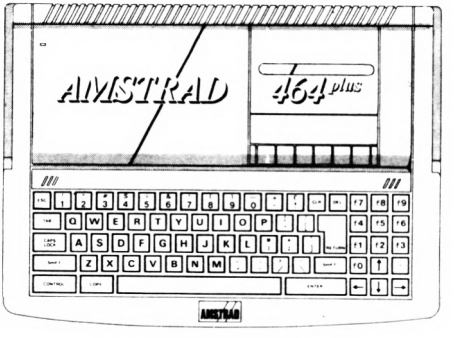
464 PLUS

6128 PLUS

AMSTRAD

464/6128 Plus
User Instructions



The product described in this manual and products for use with it are subject to continuous development and improvement.

This manual is provided to you free of charge and is intended only to assist the reader in the use of the product. The information contained in this manual and literature provided with the product is given by AMSTRAD in good faith.

AMSTRAD MAKES NO WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTIBILITY OR FITNESS FOR A PARTICULAR PURPOSE AND AMSTRAD ACCEPTS NO LIABILITY FOR ANY LOSS OR DAMAGE ARISING FROM THE USE OF ANY INFORMATION PROVIDED OR OMITTED.

This manual may include technical inaccuracies or typographical errors. Changes are periodically made to the information contained herein; these changes will be incorporated in new editions of the publications. AMSTRAD may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time but does not undertake to notify customers of these changes.

The products referred to herein are not designed and should not be used for, or in connection with, life critical functions or any activity in which an error or a fault may result in physical damage or injury to person(s) and AMSTRAD does not authorise such use.

You must carefully read these instructions and all other literature provided with the product, including the Guarantee Card, prior to operation.

IMPORTANT

The terms and instructions, contained in the Guarantee Card provided with the product, must be strictly complied with.

EXCLUSIONS OF CONSEQUENTIAL LOSS

IN ANY EVENT, AMSTRAD ACCEPTS NO LIABILITY FOR ANY CONSEQUENTIAL LOSS OR DAMAGE ARISING FROM THE USE OF THIS MANUAL OR PRODUCTS USED WITH IT OR ANY INFORMATION PROVIDED IN OR OMITTED FROM THIS MANUAL INCLUDING, BUT NOT LIMITED TO, ECONOMIC OR FINANCIAL LOSS, DAMAGE TO PERIPHERAL EQUIPMENT OR PRODUCTS, LOSS OF USE, PRODUCTIVITY OR TIME.

Nothing in this document is intended to exclude or restrict any of the consumer's rights where to do so would be illegal.

All correspondence relating to the product or this manual should be addressed to:

AMSTRAD PLC
BRENTWOOD HOUSE
169 KINGS ROAD
BRENTWOOD
ESSEX, CM14 4EF
ENGLAND

All maintenance and service on the product must be carried out by AMSTRAD authorised dealers. AMSTRAD cannot accept any liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This guide is intended only to assist the reader in the use of the product, and therefore AMSTRAD shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this guide or any incorrect use of the product.

First Published 1990.
Compiled by Roland Perry

Published by AMSTRAD PLC

PRINTED IN ENGLAND

IMPORTANT

You must read this....

Installation Notes

1. Always connect the Mains Lead to a 3-pin plug following the instructions contained in part 1 of the Foundation course.

2. Never attempt to connect the system to any Mains Supply other than 220-240V ~ 50Hz.

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

4. To avoid eye-strain, the monitor should be placed as far away as possible from the keyboard and operated in an adequately lit room. The monitor BRIGHTNESS control should be kept to as low a setting as possible.

5. Do not block or cover any ventilation holes.

6. Do not use or store the equipment in excessively hot, cold, damp, or dusty areas.

7. Do not attempt to insert or remove the cartridge while the power is switched on.

8. Do not attempt to operate the computer without a cartridge installed.

Compatibility

The 464 Plus and 6128 Plus may be used with most software and peripherals intended for the earlier CPC464, CPC664 and CPC6128 models. While every effort has been made to retain compatibility it is inevitable that some software written for the CPC464 and CPC6128 models may not operate correctly with the 464 Plus and 6128 Plus models. Before buying software or peripherals intended for the earlier models you should check with your dealer that they are suitable for use with the 464 Plus or 6128 Plus.

NB. The connectors for Expansion peripherals, Printers and Second Disc drives have the same function as those on the earlier models, but may have a different connector.

Special Disc Operation Notes (6128)

(Don't worry if you are a little baffled by the technical jargon in this section; the importance of these warnings will become clearer as you work through this manual.)

1. Never switch the 6128 on or off with a disc in the drive. Doing so will corrupt your disc, losing valuable programs or data.

2. Always make back-up (duplicate) copies of discs which contain valuable programs. It is especially important to make back-up copies of the master CP/M system disc provided with the 6128. Should you otherwise accidentally lose or corrupt your disc, replacing it could prove expensive.

3. Make sure that you do not accidentally overwrite your master CP/M system disc, by ensuring that the Write Protect holes on the disc are always open.

4. For maximum data reliability, the disc drive section of the computer should NOT be placed directly in front of the monitor, but to the right of it. Do not place the computer close to any source of electrical interference.

5. Never touch the floppy disc surface itself, inside its protective casing.

6. Do not eject a disc while it is being read from or written to.

7. Always remember that formatting a disc will erase any previous contents.

8. Always keep disc drives and discs away from magnetic fields.

9. The licence agreement for your CP/M system disc, (which is electronically serial-number encoded) permits its use on a single computer system only. In particular this means that you are prohibited from giving any other person a disc with YOUR serial-numbered copy of CP/M on it. Carefully read the Software Licence Agreement (Appendix 1), towards the end of this manual.

Contents

Chapter 4
Using Discs (6128 only)

Chapter 5
AMSDOS and CP/M

Chapter 6
For Your Reference....

Chapter 6 continued....

Chapter 7
More about the Bank Manager....

Chapter 8
At Your Leisure....

Chapter 8 continued....

Appendices

Chapter 1
Foundation Course

Part 1: Setting Up....

The 464/6128 can be set up with either:

1. The Amstrad MM12 Monochrome Monitor
2. The Amstrad CM14 Colour Monitor

Fitting a Mains Plug

The 464/6128 operates from 220-240 Volt AC 50Hz Mains supply.

Fit a proper mains plug to the mains lead of either the MM12 or CM14. 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 type of plug is used, a 5 Amp fuse must be fitted either in the plug or adaptor or at the distribution board.

WARNING: THIS APPARATUS MUST BE EARTHED

Important

The wires in the mains lead are coloured in accordance with the following code:

Green-and-yellow	**:Earth**
Blue	**:Neutral**
Brown	**:Live**

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 GREEN-AND-YELLOW must be connected to the terminal which is marked by the letter 'E' or by the safety earth symbol ⏚ or coloured green or green-and-yellow.

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.

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

Do not attempt to remove any screws, nor open the casing of the computer or monitor. Always obey the warning on the rating label which is located underneath the case of the computer and on the rear cabinet of the monitor.

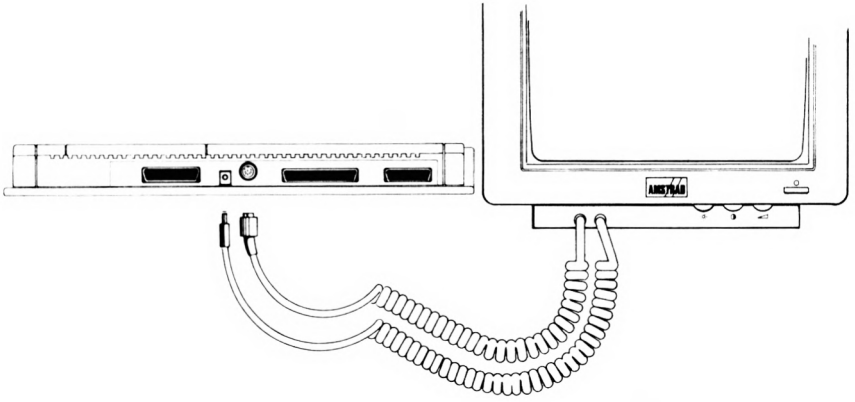
WARNING - LIVE PARTS INSIDE. DO NOT REMOVE ANY SCREWS

Setting up the 464/6128

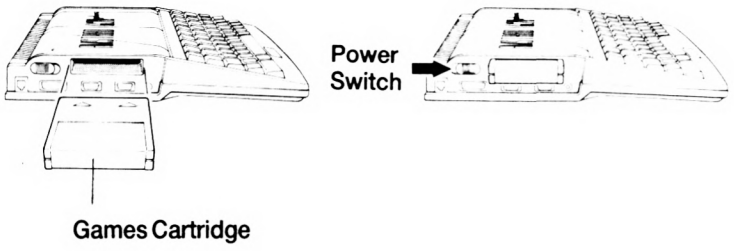
Place the computer on a suitable flat surface.

IMPORTANT: When choosing a position for the computer:

- make sure that it is installed close to, and within easy access of, the electrical mains supply socket.
- make sure that it is NOT situated near an artificial heat source, such as a radiator, near a water supply, nor in direct sunlight. These elements can damage your computer.



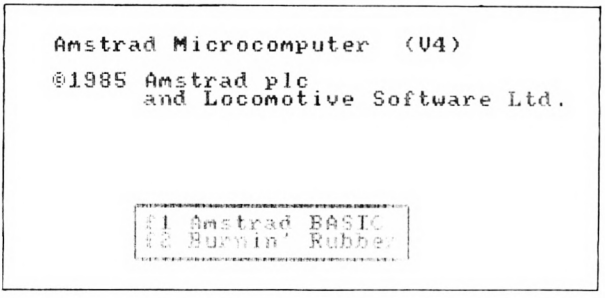
1. Make sure that the monitor is NOT plugged into the mains supply socket.
2. Connect the lead from the front of the monitor, which is fitted with the larger (8-pin DIN) plug, into the 8-pin DIN socket on the rear of the computer.
3. Connect the lead from the front of the of the monitor, which is fitted with the smaller (5V DC) plug, into the power socket on the rear of the computer.



Step 4. Insert the cartridge (power switch OFF) **Step 7. Switch on the computer**

4. Insert the supplied cartridge into the socket on the left-hand side of the computer. IMPORTANT: The cartridge can only be inserted or removed when the sliding power switch is in the OFF position.
5. Insert the mains plug into the supply socket and switch on.
6. Press the Power button on the front of the monitor, so that it is set to the IN position. When the button is set to the OUT position, the Mains supply to the system is OFF.
7. Switch on the computer using the Power switch on the left hand side. Sliding it towards you turns ON the power and sliding it away from you turns OFF the power.

When operating correctly the computer and monitor will display the following picture:



To avoid unnecessary eye-strain, adjust the controls for Brightness ☼ and Contrast ◑ on the lower front panel of the monitor. There is also a volume control ◢ to adjust the volume of the built-in loudspeakers.

Selecting BASIC or the Demonstration Game

The cartridge supplied with your 464/6128 includes both AMSTRAD BASIC and a Demonstration Game. To operate the AMSTRAD BASIC press the key with **[f1]** written on it (which is found towards the right-hand side of the keyboard) within 30 seconds of switching on the computer. If you press the key with **[f2]** written on it, or do not press the key with **[f1]** written on it within 30 seconds, then the Demonstration Game will be played.

Complete instructions on how to operate the AMSTRAD BASIC are contained in later sections of this manual.

NB. To play the demonstration game, after you have operated AMSTRAD BASIC, type in the command | GAME and then press the **[ENTER]** key. The | symbol is typed by holding down either of the **[SHIFT]** keys and then pressing the key with the @ and | symbols on it.

Part 2: Connecting your peripherals....

This section explains how the various peripherals, or add-ons, are connected to the 464/6128 system. Details concerning the use of these devices will be found in the appropriate sections of this manual. A diagram identifying the various sockets is printed in Chapter 6, Part 9.

IMPORTANT: DO NOT CONNECT OR DISCONNECT ANY PERIPHERALS WHILE THE POWER TO YOUR COMPUTER IS SWITCHED ON.

Joystick/Paddle Controller (PD-1)

The 464/6128 is supplied with one Paddle Controller. In addition a second Joystick or Paddle Controller is an item that you may wish to purchase if you are using the 464/6128 computer with games software which incorporates the facility for using more than one Joystick/Paddle Controller.

Connect the plug fitted to the supplied Paddle Controller lead into the frontmost Joystick Socket on the side of the computer. The 464/6128 can be used with two Joysticks or Paddle Controllers. The second one is fitted to the rearmost Joystick Socket on the computer.

Further information on writing your own software to use Joysticks will be found later in this manual

Light Pen/Light Gun

A Light Pen or Light Gun is an additional item that may be available if you are using the 464/6128 computer with games which incorporates the facility for a Light Gun or other software that incorporates the facility for a Light Pen.

Connect the plug fitted to the Light Gun or Light Pen lead into the AUXiliary Socket on the side of the computer.

These peripherals can only be used with specially written software supplied on tape, disc or cartridge.

Analog Joystick

An Analog Joystick is an additional item that you may wish to purchase if you are using the 464/6128 computer with games software which incorporates the facility for analog joystick control and "firing". An Analog Joystick provides more sophisticated control of the game than a normal Joystick or Games Controller.

Connect the plug fitted to the Analog Joystick lead into the Analog Joystick Socket on the side of the computer.

An Analog Joystick can only be used with specially written software, which will normally only be available on cartridge.

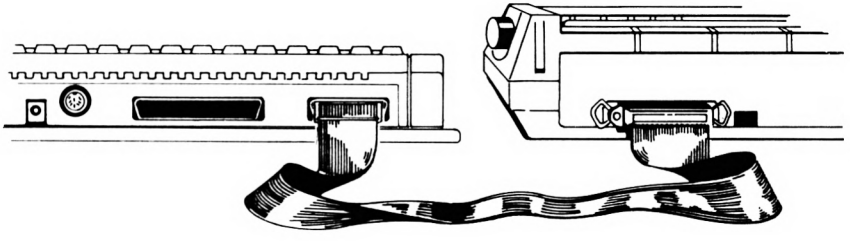
Printer

The 464/6128 can be used with any Centronics compatible parallel printer. You will require a suitable lead, which may be included with the printer.

Connect the end of the lead which is fitted with a 25 pin connector plug into the Printer Socket on the rear of the computer. Connect the other end of the lead which is fitted with a Centronics Style 36 pin plug into the socket on the rear of the printer. If the printer is equipped with security clips at each side of the socket, these may be clipped into the cut-outs at the side of the printer plug.

DO NOT plug the lead into the 36 pin Centronics Connector on the rear of the 6128. That is intended only for attaching a second disk drive.

Details of printer operation will be found later in this manual.



External Amplifier/Speakers

The 464/6128 may be connected to an external Stereo Amplifier. The input lead to your Stereo Amplifier should be terminated with a 3.5mm stereo jack plug which should be inserted into the Stereo Socket on the rear of the computer.

The computer will provide a fixed voltage signal out of the Stereo Socket and you should therefore use the controls on the amplifier itself to regulate the volume.

Expansion Devices

There is a general purpose 50-way expansion socket that is provided for add-ons such as serial interfaces, modems etc. Full connection and operation details should be included with the relevant add-on.

2nd Disc Drive (6128 only)

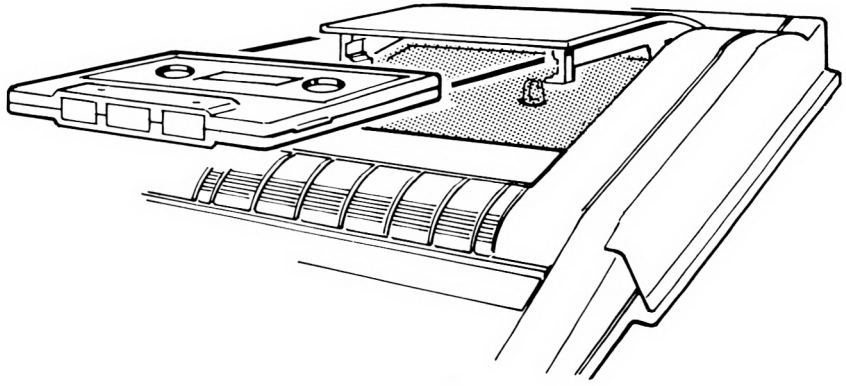
There is a socket provided for the attachment of a 2nd Disc Drive. Full instructions for attaching and operating the second drive should be included with the second drive.

Part 3: About Discs and Cassettes....

The 464/6128 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 disc.

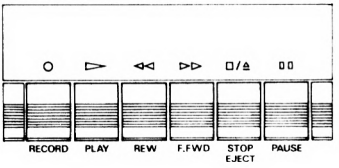
Cassette controls (464 only)

At the right hand rear of the keyboard unit you will find the cassette recorder. 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.



The right way to insert a cassette into the recorder

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



The 464 cassette 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 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.

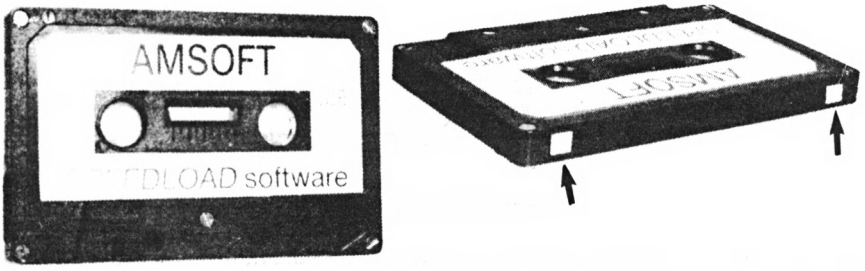
[REW] = **[REW]**ind the tape from the right hand spool to the left hand spool. There is no mechanical cancellation of this function at the tape end, so you should not leave the machine running in rewind mode or the tape drive motor may overheat when stalled.

[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 464 software, leaving the mechanical pause facility largely unused.

Write Protection



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 taut across the opening.

There are various techniques available using instructions in the 464 machine operating system to protect programs using software.

Loading the cassette

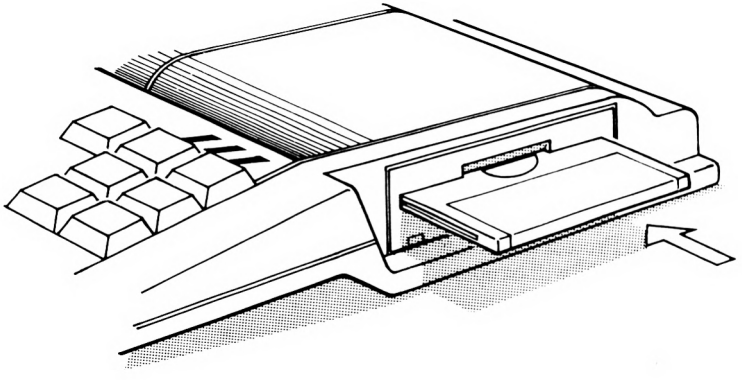
The cassette should be fully rewound (from the right hand to the left hand spool) - so if it is not, press the **[REW]** key until the motor stalls at the tape end. If the tape is accidentally looped out of the aperture on the front of the cassette, then you should rewind the tape into the body of the 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 464 memory), and throw the damaged tape away before you are tempted to try and re-use it.

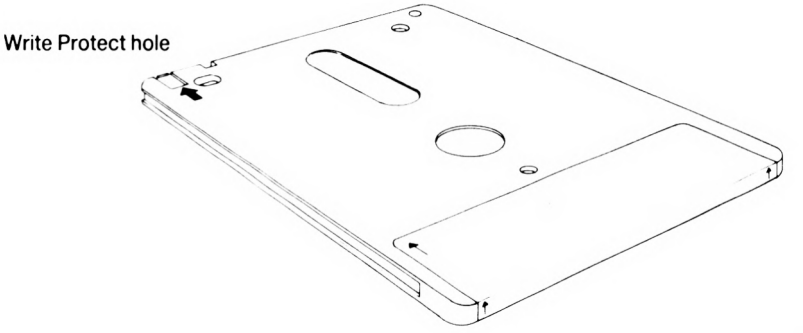
Insertion of Discs (6128 only)

Each side of a disc may be used separately. A disc should be inserted with its label facing outward from the drive, and with the side that you wish to use face up:



Write Protection

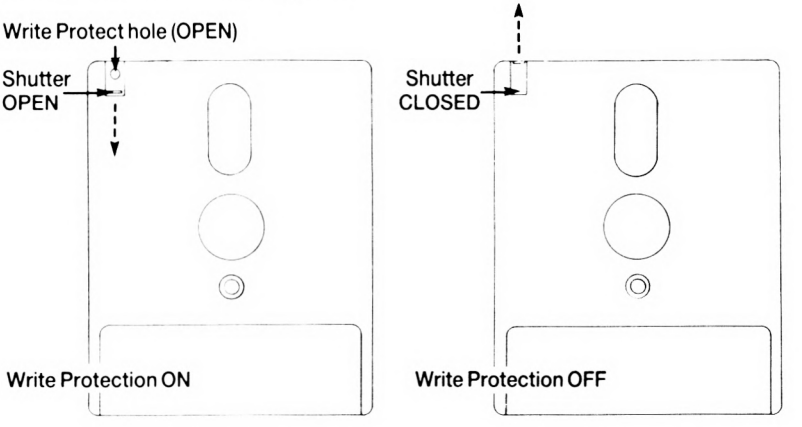
In the left hand corner of each side of a blank disc, you will see an arrow pointing to a small shuttered hole. This is called the Write Protect hole, and facilitates protection against erasure or 'overwriting':



When the hole is closed, data can be 'written' onto the disc by the computer. When the hole is open however, the disc will not allow data to be written onto it, thus enabling you to avoid accidental erasure of valuable programs.

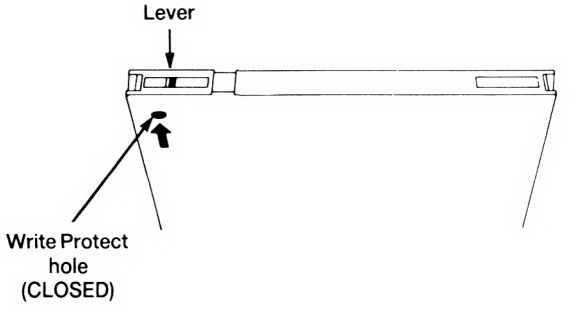
Various compact floppy disc manufacturers employ different mechanisms for opening and closing the Write Protect hole. The operation may be carried out on a typical compact floppy disc as follows:

To open the Write Protect hole, slide the small shutter located at the left hand corner of the disc, and the hole will be opened:

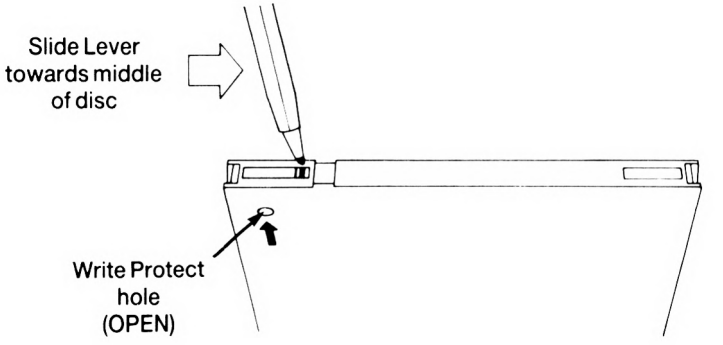


To close the Write protect hole, simply slide the shutter to its closed position.

Alternatively, other compact floppy discs employ a small plastic lever located in a slot at the left hand corner:



To open the Write Protect hole on this type of disc, slide the lever towards the middle of the disc, using the tip of a ball-point pen or similar object:



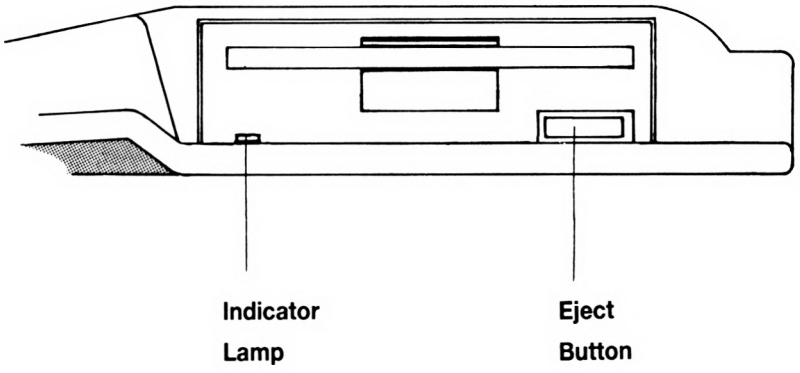
Note that regardless of the method employed to open and close the Write Protect hole, opening the hole in all cases facilitates protection against overwriting.

IMPORTANT

Always ensure that the Write Protect holes on your master CP/M system disc are open.

When Your Disc is in

At the front of the computer's disc drive which you will find in the righthand side of the 6128, you will see a red indicator lamp, and a push button for Eject:



Indicator Lamp

This indicates that data is being read from, or written to the disc.

If a 2nd disc drive is connected, the red indicator on the 2nd disc drive (Drive B) will illuminate constantly. It will extinguish when the main disc drive within the computer (Drive A) is reading or writing to disc.

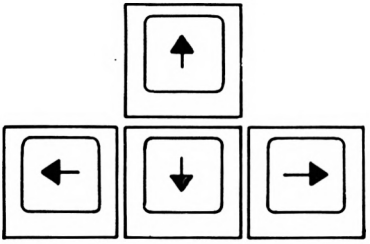
Eject Button

Pressing in the Eject button allows you to remove your disc from the drive.

Part 4: Getting Started....

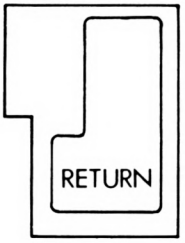
Before we start loading software and saving programs to disc or cassette, let's get familiar with some of the keys on the computer. Those of you who are experienced in using computers may skip this section.

With the computer switched on and the opening message on the screen, we're going to find out what the various keys do....



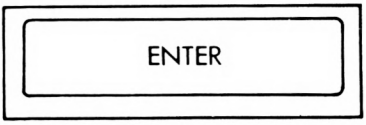
The cursor keys ↑ ↓ ← → (at the bottom right hand corner of the keyboard) move the position of the cursor (the small solid block) on the screen.

Press each of the cursor keys in turn, and you will see the cursor move about the screen.

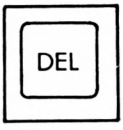


The **[RETURN]** key enters the information that you have typed into the computer. After the **[RETURN]** key is pressed, a new line is started on the screen. Each instruction that you type into the computer should be followed by pressing the **[RETURN]** key.

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

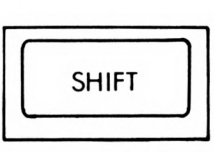


Under normal circumstances (by default), this key has the same effect as **[RETURN]** and can be used as such. However, like the function keys on the numeric keypad, the **[ENTER]** key may be re-defined for other uses. This will be explained later in the manual.



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 a b c d 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 a b c will also be removed.



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

e E

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 pressing any of the character keys, either on their own, or together with the **[SHIFT]** key.

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:

```
abcdef123456
```

On the screen you will see:

```
ABCDEF123456
```

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 number key. Type in the following while holding down the **[SHIFT]** key:

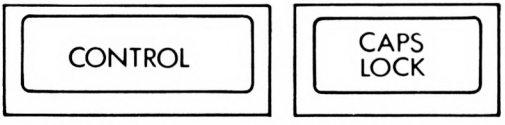
```
abcdef123456
```

On the screen you will see:

```
ABCEDF!"#$%&
```

If you wish to return to small (lower case) characters again, press the **[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 as follows:



Holding down the **[CONTROL]** key, then press the **[CAPS LOCK]** key once. This performs the function of a 'SHIFT LOCK'. Now type in:

```
abcdef123456
```

On the screen you will see:

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

Note that it is still possible to type in numbers while **[CONTROL]** and **[CAPS LOCK]** are in operation, by using the number keys (*f*0 to *f*9) at the right of the keyboard.

Part 5: Loading Software and games....

Welcome to those of you who skipped here from the beginning of the previous section!

Cartridge Games.

As we saw in Part 1, running the Demonstration Game supplied with your 464/6128 is as easy as pressing **[f2]**. If you purchase additional cartridges then these will not even need that simple step. The game will start automatically as soon as the computer is switched on.

Loading Software from Disc (6128 only)

Each software disc should come with its own instructions, but you will probably find that you are asked to type something like:

```
run"disc" [RETURN]
```

After a few seconds, your game will be loaded and ready to play.

If the program hasn't loaded, study any error message on the screen to see where you went wrong:

```
Drive A: disc missing
Retry, Ignore or Cancel?
```

....means that you have either not inserted your disc correctly, or, if you have a 2-drive system, that you have inserted it into Drive B.

```
DISC          not found
```

....means that you have either inserted the wrong disc, (or the wrong side of the disc), or you have not carefully and precisely typed in the name, `DISC`

```
Bad command
```

....means that you have probably mis-typed `DISC` by introducing an unwanted space character or punctuation mark.

```
Type mismatch
```

....means that you have omitted the quotation marks "

```
Syntax error
```

....means that you mistyped the word `run`

```
Drive A: read fail
Retry, Ignore or Cancel?
```

....means that the computer has failed to read data from your disc. Check that you have inserted the correct disc and press **R** to Retry. This message will appear if ever you corrupt your disc by switching the system on or off while the disc is in the drive.

Once you have learned how to make back-up copies of discs, always do so for valuable programs, and especially for your master CP/M system disc.

Loading Software from Tape (464 only)

Each software tape should come with its own instructions, but you will probably find that you are asked to type something like:

`run "` **[RETURN]**

There is a shorthand way of typing this, which only requires two keystrokes.

1) Press the **[CONTROL]** key, and while holding that key down, press the **[ENTER]** key.

2) Release those keys and then press the **[RETURN]** Key.

You will then be prompted by the message:

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

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 you buy, 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 GAMEPLAY 1 block 1
```

This advises you that the computer has located the first of a series of blocks of the program entitled *'Gameplay'*. 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 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 464 will do the rest.....

Read errors

If you get the message displayed that a `Read error` has occurred while the 464 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 464 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 464. 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.

Part 6: Let's Compute....

So far, we know what we must and mustn't do with the computer, and how to set it up and connect peripherals. We know what some of the keys on the computer do, and how to load software. Now we'll look at some of the instructions that you can type in to make things happen....

Like you or I, the computer can only understand instructions in a language that it knows, and that language is called BASIC, (short for Beginners' All-purpose Symbolic Instruction Code). The words in BASIC's vocabulary are called 'keywords' and each of them tell the computer to perform a specific function. All languages must conform to the rules of grammar, and BASIC is no exception. Here, grammar is referred to as 'Syntax', and the computer will always be kind enough to tell you if you've made a `Syntax error`!

An introduction to AMSTRAD BASIC keywords

In the chapter entitled 'Complete list of AMSTRAD 464/6128 BASIC keywords', you will find a description of all the keywords found in AMSTRAD BASIC. We will introduce some of the more commonly used BASIC keywords in this section.

CLS

To clear the screen, type in:

```
cls [RETURN]
```

You will notice that the screen clears and the word `Ready` with the cursor ■ will appear at the top left of the screen.

Note that you can use upper case (CAPITAL) or lower case (small) letters to enter any BASIC keyword into the computer.

PRINT

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

```
print "hello" [RETURN]
```

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 **[RETURN]** key was pressed. Type in:

```
cls [RETURN]
```

....to clear the screen.

RUN

The previous example showed a single instruction line. However, as soon as the **[RETURN]** key was pressed, the instruction was carried out then forgotten. It is possible to store a series of instructions in the computer to be carried out in a specified order. This is achieved by writing a 'program'. The sort of BASIC instructions that you write in a program are the same as just shown, but in front of each instruction line, a line number is typed in. If there is more than one line in the program, these line numbers tell the computer the order in which to carry out or 'run' the program. When **[RETURN]** is pressed, the line is stored in the memory until the program is run. Now type in:

```
10 print "hello" [RETURN]
```

Notice that when **[RETURN]** was pressed, hello was **NOT** printed on the screen, but instead was entered into the computer's memory as a one-line program. To carry out that program, the word run must be used. Type in:

```
run [RETURN]
```

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

```
10 ? "hello" [RETURN]
```

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 in:

```
list [RETURN]
```

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 **[RETURN]** to clear the screen. Note that although the screen is cleared when you type in: cls **[RETURN],** 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 in:

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

Now type:

```
run [RETURN]
```

....and you will see hello printed continuously on the screen, one under another on the left side. The reason for this, is that line 20 of the program is telling the computer to go to line 10 and carry on processing the program from there.

To pause the running of this program, 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.

Now type in:

```
cls [RETURN]
```

....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 in:

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

Note that the semicolon ; tells the computer to print the next group of characters immediately following the previous one, (unless the next group of characters is too large to fit on the same line).

Escape from this program by pressing **[ESC]** twice. Now type in line 10 again, but this time, use a comma , instead of a semicolon ;

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

You will now see that the comma , has told the computer to print the next group of characters 13 columns away from 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 group of characters will be displaced forwards by another 13 columns, so as to always maintain a space between columns.

This figure of 13 columns is adjustable by use of the ZONE command, described later in this manual.

Again, to escape from this program, press **[ESC]** twice. To clear the computer's memory completely, hold down the **[CONTROL]** **[SHIFT]** 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 "how old are you";age [RETURN]
20 print "you look younger than";age;
   "years old." [RETURN]
run [RETURN]
```

On the screen you will see:

```
what is your age?
```

Type in your age then **[RETURN]**. 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 variable, we could have just as easily used a letter, for example b.

Reset the computer to clear the memory, (**[CONTROL] [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. This type of variable is called a 'string 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$ [RETURN]
20 print "hello ";name$;" my name is Arnold" [RETURN]
run [RETURN]
```

On the screen you will see:

```
What is your name?
```

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

If the name that you entered was Fred, you will see on the screen:

```
Hello Fred my name is Arnold
```

Although we used name$ in the above example 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 **[CONTROL] [SHIFT]** and **[ESC]**. Type in the following:

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

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 **[RETURN]**.

You will then be asked:

```
what is your age?
```

Now type in your age (e.g. 18), then **[RETURN]**.

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 demonstrate this, let's type in the previous program incorrectly:

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

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 main 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 editing cursor method.

Lastly, there is the copy cursor method.

Editing Cursor Method

To correct the mistake in line 5

Type:

```
edit 5 [RETURN]
```

Line 5 will then appear under line `30` with the cursor superimposed over the `c` in `clss`

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

```
list [RETURN]
```

....to check line 5 is now correct.

The AUTO command, described later in this manual, may be used to edit a number of successive lines in a similar manner to that described in the Editing Cursor Method.

Copy Cursor Method

The copy cursor is another cursor (in addition to the one already on the screen) which comes into view when you hold down **[SHIFT]** and press one of the cursor keys. It then detaches itself from the main cursor and can then be moved around the screen independently.

To correct the mistakes in line 10 and 30, hold down the **[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 will appear 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 copied. Press **[RETURN]** 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 the 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 quotation marks 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 copied, then press **[RETURN]**.

You can now list the program to check that it is corrected in the memory by typing in:

```
list [RETURN]
```

NOTE: To move the cursor quickly (during editing) to the left or right hand end of a line, hold down the **[CONTROL]** key then press the left ◊ or right ◊ cursor key once.

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

IF

The IF and THEN commands ask the computer to test for a specified condition, then take action depending upon the result of that test. For example, in the instruction:

```
if 1+1=2 then print "correct" [RETURN]
```

....the computer will test for the specified condition, and then take action accordingly.

The keyword ELSE can be used to inform the IF THEN command as to what alternative action the computer should take if the tested condition is false, for example:

```
if 1+1=0 then print "correct" else print "wrong" [RETURN]
```

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

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

To check this program is correct, type:

```
list [RETURN]
```

Now type in :

```
run [RETURN]
```

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 lines 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 separate 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.

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

FOR and NEXT

The FOR and NEXT commands are used when you want to carry out a specified operation a number of times. The instructions for the specified operation must be enclosed by the FOR NEXT loop.

Type in:

```
5 cls [RETURN]
10 for a=1 to 10 [RETURN]
20 print "operation number";a [RETURN]
30 next a [RETURN]
run [RETURN]
```

You will see that the operation in line 20 has been carried out 10 times, as instructed by the FOR command in line 10. Note also that the value of the variable a is increased by 1 each time.

The keyword STEP may be used to inform the FOR NEXT command how much the variable should be 'stepped' per operation. For example, change line 10 to:

```
10 for a=10 to 50 step 5 [RETURN]
run [RETURN]
```

Negative steps may also be used, for example:

```
10 for a=100 to 0 step -10 [RETURN]
run [RETURN]
```

REM

REM is short for REMark. The instruction tells the computer to ignore anything that follows it on the line. Hence you can use REMarks to inform you of, for example, the title of a program, or the use of a variable, as follows:

```
10 REM Zap the invaders [RETURN]
20 L=5 :REM number of lives [RETURN]
```

The single quote mark ' (which can be typed by holding down **[SHIFT]** and pressing the **7** key) can be used as a substitute for **:REM**. For example:

```
10 'Zap the invaders [RETURN]
20 L=5 'number of lives [RETURN]
```

GOSUB

If there are a set of instructions within a program which are to be carried out a number of times, these instructions need not be typed in repeatedly every time they are needed in the program; instead, they can be made into a 'sub-routine' which when required, can be called into action by the command **GOSUB** followed by the line number. The end of the **GOSUB**-routine is marked by typing in the instruction **RETURN**. At this point the computer will return to the instruction that immediately followed the **GOSUB** command which it had just obeyed.

For example, in the following program:

```
10 a=2
20 PRINT "here is the";a;"times table"
30 FOR b=1 TO 12
40 c=a*b
50 PRINT a;"x";b;"=";c
60 NEXT
70 PRINT
80 '
90 a=5
100 PRINT "here is the";a;"times table"
110 FOR b=1 TO 12
120 c=a*b
130 PRINT a;"x";b;"=";c
140 NEXT
150 PRINT
160 '
170 a=8
180 PRINT "here is the";a;"times table"
190 FOR b=1 TO 12
200 c=a*b
210 PRINT a;"x";b;"=";c
220 NEXT
230 PRINT
240 '
250 a=9
260 PRINT "here is the";a;"times table"
270 FOR b=1 TO 12
280 c=a*b
290 PRINT a;"x";b;"=";c
300 NEXT
310 PRINT
```

....you can see a number of lines which have to be repeated at various points in the program, for example the section from line 260 to 310. Let's make that section into a sub-routine and add the instruction RETURN at the end. Then, we'll call the sub-routine using the command GOSUB 260 whenever we want to use it. The program now looks like this:

```
10 a=2
15 GOSUB 260
80 '
90 a=5
95 GOSUB 260
160 '
170 a=8
175 GOSUB 260
240 '
250 a=9
255 GOSUB 260
256 END
257 '
260 PRINT "here is the";a;"times table"
270 FOR b=1 TO 12
280 c=a*b
290 PRINT a;"x";b;"=";c
300 NEXT
310 PRINT
315 RETURN
```

See how much tedious typing we've saved ourselves? Well designed sub-routines are a principal part of computing. They lead to 'structured' programs, and develop good programming habits.

Always bear in mind when writing sub-routines, that you do not necessarily have to 'jump into' the sub-routine at the same point, i.e. its beginning. A sub-routine written from lines 500 to 800 can be called by: GOSUB 500, or GOSUB 640, or GOSUB 790.

Note in the above program, that the instruction END is used in line 256. Otherwise the program would naturally continue after line 255, and would carry out line 260, which is NOT required unless called by GOSUB.

Simple Arithmetic

Your computer 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 **[RETURN]** key is pressed.

Addition

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

Type:

```
?3+3 [RETURN]
 6
```

(Note that you do NOT type in the equals sign =)

Type:

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

Subtraction

(use unshifted = key for minus)

Type:

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

Type:

```
?8-4 [RETURN]
 4
```

Multiplication

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

Type:

```
?3*3 [RETURN]
 9
```

Type:

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

Division

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

Type:

```
?3/3 [RETURN]
 1
```

Type:

```
?8/4 [RETURN]
 2
```

Integer Division

(use \ for division with removal of the remainder)

Type:

```
?10\6 [RETURN]
 1
```

Type:

```
?20\3 [RETURN]
 6
```

Modulus

(use MOD to obtain the remainder portion after integer division)

Type:

```
?10 MOD 4 [RETURN]
 2
```

Type:

```
?9 MOD 3 [RETURN]
 0
```

Square Root

To find the square root of a number, use `sqr ( )`. The number that you want the square root of should be typed inside the brackets.

Type:

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

Type:

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

Exponentiation

(use unshifted £ key 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 [RETURN] (this means 3^3)
 27
```

Type:

```
?8↑4 [RETURN] (this means 8^4)
 4096
```

Cube Root

You can quite easily calculate cube roots by using a similar method to the last example.

To find the cube root of 27 ($_3\sqrt{27}$)

Type:

```
?27↑(1/3) [RETURN]
 3
```

To find the cube root of 125

Type:

```
?125↑(1/3) [RETURN]
 5
```

Mixed Calculations

(+,-,*,/)

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.

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 [RETURN]
 6.5
```

You can change the way the computer calculated 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 [RETURN]
 14
```

The priority of ALL mathematical operators is as follows:

↑	Exponentiation
MOD	Modulus
-	Unary minus (declares a number as negative)
* and /	Multiplication and division
\	Integer division
+ and -	Addition and Subtraction

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.

You can type in:

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

or you can type:

```
?3E1*1E1 [RETURN]
 300

?3000*1000 [RETURN] ....or.... ?3E3*1E3 [RETURN]
 3000000

?3000*0.001 [RETURN] ....or.... ?3E3*1E-3 [RETURN]
 3
```

Part 7: Save It....

Now that you've exercised your fingers by typing in a few instructions, you'll probably want to know how to save the program. If you have a 464, then you will save it to cassette. On the other hand, if you have a 6128, you will save it to disc. Later on you will then be able to re-load the program any time you want.

Apart from the obvious physical differences, there are a number of operational differences between using a cassette and a disc. A new blank disc cannot just be taken out of its wrapper and used, like a tape can. First it has to be 'formatted'. This process will be described in a moment. And while the disc stores and retrieves information automatically on various portions of the disc surface a cassette has to be positioned and rewound by hand. The computer does automatically start and stop the tape, however.

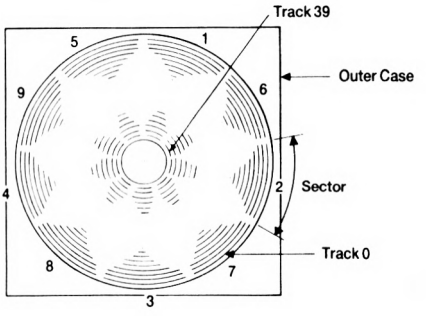
Another point worth mentioning here is the importance of correctly naming the stored files. Cassette file names generally conform to very loose standards, varying greatly in length, and can sometimes be omitted altogether. Disc file names, however, must conform strictly to certain standards, which are described later in this section.

If you have a 464 please turn to the section entitled "Saving a Program in Memory onto cassette".

Formatting discs for use

Before writing any data onto a new blank disc, the disc itself must first be formatted. Formatting can be likened to building a series of shelves and dividers onto a disc prior to the storage of information on those shelves; in other words, laying down an organised framework around which data can be put in or taken out.

Formatting divides the disc into 360 distinctly separate areas:



There are 40 tracks from the outside of the disc (Track 0), to the inside (Track 39), and the circumference of the disc is divided into 9 sectors.

Each track in a sector can store up to 512 bytes of data; hence the total available space on each side of a disc is 180Kbytes.

First Steps Using the Master CP/M System Discs Package

To prepare a new blank disc for reading and writing your own programs onto, you will need to format the new disc. This is carried out using Side 1 of your master CP/M system disc (supplied with the computer).

Switch the system on, and insert Side 1 of the master system disc into the drive.

Type in:

```
|cpm [RETURN]
```

(You will find the bar symbol | by holding down **[SHIFT]** and pressing the @ key.)

After a few seconds you will see the following message at the top of the screen:

```
CP/M Plus   for the 6128. (c)1985 Amstrad plc
```

This is a '`Sign on`' message indicating that the computer is now under the control of the CP/M Plus operating system. You will also see the letter `A>` together with the cursor displayed on the screen. This is a prompt, (similar to `Ready` during normal BASIC operation) indicating that the computer is awaiting your instructions.

Once you are operating CP/M, you cannot enter BASIC commands into the computer, as these will not be understood.

If for example, you type in the BASIC command:

```
cls [RETURN]
```

The Computer will return your entry, together with a question mark:

```
CLS?
```

....indicating that it does not understand your command.

To look briefly at some of the CP/M commands, type in:

```
dir [RETURN]
```

On the screen you will see a **DI**Rectory of CP/M and utility **COM**mands, one of which is **DISCKIT**. Type in:

```
disckit [RETURN]
```

After a few moments, you will see the DISC KIT opening message at the top of the screen, followed by:

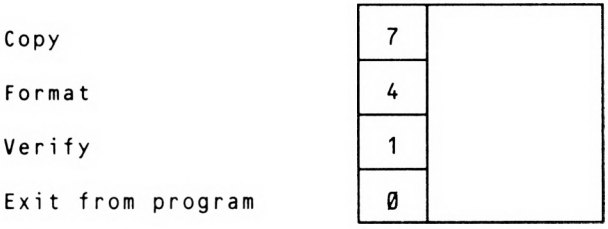
```
One drive found
```

This message tells you that you are running the DISC KIT utility program, and the computer has detected that you are operating only one disc drive (the one within the computer).

If you have connected an additional disc drive to the system, the message will say:

```
Two drives found
```

At the bottom of the screen, you will see the following:



This is known as the main DISC KIT menu. The numbers in boxes refer to the function keys at the right hand end of the keyboard (marked **fØ**, **f1**, **f4**, and **f7**), and pressing one of these keys takes you to your selected menu choice.

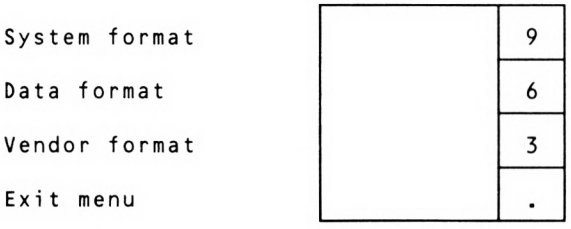
Note that pressing function key number **Ø** at this point will exit from the DISC KIT program back to CP/M Direct Console Mode (the **A>** prompt).

We now want to format a disc, so press function key number 4(**f4**).

BEWARE

FORMATTING A PREVIOUSLY RECORDED DISC WILL ERASE ITS CONTENTS

You will now see a new menu offering you a choice of different formats, namely:



As before, we can now press one of the function keys (**f3**, **f6**, or **f9**) to select the type of format we want. Each of these different formats will be explained later in the manual, but for now, select Data format by pressing function key number 6.

Note that pressing the **.** key (below **f3**, **f6**, and **f9**) exits from the Format mode back to the main DISC KIT menu.

Having pressed function key number 6, (and assuming that you have not connected an additional disc drive to the computer) you will see the message:

```
Y   Format as Data

        Any other key to exit menu
```

At this point, you should remove your master CP/M system disc, and insert the disc that you wish to format. The side of the disc to be formatted should be placed face-up in the drive.

Now press the **Y** key, (Y for Yes, meaning 'go-ahead and format the disc').

The disc will be formatted, tracks 0 to 39; the current track number being indicated at the top left of the screen.

You will not be able to format a disc which has its write protect hole open. Attempting to do so will result in the message:

```
Disc write-protected
Insert disc to format
R-etry or C-ancel
```

....whereupon you should type **C** to Cancel, remove the disc, and insert the correct disc for formatting, with its write protect hole closed.

Make sure that you don't close the write protect hole on a disc that contains programs you want to keep, and NEVER close the write protect holes on your master CP/M system disc package.

When formatting is completed, you will be asked to remove your newly formatted disc from the drive, then to press any key to continue.

Having done so, you will be able to format another disc by inserting it and pressing **Y** again. This may be repeated any number of times until you have formatted all the discs you require in that particular format.

When you have finished formatting, press any key (other than **Y**) to return to the main DISC KIT menu once again.

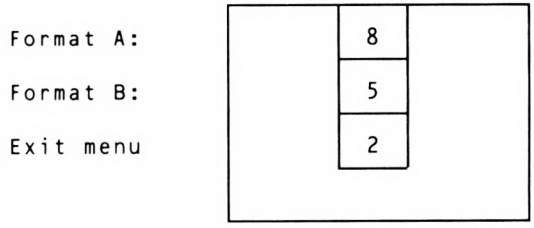
The menu options `Copy` and `Verify` will be dealt with later in this manual, but for now, having learnt to format with CP/M, reset the computer using **[CONTROL] [SHIFT] [ESC]**.

Always keep the master copy of your CP/M system disc in a safe place, as it is literally the key to your system. Later in this manual, you will be shown how to make 'working copies' of your system disc, so that you can keep your master copy safely locked away.

Formatting on a 2-drive system

Follow the instructions above, selecting the `Format` option from the main DISC KIT menu by pressing key **f4**, then select `Data format` by pressing key **f6**.

At this point you will be shown a third menu offering you the choice of drive on which to format:



Choosing the option `Format B:` (key **f5**) will enable you to leave your system disc (Side 1) in Drive A, while you place the disc to be formatted in Drive B.

After selecting `Format B:`, you may then either press **Y** to go ahead with formatting, or any other key to exit to the main DISC KIT menu.

If you have selected the `Format A:` option (key **f8**), you MUST then remember to remove your system disc from Drive A, and instead insert the disc to be formatted.

Don't forget - NEVER RISK OVERWRITING YOUR MASTER CP/M SYSTEM DISC.

Now that we have a formatted blank disc (or two), we can start to manipulate BASIC programs to and from disc.

Saving a Program in Memory onto Disc

Having typed a program into the computer's memory, save it onto disc by typing in:

```
save "filename" [RETURN]
```

Note that the naming of the program is obligatory.

A filename on disc consists of 2 parts (fields). The first part is obligatory and can contain up to 8 characters. Letters and numbers may be used but NO spaces or punctuation marks. This first field usually contains the name of the program.

The second field is optional. You can use up to 3 characters, but again no spaces or punctuation. The 2 fields are separated by a dot `.`

If you do not specify a second field, the system will automatically label it with a token of its own, such as `.BAS` for BASIC files or `.BIN` for binary (machine code) files.

As an example of saving to disc, write a short program into the memory, insert a formatted disc, then type in:

```
save "example" [RETURN]
```

After a few seconds, the prompt `Ready` will appear on the screen, and the program will have been saved onto disc. (If not, check any error message on the screen to establish whether you either forgot to insert your disc into the correct drive, forgot to close the write protect hole, or mistyped the command.)

Catalog

After saving the above program, type in:

```
cat [RETURN]
```

On the screen you will see:

```
Drive A: user 0

EXAMPLE.BAS 1K

177K free
```

The filename will be displayed, including any specified or token second field, together with the file length (to the nearest higher Kbyte). The amount of free space on the disc will also be displayed.

Loading from Disc

Programs may be loaded from disc then run, using the commands:

```
load "filename" [RETURN]
run [RETURN]
```

....or they may be run directly using the command:

```
run "filename" [RETURN]
```

Note that protected programs may be run directly only.

Saving a Program in Memory onto Cassette

Having typed a program into the computer's memory, save it onto cassette by typing:

```
SAVE "FILENAME" [RETURN]
```

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.

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 with the specified filename.

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.

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 disc 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 `CAT`alogue 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..

Supersafe and Speedload

The 464 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
```

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 enforceable.

Advanced SAVEing techniques

There are four ways in which files may be SAVEd by the 464/6128. In addition to ordinary BASIC file saving, by:

```
save "filename" [RETURN]
```

....there are three alternative methods, for more specialised purposes:

ASCII Files

```
save "filename",a [RETURN]
```

Adding the suffix ,a instructs the computer to save the program or data in the form of an ASCII text file. 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.

Protected Files

```
save "filename",p [RETURN]
```

Adding the suffix ,p tells the computer to protect the data so that the program cannot be LISTed after LOADing it, or RUNning it then stopping its execution using the **[ESC]** key function.

Programs saved in this way can only be run directly, using the commands:

```
run "filename" [RETURN]
```

....or....

```
chain "filename" [RETURN]
```

If you anticipate wanting to edit or alter the program, you should also keep a copy for yourself in unprotected form, i.e. without the ,p suffix.

Binary Files

```
save "filename",b, ‹starting address› , ‹length in bytes›
                                    [ , ‹optional entry point› ] [RETURN]
```

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

Screen Dump

This binary save feature allows data from the screen memory to be stored directly onto disc or cassette in the form of a screen dump. The contents of the screen will be saved exactly as it is seen, using the command:

```
save "scrndump",b,49152,16384 [RETURN]
```

....where `49152` is the starting address of the screen memory, and `16384` is the length of the screen memory that you wish to save.

To call it back onto the screen, type in:

```
load "scrndump" [RETURN]
```

Copying Programs from Disc to Disc

Using the commands already learnt in this section, it can be seen that disc to disc or tape to tape program copying is performed simply by loading the program into the memory from the original (source) disc or tape, removing the source disc or tape, and saving the program onto the new (destination) disc or tape.

More information on using the system to manipulate program files between discs (and cassette), will be found later in this manual.

Part 8: Understanding Modes, Colours and Graphics....

The Amstrad 464/6128 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 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 **[RETURN]** - 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 [RETURN]
```

You will see that 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 **[RETURN]** again.

Now type in:

```
mode 2 [RETURN]
```

You will see that this is the smallest mode, and if you again type in a line 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 **[RETURN]** once again.

Colours

There is a choice of 27 colours. These are shown on a mono monitor (MM12) as various shades of grey.

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

For convenience, this table also appears on the panel at the top right hand side of the computer.

MASTER COLOUR CHART

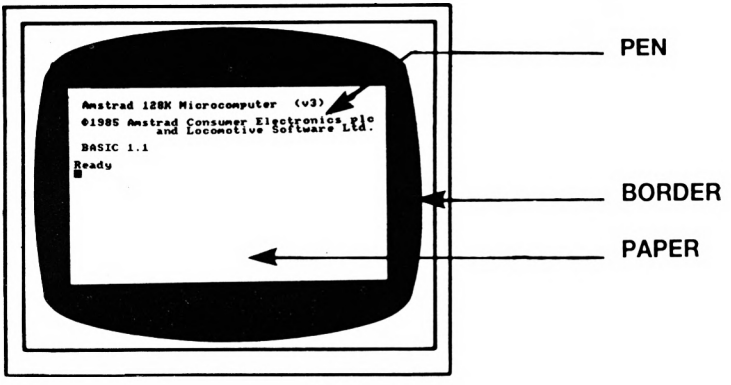
Ink No.	Colour/Ink	Ink No.	Colour/Ink	Ink No.	Colour/Ink
0	Black	9	Green	18	Bright Green
1	Blue	10	Cyan	19	Sea Green
2	Bright Blue	11	Sky Blue	20	Bright Cyan
3	Red	12	Yellow	21	Lime Green
4	Magenta	13	White	22	Pastel Green
5	Mauve	14	Pastel Blue	23	Pastel Cyan
6	Bright Red	15	Orange	24	Bright Yellow
7	Purple	16	Pink	25	Pastel Yellow
8	Bright Magenta	17	Pastel Magenta	26	Bright White

Table 1: 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 from a different mode, type in:

```
mode 1 [RETURN]
```

The screen display



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 characters on the screen can only appear inside the border. The PAPER is the background area behind a character, while the PEN writes the character itself.

Now we will explain how the colours that you see on the screen are selected, and how you can change these to your own choice.

When you first switch on, or reset the computer, the BORDER is always set to colour number 1. Look up number 1 on the master colour chart, and you will see that colour number 1 is blue. The colour of the border can be changed by using the command: BORDER followed by the colour number. To change the border to white, type in:

```
border 13 [RETURN]
```

So far, so good. Now for the tricky bit....

When you first switch on, or reset the computer, PAPER number 0, and PEN number 1 are always automatically selected. This does NOT mean that you look up numbers 0 and 1 on the master colour chart and away you go....

The important thing to remember is that 0 and 1 are PAPER and PEN numbers; they are NOT ink colour numbers. To understand this, imagine having 4 pens on your desk numbered 0,1,2, and 3, and being able to fill each of these pens with any colour of your choice out of 27 bottles of ink, numbered 0 to 26. It can be seen therefore, that pen number 1 is not necessarily always the same colour; as it can be filled with a different ink, in fact you could fill each of the 4 pens with the same ink.

So it is with the computer. Using the **PEN** and **INK** commands, you can select the pen number, and then the ink colour for that pen.

Remembering that we are operating in Mode 1 (40 columns), look at Table 2 below, and you will see from the first and third columns, that **PEN** number 1 corresponds to **INK** colour number 24. Now look up **INK** number 24 on the master colour chart (Table 1) and you will see that the colour listed is bright yellow, i.e. the colour of the characters on the screen when you first switch on.

DEFAULT SETTINGS

Paper/Pen No.	Ink Colour Mode 0	Ink Colour Mode 1	Ink Colour 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 2: PAPER/PEN/MODE/INK reference

The PAPER/PEN/INK relationships given in Table 2 are not fixed however. They are the default settings when you switch-on or reset the computer. You can change them by using the INK command. The command has two parts (or 'parameters'). The first part is the number of the PAPER or PEN that you are going to fill with ink, and the second part is the colour of the ink itself. The two parts of the command are separated by a comma ,

So, now that we know that we are using PEN number 1, let's change the colour of the INK in that pen, to orange.

Type in:

```
ink 1,15 [RETURN]
```

....and you will see that the characters on the screen have changed colour.

The background colour can also be changed using the INK command. We know that PAPER number Ø is selected at switch on, so let's change the colour of the INK in PAPER number Ø to green (colour number 9) by typing in the command:

```
ink Ø,9 [RETURN]
```

Now let's use a different pen altogether. Type in:

```
pen 3 [RETURN]
```

Notice how only the colour of the new characters (after the command) changes. You're using PEN number 3 now, and you will see from Tables 1 and 2 that the INK initially loaded into PEN number 3 is colour number 6 (bright red). Change it to pink by typing in:

```
ink 3,16 [RETURN]
```

Remember, 3 is the colour of the pen that you selected earlier with the command: pen 3, and that 16 is the ink colour - pink.

Now let's change to a new paper. When a new paper is selected, the previous background colour behind the characters will NOT change because that colour was 'printed' by a different PAPER. To see this, type in:

```
paper 2 [RETURN]
```

Once again use Tables 1 and 2 to see why the background colour for PAPER number 2 is bright cyan. Change it to black by typing in:

```
ink 2,0 [RETURN]
```

On the screen now, we have characters written by PEN numbers 1 and 3, on a background of PAPER numbers 0 and 2. INKs can be changed in a PEN or PAPER that you are not currently using. For example, type in:

```
ink 1,2 [RETURN]
```

....which changes the colour of all the previous characters typed in with PEN number 1.

Type in:

```
cls [RETURN]
```

....to clear the screen.

It should now be possible for you to instruct the computer to return to its original colours (blue border and background with bright yellow characters) using the BORDER, PAPER, PEN, and INK commands. See if you can do so. If you can't, then reset the computer using the **[CONTROL] [SHIFT]** and **[ESC]** keys.

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, reset the computer using **[CONTROL] [SHIFT] [ESC]**, and type in:

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

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 [RETURN]
```

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 using **[CONTROL] [SHIFT] [ESC]**

Note from Table 2 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, their INK commands have been pre-programmed with an extra colour parameter.

Type in the following:

```
mode 0 [RETURN]
pen 15 [RETURN]
```

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

Now type in:

```
paper 14 [RETURN]
cls [RETURN]
```

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.

PEN and PAPER numbers 14 and 15 may be re-programmed using the INK command to other flashing colours, or to one steady colour.

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 [RETURN]
```

You will now see that the BORDER is flashing between bright red and green. Note that the border may be set to any one, or pair of the 27 colours, regardless of whether you are operating in mode 0, 1, or 2.

Now reset the computer using **[CONTROL] [SHIFT] [ESC]**

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

```
10 MODE 0 [RETURN]
20 rate=600: REM sets speed of program [RETURN]
30 FOR b=0 TO 26 [RETURN]
40 LOCATE 3,12 [RETURN]
50 BORDER b [RETURN]
60 PRINT "border colour";b [RETURN]
70 FOR t=1 TO rate [RETURN]
80 NEXT t,b [RETURN]
90 CLG [RETURN]
100 FOR p=0 TO 15 [RETURN]
110 PAPER p [RETURN]
120 PRINT "paper";p [RETURN]
130 FOR n=0 TO 15 [RETURN]
140 PEN n [RETURN]
150 PRINT "pen";n [RETURN]
160 NEXT n [RETURN]
170 FOR t=1 TO rate*2 [RETURN]
180 NEXT t,p [RETURN]
190 MODE 1 [RETURN]
200 BORDER 1 [RETURN]
210 PAPER 0 [RETURN]
220 PEN 1 [RETURN]
230 INK 0,1 [RETURN]
240 INK 1,24 [RETURN]
run [RETURN]
```

IMPORTANT

In the above program, and in later chapters and listings in this manual, BASIC keywords will appear in upper case (CAPITAL) letters. This is how keywords appear when a program is LISTed by the computer. In general it is preferable that you type instructions or programs using lower case (small) letters, since it will help you spot typing mistakes when LISTing the program - (because the mis-typed BASIC keyword will NOT be converted to upper case).

For the remainder of this Foundation course, we will list programs in both upper and lower case, so that you get accustomed to this aspect of operation.

A variable's name, such as x or a$, will NOT be converted to upper case when the program is LISTed although the computer will recognise the name regardless of whether it appears in upper or lower case in the program.

Attention

From this point in the manual, you will not be instructed to press the **[RETURN]** key after each line. Therefore it is assumed that you will do it automatically.

Graphics

There are a number of character symbols in the computer's memory. To print any one of these, we use the keyword:

```
chr$( )
```

Inside the brackets should be the symbol number, which is in the range from 32 to 255.

Reset the computer, **[CONTROL] [SHIFT] [ESC]**, then type in:

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

Don't forget to press **[RETURN]**. 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 together with their associated numbers, type in the following program, once again remembering to press **[RETURN]** after each line.

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

For your reference, the range of characters together with their respective numbers appear in the chapter entitled 'For your reference....'.

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. Therefore, 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 **[RETURN]** 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:

```
10 CLS
20 FOR x=1 TO 39
30 LOCATE x,20
50 PRINT CHR$(250)
60 NEXT x
70 GOTO 10
run
```

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

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

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

(This new line 50 automatically replaces the previously typed in line 50.)

Now type in:

```
run
```

FRAME

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

```
40 frame
```

The FRAME command synchronises the movement of objects on the screen to the display frame scanning frequency. If that's a bit technical for you, just remember that the command should be used whenever you want to move characters or graphics around the screen smoothly.

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:

```
70 FOR n=1 TO 300:NEXT n
80 FOR x=39 TO 1 STEP -1
90 LOCATE x,20
100 FRAME
110 PRINT CHR$(251);" "
120 NEXT x
130 FOR n=1 TO 300:NEXT n
140 GOTO 20
run
```

PLOT

Unlike the LOCATE command, PLOT may be used to determine the position of the graphics cursor, using pixel co-ordinates. (A pixel is the smallest possible segment of the screen).

Note that the graphics cursor is not visible and is different from 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 between modes 0,1, or 2.

To see this, first reset the computer using **[CONTROL] [SHIFT] [ESC]**, then type in:

```
plot 320,200
```

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

Now change the mode by typing in:

```
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:

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

The dot is still in the centre, but 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 **[CONTROL] [SHIFT] [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 there to the right corner etc, 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 60
run
```

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

(Notice line 60 of this program; the computer is told to 'loop' around line 60 again and again, until you 'break' from the program by pressing **[ESC]** twice. This sort of instruction is useful if you do not want the computer to automatically 'break' at the end of a program and display the Ready prompt that usually appears on the screen.)

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
110 goto 110
run
```

Again, press **[ESC]** twice to break from this program.

MOVE

The MOVE command operates in a similar manner to PLOT, in that the graphics cursor is moved to the position specified by the x,y co-ordinates; however the pixel (dot) at the new graphics cursor location is NOT plotted.

Type in:

```
cls
move 639,399
```

Although we can see no sign of it on the screen, we have moved the graphics cursor to the top right corner.

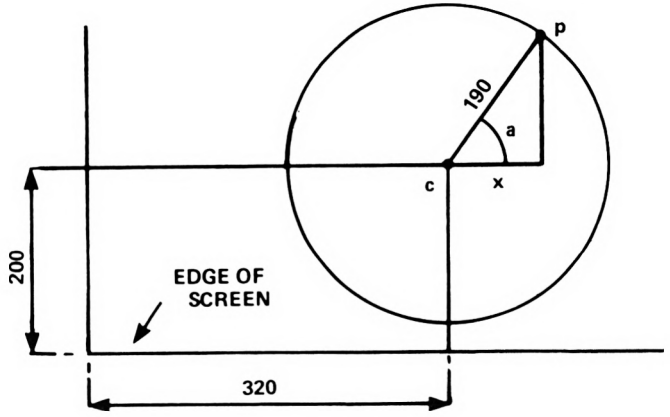
Let's prove it by drawing a line from that position to the centre of the screen, by typing in:

```
draw 320,200
```

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 diagram 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)
y=190*sin(a)
```



Plotting the points of a circle.

In previous programs we have plotted points with respect to the bottom left corner of the screen. If we 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 plot a circle would then be like this. Type in:

```
new
10 CLS
20 DEG
30 FOR a=1 TO 360
40 MOVE 320,200
50 DRAW 320+190*COS(a),200+190*SIN(a)
60 NEXT
run
```

Note the use of the keyword **NEW** before typing in this program. This tells the computer to clear any program in the memory (in a similar manner to **[CONTROL] [SHIFT] [ESC]**). However the screen itself is not cleared.

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 **20** from the program by typing in:

```
20
```

To see a solid circle drawn by lines from the centre, edit line **50**, replacing the word **plot** with the word **draw**. (Line **50** will then be):

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

Try this with and without line **20** again.

You will note that line **60** of this program is **NEXT** instead of **NEXT a**. It is permissible to simply type '**NEXT**' on its own; 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 however, you may wish to add the variable's name after the word **NEXT** in order to identify the **NEXT** statement when studying the program.

ORIGIN

In the previous program we used the **MOVE** command to establish 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 each of the x,y co-ordinates relative to the **ORIGIN**. To see this, type in:

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

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

```
new
10 CLS
20 FOR a=1 TO 360
30 ORIGIN 196,282
40 PLOT 50*COS(a),50*SIN(a)
50 ORIGIN 442,282
60 PLOT 50*COS(a),50*SIN(a)
70 ORIGIN 196,116
80 PLOT 50*COS(a),50*SIN(a)
90 ORIGIN 442,116
100 PLOT 50*COS(a),50*SIN(a)
110 NEXT
run
```

To see a different way of creating a circle, type in the following program:

```
new
10 MODE 1
20 ORIGIN 320,200
30 DEG
40 MOVE 0,190
50 FOR a=0 TO 360 STEP 10
60 DRAW 190*SIN(a),190*COS(a)
70 NEXT
run
```

This time a line is **DRAW**n from co-ordinate to co-ordinate around the circumference of the circle. Note how the circle is drawn much quicker than it is plotted.

Once again observe the effect of removing the **DEG** command, by deleting line **30**, then **RUN**ning the program again.

FILL

The **FILL** command is used to fill an area of the screen which is enclosed by drawn or edge of screen/graphics window boundaries.

Reset the computer, **[CONTROL] [SHIFT] [ESC]**, then type in:

```
new
10 cls
20 move 20,20
30 draw 620,20
40 draw 310,380
50 draw 20,20
run
```

On the screen you will see a triangle. Move the graphics cursor to the centre of the screen by typing in:

```
move 320,200
```

Using the keyword FILL followed by a pen number, for example 3, we will now FILL the screen using the specified pen, from the current graphics cursor position (centre screen) to the drawn boundaries. Type in:

```
fill 3
```

Now move the graphics cursor outside the triangle by typing in:

```
move 0,0
```

See what happens when you type in:

```
fill 2
```

The computer has used pen number 2 to FILL the area bounded by the drawn lines and by the edges of the screen.

Now alter the program by typing in the following lines, and see what happens:

```
50 draw 50,50
60 move 320,200
70 fill 3
run
```

You will note that any gaps in the drawn boundaries let the ink from the pen 'seep' through!

This point is further demonstrated by FILLing first a plotted circle, then a drawn circle. Type in:

```
new
10 CLS
20 FOR a=1 TO 360
30 ORIGIN 320,200
40 PLOT 190*COS(a),190*SIN(a)
50 NEXT
60 MOVE -188,0
70 FILL 3
run
```

Now try:

```
new
10 MODE 1
20 ORIGIN 320,200
30 DEG
40 MOVE 0,190
50 FOR d=0 TO 360 STEP 10
60 DRAW 190*SIN(d),190*COS(d)
70 NEXT
80 MOVE -188,0
90 FILL 3
run
```

We can make the outline of the circle to be filled invisible, by setting the pen ink to the same colour as the paper ink. Add:

```
45 GRAPHICS PEN 2:INK 2,1
run
```

The **GRAPHICS PEN** command selects the pen to be used for drawing graphics on the screen. The **INK** command then specifies the ink colour for that pen, which in this case, is the same as for the paper (i.e. colour number **1**).

Finally, type in this demonstration program:

```
new
10 MODE 0:BORDER 13
20 MOVE 0,200:DRAW 640,200
30 FOR x=80 TO 560 STEP 80
40 MOVE x,0:DRAW x,400
50 NEXT:MOVE -40,300
60 FOR c=0 TO 7
70 MOVER 80,0:FILL c
80 MOVER 0,-200:FILL c+8
90 MOVER 0,200:NEXT
100 GOTO 100
run
```

The colours of the filled areas can be changed after the fill. Type in:

```
100 SPEED INK 30,30
110 BORDER RND*26,RND*26
120 INK RND*15,RND*26,RND*26
130 FOR t=1 TO 500:NEXT:GOTO 110
run
```

Further details....

For a more comprehensive guide to graphics on the 464/6128, see the section 'Graphically speaking' in the chapter entitled 'At your leisure....'.

To conclude this section, here are a few graphics demonstration programs which incorporate a lot of the programming commands and keywords that you should now understand. Each of the programs draws continuous patterns on the screen.

```
new
10 BORDER 0:GRAPHICS PEN 1
20 m=CINT(RND*2):MODE m
30 i1=RND*26:i2=RND*26
40 IF ABS(i1-i2)<10 THEN 30
50 INK 0,i1:INK 1,i2
60 s=RND*5+3:ORIGIN 320,-100
70 FOR x= -1000 TO 0 STEP s
80 MOVE 0,0:DRAW x,300:DRAW 0,600
90 MOVE 0,0:DRAW -x,300:DRAW 0,600
100 NEXT:FOR t=1 TO 2000:NEXT:GOTO 20
run
```

```
10 MODE 1:BORDER 0:PAPER 0
20 GRAPHICS PEN 2:INK 0,0:i=14
30 EVERY 2200 GOSUB 150
40 flag=0:CLG
50 INK 2,14+RND*12
60 b%=RND*5+1
70 c%=RND*5+1
80 ORIGIN 320,200
90 FOR a=0 TO 1000 STEP PI/30
100 x%=100*COS(a)
110 MOVE x%,y%
120 DRAW 200*COS(a/b%),200*SIN(a/c%)
130 IF flag=1 THEN 40
140 NEXT
150 flag=1:RETURN
run
```

```
10 MODE 1:BORDER 0:DEG
20 PRINT "Please wait"
30 FOR n=1 TO 3
40 INK 0,0:INK 1,26:INK 2,6:INK 3,18
50 IF n=1 THEN sa=120
60 IF n=2 THEN sa=135
70 IF n=3 THEN sa=150
80 IF n=1 THEN ORIGIN 0,-50,0,640,0,400 ELSE ORIGIN
   0,0,0,640,0,400
90 DIM cx(5),cy(5),r(5),lc(5)
100 DIM np(5)
110 DIM px%(5,81),py%(5,81)
120 st=1:cx(1)=320:cy(1)=200:r(1)=80
130 FOR st=1 TO 4
140 r(st+1)=r(st)/2
150 NEXT st
160 FOR st=1 TO 5
170 lc(st)=0:np(st)=0
180 np(st)=np(st)+1
190 px%(st,np(st))=r(st)*SIN(lc(st))
200 py%(st,np(st))=r(st)*COS(lc(st))
210 lc(st)=lc(st)+360/r(st)
220 IF lc(st) < 360 THEN 180
230 px%(st,np(st)+1)=px%(st,1)
240 py%(st,np(st)+1)=py%(st,1)
250 NEXT st
260 CLS:cj=REMAIN(1):cj=REMAIN(2)
270 cj=REMAIN(3):INK 1,2:st=1
280 GOSUB 350
290 LOCATE 1,1
300 EVERY 25,1 GOSUB 510
310 EVERY 15,2 GOSUB 550
320 EVERY 5,3 GOSUB 590
330 ERASE cx,cy,r,lc,np,px%,py%:NEXT
340 GOTO 340
350 cx%=cx(st):cy%=cy(st):lc(st)=0
360 FOR x%=1 TO np(st)
370 MOVE cx%,cy%
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))
```

This program continues on the next page

```
440 cy(st+1)=cy(st)+1.5*r(st)*COS(sa+lc(st))
450 st=st+1
460 GOSUB 350
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
```

Part 9: Using Sound....

Sound is generated by a pair of stereo loudspeakers within the monitor.

The level of sound can be adjusted by use of the **VOLUME** control at the front of the monitor. The sound can also be fed to the auxiliary input socket of your stereo system, using the socket provided on the side of the computer. Instructions on connecting to the computer's STEREO socket will be found in part 2 of this Foundation course.

The SOUND Command

The SOUND command has 7 parts ('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›

It looks pretty complicated, but if we analyse each parameter, we can soon get to grips with it. Let's look at the parameters one by one....

Channel Status

To keep things simple at the moment, regard the ‹channel status› as the reference number for the sound channel. There are 3 sound channels, and for now we will use the ‹channel status› number 1 (Channel 1 = Left, 3 = Right, 2 = Mixture).

Tone Period

‹Tone period› is a technical way of defining the pitch of the sound, or in other words, 'what note it is' (i.e. do re mi fa so, etc). Each note has a set number, and this number is the ‹tone period›. Refer to the chapter entitled 'For your reference....', and you will see that the note middle c (do), has a tone period of 239.

Now reset the computer **[CONTROL] [SHIFT] [ESC]**, and type in:

```
10 sound 1,239
run
```

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

If you don't hear anything, make sure that the **VOLUME** control on the computer is not set to zero, then type **RUN** again.

Duration

This parameter sets the length of the sound, in other words, 'how long it lasts'. The parameter works in units of 0.01 (one hundredth) of a second, and if you don't specify the ‹duration›, the computer will assume a figure of 20, which is why the note you just heard lasted 0.2 second, i.e. 0.01 multiplied by 20.

To make the note last for 1 second, a ‹duration› of 100 would be be used; to make it last 2 seconds, 200 would be used. Type in:

```
10 sound 1,239,200
run
```

You will hear the note middle c lasting 2 seconds.

Volume

This parameter specifies the starting volume of a note. The number is in the range 0 to 15. A ‹volume› figure of 0 is minimum, while 15 is maximum. If no number is used, 12 is assumed. Type in:

```
10 sound 1,239,200,5
run
```

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

```
10 sound 1,239,200,15
run
```

You will hear that this is much louder.

Volume Envelope

To make the volume vary within the duration of the note, you can specify a volume envelope using the separate command ENV. You can in fact make a number of different volume envelopes, and like the SOUND command, each has its own reference number. If you have created a volume envelope with a reference number of 1, and you wish to use it in a SOUND command, then where the parameter ‹volume envelope› is required, type in 1. Creating a volume envelope will be explained shortly.

Tone Envelope

To make the tone or pitch vary within the duration of the note, you can specify a tone envelope using the separate command ENT. You can in fact make a number of different tone envelopes, and like the SOUND command, each has its own reference number. If you have created a tone envelope with a reference number of 1, and you wish to use it in a SOUND command, then where the parameter ‹tone envelope› is required, type in 1. Creating a tone envelope will be explained shortly.

Noise

‹Noise› is the last parameter of the SOUND command. A range of noise is available by varying the ‹noise› parameter between 1 and 31. Add a ‹noise› parameter of 2 at the end of the SOUND command and listen to the effect. Then change the ‹noise› parameter to 27 and listen to the difference. Type in:

```
10 sound 1,239,200,15,,,2
```

Note the two 'blank' parameters (,,,) before the ‹noise› parameter of 2. This is because we haven't created a ‹volume envelope› nor a ‹tone envelope›.

Creating a Volume Envelope

The volume envelope command is ENV. In its simplest form, the command has 4 parameters: The command is typed in as:

ENV ‹envelope number› , ‹number of steps› , ‹size of step› , ‹time per step›

As before, let's look at the parameters one by one.

Envelope Number

This is the reference number (between 0 and 15) given to a particular volume envelope so that it can be called up in the SOUND command.

Number of Steps

This parameter specifies how many different steps of volume you want the sound to pass through before it ends. For example, in a note which lasts 10 seconds, you may wish to have 10 volume steps of 1 second each. In such a case, the ‹number of steps› parameter used should be 10.

The available range of ‹number of steps› is 0 to 127.

Size of Step

Each step can vary in size from a volume level of 0 to 15 with respect to the previous step. The 16 different volume levels are the same as those you will hear in the SOUND command. However, the ‹size of step› parameter used can be between -128 and +127; the volume level re-cycling to 0 after each 15.

Time per Step

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of ‹time per step› numbers is 0 to 255, which means that the longest time between steps is 2.56 seconds (0 is treated as 256).

Note therefore, that the ‹number of steps› parameter multiplied by the ‹time per step› parameter shouldn't be greater than the ‹duration› parameter in the SOUND command, otherwise the sound will finish before all the volume steps have been passed through.

(In such a case, the remaining contents of the volume envelope are discarded.)

Likewise, if the ‹duration› parameter in the SOUND command is longer than the ‹number of steps› multiplied by the ‹time per step›, the sound will continue after all of the volume steps have been passed through, and will remain constant at the final level.

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

```
10 env 1,10,1,100
20 sound 1,142,1000,1,1
run
```

Line 20 specifies a SOUND with a tone period of 142 (international a), lasting for 10 seconds with a start volume of 1, and using volume envelope number 1, conisting of 10 steps, raising the volume of each step by 1, every 1 second (100 x 0.01 second).

Change line 10 in each of the following ways and then run to hear the effect of changing the envelope:

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

And finally try this:

```
10 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 volume varied was only 5 seconds, but the ‹duration› parameter in the SOUND command in line 20 was 1000, i.e. 10 seconds.

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

If you wish to create a more intricate volume envelope, the 3 parameters: ‹number of steps› , ‹size of step› , ‹time per step› may be repeated at the end of the ENV command up to 4 more times, to specify a different 'section' of the same envelope.

Creating a Tone Envelope

The tone envelope command is ENT. In its simplest form, the command has 4 parameters: The command is typed in as:

ENT ‹envelope number› , ‹number of steps› , ‹tone period of step› , ‹time per step›

Once again, let's look at the parameters one by one.

Envelope Number

This is the reference number (between 0 and 15) given to a particular tone envelope so that it can be called up in the SOUND command.

Number of Steps

This parameter specifies how many different steps of tone (pitch) you want the sound to pass through before it ends. For example, in a note which lasts 10 seconds, you may wish to have 10 tone steps of 1 second each. In such a case, the ‹number of steps› parameter used should be 10.

The available range of ‹number of steps› is 0 to 239.

Tone Period of Step

Each tone step can vary in the range -128 to +127 with respect to the previous step. Negative steps make the pitch of the note higher; positive steps make the pitch of the note lower. The shortest tone period is 0. This must be remembered when formulating the tone envelope. The full range of tone periods is shown in the chapter entitled 'For your reference....'.

Time per Step

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of ‹time per step› numbers is 0 to 255, which means that the longest time between steps is 2.56 seconds (0 is treated as 256).

Note therefore, that the ‹number of steps› parameter multiplied by the ‹time per step› parameter shouldn't be greater than the ‹duration› parameter in the SOUND command, otherwise the sound will finish before all the tone steps have been passed through. (In such a case, the remaining contents of the tone envelope are discarded.)

Likewise, if the ‹duration› parameter in the SOUND command is longer than the ‹number of steps› multiplied by the ‹time per step›, the sound will continue after all of the tone steps have been passed through, and will remain constant at the final tone pitch.

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

```
10 ent 1,100,2,2
20 sound 1,142,200,15,,1
run
```

Line 20 specifies a SOUND with a tone period of 142 (international a) lasting for 2 seconds with a start volume of 15 (max), without a volume envelope (represented by a blank parameter ,,) and with tone envelope number 1.

Line 10 is tone envelope number 1 consisting of 100 steps, increasing the tone period (reducing the pitch) by 2, every 0.02 second (2 x 0.01 second).

Now change line 10 in each of the following ways, and then **RUN** to hear the effect of changing the tone envelope:

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

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

```
10 ent 1,2,17,70
20 sound 1,71,140,15,,1
30 goto 10
run
```

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

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

```
10 env 1,100,1,3
20 ent 1,100,5,3
30 sound 1,142,300,1,1,1
run
```

Then replace line 20 by typing in:

```
20 ent 1,100,-2,3
run
```

Now replace all the lines by typing in:

```
10 env 1,100,2,2
20 ent 1,100,-2,2
30 sound 1,142,200,1,1,1
run
```

If you wish to create a more intricate tone envelope, the 3 parameters: ‹number of steps› , ‹tone period of step› , ‹time per step› may be repeated at the end of the **ENT** command up to 4 more times, to specify a different 'section' of the same envelope.

Try some more variations for yourself. Add some ‹noise› to the **SOUND** command, and try adding some extra sections to the volume and tone envelopes.

The chapter entitled 'Complete list of AMSTRAD 464/6128 BASIC Keywords' contains full details of the various sound commands. If you are interested in the more melodius aspects of sound, see the section, 'The Sound of Music' which you will find in the chapter entitled 'At your leisure....'.

Part 10: Introducing AMSDOS and CP/M (6128 only)....

What is AMSDOS?....

AMSDOS is an abbreviation of AMStrad Disc Operating System, and offers the following file handling commands and functions; modified to use the disc drive of the 6128, instead of the cassette drive of the 464.

```
LOAD "filename"
RUN "filename"
SAVE "filename"
CHAIN "filename"
MERGE "filename"
CHAIN MERGE "filename"
OPENIN "filename"
OPENOUT "filename"
CLOSEIN
CLOSEOUT
CAT
EOF
INPUT #9
LINE INPUT #9
LIST #9
PRINT #9
WRITE #9
```

Copying a Whole Disc

The entire contents of a disc can be copied from one to another using the `DISCKIT` program on Side 1 of the CP/M system disc.

You may use this method to make back-up copies of the master system disc itself.

Insert Side 1 of the system disc into the computer's disc drive, and type:

```
|cpm
```

After the `A>` prompt, type:

```
diskit
```

After a few seconds, you will see the DISC KIT opening message at the top of the screen, followed by:

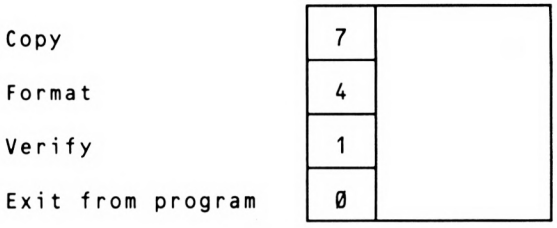
```
One drive found
```

This message tells you that you are running the DISC KIT utility program, and the computer has detected that you are operating only one disc drive (the one within the computer).

If you have connected an additional disc drive to the system, the message will say:

```
Two drives found
```

At the bottom of the screen, you will see the following:



This is known as the main DISC KIT menu. The numbers in boxes refer to the function keys at the right hand end of the keyboard (marked **fØ**, **f1**, **f4**, and **f7**), and pressing one of these keys takes you to your selected menu choice.

Note that pressing function key number **Ø** at this point will exit from the DISC KIT program back to CP/M Direct Console Mode (the **A>** prompt).

We now want to copy a disc, so press function key number 7 (**f7**).

BEWARE

COPYING ONTO A PREVIOUSLY RECORDED DISC WILL ERASE ITS CONTENTS

Copying on a 1-drive system

Assuming that you are using a 1-drive system (i.e. you have NOT connected an additional drive), you will then see the message:


```
Copy

        Any other key to exit menu
```

At this point, you should remove your master CP/M system disc, and insert the disc that you wish to copy from. If you wish to copy the CP/M system disc itself, simply leave it in the drive.

When the disc that you wish to copy from is in the drive, press the **Y** key, (Y is for Yes, meaning 'go-ahead and copy the disc').

The format of the disc will be examined by the computer, and displayed at the top of the screen for your information.

After a while, you will see the message:

```
Insert disc to WRITE
Press any key to continue
```

At this point, you should take the disc that you are copying out of the drive. Then insert the disc for copying onto, and press any key.

Information about the format of the disc to be copied onto (even if it is a new unformatted disc) will be indicated at the top of the screen.

If the disc that you wish to copy onto is not correctly formatted (or not formatted at all), this will be taken care of during copying, and you will see a message such as:

```
Disc isn't formatted (or faulty)
Going to format while copying
Disc will be system format
```

....or some other similar message, depending upon the discs that you are copying between.

When the computer is again ready for the disc to copy from, it will display the message:

```
Insert disc to READ
Press any key to continue
```

....and you should re-insert the disc to copy from.

After repeating this process a few times, the contents of the first disc will be copied onto the second disc, and the message:

```
Copy completed
Remove disc
Press any key to continue
```

....will appear, whereupon you should follow the instructions on the screen, which give you the option to either copy another disc (by typing `Y`) or 'exit'ing to the main DISC KIT menu.

Write protection

Note that you will not be able to copy onto a disc which has its write protect hole open. Attempting to do so will result in the message:

```
Disc write-protected
Insert disc to WRITE
R-etry or C-ancel
```

....whereupon you should type `C` to Cancel, remove the disc, and insert the correct disc for copying onto, with its write protect hole closed.

Make sure that you don't close the write protect hole on a disc that contains programs you want to keep, and NEVER close the write protect holes on your master CP/M system disc.

Verifying discs

The `DISCKIT` program also provides you with the facility to verify (check) a disc.

Information about the format of the disc is displayed, and all files on the disc are read. Any errors present in the files are reported on the screen.

To verify a disc, insert Side 1 of your system disc, and type:

```
Icpm
```

At the `A>` prompt, type:

```
disckit
```

Select the `Verify` option from the main DISC KIT menu (key **f1**), and then follow the instructions on the screen.

When the disc that you wish to verify is inserted in the drive, type `Y` to commence verifying.

When verifying is completed, the following message will be displayed:

```
Verify completed
Remove disc
Press any key to continue
```

You may then repeat the verifying process by typing `Y`, or exit to the main DISC KIT menu by pressing any other key.

Part 11: Introducing the Bank Manager (6128 only)....

Using the second 64K of memory

The 6128 contains 128K of RAM (Random Access Memory) in two lots of 64K. CP/M Plus uses the full 128K all the time, but BASIC does not normally use the second 64K - only the available memory in the first 64K. It would be a pity to leave the extra 64K completely unused when programming in BASIC, so a program has been provided to make special use of this extra memory. The program offers some extra commands that make it possible to use the second 64K of RAM as either a storage space for screen images, or as a storage space for strings.

The program providing these extra commands is known as the 'BANK MANAGER' where 'bank' is a technical term used to describe a section of memory.

Using the BANK MANAGER for screen images

The 6128 is displaying a screen image all the time. To do this, it requires 16K of memory in which to store the information about the colour and brightness of every pixel (dot) on the screen. The 6128's memory allows up to six screen images (each in a 16K block) to be present in the computer's memory at any one time. The BANK MANAGER provides the facilities for you to juggle and display up to five of the six possible screens from BASIC.

When you first switch on, the screen is displayed from a 16K block of memory (which we will call 'Block 1') out of the first 64K. The other four screens are held in the second 64K of memory and are called Block 2, Block 3, Block 4, and Block 5.

Only Block 1 (from the first 64K) can be used to actually *display* a screen. Therefore, to *see* a screen stored in the second 64K (Blocks 2 to 5), it is necessary to move the required screen into Block 1. The BANK MANAGER provides all the commands needed to move screens around, such as `|SCREENCOPY` which simply transfers one screen to another, overwriting the contents of the existing screen; and `|SCREENSWAP` which exchanges the contents of two screens.

The BANK MANAGER uses 'external commands' which start with the bar symbol | (obtained by typing **[SHIFT]@**).

How to use the BANK MANAGER

Reset the computer **[CONTROL] [SHIFT] [ESC]**, then insert Side 1 of your system disc package and type:

```
RUN "BANKMAN"
```

The loading procedure is detailed in Chapter 6 part 13 on RSX's (Resident System eXtensions), and it is advisable to have some understanding about RSX's and also about reserving memory space, before using these routines in your programs. However, to try the following examples, you will not need to understand the loading procedure.

Type:

```
MODE 1

PRINT "THIS IS THE DEFAULT SCREEN"
|SCREENSWAP,1,2
```

The text should have disappeared. You are now looking at what used to be stored as Screen 2 (in Block 2). If the machine has just been switched on, this will probably be a random pattern. To clear this pattern, type:

```
MODE 1
```

....then type:

```
PRINT "THIS IS SCREEN 2"
|SCREENSWAP,1,2
```

Your original text reappears. If you now repeat the **|SCREENSWAP,1,2** command, you can see that the contents of the two screens are being exchanged. You can swap the contents of any of the five screens with each other, but bear in mind that only when you do a swap involving Screen 1 will the *display* be altered.

The other command available is **|SCREENCOPY**. This allows you to copy one screen onto another, overwriting it with a new image.

Type:

```
MODE 1

PRINT "THIS IS A SCREEN TO COPY"
|SCREENCOPY,2,1
```

The contents of Screen 1 are copied into Screen 2. If you reverse the parameters by typing:

```
MODE 1
|SCREENCOPY,1,2
```

....the contents of the currently displayed screen are overwritten from screen 2.

So the first parameter is the screen to copy to; the second parameter is the screen to copy from.

When copying screens, note that you will produce a disjointed display if the screens' **MODE**s are different, or if the viewing screen has been 'scrolled' since the last **MODE** command. The BANK MANAGER's screen commands are intended for use with pictorial screens rather than text screens, as scrolling is less likely to have occurred.

Using the BANK MANAGER for String Storage

There are four more commands provided by the BANK MANAGER, which allow the use of the additional 64K of memory as a filing system for string variables.

Most programs can be divided into two parts: firstly, the actual program instructions, and secondly, the data that the program uses. A good example of this is a database program such as an address book. Such a program would use an array of strings to store the names and addresses of the people listed in the book.

Strings can be stored in the second 64K of memory, one after another, end to end. The memory that the strings are stored in can be divided into compartments, which are called records. A record can be of any fixed length between 2 and 255 characters, whereas the length of a string in BASIC varies according to its contents. The purpose of a record is to provide standard sized compartments, like a set of pigeon holes, in which to store the untidy string information. Each operation to store or retrieve data from a record is automatically followed by a step to the next record, ready for the next operation. The record to be used for the next operation is called the 'current record' and will be used automatically unless a different record is specified.

We call this system of memory management a 'RAMdisc'. This is because it operates in a similar way to random access disc systems, but using RAM instead.

Read the following descriptions of the various commands so that you understand what each command is for, if not how to use it, then go on to try out the examples.

The first of the RAMdisc commands is **|BANKOPEN**. This command specifies how many characters of string information each record can contain. Its syntax is:

```
|BANKOPEN,n
```

Where n is a number specifying the amount of characters in the record. The value n can be in the range 0 to 255, but values of 0 and 1 will have strange effects.

!BANKWRITE stores a string into the current record. The current record is then incremented to point at the record after the one that has just been written, ready for the next operation. Its syntax is:

```
!BANKWRITE,@r%,a$
```

....or....

```
!BANKWRITE,@r%,a$,n
```

....where **r%** is an integer variable in which a code is returned giving information about the operation. **a$** is a string variable containing the characters to be written into the record. In the first of the two examples, the record is the current record. In the second example, the optional parameter n specifies the record to be written to.

!BANKREAD examines the current record, and returns its contents in a string. The current record is then incremented to point at the record after the one that has just been read, ready for the next operation. Reading the contents of a record does not change it, so the record can be read over and over again. The syntax for **!BANKREAD** is:

```
!BANKREAD,@r%,a$
```

....or....

```
!BANKREAD,@r%,a$,n
```

....where **r%** is an integer variable in which a code is returned giving information about the operation. **a$** is a string variable into which the contents of the record is read. In the first of the two examples, the record is the current record. In the second example, the optional parameter n specifies which record is to be read from.

The last command is **!BANKFIND**. This command searches through the records trying to find a particular string. If the string is found, then the number of the record in which it is found, is returned. The syntax for **!BANKFIND** is:

```
!BANKFIND,@r%,a$
```

....or....

```
!BANKFIND,@r%,a$,n
```

....or....

```
!BANKFIND,@r%,a$,n,m
```

....where r% is an integer variable in which either the record number (where the string was found) or a code indicating that it was not found, is returned. a$ is the string to search for. The optional parameter n, specifies which record to begin searching from. If n is unspecified, then the search will start at the current record. The second optional parameter m specifies at which record to end the search even if a match is not yet found. If m is omitted, the search continues through the whole 64K, which may overrun the end of the data you have written.

Now try this out. If you have already run the BANK MANAGER program to look at the screen swapping commands, and have not reset the computer, then the extra commands will currently be resident in memory. If not, you will need to insert Side 1 of your system disc package, and type:

```
RUN "BANKMAN"
```

Now type:

```
!BANKOPEN,20
```

This sets the record length to 20 characters and sets the current record to be 0.
Now type:

```
a$="FIRST ENTRY"+SPACE$(9)
```

....which sets up a$ to be exactly 20 characters long.

Now type:

```
r%=0
```

....to initialise r%.

Now type:

```
!BANKWRITE,@r%,a$
```

This will write a$ into the first record (record 0). Now type:

```
d$=SPACE$(20)
!BANKREAD,@r%,d$,0
PRINT d$
```

The first command sets up d$ to contain 20 spaces. This is enough room to contain the contents of the record when it is read. The second command reads record number 0 and places the result in d$. As the current record would now be stepped to record 1 (by the previous !BANKWRITE operation) it is necessary to specify record 0, as the record to read. Remember - if no record number is specified, then the current record is read. Finally the result of the reading operation is printed out. Therefore d$ ought to contain 'FIRST ENTRY' and 9 spaces.

Now type this:

```
b$="TWO"+SPACE$(17)
c$="THREE"+SPACE$(15)
!BANKWRITE,@r%,b$,1
!BANKWRITE,@r%,c$
```

This puts b$ and c$ into records 1 and 2. Again the optional parameter in the first !BANKWRITE command is used, allowing record 1 to be specified as the record in which to place b$. The current record number is thereafter set to the following record, so it is unnecessary to specify in which record to put c$ - it will be placed in record 2 automatically.

Type:

```
PRINT r%
```

The result of PRINTing r% after the above example should be 2. This number can either be considered as the last record operated on, or as the current record minus one. In the above example the last record operated on was 2 and the next record will be 3.

The purpose of this 'return code' is to provide information about the operation just carried out. A successful operation will return a positive number indicating a record number; an unsuccessful operation will return a negative number indicating an error code. There are two possible errors that can be returned by !BANKWRITE and !BANKREAD. These are:

-1 Indicates that the end of the file has been reached. This happens when all the records have been used up or a record has been specified that does not exist.

-3 Indicates a bank switching failure. This should never happen.

Try some further examples:

```
d$=STRING$(20,"X")
!BANKOPEN,20
FOR n=1 to 3:!BANKWRITE,@r%,d$:NEXT
```

This will set up d$ to contain 20 X's. !BANKOPEN resets the current record to be 0, so the !BANKWRITE command will overwrite the contents of records 0, 1, and 2 with d$.

Now type:

```
a$="FIRST"
!BANKWRITE,@r%,a$,0
```

This puts the word "FIRST" into record 0, overwriting some of the X's. Now reset d$ to spaces by:

```
d$=SPACE$(20)
```

Now type:

```
!BANKREAD,@r%,d$,0
```

This reads back record 0 into d$.

To recap the steps taken so far:

All three records contain X's. Into record 0 has been added the word "FIRST". d$ has then been filled with spaces. Finally, record 0 has been read back into d$. Now type:

```
PRINT d$
```

The result should be "FIRSTXXXXXXXXXXXXXXX". This illustrates an important consideration when using these commands. If the string being placed into a record does not completely fill that record, the old characters that have not been overwritten will remain in the record. Therefore, it is advisable to put a string of spaces or CHR$(32)'s into the record before storing fresh information in it. This avoids reading back a string and finding that it has characters which were not intended to still be in it. The same consideration should also be made to the string into which the record will be read. If the string is longer than the length of the record, then there will be characters left unchanged at the end of the string. That is why d$ was cleared (filled with spaces) before reading record 0 into it.

It is possible to write a string into a record which is too short to accept the complete string. If this happens, the excess characters will be ignored.

Likewise, it is also possible to read the contents of a long record into a short string. and once again, excess characters (read from the record) will be ignored. In normal string operations, BASIC would automatically extend the string to accept the additional characters but this does not happen within an external command.

Finally the !BANKFIND command. This allows the records to be searched for a particular entry. For example, if record number 24 starts with the word "FRED", this could be found with the command:

```
!BANKFIND,@r%,"FRED"
```

This is extremely useful for database programs where for example, a name, or address, is being searched for.

|BANKFIND will start at the current record, and will work its way through the records until it reaches the searched-for string, or the end of the second 64K in which the records are stored.

It is possible to specify the record number at which to start from, by including a record number at the end of the command.

A further record number, after the starting record number, can be included to specify the record at which to end the search.

|BANKFIND can be used to look for a string which does not lie at the start of a record. To do this, CHR$(Ø)'s are used in the character positions before the significant characters to be searched for. The CHR$(Ø)'s will be treated as 'wildcards' (like ?'s in CP/M filenames) which means that they count as 'any character'. An example of this would be:

```
a$=STRING$(1Ø,Ø)+"FRED"
|BANKFIND,@r%,a$,Ø
```

This would find the first occurrence of the word "FRED" (lying between the eleventh and fourteenth character positions). The first ten characters could contain a phone number or some other information that |BANKFIND would ignore.

The number of the record in which the string was found is placed in the integer variable r% used for the return code (if the word "FRED" was found), otherwise the return code will contain -2.

Further details....

Further information about the BANK MANAGER will be found in Chapter 7, and you should study parts 13 and 14 of Chapter 6 on the subject of using RSX's.

Well, that concludes this 11 part Foundation course on the 464/6128. By now you should know what most of the keys on the keyboard do, how to use some of the simpler BASIC commands, how to format a brand new disc so that it's ready to use, and how to perform some of the most elementary disc functions, i.e. LOADing, SAVEing, CATaloguing etc, together with a few AMSDOS, CP/M, and BANK MANAGER commands.

The chapters that follow, look at the more specialised aspects of computing and AMSTRAD BASIC.

Good luck, and happy reading!

Chapter 2
Beyond Foundations....

So you've read the Foundation course and you have the computer switched on in front of you. You've already learnt how to make it carry out an operation several times over using a FOR NEXT loop, and how to make it test IF a condition is true THEN do something.

Well, you're soon going to tire of seeing your name printed all over the screen, and will want to get on with some serious computing - something useful or entertaining. The chapter after this, lists each of the AMSTRAD BASIC keywords at your disposal, together with a description of the 'syntax', and what the keyword is used for. Armed with this list, the scope of what you can then make the computer do is limited only by your imagination.

If you have never used a computer before, the idea of actually 'programming' may fill you with apprehension. Fear not! It's a lot easier than you think, and certainly a lot easier than the technology and jargon would have you believe. Think of BASIC not as a new language, but as a variation of English with some words abbreviated to speed things up. In other words, don't think of CLS as 3 letters of magic code, but instead as, Clear Screen.

Try not to be afraid of BASIC, and you'll soon find yourself enjoying the business of programming, as well as the fruits of your endeavours. Programming can be a very rewarding exercise, especially when you're a beginner experimenting with the machine and the language. Always remember if you have a 6128 that as long as you make sure that you don't accidentally write onto your master CP/M system disc, nothing you type in can actually harm the computer, and it's always worth trying something new.

So where do I start?

Starting is often the most difficult part of the program for the beginner. However, what you should avoid doing, is plunging straight in and hacking away at the keyboard without any forethought.

One of the first things you should establish, is what exactly you want the program to do, and how you want the results to be presented to you (in other words, what the screen should look like as the program runs).

Having decided this, you can then start writing a program to fulfil your requirements, all the time thinking of how to make the program flow smoothly from beginning to end, with the minimum of jumping around using GOTO here and GOTO there. A good program will be easy to follow when listed, and will not land you in a hopeless muddle when you try to fault find, or 'de-bug' as they say in computer talk.

Fortunately, BASIC is a very forgiving language and will often help you by producing error messages on the screen when you go wrong. BASIC also allows you to have 'afterthoughts', and new lines of program can be sandwiched in between the existing lines with a minimum of fuss.

Writing a simple program

OK let's get going. We'll write a program to keep a record of our friends' names and telephone numbers. We'll call the program 'Telephone book'. Now let's apply the above rules: 'What should the program do?' and 'How should the results be presented?'

Well, the program should let you enter up to say, 100 names and 'phone numbers for storage. When you want a number, you should be able to type in the name and get back the number. In addition, if you're not sure how one of the names was originally entered, you should be able to display a complete list of all the information on the screen. Notice by the way, that we're automatically starting to consider the question of how the results of the program are to be presented.

Right, let's put finger to keyboard! We'll start off with the title at the beginning:

```
10 REM telephone book
```

You don't have to put a title in a program, but when you start to accumulate quite a few programs, it helps to be able to know at a glance which is which.

Next, we know that we want to be able to **INPUT** (put in) a string of characters (somebody's name) into a variable; we'll call that variable **NAME$**. The same applies to the telephone number, and we'll call that variable **TEL$**.

Remember those example programs from the Foundation course? They used the **INPUT** command for you to enter the value of the variable, so if we type in:

```
20 INPUT "enter the name";NAME$
30 INPUT "enter the telephone number";TEL$
run
```

You could enter the name, for example: Joe. You could then enter the 'phone number, for example: 0277 228888

The program has stored the information, but hasn't produced any printed results on the screen. A section of program is therefore needed to retrieve the information, then display it. To get the values of **NAME$** and **TEL$** at the moment, we'd use commands like:

```
PRINT NAME$ ....and.... PRINT TEL$
```

But hang on! We said that we want to be able to store up to 100 names and 100 'phone numbers in the program. Surely we don't have to write a program with two hundred INPUT commands, each with a different variable name, and then two hundred PRINT commands to produce a list on the screen??? Don't worry, computers make provision for this with what's known as an 'array'. An array allows you to use one variable (such as NAME$) which can have any number of 'dimensions' (in our program, we require 100). Then, when you want to get at the contents of the variable, you use the variable's name followed by its reference number (inside brackets). This reference number is called a 'subscript' and the expression NAME$(27) for example, is called a 'subscripted variable'. Now if we use a number-variable as the subscript, for example NAME$(x), we can then process the whole list of variables from 1 to 100 by changing the value of x in a FOR NEXT loop, i.e. FOR x=1 TO 100. As the value of x increases by 1, so the subscript value changes and refers to a different 'element' or name in the NAME$ array.

We want two arrays; one for NAME$, and one for TEL$, each with a dimension of 100 elements. Before we can start using an array, we must declare its DIMensions at the outset. We'll overwrite lines 20 and 30 with these statements:

```
20 DIM NAME$(100)
30 DIM TEL$(100)
```

Having established our variables, let's write some program that will firstly enable us to enter the names and numbers into the arrays, for later retrieval. Add:

```
40 FOR x=1 TO 100
50 INPUT;" name";NAME$(x)
60 INPUT " phone";TEL$(x)
70 NEXT
run
```

This is all very well, but we may not want to enter all 100 names at once. What's more, the way that the program presents itself on the screen is most unsatisfactory. What's needed here is a bit of tidying up. Firstly, before taking input of each new name and number, let's rid the screen of all the previous superfluous text, by CLearing the Screen each time. Add:

```
45 CLS
```

Now, how do we get the computer to know that we've finished inputting information for the moment? Pressing **[ESC]** would stop the program alright, but as soon as you typed RUN again, you'd lose all the values of your carefully entered variables!

Here's what we can do instead. As the program takes input of a new name, we'll make the program check that at least something has been typed in for the value of NAME$(x), in other words, check whether the value of NAME$(x) is an 'empty string' or not. IF it is, THEN we'll make the program stop taking any more input. Did you get the clue on how we can do this? Add:

```
55 IF NAME$(x)="" THEN 80
80 PRINT "no more input"
```

Also, so that the program tells you how to stop inputting, add:

```
47 PRINT "Press [RETURN] to end input"
```

Now let's write some program to PRINT the information you've stored, firstly in the form of a list. Add:

```
90 FOR x=1 TO 100
100 PRINT NAME$(x);" ";TEL$(x)
110 NEXT
```

Once again the program doesn't know when to stop, before reaching the 100th element of the array, so let's add:

```
95 IF NAME$(x)="" THEN 120
120 PRINT "list finished"
```

Line 95 detects whether NAME$(x) is an empty string, and IF so, THEN stops printing by bypassing lines 100 and 110.

And so to the next of our requirements. We'll now write some program that searches for a particular name that you enter. Add:

```
130 INPUT "find";SEARCH$
140 FOR x=1 TO 100
150 IF INSTR(NAME$(x),SEARCH$)=0 THEN 180
160 PRINT NAME$(x);" ";TEL$(x)
170 END
180 NEXT
190 PRINT "name not found"
run
```

There's a new command in line 150 - INSTR. It tells the computer to INterrogate the first STRing expression to find the first occurence of the second string expression, in other words, it searches NAME$(x) for SEARCH$ (which is the variable you input in line 130 containing the name that you're looking for). If INSTR doesn't find it (or any part of it), it gives out a value of 0, which is used here to make the program pass to line 180 and try again with the NEXT value of x. If the program has passed through all values of x up to 100, it then continues to line 190 and tells you that it hasn't found the name. If however it does find the name, INSTR will not produce a value of 0, and the program will then pass from line 150 to 160 and print the name and phone number, then END at line 170.

As you can see, our program is developing quite rapidly now, but there's still so much to be done. Let's sit back for a moment and consider some drawbacks of this program, starting with the way in which the program runs: First you type in the information, then you get a list back, then you search for a specific name.

What if?

Well, what if you don't want to do it in that order? What if you want to start by searching for a name that you stored in the program yesterday? And what if you want to add some more names and numbers to those already there? These are all aspects of the program that you have to think about and find solutions for; it's what programming is all about. As previously mentioned, BASIC is kind enough to let you sandwich afterthoughts into the program, but a good programmer will have anticipated these problems beforehand.

Another major problem with this program is that the values of the variables in the arrays are all stored in a part of the computer's memory that is cleared whenever you RUN a program. You'll not want to have to type in all the information every time you use the program, so you'll need the option of being able to save the values of all the NAME$ and TEL$ variables before you switch off, together with the option to load in the values of the variables whenever you run the program.

Solutions

The first of these problems (i.e. the order in which things are carried out) can be dealt with by writing the program so that when it runs, you get a choice of the various functions that it can perform. This type of program is called 'menu-driven', and in effect displays a menu on the screen from which you can select an option. If you've ever used one of those cash dispensers outside the bank, then you'll have already operated a menu-driven computer program! Let's add a menu to this program:

```
32 PRINT "1. enter info"
33 PRINT "2. list info"
34 PRINT "3. search"
35 PRINT "4. save info"
36 PRINT "5. load info"
37 INPUT "enter menu selection";ms
38 ON ms GOSUB 40,90,130

85 RETURN
125 RETURN
170 RETURN
200 RETURN
```

As you can see, we've made the program print the menu of options, then take INPUT of your selection, putting it into the variable ms. The command ON ms GOSUB in line 38 tells the program that if ms=1 then GO to the first SUB-routine line number (40); if ms=2 then GO to the second SUB-routine line number (90), and so on.

As each of the functions are now sub-routines called by the ON ms GOSUB command, they must each have a RETURN command at the end, hence we have added all those RETURN commands above.

Do you remember what the RETURN command does? It makes BASIC return from the sub-routine to the point in the program immediately following the appropriate GOSUB command, so in this case it returns to the instruction after line 38 (which means that the program will continue at line 40 - the 'enter info' point!) We don't want that to happen, so we must add:

```
39 GOTO 32
```

....to make the program loop back and display the menu once again. Now RUN the program again to see how far we've progressed.

OK let's have a look at the listing of the program for a moment. (If the program is still running, press **[ESC]** twice.) Type in:

```
LIST
```

This is what you should have so far:

```
10 REM telephone book
20 DIM NAME$(100)
30 DIM TEL$(100)
32 PRINT "1. enter info"
33 PRINT "2. list info"
34 PRINT "3. search"
35 PRINT "4. save info"
```

```
36 PRINT "5. load info"
37 INPUT "enter menu selection";ms
38 ON ms GOSUB 40,90,130
39 GOTO 32
40 FOR x=1 TO 100
45 CLS
47 PRINT "press [RETURN] to end input"
50 INPUT;" name";NAME$(x)
55 IF NAME$(x)="" THEN 80
60 INPUT " phone";TEL$(x)
70 NEXT
80 PRINT "no more input"
85 RETURN
90 FOR x=1 TO 100
95 IF NAME$(x)="" THEN 120
100 PRINT NAME$(x);" ";TEL$(x)
110 NEXT
120 PRINT "list finished"
125 RETURN
130 INPUT "find";SEARCH$
140 FOR x=1 TO 100
150 IF INSTR(NAME$(x),SEARCH$)=0 THEN 180
160 PRINT NAME$(x);" ";TEL$(x)
170 RETURN
180 NEXT
190 PRINT "name not found"
200 RETURN
```

You'll see that in certain parts of the program, we're starting to run out of lines to insert instructions, so let's create some more space and tidy things up by **RENUM**bering the lines. Type in:

```
RENUM
LIST
```

You should now see:

```
10 REM telephone book
20 DIM NAME$(100)
30 DIM TEL$(100)
40 PRINT "1. enter info"
50 PRINT "2. list info"
60 PRINT "3. search"
70 PRINT "4. save info"
80 PRINT "5. load info"
90 INPUT "enter menu selection";ms
```

```
100 ON ms GOSUB 120,210,270
110 GOTO 40
120 FOR x=1 TO 100
130 CLS
140 PRINT "press [RETURN] to end input"
150 INPUT;" name";NAME$(x)
160 IF NAME$(x)="" THEN 190
170 INPUT " phone";TEL$(x)
180 NEXT
190 PRINT "no more input"
200 RETURN
210 FOR x=1 TO 100
220 IF NAME$(x)="" THEN 250
230 PRINT NAME$(x);" ";TEL$(x)
240 NEXT
250 PRINT "list finished"
260 RETURN
270 INPUT "find";SEARCH$
280 FOR x=1 TO 100
290 IF INSTR(NAME$(x),SEARCH$)=0 THEN 320
300 PRINT NAME$(x);" ";TEL$(x)
310 RETURN
320 NEXT
330 PRINT "name not found"
340 RETURN
```

That's better. Now on with the program! We'll now add an instruction, so that whenever you enter some new names and 'phone numbers, the computer will add them to the existing entries, by placing them in the first element of the array that it finds to be empty. This time, we'll use the new command **LEN** to tell us the **LEN**gth of the string. We'll specify the following:

IF the **LEN**gth of **NAME$(x)** is greater than 0, i.e. if there's already an entry in that element of the array, **THEN** jump to line **180** (which steps to the **NEXT** element in the array).

Notice again how similar the above instruction in English is, compared to the equivalent BASIC instruction just below. I told you that BASIC isn't really a different language!!!

```
135 IF LEN(NAME$(x))>0 THEN 180
```

Such a simple solution isn't it? Problems like these can always be sorted out with your list of BASIC keywords and a little thought. There's nearly always at least one command that will satisfy your programming needs, and the more you program, the more you'll be able to find instant solutions 'off the top of your head'.

Now to the matter of saving the contents of the variables so that they can be loaded back in when the program is run. Part 7 of the Foundation course explained how to save the program itself, using the SAVE command. However the program itself is only a framework which lets the variable values be put in (at the keyboard) and taken out (at the screen). When you SAVE the program, you are saving only that framework, not the values of the variables.

Hence we must write a section of program that will save the values of the variables onto disc. We do this by creating a separate 'data file'.

First we OPEN an OUTput file and specify a name for it such as "data". Then we WRITE the values of the variables NAME$(x) and TEL$(x) from 1 to 100, into the file, and finally we CLOSE the OUTput file and RETURN to the menu. Let's add it to the program from line 350 onwards. To save us typing in each new line number, we'll use the command:

```
AUTO 350
```

....which will start AUTOmatic line numbering from the required line:

```
350 OPENOUT "data"
360 FOR x=1 TO 100
370 WRITE #9,NAME$(x),TEL$(x)
380 NEXT
390 CLOSEOUT
400 PRINT "data saved"
410 RETURN
```

After you have typed in line 410 and pressed **[RETURN]**, press **[ESC]** to stop the AUTOmatic line numbering.

Now we need to add an extra number to the list of numbers in the ON ms GOSUB command in line 100. This is because we have added another option for the menu to select. Therefore EDIT line 100 to add this extra number:

```
100 ON ms GOSUB 120,210,270,350
```

Now, whenever you select menu option number 4, the program will save all the information that you entered, onto disc, or cassette.

Notice that in line 370, where the program WRITEs the values of NAME$(x) and TEL$(x) onto disc or cassette, the expression #9 is used after the word WRITE. The # sign is a 'stream director', in other words, it tells the computer which 'stream' to send the data to. The computer has 10 streams:

Directing data to streams 0 to 7 (#0 to #7) will result in it appearing on the screen because streams #0 to #7 are 'screen streams' or WINDOWs.

Directing data to stream #8 sends data to the printer (if connected).

Finally, directing data to stream #9 sends it to the disc drive or cassette, which is what we have done in line 370.

If I may digress....

A few words now about the AUTO command which we used a moment ago. If you leave out the line number and just type in:

```
AUTO
```

....the computer will commence line numbering from 10, advancing by 10 each time **[RETURN]** is pressed. If you have previously used lines 10, 20, 30 etc, each line's contents will be displayed on the screen as you pass through it (by pressing **[RETURN]** each time). When each line appears on the screen, it may be edited before pressing **[RETURN]**, thus providing a quick method of continuously editing regularly successive program lines.

Back to the program....

We've now added the instructions to save the information to disc or cassette, so the final main section of this program will load the data back from disc or cassette, ready for use. Therefore, we must add yet another menu option to the list of numbers in line 100. Edit line 100 again, as follows:

```
100 ON ms GOSUB 120,210,270,350,420
```

Now for the instructions to load the information. We start by OPENing the INput file from disc or cassette called "data". Then we take INPUT from disc or cassette (stream #9) of all the values of the variables NAME$(x) and TEL$(x) from 1 to 100, and lastly we CLOSE the INput file and RETURN to the menu. Type in:

```
420 OPENIN "data"
430 FOR x=1 TO 100
440 INPUT #9,NAME$(x),TEL$(x)
450 NEXT
460 CLOSEIN
470 PRINT "data loaded"
480 RETURN
```

Remember that if you have a 464 you will have to rewind the cassette before loading the data back.

The end of the beginning....

So now we have written a program which fulfils the requirements that we set out to achieve when we decided 'what the program has to do'. All that remains now, is to improve the way that 'the results are presented' on the screen.

The beginning of the end....

Let's add some instructions to tidy up the presentation of the program:

```
34 MODE 1
```

This establishes the screen mode, and clears the screen at the start of the program. Now add:

```
36 WINDOW #1,13,30,10,14
```

Don't be put off by this seemingly complicated instruction. What we're doing here is creating a small window on the screen to put the menu into. After the word **WINDOW**, we first specify which stream number this window is for, remember we can use 8 screen streams from **#0** to **#7**. Now bearing in mind that all items printed on the screen use stream **#0** unless otherwise instructed, we won't use stream **#0** for our little window, otherwise everything that the program prints will be sent to it. Instead we'll specify another stream between **#1** and **#7**, and as you can see, we've chosen **#1**. The four numbers that follow **#1**, tell the computer what size the **WINDOW** should be, and it couldn't be easier; the numbers specify the left, right, top, and bottom edges of the window, and refer to text column and row numbers (the same as those used in the **LOCATE** command). So in our example, after specifying that it's stream **#1** we're using, we say that the left edge of the window starts at column **13**, the right edge ends at column **30**, the top edge starts at row **10**, and the bottom edge ends at row **14**.

Now to make all our menu options print in stream (**WINDOW**) **#1**, we'll have to edit lines **40** to **80** as follows:

```
40 PRINT #1,"1. enter info"
50 PRINT #1,"2. list info"
60 PRINT #1,"3. search"
70 PRINT #1,"4. save info"
80 PRINT #1,"5. load info"
```

Let's now add:

```
85 LOCATE 7,25
```

This LOCATEs where the menu INPUT statement in line 90 will appear, so that it looks neater.

So that the screen is always cleared when the menu is returned to, edit line 110 to:

```
110 GOTO 34
```

So that the screen is always cleared when one of the menu options is selected, add:

```
95 CLS
```

Finally, we'll just add the following three lines to make the program pause before returning to the menu:

```
103 LOCATE 9,25
105 PRINT "press any key for menu"
107 IF INKEY$="" THEN 107
```

Line 103 tells the computer where to print the message given in line 105. Line 107 INterrogates the KEYboard to see what $tring variable is being input (by a key being pressed). IF it detects that an empty string is being input (because no key is pressed), THEN the program loops back to the same instruction again and again, until INKEY$ finds that the string is not empty because a key has been pressed. This is a useful way of creating a pause in a program, as BASIC will not pass to next line until a key is pressed.

So there it is; the finished program. Or is it? Well, you could make it have the facility to amend and delete names and 'phone numbers, to sort the list into alphabetical order, to 'print-out' the list on a printer, or if you're really ambitious, to make the program produce the signals to automatically dial the number after you've typed in the name - with of course, the permission of British Telecom to connect your 'phone to the computer! Nevertheless all these enhancements to the program are possible, and in truth you can go on forever improving and streamlining a program, especially when you've got a computer system as powerful as the 464/6128. No, we've got to draw the line somewhere, and this is where we'll leave our 'telephone book', hopefully having learned a thing or two about the art of writing a program from scratch. Tidy up the program by typing in:

```
RENUM
```

....and then save it to disc or cassette, or throw it away. You never know though, it might come in handy for keeping a record of your friends' names and telephone numbers!

Final listing:

```
10 REM telephone book
20 DIM NAME$(100)
30 DIM TEL$(100)
40 MODE 1
50 WINDOW #1,13,30,10,14
60 PRINT #1,"1. enter info"
70 PRINT #1,"2. list info"
80 PRINT #1,"3. search"
90 PRINT #1,"4. save info"
100 PRINT #1,"5. load info"
110 LOCATE 7,25
120 INPUT "enter menu selection";ms
130 CLS
140 ON ms GOSUB 190,290,350,430,500
150 LOCATE 9,25
160 PRINT "press any key for menu"
170 IF INKEY$="" THEN 170
180 GOTO 40
190 FOR x=1 TO 100
200 CLS
210 IF LEN(NAME$(x))>0 THEN 260
220 PRINT "press [RETURN] to end input"
230 INPUT;" name";NAME$(x)
240 IF NAME$(x)="" THEN 270
250 INPUT " phone";TEL$(x)
260 NEXT
270 PRINT "no more input"
280 RETURN
290 FOR x=1 TO 100
300 IF NAME$(x)="" THEN 330
310 PRINT NAME$(x);" ";TEL$(x)
320 NEXT
330 PRINT "list finished"
340 RETURN
350 INPUT "find";SEARCH$
360 FOR x=1 TO 100
370 IF INSTR(NAME$(x),SEARCH$)=0 THEN 400
380 PRINT NAME$(x);" ";TEL$(x)
390 RETURN
400 NEXT
410 PRINT "name not found"
420 RETURN
430 OPENOUT "data"
```

continued next page ....

```
440 FOR x=1 TO 100
450 WRITE #9,NAME$(x),TEL$(x)
460 NEXT
470 CLOSEOUT
480 PRINT "data saved"
490 RETURN
500 OPENIN "data"
510 FOR x=1 TO 100
520 INPUT #9,NAME$(x),TEL$(x)
530 NEXT
540 CLOSEIN
550 PRINT "data loaded"
560 RETURN
run
```

Chapter 3 Complete list of Amstrad 464/6128 BASIC keywords

IMPORTANT

It is vital that you understand the terminology and notation that we use in this chapter. You will see various types of brackets used when explaining how a particular command is typed in; each of these types of brackets has a specific meaning, and you should note them well.

Any part of a command not shown enclosed by brackets is required as given. For example, the command **END** takes the form:

```
END
```

....and you must type in the word **END** literally.

Where an item is enclosed in angled brackets ‹› for example:

```
‹line number›
```

....you are NOT required to type the brackets, nor the words within them. The example above shows you the type of data required in the command. For example:

```
EDIT ‹line number›
```

....means that you should type in:

```
EDIT 100
```

Round brackets **( )** MUST be typed in literally. For example:

```
COS(‹numeric expression›)
```

....requires that brackets be typed around the ‹numeric expression› of which the COSine is required, e.g:

```
PRINT COS(45)
```

Finally, square brackets enclose optional items in a command or function. For example:

```
RUN [‹line number›]
```

....means that you do not have to follow the keyword **RUN** with a parameter, but that you can expand the command by adding the optional parameter ‹line number›. Hence, the command could be typed in as:

```
RUN ....or.... RUN 100
```

Special Characters

& or &H	Prefix for hexadecimal constant
&X	Prefix for binary constant
#	Prefix for stream director

Data types

Strings may be from 0 to 255 characters long. A ‹string expression› is an expression which yields a value of type string. Strings may be appended to one another using the + operator, as long as the resulting string is no greater than 255 characters long.

Numeric data can be either integer or real. Integer data is held in the range −32768 to 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.

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 a screen window, printer, or disc, where the text is required to 'stream'.

A ‹list of:‹item› describes a parameter which comprises a list of items separated by commas. The list may contain one, or any number of items, limited by line length.

Type markers are:

%	Integer
!	Real (The default)
$	String

Please note that AMSTRAD 464/6128 BASIC keywords are listed here using the form:

KEYWORD

Syntax

Example

Description

Associated keywords

Keywords are either:

COMMANDS : operations that are executed directly.
FUNCTIONS : operations that are brought into action as arguments in an expression.
OPERATORS : acting upon mathematical arguments.

BASIC converts all keywords entered in lower case letters to UPPER CASE when a program is LISTed. The examples shown in this chapter use UPPER CASE, since this is how the program will appear when LISTed. Hence you should enter using lower case, as you will be able to spot typing errors more readily since the mis-typed keyword will still be displayed in lower case when LISTed.

Keywords....

ABS

ABS(‹numeric expression›)

```
PRINT ABS(-67.98)
 67.98
```

FUNCTION: Returns the ABSolute value of the given expression. This means that negative numbers are returned as positive.

Associated keywords: SGN

AFTER

AFTER ‹timer delay› [, ‹timer number›] GOSUB ‹line number›

```
10 AFTER 250 GOSUB 60:CLS
20 PRINT "Guess a letter in 5 seconds"
30 a$=INKEY$:IF flag=1 THEN END
40 IF a$<>CHR$(INT(RND*26+97)) THEN 30
50 PRINT a$;" is correct. You win"
55 SOUND 1,478:SOUND 1,358:END
60 PRINT "too late. I win"
70 SOUND 1,2000:flag=1:RETURN
run
```

COMMAND: Calls a BASIC sub-routine after a given period of time has elapsed. The ‹timer delay› parameter specifies the period of time in units of 0.02 (fiftieths) second.

The ‹timer number› (in the range 0 to 3) specifies which of the four delay timers are to be used. Timer 3 has the highest priority; timer 0 (the default timer) has the lowest.

Each of the timers may have a sub-routine associated with it

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: EVERY, REMAIN, RETURN

AND

‹argument› AND ‹argument›

```
IF "alan"<"bob" AND "dog">"cat" THEN PRINT "correct" ELSE PRINT "wrong"
correct
IF "bob"<"alan" AND "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
IF "alan"<"bob" AND "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
....
PRINT 1 AND 1
 1
PRINT 0 AND 0
 0
PRINT 1 AND 0
 0
```

OPERATOR: Performs bit-wise boolean operation on integers. Result is 0 unless both argument bits are 1.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: OR, NOT, XOR

ASC

ASC(‹string expression›)

```
PRINT ASC("x")
 120
```

FUNCTION: Returns the numeric value of the first character in the ‹string expression›.

Associated keywords: CHR$

ATN

ATN(‹numeric expression›)

```
PRINT ATN(1)
 0.785398163
```

FUNCTION: Calculates the Arc-TaNgent of the ‹numeric expression›.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: COS, DEG, RAD, SIN, TAN

AUTO

AUTO[‹line number›][,‹increment›]

```
AUTO 100,50
```

COMMAND: Generates line numbers AUTOmatically. The optional ‹line number› parameter sets the first line to be generated in case you wish to generate lines from a particular point in the program. If the parameter is omitted, line numbers will be generated from line 10 onwards.

The optional ‹increment› sets the number of lines to leave before generating the following line number. If the parameter is omitted, the line numbers will increase by 10 each time.

If a line number is generated which is already in use, then the contents of the line will be displayed on the screen and may be edited if required. The displayed line will be replaced in the memory when **[RETURN]** is pressed.

To stop automatic line numbering, press **[ESC]**.

Associated keywords: **none**

BIN$

BIN$(‹unsigned integer expression›[,‹integer expression›])

```
PRINT BIN$(64,8)
01000000
```

FUNCTION: Produces a string of BINary digits representing the value of the ‹unsigned integer expression›, using the number of binary digits instructed by the second ‹integer expression› (in the range 0 to 16). If the number of digits instructed is too great, the resulting expression will be filled with leading zeros; if the number of digits instructed is too small, the resulting expression will NOT be shortened to the instructed number of digits, but will be produced in as many digits as are required.

The ‹unsigned integer expression› to be converted into binary form must yield a value in the range −32768 to 65535.

Associated keywords: DEC$, HEX$, STR$

BORDER

BORDER ‹colour›[,‹colour›]

```
10 REM 729 border combinations!
20 SPEED INK 5,5
30 FOR a=0 TO 26
40 FOR b=0 TO 26
50 BORDER a,b:CLS:LOCATE 14,13
60 PRINT "border";a;",";b
70 FOR t=1 TO 500
80 NEXT t,b,a
run
```

COMMAND: Changes the colour of the border on the screen. If two colours are specified, the border alternates between the two at a rate determined in the SPEED INK command. The range of border colours is 0 to 26.

Associated keywords: SPEED INK

BREAK

(See ON BREAK CONT, ON BREAK GOSUB, ON BREAK STOP)

CALL

CALL ‹address expression›[, ‹list of:‹parameter›]

```
CALL Ø
```

COMMAND: Allows an externally developed sub-routine to be called from BASIC. The above call completely resets the computer.

Not a command to be used by the unwary.

Associated keywords: UNT

CAT

CAT

```
CAT
```

COMMAND: CATalogs the disc or cassette.

6128: Displays in alpha-numeric order, the full names of all files found, together with each file's length (to the nearest higher Kbyte). The free space left on the disc is also displayed, together with Drive and User identification.

Cataloguing does not affect the program currently in memory.

464: You will be instructed:

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

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer will display the names of each of the files that it finds (in sequence) on the cassette.

Each of the blocks of a file will be displayed, followed by a single character which indicates what sort of file it is:

$	is an unprotected BASIC file
%	is a Protected BASIC file
*	is an ASCII file
&	is a Binary file

The computer displays:

```
Ok
```

....at the end of the line if it has read the file successfully, indicating that the file would have loaded into memory, had the computer attempted to do so.

The CAT function will not affect the program currently in the computer's memory.

If a cassette file has been saved without a specified name, CAT will display it as:

```
Unnamed file
```

CAT is terminated by pressing **[ESC]**.

Associated keywords: LOAD, RUN, SAVE

CHAIN

CHAIN ‹filename› [, ‹line number expression›]

```
CHAIN "testprog.bas",350
```

COMMAND: Loads a program from disc or cassette into the memory, replacing the existing program. The new program then commences running, either from the beginning, or from a line specified in the optional ‹line number expression›.

Protected files, (SAVEd by ,p) can be loaded and run by CHAINing.

464: You need not specify the filename if you wish the first suitable file on the cassette to be loaded.

You will be instructed:

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

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer will load the file to be processed.

The screen will display the loading messages:

```
Loading FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is loaded.

If the first character of the filename is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the [ESC] key produces the error message on the screen:

```
Broken in
```

Associated keywords: CHAIN MERGE, LOAD, MERGE

CHAIN MERGE

CHAIN MERGE ‹filename›[, ‹line number expression›]
[, DELETE ‹line number range›]

```
CHAIN MERGE "newrun.bas",750,DELETE 400-680
```

COMMAND: Loads a program from disc or cassette, merges it into the current program in the memory, then commences running the resultant program, either from the beginning, or from a line specified in the optional ‹line number expression›. If before a program is CHAIN MERGEd, it is required to delete part of the original program, the DELETE ‹line number range› option may be used.

Note that line numbers in the old program which exist in the new program to be CHAIN MERGEd, will be over-written by the new program lines.

Protected files, (SAVEd by ,p) can NOT be loaded and run by CHAIN MERGEing.

464: You need not specify the filename if you wish the first suitable file on the cassette to be loaded.

You will be instructed:

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

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer will load the file to be processed.

The screen will display the loading messages:

```
Loading FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is loaded.

If the first character of the filename is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the **[ESC]** key produces the error message on the screen:

```
Broken in
```

Associated keywords: CHAIN, DELETE, LOAD, MERGE

CHR$

CHR$(‹integer expression›)

```
10 FOR x=32 TO 255
20 PRINT x;CHR$(x),
30 NEXT
run
```

FUNCTION: Converts an ‹integer expression› in the range 0 to 255, to its CHaRacter $tring equivalent, using the AMSTRAD character set shown in part 3 of the chapter entitled 'For your reference....'.

Note that 0 to 31 are control characters; hence the above example prints CHR$(x) in the range 32 to 255.

Associated keywords: ASC

CINT

CINT(‹numeric expression›)

```
10 n=1.9999
20 PRINT CINT(n)
run
 2
```

FUNCTION: Returns the value of the ‹numeric expression›, Converting it to a rounded INTeger in the range −32768 to 32767.

Associated keywords: CREAL, FIX, INT, ROUND, UNT

CLEAR

CLEAR

```
CLEAR
```

COMMAND: Clears all variables to zero or null. All open files are abandoned, all arrays and user functions are erased, and BASIC is set to radians mode of calculation.

Associated keywords: **none**

CLEAR INPUT

CLEAR INPUT

```
10 CLS
20 PRINT "Type in letters now!"
30 FOR t=1 TO 3000
40 NEXT
50 CLEAR INPUT
run
```

COMMAND: Discards all previously typed input from the keyboard, still in the keyboard buffer.

To see the effect of this command, RUN the above program and type in letters when asked to do so. Then delete line 50 of the program and RUN again, noting the difference.

Associated keywords: INKEY, INKEY$, JOY

CLG

CLG[‹ink›]

```
LOCATE 1,20
CLG 3
```

COMMAND: CLears the Graphics screen to the graphics paper colour. If the ‹ink› is specified, the graphics paper is set to that value.

Associated keywords: CLS, GRAPHICS PAPER, INK, ORIGIN

CLOSEIN

CLOSEIN

```
CLOSEIN
```

COMMAND: CLOSEs any INput file from disc or cassette. (See OPENIN).

Associated keywords: EOF, OPENIN

CLOSEOUT

CLOSEOUT

```
CLOSEOUT
```

COMMAND: CLOSEs any OUTput file to disc or cassette. (See OPENOUT).

464: If the file buffer is partly full and the command CLOSEOUT is encountered, the computer will save the remaining contents of the file buffer to cassette, issuing the prompt:

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

The screen will display the saving message:

```
Saving FILENAME block <x>
```

If the first character of the filename in the corresponding OPENOUT command is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to save. (You must make sure that the **RECORD** and **PLAY** buttons on your cassette unit are down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning file output using the [ESC] key produces the error message on the screen:

```
Broken in
```

Associated keywords: OPENOUT

CLS

CLS[#‹stream expression›]

```
10 PAPER#2,3
20 CLS#2
run
```

COMMAND: CLears the given Screen stream (window) to its paper ink. If no ‹stream expression› is given, screen stream #Ø is cleared.

Associated keywords: CLG, INK, PAPER, WINDOW

CONT

CONT

```
CONT
```

COMMAND: CONTinues program execution, either after the **[ESC]** key has been pressed twice, or after a STOP command has been encountered in the program. CONT will only continue to execute the program if it has not been altered, and if it is not a protected program.

Direct commands may be entered before CONTinuing.

Associated keywords: STOP

COPYCHR$

COPYCHR$(#‹stream expression›)

```
10 CLS
20 PRINT "top corner"
30 LOCATE 1,1
40 a$=COPYCHR$(#0)
50 LOCATE 1,20
60 PRINT a$
run
```

FUNCTION: COPies a CHaRacter from the current position in the stream (which MUST be specified). The above program copies a character from location 1,1 (top left), and reproduces it at location 1,20.

If the character read is not recognised, a null string is returned.

Associated keywords: LOCATE

COS

COS(‹numeric expression›)

```
DEG
PRINT COS(45)
 0.707106781
```

FUNCTION: Calculates the COSine of the ‹numeric expression›.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: ATN, DEG, RAD, SIN

CREAL

CREAL(‹numeric expression›)

```
10 a=PI
20 PRINT CINT(a)
30 PRINT CREAL(a)
run
 3
 3.14159265
```

FUNCTION: Returns the value of the ‹numeric expression›, Converting it to REAL.

Associated keywords: CINT

CURSOR

CURSOR[‹system switch›][,‹user switch›]

```
10 CURSOR 1
20 PRINT "question?";
30 a$=INKEY$:IF a$="" THEN 30
40 PRINT a$
50 CURSOR 0
run
```

COMMAND: Sets the system switch or the user switch to the cursor, on or off. The ‹system switch› and ‹user switch› parameters must be either 0 (off) or 1 (on). In the above INKEY$ command, where the cursor is not normally visible, the cursor has been turned on by the ‹system switch› setting of 1 (in line 10).

The cursor is displayed whenever both the ‹system switch› and the ‹user switch› are on (1). The ‹system switch› is automatically turned on for the command INPUT, but is turned off for INKEY$.

It is recommended that the cursor be turned off when printing text to the screen.

Either switch parameter may be omitted, but not both. If a switch parameter is omitted, that particular switch state is not changed.

Associated keywords: LOCATE

DATA

DATA ‹list of:‹constant›

```
10 FOR x=1 TO 4
20 READ name$,surname$
30 PRINT name$;" ";surname$
40 NEXT
50 DATA Hilda,Ogden,Bet,Lynch
60 DATA Rita,Fairclough,Mavis,Riley
run
```

COMMAND: Declares constant data for use within a program. Data may be read into the variable by the READ command, after which the 'pointer' moves on to the next item in the DATA list. The RESTORE command may be used to move the pointer to a specified DATA position.

Further information concerning data will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: READ, RESTORE

DEC$

DEC$(‹numeric expression›,‹format template›)

```
PRINT DEC$(10↑7,"ff##########,.##")
 £10,000,000.00
```

FUNCTION: Returns a DECimal string representation of the ‹numeric expression›, using the specified ‹format template› to control the print format of the resulting string.

The format template may contain ONLY the characters:

```
+ - £ $ * # , . ↑
```

The use of these 'format field specifiers' is described under the keyword PRINT USING.

Associated keywords: BIN$, HEX$, PRINT USING, STR$

DEF FN

DEF FN‹function name›[(‹formal parameters›)]=‹expression›

```
10 t=TIME/300
20 DEF FNclock=INT(TIME/300-t)
30 EVERY 100 GOSUB 50
40 GOTO 40
50 PRINT "program was run";
60 PRINT FNclock;"seconds ago"
70 RETURN
run
```

COMMAND: DEFines a FuNction. BASIC allows the program to define and use simple value returning functions. DEF FN is the definition part of this mechanism and creates a program-specific function which works within the program in the same way that for example COS operates as a built-in function of BASIC.

(Note in the above example how the value of the function FNclock is continually updated even if the program is paused by pressing **[ESC]** once, or stopped by pressing **[ESC]** twice, then CONTinuing.)

Associated keywords: **none**

DEFINT

DEFINT ‹list of:‹letter range›

```
10 DEFINT n
20 number=123.456
30 PRINT number
run
 123
```

COMMAND: Sets the DEFault for a variable to type INTeger. When a variable is encountered without an explicit type marker (! % $), the default type is assumed. This command sets the default for variables with the specified first letter(s) to type INTeger. There may be a list of first letters such as:

```
DEFINT a,b,c
```

....or there may be an inclusive range of first letters such as:

```
DEFINT a-z
```

Associated keywords: DEFREAL, DEFSTR

DEFREAL

DEFREAL ‹list of:‹letter range›

```
DEFREAL x,a-f
```

COMMAND: Sets the DEFault for a variable to type REAL. When a variable is encountered without an explicit type marker (! % $), the default type is assumed. This command sets the default for variables with the specified first letter(s) to type REAL. There may be a list of first letters such as:

```
DEFREAL a,b,c
```

....or there may be an inclusive range of first letters such as:

```
DEFREAL a-z
```

Associated keywords: DEFINT, DEFSTR

DEFSTR

DEFSTR ‹list of: ‹letter range›

```
10 DEFSTR n
20 name="Amstrad"
30 PRINT name
run
Amstrad
```

COMMAND: Sets the DEFault for a variable to type STRing. When a variable is encountered without an explicit type marker (! % $), the default type is assumed. This command sets the default for variables with the specified first letter(s) to type STRing. There may be a list of first letters such as:

```
DEFSTR a,b,c
```

....or there may be an inclusive range of first letters such as:

```
DEFSTR a-z
```

Associated keywords: DEFINT, DEFREAL

DEG

DEG

```
DEG
```

COMMAND: Sets DEGrees mode of calculation. The default condition for the functions SIN, COS, TAN, and ATN is radians. DEG resets BASIC to degrees until instructed otherwise by the commands RAD, and NEW, CLEAR, LOAD, RUN etc.

Associated keywords: ATN, COS, RAD, SIN, TAN

DELETE

DELETE ‹line number range›

```
DELETE 100-200
```

COMMAND: Deletes part of the current program as defined in the ‹line number range› expression.

The first or last number in the ‹line number range› may be omitted to indicate '....from the beginning of the program', or '....to the end of the program', i.e:

```
DELETE -200
```

....or....

```
DELETE 50-
```

....or....

```
DELETE
```

....which deletes the whole program.

Associated keywords: CHAIN MERGE, RENUM

DERR

DERR

```
LOAD "xyz.abc"

XYZ .ABC not found
Ready
PRINT DERR
 146
```

FUNCTION: Reports the last error code returned by the disc filing system. The value of DERR may be used to ascertain the particular Disc ERRor that occurred. See the listing of error messages given in the chapter entitled 'For your reference....'.

Associated keywords: ERL, ERR, ERROR, ON ERROR GOTO, RESUME

DI

DI

```
10 CLS:TAG:EVERY 10 GOSUB 90
20 X1=RND*320:X2=RND*320
30 Y=200+RND*200:C$=CHR$(RND*255)
40 FOR X=320-X1 TO 320+X2 STEP 4
50 DI
60 MOVE 320,0,1:MOVE X-2,Y:MOVE X,Y
70 PRINT " ";C$;:FRAME
80 EI:NEXT:GOTO 20
90 MOVE 320,0:DRAW X+8,Y-16,0:RETURN
run
```

COMMAND: Disables Interrupts (other than the **[ESC]** interrupt) until re-enabled explicitly by EI or implicitly by the RETURN at the end of an interrupt sub-routine.

Note that entering an interrupt sub-routine automatically disables interrupts of an equal or lower priority.

The command is used to make the program literally execute 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 sub-routine are competing for use of the graphics display.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, EI, EVERY, REMAIN

DIM

DIM ‹list of:‹subscripted variable›

```
10 CLS
20 DIM friend$(5),phone$(5)
30 FOR n=1 TO 5
40 PRINT "Friend number";n
50 INPUT "Enter name";friend$(n)
60 INPUT "Enter tel.number";phone$(n)
70 PRINT
80 NEXT
90 FOR n=1 TO 5
100 PRINT n;friend$(n),phone$(n)
110 NEXT
run
```

COMMAND: **DIM**ensions an array. **DIM** allocates space for arrays and specifies maximum subscript values. BASIC must be advised of the space to be reserved for an array, or it will default to 10.

An array is identified by a ‹subscripted variable› where one variable name is used with a range of subscript numbers, so that each 'element' of the array has its own individual value. Control of the array can then be achieved by for example **FOR NEXT** loops, which can step through the array, processing each element in turn

Note that the lowest value of the subscript is zero (i.e. the first available element in the array).

Arrays can be multi-dimensional, and each element of such an array is referenced by its position within the framework of the array. For example, in an array dimensioned by:

```
DIM position$(20,20,20)
```

....an element of the array would be referenced for example:

```
position$(4,5,6)
```

Associated keywords: **ERASE**

DRAW

DRAW ‹x co-ordinate› , ‹y co-ordinate›[, [‹ink›][, ‹ink mode›]]

```
10 MODE 0:BORDER 0:PAPER 0:INK 0,0
20 x=RND*640:y=RND*400:z=RND*15
30 DRAW x,y,z
40 GOTO 20
run
```

COMMAND: Draws a line on the graphics screen, from the current graphics cursor position to the absolute position specified in the x,y co-ordinates. The ‹ink› in which to draw the line may be specified (in the range 0 to 15).

The optional ‹ink mode› determines how the ink being written interacts with that already on the graphics screen. The 4 ‹ink mode›s are:

0 : Normal
1 : XOR (eXclusive OR)
2 : AND
3 : OR

Associated keywords: DRAWR, GRAPHICS PEN, MASK

DRAWR

DRAWR ‹x offset› , ‹y offset›[, [‹ink›][, ‹ink mode›]]

```
10 CLS:PRINT "coming upstairs?"
20 MOVE 0,350:FOR n=1 TO 8
30 DRAWR 50,0
40 DRAWR 0,-50
50 NEXT:MOVE 348,0:FILL 3
60 GOTO 60
run
```

COMMAND: Draws a line on the graphics screen, from the current graphics cursor position to the relative position specified in the ‹x and y offset›s. The ‹ink› in which to draw the line may be specified (in the range 0 to 15).

The optional ‹ink mode› determines how the ink being written interacts with that already on the graphics screen. The 4 ‹ink mode›s are:

0 : Normal
1 : XOR (eXclusive OR)
2 : AND
3 : OR

Associated keywords: DRAW, GRAPHICS PEN, MASK

EDIT

EDIT ‹line number›

```
EDIT 20
```

COMMAND: Displays the program line specified in the ‹line number› on the screen together with the cursor, ready for editing.

Associated keywords: AUTO, LIST

EI

EI

```
EI
```

COMMAND: Enables Interrupts which have been disabled by the DI command.

If interrupts are disabled in an interrupt sub-routine, they are automatically re-enabled when BASIC encounters the RETURN command at the end of the sub-routine.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, DI, EVERY, REMAIN

ELSE

(See IF)

END

```
END
```

```
    END
```

COMMAND: Ends the execution of a program, and returns to direct mode. Any number of **END** commands may appear in a program, and one is automatically assumed after the final line of a program.

Associated keywords: **STOP**

ENT

ENT ‹envelope number› [**,** ‹envelope section›][**,** ‹envelope section›]
[**,** ‹envelope section›][**,** ‹envelope section›]
[**,** ‹envelope section›]

```
10 ENT 1,10,-50,10,10,50,10
20 SOUND 1,500,200,10,,1
run
```

COMMAND: Sets the Tone **EN**velope specified in the ‹envelope number› (in the range 1 to 15), which is used in conjunction with the **SOUND** command. If the ‹envelope number› is negative (in the range -1 to -15), the envelope repeats until the end of the duration of the **SOUND** command.

Each of the ‹envelope section›s may contain either 2 or 3 parameters:

If 3 parameters are used, these are:

‹number of steps› **,** ‹step size› **,** ‹pause time›

Parameter 1: ‹number of steps›

This parameter specifies how many different steps of tone (pitch) you want the sound to pass through during the envelope section. For example, in a section of a note which lasts 10 seconds, you may wish to have 10 tone steps of 1 second each. In such a case, the ‹number of steps› parameter used should be 10.

The available range of ‹number of steps› is 0 to 239.

Parameter 2: ‹step size›

This parameter must be in the range −128 to +127. Negative steps make the pitch of the note higher; positive steps make the pitch of the note lower. The shortest tone period is 0. The full range of tone periods is shown in the chapter entitled 'For your reference....'.

Parameter 3: ‹pause time›

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of ‹pause time› numbers is 0 to 255 (where 0 is treated as 256), which means that the longest time between steps is 2.56 seconds.

If 2 parameters are used, these are:

‹tone period› **,** ‹pause time›

Parameter 1: ‹tone period›

This parameter gives a new absolute setting for the tone period. (See Parameter 2 of the **SOUND** command.)

Parameter 2: ‹pause time›

This parameter specifies the pausing time in 0.01 second (hundredths of a second) units. The range of ‹pause time› numbers is 0 to 255 (where 0 is treated as 256), or 2.56 seconds.

General

Note that the total length of all the ‹pause time›s should not be greater than the ‹duration› parameter in the **SOUND** command, otherwise the sound will finish before all the tone steps have been passed through. (In such a case, the remaining contents of the tone envelope are discarded.)

Likewise, if the ‹duration› parameter in the **SOUND** command is longer than the total length of all the ‹pause time›s, the sound will continue after all of the tone steps have been passed through, and will remain constant at the final tone pitch.

Up to 5 different ‹envelope section›s, (each made up of the above 2 or 3 parameters) may be used in an **ENT** command.

The first step of a tone envelope is executed immediately.

Each time a given tone envelope is set, its previous value is lost.

Specifying an ‹envelope number› with no ‹envelope section›s cancels any previous setting.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **ENV, SOUND**

ENV

ENV ‹envelope number›[, ‹envelope section›][, ‹envelope section›]
[, ‹envelope section›][, ‹envelope section›]
[, ‹envelope section›]

```
10 ENV 1,15,-1,10,15,1,10
20 SOUND 1,200,300,15,1
run
```

COMMAND: Sets the Volume ENvelope specified in the ‹envelope number› (in the range 1 to 15), which is used in conjunction with the SOUND command.

Each of the ‹envelope section›s may contain either 2 or 3 parameters:

If 3 parameters are used, these are:

‹number of steps› , ‹step size› , ‹pause time›

Parameter 1: ‹number of steps›

This parameter specifies how many different steps of volume you want the sound to pass through during the envelope section. For example, in a section of a note which lasts 10 seconds, you may wish to have 10 volume steps of 1 second each. In such a case, the ‹number of steps› parameter used should be 10.

The available range of ‹number of steps› is 0 to 127.

Parameter 2: ‹step size›

Each step can vary in size from a volume level of 0 to 15 with respect to the previous step. The 16 different volume levels are the same as those you will hear in the SOUND command. However, the ‹step size› parameter used can be between −128 and +127; the volume level re-cycling to 0 after each 15.

Parameter 3: ‹pause time›

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of ‹pause time› numbers is 0 to 255 (where 0 is treated as 256), which means that the longest time between steps is 2.56 seconds.

If 2 parameters are used, these are:

‹hardware envelope› , ‹envelope period›

Parameter 1: ‹hardware envelope›

This parameter specifies the value to send to the envelope shape register of the sound chip.

Parameter 2: ‹envelope period›

This parameter specifies the value to send to the envelope period registers of the sound chip.

Knowledge of hardware is assumed when using hardware envelopes. Unless you have such knowledge, it is suggested that you use a software envelope incorporating a suitable ‹pause time› parameter.

General

Note that the total length of all the ‹pause time›s should not be greater than the ‹duration› parameter in the **SOUND** command, otherwise the sound will finish before all the volume steps have been passed through. (In such a case, the remaining contents of the volume envelope are discarded.)

Likewise, if the ‹duration› parameter in the **SOUND** command is longer than the total length of all the ‹pause time›s, the sound will continue after all of the volume steps have been passed through, and will remain constant at the final level.

Up to 5 different ‹envelope section›s, (each made up of the above 2 or 3 parameters) may be used in an **ENV** command.

The first step of a volume envelope is executed immediately.

Each time a given volume envelope is set, its previous value is lost.

Specifying an ‹envelope number› with no ‹envelope section›s cancels any previous setting.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **ENT**, **SOUND**

EOF

EOF

```
10 OPENIN "keys.wp"
20 WHILE NOT EOF
30 LINE INPUT #9,a$
40 PRINT a$
50 WEND
60 CLOSEIN
run
```

FUNCTION: Tests to see if the disc or cassette input is at End Of File. Returns -1 (true) if no file is open or file is at the end, otherwise returns 0 (false).

Associated keywords: OPENIN, CLOSEIN

ERASE

ERASE ‹list of: ‹variable name›

```
DIM a(100),b$(100)
ERASE a,b$
```

COMMAND: Erases the contents of an array no longer required, reclaiming the memory for other use.

Associated keywords: DIM

ERL

ERL

```
10 ON ERROR GOTO 30
20 GOTO 1000
30 PRINT "error is in line";ERL
40 END
run
```

FUNCTION: Reports the Line number of the last ERror encountered. In the above example you will see that the error is in line 20, and has been reported so by the ERL function.

Associated keywords: DERR, ERR, ERROR, ON ERROR GOTO, RESUME

ERR

ERR

```
GOTO 500
Line does not exist
Ready
PRINT ERR
 8
```

FUNCTION: Reports the number of the last ERRor encountered. See the listing of error messages given in the chapter entitled 'For your reference....'. In the above example you will see that ERRor number 8 is a 'Line does not exist' error.

Associated keywords: DERR, ERL, ERROR, ON ERROR GOTO, RESUME

ERROR

ERROR ‹integer expression›

```
10 IF INKEY$="" THEN 10 ELSE ERROR 17
run
```

COMMAND: Invokes the error specified in the ‹integer expression›. A listing of error messages 1 to 32 is given in the chapter entitled 'For your reference....'. BASIC will treat the ERROR as if it had been detected as genuine, and will jump to any error handling routine, as well as reporting the appropriate values of ERR and ERL.

ERROR accompanied by an ‹integer expression› in the range 33 to 255 can be used to create customised error messages, as shown in the following example:

```
10 ON ERROR GOTO 100
20 INPUT "enter one character";a$
30 IF LEN(a$)<>1 THEN ERROR 100
40 GOTO 20
100 IF ERR=100 THEN 110 ELSE 130
110 PRINT CHR$(7)
120 PRINT "I said ONE character!"
130 RESUME 20
run
```

Associated keywords: ERL, ERR, ON ERROR GOTO, RESUME

EVERY

EVERY ‹time period›[, ‹timer number›] GOSUB ‹line number›

```
10 EVERY 50,1 GOSUB 30
20 GOTO 20
30 SOUND 1,20
40 RETURN
run
```

COMMAND: Calls a BASIC sub-routine at regular intervals. The ‹time period› specifies the interval in units of 0.02 (fiftieths) second.

The ‹timer number› (in the range 0 to 3) specifies which of the four delay timers are to be used. Timer 3 has the highest priority; timer 0 (the default timer) has the lowest.

Each of the timers may have a sub-routine associated with it.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, REMAIN

EXP

EXP(‹numeric expression›)

```
PRINT EXP(6.876)
 968.743625
```

FUNCTION: Calculates 'E' to the power given in the ‹numeric expression›, where 'E' is approximately 2.7182818 - the number whose natural logarithm is 1.

Associated keywords: LOG

FILL

FILL ‹ink›

```
10 MODE 0
20 FOR n=1 TO 500
30 PRINT "0";
40 NEXT
50 pencolour=2+RND*13
60 FILL pencolour
70 GOTO 50
run
```

COMMAND: Fills an arbitrary area of the graphics screen. The edges of the area are bounded by lines drawn either in the current graphics pen ink or in the ink being used to fill (in the range 0 to 15).

The fill starts from the current graphics cursor position, If this position lies on an edge, nothing will be filled.

Associated keywords: GRAPHICS PEN

FIX

FIX(‹numeric expression›)

```
PRINT FIX(9.99999)
 9
```

FUNCTION: Removes the part of ‹numeric expression› to the right of the decimal point, rounding towards zero.

Associated keywords: CINT, INT, ROUND

FN

(See DEF FN)

FOR

FOR ‹simple variable›=‹start› TO ‹end› [STEP ‹size›]

```
10 FOR n=2 TO 8 STEP 2
20 PRINT n;
30 NEXT n
40 PRINT ",who do we appreciate?"
run
```

COMMAND: Carries out the body of program between the FOR and NEXT commands, a given number of times, stepping the control variable between a ‹start› and ‹end› value. If the STEP ‹size› is not specified, 1 is assumed.

The STEP ‹size› may be specified as a negative ‹numeric expression› in which case the value of the ‹start› parameter must be greater than that of the ‹end› parameter, otherwise the control variable will not be stepped.

FOR NEXT loops may be 'nested', i.e. one may be carried out within another, within another, and so on.

Assigning the variable's name to the NEXT command is optional as BASIC will automatically find which FOR command is to be associated with an 'anonymous' NEXT.

Associated keywords: NEXT, STEP, TO

FRAME

FRAME

```
10 MODE 0
20 PRINT "FRAME off"
30 TAG
40 MOVE 0,200
50 FOR x=0 TO 500 STEP 4
60 IF f=1 THEN FRAME
70 MOVE x,200
80 PRINT " ";CHR$(143);
90 NEXT
100 IF f=1 THEN RUN
110 CLS
120 TAGOFF
130 PRINT "FRAME on"
140 f=1
150 GOTO 30
run
```

COMMAND: Synchronises the writing of graphics on the screen, with the frame flyback of the display. The overall effect of this is that character or graphics movement on the screen will appear to be smoother, without 'flickering' or 'tearing'.

Associated keywords: TAG, TAGOFF

FRE

FRE(‹numeric expression›)
FRE(‹string expression›)

```
PRINT FRE(0)
PRINT FRE("")
```

FUNCTION: Establishes how much FREe memory remains unused by BASIC. The form FRE("") forces a 'garbage collection' before returning a value for available space.

NOTE - BASIC uses only the first 64K of the memory.

Associated keywords: HIMEM, MEMORY

GOSUB

GOSUB ‹line number›

```
GOSUB 210
```

COMMAND: GOes to a BASIC SUB-routine by branching to the specified ‹line number›. The end of the sub-routine itself is marked by the command RETURN, whereupon the program continues execution from the instruction after the invoked GOSUB command.

Associated keywords: RETURN

GOTO

GOTO ‹line number›

```
GOTO 90
```

COMMAND: GOes TO a specified line number.

Associated keywords: **none**

GRAPHICS PAPER

GRAPHICS PAPER ‹ink›

```
10 MODE 0
20 MASK 15
30 GRAPHICS PAPER 3
40 DRAW 640,0
run
```

COMMAND: Sets the ‹ink› of the graphics paper, i.e. the area behind graphics drawn on the screen. When drawing continuous lines, the graphics paper will not be seen. In the above example, the MASK command enables a broken line to be drawn, and the graphics paper to be seen.

The graphics paper's ink (in the range 0 to 15) is used for the 'paper' area of characters written when TAG is in operation, and as the default when clearing the graphics window, using CLG.

Associated keywords: CLG, GRAPHICS PEN, INK, MASK, TAG, TAGOFF

GRAPHICS PEN

GRAPHICS PEN [‹ink›][,‹background mode›]

```
10 MODE 0
20 GRAPHICS PEN 15
30 MOVE 200,0
40 DRAW 200,400
50 MOVE 639,0
60 FILL 15
run
```

COMMAND: Sets the ‹ink›, (in the range 0 to 15) to be used for drawing lines and plotting points. The graphics ‹background mode› can also be set to either:

0 : Opaque background
1 : Transparent background

(Transparent background affects the graphics paper of characters written with TAG on, and the gaps in dotted lines.)

Either parameter may be omitted, but not both. If a parameter is omitted, that particular setting is not changed.

Associated keywords: GRAPHICS PAPER, INK, MASK, TAG, TAGOFF

HEX$

HEX$(‹unsigned integer expression›[,‹field width›])

```
PRINT HEX$(255,4)
00FF
```

FUNCTION: Produces a $tring of HEXadecimal digits representing the value of the ‹unsigned integer expression›, using the number of hexadecimal digits instructed by the ‹field width› (in the range 0 to 16). If the number of digits instructed is too great, the resulting expression will be filled with leading zeros; if the number of digits instructed is too small, the resulting expression will NOT be shortened to the instructed number of digits, but will be produced in as many digits as are required.

The ‹unsigned integer expression› to be converted into hexadecimal form must yield a value in the range −32768 to 65535.

Associated keywords: BIN$, DEC$, STR$, UNT

HIMEM

HIMEM

```
PRINT HIMEM
42619
```

FUNCTION: Returns the address of the HIghest byte of MEMory used by BASIC, (which may be altered by the MEMORY command).

NOTE - BASIC uses only the first 64K of the memory

Associated keywords: FRE, MEMORY, SYMBOL, SYMBOL AFTER

IF

IF ‹logical expression› THEN ‹option part› [ELSE ‹option part›]

```
10 MODE 1
20 x=CINT(RND*100)
30 PRINT "Guess my number (0 to 100)"
40 INPUT n
50 IF n<x THEN PRINT n;"is too low..."
60 IF n>x THEN PRINT n;"is too high..."
70 IF n=x THEN 80 ELSE c=c+1:GOTO 40
80 PRINT "Well done, you got it in";
90 PRINT c+1;"guesses!"
run
```

COMMAND: Determines whether the ‹logical expression› is true, in which case the first ‹option part› is executed. If the ‹logical expression› is false, any ‹option part› specified in the ELSE clause is executed, otherwise BASIC passes onto the next line.

IF THEN commands may be nested to any depth, and are terminated by end of line. Therefore it is NOT possible to have further statements which are independent of the IF THEN command, on the same line.

Where the result of the ‹logical expression› requires that a line should be jumped to, the command may be given as either:

Examples....

```
IF a=1 THEN 100
```

....or....

```
IF a=1 GOTO 100
```

....or....

```
IF a=1 THEN GOTO 100
```

Associated keywords: ELSE, GOTO, THEN

INK

INK ‹ink› , ‹colour› [, ‹colour›]

```
10 MODE 1:PAPER 0:PEN 1
20 FOR p=0 TO 1
30 FOR i=0 TO 26
40 INK p,i
50 LOCATE 16,12:PRINT "ink";p;",";i
60 FOR t=1 TO 400:NEXT t,i,p
70 INK 0,1:INK 1,24:CLS
run
```

COMMAND: Assigns colour(s) to a given ink. The ‹ink› parameter describes the ink reference, which must be an integer expression in the range 0 to 15, for use in the appertaining PEN or PAPER command. The first ‹colour› parameter should be an integer expression yielding a colour value in the range 0 to 26. If an optional second ‹colour› is specified, the ink alternates between the two colours, at a rate determined by the SPEED INK command.

Associated keywords: GRAPHICS PAPER, GRAPHICS PEN, PAPER, PEN, SPEED INK

INKEY

INKEY (‹integer expression›)

```
10 IF INKEY(55)<>32 THEN 10
20 PRINT "You've pressed [SHIFT] and V"
30 CLEAR INPUT
run
```

FUNCTION: INterrogates the KEYboard to report which keys are being pressed. The keyboard is scanned every 0.02 (fiftieth) second.

The function is useful for spotting whether a certain key is down or up, by detecting the returned value of −1 (which occurs regardless of **[SHIFT]** and **[CONTROL]** key status).

The above example detects when **[SHIFT]** and **V** (key number 55) are pressed together, then ends the program. An illustration of key numbers will be found in the diagram at the top right hand side of the computer, and in the chapter entitled 'For your reference....'.

The state of **[SHIFT]** and **[CONTROL]** in conjunction with the key specified in the ‹integer expression› is identified as follows:

Value returned	**[SHIFT]**	**[CONTROL]**	specified key
-1	UP/DOWN	UP/DOWN	UP
0	UP	UP	DOWN
32	DOWN	UP	DOWN
128	UP	DOWN	DOWN
160	DOWN	DOWN	DOWN

Associated keywords: `CLEAR INPUT, INKEY$, JOY`

INKEY$

`INKEY$`

```
10 CLS
20 PRINT "Select Yes or No (Y/N)?"
30 a$=INKEY$
40 IF a$="" THEN 30
50 IF a$="y" OR a$="Y" THEN 80
60 IF a$="n" OR a$="N" THEN 90
70 GOTO 30
80 PRINT "You have selected YES":END
90 PRINT "You have selected NO"
run
```

FUNCTION: `IN`terrogates the `KEY`board, returning the current $tring reflecting any key that is pressed. If no key is pressed, `INKEY$` returns an empty string. In the above example, lines `40` and `70` tell the program to loop back to line `30` after interrogating the keyboard string.

Associated keywords: `CLEAR INPUT, INKEY`

INP

`INP(`‹port number›`)`

```
PRINT INP(&FF77)
 255
```

FUNCTION: Returns the `INP`ut value from the I/O address specified in the ‹port number›.

Associated keywords: `OUT, WAIT`

INPUT

INPUT[#‹stream expression›,][;][‹quoted string›‹separator›]
‹list of:‹variable›

```
10 MODE 1
20 INPUT "Give me two numbers for multiplication,
   (separated by a comma)";a,b
30 PRINT a;"times";b;"is";a*b
40 GOTO 20
run
```

COMMAND: Accepts data input from the stated stream, (from stream #0 if not specified).

The first, optional semicolon [;] suppresses the carriage return/line feed that will otherwise occur after the command is executed.

The ‹separator› must either be a semicolon or comma. A semicolon causes a question mark to be displayed; a comma suppresses the question mark.

If an entry is made that is of the wrong type such as if the letter o is typed instead of a 0 (zero) when INPUTing a numeric variable, then BASIC will respond with:

```
?Redo from start
```

....and any original prompt text that you programmed.

All responses from the keyboard must be terminated by pressing the **[RETURN]** key.

Associated keywords: LINE INPUT, OPENIN

INSTR

INSTR([‹start position›,]‹searched string›,‹searched for string›)

```
10 CLS
20 alphabet$=alphabet$+CHR$(n+64)
30 INPUT "Enter a letter";a$
40 b$=UPPER$(a$)
50 PRINT b$;" is number";
60 PRINT INSTR(alphabet$,b$);
70 PRINT "in the alphabet.":PRINT
80 GOTO 40
run
```

FUNCTION: Searches the first ‹searched string› expression to find the ‹searched for string› expression, and reports the position of its first occurrence within the ‹searched string›. If the ‹searched for string› does not occur within the ‹searched string›, then 0 is reported.

The position at which to start searching the ‹searched string› is optionally specifiable using the ‹start position› parameter which must yield an integer number in the range 1 to 255.

Associated keywords: **none**

INT

INT(‹numeric expression›)

```
PRINT INT(-1.995)
-2
```

FUNCTION: Rounds the number to the nearest smaller INTeger, removing any fractional part. Returns the same value as FIX for positive numbers, but returns one less than FIX for negative numbers which are not already integers.

Associated keywords: CINT, FIX, ROUND

JOY

JOY(‹integer expression›)

```
10 PRINT "To stop the program - ";
20 PRINT "operate joystick"
30 IF JOY(0)<>0 THEN END
40 GOTO 10
run
```

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

Hence for example, if the main 'fire' button (Fire 2) on the first joystick is pressed while the joystick handle is being moved left, the function JOY(0) returns a decimal value of 20, corresponding to 16 (Fire 2) + 4 (Left).

Further information concerning joysticks will be found in the chapter entitled 'For your reference....'.

Associated keywords: CLEAR INPUT, INKEY

KEY

KEY ‹expansion token number›,‹string expression›

```
KEY 11,"border 13:paper 0:pen 1:ink 0,13:
ink 1,0:mode 2:list"+CHR$(13)
```

....now press the **[ENTER]** key.

COMMAND: Assigns the ‹string expression› to the key's ‹expansion token number› specified. Thirty-two expansion tokens are supported, in the range 0 to 31, these occupying the key values 128 to 159. Keys 128 (0 on numeric keypad) to 140 (**[CONTROL] [ENTER]**) are by default assigned to print numbers 0 to 9, a decimal point, **[RETURN]** and RUN"**[RETURN]** - (for cassette operation), but may be re-assigned to other ‹string expression›s as required. Expansion tokens 13 to 31 (key values 141 to 159) are empty strings by default, but may be be expanded and assigned to keys, using the KEY DEF command described in the next example.

The ‹expansion token number› given in the KEY command may be in the range 0 to 31, or optionally 128 to 159 to reflect the key values. (See the key illustration in the chapter entitled 'For your reference....'.)

A total of 120 characters may be expanded into the ‹string expression›s. Attempting to over-expand will produce an '`Improper argument`' error (5).

Associated keywords: `KEY DEF`

KEY DEF

`KEY DEF` ‹key number› `,` ‹repeat› [`,` ‹normal› [`,` ‹shifted› [`,` ‹control›]]]

```
KEY 159,"this is the tab key"
KEY DEF 68,1,159
```

....now press the **[TAB]** key.

COMMAND: `DE`Fines the `KEY` values to be returned by the specified ‹key number› in the range 0 to 79 (for an illustration of key numbers, refer to the diagram at the top right hand side of the computer, or to the chapter entitled 'For your reference....'). The ‹normal›, ‹shifted›, and ‹control› parameters should contain the values required to be returned when the key is pressed, alone, together with **[SHIFT]**, and together with **[CONTROL]**, respectively. Each of these parameters is optional.

The ‹repeat› parameter enables you to set the key auto-repeat function on or off (`1` or `Ø`), the rate of auto-repeat being adjustable by use of the `SPEED KEY` command.

In the above example, key 159 (equivalent to expansion token 31) is first assigned to an expansion string, then the `KEY DEF` command defines key `68` (the **[TAB]** key) to auto-repeat (`1`), and to return the ‹normal› value `159` when pressed alone.

In the above example, normal action would be restored by:

```
KEY DEF 68,Ø,9
```

....where 9 is the normal ASCII value for **[TAB]**

Associated keywords: `KEY, SPEED KEY`

LEFT$

LEFT$(‹string expression›,‹required length›)

```
10 CLS
20 a$="AMSTRAD"
30 FOR n=1 TO 7
40 PRINT LEFT$(a$,n)
50 NEXT
run
```

FUNCTION: Returns the number of characters (in the range 0 to 255) specified in the ‹required length› parameter, after extracting them from the LEFT of the ‹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›)

```
10 LINE INPUT "Enter a phrase";a$
20 PRINT "The phrase is";
30 PRINT LEN(a$);"characters long."
run
```

FUNCTION: Returns the total number of characters (i.e. the LENgth) of the ‹string expression›.

Associated keywords: **none**

LET

LET ‹variable›=‹expression›

```
LET x=100
```

COMMAND: Assigns a value to a variable. A remnant from early BASICs where variable assignments had to be 'seen coming'. Has no use in AMSTRAD BASIC apart from providing compatibility with the programs supplied in early BASIC training manuals. The above example need only be typed in:

```
x=100
```

Associated keywords: **none**

LINE INPUT

LINE INPUT[#‹stream expression›,][;][‹quoted string›‹separator›]
‹string variable›

```
10 LINE INPUT "Type a line of text";a$
20 CLS
30 PRINT "The variable a$ now equals:-"
40 PRINT a$
run
```

COMMAND: Accepts input of an entire line of text from the stream indicated, (from stream #0 if not specified). The first, optional semicolon [;] suppresses the carriage return/line feed that will otherwise occur after the command is executed.

The ‹separator› must either be a semicolon or comma. A semicolon causes a question mark to be displayed; a comma suppresses the question mark.

LINE INPUT from the keyboard is terminated by pressing the **[RETURN]** key.

LINE INPUT from disc (or cassette) stream #9 is terminated by a received carriage return, or by the ‹string variable› being assigned 255 characters, whichever is the sooner.

Associated keywords: INPUT, OPENIN

LIST

LIST [‹line number range›][,#‹stream expression›]

```
LIST 100-1000,#1
```

COMMAND: Lists program lines to the given stream. Stream #0 is the default screen stream, and #8 is the printer. LISTing may be paused by pressing **[ESC]** once, and re-started by pressing the space bar. Pressing **[ESC]** twice will abandon the listing and return to direct mode.

The first or last number in the ‹line number range› may be omitted to indicate '....from the beginning of the program', or '....to the end of the program', i.e:

```
LIST -200
```

....or....

```
LIST 50-
```

....or....

```
LIST
```

....which lists the whole program.

Associated keywords: OPENOUT

LOAD

LOAD ‹filename›[,‹address expression›]

```
LOAD "discfile.xyz",&2AF8
```

COMMAND: Loads a BASIC program from disc or cassette into memory, replacing any existing program. Specifying the optional ‹address expression› will cause a binary file to be loaded at that address, rather than the address from which it was saved.

A Protected BASIC program can NOT be loaded using the LOAD command as it will be immediately deleted from memory. Instead, use the RUN or CHAIN commands.

464: You need not specify the filename if you wish the first suitable file on the cassette to be loaded.

You will be instructed:

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

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer will load the file to be processed.

The screen will display the loading messages:

```
Loading FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is loaded.

If the first character of the filename is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the **[ESC]** key produces the error message on the screen:

```
Broken in
```

Associated keywords: CHAIN, CHAIN MERGE, MERGE, RUN, SAVE

LOCATE

LOCATE[#‹stream expression›,]‹x co-ordinate›,‹y co-ordinate›

```
10 MODE 1
20 FOR n=1 TO 20
30 LOCATE n,n
40 PRINT CHR$(143);"location";
50 PRINT n;",";n
60 NEXT
run
```

COMMAND: Locates the text cursor at the stream indicated, to the position specified by the x and y co-ordinates, with 1,1 being the top left corner of the stream (window). Stream #0 is the default stream.

Associated keywords: WINDOW

LOG

LOG(‹numeric expression›)

```
PRINT LOG(9999)
 9.21024037
```

FUNCTION: Calculates the natural LOGarithm of the ‹numeric expression› which must be greater than zero.

Associated keywords: EXP, LOG10

LOG10

LOG10(‹numeric expression›)

```
PRINT LOG10(9999)
 3.99995657
```

FUNCTION: Calculates the LOGarithm to base 10 of the ‹numeric expression› which must be greater than zero.

Associated keywords: EXP, LOG

LOWER$

LOWER$(‹string expression›)

```
10 a$="SEE HOW THE LETTERS CHANGE TO "
20 PRINT LOWER$(a$+"LOWER CASE")
run
```

FUNCTION: Returns a new string expression which is a copy of the specified ‹string expression› but in which all alphabetic characters in the range A to Z are converted to lower case. The function is useful for processing input which may come in mixed upper/lower case.

Associated keywords: UPPER$

MASK

MASK[‹integer expression›][,‹first point setting›]

```
10 MODE 0:INK 5,21:INK 8,16
20 MOVE -100*RND,400*RND
30 WHILE XPOS<640
40 FOR x=1 TO 8
50 MASK 2↑(8-x)
60 DRAWR 32,0,x,1:MOVER -32,0
70 NEXT
80 MOVER 34,0
90 WEND:GOTO 20
run
```

COMMAND: Sets the 'mask' or template to be used when drawing lines. The binary value of the ‹integer expression› in the range 0 to 255, sets the bits in each adjacent group of 8 pixels to ON (1), or OFF (0).

The ‹first point setting› determines whether the first point of the line is to be plotted (1) or not plotted (0).

Either of the parameters may be omitted, but not both. If a parameter is omitted, that particular setting is not changed.

Associated keywords: DRAW, DRAWR, GRAPHICS PAPER, GRAPHICS PEN

MAX

MAX(‹list of:‹numeric expression›)

```
10 n=66
20 PRINT MAX(1,n,3,6,4,3)
run
 66
```

FUNCTION: Returns the MAXimum value from the ‹list of:‹numeric expression›s.

Associated keywords: MIN

MEMORY

MEMORY ‹address expression›

```
MEMORY &20AA
```

COMMAND: Allocates the amount of BASIC memory available by setting the address of the highest byte.

Associated keywords: FRE, HIMEM, SYMBOL, SYMBOL AFTER

MERGE

MERGE ‹filename›

```
MERGE "newload.bas"
```

COMMAND: Loads a program from disc or cassette, and adds it to the current program in the memory.

Note that line numbers in the old program which exist in the new program to be MERGEd, will be over-written by the new program lines.

Protected files, (SAVEd by ,p) can NOT be MERGEd into the current program.

464: You need not specify the filename if you wish the first suitable file on the cassette to be loaded.

You will be instructed:

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

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer will load the file to be processed.

The screen will display the loading messages:

```
Loading FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is loaded.

If the first character of the filename is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the **[ESC]** key produces the error message on the screen:

```
Broken in
```

Associated keywords: CHAIN, CHAIN MERGE, LOAD

MID$

MID$(‹string expression›,‹start position›[,‹sub-string length›])

```
10 MODE 1:ZONE 3
20 a$="ENCYCLOPAEDIA"
30 PRINT "Show me how to spell ";a$
40 PRINT "OK....":PRINT
50 FOR n=1 TO LEN(a$)
60 PRINT MID$(a$,n,1),
70 FOR t=1 TO 700:NEXT t,n
80 PRINT:PRINT
90 INPUT "Now enter another word";a$
100 GOTO 50
run
```

FUNCTION: Returns a new sub-string, commencing at the ‹start position› of the ‹string expression›, being ‹sub-string length› characters long. If the ‹sub-string length› parameter is not specified, the remainder of the ‹string expression› after the ‹start position› is returned.

If the ‹start position› is greater than the total length of the ‹string expression›, then an empty string is returned. The range of ‹start position› is 1 to 255. The range of ‹sub-string length› is 0 to 255.

Associated keywords: LEFT$, RIGHT$

MID$

MID$(‹string variable›,‹insertion position›[,‹new string length›])
=‹new string expression›

```
10 a$="hello"
20 MID$(a$,3,2)="XX"
30 PRINT a$
run
heXXo
```

COMMAND: Inserts the ‹new string expression› into the string specified by the ‹string variable›, commencing at the ‹insert position›, and occupying ‹new string length› number of characters.

Note that when using MID$ as a COMMAND, a ‹string variable› such as a$ must be used, and NOT a string constant such as "hello".

Associated keywords: LEFT$, RIGHT$

MIN

MIN(‹list of:‹numeric expression››)

```
PRINT MIN(3,6,2.999,8,9,)
 2.999
```

FUNCTION: Returns the MINimum value from the ‹list of:‹numeric expression›s.

Associated keywords: MAX

MOD

‹argument› MOD ‹argument›

```
PRINT 10 MOD 3
 1
PRINT 10 MOD 5
 0
```

OPERATOR: Returns the remainder after dividing the first ‹argument› by the second ‹argument› and removing any integer component - MODulus.

Associated keywords: **none**

MODE

MODE ‹integer expression›

```
10 m=m+1:IF m>2 THEN m+0
20 MODE m
30 PRINT "this is mode";m
40 PRINT "now press a key"
50 IF INKEY$="" THEN 50 ELSE 10
run
```

COMMAND: Changes the screen mode (0,1 or 2), and clears the screen to ink 0 (which may not be the current paper ink). All text and graphics windows are reset to the whole screen, and text and graphics cursors are homed to their respective origins.

Associated keywords: ORIGIN, WINDOW

MOVE

MOVE ‹x co-ordinate› , ‹y co-ordinate›[,[‹ink›][, ‹ink mode›]]

```
10 MODE 1:TAG
20 x=RND*800-100:y=RND*430
30 MOVE x,y
40 PRINT "I moved here";
50 GOTO 20
run
```

COMMAND: Moves the graphics cursor to the absolute point specified by ‹x co-ordinate› and ‹y co-ordinate›. The optional ‹ink› parameter may be used to change the graphics pen ink, in the range 0 to 15.

The optional ‹ink mode› determines how the ink to be written next, will interact with that already on the graphics screen. The 4 ‹ink mode›s are:

0 : Normal
1 : XOR (eXclusive OR)
2 : AND
3 : OR

Associated keywords: MOVER, ORIGIN, XPOS, YPOS

MOVER

MOVER ‹x offset› , ‹y offset›[,[‹ink›][, ‹ink mode›]]

```
10 MODE 1:TAG:MOVE 0,16
20 PRINT "life has its";
30 FOR n=1 TO 10
40 MOVER -32,16
50 PRINT "ups";:NEXT:PRINT " and";
60 FOR n=1 TO 10
70 MOVER -64,-16
80 PRINT "downs";:NEXT
run
```

COMMAND: Moves the graphics cursor to a point relative to its current position. The relative position is specified by ‹x offset› and ‹y offset›. The optional ‹ink› parameter may be used to change the graphics pen ink, in the range 0 to 15.

The optional ‹ink mode› determines how the ink to be written next, will interact with that already on the graphics screen. The 4 ‹ink mode›s are:

0 : Normal
1 : XOR (eXclusive OR)
2 : AND
3 : OR

Associated keywords: MOVE, ORIGIN, XPOS, YPOS

NEW

NEW

```
    NEW
```

COMMAND: Deletes the current program and variables in the memory. Key definitions are not lost, display characteristics i.e. MODE, PEN, PAPER, INK etc, are not changed, and the screen is not cleared.

Associated keywords: **none**

NEXT

NEXT [‹list of:‹variable›]

```
10 FOR a=1 TO 3
20 FOR b=0 TO 26
30 MODE 1
40 PEN a:BORDER b
50 PRINT "pen";a;"border";b
60 FOR c=1 TO 500
70 NEXT c,b,a
run
```

COMMAND: Marks the end of a FOR loop. The NEXT command may be anonymous, or may refer to its matching FOR. Note from the above example that the ‹list of:‹variable›s must appear in reverse order to their matching FOR commands, so that 'nested' loops do not overlap.

Associated keywords: FOR, STEP, TO

NOT

NOT ‹argument›

```
IF NOT "alan"<"bob" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
IF NOT "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
correct
....
PRINT NOT -1
 0
PRINT NOT 0
-1
```

OPERATOR: Performs bit-wise operations on integers. Inverts each bit in the argument.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AND, OR, XOR

ON BREAK CONT

`ON BREAK CONT`

```
10 ON BREAK CONT
20 PRINT "The program will CONTinue when you try  to
   *Break* using [ESC]":PRINT
30 FOR t=1 TO 1000:NEXT:GOTO 20
run
```

COMMAND: Cancels the action of the **[ESC]** key from stopping the program, and instead `CONT`inues execution. Care should be taken when using this command, as the program will continue until the computer is completely reset; hence you should `SAVE` such a program before `RUN`ning it.

`ON BREAK CONT` may be disabled within a program by `ON BREAK STOP`.

Associated keywords: `ON BREAK GOSUB`, `ON BREAK STOP`

ON BREAK GOSUB

`ON BREAK GOSUB` ‹line number›

```
10 ON BREAK GOSUB 40
20 PRINT "program running"
30 GOTO 20
40 CLS:PRINT "Pressing [ESC] ";
50 PRINT "twice calls GOSUB-routine"
60 FOR t=1 TO 2000:NEXT
70 RETURN
run
```

COMMAND: Instructs BASIC to jump to the sub-routine specified in the ‹line number› when the **[ESC]** key is pressed twice.

Associated keywords: `ON BREAK CONT`, `ON BREAK STOP`, `RETURN`

ON BREAK STOP

```
ON BREAK STOP
```

```
10 ON BREAK GOSUB 40
20 PRINT "program running"
30 GOTO 20
40 CLS:PRINT "Pressing [ESC] ";
50 PRINT "twice calls GOSUB-routine"
60 FOR t=1 TO 2000:NEXT
65 ON BREAK STOP
70 RETURN
run
```

COMMAND: Disables the `ON BREAK CONT` and `ON BREAK GOSUB` command, so that future operations of the **[ESC]** key stop the program. In the above example, the `ON BREAK GOSUB` command will operate once only, as it is then disabled by line 65 in the `ON BREAK` sub-routine.

Associated keywords: `ON BREAK CONT, ON BREAK GOSUB`

ON ERROR GOTO

`ON ERROR GOTO` ‹line number›

```
10 ON ERROR GOTO 60
20 CLS:PRINT "If error is found, ";
30 PRINT "then list the program"
40 FOR t=1 TO 4000:NEXT
50 GOTO 100
60 PRINT "Error detected in line";
70 PRINT ERL:PRINT:LIST
run
```

COMMAND: Jumps to the specified ‹line number› when an error is detected in the program.

The form of the command `ON ERROR GOTO 0` turns off the error trap, and restores normal error processing by BASIC.

See also the `RESUME` command.

Associated keywords: `DERR, ERL, ERR, ERROR, RESUME`

ON ‹expression› GOSUB

ON ‹selector› GOSUB ‹list of:‹line number›

```
10 PAPER 0:PEN 1:INK 0,1
20 CLS:PRINT "MENU OF OPTIONS":PRINT
30 PRINT "1 - Change border":PRINT
40 PRINT "2 - Change pen":PRINT
50 PRINT "3 - Change mode":PRINT
60 INPUT "Enter your selection";x
70 ON x GOSUB 90,110,130
80 GOTO 20
90 b=b-1:IF b<0 THEN b=26
100 BORDER b:RETURN
110 p=p-1:IF p<2 THEN p=26
120 INK 1,p:RETURN
130 m=m-1:IF m<0 THEN m=2
140 MODE m:RETURN
run
```

COMMAND: Selects a sub-routine line to jump to, depending upon the value of the ‹selector›, which should be a positive integer expression in the range 0 to 255. The order of the ‹selector› values determines the ‹line number› to be selected from the ‹list of:‹line number›s. In the above example, selecting 1 makes BASIC jump to line **90**, selecting 2 jumps to line **110**, and 3 jumps to line **130**.

If the value of the ‹selector› is zero, or is higher than the amount of ‹line number›s listed in the command, then no sub-routine line will be selected.

Associated keywords: **RETURN**

ON ‹expression› GOTO

ON ‹selector› GOTO ‹list of:‹line number›

```
10 CLS:PRINT "MENU OF OPTIONS":PRINT
20 PRINT "1 - List program":PRINT
30 PRINT "2 - Edit and add":PRINT
40 PRINT "3 - Catalog disc":PRINT
50 INPUT "Enter your selection";n
60 ON n GOTO 80,90,100
70 GOTO 10
80 LIST
90 AUTO
100 CAT
run
```

COMMAND: Selects a line to jump to, depending upon the value of the ‹selector›, which should be a positive integer expression in the range 0 to 255. The order of the ‹selector› values determines the ‹line number› to be selected from the ‹list of:‹line number›s. In the above example, selecting 1 makes BASIC jump to line 80, selecting 2 jumps to line 90, and 3 jumps to line 100.

If the value of the ‹selector› is zero, or is higher than the amount of ‹line number›s listed in the command, then no line will be selected.

Associated keywords: **none**

ON SQ GOSUB

ON SQ (‹channel›) GOSUB ‹line number›

```
10 ENV 1,15,-1,1
20 ON SQ(1) GOSUB 60
30 MODE 0:ORIGIN 0,0,200,440,100,300
40 FOR x=1 TO 13:FRAME:MOVE 330,200,x
50 FILL x:NEXT:GOTO 40
60 READ s:IF s=0 THEN RESTORE:GOTO 60
70 SOUND 1,s,25,15,1
80 ON SQ(1) GOSUB 60:RETURN
90 DATA 50,60,90,100,35,200,24,500,0
run
```

COMMAND: GOes to a BASIC SUB-routine when there is a free slot in the given Sound Queue. The ‹channel› should be an integer expression yielding one of the values:

1: for channel A
2: for channel B
4: for channel C

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: RETURN, SOUND, SQ

OPENIN

OPENIN ‹filename›

```
10 REM OPEN an INput file.
20 OPENIN "datafile":INPUT #9,a,a$
30 CLOSEIN:PRINT "The 2 values are:"
40 PRINT:PRINT a,a$
run
```

COMMAND: OPENs an INput file from disc or cassette, for use in the current program. The INput file to OPEN must be an ASCII file.

The above example will only work after you have created the file shown in the next example (under OPENOUT).

464: You need not specify the filename if you wish the first suitable file on the cassette to be loaded.

You will be instructed:

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

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer loads the first 2K bytes of the file into a portion of the memory called the 'file buffer'. Input is taken from the file buffer until it is empty, and the computer again prompts:

```
Press PLAY then any key
```

....and loads the next 2K bytes from the file.

The screen will display the loading messages:

```
Loading FILENAME block 1
```

....and other block numbers in turn, as the file is loaded.

If the first character of the filename in the OPENIN command is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning file input using the **[ESC]** key produces the error message on the screen:

```
Broken in
```

Associated keywords: CLOSEIN, EOF

OPENOUT

OPENOUT ‹filename›

```
10 REM OPEN an OUTput file.
20 INPUT "give me a number variable";a
30 INPUT "give me a string variable";a$
40 OPENOUT "datafile"
50 WRITE #9,a,a$
60 CLOSEOUT:PRINT "Data saved."
run
```

COMMAND: OPENs an OUTput file to disc or cassette.

464: You need not specify the filename if you wish the file to be saved as an Unnamed file.

The first 2K bytes of the file to be saved to cassette will first be written into a portion of the memory called the 'file buffer'. When the file buffer is full, you will be instructed:

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

....whereupon you should press the **RECORD** and **PLAY** buttons on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer saves the contents of the file buffer. The computer then refills the file buffer with the next 2K bytes of the file, and again prompts:

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

....and saves the next 2K bytes to cassette.

If the file buffer is partly full and the command CLOSEOUT is encountered, the computer will save the remaining contents of the file buffer to cassette, issuing the prompt:

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

The screen will display the saving message:

```
Saving FILENAME block ‹x›
```

If the first character of the filename in the OPENOUT command is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to save. (You must make sure that the **RECORD** and **PLAY** buttons on your cassette unit are down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning file output using the **[ESC]** key produces the error message on the screen:

```
Broken in
```

Associated keywords: CLOSEOUT

OR

‹argument› OR ‹argument›

```
IF "alan"<"bob" OR "dog">"cat" THEN PRINT "correct" ELSE PRINT "wrong"
correct

IF "bob"<"alan" OR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong

IF "alan"<"bob" OR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
correct

....

PRINT 1 OR 1
 1
PRINT 0 OR 0
 0
PRINT 1 OR 0
 1
```

OPERATOR: Performs bit-wise boolean operation on integers. Result is 1 unless both argument bits are 0.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AND, NOT, XOR

ORIGIN

ORIGIN ‹x›,‹y›[,‹left›,‹right›,‹top›,‹bottom›]

```
10 MODE 1:BORDER 13:TAG
20 ORIGIN 0,0,100,540,300,100
30 GRAPHICS PAPER 3:CLG
40 FOR x=550 TO -310 STEP -10
50 MOVE x,206
60 PRINT "This is a graphics window ";
70 FRAME:NEXT:GOTO 40
run
```

COMMAND: Sets the graphics origin points 0,0 to the position specified by the co-ordinates ‹x› and ‹y›.

A graphics window's dimensions may also be set by specifying the last four optional parameters. If the co-ordinates specified for the graphics window describe points beyond the edge of screen, then the edge of the graphics window is taken as edge of screen.

Associated keywords: CLG

OUT

OUT ‹port number›,‹integer expression›

```
OUT &F8F4,&FF
```

COMMAND: Sends the value in the ‹integer expression› (in the range 0 to 255) OUT to the address specified in the ‹port number›.

Not a command to be used by the unwary.

Associated keywords: INP, WAIT

PAPER

PAPER[#‹stream expression›,]‹ink›

```
10 MODE 0:PEN 0:INK 0,13
20 FOR p=1 TO 15
30 PAPER p:CLS
40 LOCATE 7,12:PRINT "paper";p
50 FOR t=1 TO 500:NEXT t,p
run
```

COMMAND: Sets the background ink for characters. When characters are written to the text screen, the character cell is filled with the paper ‹ink› (in the range 0 to 15) before the character is written, (unless the transparent mode has been selected).

If the ‹stream expression› is omitted, the PAPER ink for stream #0 is assumed by default.

The number of different PAPER inks supported is dependent upon the screen MODE.

Associated keywords: GRAPHICS PAPER, INK, PEN

PEEK

PEEK(‹address expression›)

```
10 MODE 1:ZONE 7
20 WINDOW 1,40,1,2:WINDOW #1,1,40,3,25
30 PRINT "memory-address"
40 LOCATE 20,1:PRINT "memory-contents"
50 FOR n=0 TO 65535
60 p=PEEK(n)
70 PRINT #1,n,"(&";HEX$(n);")";
80 PRINT #1,TAB(20);p,"(&";HEX$(p);")"
90 NEXT
run
```

FUNCTION: Reports the contents of the Z80 memory location specified in the ‹address expression› which should be in the range &0000 to &FFFF (0 to 65535). In all cases PEEK will return the value at the RAM address specified (not the ROM), and will be in the range &00 to &FF (0 to 255).

Associated keywords: POKE

PEN

PEN[#‹stream expression›,][‹ink›][,‹background mode›]

```
10 MODE 0:PAPER 0:INK 0,13
20 FOR p=1 TO 15
30 PEN p:PRINT SPACE$(47);"pen";p
40 FOR t=1 TO 500:NEXT t,p:GOTO 20
run
```

COMMAND: Sets the ‹ink› (in the range 0 to 15) to be used when writing to the given screen stream, (stream #0 if not specified). The ‹background mode› parameter can be set to transparent (1) or opaque (0).

Either of the last 2 parameters may be omitted, but not both. If a parameter is omitted, that particular setting is not changed.

Associated keywords: PAPER

PI

PI

```
PRINT PI
 3.14159265
```

FUNCTION: Returns the value of the ratio between circumference and diameter of a circle.

Associated keywords: DEG, RAD

PLOT

PLOT ‹x co-ordinate›,‹y co-ordinate›[,[‹ink›][,‹ink mode›]]

```
10 MODE 1:BORDER 0:PAPER 0:PEN 1
20 INK 0,0:INK 1,26:INK 2,13,26:DEG
30 FOR x=1 TO 360:ORIGIN 320,200
40 DRAW 50*COS(x),50*SIN(x),1
50 PLOT 100*COS(x),25*SIN(x):NEXT
60 ORIGIN 0,0:t=TIME+700:WHILE TIME<t
70 PLOT RND*640,RND*400:WEND
80 PLOT RND*640,RND*400,2
90 GOTO 90
run
```

COMMAND: Plots a point on the graphics screen, at the absolute position specified in the x,y co-ordinates. The ‹ink› in which to plot the point may be specified (in the range 0 to 15).

The optional ‹ink mode› determines how the ink being written interacts with that already on the graphics screen. The 4 ‹ink mode›s are:

0 : Normal
1 : XOR (eXclusive OR)
2 : AND
3 : OR

Associated keywords: GRAPHICS PEN, PLOTR

PLOTR

PLOTR ‹x offset› , ‹y offset›[, [‹ink›][, ‹ink mode›]]

```
10 REM use cursor keys to draw lines
20 BORDER 0:GRAPHICS PEN 1
30 MODE 1:PLOT 320,200
40 IF INKEY(0)=0 THEN PLOTR 0,1
50 IF INKEY(1)=0 THEN PLOTR 1,0
60 IF INKEY(2)=0 THEN PLOTR 0,-1
70 IF INKEY(8)=0 THEN PLOTR -1,0
80 IF INKEY(9)=0 THEN 30:REM copy=clear
90 GOTO 40
run
```

COMMAND: Plots a point on the graphics screen at the specified position ‹x offset› and ‹y offset›, relative to the current graphics cursor position. The ‹ink› in which to plot the point may be specified (in the range 0 to 15).

The optional ‹ink mode› determines how the ink being written interacts with that already on the graphics screen. The 4 ‹ink mode›s are:

0 : Normal
1 : XOR (eXclusive OR)
2 : AND
3 : OR

Associated keywords: GRAPHICS PEN, PLOT,

POKE

POKE ‹address expression›,‹integer expression›

```
10 FOR m=49152 TO 65535
20 POKE m,100
30 NEXT
run
```

COMMAND: Writes the value of the ‹integer expression› (in the range 0 to 255) directly into the Z80 memory (RAM) at the specified ‹address expression›.

Not a command to be used by the unwary.

Associated keywords: PEEK

POS

POS(#‹stream expression›)

```
10 MODE 1:BORDER 0:LOCATE 8,2
20 PRINT "use cursor left/right keys"
30 WINDOW 1,40,12,12:CURSOR 1,1
40 FOR n=1 TO 19:PRINT CHR$(9);:NEXT
50 IF INKEY(1)<>-1 THEN PRINT CHR$(9);
60 IF INKEY(8)<>-1 THEN PRINT CHR$(8);
70 LOCATE #1,2,24
80 PRINT #1,"text cursor ";
90 PRINT #1,"horizontal position =";
100 PRINT #1,POS(#0):GOTO 50
run
```

FUNCTION: Reports the current horizontal POSition of the text cursor relative to the left edge of the text window. The ‹stream expression› MUST be specified, and does NOT default to #0.

POS(#8) reports the current horizontal carriage position for the printer, where 1 is the extreme left hand edge.

POS(#9) reports the logical position in the disc file stream, i.e. the number of printing characters sent to the stream since the last carriage return.

Associated keywords: VPOS, WINDOW

PRINT

PRINT[#‹stream expression›,][‹list of:‹print items›]

```
10 a$="small"
20 b$="this is a larger string"
30 PRINT a$;a$
40 PRINT a$,a$
50 PRINT
60 PRINT b$;b$
70 PRINT b$,b$
run
```

COMMAND: Prints the ‹list of:‹print item›s to the given stream, (to stream #0 if no ‹stream expression› is specified).

Note that when a semicolon ; is used to tell the computer to print the following ‹print item› next to the preceding item, BASIC first checks to see if the following ‹print item› can fit onto the same line. If not, it will be printed on a new line regardless of the semicolon.

Note also that when a comma , is used to tell the computer to print the following ‹print item› in the next print zone, BASIC first checks to see that the preceding item has not exceeded the length of the current zone. If it has, the following ‹print item› is printed in a further zone.

PRINT SPC
PRINT TAB

PRINT[#‹stream expression›,][‹list of:‹print item›][;]
[SPC(‹integer expression›)][‹list of:‹print item›]

PRINT[#‹stream expression›,][‹list of:‹print item›][;]
[TAB(‹integer expression›)][‹list of:‹print item›]

```
10 PRINT "this is spc function"
20 FOR x=6 TO 15
30 PRINT SPC(5)"a";SPC(x)"b"
40 NEXT
50 PRINT "this is tab function"
60 FOR x=6 TO 15
70 PRINT TAB(5)"a";TAB(x)"b"
80 NEXT
run
```

SPC prints the number of spaces specified in the ‹integer expression›, and will print any following ‹print item› immediately next to the spaces, (assuming that the following ‹print item› will fit onto the line). Hence it is not necessary to terminate SPC with a semicolon.

TAB prints the number of spaces relative to the left edge of the text window, and will print any following ‹print item› immediately next to the spaces, (assuming that the following ‹print item› will fit onto the line). Hence it is not necessary to terminate TAB with a semicolon. If the current position is greater than the required position, then a carriage return is executed, followed by spaces to reach the required position on the next line.

PRINT USING

PRINT[#‹stream expression›][‹list of:‹print item›][;]
[USING ‹format template›][‹separator›‹expression›]

```
10 FOR x=1 TO 10
20 n=100000*(RND↑5)
30 PRINT "goods";USING "########,.##";n
40 NEXT
run
```

PRINT USING enables you to specify the print format of the expression returned by the PRINT command. This is achieved by specifying a ‹format template› to which the printed result must correspond. The ‹separator› is a comma or semicolon. The ‹format template› is a string expression which is constructed using the following 'format field specifiers':

Numeric Formats

Within the number:

\# Each # specifies a digit position.
Example template: ######

. Specifies the position of the decimal point.
Example template: ######.##

, (Specifies one digit position.) May appear BEFORE the decimal point only. Specifies that digits before the decimal point are to be divided into groups of three (for thousands), separated by commas.
Example template: ######,.##

Around the number:

`ff` (Specifies two digit positions.) Specifies that a `f` sign be printed immediately before the first digit or decimal point (after any leading sign). Note that the `f` will occupy one of the digit positions.
Example template: `ff######,.##`

`**` (Specifies two digit positions.) Specifies that any leading spaces be replaced by `*` asterisks.
Example template: `**######,.##`

`**f` (Specifies three digit positions.) Acts `**` and `ff` options combined, i.e. leading `*` asterisks and `f` sign.
Example template: `**f######,.##`

`$$` (Specifies two digit positions.) Specifies that a `$` sign be printed immediately before the first digit or decimal point (after any leading sign). Note that the `$` will occupy one of the digit positions.
Example template: `$$######,.##`

`**$` (Specifies three digit positions.) Acts as `**` and `$$` options combined, i.e. leading `*` asterisks and `$` sign.
Example template: `**$######,.##`

`+` Specifies that + or − is to be printed, as appropriate. If the + appears at the beginning of the template, the + sign is printed immediately before the the number (and any leading currency sign). If the + appears at the end of the template, the sign is printed after the number (and any exponent part).
Example template: `+####.####`

`-` The − sign may only appear at the END of a template. It specifies that − is to be printed after any negative number (and exponent part). If the number is positive, a space will be printed. A − sign is printed before a negative number by default, unless countermanded by the use of this template.
Example template: `####.####-`

`↑↑↑↑` Specifies that the number is to be printed using the exponent option. The `↑↑↑↑` in the template should appear AFTER the digit positions, but BEFORE any trailing + or − sign.
Example template: `#.####↑↑↑↑+`

The ‹format template› for a number may not exceed 20 characters. Numbers are rounded to the number of digits printed.

If the format template is too small for the input expression, for example:

```
PRINT USING "####";12345678
```

....the printed result is NOT shortened to fit the template, but is instead printed in its entirety, preceded by a % sign, to indicate 'format failure'.

String Formats

```
10 CLS:a$="abcdefghijklmnopqrst"
20 PRINT "input expression= ";a$
30 PRINT:PRINT "! specifier= ";
40 PRINT USING "!";a$
50 PRINT:PRINT "\spaces\ specifier= ";
60 PRINT USING "\       \";a$
70 PRINT:PRINT "& specifier= ";
80 PRINT USING "&";a$
90 GOTO 90
run
```

! Specifies that only the first character of the string is to be printed.
Example template: !

\‹spaces›\
Specifies that only the first x characters of the string are to be printed, where x is equal to the length of the template (including the back-slashes).
Example template: \ \

& Specifies that the entire string is to be printed 'as is'.
Example template: &

The ‹format template› for a string may not exceed 255 characters.

Both numeric and string ‹format template›s may be represented by string variables, for example:

```
10 a$="ff######,.##"
20 b$="!"
30 PRINT USING a$;12345.6789;
40 PRINT USING b$;"pence"
run
```

Further information concerning print formatting will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: SPC, TAB, USING, ZONE, OPENOUT

RAD

RAD

```
RAD
```

COMMAND: Sets RADians mode of calculation. BASIC defaults to radians when the computer is switched on or reset, or when the commands NEW, CLEAR, or LOAD, RUN, etc, are issued.

Associated keywords: ATN, COS, DEG, SIN, TAN

RANDOMIZE

RANDOMIZE [‹numeric expression›]

```
RANDOMIZE 123.456
PRINT RND
 0.258852139
```

COMMAND: Randomizes the number 'seed' specified in the ‹numeric expression›. BASIC's random number generator produces a pseudo-random sequence in which each number depends on the previous number, commencing at a given number seed. The sequence is always the same. RANDOMIZE sets the new initial value for the random number generator either to the specified value, or to a value entered by the user if the ‹numeric expression› is omitted.

RANDOMIZE TIME produces a sequence that is difficult to repeat.

Associated keywords: RND

READ

READ ‹list of: ‹variable›

```
10 FOR n=1 TO 8
20 READ a$,c
30 PRINT a$;" ";:SOUND 1,c:NEXT
40 DATA here,478,are,426,8,379,notes
50 DATA 358,of,319,a,284,musical,253,scale,239
run
```

COMMAND: Reads data from DATA statements and assigns it to variables, automatically stepping the 'pointer' to the next item in the DATA statement afterwards. The RESTORE command can be used to return the pointer to the beginning of a DATA statement.

Further information concerning data will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: DATA, RESTORE

RELEASE

RELEASE ‹sound channels›

```
10 SOUND 65,1000,100
20 PRINT "Press R to release the sound"
30 IF INKEY(50)=-1 THEN 30
40 RELEASE 1
run
```

COMMAND: Releases sound channels which are set to a 'hold' state in the SOUND command.

The parameter ‹sound channels› must yield an integer value in the range 1 to 7, which operates as follows:

1 : Releases channel A
2 : Releases channel B
3 : Releases channel A and B
4 : Releases channel C
5 : Releases channel A and C
6 : Releases channel B and C
7 : Releases channel A and B and C

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: SOUND

REM

REM ‹rest of line›

```
10 REM Intergalatic Hyperspace Mega-Monster
   Invaders Deathchase by AMSOFT
20 REM Copyright AMSOFT 1985
```

COMMAND: Inserts a REMark into a program. The ‹rest of line› is ignored by BASIC, and may contain any characters, including colons : which normally separate statements.

A single quote character ' can be used in place of :REM in all applications EXCEPT in DATA statements, where the ' is treated as part of an unquoted string.

Associated keywords: **none**

REMAIN

REMAIN(‹timer number›)

```
10 AFTER 500,1 GOSUB 40
20 AFTER 100,2 GOSUB 50
30 PRINT "program running":GOTO 30
40 REM this GOSUB-routine will not be called
   as it is disabled in line 80.
50 PRINT:PRINT "Timer 1 will now be ";
60 PRINT "disabled by REMAIN."
70 PRINT "Time-units remaining were:";
80 PRINT REMAIN(1)
run
```

FUNCTION: Returns the REMAINing count from the delay timer specified in ‹timer number› (in the range 0 to 3), and disables it.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, DI, EI, EVERY

RENUM

RENUM [‹new line number›][,[‹old line number›][,‹increment›]]

```
10 CLS
20 REM this will be line 123
30 REM this will be line 124
40 REM this will be line 125
RENUM 123,20,1
LIST
```

COMMAND: RENUMbers program lines.

The parameter ‹old line number› specifies the current existing line number at which renumbering is to commence. If ‹old line number› is omitted, renumbering will commence from the beginning of the program.

The parameter ‹new line number› specifies the new starting line number for the renumbered lines. If ‹new line number› is omitted, the renumbered program will start at line 10.

The parameter ‹increment› specifies the numeric step between each of the renumbered lines. If ‹increment› is omitted, the value of the numeric step will be 10.

RENUM takes care of all GOSUB, GOTO and other line calls. However, line number references within string expressions, such as those issued in KEY commands, are not altered; neither are line references within REM statements, nor the ‹line number expression› in a CHAIN or CHAIN MERGE command.

Line numbers are valid in the range 1 to 65535.

Associated keywords: DELETE, LIST

RESTORE

RESTORE[‹line number›]

```
10 READ a$:PRINT a$;" ";
20 RESTORE 50
30 FOR t=1 TO 500:NEXT:GOTO 10
40 DATA restored data can be read again
50 DATA and again
run
```

COMMAND: Restores the position of the 'pointer' back to the beginning of the DATA statement specified in the optional ‹line number›. Omitting this parameter restores the pointer back to the first DATA statement.

Further information concerning data will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: DATA, READ

RESUME

RESUME[‹line number›]

```
10 ON ERROR GOTO 60
20 FOR x=10 TO 0 STEP-1:PRINT 1/x:NEXT
30 END
40 PRINT "go here after error"
50 END
60 PRINT "error no.";ERR;"in line";ERL
70 RESUME 40
run
```

COMMAND: Resumes normal execution of a program after an error has been trapped and processed by an ON ERROR GOTO command. If the ‹line number› to RESUME at is not specified, the program will re-commence execution from the same line in which the error was first trapped. Try removing the ‹line number› parameter in the above example, then RUN again.

```
70 RESUME
run
```

Associated keywords: DERR, ERL, ERR, ERROR, ON ERROR GOTO, RESUME NEXT

RESUME NEXT

RESUME NEXT

```
10 ON ERROR GOTO 90
20 PRINT "press [RETURN] each time"
30 INPUT "1";a
40 INPUT "2";a
50 inpot "3";a REM syntax error!
60 INPUT "4";a
70 INPUT "5";a
80 END
90 PRINT "error no.";ERR;"in line";ERL
100 RESUME NEXT
run
```

COMMAND: Resumes normal execution of a program after an error has been trapped and processed by an **ON ERROR GOTO** command.

RESUME NEXT will re-commence execution from the line after that in which the error was first trapped.

Associated keywords: **DERR**, **ERL**, **ERR**, **ERROR**, **ON ERROR GOTO**, **RESUME**

RETURN

RETURN

```
10 GOSUB 50:PRINT "after the gosub":END
50 FOR n=1 TO 20
60 PRINT "sub-routine"
70 NEXT:PRINT
80 RETURN
run
```

COMMAND: Marks the end of a sub-routine. BASIC returns from the sub-routine to the statement immediately after the **GOSUB** command which invoked it.

Associated keywords: **GOSUB**

RIGHT$

RIGHT$(‹string expression›,‹required length›)

```
10 MODE 1:a$="464/6128 computer"
20 FOR n=1 TO 15:LOCATE 41-n,n
30 PRINT RIGHT$(a$,n)
40 NEXT
run
```

FUNCTION: Returns the number of characters (in the range 0 to 255) specified in the ‹required length› parameter, after extracting them 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: LEFT$, MID$

RND

RND[(‹numeric expression›)]

```
10 RANDOMIZE
20 FOR x=1 TO -1 STEP -1
30 PRINT "rnd parameter=";x
40 FOR n=1 TO 6
50 PRINT RND(x)
60 NEXT n,x
run
```

FUNCTION: Returns the next RaNDom number in sequence if the ‹numeric expression› has a positive value or is not specified.

If the ‹numeric expression› yields a value of zero, RND returns a copy of the last random number generated.

If the ‹numeric expression› yields a negative value, a new random number sequence is started, the first number of which is returned.

Associated keywords: RANDOMIZE

ROUND

ROUND(‹numeric expression›[,‹decimals›])

```
10 FOR n=4 TO -4 STEP-1
20 PRINT ROUND (1234.5678,n),
30 PRINT "with integer expression";n
40 NEXT
run
```

FUNCTION: Rounds the ‹numeric expression› to a number of decimal places or power of ten specified in the ‹decimals› parameter. If ‹decimals› is less than zero, the ‹numeric expression› is rounded to give an absolute integer with ‹decimals› number of zeros before the decimal point.

Associated keywords: ABS, CINT, FIX, INT

RUN

RUN ‹string expression›

```
RUN "demo"
```

COMMAND: Loads a BASIC or binary program from disc or cassette and commences execution. Any previously loaded BASIC program is cleared from the memory.

Protected BASIC programs may be run directly in this manner.

Associated keywords: LOAD

RUN

RUN[‹line number›]

```
RUN 200
```

COMMAND: Commences execution of the current BASIC program, from the specified ‹line number› parameter, or from the beginning of the program if the parameter is omitted. RUN resets the value of all current program variables to zero or null.

Protected programs may NOT be run in this manner, after loading.

Associated keywords: CONT, END, STOP

SAVE

SAVE ‹filename›[, ‹file type›][, ‹binary parameters›]

```
SAVE "discfile.xyz"
```

....saves the file in normal unprotected BASIC mode.

```
SAVE "discfile.xyz",P
```

....saves the file in Protected BASIC mode.

```
SAVE "discfile.xyz",A
```

....saves the file in ASCII mode.

```
SAVE "discfile.xyz",B,8000,3000,8001
```

....saves the file in Binary mode. In this example, saves the area of the computer's memory starting at address **8000**; the length of the file being **3000** bytes; the optional entry point address being **8001**.

COMMAND: Saves the program currently in the memory to disc or cassette. A Binary file is an area of memory saved to disc or cassette. The Binary parameters are:

‹start address› , ‹file length›[, ‹entry point›]

The screen memory can be saved as a Binary file. This is known as a 'screen dump' and can be performed using the command:

```
SAVE "screen",B,&C000,&4000
```

Then, to load it back onto the screen:

```
LOAD "screen"
```

464: You need not specify the filename if you wish the program to be saved as an `Unnamed file`. Example command:

```
SAVE ""
```

You will be instructed:

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

....whereupon you should press the **RECORD** and **PLAY** buttons on your cassette unit, followed by one of the keys on the computer. The tape in the cassette will start turning, and the computer will save the program.

The screen will display the saving messages:

```
Saving FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is saved.

If the first character of the filename is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to be saved. (You must make sure that the **RECORD** and **PLAY** buttons on your cassette unit are down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the **[ESC]** key produces the error message on the screen:

```
Broken in
```

Associated keywords: CHAIN, CHAIN MERGE, LOAD, MERGE, RUN

SGN

SGN(‹numeric expression›)

```
10 FOR n=200 TO -200 STEP-20
20 PRINT "SGN returns";
30 PRINT SGN(n);"for a value of";n
40 NEXT
run
```

FUNCTION: Determines the SiGN of the ‹numeric expression›. SGN returns -1 if ‹numeric expression› is less than zero, returns 0 if ‹numeric expression› equals zero, and returns 1 if ‹numeric expression› is greater than zero.

Associated keywords: ABS

SIN

SIN(‹numeric expression›)

```
10 CLS:DEG:ORIGIN 0,200
20 FOR n=0 TO 720
30 y=SIN(n)
40 PLOT n*640/720,198*y:NEXT
50 GOTO 50
run
```

FUNCTION: Calculates the SINe of the ‹numeric expression›.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: ATN, COS, DEG, RAD, TAN

SOUND

SOUND ‹channel status› , ‹tone period›[, ‹duration› [, ‹volume›
[, ‹volume envelope›[, ‹tone envelope›[, ‹noise period›]]]]]

```
10 FOR z=0 TO 4095
20 SOUND 1,z,1,12
30 NEXT
run
```

COMMAND: Programs a sound. The command takes the following parameters:

Parameter 1: ‹channel status›

The ‹channel status› parameter must yield an integer in the range 1 to 255. The parameter is bit significant, with each bit of the binary value of ‹channel status› signifying the following:

Bit 0 : (decimal 1) send sound to channel A (Least significant bit)
Bit 1 : (decimal 2) send sound to channel B
Bit 2 : (decimal 4) send sound to channel C
Bit 3 : (decimal 8) rendezvous with channel A
Bit 4 : (decimal 16) rendezvous with channel B
Bit 5 : (decimal 32) rendezvous with channel C
Bit 6 : (decimal 64) hold sound channel
Bit 7 : (decimal 128) flush sound channel (Most significant bit)

Hence a ‹channel status› parameter of 68 for example, would mean:

Send to channel C (4), with a hold state (64).

Parameter 2: ‹tone period›

This parameter defines the pitch of the sound, or in other words, 'what note it is' (i.e. do re mi fa so, etc). Each note has a set number, and this number is the ‹tone period›. See the chapter entitled 'For your reference....'.

Parameter 3: ‹duration›

This parameter sets the length of the sound, in other words, 'how long it lasts'. The parameter works in units of 0.01 (one hundredth) of a second, and if you don't specify the ‹duration›, the computer will default to 20 (one fifth of a second).

If the ‹duration› parameter is zero, the sound will last until the end of the specified volume envelope.

If the ‹duration› parameter is negative, the specified volume envelope is to be repeated ABS(‹duration›) times.

Parameter 4: ‹volume›

This parameter specifies the starting volume of a note. The number is in the range 0 to 15. A ‹volume› figure of 0 is off, while 15 is maximum. If no number is specified, the computer will default to 12.

Parameter 5: ‹volume envelope›

To make the volume vary within the duration of the note, you can specify a volume envelope using the separate command ENV. You can in fact create up to 15 different volume envelopes referenced in the range 1 to 15. The ‹volume envelope› parameter calls up the appropriate volume envelope reference number for use in the SOUND command.

Refer to the description of the ENV command.

Parameter 6: ‹tone envelope›

To make the tone or pitch vary within the duration of the note, you can specify a tone envelope using the separate command ENT. You can in fact create up to 15 different tone envelopes referenced in the range 1 to 15. The ‹tone envelope› parameter calls up the appropriate tone envelope reference number for use in the SOUND command. If you have specified a negative envelope number in the ENT command, use the absolute value of that number (i.e. without the negative sign) in this ‹tone envelope› parameter of the SOUND command.

Refer to the description of the ENT command.

Parameter 7: ‹noise›

A range of white noise is available, which can be switched off or added to the sound by varying the ‹noise› parameter between 0 and 31.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: ENT, ENV, ON SQ GOSUB, RELEASE, SQ

SPACE$

SPACE$(‹integer expression›)

```
10 MODE 1
20 PRINT "Put 9 spaces between me";
30 PRINT SPACE$(9);
40 PRINT "and you!"
run
```

FUNCTION: Creates a string of spaces of the given length, (in the range 0 to 255) specified in the ‹integer expression›.

Associated keywords: SPC, STRING$, TAB

SPC

(See PRINT SPC)

SPEED INK

SPEED INK ‹period 1›,‹period 2›

```
10 BORDER 7,18
20 FOR i=30 TO 1 STEP-1
30 SPEED INK i,i
40 FOR t=1 TO 700:NEXT t,i
run
```

COMMAND: Sets the rate of alternation between two ink colours specified in an INK or BORDER command. ‹period 1› specifies the time, in units of 0.02 (fiftieths) second for the first colour to be used; ‹period 2› sets the time for the second colour.

You must exercise careful judgement to avoid mesmeric effects when selecting colours and repeat rates!

Associated keywords: BORDER, INK

SPEED KEY

SPEED KEY ‹start delay›,‹repeat period›

```
10 CLS:FOR k=7 TO 1 STEP-2
20 PRINT "type your name, then [RETURN]"
30 SPEED KEY k,k
40 LINE INPUT a$:NEXT
50 PRINT "That's a funny name!"
run
```

COMMAND: Sets the rate of keyboard auto repeat. The ‹start delay› parameter specifies the time, in units of 0.02 (fiftieths) second before auto repeat starts. The ‹repeat period› parameter sets the interval between each auto repeat of a key.

SPEED KEY will operate only on keys which auto repeat by default, or which have been set to auto repeat by the KEY DEF command.

When intending to use small values of ‹start delay›, it is wise to pre-program one of the numeric keys to return the keyboard to its default SPEED KEY setting of 30,2. This can be achieved by the command:

```
KEY 0,"SPEED KEY 30,2"+CHR$(13)
```

....which will reset SPEED KEY to its default values when the 0 key on the numeric keypad is pressed.

Associated keywords: KEY DEF

SPEED WRITE

SPEED WRITE ‹integer expression›

```
SPEED WRITE 1
```

COMMAND: Sets the speed at which data is to be saved or written to the 464 cassette. The cassette can be written at either 2000 baud (bits per second) if the ‹integer expression› is 1, or at the default rate of 1000 baud if the ‹integer expression› is 0. When loading a file from tape, the computer automatically selects the correct reading speed as it loads.

For higher data reliability, it is recommended that you use SPEED WRITE 0 (default).

The SPEED WRITE command has no effect upon disc operation.

Associated keywords: OPENOUT, SAVE

SQ

SQ(‹channel›)

```
10 SOUND 65,100,100
20 PRINT SQ(1)
run
 67
```

FUNCTION: Reports the state of the Sound Queue for the specified ‹channel› which must be an integer expression, yielding one of the values:

1 : for channel A
2 : for channel B
4 : for channel C

The SQ function returns a bit significant integer, comprising the following bit settings:

Bits 0, 1, and 2 : the number of free entries in the queue
Bits 3, 4, and 5 : the rendezvous state at the head of this queue
Bit 6 : the head of the queue is held
Bit 7 : the channel is currently active

....where Bit 0 is the least significant bit, and Bit 7 is the most significant bit.

It can be seen therefore, that if Bit 6 is set, Bit 7 cannot be set, and vice versa. Similarly if Bits 3, 4, or 5 are set, Bits 6 and 7 cannot be set.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: ON SQ GOSUB, SOUND

SQR

SQR(‹numeric expression›)

```
PRINT SQR(9)
 3
```

FUNCTION: Returns the SQuare Root of the specified ‹numeric expression›.

Associated keywords: **none**

STEP

(See FOR)

STOP

STOP

```
10 FOR n=1 TO 30:PRINT n:NEXT
20 STOP
30 FOR n=31 TO 60:PRINT n:NEXT
run
cont
```

COMMAND: Stops execution of a program, but leaves BASIC in a state where the program can be resumed by the CONT command. STOP may be used to interrupt the program at a particular point when de-bugging.

Associated keywords: CONT, END

STR$

STR$(‹numeric expression›)

```
10 a=&FF :REM 255 hex
20 b=&X1111 :REM 15 binary
30 c$="***"
40 PRINT c$+STR$(a+b)+c$
run
*** 270***
```

FUNCTION: Converts the ‹numeric expression› to a decimal STRing representation.

Associated keywords: BIN$, DEC$, HEX$, VAL

STRING$

STRING$(‹length›,‹character specifier›)

```
PRINT STRING$(40,"*")
****************************************
```

FUNCTION: Returns a string expression consisting of the specified character repeated the number of times (in the range 0 to 255) specified in the ‹length›. Note that the above example could be entered as:

```
PRINT STRING$(40,42)
****************************************
```

....where the ‹character specifier› 42 refers to the ASCII value of the character * i.e. equivalent to PRINT STRING$(40,CHR$(42)).

Associated keywords: SPACE$

SWAP

(See WINDOW SWAP)

SYMBOL

SYMBOL ‹character number›,‹list of:‹row›

```
10 MODE 1:SYMBOL AFTER 105
20 row1=255:REM binary 11111111
30 row2=129:REM binary 10000001
40 row3=189:REM binary 10111101
50 row4=153:REM binary 10011001
60 row5=153:REM binary 10011001
70 row6=189:REM binary 10111101
80 row7=129:REM binary 10000001
90 row8=255:REM binary 11111111
100 PRINT "Line 110 re-defines the letter i (105).
    Type in some i's and see! Then list the program."
110 SYMBOL 105,row1,row2,row3,row4,row5,row6,row7,row8
run
```

COMMAND: Re-defines the shape of a character on the screen. Each of the parameters must yield an integer in the range 0 to 255.

To allocate space in the 464/6128's memory for a newly defined character, the computer must first be prepared by issuing the command:

```
SYMBOL AFTER x
```

....where x is equal to or less than the character number you wish to re-define.

The command **SYMBOL** is then issued, followed firstly by the character number x.

Regardless of whether or not the value of x specifies a character which is directly typeable at the keyboard, the re-defined character can be printed on the screen by issuing the command:

```
PRINT CHR$(x)
```

After **SYMBOL** x, there are up to 8 parameters which specify the 8 individual horizontal rows of the character, starting from the top. Each of the parameters can be in the range 0 to 255. The binary representation of each of the 8 parameters determines the pattern of that particular row in the finished character.

For example, if the first of the 8 parameters is 1, then the top row of the character has a binary representation of 00000001. Where the 1 appears, the section of the character is printed in the **PEN** colour; where a 0 appears, the section of the character is not visible because it is printed in the **PAPER** colour. Therefore the top row of this newly defined character has a dot in the top right hand corner. Continuing this example, we will specify the other 7 parameters as `3,7,15,31,63,0,0` - the binary representation of all 8 parameters then being:

parameter (row) 1 : 00000001 binary : (decimal 1)
parameter (row) 2 : 00000011 binary : (decimal 3)
parameter (row) 3 : 00000111 binary : (decimal 7)
parameter (row) 4 : 00001111 binary : (decimal 15)
parameter (row) 5 : 00011111 binary : (decimal 31)
parameter (row) 6 : 00111111 binary : (decimal 63)
parameter (row) 7 : 00000000 binary : (decimal 0)
parameter (row) 8 : 00000000 binary : (decimal 0)

Looking at the binary representation of the above 8 parameters, it should be possible to see what the shape of the new character is going to be like. Let's assign those parameters to character number 255 using the command:

```
SYMBOL 255,1,3,7,15,31,63,0,0
```

Note that the value of Ø appearing in the 2 final parameters means that you need only type in:

```
SYMBOL 255,1,3,7,15,31,63
```

Note that you can enter the parameters in binary to save you converting the 'pattern' of the symbol that you've created into decimal form. (Remember to use the &X binary prefix.) For example:

```
SYMBOL 255,&X00000001,&X00000011,&X00000111,
&X00001111,&X00011111,&X00111111
```

....Now, to see the character:

```
PRINT CHR$(255)
```

Assigning the above parameters to a typeable character on the keyboard would result in the new character appearing whenever the appropriate key is pressed, or wherever the previous character would have been printed. Furthermore, BASIC will not reject this new character as incomprehensible, but will regard it as the equivalent of the previous character.

Further information concerning user-defined characters will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: SYMBOL AFTER

SYMBOL AFTER

SYMBOL AFTER ‹integer expression›

```
10 CLS
20 SYMBOL AFTER 115
30 PRINT "Line 40 re-defines the s ";
40 SYMBOL 115,0,56,64,64,48,8,8,112
50 PRINT "to s"
60 PRINT "Cancel this defintion of s,"
70 PRINT "by typing: SYMBOL AFTER 240"
run
```

COMMAND: Sets the number of permissible user defined characters (in the range 0 to 256). The default setting is 240, giving 16 user defined characters (from 240 to 255). If the ‹integer expression› is 32, then all characters from 32 to 255 are re-definable. SYMBOL AFTER 256 permits no characters to be re-definable.

Whenever a SYMBOL AFTER command is executed, all user defined characters are reset to their default conditions.

SYMBOL AFTER will NOT operate if invoked AFTER the value of HIMEM has been altered using the MEMORY command, or by the opening of a file buffer with OPENIN or OPENOUT. Under such circumstances, an 'Improper argument' error (5) will be reported, (unless the previous state was SYMBOL AFTER 256).

Further information concerning user-defined characters will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: HIMEM, MEMORY, SYMBOL

TAB

(See PRINT TAB)

TAG

TAG[#‹stream expression›]

```
10 INPUT "enter your name";a$:CLS
20 PRINT "You certainly get around ";a$
30 TAG
40 x=LEN(a$)*17:y=50+RND*300:MOVE -x,y
50 FOR f=-x TO 640 STEP RND*7+3
60 MOVE f,y:PRINT " ";a$;:FRAME:NEXT
70 FOR b=640 TO -x STEP-RND*7+3
80 MOVE b,y:PRINT a$;" ";:FRAME:NEXT
90 GOTO 40
run
```

COMMAND: Sends any text specified for the given ‹stream expression› to be printed at the graphics cursor position. This allows text and symbols to be mixed with graphics, or moved pixel by pixel as opposed to character by character. The ‹stream expression› defaults to #0 if omitted.

The top left of the character cell is TAGged (Text At Graphics) to the graphics cursor, and non-printing control characters (e.g. line feed and carriage return) will display if the PRINT statement is not terminated by a semicolon.

In the default stream (#0), BASIC will switch off TAG when returning to direct mode.

Associated keywords: TAGOFF

TAGOFF

TAGOFF[#‹stream expression›]

```
10 MODE 2:TAG :REM Text At Graphics-on
20 year=1984:FOR x=1 TO 640 STEP 70
30 MOVE x,400:DRAWR 0,-350
40 year=year+1:PRINT year;:NEXT
50 TAGOFF :REM Text At Graphics-OFF
60 LOCATE 34,25:PRINT "Yearly figures"
70 GOTO 70
run
```

COMMAND: Cancels TAG (Text At Graphics) for the given ‹stream expression› (stream #0 if not specified), and re-directs text to the previous text cursor position used before TAG was invoked.

Associated keywords: TAG

TAN

TAN(‹numeric expression›)

```
PRINT TAN(45)
 1.61977519
```

FUNCTION: Calculates the TANgent of the ‹numeric expression›, which must be in the range −200000 to +200000.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: ATN, COS, DEG, RAD, SIN

TEST

TEST(‹x co-ordinate›,‹y co-ordinate›)

```
10 CLS
20 PRINT "You are using pen number";
30 PRINT TEST(10,386)
40 PRINT "Try changing PENs and MODEs";
50 PRINT "....then RUN again."
run
```

FUNCTION: Moves the graphics cursor to the absolute position specified by the ‹x and y co-ordinate›s, and reports the value of the ink at the new location.

Associated keywords: MOVE, MOVER, TESTR, XPOS, YPOS

TESTR

TESTR(‹x offset›,‹y offset›)

```
10 MODE 0:FOR x=1 TO 15:LOCATE 1,x
20 PEN x:PRINT STRING$(10,143);:NEXT
30 MOVE 200,400:PEN 1
40 FOR n=1 TO 23:LOCATE 12,n
50 PRINT "pen";TESTR(0,-16):NEXT
run
```

FUNCTION: Moves the graphics cursor by the amount specified in the ‹x and y offset›s relative to its current position, and reports the value of the ink at the new location.

Associated keywords: MOVE, MOVER, TEST, XPOS, YPOS

THEN

(See IF)

TIME

TIME

```
10 CLS:REM clock
20 INPUT "hour";hour
30 INPUT "minute";minute
40 INPUT "second";second
50 CLS:datum=INT(TIME/300)
60 WHILE hour<13
70 WHILE minute <60
80 WHILE tick<60
90 tick=(INT(TIME/300)-datum)+second
100 LOCATE 1,1
110 PRINT USING "## ";hour,minute,tick
120 WEND
130 tick=0:second=0:minute=minute+1
140 GOTO 50
150 WEND
160 minute=0:hour=hour+1
170 WEND
180 hour=1
190 GOTO 60
run
```

FUNCTION: Reports the elapsed time since the computer was last switched-on or reset, (excluding periods when reading or writing to disc).

Each second of real time is equal to the returned value: TIME/300.

Associated keywords: AFTER, EVERY, WEND, WHILE

TO

(See FOR)

TROFF
TRON

TROFF
TRON

```
10 TROFF:PRINT:PRINT "TRace-OFF"
20 FOR n=1 TO 8
30 PRINT "program running":NEXT
40 IF f=1 THEN END
50 TRON:PRINT:PRINT "TRace-ON"
60 f=1:GOTO 20
run
```

COMMAND: Traces the execution of a program by printing each line number before carrying it out. The line number appears inside square brackets [ ].

TRON switches TRace ON; TROFF switches TRace OFF.

The facility is particularly useful for studying the sequence of program line execution just before an error occurs.

Associated keywords: **none**

UNT

UNT(‹address expression›)

```
PRINT UNT(&FF66)
-154
```

COMMAND: Returns an integer in the range −32768 to +32767 which is the twos-complement equivalent of the unsigned value of the ‹address expression›.

Associated keywords: CINT, FIX, INT, ROUND

UPPER$

UPPER$(‹string expression›)

```
10 CLS:a$="my, how you've grown!"
20 PRINT UPPER$(a$)
run
```

FUNCTION: Returns a new string expression which is a copy of the specified ‹string expression› but in which all alphabetic characters in the range A to Z are converted to upper case. The function is useful for processing input which may come in mixed upper/lower case.

Associated keywords: LOWER$

USING

(See PRINT USING)

VAL

VAL(‹string expression›)

```
10 CLS:PRINT "I know my times tables!"
20 PRINT:PRINT "Press a key (1 to 9)"
30 a$=INKEY$:IF a$="" THEN 30
40 n=VAL(a$):IF n<1 OR n>9 THEN 30
50 FOR x=1 TO 12
60 PRINT n;"X";x;"=";n*x
70 NEXT:GOTO 20
run
```

FUNCTION: Returns the numeric VALue, (including any negative sign and decimal point) of the first character(s) in the specified ‹string expression›.

If the first character is not a number, then 0 is returned. If the first character is a negative sign or decimal point followed by non-numeric characters, a 'Type mismatch' error (13) will be reported.

Associated keywords: STR$

VPOS

VPOS(#‹stream expression›)

```
10 MODE 1:BORDER 0:LOCATE 8,2
20 PRINT "use cursor up/down keys"
30 WINDOW 39,39,1,25:CURSOR 1,1
40 LOCATE 1,13
50 IF INKEY(0)<>-1 THEN PRINT CHR$(11);
60 IF INKEY(2)<>-1 THEN PRINT CHR$(10);
70 LOCATE #1,3,24
80 PRINT #1,"text cursor ";
90 PRINT #1,"vertical position =";
100 PRINT #1,VPOS(#0):GOTO 50
run
```

FUNCTION: Reports the current Vertical POSition of the text cursor relative to the top of the text window. The ‹stream expression› MUST be specified, and does NOT default to #0.

Associated keywords: POS, WINDOW

WAIT

WAIT ‹port number›,‹mask›[,‹inversion›]

```
WAIT &FF34,20,25
```

COMMAND: Waits until the specified I/O ‹port number› returns a particular value in the range 0 to 255. BASIC loops whilst reading the I/O port. The value read is eXclusive ORed with the ‹inversion› and then ANDed with the ‹mask› until a non-zero result occurs.

BASIC will wait indefinitely until the required condition occurs.

Not a command to be used by the unwary.

Associated keywords: INP, OUT

WEND

WEND

```
WEND
```

COMMAND: Marks the end of the body of program which is to be executed within the WHILE loop. WEND automatically selects the WHILE command it is to be associated with.

Associated keywords: TIME, WHILE

WHILE

WHILE ‹logical expression›

```
10 CLS:PRINT "Ten second timer":t=TIME
20 WHILE TIME<t+3000
30 SOUND 1,0,100,15
40 WEND:SOUND 129,40,30,15
run
```

COMMAND: Repeatedly executes a body of program while a given condition is true. The WHILE command defines the head of the loop, and specifies the condition in the ‹logical expression›.

Associated keywords: TIME, WEND

WIDTH

WIDTH ‹integer expression›

```
WIDTH 40
```

COMMAND: Tells BASIC how many characters per line are to be printed when a printer is connected. BASIC will then send the extra carriage return/line feed at the appropriate time.

The computer assumes a default value of 132 unless a WIDTH command is specified.

The command WIDTH 255 suppresses the extra carriage return/line feed altogether, allowing printing to be 'line wrapped' by the printer. Note that carriage return/line feed will still be generated by a PRINT command that isn't terminated by a semicolon or comma.

Associated keywords: POS

WINDOW

WINDOW[#‹stream expression›,]‹left›,‹right›,‹top›,‹bottom›

```
10 MODE 0:BORDER 0:REM testcard
20 INK 0,0:INK 1,25:INK 2,23:INK 3,21
30 INK 4,17:INK 5,6:INK 6,2:INK 7,26
40 PAPER 0:CLS
50 PAPER 1:WINDOW 2,4,1,18:CLS
60 PAPER 2:WINDOW 5,7,1,18:CLS
70 PAPER 3:WINDOW 8,10,1,18:CLS
80 PAPER 4:WINDOW 11,13,1,18:CLS
90 PAPER 5:WINDOW 14,16,1,18:CLS
100 PAPER 6:WINDOW 17,19,1,18:CLS
110 PAPER 7:WINDOW 2,19,19,25:CLS
120 GOTO 120
run
```

COMMAND: Specifies the dimensions of a text stream (WINDOW) on the screen. The values of the parameters ‹left›,‹right›,‹top›, and ‹bottom› should correspond with the inclusive screen character locations consistent with the screen MODE in use.

If the ‹stream expression› is not specified, BASIC defaults to stream #0.

Further information concerning windows will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: WINDOW SWAP

WINDOW SWAP

WINDOW SWAP ‹stream expression›,‹stream expression›

```
10 MODE 1:INK 1,24:INK 2,9:INK 3,6
20 WINDOW 21,40,13,25:PAPER 3
30 WINDOW #1,1,20,1,12:PAPER #1,2
40 CLS:PRINT " window number 0"
50 CLS #1:PRINT #1," window number 1"
60 LOCATE 1,6
70 PRINT " red window (0)";SPC(2)
80 LOCATE #1,1,6
90 PRINT #1," green window (1)"
100 FOR t=1 TO 1000:NEXT
110 WINDOW SWAP 0,1:GOTO 60
run
```

COMMAND: Swaps the text window specified in the first ‹stream expression› with that specified in the second ‹stream expression›.

Both ‹stream expression›s must be specified, and in this case should NOT be preceded by a # stream director.

The command may be used to re-direct messages produced by BASIC,which are normally always sent to stream #0.

Further information concerning windows will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: WINDOW

WRITE

WRITE [#‹stream expression›,][‹write list›]

```
10 REM write variables.
20 INPUT "give me a number variable";a
30 INPUT "give me a string variable";a$
40 OPENOUT "datafile"
50 WRITE #9,a,a$
60 CLOSEOUT:PRINT "Data saved."
run
```

COMMAND: Writes the values of the items in the ‹write list› to the stream specified in the ‹stream expression›. Items written will be separated by commas; strings will be enclosed by double-quotes.

In this example the values of the variables which you input, are written to stream #9 (the disc/cassette stream).

(To recall the values of those variables from disc or cassette, it would be necessary to use a program such as follows:)

```
10 REM retrieve variables
20 OPENIN "datafile":INPUT #9,a,a$
30 CLOSEIN:PRINT "The 2 values are:"
40 PRINT:PRINT a,a$
run
```

Associated keywords: INPUT, LINE INPUT, OPENOUT

XOR

‹argument› XOR ‹argument›

```
IF "alan"<"bob" XOR "dog">"cat" THEN PRINT "correct" ELSE PRINT "wrong"
wrong

IF "bob"<"alan" XOR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong


IF "alan"<"bob" XOR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
correct

....

PRINT 1 XOR 1
 0
PRINT 0 XOR 0
 0
PRINT 1 XOR 0
 1
```

OPERATOR: Performs bit-wise boolean operation on integers. Result is 1 unless both argument bits are the same - eXclusive OR.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AND, OR, NOT

XPOS

XPOS

```
10 MODE 1:DRAW 320,200
20 PRINT "graphics cursor X POSition=";
30 PRINT XPOS
run
```

FUNCTION: Reports the current horizontal (X) POSition of the graphics cursor.

Associated keywords: MOVE, MOVER, ORIGIN, YPOS

YPOS

YPOS

```
10 MODE 1:DRAW 320,200
20 PRINT "graphics cursor Y POSition=";
30 PRINT YPOS
run
```

FUNCTION: Reports the current vertical (Y) POSition of the graphics cursor.

Associated keywords: MOVE, MOVER, ORIGIN, XPOS

ZONE

ZONE ‹integer expression›

```
10 CLS:FOR z=2 TO 20
20 ZONE z
30 PRINT "X","X ZONE =";z:NEXT
run
```

COMMAND: Changes the width of the print zone (specified in PRINT statements by using a comma between print items). The default setting of the print zone is 13 columns, but may be changed as specified in the ‹integer expression› in the range 1 to 255.

Associated keywords: PRINT

Chapter 4
Using Discs (6128 only)

Part 1: Discs

Making working discs

This section discusses how to make discs to use from day to day, and introduces some facilities of CP/M and its Utility programs.

Subjects covered:

- ★ Making a backup of the Master Disc.
- ★ Getting started with CP/M Plus.
- ★ Use of Help files.
- ★ Single and Multiple drive operation.
- ★ Copying files with PIP.
- ★ Operating with a BASIC only disc.
- ★ Turnkey AMSTRAD BASIC application.
- ★ Installing a Turnkey CP/M Plus application.

Part 7 of the Foundation course described how to format a blank system disc, which you can use for BASIC and games, as well as CP/M.

Part 10 of the Foundation course showed you how to make exact copies of discs with the `DISCKIT` program (on Side 1 of your system disc package).

This section considers how to use discs with the programs that you want on them.

Backup Master Discs

It is most important to make a copy of the Master System/Utility disc provided with your computer, and keep the original safe - it will be very costly to replace if damaged! Remember that the disc supplied has two sides. Every disc, in fact, has two sides and you are free to use either side for any purposes.

Side 1 is the most important - it contains the master copy of CP/M Plus and a set of utility programs for handling discs. Side 2 has files for assembler programmers.

You should regard your copy of the master disc as a 'library' of programs. Normally, you will select the program that you require by inserting the 'library disc' on which the program is located, rather than by copying the program onto a blank disc and running it from there.

Once again, it has to be emphasised that the 'library discs' that you use, MUST BE COPIES, made from the master disc package supplied with the computer.

Remember that if you are using a new blank disc to copy onto, the `DISCKIT` program (on Side 1) will format for you as well as doing the copying.

Getting started with CP/M Plus

You will be used to AMSTRAD BASIC appearing when your 6128 is switched on. The BASIC will remain in charge until superseded by either a BINary program run from AMSDOS, or by loading CP/M Plus with the command `|CPM`.

Once CP/M Plus is loaded, the 6128 will not need to refer to Side 1 again - unless of course, you want to run any of the utility programs contained on it. Only the start-up disc need be a System disc; all others can be Data-only discs, which have a greater storage capacity.

Running a program is simply a case of inserting the disc containing the required program and typing its name. Data used by the program may be on the same, or other, disc as the program. CP/M Plus allows the user to swap discs in the same way that AMSDOS does. If a number of programs, with perhaps a few utilities, are required on one disc for convenience, then use the program called `PIP` on Side 1, as described later in this chapter and in Chapter 5.

Maintaining a Hi profile!

The system disc provides a special file called PROFILE.SUB, which contains a list of commands that are executed automatically when CP/M Plus is started. You may then (if you have not already done so) insert a COPY of Side 1 of the System disc, and at the A> prompt, type:

```
REN PROFILE.SUB=PROFILE.ENG
```

....which creates the file PROFILE.SUB from PROFILE.ENG. This profile, which will be acted on next time CP/M Plus is started, contains the commands:

```
SETKEYS KEYS.CCP
LANGUAGE 3
```

....to set the cursor keys to be suitable for typing CP/M commands, and to convert the screen output to UK (from USA), effectively making **[SHIFT]3** display as a '£' symbol.

When the keyboard has been set up with the SETKEYS KEYS.CCP command, then CP/M command lines can be edited in much the same way as a line of BASIC. Full details of SETKEYS are given in Chapter 5 part 2.

One Drive or Two?

When CP/M Plus is first loaded, it detects the number of disc drives attached. This number is displayed as part of the sign-on message. Note that the process can be fooled if the second drive has a disc partially inserted.

All error messages relating to the disc mechanisms are displayed by default, as a banner on the 25th line of the screen. The programs themselves use only the first 24 lines of the screen.

When you have only one built-in drive, the bottom line will also display the message 'Drive is A:' or 'Drive is B:'. This is CP/M Plus allowing you to work with one physical mechanism as if it was two. You will have two discs to alternate between, and the bottom line of the screen will prompt you to insert the correct disc as the program requires it. This mode of operation avoids the need to purchase a second disc drive, but often requires considerable swapping of discs, which is time-consuming and introduces the possibility of human error.

Copying files from disc to disc

A standard utility program called PIP (Peripheral Interchange Program) is provided to copy files from one disc to another.

First load PIP from Side 1, by typing at the A> prompt:

```
PIP
```

....then a new prompt * will show that PIP has loaded correctly. Normally, you will copy files from a Source disc (in Drive A:) to a Destination disc (in Drive B:). We have already seen that in a single drive system, Drives A: and B: are the same mechanism.

To copy one file, for example SUBMIT.COM, type after the * prompt:

```
B:=A:SUBMIT.COM
```

To copy all the files from the Source disc to the Destination disc use the command:

```
B:=*.*
```

To exit from PIP, press **[RETURN]** at the * prompt.

PIP is a very sophisticated program, and further details of its operation appear in Chapter 5.

A BASIC only disc

As already described, a System disc is normally only used as the disc for starting up CP/M Plus. Discs used for BASIC can therefore be Data-Only discs, which have a slightly greater capacity.

The disc must be formatted using the DISCKIT program. To copy programs onto this type of disc, you must use PIP (loaded from Side 1), or LOAD and SAVE them from BASIC.

Turnkey AMSTRAD BASIC discs

If you buy an application program written in AMSTRAD BASIC for the 6128, it should be ready to operate when you switch on. (The expression 'turnkey' comes from the days when all small computers had a key-operated power switch). As with the Master system disc package supplied with the 6128, it is strongly recommended that you keep the original safe, and work from a copy.

Turnkey CP/M discs

The CP/M operating system allows you to load and run an immense library of software which has already been written for personal computers that support CP/M. The fundamental 'logic' of these programs has already been devised; all that is required to use them on your 6128 is to establish them on a suitable disc, and maybe to inform them of the particular method that the 6128 uses to operate the screen.

A set of programs on one disc designed to fulfil a specific application is called a 'package'. These packages are normally designed to work on a large range of different computers, each of which has its own size of screen and way of moving the cursor around.

The 6128 is provided with a built-in 'terminal emulator' when running CP/M Plus programs, and the characteristics are different from the Control Codes supported by BASIC.

Sometimes the package that you buy will have already been 'installed' for the AMSTRAD system, or cater for it by offering an installation compatible with the 6128. If available, simply follow the instructions provided with the software for a Zenith Z19/Z29 protocol. If the package does not have this, nor a specific AMSTRAD variant built in, then the section ahead entitled 'Configuring a CP/M Program' indicates some of the commands that can be sent to the 6128 screen to produce the sorts of effects that packages require. Normally, the installation, or customisation, procedure will involve typing in the relevant codes when requested to. Again, follow the instructions provided with the package.

The software you have purchased must be on a disc suitable for use in this system. Almost every different computer uses a different form of disc. Although many have the same size of disc, this does not necessarily mean that there is any compatibility between one and another in the information contained on them. Ask your supplier for an AMSTRAD 3 inch version.

Creating a Turnkey CP/M disc

As well as the application program itself, it is often useful to have the utilities SETKEYS.COM and perhaps SUBMIT.COM (with their associated instruction files) on the Turnkey disc.

PIP can be used to transfer the .COM files and also to make the instruction file for SUBMIT. In this latter mode, PIP is effectively a simple one-line-at-a-time editor. For example, the file PROFILE.ENG on Side 1 could have been created with the following commands:

(Insert System disc, Side 1 in Drive A:). Type:

```
PIP
```

(Remove System disc, insert destination disc). Type:

```
PROFILE.ENG=CON:
SETKEYS KEYS.CCP
[CONTROL]J LANGUAGE 3
[CONTROL]Z
```

Configuring a CP/M Program

The 6128 supports a wide range of control codes suitable for customising a software package to run with CP/M. Most data-processing and many other packages require to be able to print messages at any part of the screen, to accept input from any part of the screen and to generally understand cursor controls.

If your package has already been customised for the AMSTRAD system, then you need not concern yourself further.

Configuring the Output from the package

The installation procedure for a package will normally consist of running a special program (often called INSTAL) which, if it does not support either a Z19/Z29 type of terminal or the 6128 specifically, will ask a number of questions about the parameters of the 6128 screen. The answers should be derived from the table overleaf, which is an extract from Chapter 6 part 15.

Control Codes		Hex	Decimal		Operation
[BEL]		&07	7		Sound Bleeper.
[BS]		&08	8		Move cursor back one position.
[LF]		&0A	10		Move cursor down one line.
[CR]		&0D	13		Move cursor to left edge of window on current line.
[ESC]A	&1B	&41	27	65	Move cursor up one line.
[ESC]C	&1B	&43	27	67	Move cursor forward one position.
[ESC]E	&1B	&45	27	69	Clear screen.
[ESC]H	&1B	&48	27	72	Home cursor.
[ESC]J	&1B	&4A	27	74	Clear from and including the current cursor position to end of screen.
[ESC]K	&1B	&4B	27	75	Clear from and including the current cursor position to right edge of screen.
[ESC]L	&1B	&4C	27	76	Insert Line.
[ESC]M	&1B	&4D	27	77	Delete line.
[ESC]N	&1B	&4E	27	78	Delete character at cursor position.
[ESC]Y	&1B	&59	27	89	Move cursor to given position on
	‹c›	‹r›	‹c›	‹r›	screen. ‹c› is column + 32, ‹r› is row + 32.
[ESC]d	&1B	&64	27	100	Clear from start of screen to and including the current cursor position.
[ESC]o	&1B	&6F	27	111	Clear from left edge of screen to and including the cursor pos tion.
[ESC]p	&1B	&70	27	112	Enter inverse video.
[ESC]q	&1B	&71	27	113	Exit inverse video.

Configuring the Input to the package

The programs in the package will expect to be able to interrogate the keyboard. Most of the keys on the 6128 keyboard return standard values except for the cursor keys. It is possible to use the SETKEYS utility to re-define the codes produced by the keyboard, although where possible, it is preferable for each different package to be configured to accept standard values.

It is an unfortunate fact that there is not a general understanding between different items of software as to which keystroke to use for control functions. Printable characters, and 'space', **[TAB]** and **[RETURN]** are fairly universal, but disagreement sets in with 'backspace', and from there-on, matters get worse! Compare for example, the different codes expected for the operation 'move cursor to start of line':

CP/M Commands require: **[CONTROL]B**
....and a typical wordprocessor might require: **[CONTROL]Q S**

Three useful sets of keyboard codes are provided as standard. Each configuration can be called up from files contained on Side 1 of your system discs package:

```
SETKEYS KEYS.CCP
```

....already described as one of the commands automatically sent by PROFILE.SUB, will set up the keyboard suitably for CP/M commands.

Starting a Turnkey CP/M Package

Normally, all that is required is to type the package's main program name at the A> prompt. For example to run a wages program called PAYROLL.COM, simply type:

```
PAYROLL
```

If any configurations require to be set up then perhaps a SUBmit file will be provided.

Autostarting a Turnkey CP/M Package

It is possible to arrange for the CP/M Plus operating system to automatically run a particular program from the start-up system disc. This is performed by including the program's name at the end of the PROFILE.SUB file on that disc.

Chapter 5
AMSDOS and CP/M (6128 only)

Part 1: AMSDOS

Subjects covered:

- ★ Introduction to AMSDOS
- ★ Disc directory
- ★ Changing discs
- ★ Filenames and filetypes
- ★ AMSDOS headers
- ★ Filenames on 2 drives
- ★ Wild cards
- ★ Example program using AMSDOS commands
- ★ Summary of AMSDOS commands
- ★ Manipulating and copying files
- ★ Reference guide to error messages

Introduction

AMSDOS extends the AMSTRAD BASIC supplied with your computer by providing a number of external commands, which are identified by the preceding | (bar) symbol.

AMSDOS allows the user to change discs freely, as long as no files are in use - in which case an error message will be displayed and there could be a loss of data if the open file was being written to.

Disc directory

Every disc has two sections, the directory and the data area. The directory contains a list of all the filenames and a 'map' of whereabouts on the disc each file is to be found. AMSDOS or CP/M can calculate the size of a particular file by inspecting its directory entry. Calculation of the amount of space left on a disc is made by adding up all the files in the directory and seeing how much remains unused.

Whenever a file is read, its directory entry is examined, giving the disc location. When a new file is created, free space is allocated to it, and when a file is erased the space is relinquished. The directory works in units of 1K and can have up to 64 different entries. Large files will have one entry for every 16K although normally this fact is hidden from the user.

Changing discs

Under AMSDOS (and CP/M Plus) a disc may be changed, or removed, whenever the drive is not being accessed and neither the input nor output files are open on that drive.

Changing a disc while it is still being written to, may corrupt the data on the disc. If a disc is changed while there are still files open on it, then as soon as AMSDOS detects this, all the open files on the drive will be abandoned and an error message produced. Any data yet to be written will be lost and the latest directory entry will not be written to disc. However, AMSDOS can only detect this change when it reads the directory, which it does every 16K of the file (and whenever a file is opened or closed). Thus, potentially 16K of data could be corrupted by changing a disc while there are still files open on it.

AMSDOS filenames and filetypes

It is standard practice to name disc files in such a way that there is an indication of which 'type' they are. This naming convention DOES NOT 'force' the computer to use the file in any particular way, however some programs will only accept a file when it has the correct type of name. AMSDOS will accept any type of name, but will search in preference for certain file types if not otherwise specified. (See 'AMSDOS headers' ahead.)

Construction of filenames

The filename is constructed from two parts with a . (dot) separating them. The first part can be up to 8 characters long, and the second up to 3 characters long. Thus for example: "ROINTIME.DEM" , "DISCKIT3.COM" , and "DISC.BAS" are all legal filenames.

The second part of the filename is called the filetype. Filenames and filetypes can be composed of a mixture of letters and numbers, but cannot have embedded spaces or punctuation marks. Some common conventional filetypes are:

.‹space›	Unspecified type. May be a data file created by an OPENOUT "‹filename›" or BASIC program saved by AMSDOS using SAVE "‹filename›",A style.
.BAS	BASIC program saved by AMSDOS using SAVE "‹filename›" or SAVE "‹filename›",P or SAVE "‹filename›.BAS",A styles.
.BIN	Program or area of memory saved by AMSDOS using SAVE "‹filename›",B,‹binary parameters› style.
.BAK	Old version of a file, where AMSDOS or a utility program has saved a newer version of a file using an existing name. This allows the user to back-track to the previous (BAcK-up) version if required.
.COM	Command file. CP/M utility programs are all of this filetype.
.SUB	Instruction file for the CP/M SUBMIT program.

AMSDOS headers

AMSDOS automatically SAVEs files with a suitable type identifier, so it is not normally necessary to specify one unless you wish to override the defaults described previously. BASIC program files, protected BASIC program files and binary files are saved to the disc with a header record, so that the AMSDOS command:

```
LOAD "‹filename›"
```

....can recognise them and take the appropriate action. If the AMSDOS command LOAD cannot find a header, it assumes that the file is a program in ASCII, i.e. plain text.

Notwithstanding the contents of the header, when AMSDOS is asked to LOAD a file where no filetype is specified, then it first looks for a file of type:

.‹space›

If that does not exist, it looks for a file of type:

.BAS

....then finally, one of type:

.BIN

This allows the user to abbreviate the filename, i.e. not needing to specify the filetype, in most instances.

A disc data file started with the command OPENOUT and subsequently written to, will have no header, and the contents will be in ASCII, i.e. plain text, from the BASIC WRITE, PRINT or LIST commands. The disc command OPENIN will search for files in the same order as LOAD, if no file type is specified.

Filenames on two drives

On a 2-drive system, i.e. if an additional drive has been connected to the computer, files can exist on either drive. The computer will not automatically look for a file on both drives, so the user must specify which drive to use. You can either employ the |A or |B or |DRIVE commands (full description ahead) to select one or other drive, and then use a normal filename, or alternatively you can override the default drive assignment by specifying the drive as an A: or B: prefix to the filename. Thus, for example:

```
|B
SAVE "PROG.BAS"
|A
```

....and....

```
|A
SAVE "B:PROG.BAS"
```

....both save the program to the second drive, Drive B.

Similarly, you can override the default USER number assignment (USER numbers allow you to partition-off the directory) by specifying the USER number (in the range 0 to 15) as a prefix to the filename. Thus, for example:

```
LOAD "15:PROG.BAS"
```

....and....

```
SAVE "15:PROG.BAS"
```

....would load and save the program to the USER number 15 section of the disc, whatever the default USER number setting. (See the |USER command, ahead.)

Finally, it is possible to override both default USER and DRIVE settings (in that order) by specifying them together in the prefix to the filename, for example:

```
RUN "15B:PROG.BAS"
```

Wild cards

It is often required to perform some disc operation (copying, erasing, etc.) on more than one disc file. When a filename is specified for a particular operation, AMSDOS scans the disc directory looking for a name which exactly matches. However, it is possible (where the command allows) to perform the operation on a group of files where some of the characters in the filenames can be 'don't care'. This is shown by using the character ? in the 'don't care' position. If the whole block (or remainder of the whole block) of any part of the filename is 'don't care', then the block of ?'s can be abbreviated to the symbol *. Thus, for example, FRED.* is shorthand for FRED.??? and F*.BAS is shorthand for F???????.BAS

Finally, the expression *.* means 'all files'.

Examples:

DIRECTORY	Match *.BAS	Match FRED?.BAS	Match F*.BA?
BERT.BAS	BERT.BAS		
FRED1.BAS	FRED1.BAS	FRED1.BAS	FRED1.BAS
FRED2.BAS	FRED2.BAS	FRED2.BAS	FRED2.BAS
FRED3.BAK			FRED3.BAK
FRED3.BAS	FRED3.BAS	FRED3.BAS	FRED3.BAS
FINISH.BAS	FINISH.BAS		FINISH.BAS

Examples of using AMSDOS commands in a program

To give you a good understanding of the AMSDOS commands, we recommend that you work through the examples, referring to the relevant sections in the rest of this chapter as you go. DO NOT type in, or run, these programs with one of your original Master CP/M system discs installed.

Saving variables and performing a screen dump

The following example program writes to the disc, and you will therefore need a blank (formatted) disc or working disc inserted in the drive to run the program. The program draws a Union Jack flag, and then saves the whole screen to disc.

```
10 dumpfile$="flagdump.srn"
20 MODE 1:BORDER 0
30 DIM colour(2)
40 FOR i=0 TO 2
50  READ colour(i): REM get colours from DATA statement
60  INK i,colour(i)
70 NEXT
80 ON ERROR GOTO 430
90 OPENIN "param.dat" ' test if file exists
100 CLOSEIN:ON ERROR GOTO 0
110 IF errnum=32 AND DERR=146 THEN CLS:
    GOTO 160 ' file doesnt exist
120 CURSOR 1:PRINT "Do you want to overwrite
    old file? Y/N ";
130 a$=INKEY$:ON INSTR(" YN",UPPER$(a$))
    GOTO 130,150,140:GOTO 130
140 PRINT a$:PRINT "Program abandoned":END
150 PRINT a$:CURSOR 0
160 OPENOUT "param.dat"
170 WRITE #9,dumpfile$,1: REM save filename and mode
180 FOR i=0 TO 2
190 WRITE #9,colour(i): REM save colours
200 NEXT i
210 CLOSEOUT
220 CLS
230 gp=1:GRAPHICS PEN gp:w=125
240 x=-65:a=240:y=400:b=-150:GOSUB 400
```

continued on the next page

```
250 y=0:b=150:GOSUB 400
260 x=575:a=-240:y=400:b=-150:GOSUB 400
270 y=0:b=150:GOSUB 400
280 gp=2:GRAPHICS PEN gp:w=40
290 a=240:x=-40:y=400:b=-150:GOSUB 400
300 x=0:y=0:b=150:GOSUB 400
310 a=-240:x=640:y=0:b=150:GOSUB 400
320 x=600:y=400:b=-150:GOSUB 400
330 ORIGIN 0,0,256,380,0,400:CLG 1
340 ORIGIN 0,0,0,640,150,250:CLG 1
350 ORIGIN 0,0,280,352,0,400:CLG 2
360 ORIGIN 0,0,0,640,168,230:CLG 2
370 SAVE dumpfile$,b,&C000,&4000
380 DATA 2,26,6
390 END
400 MOVE x,y:DRAWR a,b:DRAWR w,0:DRAWR -a,-b
410 MOVE x+a/2+w/2,y+b/2:FILL gp
420 RETURN
430 errnum=ERR:RESUME NEXT
run
```

Note the use of **.DAT** and **.SRN** filetypes. These filetypes are used to remind us of what is in the file, rather than because they have any inherent significance. The file **PARAM.DAT** will be an ASCII data file without a header, whilst **FLAGDUMP.SRN** is an AMSDOS binary file with a header.

Note how the program deliberately tries to read from the file **PARAM.DAT** before writing to it, in order to establish if the file already exists. If the file does NOT exist, then an error is reported by BASIC; the error is trapped by the program, and execution proceeds without interruption. If the file DOES already exist, then no error is reported, and the program automatically asks if you wish to overwrite the existing file.

The particulars of the screen dump, namely the screen mode, palette colours and name of file containing the actual information, are saved into a parameter file. This illustrates the use of a data file to **WRITE** program variables (`dumpfile$`) and constants (`1`), saving them for use by another program.

Loading the screen back

The following example is a general purpose screen dump displaying program, using a parameter file to control its action. Note how variables are **INPUT** from the data file, with the **EOF** function allowing automatic variation in the size of the file. It is important that the screen dump displayed by this program was saved with the screen in a known position in memory, otherwise the result will be 'skewed'. This is ensured by the saving program executing a **MODE** command and thereafter being careful not to cause the screen to scroll.

```
10 DIM colour(15): REM Provision for 16 colours
20 OPENIN "param.dat"
30 INPUT #9, filename$,screenmode
40 i=0
50 WHILE NOT EOF
60  INPUT #9,colour(i)
70 INK i,colour(i)
80 i=i+1
90 WEND
100 CLOSEIN
110 MODE screenmode:BORDER 0
120 LOAD filename$
run
```

Summary of AMSDOS external commands

|A

```
|A
```

COMMAND: Set default drive to Drive A. Equivalent to **|DRIVE** with parameter **A**. (The main drive within the computer is Drive A.)

|B

```
|B
```

COMMAND: Set default drive to Drive B. Equivalent to **|DRIVE** with parameter **B**. (The main drive within the computer is Drive A.)

|CPM

```
|CPM
```

COMMAND: Switch to alternative disc environment by loading the operating system from a system disc. The operating system supplied with the computer is CP/M Plus.

This command will fail if the computer's drive does not contain a system disc with CP/M.

|DIR

```
|DIR[,‹string expression›]

    |DIR,"*.BAS"
```

COMMAND: Display the disc directory (In CP/M style), and free space. If the string expression is omitted, the wild-card `*.*` is assumed.

|DRIVE

```
|DRIVE,‹string expression›

    |DRIVE,"A"
```

COMMAND: Set the default drive. This command will fail if AMSDOS is unable to read the disc in the requested drive.

|ERA

```
|ERA,‹string expression›

    |ERA,"*.BAK"
```

COMMAND: Erases all files which match the filename and which are not Read/Only. Wild cards are permitted.

|REN

|REN,‹string expression›,‹string expression›

```
|REN,"NEWNAME.BAS","OLDNAME.BAS"
```

COMMAND: Give a file a new name. Any other file with the chosen new name must not already exist. Wild cards are not permitted.

The USER (see |USER ahead) parameter may be specified within the ‹string expression›s to override any default settings. For example, the command |REN,"Ø:NEW.BAS","15:OLD.BAS" will rename the file in USER 15 called "OLD.BAS", to a file called "NEW.BAS" in USER Ø, regardless of any default or previously issued settings of |USER.

|USER

|USER,‹integer expression›

```
|USER,3
```

COMMAND: Determines which of up to 16 individual sections of the directory (in the range 0 to 15), disc functions (e.g. CAT, LOAD, |DIR etc.) are to be performed on.

A file on one USER number may be transferred to another, by a |REN command. For example, |REN,"15:EXAMPLE.BAS","Ø:EXAMPLE.BAS" transfers a file from USER number Ø to USER number 15, although the name of the file itself (EXAMPLE.BAS) is not changed.

Copying files from disc to disc

AMSDOS files with headers

It is possible to copy this type of file in the CP/M environment using PIP (see part 2 of this chapter). Any file created by AMSDOS which has a header record (see 'AMSDOS headers', previously described) will be copyable as a whole - disc to disc - but in general the contents of the file will not be understood by any CP/M programs.

ASCII files

Files created by AMSDOS without headers are generally in ASCII, and are both copyable and understood by CP/M programs. In particular it should be possible to exchange ASCII program files, ASCII data files and ASCII text files freely between AMSDOS and CP/M programs.

Read/Only files

It is possible, using CP/M, to set any file to be Read/Only, and/or set to a special System status in the directory. Such attributes can only be set or reset in the CP/M environment, but are honoured by AMSDOS. For further details, see part 2 of this chapter (SET utility).

Reference guide to AMSDOS error messages

When AMSDOS cannot carry out a command for some reason, it will display an error message. If there is a problem with the hardware, the error message is followed by the question:

```
Retry, Ignore or Cancel?
```

R causes the operation to be repeated, possibly after the user has taken some preventative action.

I causes the computer to continue as if the problem had not occurred, which will often lead to unexpected and possibly inconvenient results.

C causes the operation to be cancelled, which will often lead to a further error message.

Error message meanings

```
Unknown command
```

....The command is not spelt correctly.

```
Bad command
```

....The command cannot be carried out for some reason. Syntax error or inappropriate hardware configuration.

‹filename› already exists

....User is trying to rename a file with a name that's already in use.

‹filename› not found

....File does not exist.

Drive ‹drive›: directory full

....No more room in the disc directory for a new entry.

Drive ‹drive›: disc full

....No more room on the disc for new data.

```
Drive ‹drive›: disc changed, closing ‹filename›
```

....Disc has been changed with files still open on it.

```
‹filename› is read only
```

....File cannot be operated on because it is Read/Only. Files can only be set Read/Only or Read/Write in the CP/M environment.

```
Drive ‹drive›: disc missing
```

....No disc in drive, or disc is not seated and spinning properly. Recommended action is to eject and re-insert the disc, then type **R**.

```
Drive ‹drive›: disc is write protected
```

....Attempt has been made to write on a disc with the Write Protect hole open. To use the disc, eject, close the write protect hole, re-insert the disc, and then type **R**.

```
Drive ‹drive›: read fail
```

....Hardware error reading disc. Recommended action is to eject and re-insert the disc, then type **R**.

```
Drive ‹drive›: write fail
```

....Hardware error writing disc. Recommended action is to eject and re-insert the disc, then type **R**.

```
Failed to load CP/M
```

....Read error loading CP/M during the **|CPM** command, or you are not using a valid system disc containing CP/M. Note that trying to load CP/M from a Data format disc will produce a 'read fail' error.

Part 2: CP/M

CP/M Plus

Subjects covered:

- ★ Introduction to CP/M
- ★ Booting CP/M Plus
- ★ Direct Console Mode
- ★ Transient programs
- ★ Managing peripherals

CP/M Plus is a disc operating system. It is a special program which gives you access to the full power of your 6128. The 128K of RAM is used to the full, with over 61K available to user programs. CP/M incorporates random access to data files and the 6128 implementation includes a sophisticated VDU emulator.

Because CP/M is available for so many different computers, it means that there are thousands of applications packages available for you to choose from, and a whole wealth of knowledge and experience for you to draw upon.

Introduction

The CP/M operating system provides a way for you to communicate with the computer, and manipulate files and peripherals. Special commands (and programs on the disc called utilities) are there to help you get on with the main task - which is running your applications programs with your data.

Needless to say, it is possible to become quite an expert at how CP/M and all the various utilities work, and at times, such expertise can be very useful in helping out when we get into trouble. Most of us, however, only need to know enough to get us started, and the rest of this chapter is designed to introduce all the features and facilities without obscuring the vital facts with too many frills.

Whereas BASIC has its Direct Mode and the '`Ready`' prompt, CP/M has a Direct Console Mode and is identified by the `A>` or `B>` prompt. Certain built-in commands are available but the majority of the 'housekeeping' work is done by loading and running 'transient programs'. They are called 'transient' because they are only in the computer (loaded from the disc) while you are using them, as opposed to being built-in.

As well as standard CP/M error messages, the system also generates a number of specialised hardware error messages which can be distinguished by the fact that they normally appear on the bottom line of the screen in a 'banner' form.

CP/M Plus on the disc

The major part of CP/M Plus resides in a special file which has the filetype '`.EMS`', and is found on Side 1 of the system discs package. The computer loads CP/M from this file into the memory using a two stage process.

Initially, the AMSDOS command `ICPM` loads the first sector of track 0. On a system disc this sector has been arranged to be a program which then loads the `.EMS` file into memory. The remainder of the system tracks are unused.

Early morning start profile

During the loading process, when CP/M Plus is first activated, if the file `PROFILE.SUB` is present on the disc, then the instructions in that file are `SUBMIT`ted. This facility can be used to re-arrange the keyboard, customise the screen output, initialise a printer and even auto-start an application program. In chapter 4 we saw how to rename the profile file supplied on Side 1 in order to activate it.

While the profile file is working, a small temporary file is opened on the disc, which must therefore be write-enabled. This is why the master disc itself cannot include a recognisable profile file.

Profile files can be constructed using a word processor, text editor (such as ED.COM), or even from BASIC. The small BASIC program below could have been used to generate the file PROFILE.SUB:

```
10 OPENOUT "PROFILE.SUB"
20 PRINT #9,"SETKEYS KEYS.CCP"
30 PRINT #9,"LANGUAGE 3"
40 CLOSEOUT
```

Console control codes

In the CP/M environment, a variety of special key operations are used. These keystrokes replace the action of the **[ESC]**ape and cursor keys used in AMSTRAD BASIC. The control codes below are assigned after running the command:

```
SETKEYS KEYS.CCP
```

....where both the transient program SETKEYS.COM and the command file KEYS.CCP are found on Side 1 of the system discs package.

Control Code	Key	Action
[CONTROL]A	◊	Moves the cursor one character to the left.
[CONTROL]B	**[CONTROL]**◊ ....or.... **[CONTROL]**◊	Moves the cursor to the beginning of the line. If the cursor is already at the beginning, moves to the end.
[CONTROL]C	**[CONTROL] [ESC]**	Abandon.
[CONTROL]E	**[CONTROL] [RETURN]**	Physical carriage return.
[CONTROL]F	◊	Moves cursor one character to the right.
[CONTROL]G	**[CLR]**	Deletes character under cursor.
[CONTROL]H	**[DEL]**	Backspace delete.
[CONTROL]I	**[TAB]**	Moves cursor to the next tab stop.
[CONTROL]J		Send command line.

[CONTROL]K	**[CONTROL] [CLR]**	Delete to end of line.
[CONTROL]M	**[RETURN]** ....or.... **[ENTER]**	Send command line.
[CONTROL]P		Hardcopy toggle. Turn on/off log of all screen output to printer.
[CONTROL]Q		Resume screen output.
[CONTROL]R	**[CONTROL] [ENTER]**	Retype command line.
[CONTROL]S	**[ESC]**	Halts the screen output from CP/M. Use **[CONTROL]Q** to resume.
[CONTROL]U		Discard line.
[CONTROL]W	**[COPY]**	Recall last-typed command line.
[CONTROL]X	**[CONTROL] [DEL]**	Delete from beginning of line to cursor.
[CONTROL]Z		End of text.

Filenames

Many of the commands take filenames as a parameters, and where specified, the filename may contain wild-cards (see the section entitled 'Wild cards' in part 1 of this chapter). All filenames will be forced to upper case.

Direct Console Commands and most utility programs do NOT require that filenames are contained in double quotes `""`. Remember that filenames can have an `A:` or `B:` prefix to force CP/M to use the appropriate drive.

Therefore a typical CP/M command is:

```
TYPE KEYS.CCP
```

....where `TYPE` is the function required, meaning 'display on the screen', and `KEYS.CCP` is a filename specifying which file we wish to see.

Switching default drives

If you have an additional disc drive connected, then it is possible to switch the default drive selection between Drive A and Drive B by typing `A:` or `B:` at the `B>` or `A>` prompt. That prompt, of course, tells you the current default drive. Adding the `A:` and `B:` prefix to filenames overrides, but does not reset, the default drive setting.

Direct Console Commands

There are a number of Direct Console Commands which can be typed at the `A>` or `B>` prompt. Each command can be abbreviated, and although the simple functions described below are genuinely built-in, there are also more sophisticated transient commands with the same name.

DIR command

`DIR` lists the `DI`Rectory of the disc. The filenames are not sorted into any particular order, but the position of the filename in the `DIR` display indicates the position of that file's entry in the disc directory. Wild cards are permitted. Files set with the '`SYS`' attribute will not be listed.

`DIR`	will list files on the default drive.
`DIR B:`	will list files on Drive `B:`
`DIR *.BAS`	will list files of type `.BAS`
`DIR B:*.BAS`	will list files of type `.BAS` on Drive `B:`
`DIR PIP.COM`	will list only the file `PIP.COM` (if it exists).

DIRSYS or DIRS command

`DIRSYS` or `DIRS` lists only those directory entries with the '`SYS`' attribute set. Otherwise it operates as `DIR`. The `SYS` attribute is described later.

ERASE or ERA command

`ERA` is used to `ERA`se files from the directory. Only the directory entry is erased, so the data is still in the data section of the disc until the space is re-used by another file; however, the information is nevertheless not recoverable. Wild card filenames are permitted, and if used, `ERA` will ask for confirmation. `ERA` does not list the filenames that are deleted. If any file about to be erased is found to be Read/Only then the command will abort. The Read/Only attribute is described later.

`ERA PIP.COM`	will erase the file `PIP.COM`
`ERA B:PIP.COM`	will erase the file `PIP.COM` on Drive B
`ERA *.BAS`	will erase all `.BAS` files

RENAME or REN command

`REN` allows you to `REN`ame an existing file. The new filename is specified first, followed by = then the existing filename. If the new filename already exists, an error message will be displayed.

Wild cards in the filenames are outside the scope of `REN` as a built-in command, and require `RENAME.COM`, the transient program.

`REN NEWNAME.BAS=OLDNAME.BAS`	will rename the file `OLDNAME.BAS` to `NEWNAME.BAS`
`REN B:NEWNAME.BAS=OLDNAME.BAS`	will rename the file `OLDNAME.BAS` to `NEWNAME.BAS`, on Drive B.

TYPE or TYP command

`TYPE` asks for the specified file to be `TYPE`d onto the screen. If the file is not an ASCII text file, unpredictable and possibly undesirable side-effects may occur.

```
TYPE KEYS.CCP
```

....will display the file `KEYS.CCP`

USER or USE command

`USER` changes the current user number. CP/M Plus starts with the current user number set to 0. Normally you can only access files identified with the current user number, thus providing a way of partitioning-off the directory.

A file in User 0, with the '`SYS`' attribute set can be accessed from all other user numbers. This is a very powerful facility for making utility and applications programs available to all user numbers, without having an actual copy of them in each user area.

```
USER 3
```

....will set the current user number to 3.

Transient commands

To perform more sophisticated file management than permitted by the Direct Console Mode, you must employ one of the various utility programs provided. These are invoked merely by typing the program name, possibly followed by a filename and/or some parameters. You have probably already used `DISCKIT`.

The commands fall into a number of categories as indicated below.

The commands `DISCKIT`, `SETKEYS`, `SETLST`, `SETSIO`, `PALETTE`, `LANGUAGE` and `AMSDOS`, are designed by AMSTRAD, and work exclusively on the AMSTRAD system. They will not work on any other CP/M system.

It is possible to enter multiple commands on a single line, where the commands are separated by an exclamation mark. For example:

```
LANGUAGE 3!SETKEYS KEYS.WP
```

Peripheral Management

`DISCKIT` is a complete disc formatter, copier and checker. It is quicker to format whilst copying than to format and then copy. Comprehensive menus indicate which keystrokes (mainly from the function key area of the keyboard) should be made. Vendor format discs are a special form of System disc intended for software distribution, although Data format discs are perhaps more suitable for this activity in the CP/M Plus environment.

> **WARNING**
>
> The licence agreement for your CP/M (which is electronically serial-number encoded) permits its use on a single computer system only. In particular this means that you are prohibited from giving any other person a disc with YOUR serial-numbered copy of CP/M on it. Because copies of Side 1 of your master package will include your CP/M (in the `.EMS` file) on it, you must be careful not to sell, exchange, or in any other way part with, any disc with that file on it.

Language characters

The 6128 has a full set of international characters. The LANGUAGE command exchanges certain of these characters so that simple software can display alternative and accented characters on the screen. Further details are contained in part 16 of the chapter entitled 'For your reference....'.

The command:

```
LANGUAGE 3
```

....will set the 6128 to the UK displayed character set, which swaps the # and £ signs (compared to the default USA set).

Colours

The default colours of CP/M Plus on the 6128 (with colour monitor) are bright-white characters on a blue background. These colours can be changed by the PALETTE command, which takes a number of parameters, one for each ink - ink 0 includes the background and border area; ink 1 is for the text. Each colour is represented by a number in the range 0 to 63, which can be used to gauge the colour's intensity (brightness) on a white monitor.

It is possible to specify any number of inks, from one to sixteen, although only the first two will be visible in 80 column mode.

The command:

```
PALETTE 63,1
```

....will reverse the normal settings of ink 0 and ink 1, giving a background of bright-white (63) with text in blue (1).

Use the values in the following table to select the colours (or intensities) that you require. You can use either the hex or decimal representation as you prefer.

Colour	Hex	Decimal	Colour	Hex	Decimal
Black	&00	0	Pastel blue	&2B	43
Blue	&02	2	Orange	&2C	44
Bright blue	&03	3	Pink	&2E	46
Red	&08	8	Pastel magenta	&2F	47
Magenta	&0A	10	Bright green	&30	48
Mauve	&0B	11	Sea green	&32	50
Bright red	&0C	12	Bright cyan	&33	51
Purple	&0E	14	Lime green	&38	56
Bright magenta	&0F	15	Pastel green	&3A	58
Green	&20	32	Pastel cyan	&3B	59
Cyan	&22	34	Bright yellow	&3C	60
Sky blue	&23	35	Pastel yellow	&3E	62
Yellow	&28	40	Bright white	&3F	63
White	&2A	42			

Keyboard

The codes generated by the keyboard can be altered by the `SETKEYS` command. This allows suitable codes to be assigned to keys and to expansion tokens. The actual codes must be written into a file, whose name is then presented to the `SETKEYS` command. The command file can be created by a text editor, by `PIP`, or even from BASIC. For example:

```
SETKEYS KEYS.TST
```

....where the file `KEYS.TST` contains:

```
E &8C "DIR ↑M" expansion token 12
8 N S C "↑H" backspace = [CONTROL]H, ASCII 08
```

....will firstly redefine the **[CONTROL] [ENTER]** expansion token (represented by &8C) to be the string `DIR` **[RETURN]**, and then turn the cursor left key ◊ (key number 8) into a backspace.

Standard files provided with the 6128 are `KEYS.CCP` for CP/M command editing and `KEYS.WP` which is suitable for many word processors.

Printers

Initialising printers can be performed by the command:

```
SETLST ‹filename›
```

....where ‹filename› contains the string, or strings to send to the printer. As with the command file for SETKEYS, control codes can be represented by:

```
↑‹character›
```

....or by....

```
↑'‹character value›'
```

....or by....

```
↑'‹control code name›'
```

....where control code names are ESC, FF, etc., as shown in the table of ASCII characters in the chapter entitled 'For your reference....'.

A useful initialising code for many printers is the value 15, setting the printer into condensed printing.

The command:

```
PRINT #8,CHR$(15)
```

....would set this in BASIC. In CP/M, issue the command:

```
SETLST CONDENSE
```

....where the file CONDENSE contains any one of the following as a single line of text:

```
↑'SI'
↑O
↑'&F'
↑'15'
```

....which are all interpreted as the decimal value 15.

Some applications programs require a screen that is 24x80. The command SET24X80 may be used to set the screen size.

The commands:

```
SET24X80
```

....or....

```
SET24X80 ON
```

....turns 24 x 80 mode on, and:

```
SET24X80 OFF
```

....turns 24 x 80 mode off.

The normal 6128 full screen size is 24 x 80, with the bottom line reserved for status messages. Turning off 24 x 80 mode will only be noticeable if the status line is also disabled. Refer to part 15 of Chapter 6 for details of how to turn the status line on and off.

Serial interface

Built into CP/M Plus is support for a single channel serial Input/Output interface (RS232). Its vital statistics can be examined by typing the command **SETSIO** (with no parameters):

```
SETSIO
```

....or can be set using a command which may include any (or all) of the selections:

```
SETSIO, RX 1200, TX 75, PARITY NONE, STOP 1, BITS 8,
HANDSHAKE ON, XOFF OFF
```

....which would set a new configuration.

Baud rates and **XON/XOFF** status are also affected by one of the assignments possible with the **DEVICE** command. **DEVICE** deals with logical and physical devices. Logical devices are indicated by a **:** colon. To examine all the current device attributes, type:

```
DEVICE
```

....and the attributes can be altered by commands such as:

```
DEVICE SIO[1200]    ....sets SIO to 1200 baud.
DEVICE SIO[XON]     ....to turn on SIO XON/XOFF protocol.
DEVICE SIO[NOXON]   ....to turn off SIO XON/XOFF protocol.
```

The connections between logical and physical devices can be altered. Normally CON: is set to CRT (keyboard/screen), AUX: is set to SIO (the optional serial interface), and LST: is set to LPT (the Centronics printer interface). The command:

```
DEVICE LST:=SIO
```

....will send the printer output to the serial interface (if fitted).

Note how this is a channel re-direction, not to be confused with the file copying facilities provided by PIP. The two commands GET ‹filename› and PUT ‹filename› re-direct console input or output, and printer output, instructing them to use a file, rather than the device channel.

```
PIP
```

The PIP utility (Peripheral Interchange Program) allows you to transfer information between the computer and its peripherals.

In general, the form of the command is:

```
PIP ‹destination›=‹source›
```

The ‹source› and ‹destination› can be either a filename, with wild-cards allowed in the source, or a logical device. The following logical devices may be used:

As Source		As destination	
CON:	console input	CON:	console output
AUX:	auxiliary input	AUX:	auxiliary output
EOF:	an end-of-file mark	LST:	printer
		PRN:	a printer with added tab expansion, line numbers and page breaks

Examples of PIP:

```
PIP B:=A:*.COM
```

....copy all *.COM file from Drive A: to Drive B:

```
PIP KEYBOARD.CPM=KEYS.CCP
```

....make a copy of KEYS.CCP calling it KEYBOARD.CPM

```
PIP CON:=KEYS.CCP
```

....send file KEYS.CCP to screen (similar effect to TYPE KEYS.CCP)

```
PIP LST:=KEYS.CCP
```

....send file KEYS.CCP to printer

```
PIP TYPEIN.TXT=CON:
```

....accept keyboard input into a file called TYPEIN.TXT

Note that this last operation is terminated by the **[CONTROL]Z** control code, and that in order to get a new line you must type **[CONTROL]J** after **[RETURN]** every time. **[CONTROL]J** is the ASCII code for line feed.

If typed without parameters, PIP gives a * prompt, and you can then enter the commands that you require. This form of operation is particularly useful for copying files when we do not have the PIP.COM file on either the source or destination disc. We can load in PIP from System disc Side 1, remove the System disc, and then insert the discs that we are going to use during the copying.

To exit from PIP, press **[RETURN]** at the * prompt.

Note that PIP can be used to copy files from one disc to another on a single drive system, prompts will be automatically issued to change the disc. The source and destination drive identifiers must differ.

System management

DIR, ERASE, RENAME and TYPE are transient programs with more facilities than their built-in counterparts. As with many other transient programs supplied by Digital Research, secondary parameters are specified in square brackets. A few examples are:

`DIR [FULL]`	displays show file sizes and attributes
`ERASE *.COM [CONFIRM]`	will prompt for confirmation of each file it finds, individually

`RENAME`	will ask for the old and new filenames to be typed in
`RENAME *.SAV=*.BAK`	will rename all files of type `.BAK` to type `.SAV`
`TYPE KEYS.WP [NOPAGE]`	will suppress the screen pagination

The file attributes of `SYS` (System) and `RO` (Read/Only) have been mentioned earlier. These and many other attributes can be assigned using the `SET` command, which accepts wild cards.

The commands:

```
SET *.COM [RO]
SET KEYS.CCP [RO]
SET A: [RO]
```

....set files or a drive to Read/Only status, to prevent accidental erasure.

The commands:

```
SET *.COM [RW]
SET KEYS.CCP [RW]
SET A: [RW]
```

....reset files or a drive to Read/Write status.

The commands:

```
SET *.COM [SYS]
SET KEYS.CCP [SYS]
```

....give files the System attribute. Files with this attribute are not shown by the `DIR` command (`DIRS` or `DIRSYS` is required). However, the files are still available for use, and in addition, if the files are located in User area number 0, they are available to all other User area numbers.

The commands:

```
SET *.COM [DIR]
SET KEYS.CCP [DIR]
```

....removes the System attribute.

Each disc can be assigned a label, or name, and also a password. That password will protect the directory itself, rather than the files named in the directory. The individual files can also be assigned a password.

```
SET [NAME=ROLAND]
SET [PASSWORD=SALLY]
SET [PROTECT=ON]
```

....act on the default disc.

```
SET *.*[PASSWORD=SALLY]
SET *.*[PROTECT=READ]
```

....act on files on the default disc (the wild cards ***.*** used here indicate 'all files').

Date and time stamping can be activated by the **INITDIR** command (on Side 2 of your system discs package). The commands:

```
INITDIR
SET [CREATE=ON] ....or.... SET [ACCESS=ON] ....and....
SET [UPDATE=ON] ....together with....
DIR [FULL]
```

....will initiate and display date and time stamping on the default drive. Typing in:

```
DATE SET
```

....is then required each time CP/M Plus is started, to set the clock. Once set, the clock will keep reasonable time, updated automatically by the 6128 and inspected by:

```
DATE ....and....
DATE CONTINUOUS
```

WARNING

If passwords, disc labelling, or date and time stamping have been activated, then it is recommended that the disc is NEVER subsequently written to by AMSDOS which does not support these facilities.

Normally, files will accessed on the default drive only, unless a particular drive is specified. The command:

```
SETDEF *,A:
```

....(where * refers to the default drive) instructs CP/M (when it searches for files) to search firstly on the default drive, then on Drive A:. In other words, if the default drive is B: then files will be found automatically even if they only exist on Drive A:.

The commands:

```
SETDEF [PAGE] ....and....
SETDEF [NOPAGE]
```

....turn on and off the automatic pagination of console display.

Remember that most of the facilities of DEVICE, SET, and SETDEF particularly where they refer to drives (rather than files or discs) need to be set up, together with the date, each time CP/M Plus is started. This is a prime application for a suitable PROFILE.SUB file.

SUBMIT is required to execute files of individual commands automatically. The contents of the command files are text, and it is possible to include input to programs if the first character on those lines in the .SUB file is <

The drive size, amount of space, and number of directory entries left on a disc, together with the user areas containing files, and the disc label (if any) may be displayed by various combinations of the SHOW command:

```
SHOW B:
SHOW B:[LABEL]
SHOW B:[USERS]
SHOW B:[DIR]
SHOW B:[DRIVE]
```

....all specify statistics from Drive B:

Exit-ing CP/M Plus

```
AMSDOS
```

This program relinquishes control from CP/M and returns to the built-in AMSTRAD BASIC, from which the AMSDOS disc commands will be available.

Chapter 6
For your reference....

This chapter provides much of the reference information that you are likely to require as you learn to use this computer.

Subjects covered:

- ★ Cursor locations and control code extensions
- ★ Interrupts
- ★ ASCII and graphics characters
- ★ Key references
- ★ Sound
- ★ Error messages
- ★ BASIC keywords
- ★ Planners
- ★ Connections
- ★ Printers
- ★ Joysticks
- ★ Disc organisation
- ★ Resident System eXtensions (RSX's)
- ★ Memory
- ★ CP/M Plus Terminal Emulator
- ★ CP/M Plus Character Set

Part 1: BASIC 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 being as follows:

1. Writing a character

2. Drawing the cursor 'blob'

3. Obeying the control codes marked with an asterisk in the list ahead.

The procedure for forcing the cursor to a legal position is as follows:

1. 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.

2. 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.

3. 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.

4. 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 in the range 0 to 31 sent to the text screen do not produce a character on the screen but are interpreted as CONTROL CODES (and should not be injudiciously applied). Some of the codes alter the meaning of one or more of the following characters, which are the code's parameters.

A control code sent to the graphics screen will merely print the conventional symbol related to its function if accessed via the keyboard (e.g. &07 'BEL' - **[CTRL] G**). It will execute its control function if addressed using the form of the command:

`PRINT CHR$(&Ø7)`, or `PRINT "` G `"` (where the G is obtained by pressing **[CTRL] G** within the `PRINT` statement).

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.

BASIC Control characters

Value	Name	Parameter	Meaning
&00 0	NUL		No effect. Ignored.
&01 1	SOH	0 to 255	Print the symbol given by the parameter value. This allows the symbols in the range 0 to 31 to be displayed.
&02 2	STX		Turn off text cursor. Equivalent to **CURSOR** command with a ‹user switch› parameter value of 0.
&03 3	ETX		Turn on text cursor. Equivalent to **CURSOR** command with a ‹user switch› parameter value of 1. Note that to display a cursor from within a BASIC program (other than the automatic cursor generated when BASIC is awaiting keyboard input), a **CURSOR** command with a ‹system switch› parameter value of 1 must be used.
&04 4	EOT	0 to 2	Set screen mode. Parameter taken MOD 4. Equivalent to a **MODE** command.
&05 5	ENQ	0 to 255	Send the parameter character to the graphics cursor.
&06 6	ACK		Enable Text Screen. (See &15 NAK ahead.)
&07 7	BEL		Sound Bleeper. Note that this flushes the sound queues.
&08 8 *	BS		Move cursor back one character.

Value		Name	Parameter	Meaning
&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 to 15	Set Paper Ink. Parameter taken MOD 16. Equivalent to **PAPER** command.
&0F	15	SI	0 to 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.
&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.

Value	Name	Parameter	Meaning
&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 to 1	Parameter MOD 2. Transparent option. 0 disables, 1 enables.
&17 23	ETB	0 to 3	Parameter MOD 4 0 sets normal graphics ink mode 1 sets XOR graphics ink mode 2 sets AND graphics ink mode 3 sets OR graphics ink mode
&18 24	CAN		Exchange Pen and Paper Inks.
&19 25	EM	0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 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 to 80 1 to 80 1 to 25 1 to 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 to 15 0 to 31 0 to 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. (Parameter values 27 to 31 are un-defined colours.)
&1D 29	GS	0 to 31 0 to 31	Set Border to a pair of colours. Equivalent to a **BORDER** command. The two parameters (MOD 32) specify the two colours. (Parameter values 27 to 31 are un-defined colours.)
&1E 30	RS		Move cursor to top left hand corner of window.
&1F 31	US	1 to 80 1 to 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 gives the line.

The housekeeping of the 464/6128 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 between the hardware and the BASIC interpreter - for example the ink flashing function, where BASIC passes the parameters - and the operating system 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 BASIC.

If you are tempted to POKE around in the machine memory addresses, or CALL the sub-routines, save your program and listing before doing so, or you may regret it!

In order to program extensively using machine code, it will be necessary to use an assembler.

Part 2: Interrupts

The 464/6128 makes extensive use of Z80 interrupts to provide an operating system that includes several multi-tasking features, exemplified by the AFTER and EVERY structures described earlier in this manual. The precedence of the event timers 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 possible intermediate variable states at the point of interruption. The interrupt sub-routine itself should avoid unwanted interaction with the state of variables in the main program.

The sound queues have independent interrupts of equal priority. Once a sound interrupt has started, it is not interrupted by any other sound interrupt. This enables sound interrupt routines to share variables with immunity from the effects mentioned above.

When a sound queue's interrupt is enabled (by using ON SQ GOSUB), it will immediately interrupt if the sound queue for that channel is not full, otherwise it will interrupt when the current sound ends and there is room for more in the queue. The action of interrupting disables the event, so the sub-routine must re-enable itself if further interrupts are required.

Both attempting to issue a sound and testing the queue status will also disable a sound interrupt.

Part 3: BASIC ASCII and graphics characters

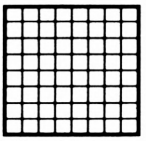
ASCII

The table below illustrates the standard ASCII reference character set using decimal, octal, and hex notation, together with the ASCII codes where appropriate. Each of the 464/6128 character cells is also represented in detail in the following pages.

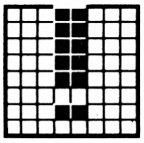
DEC	OCTAL	HEX	ASCII characters
0	000	00	NUL ([CTRL]@)
1	001	01	SOH ([CTRL]A)
2	002	02	STX ([CTRL]B)
3	003	03	ETX ([CTRL]C)
4	004	04	EOT ([CTRL]D)
5	005	05	ENQ ([CTRL]E)
6	006	06	ACK ([CTRL]F)
7	007	07	BEL ([CTRL]G)
8	010	08	BS ([CTRL]H)
9	011	09	HT ([CTRL]I)
10	012	OA	LF ([CTRL]J)
11	013	OB	VT ([CTRL]K)
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	0
49	061	31	1
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
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	~

Machine specific BASIC graphics character set

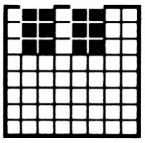
The characters reproduced here are plotted on the standard 8 x 8 cell matrix used to write the screen of the 464/6128. User defined characters may be grouped for special effects, and butted next to one another. See the section 'User Defined Characters' in the chapter entitled 'At your leisure....'.



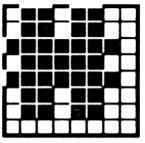
32 &H20
&X00100000



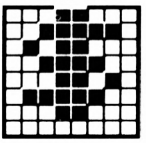
33
&H21
&X00100001



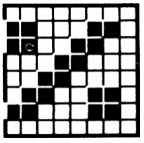
34
&H22
&X00100010



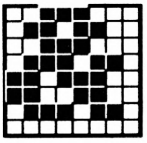
35
&H23
&X00100011



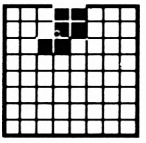
36
&H24
&X00100100



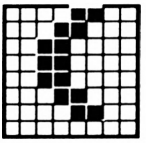
37
&H25
&X00100101



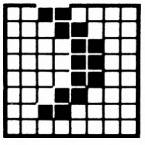
38
&H26
&X00100110



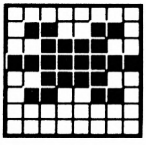
39
&H27
&X00100111



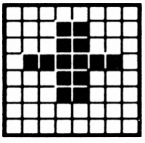
40
&H28
&X00101000



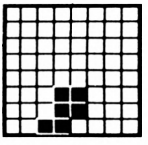
41
&H29
&X00101001



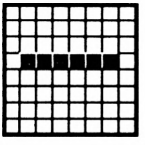
42
&H2A
&X00101010



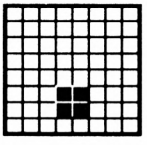
43
&H2B
&X00101011



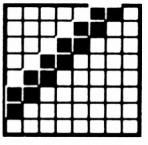
44
&H2C
&X00101100



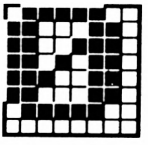
45
&H2D
&X00101101



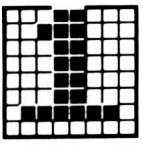
46
&H2E
&X00101110



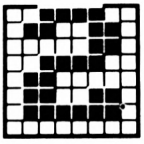
47
&H2F
&X00101111



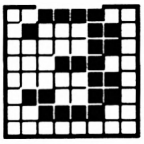
48
&H30
&X00110000



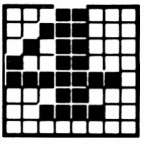
49
&H31
&X00110001



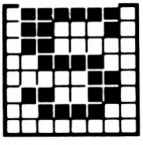
50
&H32
&X00110010



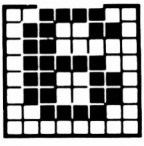
51
&H33
&X00110011



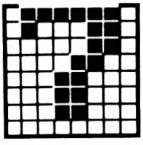
52
&H34
&X00110100



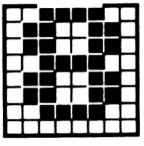
53
&H35
&X00110101



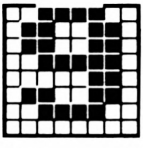
54
&H36
&X00110110



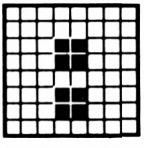
55
&H37
&X00110111



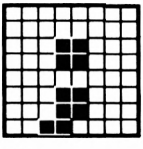
56
&H38
&X00111000



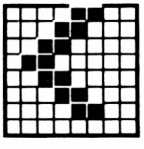
57
&H39
&X00111001



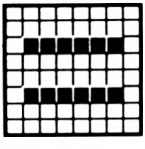
58
&H3A
&X00111010



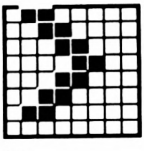
59
&H3B
&X00111011



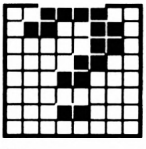
60
&H3C
&X00111100



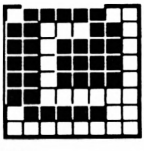
61
&H3D
&X00111101



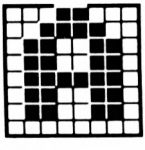
62
&H3E
&X00111110



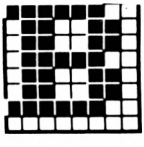
63
&H3F
&X00111111



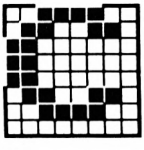
64
&H40
&X01000000



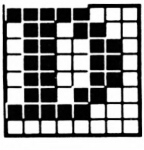
65
&H41
&X01000001



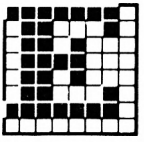
66
&H42
&X01000010



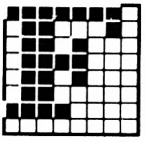
67
&H43
&X01000011



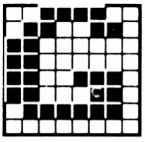
68
&H44
&X01000100



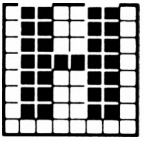
69
&H45
&X01000101



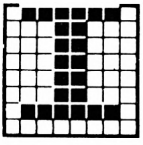
70
&H46
&X01000110



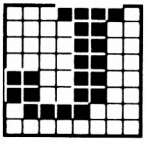
71
&H47
&X01000111



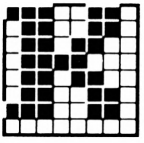
72
&H48
&X01001000



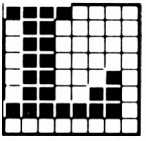
73
&H49
&X01001001



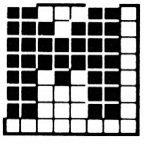
74
&H4A
&X01001010



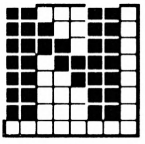
75
&H4B
&X01001011



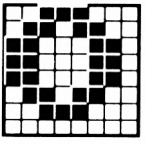
76
&H4C
&X01001100



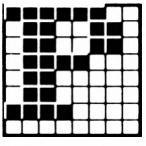
77
&H4D
&X01001101



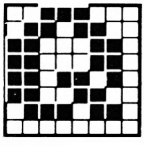
78
&H4E
&X01001110



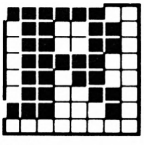
79
&H4F
&X01001111



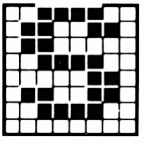
80
&H50
&X01010000



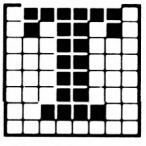
81
&H51
&X01010001



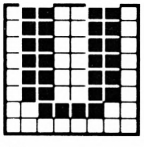
82
&H52
&X01010010



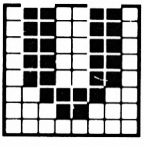
83
&H53
&X01010011



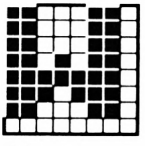
84
&H54
&X01010100



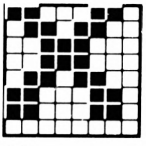
85
&H55
&X01010101



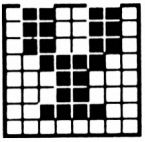
86
&H56
&X01010110



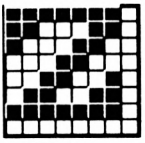
87
&H57
&X01010111



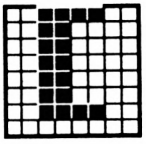
88
&H58
&X01011000



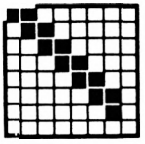
89
&H59
&X01011001



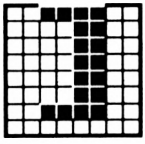
90
&H5A
&X01011010



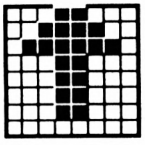
91
&H5B
&X01011011



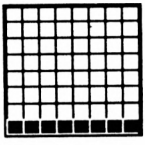
92
&H5C
&X01011100



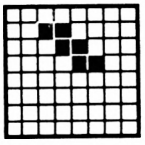
93
&H5D
&X01011101



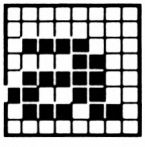
94
&H5E
&X01011110



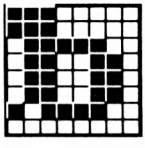
95
&H5F
&X01011111



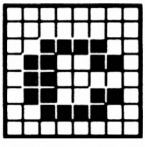
96
&H60
&X01100000



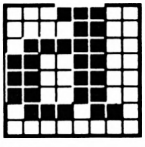
97
&H61
&X01100001



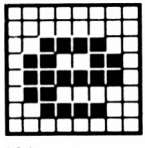
98
&H62
&X01100010



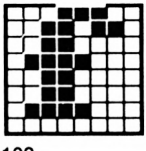
99
&H63
&X01100011



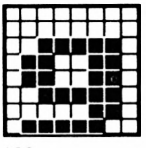
100
&H64
&X01100100



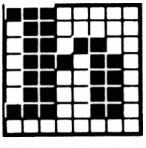
101
&H65
&X01100101



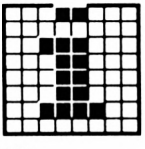
102
&H66
&X01100110



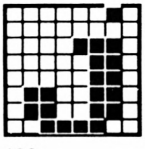
103
&H67
&X01100111



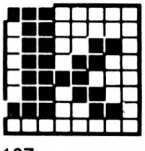
104
&H68
&X01101000



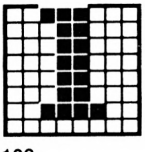
105
&H69
&X01101001



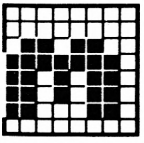
106
&H6A
&X01101010



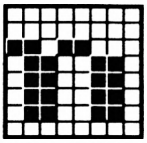
107
&H6B
&X01101011



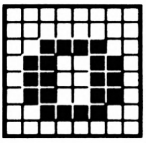
108
&H6C
&X01101100



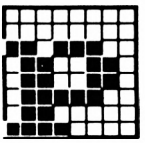
109
&H6D
&X01101101



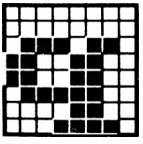
110
&H6E
&X01101110



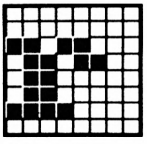
111
&H6F
&X01101111



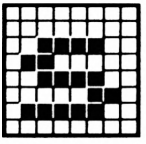
112
&H70
&X01110000



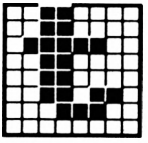
113
&H71
&X01110001



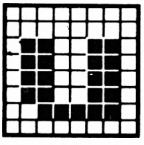
114
&H72
&X01110010



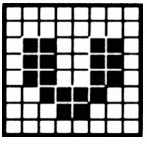
115
&H73
&X01110011



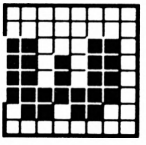
116
&H74
&X01110100



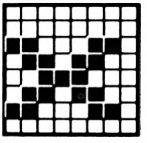
117
&H75
&X01110101



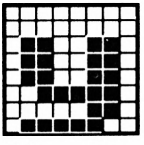
118
&H76
&X01110110



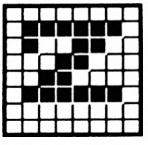
119
&H77
&X01110111



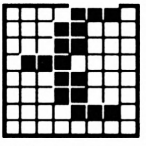
120
&H78
&X01111000



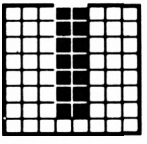
121
&H79
&X01111001



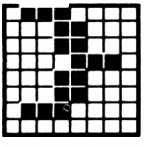
122
&H7A
&X01111010



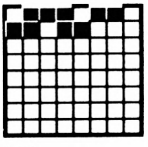
123
&H7B
&X01111011



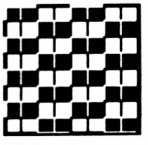
124
&H7C
&X01111100



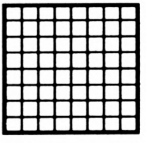
125
&H7D
&X01111101



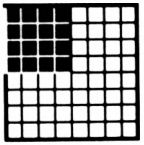
126
&H7E
&X01111110



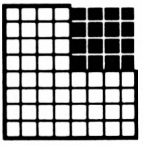
127
&H7F
&X01111111



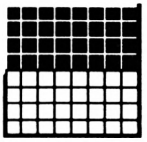
128
&H80
&X10000000



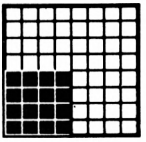
129
&H81
&X10000001



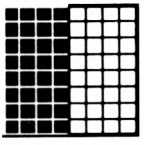
130
&H82
&X10000010



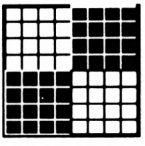
131
&H83
&X10000011



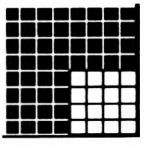
132
&H84
&X10000100



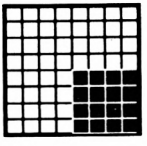
133
&H85
&X10000101



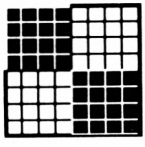
134
&H86
&X10000110



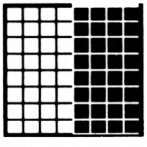
135
&H87
&X10000111



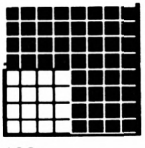
136
&H88
&X10001000



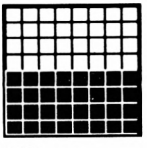
137
&H89
&X10001001



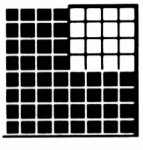
138
&H8A
&X10001010



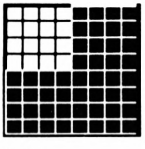
139
&H8B
&X10001011



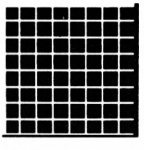
140
&H8C
&X10001100



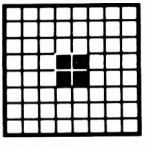
141
&H8D
&X10001101



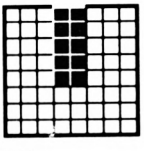
142
&H8E
&X10001110



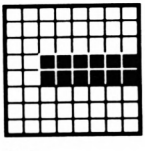
143
&H8F
&X10001111



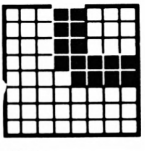
144
&H90
&X10010000



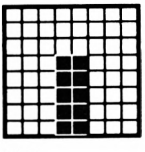
145
&H91
&X10010001



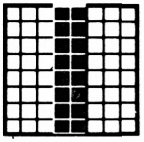
146
&H92
&X10010010



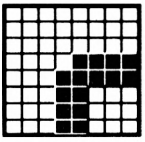
147
&H93
&X10010011



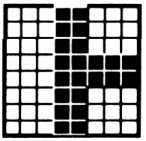
148
&H94
&X10010100



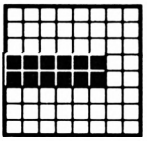
149
&H95
&X10010101



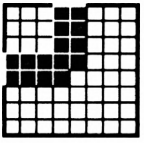
150
&H96
&X10010110



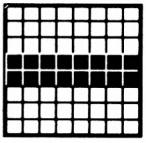
151
&H97
&X10010111



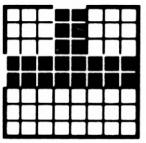
152
&H98
&X10011000



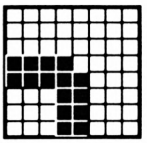
153
&H99
&X10011001



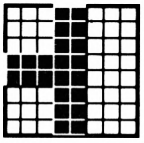
154
&H9A
&X10011010



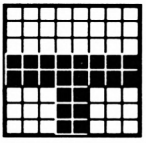
155
&H9B
&X10011011



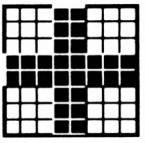
156
&H9C
&X10011100



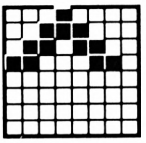
157
&H9D
&X10011101



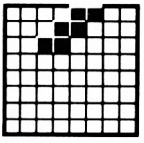
158
&H9E
&X10011110



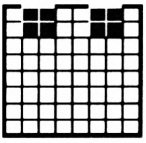
159
&H9F
&X10011111



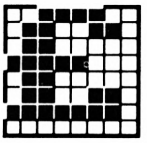
160
&HA0
&X10100000



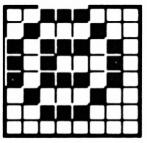
161
&HA1
&X10100001



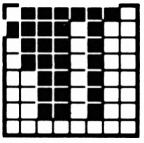
162
&HA2
&X10100010



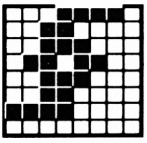
163
&HA3
&X10100011



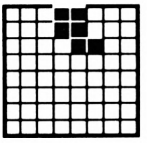
164
&HA4
&X10100100



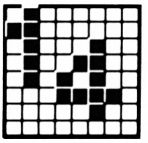
165
&HA5
&X10100101



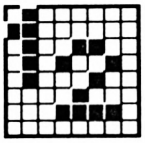
166
&HA6
&X10100110



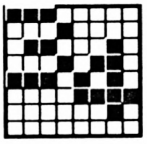
167
&HA7
&X10100111



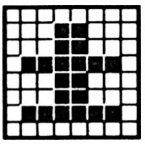
168
&HA8
&X10101000



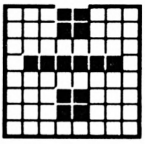
169
&HA9
&X10101001



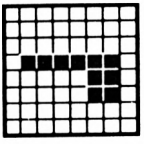
170
&HAA
&X10101010



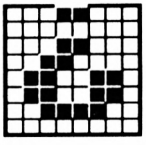
171
&HAB
&X10101011



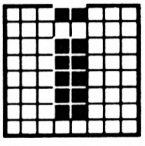
172
&HAC
&X10101100



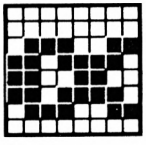
173
&HAD
&X10101101



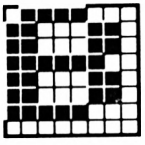
174
&HAE
&X10101110



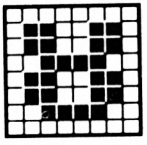
175
&HAF
&X10101111



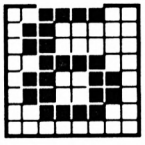
176
&HB0
&X10110000



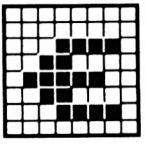
177
&HB1
&X10110001



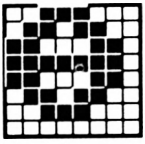
178
&HB2
&X10110010



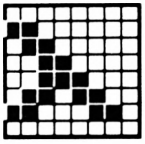
179
&HB3
&X10110011



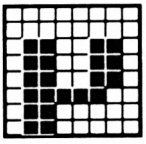
180
&HB4
&X10110100



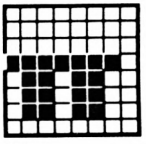
181
&HB5
&X10110101



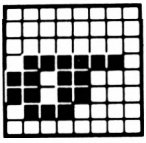
182
&HB6
&X10110110



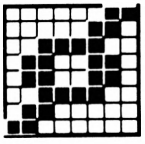
183
&HB7
&X10110111



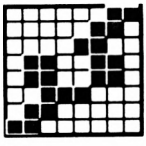
184
&HB8
&X10111000



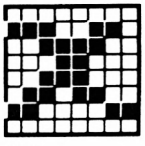
185
&HB9
&X10111001



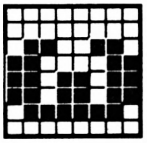
186
&HBA
&X10111010



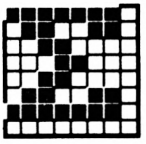
187
&HBB
&X10111011



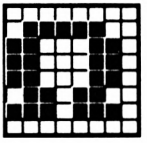
188
&HBC
&X10111100



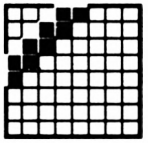
189
&HBD
&X10111101



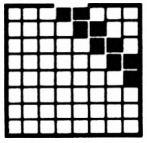
190
&HBE
&X10111110



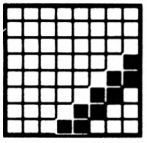
191
&HBF
&X10111111



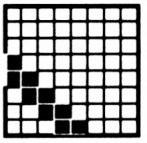
192
&HC0
&X11000000



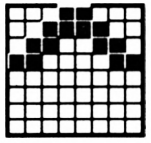
193
&HC1
&X11000001



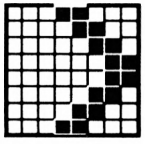
194
&HC2
&X11000010



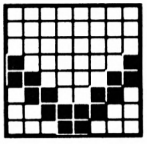
195
&HC3
&X11000011



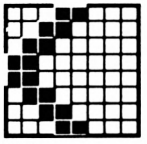
196
&HC4
&X11000100



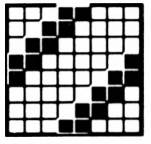
197
&HC5
&X11000101



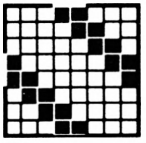
198
&HC6
&X11000110



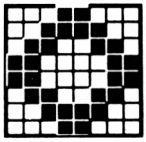
199
&HC7
&X11000111



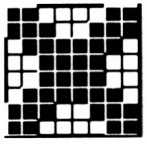
200
&HC8
&X11001000



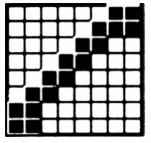
201
&HC9
&X11001001



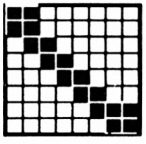
202
&HCA
&X11001010



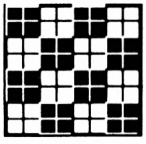
203
&HCB
&X11001011



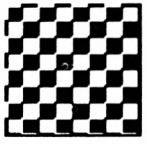
204
&HCC
&X11001100



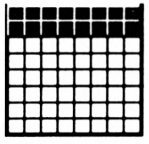
205
&HCD
&X11001101



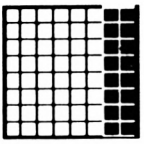
206
&HCE
&X11001110



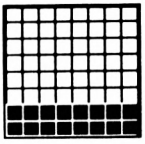
207
&HCF
&X11001111



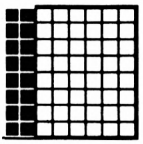
208
&HD0
&X11010000



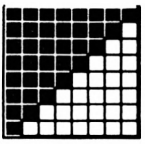
209
&HD1
&X11010001



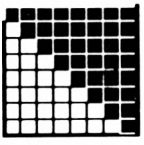
210
&HD2
&X11010010



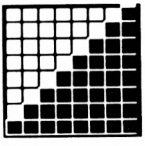
211
&HD3
&X11010011



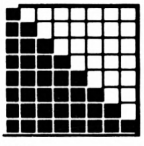
212
&HD4
&X11010100



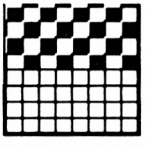
213
&HD5
&X11010101



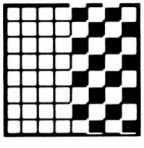
214
&HD6
&X11010110



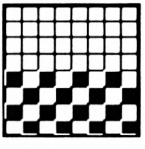
215
&HD7
&X11010111



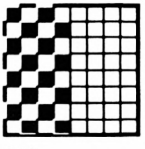
216
&HD8
&X11011000



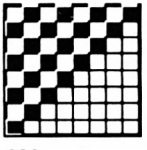
217
&HD9
&X11011001



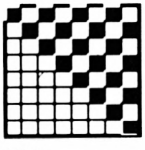
218
&HDA
&X11011010



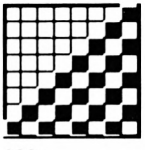
219
&HDB
&X11011011



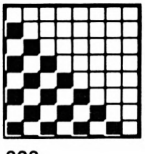
220
&HDC
&X11011100



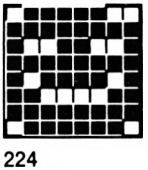
221
&HDD
&X11011101



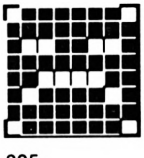
222
&HDE
&X11011110



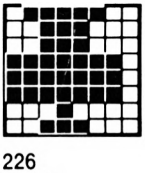
223
&HDF
&X11011111



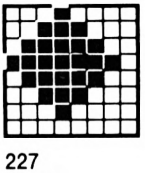
224
&HE0
&X11100000



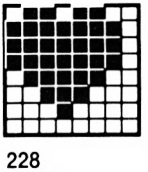
225
&HE1
&X11100001



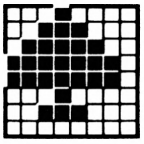
226
&HE2
&X11100010



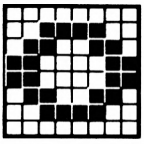
227
&HE3
&X11100011



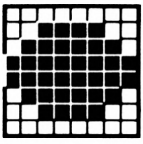
228
&HE4
&X11100100



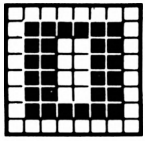
229
&HE5
&X11100101



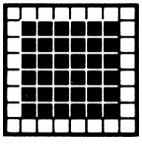
230
&HE6
&X11100110



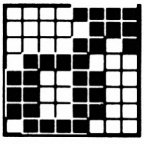
231
&HE7
&X11100111



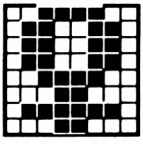
232
&HE8
&X11101000



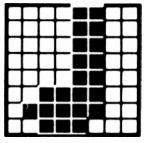
233
&HE9
&X11101001



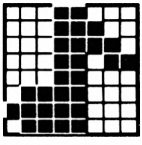
234
&HEA
&X11101010



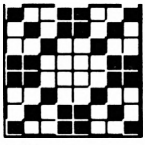
235
&HEB
&X11101011



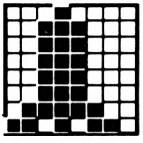
236
&HEC
&X11101100



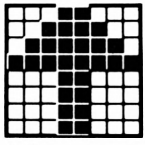
237
&HED
&X11101101



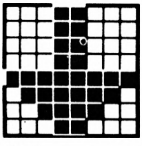
238
&HEE
&X11101110



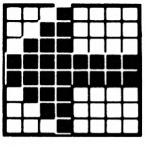
239
&HEF
&X11101111



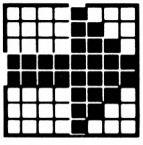
240
&HF0
&X11110000



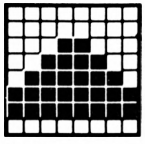
241
&HF1
&X11110001



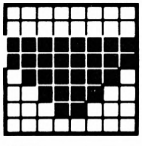
242
&HF2
&X11110010



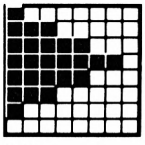
243
&HF3
&X11110011



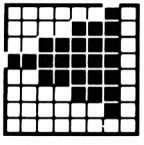
244
&HF4
&X11110100



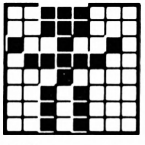
245
&HF5
&X11110101



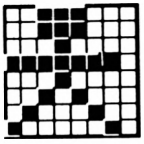
246
&HF6
&X11110110



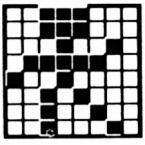
247
&HF7
&X11110111



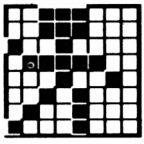
248
&HF8
&X11111000



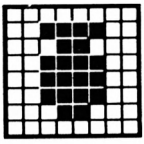
249
&HF9
&X11111001



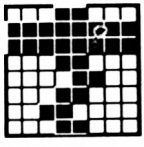
250
&HFA
&X11111010



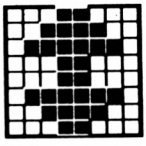
251
&HFB
&X11111011



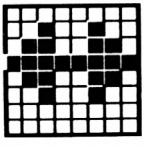
252
&HFC
&X11111100



253
&HFD
&X11111101



254
&HFE
&X11111110

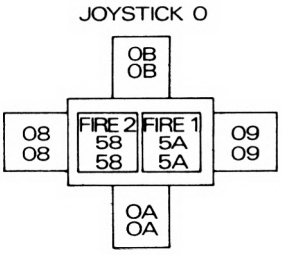


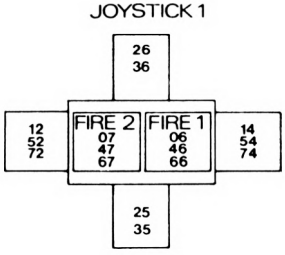
255
&HFF
&X11111111

Part 4: Key references

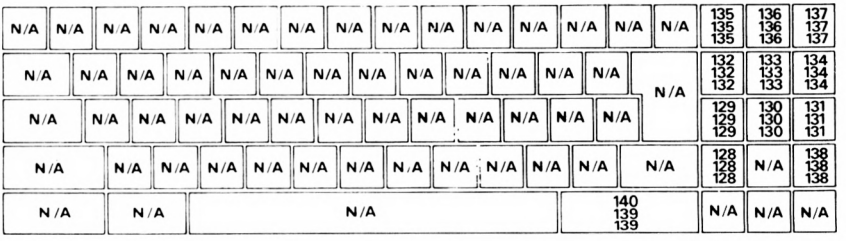
Default ASCII values (HEX)

N/A	21 31	7E 22 32	23 33	24 34	25 35	26 36	27 37	28 38	29 39	1F 5F 30	3D 2D	1E A3 5E	10 10 10	7F 7F 7F	N/A	N/A	N/A
E1 09 09	11 51 71	17 57 77	05 45 65	12 52 72	14 54 74	19 59 79	15 55 75	09 49 69	0F 4F 6F	10 50 70	00 7C 40	1B 7B 5B	0D 0D 0D		N/A	N/A	N/A
N/A	01 41 61	13 53 73	04 44 64	06 46 66	07 47 67	08 48 68	0A 4A 6A	0B 4B 6B	0C 4C 6C	2A 3A	2B 3B	1D 7D 5D			N/A	N/A	N/A
N/A	1A 5A 7A	18 58 78	03 43 63	16 56 76	02 42 62	0E 4E 6E	0D 4D 6D	3C 2C	3E 2E	3F 2F	1C 60 5C	N/A			N/A	F8 F4 F0	N/A
N/A	E0 E0 E0				20 20							N/A			FA F6 F2	F9 F5 F1	FB F7 F3





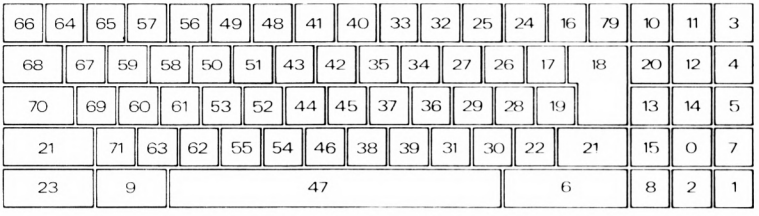
Expansion characters, default locations and values

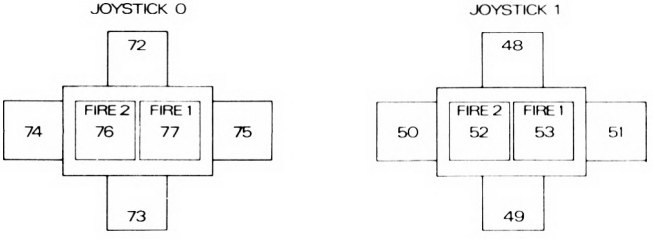


EXPANSION CHARACTER	DEFAULT SETTING	
	CHARACTER	ASCII VALUE
0 (128)	0	&30
1 (129)	1	&31
2 (130)	2	&32
3 (131)	3	&33
4 (132)	4	&34
5 (133)	5	&35
6 (134)	6	&36
7 (135)	7	&37
8 (136)	8	&38
9 (137)	9	&39
10 (138)	.	&2E
11 (139)	[RETURN]	&0D
12 (140)	RUN"[RETURN]	&52 &55 &4E &22 &0D

Note: Expansion characters 13 to 31 (141 to 159) have a null value by default. They have values assigned using the BASIC command KEY, and are assigned to keys using the command KEY DEF.

Key and joystick numbers





Part 5: Sound

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 ‹tone period› setting has to be an integer. The RELATIVE ERROR is the percentage ratio of the difference between the required and actual frequency.

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	16.352	3822	-0.007%	
C#	17.324	3608	+0.007%	
D	18.354	3405	-0.007%	
D#	19.445	3214	-0.004%	
E	20.602	3034	+0.009%	
F	21.827	2863	-0.016%	Octave -4
F#	23.125	2703	+0.009%	
G	24.500	2551	-0.002%	
G#	25.957	2408	+0.005%	
A	27,500	2273	+0.012%	
A#	29.135	2145	-0.008%	
B	30.868	2025	+0.011%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	32.703	1911	-0.007%	
C#	34.648	1804	+0.007%	
D	36.708	1703	+0.022%	
D#	38.891	1607	-0.004%	
E	41.203	1517	+0.009%	
F	43.654	1432	+0.019%	Octave -3
F#	46.249	1351	-0.028%	
G	48.999	1276	+0.037%	
G#	51.913	1204	+0.005%	
A	55.000	1136	-0.032%	
A#	58.270	1073	+0.039%	
B	61.735	1012	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	65.406	956	+0.046%	
C#	69.296	902	+0.007%	
D	73.416	851	-0.037%	
D#	77.782	804	+0.058%	
E	82.407	758	-0.057%	
F	87.307	716	+0.019%	Octave -2
F#	92.499	676	+0.046%	
G	97.999	638	+0.037%	
G#	103.826	602	+0.005%	
A	110.000	568	-0.032%	
A#	116.541	536	-0.055%	
B	123.471	506	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	130.813	478	+0.046%	
C#	138.591	451	+0.007%	
D	146.832	426	+0.081%	
D#	155.564	402	+0.058%	
E	164.814	379	-0.057%	
F	174.614	358	+0.019%	Octave -1
F#	184.997	338	+0.046%	
G	195.998	319	+0.037%	
G#	207.652	301	+0.005%	
A	220.000	284	-0.032%	
A#	233.082	268	-0.055%	
B	246.942	253	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	261.626	239	+0.046%	Middle C
C#	277.183	225	-0.215%	
D	293.665	213	+0.081%	
D#	311.127	201	+0.058%	
E	329.628	190	+0.206%	
F	349.228	179	+0.019%	Octave 0
F#	369.994	169	+0.046%	
G	391.995	159	-0.277%	
G#	415.305	150	-0.328%	
A	440.000	142	-0.032%	International A
A#	466.164	134	-0.055%	
B	493.883	127	+0.356%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	523.251	119	-0.374%	
C#	554.365	113	+0.229%	
D	587.330	106	-0.390%	
D#	622.254	100	-0.441%	
E	659.255	95	+0.206%	
F	698.457	89	-0.543%	Octave 1
F#	739.989	84	-0.548%	
G	783.991	80	+0.350%	
G#	830.609	75	-0.328%	
A	880.000	71	-0.032%	
A#	932.328	67	-0.055%	
B	987.767	63	-0.435%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	1046.502	60	+0.462%	
C#	1108.731	56	-0.662%	
D	1174.659	53	-0.390%	
D#	1244.508	50	-0.441%	
E	1318.510	47	-0.855%	
F	1396.913	45	+0.574%	
F#	1479.978	42	-0.548%	Octave 2
G	1567.982	40	+0.350%	
G#	1661.219	38	+0.992%	
A	1760.000	36	+1.357%	
A#	1864.655	34	+1.417%	
B	1975.533	32	+1.134%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	2093.004	30	+0.462%	
C#	2217.461	28	-0.662%	
D	2349.318	27	+1.469%	
D#	2489.016	25	-0.441%	
E	2637.021	24	+1.246%	
F	2793.826	22	-1.685%	
F#	2959.955	21	-0.548%	Octave 3
G	3135.963	20	+0.350%	
G#	3322.438	19	+0.992%	
A	3520.000	18	+1.357%	
A#	3729.310	17	+1.417%	
B	3951.066	16	+1.134%	

The above values are all calculated from International A as follows:

```
FREQUENCY =440*(2↑(OCTAVE+((N-10)/12)))
   PERIOD =ROUND(62500/FREQUENCY)
```

....where N is 1 for C, 2 for C#, 3 for D, etc.

Part 6: BASIC Error messages

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 sub-routine.

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 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 or 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 appending 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, 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 an 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 (i.e. in an **ON ERROR GOTO** routine).

20 Unexpected RESUME

RESUME is only valid while in error processing mode (i.e. in an **ON ERROR GOTO** routine).

21 Direct command found

When loading a file, 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 the file input stream.

25 File type error

The file being read is not of a suitable type. **OPENIN** is only prepared to open ASCII text files. Similarly, **LOAD**, **RUN**, etc, are only prepared to deal with file types produced by **SAVE**.

26 NEXT missing

Cannot find a NEXT to match a FOR command. A line number accompanying this message indicates the FOR command to which this error applies.

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, i.e. a command preceded by a bar |.

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.

31 File not open

(See the section ahead entitled 'Disc errors'.)

32 Broken in

(See the section ahead entitled 'Disc errors'.)

AMSDOS Disc errors

There are several errors that may occur during the processing of any filing operations. BASIC handles all such errors as ERRor number 32, however more specific information may be returned by the function DERR when this error number is detected. This returns values as follows:

AMSDOS error	DERR value	Cause of error
0	0 or 22	**[ESC]** has been pressed.
14	142 (128+14)	The stream is not in a suitable state.
15	143 (128+15)	Hard end of file has been reached.
16	144 (128+16)	Bad command, usually an incorrect filename.
17	145 (128+17)	File already exists.
18	146 (128+18)	File does not exist.
19	147 (128+19)	Directory is full.
20	148 (128+20)	Disc is full.
21	149 (128+21)	Disc changed while files were open.
22	150 (128+22)	File is Read/Only.
26	154 (128+26)	Soft end of file has been detected.

If AMSDOS has already reported an error, then bit 7 is set; hence the value of DERR is offset by 128.

Other values returned by DERR originate from the disc controller and are bit significant, always with bit 6 set. Bit 7 indicates whether the error has been reported by AMSDOS (as explained above). The significance of each bit is as follows:

Bit	Significance
0	Address mark missing.
1	Not writable - disc is write protected.
2	No data - can't find the sector.
3	Drive not ready - no disc in the drive.
4	Overrun error.
5	Data error - CRC error.
6	Always set to 1 to indicate error from disc controller.
7	Set to 1 if error has already been reported by AMSDOS.

ERR may also return 31 if access was attempted when no file was open. The usual way in which one may use ERR and DERR would be to include an ON ERROR GOTO which calls a short routine that checks if ERR has the value 31 or 32, and if it is 32, DERR could be interrogated to give more detailed information regarding the nature of the error. For example:

```
10 ON ERROR GOTO 1000
20 OPENOUT "myfile.asc"
30 WRITE #9,"test-data"
40 CLOSEOUT
50 END
1000 amsdoserr=(DERR AND &7F):REM mask off bit 7
1010 IF ERR<31 THEN END
1020 IF ERR=31 THEN PRINT "are you sure you've typed
     line 20 correctly?":END
1030 IF amsdoserr=20 THEN PRINT "disc is full, suggest
     you use a new data disc":END
1040 IF amsdoserr=&X01001000 THEN PRINT "put a disc in
     the drive, then press a key":WHILE INKEY$="":
     WEND:RESUME
1050 END
```

Part 7: BASIC Keywords

The following is a list of all AMSTRAD 464/6128 BASIC keywords. As such, they are reserved and can NOT be used as variable names.

```
ABS, AFTER, AND, ASC, ATN, AUTO

BIN$, BORDER

CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG, CLOSEIN,
CLOSEOUT, CLS, CONT, COPYCHR$, COS, CREAL, CURSOR

DATA, DEC$, DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE,
DERR, DI,DIM, DRAW, DRAWR

EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR, ERROR,
EVERY, EXP

FILL, FIX, FN, FOR, FRAME, FRE
```

GOSUB, GOTO, GRAPHICS

HEX$, HIMEM

IF, INK, INKEY, INKEY$, INP, INPUT, INSTR, INT

JOY

KEY

LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10, LOWER$

MASK, MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER

NEXT, NEW, NOT

ON, ON BREAK, ON ERROR GOTO Ø, 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

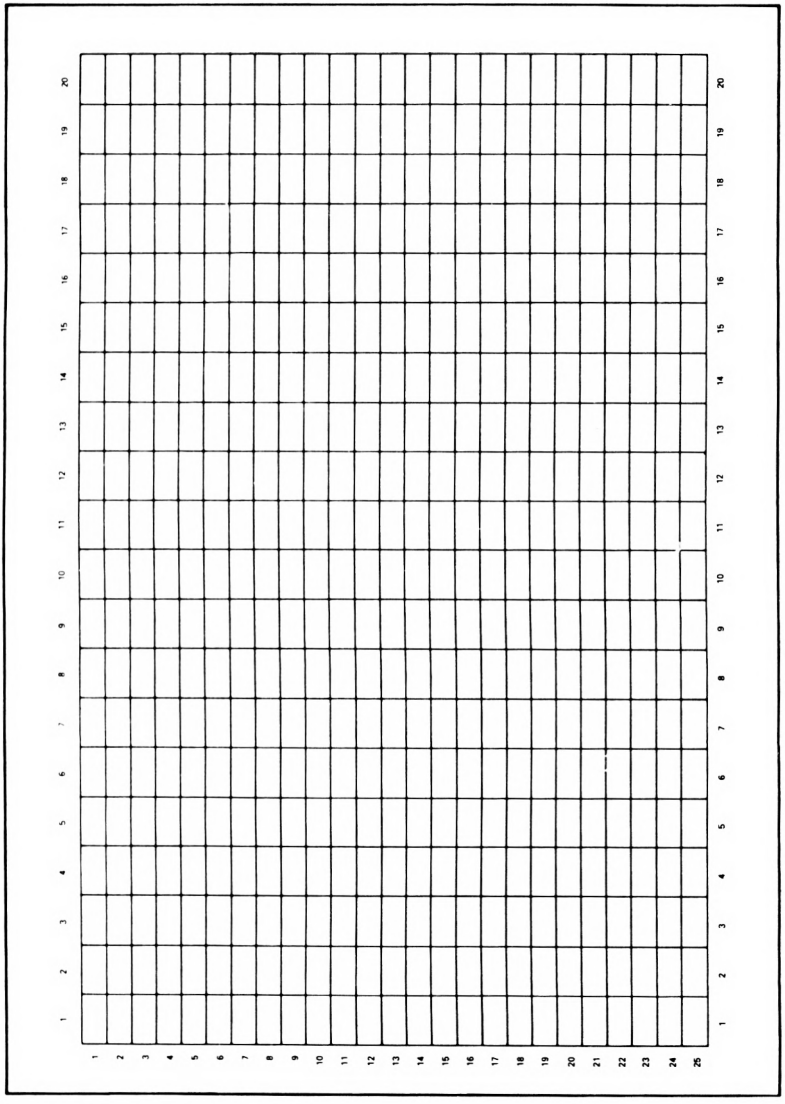
XOR, XPOS

YPOS

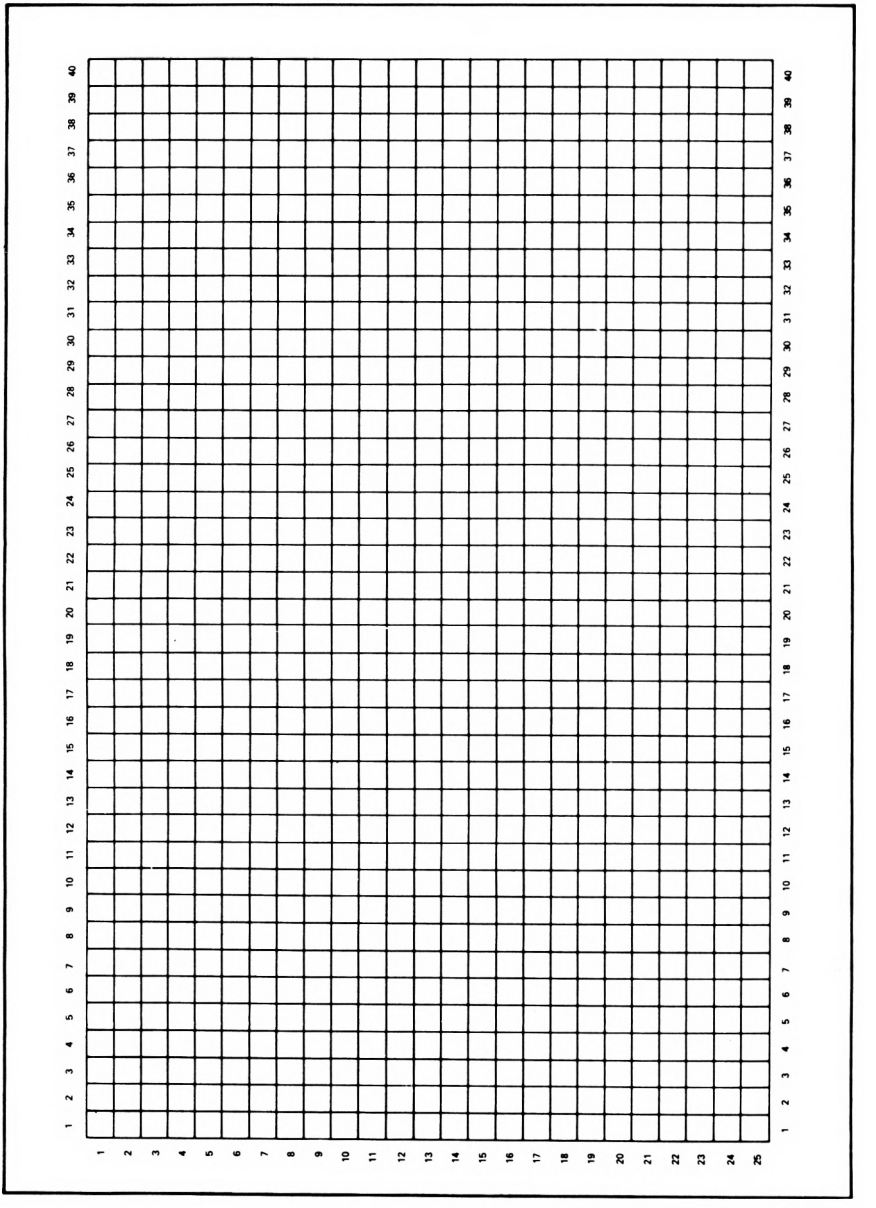
ZONE

Part 8: Planners

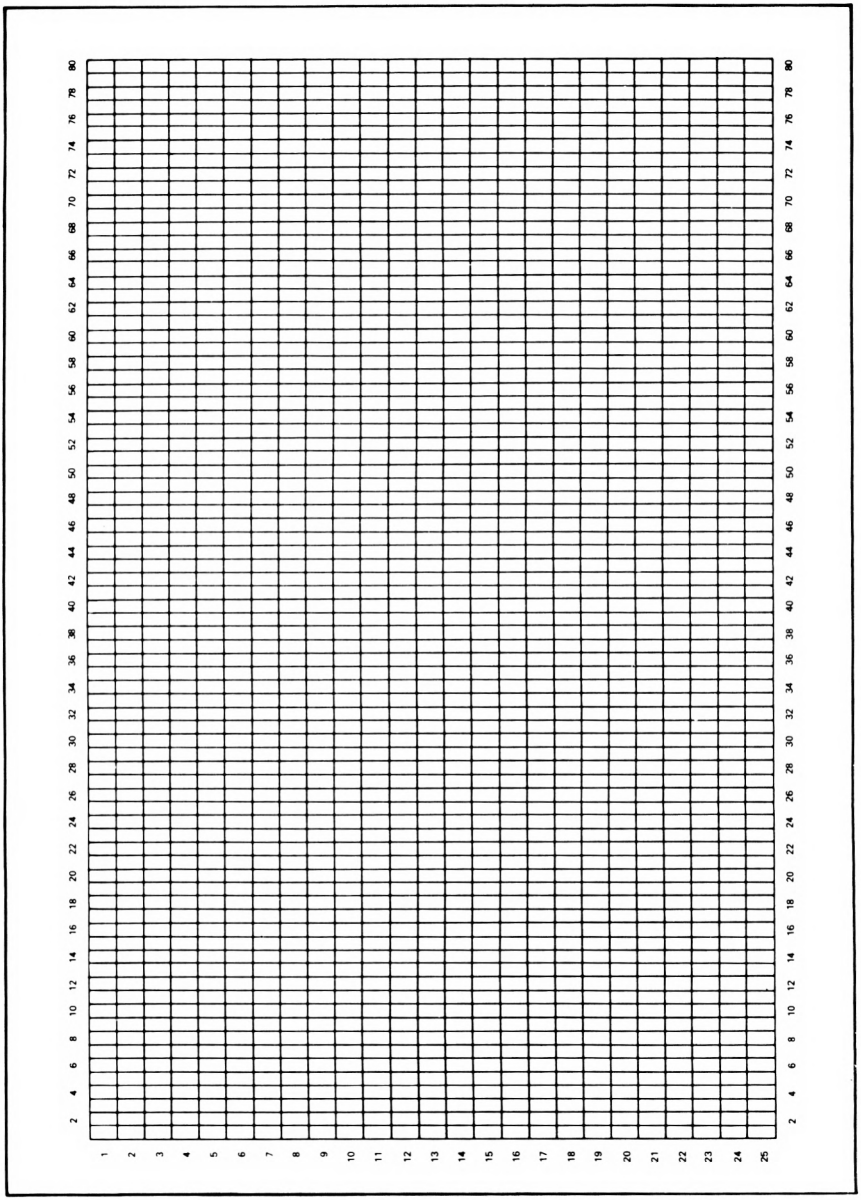
Text and window planner - MODE 0 (20 columns)



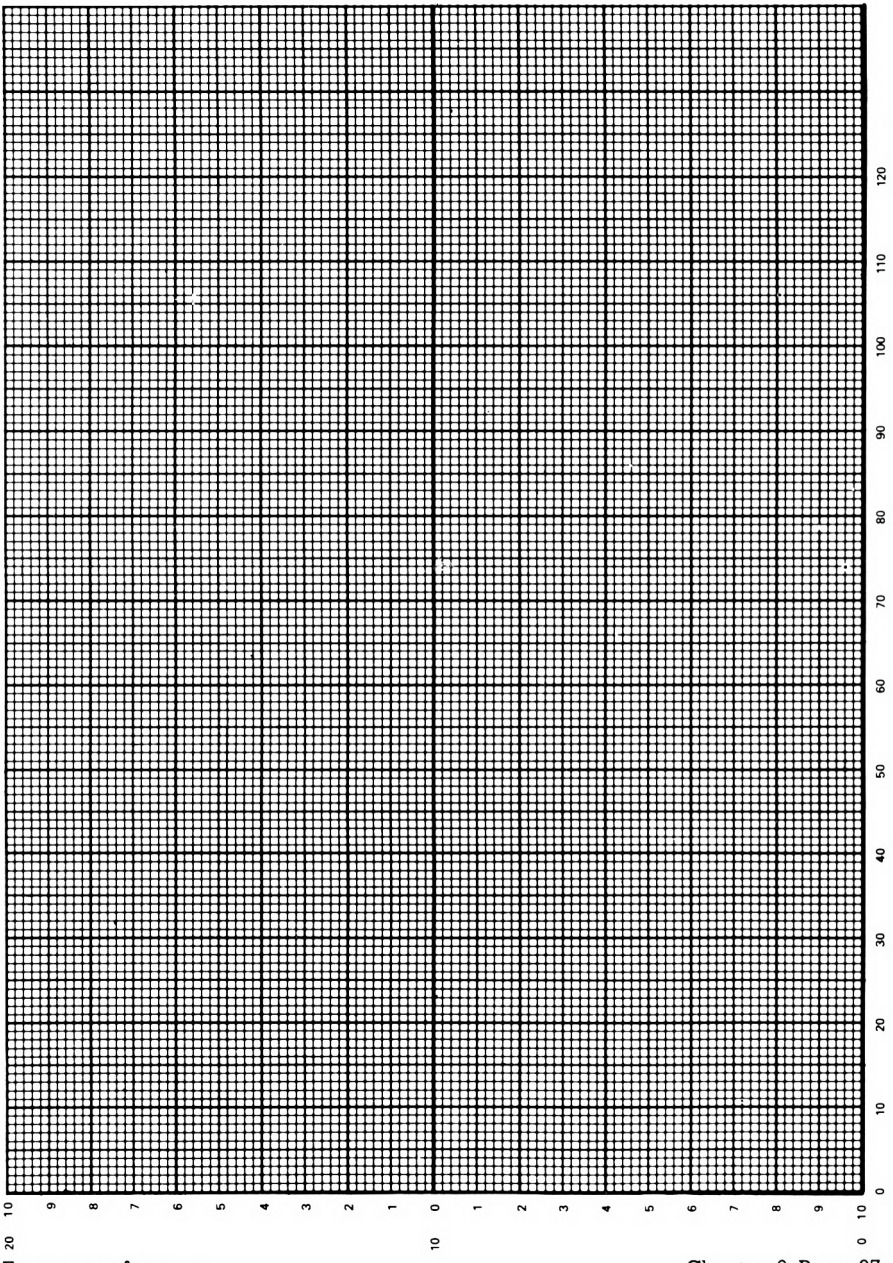
Text and window planner - MODE 1 (40 columns)



Text and window planner - MODE 2 (80 columns)

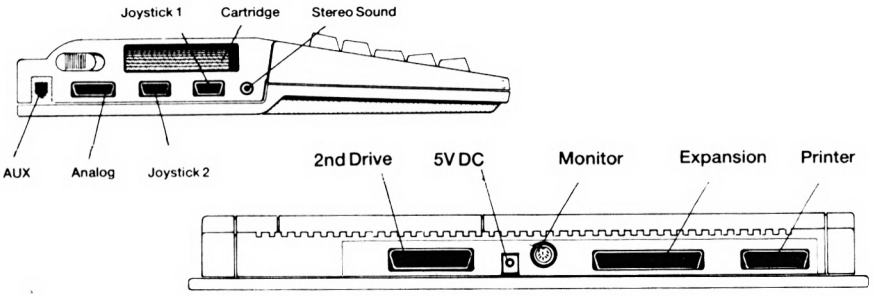


Sound envelope/music planner



Part 9: Connections

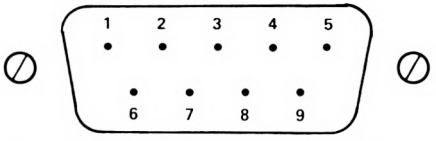
464/6128 Input/Output Sockets



Aux Socket

PIN 1	+5V	PIN 4	FIRE 2
PIN 2	COMMON	PIN 5	FIRE 1
PIN 3	LPEN	PIN 6	GND

Games Adaptor/Joystick Socket



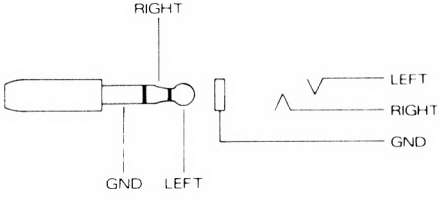
PIN 1	UP	PIN 6	FIRE 2
PIN 2	DOWN	PIN 7	FIRE 1
PIN 3	LEFT	PIN 8	COMMON
PIN 4	RIGHT	PIN 9	COM 2
PIN 5	SPARE		

NOTE: PIN 9 IS NOT COM 2 ON JOYSTICK 2 CONNECTOR

Monitor Socket

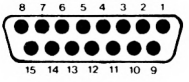
PIN 1	$\overline{\text{SYNC}}$	PIN 5	BLUE
PIN 2	GREEN	PIN 6	L SOUND
PIN 3	LUM	PIN 7	R SOUND
PIN 4	RED	PIN 8	GND

Stereo Socket



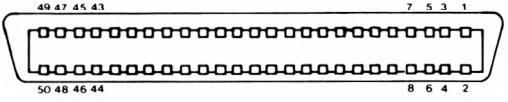
PIN 1 LEFT CHANNEL
PIN 2 RIGHT CHANNEL
PIN 3 GND

Analog Socket



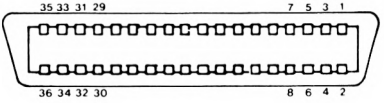
ANALOG STICK 1		ANALOG STICK 2	
PIN 1	GND (POT COMMON)	PIN 9	GND (POT COMMON)
PIN 2	FIRE 1	PIN 10	FIRE 1
PIN 3	X1	PIN 11	X2
PIN 4	COM1 (SWITCHES)	PIN 12	COM2 (SWITCHES)
PIN 5	+5V	PIN 13	Y2
PIN 6	Y1	PIN 14	FIRE 2
PIN 7	FIRE 2	PIN 15	GND (POT COMMON)
PIN 8	GND (POT COMMON)		

Expansion Socket



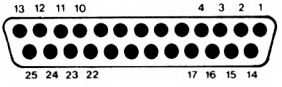
PIN	1	SOUND	PIN	18	A0	PIN	35	$\overline{\text{INT}}$
PIN	2	GND	PIN	19	D7	PIN	36	$\overline{\text{NMI}}$
PIN	3	A15	PIN	20	D6	PIN	37	**BUSRQ**
PIN	4	A14	PIN	21	D5	PIN	38	$\overline{\text{BUSAK}}$
PIN	5	A13	PIN	22	D4	PIN	39	READY
PIN	6	A12	PIN	23	D3	PIN	40	$\overline{\text{BUS RESET}}$
PIN	7	A11	PIN	24	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	○
PIN	17	A1	PIN	34	$\overline{\text{HALT}}$			

Disc Drive 2 Socket (6128 only)



PIN 1		**PIN 2**	**GND**
PIN 3		**PIN 4**	**GND**
PIN 5		**PIN 6**	**GND**
PIN 7	**INDEX**	**PIN 8**	**GND**
PIN 9		**PIN 10**	**GND**
PIN 11	**DRIVE 1 SELECT**	**PIN 12**	**GND**
PIN 13		**PIN 14**	**GND**
PIN 15	**MOTOR ON**	**PIN 16**	**GND**
PIN 17	**DIRECTION SELECT**	**PIN 18**	**GND**
PIN 19	**STEP**	**PIN 20**	**GND**
PIN 21	**WRITE DATA**	**PIN 22**	**GND**
PIN 23	**WRITE GATE**	**PIN 24**	**GND**
PIN 25	**TRACK Ø**	**PIN 26**	**GND**
PIN 27	**WRITE PROTECT**	**PIN 28**	**GND**
PIN 29	**READ DATA**	**PIN 30**	**GND**
PIN 31	**SIDE 1 SELECT**	**PIN 32**	**GND**
PIN 33	**READY**	**PIN 34**	**GND**
PIN 35		**PIN 36**	**GND**

Printer Port



PIN 1	$\overline{\text{STROBE}}$	PIN 17	GND
PIN 2	D0	PIN 18	GND
PIN 3	D1	PIN 19	GND
PIN 4	D2	PIN 20	GND
PIN 5	D3	PIN 21	GND
PIN 6	D4	PIN 22	GND
PIN 7	D5	PIN 23	GND
PIN 8	D6	PIN 24	GND
PIN 9	D7	PIN 25	GND
PIN 11	BUSY		
PIN 16	+5V	All other pins NC	

Part 10: Printers

Printer interfacing

The 464/6128 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 socket at the rear of the computer, and the connector on the parallel printer.

The actual pin interface details are illustrated in part 9 of this chapter.

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 set-up commands required, and the output is directed to the printer by specifying stream #8.

Although the 464/6128 PRINTER port is envisaged for use with low cost dot-matrix printers; with a suitable interface it will support daisywheel printers, graphics plotters, and multi-colour ink-jet printers. The key to compatibility is the standard parallel interface.

Printer configuration

A facility is provided whereby special characters which may appear on the screen and which are supported by the printer, will be printed even though the character codes for the screen and printer may be different. The majority of these symbols will only be available when the printer is switched to one of its foreign language modes. For example:

```
PRINT CHR$(&AØ)
^
PRINT #8,CHR$(&AØ)
```

^ is printed on the printer.

This works even though the character code for a circumflex accent on the printer is &5E. In other words, the printer routine has recognised &AØ as one of the codes held in a printer translation table, and has translated it to &5E so that the same character appearing on the screen will be printed by the printer. The code &5E will produce a circumflex accent on a printer no matter which language mode the printer is set to (this is not true for all the characters in the translation table). The other characters in the table are as shown in the following table:

CHR$	Character On Screen	Printer Translation	U.K.	U.S.A.	France	Germany	Spain
&A0	^	&5E	^	^	^	^	^
&A2	..	&7B	†	†	†	†	..
&A3	£	&23	£	#	#	#	Pt
&A6	§	&40	†	†	†	§	†
&AE	¿	&5D	†	†	†	†	¿
&AF	¡	&5B	†	†	†	†	¡

† For the printed character, refer to your printer instruction manual.

The above is an extract from the default translations, which can be changed if required.

Part 11: Joysticks/Paddle Controllers

The built-in software in the computer supports either one or two joysticks. These are treated as part of the keyboard, and as such, may be interrogated by `INKEY` and `INKEY$`.

Note that in the majority of cases, the main 'fire' button on a joystick is interpreted as 'Fire 2' by the 464/6128.

The functions `JOY(Ø)` and `JOY(1)` enable direct inspection of the first and second joysticks respectively. The function returns a bit-significant result which indicates the state of the joystick switches at the last keyboard scan.

The table below indicates the values returned by both joysticks. The `JOY` values are followed by values for use in statements which take key numbers as parameters (i.e. `INKEY` and `KEY DEF`).

STATUS	JOY COMMAND		KEY VALUES		
	BIT SET	VALUE RETURNED	FIRST JOYSTICK	SECOND JOYSTICK	EQUIVALENT KEY
Up	0	1	72	48	'6'
Down	1	2	73	49	'5'
Left	2	4	74	50	'R'
Right	3	8	75	51	'T'
Fire 2	4	16	76	52	'G'
Fire 1	5	32	77	53	'F'

Note that when key values for the SECOND joystick are returned, the computer cannot tell whether those values have been generated by the joystick or by the equivalent keyboard key (indicated in the last column of the previous table). This means that the keyboard can be used as a substitute for the second joystick.

Part 12: Disc organisation

The BIOS supports two different disc formats: SYSTEM format and DATA ONLY format. Under AMSDOS, the format of a disc is automatically detected each time a disc with no open files is accessed. To permit this automatic detection, each format has unique sector numbers.

3 inch discs are double sided, but only one side may be accessed at a time depending on which way round the user inserts the disc. There may be a different format on each side.

Common to both formats:

Single sided (the two sides of a 3 inch disc are treated separately).
512 byte physical sector size.
40 tracks numbered 0 to 39.
1024 byte CP/M block size.
64 directory entries.

SYSTEM format

9 sectors per track numbered &41 to &49.
2 reserved tracks.

The system format is the main format supported, since CP/M can only be loaded from a system format disc. The reserved tracks are used as follows:

Track 0 sector &41	:	boot sector for CP/M Plus
Track 0 sectors &42 to &49	:	unused
	:	
	:	
Track 1 sectors &41 to &49	:	

Note that VENDOR format is a special version of system format which does not contain any system software on the two reserved tracks. It is intended for use in software distribution.

DATA ONLY format

9 sectors per track numbered &C1 to &C9.
0 reserved tracks.

Part 13: Resident System eXtensions (RSX's)

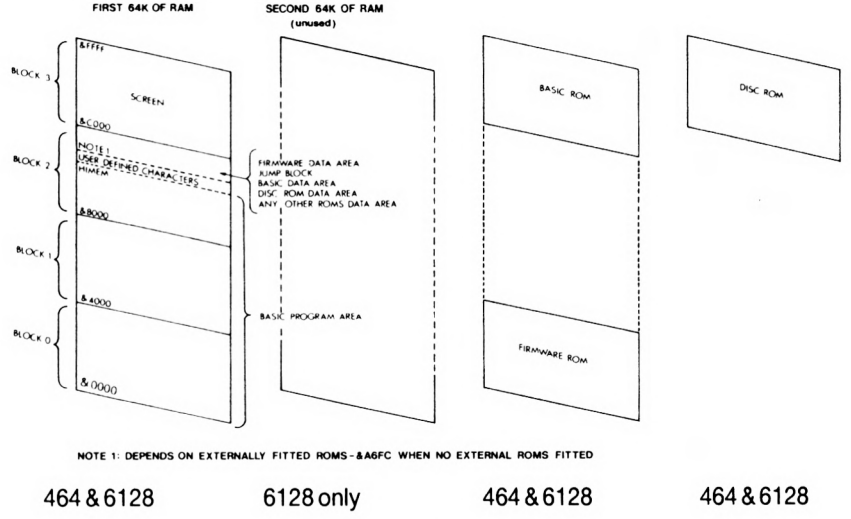
External commands were introduced in Chapter 5 (about AMSDOS). Fundamentally, an external command is a way of extending the repertoire of the BASIC by adding new commands signalled by a | prefix. The machine instructions for the AMSDOS new commands are included in a ROM, and the necessary housekeeping to add the commands is carried out automatically when the 464/6128 starts BASIC.

It is also possible to add further external commands (after BASIC has started) by loading the machine instructions into RAM. Such new commands are called RSX's and operate in exactly the same way as ROM based extensions. RSX's have to be loaded from disc (or tape) each time the 464/6128 starts (or restarts) BASIC. Normally, RSX's will be used for controlling some sort of intelligent peripheral, such as a light pen or speech synthesiser.

Chapter 7 describes the use of RSX's to access the 6128's second 64K of memory.

Part 14: Memory

The 6128 contains 128K of RAM, and uses 48K of the System Cartridge ROM. The System Cartridge ROM contains 16K for BASIC, 16K for General Firmware, 16K for Disc Firmware, 64K for a game and 16K is reserved.This is available to BASIC 1.1 as shown below. The 464 differs only by having 64K of RAM. However, the Disc Firmware is automatically ineffective. The first 64K of RAM is nominally divided into four blocks (each of 16K) numbered Block 0 to Block 3. The screen uses Block 3 and the upper section of Block 2 is filled by system variables as indicated.



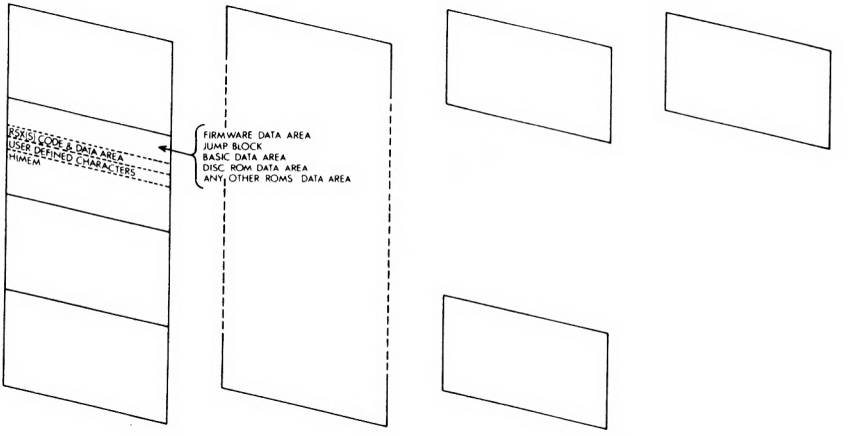
Memory Map for BASIC 1.1

The user defined characters are initially positioned immediately above HIMEM. HIMEM may be altered by a **MEMORY** command but is also automatically lowered by 4K to create a buffer whilst AMSDOS files are open. The number of user defined characters can only be altered if HIMEM is unchanged since the last time they were set (unless the last time they were set to 'no user defined characters' by a **SYMBOL AFTER 256** command). When BASIC starts, the user defined characters are set as if a **SYMBOL AFTER 240** command had been issued.

It is, therefore, prudent to collapse the user defined character area before permanently changing HIMEM, then re-establishing the user defined characters in the new position. This will allow subsequent programs to alter the **SYMBOL AFTER** assignment.

The example below shows this scheme in operation when HIMEM is lowered in conjunction with loading an RSX.

```
100 SYMBOL AFTER 256 ' collapse user defined
    characters
110 rsxaddress=HIMEM-rsxlength
120 MEMORY rsxaddress-1
130 LOAD "rsxcode",rsxaddress
140 CALL rsxaddress ' log on rsx
150 SYMBOL AFTER 140 ' restore user defined
    characters
```



Memory Map with RSX loaded in recommended position

Additional I/O

Most I/O port addresses are reserved by the computer - in particular, addresses below &7FFF should not be used at all.

It is intended that the part of the address A0 - A7 should reflect the type of external I/O device, and that address lines A8 and A9 may be decoded to select registers within the I/O device. Of the remaining address lines, only A10 must be decoded (as low) whilst lines A11 to A15 are high. Thus each device may have registers addressed as &F8??, &F9??, &FA??, and &FB??, where ?? is in the range DC to DF for communications interfaces, and E0 to FE for other user peripherals.

Note that Z80 instructions which place the B register on the upper half of the address bus (A15 - A8) must be used.

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.

Part 15: CP/M Plus Terminal Emulator

In part 1 of this chapter, a table of control characters (together with their respective actions) was illustrated. These actions are performed when text is sent to the screen by BASIC and were chosen both to be simple to use and to reflect the facilities available in the Firmware Text VDU. These facilities are unique to AMSTRAD computers, and software must therefore be adapted to their use.

In the Business and Commercial environment of CP/M Plus software, it is normal to expect a range of 'standard' Text VDU facilities in order that software is easily portable and 'installable' from one machine to another. The CP/M Plus implementation on the 6128 includes a Terminal Emulator which provides facilities very similar to a Zenith Z19/Z29 VDU. The installation procedure for CP/M Plus software will normally include, as standard, an option for this type of terminal.

The facilities offered by the CP/M Plus Terminal Emulator include many of those previously offered by the Firmware Text VDU although different control codes are required.

There are a considerable number of new and more sophisticated operations.

Characters in the range &20 to &FF are displayed at the current cursor position. If the cursor is not at the right-hand column, then it is moved right by one column. If the cursor is at the right-hand column and wrapping is enabled, then it is moved to the left-hand column on the next line, scrolling up if necessary.

Characters in the range &00 to &1F are interpreted as control codes as follows:

&07	BEL	Sounds a beep.
&08	BS	Backspace. Move left one column. If the cursor is on the left-hand column and not on the top row and wrapping is enabled, then it is moved to the right-hand column of the row above.
&0A	LF	Linefeed. Move cursor down one line, scrolling up if necessary.
&0D	CR	Carriage return. Move cursor to left-hand column of current row.
&1B	ESC	Introduce an escape sequence.

All other control codes are ignored.

The following escape sequences are recognised. Any other characters following an escape are displayed, and the cursor advanced. This feature may be used to display the characters corresponding to control codes &00 to &1F. (Note that many applications languages expand the control code &09 (TAB) to a number of spaces and therefore the sequence [ESC][TAB] will often not display the character for &09.)

[ESC]0	Disable status line. Disc system messages will appear with CRT output. CRT may use the bottom line of the screen.
[ESC]1	Enable status line. Disc system messages will appear on the bottom line of the screen.
[ESC]2 ‹n›	Change character set (see part 16 of this chapter). ‹n› is the language parameter which is masked with &07. Certain character matrices in the range &20 to &7F are swapped with other characters in the range &80 to &FF. The action of this command is very similar to that used to control printers which have software-selectable international character sets.

‹n›=0	USA
‹n›=1	France
‹n›=2	Germany
‹n›=3	UK
‹n›=4	Denmark
‹n›=5	Sweden
‹n›=6	Italy
‹n›=7	Spain

[ESC]3 ‹m›	Change screen mode. ‹m›=screen mode+&20. The value is masked with &3 to give the mode 0 to 2. Mode 3 is ignored. The screen is cleared, but the cursor position is unaffected.
[ESC]A	Cursor up. If on top row, then do nothing.
[ESC]B	Cursor down. If on bottom row, then do nothing.
[ESC]C	Cursor forward. If on right-hand column, then do nothing.
[ESC]D	Cursor backwards. If on left-hand column, then do nothing.
[ESC]E	Erase page. The cursor position is unaffected. This command clears the entire screen, even if set to 24x80 mode. (Other escape codes will only affect the 24x80 area when 24x80 mode is set).
[ESC]H	Home cursor. Move cursor to top row, left-hand column.
[ESC]I	Reverse index. Move cursor up one row. Scroll screen down if necessary.
[ESC]J	Erase to end of page. Includes character at cursor position. The cursor position is unaffected.
[ESC]K	Erase to end of line. Includes character at cursor position. The cursor position is unaffected.
[ESC]L	Insert Line. All rows below and including the cursor line are scrolled down. The cursor row is cleared. The cursor position is unaffected.
[ESC]M	Delete line. All rows below and including the cursor row are scrolled up. The bottom row is cleared. The cursor position is unaffected.
[ESC]N	Delete character. All characters to the right of the cursor are shuffled left one character position. The character at the end of the row is cleared. The cursor position is unaffected.
[ESC]Y ‹r› ‹c›	Moves cursor to given position. If position is beyond the edge of the screen, then the cursor is moved to the edge of the screen. ‹r›=row position+&20, ‹c›=column position+&20. The top left-hand corner is row 0, column 0.

[ESC]b ‹cp›

Set foreground colour. Affects all characters on the screen. ‹cp› is the colour parameter and is masked with &3F then treated as three 2-bit numbers each specifying the intensity of one of the three primary colours: bits 0,1 for Blue, bits 2,3 for Red, and bits 4,5 for Green. When running CP/M Plus, the 464/6128 provides three levels of intensity, mapped onto the four specifiable levels as:

464/6128	Zero intensity	Half Intensity	Full intensity
Colour bits	00 binary	01 or 10 binary	11 binary

[ESC]c ‹cp›

Set background colour. Affects all the background and border on the screen. The colour is specified as above.

[ESC]d

Erase to beginning of page. Includes character at cursor position. The cursor position is unaffected.

[ESC]e

Enable cursor blob. To prevent unsightly flashing, the cursor is not turned on during normal text outputting until 1/10 second after the last character was written.

[ESC]f

Disable cursor blob.

[ESC]j

Save the cursor position.

[ESC]k

Restore cursor position as saved by [ESC]j.

[ESC]l

Erase line. The cursor position is unaffected.

[ESC]o

Erase to beginning of line. Includes character at cursor position. The cursor position is unaffected.

[ESC]p

Enter inverse video mode. Printable characters are written with the foreground and background colours reversed.

[ESC]q

Exit inverse video mode.

[ESC]v

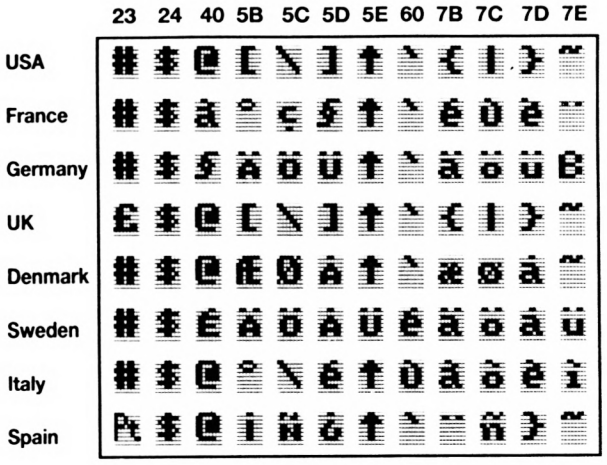
Wrap at end of line.

[ESC]w	Discard at end of line.
[ESC]x	Enter 24x80 mode. Some applications programs may require a 'standard' 24x80 screen. This command will enable such a screen regardless of the full size of the screen which may depend upon machine, country and whether or not the status line is enabled. The screen is cleared.
[ESC]y	Exit 24x80 mode. The screen is cleared.

Part 16: CP/M Plus Character Set

In part 10 of this chapter, a printer translation table was described. The purpose of this table is to convert certain of the characters from the BASIC character set into a form where they can be printed when a language selection is made on the printer. The facility is somewhat limited because very few printers' foreign characters actually appear in the BASIC character set.

Although this printer translation scheme still operates under CP/M Plus, the character set has been enhanced to allow an almost complete correspondence between the screen characters and the printed characters. (The one missing screen symbol is the Swedish Currency Symbol replacing the $). The table below confirms the arrangement:



CP/M Plus International Character Set

Operating the machine in a foreign environment requires two actions:

1. The printer itself should be set to the required language (often by small (DIP) switches, though some printers allow control codes to be sent).

2. The required screen character set must be invoked, either by the transient command:

   `LANGUAGE` ‹n›

   ....or by sending

   [ESC]2 ‹n›

   ....to the Terminal Emulator.

In practice, the initialisation can be performed as part of the `PROFILE.SUB` operation, using the `LANGUAGE` and `SETLST` facilities. CP/M Plus is shipped with the USA environment, mainly because the keyboard displays a # over the **3** key. It is common practice to operate in the UK environment when doing word processing.

7 bit software....

Although this foreign language facility is very powerful, it has to be stressed that the 'normal' (or USA) characters which have been replaced by the 'local' foreign characters are no longer displayable. This is a common and intractable compromise when dealing with 7-bit software. Nearly all available software (including most CP/M Plus utilities, Word processors, and languages) operate with only a 7 bit character set. In the UK it is fairly well accepted that the # disappears to be replaced by the £ not only in word processing (where it is regarded as desirable) but also in program listings, e.g. `LIST £8`, where it is regarded as undesirable. However, in time, everyone becomes used to performing the mental transformation.

Unfortunately, the other foreign language transformations also replace characters such as the vertical bar and square brackets, and although the resulting availability of accented characters improves the readability of text, in situations where bars and brackets are required by the application program, e.g. `DIR [FULL]`, the readability and (if the keytops have been exchanged) the typeability is markedly diminished. Remember that the application program is working in ASCII values, quite oblivious to the shape of the characters on the screen. The problem is that with 7-bit software, there are simply not enough ASCII values to go around.

Working with 8 bit character sets....

The BASIC character set has 256 different symbols, with the values 128 to 255 (&80 to &FF) containing various graphic symbols of particular use in games and home computing (dancing men, hearts/clubs/diamonds/spades, etc.). CP/M Plus also has 256 characters available, but the second 128 are different from those in BASIC, and reflect the International and Business flavour of the CP/M Plus environment:

Characters 0 to 127 (&00 to &7F) **Characters 128 to 255 (&80 to &FF)**

The Standard CP/M Plus Character Set (USA)

Software which is capable of operating with 8 bit characters may use this character set to access all of the foreign language characters at once, without having to specifically change language. In this way, all the 'bar and brackets' type characters will be available as well.

Please note that there is currently very little 8-bit software available, and that characters in the ranges 0 to 31 and 128 to 255 will only ever appear in the form shown above on the screen itself. Printers, on the other hand, will have their own, and different, ideas about what these symbols will look like.

Chapter 7
More About the Bank Manager....

Extensions to BASIC allowing access to the second 64K bank of RAM. This chapter applies only to the 6128.

Subjects covered:

- ★ Storing screen images
- ★ Pseudo-file operation

The memory map for BASIC 1.1 (shown in Chapter 6 part 14) indicates that 64K of the 128K RAM is unused. The BASIC and firmware themselves reside in a ROM which, together with the disc ROM, increase the normal memory provided from 64K to 112K (64K RAM, 48K ROM).

Each section of 16K is called a 'block', and any particular selection of four blocks (to make 64K in total) is called a 'bank'. The technique of selecting blocks is therefore called 'bank switching'.

The Z80 microprocessor can accomodate only 64K of memory at any one time, so the operating system contains instructions to switch the firmware ROM into play instead of Block 0 of RAM, and to switch either the BASIC ROM or the disc ROM into play instead of Block 3. This switching takes place automatically when the BASIC or firmware is required. Bank switching of RAM merely extends this concept to include the overlaying of RAM, rather than ROM. The switching is taken care of by an assembler program.

The program `BANKMAN.BAS` is provided on Side 1 of the system discs package. If the program is run after BASIC has started, it will install the standard bank management RSX code. Hence, the program is known as the 'BANK MANAGER'.

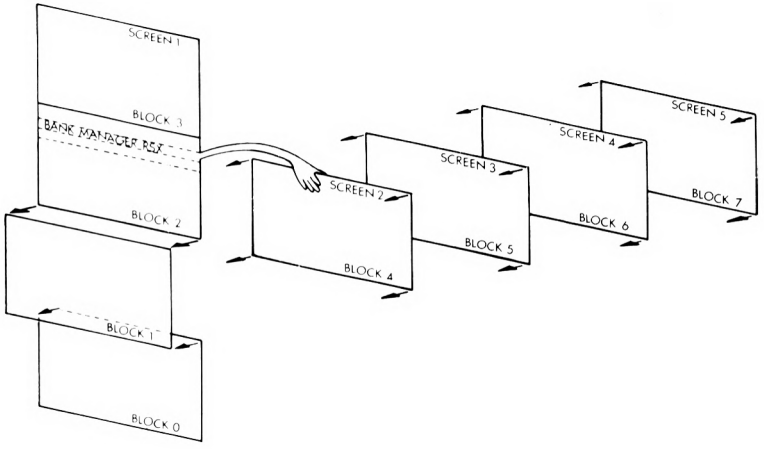
One use for the memory in the second 64K is as temporary storage space for picture screens. Such applications may include, for example, a 'Screen Designer' program which stores a number of different screens, or a video game which may have a number of different screens already prepared.

Another use for the second 64K is as an extension to the variable workspace, which can be regarded as either an extension to the string array space, or as a simple 'RAMdisc'.

Part 1: Storing screen images

Choose your screen....

The BANK MANAGER is able to switch out Block 1, and switch in one of the four blocks from the second 64K in its place. The diagram below illustrates this action. Note how each of the blocks from the second 64K occupies the same address space (&4000 to &7FFF). The contents of Block 1 (probably the middle of your BASIC program!) is preserved, and is restored when the BANK MANAGER has finished. There are three other bank selections possible (apart from the five shown below), but they are only useful to the implementation of CP/M Plus.



Hardware bank switching

The BANK MANAGER supports two commands for moving screen-fulls of information between one block and another. Blocks 4 to 7 are switched in and out automatically as required, and the memory map is left with Block 1 switched in.

The command:

!SCREENSWAP,[‹screen section›,]‹screen number›,‹screen number›

....exchanges the contents of two blocks, whereas:

!SCREENCOPY,[‹screen section›,]‹destination screen number›,
‹source screen number›

....copies the contents of one block into another block.

The optional ‹screen section› parameter causes the software to copy only 1/64th of the block (256 bytes out of 16K). ‹screen section› therefore takes values in the range 0 to 63. This mode of operation is useful if it is required to interleave any other processing with screen moving. A screen-swap can take around 150/300ths of a second (150 TIME-counts).

The ‹screen number›s required are 1 (the normal screen), and then either 2, 3, 4, or 5. Copy and swap operations where the source or destination is screen 1, will operate much faster. Be careful of the screen hardware-roll effect, as experienced when dealing with disc screen dumps. It should be arranged that all the screen images are constructed (and viewed) with screen 1 set to the same hardware position. The simplest (default) position is that set by a **MODE** command.

Try out the screen switching commands....

First, run the BANK MANAGER program from Side 1 of your system discs package by:

```
RUN "BANKMAN"
```

Then type:

```
MODE 1
```

Screen clears. Now type:

```
' This is the original screen
!SCREENCOPY,3,1 'Send original screen to memory 3

CLS
```

Screen clears again. Now type:

```
' This is the intermediate screen
!SCREENCOPY,2,1 ' Send intermediate screen to memory 2
!SCREENSWAP,2,3 ' Exchange memory 2 and memory 3
!SCREENCOPY,1,3 ' Restore intermediate screen from memory 3
!SCREENCOPY,1,2 ' Restore original screen from memory 2
```

Finally on this subject, the last part of Chapter 9 includes a comprehensive 'Screen Designer' program, which incorporates the screen switching facilities provided by the BANK MANAGER.

Part 2: Pseudo-file operation

Curses filed again....

When regarded as a RAMdisc, the whole of the second 64K is divided into a 'RAMfile' comprising a number of fixed length records. The record length can be between 0 and 255 bytes, although 2 bytes is recommended as the minimum. Once the ‹RAMrecord length› has been established, each record can be accessed by its ‹RAMrecord number›. It is perfectly acceptable to write the RAMfile using one record length, and read it back using another.

NOTE - The RAMfile must contain only data; there is NO facility for it to contain program instructions.

As is common with random access disc filing schemes, there is the concept of 'current record number'. This provides, in effect, a default record number, which is particularly useful when auto-stepping through the RAMfile.

The command:

!BANKOPEN,‹RAMrecord length›

....sets the fixed length of all records, initialises the current record number to zero, but DOES NOT clear the memory in any way.

The command:

!BANKWRITE,@‹return code›,‹string expression›[,‹RAMrecord number›]

....writes the ‹string expression› to the RAMfile.

The ‹RAMrecord number› specifies which record is written. If the parameter is omitted, then the current record number is used. The current record number is then set to point at the next record.

If the ‹string expression› does not completely fill the record, then old characters (which have not been overwritten) will remain at the end of the record. If the ‹string expression› is longer than the record, then the excess characters are discarded to avoid spillage into the next record.

‹return code› is an integer variable which returns the record number written to (if the operation was successful), or a negative error code if the write operation has failed in some way.

-1 End of File error. The requested record number's address exceeds 64K.

-3 Bank switching failure (should never happen).

Examples:

```
|BANKOPEN,1Ø
|BANKWRITE,@r%,"123 testing",Ø
|BANKWRITE,@r%,w$
```

The command:

|BANKREAD,@‹return code›,@‹string variable›[,‹RAMrecord number›]

....reads a record into the ‹string variable› from the RAMfile.

The ‹RAMrecord number› specifies which record is to be read. If the parameter is omitted, then the current record number is used. The current record number is then set to point at the next record.

If the record contents do not completely fill the ‹string variable› then old characters (which have not been overwritten) will remain at the end of the ‹string variable›. If the record contents are longer than the length of ‹string variable›, the excess characters are discarded, as it is not possible to increase the length of a string variable during an external command.

‹return code› is an integer variable which returns the record number read from (if the operation was successful), or a negative error code if the read operation has failed in some way.

-1 End of File error. The requested record number's address exceeds 64K.

-3 Bank switching failure (should never happen).

Example:

```
|BANKREAD,@r%,i$,Ø
```

Searching....

It is possible to search through the stored records looking for a particular entry.

The command:

```
!BANKFIND,@‹return code›,‹searched for string›
                [,‹starting record number›[,‹finishing record number›]]
```

....will scan all the RAMrecords. The ‹starting record number› specifies from which record the search starts. If the parameter is omitted, then the current record number is used.

The search will proceed, in steps of ‹RAMrecord length›, throughout the whole of the second 64K of memory until a match is found.

If a ‹finishing record number› is specified, then the search will terminate after that record has been tested (unless a match has been found before that point).

If the search is successful, then the current record number is set to point at the record where the match was found (otherwise it remains unaffected).

‹return code› is an integer variable which returns the record number where the match was found (if the operation was successful) or a negative error code if the search operation has failed in some way.

-1 End of File. The starting record number's address exceeds 64K or exceeds the finishing record number.

-2 No match found.

-3 Bank switching failure (should never happen).

The ‹searched for string› may contain wild-cards, signalled by nulls - `CHR$(0)`, and the comparison is made with reference to either the ‹RAMrecord length› or the length of the ‹searched for string›, whichever is the shorter.

Examples:

```
!BANKFIND,@r%,"123 test",0
!BANKFIND,@r%,f$,100,200
```

Beware the mismatch....

Obvious errors, such as the wrong number of parameters, are reported as a 'Bad Command' error. However, the mechanism of external commands does not detect errors of the 'Type mismatch' variety, and the user must ensure that the correct form of parameters are employed.

The program below uses the RAMdisc commands to set up and interrogate a database containing anagrams of 7-letter words. It searches for matches, and you may use wild cards.

For example, anagrams of the word FIGURES which match with ?RUGS?? (the two last ?? can be left off if you wish) are FRUGSIE, FRUGSEI, IRUGSFE, IRUGSEF, ERUGSFI, and ERUGSIF.

The database takes some time to create, but then 64K is a lot of memory to fill!

```
10 'ANAGRAMS by ROLAND PERRY
20 ' copyright (c) AMSOFT 1985
30 '
40 'Remember to RUN "BANKMAN" before running program
50 '***********************************************
60 '
70 MODE 2
80 DEFINT a-z
90 r%=0:|BANKOPEN,7
100 INPUT"What 7 letter word to scramble ";s$
110 IF LEN(s$)<>7 THEN 100
120 PRINT"Please wait..."
130 LOCATE 1,5:PRINT"Computing:"
140 FOR c1=1 TO 7
150 FOR c2=1 TO 7
160 IF c2=c1 THEN 370
170 FOR c3=1 TO 7
180 IF c3=c2 OR c3=c1 THEN 360
190 FOR c4=1 TO 7
200 IF c4=c3 OR c4=c2 OR c4=c1 THEN 350
210 FOR c5=1 TO 7
220 IF c5=c4 OR c5=c3 OR c5=c2 OR c5=c1 THEN 340
230 FOR c6=1 TO 7
240 IF c6=c5 OR c6=c4 OR c6=c3 OR c6=c2 OR c6=c1
    THEN 330
250 FOR c7=1 TO 7
260 IF c7=c6 OR c7=c5 OR c7=c4 OR c7=c3 OR c7=c2
    OR c7=c1 THEN 320
```

Continued on the next page

```
270 o$=MID$(s$,c1,1)+MID$(s$,c2,1)+MID$(s$,c3,1)
    +MID$(s$,c4,1)+MID$(s$,c5,1)+MID$(s$,c6,1)
    +MID$(s$,c7,1)
280 LOCATE 12,5:PRINT x;o$
290 !BANKWRITE,@r%,o$
300 IF r%<0 THEN STOP
310 x=x+1
320 NEXT c7
330 NEXT c6
340 NEXT c5
350 NEXT c4
360 NEXT c3
370 NEXT c2
380 NEXT c1
390 lastrec=r%
400 REM now look them up
410 r%=0:g$=SPACE$(7)
420 PRINT:INPUT"What match do you require:
    use ? as wild card: ",m$
430 m$=LEFT$(m$,7)
440 FOR x=1 TO LEN(m$)
450 IF MID$(m$,x,1)="?"THEN MID$(m$,x,1)=CHR$(0)
460 NEXT
470 !BANKFIND,@r%,m$,0,lastrec
480 IF r%<0 THEN GOTO 420
490 !BANKREAD,@r%,g$
500 PRINT g$,
510 !BANKFIND,@r%,m$,r%+1,lastrec
520 GOTO 480
```

Chapter 8
At your leisure....

This chapter takes a leisurely look at some background information to computing in general, and at the 464/6128 in particular. It is not vital that you read this chapter before operating the computer, but it may help you to understand a little of what's going on 'under the bonnet'.

Part 1: Generally speaking....

Zap the wotsit!

Even if the only reason that you bought your 464/6128 was to take advantage of the sophisticated computer games available, you may still probably be wondering about several aspects of the computer that come under the heading of 'hardware'.

The hardware is the equipment that you can pick up and carry around, i.e. the main computer keyboard, the monitor, the connecting leads etc. In fact, it's just about everything that isn't specifically the 'software' - programs, manuals, and disc or cassette based information.

Certain features of the way that the computer behaves, are produced by courtesy of the hardware - things like the coloured display on the TV set (or monitor). It's up to the software to make use of these hardware capabilities 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.

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 distinctly different areas that can be resolved on the display (i.e. the pixels), and the number of text characters that can be displayed on a single screen area.

You will find that your 464/6128 compares very favourably with any similarly priced machine in each of 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 464/6128 is one of the most widely compatible dialects of BASIC. 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 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. On the other hand, 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. Expansibility

Most computers pay attention to the need to 'add-on' additional items of hardware: printers, joysticks, extra disc 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, because a machine that incorporates a properly supported parallel printer (Centronics compatible) and a games joystick port may actually be cheaper in real terms.

The 464/6128 computer features a built-in Centronics printer port, a port for an additional disc drive (6128 only), facilities for a games controller or up to two joysticks, a stereo sound output - and a comprehensive expansion bus that can be used to attach a serial interface, a MODEM, a speech synthesiser, etc.

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 464/6128 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, 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 up your own mind 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 464/6128 is apparently unable to perform tasks seen on any TV set. 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 be 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 twenty times greater than the converted equivalent of a home computer 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 price 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 fair way from cheap computers that can reproduce flowing motions and lifelike pictures in the same way that even a moderate animated cartoon can produce.

That keyboard looks familiar....

Why can't you simply walk up to the computer and type a page of simple text into the machine?

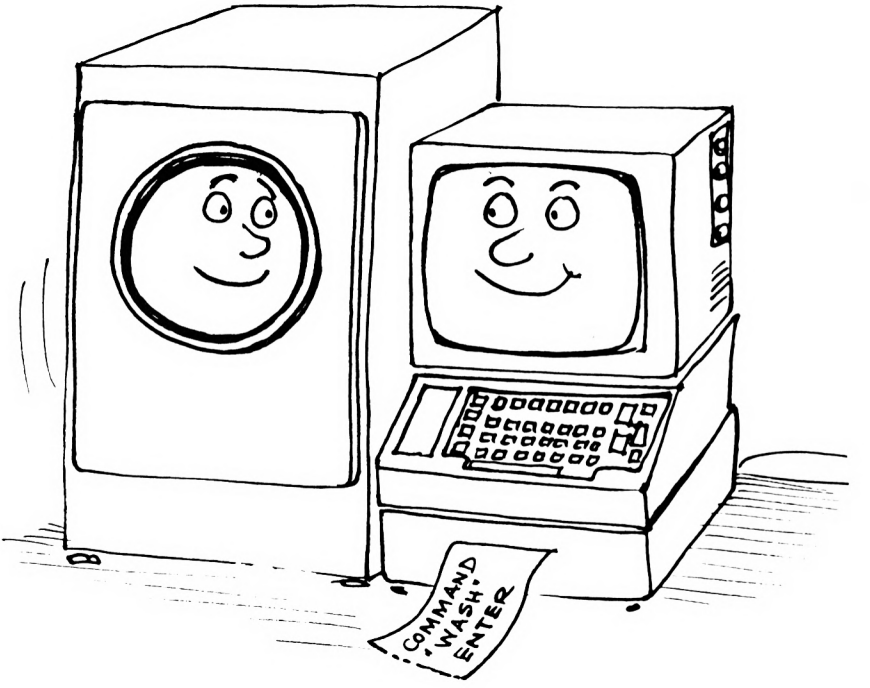
Don't be misled by the fact that the computer looks like a typewriter with an electronic display. The screen is not a piece of electronic paper - it's a 'command console' - jargon which means that it simply provides you with the means of communicating with the programming language (and the programs) 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 **[RETURN]** 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 **[RETURN]** and carry on typing as if the system WERE operating as an electronic piece of paper in an electronic typewriter. But to do this, you must have first loaded a word processor program into the machine's memory.

The computer 'seems' to combine several items of equipment that have become familiar around the home and office such as the TV-like screen and the keyboard. 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!



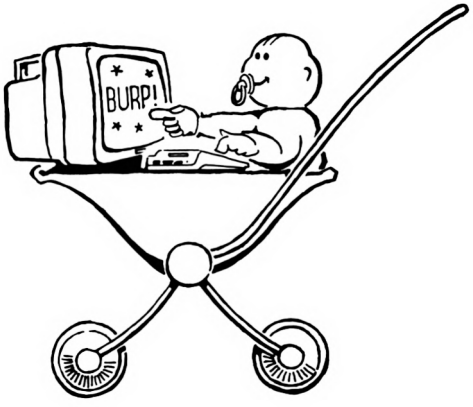
Who's afraid of the jargon?

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 terminology - 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 in the case of 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 minimise 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, despite 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 the micro revolution is settling down.

Computing is rapidly being 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 ten 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 no longer has any particular significance as to the degree of the sophistication of the language, 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.

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 5000-8000 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 (i.e. plain language) will result in the cold and clinical message:

```
Syntax error
```

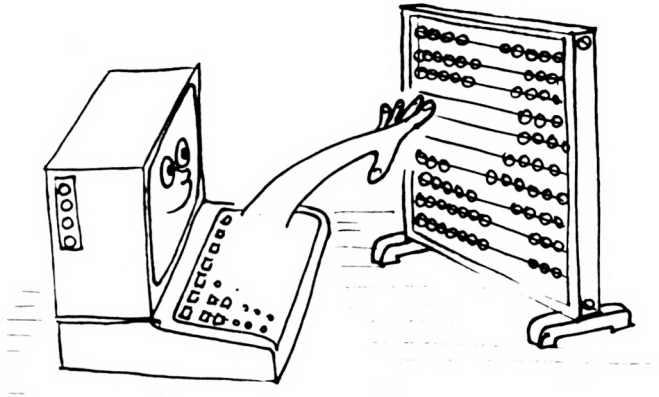
This is not as restrictive as it first appears, since the language of BASIC (the syntax) is primarily designed to manipulate numbers - the numeric data. The words are essentially an extension of the familiar mathematical operators +/− etc., and the most important concept for newcomers to grasp is the fact that a computer can only work with numeric data. Information that is supplied to the CPU (Central Processor Unit) 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 will not be a single letter or word to be found anywhere in the system. Every piece of information is first converted into a number which the computer can locate and 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 permit the computer to compare data and select for certain attributes - in other words, check to see if one number is greater than 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.



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 repetitive tasks very quickly and with absolute precision. Thus BASIC interprets the instructions as 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 linear progression. In other words, the transition is made by following the path of a line between the two states - in an ideal digital environment 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 delay - 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, and the stages of transition via the various shades of grey have NO significance whatsoever. Conversely, the linear or 'analogue' progression IS via the 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 'oversized' data to be squeezed down to fit the space available , or 'truncated', leading to rounding errors. e.g.999,999,999 becomes 1,000,000,000.

In a world where the only two numbers available are 0 or 1, how then do you 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 - i.e. 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 'Binary digIT'.

The relationship between bits and decimal notation is simple to understand:

It's 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:

e.g. 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 of 2 is present; 1=yes, 0=no.

$2^0 = 1$
$2^1 = 2 = 2 = 2(2^0)$
$2^2 = 4 = 2x2 = 2(2^1)$
$2^3 = 8 = 2x2x2 = 2(2^2)$
$2^4 = 16 = 2x2x2x2 = 2(2^3)$

....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 maximum number that 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 achieve an acceptable means of handling the memory, two bytes are used to provide a method of addressing memory which is in the form of array, with a horizontal and vertical address by which the elements of that array can be located:

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5				1		1				
6										
7										
8										
9										

The array can locate up to (10x10) items of information using address numbers that lie in the range 0 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 different elements.

The next level of shorthand for binary is the kilobyte (kByte or simply 'K') which is 1024 bytes. 1024 is the nearest binary multiple to the more familiar decimal use of the term 'kilo' (1000) - 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 being the bedrock of computer operation in the foreseeable 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 widely used in microcomputing is called HEX (an abbreviation of hexadecimal).

Here the number is based on 16 (0 to 15), 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 absolute elegance of binary and hex begin to emerge.

Reconsidering the table that introduced binary notation:

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

An 8-bit number 11010110 (&D6 hex) can be subdivided, and then considered as two 4-bit numbers (known as nibbles). Throughout this manual, a hex based number is introduced by the '&' symbol e.g. &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 programmer 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 number (also known as a Denary number) such as '89' - i.e. (8x10)+(9). It just happens that multiplying by ten is a great deal simpler, unless you've had a lot of practice 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, but 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 that your computer deals with them.

Part 2: More about the 464/6128 specifically....

This section gently expands upon some of the machine-specific aspects of the 464/6128. Background information to these matters will be found both in the Foundation course, and in the chapter entitled 'Complete list of AMSTRAD 464/6128 BASIC keywords'.

Subjects covered in this section:

- ★ Character set
- ★ ASCII
- ★ Variables
- ★ Logic
- ★ User defined characters
- ★ Print formatting
- ★ Windows
- ★ Interrupts
- ★ Data
- ★ Sound
- ★ Graphics
- ★ Graphics Using the Extra Memory

A bit of character....

As you type at your 464/6128 keyboard, you should not take for granted the fact that recognisable letters and numbers etc, appear on the screen. After all, we've already discussed the fact that your computer is not a typewriter. What's actually happening is a result of you pressing a combination of electrical switches. The electrical signals produced when you press these switches are translated by the circuitry inside the equipment to produce a pattern of dots on the screen. We recognise that pattern of dots as a letter, number, or other character from the 464/6128's 'character set'.

Some of the characters that you will see are not directly accessible by pressing the keys on the keyboard, but are only available for display using the PRINT CHR$(‹number›) statement. This is because each element stored in the computer is stored in the unit of data known as the 'byte' - and as just discussed in part 1 of this chapter, a byte has 256 different possible combinations of value. 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 464/6128 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. The chapter entitled 'For your reference....' lists the ASCII display range, together with the additional characters available on the 464/6128, and the corresponding numeric codes.

How we get there....

You are by now probably quite familiar with the program:

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

....which makes the computer display the character set on the screen. Let's now examine the essence of this small program:

The first point to notice is that the computer has not been instructed to PRINT "abcdefghijklmn..... etc"; instead it has been asked to PRINT CHR$(n). n just happens to be a convenient shorthand note for a 'variable'. A variable is an item of computer information that 'varies' according to the instructions given in the program. (The choice of the letter n for the variable is arbitrary - it can be any letter(s) as long as it's not a keyword.)

How can you tell what is a variable?....

A number like 5 is fixed, it occurs between the numbers 4 and 6 - thus it is not a variable. The 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, i.e. "n", and the computer would have responded with the message Syntax error - because it does not understand the command sequence FOR "n"=32 TO 255.

Simply by using n without quotation marks, 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, finishing at 255.

Having declared this variable, we should then instruct the computer what it should do with it - line 20 does just that:

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

This specifies that whatever the current numeric value of n, the computer should look into its memory to see which character number corresponds to that value, and print the character on the screen.

The semicolon at the end of the line instructs the computer to prevent a carriage return and line feed. (Otherwise each new character will be printed in the first column of a new line.)

Line 30 tells the computer that after it has performed the task with the first value of n in the sequence (which is 32), it should return to the line where the FOR is located, and do the same again with the NEXT value that it assigns to the variable n. This process is known as 'looping', and is one of the most vital and fundamental aspects of computer programming and operation. It saves typing long repetitious sequences manually, and you will quickly come to use it in your own programming.

When this FOR NEXT 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 ends, and returns to direct mode, displaying the Ready prompt. This tells you that the computer is ready to accept further instructions - or you can enter 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 turn the power off.

This program neatly illustrates a fundamental point about computing - i.e. 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 the 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 part 3 of the chapter entitled 'For your reference....'.)

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

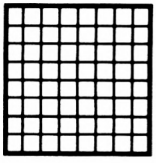
For example, the ASCII code for the letter A is 97. The computer doesn't understand 97 either (awkward blighter, eh?), and this number has to be translated from the human decimal code into a code that computer can relate to - it's generally referred to as 'machine code', and the principles underlying this aspect of the machine are discussed earlier in this chapter.

At first, the translation from the decimal number notation we are used to in everyday 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.

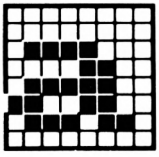
A degree of mental dexterity must be acquired to understand hex notation, but once you do, many things about computing will fall into place and the elegant structure of the numbering system will become apparent.

If you are unsure about the binary and hexadecimal numbering systems, we suggest that you thoroughly read part 1 of this chapter (if you have not already done so).

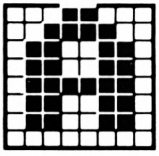
Once the computer has translated the pressing of the **A** key into the type of number it understands, it looks into that part of the memory indicated, and the result is another series of numbers that define the character. Hence the character that you see displayed on your screen, is built up from a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



A BLANK CHARACTER MATRIX (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 matrix or 'cell' on the 464/6128 display, and if you don't find a character you want out of the set of 255 that are provided, then you can re-define your own characters using the keyword **SYMBOL** described later in this section.

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

Logic....

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 (bit-wise), 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 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 `ERR`or 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 `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 their binary representations:

```
      1010
1111101000
```

The **AND** operation checks each corresponding bit at a time, and where the bit in the top **AND** the bottom row is the **1**, the answer is **1**:

```
0000001000
```

...which is our result of **8**. The logical operator **AND** is used to detect when two conditions are present simultaneously. Here's a self explanatory application:

```
10 INPUT "The number of the day";day
20 INPUT "The number of the month";month
30 IF day=25 AND month=12 THEN 50
40 CLS: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
```

Bit-wise:

```
      1010
1111101000
```

Resulting in the answer:

```
1111101010
```

And in a program example:

```
10 CLS
20 INPUT "The number of the month";month
30 IF month=12 OR month=1 OR month=2 THEN 50
40 GOTO 10
50 PRINT "It must be winter!"
```

The NOT operator inverts each bit in the argument (0 becomes 1, and vice versa):

```
10 CLS
20 INPUT "The number of the month";month
30 IF NOT(month=6 OR month=7 OR month=8) THEN 50
40 GOTO 10
50 PRINT "It can't be summer!"
```

Another major feature to consider is the fact that you can add together any number of logical conditions (up to the maximum line length) to distill the facts yet further.

```
10 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 THEN 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 = 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 lines 60 and 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, 2 for the month will produce the answer in line 50, and the actual values returned by the logical expressions in lines 60 and 70.

Finally, XOR (eXclusive OR) produces a true result as long as both arguments are different.

The following summarises all these features in what's known as a 'truth table'. It's a convenient way of illustrating what happens in a bit-wise logical operation.

Argument A	1010
Argument B	0110
AND result	0010
OR result	1110
XOR result	1100

User defined characters

One of the first applications of binary numbers that you are likely to come across will be when designing characters for use with the SYMBOL command. If the character is drawn on an 8 by 8 grid then each of the eight rows can be converted to a binary number by putting a 1 for each pixel that is to be inked and a zero for each that should be invisible, i.e. set to the paper colour. These eight numbers are then passed to the SYMBOL command. For example, to define a house character:

```
   *     = 00001000 = &08 =                8        = 8
  ****   = 00111100 = &3C =          32+16+8+4      = 60
 *    *  = 01000010 = &42 =      64              +2 = 66
* *  * * = 10100101 = &A5 = 128   +32      +4   +1 = 165
*       * = 10000001 = &81 = 128                +1 = 129
* ** * * = 10110101 = &B5 = 128   +32+16  +4   +1 = 181
* **   * = 10110001 = &B1 = 128   +32+16       +1 = 177
******** = 11111111 = &FF = 128+64+32+16+8+4+2+1 = 255
```

....the command is:

```
SYMBOL 240,8,60,66,165,129,181,177,255
```

....or....

```
SYMBOL 240,&08,&3C,&42,&A5,&81,&B5,&B1,&FF
```

....or....

```
SYMBOL 240,&X00001000, &X00111100, &X01000010, &X10100101,
          &X10000001, &X10110101, &X10110001, &X11111111
```

To print the user defined character, you would type:

```
PRINT CHR$(240)
```

Finally, to group blocks of characters together, you may for example specify:

```
semi$=CHR$(240)+CHR$(240)
PRINT semi$
```

.... or ....

```
terrace$=STRING$(15,240)
PRINT terrace$
```

Printing press....

PRINT is one of the first ever commands that you use when you start to learn computing. It's one of those BASIC commands that does what it says really.... or does it? In fact there's a lot more to PRINT than at first it seems, for instance WHERE should it print? and HOW should it print?....

Print formatting

The PRINT command has several ways in which it can be used. The simplest is to follow it by an item to be printed. This item can be a number, a string or a variable name.

```
PRINT 3
 3

PRINT "hello"
hello

a=5
PRINT a
 5

a$="test"
PRINT a$
test
```

Several items may be placed in one PRINT statement with each being separated by a separator, or TAB or SPC. The possible separators are either a semicolon or a comma. The semicolon causes direct continuation of printing, while a comma forces printing to continue in the next zone. The initial zone width is 13, but may be changed using the ZONE command.

```
PRINT 3;-4;5
 3 -4  5

PRINT "hello ";"there"
hello there

PRINT "hello","there"
hello         there

PRINT 3,-4,5
 3            -4             5

ZONE 4
PRINT 3,-4,5
 3  -4   5
```

A point to note here is the fact that positive numbers are printed with a leading space, while negative numbers have a leading minus sign. All numbers have a trailing space. Strings are printed exactly how they appear between the quotes.

The function SPC takes a numeric expression as a parameter, and will print as many spaces as are specified by the expression. If the value is negative then zero is assumed, if it is greater than the current stream (window) width, then the stream width is assumed.

```
PRINT SPC(5)"hi"
    hi

x=3
PRINT SPC(x*3)"hi"
        hi
```

TAB is very similar except that it will print as many spaces as are needed so that the item to be printed will appear in the specified column.

The stream in which all printed output will appear is window 0 unless a stream director (#) is included before the list of items to be printed. Other streams may be used to output to the other windows. Streams 8 and 9 are special cases - anything printed on stream 8 will appear on the printer (if connected). Stream 9 directs output to a disc (or cassette) file. Note however, that the WRITE command should be used instead of PRINT for this purpose.

```
PRINT "hello"
hello                          - window 0

PRINT #0,"hello"
hello                          - also window 0

PRINT #4,"hello"
hello                          - window 4
                                 (At the top of the screen)

PRINT #8,"hello"
hello                          - on the printer
                                 (If connected)
```

TAB and SPC are fine for simple print formats, but to specify a more detailed format, the PRINT USING command, together with a suitable format template can be used. A format template consists of a string expression containing special characters, each of which will specify a particular type of format. These characters, called 'Format field specifiers', are detailed in the description of the keyword PRINT USING, earlier in this manual. Some of the following examples however, may make their use a little clearer.

Firstly, here are the formats available for the printing of strings:

```
PRINT USING "\     \";"test string"
test s
```

"!" can be used to print the first character of a string.

```
PRINT USING "!";"test string"
t
```

....But probably the most useful string format is "&". This can be used to override the string wrapping feature of BASIC if required. By default, BASIC will start the printing of any string on a new line if it is too long to fit onto the current line. PRINT USING "&"; can be used to override this.

(Use BORDER Ø, so that you can see the edges of the paper.)

```
MODE 1:LOCATE 39,1:PRINT "too long"

                                                    - line 1
too long                                            - line 2

MODE 1:LOCATE 39,1:PRINT USING "&";"too long"

                                              to    - line 1
o long                                              - line 2
```

A large number of templates are available for the printing of numbers. Probably the simplest is PRINT USING "#####", one digit is printed for each "#" that appears in the template.

```
PRINT USING "######";123
   123
```

The position of the decimal point may be included by the use of "."

```
PRINT USING "####.#####";12.45
  12.45000
```

The digits before the decimal point may be grouped into threes, separated by commas if "," is included in the template before the decimal point.

```
PRINT USING "########,.####";123456.78
  123,456.7800
```

Floating dollar and pound signs may be included in the format - i.e. a currency sign that will always be printed directly before the first digit of the number, even if it does not fill the complete format. This is achieved by the use of "**$$**" and "**££**" in the template.

```
PRINT USING "$$###";7
  $7

PRINT USING "$$###";351
$351

PRINT USING "££####,.##";1234.567
 £1,234.57
```

Note the rounding of the result.

The space before the result may be padded with floating asterisks by the use of "******" in the template.

```
PRINT USING "**#####.#";12.22
****12.2
```

This may be combined with the currency symbols, (and then only one currency symbol is used) - i.e. "****$**....etc" or "****£**....etc".

A "+" at the start of the template specifies to always print the sign of the result before the first digit. A "+" at the end of the template prints a trailing sign.

A "-" can only be placed at the end of the format, and specifies that a trailing minus sign be printed if the number is negative.

```
PRINT USING "+##";12
+12

PRINT USING "+##";-12
-12

PRINT USING "##+";12
12+

PRINT USING "##-";-12
12-

PRINT USING "##-";12
12
```

A "↑↑↑↑" format template can be used to print a number in exponential format.

```
PRINT USING "###.##↑↑↑↑";123.45
 12.35E+Ø1
```

When using print formats for numbers, note that if a number is too long for the specified template, then a % symbol is printed before the result, to indicate that this has happened, and the result is NOT shortened to fit the specified template.

```
PRINT USING "####";123456
%123456
```

Want your windows done?....

The BASIC of the 464/6128 provides a comprehensive method for setting up a maximum of eight text windows. Any of the text screen driving commands may then be directed to any one of these windows.

The command that is used to set up a window is, simply enough: **WINDOW**. This is followed by 5 values. The first is optional and is used to specify which window is to be defined - if omitted, then window zero is assumed, all the normal BASIC prompts and messages (for example, '**Ready**') are produced in window zero. The hash symbol (#) precedes this number to identify it as being a stream director. The next four numbers specify the left, right, top and bottom limits of the window. These values are column and row numbers, so they must lie in the range 1 to 80 for left/right and 1 to 25 for top/bottom.

The following example will define **WINDOW** (stream) number 4 to start in column 7 (left) and go on to column 31 (right), and to start at row 6 (top) and go down to line 18 (bottom). Reset the computer then type:

```
WINDOW #4,7,31,6,18
```

Nothing will appear to have happened after this command, however, try typing the following:

```
INK 3,9
PAPER #4,3
CLS #4
```

This will cause a large green rectangle to appear on the screen, and this is window number 4. The above also shows that **PAPER** and **CLS** may be used with any one of the eight windows by the inclusion of a stream director; its omission causes the command to operate on window 0 - the default window.

Each of the following commands may include a stream director to identify the WINDOW in which the command is to be carried out.

```
CLS
COPYCHR$
INPUT
LINE INPUT
LIST
LOCATE
PAPER
PEN
POS
PRINT
TAG
TAGOFF
VPOS
WINDOW
WRITE
```

The new green window which you have put on the screen, will have obscured some of the previous text (written on window number 0).

Text may be directed to any window by including a stream director in a PRINT statement:

```
PRINT #4,"hello there"
```

These words will appear at the top of the green rectangle, rather than on the following line as would have happened if....

```
PRINT "hello there"
```

....had been used. While typing in the earlier command, you will have noticed that part of the green window was overwritten by the text.

If you want all the normal BASIC messages to appear in window 4, then it can be swapped with the default window (Ø) by use of the WINDOW SWAP command:

```
WINDOW SWAP Ø,4
```

The 'Ready' that follows this command will be printed in the green window. The cursor will be positioned directly beneath it. Now try typing the following:

```
PRINT #4,"hello there"
```

....and the words 'hello there' will appear directly beneath the **WINDOW SWAP** command in the old window **Ø**, which is now window **4**. It may also be apparent from this, that the current print position in each window is stored, so that even after a **WINDOW SWAP**, text is printed part way down the new stream **4** rather than starting at the top. Try the following:

```
LOCATE #4,20,1
PRINT "this is window Ø"
PRINT #4,"this is window 4"
```

The '**window Ø**' message will appear on the line after the **PRINT**, while the '**window 4**' message will appear at the middle of the top line of the whole screen.

Before a **WINDOW** command has been issued, all eight windows cover the entire screen. This is also true after a **MODE** command has been issued - so, if after using windows you find that the cursor ends up in a very small window, just type in **MODE 1**, as shown:

```
MODE 1
WINDOW 20,21,7,18
MO
DE
 1
```

Don't worry about the word '**MODE**' being split up - it will still work, and don't forget to leave a space between **MODE** and **1**.

Now that you know a little about the way in which windows operate - try typing in the following short program :

```
10 MODE Ø
20 FOR n=Ø TO 7
30 WINDOW #n,n+1,n+6,n+1,n+6
40 PAPER #n,n+4
50 CLS #n
60 FOR c=1 TO 200:NEXT c
70 NEXT n
```

This sets up 8 overlapping windows and clears each to a different paper colour. When the program has finished running and '**Ready**' appears, try pressing **[RETURN]** a few times to see how the scrolling of window 0 affects the coloured blocks on the screen. However, although these coloured blocks may be scrolling, the locations of the other windows do not actually move. Try the following:

```
CLS #4
```

....and you will see that the 4th window is still in the same position - the new coloured block having obscured those beneath it as one would expect. As a matter of interest, observe the differences when you type:

```
LIST
LIST #4
LIST #3
```

A further feature of the **WINDOW** command, demonstrated by the final program in this section, is that it does not matter if you specify the left and right window dimensions in reverse order. This means that if the value of the first parameter is greater than the second, BASIC will automatically sort the dimensions into the correct order. This also applies to the top and bottom window dimensions.

```
10 MODE 0
20 a=1+RND*19:b=1+RND*19
30 c=1+RND*24:d=1+RND*24
40 e=RND*15
50 WINDOW a,b,c,d
60 PAPER e:CLS
70 GOTO 20
```

If I may interrupt....

If you haven't already noticed, a major innovation in the software of the AMSTRAD range of computers is their 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 'multi-tasking', and it is implemented by the application of the 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 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; i.e. **AFTER** the time that you have preset in the command, the program will divert to the designated sub-routine and perform the task defined therein.

The 464/6128 maintains a real time clock. The AFTER command allows a BASIC program to arrange for sub-routines to be called at some time in the future. Four delay timers are available, each of which may have a sub-routine associated with it.

When the time specified has passed, the sub-routine is called automatically, just as if a GOSUB had been issued at the current position in the program. When the sub-routine finishes, using a normal RETURN command, the main program continues running where it was interrupted.

The EVERY command allows a BASIC program to arrange for sub-routines to be repeatedly called at regular intervals. Once again, four delay timers are available, and each may have a sub-routine associated with it.

The timers have different interrupt priorities. Timer 3 has the highest priority and timer 0 the lowest (see the chapter entitled 'For your reference....').

```
10 MODE 1:n=14:x=RND*400
20 AFTER x,3 GOSUB 80
30 EVERY 25,2 GOSUB 160
40 EVERY 10,1 GOSUB 170
50 PRINT "test your reflexes"
60 PRINT "press the space bar.";
70 IF flag=1 THEN END ELSE 70
80 z=REMAIN(2)
90 IF INKEY(47)=-1 THEN 110
100 SOUND 1,900:PRINT "cheat!":GOTO 150
110 SOUND 129,20:PRINT "NOW":t=TIME
120 IF INKEY(47)=-1 THEN 120
130 PRINT "you took";
140 PRINT (TIME-t)/300;"seconds"
150 CLEAR INPUT:flag=1:RETURN
160 SOUND 1,0,50:PRINT ".";:RETURN
170 n=n+1:IF n>26 THEN n=14
180 INK 1,n:RETURN
```

AFTER and EVERY commands may be issued at any time, resetting the sub-routine and time associated with the given delay timer. The delay timers are shared by the AFTER and EVERY commands, so an AFTER overrides any previous EVERY for the given timer, and vice versa.

The DI and EI commands disable and enable timer interrupts whilst the commands between them are executed. This has the effect of delaying a higher priority interrupt from ever occurring during the processing of a lower priority interrupt. The REMAIN function disables, and returns the remaining count for one of the four delay timers.

Using data....

In a program that always requires the same set of information to be input at the start, it would make more sense if there were some way of entering all the values without having to ask the user to type them in every time. This facility is provided by the **READ** and **DATA** commands. The word **READ** is very similar to **INPUT** in that it can be used to assign values to variables. It differs, however, in the fact that values are read from **DATA** statements, rather than prompting for input from the keyboard. The following two examples show this:

```
10 INPUT "enter 3 numbers separated by commas";a,b,c
20 PRINT "the numbers are";a;"and";b;"and";c
run
```

```
10 READ a,b,c
20 PRINT "the numbers are";a;"and";b;"and";c
30 DATA 12,14,21
run
```

In the same way that different items in an **INPUT** statement are separated by commas, so it is with items in a **DATA** statement.

In addition to numeric values, constant strings may also be held in **DATA** statements:

```
10 DIM a$(8)
20 FOR i=0 TO 8
30 READ a$(i)
40 NEXT
50 FOR i=0 TO 8
60 PRINT a$(i);" ";
70 NEXT
80 DATA The,quick,brown,fox,jumps,over,the,lazy,dog
run
```

You may notice that although the **DATA** contains strings, the strings are not enclosed by double quotes "". The use of double quotes in **DATA** statements to delimit (separate) strings is optional, just as they are when typing a string in answer to an **INPUT** statement. One occasion that double quotes are useful however, is when the string **DATA** itself contains commas. If strings are not delimited by double quotes under these circumstances, the **READ** statement will use the commas to delimit the strings in the **DATA** statement.

```
10 READ a$
20 WHILE a$<>"*"
30 PRINT a$
40 READ a$
50 WEND
60 DATA The old, desolate, battered house creaked in
   the wind
70 DATA "The tall, slim, dark man coughed loudly."
80 DATA *
run
```

The string in line 60 contains commas, so each part will be READ and printed separately. The string in line 70 however, is delimited by double quotes and will be printed as a whole, as intended.

The above example illustrates the fact that data can be spread over a number of lines. READ will work down the lines in number order (60, 70, 80, etc.). Another fact that may not be obvious is that DATA statements can be placed anywhere within a program; before or after the READ statement that picks up the information.

If a program contains more than one READ statement, then the second READ will continue from the point at which the first READ stops:

```
10 DATA 123, 456, 789, 321, 654, 2343
20 FOR i=1 to 5
30 READ num
40 total=total+num
50 NEXT
60 READ total2
70 IF total=total2 THEN PRINT "the data is ok"
   ELSE PRINT "there is an error in the data"
run
```

Try editing line 10 so that one of the first 5 numbers is wrong, then run the program again. This technique of adding an extra value to the end of DATA statements which is the sum of all the other values, is a good method of detecting errors in DATA, especially if there are a large number of DATA lines - this is known as a 'checksum'.

If a program requires mixed data (strings and numbers), it is permissible to combine string and numeric items in READ and DATA statements, as long as the items are read correctly. For instance, if the DATA contained sequences of two numbers followed by a string - then it would only make sense to use a READ that was followed by two numeric variables, then a string variable:

```
10 DIM a(5),b(5),s$(5)
20 FOR i=1 TO 5
30 READ a(i),b(i),s$(i)
40 NEXT
50 DATA 1,7,fred,3,9,jim,2,2,eric,4,6,peter,9,1,
   alfonzo
60 FOR i=1 TO 5
70 PRINT s$(i),":";a(i)*b(i)
80 NEXT
```

Alternatively, you may wish to separate the different types of data:

```
10 DIM a(5),b(5),s$(5)
20 FOR i=1 TO 5
30 READ a(i),b(i)
40 NEXT
50 FOR i=1 TO 5
60 READ s$(i)
70 NEXT
80 DATA 1,7,3,9,2,2,4,6,9,1
90 DATA fred,jim,eric,peter,alfonzo
100 FOR i=1 TO 5
110 PRINT s$(i),":";a(i)*b(i)
120 NEXT
```

If the FOR loop in line 20 is now changed to:

```
20 FOR i=1 TO 4
```

....then the first two attempts to read strings in line 60 will produce '9' then '1'. These values are of course valid strings, but the result is not exactly what was planned! One method by which the program could be forced to work properly, would be to include the following commands:

```
15 RESTORE 80
45 RESTORE 90
```

The RESTORE statement will move the DATA-reading 'pointer' to the line specified, and can therefore be used in a conditional statement to pick a certain block of data to be read depending upon some criterion. For instance, in a multi-level game which has a number of different screens, the DATA for each screen may be picked according to some variable - for example 'level'. The following is just an example section of such a program:

To play on more than one channel, add up the numbers for the desired channels. For example to play on A and C use 1+4 = 5.

```
SOUND 5,284
```

You may be wondering why channel C is given the number 4 and not 3 as you might expect. This is because each of these numbers is a power of two (1=2 ↑ 0, 2=2 ↑ 1, 4=2 ↑ 2) so that they combine to form a binary number. If you think about a three digit binary number, then each of the three digits can be either 0 or 1, and this is used in the channel number to indicate whether the corresponding channel should be on or off. From the above example:

5 in decimal is equivalent to `1*4 + Ø*2 + 1*1` or 101 in binary. It follows then, that if each column of this binary number is labelled C, B, and A, this gives:

```
C  B  A
1  0  1
```

In other words, channel C is ON, B is OFF, and A is ON. If the note was to be played on channels A and B, then this would become:

```
C  B  A
0  1  1
```

And the binary number 011 is the same as `Ø*4 + 1*2 + 1*1 = 3`. So the `SOUND` command would be:

```
SOUND 3,142
```

This is, of course, the same value that would be found if you just added up the numbers for the channels to play on (remember A=1, B=2, C=4). So to play on A and B, the channel number is given by 1+2 = 3.

If you didn't understand that - don't worry. As long as you can see that a combination of channels can be chosen by adding up the numbers for each of the channels to be used, then that's all you really need to know.

Unfortunately, there are yet more values that can be used in the channel number. The numbers 8, 16 and 32 are used to specify that the sound should 'rendezvous' with another channel (A, B, C respectively). You're probably wondering what is meant by the term *rendezvous*. Well, up to now the sounds that we have produced have gone straight to the specified channels. Try this:

```
SOUND 1,142,2ØØØ
SOUND 1,9Ø,2ØØ
```

Unless you are a very slow typist, you will have noticed that you were able to type in the second of these commands before the first had finished. This is because the sound system is able to hold up to 5 sound commands for each channel in a 'queue'. If we wanted a sound to play on channel A, and then two sounds to play simultaneously on A and B, we would need some way of letting the computer know that the sound on B should not start until the second note on A was ready to play - i.e. making one channel wait for another. This is known as a *rendezvous*, and there are two ways in which this can be achieved :

```
SOUND 1,200,1000
SOUND 3,90,200
```

In this, the second note is directed to A and B, so it cannot start until the note on A finishes. The limitation with using this method (to make a note combination wait until all channels that it should play on are free) is that the same sound will be directed to each of the channels (in this case `,90,200` went to both A and B). The alternative method is to use the following:

```
SOUND 1,200,2000
SOUND 1+16,90,200
SOUND 2+8,140,400
```

Here, the second note on A is made to rendezvous with the sound on B (and the sound on B is made to rendezvous with the note on A). The advantage here is clear - although the second note on A was different to the note on B, the two were still linked so that neither could start until both channels were free - this is a *rendezvous*. Once again the values are bit-significant:

$8 = 2 \uparrow 3, 16 = 2 \uparrow 4,$ and $32 = 2 \uparrow 5$

....so now the channel number can be seen as a binary number where the columns are headed:

Rendezvous C Add 32	Rendezvous B Add 16	Rendezvous A Add 8	Play on C Add 4	Play on B Add 2	Play on A Add 1
0	0	1	1	0	0

So for a note to play on C and rendezvous with A, you would use:

This is the binary number 1100, which is equal to 8 + 4 = 12

Hence, a channel number of 12 would tell the computer to play a note on channel C, and wait for a note that has been marked to rendezvous with it on channel A.

If 64 (2 ↑ 6) is added to the channel number, then this indicates that the note should be held. In effect, this means that the note will not play until the command RELEASE is used.

And finally, if 128 (2 ↑ 7) is added to the number, then the queue of the channel specified will be cleared (or flushed). Therefore, if you start a sound that is going to continue for too long on a particular channel, then a quick way to stop it is to flush the channel:

```
SOUND 1,248,30000          (this would play for 5 minutes)
SOUND 1+128,0              (this will stop it short)
```

In direct command mode, a quicker way of stopping any long sounds is to press the **[DEL]** key at the start of a line; the short warning beep flushes all sound channels.

Now that we can hopefully send a sound to any of the three channels that we choose, (with rendezvous if necessary), it would be nice to be able to produce a little more than the rather unmelodious 'beep' that the simple SOUND command produces. The way to do this is to play the sound with an envelope - a pattern that defines how the note gets louder and quieter during the short time it is playing. A note produced by an instrument has an initial attack, where the volume rises very sharply. The volume of the note then falls away to a lower level which is sustained for a time, after which it fades away to zero. It is possible to give an envelope of this nature to the notes produced by the SOUND command. The associated ENV command is used to do this. First let's look at a simple example:

```
ENV 1,5,3,4,5,-3,8
SOUND 1,142,0,0,1
```

The ENV must come before the SOUND command for which it is used. To use this envelope in a SOUND command, its number is included as the fifth part of the SOUND command - in this case, the envelope is number 1. The first number in an ENV command is the number of the envelope that it defines. The ENV instruction contains information about how long the note will last and how loud it will get, so the duration and volume parts of the SOUND command are set to zero. The envelope defined above causes the sound to be increased in 5 steps, each step increasing the volume by 3, and each step being 4 hundredths of a second long. The volume is then to be reduced in 5 steps, each step decreasing the volume by -3, each step being 8 hundredths of a second long. In other words, the first number identifies which envelope is being defined, and this is then followed by two groups of three numbers, and in each of these groups, the first number indicates by how many steps the volume is to go up or down. The second number indicates by how much the volume is to go up or down at each of these steps, and the third number determines by how long each step of volume is to be held for.

The total length of time that each section will take is equal to the first value (number of steps) multiplied by the third value (pause time). The total increase or decrease in volume is equal to the number of steps multiplied by the step size. The overall length of an envelope with more than one section, is equal to the sum of the lengths of each section.

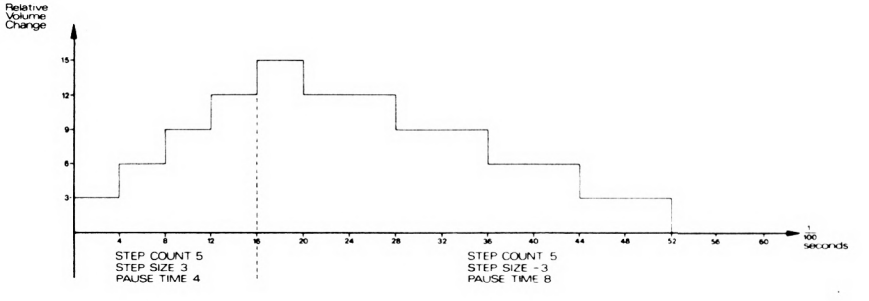
Of course, the starting volume needn't always be **Ø** (as set by the **SOUND** command). The above example produced a note that went up, then back down again. The following will go down, then back up:

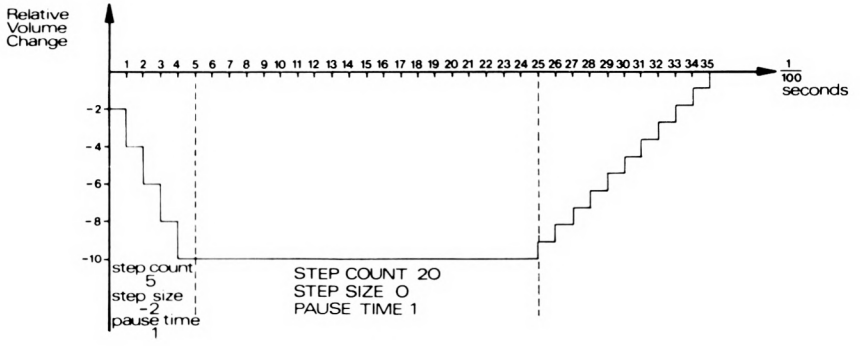
```
ENV 2,5,-2,1,2Ø,Ø,1,1Ø,1,1
SOUND 1,248,Ø,15,2
```

This envelope is given the number **2** and has three sections. In the first, the volume is reduced in **5** steps of **-2**. That is, it goes down by 2 at each step and there are 5 steps. The length of each of these steps is **1** hundredth of a second. The second section consists of **2Ø** steps, but with **Ø** (zero) reduction or increase in the volume (i.e. constant volume) per step. Again, the length of each of these steps is **1** hundredth of a second. Finally, the third section has **1Ø** steps, each increases the volume by **1**, and once again each step is **1** hundredth of a second long.

The **SOUND** command has a starting volume of **15**, so after the first section the volume will be reduced to 5, this is held constant for 20 hundredths of a second, then gets increased to 15 in the final section of the envelope.

It is perhaps a little difficult to visualise the shape of these envelopes. It often helps to draw them out on graph paper and take the values for the **ENV** command from this. The following show the shape of the two envelopes that have been defined so far:

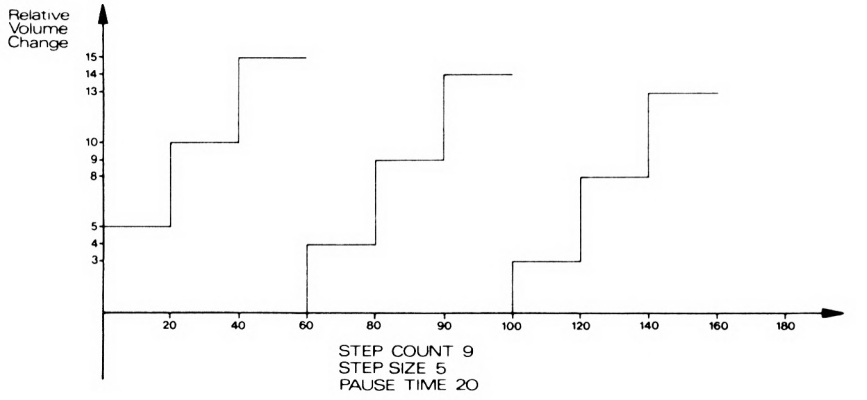




The maximum number of sections in an envelope is 5, and each section takes 3 values, so the **ENV** command can have up to 16 parts (including the first, that identifies which of the 15 envelopes (1 to 15) is being created). If the steps up or down cause the volume to go above 15 or below 0 then the value will wrap around; so the step above 15 is 0, and the step below 0 is 15:

```
ENV 3,9,5,20
SOUND 1,142,0,0,3
```

This simple envelope produces 9 steps, each step increasing the volume by 5, and each step lasting 20 hundredths of a second. After the first three steps therefore, the volume will be 15, so the next step will take it back round to 4, then 9 and so on. The diagram on the following page shows the effect:



The range of values for the number of steps is 0 to 127. The step size can be varied from −128 to +127 (negative values decrease the volume), and the pause time (i.e. the time between steps) can be anywhere in the range 0 to 255.

Now that we can create a volume envelope to give a characteristic shape, we may also want to define a characteristic tone pattern to include things such as vibrato - where the sound 'wavers' about the main tone of the note.

This is done in a very similar way to that in which volume envelopes are defined. Tone envelopes are defined using the **ENT** command. For example:

```
ENT 1,5,1,1,5,-1,1
SOUND 1,142,10,15,,1
```

A tone envelope is called into the **SOUND** command by entering the tone envelope's number as the sixth part of the **SOUND** command. Again the **ENT** must come before the **SOUND** command.

This first example of ENT specifies that in tone envelope number 1, there is to be 5 steps, each step increasing the tone period by 1, each step being 1 hundredth of a second. The next section of the envelope again specifies 5 steps, this time each step decreasing the tone period by -1, again each step being 1 hundredth of a second. This is a total length of 5 + 5 = 10. Notice that this length was specified in the SOUND command because a tone envelope does NOT determine the length of time that a note will play (in the same way that a volume envelope does). If the length used in the SOUND command is shorter than the length of the tone envelope, then the last part of the tone envelope will be lost. If it is greater, then the last part of the note will play with a constant tone. This also applies if a volume envelope is used to determine the note length.

(Notice the absence of any number in the volume envelope part of the SOUND command - this is because we have not created a volume envelope for this sound.)

The majority of tone envelopes will probably be much shorter than the expected length of the note. For this reason, a tone envelope can be made to repeat throughout the time that the note is playing. This is done by specifying a negative envelope number, the positive equivalent of which must be used in the SOUND command:

```
ENT -5,4,1,1,4,-1,1
SOUND 1,142,100,12,,5
```

This will produce a vibrato effect on the note. When defining tone envelopes, it is usually best if they can be made to vary symmetrically about the initial tone period, so that when the note repeats, it will not go further and further off the initial frequency (pitch) - try this:

```
ENT -6,3,1,1
SOUND 1,142,90,12,,6
```

You will notice that this tone will fall in frequency quite dramatically because each time the envelope is repeated, the tone period will increase by 3 and this will happen 30 times (90/3). This sort of effect can be quite useful for 'warbling' notes and sirens:

```
ENT -7,20,1,1,20,-1,1
SOUND 1,100,400,12,,7

ENT -8,60,-1,1,60,1,1
SOUND 1,100,480,12,,8
```

The number of possible tone envelopes is 15 (in the range 1 to 15), with a negative value indicating that the envelope should repeat. The number of steps (the first value in each group of three), can be anywhere between 0 and 239. As in the volume envelope, the step size can be from -128 to 127, and the pause time should be between 0 and 255. Once again, an ENT command, like an ENV command, can have a maximum of five sections (each containing 3 values).

The final part that can be added onto a **SOUND** command is a seventh value that indicates the level of noise which should be included in the sound. One point that should be borne in mind when including noise in a sound, is the fact that there is only one noise channel, so each subsequent noise period value overrides any previous one.

The noise can be added to a tone to give it a different sound, or can be used quite separately, by setting the tone period (second part) of the **SOUND** command to **Ø**, so that only noise is present. This is of use in making percussive type sounds. Try this

```
ENT -3,2,1,1,2,-1,1
ENV 9,15,1,1,15,-1,1
FOR a=1 TO 1Ø: SOUND 1,4ØØØ,Ø,Ø,9,3,15: NEXT
```

This could form the basis of a train noise. You will notice that this combines both types of envelopes and noise.

The duration and volume parts of the **SOUND** command are both set to 0 as they are determined by the volume envelope.

Now that we can hopefully use the **SOUND**, **ENV** and **ENT** commands to their full, we can look at various other associated commands and functions.

In describing the channel number of the **SOUND** command, you will remember that by adding 64 to it, the sound was marked as 'held' so that it would remain in the queue, without playing until released. The way in which a sound can be released is by use of the **RELEASE** command. The word **RELEASE** is followed by a bit significant number, where each bit is used to indicate one of the three possible channels to be released. Once again, it is not important to fully understand this, as long as you realise that:

4 means channel C
2 means channel B
1 means channel A

....and a combination of channels is released by adding up the values for each of the channels. So, to release held sounds on all three channels the following would be used:

```
RELEASE 7
```

....where 7 = 1 + 2 + 4. If no channels are held, then the **RELEASE** command is ignored. Try the following:

```
SOUND 1+64,9Ø
SOUND 2+64,14Ø
SOUND 4+64,215
RELEASE 3: FOR t=1 TO 1ØØØ: NEXT: RELEASE 4
```

The sounds that you might expect from these SOUND commands are not produced until the first RELEASE command which allows the sounds on channels A and B to play. After the delay, the sound on channel C is RELEASEd.

There is yet another method by which more than one sound can be made to rendezvous. When a sound is added to a queue that has the hold bit set (64 added), then it is not just that sound which is held, but all subsequent sounds sent to that queue. If more than four further sounds are sent to the held queue, then the machine will pause until the queue is released, perhaps by using a sub-routine that is called after a fixed period of time (using AFTER or EVERY). However this is not a particularly good method for using the sound system, as the program that contains the sound commands may pause from time to time as the sound queues fill up. This is also true if a lot of long sounds are added in quick succession. Try this:

```
10 FOR a=1 TO 8
20 SOUND 1,100*a,200
30 NEXT
40 PRINT "hello"
run
```

You will notice that the word 'hello' does not appear instantly, but only after the first three sounds. This is because program execution cannot continue until there is a free space in the queue.

The BASIC contains an interrupt mechanism, rather like that used in the AFTER and EVERY commands, and in ON BREAK GOSUB. This enables you to specify a sound playing sub-routine that is only called when a free space appears in the required queue. Try this:

```
10 a=0
20 ON SQ(1) GOSUB 1000
30 PRINT a;
40 GOTO 30
1000 a=a+10
1010 SOUND 1,a,200
1020 IF a<200 THEN ON SQ(1) GOSUB 1000
1030 RETURN
run
```

You will notice that the program never pauses. The SOUND command is only called when channel A's queue (number 1) has a free slot. This condition is detected by the ON SQ(1) GOSUB command in line 20. The command initialises an interrupt mechanism that will call the sound sub-routine when a free slot appears in the specified queue. Once ON SQ GOSUB has been used, it must be re-initialised, and this is done by line 1020 in the sound sub-routine. In this example, the sound sub-routine only re-initialises itself while the value of 'a' is less than 200.

In a complete program that may be moving objects on the screen, adding up totals and the like, a background tune can be kept continually playing by having a sub-routine called to play each note only when there is a free slot in the queue. This ensures that the program does not pause while waiting for a free slot to appear. If the note values for this tune were being read from DATA statements, then the sound routine could be set to stop re-initialising itself just before the data is exhausted.

The number within the brackets of the ON SQ() GOSUB command can be 1, 2 or 4 depending on which channel queue is to be tested for a free slot.

There is a function SQ() that may be used within a program to read the current state of any of the sound channels. The number within the brackets after SQ is again 1, 2, or 4, to specify for which channel the information should be returned. The function returns a bit-significant number and will again require an understanding of binary numbers to decipher it. The bits of the value returned have the following significance:

BIT	DECIMAL	MEANING
0,1,2	1,2,4	Number of free spaces in the queue
3	8	Note at head of queue is marked to rendezvous with A
4	16	Note at head of queue is marked to rendezvous with B
5	32	Note at head of queue is marked to rendezvous with C
6	64	Top note in queue has hold bit set (the queue is held)
7	128	A note is currently playing

Try this simple example:

```
10 SOUND 2,200
20 x=SQ(2)
30 PRINT BIN$(x)
run
```

This will print the binary number 10000100, in which bit 7 is set, indicating that the channel was currently playing when the SQ function was used. The last three digits 100, when converted to decimal give the value 4 indicating that there were 4 free entries in the queue. This function may be used to test a queue's status at a given point within a program - unlike ON SQ() GOSUB which will test and react to a queue's status at an indeterminate point.

So far, all the examples have just dealt with one or two notes of sound. Processing a whole group of unrelated notes, for example in a piece of music, can be achieved by listing the required notes in a DATA statement, from where they can be READ into a SOUND command:

```
10 FOR octave=-1 TO 2
20 FOR x=1 TO 7: REM notes per octave
30 READ note
40 SOUND 1,note/2↑octave
50 NEXT
60 RESTORE
70 NEXT
80 DATA 426,379,358,319,284,253,239
run
```

The final example program in this section greatly elaborates on this basic principle. The tune and rhythm play on channels A and B, using rendezvous to keep in step. The example demonstrates one of the ways in which **DATA** can be formatted so as to include note, octave, length and rendezvous information:

```
10 REM line 180 gives treble clef tune
20 REM line 190 gives bass clef tune
30 DIM scale%(12):FOR x%=1 TO 12:READ scale%(x%):NEXT
40 ch1%=1:READ ch1$:ch2%=1:READ ch2$
50 CLS
60 Spd%=12
70 scale$=" a-b b c+c d-e e f+f g+g"
80 ENV 1,2,5,2,8,-1,10,10,0,15
90 ENV 2,2,7,2,12,-1,10,10,0,15
100 ENT -1,1,1,1,2,-1,1,1,1,1
110 DEF FNM$(s$,s)=MID$(s$,s,1)
120 ch1%=1:GOSUB 200
130 ch2%=1:GOSUB 380
140 IF ch1%+ch2%>0 THEN 140
150 END
160 DATA &777,&70c,&6a7,&647,&5ed,&598
170 DATA &547,&4fc,&4b4,&470,&431,&3f4
180 DATA 4cr4f4f1f1g1A1-B2C2f4g2g1A1-B6A2Cr1f1g1f1g1
    a1-b1A1-b2C2g2A2g2f1g1a2g2f6e2c2e2c2g2e2c1-B1A2g
    2f4e4d8c4f3f1c2d4-b2fr2-B2A2g2f6e2gr4C4-B1a1f1-b
    1g2c2-b4a4g4fr6A2A2-B4-B2Ar2-B2A2g2f6e2g4C4-B1A1
    f1-B1g2C2-B4A4g8f.
190 DATA r4f4f8f4e4c4fr8f4e2f2e4d2e2d8c8c6e2f4g4g8e4
    f3f1c4dr8g4cr4e4c6f2d4c4c8fr8-e4dr8g8c4e4c6f2d4c
    4c8f.
200 REM send sound to channel A
210 p1$=FNM$(ch1$,ch1%)
220 IF p1$<>"r" THEN r1%=0:GOTO 240
230 r1%=16:ch1%=ch1%+1:p1$=FNM$(ch1$,ch1%)
```

continued on the next page

```
240 IF p1$="." THEN ch1%=0:RETURN ELSE l1%=VAL(p1$)
250 ch1%=ch1%+1
260 n1$=FNM$(ch1$,ch1%)
270 ch1%=ch1%+1
280 IF n1$="+" OR n1$="-" THEN 350
290 n1$=" "+n1$
300 nd1%=(1+INSTR(scale$,LOWER$(n1$)))/2
310 IF ASC(RIGHT$(n1$,1))>96 THEN o1%=8 ELSE o1%=16
320 SOUND 1+r1%,scale%(nd1%)/o1%,Spd%*l1%,0,1,1
330 ON SQ(1) GOSUB 200
340 RETURN
350 n1$=n1$+FNM$(ch1$,ch1%)
360 ch1%=ch1%+1
370 GOTO 300
380 REM send sound to channel B
390 p2$=FNM$(ch2$,ch2%)
400 IF p2$<>"r" THEN r2%=0:GOTO 420
410 r2%=8:ch2%=ch2%+1:p2$=FNm$(ch2$,ch2%)
420 IF p2$="." THEN ch2%=0:RETURN ELSE l2%=VAL(p2$)
430 ch2%=ch2%+1
440 n2$=FNM$(ch2$,ch2%)
450 ch2%=ch2%+1
460 IF n2$="+" OR n2$="-" THEN 530
470 n2$=" "+n2$
480 nd2%=(1+INSTR(scale$,LOWER$(n2$)))/2
490 IF ASC(RIGHT$(n2$,1))>96 THEN o2%=4 ELSE o2%=8
500 SOUND 2+r2%,scale%(nd2%)/o2%,Spd%*l2%,0,2
510 ON SQ(2) GOSUB 380
520 RETURN
530 n2$=n2$+FNM$(ch2$,ch2%)
540 ch2%=ch2%+1
550 GOTO 480
run
```

Graphically speaking....

This section describes the graphics facilities available. The first example builds up slowly demonstrating each major feature in turn.

To start off with, we will divide the screen into a text window (line 40) and a graphics window (line 30), setting the MODE and a couple of flashing colours along the way (line 20):

```
10 REM mask and tag in window
20 MODE 1:INK 2,10,4:INK 3,4,10
30 ORIGIN 440,100,440,640,100,300
40 WINDOW 1,26,1,25
50 CLG 2
```

If you **RUN** this program you will see a square of flashing colour halfway down the right-hand side of the screen. This square has been cleared to ink number 2 (flashing magenta/cyan) by line **50**, and the origin of co-ordinates has been moved to the bottom left hand corner of the square. The **MODE** command has set the graphics cursor to the origin of co-ordinates (X=0, Y=0) so we can draw a diagonal line across the square with line **60**:

```
60 DRAW 200,200,3
```

RUN the new program to see the effect. Now add:

```
80 MOVE 0,2:FILL 3
```

Line **80** puts the graphics cursor just inside one of the two halves of the square and fills it with ink 3. The boundary of the fill is the edge of the graphics window (in this case also the edge of the square) and anything drawn in the current graphics pen (3) or anything drawn in the ink being used to fill (also 3).

Now **RUN** the resulting program.

To prove the point about 'fill' edges, add line **70** below. Note that it is only the fact that the fill is in the same ink as the diagonal line that restricts the fill to half the square.

```
70 GRAPHICS PEN 1
run
```

Edit line **80** to **FILL** with ink **1** then **RUN** to prove this last point. Then restore it back to the original (**FILL 3**).

Now add lines **100** to **140**, which draw a box:

```
100 MOVE 20,20
110 DRAW 180,20
120 DRAW 180,180
130 DRAW 20,180
140 DRAW 20,20
run
```

The box is drawn in ink 1 because of line 70. If line 70 were omitted then we would need to add a ',1' either as the third parameter of the MOVE command in line 100, or the third parameter of the DRAW command in line 110 in order to instruct the computer to change graphics pens.

Join the dots....

Lines need not be solid, they can be dotted. The MASK command allows us to specify the size of the dots. The pattern will repeat every 8 pixels and each subsequent line will continue the dotting scheme from where the last line left off. A new MASK command (probably with the same parameter as the current one) will reset the dotting logic to the start of the 8 pixel pattern.

The dotting pattern is actually a single byte binary number where the bits set indicate where to put pen ink. In our example we will use a binary constant (called up by the '&X') indicating that we want four pixels drawn in the middle of each 8 pixel group, with the two either side not drawn. This will give a dashed line with four pixels on and four pixels off. To do this, add:

```
90 MASK &X00111100
run
```

But hold on a moment! This program does not give us a smooth continuation of the dotting around the corners as we expected. The reason for this is that each corner point is actually plotted twice, once as the last pixel of one line, and again as the first pixel of the next line. A clumsy way around this is to type in:

```
115 MOVE 180,22
125 MOVE 178,180
135 MOVE 20,178
run
```

....which produces the desired effect. However there is a simpler way, which is to add a second parameter ',0' to the end of the MASK command which tells the computer NOT to plot the first point of each line. Edit line 90 to read:

```
90 MASK &X00111100,0
```

....and delete the lines you just added by typing:

```
115
125
135
```

Now RUN, and once again the dotted box is symmetrical. Note that if the second MASK parameter is ',1' the command will be reset to draw whole lines including the first pixel.

Now look in between the dashes of the line. There is something in the bottom right triangle which is not in the top left triangle. This is the graphics paper, which is set to ink 2 by the CLG 2 command in line 50, but is invisible in the top left triangle because it is the same colour as the background. Alter line 50 to read:

```
50 CLG 2:GRAPHICS PAPER 0
```

....and re-run the program. The paper now shows up clearly all round the box.

It is possible to make the graphics paper invisible, or, as we call it 'transparent'. This means that a dotted line drawn over an existing picture will preserve the gaps between the dots. The graphics drawing is made transparent by adding a ',1' parameter to the GRAPHICS PEN command. (It is reset to non-transparent or 'opaque' by a ',0' parameter). Alter line 70 to read:

```
70 GRAPHICS PEN 1,1
run
```

....and observe the result.

As well as drawing lines (and plotting points) it is possible to write normal text characters at the position of the graphics cursor. This has the advantage that we can position the characters with much greater accuracy (within one pixel rather than within 8 pixels) and also that we can draw characters with the added flavour of graphics ink modes (see ahead).

To write characters at the graphics cursor location, simply position the graphics cursor at the top left-hand corner of where you want the character to be drawn, then issue the command TAG (or TAG #1 etc, for other text streams) followed by normal PRINTing commands. The graphics cursor is automatically stepped right by 8 pixels after each character is drawn. Add:

```
160 MOVE 64,108
170 TAG
180 PRINT "SALLY"
190 TAGOFF
run
```

(Any messages output by BASIC will be sent to the text screen irrespective of the state of the TAG/TAGOFF switch, but it is good practice to cancel the TAG assignment as soon as the TAGging is over.)

But what are the arrows after the name? Well, these are carriage-return CHR$(13) and line-feed CHR$(10) symbols. The graphics drawing software translates all of the first 32 ASCII characters into their printable versions (as if they were each preceded by a CHR$(1); when sent to the text screen). The reason for this is that most of the first 32 control codes are meaningful only to the text screen. By the same token for example, if we overflow to the right of the graphics window, no wrap-around is performed.

The carriage-return and line-feed symbols can be suppressed by the normal technique of ending the PRINT statement with a semicolon, i.e:

```
180 PRINT "SALLY";
run
```

Text sent to the graphics screen using TAG is influenced by the same GRAPHICS PEN commands as the line drawing. Thus currently, the name is written in GRAPHICS PEN 1, and is transparent. The command:

```
150 GRAPHICS PEN 1,0
run
```

....will switch back to opaque paper, whilst:

```
150 GRAPHICS PEN 0,1
run
```

....will write with ink number 0, transparently.

Now delete line 150 and RUN again. Ink number 1 + transparent mode (previously set in line 70) will be restored to the graphics pen.

Transparent characters

It is also possible to write characters to the text screen transparently, by using one of the control codes provided. Add:

```
200 PRINT #2,CHR$(22);CHR$(1)
210 LOCATE #2,32,14:PRINT #2,"******"
220 LOCATE #2,32,14:PRINT #2,"______"
230 PRINT #2,CHR$(22);CHR$(0)
run
```

Line 200 sets stream #2 to transparent mode. Note how the underlining appears to 'overstrike' the asterisks. This demonstrates that it is possible to build up composite characters, even in multiple colours. Line 230 switches off transparent mode, setting stream #2 back to opaque mode.

Ink modes

It is possible to draw using a graphics ink mode which combines the current drawing with the ink already on the screen. The final ink for each pixel is calculated by forming the logical combination of the old ink for the pixel with the graphics ink (pen or paper) being plotted. The logical combinations provided are XOR, AND and OR. The ink mode can be set either as the fourth parameter of **DRAW/DRAWR**, **PLOT/PLOTR** and **MOVE/MOVER** commands or by **PRINT**ing a **CHR$(23);** **CHR$(**‹mode›**)** control code sequence. In each case a value of 1 sets XOR plotting, 2 sets AND plotting and 3 sets OR plotting. A value of **0** restores the 'force' mode, where the graphics ink is used 'as is'.

The next example demonstrates the use of the XOR combination. XOR is often used in so-called Turtle Graphics because it has the property that drawing the same pattern twice restores the original image. Thus the square drawing routine is executed twice (in lines **110** and **130**), and the **TAG**ged printing is also executed twice (in lines **170** and **190**). The **FRAME** commands cause enough delay to make the effect visible. Note the use, in line **90**, of commands without a first parameter. This is quite in order in these commands, and simply leave the current settings of the first parameter unchanged.

The third parameter (**,1**) of the **MOVE** command in line **220** sets the **GRAPHICS PEN** to **1**, overriding the '3' set in line **60**. The XOR mode is set by the fourth parameter of the **DRAWR** command in line **230**. Note once more the missing parameter.

A reminder of the first-point plotting effect can be seen by removing the **MASK** command in line **90**. The corners of the square disappear because they are drawn twice (at the end of one line and the start of the next) and are therefore cancelled out by the XORing action.

```
10 REM XOR ink modes
20 MODE 1:INK 2,10:INK 3,4
30 ORIGIN 440,100,440,640,100,300
40 WINDOW 1,26,1,25
50 CLG 2:GRAPHICS PAPER 0
60 DRAW 200,200,3
70 MOVE 2,0:FILL 3
80 ORIGIN 440,0,440,640,0,400
90 GRAPHICS PEN ,1:MASK ,0
100 FOR y=60 TO 318 STEP 2
110 GOSUB 220
120 FRAME:FRAME
130 GOSUB 220
```

continued on the next page

```
140 NEXT
150 TAG
160 FOR y=60 TO 318 STEP 2
170 MOVE 96,y:PRINT CHR$(224);
180 FRAME:FRAME
190 MOVE 96,y:PRINT CHR$(224);
200 NEXT
210 END
220 MOVE 90,y,1
230 DRAWR 20,0,,1
240 DRAWR 0,20
250 DRAWR -20,0
260 DRAWR 0,-20
270 RETURN
run
```

Animation

It is possible to produce an animation effect by switching the colours assigned to inks. Although the contents of the screen memory are unchanged, there appears to be movement. The next example uses ORing of inks to write the numbers 1 to 4 onto the screen. (The shape is determined by scanning the character printed at the bottom left hand corner and reproducing what is found, in big block graphics.) The numbers are written in turn using inks 1,2,4 and 8 with the OR mode turned on -in this case with a control character sequence, see line **50**.

Lines **160** onwards rotate the palette according to a mathematical formula which results in one block graphics number at a time being displayed. The inks are set by inspecting each ink in turn and determining if it includes the binary component that we are looking for. For example, the number 3 was drawn in ink 4 and therefore to show the number 3 we must allocate a visible colour to all inks whose number contains a binary 4. Those inks are:

4 (0100), 5 (0101), 6 (0110), 7 (0111), 12 (1100), 13 (1101), 14 (1110), 15 (1111)

In a practical application the inks which require to be changed at each stage in the animation would be calculated, and lines **180** to **200** would be replaced by a speedier section of program.

```
10 REM latch animation
20 ON BREAK GOSUB 220
30 FOR i=1 TO 15:INK i,26:NEXT
40 m(1)=1:m(2)=2:m(3)=4:m(4)=8
50 MODE 0:PRINT CHR$(23);CHR$(3);:TAG
60 FOR P=1 TO 4
70 GRAPHICS PEN m(p),1
80 LOCATE #1,1,25:PRINT#1,CHR$(48+p);
90 FOR x=0 TO 7
100 FOR y= 0 TO 14 STEP 2
110 IF TEST(x*4,y)=0 THEN 140
120 MOVE (x+6)*32,(y+6)*16:PRINT CHR$(143);
130 MOVE (x+6)*32,(y+7)*16:PRINT CHR$(143);
140 NEXT y,x,p
150 LOCATE #1,1,25:PRINT#1," ";
160 FOR p=1 TO 4
170 FOR i= 1 TO 25:FRAME:NEXT
180 FOR i=0 TO 15
190 IF (i AND m(p))=0 THEN INK i,0 ELSE INK i,26
200 NEXT i,p
210 GOTO 160
220 INK 1,26
run
```

Colour plane sprites

In the example above we have seen how, having written graphics in inks 1,2, 4 and 8, an animation effect can be produced by colour changing. If the same inks are used, but the colours set up in a different way then a completely different effect can be produced. This effect is known as 'colour planes' and is demonstrated in the example below.

```
10 REM mountains
20 DEFINT a-z
30 INK 0,1:INK 1,26
40 INK 2,6:INK 3,6
50 FOR i=4 TO 7:INK i,9:NEXT
60 FOR i=8 TO 15:INK i,20:NEXT
70 MODE 0:DEG:ORIGIN 0,150:CLG:MOVE 0,150
80 FOR x=16 TO 640 STEP 16
```

continued on the next page

```
90 DRAW x,COS(x)*150+RND*100,4
100 NEXT
110 MOVE 0,0:FILL 4
120 cx=175:GOSUB 320
130 cx=525:GOSUB 320
140 SYMBOL 252,0,0,&C,&1F,&30,&7F,&FF
150 SYMBOL 253,0,6,&E,&F2,2,&F2,&FE
160 SYMBOL 254,0,&60,&70,&7F,&7F,&7F,&7F
170 SYMBOL 255,0,0,0,&F8,&EC,&FE,&FF
180 pr$=CHR$(254)+CHR$(255)
190 pl$=CHR$(252)+CHR$(253)
200 TAG:t!=TIME
210 FOR x=-32 TO 640 STEP 4
220 x2=((608-x)*2)MOD 640:hl=RND*10:hr=50*SIN(x)
230 GRAPHICS PEN 8,1:MOVE x,100+hr,,3:PRINT pr$;
240 GRAPHICS PEN 2,1:MOVE x2,115+hl,,3:PRINT pl$;
250 IF (TEST(x2-2,115+hl-12) AND 8)=8 THEN 380
260 IF TIME-t!<30 THEN 260
270 FRAME:t!=TIME
280 GRAPHICS PEN 7,1:MOVE x,100+hr,,2:PRINT pr$;
290 GRAPHICS PEN 13,1:MOVE x2,115+hl,,2:PRINT pl$;
300 NEXT
310 GOTO 210
320 MOVE cx,100
330 FOR x=0 TO 360 STEP 10
340 DRAW cx+SIN(x)*50+10*RND,100+COS(x)*25+10*RND,1
350 NEXT
360 DRAW cx,100:MOVE cx,90:FILL 1
370 RETURN
380 ENT -1,1,1,1
390 SOUND 1,25,400,15,,1,15
400 FOR y=100+hr TO -132 STEP -2
410 GRAPHICS PEN 7,1:MOVE x,y,,2:PRINT pr$;
420 GRAPHICS PEN 8,1:MOVE x,y-2,,3:PRINT pr$;
430 NEXT
440 GOTO 70
run
```

To explain how this works we must, once again, visualise the **INK** number in binary. Starting with the highest **INK** number (15), all the **INK**s with the '8' bit present (15 down to 8) are set to cyan. Then all the **INK**s with the '4' bit present (7 down to 4) are set to green. **INK**s 2 and 3, each with the '2' bit present are set to red, and finally **INK** 1 is set to bright white, with **INK** 0 remaining as blue.

On the screen the graphics are ORed into place - see lines 230 and 240. The colour seen on the screen in any particular pixel is determined by the most significant bit of the resultant at that point. Therefore an image in a 'more significant' plane will always obscure an image in a 'less significant' plane, but the background will be preserved and can be seen again if the 'more-significant' image is removed. The way to remove an image is to plot it using the AND ink mode with INK numbers of 7,11,13 or 14 removing original INKs of 8,4,2 and 1 respectively - see lines 280 and 290.

Graphics Using the Extra Memory (6128 only)

To conclude this chapter, we provide you with a comprehensive 'Graphics Screen Designer program that makes use of the extra 64K of RAM in the 6128.

```
10 'SCREEN DESIGNER by DAVID RADISIC
20 ' copyright (c) AMSOFT 1985
30 '
40 'Remember to RUN "BANKMAN" before running program!
50 '************************************************
60 '
70 ON ERROR GOTO 2740
80 DEFINT a-x
90 MODE 1:ch=127:cmnd=1:pn(0)=0:pn(1)=26:pn(2)=
   15:pn(3)=6:pn(4)=0:pn=1:norx=1:menu=1:zzz=HIMEM
100 DIM command$(22)
110 norx$(0)="Normal":norx$(1)="XOR    ":norx$(2)=
    "Transp":norx$(3)="XOR    "
120 RESTORE:READ cmnds$(1),cmnds$(2):cmnd$=CHR$(16)+
    CHR$(&7F)+cmnds$(1)+cmnds$(2)
130 READ cmno:FOR i=1 TO cmno:READ command$(i):NEXT
140 READ st$:IF st$<>"**" THEN cmnd$(cmnd)=st$:cmnd=
    cmnd+1:GOTO 140
150 WINDOW #0,1,40,1,3:PAPER #0,0:PEN #0,1:CLS #0
160 WINDOW #1,1,40,4,4:PAPER #1,3:PEN #1,1:CLS #1
170 ORIGIN 0,0,0,640,0,334
180 x=320:y=200:MOVE x,y
190 BORDER pn(4):FOR i=0 TO 3:INK i,pn(i):NEXT
200 MASK 255,0:PAPER 0:PEN 1:PAPER #1,3:PEN #1,1:
    GRAPHICS PEN pn,norx
210 IF flag<>5 THEN 280
```

Continued on the next page

```
220 IF pn<2 THEN pnt$=CHR$(240):px=(pn+1)*13 ELSE IF
    pn<4 THEN pnt$=CHR$(241):px=(pn-1)*13 ELSE pnt$
    =CHR$(243):px=37
230 LOCATE px,2:PRINT pnt$;
240 LOCATE 1,1:PRINT USING"  PEN 0 : ##  PEN 1 : ##";
    pn(0);pn(1);
250 LOCATE 29,2:PRINT USING"Border : ##";pn(4)
260 LOCATE 1,3:PRINT USING"  PEN 2 : ##  PEN 3 : ##";
    pn(2);pn(3);
270 LOCATE px,2:PRINT " ";
280 LOCATE #1,1,1:PRINT#1,USING"X :#### Y :####    ";
    x;y;:PRINT #1,"Plot mode : ";norx$
    (norx+(undraw*2));" ";
290 IF flag=0 THEN GOSUB 2260
300 '
310 GOSUB 970
320 '
330 IF flag>0 THEN 390
340 IF i$="" THEN 390
350 cmnd=INSTR(cmnd$,i$):IF cmnd=0 THEN 390
360 IF cmnd=1 THEN CLG:x=320:y=200:GOTO 390
370 IF cmnd=2 THEN RUN 70
380 ON cmnd-2 GOSUB 1240,1410,1520,1640,1840,1860,
    1950,2020,2090,2120,2170,2200,2660,2660,2660,
    2660,2390,2330,2200
390 IF tx=0 AND ty=0 THEN 200
400 IF flag>0 THEN 440
410 GOSUB 630
420 GOSUB 680:FRAME:GOSUB 680
430 GOTO 200
440 MOVE tempx,tempy,pn,1
450 ON flag GOSUB 470,490,550,640
460 GOTO 200
470 PLOT x,y:GOSUB 630:PLOT x,y
480 RETURN
490 DRAW tempx+x,tempy:DRAW tempx+x,tempy+y
500 DRAW tempx,tempy+y:DRAW tempx,tempy
510 GOSUB 630
520 DRAW tempx+x,tempy:DRAW tempx+x,tempy+y
530 DRAW tempx,tempy+y:DRAW tempx,tempy
540 RETURN
550 MOVE tempx,tempy:DRAWR x,y
560 IF triside=0 THEN 580
570 DRAW tempxx,tempyy:DRAW tempx,tempy
```

Continued on the next page

```
580 GOSUB 630
590 MOVE tempx,tempy:DRAW tempx+x,tempy+y
600 IF triside=0 THEN RETURN
610 DRAW tempxx,tempyy:DRAW tempx,tempy
620 RETURN
630 x=x+tx:y=y+ty:RETURN
640 MOVE tempx,tempy:DRAW x,y
650 GOSUB 630
660 MOVE tempx,tempy:DRAW x,y
670 RETURN
680 ' draw and undraw cursor
690 IF flag=5 THEN RETURN
700 MASK 255,1
710 IF flag>1 THEN xx=tempx+x:yy=tempy+y ELSE xx=
    x:yy=y
720 IF flag=4 THEN xx=x:yy=y
730 IF flag=1 THEN xx=x:yy=y
740 IF undraw=1 THEN 820
750 GOSUB 790
760 MASK 255,0
770 IF i$=" " THEN GOSUB 2150:i$=""
780 RETURN
790 MOVE xx-4,yy,pn,1:DRAW xx+4,yy
800 MOVE xx,yy-4:DRAW xx,yy+4
810 MOVE xx,yy,,xorn:RETURN
820 nx=1:GOSUB 1220
830 FRAME:GOSUB 1220
840 IF i$=" " THEN nx=norx:GRAPHICS PEN pn,1:GOSUB
    1220
850 i$=""
860 IF flag<>6 THEN 760
870 IF moved=0 AND j$<>"" AND (j$<CHR$(240) OR j$>
    CHR$(247)) THEN ch=ASC(j$):moved=1
880 IF moved=0 THEN RETURN
890 LOCATE 5,2
900 FOR i=ch-5 TO ch+5
910 PEN ABS(i<>ch)+1
920 ch$=CHR$(1)+CHR$(ABS(i+256)MOD 256)
930 IF ch=i THEN PRINT" "ch$" "; ELSE PRINT ch$;
940 NEXT
950 PEN 1:PRINT"   = "ch"  ";
960 GOTO 760
970 ty=0:tx=0:GOSUB 680:FRAME:GOSUB 680
980 IF INKEY(0)<>-1 OR INKEY(72)<>-1 THEN ty=16
990 IF INKEY(2)<>-1 OR INKEY(73)<>-1 THEN ty=-16
```

Continued on the next page

```
1000 IF INKEY(8)<>-1 OR INKEY(74)<>-1 THEN tx=-16
1010 IF INKEY(1)<>-1 OR INKEY(75)<>-1 THEN tx=16
1020 IF INKEY(21)<>-1 OR INKEY(76)<>-1 THEN tx=tx/8:
     ty=ty/8
1030 IF tx=0 AND ty=0 THEN moved=0 ELSE moved=1
1040 j$=INKEY$:i$=UPPER$(j$)
1050 IF (i$=" " OR i$=CHR$(13)) AND flag>0 THEN 1090
1060 IF flag=5 THEN 1120
1070 IF flag=6 THEN 1170
1080 RETURN
1090 ON flag GOSUB 1240,1410,1640,1860,1950,2020
1100 i$=""
1110 RETURN
1120 IF moved=0 THEN RETURN
1130 IF tx>2 THEN pn=(pn+1) MOD 5 ELSE IF tx<-2 THEN
     pn=ABS((pn<1))*5-1+pn
1140 IF ty>2 THEN pn(pn)=(pn(pn)+1) MOD 27 ELSE IF
     ty<-2 THEN pn(pn)=ABS((pn(pn)<1))*27-1+pn(pn)
1150 GRAPHICS PEN pn:PEN #1,pn
1160 tx=0:ty=0:BORDER pn(pn):RETURN
1170 IF tx<0 THEN ch=ABS(ch+255) MOD 256
1180 IF ty<0 THEN ch=ABS(ch+246) MOD 256
1190 IF tx>0 THEN ch=(ch+1) MOD 256
1200 IF ty>0 THEN ch=(ch+10) MOD 256
1210 tx=0:ty=0:RETURN
1220 TAG:MOVE xx-8,yy+6,pn,nx:PRINT CHR$(ch);:TAGOFF
1230 RETURN
1240 ' C
1250 IF flag=1 THEN 1290
1260 ro=1:GOSUB 2240
1270 tempx=x:tempy=y:flag=1
1280 RETURN
1290 IF tempx=x AND tempy=y THEN 1390
1300 PLOT x,y,,1
1310 tix=MAX(x,tempx)-MIN(tempx,x):tiy=MAX(y,tempy)-
     MIN(tempy,y)
1320 ti=SQR((tix↑2)+(tiy↑2))
1330 ORIGIN tempx,tempy
1340 PLOT 0,0,pn,0:MOVE 0,-ti
1350 FOR z=0 TO PI*2+0.01 STEP PI/(ti/2)
1360 DRAW SIN(z+PI)*ti,COS(z+PI)*ti,pn,norx
1370 NEXT z
1380 ORIGIN 0,0
1390 x=tempx:y=tempy:tempx=0:tempy=0:flag=0
1400 RETURN
```

Continued on the next page

```
1410 ' B
1420 IF flag=2 THEN 1470
1430 ro=2:GOSUB 2240
1440 tempx=x:tempy=y:flag=2
1450 x=0:y=0
1460 RETURN
1470 IF norx=1 THEN 1500
1480 MOVE tempx,tempy:DRAW tempx+x,tempy,,norx
1490 DRAW tempx+x,tempy+y:DRAW tempx,tempy+y:DRAW
     tempx,tempy
1500 x=tempx:y=tempy:flag=0
1510 RETURN
1520 ' F
1530 ro=3:GOSUB 2240
1540 GOSUB 1620:IF i$=" " THEN 1600
1550 edgecol=VAL(i$)
1560 ro=4:GOSUB 2240
1570 GOSUB 1620:IF i$=" " THEN 1600
1580 filler=VAL(i$)
1590 MOVE x,y,edgecol:FILL filler
1600 flag=0:i$=""
1610 RETURN
1620 i$=INKEY$:IF (i$<"0" OR i$>"3") AND i$<>" " THEN
     1620
1630 RETURN
1640 ' T
1650 IF flag=3 THEN 1700
1660 flag=3:ro=5:GOSUB 2240
1670 tempx=x:tempy=y
1680 x=0:y=0
1690 RETURN
1700 IF triside<>0 THEN 1770
1710 ro=6:GOSUB 2240
1720 MOVE 0,0,pn,1:GOSUB 590
1730 tempxx=tempx+x:tempyy=tempy+y:x=x/2:y=20
1740 triside=1
1750 GOSUB 550:GOSUB 590
1760 RETURN
1770 IF norx=1 THEN 1800
1780 MOVE tempxx,tempyy,,norx:DRAW tempx,tempy
1790 DRAW tempx+x,tempy+y:DRAW tempxx,tempyy
1800 tempxx=0:tempyy=0
1810 x=tempx:y=tempy:triside=0
1820 tempx=0:tempy=0:flag=0
```

Continued on the next page

```
1830 RETURN
1840 ' @
1850 norx=1:undraw=undraw XOR 1:RETURN
1860 ' L
1870 IF flag=4 THEN 1910
1880 ro=7:GOSUB 2240
1890 tempx=x:tempy=y:flag=4
1900 RETURN
1910 IF norx=1 THEN 1930
1920 MOVE tempx,tempy,,norx:DRAW x,y
1930 x=tempx:y=tempy:flag=0
1940 RETURN
1950 ' I
1960 IF flag=5 THEN flag=0:CLS:INK 3,tmpcol:INK pn,
     col:GOTO 1990
1970 CLS:flag=5:BORDER pn(pn)
1980 RETURN
1990 FOR i=0 TO 3:INK i,pn(i):NEXT:BORDER pn(4)
2000 IF pn=4 THEN pn=1
2010 CLS:RETURN
2020 ' A
2030 IF flag=6 THEN 2070
2040 tempx=0:tempy=0:CLS
2050 undraw=1:flag=6:norx=1:moved=1
2060 RETURN
2070 flag=0
2080 RETURN
2090 ' N
2100 norx=0
2110 RETURN
2120 ' E
2130 GRAPHICS PEN pn,0:TAG:MOVE xx-8,yy+6,,0:PRINT
     " ";:TAGOFF
2140 RETURN
2150 '<SPACE>
2160 PLOT x,y,pn,norx:RETURN
2170 ' X
2180 norx=1
2190 RETURN
2200 ' M
2210 menu=menu MOD 2+1
2220 GOSUB 2260:RETURN
2230 i$=UPPER$(INKEY$):IF i$="" OR INSTR(ser$,i$)=0
     THEN 2230 ELSE RETURN
```

Continued on the next page

```
2240 CLS:undraw=0:PRINT cmnd$(ro);:LOCATE 1,3:PRINT
     "<SPACE> ";:IF ro=3 OR ro=4 THEN PRINT"To exit"
2250 RETURN
2260 CLS:flag=-1
2270 FOR i=1 TO LEN(cmnds$(menu))
2280 ps=i+ABS(menu=2)*LEN(cmnds$(1))
2290 PEN 1:PRINT"<"MID$(cmnds$(menu),i,1)">"MID$
     (command$(ps),2,4)" ";
2300 NEXT
2310 PRINT"<CLR>   <DEL>   <SPACE>";
2320 RETURN
2330 ' S
2340 GOSUB 2460:IF filename$="" THEN 2370
2350 GOSUB 2550
2360 SAVE filename$,b,&C000,&4000
2370 GOSUB 2260
2380 RETURN
2390 ' R
2400 GOSUB 2460:IF filename$="" THEN 2440
2410 GOSUB 2730
2420 LOAD filename$,&C000
2430 GOSUB 2570
2440 GOSUB 2260
2450 RETURN
2460 CLS:LOCATE 10,3:PRINT"<RETURN> to Abort!";
2470 LOCATE 1,1:PRINT"Enter Filename :";
2480 INPUT "",filename$:IF filename$="" THEN RETURN
2490 n=INSTR(filename$,"."):IF n=0 THEN 2520
2500 IF n=1 THEN 2460
2510 filename$=LEFT$(filename$,n-1)
2520 filename$=LEFT$(filename$,8)+".scn"
2530 CLS
2540 RETURN
2550 FOR i=0 TO 4:POKE &C000+i,pn(i):NEXT
2560 RETURN
2570 FOR i=0 TO 4:pn(i)=PEEK(&C000+i) MOD 27:NEXT
2580 cn=0:FOR i=0 TO 2:IF pn(i)=pn(i+1) THEN cn=cn+1
2590 NEXT:IF cn=3 THEN 2630
2600 FOR i=0 TO 3:INK i,pn(i):NEXT
2610 BORDER pn(4):pn=1:GRAPHICS PEN pn
2620 RETURN
2630 pn(0)=0:pn(1)=26:pn(2)=15:pn(3)=6:pn(4)=0
2640 GOTO 2600
2650 ' 1, 2, 3, & 4
```

Continued on the next page

```
2660 CLS:PRINT"Do you wish to <S>tore":PRINT TAB(16)
     "<R>etrieve":PRINT TAB(13)"or <E>xchange  the
     screen?"
2670 ser$="SRE"+CHR$(13):GOSUB 2230:IF i$=CHR$(13)
     THEN 2260
2680 bnk2=(cmnd-13):bnk1=1
2690 IF i$="S" THEN CLS:GOSUB 2550:|SCREENCOPY,bnk2,
     bnk 1
2700 IF i$="R" THEN GOSUB 2730:|SCREENCOPY,bnk1,bnk2:
GOSUB 2570
2710 IF i$="E" THEN CLS:GOSUB 2730:GOSUB 2550:|SCREEN
SWAP,bnk2,bnk1:GOSUB 2570
2720 GOSUB 2260:RETURN
2730 FOR i=0 TO 3:INK i,0:NEXT:BORDER 0:RETURN
2740 CLS:GOSUB 2600:RESUME 2260
2750 DATA "CBFT@LIANEXM","1234RSM"
2760 DATA 19,Circle,"Box  ","Fill ",Triangle,
     Alternate,"Line ","Inks ",ASCII,Normal,Erase,
     "Xor  ","Menu ","1st  ","2nd  ","3rd  ","4th  ",
     Restore,"Save ","Menu "
2770 DATA Circle,Box,Edge colour,Filler colour,
     Triangle 1,Triangle 2,Line,**
```

The two RSX commands **|SCREENCOPY** and **|SCREENSWAP** used in this program are provided by the 'Bank Manager' utility on Side 1 of your system disc. The commands facilitate the copying and swapping of the various screens between the appropriate memory blocks, and between the two 64K banks of memory.

You must therefore run the 'Bank Manager' utility BEFORE running the Screen Designer program, and to do this, insert Side 1 of your system disc, and type:

```
run "bankman"
```

You may then run the Screen Designer program.

What happens next?

When you have run the Bank Manager utility followed by the Screen Designer program, you will see a menu of options displayed, together with a flashing graphics-cursor at the centre of the screen.

From then on, simply press the appropriate letter (e.g. ‹C› for Circle) to carry out the desired menu option.

As a demonstration, type:

```
C
```

....then press the cursor up key ⌃ until the flashing graphics-cursor moves up about an inch from the centre of the screen.

Finally, press the ‹SPACE› bar to execute the Circle function, and you will then be returned to the menu.

Typing **M** (for ‹M›enu) will transfer you to an alternative menu from where you may ‹S›ave and ‹R›estore screens, and manipulate the contents of the ‹1›st, ‹2›nd, ‹3›rd, and ‹4›th screens (held in the second 64K bank of memory).

To use these screen functions, type the 'memory number' (**1**, **2**, **3**, or **4**) and a further menu will be displayed.

From this you can select:

S	to Store the screen
R	to Retrieve a screen
E	to Exchange screens
[RETURN]	to abort the operation

If for example, you wish to store the current screen into memory number 2, then type **2** followed by **S**.

When returning to the alternative menu (after a ‹R›estore, ‹S›ave, or a screen manipulation), typing **M** again will toggle back to the original menu.

Screen Designer provides a range of drawing facilities, including boxes, circles, triangles, lines, setting/resetting points, filling, and character drawing.

When a screen has been completed, it may be saved to disc for later viewing/editing using the ‹S›ave option from the alternative menu, and may be loaded back at any time using the ‹R›estore option.

Well, that concludes this last chapter in the manual. By using, analysing, and experimenting with the programs that we have provided throughout this guide, you should have acquired a fair understanding of AMSTRAD BASIC and of the 464/6128 itself.

Further details....

Finally, we provide a set of 4 appendices which comprise: the CP/M End User Licence Agreement, a glossary of terms, some games programs' listings, and a comprehensive index to the manual.

We hope that you have found this manual informative and interesting, and thank you for purchasing the 464/6128.

Appendix 1
AMSTRAD

Software Licence Agreement

Amstrad Software licence agreement for all software described in this manual.

VERY IMPORTANT

Licence relating to the Software described in this manual.

Software licence agreement

Amstrad provides the Software and the Program described in this manual incorporated in or embedded on magnetic disc, tape, hard disc or other media and the descriptive material related to that Program in this manual or in other documents strictly subject to the terms and conditions of this Licence Agreement.

By breaking the Seal on the package in which the Software is provided and/or by entering, utilising, running, listing, copying or otherwise manipulating the magnetic disc, tape, hard disc, other media, Program, manual or other documentation other than for the purpose of erasure or destruction as set out below you agree to be bound by all the terms and conditions of this Licence Agreement.

If you do not agree to be bound by any one or more of the terms and conditions of this Licence Agreement you must destroy the Software Package, manual and documents and erase the Program and Software from the disc, tape, hard disc or other media on or in which it is incorporated or embedded.

Definitions

1. "Machine" means the single microcomputer, or PC on which you use the Software. Multiple CPU systems require additional licences.

2. "Software" means the set of programs, discs, tapes, documentation and related materials described in and supplied with this manual.

3. "Program" is a part of the Software and means the instructions, codes, messages or other information contained in, or embedded on magnetic disc, tape, hard disc or any other media, or any part of it.

4. "Amstrad" means AMSTRAD PLC, 169 Kings Road, Brentwood, Essex, CM14 4EF.

5. "Licensee" means the purchaser if purchased for own use or, if purchased for company use, the purchaser, that company and its employees.

Licence

Amstrad grants a non-exclusive Software Licence to Licensee to:

1. Use the supplied Program on a single Machine.

2. Where copy protection is not incorporated, copy the Program into any machine-readable or printed form for backup or modification purposes in support of Licensee's use of the Program on a single Machine. Licensee may make only one (1) copy of the Program for such purposes. Copying of documentation and other printed materials is prohibited. Disassembly, reverse compilation and reverse engineering of the Program or Software is prohibited.

Terms

1. Software supplied by Amstrad is copyright. Licensee agrees to uphold these copyrights.

2. This Licence Agreement enters into effect at the time you open or break the seal on the Software package, or enter, use, run, list, copy or otherwise manipulate the Program, and is effective until terminated.

3. Licensee may terminate this Licence Agreement at any time by destroying the Software together with all copies. The benefit derived by Licensee from the Licence will terminate automatically if the terms of this Agreement are violated and Amstrad may demand the return of the Software immediately. Any such termination shall be without prejudice to any accrued rights of the parties.

4. This Licence is personal to Licensee and may not be assigned to any other person, persons or company.

5. The right to "lend", "hire", "rent", "sell", or otherwise transfer the Software in part or in whole is not granted to Licensee.

6. Licensee may not use, copy, modify, or transfer the Software or documents, or any copy, modification or merged portion, in whole or in part, except as expressly provided for in this Licence Agreement.

7. The Program cannot be transferred via any media other than that on which it is supplied. It cannot therefore be transferred via such media as telecommunication lines.

8. Licensee agrees to take all possible steps to protect the Software from unauthorised use, reproduction or distribution.

Limited Warranty

THE SOFTWARE IS PROVIDED AND LICENSED ON AN "AS IS" BASIS. LICENSEE ASSUMES RESPONSIBILITY FOR SELECTION OF A PROGRAM TO ACHIEVE LICENSEE'S INTENDED RESULTS AND LICENSEE IS RESPONSIBLE FOR INSTALLATION USE AND RESULTS OF USE. EXCEPT AS IS HEREIN EXPRESSLY STATED, AMSTRAD MAKES NO REPRESENTATION, GIVES NO WARRANTY AND ACCEPTS NO CONDITION OF ANY KIND WHATSOEVER EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY AND FITNESS FOR PURPOSE (SPECIFIC OR GENERAL).

AMSTRAD DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS OR THAT THE OPERATION OF THE PROGRAM WILL BE UNINTERRUPTED OR ERROR FREE.

AMSTRAD DOES NOT RECOMMEND USE OF THE SOFTWARE FOR LIFE CRITICAL FUNCTIONS OR FOR ANY APPLICATIONS WHERE A SOFTWARE ERROR MAY CAUSE FINANCIAL LOSS, DAMAGE OR EXPENSE.

However, Amstrad gives a limited warranty that the discs, tapes or other non-hardware media on which the Program is embedded and supplied by Amstrad, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of receipt by, of delivery to Licensee as evidenced by a dated copy of Licencee's proof of purchase receipt. In the event that the discs, tapes or other non-hardware media are found to be so defective within the said period, Amstrad will replace them free of charge, if they are returned postage prepaid.

Amstrad will also, AT ITS DISCRETION, by modification or alteration endeavour to remedy any Program errors, attributable to Amstrad, which are provable and demonstrable, and reported to Amstrad in writing within the aforesaid period.

Limitations

UNDER NO CIRCUMSTANCES WILL AMSTRAD, THE AUTHOR, OR THE MANUFACTURER, DEVELOPER OR SUPPLIER OF THE LICENSED SOFTWARE BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR EXEMPLARY DAMAGES OR OTHER ECONOMIC OR PHYSICAL LOSS, EVEN IF AMSTRAD OR SUCH OTHER PARTY(IES) HAS/HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND LOSS. UNDER NO CIRCUMSTANCES WILL LIABILITY EXCEED THE PURCHASE PRICE OF THE SOFTWARE.

Indemnity

The Licensee shall indemnify Amstrad in respect of any liability incurred by Amstrad or the Software author as a result of a claim brought by Licensee against Amstrad directly or against a third party, for negligence, breach of contract or product liability.

Governing Law

This agreement shall be construed, interpreted and governed in accordance with the Laws of England (or in the case of Microsoft products, Washington, USA).

U.S. Government restricted rights in respect of Microsoft programs

The Program and documentation are provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the US Government is subject to restrictions as set forth in subdivision b(3)(ii) of The Rights in Technical Data and Computer Software clause of 252.227-7013 of the DODFAR.

Contractor/manufacture is Microsoft Corporation/160 11 NE 36th Way/Box 9701/Redmond, WA 98073-9717.

General

If any of the above provisions thereof are invalid or unlawful under any applicable Law they are to that extent only deemed omitted.

SOME STATES DO NOT ALLOW EXCLUSION OF IMPLIED WARRANTIES AND CONDITIONS OR LIMITATIONS OF LIABILITY. IN SUCH CIRCUMSTANCES THE ABOVE EXCLUSIONS AND LIMITATIONS MAY NOT APPLY TO LICENSEE. THIS WARRANTY GIVES THE LICENSEE CERTAIN LEGAL RIGHTS BUT LICENSEE MAY ALSO HAVE OTHER LEGAL RIGHTS WHICH VARY FROM STATE TO STATE.

Export of US Technology

No US technology, Program or Software may be exported or sold or transferred directly or indirectly to any person in the knowledge that export may occur without application for and obtaining necessary US Government Export or Re-Export licence issued by the US Department of Commerce, Washington DC 20230.

AMSTRAD PLC
169 Kings Road
Brentwood
Essex, CM14 4EF

Appendix 2
Glossary of Terms

Some commonly used terms from the world of computing....

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, and with other users of home computers, in order 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 written and read.

Adventure game

A cult with some, and a bore to others. A text-based computer game in which the player 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 arithmetic 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 arithmetic and logical operations - not of direct concern except in machine code programming.

Ambiguous File Name

A file name containing one or more wildcard characters. Ambiguous filenames refer to more than one specific file name and are used to refer to one or more files at a time.

AMSDOS

AMStrad Disc Operating System. The program that allows Locomotive BASIC to access disc files.

A/D
Analogue

A state where change between a start and finish point occurs gradually rather than instantaneously. 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 for example, spacemen invade and are vanquished, or 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'. 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.

Argument

An independent variable, e.g. in the expression x+y=z, the terms x, y, and z are the arguments.

Array

A 2 dimensional matrix (grid) in which data is stored by addressing with the 'horizontal' and 'vertical' co-ordinates.

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.

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).

Backup

A duplicate copy of information used as a safeguard in case the original is lost or accidentally damaged. Making a backup refers to the process of duplicating a disc or disc file.

Bar code

A computer readable printed code that can be read by optical techniques such as scanning by a low power laser. Look at the bottom of a box of soap powder to see an example.

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.

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 compiled 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.

BDOS

Basic Disc Operating System. This is the part of the CP/M operating system which provides an interface for a user program to use the functions of CP/M.

Benchmark

A standard task that can be given to different computers to compare their speed, efficiency and accuracy, e.g. calculating the square root of 99.999 squared.

Binary

The number system with base 2, in which all numbers are made up from the two binary digits 0 and 1. (See part 1 of the chapter entitled 'At your leisure....'.)

Binary number

A number represented in binary notation. Signified in 464/6128 programming by the prefix &X. e.g. &X0101 = (decimal) 5.

BIOS

Basic Input/Output System. This is the hardware dependent part of CP/M that is written specifically for one type of computer. All the input and output to the screen, keyboard, disc and so on is performed through the BIOS.

Bit

Shortform of BInary digiT. A binary digit is 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.

Boolean algebra

The statement of logical relationships where there can only be two answers: true or false. Usually signified as 1 or 0.

Boot

The process of loading an operating system into memory. When CP/M is started from BASIC a small boot program is loaded automatically from the disc, which then loads the rest of the operating system into memory.

Booting or Bootstrapping

Programs and operating systems don't load themselves, they are 'bootstrapped' by a small routine in ROM (usually), that initiates the loading processes at a specific location in memory.

Buffer

An area of memory reserved for temporarily storing, or buffering, information during an information transfer.

Bug

A problem on a scale ranging from an 'unexpected feature' based on some obscure aspect of the use of a program (e.g. 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.

Built-in commands

Commands that are part of an operating system. They are always quicker than transient commands because they are not accessed from disc.

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 464/6128 bus is presented on the connector, at the rear of the computer.

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.

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 buzzphrases 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 quickly and easily, but costs considerably more than software supplied on disc.

Cassette

Apart from the obvious recording tape variety, a generic term that encompasses a variety of 'packages' - including ROM software etc.

CCP

Console Command Processor. This is a module of CP/M that interprets and executes user input from the keyboard. Usually commands are input which the CCP loads and executes.

Character

Any symbol that can be represented in a computer and displayed by it, including letters, numbers and graphics symbols.

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

A 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'.

Cold start

The process of booting and initialising an operating system. A cold start of CP/M is performed when the `ICPM` command is used.

Command

A programming instruction.

Compiler

A complex program that converts complete programs written in a high level interpretive languages like BASIC into 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.

Console mode

CP/M direct mode; the `A>` appears on the screen, and the system awaits input of a CP/M or utility command.

Corruption

The destruction or alteration of the contents of a disc file or memory, in an undesirable and potentially unrecoverable manner.

CP/M

Control Program for Microcomputers. A disc based operating system by Digital Research that provides a standard systems interface to software written for a wide range of microprocessor based computer systems.

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 microprocessor 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.

Default

The value assumed in the absence of any user output. For example, when CP/M is started Drive `A:` is assumed to be the default drive.

Delimiter

(See Separator)

Diagnostic

A message automatically produced by a computer to indicate and identify an error in a program.

Digital

Describes the transition 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.

Directory

A section of disc containing entries for each file on the disc. A list of the contents of a disc.

Disc (or 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 in a protective envelope, with access for the reading head provided by a 'window'. On a 3″ disc the window is covered by a metallic shutter, which automatically slides across when the disc is out of the disc drive.

Disc drive

The mechanism used to spin and access the data on the surface of the disc.

Documentation

The manuals that are supplied with computers or software to explain how they are operated.

DOS

Disc Operating System. The software that controls all the operations of a disc drive.

Dot matrix

A rectangular grid of dots on which a character can be displayed by the selection of certain dots.

Double sided

A disc that can store information on both sides. A double sided disc drive can access both sides of a disc without the need to turn the disc over.

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.

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.

Edit

To correct or make changes to data, a program or text.

Editor

A program that is usually in the ROM of the computer, which 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 by ultra-violet light, and new program recorded. An EEPROM is similar, except that it may be electronically 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 data, generally stored on cassette or disc.

File name

The name of a file. In CP/M or AMSDOS, an additional three character file type, preceded by a dot . is allowed.

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 its decimal point permitted to settle in the required position. The method is particularly useful when dealing with large numbers.

Floppy disc

(See Disc)

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 it may execute in addition to, or instead of the main purpose inscribed upon that key.

Gate

Logical gates permit the passage of data when certain conditions are fulfilled. There are many different types of gate (OR, AND, XOR etc).

Graphics

The part of the screen display of the computer that is not related to the display of 'characters'. Graphics encompasses the drawing and plotting of lines, circles, patterns, etc.

Graphics character

A shape or pattern specially designed to be useful in creating images.

Graphics cursor

Similar to the text cursor, but addressing the graphics screen. An invisible concept on the 464/6128 - but nevertheless an indispensible facility for locating drawn graphics. Not to be confused with graphics characters which are still part of the 'character set', and are 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 print-out 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 (or HEX)

The number system with base 16. Hexadecimal numbers are signified in 464/6128 programming by the prefix & or &H. e.g. &FF =(decimal) 255. (See part 1 of the chapter entitled 'At your leisure....'.)

Hex file

An ASCII representation of a command or machine code file.

High-level language

Languages like BASIC, 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.

IEEE-488

One of the standard Interfaces for connecting devices to a microcomputer. Similar to - but not wholly compatible with - the Centronics parallel interface.

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 - e.g. dimensioning arrays, declaring variables to be integers, etc.

Input

Anything that enters the computer memory from its keyboard, disc 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's CPU. A single high level command may invoke a large number of elements from the CPU's instruction set.

Integer

A whole number with no decimal point.

Integer number

A number with no fractional part, i.e. 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 464/6128 interface is the keyboard (input), and the screen (output) - as well as the facility for the connection of user peripherals to the various sockets.

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. e.g It converts BASIC code as entered via the keyboard into the computer's 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 fro' 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 short form 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.

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, 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.

Logical device

The representation of a device that may be different to its physical form. For example the CP/M logical device `LST` may be assigned to the Centronics port or perhaps the VDU.

LOGO

The name of a programming language derived from the Greek word logos, which means word. Logo is designed to teach the fundamentals of computer programming.

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. Discs and tape are examples of 'bulk memory', although 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 functions, such as the screen, the disc 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 - including fibre optics. (See also 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 operation of the computer.

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'.

MSB

The Most Significant Bit of a binary number, i.e. the bit at the left end of the binary expression.

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 expression. Each of the hexadecimal digits in the expression &F6 represents 'one nibble'.

Noise

The 464/6128 sound facilities include the ability to inject a variable amount of random noise 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 464/6128, 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 of 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.

Operator

The part of an arithmetic expression that causes one number to operate on another, e.g. + - * / etc.

Output

Anything that comes from a computer as the result of some computational function.

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'.

Page zero

This refers to the region of memory in a CP/M environment between &0000 and &0100 which is used to hold vital system parameters.

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 464/6128 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 transferred many times more quickly using a parallel interface than it is through 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 at the specified location.

Peripheral

Printers, modems, joysticks, cassette units - anything that plugs into the computer to expand its capabilities.

Physical device

An actual device, consisting of hardware, that exists. Physical devices may be represented by logical devices.

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.

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 Digital Research's CP/M.

Printer

Any hardcopy method for printing out text.

Procedure

A series of expressions or program statements that dictate how to perform a particular task.

Program

A combination of instructions that cause the computer to execute a task. It 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. (See also EPROM)

Prompt

A short message or character sequence reminding the user that some type of input is expected. For example, the CP/M prompt is the > character.

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 UK or US 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 access

The ability to read and write information in memory or on a disc in any desired order.

Random number

A number that is generated by the computer program that is neither repeatable, nor predictable. The 464/6128 is capable of generating a pseudo random number sequence.

Raster

A system of 'writing' on the screen where the images are built from a number of horizontal scan lines. (Raster scan).

Read only R/O

An attribute assigned to a disc, a disc file or a disc drive that prevents writing or changing of data.

Read write R/W

An attribute assigned to a disc, a disc file or a disc drive that allows both reading and writing of data.

Real number

A number that has both integer and fractional parts. i.e. both sides of the decimal point are used.

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.

Record

A group of bytes in a file. CP/M uses 128 byte records.

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 statement in a program that is included to remind the programmer what part of the program he is doing, and 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 pre-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 arithmetic manipulations on large numbers.

Reverse Polish notation

(RPN) A method of describing arithmetic 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 aerial 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. e.g. 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 Centronics parallel interface where the interconnection is virtually standard everywhere.

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.

Sector

A block of data on a disc. The AMSTRAD disc system uses a sector size of 512 bytes.

Separator

A separator performs the same function as a delimiter, i.e. 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.

Single sided

Refers to a disc which has only one side available for data storage.

Soft key

(See UDK - user defined key)

Software

Programs themselves. May be based on disc, cassette, ROM, etc.

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 or 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 re-run, 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. e.g. the screen, the printer, or the disc.

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 as such unless specifically converted to a corresponding numeric variable by the appropriate command.

Structured programming

A logical and premeditated programming technique that results in programs that flow from 'top to bottom', with clearly described steps.

Sub-routine

(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.

System tracks

Tracks reserved on the disc for the CP/M system.

Terminal

A keyboard input device, with either a VDU screen or teletype typewriter output system.

TPA

Transient Program Area. An area in memory commencing at &0100 where CP/M user programs run and store data.

Track

Tracks are concentric rings on a disc. Each track holds a fixed number of sectors. The tracks and sectors are written to a specific area of a disc during formatting.

Transient program

A CP/M utility program such as PIP, which can be loaded into the TPA and run by typing its name at the keyboard.

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.

Turnkey

A word used to describe a program which executes automatically when the system is booted.

UDK

User defined keys. The 464/6128 has up to 32 keys which may be redefined to perform a variety of tasks.

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.

Utility program

A program on disc that enables the user to perform certain operations. (See Transient program)

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.

Warm start

This is performed when **[CTRL]C** is pressed during CP/M. A warm start reinitialises the disc sub-system and returns control to CP/M ready for commands to be entered.

Wildcard character

Either of the characters * or ?. The * wildcard character simply means any number of ?s. When referencing files, wildcard characters are used to make up an ambiguous file name. Any ?s in the file name refer to any alphabetic or numeric character.

Write protection

A safeguard used to prevent re-writing of a disc or disc file. A write protected disc or file is Read Only.

XYZY

A magic word to get out of sticky corners in adventure games.

Appendix 3
Some Programs For You....

Bustout

So simple yet so addictive! For one player against the computer. Keyboard or Joystick.

```
10 'BUSTOUT by ALEXANDER MARTIN
20 'copyright (c) AMSOFT 1984
30 '
40 MODE 1:BORDER 1:INK 0,1:INK 1,26:INK 2,24:INK 3,6
50 SPEED KEY 15,2
60 ENV 1,1,18,0,11,0,10
70 ENT 1,10,2,2
80 ENV 3,1,0,16,5,-3,2
90 ENV 2,5,3,3,1,-21,22,9,-3,2
100 ENT -2,10,2,2,5,-7,1,2,11,3,2,-4,8
110 '
120 '
130 MOVE 30,32:DRAWR 0,400,1:MOVE 610,32:DRAWR 0,400,1
140 PEN 3:LOCATE 3,1:PRINT STRING$(36,143)
150 PEN 2:LOCATE 3,2:PRINT STRING$(36,143)
160 PEN 1:FOR r=5 TO 6:LOCATE 3,r:PRINT STRING$(36,143)
    :NEXT r
170 bx=9
180 lives=5:score=0
190 PEN 1:GOSUB 680:CLEAR INPUT
200 IF INKEY$<>CHR$(32) AND JOY(0)<16 THEN 200
210 LOCATE 11,23:PRINT SPACE$(20):LOCATE 1,24:PRINT SPA
    CE$(40);
220 GOSUB 690:GOSUB 660:GOTO 280
230 '
240 '
250 LOCATE bx,24:PRINT"  ";STRING$(4,131);"  ":RETURN
260 '
270 '
280 xa=1:ya=1:IF INT(RND*2)=1 THEN xa=-xa
290 PEN 1:GOSUB 250
300 ORIGIN 0,400
310 x=bx+4:y=11:x1=x:y1=y
320 '
330 '
340 x1=x+xa:y1=y+ya
```

continued on the next page

```
350 IF x1=3 OR x1=38 THEN xa=-xa
360 GOSUB 540
370 IF y1=24 AND x1>bx+1 AND x1<bx+6 THEN ya=-ya:y1=y1-
    2:SOUND 130,44,8,7,1,1:a=((x>bx+5)OR(x<bx+2)):IF a=
    -1 THEN xa=xa*a:x1=x1+xa:y1=y1+1
380 IF y1=25 THEN LOCATE x,y:PRINT" ":GOTO 500
390 GOSUB 250
400 t=TEST((16*x1)-1,-(16*y1)-1)
410 IF t<>0 THEN ya=-ya:xz=x1:yz=y1:y1=y1+ya:GOSUB 590:
    IF t=2 THEN score=score+10:GOSUB 660
420 IF t=3 THEN score=score+20:GOSUB 660
430 IF t=1 THEN score=score+5:GOSUB 660
440 IF y1=1 THEN ya=1
450 LOCATE x,y:PRINT " ":LOCATE x1,y1:PRINT CHR$(233):x
    =x1:y=y1
460 IF y=1 OR x=3 OR x=38 THEN SOUND 129,78,8,7,1,1
470 GOTO 340
480 '
490 '
500 lives=lives-1:SOUND 132,19,46,12,2,2:IF lives=0 THE
    N GOTO 620
510 GOSUB 660:GOTO 280
520 '
530 '
540 IF (INKEY(8)=0 OR INKEY(74)=0) AND bx>2 THEN bx=bx-
    2:RETURN
550 IF (INKEY(1)=0 OR INKEY(75)=0) AND bx<32 THEN bx=bx
    +2:RETURN
560 RETURN
570 '
580 '
590 LOCATE xz,yz:PRINT " ":RETURN
600 '
610 '
620 IF score>=hiscore THEN hiscore=score
630 GOSUB 660:score=0:lives=5:GOTO 130
640 '
650 '
660 SOUND 130,0,20,13,3,0,31:LOCATE 1,25:PRINT TAB(4)"H
    ISCORE";hiscore;
670 LOCATE 18,25:PRINT"SCORE";score:LOCATE 30,25:PRINT"
    LIVES";lives:RETURN
680 LOCATE 11,23:PRINT"PRESS SPACE TO START":RETURN
690 LOCATE 1,25:PRINT SPACE$(40);:RETURN
```

Bomber

A variation on a classic theme! For one player against the computer. Keyboard only.

```
10 'BOMBER by DAVE TOWN
20 'copyright (c) AMSOFT 1984
30 '
40 MODE 1:CLS:INK 0,0:BORDER 0:INK 1,18:INK 2,6:INK 3,4
   :INK 5,15:INK 6,2:INK 7,24:INK 8,8:INK 9,26:INK 10,1
   0:INK 11,20:INK 12,12:INK 13,16:INK 14,14:INK 15,21
50 SYMBOL AFTER 240:SYMBOL 241,&40,&60,&70,&7F,&7F,&3F,
   &7,&0:SYMBOL 242,&0,&32,&7A,&FE,&FA,&F2,&E0,&0
60 score=0:hiscore=0:plane$=CHR$(241)+CHR$(242):x=2:y=2
   :drop=0:a=2:b=2
70 GOSUB 480
80 CLS
90 PEN 2:LOCATE 1,15:INPUT"Enter skill : 0 (ACE) to 5 (
   NOVICE) : ",skill
100 IF skill<0 OR skill>5 GOTO 90
110 skill=skill+10
120 LOCATE 1,15:PRINT CHR$(18);:LOCATE 1,15:INPUT"Enter
     speed 0 (FAST) to 100 (SLOW) : ",rate
130 IF rate>100 OR rate<0 GOTO 120
140 '
150 'Buildings
160 '
170 MODE 0:FOR base=5 TO 15:FOR height=21 TO INT(RND(1)
    *8+skill) STEP -1:LOCATE base,height:PEN base-2:PRI
    NT CHR$(143)+CHR$(8)+CHR$(11)+CHR$(244);:NEXT :NEXT
180 PLOT 0,20,4:DRAW 640,20,4
190 LOCATE 1,25:PEN 2:PRINT"SCORE";score;:LOCATE 13,25:
    PRINT"HI";hiscore;
200 '
210 'Main Game
220 '
230 LOCATE x-1,y:PRINT"   ";
240 PEN 1:LOCATE x,y:PRINT plane$;:PEN 2
250 IF y=21 AND x=15 THEN GOTO 290:ELSE GOTO 340
260 '
270 'Landed
280 '
290 FOR c=0 TO 1000:NEXT
```

continued on next page

```
300 score=score+100-(skill*2):skill=skill-1:x=2:y=2:a=2
    :b=2:drop=0
310 IF skill<10 THEN skill=10:rate=rate-20
320 IF rate<0 THEN rate=0
330 GOTO 150
340 FOR c=0 TO rate:NEXT
350 x=x+1
360 IF x=18 THEN LOCATE x-1,y:PRINT CHR$(18);:x=2:y=y+1
    :LOCATE x,y:PEN 1:PRINT plane$;:PEN 2
370 a$=INKEY$:IF a$=" " AND drop=0 THEN drop=1:b=y+2:a=
    x
380 IF y=21 THEN drop=0
390 IF drop=1 THEN LOCATE a,b:PRINT CHR$(252);:LOCATE a
    ,b-1:PRINT" ";:b=b+1:IF b>21 THEN LOCATE a,b:PRINT"
     ";:LOCATE a,b-1:PRINT" ";:a=0:b=0:drop=0:SOUND 3,4
    000,10,12,0,0,10
400 ga=(a-0.5)*32:gb=400-(b*16):bomb=TEST(ga,gb)
410 IF bomb>0 THEN GOTO 670
420 gx=((x+1.5)*32):gy=408-(y*16):crash=TEST(gx,gy)
430 IF crash>0 GOTO 570
440 GOTO 230
450 '
460 'Instructions
470 '
480 LOCATE 1,2:PEN 1:PRINT"You are piloting an aircraft
     over a des-erted city and must clear the buildings
     in order to land and refuel.  Your air- craft move
    s across  the screen from left to right.";:PRINT
490 PRINT:PRINT"On reaching the right, the aircraft
     returns to the left A LINE FURTHER DOWN.You have a
    n unlimited supply of bombs   and you can drop them
     on the buildings  below by pressing the SPACE BAR.
    ";:PRINT
500 PRINT:PRINT"Each time you land, the height of the
     buildings or the speed of your aircraft increases.
    ";:PRINT:PRINT:PRINT"ONCE YOU HAVE RELEASED A BOMB,
     YOU WILL NOT BE ABLE TO RELEASE ANOTHER UNTIL THE
    FIRST HAS EXPLODED !!!!";
510 PEN 2:LOCATE 1,24:PRINT:PRINT"Press any key to star
    t.";
520 a$=INKEY$:IF a$="" GOTO 520
530 RETURN
540 '
550 'Collision
```

continued on the next page

```
560 '
570 LOCATE x-1,y:PRINT CHR$(32)+CHR$(32)+CHR$(32)+CHR$(
    253)+CHR$(8)+CHR$(238)+CHR$(8);
580 FOR t=1 TO 10:SOUND 7,4000,5,15,0,0,5:PEN t:PRINT C
    HR$(253)+CHR$(8)+CHR$(238)+CHR$(8)+CHR$(32)+CHR$(8)
    ;:FOR tm=0 TO 50:NEXT:NEXT:PEN 2
590 CLS:LOCATE 1,5:PRINT"You scored";score;
600 IF score>hiscore THEN hiscore=score:LOCATE 1,8:PRIN
    T"TOP SCORE!!";
610 score=0:LOCATE 1,12:PRINT"Press R to restart";
620 a$=INKEY$:IF a$="r" OR a$="R" GOTO 630 ELSE GOTO 62
    0
630 PEN 1:MODE 1:x=2:y=2:a=2:b=2:GOTO 90
640 '
650 'Bombed building
660 '
670 LOCATE a,b-1:PRINT" "+CHR$(8);:PEN 4:FOR tr=1 TO IN
    T(RND(1)*3)+1:score=score+5:SOUND 3,4000,10,12,0,0,
    10:LOCATE a,b:FOR t=0 TO 4:PRINT CHR$(253)+CHR$(8)+
    CHR$(32)+CHR$(8);:NEXT:b=b+1
680 IF b=24 THEN b=b-1
690 NEXT
700 LOCATE 6,25:PRINT score;:drop=0:a=x:b=y:GOTO 230
```

Telly tennis

The one that started it all, but still great fun! For two players, or one player against the computer. Keyboard or Joystick(s).

```
10 'TELLY TENNIS by DAVID RADISIC
20 'copyright (c) AMSOFT 1985
30 '
40 DEFINT a-z
50 comp=1
60 ENV 1,=11,20,=9,5000
70 MODE 1:INK 0,10:BORDER 10:INK 1,26:INK 2,18:INK 3,0
80 GOSUB 710
90 GOSUB 150
100 GOSUB 330
110 GOSUB 420
120 LOCATE 13,1:PRINT USING"#### ";score1;
130 LOCATE 35,1:PRINT USING"#### ";score2;
140 GOTO 100
150 PEN 2
160 x(1)=3:y(1)=5
170 x(2)=37:y(2)=22
180 edge$=CHR$(233):edge2$=STRING$(2,207)
190 LOCATE 1,3:PRINT STRING$(39,edge$):PRINT STRING$(39
    ,edge$)
200 FOR i=1 TO 19
210 PRINT edge2$;TAB(38);edge2$
220 NEXT
230 PRINT STRING$(39,edge$):PRINT STRING$(39,edge$);
240 WINDOW #1,3,37,5,23
250 CLS#1
260 SYMBOL 240,0,60,126,126,126,126,60,0
270 bat$="I"+CHR$(8)+CHR$(10)+"I"
280 clr$=" "+CHR$(8)+CHR$(10)+" "
290 ball$=CHR$(240)
300 PEN 3
310 LOCATE 2,1:PRINT"Player 1 :    0";:LOCATE 24,1:PRIN
    T"Player 2 :    0";
320 RETURN
330 n=INT(RND*2):CLS #1:scored=0
340 PEN 3
350 FOR i=1 TO 2:LOCATE x(i),y(i):PRINT bat$;:NEXT
360 ON n GOTO 390
```

continued on the next page

```
370 xb=21:dx=1
380 GOTO 400
390 xb=19:dx=-1
400 yb=12:dy=INT(RND*3)-1
410 RETURN
420 GOSUB 600
430 oxb=xb:oyb=yb
440 GOSUB 500
450 IF note>0 THEN SOUND 129,note,50,15,1
460 LOCATE oxb,oyb:PRINT" ";
470 LOCATE xb,yb:PRINT ball$
480 IF scored=0 THEN 420
490 RETURN
500 LOCATE xb+dx,yb+dy:ch$=COPYCHR$(#0)
510 note=0
520 IF ch$=" " THEN xb=xb+dx:yb=yb+dy:RETURN
530 IF ch$="I" THEN dx=2-dx-2:dy=INT(RND*3)-1:note=200:
    RETURN
540 IF ch$=LEFT$(edge2$,1) THEN 570
550 IF ch$=edge$ THEN dy=2-dy-2:note=250
560 RETURN
570 IF dx>0 THEN score1=score1+1 ELSE score2=score2+1
580 scored=1:note=2000
590 RETURN
600 p(1)=(INKEY(69)>=0)+(INKEY(72)>=0)+ABS((INKEY(71)>=
    0)+(INKEY(73)>=0))*2
610 IF comp=1 THEN p(2)=ABS(y(2)<yb)*2+(y(2)>yb):GOTO 6
    30
620 p(2)=(INKEY(4)>=0)+(INKEY(48)>=0)+ABS((INKEY(5)>=0)
    +(INKEY(49)>=0))*2
630 PEN 3
640 FOR i=1 TO 2
650 LOCATE x(i),y(i)+p(i):ch$=COPYCHR$(#0)
660 IF ch$=" " THEN LOCATE x(i),y(i):PRINT clr$;:y(i)=y
    (i)+ROUND(p(i)/2)
670 LOCATE x(i),y(i):PRINT bat$;
680 NEXT
690 PEN 1
700 RETURN
710 PEN 2:PRINT:PRINT TAB(15)"Ping-Pong":PRINT TAB(15)"
    ---------"
720 PEN 3:PRINT:PRINT TAB(14)"To move bats :
730 PRINT:PRINT:PEN 1
740 PRINT" Player 1       Player 2       Direction":PRI
    NT
```

continued on the next page

```
750 PRINT"     A                6                UP"
760 PRINT"     Z                3                DOWN":PRINT
770 PEN 3:PRINT:PRINT TAB(14)"Or joysticks"
780 PRINT:PRINT:PRINT:PRINT
790 PEN 2
800 PRINT TAB(6)"Select <1> or <2> players"
810 i$=INKEY$:IF i$<>"1" AND i$<>"2" THEN 810
820 IF i$="1" THEN comp=1 ELSE comp=0
830 MODE 1:RETURN
```

Electric fencing

Try to 'foil' your opponent! For two players only. Keyboard or Joysticks.

```
10 'ELECTRIC FENCING by ALEXANDER MARTIN
20 'copyright (c) AMSOFT 1985
30 '
40 DEFINT a-z
50 MODE 0
60 GOSUB 980
70 GOSUB 1370
80 GOSUB 270
90 GOSUB 1520
100 GOSUB 1370
110 GOSUB 1270
120 '
130 '
140 REM start
150 IF finished THEN GOTO 100
160 GOSUB 240
170 FRAME:IF p1dir THEN GOSUB 570 ELSE FRAME:FRAME
180 FRAME:IF p2dir THEN GOSUB 620 ELSE FRAME:FRAME
190 IF p1sa=-1 THEN GOSUB 670
200 IF p2sa=-1 THEN GOSUB 720
210 GOTO 140
220 '
230 '
240 IF j THEN  380 ELSE  480
250 '
260 '
270 CLS:PEN 6
280 PRINT:PRINT"   CHOOSE CONTROL"
290 PRINT:PRINT:PRINT:PRINT"press J/K then ENTER"
300 LOCATE 4,10:PRINT"JOYSTICK";TAB(5);"OR KEYS"
310 LOCATE 12,10:IF j THEN PRINT"*":ELSE PRINT" "
320 LOCATE 12,11:IF j THEN PRINT" ":ELSE PRINT"*"
330 IF NOT(INKEY(45)) THEN j=-1
340 IF NOT(INKEY(37)) THEN j=0
350 IF NOT(INKEY(18)) THEN RETURN ELSE 310
360 '
370 '
380 p1=JOY(0):p2=JOY(1)
390 p1dir=(p1 AND 1)*-1+(p1 AND 2)*0.5
```

continued on the next page

```
400 p2dir=(p2 AND 1)*-1+(p2 AND 2)*0.5
410 IF P1 AND 16 THEN p1sa=p1sa-1:IF p1sa=-1 THEN AFTER
     15 GOSUB 770
420 IF P2 AND 16 THEN p2sa=p2sa-1:IF p2sa=-1 THEN AFTER
     15 GOSUB 770
430 IF p1sa THEN p1dir=0
440 IF p2sa THEN p2dir=0
450 RETURN
460 '
470 '
480 p2dir=((INKEY(4)=0)*1)+((INKEY(5)=0)*-1)
490 p1dir=((INKEY(69)=0)*1)+((INKEY(71)=0)*-1)
500 IF INKEY(63)=0 THEN p1sa=p1sa-1:IF p1sa=-1 THEN AFT
    ER 15 GOSUB 770
510 IF INKEY(10)=0 THEN p2sa=p2sa-1:IF p2sa=-1 THEN AFT
    ER 15 GOSUB 770
520 IF p1sa THEN p1dir=0
530 IF p2sa THEN p2dir=0
540 RETURN
550 '
560 '
570 pt=p1wp+p1dir:IF pt>25 OR pt<6 THEN RETURN ELSE p1w
    p=pt
580 p1dir=0
590 PEN 1:LOCATE 3,p1wp:CLS #3:PRINT CHR$(209);:RETURN
600 '
610 '
620 pt=p2wp+p2dir:IF pt>25 OR pt<6 THEN RETURN ELSE p2w
    p=pt
630 p2dir=0
640 PEN 2:LOCATE 18,p2wp:CLS #5:PRINT CHR$(211);:RETURN
650 '
660 '
670 PAPER #4,4:WINDOW #4,4,17,p1wp,p1wp:CLS#4:FRAME:FRA
    ME
680 PAPER #4,0:CLS#4
690 GOTO 570
700 '
710 '
720 PAPER #6,5:WINDOW #6,4,17,p2wp,p2wp:CLS#6:FRAME:FRA
    ME
730 PAPER #6,0:CLS#6
740 GOTO 620
750 '
```

continued on the next page

```
760 '
770 pwpe=(p1wp=p2wp):IF p1sa AND NOT(p2sa) AND pwpe THE
    N p1sc=p1sc+1:SOUND 132,120,10,0,1,0:PRINT#1,a$(p1s
    c);:IF p1sc=9 THEN 860
780 IF p2sa AND NOT(p1sa) AND pwpe THEN p2sc=p2sc+1:SOU
    ND 132,100,10,0,1,0:PRINT#2,a$(p2sc);:IF p2sc=9 THE
    N 860
790 IF p1sa THEN SOUND 132,40,70,0,1,1
800 IF p2sa THEN SOUND 132,56,70,0,1,1
810 p1sa=0
820 p2sa=0
830 RETURN
840 '
850 '
860 PEN 6
870 LOCATE 6,10:PRINT"GAME OVER"
880 IF p1sc=9 THEN INK 1,2,20:INK 2,0 ELSE INK 2,6,17:I
    NK 1,0
890 SOUND 129,1000,0,12,3:SOUND 130,900,0,12,3
900 WHILE INKEY$<>"":WEND
910 t!=TIME:WHILE t!+2000>TIME:WEND
920 WHILE INKEY$="":WEND
930 CLS
940 finished=-1
950 RETURN
960 '
970 '
980 a$(0)="1111011011101111"
990 a$(1)="0010010010010001"
1000 a$(2)="1110011111100111"
1010 a$(3)="1110011111001111"
1020 a$(4)="1001001011111001"
1030 a$(5)="1111001111001111"
1040 a$(6)="1111001111101111"
1050 a$(7)="1110010010100010"
1060 a$(8)="1111011111101111"
1070 a$(9)="1111011111001001"
1080 FOR n=0 TO 9
1090 howlong=LEN(a$(n))
1100 FOR n2=1 TO howlong
1110 IF MID$(a$(n),n2,1)="1"THEN MID$(a$(n),n2,1)=CHR$(
     143)ELSE MID$(a$(n),n2,1)=CHR$(32)
1120 NEXT n2,n
1130 '
```

continued on the next page

```
1140 '
1150 b$="ELECTRIC FENCING"
1160 c$=CHR$(32)+CHR$(164)+" Alexander Martin"
1170 ENV 1,=9,2000:ENT -1,6,3,1
1180 ENV 2,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0
1190 ENV 3,=9,9000
1200 '
1210 '
1220 BORDER 0
1230 INK 0,12:PEN #4,1:PEN #6,2:PEN #1,1:PEN #2,2:PAPER
     #1,3:PAPER #2,3:PEN #0,6
1240 RETURN 'FROM SETTING UP CONSTANTS
1250 '
1260 '
1270 INK 0,12:INK 1,2:INK 2,6:INK 3,13:INK 4,20:INK 5,1
     7:INK 6,20
1280 WINDOW #3,3,3,6,25:WINDOW #5,18,18,6,25
1290 WINDOW #1,3,5,1,5:WINDOW #2,16,18,1,5:WINDOW #7,1,
     20,1,5:PAPER #7,3
1300 CLS:CLS#7:PRINT#1,a$(0);:PRINT#2,a$(0);:p1sc=0:p2s
     c=0:p1wp=5:p2wp=24:p1dir=1:p2dir=1
1310 GOSUB 570:GOSUB 620
1320 SOUND 1,1000,0,12,2:SOUND 2,900,0,12,2
1330 p1sa=0:p2sa=0:finished=0
1340 RETURN 'FROM GAME SHEET RESTORE
1350 '
1360 '
1370 CLS
1380 PEN 7
1390 FOR n=1 TO LEN(b$)
1400 LOCATE 2+n,10
1410 FOR n2=LEN(b$) TO n STEP-1
1420 PRINT MID$(b$,n2,1)
1430 LOCATE 2+n,10
1440 SOUND 135,20*n2,5,12,2,1
1450 NEXT n2,n
1460 SOUND 135,100,0,13,3,1,20
1470 PEN 6:PRINT:PRINT:PRINT:PRINT c$
1480 FOR n=1 TO 5000:NEXT
1490 RETURN
1500 '
1510 '
1520 IF j THEN RETURN
1530 CLS
```

continued on the next page

```
1540 LOCATE 7,5
1550 PRINT"CONTROLS"
1560 PRINT
1570 PRINT"  PLAYER1  PLAYER2"
1580 PRINT
1590 PRINT"  A       up       6"
1600 PRINT"  Z      down      3"
1610 PRINT"  X      fire      7"
1620 t!=TIME:WHILE t!+1000>TIME:WEND
1630 RETURN
```

Amthello

The thinking person's game. Try to surround and capture your opponent's squares without leaving your own squares open to capture! For one player against the computer. Keyboard only.

```
10 'AMTHELLO by M.J.GRIBBINS
20 'copyright (c) AMSOFT 1984
30 '
40 BORDER 14
50 CLEAR
60 MODE 1:PEN 0:PAPER 1:CLS
70 INK 0,0:INK 1,14:INK 2,18:INK 3,26
80 LOCATE 2,3:PEN 3:PRINT"A":LOCATE 3,4:PRINT"M":LOCATE
    4,5:PRINT"T":LOCATE 5,6:PRINT"H"
90 LOCATE 6,7:PRINT"E":LOCATE 7,8:PRINT"L":LOCATE 8,9:P
   RINT"L":LOCATE 9,10:PRINT"O"
100 WINDOW #1,2,39,22,25:PAPER #1,1:PEN #1,0:CLS #1
110 PEN 0
120 LOCATE #1,8,1:PRINT #1,"BLACK ALWAYS PLAYS FIRST"
130 LOCATE #1,1,3:PRINT #1,"PRESS B OR W TO CHOOSE BLAC
    K OR WHITE"
140 B$=INKEY$:IF B$="" THEN 140
150 IF B$="W" OR  B$="w" THEN Q%=3:N%=0:GOTO 210
160 IF B$="B" OR  B$="b" THEN Q%=0:N%=3:GOTO 210
170 CLS #1:LOCATE #1,4,3
180 PRINT #1,"       BLACK OR WHITE ONLY"
190 FOR T=0 TO 1000:NEXT T
200 GOTO 140
210 DIM C%(10,10),P%(9,9),C1%(8),C2%(8),CX%(9),CY%(9)
220 I1%=2:J1%=2:I2%=7:J2%=7
230 FOR I%=0 TO 9
240 C%(I%,0%)=6:C%(0,I%)=6
250 C%(9,I%)=6:C%(I%,9)=6
260 NEXT I%
270 FOR I%=1 TO 8
280 READ C1%(I%),C2%(I%)
290 FOR J%= 1 TO 8
300 READ P%(I%,J%)
310 C%(I%,J%)=6
320 NEXT J%:NEXT I%
330 C%(4,4)=3:C%(4,5)=0:C%(5,4)=0:C%(5,5)=3
```

continued on the next page

```
340 FOR K%=1 TO 58
350 READ AR%,BR%,CR%,DR%
360 PLOT AR%,BR%:DRAW CR%,DR%,0
370 NEXT K%
380 GOSUB 1460
390 IF Q%=3 GOTO 770
400 CLS #1:INPUT #1,"  WHICH LINE DO YOU WANT ";E%
410 IF E% <1 OR E% >8 GOTO 400
420 LOCATE #1,1,3:INPUT #1,"WHICH COLUMN DO YOU WANT ";
    D%
430 IF D% <1 OR D% >8 GOTO 420
440 IF C%(D%,E%)=6 GOTO 480
450 CLS #1:LOCATE #1,5,2:PRINT #1,"THAT SQUARE IS ALREA
    DY OCCUPIED !"
460 FOR T=1 TO 1000:NEXT T
470 GOTO 400
480 PLOT 270+(30*D%),70+(30*E%):DRAW 290+(30*D%),89+(30
    *E%),Q%
490 PLOT 290+(30*D%),70+(30*E%):DRAW 270+(30*D%),89+(30
    *E%),Q%
500 GOTO 540
510 FOR M%= 0 TO 19 STEP 2:PLOT 270+(30*D%),70+M%+(30*E
    %)
520 DRAW 290+(30*D%),70+M%+(30*E%),6:NEXT M%
530 GOTO 400
540 VRX%=0
550 FOR K%=1 TO 8
560 VR%=0:C3%=D%:C4%=E%
570 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
580 IF C%(C3%,C4%)=N% GOTO 590 ELSE 600
590 VR%=VR%+1:GOTO 570
600 IF C%(C3%,C4%)=6 GOTO 610 ELSE 620
610 NEXT K%:GOTO 670
620 IF VR%=0 GOTO 610 ELSE 630
630 VRX%=VRX%+VR%
640 C3%=C3%-C1%(K%):C4%=C4%-C2%(K%)
650 IF C%(C3%,C4%)=6 GOTO 610 ELSE 660
660 C%(C3%,C4%)=Q%:GOTO 640
670 IF VRX%=0 GOTO 680 ELSE 710
680 CLS #1:PRINT #1," THIS IS NOT A POSSIBLE CHOICE"
690 FOR T=1 TO 1000:NEXT T
700 GOTO 510
710 E%=E%:D%=D%:VRX%=VRX%
720 CLS #1:PRINT #1,"YOU HAVE PLAYED LINE NUMBER ";E%
```

continued on the next page

```
730 PRINT #1,"           AND COLUMN NUMBER ";D%
740 LOCATE #1,2,4:PRINT #1,"THAT GIVES YOU ";VRX%;" SQU
    ARES(S)"
750 C%(D%,E%)=Q%:GOSUB 1710
760 GOSUB 1460
770 CLS #1:LOCATE #1,10,2:PRINT #1,"NOW IT'S MY TURN ..
    .!"
780 P%=0:VRX%=0:VRY%=0
790 IF I1%*J1%=1 AND I2%*J2%=64 GOTO 860
800 FOR K%=2 TO 7
810 IF C%(2,K%) <> 6 THEN I1%=1
820 IF C%(7,K%) <> 6 THEN I2%=8
830 IF C%(K%,2) <> 6 THEN J1%=1
840 IF C%(K%,7) <> 6 THEN J2%=8
850 NEXT K%
860 FOR I%=I1% TO I2%
870 FOR J%=J1% TO J2%
880 IF C%(I%,J%)=6 GOTO 1030
890 NEXT J%:NEXT I%
900 IF P% > 0 GOTO 1000
910 IF PAS%=1 GOTO 920 ELSE 940
920 CLS #1:PRINT #1," DEADLOCK ! I MUST PASS ALSO.GAME
    OVER"
930 FOR T=1 TO 1000:NEXT T:GOTO 1550
940 CLS #1:LOCATE #1,18,2:PRINT #1,"I MUST PASS"
950 GOSUB 2720
960 IF PAS%=1 GOTO 970 ELSE 990
970 CLS #1:PRINT #1,"DEADLOCK! YOU MUST PASS ALSO.GAME
    OVER"
980 FOR T=1 TO 1000:NEXT T:GOTO 1550
990 GOTO 400
1000 IF LC%=0 THEN LC%=1:RANDOMIZE LC%:RL%=RND(LC%)
1010 CX1%=CX%(RL%):CX2%=CY%(RL%)
1020 GOTO 1220
1030 VRX%=0
1040 FOR K%=1 TO 8
1050 VR%=0:C3%=I%:C4%=J%
1060 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
1070 IF C%(C3%,C4%)=Q% GOTO 1080 ELSE 1090
1080 VR%=VR%+1:GOTO 1060
1090 IF C%(C3%,C4%)=6 GOTO 1100 ELSE 1110
1100 NEXT K%:GOTO 1130
1110 IF VR%=0% GOTO 1100 ELSE 1120
1120 VRX%=VRX%+VR%:GOTO 1100
```

continued on the next page

```
1130 IF VRX%=0 GOTO 890
1140 IF P%(I%,J%) < P% GOTO 890
1150 IF P%(I%,J%) > P% GOTO 1160 ELSE 1170
1160 P%=P%(I%,J%):VRY%=VRX%:LC%=0:CX%(0)=I%:CY%(0)=J%:G
     OTO 890
1170 IF VRY% > VRX% GOTO 890
1180 IF VRY% < VRX% GOTO 1190 ELSE 1200
1190 LC%=0:VRY%=VRX%:CX%(0)=I%:CY%(0)=J%:GOTO 890
1200 LC%=LC%+1:CX%(LC%)=I%:CY%(LC%)=J%
1210 GOTO 890
1220 CX2%=CX2%:CX1%=CX1%:VRY%=VRY%
1230 CLS #1:PRINT #1," I CHOOSE LINE NUMBER ";CX2%
1240 PRINT #1,"     AND COLUMN NUMBER ";CX1%
1250 LOCATE #1,1,4:PRINT #1,"THAT GIVES ME ";VRY%;" SQU
     ARE(S)"
1260 PLOT 270+(30*CX1%),70+(30*CX2%):DRAW 290+(30*CX1%)
     ,89+(30*CX2%),N%
1270 PLOT 290+(30*CX1%),70+(30*CX2%):DRAW 270+(30*CX1%)
     ,89+(30*CX2%),N%
1280 FOR T=1 TO 1000:NEXT T
1290 FOR K%=1 TO 8
1300 VR%=0:C3%=CX1%:C4%=CX2%
1310 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
1320 IF C%(C3%,C4%)=Q% GOTO 1330 ELSE 1340
1330 VR%=VR%+1:GOTO 1310
1340 IF C%(C3%,C4%)=6 GOTO 1350 ELSE 1360
1350 NEXT K%:GOTO 1400
1360 IF VR%=0 GOTO 1350
1370 C3%=C3%-C1%(K%):C4%=C4%-C2%(K%)
1380 IF C%(C3%,C4%)=6 GOTO 1350
1390 C%(C3%,C4%)=N%:GOTO 1370
1400 C%(CX1%,CX2%)=N%
1410 GOSUB 2720
1420 GOSUB 1460
1430 IF PAS%=1 GOTO 1440 ELSE 1450
1440 CLS #1:PRINT #1,"      YOU MUST PASS":FOR T=1 TO 10
     00:NEXT T:GOTO 770
1450 GOTO 400
1460 FOR I%=1 TO 8
1470 FOR J%=1 TO 8
1480 FOR M%=0 TO 19 STEP 2
1490 Z%=270+(30*I%):H%=70+(30*J%):W%=H%+M%
1500 PLOT Z%,W%:DRAW Z%+20,W%,C%(I%,J%)
1510 NEXT M%:NEXT J%:NEXT I%
```

continued on the next page

```
1520 X%=X%+1
1530 IF X%=61 GOTO 1550
1540 RETURN
1550 CQ%=0:CN%=0
1560 FOR I%=1 TO 8
1570 FOR J%=1 TO 8
1580 IF C%(I%,J%)=Q% THEN CQ%=CQ%+1
1590 IF C%(I%,J%)=N% THEN CN%=CN%+1
1600 NEXT J%:NEXT I%
1610 IF CQ% > CN% GOTO 1680
1620 IF CQ%=CN% GOTO 1630 ELSE 1650
1630 CLS #1:LOCATE #1,25,2:PRINT #1,"DEADLOCK"
1640 END
1650 CLS #1:LOCATE #1,5,1:PRINT #1,"YOU HAVE ";CQ%;" SQ
     UARES;I HAVE ";CN%
1660 LOCATE #1,11,3:PRINT #1,"I HAVE WON....!!!!"
1670 END
1680 CLS #1:LOCATE #1,5,1:PRINT #1,"YOU HAVE ";CQ%;" SQ
     UARES;I HAVE ";CN%
1690 LOCATE #1,5,3:PRINT #1,"WELL DONE. YOU HAVE WON !!
     "
1700 END
1710 IF C%(2,2)=Q% AND (C%(3,1)=N% OR C%(1,3)=N%) GOTO
     1720 ELSE 1730
1720 P%(3,1)=1:P%(1,3)=1
1730 IF C%(7,7)=Q% AND (C%(8,6)=N% OR C%(6,8)=N%) GOTO
     1740 ELSE 1750
1740 P%(8,6)=1:P%(6,8)=1
1750 IF C%(2,7)=Q% AND (C%(1,6)=N% OR C%(3,8)=N%) GOTO
     1760 ELSE 1770
1760 P%(1,6)=1:P%(3,8)=1
1770 IF C%(7,2)=Q% AND (C%(6,1)=N% OR C%(8,3)=N%) GOTO
     1780 ELSE 1790
1780 P%(6,1)=1:P%(8,3)=1
1790 IF D%=1 OR D%=8 OR E%=1 OR E%=8 GOTO 1820
1800 IF CX1%=1 OR CX1%=8 OR CX2%=1 OR CX2%=8 GOTO 1820
1810 RETURN
1820 FOR J%=1 TO 8 STEP 7
1830 FOR I%=2 TO 7
1840 IF C%(I%,J%)=N% GOTO 1850 ELSE 1860
1850 P%(I%+1,J%)=21:P%(I%-1,J%)=21
1860 IF C%(J%,I%)=N% GOTO 1870 ELSE 1880
1870 P%(J%,I%+1)=21:P%(J%,I%-1)=21
1880 NEXT I%
```

continued on the next page

```
1890 FOR I%=2 TO 7
1900 IF C%(I%,J%)=Q% GOTO 1910 ELSE 1920
1910 P%(I%+1,J%)=2:P%(I%-1,J%)=2
1920 IF C%(J%,I%)=Q% GOTO 1930 ELSE 1940
1930 P%(J%,I%+1)=2:P%(J%,I%-1)=2
1940 NEXT I%:NEXT J%
1950 P%(1,2)=1:P%(1,7)=1:P%(2,1)=1:P%(7,1)=1
1960 P%(2,8)=1:P%(7,8)=1:P%(8,2)=1:P%(8,7)=1
1970  FOR I%=2 TO 7
1980 IF C%(1,I%-1)=Q% AND C%(1,I%+1)=Q% THEN P%(1,I%)=2
     5
1990 IF C%(8,I%-1)=Q% AND C%(8,I%+1)=Q% THEN P%(8,I%)=2
     5
2000 IF C%(I%-1,1)=Q% AND C%(I%+1,1)=Q% THEN P%(I%,1)=2
     5
2010 IF C%(I%-1,8)=Q% AND C%(I%+1,8)=Q% THEN P%(I%,8)=2
     5
2020 NEXT I%
2030 FOR J%=1 TO 8 STEP 7
2040 FOR I%=4 TO 8
2050 IF C%(J%,I%) <> N% GOTO 2140
2060 IC%=I%-1:IF C%(J%,IC%)=6 GOTO 2140
2070 IF C%(J%,IC%)=Q% GOTO 2080 ELSE 2090
2080 IC%=IC%-1:GOTO 2070
2090 IF C%(J%,IC%)=6 GOTO 2110
2100 GOTO 2140
2110 IF IC%=0 GOTO 2140
2120 IF C%(J%,I%+1)=Q% AND C%(J%,IC%-1)=6 GOTO 2140
2130 P%(J%,IC%)=26
2140 IF C%(I%,J%) <> N% GOTO 2230
2150 IC%=I%-1:IF C%(IC%,J%)=6 GOTO 2230
2160 IF C%(IC%,J%)=Q% GOTO 2170 ELSE 2180
2170 IC%=IC%-1:GOTO 2160
2180 IF C%(IC%,J%)=6 GOTO 2200
2190 GOTO 2230
2200 IF IC%=0 GOTO 2230
2210 IF C%(I%+1,J%)=Q% AND C%(IC%-1,J%)=6 GOTO 2230
2220 P%(IC%,J%)=26
2230 NEXT I%
2240 FOR I%=1 TO 5
2250 IF C%(J%,I%) <> N% GOTO 2340
2260 IC%=I%+1:IF C%(J%,IC%)=6 GOTO 2340
2270 IF C%(J%,IC%)=Q% GOTO 2280 ELSE 2290
2280 IC%=IC%+1:GOTO 2270
```

continued on the next page

```
2290 IF C%(J%,IC%)=6 GOTO 2310
2300 GOTO 2340
2310 IF IC%=9 GOTO 2340
2320 IF C%(J%,I%-1)=Q% AND C%(J%,IC%+1)=6 GOTO 2340
2330 P%(J%,IC%)=26
2340 IF C%(I%,J%) <> N% GOTO 2430
2350 IC%=I%+1:IF C%(IC%,J%)=6 GOTO 2430
2360 IF C%(IC%,J%)=Q% GOTO 2370 ELSE 2380
2370 IC%=IC%+1:GOTO 2360
2380 IF C%(IC%,J%)=6 GOTO 2400
2390 GOTO 2430
2400 IF IC%=9 GOTO 2430
2410 IF C%(I%-1,J%)=Q% AND C%(IC%+1,J%)=6 GOTO 2430
2420 P%(IC%,J%)=26
2430 NEXT I%:NEXT J%
2440 IF C%(1,1)=N% GOTO 2450 ELSE 2460
2450 FOR I%=2 TO 6:P%(1,I%)=20:P%(I%,1)=20:NEXT I%
2460 IF C%(1,8)=N% GOTO 2470 ELSE 2480
2470 FOR I%=2 TO 6:P%(I%,8)=20:P%(1,9-I%)=20:NEXT I%
2480 IF C%(8,1)=N% GOTO 2490 ELSE 2500
2490 FOR I%=2 TO 6:P%(9-I%,1)=20:P%(8,I%)=20:NEXT I%
2500 IF C%(8,8)=N% GOTO 2510 ELSE 2520
2510 FOR I%=3 TO 7:P%(I%,8)=20:P%(8,I%)=20:NEXT I%
2520 IF C%(1,1) <> 6 THEN P%(2,2)=5
2530 IF C%(1,8) <> 6 THEN P%(2,7)=5
2540 IF C%(8,1) <> 6 THEN P%(7,2)=5
2550 IF C%(8,8) <> 6 THEN P%(7,7)=5
2560 P%(1,1)=30:P%(1,8)=30:P%(8,1)=30:P%(8,8)=30
2570 FOR I%=3 TO 6
2580 IF C%(1,I%)=N% THEN P%(2,I%)=4
2590 IF C%(8,I%)=N% THEN P%(7,I%)=4
2600 IF C%(I%,1)=N% THEN P%(I%,2)=4
2610 IF C%(I%,8)=N% THEN P%(I%,7)=4
2620 NEXT I%
2630 IF C%(7,1)=Q% AND C%(4,1)=N% AND C%(6,1)=6 AND C%(
     5,1)=6 THEN P%(6,1)=26
2640 IF C%(1,7)=Q% AND C%(1,4)=N% AND C%(1,6)=6 AND C%(
     1,5)=6 THEN P%(1,6)=26
2650 IF C%(2,1)=Q% AND C%(5,1)=N% AND C%(3,1)=6 AND C%(
     4,1)=6 THEN P%(3,1)=26
2660 IF C%(1,2)=Q% AND C%(1,5)=N% AND C%(1,3)=6 AND C%(
     1,4)=6 THEN P%(1,3)=26
2670 IF C%(8,2)=Q% AND C%(8,5)=N% AND C%(8,3)=6 AND C%(
     8,4)=6 THEN P%(8,3)=26
```

continued on the next page

```
2680 IF C%(2,8)=Q% AND C%(5,8)=N% AND C%(3,8)=6 AND C%(
     4,8)=6 THEN P%(3,8)=26
2690 IF C%(8,7)=Q% AND C%(8,4)=N% AND C%(8,5)=6 AND C%(
     8,6)=6 THEN P%(8,6)=26
2700 IF C%(7,8)=Q% AND C%(4,8)=N% AND C%(5,8)=6 AND C%(
     6,8)=6 THEN P%(6,8)=26
2710 RETURN
2720 PAS%=0
2730 FOR I%=1 TO 8
2740 FOR J%=1 TO 8
2750 IF C%(I%,J%)=Q% GOTO 2780
2760 NEXT J%:NEXT I%
2770 PAS%=1:RETURN
2780 FOR K%=1 TO 8
2790 VR%=0:C3%=I%:C4%=J%
2800 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
2810 IF C3% < 1 OR C3% > 8 GOTO 2820 ELSE 2830
2820 NEXT K%:GOTO 2760
2830 IF C4% < 1 OR C4% > 8 GOTO 2820 ELSE 2840
2840 IF C%(C3%,C4%)=N% GOTO 2850 ELSE 2860
2850 VR%=VR%+1:GOTO 2800
2860 IF C%(C3%,C4%)=Q% GOTO 2820 ELSE 2870
2870 IF VR% > 0 THEN RETURN
2880 GOTO 2820
2890 DATA 1,0,30,1,20,10,10,20,1,30,1,1,1,1,3
2900 DATA 3,3,3,1,1,0,1,20,3,5,5,5,5,3,20,-1,1,10,3,5
2910 DATA 0,0,5,3,10,-1,0,10,3,5,0,0,5,3,10,-1
2920 DATA -1,20,3,5,5,5,5,3,20,0,-1,1,1,3,3,3,3,1,1,1,-
     1,30,1,20,10,10,20,1,30
2930 DATA 263,100,263,120,270,130,255,130,255,130,255,1
     40,255,140,270,140
2940 DATA 270,140,270,150,270,150,255,150,255,160,270,1
     60,270,160,270,180
2950 DATA 270,180,255,180,270,170,255,170,270,190,270,2
     10,270,200,255,200
2960 DATA 255,200,255,210,255,220,270,220,270,220,270,2
     30,270,230,255,230
2970 DATA 255,230,255,240,255,240,270,240,255,250,270,2
     50,270,250,270,260
2980 DATA 270,260,255,260,255,250,255,270,270,280,270,3
     00,270,300,255,300
2990 DATA 255,310,255,330,255,330,270,330,270,330,270,3
     10,270,310,255,310
3000 DATA 255,320,270,320
```

continued on the next page

```
3010 DATA 310,355,310,375,350,355,335,355,335,355,335,3
     65,335,365,350,365
3020 DATA 350,365,350,375,350,375,335,375,365,355,380,3
     55,380,355,380,375
3030 DATA 380,375,365,375,380,365,365,365,410,355,410,3
     75,410,365,395,365
3040 DATA 395,365,395,375,425,355,440,355,440,355,440,3
     65,440,365,425,365
3050 DATA 425,365,425,375,425,375,440,375,455,375,455,3
     55,455,355,470,355
3060 DATA 470,355,470,365,470,365,455,365,485,375,500,3
     75,500,375,500,355
3070 DATA 515,375,515,355,515,355,530,355,530,355,530,3
     75,530,375,515,375
3080 DATA 515,365,530,365
```

Raffles

Break into His Lordship's house and steal the booty. Lots of obstacles to trip you up, lights to switch on, and you must beware of the dog! For one player against the computer. Keyboard or Joystick.

```
10 'RAFFLES by DAVID RADISIC
20 'copyright (c) AMSOFT 1985
30 '
40 MODE 0:INK 0,0:BORDER 0:INK 1,26:INK 2,15:INK 3,25
50 INK 4,14:INK 5,24,12:INK 6,0:INK 7,0:INK 8,0:PAPER #
   1,7
60 delay=200
70 DIM objx(5,20),objy(5,20),gemx(5,20),gemy(5,20)
80 GOSUB 380
90 GOSUB 720
100 pause=200:GOSUB 340
110 IF gems=0 THEN GOSUB 970
120 PEN 4
130 FOR i=10 TO 12
140 LOCATE 15,i:PRINT"SWAG";
150 NEXT
160 PAPER 0:CLS#2:PAPER 8
170 GOSUB 1170
180 GOSUB 1230
190 GOSUB 1370
200 GOSUB 1510
210 IF rm=0 THEN GOSUB 1900
220 IF dead=0 THEN 160
230 pause=100:GOSUB 340
240 PAPER 0:CLS:PEN 1
250 LOCATE 4,3:PRINT"Do you want";
260 LOCATE 4,5:PRINT"Another game";
270 PEN 5:LOCATE 9,7:PRINT"Y/N";
280 i$=UPPER$(INKEY$):IF i$<>"Y" AND i$<>"N" THEN 280
290 IF i$="N" THEN MODE 2:PEN 1:STOP
300 RUN
310 IF dog=1 THEN RETURN
320 dog=1:dogx=minx(rm):dogy=miny(rm)
330 RETURN
340 FOR loop=1 TO pause
350 FRAME
360 NEXT
370 RETURN
```

continued on the next page

```
380 rm=1:xp=6:yp=4:man$=CHR$(224):dog=0:stolen=0
390 SYMBOL 240,8,8,8,8,8,8,8,8
400 SYMBOL 241,0,0,0,0,255,0,0,0
410 SYMBOL 242,0,0,0,0,15,8,8,8
420 SYMBOL 243,0,0,0,0,248,8,8,8
430 SYMBOL 244,8,8,8,8,248,0,0,0
440 SYMBOL 245,8,8,8,8,15,0,0,0
450 SYMBOL 246,8,12,13,14,12,12,8,8
460 SYMBOL 247,8,12,12,14,13,12,8,8
470 SYMBOL 248,8,24,88,56,24,24,8,8
480 SYMBOL 249,8,24,24,56,88,24,8,8
490 SYMBOL 250,0,0,255,129,129,129,255,0
500 SYMBOL 251,28,20,20,20,20,20,20,28
510 SYMBOL 252,0,0,255,255,255,255,255,0
520 SYMBOL 253,28,28,28,28,28,28,28,28
530 SYMBOL 255,195,165,60,126,90,60,36,24
540 ENT 1,12,-4,1
550 ENT -2,=1000,60,=3000,40
560 ENV 1,10,1,5,2,-4,1,2,-1,20
570 windw$(1)=STRING$(2,250):windw$(2)=CHR$(251)+CHR$(8
    )+CHR$(10)+CHR$(251)+CHR$(8)+CHR$(10)+CHR$(251)
580 door$(1)=STRING$(2,252):door$(2)=CHR$(253)+CHR$(8)+
    CHR$(10)+CHR$(253)+CHR$(8)+CHR$(10)+CHR$(253)
590 switch$(1,0)=CHR$(246):switch$(1,1)=CHR$(247)
600 switch$(2,0)=CHR$(248):switch$(2,1)=CHR$(249)
610 gem$=CHR$(144):obj$=CHR$(233):dog$=CHR$(255)
620 hit$=CHR$(246)+CHR$(248)+CHR$(247)+CHR$(249)+CHR$(2
    52)+CHR$(253)+CHR$(250)+CHR$(251)+gem$+obj$+dog$
630 RESTORE 3010
640 FOR i=1 TO 5
650 READ minx(i),miny(i),maxx(i),maxy(i)
660 READ dir(i,1),dir(i,2),dir(i,3),dir(i,4)
670 NEXT
680 WINDOW #1,minx(rm)-1,maxx(rm)+1,miny(rm)-1,maxy(rm)
    +1
690 WINDOW #2,1,14,1,25
700 CLS #1:PAPER #0,8
710 RETURN
720 ORIGIN 50,50
730 INK 6,24,12
740 RESTORE 3060
750 GOSUB 1280
760 LOCATE 3,20
770 PEN 5:PRINT">";
```

continued on the next page

```
780 PEN 1:PRINT"Escape  Routes";
790 PEN 5:PRINT"<";:PEN 1
800 LOCATE 10,2:PRINT"IN";
810 pause=300:GOSUB 340
820 CLS:LOCATE 1,3:INK 6,0
830 PEN 1:PRINT man$;"  You The Thief":PRINT
840 PEN 2:PRINT LEFT$(door$(1),1);LEFT$(door$(2),1);" D
    oors":PRINT
850 PEN 3:PRINT switch$(1,0);switch$(2,0);" LightSwitch
     (OFF)"
860 PEN 3:PRINT switch$(1,1);switch$(2,1);" LightSwitch
     (ON)":PRINT
870 PEN 4:PRINT LEFT$(windw$(1),1);LEFT$(windw$(2),1);"
     Windows":PRINT
880 PEN 5:PRINT gem$;"  Precious Gems":PRINT
890 PAPER 1:PEN 0:PRINT obj$;"  Obstructions ":PEN 1:PA
    PER 0:PRINT
900 PEN 1:PRINT dog$;"  The Dog"
910 PEN 5:PRINT:PRINT:PRINT
920 PRINT"  Use Joystick":PRINT"   Or Cursor keys"
930 dummy=REMAIN(1)
940 AFTER delay*4,1 GOSUB 340
950 RETURN
960 '
970 'Generate Gems/obstacles
980 '
990 FOR room=1 TO 5
1000 gemr=INT(RND*8)+2:objr=INT(RND*10)+5
1010 minx=minx(room):miny=miny(room):maxx=maxx(room):ma
     xy=maxy(room)
1020 FOR i=1 TO gemr
1030 x=INT(RND*(maxx-minx+1))+minx
1040 y=INT(RND*(maxy-miny+1))+miny
1050 gemx(room,i)=x:gemy(room,i)=y
1060 gems=gems+1
1070 NEXT i
1080 FOR i=1 TO objr
1090 x=INT(RND*(maxx-minx+1))+minx
1100 y=INT(RND*(maxy-miny+1))+miny
1110 objx(room,i)=x:objy(room,i)=y
1120 NEXT i
1130 gems(room)=gemr:obj(room)=objr
1140 NEXT room
1150 CLS
```

continued on the next page

```
1160 RETURN
1170 ON rm GOTO 1180,1190,1200,1210,1220
1180 RESTORE 2670:RETURN
1190 RESTORE 2740:RETURN
1200 RESTORE 2810:RETURN
1210 RESTORE 2880:RETURN
1220 RESTORE 2960:RETURN
1230 PAPER 0:READ rm$:PAPER 8
1240 WINDOW #1,minx(rm)-1,maxx(rm)+1,miny(rm)-1,maxy(rm
     )+1:CLS #1
1250 PEN 1:LOCATE 1,1:PRINT SPACE$(19);
1260 LOCATE 1,1:PRINT "Room :";rm$;
1270 IF lights(rm) THEN INK 7,10:INK 8,10 ELSE INK 7,0:
     INK 8,0
1280 READ a$:IF a$="END" THEN RETURN
1290 IF a$="D" THEN 2180
1300 IF a$="W" THEN 2260
1310 IF a$="L" THEN GRAPHICS PEN 1:GOTO 2340
1320 IF a$="S" THEN 2420
1330 IF a$="F" THEN GRAPHICS PEN 6:GOTO 2340
1340 PRINT"***ERROR ***";
1350 STOP
1360 '
1370 'Display gems/objects
1380 '
1390 PEN 6
1400 FOR i=1 TO obj(rm)
1410 LOCATE objx(rm,i),objy(rm,i)
1420 PRINT obj$;
1430 NEXT
1440 PEN 5
1450 FOR i=1 TO gems(rm)
1460 LOCATE gemx(rm,i),gemy(rm,i)
1470 PRINT gem$;
1480 NEXT
1490 PEN 1:LOCATE xp,yp:PRINT man$;
1500 RETURN
1510 xf=0:yf=0:PEN 1
1520 IF INKEY(0)<>-1 OR INKEY(72)<>-1 THEN yf=-1
1530 IF INKEY(2)<>-1 OR INKEY(73)<>-1 THEN yf=1
1540 IF INKEY(8)<>-1 OR INKEY(74)<>-1 THEN xf=-1
1550 IF INKEY(1)<>-1 OR INKEY(75)<>-1 THEN xf=1
1560 IF xf=0 AND yf=0 THEN 1630
1570 LOCATE xp+xf,yp+yf:ht$=COPYCHR$(#0)
```

continued on the next page

```
1580 IF ASC(ht$)>239 AND ASC(ht$)<246 THEN 1510
1590 IF ht$<>" " THEN 1660
1600 LOCATE xp,yp:PRINT " ";
1610 PAPER 0:LOCATE 15,5:PRINT"     ";:PAPER 8
1620 xp=xp+xf:yp=yp+yf
1630 LOCATE xp,yp:PRINT man$;
1640 IF dog>0 THEN dog=dog MOD 2+1:IF dog=2 THEN 2550
1650 GOTO 1510
1660 hit=INSTR(hit$,ht$):char=ASC(MID$(hit$,hit,1))
1670 ON hit GOTO 1690,1690,1690,1690,1750,1750,1850,190
     0,1970,2090,2650
1680 GOTO 1600
1690 IF hit>2 AND hit<5 THEN char=char-1
1700 IF hit<3 THEN char=char+1
1710 PEN 3:LOCATE xp+xf,yp+yf:PRINT CHR$(char);
1720 lights(rm)=lights(rm) XOR 1
1730 IF lights(rm) THEN INK 7,10:INK 8,10 ELSE INK 7,0:
     INK 8,0
1740 GOTO 1510
1750 IF xf<>0 AND yf<>0 THEN 1630
1760 IF xf<0 THEN dir=4 ELSE IF xf>0 THEN dir=3
1770 IF yf<0 THEN dir=1 ELSE IF yf>0 THEN dir=2
1780 IF dir(rm,dir)=-1 THEN 1630 ELSE rm=dir(rm,dir)
1790 IF dog>0 THEN GOSUB 310
1800 IF dir=1 THEN xp=6:yp=maxy(rm)
1810 IF dir=2 THEN xp=6:yp=miny(rm)
1820 IF dir=3 THEN xp=minx(rm):yp=13
1830 IF dir=4 THEN xp=maxx(rm):yp=13
1840 RETURN
1850 IF xp>5 AND xp<8 THEN 1880
1860 IF xp<6 THEN dir=4 ELSE dir=3
1870 GOTO 1780
1880 IF yp>13 THEN dir=2 ELSE dir=1
1890 GOTO 1780
1900 PAPER 0:CLS:PEN 1
1910 LOCATE 3,3:PRINT"You have escaped";
1920 LOCATE 9,5:PRINT"with";
1930 IF gems=stolen THEN LOCATE 8,7:PRINT"All"; ELSE LO
     CATE 9,7
1940 PRINT USING " ##";stolen;
1950 PEN 5:LOCATE 9,9:PRINT"Gems";
1960 dead=1:RETURN
1970 LOCATE xp,yp:PRINT" ";:xp=xp+xf:yp=yp+yf
```

continued on the next page

```
1980 i=0
1990 i=i+1
2000 IF i>gems(rm) THEN 1510
2010 IF gemx(rm,i)<>xp OR gemy(rm,i)<>yp THEN 1990
2020 IF i=gems(rm) THEN 2050
2030 gemx(rm,i)=gemx(rm,gems(rm))
2040 gemy(rm,i)=gemy(rm,gems(rm))
2050 gems(rm)=gems(rm)-1:stolen=stolen+1
2060 MOVE 400,150+(stolen*2),1,1:DRAW 520,150+(stolen*2
     ),1,1
2070 SOUND 129,248,10,12,0,1
2080 GOTO 1980
2090 noise=INT(RND*15)
2100 SOUND 1,3000,10,noise,0,0,10
2110 PAPER 0:LOCATE 15,5:PRINT"Crash ";:PAPER 8
2120 IF noise<10 OR delay=50 THEN 1630
2130 delay=delay-50
2140 dummy=REMAIN(1)
2150 AFTER delay*4,1 GOSUB 310
2160 GOTO 1630
2170 '
2180 'Draw doors
2190 '
2200 READ no,dr$
2210 IF dr$="V" THEN dr=2 ELSE dr=1
2220 PEN 2
2230 pic$=door$(dr):GOSUB 2500
2240 GOTO 1280
2250 '
2260 'Draw windows
2270 '
2280 READ no,wi$
2290 IF wi$="V" THEN wi=2 ELSE wi=1
2300 PEN 4
2310 pic$=windw$(wi):GOSUB 2500
2320 GOTO 1280
2330 '
2340 ' Draw lines
2350 '
2360 READ x1,y1,x2,y2
2370 MOVE x1,y1,,0
2380 DRAW x1,y2,,0:DRAW x2,y2,,0
2390 DRAW x2,y1,,0:DRAW x1,y1,,0
2400 GOTO 1280
```

continued on the next page

```
2410 '
2420 'Draw switches
2430 '
2440 READ no,sw$
2450 IF sw$="L" THEN sw=1 ELSE sw=2
2460 PEN 3
2470 pic$=switch$(sw,0):GOSUB 2500
2480 GOTO 1280
2490 '
2500 'Print char
2510 '
2520 READ x,y:LOCATE x,y:PRINT pic$;
2530 no=no-1:IF no>0 THEN 2520
2540 RETURN
2550 PEN 1:LOCATE dogx,dogy:PRINT" ";
2560 man$=CHR$(225)
2570 IF (dogx=xp AND dogy=yp) OR (dogx=xp+xf AND dogy=y
     p+yf) THEN 2650
2580 IF dogx<xp THEN dogx=dogx+1
2590 IF dogx>xp THEN dogx=dogx-1
2600 IF dogy<yp THEN dogy=dogy+1
2610 IF dogy>yp THEN dogy=dogy-1
2620 LOCATE dogx,dogy:PRINT dog$;
2630 SOUND 1,0,RND*40,10,1,2,31
2640 GOTO 1510
2650 PRINT"SNAP";
2660 dead=1:RETURN
2670 DATA Hallway
2680 DATA L,64,308,226,4
2690 DATA D,2,H,6,3,6,22
2700 DATA D,2,V,4,12,9,11
2710 DATA S,1,L,4,11
2720 DATA S,1,R,9,14
2730 DATA END
2740 DATA Lounge
2750 DATA L,2,308,258,4
2760 DATA D,1,V,10,12
2770 DATA W,1,H,6,3
2780 DATA W,1,V,2,12
2790 DATA S,2,R,10,11,10,15
2800 DATA END
2810 DATA Dining room
2820 DATA L,2,308,258,4
2830 DATA W,1,V,10,12
```

continued on the next page

```
2840 DATA W,1,H,6,3
2850 DATA D,1,V,2,12
2860 DATA S,2,L,2,11,2,15
2870 DATA END
2880 DATA Kitchen
2890 DATA L,2,276,384,4
2900 DATA D,2,H,6,5,6,22
2910 DATA W,1,H,10,22
2920 DATA W,1,V,14,13
2930 DATA D,1,V,2,13
2940 DATA S,1,L,2,16
2950 DATA END
2960 DATA Pantry
2970 DATA L,2,276,256,4
2980 DATA D,1,V,10,12
2990 DATA S,1,R,10,11
3000 DATA END
3010 DATA 5,4,8,21,0,4,3,2
3020 DATA 3,4,9,21,-1,-1,1,-1
3030 DATA 3,4,9,21,-1,-1,-1,1
3040 DATA 3,6,13,21,1,0,-1,5
3050 DATA 3,6,9,21,-1,-1,4,-1
3060 DATA L,64,308,480,100
3070 DATA F,250,98,294,102
3080 DATA F,250,306,294,310
3090 DATA F,390,94,430,106
3100 DATA F,390,302,430,314
3110 DATA F,474,240,488,270
3120 DATA F,474,124,488,154
3130 DATA F,58,240,72,270
3140 DATA L,226,308,322,180
3150 DATA L,160,180,480,100
3160 DATA L,64,180,160,100
3170 DATA END
```

Appendix 4
Index

(Note that all references given in this index correspond to chapter and page number, e.g. 1.44 refers to Chapter 1 page 44.)

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

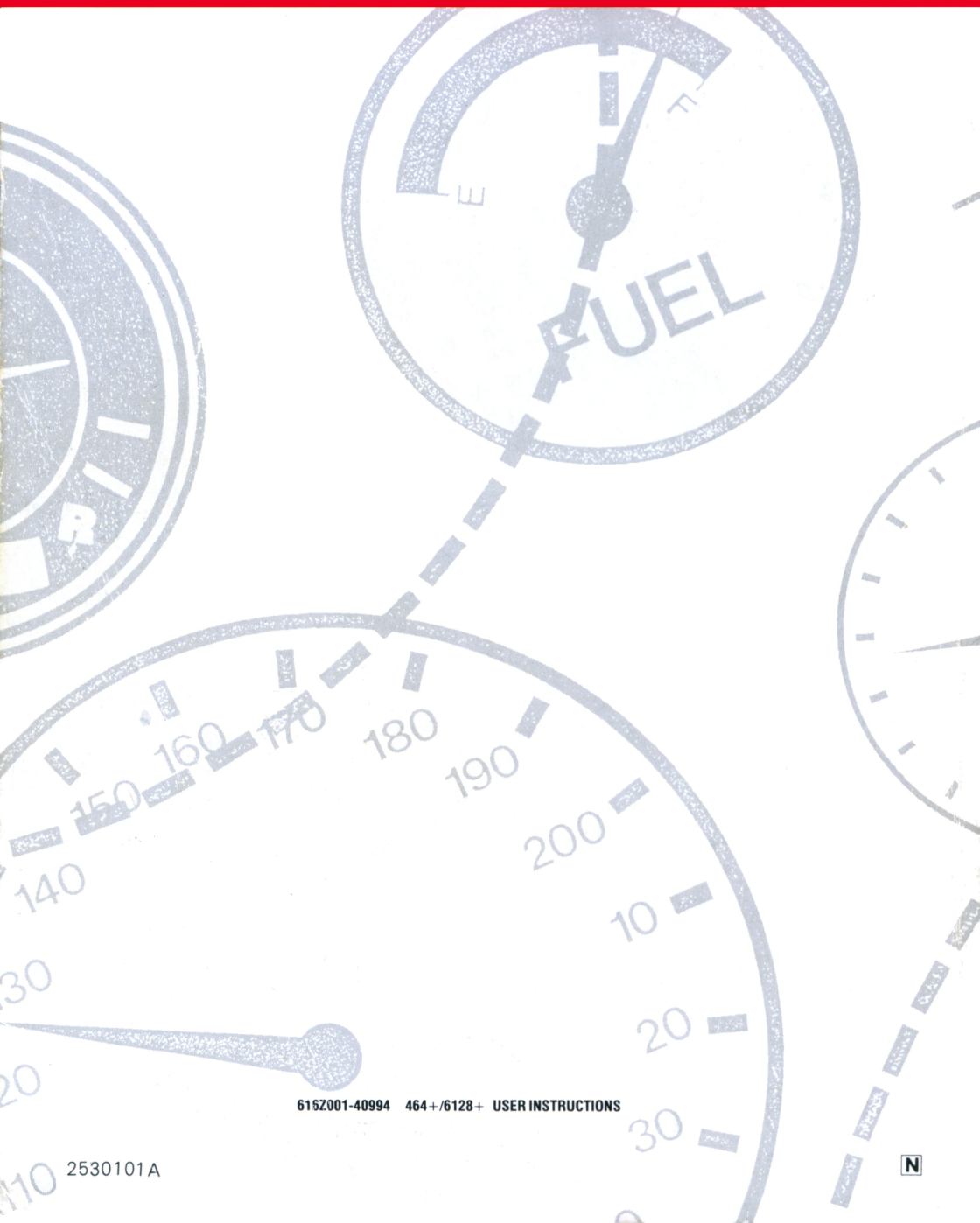
V

W

X

Y

Z



615Z001-40994 464+/6128+ USER INSTRUCTIONS

2530101A

N

AMSTRAD

464 PLUS 6128 PLUS

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/

464 PLUS

6128 PLUS

AMSTRAD

464 PLUS

6128 PLUS

AMSTRAD