic piece of paper in an electronic typewriter.

But to do this, you must first have loaded a wordprocessor program into the machine's memory using the datacassette input.

A computer 'seems' to combine several items of equipment that have become familiar around the home and office - the TV-like screen, the keyboard, the cassette recorder - you must remember that the similarities are generally strictly superficial, and that the computer is a combination of familiar looking hardware that has an entirely different personality of its own.



GLOSSARY OF TERMS

Some commonly used terms from the world of computing, explained for CPC464 users.....

Accumulator

A memory location within the microprocessor circuit at the heart of the microcomputer that stores data temporarily while it is being processed. Used extensively in machine code programming — BASIC users need never know it exists !

Acoustic coupler

Also known as an Acoustic Modem. An electronic attachment for a computer that connects a telephone handset to the computer and enables the latter to communicate over the normal voice telephone network. In this way, a computer can communicate with public information systems such as PRESTEL, other users of home computers etc., to exchange software, get data and information etc.

Address

The number in an instruction that identifies the location of a 'cell' in a computer's memory. By means of its address, a particular memory location can be selected so that its contents can be found and 'read', in the case of RAM, both read and written back after alteration.

Adventure game

A cult with some, and a bore to others. A text-based computer game in which the user playing is invited to participate in a series of pseudo random events based on trying to find a way around a maze or labyrinth.

Algorithm

A grandiose name for a complicated formula or sum. A sequence of logical and arithmetical steps to perform a defined task in computing.

Alphanumeric

The attribute that describes the difference between a letter or number and a graphics character.

ALU

Arithmetic Logic Unit. The part of a microprocessor that carries out arithmetical and logical operations - not of direct concern except in machine code programming.

AMSOFT

AMSTRAD's specialist computer support division, supplying software, peripherals, publications specifically to enhance the CPC464 and its many applications.

A/D
Analogue

A state where change between a start and finish point occurs gradually rather than by instantaneous steps. Computers are digital devices - most of the natural world is based on analogue principles, thus the computer has to perform an analogue to digital (A/D) conversion before it can process data from an analogue source.

Animation

Cartoons are the best known form of animation - computer animation is based on the concept of moving graphics to simulate the idea of 'live' movement.

Applications program

A program with a specific task rather than a general purpose software 'tool' such as an assembler, printer driver etc.

Arcade Game

The type of moving action computer video game where: spacemen invade and are vanquished, insatiable monsters chase around a maze and gobble up the unwary, figures under the control of the user have to avoid all manner of unpleasant 'deaths'. Fun and generally good for the reflexes but of little educational benefit for the computer student.

Architecture

The plan relationship of the databus, peripheral and CPU handling aspects of a microcomputer. Not a subject that will be concerning readers of this glossary -yet!

Argument

An independent variable. eg. in the expression `x+y=z`, the terms x,y and z, are the *arguments*.

Array

A matrix in which data is stored by addressing with the 'horizontal' and 'vertical' coordinates.

Artificial intelligence AI

A structure in program techniques that enables the program to learn from its past experience.

ASCII

American Standard Code for Information Interchange. A commonly used way of representing the numbers, letters and other symbols that can be entered from the computer's keyboard or invoked using a variety of other commands. The CPC464 codes are listed in Appendix III.

Assembler

The practical method for programming in machine code, where the machine code instructions are invoked by mnemonics (letters that suggest the function being performed by the corresponding machine code routine).

Bar code

Look on the bottom of an soap powder packet to see an example. A computer readable printed code that can be read by optical techniques such as scanning by a low power laser.

Base

The prime numeric consideration of any mathematician. The basis of any system of number representation. The binary system has base 2; the decimal system has base 10, and the hexadecimal system has the base 16 - see Appendix II for a fuller explanation.

BASIC

Beginners' All-purpose Symbolic Instruction Code. An interpretive programming language used in almost all home computers, BASIC was specifically designed to be easy to learn and simple to use since it allows for programs to be 'glued' together and tested at any point in their development, as opposed to compilers types where the complete program must be run before any aspect can be properly tested.

Baud

A bit per second :the unit for measuring the rate at which digital data is transmitted in serial communication systems.

BCD

Binary Coded Decimal. A coding system for decimal numbers in which each digit is represented by a group of four binary digits.

Benchmark

A standard task that can be given to different computers to compare their speed, efficiency and accuracy eg. the square root of 99.999 squared.

Binary

(See Base) The number system with base 2, in which all numbers are made up from the two binary digits 0 and 1.

Binary number

A number represented in binary notation. Signified in the CPC464 programming by the prefix &X. eg &X0101 (decimal) 5.

Bit

Shortform of BInary digiT. A binary digit is either one of the two digits, represented by 0 and 1, that are used in the binary number system.

Bit significant

Where the information contained in a number is extracted by considering the state of each of the eight bits that make up the complete byte. The overall decimal value may have no meaning.

Booting or Bootstrapping

Programs and operating systems don't load themselves, they are 'boostrapped' by a small routine in ROM (usually), that initiates the loading processes at a specific location in memory.

Boolean algebra

Is the statement of logical relationships where there can only be two answers: true or false. Usually signified as 0 or 1.

Buffer

A transient or temporary storage area to hold information during transfers from one part of the system to another, for example from the cassette drive software to the computer's central processing unit (CPU) and main RAM. A buffer regulates the way data is passed between devices operating at different speeds, such as a modem or a printer.

Bug

A problem on a scale ranging from an 'unexpected feature' based on some obscure aspect of the use of a program (eg if you press four keys at once, the screen changes colour), to a sequence that completely and irrevokably crashes a computer program and wipes the memory clean of all data.

Bus

A group of connections either within the computer, or connecting it to the outside world that carries information on the state of the CPU, the RAM and other hardware features. The CPC464 bus is presented on the larger of the two circuit board connector strips, accessible through the holes in the rear of the case.

Byte

A group of eight bits, which forms the smallest portion of memory that an 8-bit CPU can recall from, or store in memory. See additional information Appendix II.

CAD

Computer Aided Design. Usually an interaction of computing power and graphics to provide an electronic drawing board, although any calculation performed on a computer in pursuance of a 'design' comes under the heading of CAD.

CAE

Computer Aided Education. Further nourishment for the buzzphrase of computing. The use of the computer to help with education. CAI (Computer Aided Instruction) and CAL (Computer Aided Learning) are two aspects of CAE.

Cartridge

A specially packaged memory integrated circuit containing software which can be plugged directly into a socket specifically provided for the purpose on the computer. Cartridge software loads and runs more quickly and easily, but costs considerably more than software supplied on cassette.

Cassette

Apart from the obvious recording tape variety, a generic term that encompasses a variety of 'packages' -including ROM software etc.

Character

Any symbol that can be represented in a computer and displayed by it, including letters, numbers and graphics symbols. (See Appendix II)

Character cell

The matrix of dots on a display screen used to represent a single character may be displayed by selective illumination of some of the dots. (See Appendix II)

Character set

All the letters, numbers and symbols available on a computer or printer. The fact that a character exists on a computer does not imply it is accessible on any printer.

Character string

A piece of ‹variable› data comprising a sequence of characters that can be stored or manipulated as a single unit, e.g. a word or a collection of words.

Chip

An misleading but popular reference to any form of monolithic electronic integrated circuit. The 'chip' is actually a small slice of specially processed silicon material, on which the circuit is fabricated.

Clock

The reference timing system in the computer used to synchronise and schedule the operations of the computer. A *real time* clock is one that maintains the hour, date etc.

Code

Apart from the more literal implications, frequently used by programmers as an abbreviation of 'machine code'.

Command

A programming instruction.

Compiler

A complex program that converts complete programs written in a high level interpretive languages like BASIC into a the direct instruction code of the microprocessor, thereby enabling operation at much greater speeds.

Computer generations

Technological landmarks have delineated several distinct steps in computer technology, and the groupings within these various strata are known as the 'generations' of design technologies.

Computer literacy

Another grandiose expression meaning understanding computers.

CP/M

Digital Research's *de facto* standard for 8-bit computer disk operating sytems.

CPU

Central Processing Unit. The component at the heart of any computer system that interprets instructions to the computer and causes them to be obeyed, in a microcomputer, the CPU is the microprocesor device itself.

Cursor

A movable marker, indicating where the next character is to appear on the screen.

Cursor control keys

Keys that move the Cursor around the screen, and are frequently used to control the direction of action in arcade games, indicated by arrows printed on the top.

Daisy-wheel printer

A printer that can produce high quality or 'typewriter quality' documents. Printed characters are created by the impact of letters against the ink or film ribbon.

Database

An array of any type of data in a variety of computer addressable formats.

Data Capture

The term which describes the collection of data from any outside sources that are linked in some way to a central computer.

Debugging

The process of fixing the bugs in a program by a combination of 'suck it and see' and more scientific methods.

Decimal notation

Also known as the Denary system, for numbers with base 10, using the digits 0 to 9, representing numbers of units, tens, hundreds, thousands and so on.

Diagnostic

A message automatically produced by a computer to indicate and identify an error in a program.

Digital

Describes the expression of a changing quantity in terms of discrete steps rather than by a continuous process. The opposite of analogue.

Digitiser

A means of plotting analogue information into a computer. Commonly referred to in conjunction with graphics tablets.

Disk

A flat, thin circular piece of plastic coated on one or both sides with a magnetic oxide surface and used as a medium for storing data. The disk is housed at all times in a square protective envelope or plastic box, with access for the reading head provided by a 'window' in the case. Also see Floppy disk and Winchester.

Disk drive

The unit that records information on the magnetisable surface of a spinning disk and 'reads' (recovers) information recorded on it.

Documentation

The manuals that are supplied with computers or software to explain how they are operated.

Download

The transfer of information from one computer to another - the computer receiving the data is generally referred to as the machine downloading. The other end of the link is uploading.

DOS

Disk Operating System. The software that controls all the operations of a disk drive - acts like the attendant in the 'car park' represented by the logical layout of the disk.

Dot matrix

A rectangular grid of dots on which a character can be displayed by the selection of certain of the dots.

Dumb terminal

A computer terminal that simply acts as a medium for input and output without any processing of the information passing through. Note that a *mindless terminal* is one where even the display drive electronics are absent, and that the screen display information is fed in as pure video..

Editing

Correcting or making changes to data, a program or text.

Editor

A program that is usually in the ROM of the computer that enables the editing process to be carried out.

EPROM

Erasable Programmable Read only Memory. Similar to the PROM, except that the data contained in the chip can be erased using ultra-violet light and new programming recorded. An EEPROM is similar, except that it may be electrically erased.

Expression

A simple or complex formula used within a program to perform a calculation on data - the expression will usually define the nature of the data it can handle.

Fifth generation computers

Mainly large mainframe computers that are promised to arrive with the ability for self-programming using the developments of artificial intelligence.

File

A collection of information, generally stored on cassette or disk, although RAM files are supported on some computers .

Firmware

Software contained in ROM - a cross between pure software and pure hardware.

Fixed-point number

A number represented, manipulated and stored with the decimal point in a fixed specified position.

Floating-point number

A Real number, manipulated and stored with a decimal point permitted to settle in the required position. The method is particularly useful when dealing with large numbers.

Floppy disk

A removable 5.25 or 8 inch diameter magnetic disk, that is used to store computer data. Housed inside a protective square envelope. Much greater data storage capacity than a cassette, much faster, more expensive.

Flowchart

A diagrammatic representation of the progression of program steps and logical processes tracing the sequence of events during program execution.

Forth

A high speed programming language, with speed and complexity falling between a High-level language and Machine code. Not a beginners language.

Function key

A key on the keyboard that has been assigned a specific task -which may in addition to, or instead of the main purpose inscribed upon it. The CPC464 has a number of keys that may be defined as function keys, whereupon a single keystroke can issue up to 32 characters of text in the form of commonly used instructions, or instructions controlling peripheral equipment, such as modems or printers.

Gate

Logical gates permit the passage of data when certain conditions are fulfilled. There are many different types (OR, AND, XOR etc). See the entry for Boolean algebra.

Graphics

The part of the screen display of the computer that is not related to the display of 'characters' eg. drawing lines, circles, graphs etc. In conjunction with an appropriate printer, a paper print copy may also be obtained.

Graphics character

A shape or pattern specially designed to be useful in creating images. The CPC464 has a complete set of these described in Appendix III.

Graphics cursor

Similar to the text cursor, but addressing the graphics screen. An invisible concept on the CPC464 - but nevertheless an indispensible facility for locating drawn graphics. Not to be confused with graphics characters (Appendix II) which are still part of the 'character set', and printed at the text cursor.

Graphics mode

Early microcomputers required to be specifically set to either handle characters or graphics. Modern personal computers are capable of mingling text and graphics simultaneously.

Graphics tablet

A device that plots the co-ordinate points of a given picture or drawing for manipulation within the computer. A form of A/D.

Handshaking

A sequence of electronic signals which initiates, checks and synchronises the exchange of data between a computer and a peripheral, or between two computers.

Hard Copy

Paper printout of a program or other text - or of a graphics display. The transitory screen equivalent is known as 'soft copy'.

Hardware

The electronic and mechanical parts of a computer system -anything that isn't software or firmware.

Hexadecimal notation

Numbers based on 16. See Appendix II. Signified in the CPC464 by the prefix & or &H.

High-level language

Languages which are written in 'near literal' form, where the actual language does most of the work of interpreting. Slower than machine code orientated programs, but far simpler to understand, like BASIC.

IEEE-488

One of the standard Interfaces for connecting devices to a microcomputer. Similar to - but not wholly compatible with - the Centronics parallel interface in many respects

Information technology

Anything relating to the use of electronics in the processing of information and communications: wordprocessing, data communications, PRESTEL etc.

Initialise

Switch on a system, or declare specific values for variables before beginning to execute the body of program - such as declaring them to be integers etc.

Input

Anything that enters the computer memory from its keyboard, cassette unit, disk unit, serial interface or other input source.

Instruction

A request/command to a computer to perform a particular operation. A collection or sequence of instructions form a program.

Instruction set

The prime logical and mathematical processes carried out by the microprocessor. Every high level instruction (including assembler mnemonics) have to be capable of being distilled down to an instruction that is recognised by the computer CPU. A single high level command may invoke a large number of elements from the CPU's instruction set.

Integer number

A number with no fractional part. ie. a number with no part to the right of the decimal point - as opposed to a real number which is the integer part plus the fractional part.

Integrated circuit

A collection of electronic circuit components miniaturised and built onto a single piece of silicon. See also 'Chip'.

Intelligent terminal

A terminal where as well as handling the requirements of the computer's input and output, local processing power is also available when the terminal is 'off line'.

Interactive

Usually a reference to programs where the hardware computer prompts the user to provide various types of input - ranging from controlling the spaceship in an arcade game, to answering questions in educational programs. The action of the user has an effect in 'real time' on the behaviour of the program.

Interface

The way in and out of a computer, both in electrical and human terms. The CPC464 interface is the keyboard (input), and the screen (output) - as well as the facility at the rear for the connection of user peripherals to the interface bus.

Interpreter

A further extension of the analogy of computer instruction sets and language. The element of the system software that interprets the high level language to the level that can be understood by the CPU. eg. It converts BASIC code as enetered via the keyboard into the computers own internal language..

I/O

Input/Output.

Iteration

One of the elements of computing. The computer performs all tasks by breaking them down into the simple tasks that can be handled by the CPU. To do this, the computer must go to and from between many simple elements until a given condition is fulfilled.

Joystick

An input device that generally replaces the function of the cursor keys, and makes games playing faster and easier.

K

A shortform of the metric measure prefix for 1000, 'kilo' - which in computing has come to be widely used to refer to a 'kilobyte' - which is actually 1024 (decimal) in view of the binary association of 2 raised to the power of 10. See Appendix II.

Keyboard

The matrix of alphanumeric key switches, arranged to provide the means of typing commands and other information into the computer.

Keyword

A word whose use in the computer program or language is reserved for a specific function or command..

Least significant bit

In a binary number, (see Appendix II), the Least Significant Bit (LSB) is the bit at the extreme right hand end of the expression.

Light Pen

Another alternative input method, using a pen or 'wand'.

Line number

BASIC and some other languages use programs that are arranged in line number order.

Lisp

The acronym formed from LISt Processor language. Another high level computer language.

Logic

The electronic components that carry out the elementary logical operations and functions, from which every operation of a computer is ultimately built up.

Logo

A simple to learn graphics orientated high-level computer language frequently used in schools as an aid to teaching computing.

Loop

A process in a program that is executed repeatedly by the computer until a certain condition is satisfied.

Low-level language

Such as assembly language. A programming language in which each instruction corresponds to the computer's machine code instruction.

LSI

Large Scale Integration. The development of integrated circuits, packing more functions onto ever smaller pieces of silicon.

Machine Code

The programming language that is directly understood by a microprocessor, since all its commands are represented by patterns of binary digits.

Machine readable

A medium of data or any other information that can be immediately input to a computer without additional work on keyboarding etc.

Man-machine interface

A point of interaction between the computer and the operator: keyboard, screen, sound etc.

Matrix

The arrangement of the dots that form the character cell on the screen, or on the print head of a 'dot matrix' printer. Also a term used in mathematics and computer science to encompass arrays.

Memory

The computer's parking lot for information and data, neatly arranged in logical rows with each item individually accessible. Either known as RAM (random access memory) where information can be both stored and retrieved, or ROM, where the information may be read, but not re-written in another form. Disks and tape are example of 'bulk memory', althouth the term has evolved to mean the memory that is directly addressed by the CPU.

Memory map

The layout of the memory, showing the various addresses, and the allocation of the memory to specific function, such as the screen, the tape operating system etc.

Menu

A bill of fare of the different options that may be carried out by the program in the computer, left to the user to select.

Microprocessor

An integrated circuit that sits at the heart of a microcomputer and executes the instructions that are presented to it by the BASIC intepreter, in order to control the various output devices and options.

Modem

A MOdulator DEModulator that connects the computer's I/O to a telephone line or other serial data transmission medium -incuding fibre optics. Also see acoustic coupler.

Monitor

The screen section of a computer terminal system, and also a term describing a machine language program that provides access to the fundamental machine language operation of the computer.

MSB

The Most Significant Bit of a binary number: the bit at the left end of the expression.

Mouse

An upside-down tracker or roller ball. Pushed around a table top by hand, a mouse is generally used to move a cursor around the screen. Originally designed to overcome the fear of keyboards and make software appear more 'user friendly'.

Network

When two or more computers are linked together to exchange data and information - either by wiring, or via MODEMs.

Nibble

Half a byte: a four bit binary expression. Each of the hexadecimal digits in the expression &F6 represents 'one nibble'.

Noise

The CPC464 sound facility includes a facility to inject a variable amount of random noise by using the SOUND command to create effects such as explosions.

Numeric keypad

The area on the keyboard where number keys are grouped to facilitate entry of numeric data, and in the case of the CPC464, to provide the additional facility of user definable function keys.

OCR

Optical Character Recognition. A means of reading printed or written characters with an optical reader and translating them directly into computer readable data.

Octal

A number system to the base 8, where each digit (0-7) is constructed from three Bits.

Off line

A computer peripheral - usually a display terminal or a printer - that is not actively connected to , or accessible by, the main processing unit.

On line

The opposite of Off line.

Operating system

The attendant in the parking lot referred to under the entry for Memory. Software that allocates precedence and timing to the operations of the computer.

Output

Anything that comes from a computer as the result of some computational function.

Operator

The part of arithmetical expression that causes one number to operate on another - ie. + − ÷ etc..

Overwrite

Erase an area of memory by replacing its contents with new data.

Paddle

An alternative name for a joystick. Also referred to as a 'games paddle'.

Paperware

Another description for the printed 'hardcopy' of computing. Occasionally computers launched before they have actually finished development are described as a 'paperware exercise'.

Parallel interface

The CPC464 printer interface supports a parallel printer: which means that each data line from the bus is connected to a corresponding input on the printer. Data is transfered more quickly using a parallel interface than a serial interface since the serial interface must first format each byte, and frame it with synchronisation information.

Pascal

A high-level structured programming language that must be compiled before it will execute - and therefore runs very quickly. Generally the next language that the keen BASIC student will pursue.

PEEK

The BASIC function that looks directly into the computer's memory, and reports the value of the specified location.

Peripheral

Printers, modems, joysticks, disk drives - anything that plugs into the computer to expand its capabilities.

Pixel

The smallest accessible area of the screen that can be controlled by the hardware.

Plotter

A specific type of printer that draws 'longhand' using pens rather than an impact print head. Used for technical and graphical drawing output.

POKE

The statement in BASIC that is used to place a value in a specified memory location - see chapter 8.

Port

A specifically addressable point on the interface for input or output of data.

Portability

Other than the literal use, means the ability for software to operate on different makes of computer - usually as a result of a compatible operating system, such as the Digital Research CP/M.

Primer

A set of elementary instructions and guidelines to introduce the user to specific aspects of computing.

Printer

Any hardcopy method for printing out text.

Program

Spelt 'program' not 'programme' in computing. The combination of instructions that cause the computer to execute a task, and can be anything from a simple machine code 'routine' to a complete applications program, such as a wordprocessor.

Programming language

The medium through which the program is written, being comprised of rigid rules on the use of words, numbers and the sequence in which they are implemented.

PROM

Programmable Read Only Memory. An integrated memory circuit that once written with data, cannot be erased. (Also see EPROM)

Prompt

A symbol or message displayed on the screen that invites the user to respond with an answer or further 'input'. Basic uses a simple question mark ? when requiring input, or the word `Ready` when waiting for a command to be entered.

PSU

Power Supply Unit. The means of converting the domestic mains electricity supply into the necessary voltages to operate the computer (and peripheral devices).

QWERTY keyboard

The colloquial term to describe a keyboard with the conventional typewriter key layout.

RAM

Random Access Memory. Memory that may be both read from, and written to, using the internal circuitry of the computer during the normal course of program execution.

Random number

A number that is generated by the computer program that is neither repeatable, nor predictable.

Raster

A system of 'writing' on the screen where the images are built from a number of horizontal scan lines. (Raster scan).

Real number

A number that has both integer and fractional parts. ie. both sides of the decimal point are used. A variable may still be considered real, although when examined at some point in the program execution, it is only occupying an integer position..

Real time

Events that occur before your eyes, as opposed to those which only become evident after the termination of the process that produced them.

Recursion

The series of repeated steps (also sometimes imprecisely described as reciprocation) within a program or routine in which the result of each repeated cycle of events is related to the previous one.

Refresh

To update information, either on the screen of a VDU, or in the memory. Need not be a destructive process, but merely reinforcing whatever was already present in memory or on the screen.

Register

A transient memory location within the CPU that is used for temporary storage.

Remark

A non-executing comment statement in a program that is installed to REMind the programmer what part of the program is doing, and possibly also to date and time stamp that particular 'edition'.

Reserved word

A word which has particular significance to the computer program, and cannot be used other than in its previously defined context. For example, BASIC will not accept the word `NEW` as a variable -it is already 'reserved' for another purpose.

Resolution

The ability to determine where one element of the display ends, and the next one begins. Also loosely applied to the ability of a computer to perform arithmetical manipulations on large numbers.

Reverse Polish notation

(RPN) A method of describing arithmetical operations favoured by some calculator manufacturers, where the operators (+, -, ×, /) are placed behind the values to which they apply.

RF Modulator

The means by which the video signals from the computer are encoded and 'transmitted' to the aeriel input of a standard TV set.

ROM

Read Only Memory. Generally with reference to semiconductor memory systems: once written, neither erasable, nor re-writable.

Routine

A part of the program that performs a 'routine' task. A 'sub' program that resides either within a main program, or may exist as a separate module for incorporation into a variety of applications programs. eg A program to derive a 12 hour display from the system's clock may be considered as a routine.

RS232C

A specific standard for serial data communications interfaces. Devices at both ends of the data link using an RS232 interface require to be 'configured' to the particular conditions of the RS232 data standard used. Compare this with the Centronix parallel interface where the interconnection is a rigid standard.

Screen Editor

A text or program editor where the cursor may be taken to any part of the screen display in order to alter the characters appearing there.

Scrolling

The term describing the way in which the screen display 'rolls up' when the display fills up to the bottom, and needs to make space for the next line of entry or output to appear.

Separator

A separator performs the same function as a delimiter: ie. marking the boundary between reserved words and other elements of the program or data..

Serial interface

Although this term nearly always refers to an RS232 interface, other serial standards exist for the sequential transmission of computer data.

Simulation

A technique for emulation of real life interactive processes using the computer, such as flight simulation, driving simulation etc.

Soft key

See UDK (user defined key).

Software

Programs.

Software engineering

A grandiose expression meaning computer programming, implying a structured and considered approach, as opposed to arbitrary techniques.

Sound generator

The part of a computer (it may be either hardware of software) that creates the sound and noise.

Speech recognition

The conversion of the spoken word into machine readable instructions.

Speech synthesis

Generation of simulated speech using a combination of hardware and software.

Spreadsheet

A program that allows rows and columns of numbers to be entered and arithmetically manipulated. Changing one entry causes all the associated calculations to be rerun, and produces an updated result.

Sprite

A screen character that moves freely around the display, generated by specific hardware or software that allows it to appear and disappear apparently at random.

Stack

An area of memory allocated for 'stacking' information, but where only the last entry on the stack can be recalled.

Statement

An instruction, or sequence of instructions, in a computer program.

Stream

The route used for the output from the computer. eg the screen, the printer, or the cassette.

String

A type of data comprising an assortment of characters that may not be treated as a numeric variable. It may be purely numeric, but it is not treated at such unless specifically converted to a corresponding numeric variable by the appropiate command.

Structured programming

A logical and premeditated programming technique that results in programs that flow from 'top to bottom', with clearly described steps.

Subroutine

See routine.

Syntax error

When the rules of the program are broken by the incorrect use of keywords and variables, BASIC will prompt the user with this message. See Appendix VIII.

Terminal

A keyboard input device, with either a VDU screen or teletype typewriter output system.

Truncated

A number or string that has been shortened by removal of leading or trailing characters. Where intentional the process may involve rounding the value. Where unintentional, the extra characters are simply discarded to enable the number or string to fit the available space.

Truth table

The results of a logical operation are either 'true' or 'false'. The computer interprets these as being either 1 or 0, and the truth table lists the possible results of a logical operation (IF A› B THEN C) accordingly.

UDK

User defined keys. The CPC464 has up to 32 keys which may be redefined to perform a variety of tasks, including strings of up to 32 characters.

Unsigned number

A number with no prefix to signify whether its value is positive or negative.

Utility

Any complete program used to perform a common operation, such as sorting data or copying files.

Variable

An item included in a computer program that can be identified by name, but whose actual value may be made to vary during the execution of a program.

Appendix II

An introducion to the background of computing

Whose afraid of the jargon ?

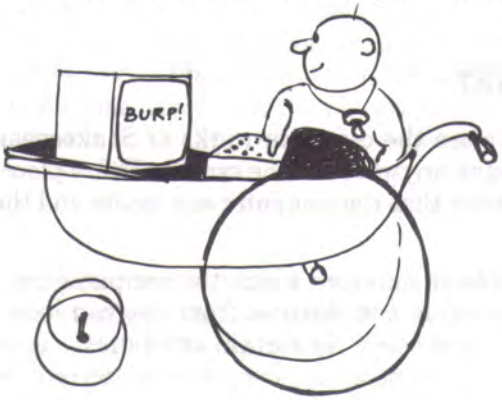
As with all 'specialist' industries, computing has developed its own jargon as a short-hand form of communicating complicated concepts that require many words of 'plain language' explanation. It's not just the high technology business that's guilty of hiding itself behind an apparent smokescreen of 'buzz words', jargon and terminolgy - most of us have come up against the barriers to understanding, erected by all the main professions and trades.

A major difference is that the confusion in legal jargon arises from the way the words are used - rather than the words themselves, as is the case with computing. Most people who grow familiar with computing terminology will go out of their way to use the words in the most straightforward possible manner, so as to minimize the complexity of the communication.

Don't be mislead by the 'plain language' used in computing, it is not a literary subject, but a precise science, and apart from the 'syntax' of the wording, the structure of the communication is very straightforward, and not in the least confusing or ambiguous. Teachers of computing have not yet managed to make an art form out of trying to analyse the exact meaning intended by a programmer in his program construction.

Having said that, although whether or not the meaning of a computer program is obvious, there are still many aspects that can be analysed as either elegant or untidy, and more emphasis is being put on a formal approach to program construction now that the initial mayhem brought about by the micro revolution is settling down.

Computing is rapidly understood by many young people who appreciate the precision and simplicity of the ideas and the way they can be communicated - you don't find too many 10 year old lawyers - but you can find plenty of ten year old programmers !



Basics of BASIC

Virtually all home computers provide a language known as BASIC, which allows programs to be written in the nearest thing to plain language presently available. BASIC is an acronym of 'Beginners All-purpose Symbolic Instruction Code - it no longer has any particular significance as to the degree of the sophistication of the languages, and many extremely complex and powerful programs are written using BASIC.

However, there's no doubt that the name has attracted many newcomers for its promise of providing a starting place in the maze of computer program languages, and this has contributed significantly to its universality.

From here onwards, the commonly used words that form the glossary of terms of computing will be introduced by first printing them in italics. Don't worry about trying to learn them from the following sections - there's an index to them in the glossary.

Basics

BASIC is a computer language that interprets a range of permitted commands, and then performs operations on data while the program runs. Unlike the average human vocabulary of 5-8 thousand words (plus all the different ways verbs can be used etc.), BASIC has to get by with about two hundred. Computer programs written using BASIC have to follow rigid rules concerning the use of these words. The *syntax* is precise, and any attempt to communicate with the computer using literal or colloquial expressions (plain language) will result in the cold and clinical message:

```
Syntax error
```

This is not as restrictive as it first appears, since the language of BASIC (the Syntax) is primarily designed to manipulate numbers - the numeric data. The words are essentially an extension of the familiar mathematical operators + / - etc. - and the most import concept for newcomers to grasp is the fact that a computer can only work with numeric data - information that is supplied to the *Central Processor integrated circuit* is only supplied in the form of numerical data.

NUMBER PLEASE!

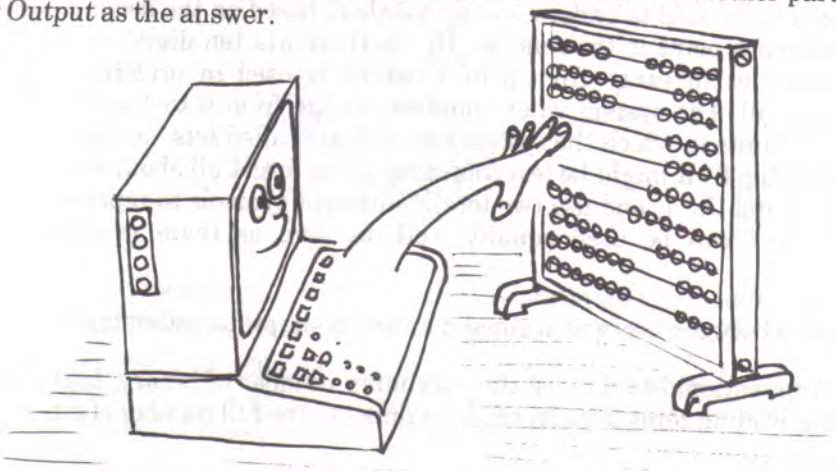
If a computer is used to store the complete works of Shakespeare, there is not a single letter or word to found anywhere in the system. Every piece of information is first converted into a number that the computer can locate and then manipulate as required.

BASIC interprets the words as numbers which the computer can then manipulate using only addition, subtraction and features from *Boolean logic* that permits the computer to compare data and select for certain attributes. - ie check to see if one number is bigger then or the same as another, or to perform a defined task if one number OR another meets certain criteria.

Through the medium of the program, the computer breaks down every task into a simple series of Yes/No operations.

The process of multiplication is performed using multiple additions - the BASIC instruction to multiply 35 by 10 (35*10) gets to the answer by adding 35 to itself ten times.

Part of the *Central Processor Unit (CPU)* is loaded with the numeric data for 10, and another part of the *CPU* is loaded with 35. Each time 35 is added to itself, the part of *memory* containing 10 is decreased by one until it reaches zero, when the process stops, and the accumulated result of 350 is sent to another part of the *CPU* for *Output* as the answer.



If this process sounds cumbersome, then you're quite right, as you have uncovered the first and most important truth about computing. A computer is primarily a tool for performing the simplest of repetetive tasks very quickly and with absolute precision. Thus BASIC interprets the instructions given in the form of the *program*, and translates them into the language that can be handled by the *CPU*. Only two states are understood by the logic of a computer - 'yes' or 'no', represented in *binary* notation as '1' and '0'. The representation in *Boolean logic* is simply 'true' and 'false' - there's no such thing as a 'maybe' or 'perhaps'.

The process of switching between these two distinct states is the essence of the term *digital*, and is sometimes referred to as toggling. In the world of nature, most processes move gradually from one completely 'stable' state to another in a smooth progression. In other words, the transition is made by following the path of a line between the two states - in an ideal digital enviroment the switch from one state to the next is made in no time at all - but the physics of semiconductor science dictate that there will be some minor delay, referred to as propagation delays - and it is the accumulation of many of these propagation delays that provides the reason why a computer has to spend some time processing the information before the answer comes out.

In any case, the computer would have to wait a finite time for one task to have finished before it can start work on the result of that first task - so there would need to be some artificial delay imposed anyway. The digital process is black or white where the stages in the transition via shades of grey have no significance whatsoever. The smooth progression is via various shades of grey.

If the ultimate answer is either 0 or 1, then there is no possibility of it being 'nearly' correct. The fact that computers can sometimes appear to make errors when handling numeric data is due to the limitation of the size of numbers it can process requiring 'oversize' data to be squeezed down to fit the space available, or *truncated*, leading to rounding errors. eg 999,999,999 becomes 1,000,000,000.

In a world where the only two numbers available are 0 or 1, how do you then count beyond 1 ?

Bits and Bytes

We just happen to be used to understanding numbers based on the *decimal* system where the reference point is the number 10 - ie there are ten digits available to represent quantities in range from 0 to 9 (which is used in preference to the expression 1 to 10). The system where numbers range from 0 to 1 is the *binary* system, and the units in which the system operates are called *bits* - an abbreviated form of '**BI**nary digi**T**'. It might be less confusing if you forget all about the decimal context of the 0 and the 1, and use two totally different symbols to represent their function: fl and * can be used equally well, as long as there is consistency throughout.

The relationship between *bits* and decimal notation is simple to understand:

It's actually conventional to declare the maximum number of binary digits being used by adding leading zeros to make up the number to the full number of bits:

decimal 7 becomes
00111 binary
using 5 bit notation

In the binary system, the figures may be considered merely as indicators in columns to specify whether or not a given power 2 is present (1), or not (0).

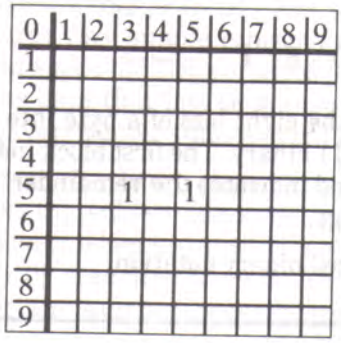
$2^0 = 1$
$2^1 = 2 = 2 = 2(2^0)$
$2^2 = 4 = 2x2 = 2(2^1)$
$2^3 = 8 = 2x2x2 = 2(2^2)$
$2^4 = 16 = 2x2x2x2 = 2(2^3)$

so the columns look like

2^4	2^3	2^2	2^1	2^0	
1	0	0	1	1	
(16	0	0	2	1)	= 19

In order to provide a shorthand method of referring to binary digit information, the term byte is used to denote 8 bits of information. The highest number that can be stored in a byte is then (binary) 11111111 - or (decimal) 255. This implies 256 actual variations, including 00000000, which is still perfectly valid data to a computer.

Computers tend to manipulate data in 8 bit multiples. 256 is not a very large number, so in order to acheieve an acceptable means of handling the memory, two bytes are used to provide a method of addressing memory which is in the form of an *array*, with a horizontal and vertical *address* by which the elements of that *array* can be located:



This array can locate up to (10x10) items of information using address numbers that lie in the range 1 to 9. The item stored at position 3,5 is a '1' - as is the item at 5,5.

So a binary array of 256x256 can handle 65,536 individual locations using 8 bit addresses for the vertical and horizontal axes of the array. So our '0' and '1' have progressed to being capable of identifying one of 65,536 unique elements.

The next level of shorthand for binary is the *kilo*byte (kByte or simply 'K') which is 1024 bytes. 1024 is the nearest binary multiple to the more familiar decimal use of the term 'kilo' - and explains why a computer described as having a '64K' memory does in fact have a memory of 65,536 bytes (64 x 1024).

Thankfully, the BASIC interpreter does all the necessary conversions for you, and it is quite possible to become a proficient programmer without a complete understanding of binary. Although an appreciation of the significance of binary will help you spot the many 'magic' or significant numbers that inevitably crop up as you work through the science of computing.

It's worth spending some effort to acquire an understanding of binary and the various significant numbers 255, 1024 etc., since it is very unlikely that these will change from the being the bedrock of computer operation in the forseeable future. The certainty and simplicity that comes from working in only two states will prevail over the enormously increased complexity that would result from any other number base.

However....

Simple and elegant as it is, binary notation is longwinded and prone to inaccuracy as it cannot be easily read at a glance. Binary has a number of associated counting systems that act as shorthand for programmers. One such number system is widely used in microcomputing called *Hex* (an abbreviation of *hexadecimal*).

Here the number is based on 16, and is represented in a single character:

Decimal

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Hex

0 1 2 3 4 5 6 7 8 9 A B C D E F

The hexadecimal system can break the eight bits of a byte into two blocks of four bits, since 15 is a *four bit* number: 1111 binary. The first block indicates the number of complete units of '15', and the second indicates the 'remainder' - and this is where the elegance of binary begins to emerge.

Reconsidering the table that introduced binary notation

Decimal	Binary	CPC464-ese	Hexadecimal
0	0	fl	0
1	1	*	1
2	10	*fl	2
3	11	**	3
4	100	*flfl	4
5	101	*fl*	5
6	110	**fl	6
7	111	***	7
8	1000	*flflfl	8
9	1001	*flfl*	9
10	1010	*fl*fl	A
11	1011	*fl**	B
12	1100	**flfl	C
13	1101	**fl*	D
14	1110	***fl	E
15	1111	****	F
16	10000	*flflflfl	10

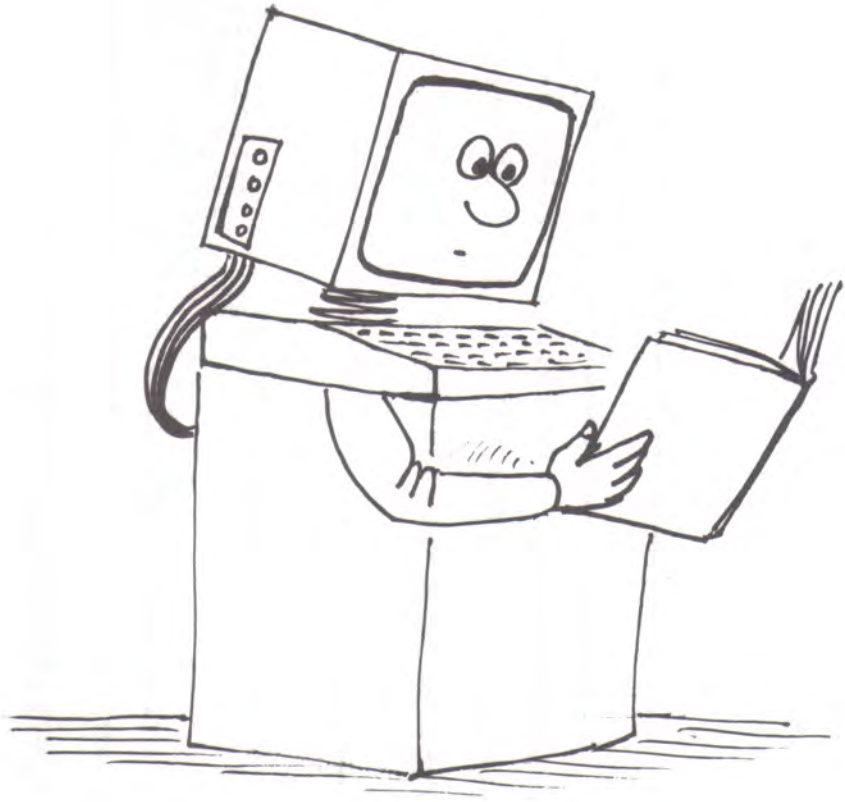
An 8-bit number 11010110 can be subdivided, and then considered as two 4-bit numbers known as *nibbles* , Hex D6. Throughout this guide a hex based number will be introduced by the '&' symbol eg &D6, and this is the number base most commonly used by programmers using assembly language techniques. An assembly language program is the nearest most programmers get to programming directly in *machine code*, since the assembly language program allows the program to use simple letter 'mnemonics' to specify the actual machine code 'numbers'.

When using HEX, you must first work out the value of the first digit to obtain the number of 16's in the final number, and then add the remainder designated by the second 'half' of the hex notation to obtain the total decimal equivalent. There's a powerful temptation to regard a number like &D6 as 13+6, or 136. But it's (13x16)+(6) = 214.

It's the same process you use when you read a decimal (also known as a Denary number) number such as '89' - ie (8x10)+(9). It just happens that multiplying by ten is a great deal simpler unless you've had a lot of practise at multiplying by 16.

If you've got this far without becoming too confused, then you are well on your way to getting a grasp of the basic principles of the computer. You may even be wondering what all the fuss is about - and you'd be quite correct. A computer is a device that manages very simple concepts and ideas: it just happens to perform these tasks at great speed (millions of times per second), and with a huge capacity to remember both the data that has been input, and the intermediate results of the many thousands of very simple sums along the way to the result.

If you want to pursue the theory of your computer, there are literally thousands of books available on the subject of computing. Some will tend to leave you more confused than you were when you started reading them, and a few will actually lead you along the way by revealing the simplicity and the fundamental relationships that exist between the number systems, and the way your computer deals with them.



Appendix III

The ASCII characters, and the graphics characters of the CPC464

III.1 ASCII

For convenient reference, we reproduce here the standard ASCII reference character set using decimal, octal and hex notation, together with the ASCII code where appropriate. Each of the CPC464 character cells is also represented in detail.

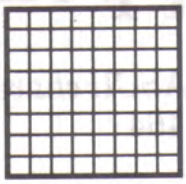
DEC	OCTAL	HEX	ASCII characters
0	000	00	NUL ([CTRL]@)
1	001	01	SOH ([CTRL]A)
2	002	02	STX ([CTRL]B)
3	003	03	ETX ([CTRL]C)
4	004	04	EOT ([CTRL]D)
5	005	05	ENQ ([CTRL]E)
6	006	06	ACK ([CTRL]F)
7	007	07	BEL ([CTRL]G)
8	010	08	BS ([CTRL]H)
9	011	09	HT ([CTRL]I)
10	012	OA	LF ([CTRL]J)
11	013	OB	VT ([CTRL]K)
12	014	OC	FF ([CTRL]L)
13	015	OD	CR ([CTRL]M)
14	016	OE	SO ([CTRL]N)
15	017	OF	SI ([CTRL]O)
16	020	10	DLE ([CTRL]P)
17	021	11	DC1 ([CTRL]Q)
18	022	12	DC2 ([CTRL]R)
19	023	13	DC3 ([CTRL]S)
20	024	14	DC4 ([CTRL]T)
21	025	15	NAK ([CTRL]U)
22	026	16	SYN ([CTRL]V)
23	027	17	ETB ([CTRL]W)
24	030	18	CAN ([CTRL]X)
25	031	19	EM ([CTRL]Y)
26	032	1A	SUB ([CTRL]Z)
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	SP
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/
48	060	30	Ø
49	061	31	1

DEC	OCTAL	HEX	ASCII
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑
95	137	5F	←
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c

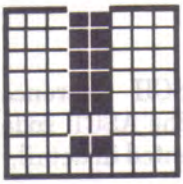
DEC	OCTAL	HEX	ASCII
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~

III. 2 CPC464 Machine-specific firmware character set

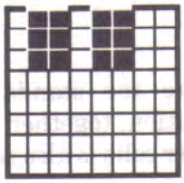
The characters reproduced here are plotted on the standard 8x8 cell matrix used to write the screen of the CPC464. User Defined Characters may be grouped for special effects and butted close against each other - see the SYMBOL command description given in Chapter 8.



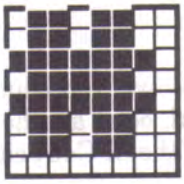
32 &H20
&X00100000



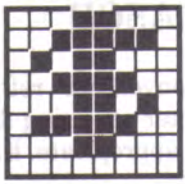
33
&H21
&X00100001



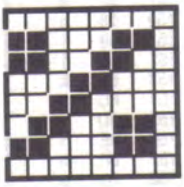
34
&H22
&X00100010



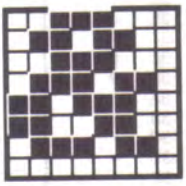
35
&H23
&X00100011



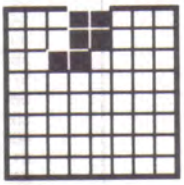
36
&H24
&X00100100



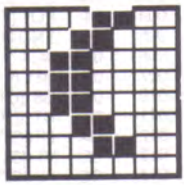
37
&H25
&X00100101



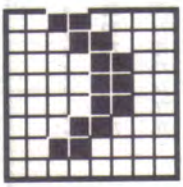
38
&H26
&X00100110



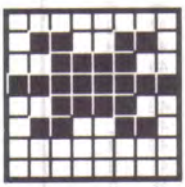
39
&H27
&X00100111



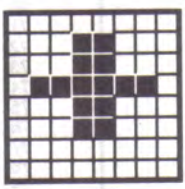
40
&H28
&X00101000



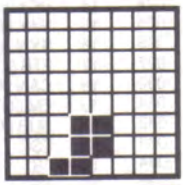
41
&H29
&X00101001



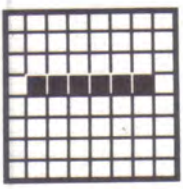
42
&H2A
&X00101010



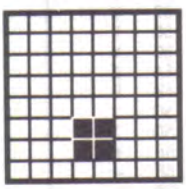
43
&H2B
&X00101011



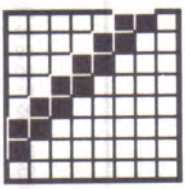
44
&H2C
&X00101100



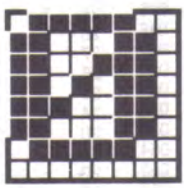
45
&H2D
&X00101101



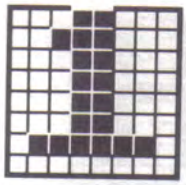
46
&H2E
&X00101110



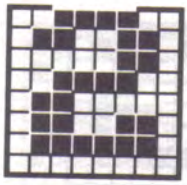
47
&H2F
&X00101111



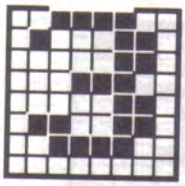
48
&H30
&X00110000



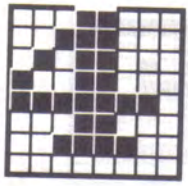
49
&H31
&X00110001



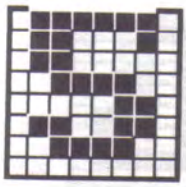
50
&H32
&X00110010



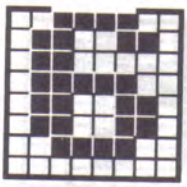
51
&H33
&X00110011



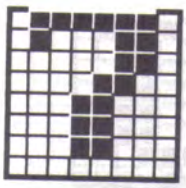
52
&H34
&X00110100



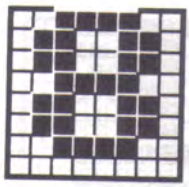
53
&H35
&X00110101



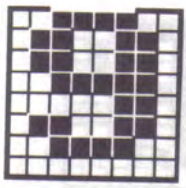
54
&H36
&X00110110



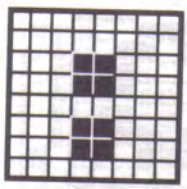
55
&H37
&X00110111



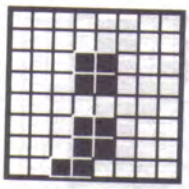
56
&H38
&X00111000



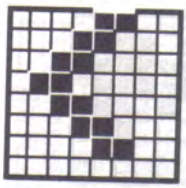
57
&H39
&X00111001



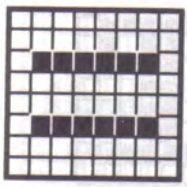
58
&H3A
&X00111010



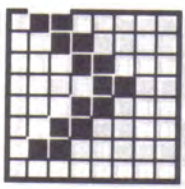
59
&H3B
&X00111011



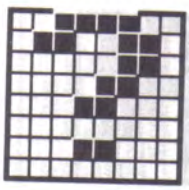
60
&H3C
&X00111100



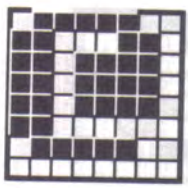
61
&H3D
&X00111101



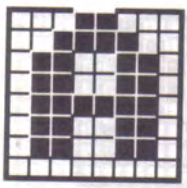
62
&H3E
&X00111110



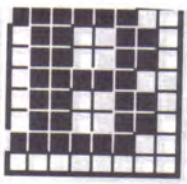
63
&H3F
&X00111111



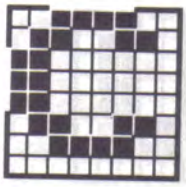
64
&H40
&X01000000



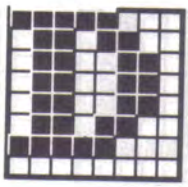
65
&H41
&X01000001



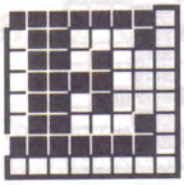
66
&H42
&X01000010



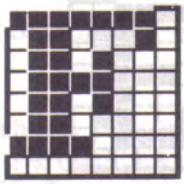
67
&H43
&X01000011



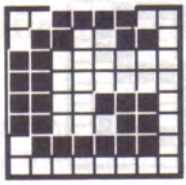
68
&H44
&X01000100



69
&H45
&X01000101

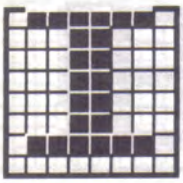


70
&H46
&X01000110

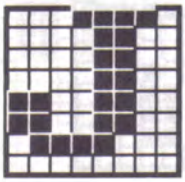


71
&H47
&X01000111

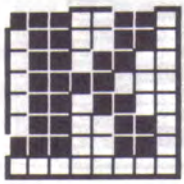
72
&H48
&X01001000



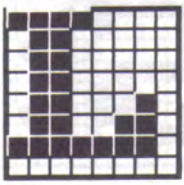
73
&H49
&X01001001



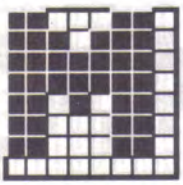
74
&H4A
&X01001010



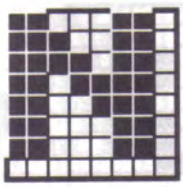
75
&H4B
&X01001011



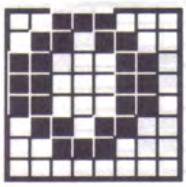
76
&H4C
&X01001100



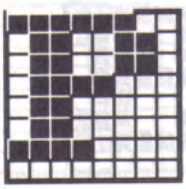
77
&H4D
&X01001101



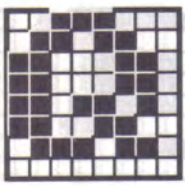
78
&H4E
&X01001110



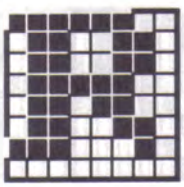
79
&H4F
&X01001111



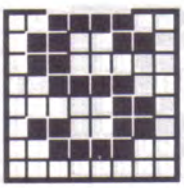
80
&H50
&X01010000



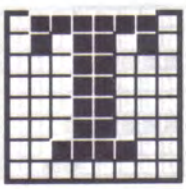
81
&H51
&X01010001



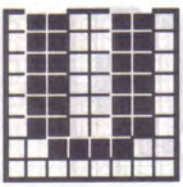
82
&H52
&X01010010



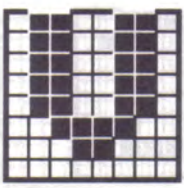
83
&H53
&X01010011



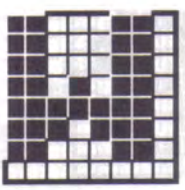
84
&H54
&X01010100



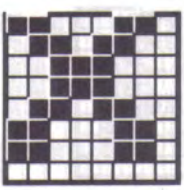
85
&H55
&X01010101



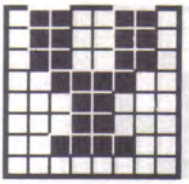
86
&H56
&X01010110



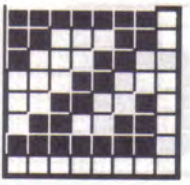
87
&H57
&X01010111



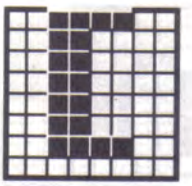
88
&H58
&X01011000



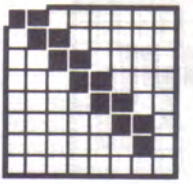
89
&H59
&X01011001



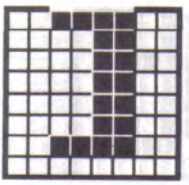
90
&H5A
&X01011010



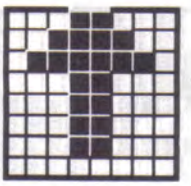
91
&H5B
&X01011011



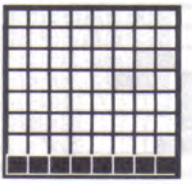
92
&H5C
&X01011100



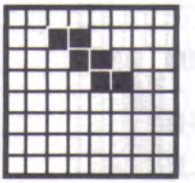
93
&H5D
&X01011101



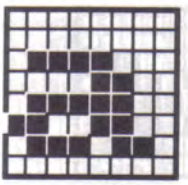
94
&H5E
&X01011110



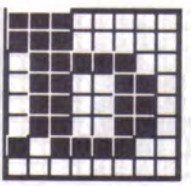
95
&H5F
&X01011111



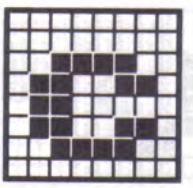
96
&H60
&X01100000



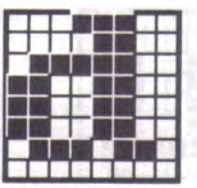
97
&H61
&X01100001



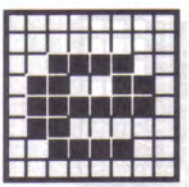
98
&H62
&X01100010



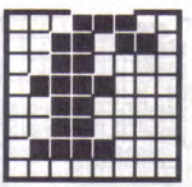
99
&H63
&X01100011



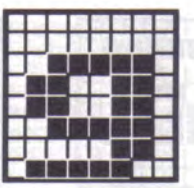
100
&H64
&X01100100



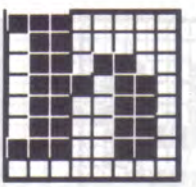
101
&H65
&X01100101



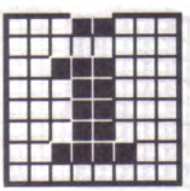
102
&H66
&X01100110



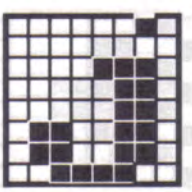
103
&H67
&X01100111



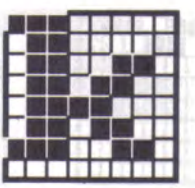
104
&H68
&X01101000



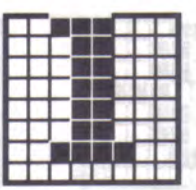
105
&H69
&X01101001



106
&H6A
&X01101010

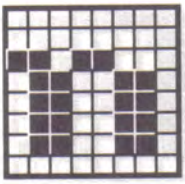


107
&H6B
&X01101011

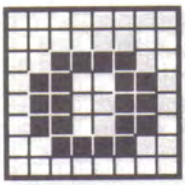


108
&H6C
&X01101100

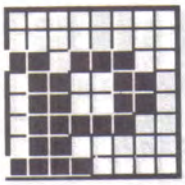
109
&H6D
&X01101101



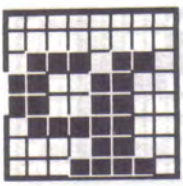
110
&H6E
&X01101110



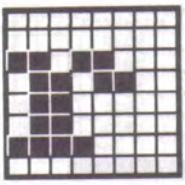
111
&H6F
&X01101111



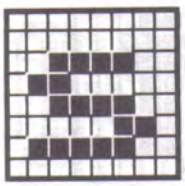
112
&H70
&X01110000



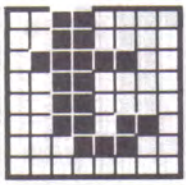
113
&H71
&X01110001



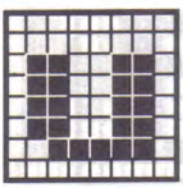
114
&H72
&X01110010



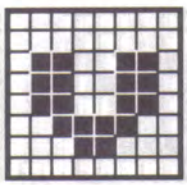
115
&H73
&X01110011



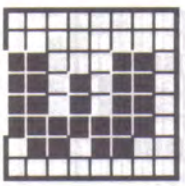
116
&H74
&X01110100



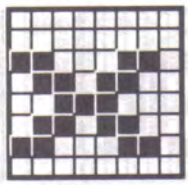
117
&H75
&X01110101



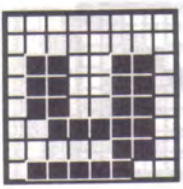
118
&H76
&X01110110



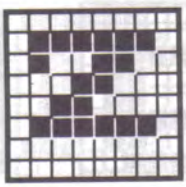
119
&H77
&X01110111



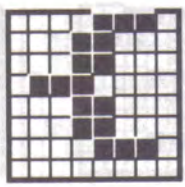
120
&H78
&X01111000



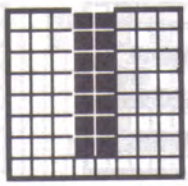
121
&H79
&X01111001



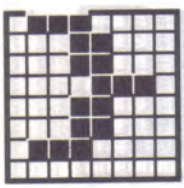
122
&H7A
&X01111010



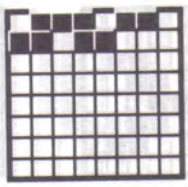
123
&H7B
&X01111011



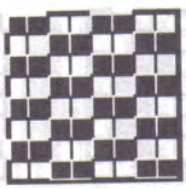
124
&H7C
&X01111100



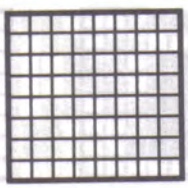
125
&H7D
&X01111101



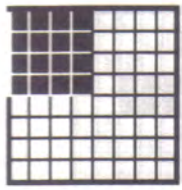
126
&H7E
&X01111110



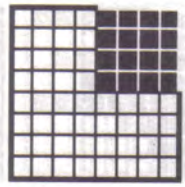
127
&H7F
&X01111111



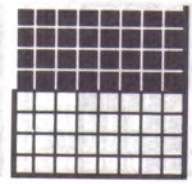
128
&H80
&X10000000



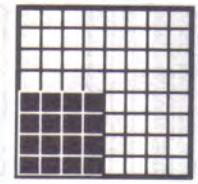
129
&H81
&X10000001



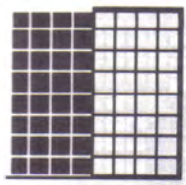
130
&H82
&X10000010



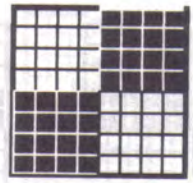
131
&H83
&X10000011



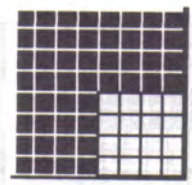
132
&H84
&X10000100



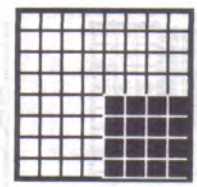
133
&H85
&X10000101



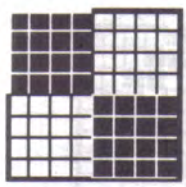
134
&H86
&X10000110



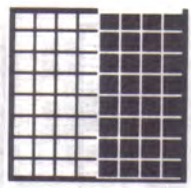
135
&H87
&X10000111



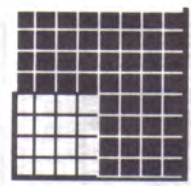
136
&H88
&X10001000



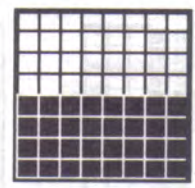
137
&H89
&X10001001



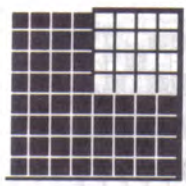
138
&H8A
&X10001010



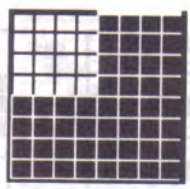
139
&H8B
&X10001011



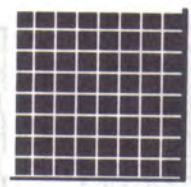
140
&H8C
&X10001100



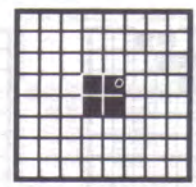
141
&H8D
&X10001101



142
&H8E
&X10001110

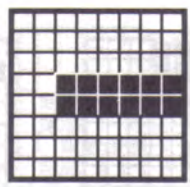


143
&H8F
&X10001111

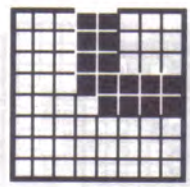


144
&H90
&X10010000

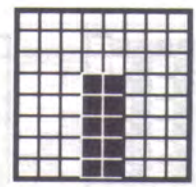
145
&H91
&X10010001



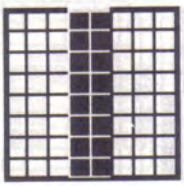
146
&H92
&X10010010



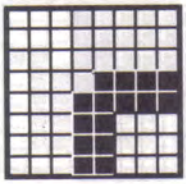
147
&H93
&X10010011



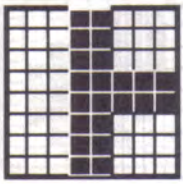
148
&H94
&X10010100



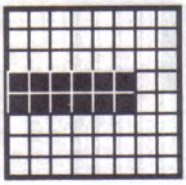
149
&H95
&X10010101



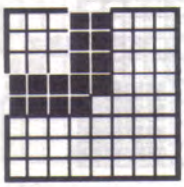
150
&H96
&X10010110



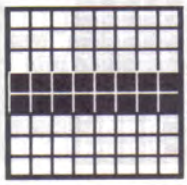
151
&H97
&X10010111



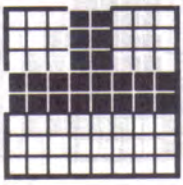
152
&H98
&X10011000



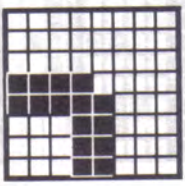
153
&H99
&X10011001



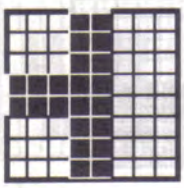
154
&H9A
&X10011010



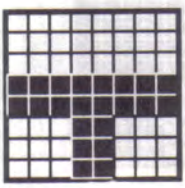
155
&H9B
&X10011011



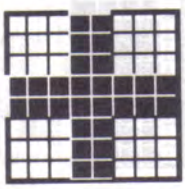
156
&H9C
&X10011100



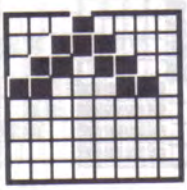
157
&H9D
&X10011101



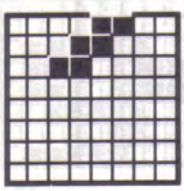
158
&H9E
&X10011110



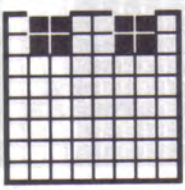
159
&H9F
&X10011111



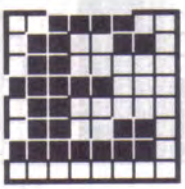
160
&HA0
&X10100000



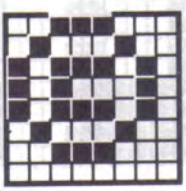
161
&HA1
&X10100001



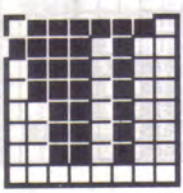
162
&HA2
&X10100010



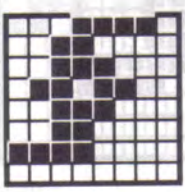
163
&HA3
&X10100011



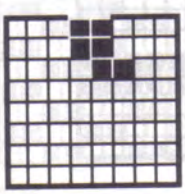
164
&HA4
&X10100100



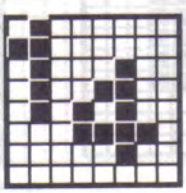
165
&HA5
&X10100101



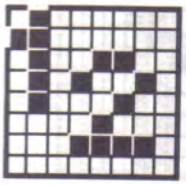
166
&HA6
&X10100110



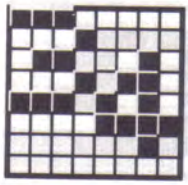
167
&HA7
&X10100111



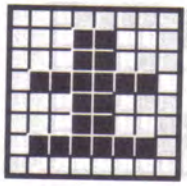
168
&HA8
&X10101000



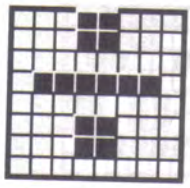
169
&HA9
&X10101001



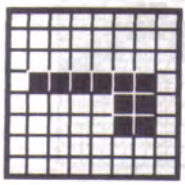
170
&HAA
&X10101010



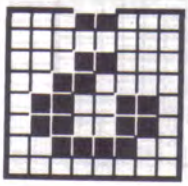
171
&HAB
&X10101011



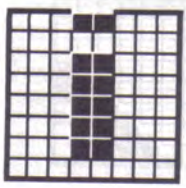
172
&HAC
&X10101100



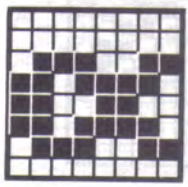
173
&HAD
&X10101101



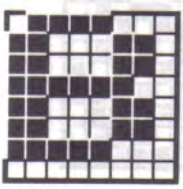
174
&HAE
&X10101110



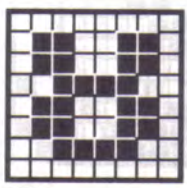
175
&HAF
&X10101111



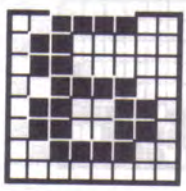
176
&HB0
&X10110000



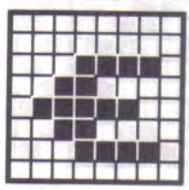
177
&HB1
&X10110001



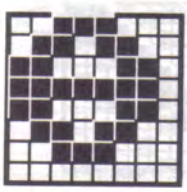
178
&HB2
&X10110010



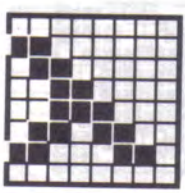
179
&HB3
&X10110011



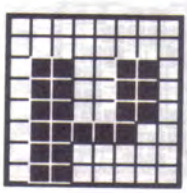
180
&HB4
&X10110100



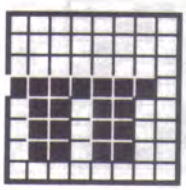
181
&HB5
&X10110101



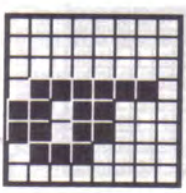
182
&HB6
&X10110110



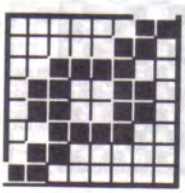
183
&HB7
&X10110111



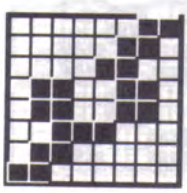
184
&HB8
&X10111000



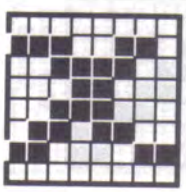
185
&HB9
&X10111001



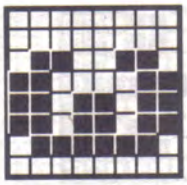
186
&HBA
&X10111010



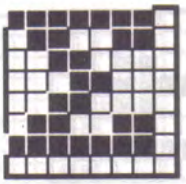
187
&HBB
&X10111011



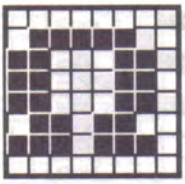
188
&HBC
&X10111100



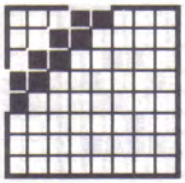
189
&HBD
&X10111101



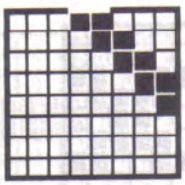
190
&HBE
&X10111110



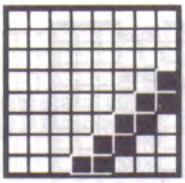
191
&HBF
&X10111111



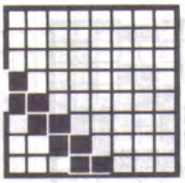
192
&HC0
&X11000000



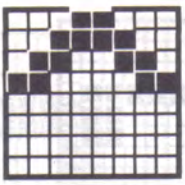
193
&HC1
&X11000001



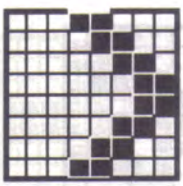
194
&HC2
&X11000010



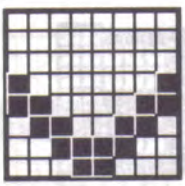
195
&HC3
&X11000011



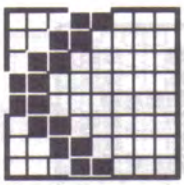
196
&HC4
&X11000100



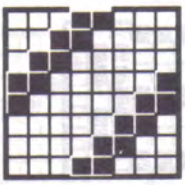
197
&HC5
&X11000101



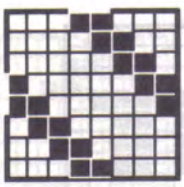
198
&HC6
&X11000110



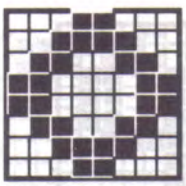
199
&HC7
&X11000111



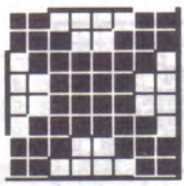
200
&HC8
&X11001000



201
&HC9
&X11001001

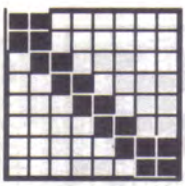


202
&HCA
&X11001010

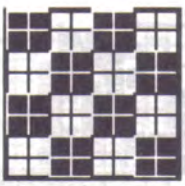


203
&HCB
&X11001011

204
&HCC
&X11001100

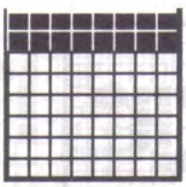


205
&HCD
&X11001101

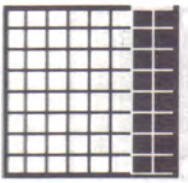


206
&HCE
&X11001110

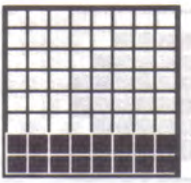
207
&HCF
&X11001111



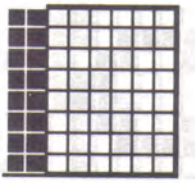
208
&HD0
&X11010000



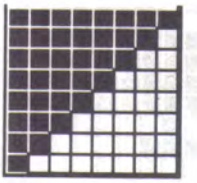
209
&HD1
&X11010001



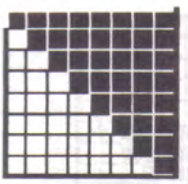
210
&HD2
&X11010010



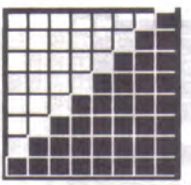
211
&HD3
&X11010011



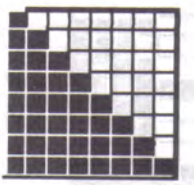
212
&HD4
&X11010100



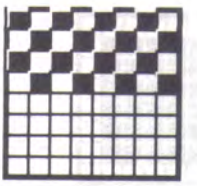
213
&HD5
&X11010101



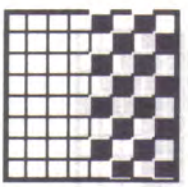
214
&HD6
&X11010110



215
&HD7
&X11010111

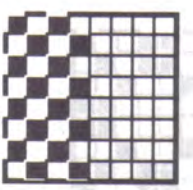


216
&HD8
&X11011000

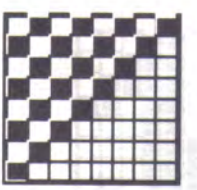


217
&HD9
&X11011001

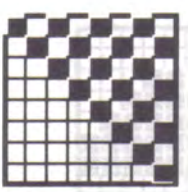
218
&HDA
&X11011010



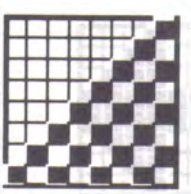
219
&HDB
&X11011011



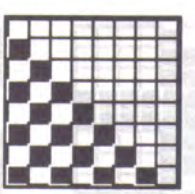
220
&HDC
&X11011100



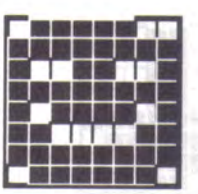
221
&HDD
&X11011101



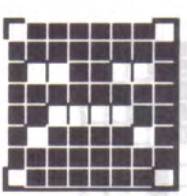
222
&HDE
&X11011110



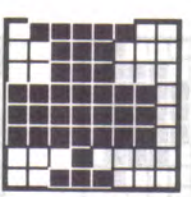
223
&HDF
&X11011111



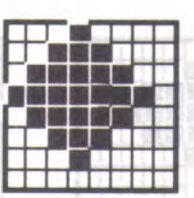
224
&HE0
&X11100000



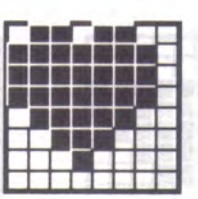
225
&HE1
&X11100001



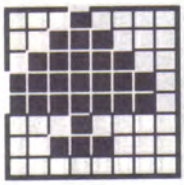
226
&HE2
&X11100010



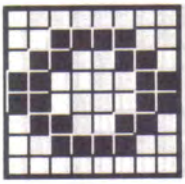
227
&HE3
&X11100011



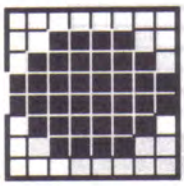
228
&HE4
&X11100100



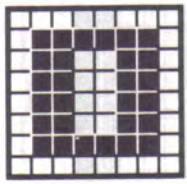
229
&HE5
&X11100101



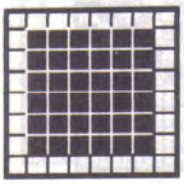
230
&HE6
&X11100110



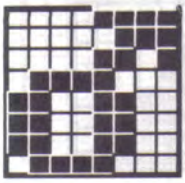
231
&HE7
&X11100111



232
&HE8
&X11101000

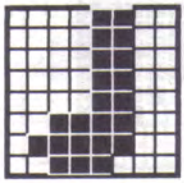


233
&HE9
&X11101001

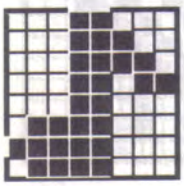


234
&HEA
&X11101010

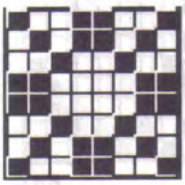
235
&HEB
&X11101011



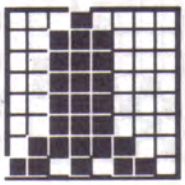
236
&HEC
&X11101100



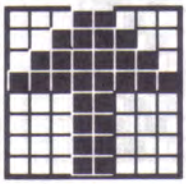
237
&HED
&X11101101



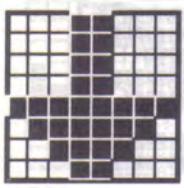
238
&HEE
&X11101110



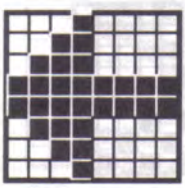
239
&HEF
&X11101111



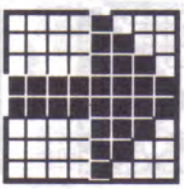
240
&HF0
&X11110000



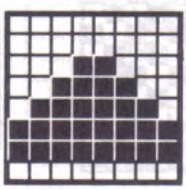
241
&HF1
&X11110001



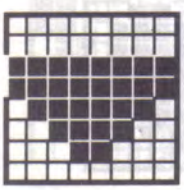
242
&HF2
&X11110010



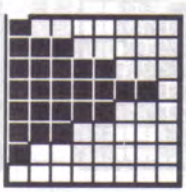
243
&HF3
&X11110011



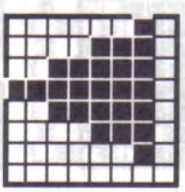
244
&HF4
&X11110100



245
&HF5
&X11110101



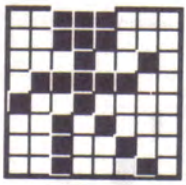
246
&HF6
&X11110110



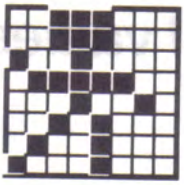
247
&HF7
&X11110111

248
&HF8
&X11111000

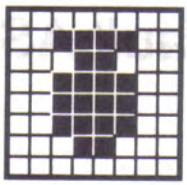
249
&HF9
&X11111001



250
&HFA
&X11111010

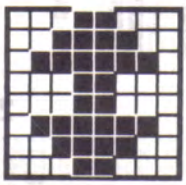


251
&HFB
&X11111011

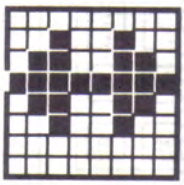


252
&HFC
&X11111100

253
&HFD
&X11111101

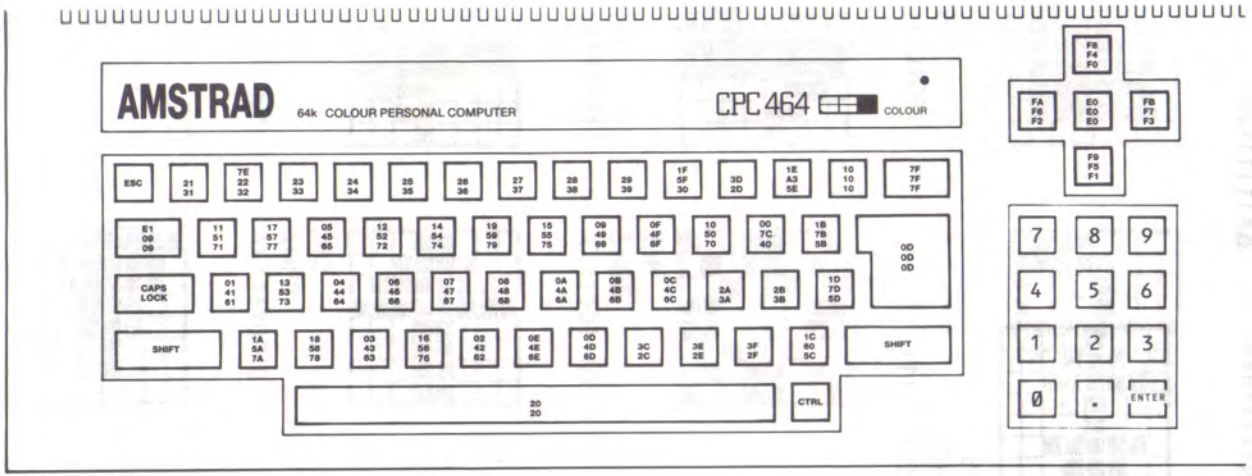


254
&HFE
&X11111110



255
&HFF
&X11111111

Default ASCII values



JOYSTICK 0

0B
0B

FIRE 2 FIRE 1

08 58 5A 09
08 58 5A 09

0A
0A

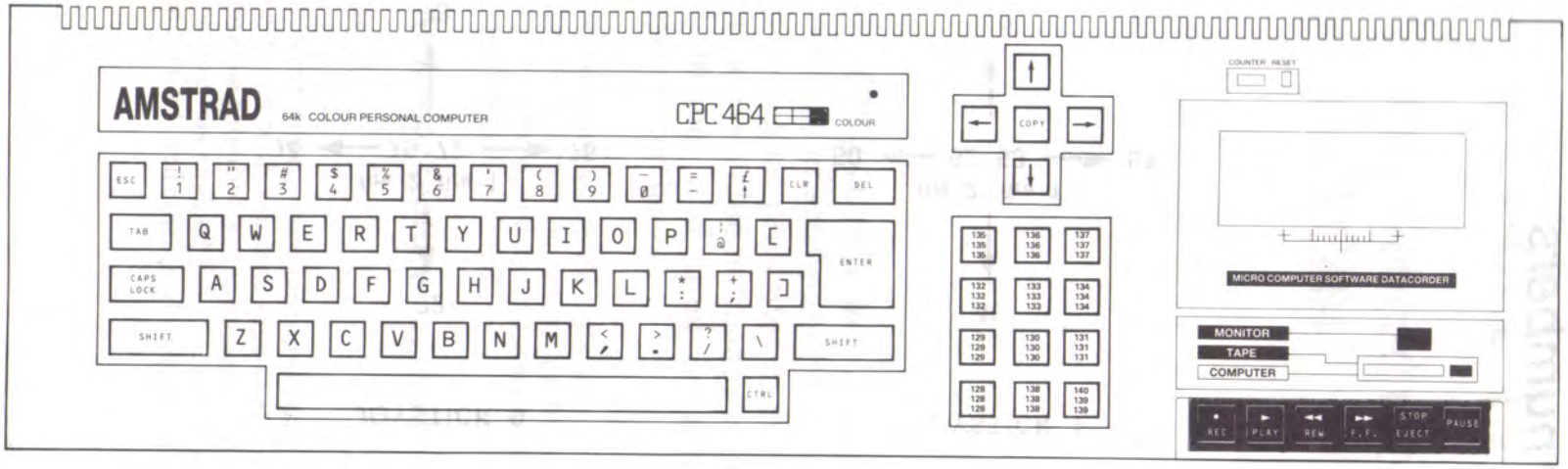
JOYSTICK 1

26
36

FIRE 2 FIRE 1

12 07 06 14
52 47 46 54
72 67 66 74

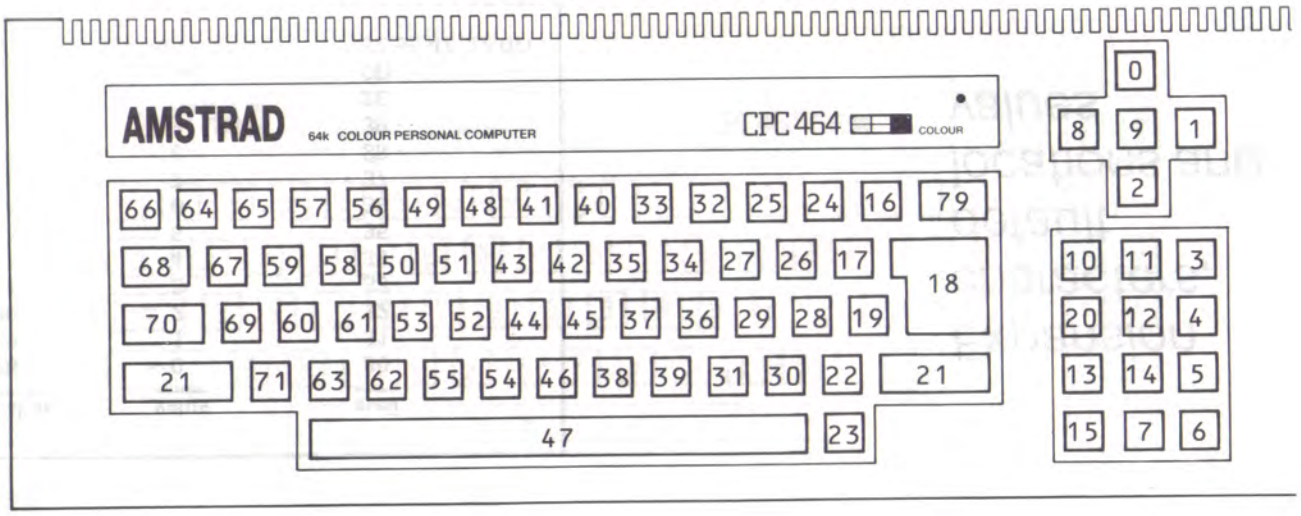
25
35



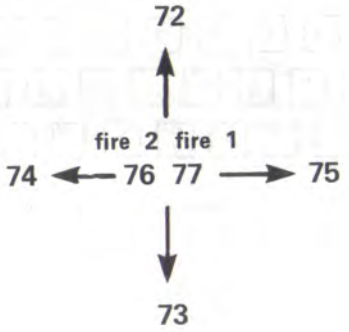
exp char	value	ascii
128	0	30
129	1	31
130	2	32
131	3	33
132	4	34
133	5	35
134	6	36
135	7	37
136	8	38
137	9	39
138	.	2E
139	[enter]	0D
140	run"	52,55,4E,22,0D

Expansion characters, default locations and values

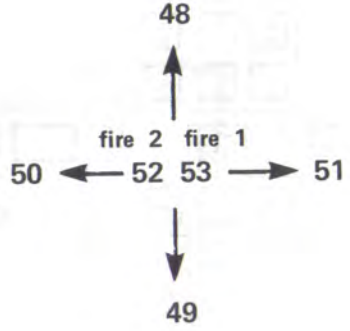
Key and joystick key numbers



JOYSTICK 0



JOYSTICK 1



Appendix IV:

An experienced users introduction to the CPC464

The CPC464 is a low cost colour personal computer with a generous, effective and thorough implementation of established technologies, presented in format designed to offer exceptionally attractive value for money coupled with substantial capacity for expansion, and thus be attractive to both first time and experienced users.

The hardware and ROM software have been designed to provide newcomers and experienced programmers with a friendly enviroment in which existing software may be tailored, and new software written to take advantage of the many features supported by the CPC464.

The major features of the system are :

Z80CPU

The most widely used microprocessor found in home computers throughout the world, and with the best supported software base - especially since the CPC464 offers the potential for the implementation of CP/M. The unique interrupt handling structure has enabled the CPC464 to innovate with the BASIC features AFTER and EVERY, and other 'real time' facilities controlling sound and timers.

64K RAM

A generous amount of RAM is supplied as standard, over 42K of which is actually available to the user, thanks to the implementation of ROM overlay techniques when implementing the BASIC.

Screen

The CPC464 supports three basic screen handling modes, including 80 column text, a palette of 27 colours, and resolution up to 640x200 pixels.

Real Keyboard

A full feature 'typewriter-style' keyboard, with a logical cursor key cluster and standard numeric entry keypad which doubles use for function key purposes.

Built-in cassette

Local cassette storage is available as a built-in feature, thus providing operation with no additional user complications with hookup, level setting etc. Writing at either 1kBaud or 2kBaud (software selectable), with read speeds automatically established by software.

BASIC

A 'written in England' industry standard BASIC, faster and more versatile than you have come to expect from such BASICs, with many extensions for graphics and sound, plus extensive support in firmware.

Extended character set

A full 8-bit character set including symbols and graphics is accessible largely via the keyboard, and using `CHR$(n)` functions.

Elapsed Time

Interrupts are generated by frame scans providing for elapsed time facilities.

User Defined Keys

Up to 32 Keys may be user defined with up to 32 character strings. Redefinition capability includes the repeat parameters. A complete 255 character set (all ASCII plus over 100 more), optionally user re-definable.

Subroutines

Many assembler sub-routines are available to be called from BASIC.

SCREEN MODES

There are three modes of screen operation:

a) Normal

Mode 1: 40 columns x 25 lines, 4 'ink' text
320x200 pixels, individually addressable in 4 colours

b) Multicolour mode

20 columns x 25 lines, 16 'ink' text
160x200 pixels, individually addressable in 16 colours

c) High Resolution mode

80 columns x 25 lines, 2 ink text
640x200 pixels, individually addressable in 2 colours

Colour Selection

(NB Throughout this guide, 'black' ie nil luminance is considered as a colour for the purposes of the following descriptions)

The border can be set to a ANY pair of colours regardles of the screen mode: ie flashing; or a single colour ie steady.

The number of available `INK`s depends on the screen modes selected. Each `INK` can be set to a pair of colours ie flashing; or a single colour ie steady. The number of usable inks at any time depends on the screen mode as previously defined. The text `PAPER`, text `PEN`, and graphics `PEN` can then be set to an available ink.

The text writing can be set to be translucent or opaque: ie it will either ignore the paper colour and overwrite the graphics, or completely overwrite the background.

Windows

The user can select up to eight text windows into which characters are written, and also a graphics window into which plotting may be performed.

Windows are reset to defaults when the screen mode is set.

NB: If the text window is equivalent to the entire screen (default), then rapid rolling is achieved by hardware. If the text window is less than the available screen, then rolling is achieved by software, which is correspondingly slower.

Cursor

The cursor is disabled during periods when the CPU is not requiring keyboard input, thus acting as an automatic prompt. The cursor is represented by an inverse square of colour.

Polyphonic Sound

The sound facilities of the CPC464 are generated using the industry standard sound generator device from General Instrument's AY8910 family. The device operates with 3 channels (voices), each of which can be independently set for tone and amplitude. White noise may be added as required.

The three channels appear as left, right and centre (using the stereo extension jack). The internal loudspeaker produces a mixed monaural output.

The software provides envelope control facilities for amplitude and tone. The sound generator device's internal amplitude enveloping is normally not used.

Printer Port

An industry standard Centronics compatible parallel printer port is provided, using the 'Busy' signal line for performing handshake operations.

Expansion Support

Many hardware interfaces are available via jump blocks or indirection to provide software expansion facilities. AMSTRAD will be introducing, amongst other items, disc drives, serial interfaces with driver software in ROM etc.

Coordinates

The text origin is the top left hand corner of the screen, and physical positions on the screen change with the screen mode.

The graphics origin is the bottom left hand corner, and assumes the screen is is High Resolution mode at all times -although calculations for inks will be performed correctly in all screen modes.

NB

In normal screen mode, each pixel has *TWO* horizontal addresses, and either may be used. In multicolour mode, each pixel has *FOUR* horizontal addresses, and any of the four can be used to define the pixel position.

The vertical axis has co-ordinates from 0....399 which are divided by two and truncated to give a physical position in the range 0....199. This ensures that expected aspect ratios are preserved on screen.

Expansion ROMS

All ROMs occupy the top 16K of memory (where the the BASIC lives) and there are facilities in the firmware to call up to 240 additional 16K add-on ROMs (a certain amount of address decoding hardware must be employed externally to the base machine as part of the ROM expansion hardware.)

Overview:

A brief summary of the main features of the hardware and firmware of the CPC464.

1) Hardware

1.1) Inside the main CPC464 case.

Computer, keyboard, cassette datacorder and loudspeaker. RGB and luminance outputs.

1.1.1) LSI Chips

Z80A processor running at 4MHz.

64K bytes of 64K x 1 dynamic RAM refreshed by accesses to the screen memory.

32K bytes of ROM containing BASIC and the operating system (OS).

A custom logic array incorporates nearly all logic not already inside LSI chips; particularly timing, colour generation and DMA circuitry.

The 6845 CRT controller device generates scanning signals for the video RAM.

NB

The mapping is intricate and changes with the screen mode. The CRTC can be used to perform scrolling (sideways by 1/40 screen width) and rolling (up and down by 8 scan lines) if the software requires. Parameters within the CRTC select the number of lines, the frame rate, width and positioning of the borders.

GI sound generator chip AY-3-8912: 3 voices. Sound is taken from the three channels and mixed equally to produce a mono output on the integral loudspeaker, controlled by a volume control. There is also an external stereo output where .....

Left = Channel A + 1/2 Channel C. Right = Channel B + 1/2 Channel C.
This chip also receives keyboard and joystick port scan information.

An 8255 parallel I/O device interfaces the bus to the GI sound chip. It also scans the keyboard, joystick port, option links and controls the cassette.

1.1.2) External sockets.

PCB edge connectors for general purpose expansion (12) and an external printer (12) (parallel, Centronics). Sockets for joystick (14) (9 pin D-Type), stereo sound output (15), video output (10) (RGB and sync or composite video or luminance and sync).

1.2) Outside the case.

The CPC464 system comes with a choice of two types of direct input video monitor, each includes a 5v power supply for the computer, designed to suit local mains voltage standards in the country of sale. In addition, there is an optional PSU and UHF TV modulator, the MP1.

Cables will be required to connect to a Centronics compatible printer and Hi-fi unit.

1.3) Display specification.

The screen operates from 16K of memory. The machine has a set of 27 colours which can be freely selected to form a palette. The number of inks on the palette depends on the screen mode. A number of inks can be set to the same colour if required. Pixels on the screen are defined as dots of a particular ink. (Note that the background, or paper, requires one ink from the available palette). There is a BORDER surrounding the active picture area which can be independantly set to any of the 27 colours.

Mode	Number of Inks	Vert dots	Horiz dots	Horiz chars
Normal	4	200	320	40
High Res	2	200	640	80
Multi Colour	16	200	160	20

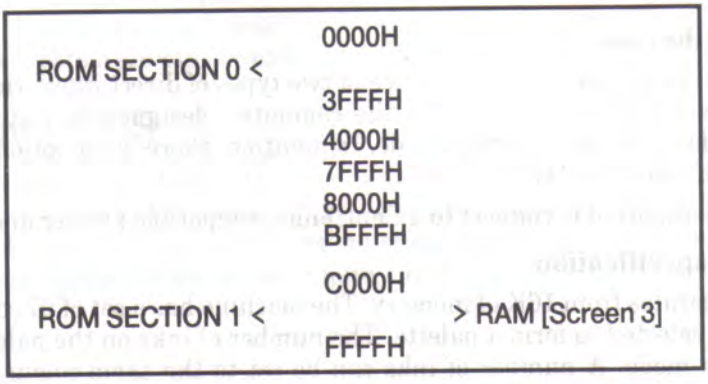
NB :The computer will produce a grey scale effect if viewed on a monchrome monitor. The order of colours in increasing brightness is as follows:-

GREY LEVEL	COLOUR	GREY LEVEL	COLOUR
0	BLACK	13	WHITE
1	BLUE	14	PASTEL BLUE
2	BRIGHT BLUE	15	ORANGE
3	RED	16	PINK
4	MAGENTA	17	PASTEL MAGENTA
5	MAUVE	18	BRIGHT GREEN
6	BRIGHT RED	19	SEA GREEN
7	PURPLE	20	BRIGHT CYAN
8	BRIGHT MAGENTA	21	LIME GREEN
9	GREEN	22	PASTEL GREEN
10	CYAN	23	PASTEL CYAN
11	SKY BLUE	24	BRIGHT YELLOW
12	YELLOW	25	PASTEL YELLOW
		26	BRIGHT WHITE

1.4) Memory Map

The 64K of RAM is allocated as follows.

Note that part of the ROM overlays the screen RAM, thereby releasing the maximum possible area to user RAM during BASIC operations.



When there is both RAM and ROM at an address, then READING accesses the ROM and WRITING accesses the RAM. Either ROM section can be turned off, allowing read access to the RAM at the same address.

1.5) Add-on ability

1.5.1) Sideways ROMS

Provision is made for additional ROMS to be selected in place of any part of the on-board ROM. The address arbitration and bank selection logic will be contained in a module connected to the expansion bus, but all the signals required are brought to the expansion bus.

1.5.2) Additional RAM

Additional RAM can be switched in place of any part of the on-board RAM. The address arbitration and bank selection logic will be contained in a module connected to the expansion bus, but all the signals required are brought to the expansion bus. This memory will be read-only and a special scheme involving I/O mapping will be required to write to this additional RAM from the computer.

1.5.3) Additional I/O

Most I/O port addresses are reserved by the computer, in particular addresses below **7Fxx** should not be used at all. The following can be used by external hardware.

F8xx, F9xx, FAxx, FBxx

Expansion bus peripherals must decode addresses on **A0** to **A7** whilst address **A10** is low. Expansion bus I/O channels in the address range **F800** to **FBFF** are reserved as follows:-

Addresses	A0-A7
00 - 7B	** Do not use **
7C - 7F	Reserved for disk interface
80 - BB	** Do not use **
BC - BF	Reserved for future use
C0 - DB	** Do not use **
DC - DF	Reserved for communications interfaces
E0 - FF	Available for user peripherals

Note that Z80 instructions which place the **B** register on the upper half of the address bus (**A15-A8**) must be used.

2) Keyboard.

A full reset is operated by **[CTRL][SHIFT][ESC]** pressed together. Keys causing printing characters or cursor movement will auto-repeat under firmware control, excluding all keys on the numeric keypad..

[ESC] Suspends program execution. Followed by a further **[ESC]** terminates execution. Followed by any other character resumes program execution.

[CAPS LOCK] is a toggle, operated by the caps lock key. Shift lock is a toggle operated by **[CTRL][CAPS LOCK]** pressed together.

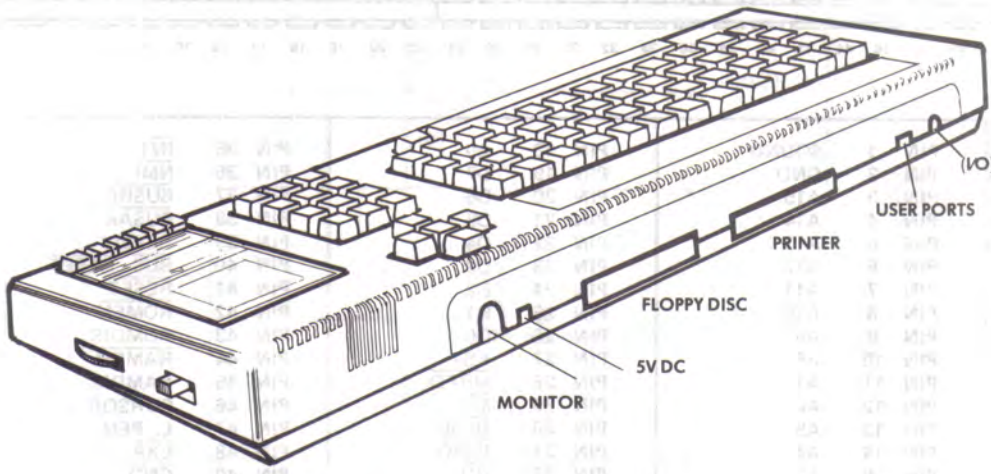
The copy cursor is detached from the input cursor by operating **[SHIFT]** together with the cursor keys. Input can be obtained from the character beneath the copy cursor by pressing the **[COPY]** key.

The cursor keys are intended to allow editing of the input buffer, which may spread over a number of screen lines. The cursor keys may be used to position the start of keyboard input to any screen position prior to any keyboard input being received. Once any keyboard input has been received then the screen position is fixed. The new input text will overwrite any existing content on the screen at that position.

[DEL] is a backwards delete and **[CLR]** is a forward delete.

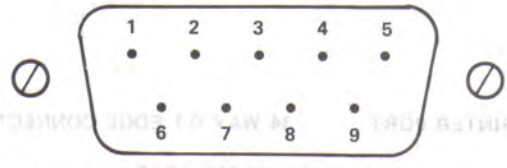
Appendix V:

Rear panel connection to the CPC 464



PADDLE PORT CONNECTOR (9 PIN D)

VIEWED FROM REAR



PIN 1	UP	PIN 6	FIRE 2
PIN 2	DOWN	PIN 7	FIRE 1
PIN 3	LEFT	PIN 8	COMMON
PIN 4	RIGHT	PIN 9	COM 2
PIN 5	SPARE		

VIDEO OUTPUT CONNECTOR (6 PIN DIN)

VIEWED FROM REAR

PIN 1	RED	PIN 4	SYNC
PIN 2	GREEN	PIN 5	GND
PIN 3	BLUE	PIN 6	LUM

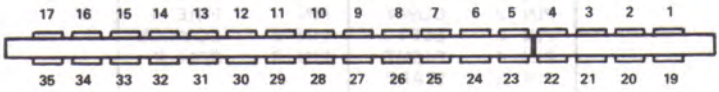
EXPANSION PORT 50 WAY 0.1 EDGE CONNECTOR

VIEWED FROM REAR

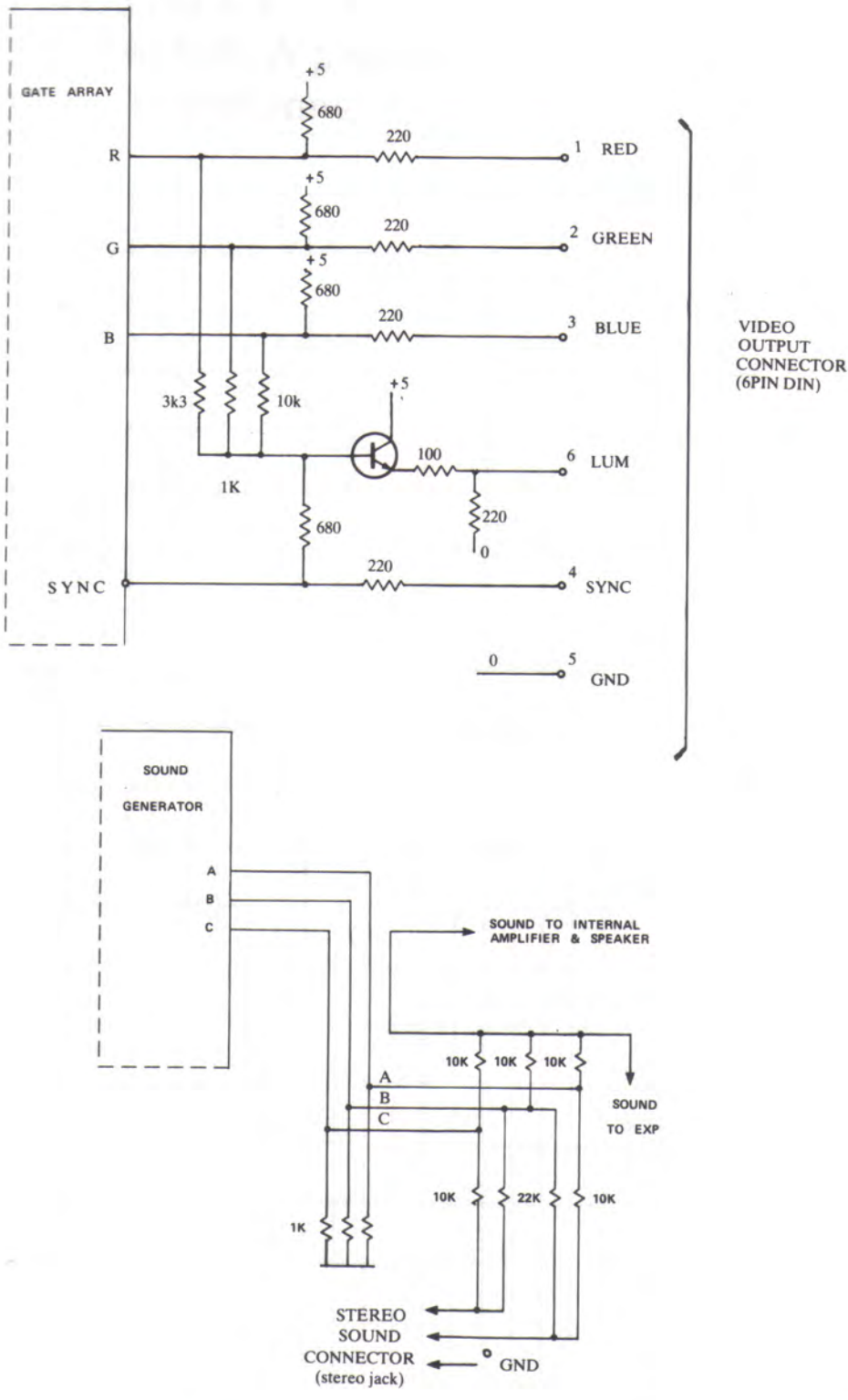
PIN	1	SOUND	PIN	18	A0	PIN	35	$\overline{\text{INT}}$
PIN	2	GND	PIN	19	D7	PIN	36	$\overline{\text{NMI}}$
PIN	3	A15	PIN	20	D6	PIN	37	$\overline{\text{BUSRD}}$
PIN	4	A14	PIN	21	D5	PIN	38	$\overline{\text{BUSAK}}$
PIN	5	A13	PIN	22	D4	PIN	39	READY
PIN	6	A12	PIN	23	D3	PIN	40	$\overline{\text{BUS RESET}}$
PIN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

PRINTER PORT 34 WAY 0.1 EDGE CONNECTOR

VIEWED FROM REAR



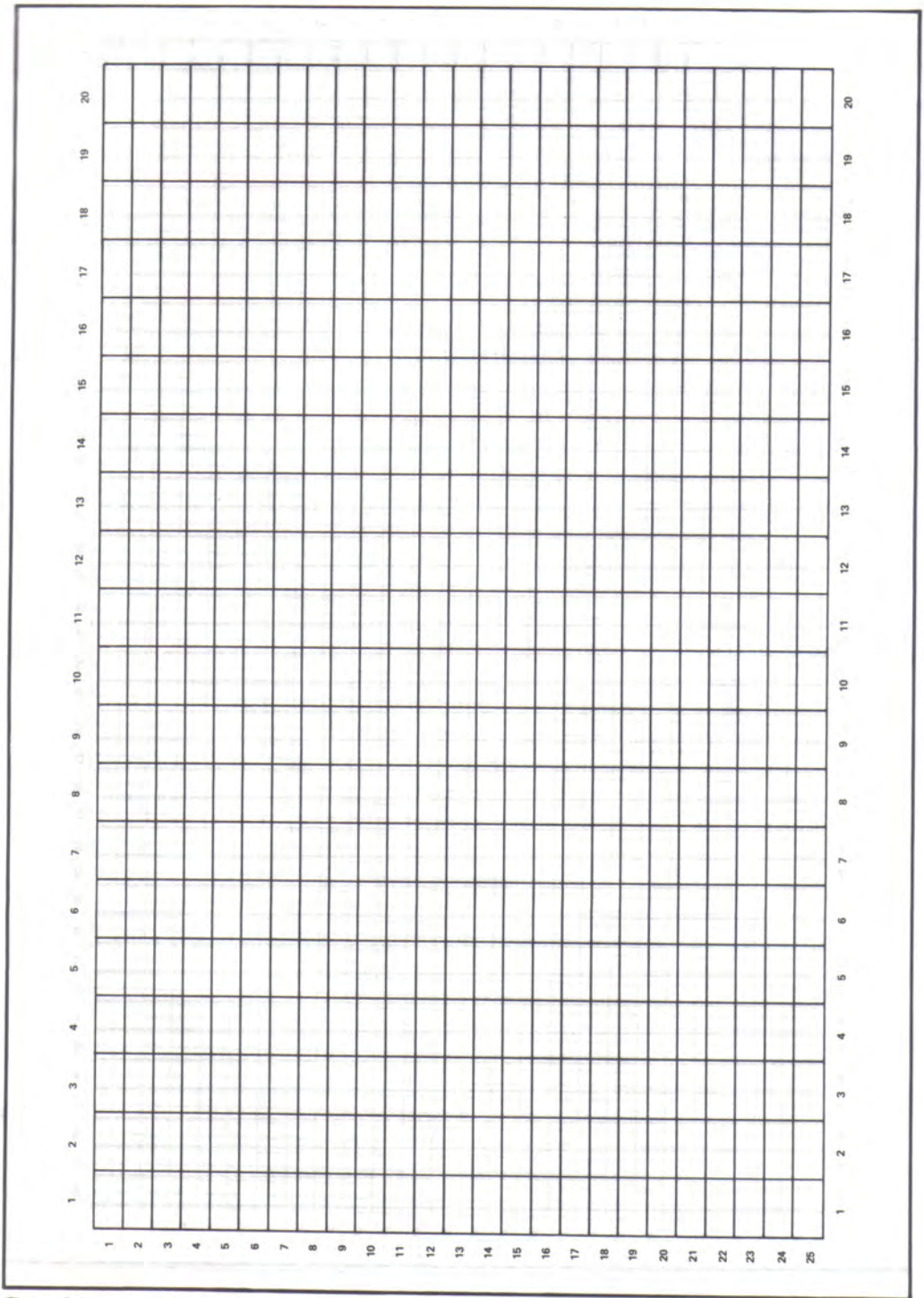
PIN	1	$\overline{\text{STROBE}}$	PIN	19	GND
PIN	2	D0	PIN	20	GND
PIN	3	D1	PIN	21	GND
PIN	4	D2	PIN	22	GND
PIN	5	D3	PIN	23	GND
PIN	6	D4	PIN	24	GND
PIN	7	D5	PIN	25	GND
PIN	8	D6	PIN	26	GND
PIN	9	GND	PIN	28	GND
PIN	11	BUSY	PIN	33	GND
PIN	14	GND			
PIN	16	GND	All other pins NC		


GATE ARRAY
R
G
B
SYNC
+5
680
220
1 RED
2 GREEN
3 BLUE
3k3
10k
1K
100
6 LUM
220
0
4 SYNC
0 5
GND
VIDEO
OUTPUT
CONNECTOR
(6PIN DIN)
SOUND
GENERATOR
A
B
C
SOUND TO INTERNAL
AMPLIFIER & SPEAKER
10K
22K
SOUND
TO EXP
STEREO
SOUND
CONNECTOR
(stereo jack)
GND

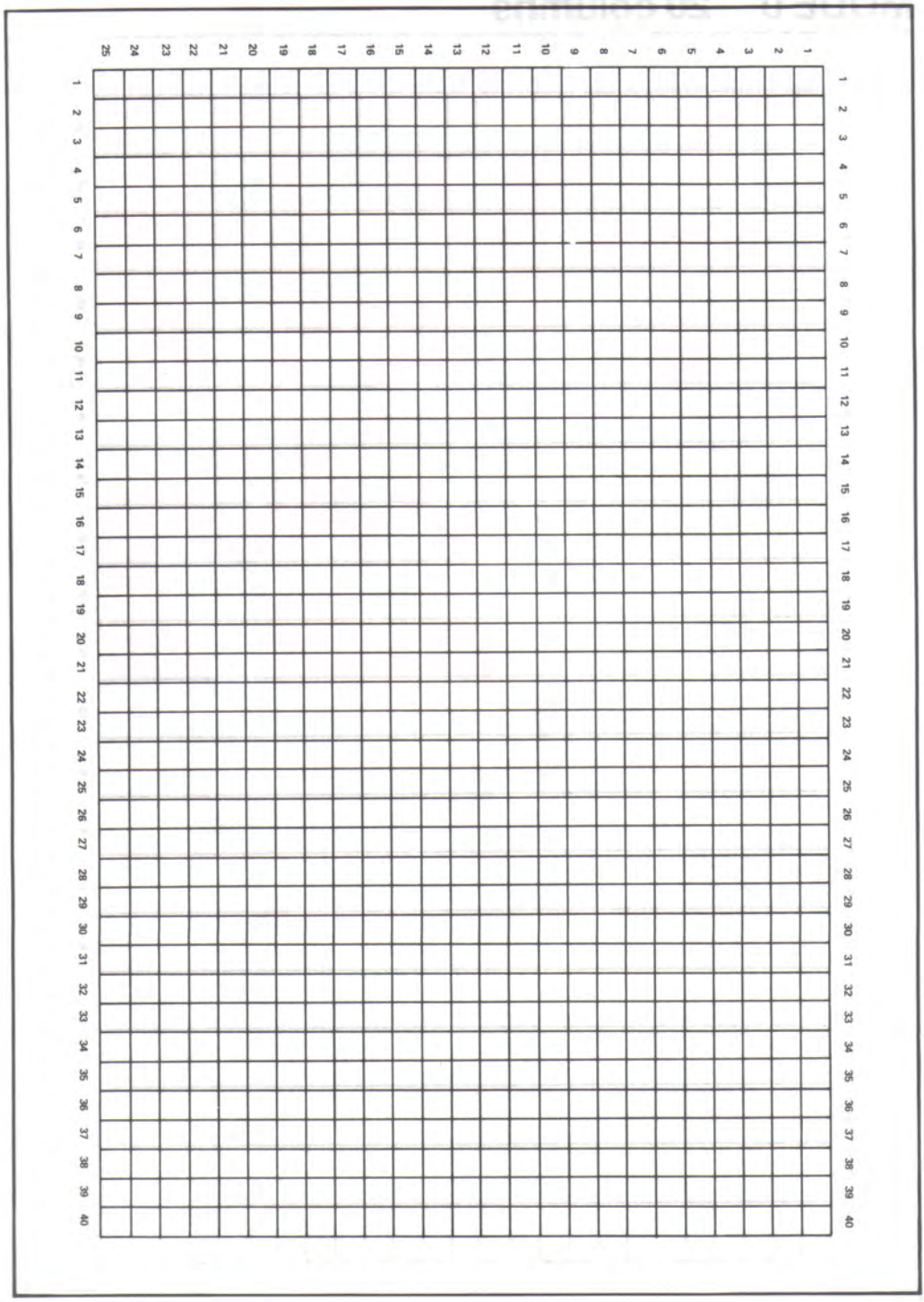
Appendix VI:

TEXT and WINDOW planner

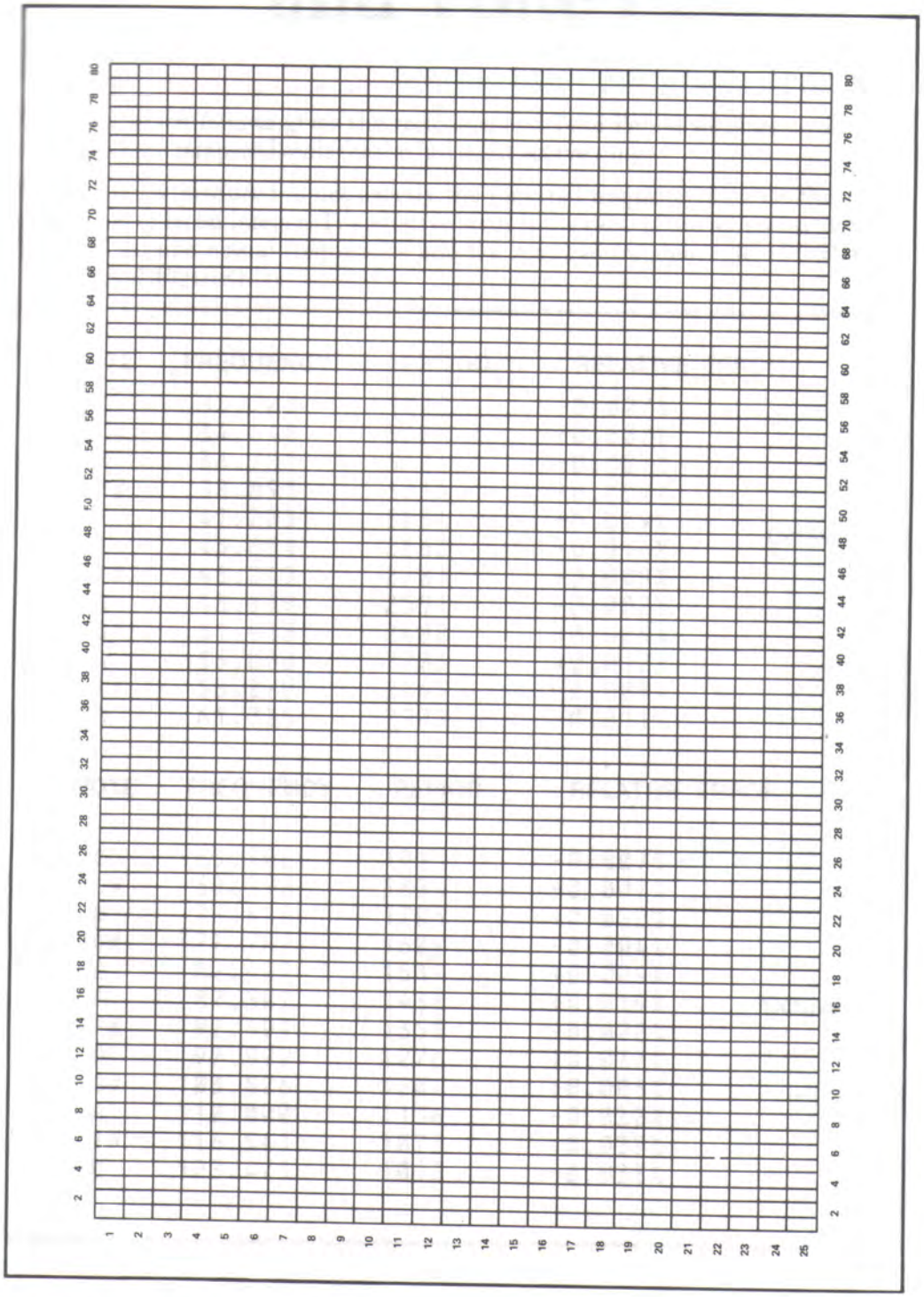
MODE 0 20 columns



Text and WINDOW planner
Mode 1 40 columns



Text and WINDOW planner
Mode 2 80 Columns



Appendix VII: Notes and Tone Periods.

The table which follows gives the recommended Tone Period settings for notes in the usual even tempered scale for the full eight octave range.

The frequency produced is not exactly the required frequency because the period setting has to be an integer. The relative error is the ratio of the difference between the required and actual frequencies and the required frequency, ie: (REQUIRED -ACTUAL)/REQUIRED).

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	32.703	3822	-0.007%	
C#	34.648	3608	+0.007%	
D	36.708	3405	-0.007%	
D#	38.891	3214	-0.004%	
E	41.203	3034	+0.009%	
F	43.654	2863	-0.016%	Octave -3
F#	46.249	2703	+0.009%	
G	48.999	2551	-0.002%	
G#	51.913	2408	+0.005%	
A	55.000	2273	+0.012%	
A#	58.270	2145	-0.008%	
B	61.735	2025	+0.011%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	65.406	1911	-0.007%	
C#	69.296	1804	+0.007%	
D	73.416	1703	+0.022%	
D#	77.782	1607	-0.004%	
E	82.407	1517	+0.009%	
F	87.307	1432	+0.019%	Octave -2
F#	92.499	1351	-0.028%	
G	97.999	1276	+0.037%	
G#	103.826	1204	+0.005%	
A	110.000	1136	-0.032%	
A#	116.541	1073	+0.039%	
B	123.471	1012	-0.038%	

Appendix VII: Notes and Tone Periods.

The table which follows gives the recommended Tone Period settings for notes in the usual even tempered scale for the full eight octave range.

The frequency produced is not exactly the required frequency because the period setting has to be an integer. The relative error is the ratio of the difference between the required and actual frequencies and the required frequency, ie: (REQUIRED -ACTUAL)/REQUIRED).

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	32.703	3822	-0.007%	
C#	34.648	3608	+0.007%	
D	36.708	3405	-0.007%	
D#	38.891	3214	-0.004%	
E	41.203	3034	+0.009%	
F	43.654	2863	-0.016%	Octave -3
F#	46.249	2703	+0.009%	
G	48.999	2551	-0.002%	
G#	51.913	2408	+0.005%	
A	55.000	2273	+0.012%	
A#	58.270	2145	-0.008%	
B	61.735	2025	+0.011%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	65.406	1911	-0.007%	
C#	69.296	1804	+0.007%	
D	73.416	1703	+0.022%	
D#	77.782	1607	-0.004%	
E	82.407	1517	+0.009%	
F	87.307	1432	+0.019%	Octave -2
F#	92.499	1351	-0.028%	
G	97.999	1276	+0.037%	
G#	103.826	1204	+0.005%	
A	110.000	1136	-0.032%	
A#	116.541	1073	+0.039%	
B	123.471	1012	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	130.813	956	+0.046%	
C#	138.591	902	+0.007%	
D	146.832	851	-0.037%	
D#	155.564	804	+0.058%	
E	164.814	758	-0.057%	
F	174.614	716	+0.019%	**Octave -1**
F#	184.997	676	+0.046%	
G	195.998	638	+0.037%	
G#	207.652	602	+0.005%	
A	220.000	568	-0.032%	
A#	233.082	536	-0.055%	
B	246.942	506	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	261.626	478	+0.046%	Middle C
C#	277.183	451	+0.007%	
D	293.665	426	+0.081%	
D#	311.127	402	+0.058%	
E	329.628	379	-0.057%	
F	349.228	358	+0.019%	**Octave 0**
F#	369.994	338	+0.046%	
G	391.995	319	+0.037%	
G#	415.305	301	+0.005%	
A	440.000	284	-0.032%	International A
A#	466.164	268	-0.055%	
B	493.883	253	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	523.251	239	+0.046%	
C#	554.365	225	-0.215%	
D	587.330	213	+0.081%	
D#	622.254	201	+0.058%	
E	659.255	190	+0.206%	
F	698.457	179	+0.019%	**Octave 1**
F#	739.989	169	+0.046%	
G	783.991	159	-0.277%	
G#	830.609	150	-0.328%	
A	880.000	142	-0.032%	
A#	932.328	134	-0.055%	
B	987.767	127	+0.356%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	1046.502	119	-0.374%	
C#	1108.731	113	+0.229%	
D	1174.659	106	-0.390%	
D#	1244.508	100	-0.441%	
E	1318.510	95	+0.206%	
F	1396.913	89	-0.543%	Octave 2
F#	1479.978	84	-0.548%	
G	1567.982	80	+0.350%	
G#	1661.219	75	-0.328%	
A	1760.000	71	-0.032%	
A#	1864.655	67	-0.055%	
B	1975.533	63	-0.435%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	2093.004	60	+0.462%	
C#	2217.461	56	-0.662%	
D	2349.318	53	-0.390%	
D#	2489.016	50	-0.441%	
E	2637.021	47	-0.855%	
F	2793.826	45	+0.574%	
F#	2959.955	42	-0.548%	Octave 3
G	3135.963	40	+0.350%	
G#	3322.438	38	+0.992%	
A	3520.000	36	+1.357%	
A#	3729.310	34	+1.417%	
B	3951.066	32	+1.134%	

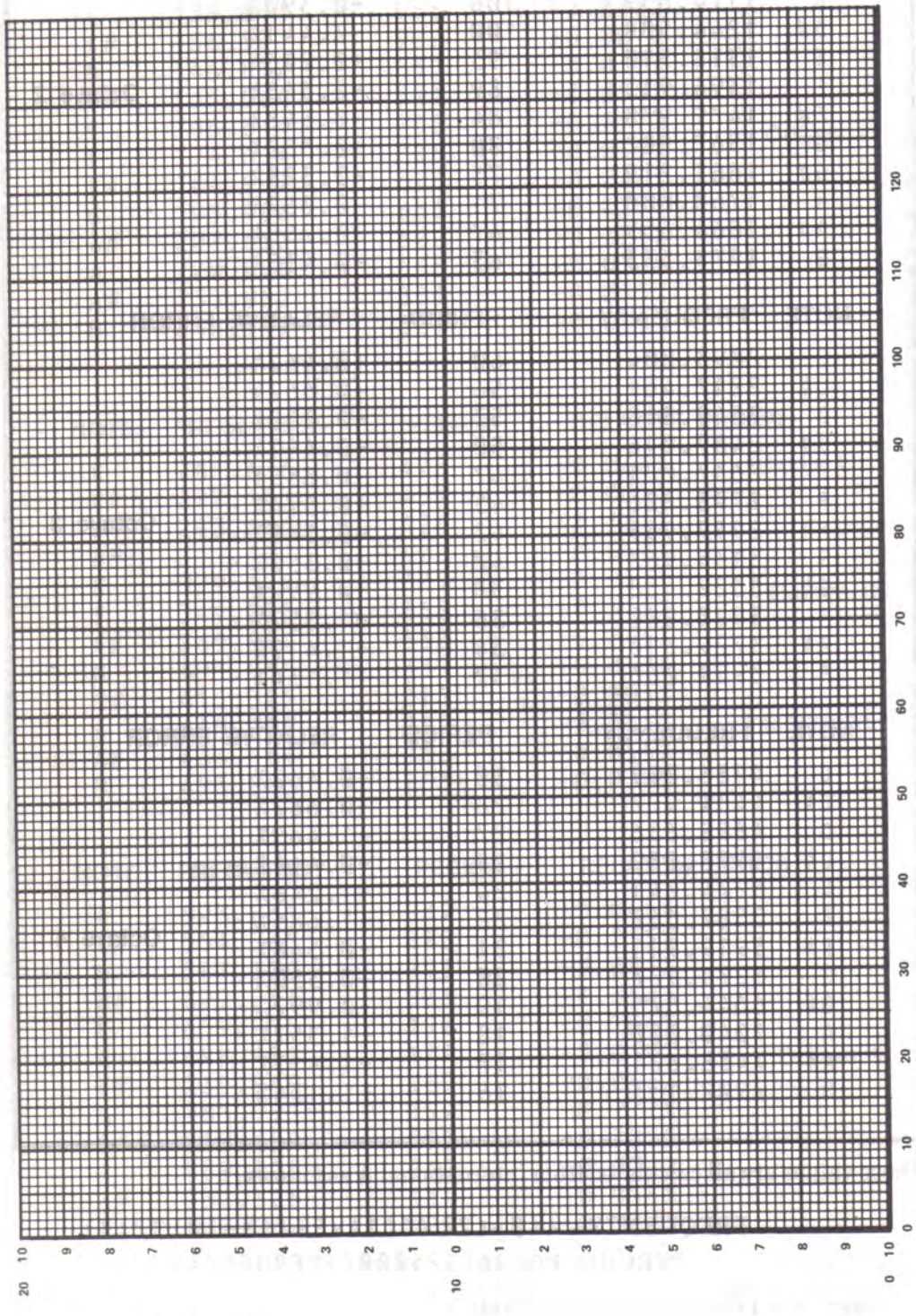
NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	4186.009	30	+0.462%	
C#	4434.922	28	-0.662%	
D	4698.636	27	+1.469%	
D#	4978.032	25	-0.441%	
E	5274.041	24	+1.246%	
F	5587.652	22	-1.685%	
F#	5919.911	21	-0.548%	Octave 4
G	6271.927	20	+0.350%	
G#	6644.875	19	+0.992%	
A	7040.000	18	+1.357%	
A#	7458.621	17	+1.417%	
B	7902.133	16	+1.134%	

These values are all calculated from International A as follows :

```
FREQUENCY = 440*(2↑(OCTAVE+((N-10)/12)))
PERIOD = ROUND(125000/FREQUENCY)
```

where N is 1 for C, 2 for C#, 3 for D etc.

Sound envelope/Music planner sheet



Appendix VIII: Error codes and reserved words

Error Numbers and Error Messages

When BASIC encounters a program statement, word or variable that it cannot understand or process, it will stop and display an error message. The form of the message will generally indicate what went wrong - and sometimes, if the error is a typographical error during program entry, BASIC will prompt in edit mode, with the line where the incorrect entry was made.

The most popular error to greet the inaccurate typist is the `Syntax Error` (number 2), and BASIC prompts with the line to edit if encountered in program mode. In direct mode, it simply states that an error occured, and assumes that the last line typed is visible to spot the problem.

If the `ON ERROR GOTO` command is included at the beginning of a program, it may refer the computer to a given line number when detecting an error. In the following example, the computer is referred to line 1000 when detecting an error:

```
10 ON ERROR GOTO 1000
```

program

```
1000 PRINT CHR$(7):MODE 2:INK 1,0: INK 0,9: CLS :LIST
```

Whereupon the CPC464 will beep, clear the current screen, change to a suitable colour combination for the 80 column display, and list the program ready for examination. If the error is a `Syntax error` it will appear at the foot of the listing, awaiting correction in the line edit mode, although the `Syntax error` message is suppressed.

Remember to `END` the program on the last line before 1000 if you wish to save the results on the screen.

BASIC will not produce error messages for valid input - so it must be assumed that whenever an error does occur, it can be traced back to an error in the form of the program, usually guided by the message produced to help in the process of de-bugging. As with most things, you will learn most readily from your mistakes, so make the most of the fact that the CPC464 is the most tolerant of tutors: you will tire of trying long before the CPC464 loses its patience!

All errors generated by BASIC are listed here, in error number order. The messages produced by BASIC are given, as well a brief description of possible causes.

1 Unexpected NEXT

A NEXT command has been encountered while not in a FOR loop, or the control variable in the NEXT command does not match that in the FOR.

2 Syntax Error

BASIC cannot understand the given line because a construct within it is not legal.

3 Unexpected RETURN

A RETURN command has been encountered when not in a subroutine.

4 DATA exhausted

A READ command has attempted to read beyond the end of the last DATA.

5 Improper argument

This is a general purpose error.The value of a function's argument, or a command parameter is invalid in some way.

6 Overflow

The result of an arithmetic operation has overflowed. This may be a floating point overflow, in which case some operation has yielded a value greater than 1.7E–38 (approx.). Alternatively, this may be the result of a failed attempt to change a floating point number to a 16 bit signed integer.

7 Memory full

The current program or its variables may be simply too big, or the control structure is too deeply nested (nested GOSUBs, WHILEs or FORs).
A MEMORY command will give this error if an attempt is made to set the top of BASIC's memory too low, or to an impossibly high value. Note that an open cassette file has a buffer allocated to it, and that may restrict the values that MEMORY may use.

8 Line does not exist

The line referenced cannot be found.

9 Subscript out of range

One of the subscripts in an array reference is too big or too small.

10 Array already dimensioned

One of the arrays in a DIM statement has already been declared.

11 Division by zero

May occur in Real division, integer division, integer modulus or in exponentiation.

12 Invalid direct command

The last command attempted is not valid in Direct Mode.

13 Type mismatch

A numeric value has been presented where a string value is required. and vice versa, or an invalidly formed number has been found in READ or INPUT.

14 String space full

So many strings have been created that there is no further room available, even after 'garbage collection'.

15 String too long

String exceeds 255 characters in length. May be generated by adding a number of strings together.

16 String expression too complex

String expressions may generate a number of intermediate string values. When the number of these values exceeds a reasonable limit, BASIC gives up, and this error results.

17 Cannot CONTinue

For one reason or another the current program cannot be restarted using CONT. Note that CONT is intended for restarting after a STOP command, **[ESC][ESC]** or error, and that any alteration of the program in the meantime makes a restart impossible.

18 Unknown user function

No DEF FN has been executed for the FN just invoked.

19 RESUME missing

The end of the program has been encountered while in Error Processing Mode (ie in an ON ERROR GOTO routine).

20 Unexpected RESUME

RESUME is only valid while in Error Processing Mode (ie in an ON ERROR GOTO routine).

21 Direct command found

When loading a program from cassette a line without a line number has been found.

22 Operand missing

BASIC has encountered an incomplete expression.

23 Line too long

A line when converted to BASIC internal form becomes too big.

24 EOF met

An attempt has been made to read past end of file on the cassette input stream.

25 File type error

The cassette file being read is not of a suitable type. OPENIN is only prepared to open ASCII text files. LOAD, RUN etc, are only prepared to deal with the file types produced by SAVE.

26 NEXT missing

Cannot find a NEXT to match a FOR command.

27 File already open

An OPENIN or OPENOUT command has been executed before the previously opened file has been closed.

28 Unknown command

BASIC cannot find a taker for an external command.

29 WEND missing

Cannot find a WEND to match a WHILE command.

30 Unexpected WEND

Encountered a WEND when not in a WHILE loop, or a WEND that does not match the current WHILE loop.

BASIC Keywords

The following are the BASIC keywords, they are reserved and cannot be used as variable names.

```
ABS, AFTER, AND, ASC, ATN, AUTO

BIN$, BORDER

CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG, CLOSEIN,
CLOSEOUT, CLS, CONT, COS, CREAL

DATA, DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE,
DI, DIM, DRAW, DRAWR

EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR,
ERROR, EVERY, EXP

FIX, FN, FOR, FRE

GOSUB, GOTO

HEX$, HIMEM

IF, INK, INKEY, INKEY$, INP, INPUT, INSTR, INT

JOY
```

KEY

LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10, LOWER$

MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER

NEXT, NEW, NOT

ON, ON BREAK, ON ERROR GOTO, ON SQ, OPENIN, OPENOUT, OR, ORIGIN, OUT

PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT

RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM, RESTORE, RESUME, RETURN, RIGHT$, RND, ROUND, RUN

SAVE, SGN, SIN, SOUND, SPACE$, SPC, SPEED, SQ, SQR, STEP, STOP, STR$, STRING$, SWAP, SYMBOL

TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF, TRON

UNT, UPPER$, USING
VAL, VPOS

WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE

XOR, XPOS

YPOS

ZONE

INDEX