WELCOME GUIDE

BRITISH BROADCASTING CORPORATION
MASTER SERIES MICROCOMPUTER

The BBC Microcomputer System

Master Series

WELCOME GUIDE

Part number 0443,000
Issue 1
January 1986

WARNING: THIS COMPUTER MUST BE EARTHED

Important:

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

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

The moulded plug must be used with the fuse and fuse carrier firmly in place. The fuse carrier is of the same basic colour (though not necessarily the same shade of that colour) as the coloured insert in the base of the plug. Different manufacturers' plugs and fuse carriers are not interchangeable. In the event of loss of the fuse carrier, the moulded plug MUST NOT be used. Either replace the moulded plug with another conventional plug wired as described below, or obtain a replacement fuse carrier from an Acorn Computers authorised dealer. In the event of the fuse blowing it should be replaced, after clearing any faults, with a 3 amp fuse that is ASTA approved to BS1362.

If the socket outlet available is not suitable for the plug supplied, the plug should be cut off and the appropriate plug fitted and wired as noted below. The moulded plug which was cut off must be disposed of as it would be a potential shock hazard if it were to be plugged in with the cut off end of the mains cord exposed.

As the colours of the wires 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 in the plug 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.

Exposure

The computer should not be exposed to direct sunlight or moisture for long periods.

Ventilation

Do not block the ventilation slots in the case – see text for details.

Internal battery

The computer is fitted with a Lithium non-rechargeable cell which contains Lithium, either Manganese Dioxide or Chromium Dioxide and a small quantity of Thionyl Chloride. The cell is completely safe when correctly fitted but the following precautions should be observed if the cell is removed from the computer's case:

Keep away from children and animals.
Do not attempt to recharge the cell.
Do not crush, puncture, open, dismantle, or otherwise mechanically interfere with or abuse the cell.
Do not dispose of in fire.
Do not solder.
Do not short circuit.

The cell has a very high energy level for its size and should not be carried in pockets with keys, loose change etc. or placed in contact with metal objects.

Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

The product described in this manual and products for use with it, are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained upon request from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:

Technical Enquiries
Acorn Computers Limited
Cambridge Technopark
Newmarket Road
CAMBRIDGE CB5 8PD

All maintenance and service on the product must be carried out by Acorn Computers' authorised dealers. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of this product, and therefore Acorn Computers 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 manual, or any incorrect use of the product.

This book is part of the BBC Computer Literacy Project.
Cover design concept by Carrods Graphic Design

Written by ACORNCES

First published 1986
Published by Acorn Computers Limited

Contents

Appendices

Foreword

A few years ago, the suggestion that you might have a computer in your own home would have been greeted with disbelief. Now, home computers are an accepted fact and more and more people are beginning to investigate their potential.

Thankfully, the intervening years have seen many developments and today's microcomputers offer real processing power – making them suitable for use not only in the home but also in classrooms, laboratories and our increasingly automated offices.

BBC Microcomputer systems have been available throughout this period of change and, unlike many comparable machines, they have provided flexibility and expandability, enabling them to grow to meet the changing needs of their users. It is therefore no accident that this, the latest BBC Microcomputer, is one of the most advanced in its class, offering a large user memory and a number of powerful standard facilities including:

- BBC BASIC;
- VIEW, a professional word-processing package;
- ViewSheet, a powerful electronic spreadsheet;
- The Acorn Editor, a full-screen text editor;
- Terminal emulation software.

Where appropriate, each of these features is compatible with its implementation on earlier BBC microcomputers and, as before, your computer is readily expandable by the addition of disc units, printers, high-resolution monitors and expansion units such as the Teletext and PRESTEL adaptors.

Introduction

This book is the 'Welcome Guide' for your Master Series computer. It provides an introduction to the system for all new owners – including those who have never used a computer before.

The book covers the initial setting-up of the system, an overview of the computer's capabilities, information on expansion options and a series of useful appendices, designed for quick reference to particular features. It is **not** intended to be a comprehensive technical manual or a self-study guide – users requiring this type of material should refer to the following publications, which are available from dealers:

The Master Series Reference Manual – Part 1
The Master Series Reference Manual – Part 2
The VIEW Guide
The ViewSheet Guide

An Advanced Reference Manual for the Master Series computers is also planned for later availability.

The first chapter describes how you should start using your computer. It also provides some background information on the accompanying Welcome software which, for first-time users, will give an indication of the computer's power and speed.

Subsequent chapters introduce:

- BBC BASIC;
- the VIEW word-processor;
- the ViewSheet electronic spreadsheet;
- alternatives to cassette-tape storage;
- the Editor;
- the terminal emulation software;
- expansions to the system.

1. Getting Started

Unpacking the equipment

Having already opened the packaging to retrieve your copy of this Welcome Guide, you should now have in front of you:

- empty packaging;
- your computer;
- an aerial lead;
- a welcome tape;
- a welcome disc;
- two reversible keyboard inserts;
- a VIEW reference card;
- a ViewSheet reference card;
- a guarantee card.

Put the guarantee card in a safe place, then flatten the box and keep it together with the two polystyrene inserts so that, in the unlikely event of a fault, you can return the machine to your supplier in safety.

If you do not have a disc unit, you will also require an ordinary cassette tape recorder and a lead to connect it to the computer. The lead is not supplied with the computer because of the wide variety of cassette recorder socket types. A full specification of the various types of lead is given on page 239 and your supplier or an Acorn dealer will be pleased to provide one suitable for your needs.

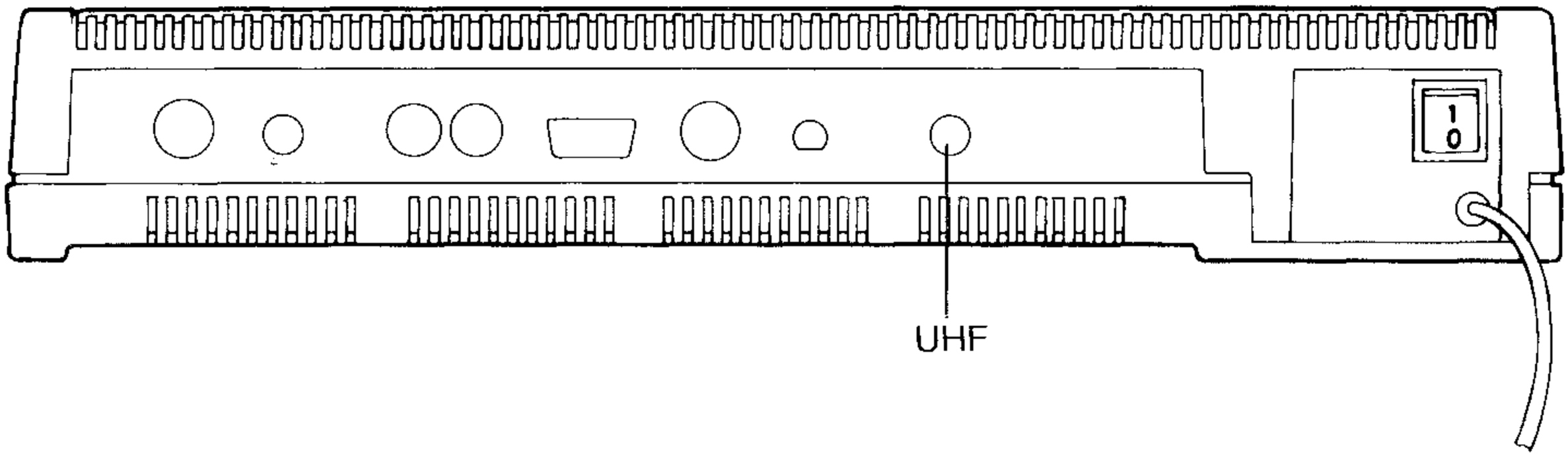
Preparation

This section deals with getting the computer going for the first time and the assumption is that you will be using an ordinary domestic colour television for displaying output from the computer. You should refer to Chapter 8 for instructions on how to connect a black-and-white or a colour monitor. Connection and use of your cassette recorder is discussed later in this chapter, under the heading *The Welcome programs*.

The computer should be placed on a flat surface, such as a desk or table top, within easy reach of the television. Soft surfaces (such as a carpeted floor) should be avoided as they may block the ventilation slots in the casing and cause overheating. You will need access to mains power for the computer and your television or monitor.

Making the connections

Take the aerial lead supplied and identify the end with the long central pin. This connector should be fixed firmly to the socket marked 'UHF' on the back of the computer:

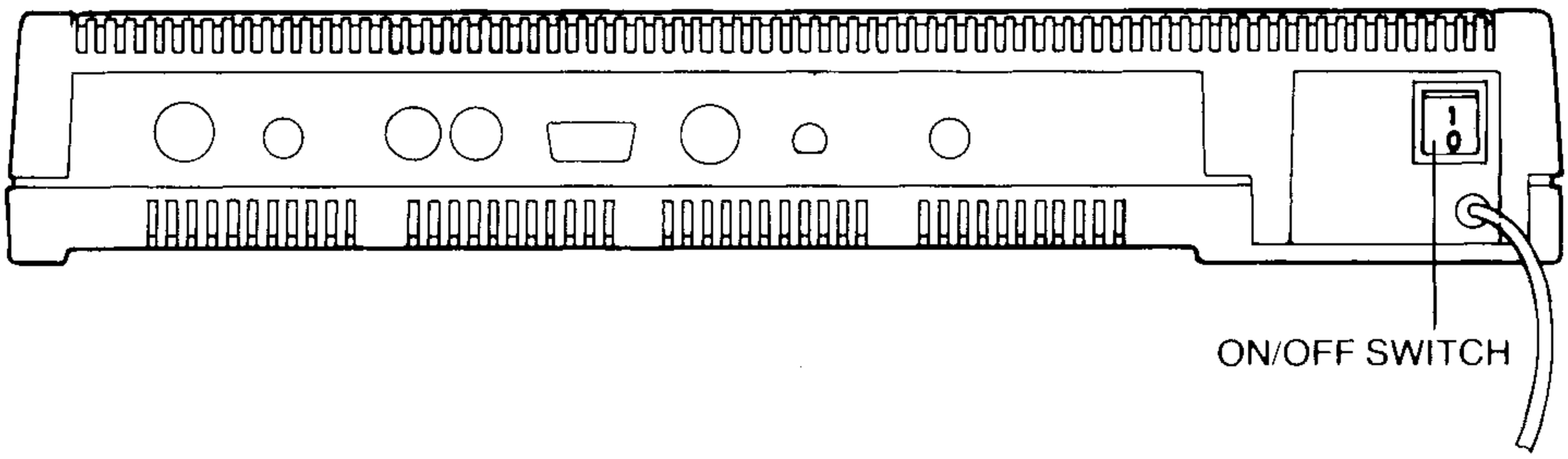


The other end of the aerial lead should be connected to the television set in place of the aerial used to receive conventional television pictures.

Switching on

Plug the computer and the television into the mains and switch on the power. Make sure that the television is switched on and that the volume is turned right down – the computer uses its own internal speaker for generating sound.

Switch the computer on by means of the ON/OFF switch on the back:



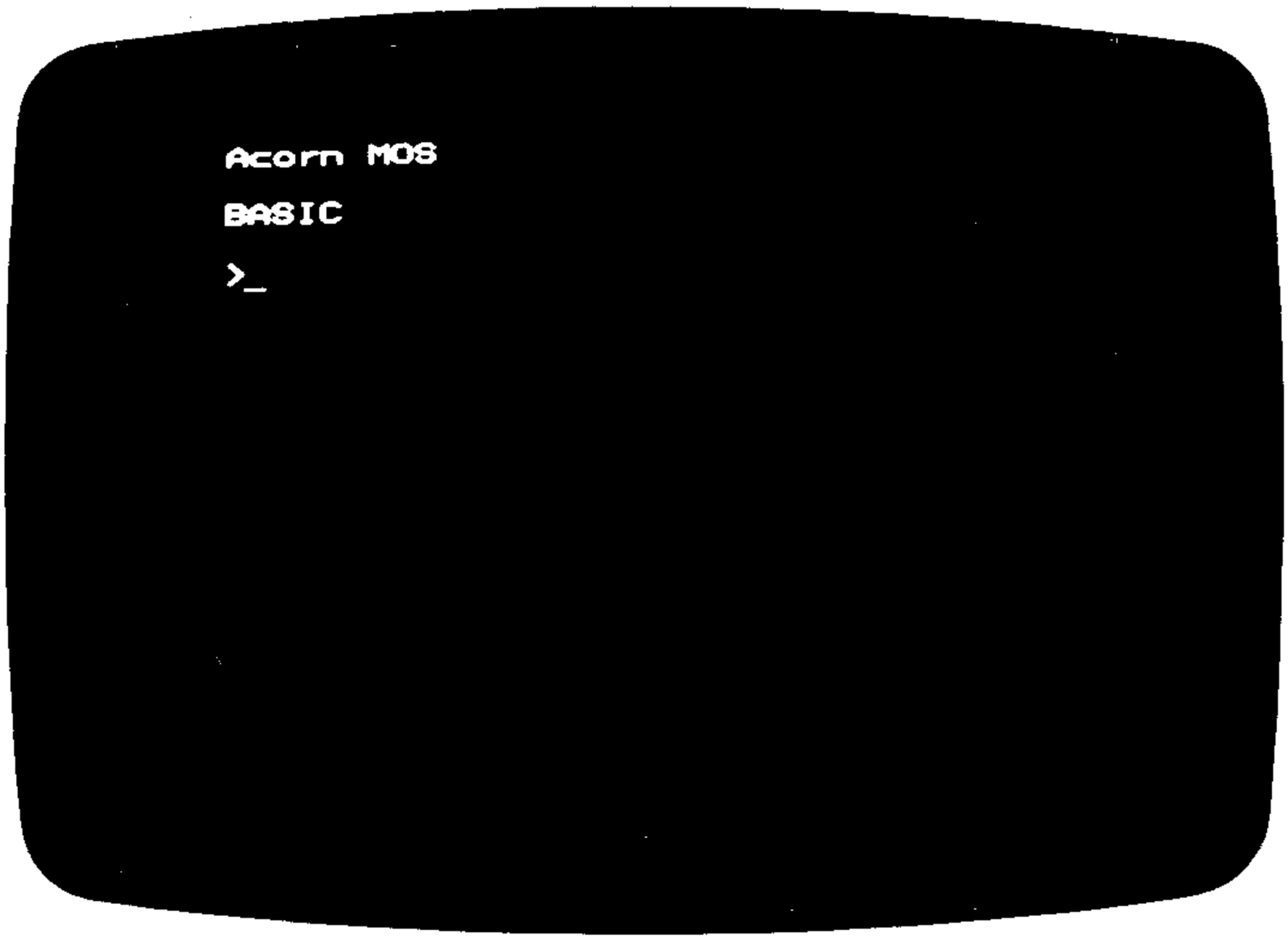
and, if everything has gone according to plan, you should be rewarded by a 'bleep' from the computer's speaker and the appearance of two red indicators at the top left of the keyboard. At this stage, the screen is likely to be blank or to show a 'snowstorm' effect.

Tuning the television

The next stage is to tune your television so that it can receive the transmissions from your computer, which are made on channel 36. The method of achieving this will vary from television to television but, if your set is operated by means of push-buttons, you are advised to select and tune one of the buttons you do

not normally use for receiving television broadcasts. In this way you will always be able to use the computer without interfering with the other settings.

The aim of the tuning exercise is to achieve the following image as clearly and sharply as possible – it contains white letters on a black background:



If the image appears blurred or distorted, try some further adjustments to the tuning – you may be looking at a weaker version of the true signal. However, if all else fails, you will have to consult your supplier or an Acorn dealer for advice.

Using the computer

This section is intended to familiarise you with the operation of the computer at its most basic level, i.e. using the keyboard for giving simple commands. We start with a description of the keyboard itself and introduce the conventions we shall use to describe key depressions in subsequent sections.

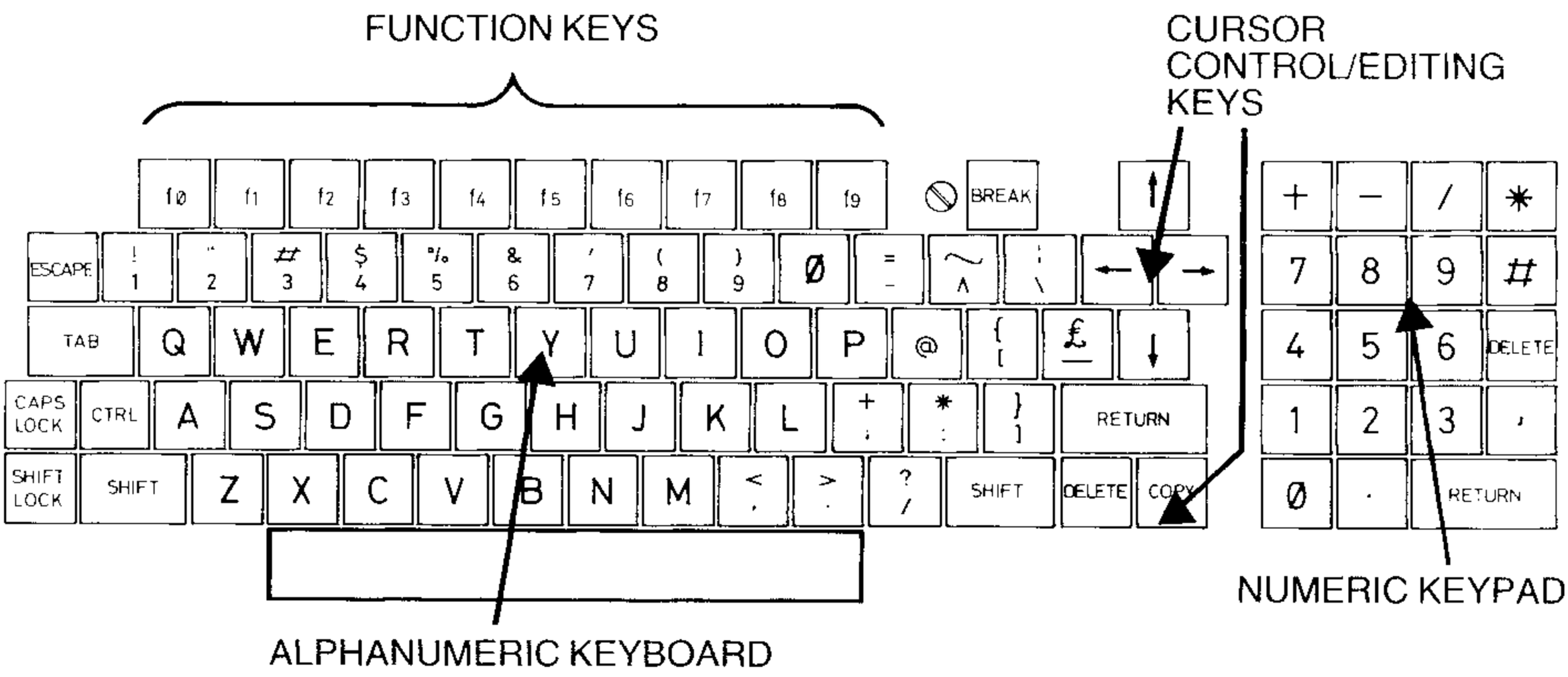
The keyboard

For descriptive purposes, the computer's keyboard can be divided into four separate areas:

- the main, **alpha-numeric** keyboard, which is laid out in the same format as found on a conventional typewriter, with one or two additions;
- a smaller, **numeric keypad**, which contains keys associated with the input of numeric data;

– a group of grey-green **cursor control / editing keys**;

– a row of red **function keys**, labelled f0 - f9.



The keyboard's 'touch' is similar to most electric typewriters in that only brief, light pressure is required to activate each key. The difference, of course, is that the characters produced by each key depression are displayed on the screen, rather than being printed on paper. Under normal circumstances, the response is immediate, although there are occasions (when the computer is busy doing something else) when there may be a momentary delay before the characters appear.

The keyboard also incorporates a feature known as **auto-repeat** – if a key is pressed and held down, the corresponding character will be repeated, after a short initial delay. Repetition continues until the key is released or until the computer runs out of space to store the line being input (indicated by a continuous tone from the speaker).

Throughout the remainder of this guide, we shall use

```
text like this
```

to denote input from the keyboard and output on the screen whereas symbols like

[RETURN]

denote specific key depressions. The simultaneous depression of two keys is indicated like this:

[SHIFT]+[BREAK]

The alpha-numeric keyboard

The alpha-numeric keyboard contains keys denoting all the letters of the alphabet (including space), the numbers 0 to 9, various special symbols (such as punctuation, £ % etc.) plus a number of other special-purpose keys. It also contains, in the top left-hand corner, a row of three red indicators labelled *power*, *caps lock* and *shift lock*.

The *power* indicator is illuminated while the computer is switched ON.

If *caps lock* is ON (i.e. illuminated), depression of any alphabetic key will produce a capital (upper case) letter; depression of any keys containing two symbols will produce the lower of the two characters.

If *shift lock* is ON , the alphabetic keys will still produce upper case letters but depression of any key containing two symbols will produce the upper of the two characters.

If neither *caps lock* nor *shift lock* is ON, depression of the alphabetic keys will produce small (lower case) letters and the keys containing two symbols will once again produce the lower of the two characters.

The state of the *caps lock* and *shift lock* indicators is controlled by the [CAPS LOCK] and [SHIFT LOCK] keys – each depression switches the corresponding indicator ON or OFF, depending on its current state. Note that it is impossible to illuminate both caps lock and shift lock from the keyboard – the computer uses this simultaneous indication to denote a particular circumstance, as described below.

The two [SHIFT] keys have no effect while *shift lock* is ON. If *shift lock* is OFF, (regardless of the setting of *caps lock*) the [SHIFT] keys cause upper case letters and symbols to be produced if either is held down while another key is depressed. The [SHIFT] keys do not affect the *shift lock* indicator.

A further option is provided by pressing [SHIFT]+[CAPS LOCK]. In this case, *caps lock* is switched ON as usual but in the input of subsequent characters lower case letters of the alphabet may be obtained by holding down a [SHIFT] key.

Whether *caps lock* or *shift lock* (or neither) is ON for a particular session at the computer is a matter of personal preference although the choice will also depend upon the type of input, for example:

- Conventional text, such as an item of correspondence input to the VIEW word-processor, consists mainly of lower case characters interspersed with a few capital letters;
- A BASIC program consists of a mixture of special upper case words (called keywords) interspersed with other, often lower case words (called variable names).

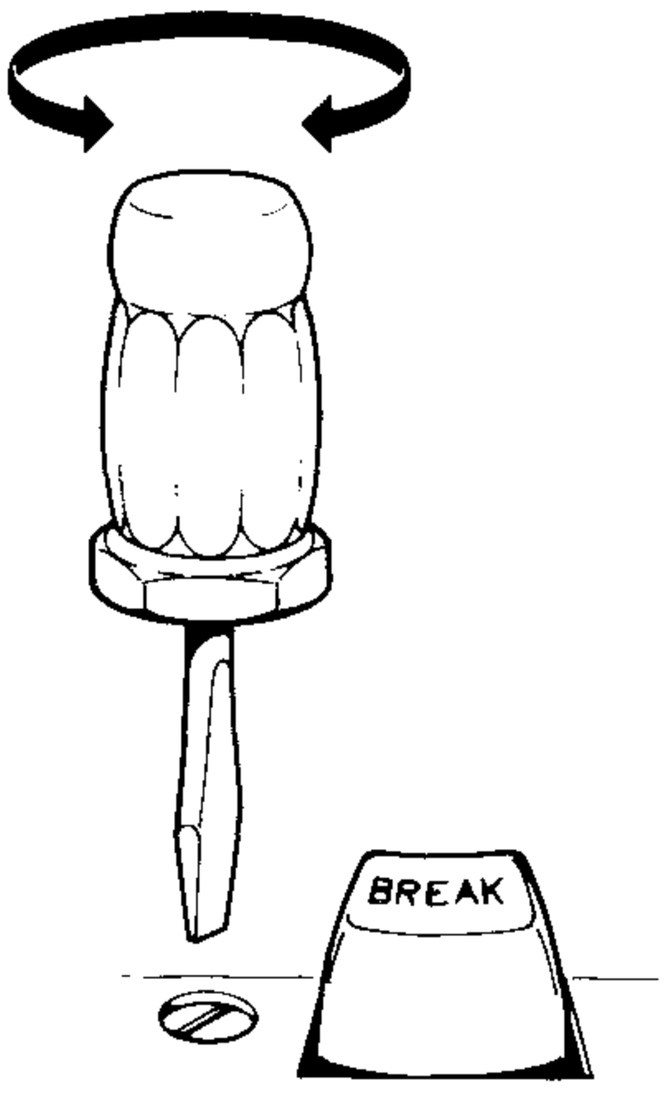
[CTRL] (which is an abbreviation for *control*) has no effect on its own but it may be used in conjunction with other keys on the keyboard to invoke a number of special effects. For example, [CTRL]+G causes the computer to emit a short bleep; [CTRL]+L clears the display screen. Other examples are given in the remainder of this guide and a summary of the various effects is given in Appendix G.

[TAB] normally acts like a space-bar depression although it has a special significance when using VIEW, ViewSheet or the Editor, as described later in this guide.

[RETURN] is used to indicate that a particular line of input is complete – prior to the depression of [RETURN], [DELETE] may be used to erase the most recent character(s) you have typed.

[ESCAPE] and [BREAK], as their names imply, are provided to enable you to interrupt what the computer is doing, although [ESCAPE] also has a special significance when using VIEW, ViewSheet and the Editor. [ESCAPE] should be considered to be a 'polite request', which normally stops the computer without any side-effects, whereas [BREAK] is a definite command which stops the computer at all costs.

Depression of [BREAK] alone is sometimes referred to as a **soft break** because it has the effect of resetting the computer to the condition it was in at the start of the current session (BASIC, VIEW, ViewSheet, Editor etc.). A **hard break** is achieved by pressing [CTRL]+[BREAK]; this resets the computer so that it assumes the state in which it would normally be immediately after switching on.



[SHIFT]+[BREAK] has a special significance if you are using a disc unit and further details are given in the section entitled *The Welcome programs*, on page 15.

The potentially hazardous side-effects of accidental depression of [BREAK] can be avoided by turning the **break key lock** (see illustration) clockwise through 90 degrees using a suitable flat-bladed screwdriver. Normal operation is restored by returning the screwhead to its original position.

The numeric keypad

The numeric keypad is provided as a convenient means of entering large quantities of numeric data – it contains:

- The digits 0 - 9;
- Symbols denoting the four arithmetic operations (* being used for multiplication, / for division);
- Full stop (decimal point) and comma;
- Separate [RETURN] and [DELETE] keys;
- The # symbol.

Each key replicates the function of the corresponding key in the main keyboard, with the added advantage that +, * and # may be obtained directly (i.e. without the use of [SHIFT]).

The cursor control / editing keys

Under normal circumstances, the screen will show a flashing symbol known as the **cursor**; it indicates the position at which the next character to be typed will be displayed. The cursor moves one character position to the right for each normal key depression, one character position to the left for each depression of [DELETE] and to the start of a new line for each depression of [RETURN].

The four arrowed cursor control keys may be used to move the cursor around the screen and it will be seen from later chapters that this facility is fundamental to the use of VIEW, ViewSheet and the Editor.

[COPY] has a special function in each of the above but it is also used in conjunction with the cursor control keys for **cursor editing** – a technique mainly used during the input and correction of programs and which is described on page 40.

The function keys f0 – f9

In certain applications, such as VIEW, ViewSheet and the Editor, it is convenient to make use of a single key depression to denote a particular action and the 10 red function keys across the top of the main keyboard are provided for this purpose. Each key may be used on its own, in conjunction with [SHIFT], [CTRL] or, indeed, [SHIFT]+[CTRL], giving a total of 40 additional keyboard

functions. In these cases, it is usual to define the function invoked by each type of depression on a special keyboard insert, such as those supplied with your computer.

In addition, the function keys may be 'programmed' to produce a sequence of one or more characters, thereby minimising the number of keystrokes required to carry out frequently-used tasks. A brief description of function key programming is given in the next section and full instructions (including the way in which the cursor control keys, [COPY] and the numeric keypad can be programmed) are contained in the Reference Manual.

The screen display

This section introduces the various screen displays that are available and gives you an opportunity to try out your newly-acquired keyboard skills. For the time being, however, do not worry about the meaning of what you are asked to type but concentrate on pressing the correct keys. If you type a line incorrectly (i.e. you press [RETURN] before you spot the mistake), the computer will respond with a simple message, such as:

```
Mistake
```

or

```
No such variable
```

Ignore these messages for the time being and merely type the line in again; their significance is explained in later chapters. One of the most likely mistakes at this stage is to type the letter O instead of the number 0, which are denoted by 0 and Ø respectively. If things appear to have gone irretrievably wrong, try pressing [ESCAPE] and, if that has no effect, press [BREAK].

The computer is able to display output on the screen in a variety of different **modes**, each of which has its own characteristics, in terms of the number and length of its lines of text, the size and shape of the characters displayed and its ability to present **graphics** (points, lines and areas of colour). Each screen mode is identified by a number, which may be in the range 0 – 7 or 128 – 135. These two sets of modes are identical in terms of what is actually displayed on the screen; they differ only in the size and location of the area of memory set aside for storing the current content of the screen. Modes 0 – 7 are identical to the eight modes available on the BBC Model B microcomputer; modes 128 – 135 are referred to as the **shadow** screen modes (identical to those available on the BBC Model B+ microcomputer) which provide the maximum amount of user memory for a given type of display. We shall use modes 128 – 135 in all the examples in this guide.

You have a means of instructing the computer to start up in any of the

available modes (see page 25) but the standard setting is mode 7, which provides:

– 25 lines of text, each 40 characters in length;
– the **teletext** character set (see below);
– limited graphics in the form of small blocks of colour.

The > symbol immediately to the left of the flashing cursor is an example of a **prompt** and its appearance indicates that the computer is waiting for you to type something. Try typing these lines to see the effect; in each case the computer will respond by displaying the characters inside the quotation marks:

```
PRINT"White on black"[RETURN]
PRINT"[SHIFT]+[f1]Red on black"[RETURN]
```

In mode 135, [SHIFT]+[f1] and [CTRL]+[f2] etc. generate what are known as **teletext control codes** which affect the way in which the remaining characters on a particular line are displayed. Examples of this type of screen display can be seen on pages from either the BBC's CEEFAX or the IBA's Oracle services and further information is provided in Chapter 2 of this guide and in the Reference Manual.

If you type:

```
MODE128 [RETURN]
```

the screen will clear and a smaller prompt will appear in the top left-hand corner.

You have now selected mode 128 which provides:

– 32 lines of text, each 80 characters in length;
– the full ASCII character set (see below);
– high-resolution, 2-colour graphics.

Now type:

```
PRINT"White on black"[RETURN]
COLOUR0:COLOUR129:PRINT"Black on white"[RETURN]
MOVE 600,500:PLOT149,750,500[RETURN]
MOVE 600,500:PLOT157,700,500[RETURN]
```

You may like to try repeating the same sequence of examples in each of modes 129, 130, 132 and 133 – the remaining modes which offer a graphics facility. Notice the effect that each change of mode has on the size and shape of each character you type, the colours produced and the 'crispness' of the circle.

Modes 131 and 134 offer a text-only display consisting of 25 lines of 80 and 40 columns respectively.

The welcome software contains a demonstration of the capabilities of the

various screen modes and Appendix A, on page 181 gives a full specification of the characteristics of each mode.

A note on character sets

Computers use simple codes to represent characters which are stored in memory or displayed on the screen and your computer offers two, internationally accepted coding conventions, namely **teletext** and **ASCII**. (ASCII is an abbreviation for American Standard Code for Information Interchange.) The teletext set is available only in modes 7 and 135 and the ASCII set is available in all others.

It is the ASCII character set which is etched into the keytops on the computer's keyboard and in any mode other than 7 or 135 a representation of the corresponding character will be displayed on the screen. The Teletext character set is identical for all the letters of the alphabet, the digits 0 – 9 and all except eight of the special symbols:

ASCII symbol: | \ ] ^ { ¦ } ¯

Teletext symbol: ← ½ → ↑ ¼ ‖ → ÷

In addition, the teletext character set contains the elementary graphics characters and teletext control codes mentioned on page 11, full details of which are given in Appendix B.

Matters are made somewhat more complicated by the fact that your computer allows the ASCII character set to be redefined and extended, thereby enabling foreign, italic and a variety of user-defined characters to be displayed. The example below redefines the @ key so that it displays the mathematical symbol used to denote pi:

```
MODE134[RETURN]
VDU23,64,0,2,124,168,40,40,40,0[RETURN]
```

A utility to help you design your own characters is provided as part of the Welcome software.

Communicating with the computer

You have now spent a short time typing things at the computer's keyboard and witnessing the result. Initially, it does not seem particularly surprising that when you press, say, A, the computer displays an A on the screen – this is exactly what you would expect. In fact, one part of the computer, called the **machine operating system** (MOS) works incredibly hard to produce this simple result and it is in action for every instant that the computer is switched on. Even when the computer appears to be idle, waiting for you to type

something at the keyboard, the MOS is busy maintaining the screen display and carrying out other vital functions.

The MOS is also responsible for calling up each of the other systems provided in your computer i.e. VIEW, ViewSheet etc. Only one system may be operational at a given time and, unless you tell it otherwise, the MOS will automatically select the BASIC language system for you when the computer is switched on – hence the appearance of the word BASIC in the screen display shown on page 5. Thereafter, all input from the keyboard is collected by the MOS and passed to the system you have chosen – you have (perhaps without realising it) been typing BASIC instructions in the previous section. Any messages you received, such as `Mistake` or `Missing "`, were produced by the BASIC system to indicate that it was unable to make sense of the line it received from the MOS. Needless to say, it was the MOS which actually did the job of putting the characters on the screen.

There are, however, occasions when it it neccessary to communicate directly with the MOS, regardless of the system currently in use. These **operating system commands** have an asterisk (*) as their first character and this symbol is used to tell the MOS that it must deal with the remainder of the line itself.

For example, if you type

`*TIME` RETURN

the MOS will respond with the day, date and time from its internal clock (which is maintained by battery when the computer is switched off). See the section headed *The Welcome utilities* on page 22 for instructions on how to reset the clock if it is wrong.

`*ROMS` RETURN

will cause the MOS to list the various systems and languages resident in the computer's **read-only memory** (ROM) sockets. These will include VIEW, ViewSheet and the Editor.

Now try typing:

`*WORD` RETURN

Immediately, the MOS clears the screen and selects the VIEW word-processor. Similarly, typing:

`*SHEET` RETURN

tells the MOS to select ViewSheet. The BASIC language system can be reinstated by typing:

`*BASIC` RETURN

The *KEY command tells the MOS to associate a sequence of characters with a particular function key. For example, if you type:

```
*KEY0function RETURN
```

each subsequent depression of [f0] will produce the characters function, so you could abbreviate the input of the phrase function keys have lots of functions by typing:

```
[ f0 ] keys have lots of [ f0 ]s
```

In this somewhat trivial example, the line remains incomplete (i.e. you can add further characters to it, delete characters from it etc.) exactly as if the characters were being typed one at a time from the keyboard. You can, however, include a special sequence (|M) to simulate depression of RETURN so that a function key depression becomes equivalent to one or more complete lines. There is also no reason why the string associated with a particular function key should not itself contain operating system commands, for example:

```
*KEY1*TIME|M*ROMS|M RETURN
```

This causes each depression of [f1] to produce the time and date followed by a listing of the computer's ROM contents.

Other operating system commands can be used to tell the MOS to change the way it behaves. You will recall, for example, that pressing and holding down a key on the keyboard invokes the *auto-repeat* facility in which the character is repeated after an initial delay. Both the initial delay and the speed at which the character is repeated are controlled by the MOS and they can be changed if required. Remind yourself of the normal settings by producing a sequence of characters using auto-repeat (and RETURN), then type:

```
*FX12,1 RETURN
```

and repeat the sequence. Now see what happens if you try to produce the same sequence after typing:

```
*FX11,0 RETURN
```

In other words, *FX12 enables you to adjust the speed at which characters are repeated and *FX11 enables you to adjust the delay before auto-repeat commences. (*FX11,0 actually switches the auto-repeat facility off altogether). You can restore both the speed and delay to their initial settings by typing:

```
*FX12,0 RETURN
```

A summary of these, and the host of other special effects is given in Appendix D.

Finally, the MOS also responds directly to control key depressions, such as [CTRL]+G and [CTRL]+L mentioned above. These two examples are complete in themselves but others, such as [CTRL]+S (which can be used to change the screen colours in modes 0 – 6 and modes 128 – 134) need further keystrokes to achieve their effect. Select, say, mode 3 and press:

[CTRL]+S followed by `0 4 0 0 0`

The five additional characters do not appear on the screen but the MOS interprets them as a request to change the background colour (0) to blue (4). Similarly:

[CTRL]+S followed by `7 1 0 0 0`

changes the text colour (7) to red (1).

[CTRL]+T, or a subsequent change of mode resets the screen to its default values of white text on a black background.

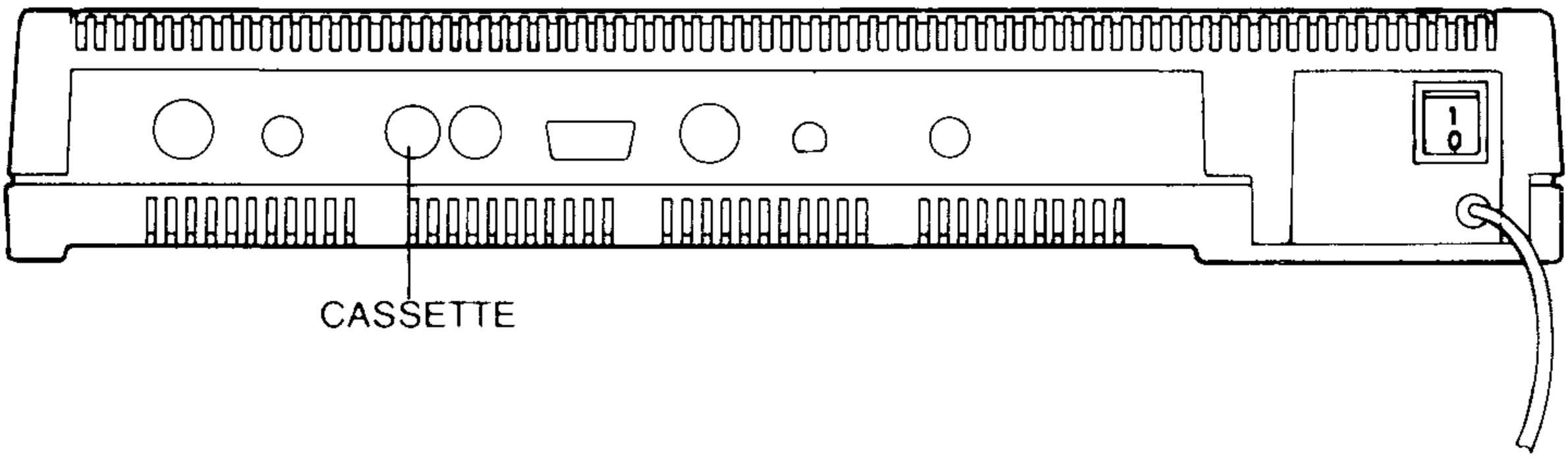
The Welcome programs

This section describes how to connect your tape recorder and how to run the programs and utilities provided on the Welcome tape and its disc equivalent.

Connecting a tape recorder

If you intend to use a disc unit from the outset, you should ignore these instructions and refer instead to the section entitled *Connecting a disc unit* below. Ideally, your tape recorder should be mains powered but if it is battery powered make sure that the batteries are in good condition – you will encounter difficulties if the recorder is not operating at the correct speed.

Your tape recorder lead will be one of those described in Appendix K. The single plug corresponding to the end shown under the heading COMPUTER should be inserted into the socket marked 'cassette' on the back of the computer:



Plug(s) shown under the heading 'TAPE RECORDER' should be inserted into the corresponding sockets on your tape recorder.

Motor control

It is normally desirable to allow the computer to start and stop the tape recorder automatically but this facility is not available with all combinations of tape recorders and leads. The procedure outlined below will enable you to determine whether 'motor control' is available with your equipment.

1. Ensure that the equipment is connected to the mains and switched on. If you have just been using the computer, execute a hard break (CTRL+BREAK) to reset it to its initial state.

2. Place the Welcome tape in the recorder such that side 1 is ready to be played.

3. Now press the 'Fast Forward' button (variously labelled 'FF', 'F FWD' or '>>') on the tape recorder and observe the effect:

– If the tape winds forward, stop the tape recorder and fully rewind the tape – your equipment does **not** provide the motor control facility.

– If nothing happens, leave the tape recorder on 'Fast Forward', type:

```
*MOTOR 1 RETURN
```

and observe the effect:

– If the tape now begins to wind forward, stop the recorder, fully rewind the tape and (only when the tape is rewound) type:

```
*MOTOR Ø RETURN
```

Your equipment **does** provide the motor control facility.

– If nothing happens, disconnect the lead from the tape recorder and check that it is operating correctly on its own. Then reconnect the lead carefully and repeat the procedure – if it fails again you should consult your supplier or an Acorn dealer.

Tone and volume settings

In order to be reasonably sure of being able to load the programs from the Welcome tape successfully, you should first adjust the tone and volume settings on your tape recorder. For the majority of modern tape recorders, you should select maximum 'treble response' (i.e. turn the tone control to its maximum) and set the volume at about one third of its maximum.

Programs recorded on cassette tape consist of a series of rather unpleasant sounds and, in many tape recorders, this sound is played back through the internal speaker at the same time as it is transmitted along the connecting lead. If possible, therefore, you should also switch the tape recorder's internal speaker OFF. If the speaker cannot be switched off, an equivalent effect can

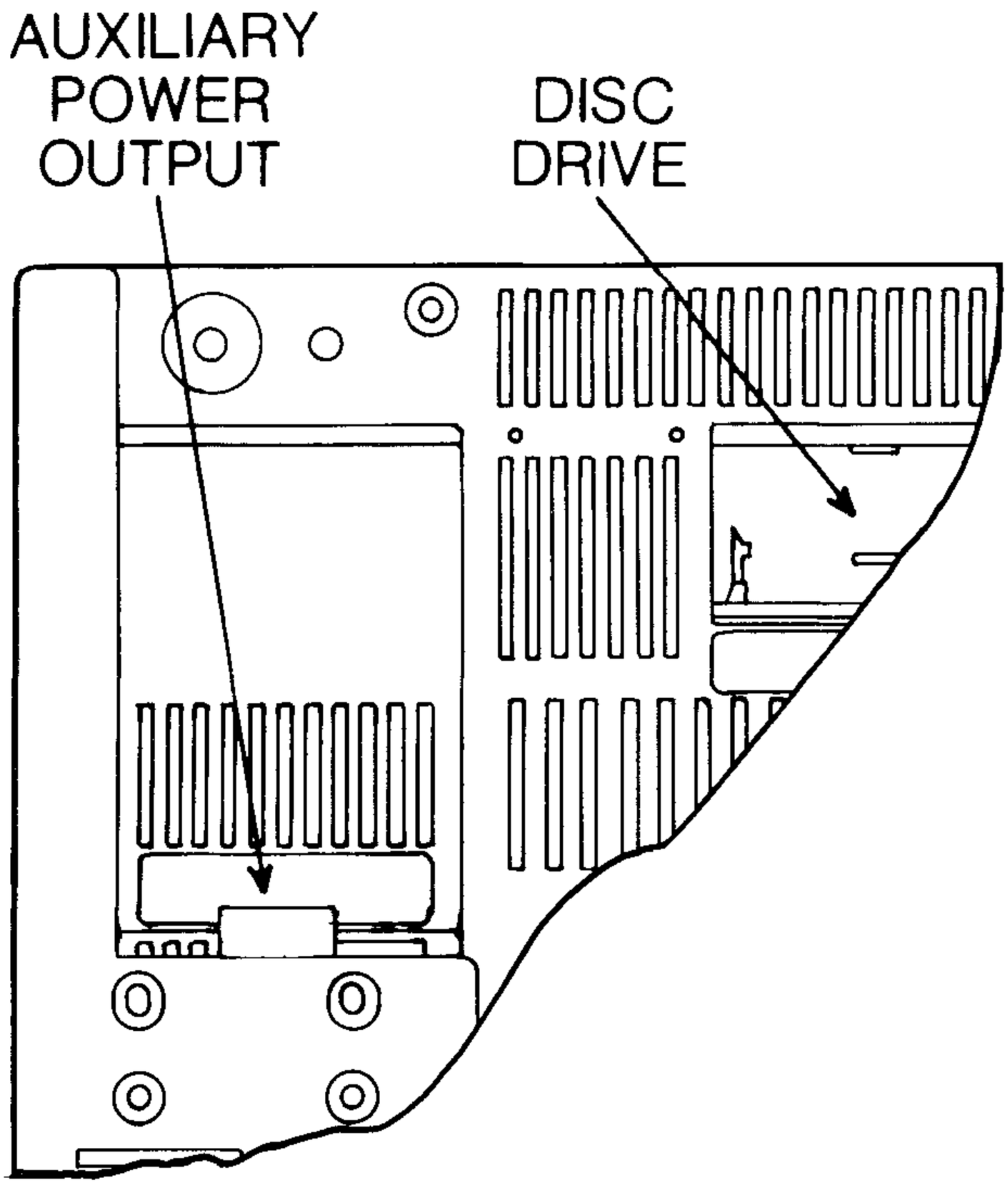
often be achieved by inserting a jack plug into the socket marked 'ear', if it is not already occupied.

You must not turn the volume control down unless you can be absolutely certain that it affects ONLY the speaker and, if all else fails, you will have to muffle the speaker with some suitable material.

Connecting a disc unit

Chapter 8 contains instructions for connecting a variety of disc units to your computer. In order to run the Welcome programs and utilities from the disc provided, your disc unit must accept either 40- or 80-track 5.25in flexible (floppy) discs.

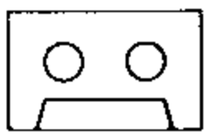
The disc unit will have a flat connecting cable and connector which should be inserted into the socket marked 'disc drive' on the front underside of the computer:



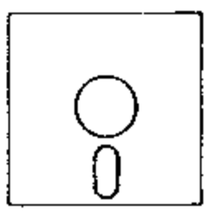
Some disc units are mains powered (in which case they will have a mains power lead and a separate ON/OFF switch); other units draw their power from the computer itself and the power cable should be connected to the auxiliary power output socket shown in the illustration.

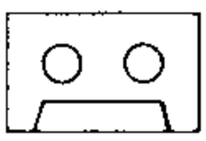
Running the Welcome programs

All the Welcome programs are written in BBC BASIC and in this section you will encounter the keyword CHAIN, which is used to load and run BASIC programs from either tape or disc. However, whilst your computer is fitted with the necessary circuitry to enable it to control the operation of a disc unit, it has been set up (initially at least) to handle only input from a tape recorder, under the control of what is called the **Cassette Filing System** (CFS). Instructions below relating specifically to the CFS are indicated by this symbol:



Disc users, on the other hand, will need the **Advanced Disc Filing System** (ADFS) and the necessary instructions are indicated by:





Load the Welcome cassette into the recorder so that side 1 can be played back. Reset the tape counter, then type:

```
CHAIN"WELCOME" RETURN
```

and press the PLAY button on the recorder.

The screen will first show the message:

```
Searching
```

then:

```
Loading  WELCOME
```

When program WELCOME has been loaded, a short bleep will be emitted from the computer's speaker, the screen will fill with a title page and you will be asked the question:

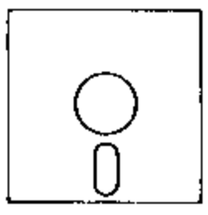
```
Do you have motor control ? (Y/N)
```

If your equipment provides motor control, press `Y` and leave the PLAY button on the recorder depressed – the computer will switch the motor on and off automatically. If your equipment does not provide motor control, press `N` and be prepared to press the STOP button on the recorder when told to do so throughout the Welcome sequence.

A bleep accompanied by one of the messages:

```
?Block , ?Data or Rewind tape
```

indicates a failure to read the contents of the tape correctly and you should start again, using a different volume setting.



Remove the disc from its protective jacket and insert it into the drive labelled (or specified in the disc drive's documentation) as drive 0. Leave the disc drive latch open and then type:

```
*ADFS[RETURN]
```

Wait for the disc drive to whirr and for the lamp adjacent to drive 0 to come on, then close the latch. There will be a short delay while the ADFS retrieves essential information from the disc and, when the > prompt reappears, type:

```
CHAIN"WELCOME40"[RETURN]
```
(for a 40-track disc unit)

or:

```
CHAIN"WELCOME80"[RETURN]
```
(for an 80-track disc unit)

This will bring up the title page; the remaining programs will be loaded from the disc automatically.

About the Welcome programs

As each of the Welcome programs is loaded, make a note of the tape counter value in the blank, rectangular box adjacent to each program name below. This will enable you to locate a particular program quickly and easily if you wish to look at it again.

MODES ☐  loading time 2 minutes

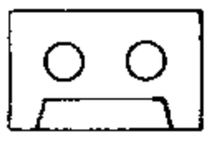
MODES cycles through the 8 basic screen modes and displays examples of the text, the available colours and, where possible the basic graphics capability.

CASTLE ☐  loading time 1 minute

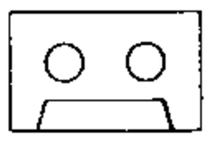
CASTLE illustrates the computer's ability to produce high-speed, multi-colour graphics. It uses a variety of shapes (squares, rectangles, circles and triangles) filled with either plain colours or patterns.

CLOWN ☐  loading time 1 minute

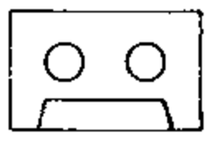
CLOWN is a similar illustration which incorporates other shapes.

SHAPES ☐  loading time 1 minute

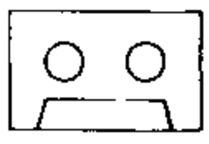
SHAPES is a sequence of examples showing the basic shapes which can be produced directly using built-in graphics commands. For the purposes of the demonstration, each shape is drawn as a solid figure, each superimposed upon the previous one, but it is also possible to produce outline shapes using solid or broken lines.

CLOUD ☐  loading time 1 minute

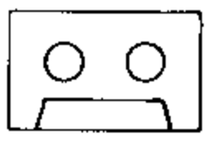
CLOUD is a simple animated sequence in which various parts of a graphic image are moved about the screen. The smoothness of movement is achieved by switching between the normal and the *shadow* screens. Changes of colour are used to produce a pleasing effect.

PATTERNS ☐  loading time 1 minute

PATTERNS produces a sequence of complex figures, facinating to watch in themselves, but which are then used to illustrate the speed with which the computer can *flood-fill* an area with either a plain colour or a more complex pattern.

KEYBOARD ☐  loading time 2 minutes

KEYBOARD is an program designed to help you to familiarise yourself with the operation of the keyboard. You will be shown a character which you must find and press; the computer will time you and display your score out of ten in each of five different tests, together with the average time that you take to find each key.

TURTLE ☐  loading time 1 minute

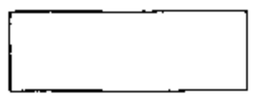
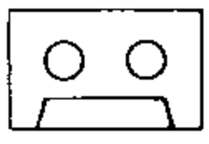
TURTLE is a program which allows you to control the movement of a screen pointer (the turtle) using a simple set of commands. The turtle normally leaves a trail in the form of a simple line and this feature can be used to create a number of interesting graphic effects. In fact, the nine red function keys are preprogrammed with sequences of commands to produce different shapes and patterns.

The commands which may be given to the turtle are shown below – you may also use the abbreviation given in brackets.

FORWARD (FD) *n*	moves the turtle *n* steps in the direction in which it is pointing.
BACK (BK) *n*	moves the turtle *n* steps backwards.
RIGHT (RT) *a*	turns the turtle right through *a* degrees.
LEFT (LT) *a*	turns the turtle left through *a* degrees.
HOME	returns the turtle to its starting position.
PENUP (PU)	stops the turtle leaving a trail.
PENDOWN (PD)	makes the turtle leave a trail.

PENCOLOUR (PC) *c*	changes the colour of the turtle's trail: PENCOLOUR 0 leaves a blue trail; PENCOLOUR 1 leaves a red trail; PENCOLOUR 2 leaves a yellow trail; PENCOLOUR 3 leaves a white trail.
HIDETURTLE (HT)	makes the turtle invisible.
SHOWTURTLE (ST)	restores the turtle to the screen.
CLEAN (CL)	wipes the screen clean.
CLEARTEXT (CT)	clears the area where commands appear.
REPEAT *n* [....]	enables a sequence of commands to be repeated *n* times. For example: REPEAT 4 [FORWARD 100 RIGHT 90] will draw the four sides of a square.

CTRL may be used to interrupt a sequence of commands.

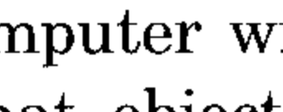
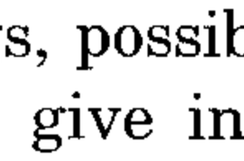
ADVENTURE   loading time 3 minutes

ADVENTURE is an adventure game in which you must explore a world revealed to you by the computer – the aim is to find the hidden treasure.

The computer will describe your surroundings, possible routes you may take and what objects (if any) are to hand. You give instructions using simple commands of one or two words. For example, to 'go north', you could type GO NORTH or simply NORTH. (In fact, NORTH, SOUTH, EAST, WEST, UP and DOWN can also be abbreviated to N, S, E, W, U and D respectively.) You can collect any objects you come across (such as a key) by typing TAKE KEY or GET KEY.

IF you type INVENTORY (or simply INV) you will be given a list of the objects which you are carrying.

Do not be afraid to experiment with a wide range of words – you may be surprised to learn how many commands the program can understand !

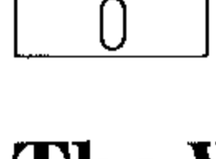
AQUA   loading time 2 minutes

Aqua attack is an arcade-style game in which you score points by destroying various objects in an under sea scene. You control the movement of the black submarine using either the keyboard or a joystick. (See Chapter 6 for instructions for connecting joysticks).

Points are scored for each direct hit on an object but some require several hits before they are destroyed. If you hit the rapidly-moving 'sea-snake', it splits

into two segments. You lose one of your three lives if you steer your submarine into any object or if you are caught by either the octopus or a falling mine !

AQUA is the last of the Welcome programs although a number of other items are contained on the second side of the cassette. At this stage, however, you may wish to see some or all of the programs again. First reset the computer with [CTRL]+[BREAK], then proceed as follows:

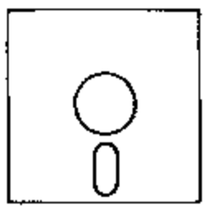


You will need to rewind the tape either to the start or to the appropriate tape counter setting. If your equipment provides motor control, you will be unable to rewind the tape until you switch the motor ON by typing:

```
*MOTOR 1[RETURN]
```

When the tape is correctly positioned, execute the program of your choice by typing:

```
CHAIN"program name"[RETURN]
```



With the disc correctly loaded in the disc unit, merely type:

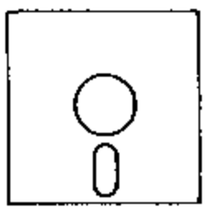
```
CHAIN"program name"[RETURN]
```

The Welcome utilities

The Welcome disc and side 2 of the Welcome cassette contain a number of utility programs. Three utilities, PANEL and TIMPAINT and DBASE are described here, the remainder, which provide facilities which you may wish to incorporate into your own BASIC programs, are described at the end of Chapter 2.

The procedure for loading each utility is given under the appropriate section heading.

If you have just finished running the Welcome programs you will need to wind the tape forward (using `*MOTOR 1` if necessary), remove the cassette and replace it so that side 2 is ready to be played. Remember to reset the tape counter.



You may use the commands described below to run any of the Welcome utilities individually. However, you may call up a menu system which provides access to all the utility programs by typing:

```
*ADFS[RETURN]
CHAIN"UTILITIES"[RETURN]
```

You should refer to Chapter 2 for a description of the Programming utilities and to Chapter 5 for a description of the Advanced Disc Filing System utilities.

PANEL | | loading time 2 minutes

PANEL is the first of the utilities provided with your computer – it is described here because it enables you to carry out such functions as resetting the date and time and, if necessary, reconfiguring your machine so that it selects something other than the Cassette Filing System when it is switched on.

PANEL differs from the other welcome and utility programs in that it is written, not in BASIC, but in the computer's own **machine code**. CHAIN, which is the command for loading and running BASIC programs, is therefore inappropriate and PANEL is executed using the command:

```
*RUN PANEL RETURN
```

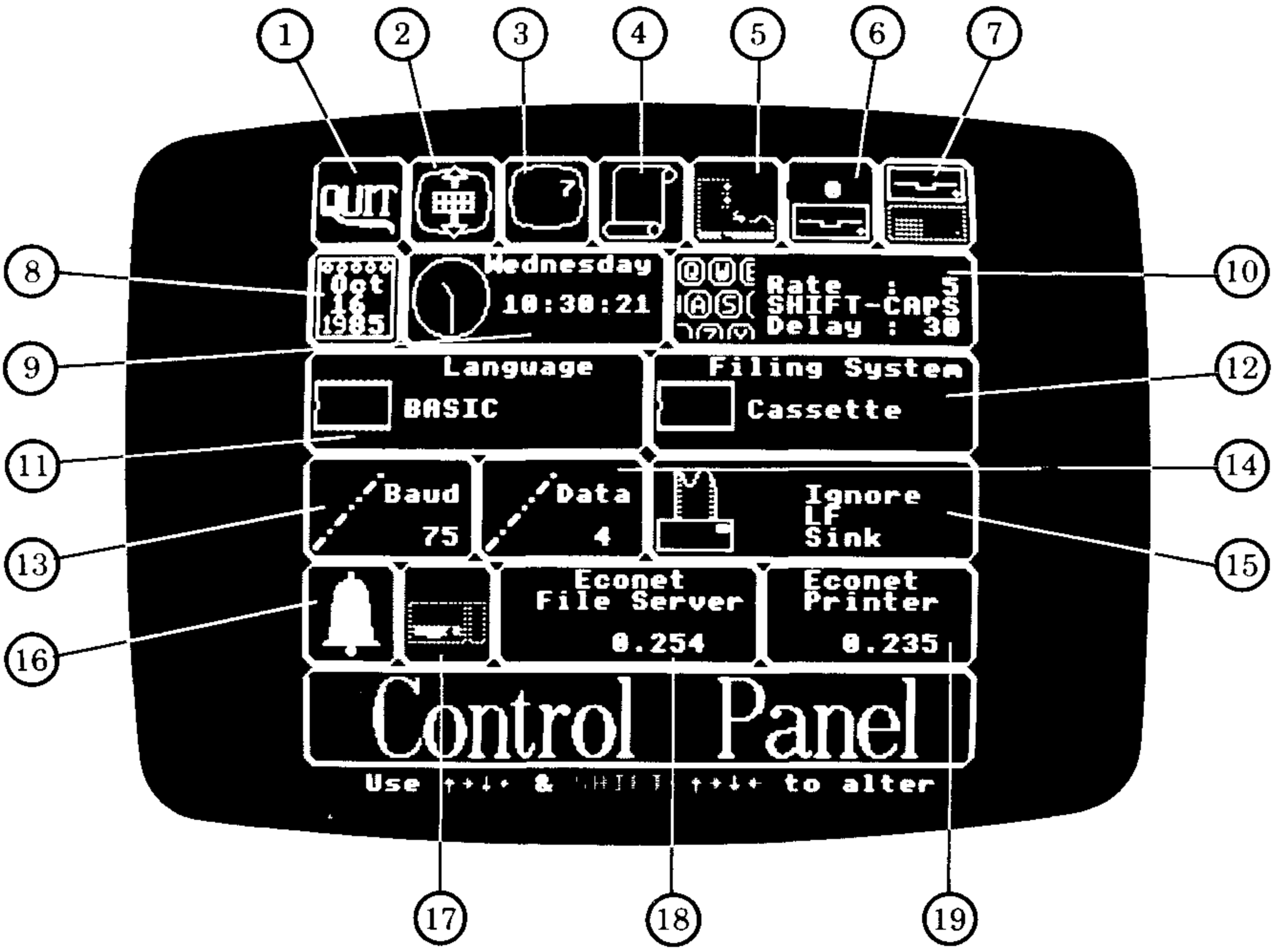
*RUN PANEL may be abbreviated to *PANEL if you are using a disc system. Since PANEL is not one of the standard commands recognised by the MOS, this line causes the computer to search the disc for a machine code program with the name PANEL.

Once loaded, PANEL replaces the screen with a **control panel** consisting of a series of boxes showing the settings which are selected when the computer is switched on. These settings are held in a special, battery-backed memory referred to as the CMOS RAM and they are maintained even when the computer is switched off.

If your equipment does not provide motor control, stop the tape recorder as soon as the control panel appears. Note the reading of the tape counter – you will need it when you come to examine the remaining utilities.

WARNING: READ THE FOLLOWING INSTRUCTIONS FULLY BEFORE ATTEMPTING TO USE THE CONTROL PANEL. FAILURE TO DO SO MAY RECONFIGURE YOUR MACHINE INTO A STATE FROM WHICH YOU MAY FIND IT DIFFICULT TO RECOVER.

The control panel layout is shown below.



The content of each of the boxes may be changed and the box to which changes will be made at any one time is outlined in blue; all the remaining boxes will have a white outline. Initially, the box marked QUIT (1) is the one so marked and depression of [ESCAPE] while QUIT is highlighted terminates execution of the control panel program.

Movement between the various boxes is achieved using the four cursor control keys (←, →, ↑, ↓) and any necessary changes are made by using the same keys in conjunction with simultaneous depression of either of the [SHIFT] keys.

The meaning of the various symbols shown in the control panel, a list of the possible options and a discussion of their functions are given below. The 'default settings' are those incorporated into the machine prior to its purchase.

(1) QUIT

This box provides a means of leaving the control panel, as descibed above.

(2) Vertical screen alignment

On some domestic televisions and (fewer) monitors, the top line of the screen display is either totally or partially lost and selection of this option allows you to shift the whole screen display up or down as necessary.

Default setting: no vertical screen shift

(3) Mode select

The computer can be configured to start up in any of the available modes (0 – 7 or 128 – 135).

Default setting: mode 7.

(4) Scroll-protect option

Under normal circumstances, the cursor will automatically **wrap around** to the start of the next screen line whenever a particular line becomes full. If the line happens to be the bottom line on the screen, the wrap around also causes the screen to be **scrolled** up by one line (i.e. the top line disappears from view and a new, blank line is created at the bottom.

This feature does not normally cause problems but one side-effect is that it is impossible to type a character in the last position on the last line without invoking the scroll. This option allows you to overcome this minor irritation by effectively creating an 81st character position (or a 41st, depending on the current mode) on the last screen line – this character position is used to hold the cursor on the last line if a character is placed in the 80th (or 40th) position.

Default setting: no scroll-protect.

(5) Boot option

Disc and network users have the option of selecting the *auto-boot* option, which enables a predefined sequence of actions to be carried out whenever SHIFT+BREAK is depressed. Normally, this involves executing the instructions contained in the file with the special name !BOOT, held either on the disc or in the user's own area of the network file server. This option has no effect for users of the Cassette Filing System.

Default setting: auto-boot OFF.

(6) Drive control parameters

This option allows the user to make adjustments to the way in which the disc drive(s) connected to the computer are controlled. The value is a decimal representation of a sequence of binary digits (bits), each of which is used to indicate a particular setting, further details of which will be found in the technical documentation accompanying your disc unit. The default setting is appropriate to most commonly-available disc units.

Default setting: 3

(7) Disc-type select

Your computer is capable of controlling the operation of most industry-standard disc drives, i.e.:

– 5.25in flexible ('floppy') disc drives;
– 3.5in or 3in mini-disc drives;
– 'Winchester' (hard) disc drives.

In this option, the top symbol denotes 5.25in flexible disc, 3.25in or 3in mini-disc drive(s); the lower symbol denotes a Winchester drive.

In addition, each symbol contains a small indicator which may be switched ON or OFF to indicate whether the Advanced Disc Filing System (see Chapter 5) will carry out an automatic start-up sequence.

Default setting: 5.25in flexible disc, 3.25in or 3in mini-disc drive(s). No automatic initialisation.

(8) Date

The CMOS RAM holds and maintains a perpetual calendar which caters for all dates (including leap years) until the year 2000. This option allows the day, month and year numbers held in the RAM to be altered if required.

Default setting: not applicable.

(9) Day and time

In addition to the day month and year numbers, the CMOS RAM holds a representation of the actual day and the current time. Both can be altered by selecting this option.

Default setting: not applicable.

(10) Keyboard status

This option allows the power-on setting of the *caps lock* and *shift lock* indicators to be defined. It also allows the auto-repeat rate and the initial auto-repeat delay to be specified.

Default settings: auto-repeat rate one tenth of a second;
caps lock ON;
auto-repeat delay half a second.

(11) Default language

In its standard form, your computer comes equipped with the BASIC language, the VIEW word-processor, the ViewSheet spreadsheet package and the Acorn Editor. It is possible (using the various internal ROM sockets and the two external cartridge ROM sockets) to install other language systems, such as PASCAL, COMAL, LOGO etc. This option allows any of the available language systems, VIEW, ViewSheet or the Editor to be selected when the computer is switched on.

Default setting: the BASIC language.

(12) Default filing system

In its standard form, your computer comes equipped with four different filing systems:

- The Cassette Filing System (CFS);
- The ROM Filing system (RFS);
- The Disc Filing System (DFS) – supplied for compatibility with BBC Model B microcomputers;
- The Advanced Disc Filing System (ADFS).

and this option allows any of the filing systems to be selected when the computer is switched on.

The ROM Filing System must be selected if you wish to use software supplied on cartridge ROMs (i.e. which plug into either of the two external cartridge ROM sockets). The Disc Filing System is fitted in order to provide compatibility with previous versions of the BBC Microcomputer (i.e. Models B and B+).

Default setting: Cassette Filing System

(13) RS423 baud rate

The RS423 serial interface is used, amongst other things, for connecting serial printers and it is necessary to specify the rate at which data is to be transmitted. This option defines the transmission (baud) rate which will be selected when the computer is switched on.

Default setting: 1200 baud

(14) RS423 data format

As with the Drive control parameters described above, this value is a decimal representation of a sequence of bits used to define the format of data transmitted or received via the RS423 serial interface. Users wishing to make extensive use of the RS423 interface should consult the Reference Manual for further information on this setting.

Default setting: 4

(15) Printer options

A variety of different types of printer may be connected to the computer, some of which produce automatic *line-feeds*, others of which do not. Those which do produce an automatic line-feed will print in *double spacing* (i.e. with a blank line between each printed line) unless the printer is instructed to ignore the line-feed character generated by the computer. This option allows you to choose whether line-feeds (or, indeed, any other character) will be ignored.

This option also allows you to specify the type of printer currently in use, i.e. parallel, serial or (in the appropriate environment) a network printer. If you do not have a printer connected to the computer, you may also select the *printer sink* option, which ensures that programs will not fail if they attempt to direct output to a printer which is not connected or not switched on.

Default setting: ignore line-feeds; parallel printer type.

(16) Sound option

The computer produces sounds through its internal speaker and this option allows you to select either full or half volume for standard tones. Selecting half volume has the additional effect of suppressing the sounds generated when the computer is switched on.

Default setting: full volume.

(17) Co-processor options

Chapter 8 of this guide describes a number of **co-processor** options which provide your computer with increased power and versatility. This option allows you to specify if a co-processor is to be selected when the computer is switched on.

The large symbol denotes a co-processor installed inside the computer's casing (i.e. an internal co-processor) and the smaller symbol to its right denotes an external co-processor unit (such as a BBC Microcomputer System Second Processor).

Default setting: co-processor not selected.

(18) Default file server (Network users only)

If your computer is connected to a network of other BBC computers (using the optional Advanced Network Filing System (ANFS)), your storage and communication needs will be met by one of the computers – referred to as the **file server**. This option allows you to specify which network file server is to be selected when the computer is switched on.

Default setting: 0.254 (i.e. station number 254 in network 0)

(19) Default printer server (Network users only)

Most networks incorporate a **printer server** which is a computer dedicated to controlling the printed output from all stations in the network. This option allows you to specify the identity of that network station.

Default setting: 0.235 (i.e station number 235 in network 0)

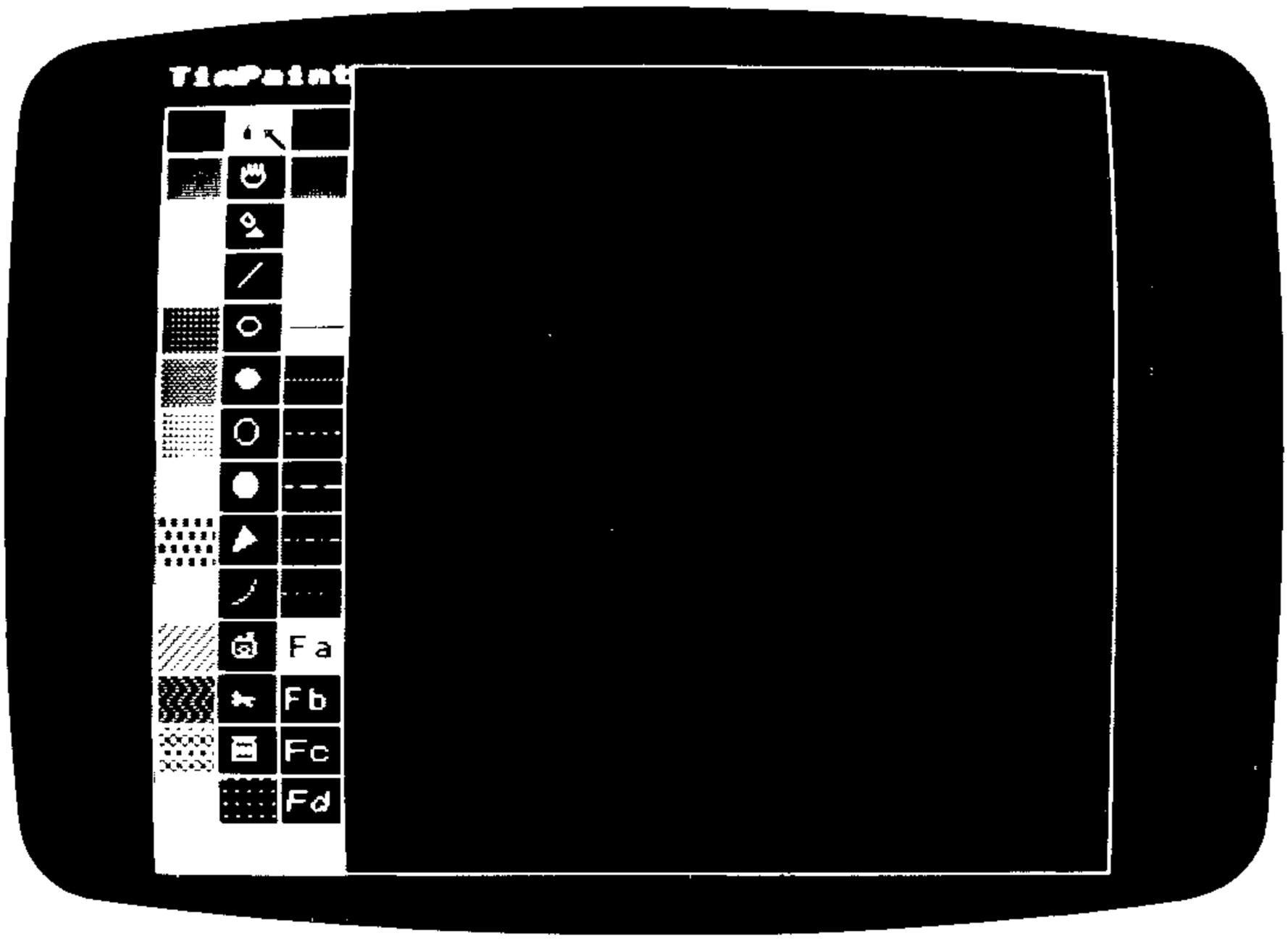
(References to 'network 0' in items 18 and 19 mean 'the normal network' – it is

possible to link different networks together by means of **bridges** and in these cases, each individual network has a unique number.)

TIMPAINT [ ] [cassette] loading time 3 minutes
This program allows you to create and save your own pictures using many of the advanced graphics features provided by your computer. To load TIMPAINT, type:

```
CHAIN"TIMPAINT"[RETURN]
```

Once loaded, TIMPAINT shows the following display:



The boxed area on the left is the **menu**, from which you select the various functions, colours and typestyles (**fonts**) that you wish to use. The larger, empty area to the right is the 'canvas' on which you create your artwork.

The menu is divided into 3 columns:

- the leftmost column determines which colour will be used for all subsequent operations, the one selected being shown in the larger rectangle at the bottom of the menu;
- the middle column contains all the available functions, each identified by a special symbol, such as camera, typewriter, scissors etc, each of which is described below;
- the rightmost column is further subdivided:
  - the top four boxes contain the colour 'palette';

- the boxes immediately below the palette contain the four types of line which can be used;
- the bottom four boxes show which font will be used when text is placed on the screen.

Selections from the menu are made by moving the arrow pointer to the required box and pressing [CTRL]. Slow movement is provided by the four cursor control keys; fast movement (eight times normal) is provided by the simultaneous depression of [SHIFT]. (It is also possible to operate TimPaint using a joystick, in which case the 'fire' button replaces the function of [CTRL]. See Chapter 8 for instructions on connecting a joystick to your computer.)

The current menu selections are normally highlighted and, when TIMPAINT is first loaded, the selections are:

- black background;
- white foreground;
- spray can option (see below);
- joysticks off;
- grid off;
- solid lines;
- normal font (Fa).

Thus, to select a background other than black, move the pointer to the box containing the colour or pattern of your choice and press [CTRL]; the box at the bottom of the menu will then fill with your selection.

You can change the palette (i.e. the range of available colour combinations) using the four boxes at the top of the third menu column. For example, to change red to green, move the pointer to the red box and keep pressing [CTRL] until the box shows green.

Similarly, the drawing function, line and type styles are selected by moving the pointer to the appropriate box and pressing [CTRL].

Whenever you move the pointer outside the menu area the arrow is replaced with the symbol denoting the function you have selected. The various procedures are described below, on the assumption that the corresponding function has already been selected from the menu.

The spray gun allows you to draw one or several lines at a time, each dot of the spray leaves one line behind it when it moves. Press [COPY] to increase the number of dots and [DELETE] to reduce it. To use the spray gun, move to the place where you want to start your line and press [CTRL]. Then, with [CTRL] held down, move around the screen and release [CTRL].

The hand will move the whole screen in any direction. To start, move the hand symbol to a readily identifiable point on the screen and then press [CTRL]. With

[CTRL] held down, move the hand to the position to which you wish to move the original point and release [CTRL] – the whole screen will then be moved. Note that any part of the picture which is shifted off the screen will be lost.

The flood fill option can be used to fill any enclosed area of the screen with the current colour. Simply move to any point within the area you want to fill and press [CTRL]. Note that if you try to fill an outline which has a gap in it then the colour will escape out of the gap and carry on until it hits a solid boundary or the edge of the screen. Areas can be 'unflooded' by pressing [TAB]. Sometimes, however, this operation does not only reverse the action of the flood but affects other shapes on the canvas, depending on the colours used.

The line allows single lines to be drawn anywhere on the canvas. Press [CTRL] to start a line and release it when you are happy with its position.

Ellipse outlines and solid ellipses are produced by moving to the point which is to be the centre of the ellipse and pressing [CTRL]. Then, while holding [CTRL] down, the width and height of the ellipse can be altered using the arrow keys.

Circle outlines and solid circles are drawn in a similar manner – press [CTRL] to indicate the centre and alter the position of the circle symbol to produce the size of the circle you want – then release [CTRL].

The camera allows copies of any rectangular area of the canvas to be made. Move the symbol to one corner of the area and press [CTRL]. Then hold down [CTRL] whilst moving the cursor keys to increase the depth and width of the box. Release [CTRL] when you have enclosed the area to be copied. Pressing the cursor keys now will move a second box which should be placed where you want the copy to be put. The copy is made by pressing [CTRL].

The scissors will move a rectangular area of the screen, replacing it by a block of the background colour. This is preformed in the same manner as the copy routine described above.

The polygon allows a series of lines to be drawn, each one beginning where the previous one finished, thereby producing a continuous line drawing. Press and release [CTRL] to begin and then each time a line is to be drawn.

The typewriter can be used to print text on the screen. On the canvas, the typewriter symbol is replaced by a 'pencil' and the start point for the the text is identified by pressing [CTRL]. Any subsequent key depressions produce characters in the screen in the current font and the end of the text is marked by pressing [RETURN]. Note that text can only be placed between the starting point and the right hand side of the canvas – it is not allowed to wrap round to the beginning of the next line.

When selected, the joystick option allows the movement of the symbols to be

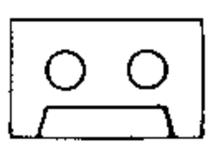
controlled by a joystick rather than by pressing the arrow keys. In addition the fire button replaces the [CTRL] key.

The grid option restricts movement of the symbols to positions in an invisible grid on the screen, making it easier to draw several circles with the same centre point etc.

Loading and saving can be carried out by pressing L or S. This clears the menu area and allows you to type in the name you wish to use.

To clear the screen and start again press [CTRL]+[TAB]. This will reset all the options to their initial values.

DBASE

 Loading time 2 minutes

This program is a very simple database which has been set up in the style of a card-index address book. To load DBASE, type:

```
CHAIN"DBASE"[RETURN]
```

Each 'card' contains 4 slots; one each for the name, address, telephone number and birthday of an individual. Details of up to 100 people can be stored.

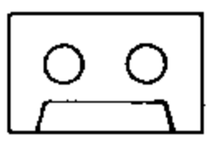
You can move through the database using ← and → which will move you on back and forward one card respectively.

To move between the different slots within a particular card use ↑ and ↓. The current slot is always highlighted in black.

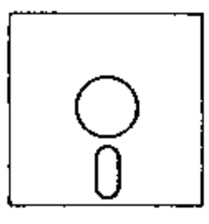
Initially the address book contains no information, each of the slots in each card is blank. To enter information for a person you should move to, and then edit each slot in turn by pressing E. Any characters you type will be placed in the current slot until the slot is full or you press [RETURN]. The maximum number of characters for each of the slots is:

name	20 characters;
address	80 characters;
telephone number	12 characters;
birthday	12 characters.

You can SAVE the information currently held in the database by pressing S at any stage other than entering or editing a slot. S saves the complete database with the name FILE.



Do not attempt to SAVE a database to the Welcome cassette itself – you will overwrite the next utility program.



The new file will automatically overwrite the previous version.

Pressing L will LOAD the information contained in FILE into the computer.

If you wish to edit the contents of an existing slot you can do it in a similar way to entering new details – when you press E the current slot will be cleared so that your new data can be typed in.

Once you have created your database, you can use the 'find' facility to select cards satisfying specific criteria. For example you can find someone's telephone number by entering their name or you can find the names and addresses of, say, all the people who have birthdays in the next month.

To do this you should initially move to the slot which corresponds to the piece of information you know. In the first example this would be the 'name' slot, in the second example it would be the slot marked 'Birthdy'. Press F and type in the information you know, (for example SMITH or AUGUST) and press [RETURN].

The computer will FIND the cards you are looking for. A message at the bottom of the screen will say how many cards have been found and you will be moved to the first one. Now when you use ← and → you will move only between the cards found for you by the computer.

If you do a further search then this will only be carried out on the cards found previously. Hence a search for those named SMITH followed immediately by a search for those whose address is in LONDON will find all the people who are called SMITH and who live in LONDON.

Note that the computer will treat upper and lower case letters as the same (i.e. BELL is considered to be identical to Bell) and that it looks for an exact match with what you have typed. Hence, searching for D BELL will not find entries like D J BELL or DAVE BELL.

If you wish to return to the whole file then press R. This will restore the file for you. The file is automatically restored if no cards are found in a search.

2. The BASIC Language

Writing a program

Languages such as English are too ambiguous to be used for communication with a computer. Instead, all instructions are given using a computer language consisting of just a few hundred words that the computer can interpret.

Your computer comes complete with the powerful and flexible computer language, BBC BASIC. This is composed of a number of English-like words, which make the language easy to learn and use. (You may already be familiar with some other computer language such as Pascal. Chapter 8 describes how you can expand your system to include such a language).

In the last section you learnt that the computer can obey some commands immediately. For example, if you type:

```
PRINT "Hello"RETURN
```

the computer displays the the word Hello.

PRINT is a BBC BASIC **keyword** that the computer recognises. It tells the computer to display on the screen whatever follows the PRINT statement. The most important BASIC keywords are described in this section of the book, and a full list of all the keywords and their meanings is given in Appendix F.

You may already have found out what happens if you give a command to the computer that it cannot interpret. For example, if you type:

```
PRINT "HelloRETURN
```

the computer responds with the message:

```
Missing "
```

The computer gives an error message to show that it cannot obey your command because you have not followed the rules of the BASIC language. It is easy to make mistakes when giving the computer instructions, and error messages are helpful in tracking down and correcting these mistakes.

If you want the computer to carry out a calculation in BASIC, you can use either the normal keyboard or the numeric keypad. Try:

```
PRINT 8+7RETURN
PRINT 20-9.5RETURN
```

Multiplication involves using the * symbol, and division, the /:

```
PRINT 12*9 RETURN
PRINT 25/2 RETURN
```

Basic contains many other arithmetic functions which can be used to find things like the square root of a number, or calculate its logarithm. Try typing:

```
PRINT SQR(9) RETURN
PRINT LOG(75) RETURN
```

If you intend to use your computer mainly for carrying out many complex calculations, you may find your needs are better met by ViewSheet which is described in Chapter 4.

The screen is looking rather cluttered now, so type:

```
CLS RETURN
```

which is the BASIC instruction to clear the screen.

You have been giving commands which the computer obeys immediately. More commonly, you will give the computer a series of numbered instructions to obey. These instructions are stored in the memory and are called the program. The computer only obeys program instructions when you want it to do so.

You can see the difference between the methods of giving instructions by typing:

```
10 PRINT "Hello" RETURN
```

This time nothing happens and the > prompt reappears.

At the start of the line you typed the number 10. This is called the **line number**, and it tells the computer that the statement which follows is not to be obeyed immediately. Instead the line is stored in the memory, as you can see by typing:

```
LIST RETURN
```

Your one-line program is listed on the screen. You can make the computer carry out or execute this very short program by typing:

```
RUN RETURN
```

Once the computer finishes executing the program, the > prompt returns to the screen. This shows that the computer is ready to accept further commands at the keyboard. The program is still stored in memory, as you can confirm by typing:

```
LIST RETURN
```

If you add another line to the program the computer automatically puts the lines into line number order. For example, type:

```
3 PRINT "This is another line"[RETURN]
```

and LIST the full program.

Program lines are usually numbered in tens as this makes it easy to insert extra lines later. If you type:

```
RENUMBER[RETURN]
```

the computer automatically renumbers the program, making the first line 10.

Once a program is complete it can be saved onto cassette or disc so that it can be used again. The Welcome software contains a series of programs which have been saved in this way. You probably don't want to save the present program, so type:

```
NEW[RETURN]
```

which tells the computer to 'forget' the program – you can confirm this by trying to LIST it.

You may accidentally lose a program by pressing [BREAK], or by typing NEW before you realise that you have not saved a copy of the program. Normally, the program can be recovered provided no new program lines have been typed. Use the command:

```
OLD[RETURN]
```

to restore the old program.

A simple program using variables

Throughout this section and the remainder of the chapter, you will be required to type a number of short programs and, for clarity, we shall omit the [RETURN] symbol at the end of each line.

Type the following program in:

```
10 PRINT "Can you give me a number ";
20 INPUT yournumber
30 PRINT "You typed ";yournumber
```

and then RUN the program. The computer obeys line 10 and displays the question:

```
Can you give me a number ?
```

The question mark is added automatically by the execution of line 20. The

INPUT statement makes the computer wait for you to type something – in this case a number. Type:

6 RETURN

Once you have typed the number, line 30 is obeyed and the message displayed is:

```
You typed 6
```

Line 20 causes the computer to store your number in a **variable**, so-called because its value can vary. Here the variable is called *yournumber*. You can think of a variable as an internal pigeon-hole which the computer fills with a value, in this case 6.

Whenever the computer comes across any reference to *yournumber* in the program, it uses the current value of the variable. So line 30 causes the computer to print `You typed`, then find the value of the variable *yournumber*, and finally print that value, 6.

RUN the program again, inputting a different number, and watch the effect. *yournumber* is a numeric variable – it can be used to store the value of whole numbers, decimals, or negative numbers. Variables can be used in arithmetic – add these lines to the program and RUN it again:

```
40 PRINT "Twice ";yournumber;" is ";2*yournumber
50 PRINT "Subtract 5 from ";yournumber;" and you get ";yournumber-5
60 PRINT"Add 20 to ";yournumber;" and you get "; 20+yournumber
```

The value of a variable does not have to be input, it can be given directly. For example, type:

```
LET height=2.1 RETURN
```

Then type:

```
PRINT height RETURN
```

```
PRINT height*2 RETURN
```

You can change your program to include a LET statement by adding these lines:

```
35 LET yournumber=10
36 PRINT "But the new value is ";yournumber
```

LIST the program so that you can see the order in which the computer obeys the instructions, and then RUN the extended program.

In the versions of BASIC provided on some computers only very short variable names like Q or AB are allowed. BBC BASIC, on the other hand lets you use

long variable names, which makes a program easier to follow and easier to modify. For example, the following are all allowed statements:

```
LET LengthOfCarpet=7.56

LET costof3Tins=1.21

LET SPEED_OF_CAR=60
```

(the underline character is on the same key as the £)

Although all the examples above have LET before the variable name, its inclusion is optional. The example program runs just as well if you type:

```
35 yournumber=10
```

As LET is optional you will find it is omitted in most programs.

Whilst variable names may be of any length, they must obey a few simple rules:

- The variable must begin with an upper- or lower-case letter, the £ or underline character.
- The other characters can be upper- or lower-case letters, the £ or underline character, or numbers.
- Variables that begin with Basic keywords such as PRINT or LET are not allowed.

As all Basic keywords are capitalised, it is easy to avoid including keywords at the start of a variable name by using only lower-case letters in the variable. This also makes program listings more readable, as the variables stand out.

Integer variables

The variables described so far are known as **real** variables, because they can be used to store real numbers – those with a decimal point. A real variable can be used to store numbers with up to 9 figure accuracy.

The computer always uses the same amount of memory to store a real quantity, even if the number stored there is an **integer** (a whole number). Some programs only need integers, and using real variables wastes computer memory. It also slows the program down, because the computer will treat 9×8 as $9.00000000 \times 8.00000000$ with all the extra calculation this entails!

An integer variable is another sort of numeric variable, and is used to store only whole numbers in the range -2147483648 to 2147483647. Calculations with integer variables are much more rapid, and the variables themselves use less memory than real variables.

An integer variable always ends in a percentage sign, as shown in the program below:

```
10 PRINT "Type any number ";
20 INPUT whole%
30 PRINT "You typed ";whole%
```

RUN the program and input, say 4.5. The result shows why you must not use an integer variable unless you are certain that the value stored there will always be a whole number.

The variables A% to Z% are known as the **resident integer variables** and memory space is automatically given to these variables when the computer is switched on, so no extra memory is taken up if they are used in a program.

The computer loses the values of other variables after a program is run, but the values of A% to Z% remain unchanged, even after typing NEW or pressing [BREAK]. They provide a means of passing information from program to program. For example, the Welcome cassette uses a resident integer variable to tell each program whether or not the user's cassette recorder has automatic motor control.

There is one other special resident integer variable, @%. The value of @% is used to control the way the computer prints numbers. @% is described in more detail on page 57.

String variables

The variables described so far are numeric variables – they can be used only to store numbers. The computer can also store strings of characters (ie words and phrases) in what are called **string variables**. A string variable always ends in a dollar sign, as you can see in these examples:

```
Type_of_car$="Mini Metro"

CURRENCY$="Francs"

Weather$="Wet"
```

The characters within the inverted commas are called **strings**. Type in and run this brief program:

```
10 PRINT "What is your name ";
20 INPUT name$
30 PRINT "Pleased to meet you ";name$
```

The string variable name$ in line 20 is used to store any name typed in. The contents of name$ are printed out by line 30. A string variable can hold from zero to 255 characters. You can prove this for yourself by running the program a few times and inputting names of different lengths.

Any set of characters can be stored in a string variable, for example:

```
a_mixed_string$="123%.abc@*"
```

However, you cannot carry out arithmetic on strings, even if the variable contains only numbers. Thus, although:

```
example$="365"
```

is an acceptable string,

```
PRINT example$+5
```

is meaningless to the computer. The contents of a string variable are treated as a series of characters. You cannot reasonably carry out arithmetic on a house number or a shoe size, and numbers stored as a string fall into the same category.

Help that BBC BASIC can give you

BBC BASIC has many features to make programming simpler. You may already have made a few mistakes when typing the example programs. If not, type:

```
10 PRONT "This is a mistake."[RETURN]
```

and see what happens when you run the program. The most long-winded way of correcting the error is to type the entire line again. Alternatively you can edit or alter the line using the cursor control and [COPY] keys at the right-hand side of the main keyboard.

Press ↑. As soon as you press the key, the cursor splits into two – the flashing cursor is the **copy cursor**, which you can move around to copy text from elsewhere on the screen; the white block is the **write cursor**, showing where anything you type or copy will appear. The write cursor moves only after a character has been typed or copied.

Move the copy cursor around until it is underneath the first character in the erroneous line and then press [COPY] once. The '1' is copied into the character position indicated by the write cursor. Now press [COPY] key four times more to give:

```
10 PR
```

You do not want to copy the next character because it is incorrect. Type `I` at the keyboard, and it will appear on your new line then use → to move the copy cursor until it is under the N in `10 PRONT`. You can now copy the rest of the line to give:

```
10 PRINT "This is a mistake."
```

(If you make any errors when copying, you can use [DELETE] to remove the most recent characters on your new line). When you have copied the last character, press [RETURN], and the corrected version of the line will replace the old one.

You can move the copy cursor elsewhere on the screen at any time whilst copying, so you can copy sections from several different lines to create a completely new line. If you want to abandon editing a line half-way through, press [ESCAPE]. Do **not** press [RETURN], as your old line will be replaced by the partially-edited version.

It is worth spending some time learning how to edit lines, as it speeds up program-writing considerably. However, once you begin to write longer programs you will probably want to use the more powerful editing facilities provided by the Editor, which is described in Chapter 6.

AUTO

Earlier you saw that program lines are usually numbered in tens. This leaves plenty of free line numbers for any statements that are inserted later. If you wish, the computer can automatically number lines for you. Remove the current program using NEW and then type:

```
AUTO[RETURN]
```

The computer prints 10 and waits for you to type a statement. Type the following, remembering to pressing [RETURN] after each line. (You can still use the editing facilities: most of line 40 can be copied from line 20, for example).

```
10 PRINT "A short program"
20 PRINT "What is your first number ";
30 INPUT first
40 PRINT "What is your second number ";
50 INPUT second
60 PRINT first;" plus ";second;" gives ";first+second
```

After the last line the computer prints 70. As the program is complete, press [ESCAPE] – you no longer want the computer to generate new line numbers. You can now LIST or RUN the program.

AUTO can be used to begin numbering at any line number, with any interval in between. The default interval is ten, so AUTO 100 produces line 100, 110, 120 and so on. AUTO 15,1 would produce line numbers 15, 16, 17 etc.

LIST

You have already used LIST, but an extended LIST command is also available which is useful as a cross-reference in longer programs. Try typing:

```
LIST IF PRINT[RETURN]
```

and

```
LIST IF first[RETURN]
```

In other words LIST IF displays only those lines containing the specified sequence of characters.

DELETE

Sometimes you will find you need to remove lines from a program. Single lines can be deleted by typing the line number and pressing [RETURN]. A number of lines in sequence can be deleted using the DELETE command. Try typing:

```
DELETE 20,50[RETURN]
LIST[RETURN]
```

which deletes all line numbers from 20 to 50 inclusive

RENUMBER

If you have inserted many extra lines in a program you can tidy it up by using RENUMBER to spread the line numbers out at intervals of 10. Renumbering always begins from the first line of the program. Like AUTO, you can use variations such as RENUMBER 100,5 to make the first line 100 and successive lines 105, 110, etc.

REM

The REM statement enables you to put remarks within a program to remind yourself or others what parts of the program do. Sensible variable names can make a program largely self-documenting, but REMs are useful to summarise the purpose of a number of lines:

```
100 REM Lines 110 to 150 plot a circle

500 REM Find the largest number and print it
```

The computer ignores any line beginning with a REM statement when a program is run.

Minimum abbreviations

If you are not used to a keyboard you may find it tedious to pick out the correct letters to type PRINT, for example. The computer recognises BASIC keywords if they are spelt in full **or** if an allowed abbreviation is used. Type:

```
P."Hello"[RETURN]
```

This is exactly the same as:

```
PRINT"Hello"[RETURN]
```

and is obeyed as such. Similarly, I. is the abbreviation for INPUT. Use NEW to remove the current program, select AUTO line numbering and then type in the following program, which uses several abbreviated keywords:

```
10 P."Pick a number ";
20 I.choice
30 P."Number ";choice;"!"
40 P."A good selection!"
```

Now LIST the program – abbreviations used in program lines are expanded to their full length automatically when a program is listed.

The abbreviations for all BASIC keywords are given in Appendix F.

Using the function keys

Most of the keys on the keyboard print a particular character whenever they are pressed. Across the top of the keyboard are a group of red keys which act differently. They are called the **function keys**. Each key can be programmed to produce a character or string of characters when it is pressed.

For example, you can program [f0] to produce the word PRINT by typing:

```
*KEY0 PRINT[RETURN]
```

[f1] can be programmed to produce INPUT if you type:

```
*KEY1 INPUT[RETURN]
```

Now press [f0] and [f1] to see the effect. You will notice that after the characters have been printed the cursor remains at the end of the line. Sometimes it is useful to program a function key so that it behaves as if [RETURN] had been pressed after the characters are printed and this is achieved by including the characters !M in the key definition. For example:

```
*KEY2 LIST!M[RETURN]
```

causes the current program to be listed whenever [f2] is pressed. Some screen modes only print 20 characters per line, which makes a listing very difficult to read so it would probably be better to define [f2] so that the computer switches to mode 135, the most readable mode, before listing a program:

```
*KEY2 MODE135!MLIST!M[RETURN]
```

It is useful to write a brief program that defines the keys. This program can be loaded and run at the start of a computing session. The key definitions remain set until:

– the keys are redefined;

– a *FX18 command is given, which clears the keys;

– there is a hard break (i.e. [CTRL]+[BREAK])

Type in the following program, which sets all the function keys:

```
10*KEY0 MODE135¦MLIST¦M
20*KEY1 RUN¦M
30*KEY2 MODE
40*KEY3 PRINT
50*KEY4 INPUT
60*KEY5 COLOUR
70*KEY6 MOVE
80*KEY7 DRAW
90*KEY8 PLOT
100*KEY9 GCOL
```

Later you may want to use key definitions of your own, but you will find the above program useful in the next few chapters. The next section shows how you can save the program you have just written so that it is available whenever you need it.

Saving and loading programs

Most of the programs you have just typed in have been fairly short and do not really do anything worthwhile. It is therefore not really worth keeping a permanent copy on cassette or disc but, as you learn more about BASIC programming, you will probably want to keep versions of your masterpieces so that you can run them without having to retype all the instructions from scratch.

Making a permanent copy of a program is referred to as **saving** a program and the BASIC language provides a special command for this purpose. Its format is:

```
SAVE "name"[RETURN]
```

where *name* (which must be enclosed in double quotation marks) is something you choose to identify the program from all others.

The SAVE command works equally well if you are using cassette tape or disc as a storage medium and the only difference is the number of characters which a name may contain (see Chapter 5).

Note: You will need either a cassette (**not** the Welcome cassette) or a so-called **formatted** disc (again **not** the Welcome disc) if you wish to carry out the commands given below.

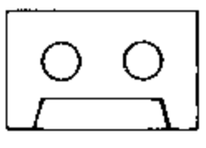
So, to SAVE the function key definition program you have just entered, you could type:

```
SAVE "KEYS"[RETURN]
```

or possibly:

```
SAVE "KEYDEFS"[RETURN]
```

or a SAVE command including any name you wish.



Load the cassette into the recorder and rewind it to the start of the tape (not the transparent 'leader') then type the save command of your choice.

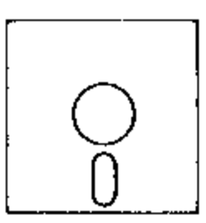
As soon as you press [RETURN] the computer responds with the message:

```
RECORD then RETURN
```

which is to remind you to press the RECORD button on the recorder – saving does not actually begin until [RETURN] is depressed.

There will be a momentary delay before the computer begins to store a copy of your program and a message giving the program name and an indication of its length is displayed while the transfer takes place.

A short bleep and the reappearance of the > prompt indicate that saving is complete and, if your equipment does not provide motor control, you will have to press the STOP button on the recorder.



Load the disc into the disc unit and then type the SAVE command of your choice.

As soon as you press [RETURN], the disc unit light comes on and the motor begins to whirr before the program is saved.

The > prompt reappears when the program has been saved.

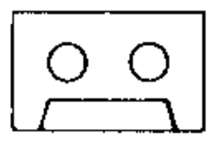
Note that SAVE merely transmits a copy of your program, it remains in memory for you to RUN, LIST or modify.

The process of retrieving a program from either cassette or disc is referred to as **loading** and, once again, the BASIC language provides a special command:

```
LOAD "name"[RETURN]
```

Clearly, the named program must exist on the medium in question.

Take a deep breath and remove your function key definition program from memory by using NEW.



Rewind the tape to the start. You will first have to use:

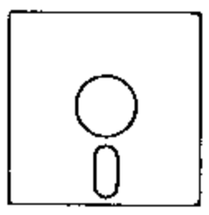
```
*MOTOR 1[RETURN]
```

if your equipment provides motor control. Then type the LOAD command containing the name of your function key definition program.

As soon as you press [RETURN], the computer will display the message:

```
Searching
```

and, after a short delay, you will see the name of your program together with a slowly-moving count. A short bleep and the reappearance of the > prompt indicates that loading is complete and, if your equipment does not provide motor control, you should press the STOP button on the recorder.



Simply type the LOAD command containing the name of your function key definition program and press [RETURN]. The > prompt will reappear as soon as the program is loaded.

LIST the program to prove that it has been retrieved.

Note that the LOAD operation replaces the current program, so you must be sure that you have SAVEd it if necessary.

To program or not to program

In the previous sections you have been introduced to a few of the BASIC programming facilities on the computer. You may be eager to learn more – in which case the next few sections are for you.

Or you may feel you have learned quite enough about programming. Is it really necessary to know so much before you can use the computer?

It is worth emphasising at this stage that it is up to you how you choose to use your computer.

Many thousands of people enjoy computing as a hobby. They write programs to play games or work out the monthly budget. They attend computer clubs and swap hints and tips with other enthusiasts.

Other computer owners never bother to learn beyond the rudiments of programming. When they want software to catalogue their stamp collection, they buy a pre-written program from a local shop or via mail order. Rather than struggling to write a program to play noughts and crosses they prefer to purchase complex games such as *Elite*, which have taken professional programmers months to produce.

Some computer owners play the occasional game but primarily use the computer for more serious purposes. They prepare and print out letters with the help of a word processor, use spreadsheets to help them make financial decisions, and store important information on tape or disc.

The computer is a tool; complex and sophisticated, but a tool nonetheless. Do not feel that you must learn to program to use it properly. Your computer contains other powerful facilities which are described in later sections. A wide

range of software to meet virtually any need is available – you do not need to program to find the computer a valuable and useful aid.

The next few sections demonstrate some of the possibilities of programming. They are intended only as an introduction to BASIC, but you will learn more from them if you experiment with the example programs that are listed. Change the values of the variables or add some lines of your own. Don't worry if you make a mistake that seems to keep the program going forever. You can always press [ESCAPE], which stops the program and brings back the > prompt to show you can again input instructions.

Simple graphics

The computer provides eight different screen display modes and the Welcome software showed you one of the most obvious differences between the modes – the number of characters that can be displayed on a line. The Welcome software also demonstrated how some modes allow both the printing of text and the display of graphics.

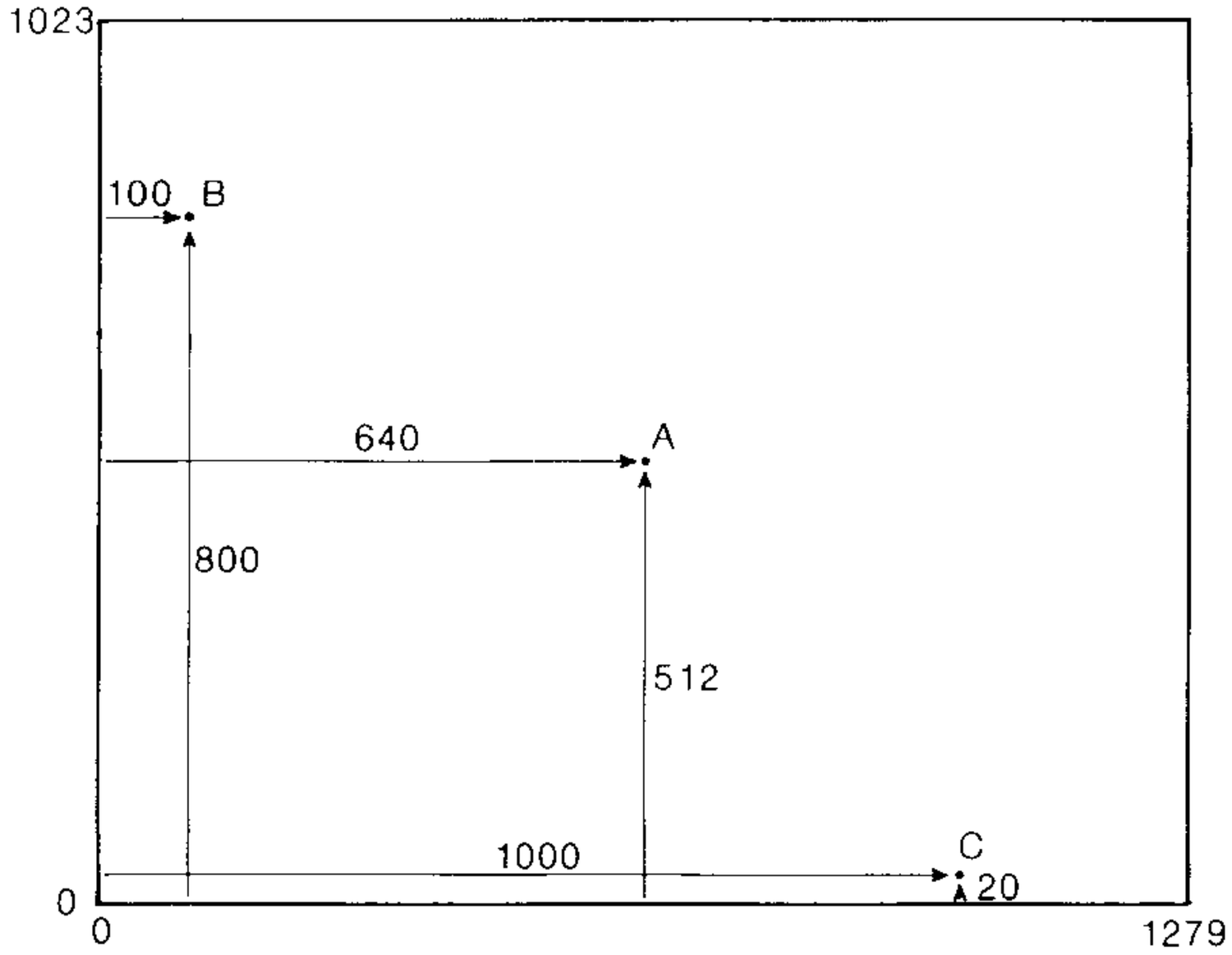
The modes differ in a number of important ways. Some of these differences will be explained in this chapter, but a full list of the characteristics of the modes is also provided in Appendix A.

Two sets of modes are available, modes 0 to 7 and modes 128 to 135. Each low-numbered mode N has a high-numbered counterpart mode N+128 which behaves visibly in exactly the same way and has the same features. For example, mode 7 and mode 135 are identical in appearance.

You should use modes 0 to 7 if you are writing programs which may also be run on the original BBC Model B microcomputer. In the Master Series computers, modes 128 to 135 leave more memory free so that longer programs can be written. For this reason you should always use the high-numbered modes but, in the example programs that follow, all references to a particular mode apply equally well to its lower-numbered counterpart.

Modes 128, 129, 130, 132 and 133 are known as **graphics modes** because they allow the use of both text and graphics. Modes 131 and 134 are text-only modes. Mode 135 allows the use of graphics, but the commands involved are very different and so are dealt with in a separate section beginning on page 94.

In each of the graphics modes, points on the screen are given coordinates so that their position can be identified.



The point A in the figure has coordinates 640 across and 512 up, roughly the middle of the screen. The point B is at position 100,800 and C is at 1000,20.

Type in and run this program:

```
10 MODE 128
20 MOVE 100,100
30 DRAW 800,100
40 DRAW 800,900
50 DRAW 100,100
```

Line 10 changes to a graphics mode, and as a result the invisible graphics cursor is automatically positioned at point 0,0 – the bottom left corner of the graphics screen.

Line 20 causes the computer to move to 100,100 without drawing a line.

The DRAW command draws a line from the last point visited (which was 100,100) to 800,100. The remaining DRAW commands produce a series of joined lines making a triangle.

Earlier you saw that after running a program you can clear the screen by typing:

CLS RETURN

You can also clear the screen with:

CLG RETURN

Although both commands appear to have the same effect, CLS actually clears the text screen and CLG clears the graphics screen. Normally these are exactly the same and fill the whole screen. Later you will see that the areas in which

text and graphics appear can be separated, and so it is useful to have two commands for clearing the screen.

The lines drawn in mode 128 are the finest that your computer can produce, and so this mode is used whenever very accurate **high-resolution** graphics are needed. The same program runs in other graphics modes, as you can see if you edit line 10 and then run the program again, i.e. type:

```
10 MODE 129 RETURN
RUN RETURN
```

This time the lines produced are thicker – mode 129 is a medium-resolution mode. The main advantage it offers over mode 128 is that it allows the display of four colours at the same time. You can change the colour of the lines by adding:

```
35 GCOL 0,1
45 GCOL 0,2
```

and running the program again. GCOL is used to select the colour to be used in the DRAW statement. The number following GCOL 0, is related to a particular colour in each mode. In mode 129:

GCOL 0,0 gives black lines
GCOL 0,1 gives red lines
GCOL 0,2 gives yellow lines
GCOL 0,3 gives white lines

Once a colour has been selected, it is automatically used in all further DRAW statements until a new GCOL command is given.

GCOL can also be used to change the background graphics colour. For example, type:

```
MODE 129 RETURN
GCOL 0,130 RETURN
CLG RETURN
```

This sets the background to yellow, and then clears the whole graphics screen to that colour. All GCOL numbers greater than 127 change the background colour.

RUN the program again after editing line 10 to be:

```
10 MODE 130
```

Mode 130 is a low-resolution mode giving much thicker lines, but up to 16 colours can be displayed simultaneously, eight of these being flashing colours.

Note that GCOL 0,2 gives green and not yellow in this mode. The numeric

references to colour are not the same in all the graphics modes. You must refer to Appendix A for the correct GCOL number to produce a particular colour.

In mode 130, GCOL 0 can be followed by any number from 0 to 15 to select a colour. Try changing the GCOL statements to see its effects.

The PLOT command

The PLOT command is an all-purpose drawing command. MOVE and DRAW are special examples of PLOT. Because moving and drawing are used so frequently, the PLOT commands that produce these effects have been given equivalent keywords:

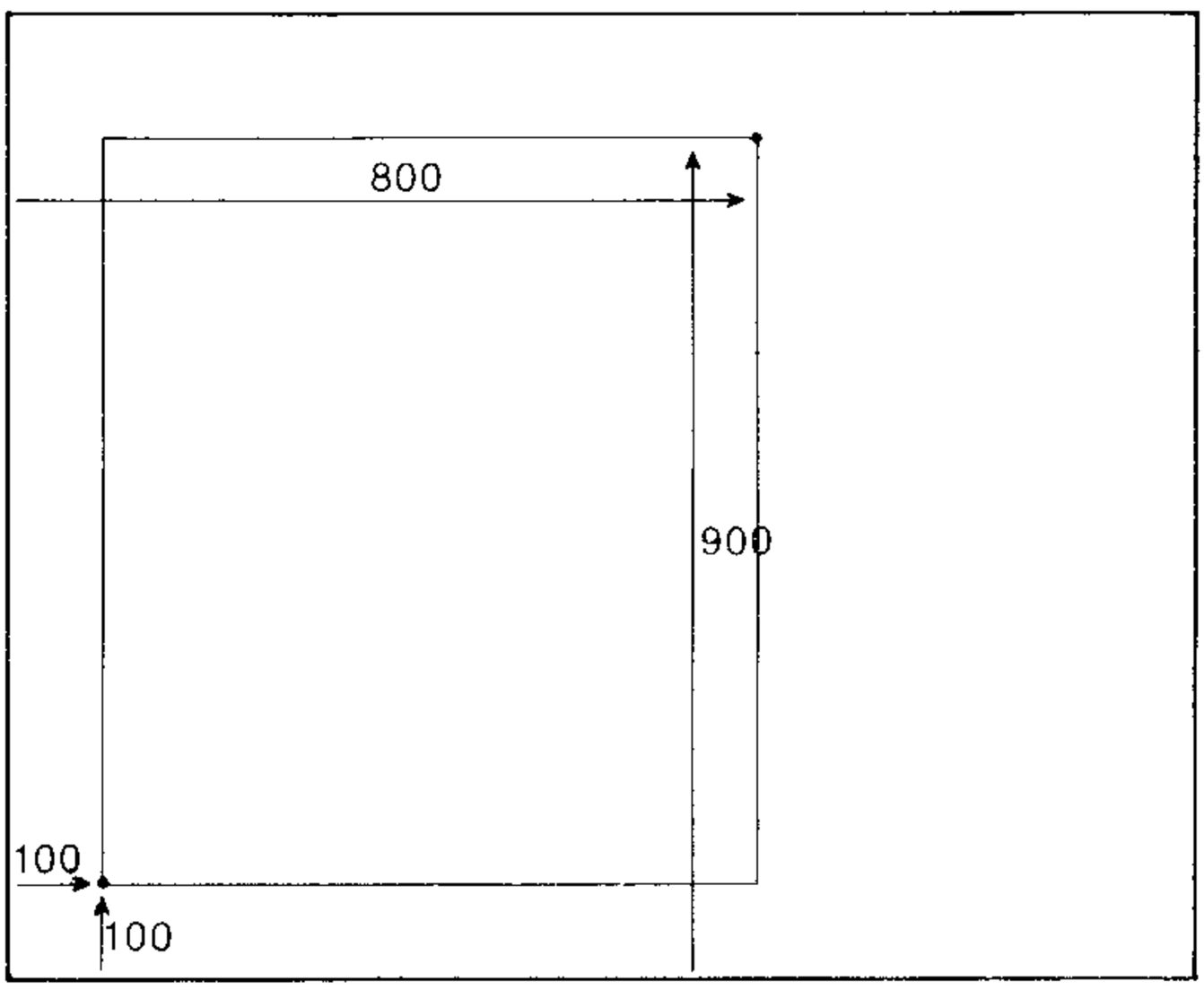
PLOT 4,100,100 is the same as MOVE 100,100

PLOT 5,800,100 is the same as DRAW 800,100

The first number after PLOT decides how the lines are plotted. PLOT commands enable rectangles, parallelograms, circles, segments, sectors, arcs, triangles or ellipses to be drawn in outline, solid colour or patterned. You have seen this demonstrated in the Welcome software. A program to draw a solid rectangle is:

```
10 MODE 129
15 REM move to one corner of rectangle
20 MOVE 100,100
25 REM move to diagonally opposite corner
30 PLOT 101,800,900
```

PLOT 101 tells the computer to draw a rectangle with opposite corners at the last point visited and the present point:

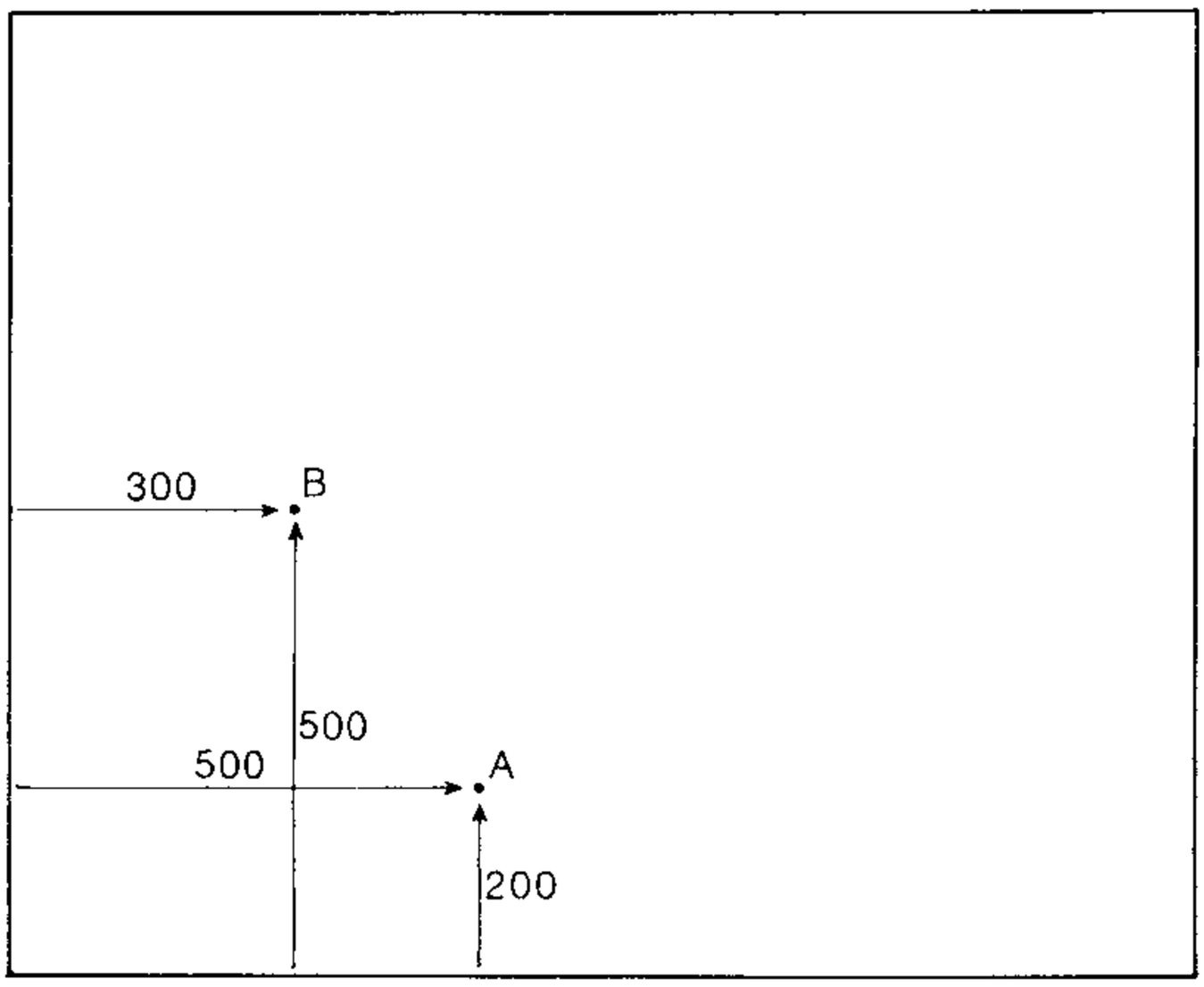


Change line 20 and RUN the program again:

```
20 MOVE 300,100
```

You may not be surprised to see that the rectangle gets smaller because the position of the first corner has changed. If you wanted to plot a whole series of identical rectangles in different positions, you would have to calculate the new position of the opposite corner for every rectangle. There is, however, another PLOT command which avoids this problem by describing the rectangle slightly differently.

Instead of giving the exact or absolute position of a point on the screen, its distance from another point can be given – this is the relative position of the point:



The point A in the figure is at 500,200. The point B is 200 to the left and 300 above A, so its relative position is -200,300. This program draws a rectangle using A and B as the positions of the two corners:

```
10 MODE 129
20 MOVE 500,200
30 PLOT 97,-200,300
```

Now change line 20 and RUN the program again:

```
20 MOVE 300,100
```

PLOT 97 tells the computer to draw a rectangle using the two points given, with the second point being relative to the first point. This means that the computer always draws the same size rectangle, wherever the first point is placed. Relative positioning is very useful if a drawing needs to be moved around on the screen.

The PLOT commands are very versatile and provide a great deal of control over how images are drawn. Lines or figures can be drawn absolutely or relatively, solid or dotted, in the foreground or background colour. Figures can be drawn in outline or as solid blocks of colour. A full list of the PLOT commands is given in Appendix H.

A circle can be drawn by giving the position of its centre and a point on its circumference:

```
10 MODE 1
15 REM coordinates of centre
20 MOVE 300,300
25 REM coordinates of point on circumference
30 PLOT 149,550,300
```

Here is an example of a PLOT command that draws a solid figure. Edit line 30 to:

```
30 PLOT 157,550,300
```

and RUN the program again. You can get a red circle by adding:

```
16 GCOL 0,1
```

Other PLOT commands enable you to create solid rectangles, ellipses, sectors of a circle, and so on. More complex figures must be built up using these simpler shapes. Any shape can be *flood filled* with colour once it has been drawn:

```
1  REM Teddy - an unfinished masterpiece
10 MODE 130
19 REM select red
20 GCOL 0,1
29 REM draw circular head
30 MOVE 500,500
40 PLOT 149,800,500
49 REM right eye
50 MOVE 620,600
60 PLOT 149,680,600
69 REM left eye
70 MOVE 380,600
80 PLOT 149,440,600
89 REM rectangular nose
90 MOVE 460,600
100 PLOT 101,540,400
109 REM use arc for the smile
110 MOVE 500,600
120 MOVE 350,350
130 PLOT 165,650,350
```

```
139 REM change to yellow for flood-fill
140 GCOL 0,3
150 PLOT 133,500,320
```

You might like to finish the picture by adding ears and colouring the eyes.

The Teddy program runs in mode 130, which allows 16 different colours. In other modes, such as mode 129, only four colours can be displayed at the same time. The range of colours is increased by four extra patterns made up of various colour combinations. For example, in mode 129:

GCOL 16,0 red-orange
GCOL 32,0 orange
GCOL 48,0 yellow-orange
GCOL 64,0 cream

These colours are produced regardless of the second number used. The effect of the commands varies from mode to mode, as the patterns are built up from the colours available in that mode. Change the GCOL commands in the Teddy program to see some of the patterns available in mode 130.

You can create your own colour patterns in place of those provided – this is described on page 103.

Printing text

Text can be displayed in any of the eight modes, but the number of characters per line varies from mode to mode, and can be 20, 40 or 80 characters. Try:

```
10 MODE 128
20 PRINT "Here is a sentence"
30 PRINT "to demonstrate printing."
```

Edit the program and run it a few times with line 10 altered to produce mode 129, 130 or 135. Mode 135 gives the clearest display. If you are using a TV rather than a monitor you may find mode 128 text rather hard to read.

After obeying any PRINT statement the computer moves to the start of a new line unless instructed to do otherwise. Run the program again after changing line 20 to:

```
20 PRINT "Here is a sentence";
```

The semi-colon at the end of the line tells the computer to stay on the same line after printing the string. The result is:

```
Here is a sentenceto demonstrate printing
```

The semi-colon is useful if you are printing a variable within a sentence, and want all the text to be on the same line. Add these lines:

```
40 my_age=105
50 PRINT "I am ";my_age;" years old."
```

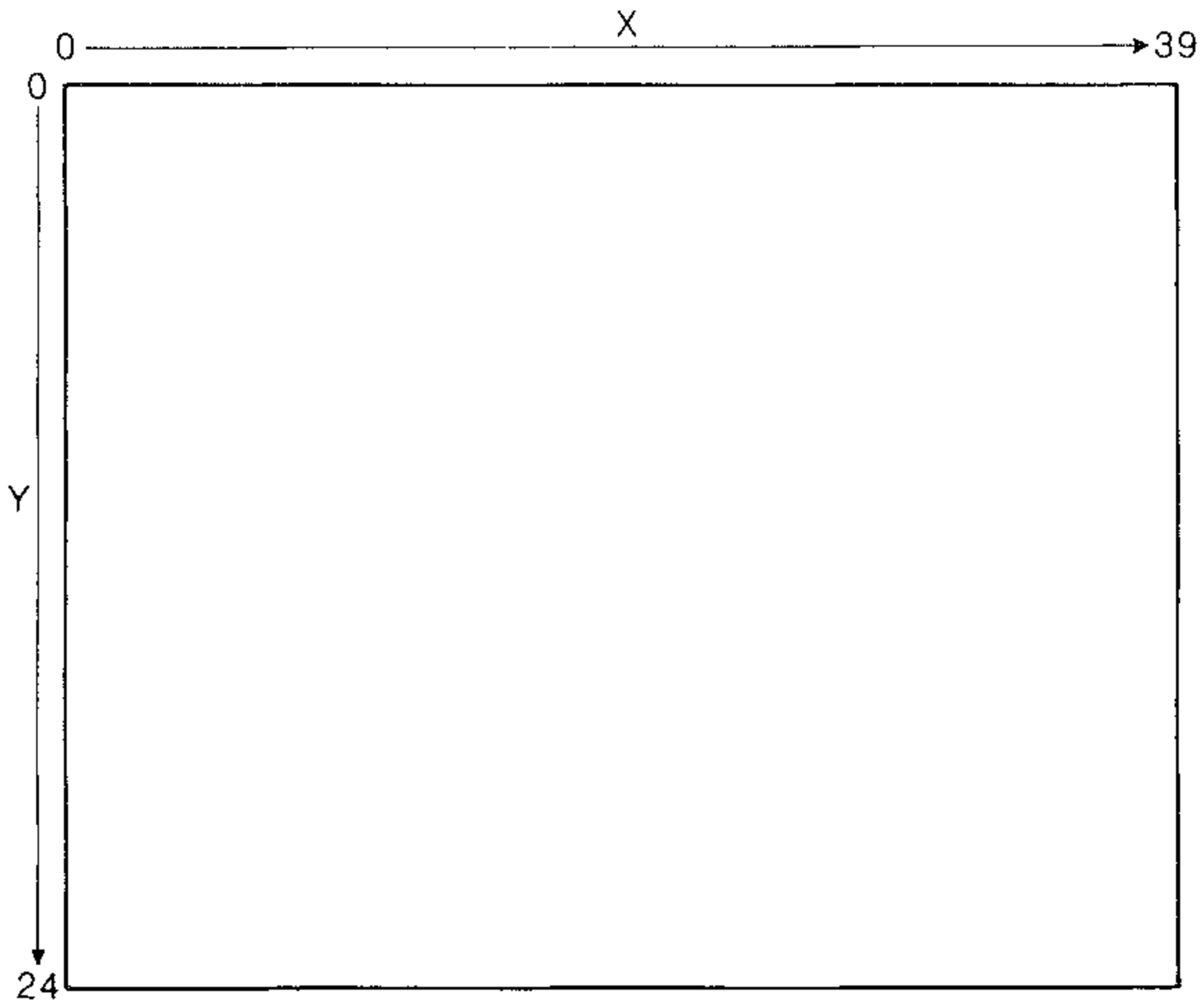
The spaces within line 50 are very important, as they stop the text running together untidily, as in the first example.

Including apostrophes causes extra blank lines to be printed. For example:

```
50 PRINT '''I am ";my_age;" years old."
```

prints two blank lines before the actual line of output.

The position of any character on the screen can be described in terms of its **text coordinates**. Text coordinates are given relative to the top left of the screen, unlike graphics. In mode 135, the text coordinates are:



Notice that although there are 40 character positions on a line, the positions are numbered zero to 39, and the lines are numbered similarly.

The PRINT TAB statement enables you to control the position at which printing begins. Use NEW to remove the current program then type:

```
10 MODE 135
20 PRINT "0123456789"
30 PRINT TAB(5);"An example of TAB."
```

When run, this gives:

```
0123456789
     An example of TAB
```

Printing begins at character position 5 on the line, ie the 6th column. More than one TAB can be used on the same line, but if the computer has already

moved beyond the required TAB position it begins a new line. For example:

```
30 PRINT TAB(5);"An";TAB(10);"example";TAB(15);"of";TAB(20);"TAB."
```

gives:

```
0123456789
     An   example
                of   TAB.
```

The computer is already at character position 17 when it comes to the TAB(15) command, and so it starts a new line.

By also giving the line number, you can use PRINT TAB to place text anywhere on the screen, for example:

```
10 MODE 135
20 PRINT TAB(8,24) "It can go at the bottom"
30 PRINT TAB(14,0) "Or the top"
40 PRINT TAB(1,11) "Or the left";TAB(27);"Or the right"
```

Line 30 should remind you that although mode 135 has 25 lines these are numbered from zero to 24. Line 40 shows that once you are on a line you can use TAB as before without referring to the line number.

Printing text in colour

The computer lets you change the colours used in printing text with the COLOUR command. Type:

```
MODE 129 RETURN
COLOUR 1 RETURN
```

The number after COLOUR indicates red in mode 129, and tells the computer that the new text foreground colour is to be red. Anything you type from now on will be printed in red. Type:

```
COLOUR 2 RETURN
COLOUR 129 RETURN
```

The first COLOUR command changes the text colour to yellow, and the second changes the background colour to red. All text from now on will be printed as yellow on red. You can change the entire screen to the new background colour by typing:

```
CLS RETURN
```

The COLOUR commands apply in all modes except modes 7 and 135. As with GCOL, the numbers used to indicate a particular colour vary from mode to mode. Consult Appendix A for a full list of the numeric colour references for each mode.

More advanced print formatting

When producing a table of figures, it is useful to print at particular positions without having to use TAB every time. If the printed items are separated by commas the computer does this automatically:

```
10 MODE 135
15 REM lines 20 & 30 help
16 REM to show character positions
20 PRINT TAB(10) "11111111112222222223"
30 PRINT "01234567890123456789012345678 90"
40 PRINT 1.23,4.567,89
```

Running the program gives:

```
          11111111112222222223
0123456789012345678901234567890
      1.23     4.567        89
```

Each number is printed at the right hand side of a column 10 characters wide. These columns are called **fields** and the width of each field is set to 10 characters when the computer is switched on.

Text is printed to the left of the field, as you can see by adding:

```
50 PRINT "Hello","my","friend"
```

which gives:

```
          11111111112222222223
0123456789012345678901234567890
      1.23     4.567        89
Hello     my         friend
```

If a number or word is longer than the field width, the item following is printed in the next empty field. For example:

```
          11111111112222222223
0123456789012345678901234567890
      1.23     4.567        89
Congratulations      my         friend
```

The field width can be altered to vary from zero to 255 characters:

```
10 MODE 135
18 REM set field width to 8 characters
19 REM giving 5 fields across the screen
20 @%=&08
30 PRINT TAB(8) "Income from sales regions"
40 PRINT '"Jan",1234.56,789,123.45,678.9
50 PRINT "Feb",234.5,67.89,12,3456.78
```

@% in line 20 is a special variable which controls the printing of numbers. In this case it is used only to reduce the field width to eight characters. The result is very untidy and confusing because the numbers are not aligned vertically about the decimal point. Make line 20:

```
20 @%=&020208
```

This tells the computer to print each number to two decimal places and with a field width of eight.

@% offers a great deal of control over the way the computer prints numbers, and is discussed in more detail in the Reference Manuals. Note that once @% is set, its effects remain until you reset @%, perform a hard break, or switch the computer off.

Text and graphics

It is sometimes useful to restrict the printing of text to part of the screen. You saw an example of this in the Welcome software. The Turtle Graphics (TURTLE) program lets you type commands at the keyboard, but these are printed on the bottom four lines only, so that the screen display is not disturbed.

Type:

```
MODE 135 RETURN
```

You can set up a text 'window' within which text is displayed by typing:

```
VDU 28,12,15,30,10 RETURN
```

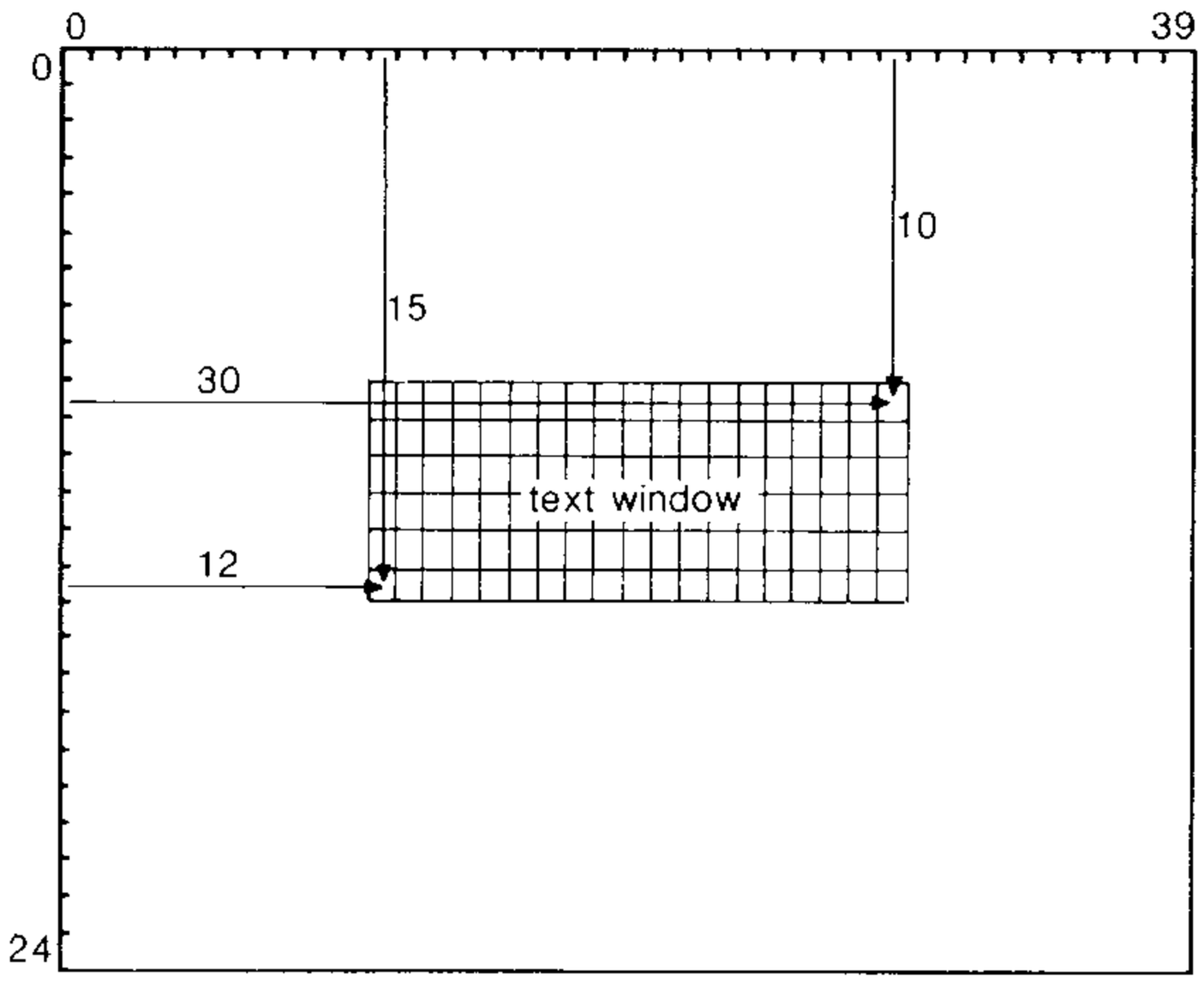
Type a few characters at the keyboard (anything will do). The text is printed inside a window in the middle of the screen.

The VDU 28 command is one of a series of VDU commands which enable you to control the way text and graphics are displayed on the screen. VDU commands can be used to change the colours of text or graphics, move the cursor, clear the screen, etc. VDU 28 is used specifically to set up a text window.

The first two numbers following the VDU 28 give the position of the bottom left character within the new text window. The remaining two numbers give the position of the character at the top right of the text window (see illustration at the top of the next page).

Once you have created a text window, the top left position within the window becomes 0,0. All PRINT TAB commands are relative to this new position, as you can see if you type:

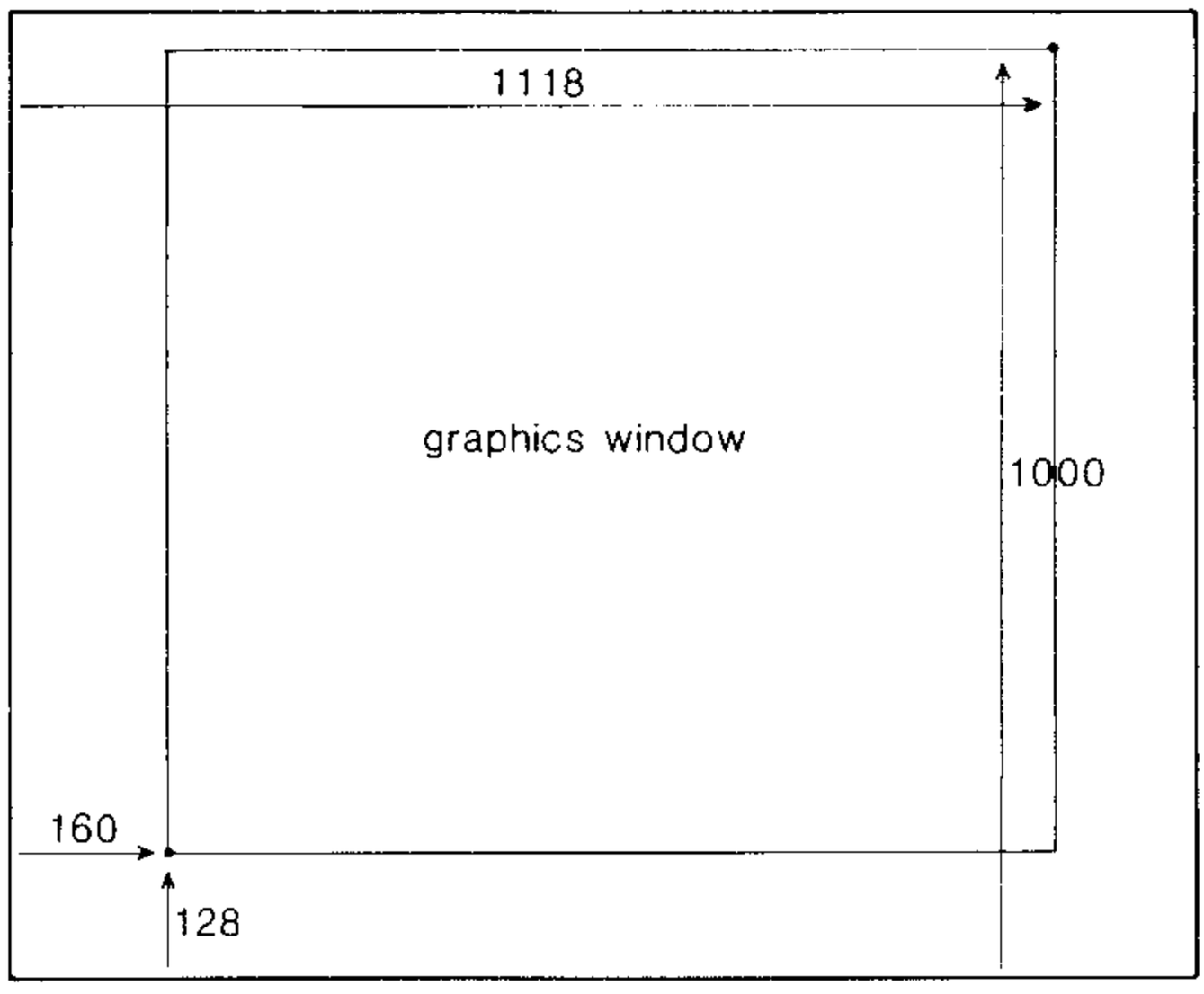
```
CLS RETURN
PRINT TAB(4,3)"The middle" RETURN
```



A graphics window can be set up in any mode which allows the use of graphics. Type:

```
MODE 129 RETURN
VDU 24,160;128;1118;1000; RETURN
```

VDU 24 is followed by the graphics positions which are at the bottom left and top right of the new window:



In this case graphics coordinates are used – notice the numbers are separated by semi-colons unlike the VDU 28 command.

You can see the graphics window by typing:

```
GCOL 0,130[RETURN]
CLG[RETURN]
```

Although graphics are now displayed only within the window, the whole screen is still used for text. To completely separate text and graphics we must set up both a graphics and a text window. Create a text window below the graphics window by typing:

```
VDU 28,5,31,34,28[RETURN]
```

Change the background text colour to red:

```
COLOUR 129[RETURN]
CLS[RETURN]
```

Type in some MOVE and DRAW commands. The text is displayed within the text window, and any lines drawn only appear within the graphics window.

After a VDU 26 command the whole screen is used for text and graphics once again, so return things to normal by typing:

```
VDU 26[RETURN]
```

Note that using a MODE command has the same effect as it automatically destroys all windows.

Printing text at graphics positions

TAB is used to print characters on the screen at any text coordinate. It is sometimes helpful to position text more accurately on the screen than PRINT TAB allows, especially if graphics are also used.

In modes in which graphics can be used, text can be printed at graphics coordinates after a VDU 5 command. This program gives a three dimensional effect by printing the same message twice and slightly off-setting the second set of characters which are printed in a different colour:

```
10 MODE 1
20 PRINTTAB(16,15)"H e l l o"
30 VDU 5
40 GCOL 0,1
50 MOVE 516,540
60 PRINT"H e l l o"
70 VDU 4
```

Line 20 prints characters using the usual text coordinates. The VDU 5 in line 30 joins the text and graphics cursors. In line 50 MOVE is used to position the text, which can now only be printed at graphics coordinates. Finally, the VDU 4 command returns the text cursor to normal so that PRINT TAB is usable again.

Input

Earlier you saw that you can type in information while a program is running if the program contains an INPUT statement:

```
10 MODE 135
20 PRINT "How old are you";
30 INPUT age
40 PRINT "So you're ";age;" years old."
50 PRINT "You don't look it!"
```

The INPUT statement in line 30 causes the computer to print a question mark, and then wait for information to be typed at the keyboard. The computer expects a number to be typed, because *age* is a numeric variable. Once RETURN is pressed, the computer stores the value typed in the variable *age*. If you type text rather than a number, the computer assumes the number is zero.

If you want to input text, you must use a string variable in the INPUT statement:

```
10 MODE 135
20 PRINT "What is your name";
30 INPUT name$
40 PRINT "Hello ";name$;" and how are you?"
```

You can use a single INPUT statement to ask for several inputs:

```
10 MODE 135
20 PRINT "What is your name and age ";
30 INPUT name$, age
40 PRINT "Hello ";name$;". So you are ";age;" years old."
```

In this case the computer will expect two inputs, one a string and one a numeric variable. They can either be typed in separated by commas, or both can be followed by RETURN.

The PRINT statement just before the INPUT is there to give a message to remind you what you should type. This message can be included within the INPUT statement:

```
10 MODE 135
20 INPUT "What is your name ";name$
30 PRINT "Hello ";name$;". Pleased to meet you."
```

INPUT ignores any spaces at the beginning of an input or anything typed after a comma:

```
What is your name ? Nero,Emperor of Rome
Hello Nero. Pleased to meet you.
```

If you need to type in text that includes spaces at the start or may include commas, you should use INPUT LINE rather than INPUT:

```
20 INPUT LINE "What is your name ";name$
```

This gives:

```
What is your name ? Nero,Emperor of Rome
Hello Nero,Emperor of Rome. Pleased to meet you.
```

GET and INKEY

In some programs, such as games, the computer needs to respond as soon as a key is pressed. Programs like this use GET or INKEY rather than INPUT statements. GET waits until a key is pressed before continuing:

```
10 MODE 135
20 PRINT "Press any key to continue"
30 chosen=GET
40 PRINT "The program has ended."
```

INKEY causes the computer to wait for a key to be pressed within a fixed time:

```
10 MODE 135
20 PRINT "Press any key to continue"
30 PRINT "You have 3 seconds only!"
40 chosen=INKEY(300)
50 PRINT "The program has ended."
```

The timing is in hundredths of a second, so line 40 makes the computer wait for a key depression for three seconds (300 hundredths of a second). If no key is pressed within three seconds, the computer moves on to the next line of the program. If a key is pressed, the computer immediately continues with the next line of the program.

ASCII codes

Both GET and INKEY produce what is called the ASCII code of the depressed key. Internally, the computer uses a number from 0 to 255 to represent each character that it stores in its memory. This number is the character's ASCII code. For example, the ASCII code for A is 65, B is 66 and C is 67, so the computer would store the word CAB as the numbers 67, 65 and 66.

The computer can give you the ASCII code for a character. For example:

```
PRINT ASC("A") RETURN
```

prints 65. Note that ASC works with single characters only. If you want the ASCII codes for a series of characters you should consult the table showing the full character set in Appendix B.

In the previous two programs the ASCII code is stored in the variable *chosen*. If no key is pressed before the INKEY time limit is reached, *chosen* is given the value -1.

GET or INKEY do not automatically display the character typed at the keyboard. This is useful in programs where printing would spoil the screen display. If you do want to print the character, use PRINT CHR$ to convert the ASCII code into a string:

```
10 MODE 135
20 PRINT "Type any character - ";
30 chosen=GET
40 PRINT CHR$(chosen)
50 PRINT "You typed ";CHR$(chosen)
```

VDU followed by an ASCII code has the same effect as PRINT CHR$. For example, both PRINT CHR$(65) and VDU 65 would print the letter A. If you type:

```
VDU 66,66,67 RETURN
```

the computer prints:

```
BBC
```

The ASCII codes for the characters start at 32. Lower codes are used to give commands to the computer, as you have seen with VDU 26 and VDU 28.

Structured programs

In the last section you were introduced to some of the most commonly used commands from BBC BASIC. Most of these commands dealt with the ways in which you can communicate with the computer while a program is running, and how you can effect the way the computer displays information on the screen.

You have already seen that programs are much more readable if they contain sensible variable names. Additionally, all the example programs have used only a single statement per program line. Programs can be written to contain more than one statement per line, provided the statements are separated by colons:

```
10 MODE 135:PRINT "Type any character - ";:chosen=GET:PRINT CHR$(ch
osen):PRINT "You typed ";CHR$(chosen)
```

You can imagine that a program with many multi-statement lines like this is not easy to follow!

The next section deals with the facilities BBC BASIC offers to simplify the development and modification of programs. So far you have only used programs

made up of a sequence of instructions. The computer can also repeat instructions, or choose which of several instructions it will obey. All programs are built up from a combination of the three program structures *sequence*, *repetition* and *choice*. The next few sections describe how you can use these structures in BBC BASIC.

Planning your programming

The programs in the earlier chapters have all been fairly short, but the easiest way to write more complex programs is to organise them differently.

Look back at the Teddy program on page 52. The program consists of a sequence of instructions which the computer obeys in line-number order. A longer program might contain several hundred lines, and it simpler to write if it is broken into small sections or **procedures**.

This program shows the use of procedures:

```
 1 REM draw butterfly
10 MODE 130
20 PROCbody
30 PROCleft_wing
40 PROCright_wing
50 END
60 DEFPROCbody
 70 GCOL0,2
 80 MOVE 640,500
 90 MOVE 700,500
100 PLOT 205,640,700
110 ENDPROC
120 DEFPROCleft_wing
130 GCOL 0,1
140 MOVE 200,200
150 DRAW 600,500
160 PLOT 85,200,800
170 ENDPROC
180 DEFPROCright_wing
190 GCOL 0,1
200 MOVE 1080,200
210 DRAW 680,500
220 PLOT 85,1080,800
230 ENDPROC
```

The main program is really only lines 10 to 50:

```
10 MODE 2
20 PROCbody
30 PROCleft_wing
40 PROCright_wing
50 END
```

Lines 20 to 40 are known as **procedure calls**. Each PROC tells the computer not to obey the next line number. Instead it must search the program for a DEFinition of the PROCedure (DEFPROC) with the correct procedure name, and obey the instructions in that procedure.

For example, after line 20 the computer moves to line 60 and then executes lines 70 to 100 which draw the butterfly's body. Line 110 is the END of the PROCedure (ENDPROC).

After carrying out the procedure, the computer returns to the line after the procedure call to carry on with the program. Here the line following line 20 is another procedure call. *PROCleft_wing* draws the butterfly's left wing, and the computer then executes *PROCright_wing*, which draws the right wing.

The END in line 50 tells the computer that the program is finished. END is optional in some programs, such as the Teddy program. It must be used here as otherwise the computer will carry on and execute line 60 (and attempt to draw the butterfly body again!

The order in which procedures appear does not matter and you can place procedures wherever you want within a program, except at the very beginning. Procedure names follow much the same rules as for variable names, although a procedure name can begin with a number.

A procedure can be called more than once in a program, saving you the trouble of repeating program lines:

```
10 MODE 130
20 PROCvariables
30 PROCengine
40 END
50 DEFPROCvariables
60 scale=0.6
70 xstart=300
80 ystart=200
90 xdoor=360
100 xdist=900
110 ydoor=500
120 ydist=300
130 xbump=300*scale
```

```
140 bump_rad=50*scale
150 xchim=100*scale
160 ychim=200*scale
170 chimstart=60*scale
180 xdoorstart=xstart+(xdist*scale)
190 xwind=xdoorstart-30*scale
200 ywind=ystart+(ydist-30)*scale
210 xwinddist=300*scale
220 ywinddist=200*scale
230 wheel_dist=130*scale
240 wheel_rad=100*scale
250 ENDPROC
260 DEFPROCengine
270 PROCrectangle(xstart,ystart,xdist*scale,ydist*scale,1)
280 PROCrectangle(xdoorstart,ystart,-xdoor*scale,ydoor*scale,1)
290 PROCrectangle(xwind,ywind,-xwinddist,ywinddist,6)
300 PROCrectangle(xstart+chimstart,ystart+(ydist*scale),xchim,ychim,1)
310 PROCcirc(xstart+wheel_dist,ystart,wheel_rad,4)
320 PROCcirc(xdoorstart-wheel_dist,ystart,wheel_rad,4)
330 PROCcirc(xstart+xbump,ystart+(ydist*scale),bump_rad,1)
340 ENDPROC
350 DEFPROCrectangle(x,y,xmove,ymove,col)
360 GCOL 0,col
370 MOVE x,y
380 PLOT 97,xmove,ymove
390 ENDPROC
400 DEFPROCcirc(x,y,rad,col)
410 GCOL 0,col
420 MOVE x,y
430 PLOT 153,rad,0
440 ENDPROC
```

The procedures to draw the engine use relative MOVE and DRAW commands. *PROCvariables* sets the values of the variables used throughout the rest of the program. You can change the size and position of the engine by changing the values of *scale%*, *xstart%* and *ystart%*.

Information can be passed to a procedure from the main program:

```
10 MODE 130
20 PROCcircle(400,300,200)
30 PROCcircle(600,600,100)
40 PROCcircle(690,750,50)
50 END
60 DEFPROCcircle(xcentre%,ycentre%,radius%)
```

```
70 MOVE xcentre%,ycentre%
80 PLOT 157,xcentre%+radius%,ycentre%
90 ENDPROC
```

The values in brackets at line 20 are called **parameters**. The computer takes the parameters and stores them in the variables *xcentre%*, *ycentre%*, and *radius%* in line 60 when it obeys the procedure call. It uses these variables in the rest of the procedure to draw a circle with its centre at 400,300 and a radius of 200.

Lines 30 and 40 demonstrate how this same procedure can be used whenever a circle is drawn. Only the parameters need to be changed.

A procedure like *PROCcircle* is very useful because:

- it can be used many times in the same program with different parameters to give different results;
- it can be used even if you do not know or remember how the procedure works;
- it can be used in other programs.

You might use *xcentre%* and *ycentre%* to hold the coordinates of the screen centre in a program. It seems as if these values will be lost if *PROCcircle* is used in the same program, because this also has variables called *xcentre%* and *ycentre%*:

```
10 MODE 130
15 xcentre%=640:ycentre%=512
20 PROCcircle(400,300,200)
30 PROCcircle(600,600,100)
40 PROCcircle(690,750,50)
45 PRINT"xcentre% remains ";xcentre%
46 PRINT"ycentre% remains ";ycentre%
50 END
60 DEFPROCcircle(xcentre%,ycentre%,radius%)
70 MOVE xcentre%,ycentre%
80 PLOT 157,xcentre%+radius%,ycentre%
90 ENDPROC
```

RUN the program. The values of *xcentre%* and *ycentre%* are not affected by *PROCcircle*. This is because any parameters passed to a procedure are automatically **local** to that procedure. The *xcentre%*, *ycentre%* and *radius%* in *PROCcircle* exist only within the procedure, and do not change the value of variables with the same name elsewhere in the program.

All variables except parameters are **global** to a program. The whole program, including procedures, 'knows' the value of the variables:

```
10 MODE 135
20 PROCname
30 PROCprint
40 END
50 DEFPROCname
60 INPUT"What is your name ",name$
70 ENDPROC
80 DEFPROCprint
90 PRINT"This procedure is called PROCprint"
100 PRINT"It 'knows' your name is ";name$
110 ENDPROC
```

The string variable *name$* is global. It is set up in *PROCname*, but *PROCprint* also 'knows' *name$* and uses it.

The distinction between local and global variables only becomes important if a procedure contains global variables. For example, here is a procedure which centres text on a given line:

```
100 DEFPROCcentre(text$)
110 length%=LEN(text$)
120 x_position%=(40-length%)/2
130 PRINT TAB(x_position%) text$
140 ENDPROC
```

A useful procedure which might be called several times in a single program. However, the procedure contains two global variables, *length%* and *x_position%*. If variables of the same name are used in the program, their values are lost after *PROCcentre* is called:

```
10 MODE 135
20 length%=5
30 x_position%=15
40 PRINT"length% is ";length%
50 PRINT"x_position% is ";x_position%
60 PROCcentre("A few characters")
70 PRINT"length% is now ";length%
80 PRINT"x_position% is now ";x_position%
90 END
100 DEFPROCcentre(text$)
110 length%=LEN(text$)
120 x_position%=(40-length%)/2
130 PRINT TAB(x_position%) text$
140 ENDPROC
```

You can make sure that variables within a procedure do not interfere with the rest of the program by declaring the variables as *local*. Add this line to the previous program and run it again:

```
105 LOCAL length%,x_position%
```

This time *length%* and *x_position%* are unchanged despite *PROCcentre*. There are effectively two copies of the variables: the global values, available to the whole program, and the local values, which exist only within *PROCcentre*.

PROCcentre is now completely isolated, and it can be used in any program without giving unexpected side-effects.

Note that variables can also be used as parameters. This brief program contains an improvement on *PROCcircle* so that you can select the colour used:

```
10 MODE 135
20 PROCchoose
30 MODE 130
40 PROCcircle(xchoice%,ychoice%,radius_choice%,colour_choice%)
50 END
60 DEFPROCchoose
70 INPUT"Centre of circle ",xchoice%,ychoice%
80 INPUT"Radius ",radius_choice%
90 INPUT"Colour number (1 to 15) ",colour_choice%
100 ENDPROC
110 DEFPROCcircle(xcentre%,ycentre%,radius%,colour%)
120 GCOL 0,colour%
130 MOVE xcentre%,ycentre%
140 PLOT 157,xcentre%+radius%,ycentre%
150 ENDPROC
```

Throughout the rest of this chapter on BBC BASIC, procedures are used extensively. This is because it is simpler to write and modify programs that are broken into smaller sections. Some of the procedures will be specific to a particular program, but others, such as *PROCcircle*, are more general-purpose. You may like to use these procedures in programs of your own.

Functions

A function is a routine which takes one or more parameters and uses them to calculate a result. BBC BASIC contains some built-in functions. Try:

```
PRINT LEN("Acorn Computers")
```
RETURN

LEN is a function which takes a string as a parameter and produces the length of the string as the result. Now try:

```
PRINT SQR(9)
```
RETURN

SQR is a function taking a number as a parameter and producing its square root as the result.

BBC BASIC allows you to set up your own functions, as this example shows:

```
10 MODE 135
20 PROCinput_time
30 END
40 DEFPROCinput_time
50 PRINT'"Input a time in minutes and seconds."
60 PRINT'"The function will convert it into"
70 PRINT"seconds."
80 INPUT'"How many minutes and seconds ",minutes%,seconds%
90 total%=FNconvert(minutes%,seconds%)
100 PRINT'"That is ";total%;" seconds."
110 ENDPROC
120 DEFFNconvert(mins%,secs%)
130 =mins%*60+secs%
```

Line 90 calls the function. The computer scans the rest of the program until it finds the DEFinition of the FuNction (DEFFN) at 120.

Line 130 begins with an equals sign. This tells the computer that the calculation which follows will produce the required result, and that the function ends on this line. The calculation is carried out, the function ends, and the program returns to line 90 and stores the result in *total%*.

The function here is a trivial example, as it is simpler to just put:

```
90 total%=minutes%*60+seconds%
```

The program below uses a much more complex function, containing statements which are explained in the next few sections:

```
10 MODE 135
20 PROCinput_word
30 END
40 DEFPROCinput_word
50 INPUT"Type in a word ",word$
60 PRINT'"An anagram of that word is ";FNanagram(word$)
70 ENDPROC
80 DEFFNanagram(choice$)
90 length%=LEN(choice$)
100 FOR count=1 TO length%
110 random_letter%=RND(length%-1)
120 choice$=RIGHT$(choice$,length%-random_letter%)+MID$(choice$,rand
om_letter%,1)+LEFT$(choice$,random_letter%-1)
```

```
130 NEXT
140 =choice$
```

Loops

FOR...NEXT

The real power of computers comes from their ability to repeat instructions. This can transform trivial programs so that they produce very impressive results.

The FOR..NEXT loop makes the computer repeat a set of instructions a fixed number of times:

```
10 MODE 128
20 FOR count=1 TO 100
30 PRINT count
40 NEXT count
```

Line 20 is the start of the loop, with the variable *count* being set to 1 initially. After printing the value of *count* at line 30, the computer finds the NEXT statement which indicates the end of the loop.

At this point count is increased by 1. Provided that count has not gone beyond the end value of 100 the computer now repeats all the instructions again.

Line 40 can be written as just:

```
40 NEXT
```

The use of the variable name is optional, but if you are using many loops in a program, including the name makes the program easier to follow.

You can change the step size so that *count* does not increase by 1:

```
20 FOR count=7 TO 50 STEP 2
```

The step size can be decimal:

```
20 FOR count=3 TO 10 STEP 1.6
```

It can even be negative, although the start and end values for the loop must also be adjusted so that the loop starts with the highest value:

```
20 FOR count=20 TO 1 STEP -1
```

Of course, the loop values can also be variables. You can experiment with loops by adding these lines and running the program a few times:

```
15 INPUT "What is the start, end and step size ",start,end,step
20 FOR count=start TO end STEP step
```

Here is a brief program which shows the power of the loop:

```
10 MODE 2
20 PROCmodern_art
30 END
40 DEFPROCmodern_art
50 FOR count=1 TO 50
60 PROCcircle(RND(1279),RND(1023),RND(200),RND(7))
70 NEXT count
80 ENDPROC
90 DEFPROCcircle(xcentre%,ycentre%,radius%,colour%)
100 GCOL 0,colour%
110 MOVE xcentre%,ycentre%
120 PLOT 157,xcentre%+radius%,ycentre%
130 ENDPROC
```

RND produces a random whole number between 1 and the bracketed value. Line 60 draws a random-sized circle in a random position and random colour by calling *PROCcircle* with random parameters.

You may get an idea how some of the Welcome software works by running the program again using these lines:

```
50 FOR count=7 TO 1 STEP -1
60 PROCcircle(640,512,count*50,count)
```

One FOR...NEXT loop can be included within another. These are called **nested loops**:

```
10 MODE 7
20 PROCtables
30 END
40 DEFPROCtables
50 FOR table=1 TO 12
60 PRINT''TAB(8)"The ";table;" times table"''''
70 FOR count=1 TO 10
80 PRINT count;" times ";table;" is ";count*table
90 NEXT count
100 PROCinput
110 NEXT table
120 ENDPROC
130 DEFPROCinput
140 PRINT''''"Press any key when you're ready for"
150 PRINT'TAB(2)"the next multiplication table"
160 key=GET
170 CLS
180 ENDPROC
```

The main loop running from line 50 to 110 counts through the multiplication tables from 1 to 12. The other loop from 70 to 90 nests completely within the main loop. It multiplies the value of table by all the numbers from 1 to 10.

The effect of LIST may be altered so that it automatically produces indentations for every FOR and NEXT pair (and certain other structures). Type:

```
LISTO 7 RETURN
LIST RETURN
```

Notice that the start and end of the loops are in line vertically. This makes it easier to pick out the loops and spot errors.

Delete line 90, which contains a NEXT, and LIST the program again. The start and finish of the loop at lines 50 and 110 no longer line up. This is a sure sign that a loop somewhere in the program is missing a FOR or NEXT.

The option provided by LISTO remains in force until you reset it (using LISTO 0), execute a hard break or switch the computer off. However, leave it in force for the next section.

REPEAT...UNTIL

Imagine a program based on the BBC quiz *Mastermind*. The program needs to repeatedly ask questions until the time limit of one minute is reached. Can we use a FOR...NEXT loop here?

FOR...NEXT loops always end as the result of a count reaching a certain value. Here we have no idea beforehand exactly how many questions will be answered in one minute. One person running the program may answer only three questions, whereas another may answer a dozen.

In this case we must use a different sort of loop, the REPEAT...UNTIL loop. This is a loop that ends when a condition is satisfied, rather than as a result of a count. For example, many programs include a procedure that prevents the program from rushing on until a particular key is pressed:

```
10 MODE 7
20 PROCwait
30 END
40 DEFPROCwait
50 PRINT TAB(0,24)"Press C to continue"
60 REPEAT
70 key$=GET$
80 UNTIL key$="C"
90 ENDPROC
```

Lines 60 to 80 REPEATedly scan the keyboard UNTIL the C is pressed. Press

some other key at line 70. The computer finds that *key$* does not satisfy the condition at line 80, and so it executes the loop again from line 60.

The *Mastermind* program might look something like this:

```
10 MODE 7
20 PROCquiz
30 END
40 DEFPROCquiz
50 TIME=0
60 answers=0
70 REPEAT
80 first=RND(12)
90 second=RND(12)
100 PRINT'"What is ";first;" times ";second;
110 INPUT response
120 answer=answer+1
130 UNTIL TIME>=6000
140 PRINT'"You answered ";answer;" questions"
150 ENDPROC
```

Line 50 introduces TIME, which gives the value of the computer's internal clock. TIME counts in hundredths of a second from the moment the computer is switched on, or from when it is reset. Line 50 sets TIME back to zero, so that it can be used to count the minute allowed for questions.

The variable *answer* is used to count the number of answers given, and is initially set to zero by line 60. The loop runs from 70 to 130, and repeatedly asks random multiplication questions until TIME is >= (greater than or equal to) 6000 hundredths of a second, one minute.

The program has one big flaw – unlike Magnus Magnusson, it doesn't check the answers ! You will find out how to extend the program to do that in the next section , so you might like to save the program before you continue.

Making choices

You have already seen that the computer can obey a series of instructions, or repeat instructions a number of times. It can also choose whether to obey an instruction or not:

```
10 MODE 7
20 PROCinput_age
30 END
40 DEFPROCinput-age
50 INPUT "How old are you ",age
60 IF age<=18 THEN PRINT'"So you can't vote in elections."
70 ENDPROC
```

RUN the program a few times, inputting different ages. In line 60, the computer checks the statement after the IF, and if it is true, it executes the instructions after the THEN. If the statement is false, the computer ignores the rest of the IF...THEN and carries on to the next line.

Now add these lines to the program and run it several times, so that you are sure you understand how IF...THEN works:

```
63 IF age=32 THEN PRINT'"You are the same age as me!"
66 IF age<65 THEN PRINT'"You are below retiring age."
```

The quiz program can now be extended so that it checks your answers. The new lines are 65, 115 and 145:

```
10 MODE 7
20 PROCquiz
30 END
40 DEFPROCquiz
50 TIME=0
60 answers=0
65 wrong=0
70 REPEAT
80 first=RND(12)
90 second=RND(12)
100 PRINT'"What is ";first;" times ";second;
110 INPUT response
115 IF response<>first*second THEN wrong=wrong+1
120 answer=answer+1
130 UNTIL TIME>=6000
140 PRINT'"You answered ";answer;" questions"
145 PRINT'"You had ";wrong;" wrong"
150 ENDPROC
```

Line 115 can be extended so that it gives the correct answer as well as counting the wrong answers:

```
115 IF response<>first*second THEN wrong=wrong+1:PRINT "No, the ans
wer is ";first*second
```

Where there are only two possible outcomes, such as an answer being right or wrong, an extended form of IF...THEN can be used:

```
115 IF response<>first*second THEN wrong=wrong+1:PRINT "No, the ans
wer is ";first*second ELSE PRINT "Well done!"
```

In other words, IF the response is wrong,THEN the computer gives the right answer, ELSE it congratulates you on getting it correct.

The line is beginning to get rather long. To make the program easier to read and understand it is better to use:

```
115 IF response<>first*second THEN PROCwrong ELSE PROCright
```

and add extra procedures at the end:

```
160 DEFPROCwrong
170 wrong=wrong+1
180 PRINT "No, the answer is ";first*second
190 ENDPROC
200 DEFPROCright
210 PRINT "Well done!"
220 IF wrong<2 THEN PRINT "Keep it up"
230 ENDPROC
```

Conditions

A REPEAT...UNTIL loop can be set so that it ends under a variety of conditions:

```
10 MODE 135
20 PROCreaction
30 PROCtest
40 PROCcomment
50 END
60 DEFPROCreaction
70 PRINT TAB(0,8)"Press the correct key when it is"
80 PRINT"flashed on the screen."
90 PRINT TAB(0,13)"You have 2 seconds to respond, and you"
100 PRINT"can continue until you miss twice or"
110 PRINT"20 seconds is up."
120 PRINT TAB(0,24)"Press any key when you're ready.";
130 key=GET
140 ENDPROC
150 DEFPROCtest
160 CLS
170 missed=0
180 right=0
190 TIME=0
200 REPEAT
210 letter=RND(26)+64
220 PRINTTAB(19,11)CHR$(letter)
229 REM VDU7 gives a bleep
230 VDU7
240 response=INKEY(200)
250 IF response=-1 THEN missed=missed+1
```

```
260 IF response=letter THEN right=right+1
270 UNTIL missed=2 OR TIME>2000
280 ENDPROC
290 DEFPROCcomment
300 CLS
310 PRINT"You got ";right;" right"
320 PRINT"You missed ";missed
330 IF right>10 THEN PRINT'"A very good result."
340 IF right<4 THEN PRINT'"Rather poor."
350 ENDPROC
```

The loop runs from 200 to 270, and ends either when two keys are missed or 20 seconds is up. The OR can also be used in IF...THEN statements:

```
325 IF right>10 OR missed=0 THEN PRINT'"A very good result."
```

You may want an IF...THEN statement to be executed only if several conditions are true at the same time:

```
345 IF right<3 AND missed=2 THEN PRINT'"Quite pathetic."
```

AND can also be used to end a REPEAT...UNTIL loop:

```
270 UNTIL missed=2 AND right=5
```

Now the loop only ends after you have both missed two letters and have five correct – not a very sensible test!

There is (almost) no limit to the number of conditions, for example you might have:

```
270 UNTIL missed=2 OR TIME>2000 OR right>5
```

Multiple choices

IF...THEN...ELSE is useful if there are only two alternative choices of action, but often in a program there may be many more. For example, programs often contain a **menu** which allows the user to choose one of a number of actions. Here is the start of a drawing program which contains a menu:

```
10 MODE 135
20 PROCmenu
30 END
40 DEFPROCmenu
50 REPEAT
60 CLS
70 PRINT TAB(7,5)"Do you want to:"
80 PRINT TAB(8,9)"1 Load a picture"
90 PRINT TAB(8,12)"2 Save a picture"
100 PRINT TAB(8,15)"3 Draw a picture"
```

```
110 PRINT TAB(8,18)"4 End the program"
120 PRINT TAB(7,22)"Your choice, 1 to 4 ";
130 REPEAT
140 response=GET
150 UNTIL response>48 AND response<53
160 choice=response-48
170 ON choice PROCload, PROCsave, PROCdraw, PROCmake_sure
180 UNTIL choice=4
190 ENDPROC
```

(This program is incomplete and gives an error message if you run it).

The loop from lines 130 to 150 only ends when a key with an ASCII code between 48 and 53 is pressed. These are the ASCII codes for the numbers 1 to 4 on the keyboard, so this loop screens out accidental key depressions like Q or W.

Subtracting 48 from the ASCII code in line 160 gives a number from 1 to 4 again and line 170 uses this number to choose which of four procedures to execute. If *choice*=1, the computer carries out *PROCload*, with *choice*=2, it executes *PROCsave*, and so on. After carrying out the procedure the computer continues with line 180.

The program avoids the problems that might arise with a wrong key depression by only continuing when one of the keys 1 to 4 is pressed. However, the loop from lines 130 to 150 can be omitted and the problem of incorrect keys handled by an extension of the ON...PROC statement:

```
120 PRINT TAB(7,22)"Your choice, 1 to 4 ";
140 response=GET
160 choice=response-48
170 ON choice PROCload, PROCsave, PROCdraw, PROCmake_sure ELSE
PROCwrong_key
```

The computer executes *PROCwrong_key* if choice does not fall in the range 1 to 4. Only a single statement can follow the ELSE, although it need not be a PROC, for example:

```
170 ON choice PROCload, PROCsave, PROCdraw, PROCmake_sure ELSE
PRINT"Wrong key!"
```

ON...PROC is very useful, but note that it only works with numbers which must range from one upwards in steps of one. Normally, therefore you will need to carry out some kind of calculation in order to produce a suitable range of values.

Error handling

You can reduce the time you spend correcting errors in your programs by using procedures and sensible variable names, but it is inevitable that you will make some mistakes. The computer is able to identify some types of error itself, and gives an error message to let you know what is wrong.

You should always include an error-handling routine in your program. This tells you (or anyone else using the program) as much about the error as possible, and makes correcting it easier:

```
10 ON ERROR GOTO50
20 MODE 130
30 PROCmain_program
40 END
50 MODE 7
60 PRINT"Error number ";ERR;" at line ";ERL
70 END
```

This program contains a major error – there is no procedure called *PROCmain _ program*! Running the program gives this result:

```
Error number 29 at line 30
```

The ON ERROR statement at line 10 tells the computer that if it finds an error while it is running the program it should go to line 50. Every sort of error the computer can detect has an error number, and the computer uses the variable ERR to store this number. It uses ERL to store the line number at which the error occurred.

The Reference Manual gives a full list of the error numbers and describes the errors themselves in detail. However, you can get more information about the error from the computer itself by including a REPORT statement in the error-handling routine:

```
55 REPORT
60 PRINT " at line ";ERL
```

Running the program gives:

```
No such FN/PROC at line 30
```

This shows that the computer could not find a procedure called *PROCmain _ program* at line 30.

You have probably already had some experience of the computer giving error messages. As it does this automatically, you may wonder why you should bother including an error-handling routine at all. The main reason is that the routine can restore the computer to normal. Error messages can otherwise prove unreadable, as you will see if you RUN this program:

```
10 MODE 2
20 VDU 28,19,31,19,0
30 COLOUR 135
40 a terrible mistake
50 END
```

Add these lines to see the advantage of an error-trapping routine:

```
5 ON ERROR GOTO 60
60 MODE 7
70 REPORT
80 PRINT "at line ";ERL
90 END
```

More about strings

Strings are merely groups of characters and this section deals with their manipulation in BBC BASIC.

You can join together (concatenate) several strings simply by telling the computer to 'add' one string to the end of another:

```
10 MODE 135
20 first$="The start"
30 second$="and the end."
40 all$=first$+second$
50 PRINT all$
```

Running the program gives:

```
The startand the end.
```

Other than that the composite string may not exceed 255 characters, there is effectively no limit on the number of strings which may be joined together at one time. You could, for example, include an additional space in the line shown above using:

```
40 all$=first$+" "+second$
```

Two strings can also be compared using < = and > (or any combination of the three). The two strings are compared character by character until a difference is found. The string containing the character earliest in the alphabet is 'less than' the other string. For example:

- PUPPY is less than SHARK because P comes before S;
- PUPPY is less than RAT because P comes before R;
- PUPPY is greater than POPPY because both words begin with the letter P and U comes after O;

– PUPPY is greater than MONKEY because P comes after M.

If you are still not sure about this, run the following program which lets you compare pairs of strings:

```
10 MODE 135
20 REPEAT
30 INPUT LINE "What is the first string", first$
40 INPUT LINE "What is the second string", second$
50 IF first$<second$ THEN PRINT first$;" is earlier alphabetically
than ";second$
60 IF first$=second$ THEN PRINT "The two strings are identical."
70 IF first$>second$ THEN PRINT first$;" is later alphabetically
than ";second$
80 UNTIL first$="STOP"
```

Can you see how to stop execution of the program ?

The computer can sort a long list of strings into alphabetic order by using string comparisons like the above. Strictly speaking, the computer compares the ASCII codes of the characters concerned. A lower case letter like *a* is considered to be greater than the upper case *A* because the ASCII code for *a* is 97 and the ASCII code for *A* is only 65.

LEN enables you to find out how many characters there are in a string:

```
10 MODE 135
20 INPUT "What is your string ";choice$
30 length=LEN(choice$)
40 PRINT choice$;" contains ";length;" characters."
```

The earlier anagram program (see page 69) used LEN to find the length of the word supplied as input. It then rearranged the characters by combining parts of the string in a different order. There are several functions which enable you to copy part of a string:

```
10 MODE 135
20 example$="Yellow submarine"
30 PRINT "The word is ";example$
40 part1$=LEFT$(example$,4)
50 PRINT "The leftmost 4 letters are ";part1$
60 part2$=RIGHT$(example$,6)
70 PRINT "The rightmost 6 letters are ";part2$
80 part3$=MID$(example$,5,6)
90 PRINT "The middle 6 letters are ";part3$
100 part4$=MID$(example$,4)
110 PRINT "The letters from the 4th character are: ";part4$
```

LEFT$ and RIGHT$ work in a similar way, by taking the chosen number of characters from the left or right of the string respectively. MID$ is slightly different – in line 80 it is used to extract letters beginning at the 5th letter and going on for 6 letters. In line 100 the second number is omitted, which causes MID$ to extract all the characters from the 4th to the end of the string. Needless to say the numbers in each of the examples above may be replaced by numeric variables.

A string can be created which consists of a series of copies of another string using STRING$:

```
10 MODE 135
20 INPUT "What is your string ",text$
30 copy$=STRING$(10,text$)
40 PRINT "A string containing 10 copies looks like this:"
50 PRINT copy$
```

INSTR is used to check for the first occurrence of one string within another, for example:

```
10 MODE 135
20 INPUT LINE "Please type in any sentence",sentence$
30 check=INSTR(sentence$,"e")
40 PRINT "Your sentence contains ";
50 IF check>0 THEN PRINT "an 'e' at position ";check  ELSE PRINT "d
oes not contain an 'e'"
```

The variable *check* at line 30 contains the position within *sentence$* at which the first letter *e* occurs. If *sentence$* does not contain an *e*, *check* is 0. You can also search for groups of letters using INSTR. For example replacing line 30 with:

```
30 check=INSTR(sentence$,"the")
```

makes the program check for the string *the* within *sentence$*.

You cannot carry out arithmetic on a string variable, even if that string variable contains only numeric characters. This can be inconvenient, so there are two functions which enable you to change a number to a string and vice versa:

```
10 MODE 135
20 INPUT "What is today's date (eg 27) ";number
30 number$=STR$(number)
40 INPUT "What month is it ";month$
50 date$=month$+" "+number$
60 PRINT "Today's date is ";date$
```

STR$ in line 30 converts the numeric variable *number* into a string variable

number$. Lines 50 and 60 are included to demonstrate that the string version can be concatenated with other strings.

VAL gives the numeric value of a string:

```
10 MODE 135
20 INPUT "Type in any mixture of numbers and letters ";mixture$
30 number=VAL(mixture$)
40 IF number>0 THEN PRINT "The string begins with the numbers ";
number
```

If a string begins with numeric characters, a + or − sign, VAL converts those characters to their numeric equivalent. Note that VAL ignores the remainder of the string following the first non-numeric character it discovers,for example:

```
PRINT VAL("123g456") RETURN
```

produces 123.

READ, DATA and RESTORE

Many programs need some basic data before they can run, and it is often convenient to store that data as part of the program. For example, here is a simple quiz program that includes the questions and answers in DATA statements:

```
10 MODE 135
20 PROCstart
30 PROCquiz
40 END
50 DEFPROCstart
60 correct=0
70 READ how_many
80 PRINT TAB(14)"A quiz game"
90 PRINTSTRING$(40,"=")
100 ENDPROC
110 DEFPROCquiz
120 FOR question=1 TO how_many
130 READ question$,answer$
140 PRINT'question$
150 INPUT response$
160 IF response$=answer$ THEN PROCright ELSE PROCwrong
170 NEXT question
180 PRINT'"You had ";correct;" right out of ";how_many
190 ENDPROC
200 DEFPROCright
210 correct=correct+1
220 IF RND(2)>1 THEN PRINT'"That's right!" ELSE PRINT'"Well done!"
```

```
230 ENDPROC
240 DEFPROCwrong
250 PRINT'"No, the answer is:"
260 PRINTanswer$
270 ENDPROC
280 DATA3
290 DATAWhich century is this,20th
300 DATAWhich British king had six wives,Henry VIII
310 DATAWhat is the seed of an oak called,Acorn
```

The READ statement in line 70 makes the computer search through the program until it finds the first line beginning with the word DATA, which is 280. The computer reads the first value after the word DATA and stores it in the variable *how_many*. DATA items can either be numbers or strings, and are separated by commas.

The loop from lines 120 to 170 is carried out 3 times (the value of *how_many*). Line 130 successively reads a question and answer from the DATA statements. Each time through the loop the computer carries on reading data at the point at which it left off, so each time it reads a different question and answer.

Any number of data items can be included in a DATA statement, up to the maximum line length of 255 characters. Thus all the data in the program could be confined to a single line:

```
280 DATA3,Which century is this,20th,Which British king had six wiv
es,Henry VIII,What is the seed of an oak called,Acorn
```

The main reason for breaking the data up is that it makes changes easier. For similar reasons DATA lines are usually collected together although they can be placed anywhere within the program. You can add an extra question simply by inputting these lines:

```
280 DATA4
310 DATAWho won the 1984 World Cup,Italy
```

Running the program reveals one of the problems of using strings. The computer only accepts as correct a response that exactly matches its stored answer – for example, *Henry the Eighth* is treated as a wrong answer to question 2!

You can use the RESTORE statement to make a program read DATA beginning at a particular line. Add these lines to the quiz program to offer alternative questions:

```
91 PRINT "Do you want (1) general knowledge questions"
92 PRINT TAB(13)"(2) questions on animals"
93 INPUT "1 or 2",choice
```

```
94 IF choice=1 THEN RESTORE 280 ELSE RESTORE 500
500 DATA3
510 DATAWhat is the young of a wolf called,wolverine
520 DATAWhat is the largest mammal,whale
530 DATAWho killed Cock Robin,sparrow
```

Line 94 uses RESTORE to make the program read data beginning at line 280 or at 500, depending upon the set of questions are chosen.

Arrays

The computer is very useful for finding a particular data item in a long list or for sorting sets of data into a particular order. For example, you might want to sort a list of names into alphabetical order. The computer is quite able to do this, but it needs to compare every name with every other name to decide upon their order. All the names must be accessible at the same time, and it is easier to compare them if they are all stored in a list or **array**.

This program reads 10 names into an array and then displays any selected name:

```
10 MODE 135
20 PROCset_up_array
30 PROCfind
40 END
50 DEFPROCset_up_array
60 DIM name$(10)
70 FOR count=1 TO 10
80 READ name$(count)
90 NEXT count
100 ENDPROC
110 DATA Smith,Bloggs,Hutchings,Postlethwaite,Broome
120 DATA Turner,Dick,James,Neale,Sewell
130 DEFPROCfind
140 INPUT "Which name do you want (1-10)",number
150 PRINT'"Number ";number;" in the list is ";name$(number)
160 ENDPROC
```

The DIM statement in line 60 tells the computer how many items there are in the array – in this case, 10. The loop from lines 70 to 90 reads the names from data statements and automatically stores them in the array *name$*, so that *name$(1)* is Smith, *name$(2)* is Bloggs, and so on. *PROCfind* at 130 is included so that you can confirm for yourself that the computer has stored the names in the order they are given in the DATA statements.

The program can search through the array very rapidly to find a name or set of

names which meet certain requirements. For example, to find all names beginning with a particular letter, change the last few lines to:

```
140 INPUT "Which letter should the name begin with ",letter$
150 FOR count=1  TO 10
160 name$=name$(count)
170 IF LEFT$(name$,1)=letter$ THEN PRINT name$
180 NEXT count
190 ENDPROC
```

This program contains only a few names, but the computer can deal just as easily with a list of several hundred names – the limiting factor is the computer's memory capacity.

It is more common to deal not with a single array but with several simultaneously. We usually make lists of data items that are in some way associated – names and addresses, books and their authors, and so on. For example, if names and ages are being stored we can set up two arrays. The association between the arrays makes it easy for the computer to carry out searches. If Broome is the fifth name in the *names* array, his or her age is fifth in the *age* array:

name$(5)="Broome" age(5)=27

Here the age is stored in a numeric array age( ) rather than a string array, because we may want to carry out a calculation involving the age.

This program stores the names and ages of 10 people, and searches the array to find the age of any person once you have input their surname:

```
10 MODE 135
20 PROCset_up_array
30 PROCfind_age
40 END
50 DEFPROCset_up_array
60 DIM name$(10), age(10)
70 FOR count=1 TO 10
80 READ name$(count), age(count)
90 NEXT count
100 ENDPROC
110 DATA Smith,42,Bloggs,35,Hutchings,57
120 DATA Postlethwaite,35,Broome,49,Turner,23
130 DATA Dick,39,James,24,Neale,63,Sewell,75
140 DEFPROCfind_age
150 INPUT "Whose age do you want ",search$
160 count=1
170 REPEAT
```

```
180 name$=name$(count)
190 IF name$=search$ THEN PRINT name$;" is ";age(count)
200 count=count+1
210 UNTIL count=11 OR name$=search$
220 IF name$<>search$ THEN PRINT search$;" is not in the list"
230 ENDPROC
```

Only one DIM statement is needed to set up the size of both arrays, line 60. The loop from 170 to 210 examines each name in the array to see if it is the one required.

It is also possible to use integer arrays. In the previous program all the ages were integers, and could have been stored in an array *age%( )*.

Files

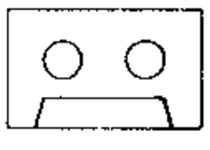
The last section showed how you can store data in arrays. One weakness of this storage method is that it wastes computer memory. Every data item is stored twice: once as part of the DATA statements within the program, and again elsewhere in memory when the computer copies each data item into the array.

A more sensible method is to store the data completely separately from the program, as a **data file**. The file can be saved onto cassette or disc (in a similar manner to a program) and can be loaded back when required.

This program creates a file of names and telephone numbers:

```
10 MODE 135
20 PROCtake_names
30 PROCmake_file
40 END
50 DEFPROCtake_names
60 DIM name$(100), tele$(100)
70 PRINT'"Please type in the names and"
80 PRINT"telephone numbers of your friends."
90 PRINT"You can end by typing XXX when you"
100 PRINT"are asked for a name."'
110 count=0
120 REPEAT
130 count=count+1
140 INPUT "Name",name$(count)
150 IF name$(count)<>"XXX" THEN INPUT "Telephone number",tele$(count)
160 UNTIL name$(count)="XXX" OR count=100
170 ENDPROC
180 DEFPROCmake_file
190 CLS
200 PRINT'"What name do you want to give to"
```

```
210 INPUT"your data file",file$
220 this_one=OPENOUT(file$)
230 FOR number=1 TO count-1
240 PRINT#this_one,name$(number),tele$(number)
250 NEXT number
260 CLOSE#this_one
270 ENDPROC
```



The program will display the message:

RECORD then RETURN

when it wants to save the data file – you will also have to stop the tape recorder if your equipment does not provide motor control.

PROCtake_names sets up two string arrays which can hold up to 100 names and telephone numbers. The loop from 120 to 160 takes input from the keyboard and stores the names and numbers in the two arrays.

PROCmake_file creates the file, which is given a name at line 210. Line 220 **opens** the file using OPENOUT so that data can be output to it.

BBC BASIC allows you to have up to five files open at the same time. Each file is given a number by the computer so that it can distinguish between files. This number is called the **channel number**. All references to the file are made via the channel number, so it is vital that it is saved. Line 220 stores the channel number for the file in the variable *this_one*.

The loop from lines 230 to 250 writes the data out to the file. Line 240 tells the computer to print the data out via channel *this_one*.

The computer needs to be told that there is no more output, so line 260 **closes** the channel once all the data has been printed to the file.

Note that running the program only saves the file containing the names and telephone numbers. The program itself must be saved in the same way you would save any other program.

A file is of little use unless you can read the information stored in it and the followings program reads the names and phone numbers in the file back into memory, and finds the phone number for any friend whose name you have stored on the file:

```
10 MODE 135
20 PROCread_file
30 PROCfind_number
40 END
50 DEFPROCread_file
60 DIM friend$(100), numb$(100)
```

```
70 PRINT'"What name did you give to"
80 INPUT"your data file",file$
90 that_one=OPENIN(file$)
100 count=0
110 REPEAT
120 count=count+1
130 INPUT#that_one,friend$(count),numb$(count)
140 UNTIL EOF#that_one
150 CLOSE #that_one
160 ENDPROC
170 DEFPROCfind_number
180 CLS
190 INPUT "Whose number do you want",name$
200 search=0
210 REPEAT
220 search=search+1
230 IF name$=friend$(search) THEN PRINTname$;" has the number
";numb$(search)
240 UNTIL search=count OR name$=friend$(search)
250 IF name$<>friend$(search) THEN PRINT"I can't find this name"
260 ENDPROC
```

PROCread_file reads the contents of the file back into memory and stores the names and phone numbers in two arrays *friend$( )* and *numb$( )*.

Line 90 opens the file using OPENIN so that data can be input from it. Once again we save the channel number, this time storing it in the variable *that_one*.

The loop from lines 110 to 140 reads in items from the file and stores the data in the arrays. Line 130 inputs data via the channel *that_one*.

The computer does not know how many data items there are in the file, so it continues to read data until it reaches the End Of File (EOF) at line 140. As there is no more data, line 150 closes the file.

All the data has now been copied from the file into the arrays *friend$( )* and *numb$( )*, and *PROCfind_number* searches those arrays for the phone number if you input as friend's name.

The previous two programs are very simple and only illustrate the principles of using files. Much more sophisticated software is available which allows you to create and modify files of data of any nature, rather than specifically names and phone numbers. If you have a disc-based machine you can expand your system to include ViewStore, a very powerful file-handling program, details of which are available from Acorn.

More about graphics

In any graphics mode a fixed number of pure colours can be shown on the screen simultaneously. Four other patterns made up from a combination of the pure colours can also be displayed. For example, in mode 129 four pure colours are available, and four patterns. This program displays all eight colours at the same time by drawing seven rectangles on a background of black:

```
10 MODE 129
20 PROCpure
30 PROCmixed
40 END
50 DEFPROCpure
60 FOR colour=1 TO 3
70 GCOL0,colour
80 corner=80*colour
90 PROCrectangle(corner,corner,corner+100,corner+100)
100 NEXT colour
110 ENDPROC
120 DEFPROCmixed
130 FOR colour=16 TO 64 STEP 16
140 GCOL colour,0
150 corner=80*((colour/16)+3)
160 PROCrectangle(corner,corner,corner+100,corner+100)
170 NEXT colour
180 ENDPROC
190 DEFPROCrectangle(x,y,x1,y1)
200 MOVE x,y
210 PLOT 101,x1,y1
220 ENDPROC
```

PROCpure draws rectangles in the pure colours following the GCOL 0 command at line 70.

PROCmixed draws rectangles using the patterns. Each time through the loop the pattern is dictated by the GCOL command at line 140. The first time this is GCOL 16,0; the next GCOL 32,0; and so on.

The patterns dictated by GCOL 16,0 and the other high-numbered GCOL commands are not fixed, and can be changed by a VDU command. Add the following lines to the program and run it again:

```
121 REM gives yellow/black shading for GCOL 16,0
122 VDU23,2,160,80,160,80,160,80,160,80
```

The command VDU 23,2 changes the pattern produced by GCOL 16,0. The eight numbers following describe the new pattern – in this case alternate black

and yellow areas. Similarly, VDU 23,3 can be used to give a new pattern following GCOL 32,0 and VDU 23,4 changes the pattern produced by GCOL 48,0. Add these lines to get a completely new set of patterns:

```
123 REM gives large block red/yellow shading for GCOL 32,0
124 VDU23,3,60,195,60,195,60,195,60,195
125 REM gives black/red shading for GCOL 48,0
126 VDU23,4,5,10,5,10,5,10,5,10
127 REM gives black/white shading for GCOL 64,0
128 VDU23,5,85,170,85,170,85,170,85,170
```

Working out which eight numbers produce which pattern is a little complex, and the procedure varies from mode to mode – you will find it easier to use the pattern generator utility (PFILL) from the Welcome software and which is described at the end of this chapter. Further information about the way the VDU23 command works is given in the Reference Manual.

PFILL lets you define your own pattern and displays the numbers needed to recreate it. The numbers are shown in **hexadecimal** form (counting in 16s). Do not worry if you are not familiar with hexadecimal. You need only put these numbers in a suitable VDU statement to use the pattern in your own programs. For example:

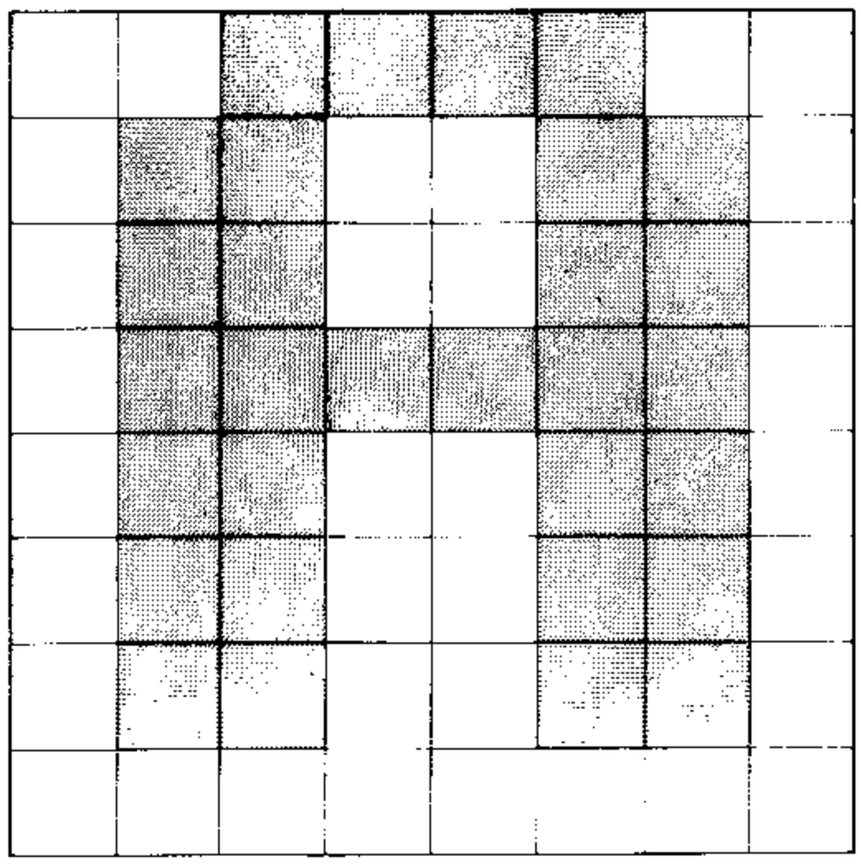
```
122 VDU23,2,&A0,&50,&A0,&50,&A0,&50,&A0,&50
```

is the hexadecimal equivaalent of the previous line 122 and has the same effect. (The & symbol is used to denote that the number following is in hexadecimal notation.)

Defining your own characters

VDU 23 can also be used to define new characters for games or for specialist programs in science or mathematics which require unusual notation. You may recall that the symbol for *pi* was used as an example in the introduction to this guide.

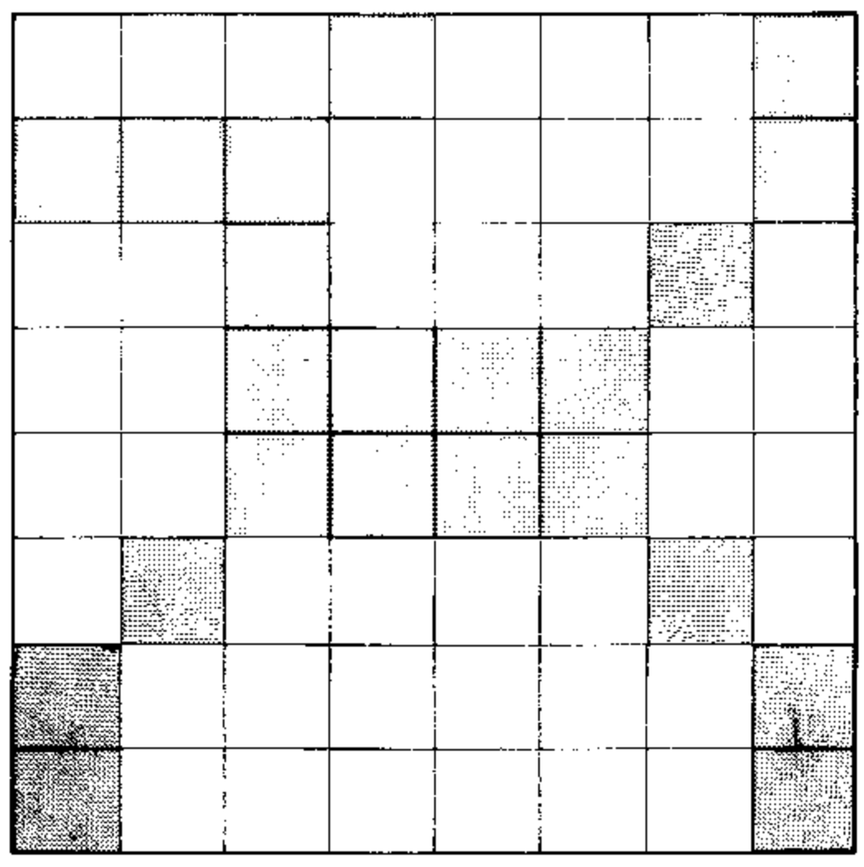
All the normal characters are based on an 8 by 8 grid, so the uppercase A looks like this:



Any of the characters can be redefined, but changing the upper case A to some other shape does not help the readability of programs!

Here is a 'dog' character which has been drawn on the 8 by 8 grid:

128 64 32 16 8 4 2 1



You can redefine character 255 as the dog by typing:

```
VDU 23,255,17,225,34,60,60,66,129,129
```

To see the character, try typing:

```
MODE 129 RETURN
PRINT CHR$(255) RETURN
```

Each number after VDU 23,255 describes one of the eight rows of points which together make up the character, from top to bottom. To get this number you must first note the points within the row which will be 'lit' when the figure is

displayed. For example, in the top row only the fourth and last points will be lit. The number to describe this row is 16+1=17, obtained by adding the figures above these two points.

Similarly, the second row is described by the number

128+64+32+1=225, the third row by 32+2=34, and so on.

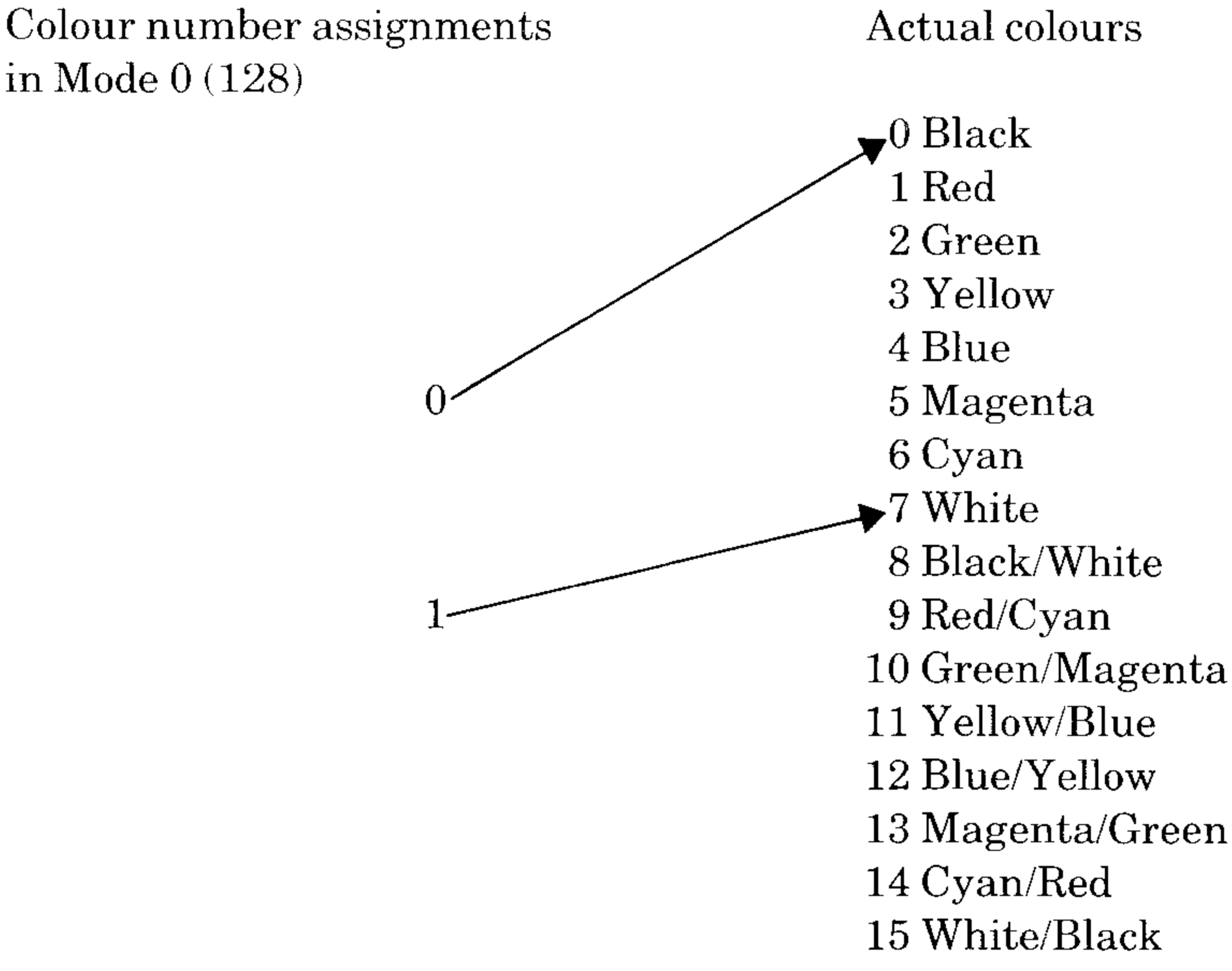
Draw up an 8 by 8 grid and try defining a character of your own.

The Welcome software also contains a character design utility called CHARDES which provides an automated method of changing the characters which the computer can display. This utility is described at the end of this chapter.

Changing the range of colours

Modes like 128 only allow two colours to be displayed on the screen at any one time – for example, the normal colours for mode 128 are black and white. Although there is no way you can use more than two colours simultaneously in mode 128, you can change the range of colours available i.e. instead of black and white you could choose red and yellow.

However, the numbers used in GCOL and COLOUR commands produce different effects in different modes and the colour displayed depends upon two sets of information.

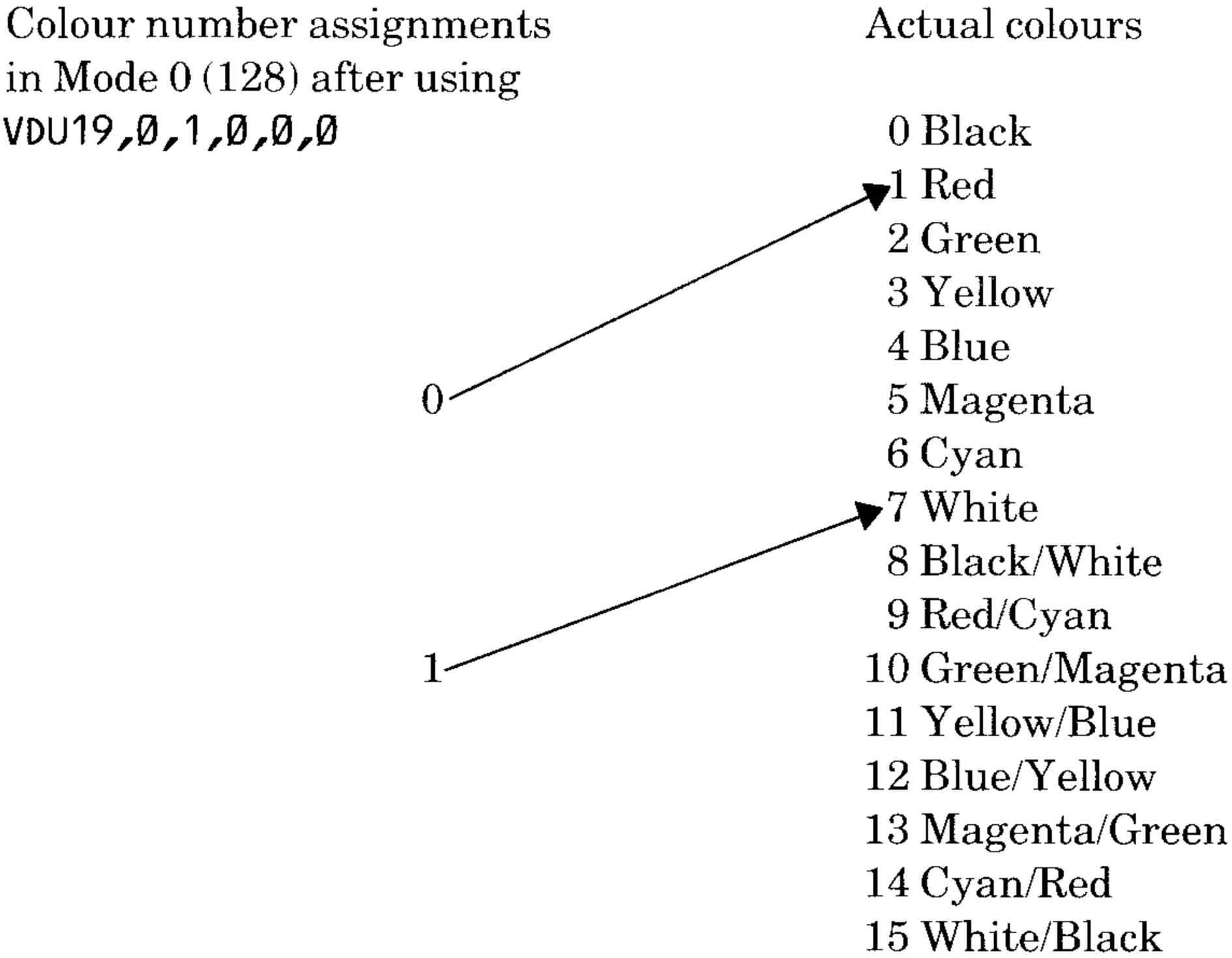


The list on the right shows what are called the **actual** colour numbers of the 16 pure colours. This list never changes and is the same for every mode. The way the colour numbers for the mode are associated with this actual colour list can be varied by using the VDU 19 command. For example, type:

```
MODE 128[RETURN]
VDU 19,0,1,0,0,0[RETURN]
```

This instantly changes the normal black background colour to red.

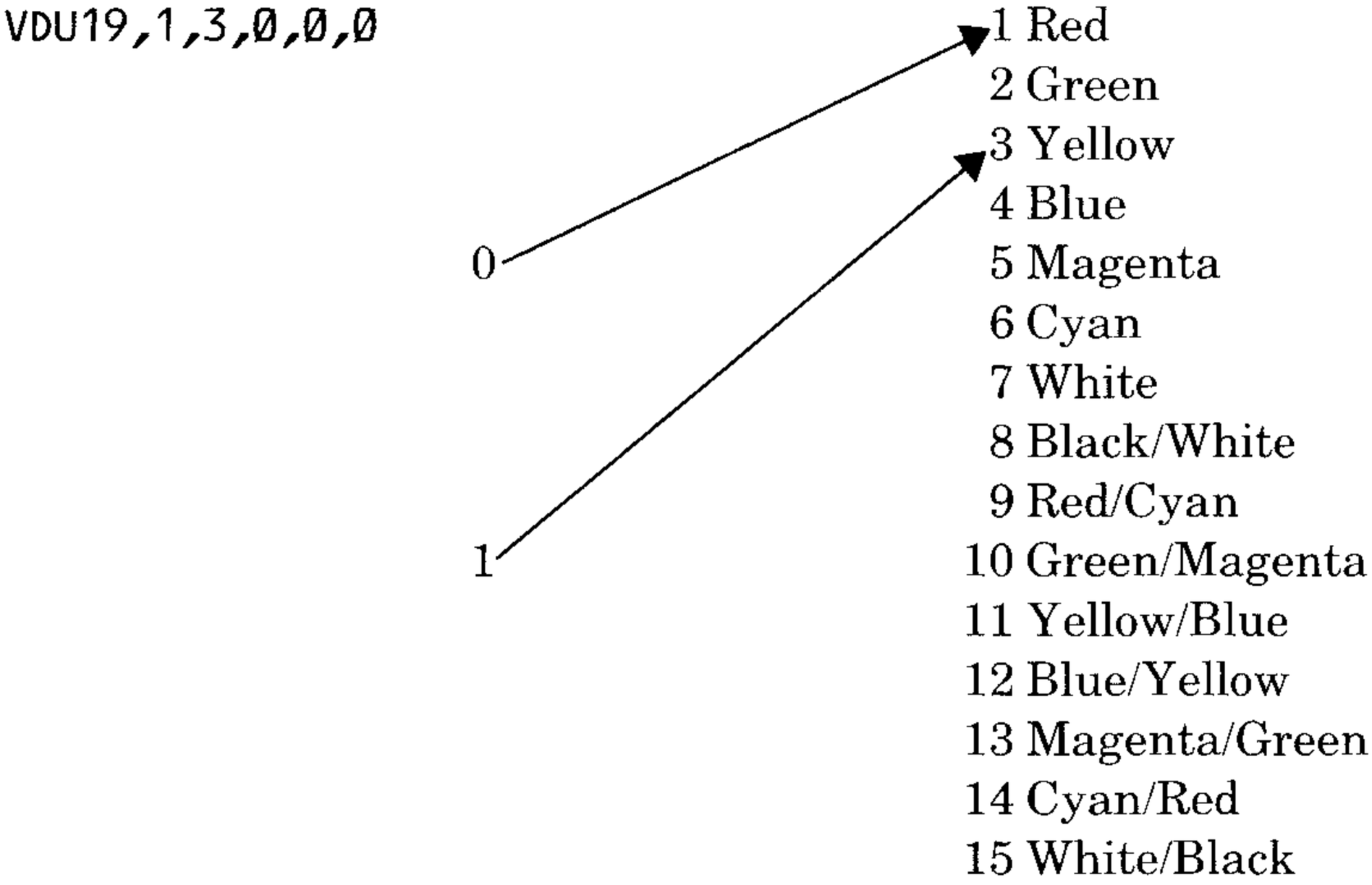
The first number after VDU 19 is 0, which normally produces black in mode 128. The second number refers to the actual colour number 1, which always stands for red. The VDU 19 command effectively changes the association between the colour numbers and the actual colours:



Similarly, white can be changed to yellow by:

```
VDU 19,1,3,0,0,0[RETURN]
```

(The last three zeroes are for future expansions and they must be included even though they and have no effect.)



The same principle applies in all other modes except 7 and 135 – a full list of the normal colour number assignments is given in Appendix A.

The teletext mode

Modes 7 and 135 are unique in the way they display text and graphics. Commands such as COLOUR, GCOL, MOVE and DRAW do not work in these modes. Instead, colourful displays are produced using what are known as **teletext control codes**.

You may have seen teletext pages broadcast by CEEFAX or Oracle – modes 7 and 135 are teletext compatible modes.

The computer lets you produce your own teletext displays using mode 7 or 135. These modes use very little memory, and offer a wide range of colours for simultaneous display on-screen. The graphics are limited but effective. Throughout the rest of this section only mode 135 will be discussed, but all comments apply equally to mode 7.

This program demonstrates the text colours available in mode 135:

```
10 MODE 135
20 PRINT "This";CHR$(129);"shows how a control code"
30 PRINT "only effects the";CHR$(130);"characters"
40 PRINT "after it on the";CHR$(131);"same";CHR$(129);"line."
```

The PRINT statement at line 20 prints some text containing a series of

invisible *control codes*. Each code takes up a character position, so the words are printed with spaces between. The codes affect the way the remaining characters on that particular line are displayed. For example, printing CHR$(129) before "shows" makes the computer display the text in red, CHR$(130) causes the text after it to be printed in green, and so on.

Printing any of the ASCII codes 129 to 135 affects the colour of any characters printed after the code on the same line. Try:

```
PRINT CHR$(130)"Test"[RETURN]
```

which prints in green. A full list of the teletext control codes is given in Appendix B.

The colour of text can be changed directly from the keyboard. Hold down [SHIFT] and at the same time press the red function key [f1]. This prints the control code 129. Any characters you type on the same line will now be displayed in red. Pressing [SHIFT] and any of the function keys [f1] to [f7] gives a different colour for any text printed afterwards on the same line.

You can also make text flash:

```
PRINT CHR$(136);"Flash";CHR$(137);"no flash";CHR$(136);"flash" [RETURN]
```

Flashing coloured text can be produced by using two control codes:

```
PRINT "Flashing";CHR$(129);CHR$(136);"red"[RETURN]
```

The codes each occupy a character position, so the words are printed separated by two spaces.

Again, the same effects are possible using the function keys. SHIFT and f8 print the control code for flashing, SHIFT and f9 print the non-flashing code.

Double height characters can be printed using CHR$(141):

```
10 MODE 135
20 PRINT CHR$(141);"Double height"
30 PRINT CHR$(141);"Double height"
```

The same text must be printed on two successive lines beginning with CHR$(141), otherwise only the top half of the letters is displayed.

Changing the background colour uses two codes:

```
PRINT CHR$(131);CHR$(157)[RETURN]
```

The first code is for yellow text. CHR$(157) tells the computer to use the previous control code as the background colour. The net effect of the two codes is to give yellow text on a yellow background, as you can see if you type:

```
PRINT CHR$(131);CHR$(157);"Hello"[RETURN]
```

This is obviously not very useful, as the text is unreadable. To print text visibly on a coloured background requires three control codes, two codes to change the background colour and a third to change the colour of the text:

```
PRINT CHR$(131);CHR$(157);CHR$(132);"Blue on yellow"[RETURN]
```

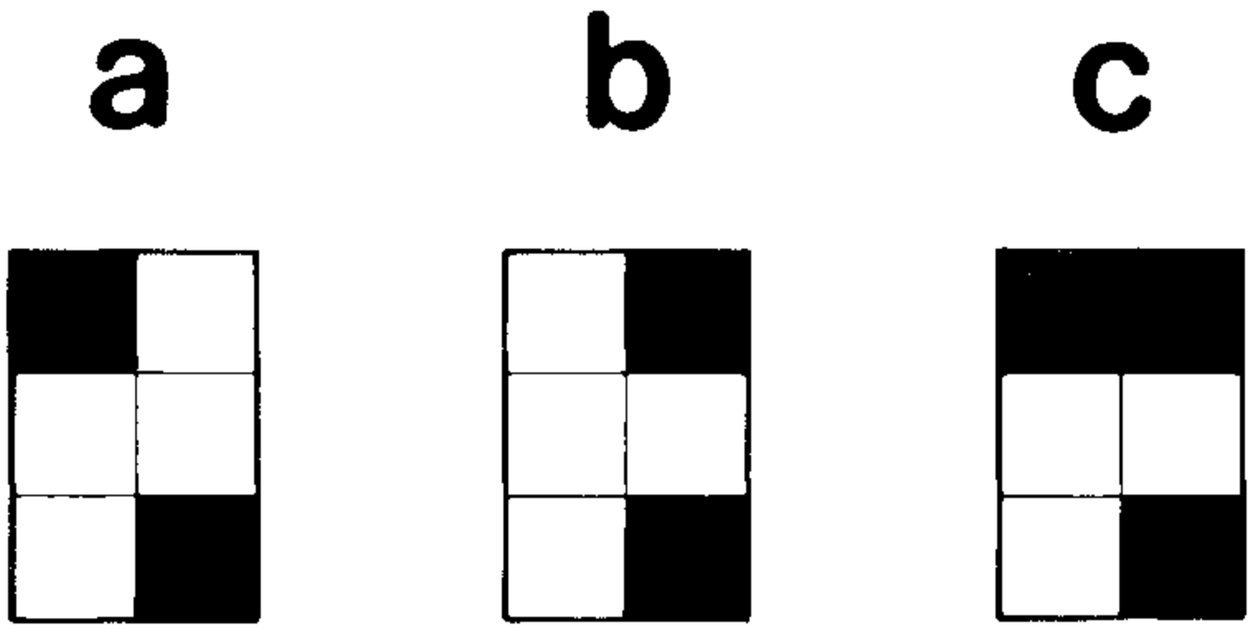
The first two codes set the yellow background and CHR$(132) is the code for blue text.

All of these codes can be combined and incorporated into strings. If you intend to use a particular set of codes many times within a program it is useful to set up a single string containing those codes:

```
10 MODE 135
20 ryflash$=CHR$(131)+CHR$(157)+CHR$(129)+CHR$(136)
30 PRINT ''ryflash$;"A demonstration"
40 PRINT "of normal printing";ryflash$;"and in colour"
```

Teletext graphics

All graphics in mode 135 are produced as the result of printing characters. If any line contains a graphics control code, any characters other than uppercase letters that appear after it on the same line are printed as graphics shapes. Each letter corresponds to a particular shape which is based on a two by three grid for example:



A full table showing the graphics shape associated with each character, together with the graphics control codes, is given in Appendix B.

The printing of any of the ASCII codes 145 to 151 causes characters on the same line to be printed in their graphics form. Upper case letters are unaffected:

```
PRINT CHR$(145);"Aa";CHR$(146);"Bb";CHR$(147);"Cc"[RETURN]
```

It is easier to appreciate the effectiveness of teletext graphics when a series of graphics characters are displayed together:

```
PRINT CHR$(148);STRING$(30,"9")[RETURN]
```

Graphics characters can be displayed in double height, on different backgrounds, or flashing.

You can produce the graphics control codes directly from the keyboard by pressing [CTRL] and any of the function keys [f1] to [f7] simultaneously. Any non-upper case characters you subsequently type on the same line will be displayed as graphics shapes.

Sound

Your computer contains a **sound generator** with four **channels**.

Two BASIC commands are available that give a wide degree of control over sound. The SOUND command is used to play single notes. For example:

```
SOUND 1,-15,53,20[RETURN]
```

plays a note on channel 1 at maximum loudness for 1 second. The command can be summarised as:

SOUND channel,loudness,pitch,duration

The first of the four parameters after SOUND denotes the channel number. This can be 0 to 3, with channel 0 producing noises for special effects, and channels 1 to 3 producing musical notes. For example:

```
SOUND 0,-15,53,20[RETURN]
```

changes only the channel number from the previous example but gives a very different effect.

The second parameter controls the loudness or amplitude of the note, and can have any value from -15 to 16. The loudest is -15, -14 is quieter and other negative numbers give softer sounds up to 0, which is silence. Any positive number from 1 to 16 indicates the sound is under the control of an ENVELOPE command (discussed shortly).

The third number gives the pitch of the note, and can have any value from 0 to 255. Low values produce deep notes; high values, high notes. The pitch value has a different effect if channel number 0, the noise channel, is used. In this case the range for the third parameter is only 0 to 7, producing various pitches of noise.

The last parameter shows the duration of the sound in twentieths of a second, and can have any value from 0 to 255. In the example, this value is 20, so the note sounds for 1 second (20 twentieths of a second). A value of 255 produces a continuous sound that stops only if you press [ESCAPE].

To play a simple tune you need only sound several notes in succession:

```
10SOUND 1,-15,97,10
20SOUND 1,-15,105,10
30SOUND 1,-15,89,10
40SOUND 1,-15,41,10
50SOUND 1,-15,69,20
```

Notes can be sounded simultaneously on another channel if you add:

```
15SOUND 2,-15,97,10
25SOUND 2,-15,105,10
35SOUND 2,-15,89,10
45SOUND 2,-15,41,10
55SOUND 2,-15,69,20
```

Sounds with a loudness parameter of 1 to 16 are controlled by the envelope with the corresponding number. The envelope can affect both the pitch and amplitude of a note. For example:

```
SOUND 1,-15,255,255 RETURN
```

plays a continuous loud note. Change the second parameter to 1 and the note comes under the control of envelope 1. The ENVELOPE command requires 14 parameters:

```
ENVELOPE 1,1,-26,-36,-45,255,255,255,127,0,0,-127,126,0 RETURN
```

The number immediately after ENVELOPE is the envelope number, which can vary from 1 to 16. The remaining parameters control and vary the pitch and amplitude of the note. Try the same note as before, but under the control of envelope 1:

```
SOUND 1,1,255,255 RETURN
```

The SOUND and ENVELOPE commands are extremely versatile and together enable the computer to function as a music-maker superior to much more costly synthesisers – both commands are discussed in detail in the Reference Manual. In addition, the Welcome software includes an envelope editor (called ENVELOPE) which allows you to experiment with the parameters in the envelope command.

Changing the time

The introduction to this User Guide describes how you can use the control panel utility to reconfigure your machine – one of the facilities offered is to reset the date and time. You can do the same thing more directly by using the variable TIME$ which enables you to read or alter the time.

For example:

```
PRINT TIME$[RETURN]
```

displays the day, date and time. Typing:

```
TIME$="Tue,7 Jan 1986.09:00::00"[RETURN]
```

sets the time to 9.00am on Tuesday January 7th 1986. The comma, spaces, full stop and colon characters are important as they separate the day, date and time. Either the date or the time can be omitted, so the following are also valid:

```
TIME$="12:27:35"
TIME$="Fri, 6 Jun 1986"
```

128K BASIC

Your computer is equipped with a ROM-based version of BBC BASIC which, in conjunction with the shadow memory facilities can access up to 64K of the available 128K of random-access memory (RAM). Access to the remaining 64K of paged RAM is possible using a disc-based version of BBC BASIC (referred to as BAS128) which is available from your supplier.

Assembly language

Although programs in BBC BASIC run very quickly, some programs – such as games – need to run even more rapidly if they are to be effective. Every time the computer runs a program written in BASIC, it has to translate (or **interpret**) each statement so that it can carry out the necessary function using routines written in the computer's internal language – *machine code*. It is the translation process which slows the computer down.

Writing a program directly in machine code means the computer need not interpret each statement, so a machine code program runs many times faster than its BASIC equivalent. However, writing a program as a series of numbers is extremely difficult. Instead the program is written in **assembly language**.

The computer translates an assembly language program into machine code using a built-in program called an **assembler**. The machine code translation of the program can be saved on its own. When the computer next runs the program it does not need to translate any of the instructions, and so execution is very rapid.

Assembly language is more difficult to use than BASIC, although it results in faster and shorter programs. Fortunately, however, your computer allows you to mix BASIC and assembly language in one program, and it is sensible to only use assembly language in sections of a program where speed is vital.

This brief program demonstrates the use of assembly language – be sure to type it in *exactly* as shown:

```
10 MODE 135
20 DIM demo 10
30 P%=demo
40 [
50 LDA #67
60 JSR &FFEE
70 RTS
80 ]
90 P%=demo
100 END
```

Line 20 reserves 10 memory locations to hold the machine code version of the program. *P%* at line 30 is used to indicate to the computer the first memory location to be used for the machine code program.

The brackets at lines 40 and 80 indicate the beginning and end of the assembly language section of the program. The short program from line 50 to 70 prints the letter C on the screen. If you run the program you will see the following:

```
E71
E71 A9 43               LDA #67
E73 20 EE FF            JSR &FFEE
E76 60                  RTS
```

The computer has used the assembler to translate the assembly language instructions into machine code. The numbers in the left-most column are the hexadecimal memory locations where the machine code is stored. Each remaining hexadecimal number on the line is the equivalent of one assembly language instruction.

Notice that running the program has only translated the assembly language into machine code, and does not run the machine code program itself. To actually execute the machine code, type:

```
CALL P%
```

CALL is a statement to the computer to execute a piece of machine code. It is followed by the memory location at which the execution of the machine code is to begin. You should find that the computer prints the letter C.

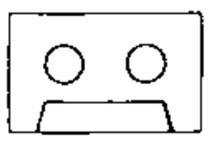
Once an assembly language program has been translated by the assembler, the machine code program can be run independently. If you use NEW, the original program is removed but the machine code remains in memory, as you can prove by typing `CALL P%` again. The section of memory containing the machine code can be saved on its own and used again without any need for the original

program containing the assembly language instructions.

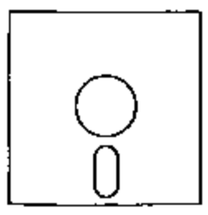
The facility for mixing BASIC and assembly language instructions is a powerful one, but any further discussion on the subject is outside the scope of this guide. Assembly language is discussed in detail in the Reference Manual.

Utility programs

Three further utility programs are provided on the Welcome tape and its disc equivalent.

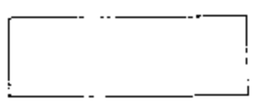
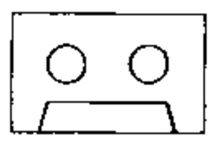


The programming utilities are located on side 2 of the tape, immediately following the Welcome utilities. Load each one using the command given within each description.



You may use the commands described below to run any of the programming utilities individually. Alternatively, you may use the menu system by typing:

```
*ADFS RETURN
CHAIN"UTILITIES" RETURN
```

CHARDES   loading time 2 minutes

This utility allows you to alter the shapes of the letters and numbers that appear on the screen; in other words, it allows you to design your own *fonts*. The utility is executed by means of the command:

```
CHAIN"CHARDES" RETURN
```

The screen display is divided into three areas, the top showing all the current character shapes, the central box showing an enlarged version of the currently-selected character (together with a normal size version to its right) and the bottom area providing a summary of the operating keys.

Characters to be redefined may be selected in one of two ways:

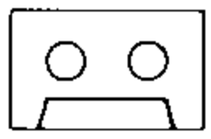
- by pressing the appropriate key on the keyboard (for standard keyboard characters);
- by using the cursor control keys to position the cursor under one of the characters at the top of the screen and pressing DELETE. This method may be used to select both standard characters and those which cannot be obtained directly from the keyboard.

Once a character has been selected, an enlarged version is shown in the central box. Thereafter, the cursor keys may be used to select a particular element in the central box and depression of RETURN changes its state (i.e. if it is currently is white, RETURN switches it to black and vice versa). The effect of any change is reflected immediately in the character to the right of the grid.

ESCAPE is used to end execution of the program.

[COPY] is used to save the current font to either tape or disc, making it possible to design a number of fonts, each of which may be reloaded when required.

[TAB] is used to reset the font to normal. Note that unless you use [TAB] before ending the utility, the effects of any changes to the font will remain until you switch the computer off or execute a hard break.

ENVELOPE [ ]  loading time 1 minute

Brief mention of the BASIC ENVELOPE command has been made on page 98, but the fact that it takes no less than 14 parameters makes it unsuitable for description in a guide of this nature. However, ENVELOPE is a utility program which enables you to experiment with the envelope command. It allows you to change some or all of the various parameters and to listen to the effect that the changes have upon the sounds generated by the computer. It may also be used to determine the parameters necessary to generate a particular sound for inclusion in say, a computer game.

ENVELOPE is loaded by means of the command:

```
CHAIN"ENVELOPE"[RETURN]
```

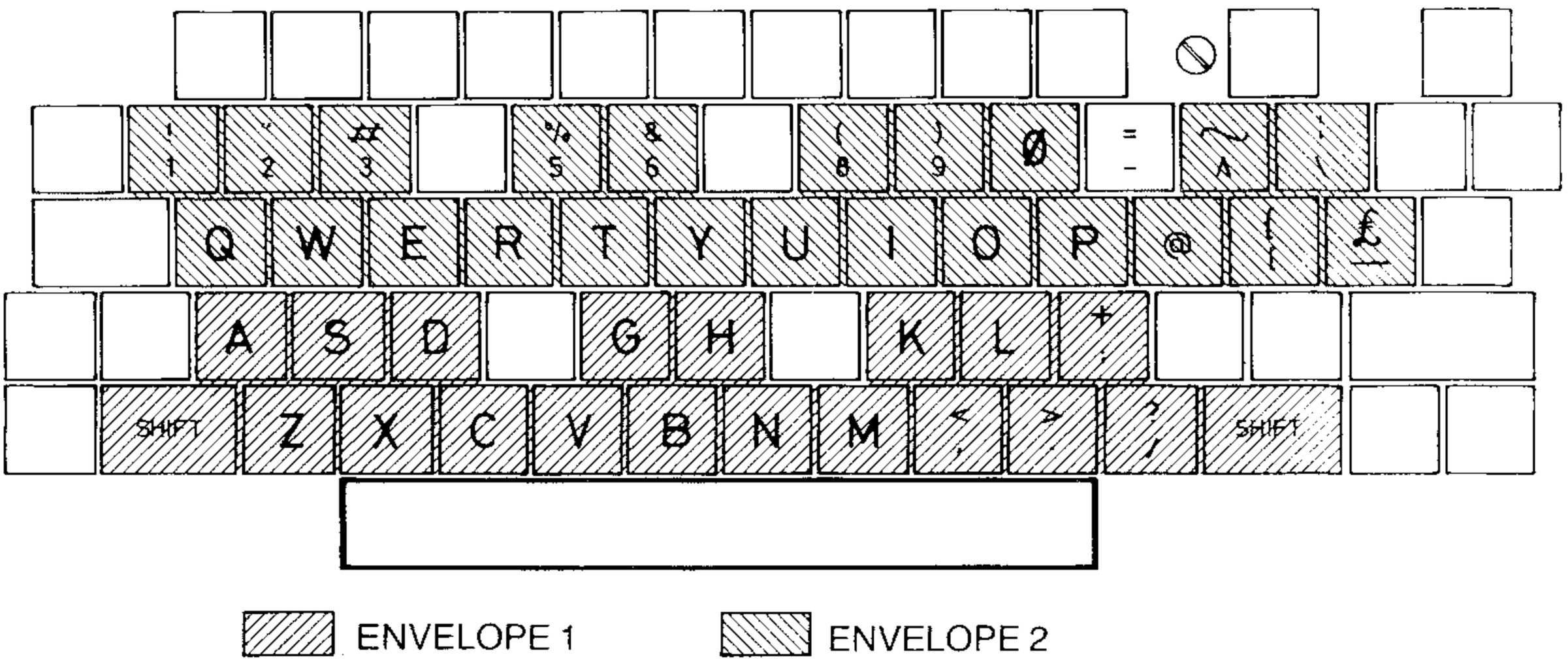
One loaded, ENVELOPE displays a number of boxes. Two different envelopes may be defined by changing the content of the box marked `Number`; the remaining boxes represent the settings of the fourteen parameters associated with the currently-selected envelope:

Length	–	Length of each step 1/100sec
Pstep1	–	Change of pitch per step in 1
Pstep2	–	Change of pitch per step in 2
Pstep3	–	Change of pitch per step in 3
Steps1	–	Number of steps in section 1
Steps2	–	Number of steps in section 2
Steps3	–	Number of steps in section 3
AstepA	–	Amplitude change in attack
AstepD	–	Amplitude change in decay
AstepS	–	Amplitude change in sustain
AstepR	–	Amplitude change in release
Peak	–	Target level at end of attack
Level	–	Target level at end of decay

You can move between the boxes by using ← and →; the current box will be

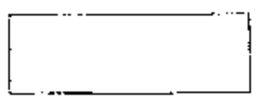
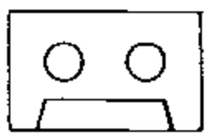
highlighted in black. To increase or decrease the value held in the current box press ↑ or ↓ as required. Alternatively, you can load a number of 'pre-set' envelopes by pressing any of the function keys.

The effect of the current set of ENVELOPE parameters can be heard using the keyboard, which is divided up into two 'piano-style' keyboards:



You may press one or several keys at a time, from either one or both envelopes.

You may replace any of the preset envelopes with one of your own by pressing [COPY] followed by the number of the envelope you wish to redefine.

PFILL   loading time 1 minute

PFILL allows you to design your own colour patterns in any of the graphics modes. It is executed by means of the command:

```
CHAIN"PFILL"[RETURN]
```

and you will first be asked which mode you wish to use. Choose which you would like to use and type the number, the possibilities are :

Mode 0 (128)	–	2 colours, 640 by 256 pixels.
Mode 1 (129)	–	4 colours, 320 by 256 pixels.
Mode 2 (130)	–	16 colours, 160 by 256 pixels.
Mode 4 (132)	–	2 colours, 320 by 256 pixels.
Mode 5 (133)	–	4 colours, 160 by 256 pixels.

The other modes are unsuitable because they do not allow the display of graphics.

Once the mode has been selected, a grid will be shown on the screen, with a flashing cross in the top left-hand box; the cross can be moved around using the cursor control keys.

The range of colours you can use to fill each box in the grid is given at the bottom of the screen and to fill the box currently marked with a cross, simply press the corresponding number. Each time you fill a box the large rectangle at the top of the screen will be filled with the current pattern from the grid. The eight parameters required to specify the current pattern are always displayed at the side of the grid. Note, however, that the parameters are shown in hexadecimal notation (i.e. the values are preceded with &).

Having produced a satisfactory pattern you can note down these numbers and use them in your own programs to fill any of the solid shapes (such as triangles, circles or ellipses) which the computer can plot.

Down the right hand side of the screen you will see a strip of coloured blocks. This is the palette – it allows you to change the relationship between the colour numbers and the actual colour which is seen. To alter it type P.

The pointer by the palette will then start flashing and can be moved up and down the strip using ↑ and ↓. To change the appearance of a colour number press one of the keys 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E or F; this will select the actual colour from the sixteen which are available. To return to the grid press RETURN.

3. Introducing VIEW

What is word processing?

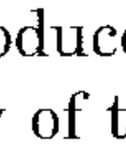
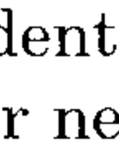
Word processing has had a more profound impact on office practice than any other application of computer technology. Consider the number and variety of documents that are produced daily in homes and offices. Letters, memos, membership lists, agendas, reports.... the list is endless.

In many cases, a document will undergo several changes before appearing in its final printed, or written, form. Using a conventional typewriter, for example, a rough draft may be produced, edited by hand, then retyped to obtain a final copy. If, perhaps at a later date, a similar document is required but with minor modifications, the whole document will have to be retyped.

A word processor offers significant advantages over the typewriter. Text entered on a word processor appears on the monitor screen for editing and inspection before it is committed to paper. Corrections and modifications are simple to implement. Characters, lines or complete paragraphs can be inserted, deleted or moved about at will. Furthermore, text may be stored for later use with modified details such as names and addresses. Personalised copies of a standard letter can be produced, all identical except for individual names and addresses. The main body of the letter need be typed only once.

The VIEW word processor

Your computer is supplied with **VIEW**, a powerful built-in word processor. VIEW has established itself as one of the more popular word processors available for use on microcomputers. Whether your particular requirements are business or domestic, VIEW will save time and effort in the production of all kinds of text.

Although for serious word processing a disc drive is essential, a cassette system is adequate for initial familiarisation with word processing techniques. Throughout the chapter, sections applying exclusively to either disc or cassette users will be marked  and  respectively. Similarly you will eventually need – or need access to – a printer but this will not be assumed for the purposes of this chapter.

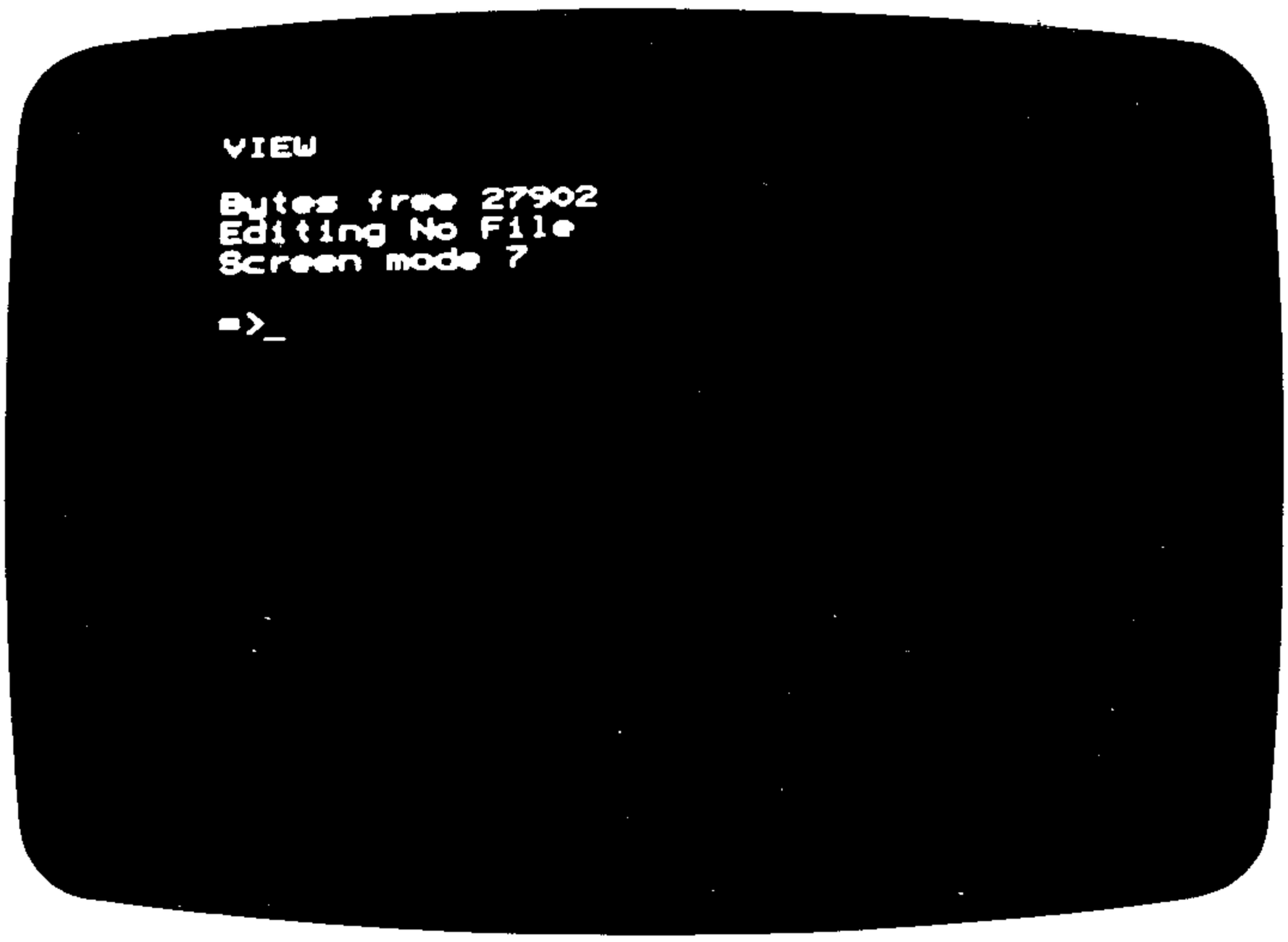
Using VIEW

Before starting to use VIEW, place the function key card supplied behind the clear plastic strip at the top of the keyboard. Ensure that FORMAT PARAGRAPH on the card is aligned with key [f0].

When you switch your computer on it will probably be ready to run programs in BASIC. In order to change from BASIC to VIEW, type:

*WORD RETURN

The screen will look like this.



If you are currently using screen mode 7, there will be only 40 character positions across the screen. Mode 131, with 80 character positions, is far more useful. We shall see later that a number of the available character positions on each line are reserved for special purposes. To select mode 131, type:

MODE131 RETURN

Throughout this chapter, it will be assumed that you are using screen mode 131. Remember that modes 128 - 135 are identical to modes 0 - 7 except for the memory that is available to hold your text. Note, however, that the VIEW command screen will always show one of modes 0 - 7.

You are looking at the VIEW **command screen**. This is the screen from which general commands such as SAVE will be issued. Note also that commands to the operating system (*commands) can be issued from the VIEW command screen. For example, you can speed up the cursor movement by typing:

*FX12,3 RETURN

To return to the standard cursor speed, type:

*FX12,0 RETURN

Another useful command is *CAT which displays a list of the files stored on a cassette or disc. More information on operating system commands can be found in Appendix C.

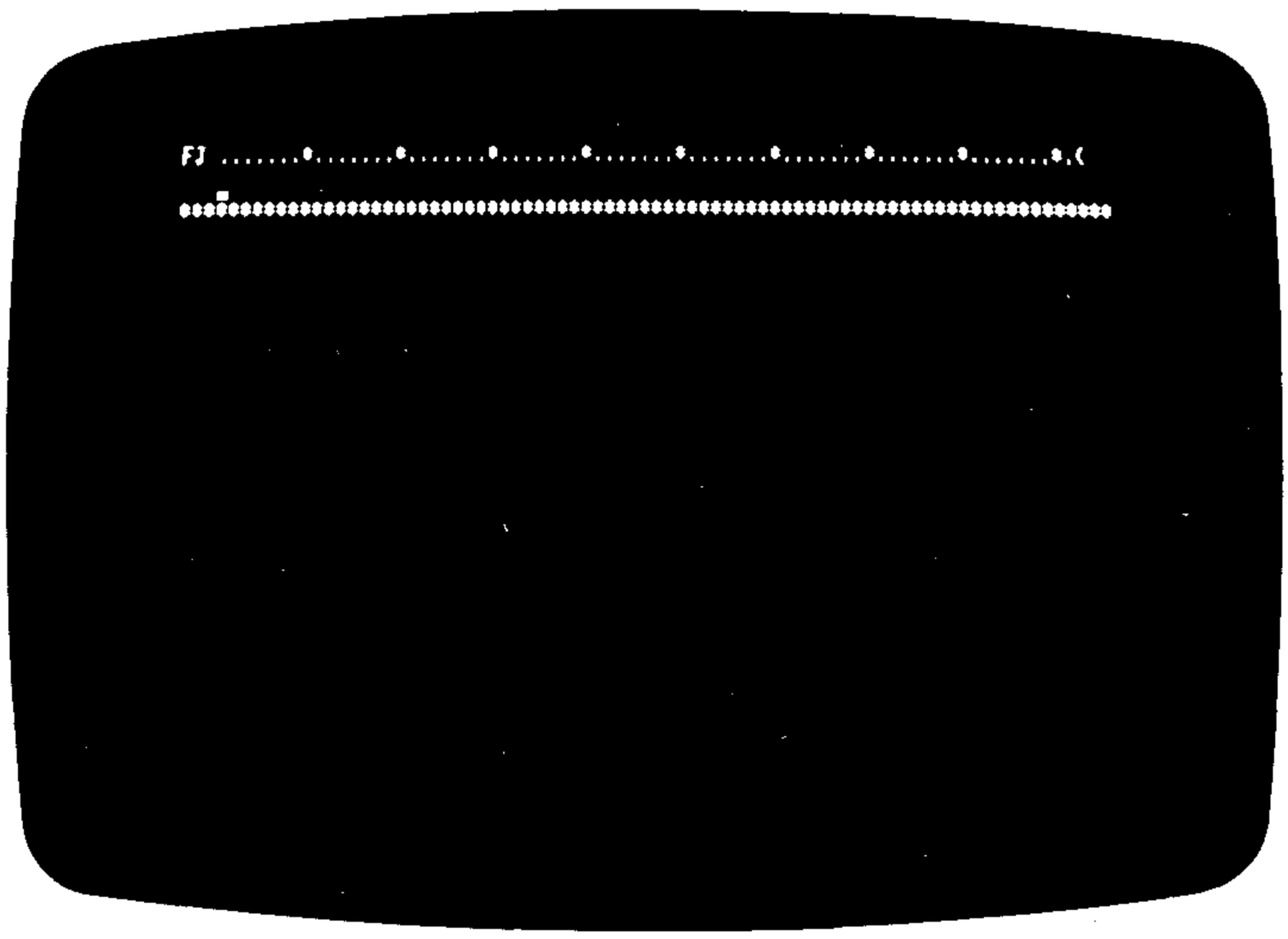
We will return to the command screen later in the chapter, but for the moment type:

NEW [RETURN]

then press [ESCAPE] and you will switch to the **text screen**. [ESCAPE] always switches, or **toggles**, between command and text screens. Note that any text that you have typed in will not be affected by pressing [ESCAPE].

Entering text

The text screen looks like this.



The flashing white symbol is the cursor – any text typed in will appear at the current cursor position. Type a few lines of text without pressing [RETURN]. Notice what happens as you reach the end of each line. If a word will not fit on the current line, it is automatically carried over to the next line. VIEW takes care of new lines for you ensuring that no lines are too long and that no words are split.

All the usual keyboard functions are operative in VIEW just as in BASIC so that if, for example, all your text appears in capitals, pressing [CAPS LOCK] will switch to lower case characters.

It will soon be obvious that VIEW is doing more to your text than just carrying

over words that will not fit on a line. VIEW always comes on with the **justification** feature on, – indicated by the J in the top left corner of the screen. This means that all text is vertically aligned at both the left and right hand ends of each line. In order to make all lines of text the same length, spaces are automatically inserted between some of the words.

If automatic justification is not required, it can be switched off by holding down [CTRL] and pressing [f3]. Try it now and you will see the J disappear from the top of the screen. If you now type a few lines of text, all word spacing will be identical but lines will be of varying lengths.

Switching justification on and off is just one of the facilities you will see labelled on the fuction key strip. VIEW has been designed in such a way that the most frequently used commands are obtained by pressing a function key. These functions are the ones you can see labelled along the bottom of the key strip. The row above consists of functions called by simultaneously pressing [SHIFT] and a function key, and the top row facilities require simultaneous depression of [CTRL].

These are called **immediate commands** because they can be issued directly from the text screen without switching to the command screen. Throughout this chapter, immediate commands will be referenced by the key number together with its function. For example:

[SHIFT]+[f7] (SET MARKER)

means 'hold down SHIFT and press function key f7'.

The line of dots and asterisks along the top of the screen is called the **ruler**. Amongst other things, it determines the maximum length of your lines of text. By adjusting the ruler, you can reduce or increase the number of characters per line for the text that follows it. Press [RETURN] a couple of times to leave some space, then press:

[CTRL]+[f5] (RULER)

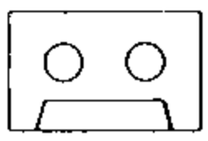
Another copy of the mode 131 standard ruler will appear. Use the arrow keys to take the cursor to the leftmost end of the ruler, then erase part of the ruler by pressing the SPACE BAR about ten times. Now enter a left margin stop >. Press [RETURN] to begin a new line and type in another two or three lines of text to observe the effect of shortening the ruler. The rightmost end of the ruler can be adusted in the same way but using the right margin stop <.

Editing a text file

The real power of a wordprocessor lies in the facility for editing and correcting text that has already been entered. To save time, a document called GRANT1, on which you can try out the editing facilities, is provided as part of your Welcome software.

Ensure that you are looking at the VIEW command screen and clear the VIEW workspace by typing:

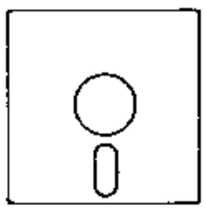
NEW RETURN



Load the Welcome cassette into the recorder so that it is ready to read side 2, reset the tape counter and wind through the tape to the end of PFILL. Remember that you may need to use *MOTOR 1 to enable you to control the recorder manually. Then type:

READ GRANT1 RETURN

The computer will search for file GRANT1 and append it to any text currently in the VIEW workspace. If necessary, stop your cassette recorder once loading is complete.



Insert the Welcome disc into your disc unit, then type:

LOAD GRANT1 RETURN

The LOAD command causes any text currently in memory to be overwritten by the new file. If you had wanted to append GRANT1 to to text currently in memory you could have typed:

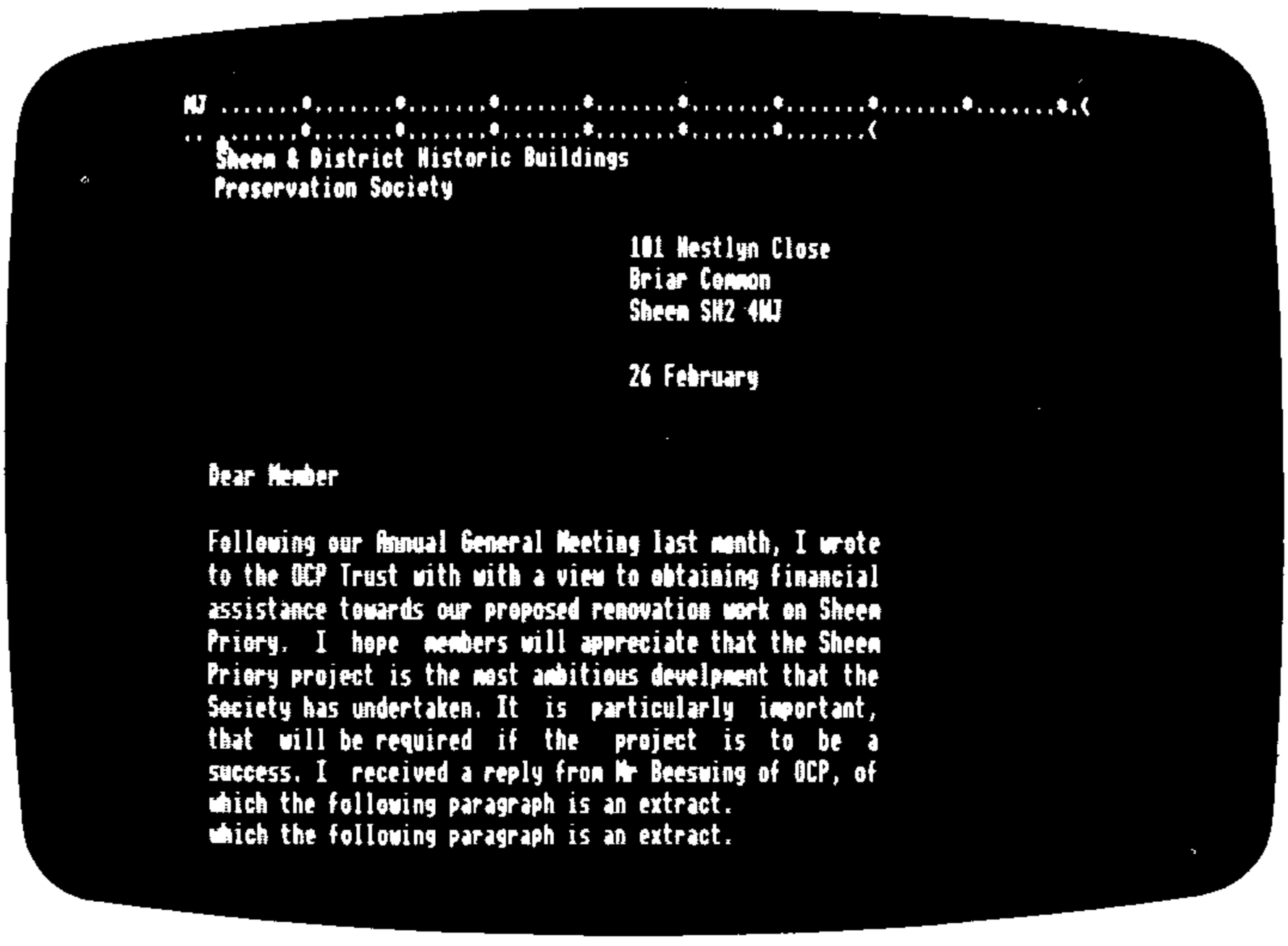
READ GRANT1 RETURN

as described above.

Note that the LOAD command cannot be used to load files from cassette.

Press ESCAPE to switch back to the text screen.

The screen will look like this:



You are looking at the first part of the document GRANT1. Hold down the downward arrow key and watch what happens as the cursor reaches the bottom of the screen. The VIEW text area is not limited to the screen itself. The text area is a very large 'page', only a little of which is visible to you through the screen. The screen is rather like a window which you can move (using the arrow keys) to any part of the page you wish.

Use the downward arrow key to scroll to the end of the document. You will find that the cursor will go no further than the last line of text. If you need to move further down, perhaps to begin another paragraph, you must press [RETURN] to add extra lines.

Clearly, moving through a long document using cursor keys alone can be somewhat tedious. Take the cursor back up the text by holding down [SHIFT] and pressing the up arrow key. The effect of [SHIFT] is to make the cursor jump in blocks of one screenful rather than one line at a time; useful for scanning quickly through a document. Another facility for speeding up movement around a document can be seen on function keys [f1] (TOP OF TEXT) and [f2] (BOTTOM OF TEXT). Their purpose is self evident.

GRANT1 contains several mistakes, each of which can be easily corrected using VIEW. Firstly, if the letter has been delayed, the date may have to be changed. Using a typewriter the alteration could be made with correcting fluid but the result is unlikely to be entirely satisfactory. In fact, when a letter contains more than one or two mistakes, the only realistic option is to retype the letter. VIEW enables you to correct such mistakes quickly and undetectably.

Take the cursor to the first character of the date and type:

```
5 March
```

Your new date will replace or **overtype** the one on the screen. A few characters from *26 February* will remain but these can be removed by moving the cursor to the space beyond *y* and using [DELETE].

As a result of changing the date, the reference in the first sentence to *last month* will have to change to *in January*. Do this now by overtyping.

The next mistake can be seen in line 2 where the word *with* has been inadvertently typed twice. It is possible to overtype one of the words with spaces, but unless the whole line were to be retyped this would leave a large gap between two words. Take the cursor to the *w* of the first *with* and press:

[f9] (DELETE CHARACTER)

You will see the character disappear and everything to the right of the cursor will move over to close the gap. The cursor should now be on the letter *i*. Press [f9] to delete this and twice more to delete the *t* and the *h*.

On line 5, a letter has been omitted from the word *development*. Place the cursor on the letter *p* and press:

[f8] (INSERT CHARACTER)

Everything to the right of, and including, the cursor position will move to the right to make an extra space. Now type *o* and the correction is complete.

The next mistake occurs towards the end of the paragraph where a line of text has been omitted after the word *important*. Place the cursor anywhere on the line below and press:

[f6] (INSERT LINE)

All lines of text below and including that line will move down to make room for a new line to be inserted. Take the cursor to the left hand end of the blank line and type:

```
therefore, that all members contribute to the effort
```

The final error in this paragraph can be seen at the bottom where a line has been typed twice. Put the cursor on the bottom line and press:

[f7] (DELETE LINE)

By this time, your paragraph will have lost its neatly formatted appearance. Insertions and deletions will have left some lines shorter than they should be, others will be too long. This situation can be easily remedied. Place the cursor on the top line of the paragraph and press:

[f0] (FORMAT PARAGRAPH)

The effect of this is to reposition all text from the line containing the cursor down to the end of the paragraph so that a neat format is maintained. Note that *justification* should be switched on, as shown by a J at the top of the screen, so that the paragraph will be formatted as justified text. If, after formatting, the paragraph is unjustified, press:

[CTRL]+[f3] (JUSTIFICATION)

and format the paragraph again.

Having corrected the first paragraph, you should be able to correct the errors in the remainder of GRANT1. Remember to format blocks of text as necessary, either following each correction or after editing a complete paragraph - the end result should be the same.

Your new version of GRANT1 will look something like this:

Sheem & District Historic Buildings
Preservation Society

101 Nestlyn Close
Briar Common
Sheem SH2 4WJ

5 March

Dear Member

Following our Annual General Meeting in January, I wrote to the OCP Trust with a view to obtaining financial assistance towards our proposed renovation work on Sheem Priory. I hope members will appreciate that the Sheem Priory project is the most ambitious development that the Society has undertaken. It is particularly important, therefore, that all members contribute to the effort that will be required if the project is to be a success. I received a reply from Mr Beeswing of OCP, of which the following paragraph is an extract.

"The OCP Trust does not normally contribute towards restoration work on buildings intended for business use. However, we are aware that if the priory were not restored, it could mean the loss of a building of great historic interest. Consequently, an application for assistance from the Trust would be favourably considered."

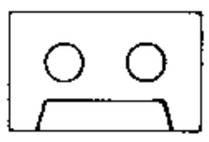
It would seem that Mr Beeswing is sympathetic to our cause and I suggest that we forward an application to the OCP Trust as soon as possible. I would be interested to hear suggestions from members as to what form such an application should take. Suggestions should be sent to me by the end of March, in time for me to present them to the executive meeting on April 6th. A prompt reply would be much appreciated in order that I might meet that deadline.

Yours sincerely

Martyn Gilbert (secretary)

After checking that all mistakes have been corrected, you will want to save your new document onto a cassette or a disc. Even if you intend further editing in the same session, it is a wise precaution to save your text at regular intervals. Then, if you should accidentally lose the document from memory (perhaps because of a power failure), only your most recent alterations will need to be done again.

To save your text file, first decide upon a filename. As the original document is called GRANT1 we would probably name the second version GRANT2.



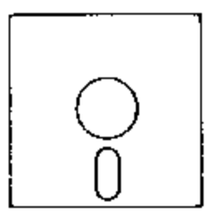
Remove the Welcome cassette from the recorder (without rewinding it) and replace it with a new cassette (or ensure that you are not likely to record over anything that you want to keep on an old cassette). Type:

```
SAVE GRANT2 RETURN
```

and the screen will show

```
RECORD then RETURN
```

Press RECORD on the cassette recorder, then press RETURN. Once the file has been saved, the computer will emit a bleep and the cursor will reappear. Switch off the cassette recorder.



Remove the Welcome disc from your disc drive and replace it with a disc onto which files can be saved. Type:

```
SAVE GRANT2 RETURN
```

Next time you come to work on your document, you will be able to load it by typing:

```
READ GRANT2 RETURN
```
(for cassette systems) or

```
LOAD GRANT2 RETURN
```
(for disc systems)

Block Operations

The editing facilities that you have used so far, with exception of f0 (REFORMAT PARAGRAPH), affect no more than one line of text at a time. However, facilities are available that operate on complete blocks of text. You can try these techniques by entering the song *Ten Green Bottles*.

Press ESCAPE to return to the command screen and clear the workspace by typing:

```
NEW RETURN
```

Press ESCAPE again to switch to the text screen and type in the first verse as shown on the next page. Note that in this case you will have to press RETURN at the end of each line because each is shorter than the standard ruler shown at the top of the screen.

```
Ten green bottles, hanging on the wall
Ten green bottles, hanging on the wall
And if one green bottle should accidentally fall
There'd be nine green bottles, hanging on the wall
```

You can quickly produce the entire song by using the COPY BLOCK facility. First indicate which block of text is to be copied. This is done by setting markers at the start and finish of the relevant block which, in this case, is the entire verse.

Position the cursor on the *T* at the beginning of the first line then press:

SHIFT + f7 (SET MARKER)

The characters MK appear at the top left of the screen. Then press:

1 to indicate that you are setting the position of marker 1.

A white block will appear indicating its position.

Now move the cursor to the line below the end of the verse and press:

SHIFT + f7 (SET MARKER)

Press:

2 to indicate that you are setting the position of marker 2.

Another white block appears, indicating the position of the second marker.

If you press ESCAPE and examine the command screen header, you will see that confirmation of the fact that you have positioned markers 1 and 2 is given by the additional line:

```
Marker(s) set 1,2
```

Press ESCAPE to return to the text screen and move the cursor to the point at which you want the copy to appear; in this case, immediately below your second marker. Finally, to execute the copy, simply press:

COPY

There should now be two identical verses in the document and the screen will look something like this.

Note that the two markers are still set, so by positioning the cursor and pressing [COPY] the verse can be reproduced as many times as needed. Complete the song by creating ten copies of the verse then editing each one as necessary.

Another useful operation enables blocks of text to be moved from one part of a document to another. Set markers to indicate one of the verses in your *Ten Green Bottles* document. Position the cursor elsewhere in the document and press:

[SHIFT]+[*f0*] (MOVE BLOCK)

The marked verse will be transferred to the cursor position. You will notice that, unlike the COPY operation, the markers are automatically cleared after MOVE BLOCK. The reason for preserving markers after copying is to facilitate a repeated copy as when compiling *Ten Green Bottles*.

The other block operation to be aware of is deletion, executed by setting markers and pressing:

[CTRL]+[*f0*] (DELETE BLOCK)

Any text below the deleted block moves up to close the gap.

Using CHANGE

Suppose you have typed in a document and you realise that a word has been consistently mis-spelt. You could search for each mistake and edit it

independently. In a long piece of text, however, you may have to make the same correction many times and one or two occurrences may be missed.

It is easier using the CHANGE facility, which can be illustrated with your *Ten Green Bottles* document.

From the command screen, type:

```
CHANGE/green/red/[RETURN]
```

VIEW responds with a message such as

50 string(s) changed

If you switch to the text screen and examine the document you will see that all occurrences of *green* have been changed to *red*.

You can also change *the* into *a* by typing:

```
CHANGE/the/a/[RETURN]
```

but the result may not be quite what you expect. The problem is that VIEW has, quite rightly, identified every occurrence of *the* whether it occurs alone or as part of **the**re, **the**y, li**the** or pa**the**tic. One way to overcome this problem is to apply CHANGE not to *the* alone, but to *the* together with spaces before and after. In other words, VIEW will search for / *the* / rather than /*the*/.

You can try this technique by changing *a* back to *the*, avoiding the creation of words like h**the**nging and f**the**ll. Switch back to the command screen and type:

```
CHANGE/ a / the /[RETURN]
```

Switch to the text screen and observe the effect.

You can apply CHANGE to phrases as well as single words. For example:

```
CHANGE/ insect / small invertebrate segmented animal /[RETURN]
```

The slash (/) in a CHANGE command is known as a **delimiter** because its function is to mark the beginning and end of a word or phrase. A space may be used instead of a slash provided no other spaces are required in the command. For example:

```
CHANGE kangaroo wallaby[RETURN]
```

As a diversion, readers may like to use CHANGE on single characters in order to decode the following passage. Despite its appearance, only five CHANGE operations are necessary!

> Kzch ykzj, zw whk hkighw of whk woujiqw qkzqon, ouj ciwy zcwq zq hoqw wo whouqzndq of viqiwojq fjom homk znd ovkjqkzq. Ykw in whk midqw of zll whiq zcwiviwy, whkjk zjk liwwlk ozqkq of chzjm znd pkzck.

The CHANGE operation is just one of a group of **global** operations that provide very powerful editing facilities. Treatment of more advanced techniques is outside the scope of this introduction and users are advised to consult the VIEW User Guide.

More on rulers

Clear any text that you have typed in by switching to the command screen and typing:

NEW [RETURN]

If you are not already in mode 131 as shown at the top of the command screen, type:

MODE 131 [RETURN]

Press [ESCAPE] to switch to the text screen.

As you saw earlier, the state of the text ruler determines the maximum line length for the text below it. The ruler at the top of the current screen is the standard ruler for mode 131 and it corresponds to a line length of 74 characters. Each mode has its own standard ruler and that for mode 135, for example, corresponds to a line length of 34 characters.

Put another standard ruler on the screen by pressing:

[CTRL]+[*f5*] (RULER)

It is good practice always to put in a ruler before starting to enter text. This ensures that your document is not mistakenly reformatted under a different ruler at a later date.

Now type in the text shown below. Remember that there is no need to press [RETURN] at the end of each line.

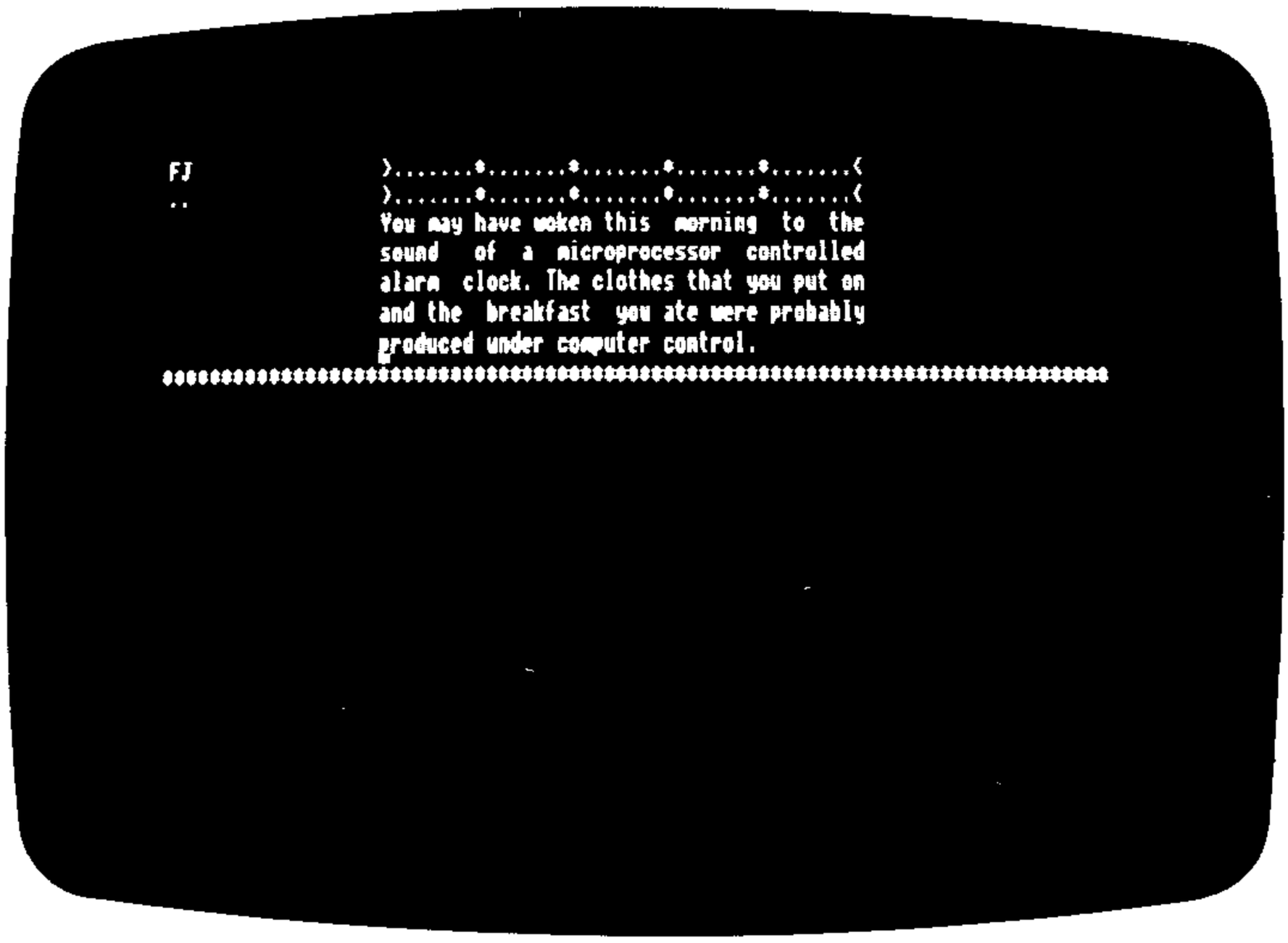
```
You may have woken this morning to the sound of a
microprocessor controlled alarm clock. The clothes that
you put on and the breakfast you ate were probably
produced under computer control.
```

The layout of the text can be altered by editing the current ruler. Take the cursor up to the ruler and change it to look like the one shown below. Remember, you can use any of the usual editing facilities such as overtyping and deleting characters.

```
>.......*.......*.......*.......*.......<
```

Now press [RETURN] to move the cursor from the newly edited ruler to the first line of the text. You will notice that the ruler at the top of the screen now matches the new current ruler. The top ruler always acts as a reminder as to which ruler is operative in the current cursor position. Press:

[*f0*] (FORMAT PARAGRAPH)



You may want the next paragraph to have a different ruler setting, in which case a new ruler must be added to the document. You can do this by pressing:

[CTRL]+[*f5*] (RULER)

to put a standard ruler in the required position, then editing the ruler as appropriate. Sometimes it may be more convenient to copy the current ruler and edit that - pressing [SHIFT]+[COPY] together will generate a copy of the current ruler at the current cursor position. Having created your new ruler, any text typed in below it will be subject to the new margin setting, as shown on the next page.

VIEW recognises a ruler by the two dots in the left margin. They are normally followed by a line of dots and asterisks bounded by margin stops > and <. The left margin stop is omitted on standard rulers. With two exceptions, the characters that appear between the margin stops > and < are irrelevant so it makes sense to adopt the convention of using a line of dots as this renders the ruler immediately recognisable to the user.

The asterisks in standard rulers are TAB stops. Their function can best be illustrated by putting the cursor on a blank line then pressing the [TAB] key two or three times. The cursor jumps from one TAB position to the next. This facility is particularly useful in constructing tables. Having used [TAB] to move the cursor across the screen, the effect of pressing [DELETE] may surprise you. Instead of moving by one character position at a time, the cursor jumps back from each TAB stop to the next. This effect is less surprising when you realise that TAB is, in fact, an invisible *character*. Cursor movement, therefore, by means of arrow keys or the [DELETE] key, is still taking place from character to character.

The other special character that may be used in a ruler is *b* for *bleep*. This corresponds to the bell that signals an approaching end-of-line on a typewriter. A bleep will sound whenever you type past a position at which a *b* has been inserted in the ruler.

Back to GRANT2

Load the document GRANT2.

It is often necessary to carry out wholesale changes to the way in which a document is structured and formatted. Consider the suggested changes that have been marked up on GRANT2 as shown below.

centre heading

Sheem & District Historic Buildings
Preservation Society

101 Nestlyn Close
Briar Common
Sheem SH2 4WJ

5 March

right justify

Dear Member

Following our Annual General Meeting in January, I wrote to the OCP Trust with a view to obtaining financial assistance towards our proposed renovation work on Sheem Priory. I hope members will appreciate that the Sheem Priory project is the most ambitious development that the Society has undertaken. It is particularly important, therefore, that all members contribute to the effort that will be required if the project is to be a success. I received a reply from Mr Beeswing of OCP, of which the following paragraph is an extract.

move text

"The OCP Trust does not normally contribute towards restoration work on buildings intended for business use. However, we are aware that if the priory were not restored, it could mean the loss of a building of great historic interest. Consequently, an application for assistance from the Trust would be favourably considered."

re-format to new text width

It would seem that Mr Beeswing is sympathetic to our cause and I suggest that we forward an application to the OCP Trust as soon as possible. I would be interested to hear suggestions from members as to what form such an application should take. Suggestions should be sent to me by the end of March, in time for me to present them to the executive meeting on April 6th. ~~A prompt reply would be much appreciated in order that I might meet that deadline.~~

delete this sentence

Yours sincerely

Martyn Gilbert (secretary)

Note - it is OPC not OCP

Firstly, the name of the society should be centred on the page. You could do this by inserting spaces but you would have to count characters or judge the right position for both lines. Also, if the width of the text were to be adjusted at a later date the positioning would no longer be accurate.

It is far easier to use one of the **stored commands** available in VIEW. These commands are entered in the **stored command margin** to the left of the text area. They have no immediate effect but are simply stored until the document is printed, whereupon they come into operation. On this occasion you need the stored command CE, which stands for *CEntre*. Its function is to centre text according to the current ruler, so if a new ruler is inserted the relative position of the text will still be correct.

Switch to the text screen and take the cursor to the first of the two lines to be centred. Press:

[SHIFT]+[f8] (EDIT COMMAND)

The cursor moves into the left margin. Now type:

CE[RETURN]

The command CE remains in the margin and the cursor moves back into the text area.

Take the cursor to the second line to be centred and repeat the operation.

The stored command CE has no immediate effect, but there is a way to preview the document as it would appear if it were printed. Switch to the command screen and type:

SCREEN[RETURN]

You should see the first part the document with no ruler and with the name of the society centred. In order to preview the next screenful, press and release [SHIFT]. Once the complete document has been SCREENed, press [ESCAPE] to return to the text screen.

The SCREEN command is a convenient way of checking on the appearance of text before it is printed. The effects of rulers and of stored commands can be previewed before committing anything to paper.

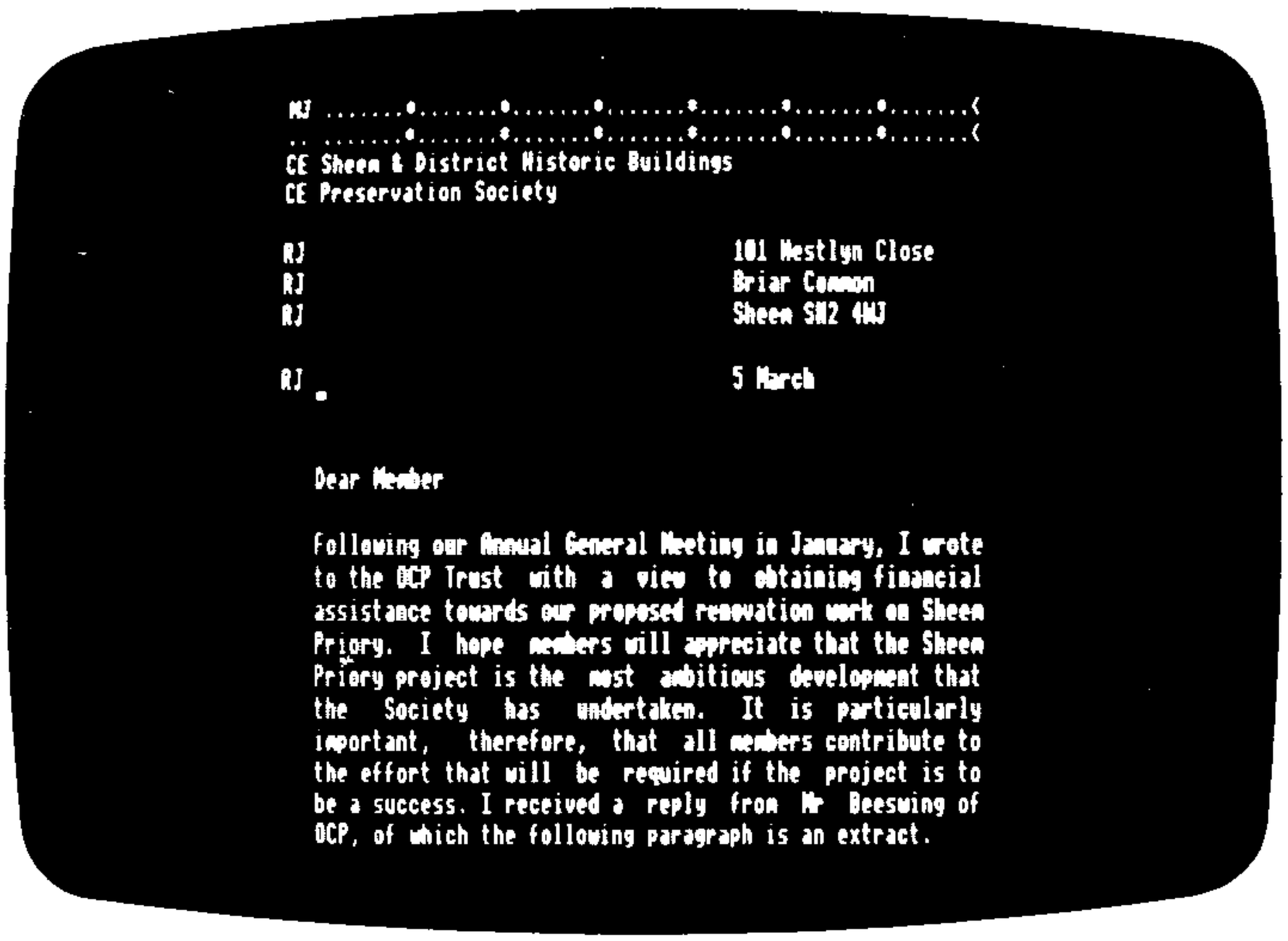
You can use another stored command to position the address at the top of the letter. Take the cursor to the first line of the address and press:

[SHIFT]+[f8] (EDIT COMMAND)

Now type:

RJ[RETURN]

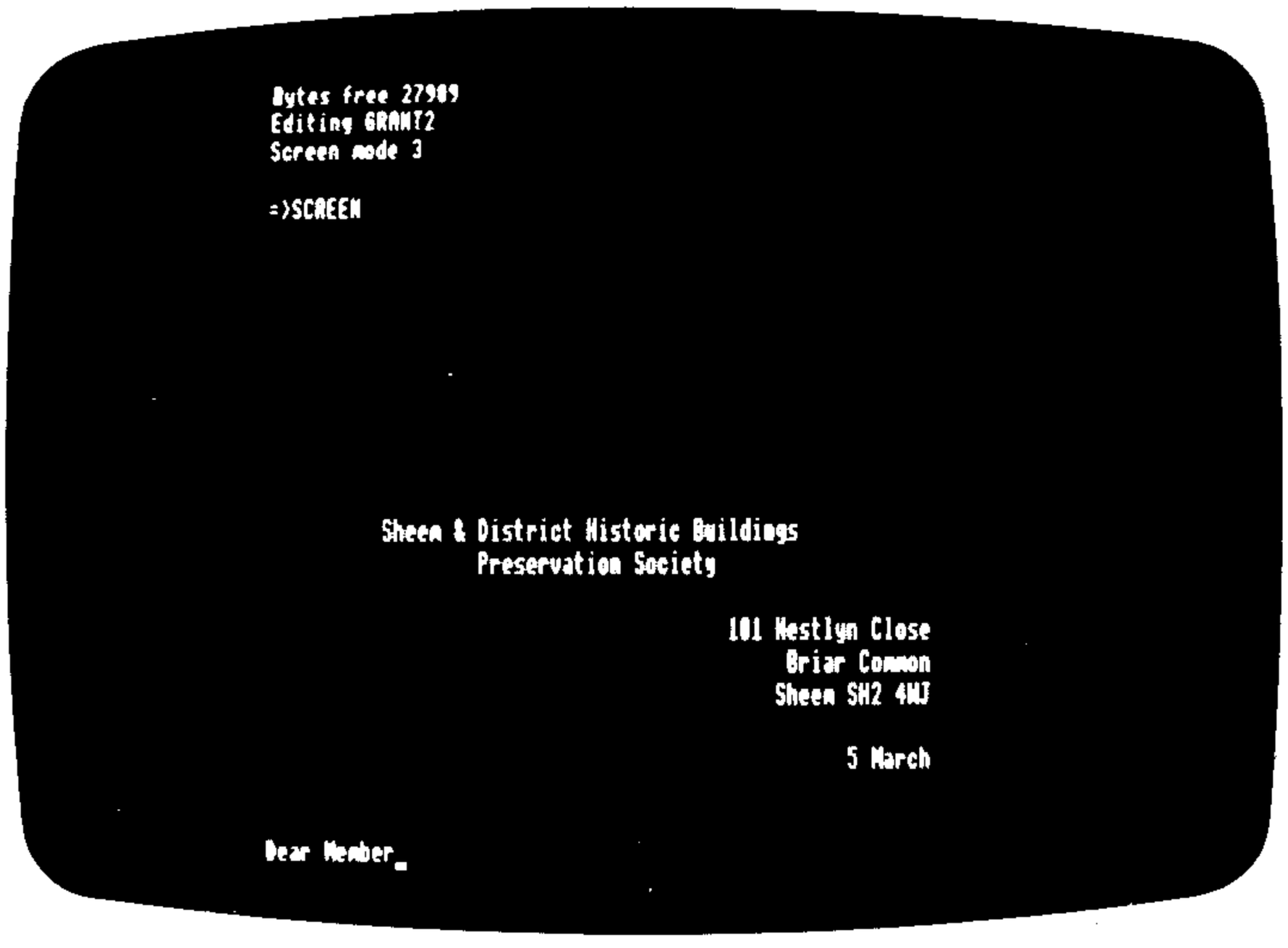
Enter the same stored command into the left margins of the other lines of the address and the date, as shown below.



RJ stands for **Right Justify**. Again, its effect is not immediate, but if you switch to the command screen and type:

SCREEN RETURN

the result should be something like this.



Press [SHIFT] until the screening is complete, then [ESCAPE] to return to the text screen.

You will come across more stored commands later in this chapter. For the moment, we will move on to the other changes to be carried out to GRANT2.

A block of text has to be moved from the first paragraph to the end of the letter. To do this, set markers 1 and 2 to indicate the beginning and end of the block. Unlike the example you saw in the last section, this block lies in the middle of a paragraph so you cannot set markers on blank lines. Simply set marker 1 on the *I* of line 4 and marker 2 in the space immediately following the last character of the block. This is before the *I* on line 9.

Take the cursor to the point at which you want the block to appear; in this case, two lines down from the end of the final paragraph. Press:

[SHIFT]+[*f0*] (MOVE BLOCK)

If, as in this instance, moving a block has destroyed the format of the text, simply reformat the affected paragraphs by positioning the cursor on the first line and pressing:

[*f0*] (FORMAT PARAGRAPH)

The next alteration to GRANT2 involves paragraph two, which has to be reformatted to a narrower text width. This involves inserting and editing a new ruler above the relevant paragraph. As you have seen, however, each ruler affects all the text below it until the occurrence of the next ruler. As we only want one paragraph to be reformatted, it will be necessary to insert a new ruler below the paragraph as well as above it.

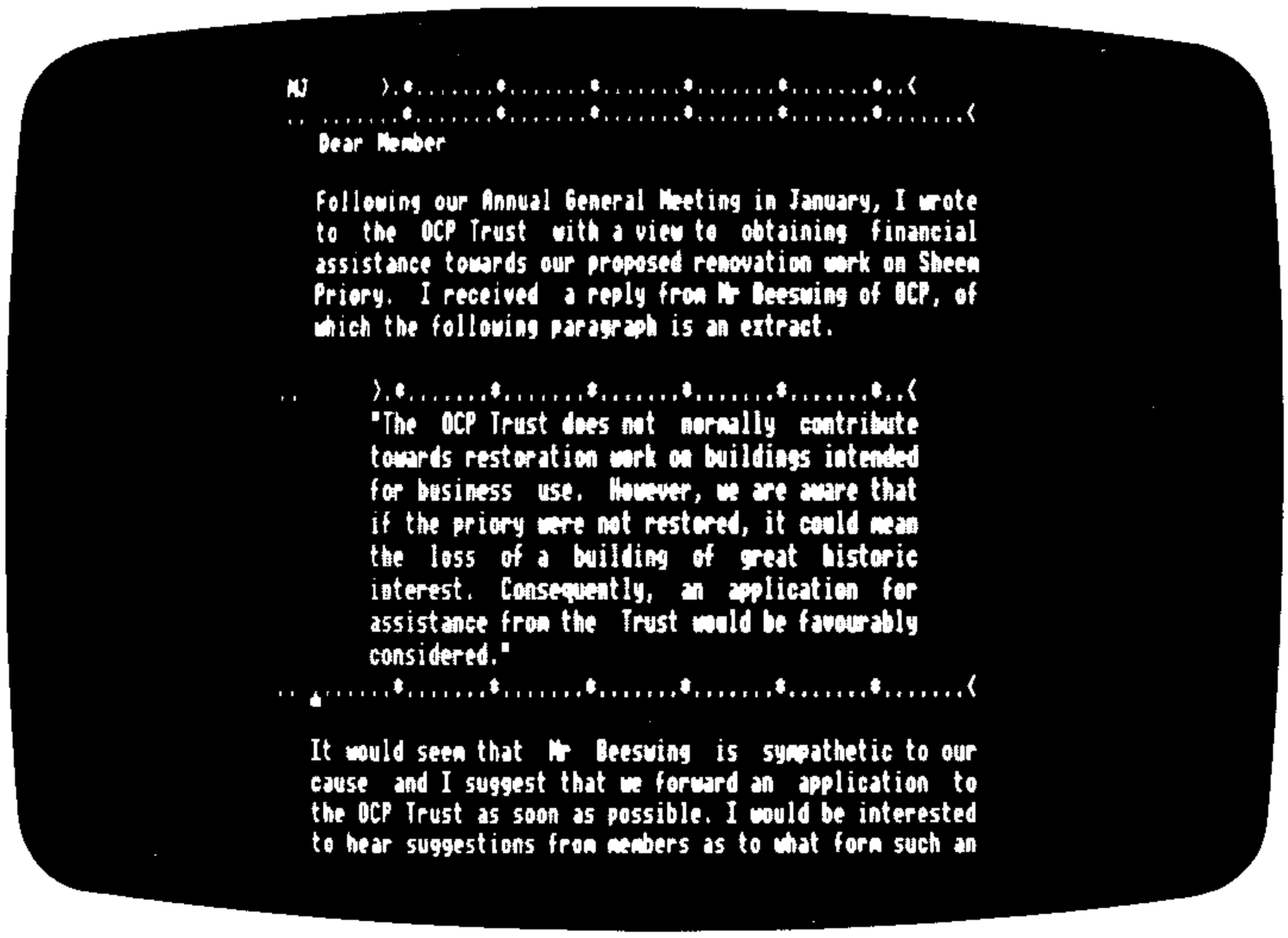
Position the cursor above the first line of the paragraph and press:

[SHIFT]+[COPY]

to obtain a copy of the current ruler. Now put another copy of the current ruler below the last line of the paragraph. The first of these two rulers can now be edited to obtain the required format.

Note that no account has been taken of the tab positions (*) as tabs are not be used in this particular document. For documents in which tabulation is required the positions of the asterisks must be adjusted accordingly.

Now all that remains is to position the cursor on the first line of the paragraph and press [*f0*] to reformat.



Another alteration to GRANT2 involves deleting the final sentence of what is now paragraph three. Position the cursor on the letter *A* at the beginning of the sentence and press:

[*f3*] (DELETE END OF LINE)

The following two lines of text can be deleted by positioning the cursor and pressing:

[*f7*] (DELETE LINE)

Finally, it appears that OCP should have been typed as OPC. Carry this out using CHANGE.

Having completed your second edit, the document will be ready to save again onto a cassette or a disc. If you choose to keep the same name that appears at the top of the command screen, then all that is necessary is to type:

SAVE [RETURN]

VIEW will assume that you require the filename currently shown.

Note that the filename on the command screen can be changed at any time. For example, to change the filename to FRED type:

NAME FRED [RETURN]

It is not always wise to save an edited document under its former name. If you are using a disc system, the new file will overwrite any other file with the same name. In many cases, this is perfectly acceptable but you may decide you want to keep older versions, perhaps as a security measure. If this is the case, then it makes sense to use a numbering system such as GRANT1, GRANT2, GRANT3 and so on. Never include spaces in your filenames. Everything following a space will be ingored so that GRANT 1, GRANT 2 and GRANT 3 will be treated as the same name.

Printing from VIEW

When instructed to print, VIEW sends codes to a **printer driver** program which controls the operation of the printer. The driver program contained in VIEW is perfectly adequate for straightforward printing and will operate most types of printer although it is possible to use more sophisticated drivers. We will come to these later but for the moment it is assumed that you are using the default printer driver contained in VIEW.

There are two ways to print. If you want to print a copy of the file that is currently in the computer's memory, all that is necessary is to switch to the command screen and type:

PRINT [RETURN]

Alternatively, you can print from a file held on disc or cassette without affecting the text currently in memory. To print a file called GRANT3, for example, type:

PRINT GRANT3[RETURN]

Try printing a few pieces of text to get used to the method. Remember, if you want to see a simulated print-out on the screen before committing anything to paper you can SCREEN a file as described on page 121.

If you print a document that carries over to more than one page, VIEW assumes a page length of 66 lines of text. If this is inappropriate, it can be changed by entering a *stored command* at the beginning of the document. Press:

[SHIFT]+[f8] (EDIT COMMAND)

The cursor moves into the left margin. Now type:

PL[RETURN]

The command PL remains in the margin and the cursor moves back into the text area. Now type:

50[RETURN]

This number specifies the number of lines in the new page length.

Subsequent PRINT or SCREEN operations on that file will now result in a page length of 50 lines.

Sometimes page breaks occur at inconvenient points such as in the body of a table or immediately following a heading. This may be remedied using the stored command PE – which stands for Page Eject. The command is placed in the margin wherever the printer is required to move on to a new page. It overrides the automatic page breaks specified in the PL stored command.

On page121you were introduced to the *stored commands* CE and RJ. Now you can add PL and PE to the list. These commands serve to illustrate the way in which stored commands operate in VIEW and they are sufficient for most general printing tasks. However, there are many more stored commands available in VIEW and users are referred to Appendix I and the VIEW User Guide.

Printer drivers

A wide range of printers is available, some offering sophisticated features such as bold type, underlining and sub-scripts. You may have noticed the labels HIGHLIGHT 1 and HIGHLIGHT 2 on the function key strip. They are used to insert codes that identify words to be printed in a special way, normally underlined or printed in bold type. In order to utilise these special features each kind of printer uses a different set of codes and requires a tailor-made printer driver.

The *Printer Driver Generator* is a program for producing printer drivers to meet the needs of particular printers. Once generated, a driver can be saved on disc or cassette to be called up whenever required. The generator is available from your dealer.

Additional features of VIEW

This concludes the introduction to word processing using VIEW. By now you should have an understanding of what word processing is all about and a working knowledge of VIEW. As you will probably appreciate, we have been able to describe only the basic features of the word processor. Given below are outlines of some of the more advanced features of VIEW, operational details of which can be found in the VIEW User Guide.

Macros

A **macro** is a piece of text that can be printed by entering a unique name in the margin, rather like a stored command. The macro may be printed as often as required merely by repeating the name in the margin. A macro can be *modified* so that each time it is called, different words or phrases are inserted at particular positions.

A common use of macros is in the printing of personalised standard letters as mentioned in the introduction to this chapter. The letter can be printed as many times as needed and each copy will include a different name and address.

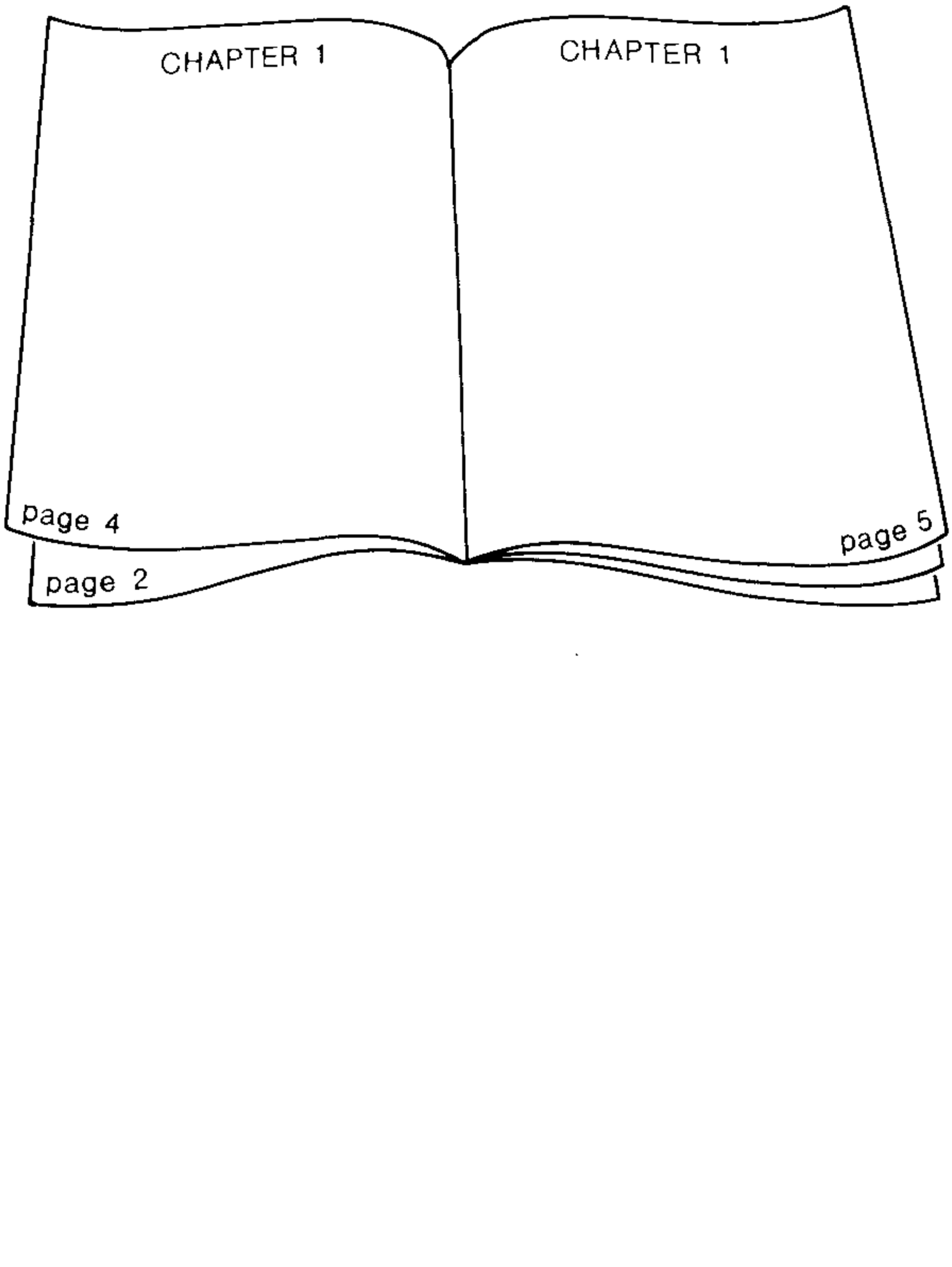
Global editing

You have used the CHANGE command to alter words throughout a document. Similar operations include SEARCH, which will locate items of text and REPLACE which performs the same job as CHANGE but gives you the option to accept or reject each alteration.

Special symbols can be used which enable all of these facilities to operate on invisible characters such as tabs and carriage returns. In addition there is a **wildcard** symbol which can be used, for example, to search for all occurrences of a word even though some occurrences may be mis-spelt.

Further printing facilities

Additional printing facilities include the use of **headers** and **footers**. They can be used to print a document with, for example, automatic page numbering and a running title. The illustration below shows one effect that can be achieved by differentiating between odd and even numbered pages.

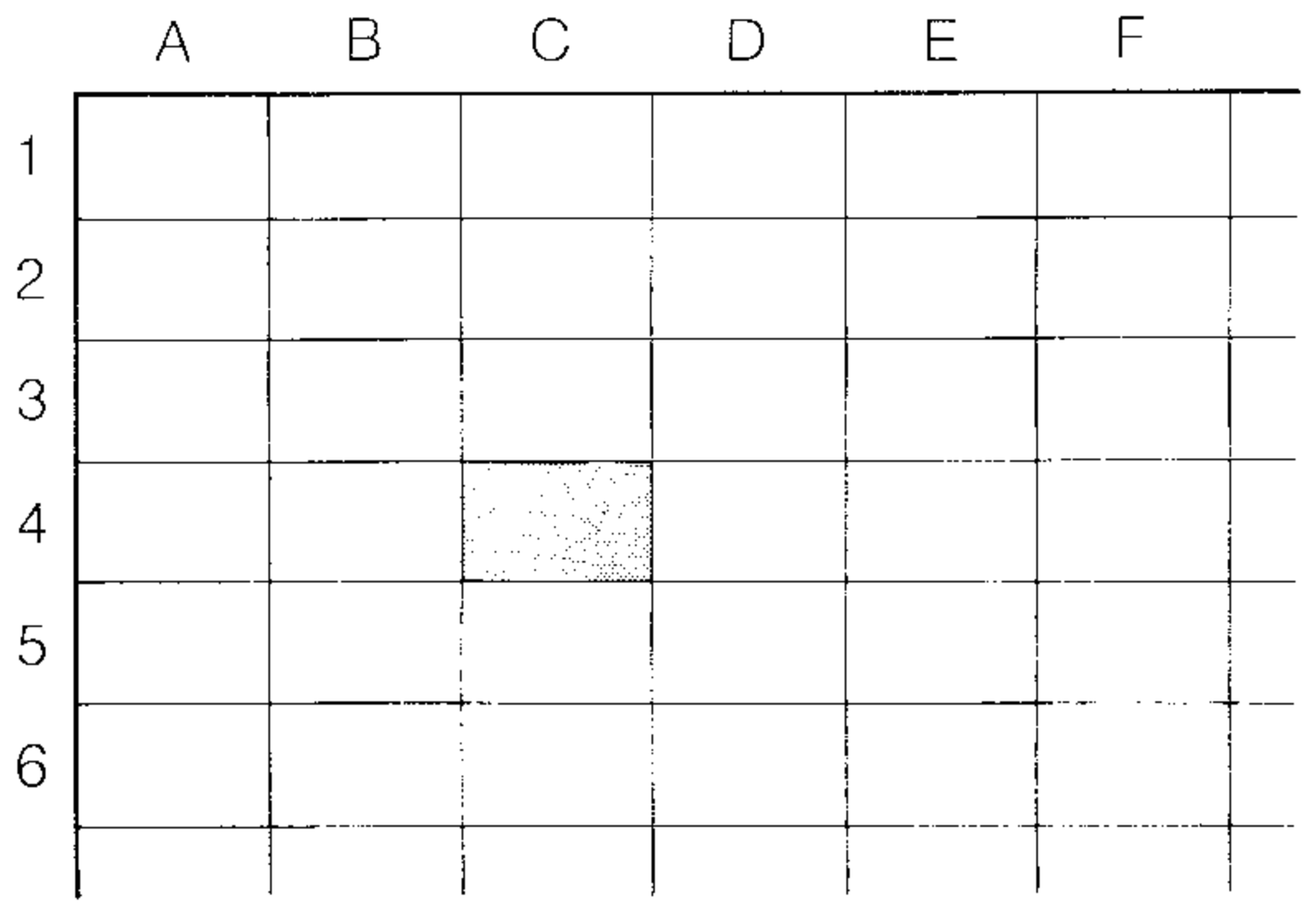


4. Introducing ViewSheet

What is a Spreadsheet?

Imagine a huge sheet of paper marked out in rectangles by a series of horizontal and vertical lines. Each of the boxes so formed may be individually identified by reference to the appropriate *column* and *row*. The columns are labelled A, B, C, and so on whereas the rows are numbered from 1 to 255. Thus an individual box, or **slot**, might be called A2 or B4 or F99.

In the diagram below, slot C4 has been shaded:



A spreadsheet program simulates such a sheet of paper and allows you to enter, into any slot, one of three things:

- A **label**, that is a piece of text
- A **number**, such as 7 or 1234.56
- A **formula** containing references to other slots

It is the facility for entering formulae that makes the spreadsheet such a powerful tool. Once a spreadsheet has been set up, any new values entered can be automatically be related to the items already recorded.

For example, you might enter the number 27 into slot B3 and 19 into B4. If you were to put the formula B3-B4 into the slot B5, the result would be as shown opposite.

	A	B	C	D	E	F
1						
2						
3		27				
4		19				
5		8				
6						

Labels can be added to make your spreadsheet more meaningful:

	A	B	C	D	E	F
1						
2						
3	SALES	27				
4	COSTS	19				
5	PROFITS	8				
6						

The contents of any slot may be changed and the effects of such changes can be observed over the rest of the sheet. In our simple example, changing the COSTS in B4 will automatically adjust the PROFITS in B5.

This is obviously a trivial example, but it points to the value of spreadsheets in investigating 'what if..?' type questions such as:

"If I borrow £1000 over three years and the interest rate is held at 13.5% my payments will be £39.03 but if I increase my payment to £45 I can....."

Calculations that would be very repetitive if carried out by conventional means are achieved with no greater effort than entering the data on which they are based. Thus spreadsheets are widely used in industry and commerce for financial modelling and forecasting. At home and in small businesses, they are used for budgeting and accounts.

ViewSheet

Your BBC Microcomputer has a powerful built-in spreadsheet program, **ViewSheet**. It allows you to set up spreadsheet displays, vary them at will,

save them onto disc or cassette, retrieve them, and print them in whole or in part.

Furthermore, ViewSheet is compatible with VIEW. In other words, spreadsheets can be incorporated into word processed text, edited as required, and printed as one file. If you have read the Word Processing chapter you will notice that there are many similarities between the operational details of VIEW and ViewSheet. Both are members of the *View Family* from Acornsoft.

Using ViewSheet

Before starting to use ViewSheet, place the function key card under the clear plastic strip at the top of the keyboard. Ensure that DELETE CHARACTER is aligned with key f9.

When you switch your computer on, it will be ready to receive BASIC programs. In order to change from BASIC to ViewSheet, type:

```
*SHEET RETURN
```

The screen will look like this.

In screen mode 7, there are only 40 character positions across the screen. As in VIEW, mode 131 with its 80 character positions is far more useful. To select mode 131, type:

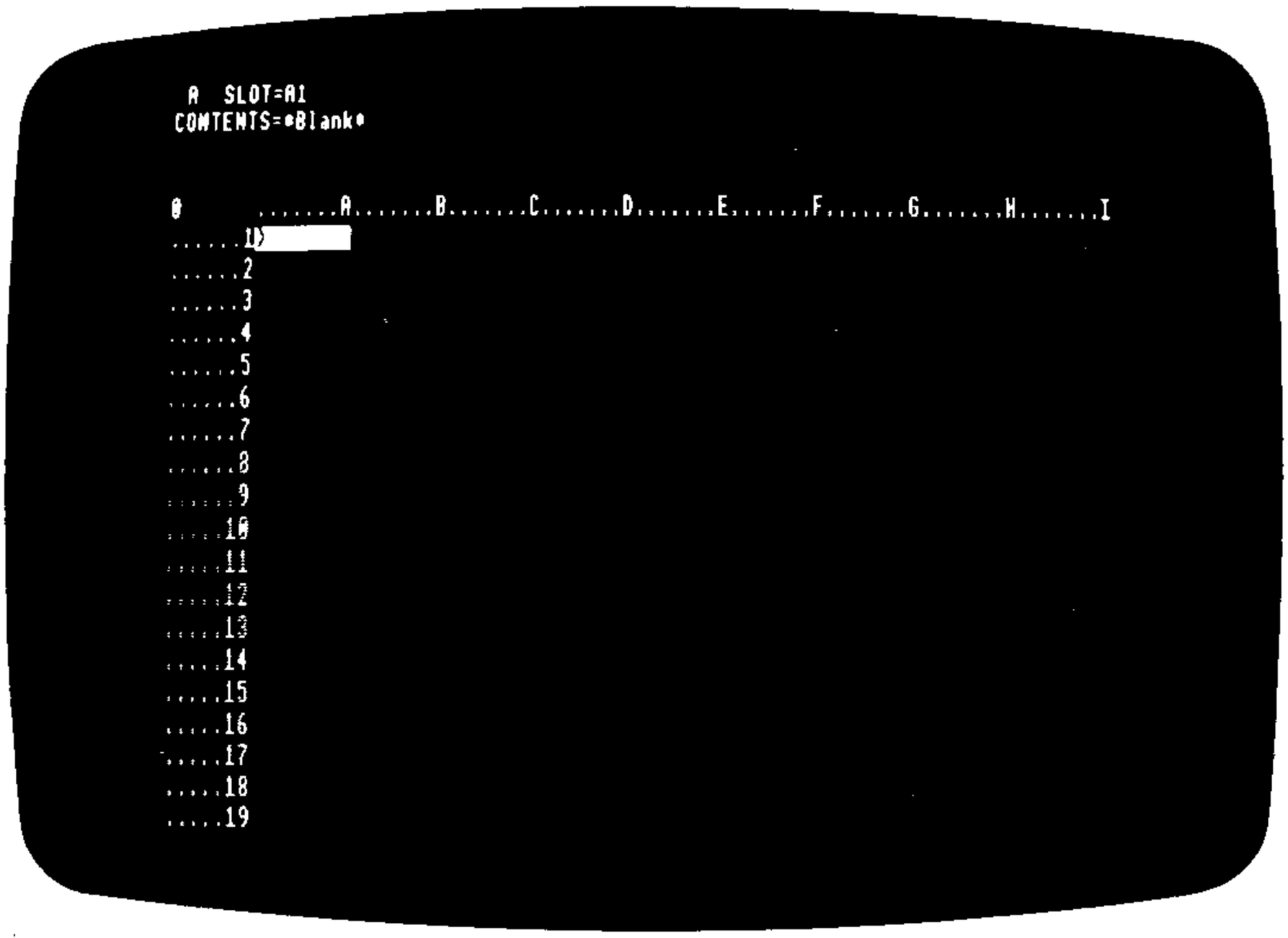
```
MODE 131 RETURN
```

Throughout this chapter, it will be assumed that you are using screen mode 131 although the command screen will always show the mode number in the range 0 – 7.

ViewSheet is now displaying the **command screen**. Press [ESCAPE] and you will switch to the **sheet screen**. Pressing [ESCAPE] always switches, or **toggles**, between the command screen and the sheet screen. Note that the contents of your spreadsheet are not affected by pressing [ESCAPE].

Moving around the spreadsheet

Now the screen looks like this.



What you are looking at is the top left-hand corner of the spreadsheet. Along the top you will see the letters A to I and down the side the numbers 1 to 19. Thus, as mentioned earlier, any slot may be uniquely referenced by a letter/number pair such as A1 or E15.

The white rectangle in slot A1 is the cursor. One of its functions is to provide a means of moving about within the spreadsheet so that you are not permanently looking at the top left-hand corner. Press the downward arrow key several times to take the cursor down the screen and notice what happens when it reaches the bottom. In effect, you are moving the whole screen down the sheet one step at a time. If you were to carry on long enough, you would reach the bottom of the sheet as shown by row 255.

You can move across the sheet in the same way, using the right and left arrow keys to move the cursor. The columns are labelled A to Z followed by AA to AZ and so on, through to the 255th column which is labelled IU.

In order to move around the spreadsheet more quickly, you can use the *auto-repeat* facility. Press one of the arrow keys and hold it down for a few

seconds. The cursor will jump quickly from slot to slot as though you were pressing the key repeatedly.

Another way to speed up cursor movement is to hold down [SHIFT] as you press the arrow key. Instead of moving at a rate of one slot at a time, the cursor now jumps in blocks equivalent to one screenful. Use this method now to take the cursor back to slot A1.

When in the sheet screen the current cursor position is always shown at the top of the screen. It should now read:

```
SLOT=A1
```

but it will change as you move the cursor around. Also shown are the contents of the slot. At the moment this will appear as:

```
CONTENTS=*BLANK*
```

Entering information

Make sure the cursor is on slot A1 then type:

RENT [RETURN]

You will see the word RENT appear in the current cursor position. Now move the cursor down to A2 and type:

RATES (do not press [RETURN])

The word appears near the top of the screen but not, as yet, in the spreadsheet. The word RATES is on the editing line. It will stay there until you press [RETURN] and in the meantime you can edit it or delete it (using the [DELETE] key) without affecting the contents of the sheet. The editing line is particularly useful when making a complex entry that you want to check before committing it to the sheet. We will look at editing facilities later in the chapter.

Press:

[RETURN]

to transfer RATES to the sheet, then enter FUEL into slot A3 and TOTAL into slot A5. A4 will be left blank to aid clarity of presentation.

```
RENT
RATES
FUEL

TOTAL
```

So far we have been entering words or **labels** into the sheet. Naturally they cannot be used in calculations but labels are necessary as headings or explanations. The letter L at the top of the screen shows that the slot currently indicated by the cursor contains a label.

Now take the cursor to slot B1 then type:

126 RETURN

You have now entered a **value** into B1. A value may take the form of a number or or a formula – anything that can be used in calculation. Whilst the cursor is on B1, you should see the letter V at the top of the screen. This indicates that ViewSheet recognises the contents of B1 as a value rather than a label.

Now enter, say, 37 and 66 into B2 and B3 respectively. Your spreadsheet now contains:

```
RENT        126
RATES       37
FUEL        66

TOTAL
```

Take the cursor to B5 and enter:

B1+B2+B3 RETURN

The number 229 should appear as the total in slot B5. Note, however, that the CONTENTS= line at the top of the screen still shows the formula B1+B2+B3. We can see, then, that there are two pieces of information associated with a slot containing a formula. Firstly there is the formula itself – in this case B1+B2+B3; secondly there is the number obtained by evaluating the formula – in this case 229. When we look at a spreadsheet, all values appear as numbers whether entered directly or calculated from a fomula. However, the CONTENTS= line always shows how the number in a particular slot was originated.

If at this point slot B5 does not hold the total, the chances are that you have mis-typed the formula. Each of the variables used in a formula must be a slot reference such as B1 or AC123. If ViewSheet cannot recognise it as a slot reference it will assume the entry to be a label and will reproduce it in the slot.

Below are some examples of formulae that could be used in ViewSheet. Note that multiplication and division are carried out using * and /.

```
(4*A1)/B7 5*(C3-B3) (D1+E1+F1)/(H8*H9)
```

As mentioned earlier, the real power of ViewSheet lies in the ability to investigate the effects of changing one or more of the values held. Simply overwriting the contents of the slot is often the most convenient method of making such changes. Take the cursor to B1 and type:

131 RETURN

The number 131 will overwrite the existing contents of slot B1.

Furthermore, the contents of B5 will automatically adjust to show the new total.

At this point, you may wish to experiment by entering values and using them in formulae. Clearly, the example we have looked at is particularly simple. In the next section, we will examine a more realistic spreadsheet that has already been created and discover some of the ways in which it might be used.

Using a spreadsheet

An example spreadsheet is provided with the Welcome software.

((c)) If you have used the VIEW file GRANT1 your cassette should already be wound to the appropriate point.

Press [ESCAPE] to switch to the command screen then type:

LOAD ACCOUNT [RETURN]

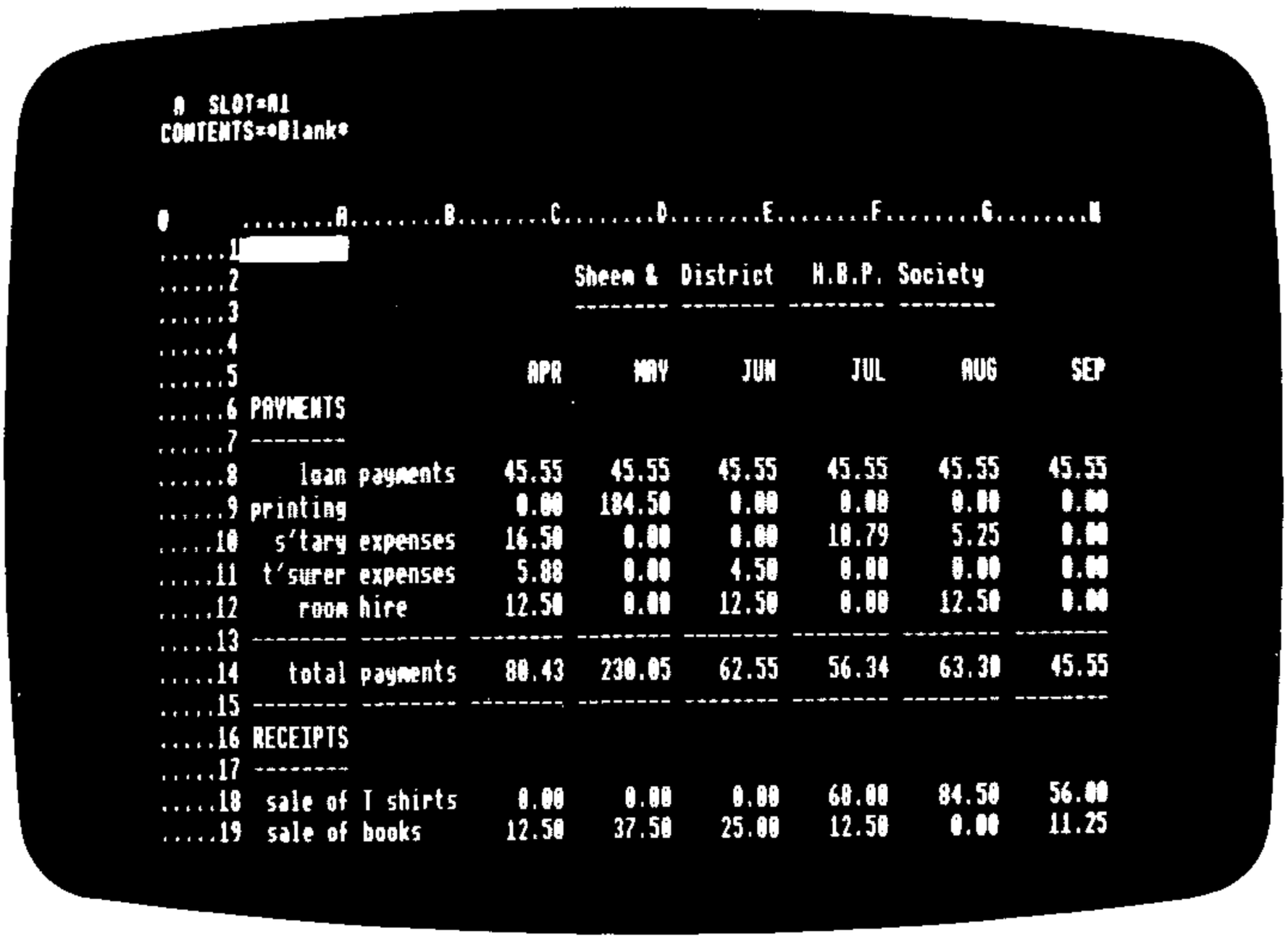
The computer will search for the file called ACCOUNT and will load it into the ViewSheet workspace. Once loading is complete stop the cassette recorder.

(d) Press [ESCAPE] to switch to the command screen then type:

LOAD ACCOUNT [RETURN]

The computer will load a file called ACCOUNT into the ViewSheet workspace.

Press [ESCAPE] to switch back to the sheet screen. The screen will look like this:



ACCOUNT shows the Sheem Preservation Group's annual accounts for the year 1985/86. The full sheet is shown opposite.

Sheem & District H.B.P. Society Accounts for: '1985-86

	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	JAN	FEB	MAR	TOTALS
PAYMENTS													
loan payments	45.55	45.55	45.55	45.55	45.55	45.55	45.55	45.55	45.55	45.55	45.55	45.55	546.60
printing	0.00	184.50	0.00	0.00	0.00	0.00	0.00	99.00	0.00	0.00	0.00	0.00	283.50
s'tary expenses	16.50	0.00	0.00	10.79	5.25	0.00	0.00	8.30	6.80	0.00	0.00	0.00	47.64
t'surer expenses	5.88	0.00	4.50	0.00	0.00	0.00	0.00	0.00	0.00	22.00	3.60	0.00	35.98
room hire	12.50	0.00	12.50	0.00	12.50	0.00	12.50	0.00	12.50	0.00	15.00	0.00	77.50
total payments	80.43	230.05	62.55	56.34	63.30	45.55	58.05	152.85	64.85	67.55	64.15	45.55	991.22
RECEIPTS													
sale of T shirts	0.00	0.00	0.00	68.00	84.50	56.00	40.50	26.00	58.00	0.00	17.50	22.50	373.00
sale of books	12.50	37.50	25.00	12.50	0.00	11.25	64.00	80.00	125.00	16.00	0.00	37.50	421.25
Wynne's donation	0.00	650.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	650.00
members fees	135.00	320.00	200.00	57.00	88.00	275.00	300.00	48.00	25.00	35.00	108.00	321.00	1912.00
total receipts	147.50	1007.50	225.00	137.50	172.50	342.25	404.50	154.00	208.00	51.00	125.50	381.00	3356.25
net total	67.07	777.45	162.45	81.16	109.20	296.70	346.45	1.15	143.15	-16.55	61.35	335.45	2365.03

The top half of the sheet shows the society's monthly expenditure throughout the year with monthly totals shown in row 14. To the right (column O) is a list of annual totals for each item. The lower half of the sheet shows a corresponding table of receipts and the bottom row (25) gives a net total for each month. The net total for the whole year is given in slot O25.

Most of the labels on the left of the sheet occupy two slots. In this sheet, all slots have been set to a width of eight characters so some abbreviation of labels has been necessary. This could have been avoided by placing the labels in a 'window' of, say, fifteen characters in width. However, the techniques of extending slots and setting windows are outside the scope of this guide and users are advised to consult the ViewSheet User Guide.

Initially, the cursor will rest in slot A1. Use the downward arrow key to take the cursor down column A. The L that appears in the top left corner of the screen indicates that the contents of these slots are labels. Note that the labels appear in the `CONTENTS=` line at the top of the screen.

Now take the cursor to C8. At the top of the screen, the L changes to a V, indicating that this slot contains a value.

Move the cursor down to C14 and look at the `CONTENTS=` line. This slot contains a formula giving the total payments for April. In fact, C8C12 specifies a **range** of slots, and when used as a slot formula it means the total of all the slot contents in the range, i.e. C8+C9+C10+C11+C12.

This is an example of a sheet that might be used by a small business for calculating the weekly wages of employees and the total wage bill. This type of application is ideally suited to a spreadsheet as it often involves extensive use of calculations, particularly if the workforce is large.

Spend a few minutes exploring the sheet using the techniques described earlier for cursor movement. The current values should be the same as those shown in the illustration on the previous page.

You will see that the label in K2 includes single quotes. It is necessary in this case to include quotes – or any other character that could not be part of a formula – in order to indicate that 1985-86 is intended to be a label. Otherwise, ViewSheet will subtract 86 from 1985 and place the result in K2!

When you are ready to return to the top left of the sheet, press:

[*f7*] (GO TO SLOT)

then enter the slot reference:

A1 [RETURN]

This is a useful facility for moving quickly to anywhere on the sheet provided an appropriate slot reference is known.

Take the cursor to C12 showing the payment for room hire in April. Change the value in this slot to, say, 10.25. Almost immediately, the value in slot C14 adjusts to give the new total payments for April. All the other values that are dependant on C12 will also have been adjusted or **recalculated**. This may be verified by inspecting, for example, the net total in C25 or the final total in O25, and comparing these values with the original ones as shown on page 135.

It is important to remember the way in which ViewSheet recalculates. Recalculation takes place from left to right along each succeeding row from top to bottom. Thus if a formula contains references to slots that appear later in the sheet, it may not automatically recalculate when those values change. In this case, it is necessary to press:

SHIFT + f7 (RECALCULATE)

or simply TAB, which has the same effect.

One other facility is worth noting before you go on to create a spreadsheet. In a sheet like ACCOUNT, it is all to easy to inadvertantly omit a row or column and the omission might not even be noticed until the sheet is virtually complete.

In ACCOUNT, for example, a row of receipts from a council grant may have been missed out. Place the cursor anywhere on row 21 and press:

SHIFT + f2 (INSERT ROW)

Everything below the cursor will be shifted down to accomodate a new row. Place the label `council` into slot A21 and `grant` into B21.

Take the cursor to slot C21 and enter a value, say, 75. The first thing you will notice is that, unlike the other values on the sheet, this one is not displayed to 2 decimal places. Ignore this for the moment. You will see later in the chapter how a slot may be formatted to display a specified number of decimal places.

The interesting point to note is that the value in C24 has recalculated to display the new total, even though the original formula in C23 did not allow for the new row. This is because ViewSheet has automatically adjusted the range specified in C24 to take account of the new row. This may be verified by placing the cursor on C24 and looking at the `CONTENTS=` line. Originally it contained C18C21, now it contains C18C22.

Similarly, columns can be inserted by pressing:

SHIFT + f1 (INSERT COLUMN)

Before finishing with ACCOUNT, experiment with INSERT ROW and INSERT COLUMN as well as the complementary functions given by:

CTRL + f1 (DELETE COLUMN) and

[CTRL]+[f2] (DELETE ROW)

You will find that initially you are unable to delete rows and columns. This is because of a **protection** feature which has to be switched off. Press [ESCAPE] to switch to the command screen then type:

```
PROTECT OFF[RETURN]
```

You will see the message

```
Protection off
```

Press [ESCAPE] to return to the sheet screen and you will be able to delete rows and columns. It is good practice to switch protection back on once the required deletion has been carried out. This is done using by typing:

```
PROTECT ON[RETURN]
```

from the command screen.

Creating a Spreadsheet

Having examined a spreadsheet and tried some ViewSheet techniques you are now going to create a spreadsheet from scratch. The 'pen and paper' equivalent of the spreadsheet is shown below.

Weekly wages — week 21

normal hours = 40 — ovtime rate = 1.75

EMPLOY NAME	HOURS TOTAL	HOURS NORMAL	HOURS OVTIME	HOURLY PAY	GROSS PAY
Britton	40	40	0	4.50	180.00
James	34	34	0	4.50	153.00
Lewis	48	40	8	5.00	270.00
Monk	43	40	3	4.50	203.62
Toon	38	38	0	5.25	199.50
Tyler	52	40	12	5.25	320.25
TOTALS	255	232	23	29.00	1326.37

In our example, wages are calculated on the basis of a 'normal' working week of 40 hours. Any hours worked in excess of 40 qualify for the overtime rate, in this case 1.75 times the basic rate of pay.

The top row of the sheet gives a title and the week number. The next row specifies the number of hours in a normal working week along with the current overtime rate.

The remainder of the sheet consists of a table with six columns. Column 1 lists the names of the employees and column 2 lists the hours worked by each one. Columns 3 and 4 split the hours worked into normal (40 or less) and overtime. Column 5 lists the hourly rate of pay for each employee and the final column shows the gross pay. At the foot of each column is a total so the figure at the foot of column 6 shows the total wage bill for the firm.

Entering the spreadsheet

From the command screen, select mode 131 then clear the workspace using `NEW`.

Press [ESCAPE] to switch to the sheet screen.

The cursor should be in slot A1. This seems to be an appropriate slot for the title so type:

`Weekly` [RETURN]

Remember to press [CAPS LOCK] if all your letters are appearing as capitals.

This spreadsheet uses a slot width of seven characters. Although you can enter up to 239 characters in one slot, only the first seven will be displayed. The rest of the title will have to go into B1. Take the cursor into B1 and type:

`wages` [RETURN]

In the same way, enter the underline characters into slots A2 and B2.

Now enter `week` in E1 and 21 in F1

You could fit the whole of *week 21* into one slot, but it would then be treated as a label. As the same spreadsheet would probably be used for every other week of the year, it is better to enter 21 as a value so that it is easier to change.

Now complete row 4 by entering:

`normal`	into A4
`hours =`	into B4
`40`	into C4
`ovtime`	into D4
`rate =`	into E4
`1.75`	into F4

By now you will have noticed that the spacing of your entries leaves much to be desired. Labels, such as `week` and `normal`, are automatically ranged left whereas values, like 40 and 1.75, are ranged right.

Put the cursor back into E1 and press

[SHIFT]+[f8] (JUSTIFY LABEL)

The label in E1 moves from the left of the slot to the right. You may like to do the same to slots A1, A2, A4 and D4 in order to improve the spacing.

To further improve the spacing you need to move the values in F1, C4 and F4 from the right to the left of their respective slots. For reasons which will become clear later, the method for justifying values is different to that used for labels.

Place the cursor in one of the three slots mentioned above and press:

[f6] (EDIT SLOT FORMAT)

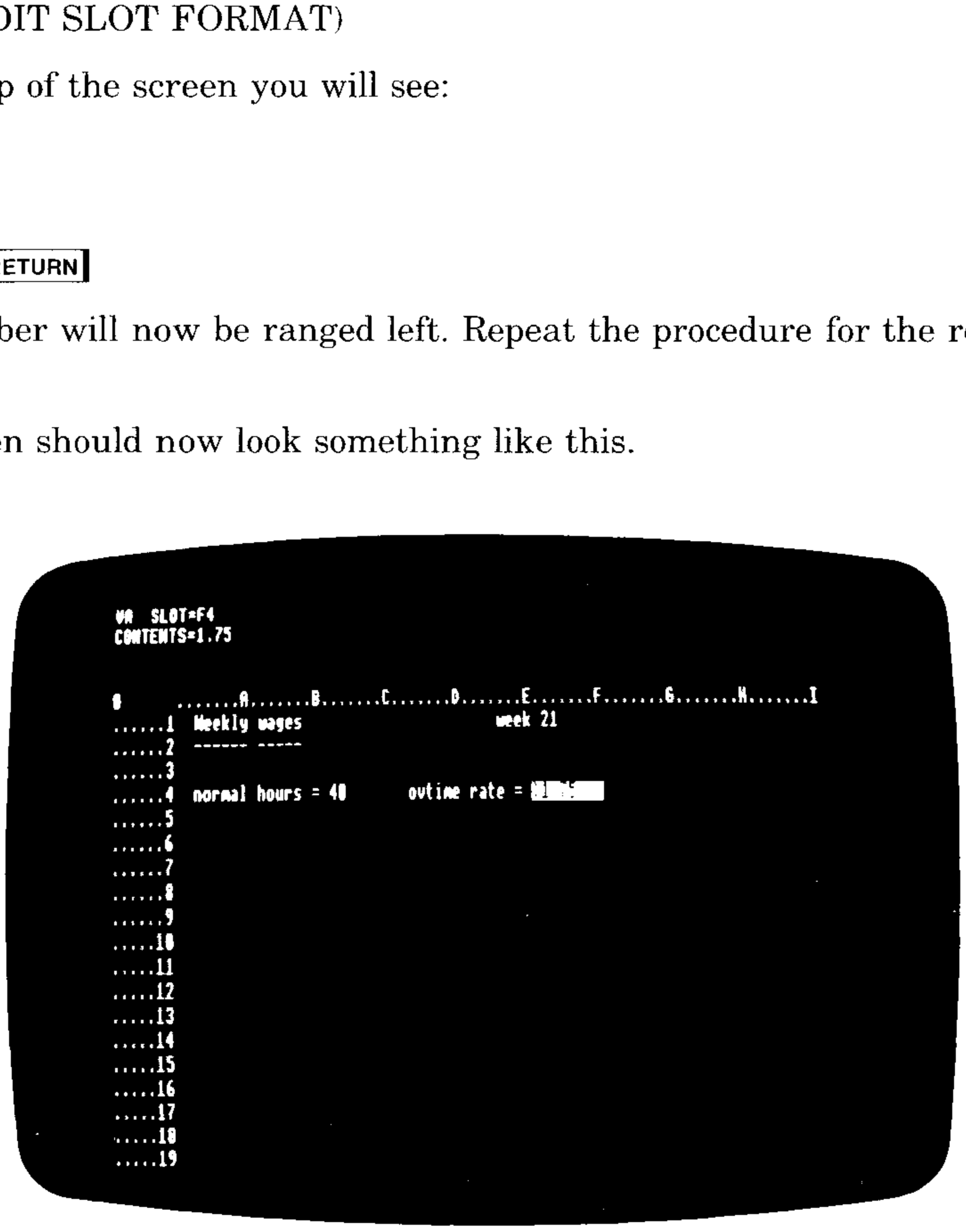
At the top of the screen you will see:

```
Format?
FRM
```

Type: L[RETURN]

The number will now be ranged left. Repeat the procedure for the remaining two slots.

The screen should now look something like this.



Now enter the column headings into columns A to F of rows 7 and 8. To maintain the style of the written spreadsheet this is best done with *caps lock* on.

Before entering the list of names in column A, press:

[CTRL]+[*f0*] (AUTO ENTRY)

The letter R appears at the top left of the screen.
Press it again and the R changes to D.
Press it again and the D disappears.

R stands for Right and D stands for Down. What AUTO ENTRY does is to save you the trouble of pressing the arrow keys every time you want to move on to an adjacent slot. It is useful when entering a long row or column of items. Just press [RETURN] and the cursor moves on to the next slot automatically.

To enter the list of names you need the D (Down) operation. You can carry on to enter the lines in A15 and A17 and the label TOTALS in A16.

Whilst AUTO ENTRY is switched on, you can use it to enter the HOURS TOTAL list in column B but for the moment do not make any entries below B14.

Switch AUTO ENTRY off by pressing:

[CTRL]+[*f0*] (AUTO ENTRY)

You will have noticed that the items in column B are not aligned with the heading. As mentioned earlier, labels automatically range left whereas values range right. Columns of figures are best left ranged right, so the solution is to justify the labels that comprise the column heading.

Position the cursor in B7 and press:

[SHIFT]+[*f8*] (JUSTIFY LABEL)

Repeat for the other half of the heading, in slot B8.

The spreadsheet will look neater if all the headings range right. Repeat the JUSTIFY procedure for all the labels in rows 7 and 8.

Each item in row 15 is identical. It would not be too much trouble to type the label '– – – – – – –' six times, but there is an easier way that is particularly useful for large spreadsheets. Press:

[*f0*] (REPLICATE)

The system responds with a prompt:

```
From - To?
```

Type:

```
A15 - B15F15[RETURN]
```

This **replicates** the contents of A15 into columns B to F of row 15. As you have

seen, the specification of a range, like B15F15, is useful for a variety of operations in ViewSheet.

Repeat this procedure to enter row 17.

Your spreadsheet should now look like this.

Before continuing, it is important to distinguish between values that must be entered directly, and those which ViewSheet can calculate from a fomula. According to the rules outlined earlier, the HOURS NORMAL and HOURS OVTIME values can be calculated from the HOURS TOTAL. Similarly, the GROSS PAY will be calculated from values in the preceding columns. Therefore the only remaining values to be entered directly are those in column E, HOURLY PAY.

Enter the first of these, 4.50, into E9. A problem is immediately apparent. Trailing zeros are ignored and you end up with the figure 4.5. For sums of money, you need a slot format that maintains two decimal places whatever the value. To do this, press:

[f6] (EDIT SLOT FORMAT)

In response to:

```
Format?
FRM
```

Type: `D2`

This stands for two decimal places. Note that you can combine instructions for

editing slot formats by typing, for example, D2L which specifies two decimal places and ranges left.

Try entering the remaining items in column E using AUTO ENTRY. Remember to edit each slot format to two decimal places.

Formulae

You are now left with three empty columns and a blank TOTALS row along the bottom of the table. The values in these positions will all be calculated by ViewSheet. All that remains is to devise a suitable formula for each slot.

Begin with the formulae in the TOTALS row as these are the most straighforward. To obtain a total for column B HOURS TOTAL, place the cursor in B16 and type:

B9+B10+B11+B12+B13+B14 [RETURN]

Almost immediately, the sum of the six numbers appears in B16. Although this method is perfectly acceptable for totaling fairly short lists of values, it is clearly impractical for lengthy additions. A better way is to specify a range of values, as shown earlier in the ACCOUNT spreadsheet. Leaving the cursor in B16, press:

[SHIFT]+[f9] (DELETE SLOT)

This erases the current contents of B16. Now type:

B9B14 [RETURN]

The result should be the sum of the values in all slots from B9 to B14 inclusive.

You will need to specify ranges in a similar way for the other slots in row 16. You can do this using replication even though each formula contains different slot references. Press:

[f0] (REPLICATE)

The screen will show:

```
From - To?
```

Type: B16 - C16F16 [RETURN]

The screen will show:

```
R)elative, N)o change?
B9 B14
```

Note that the slot reference B9 is highlighted. Your earlier use of replication involved slot contents that were not formulae. That is, they did not include slot references. This time, as you want to replicate a formula, ViewSheet offers you the option of replicating each slot reference **absolutely** or **relatively**.

Absolute replication copies the exact slot contents to the specified range, just as you did earlier when replicating a label. Relative replication will change B16 to C16, D16, E16 and F16 in each successive slot. To select relative replication on B9 press R.

The screen will show:

```
R)elative, N)o change?
```

B14

Now you have the option of selecting absolute or relative replication for the slot reference B14. Again, you do not want B14 to be copied exactly, but to change to C14, D14, E14 and F14, so press R.

Immediately the row fills with figures. If you pass the cursor along you will see in the CONTENTS= line at the top of the screen how the formula has been copied with the slot reference updated each time.

Now try using replication to enter formulae in column D, HOURS OVTIME. The overtime is calculated simply by subtracting the HOURS BASIC from the HOURS TOTAL. So D9, for example will contain B9-C9. Type this formula into D9 then use relative replication on both B9 and C9 in order to complete the column.

A simple formula is not sufficient to complete column C, HOURS BASIC. This value is calculated for each employee by looking at the HOURS TOTAL. if this is 40 or less then HOURS BASIC will be the same figure. If it exceeds 40, then HOURS BASIC will be 40. For the first employee, Ms Brittan, you could write this as:

'If B9 is more than 40, enter 40. Otherwise enter B9.'

which, as a ViewSheet formula, becomes:

```
IF(B9>40,40,B9)
```

This is a very powerful spreadsheet facility that enables a condition (B9>40) to dictate the value (40 or B9) to be given to a slot.

Enter the above formula into slot C9 and use replication to put corresponding formulae into the rest of column C.

At this point, your spreadsheet should look like this:

```
VA  SLOT=C9
CONTENTS=IF(B9>40,40,B9)

0      .......A.......B.......C.......D.......E.......F.......G.......H.......I
.......1 Weekly  wages                        week 21
.......2 ------  -----
.......3
.......4  normal hours = 40       outime rate =  1.75
.......5
.......6
.......7  EMPLOY   HOURS   HOURS   HOURS  HOURLY   GROSS
.......8    NAME   TOTAL  NORMAL  OVTIME     PAY     PAY
.......9 Brittan      40      40       0    4.50
......10 Janes        34      34       0    4.50
......11 Lewis        48      40       8    5.00
......12 Monk         43      40       3    4.50
......13 Toon         38      38       0    5.25
......14 Tyler        52      40      12    5.25
......15 ------- ------- ------- ------- ------- -------
......16 TOTALS      255     232      23      29       0
......17 ------- ------- ------- ------- ------- -------
......18
......19
```

All that remains is to complete the final column, GROSS PAY. This value is made up from two parts, basic pay and overtime pay. Taking Ms Brittan as an example, gross pay could be calculated as follows:

basic pay
= HOURS BASIC × BASIC RATE
= 40 × £4.50
= £180

overtime pay
= HOURS OVTIME × BASIC RATE × OVTIME RATE
= 2 × £4.50 × 1.75
= £15.75

gross pay
= basic pay + overtime pay
= £195.75

Expressing this as a ViewSheet formula:

```
(C9*E9)+(D9*E9*F4)
```

Enter this into F9 then replicate it into slots F10 to F14. Note that F4 is the overtime rate for all employees and as such should be copied exactly to each slot. Therefore when you see:

```
R)elative, N)o change?
F4 )
```

you should press N to signify *No change.*

Your complete spreadsheet should look like this:

```
VA  SLOT=F9
CONTENTS=(C9*E9)+(D9*E9*F4)

0      .......A.......B.......C.......D.......E.......F.......G.......H.......I
.....1 Weekly  wages                        week 21
.....2 ------  -----
.....3
.....4  normal hours = 40       ovtime rate =  1.75
.....5
.....6
.....7  EMPLOY   HOURS   HOURS   HOURS  HOURLY   GROSS
.....8    NAME   TOTAL  NORMAL  OVTIME     PAY     PAY
.....9 Brittan      40      40       0    4.50  180.00
....10 James        34      34       0    4.50  153.00
....11 Lewis        48      40       8    5.00  270.00
....12 Monk         43      40       3    4.50  203.62
....13 Toon         38      38       0    5.25  199.50
....14 Tyler        52      40      12    5.25  320.25
....15 ------- ------- ------- ------- ------- -------
....16 TOTALS      255     232      23      29 1326.37
....17 ------- ------- ------- ------- ------- -------
....18
....19
```

This may have seemed a long and complex process to produce what is a relatively trivial spreadsheet. However, the principles involved are the same whether your sheet caters for six employees or sixty. The methods and devices that you have used in constructing the WAGES sheet will stand you in good stead for creating more complex and useful spreadsheets.

As mentioned previously, spreadsheets are particularly useful for answering questions of the type 'What if....?'. Below are four problems which you may like to solve using the WAGES sheet. Although with the help of a spreadsheet they might seem trivial, you can imagine the amount of calculation that might be involved if you were using the 'pen and paper' equivalent shown on page 138!

1 Having completed this week's wages sheet you find that Mr Monk has been fiddling his time sheet and should really be credited only 35 hours. Edit your spreadsheet accordingly.

2 The company is considering increasing the overtime rate from 1.75 to 1.95. Naturally, the boss is concerned about the possible effect on the total wage bill. What difference would it have made for week 21?

3 Another scheme under consideration is to cut the number of hours in a normal working week from 40 to 38.

This problem points to a deficiency in the spreadsheet as it stands. Although you entered 40 as a *value* in slot C4, all of the relevant formulae refer to the

constant 40 instead of the slot reference C4. If slot references are used in formulae, the whole spreadsheet will be recalculated each time a single slot value is altered. Therefore, if there is a choice, formulae should contain slot references instead of just values.

Printing

Printing from ViewSheet is very like VIEW in that codes are sent to a *printer driver* which controls the operation of the printer. There is a built-in driver for straightforward printing, whereas to utilise more sophisticated features available on some printers a *driver generator* is available separately.

Provided the system is correctly set up, printing the sheet that you have just created is very easy. Switch to the command screen and type:

```
PRINT RETURN
```

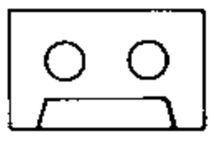
The area of the sheet that is covered by the screen will be printed. In mode 3, that normally includes everything from A1 to I19 which, in this case, includes your complete spreadsheet.

For printing anything outside the area from A1 to I19, ViewSheet uses **print windows**. A description of print windows is outside the scope of this guide and users are referred to the ViewSheet User Guide.

Using spreadsheets with VIEW

You can incorporate spreadsheets into VIEW files. This facility is useful for creating documents that include tables.

To save your spreadsheet in a suitable format (called, say, MONEY) proceed as follows.



Type:

```
*SPOOL MONEY RETURN
```

The screen will show:

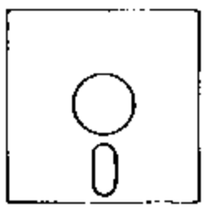
```
RECORD then RETURN
```

Press RECORD on your tape recorder followed by RETURN, then type:

```
SCREEN RETURN
```

The sheet is displayed on the screen as the file is created. Now type:

```
*SPOOL RETURN
```



Type:

*SPOOL MONEY [RETURN]
SCREEN [RETURN]

The sheet is displayed on the screen. Now type:

*SPOOL[RETURN]

The effect of this procedure is to save the top corner of the screen from A1 to I19. For spreadsheets that extend outside this range, print windows must be set and the necessary information can be found in the ViewSheet User Guide.

Once saved in this way, the file can be read into VIEW and subsequently edited or incorporated into another document. Note that spreadsheet files created in this way must be loaded using VIEW's READ command and **not** the LOAD command.

Other features of ViewSheet

With your experience, albeit brief, of using ViewSheet, you will be able to appreciate its power and recognise some potential applications. We have had to omit many of the more advanced facilities that ViewSheet offers, some of which are outlined below.

A summary of ViewSheet command screen commands is given in Appendix J but the ViewSheet User Guide should be consulted for full details of their use.

When using a large spreadsheet it is often convenient to display different sections of the sheet on the screen simultaneously. For example, suppose you were totalling a list of values in column A. If you were not sure how many items the list would contain, you might put the total at the bottom of the column in slot A255. However, if would be tedious to have to keep displaying different parts of the sheet in order to refer to items in the list and the total in A255.

This problem may be overcome using ViewSheet's **screen windows**. A255 can be displayed in a window at the foot of the screen whilst the list of values in column A is scrolled independently.

Up to ten windows can be set up at the same time, each of which may be set to a different width. This enables displays to be set up using wide slots for labels and narrower ones for values.

As mentioned earlier, windows are also useful in printing. Different sections of a sheet can be printed adjacently, or incorporated into a VIEW document.

ViewSheet provides a variety of **functions** for use in slot formulae. You have already used one of them – the IF function. There are functions that can, for example, give the maximum, minimum or average values for a specified range.

Some functions have counterparts in BBC BASIC, for example INT, LOG and SIN.

A **lookup table** may be set up within a spreadsheet. ViewSheet will search through a list of values until it locates a specified item. The corresponding value in a second table will then be located and used as required. For example, suppose the first list contained numeric stock codes and the second list, prices. A code could be referenced within a slot elsewhere in the sheet as a result of which the corresponding price would be displayed.

Another useful facility allows data to be displayed in the form of bar charts. Within a specified section of the sheet, numbers are automatically represented using lines of asterisks.

5. Filing Systems

What is a filing system ?

Virtually every computer application (barring the most trivial) requires some kind of access to an external storage medium, such as cassette tape or magnetic disc (either connected directly to the computer or provided as part of a network of computers). In the main, this is because the portion of the computer's memory which can be used to hold programs and data (the **RAM**) is unable to maintain its contents when the power is switched off. Also, in certain circumstances, the RAM may not be large enough to hold both a large quantity of data **and** the program which processes it. Clearly, if programs and data are to be held outside the memory in this way, the user must be provided with a convenient means of referring to them, in order to:

- retrieve (LOAD) existing items;
- access existing items (i.e. selectively retrieve parts of items without having to load them in their entirety);
- store (SAVE) new items.

The items are normally referred to as **files** and it is a **filing system** which provides these, and many other facilities.

Standard Filing Systems

Your computer comes equipped with four standard filing systems:

- the Cassette Filing System (CFS);
- the ROM Filing System (RFS);
- the Disc Filing System (DFS);
- the Advanced Disc Filing system (ADFS).

therefore enabling the computer to access files held on cassette tape, cartridge ROM sockets, conventional flexible (floppy) discs, and even hard (Winchester) discs. In addition, optional filing systems may be fitted to the machine enabling it, for example, to act as a station in an Econet network.

Whenever the computer is switched on, or subjected to a hard or soft break (CTRL+BREAK or BREAK), it automatically selects the filing system designated by the contents of the CMOS RAM (see page 23). This becomes the current filing system and it remains in force until you tell the MOS that you wish to use a different system (using one of the commands described below). The filing system selected on power-up may be changed by use of the *Control Panel* utility (see page 23) or by means of the *CONFIGURE command which is fully described in the Reference Manual.

It is important to realise that each filing system is, as far as is possible, compatible with all the others. This means that the command required to, say, load a file into memory, is equally applicable to the Cassette Filing System, the Disc (or Advanced Disc) Filing System and even the Advanced Network Filing System. It is therefore extremely easy to carry out operations such as transferring a file from one medium to another, merely by changing filing systems once the file is resident in the computer's memory. However, the range of available options increases with the sophistication of the storage medium and, whilst all Cassette Filing System commands are applicable to the Disc Filing System, the reverse is not necessarily true. Similarly, the Disc Filing System commands are applicable to the Advanced Disc Filing System, which itself contains a number of specialised commands. The ROM Filing System is an exception to this general rule because of the *read-only* nature of the medium.

The BASIC Language, VIEW, ViewSheet and the Editor (see page 164) each have their own built-in commands for communicating with the current filing system (i.e. LOAD, SAVE, SCREEN etc.) and their purpose and effects are described in the appropriate chapters. What follows in the remainder of this chapter is a description of the way in which files are organised by each filing system and mention of the operating system commands necessary to use the filing system at its most elementary level. The complete range of commands is summarised in Appendix E and full details can be found in the Reference Manual.

The Cassette Filing System (CFS)

Selected by: *TAPE RETURN

Description

The Cassette Filing System is the most basic, and hence the least sophisticated of the available filing systems; the storage medium is standard audio recording tape accessed via an ordinary domestic cassette tape recorder.

Files are stored on the tape in strict sequence, as a series of short **blocks** and the Cassette Filing System makes no attempt to maintain a record of the files stored on a particular cassette. This, and a great many other functions are left to the user. For example, he or she must ensure that:

- the correct tape is loaded into the recorder;
- when a file is to be LOADed, the tape is positioned (approximately) at the start of the required file;
- when a file is to be SAVEd, the tape is positioned such that the new file does not overwrite any existing files which might be needed at some later stage. The user must even ensure that the file is actually recorded by depressing the RECORD button on the recorder!

The solution to these problems is merely one of sensible cassette management, i.e. ensuring that each cassette is adequately labelled with its contents and approximate tape counter settings for both the start and end of each file. However, the filing system itself provides what help it can, as described below.

When a file is to be LOADed, a search is made (from the current tape position) for the beginning of the file with the corresponding name. The message:

```
Searching
```

is displayed together with the names of any files (or parts of files) which are encountered. This information can often tell you whether you are in approximately the correct position and 'Fast Forward' and 'Fast Rewind' may be used to reposition the tape if necessary.

Once the start of the required file is located, the message:

```
Loading
```

is displayed together with the name of the file and a count of the blocks as they are loaded. (Note that this count is in hexadecimal rather than conventional decimal notation, such that the tenth block is displayed as `0A`, the eleventh as `0B`, the fifteenth as `0F`, the sixteenth as `10` and so on.) As a (very) rough guide, each block takes approximately 2 seconds to load, with a gap of about half a second's duration between each pair. A bleep is emitted from the speaker when loading is complete and, if motor control is available, the cassette motor is switched off automatically.

When a file is to be SAVEd, the filing system reminds the user to press the correct buttons on the recorder by displaying the message:

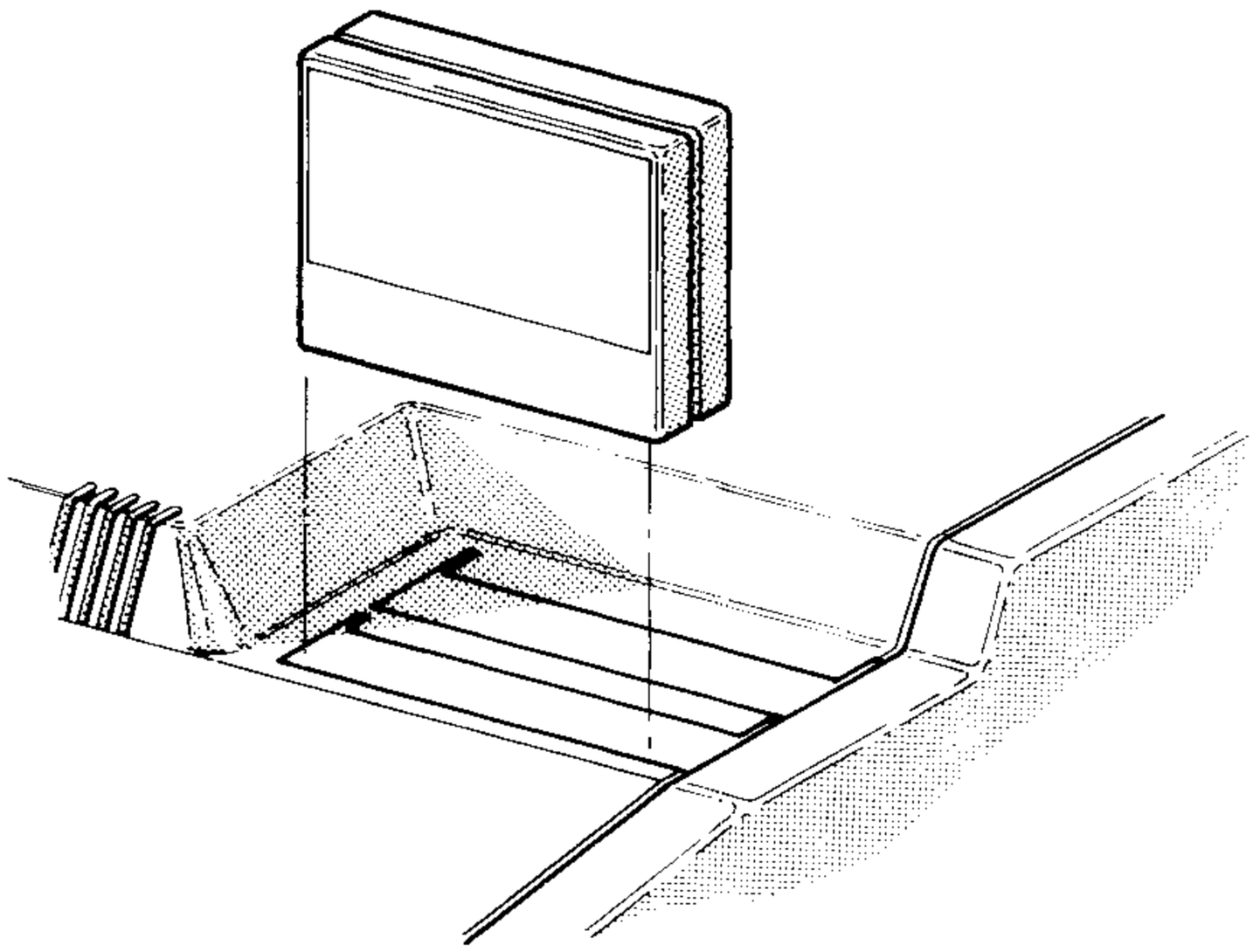
```
RECORD then RETURN
```

Actual transmission of the content of the file does not start until [RETURN] has been pressed – although the filing system will still have no means of knowing whether the recorder is ready or, indeed, if a recorder is connected at all! As with loading, the filing system displays the name of the file and a hexadecimal count of the blocks as they are transmitted – a bleep indicates that the transfer is complete.

The ROM Filing System (RFS)

Selected by: `*ROM`[RETURN]

The ROM (Read-Only Memory) Filing System is provided for the purpose of accessing files held in *cartridge ROMs*, which are inserted into the two special sockets above the numeric keypad:



Either or both of the sockets may be occupied at any time although it is advisable to switch the computer off when inserting or removing a cartridge.

The ROM Filing System provides similar facilities to the Cassette Filing System – the obvious difference being the *read-only* nature of the storage medium and the speed of access to files. A typical ROM Filing System cartridge will contain one or more files which are accessed using ordinary commands; a machine code game, for example, would be retrieved by means of a command such as:

```
*RUN GAME RETURN
```

which causes the machine code program GAME to be loaded into memory and executed – the similarity between the ROM Filing System and the Cassette Filing System is reinforced by the brief appearance of the `Searching` message.

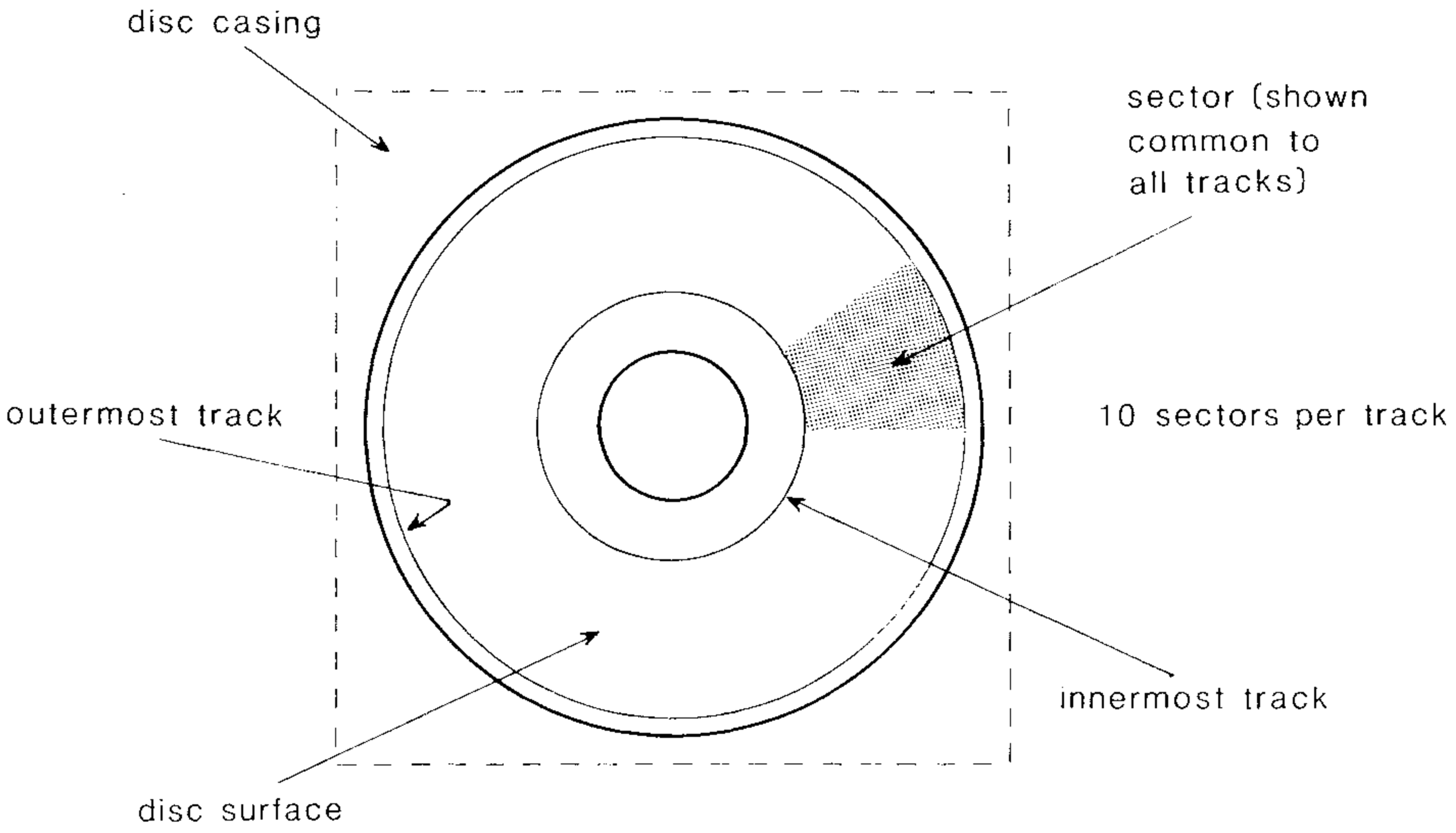
Note. It is also possible to use either of the two cartridge ROM sockets as extensions to the computer's existing Read-Only Memory although this use is independent of the ROM Filing System. Owners of BBC Model B / B+ microcomputers, for example, may already possess a variety of ROM software, packaged simply in a **chip** which is inserted directly into the printed circuit board. Assuming that the original manufacturer followed the guidelines laid down by Acorn these chips may be used with the new model by means of a special **User ROM cartridge** which is available through Acorn registered dealers.

The Disc Filing System

Selected by: `*DISC RETURN`

The conventional Disc Filing System is supplied in order to provide compatibility with previous BBC Microcomputers in which the storage medium

is the conventional 5.25" flexible (floppy) disc. On each disc, files are recorded on concentric rings of specially-formatted magnetic material called **tracks**, each of which is divided up into 10 **sectors**.



Depending upon the type of disc unit in use, either 40 or 80 tracks will be available – some disc units are said to be 'switchable' and can be set to read discs in either format. The recording format used by the Disc Filing System gives each surface of a 40-track disc a capacity of 102400 (100K) characters, and each surface of an 80-track disc 204800 (200K) characters.

All discs must be appropriately formatted by the computer before they can be used with the Disc Filing System and the necessary commands are described in Appendix E.

The actual process of loading and saving files is carried out by **read/write heads** which can be positioned over any of the available tracks on a particular disc surface as it rotates and this mechanism is referred to as a **drive**. The number of available drives is also determined by the type of disc unit:

- One drive is provided by the simplest unit which is referred to (quite logically) as a **single-sided** disc unit;
- Two drives may be provided either by a **double-sided** disc unit (which allows files to be stored on both surfaces of a single disc) or by a **twin single-sided** disc unit (which allows files to be stored on one surface of either of two separate discs);

– Most sophisticated (and hence most expensive) are the **twin double-sided** disc units which provide four drives by allowing files to be stored on both surfaces of either of two separate discs.

Each drive is referred to by a number, which will be 0 when the Disc Filing System is first selected and which may changed (if your disc unit has more than one drive) by means of the command:

```
*DRIVE number RETURN
```

The number sets the current drive and specifies the disc surface that will be used in all subsequent disc transfers. It is possible to access a particular drive without changing the current drive number by including a drive specification in commands like LOAD and SAVE. For example, if PROG1 is a BASIC program stored on the disc surface corresponding to drive 2, the command:

```
CHAIN ":2.PROG1" RETURN
```

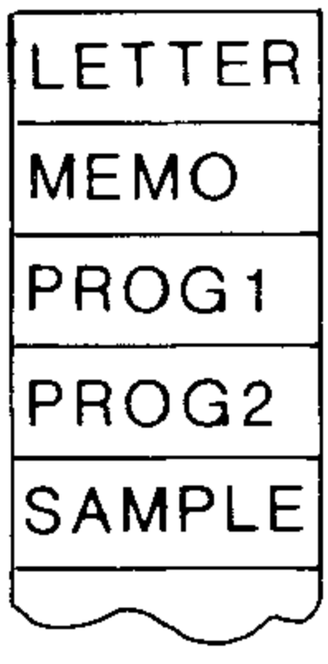
would load and run the program regardless of the current drive number. The **:** symbol indicates that a drive number is to follow and the **.** separates the drive number from the name of the file.

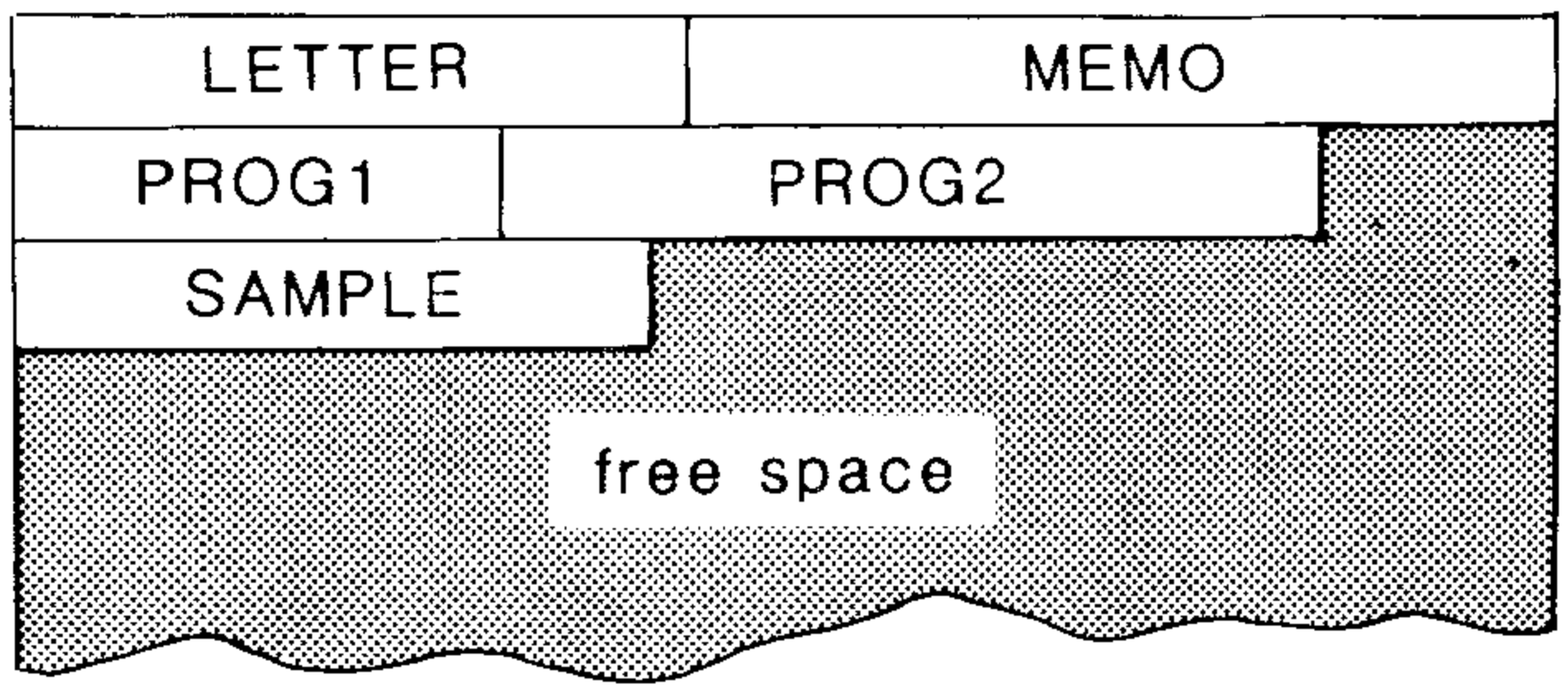
The fact that the read/write heads on a particular drive may be positioned over any of the available tracks, coupled with the fact that the disc rotates, means that it is possible to access a particular file merely by waiting the relatively short time for the appropriate sector(s) to pass under the heads. This is in stark contrast to the strictly *serial* access provided by cassette tape storage.

The disc catalogue

The names and information relating to the position of each file on the current drive are held in a **catalogue** stored on the disc itself and an empty catalogue is created by the formatting procedure mentioned above. The catalogue may contain a maximum of 31 entries and filenames may be up to 7 characters in length.

When a file is SAVEd, the Disc Filing System first examines the catalogue to determine a suitable free area on the current drive. The name of the file is then entered into the catalogue together with its position and the file itself is written to the designated area. After several such operations, the arrangement on a particular disc surface might be like that shown in the diagram overleaf:





When a file is loaded, the filing system first consults the catalogue to determine whether the file exists on the current drive. If it does, the filing system uses the remainder of the information in the corresponding catalogue entry to locate and retrieve the contents of the file.

Files may be deleted, renamed or copied to a different drive using the commands described in Appendix E.

Directories

The diagram in the previous section shows the catalogue in its simplest form – it is also possible to group files together into units called **directories** which are identified by a single character (normally an upper case letter). You might, for example, wish to group all your BASIC programs together in directory B and all your word-processed documents in directory W and this can be achieved in one of two ways:

– instruct the filing system to select a particular directory letter and to apply it to all subsequent LOAD and SAVE operations. The command:

  ```
  *DIR directory_letter RETURN
  ```

  is provided for this purpose. The letter chosen becomes the current directory.

  When first selected, the Disc Filing System automatically assigns directory $

as the current directory; the files shown in the diagram above would therefore be part of directory $ (which happens, in this case, to contain all the entries in the catalogue).

– incorporate a directory specification in all LOAD and SAVE commands, for example:

SAVE"B.PROG1" RETURN (note the quotes required by BASIC's LOAD and SAVE commands)

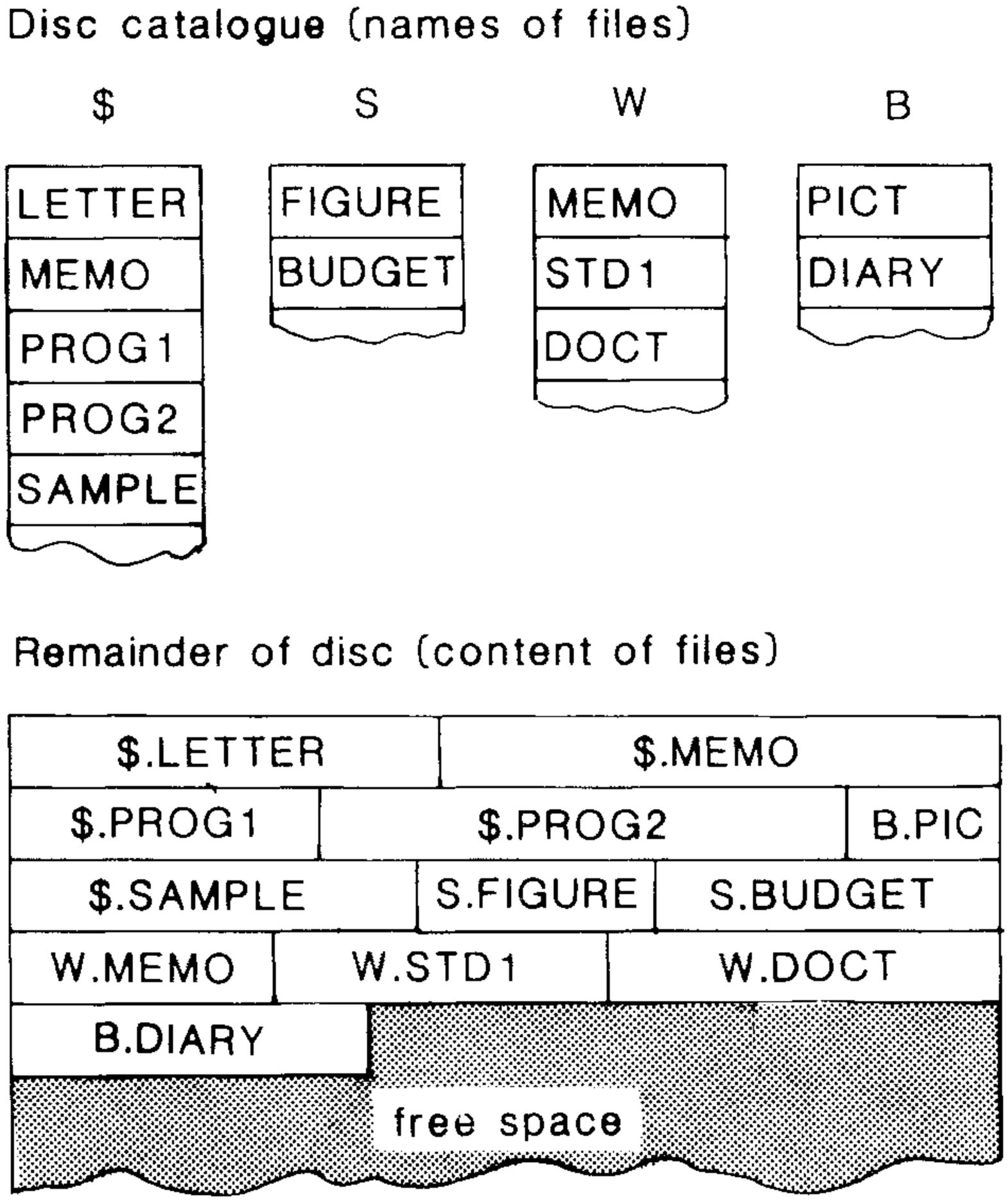
LOAD W.LETTER RETURN

The . is used to separate the directory letter from the file name. The absence of a directory specification causes the filing system to use the current directory.

It is, of course, possible to include both a drive and directory specification in a single command, such as:

SAVE :2.W.MEMO RETURN

The diagram below shows a possible Disc Filing System catalogue / directory structure for one drive – note that the use of different directories does not alter the maximum of 31 files per catalogue.



Libraries

The discussion of the *Control Panel* on page 23 makes mention of the fact that machine code programs are executed by means of the command:

```
*program_name RETURN
```

which (on the assumption that program_name is not a recognised operating system command) causes the current filing system to load and run the corresponding program. Such commands will cause the output of message such as:

```
Bad command
```

or

```
File not found
```

if the program cannot be located in the catalogue for the current drive / directory. Whilst it is perfectly possible to change drive / directory or even to issue a command such as:

```
*:2.U.UTILITY RETURN
```

(which explicitly states both drive and directory) the Disc Filing System allows you to specify a library which will always be referenced if a * command fails to find the file in the current drive / directory. A library is specified by means of the command:

```
*LIB drive_specification directory_specification RETURN
```

If only one of either the drive or directory specifications is given, the current setting of the other value is assumed.

The library facility is of most use with multiple drives, when it is possible to use choose one drive for general use and a different drive for utilities, which will always be directly accessible if the appropriate *LIB command is given at the start of a session.

Displaying a disc catalogue

A special command is provided in order to display a disc catalogue and its format is:

```
*CAT drive_number RETURN
```

If the drive number is omitted, the current drive is assumed.

The Advanced Disc Filing System

Selected by: *ADFS RETURN

The Advanced Disc Filing System is an alternative to the traditional Disc

Filing System; the term 'advanced' referring to its capabilities and performance rather than its suitability for new users of disc systems.

In essence, the Advanced Disc Filing System provides the same means of controlling the operation of one or more disc drives as the Disc Filing System but, in addition to a number of additional commands, it provides:

- a recording format which gives each surface of a 40-track disc a capacity of 163840 (160K) characters, and each surface of an 80-track disc a capacity of 327680 (320K) characters;
- a *hierarchical* directory structure, which overcomes the limit of 31 files per disc surface in the DFS, and which therefore means that ADFS can make optimum use of alternative types of disc unit, for example Winchester hard discs;
- the use (if possible) of both sides of a single disc as one entity;
- faster access to certain types of file (particularly in a network environment).

5.25" and 3.5" discs must be formatted before they can be used by the Advanced Disc Filing System and the necessary commands are described in Appendix E.

One of the major limitations of the Disc Filing System is the maximum of 31 filenames which can be held in the catalogue stored on each disc surface. This means that a disc may become 'full' merely because there is insufficient room to enter further filenames and not because there is insufficent room to accommodate the files themselves. The Advanced Disc Filing System overcomes this problem by means of the heirarchical directory structure which enables any number of files to be stored on a single disc, subject only to the amount of available disc space (and sensible use of the structure by the user).

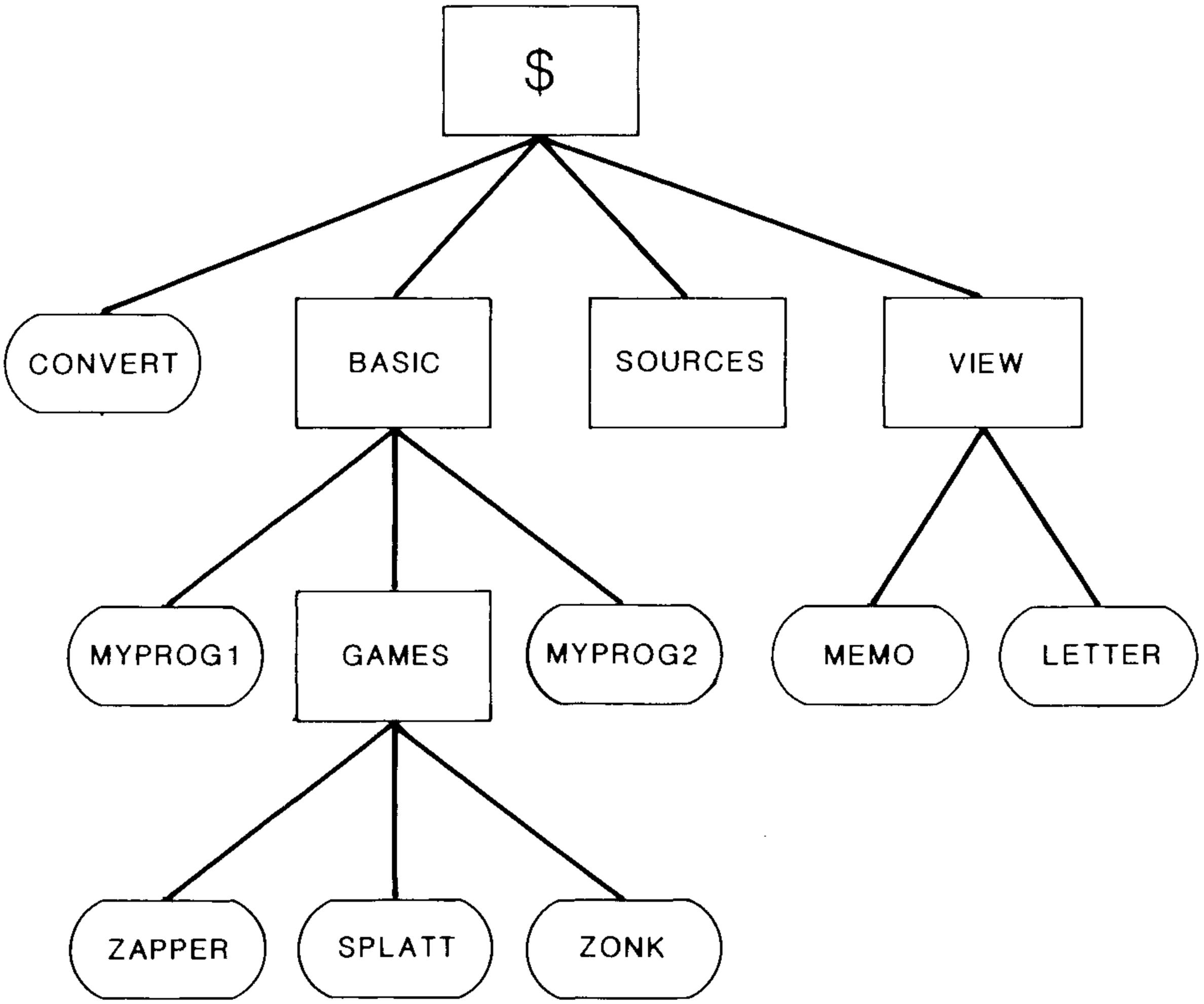
The basic unit of storage (as with the Disc Filing System) is a file, which is identified by a name of up to 10 (rather than 7) characters. Files are grouped into directories, which in the case of The Advanced Disc Filing System, may also have names of up to 10 characters (rather than a single letter). However, whereas grouping files together into directories is merely a convenience under the Disc Filing System, it becomes both a necessity and a positive advantage under the Advanced Disc Filing System, as described below.

The directory structure

The process of formatting a disc for use under the Advanced Disc Filing System creates a blank copy of what is known as the **root directory** which, for compatibility with the Disc Filing System, has the (unalterable) name $. The root directory may contain up to 47 entries, each of which is either:

- a simple file;
- a reference to a subordinate directory, which itself may contain both files and further subordinate directories.

This idea is illustrated in the following diagram, which shows a typical, simple, directory structure and which is described below:



The root directory contains one file (CONVERT) and three subordinate directory names (BASIC, SOURCES and VIEW);

- Directory BASIC contains two files (MYPROG1 and MYPROG2) and one further subordinate directory (GAMES). GAMES contains three files (ZAPPER, SPLATT and ZONK);
- Directory SOURCES is empty;
- Directory VIEW contains two files (MEMO and LETTER).

Referring to files

When the Advanced Disc Filing System is first selected it will normally read the contents of the root directory from the disc in drive 0. The root directory becomes the current directory and you may access any of the files which it contains (i.e. CONVERT in the example above), for example:

```
CHAIN"CONVERT"[RETURN]
```

If you attempt to, say, LOAD or SAVE the name of a subordinate directory, the filing system will respond with a message such as:

```
Is a directory
```

because the name is merely a pointer to the directory itself and not to the files it contains.

As with the Disc Filing System, access to files in other directories may be made in one of two ways:

- use the *DIR command to make the directory containing the required file the current directory (which also gives direct access to any other files in that directory);
- include a directory specification in the command itself, (leaving the current directory unchanged).

For example, to LOAD file MYPROG1 from directory BASIC, you could type:

```
*DIR BASIC RETURN
LOAD"MYPROG1" RETURN
```

which leaves BASIC as the current directory, or type:

```
LOAD BASIC.MYPROG1 RETURN
```

which leaves the current directory unchanged (i.e. as directory $).

Similarly, ZONK (if it is a BASIC program) could be executed by any of the command sequences given below:

- ```
  *DIR BASIC.GAMES RETURN
  CHAIN"ZONK" RETURN
  ```

  which leaves directory GAMES as the current directory;

- ```
  *DIR BASIC RETURN
  CHAIN"GAMES.ZONK" RETURN
  ```

  which leaves BASIC as the current directory;

- ```
  CHAIN"BASIC.GAMES.ZONK RETURN
  ```

  which leaves the root as the current directory.

Sequences of directory names, each separated by a . are sometimes referred to as **pathnames**.

A pathname may start only from the current directory or from the root directory – if the current directory is, say, BASIC, file MEMO may be accessed only by the pathname:

$.VIEW.MEMO
  ```

Similarly, if VIEW is the current directory, file ZAPPER may be accessed only by means of the pathname:

$.BASIC.GAMES.ZAPPER

There are, however, two exceptions to this general rule – the first being the use of the *BACK command; the second being the use of the ^ symbol. In addition to 'remembering' the current directory, the Advanced Disc Filing System also remembers the last directory accessed and the command:

```
*BACK RETURN
```

makes the previously selected directory current, i.e. it provides a convenient means of switching between two commonly-used directories.

The ^ symbol, when used as part of a pathname, means *parent* directory (i.e. the directory which contains the current directory). Thus, if GAMES is the current directory, file NEWPROG could be stored in directory BASIC by means of the command:

```
SAVE "^.NEWPROG" RETURN
```

instead of using the full pathname:

```
SAVE $.BASIC.NEWPROG RETURN
```

The symbol @ is used to denote the current directory and it may be used (together with a number of other, special-purpose symbols) with many of the commands described in Appendix E.

If a file is resident on any drive other than the current drive, it may be accessed by including a drive specification at the start of the necessary pathname, for example:

```
*DIR :2.BACKUP.VIEW RETURN
```

which makes directory VIEW, within directory BACKUP (itself in the root directory) on drive number 2 the current directory. A command containing a drive specification only, such as:

```
*DIR :1 RETURN
```

causes the Advanced Disc Filing System to load the root directory from the designated drive; it is therefore similar in effect to the Disc Filing System's *DRIVE command.

Creating subordinate directories

Whereas SAVE (or its equivalent) is used to store files in the current directory, the *CDIR command is provided in order to allow the creation of a new subdirectory in the current directory. Thus, directory SOURCES could be

divided into two subdirectories, PASCAL and COMAL, by means of the following command sequence:

`*DIR SOURCES` RETURN	(Sets current directory)
`*CDIR PASCAL` RETURN	(Creates subdirectory PASCAL)
`*CDIR COMAL` RETURN	(Creates subdirectory COMAL)

In each case, the effect is to create a new (empty) directory with the corresponding name.

Libraries

Under the Advanced Disc Filing System (as with the Disc Filing System), it is possible to specify a *library* to contain any frequently-used utility programs. This may be assigned explicitly using the *LIB command (as described on page 215)or implicitly by ensuring that the library directory begins with the characters LIB and that it is an entry in the root directory.

Displaying a directory catalogue

Under the Disc Filing System, the *CAT command is used to display the catalogue for the current (or a designated) drive. Under the Advanced Disc Filing System, *CAT is used to display the catalogue for the current (or a designated) directory, and it may therefore be followed by a pathname. Thus:

`*CAT` RETURN	displays the catalogue of the current directory.
`*CAT $.BASIC` RETURN	displays the catalogue of directory BASIC (regardless of the current directory).
`*CAT :2.BACKUP.VIEW` RETURN	displays the catalogue of directory VIEW on drive 2 (VIEW being subordinate to BACKUP in the root directory).

6. The Editor

This chapter describes the **Editor** and shows, briefly, how it can be used to help in the preparation of both text and programs.

In essence, the Editor offers a similar range of functions to those available in the VIEW word processor in that it provides a large workspace into which text may be loaded, entered, edited and subsequently saved. The major difference, however, is that the Editor does not carry out on-screen formatting and it is therefore recommended that VIEW is used for all conventional word processing tasks (i.e. the production of letters, memos, reports etc.) and that the Editor is used primarily for creating and editing programs. It is mainly the latter use which is described here, with specific reference to the BASIC language. The Reference Manual contains a great deal of further information about the Editor, including a discussion of the powerful formatting commands which can be used in the generation of bulk text for display or printing.

Before you can learn how to use the Editor, it is important to learn the distinction between a text file and a file containing a BASIC program because it affects both the way in which you select the Editor and what you will see while it is in use.

- A **text file** is merely a sequence of characters, each stored as an ASCII code. It has no special format except that individual lines are separated from one another by means of a carriage return character (ASCII 13) and the end of the whole file is indicated by means of special *end of file* marker.

  A text file may also contain ASCII control codes, (i.e. ASCII code which does not correspond to one of the normal, printable characters).

- A **BASIC program file** is also a sequence of characters but the important difference is that it has a special format:

  - the file begins with a carriage return character (ASCII 13);
  - each program line begins with a line number followed by a count of the number of characters the line contains;
  - within each line, all keywords (PRINT, INPUT, REPEAT etc.) are represented by special **tokens** which are merely internal codes and which help to reduce the overall length of a program when it is stored or loaded into memory;
  - each line is terminated by a carriage return character.

The Editor has no way of knowing which type of file is being edited and it therefore treats every file as a sequence of characters, which means that text

files may be loaded directly into the Editor and, consequently, saved without any modification. Clearly, however, loading a BASIC program file into the Editor directly will produce a rather peculiar effect, because the Editor will attempt to display a single character corresponding to items such as the line number, the character count and each keyword token. This dificulty is overcome using the BASIC language's EDIT command, as described below.

Selecting the Editor

There are several ways of selecting the Editor and the choice of method will depend upon the type of file and the purpose of the editing session:

– If you intend to create either a text file or a BASIC program file from scratch, The Editor is selected by means of the command:

```
*EDIT RETURN
```

This has the effect of clearing the Editor's workspace ready for subsequent input in the form of pure text or BASIC program statements.

Text files may be loaded (and even inserted) into the workspace using the function key commands described below.

– If you intend to edit an existing text file, you may use the procedure mentioned above or, alternatively, use the command:

```
*EDIT textfilename RETURN
```

which selects the Editor and automatically loads the named text file into the workspace.

– If you intend to edit an existing BASIC program file, you must first select BASIC and load the required program:

```
*BASIC RETURN
LOAD"programfilename" RETURN
```

and then select the Editor by means of the BASIC language's own EDIT command, i.e:

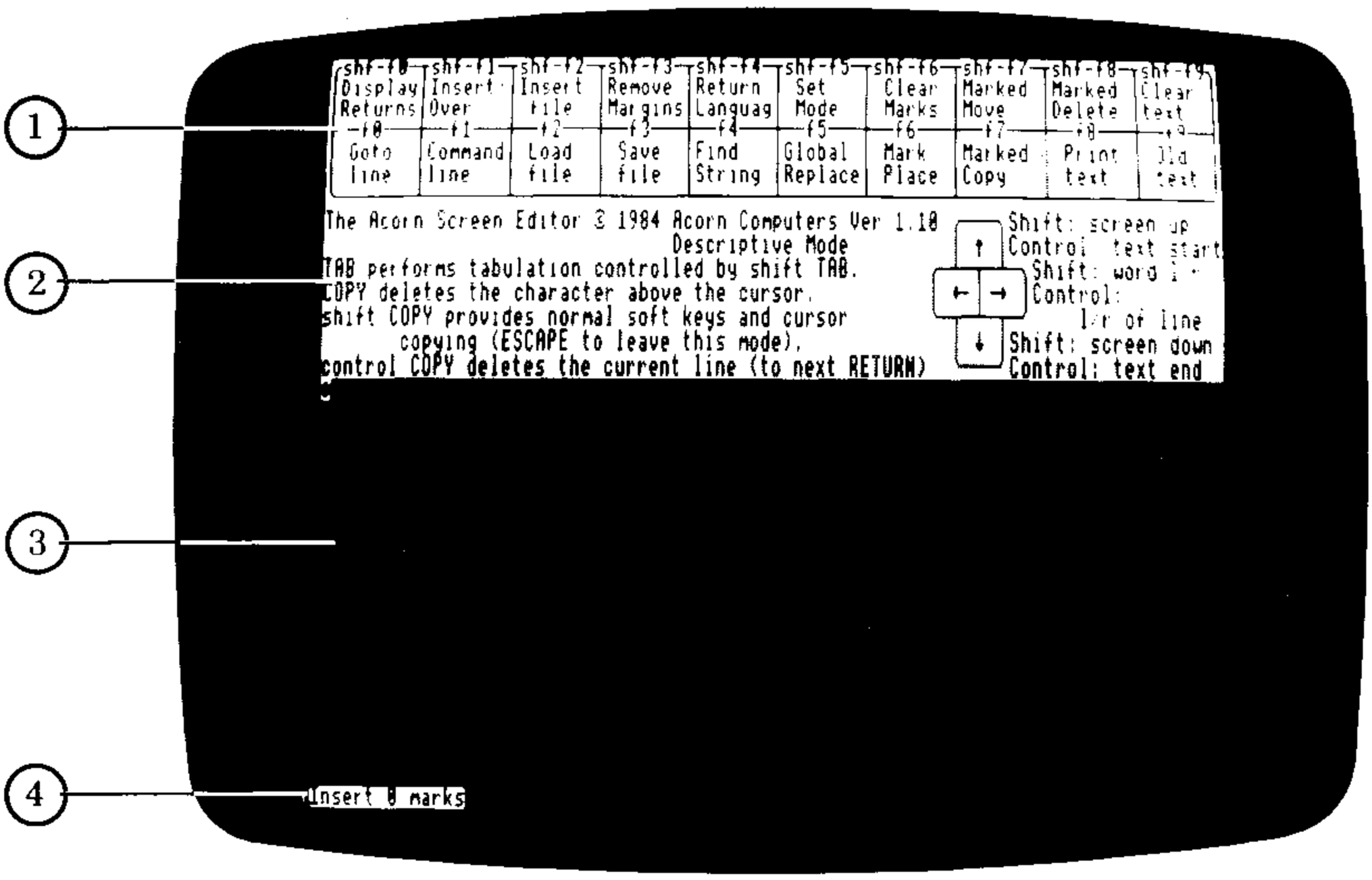
```
EDIT RETURN
```

This command merely LISTs the current program, not to the screen, but directly to the Editor's workspace – an action which effectively converts the program into a text file.

The reverse operation, that is reconverting the contents of the workspace into BASIC's internal format is achieved by means of another of the Editor's function key commands, as described below.

The standard EDIT Screen

Switch the computer on, or execute a hard break, then select the Editor using the *EDIT command. Almost immediately, the screen will clear and be replaced with the EDIT screen, which is made up of four components:



(1) The function key legends briefly describe the effect of the ten function keys, the bottom row being the effect if the corresponding function key is depressed on its own; the top row being the effect if the corresponding key is pressed in conjunction with [SHIFT]. Two additional functions (not shown on the screen) are provided by [CTRL]+[*f6*] and [CTRL]+[*f7*].

(2) The area below the function key legends is reserved for a brief description of the effect of various keys and, on depression of one of the function keys (with or without [SHIFT]), a summary of the effect of the selected command.

(3) The large, currently empty area in the lower half of the screen is the text display / entry area which, as in VIEW, provides a *window* containing a segment of the text in the workspace. As you would expect by now, you are able to alter the content of the text displayed in the window and move it in order to examine other segments of the text.

The black asterisk in the white rectangle is the *end of text* marker, and, if you look carefully, you will see the cursor flashing underneath it. This is because the workspace is empty and the current cursor position therefore corresponds with the end of the text.

(4) The small segment at the bottom of the EDIT screen is used to display information about the Editor's current status, messages and prompts for replies from the user.

For fairly obvious reasons, the display format described above is referred to as the Editor's **descriptive mode** and it operates in screen mode 128, thereby giving you the maximum available workspace size.

Other display modes

Other display modes may be selected by means of [SHIFT]+[f5] (SET MODE), which produces the prompt:

```
New mode  ?
```

at the bottom of the screen.

Press [SHIFT]+[f5] (SET MODE) and then K in response to the prompt. You have now selected **keyword mode** in which the area of the standard screen reserved for the key and command summaries is no longer present – the additional space being taken up by a larger text display / entry area.

Press [SHIFT]+[f5] (SET MODE) again but, in this case respond to the prompt with 7 [RETURN]. You have now selected the mode 7 Edit screen in which, as you can see, only the text entry / display area and the status line appear. In fact, any of modes 0, 1, 3, 4, 6 or 7 may be selected and they produce a display with the normal characteristics of the selected mode. Note, however, that despite the use of mode numbers in the range 0 to 7, the Editor always selects the corresponding shadow screen mode, in order to give the maximum workspace size.

Whilst this last type of Edit screen is less informative that either of descriptive or keyword modes, it does have the advantage of providing the largest possible number of lines in the edit window and most experienced Editor users use either the mode 0 or mode 3 screen display in conjunction with the keyboard insert supplied with the computer. However, six pages of the Welcome Guide hardly puts you in the experienced user category and you should reselect descriptive mode (using [SHIFT]+[f5] followed by D [RETURN]) before continuing with this section.

Note: the Editor display mode is one of the items stored by the CMOS RAM and when you next select the Editor (regardless of whether the computer has been switched off in between) it will automatically reselect the display mode you used last.

Entering text in the workspace

Try typing in the first two sentences from this paragraph, pressing [RETURN] at the end of each sentence and watch the effect on the screen. Each time a key is depressed, the corresponding character is displayed and the cursor (together with the end of text marker) moves one position to the right. At the end of each line, the cursor and marker move to the beginning of the next line and, if you were to continue typing sufficient text to fill the current window, all preceding lines would be scrolled up to make space for the new line, just as in VIEW.

Now press [SHIFT]+[*f0*] (DISPLAY RETURNS) and the position of each depression of [RETURN] will be shown as a letter M reversed out of a white block. We shall see the usefulness of this feature later in the chapter.

Now use the cursor keys to reposition the cursor under a character anywhere in the text you have just typed (noting that the position of the *end of text* marker remains unchanged. If you now type any characters, you will see that everything to the right of the cursor on the same line and all characters on subsequent lines down to the next return character, is moved along to accommodate the new characters. This effect is produced because the Editor always starts in what is referred to as **Insert mode**, as indicated by the word `Insert` at the bottom of the screen.

Now press [SHIFT]+[*f1*] (INSERT/OVER). This changes the word to `Over` and selects **Overtype mode**, in which a new character replaces any character in the current cursor position. [SHIFT]+[*f1*] acts as a toggle between insert and overtype modes. You might like to see the effect by repositioning the cursor and typing some further characters.

Further information about the use of the two different text entry modes is given in the next section but, with your rudimentary knowledge of the VIEW word processor, you should be able to understand the following features, which are unaffected by the choice of text entry mode:

– [SHIFT] and [CTRL] may be used in conjunction with the four cursor control keys:
  [SHIFT]+ ↑ moves the text display up one 'screenful';
  [SHIFT]+ ↓ moves the text display down one 'screenful';
  [SHIFT]+ ← moves the cursor to the start of the previous word in the workspace;
  [SHIFT]+ → moves the cursor to the start of the next word in the workspace;
  [CTRL]+ ↑ moves the cursor to the start of the text;
  [CTRL]+ ↓ moves the cursor to the end of the text (i.e. so that it coincides with the end of text marker);
  [CTRL]+ ← moves the cursor to start of the current line;
  [CTRL]+ → moves the cursor to the end of the current line.

– [TAB] may be used to move the cursor in one of two different modes referred to as *TAB below words* and *TAB columns of 8*. The choice of TAB mode is made by pressing [SHIFT]+[TAB] which displays the current TAB mode and toggles between the two.

TAB below words causes the cursor to be positioned immediately below the first character on the previous line, thereby providing a convenient means of producing sensibly indented program listings (such as provided by BASIC's LISTO commmand).

TAB columns of 8 causes the cursor to be moved across the screen in steps of 8 character positions. Note that the effect is cyclic, i.e. movement off the right of a particular line brings the cursor back on the left of the same line.

The effect of pressing other keys, such as [RETURN], [COPY] and [DELETE] is determined by the choice of text entry mode, as described below.

An example of text entry in insert mode

If necessary, reselect insert mode (using [SHIFT]+[*f1*]) and then clear the Editor's workspace by pressing [SHIFT]+[*f9*] (DELETE TEXT). Note that you must confirm your intention by pressing any other key. Leave, or reselect the DISPLAY RETURNS option (using [SHIFT]+[*f0*].

Now type in the short, and somewhat uninspiring BASIC program below, pressing [RETURN] at the end of each line. Note that it contains a few (!) deliberate mistakes which we shall use to illustrate the effect of some special keys.

```
10 REM Noddy's program (with apologies to A A Milne)
20 INPUT "What is your name "name$
30 PRINT '"HELLO "name$", how old are you ?";40 INPUT age%
50 PRINT '"Did you know that you are ";ABS(age%-5);
60 IF age%>5 THEN PRINT "older"; ELSE PRINT "YOUNGER";
70 PRINT "than Big Nose ?"
```

Clearly, if the program is indeed Noddy's work, the reference to *A A Milne* in line 10 is incorrect and needs to be replaced by *Enid Blyton*. This change can be achieved in a number of ways, the most obvious being to use [DELETE]. In Insert mode, the DELETE key removes the character immediately to the left of the current cursor position, closing up any remaining characters down to the next return character. We can therefore achieve the necessary deletion by positioning the cursor under the *)* in line 10 and pressing [DELETE] the appropriate number of times. The fact that we are in insert mode means that the characters *Enid Blyton* may simply be typed once the erroneous characters have been deleted, the *)* and the return character will move across automatically.

Line 30 also contains a mistake, in that it also contains line 40 (without an intervening carriage return). This mistake can be simply corrected by placing

the cursor on the *4* and pressing [RETURN], but watch the effect carefully – the return symbol is inserted at the current cursor position but the very fact that it is a return symbol causes the remainder of the line to be carried over to the start of a new line. We shall see later that the effect of [RETURN] in overtype mode is rather different.

Note that in insert mode, it is possible to join two lines together (i.e. delete the separating return character) by position the cursor at the start of the second of the two lines and pressing [DELETE].

Line 50 has the characters `" years ";` missing from the end of the PRINT statement and these can merely be inserted by pressing the appropriate keys once the cursor has been positioned over the return symbol.

The appearance of the output from the program would probably be improved if the characters `YOUNGER` were replaced by `younger` and this change could be made using the [DELETE] procedure described above. However, the occurence of a second mistake of this type gives us the opportunity to examine the effect of [COPY] which, somewhat surprisingly, actually deletes characters, but to the right of the current cursor position. The deletion can therefore be made by positioning the cursor under the " character preceding the letter *Y* and pressing [COPY] to delete the incorrect characters. Once again, the required characters may merely be typed in order to achieve the correction.

The final mistake (unless you introduced some more of your own) is the reference to *Big Nose* rather than *Big Ears*. Either of the two deletion and replacement techniques could be used again in this case, but take the opportunity to see the effect of another function key command – [f5] (GLOBAL REPLACE).

If you press [f5] the prompt:

```
Global replace:
```

will appear at the bottom of the screen and you can then specify both a target string and a replace string, separated by a / character. Our simple change may be achieved merely by typing:

```
Nose/Ears[RETURN]
```

and the Editor will confirm that,in this case, it has found 1 such target string and replaced it with the replace string. However, one of the most powerful features of the Editor is the sophistication of its search and replace functions ([f4] (FIND STRING) and [f5] (GLOBAL REPLACE)), an indication of which can be gleaned from the brief description produced in response to your depression of [f5]. Further information on this somewhat advanced editing technique can be found in the Reference Manual.

Remember that the content of the Editor's workspace is merely a sequence of

text which happens, in this case, to contain the line numbers, keywords and other symbols which make it obey the rules of the BASIC language. If the file is saved (using [f3] (SAVE FILE)), it will be saved as a text file and any subsequent attempt to LOAD it in BASIC will produce a Bad program message. If you wish to verify that Noddy's masterpiece actually works, you must supply the content of the Editor's workspace to BASIC using [SHIFT]+[f4] (RETURN LANGUAGE), which produces the prompt:

```
Language ?
```

At this point you should type BASIC[RETURN] and, after a momentary delay while the necessary conversion takes place, the screen will clear and be replaced by familar > prompt from the BASIC language. You may, of course, RUN or LIST the program and possibly make some minor alterations using the normal cursor editing functions.

To SAVE the program as a BASIC program file you must give an appropriate SAVE command whilst still in the BASIC system but you may return the the Editor at any time merely by typing:

```
EDIT[RETURN]
```

An example of text entry in overtype mode

Insert mode is the most commonly used of the two text entry modes but it is instructive to repeat the same sequence as above in overtype mode in order to see the differences between the two. In the main, these centre upon the effect of the [RETURN], [DELETE] and [COPY] keys.

Clear the Editor's workspace and retype the sample program on page 169, including the errors. Ensure also that the DISPLAY RETURNS option is ON and then select overtype mode (using [SHIFT]+[f1]).

To change *A A Milne* to *Enid Blyton* in insert mode, we positioned the cursor under the *)* character and made several depressions of [DELETE]. Try this technique again in overtype mode and watch the effect. The first difference is that the characters to the right of the cursor are not closed up as each character is deleted. The second difference is that when you type the new characters to correct the error, you 'run out of space' after Enid Blyt. In this case, the simplest solution is to continue typing the characters on), replacing the existing *)*. Notice, however, that while the *)* symbol is overtyped, the return symbol obligingly moves to the right for each remaining character depression. This is because its replacement would have the effect of removing the separator between two lines and the Editor would then have to make some judgement on the content of the unused character positions beyond the end of the new text. Clearly, the more sensible approach is that adopted, which considers any text immediately before a return to be an extension to the current line. The same argument is applied to any attempt to delete a return character.

If the mistake had been nearer the beginning of the line the technique above would mean retyping almost the whole line and a better approach would be to switch to insert mode (possibly temporarily) in order to add the necessary characters.

Now try the technique for splitting the two lines; line 40 steadfastly refuses to budge and depression of [RETURN] merely repositions the cursor at the start of the next line. This is because overtype mode considers [RETURN] to be just that, an instruction to return the cursor to the start of the next line and the only instance when a return character is inserted into the workspace is if the cursor coincides with the end of text marker, i.e. when a completely new line is created. The two lines may therefore be separated only by switching back to Insert mode.

The missing characters in line 50 may merely typed in after positioning the cursor over the return character, as in the example above.

Replacing YOUNGER with younger is a relatively straightforward task as both strings are the same length but, once again, it is useful to see the effect of the [COPY] key in Overtype mode. If [COPY] is pressed with the cursor under the " character preceding *Y*, it is the " which disappears, and not the *Y* (as in Insert mode). Furthermore, you may hold your finger down on [COPY] for as long as you like because the cursor does not move. In other words, in Overtype mode, [COPY] has the effect of deleting the character curently under the cursor and it is therefore foolish to allow repeated deletion in one direction or the other.

[f4] (FIND STRING) and [f5] (GLOBAL REPLACE) operate identically in both Insert and Overtype modes.

Block operations

If you examine the function key legend at the top of the Descriptive mode display (or, indeed, the Editor keyboard insert) you will find several references to *marks* which are the basis of all block operations within the Editor and which are briefly described below. It must be said, however, that block operations are more appropriate to conventional text editing rather than the editing of BASIC programs.

The display at the bottom of the screen always shows the current number of marks (initially 0) and a mark may be inserted at the current cursor position by pressing [f6] (MARK PLACE).

Deletion of a block is achieved by positioning one mark over either the start or the end of the block to be deleted and using the cursor itself to identify the other end of the block. The actual deletion is carried out using [SHIFT]+[f8] (MARKED DELETE).

In order to either move or copy a block of text, both the start and end of the required block must be marked – the cursor is used to indicate the destination position. Blocks are moved using [SHIFT]+[f7] (MARKED MOVE) and copied using [f7] (MARKED COPY). Movement (and deletion) of a block removes the current markers, but the copy operation does not, thereby enabling multiple copies of a particular block to be made if required.

Scroll margins

If you examine the function key legend in either descriptive or keyword mode (or, indeed, the keyboard insert) you will see that [SHIFT]+[f3] represents CLEAR MARGINS. In the context of the Editor, the margins refer not to the width of the text (as in VIEW) but to the points at which the content of the screen begins to **scroll** (i.e. move up or down one line, thereby bringing a new line onto the screen). Unless you specify otherwise, the Editor begins scrolling whenever the cursor reaches the fifth line from the top or bottom of the current text display / entry area – the idea being to enable you to see the context of the line you are editing.

[SHIFT]+[f3] (CLEAR MARGINS) sets the top and bottom scroll margins to the top and bottom lines of the text entry / display area respectively. However, either the top or the bottom scroll margin may be reset by using [SHIFT]+[f6] and [SHIFT]+[f7] respectively – the line chosen being that containing the cursor at the time of the function key depression. Thus, by setting the top and bottom scroll margins to adjacent lines, it is possible to make the whole screen scroll whenever the cursor is moved to another line.

7. The Terminal Emulator

Preceding chapters have described the BASIC language, the VIEW word processor, ViewSheet and the System Editor – all examples of *applications software* designed to make your computer a powerful, general-purpose tool. This chapter makes brief mention of the remaining item of standard applications software – the **Terminal Emulator** – which provides the facilities necessary to use your computer as either a local or a remote terminal to other computer systems. A full description is beyond the scope of this guide and users wishing to make use of the Terminal Emulator should consult the appropriate chapter in the Reference Manual.

In essence, the Terminal Emulator is a machine code program in ROM which, when called by the command:

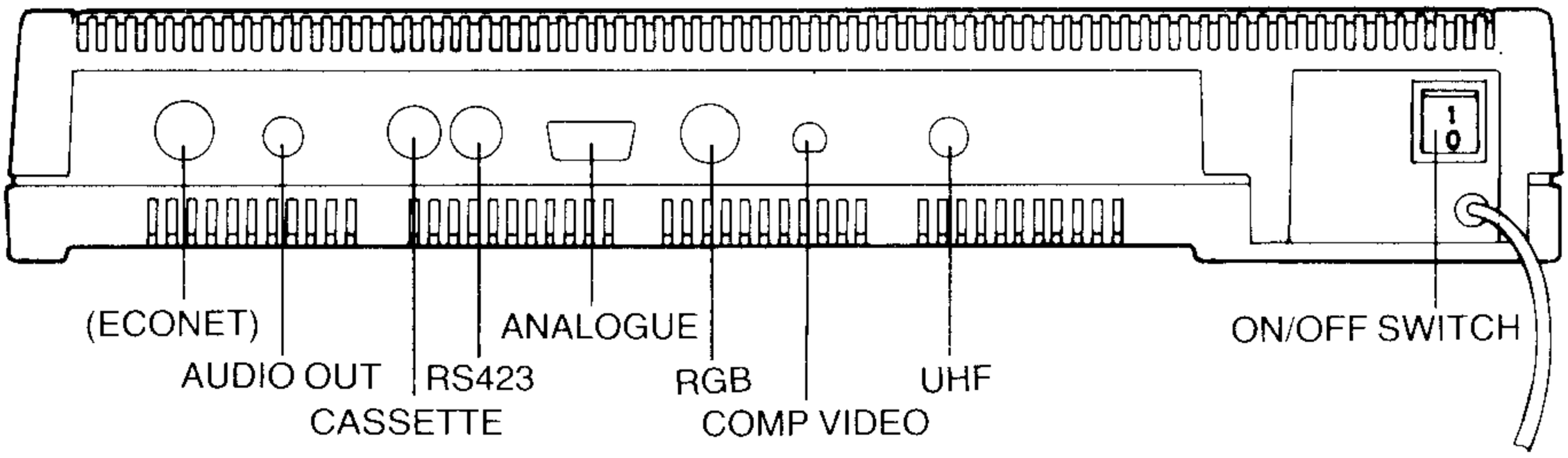
```
*TERMINAL RETURN
```

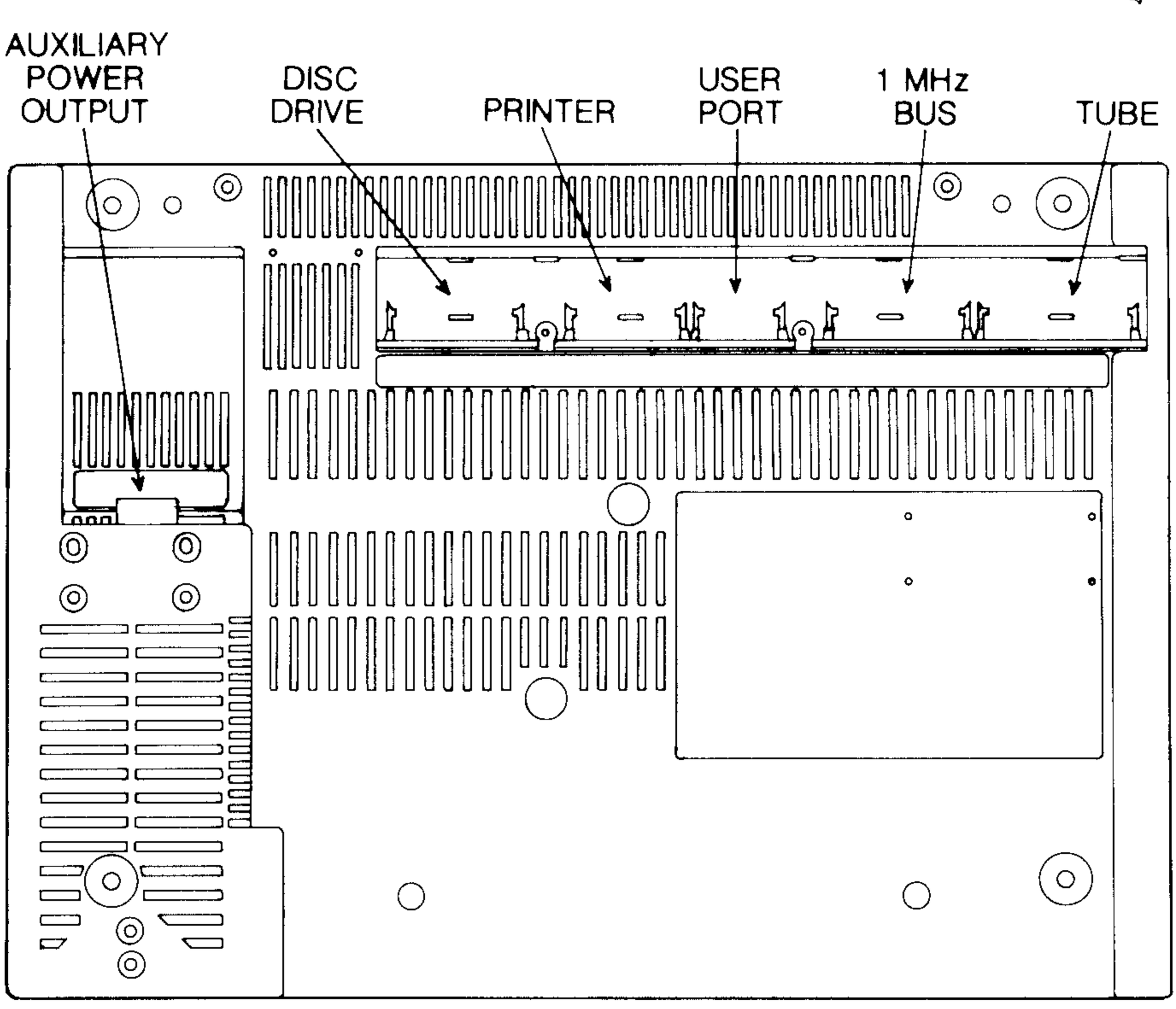
controls the transmission and reception of data between the host computer system and your microcomputer; all transmission takes place via the RS423 serial interface. The computer may be connected directly to a local system; connection to a remote system requires the use of a suitable modem or acoustic coupler which is itself connected to the host system via telephone lines (or a dedicated data network). Space for an internal modem has been left inside the computer's case.

8. Expanding the System

This section covers expansions to the basic system, both in the form of peripherals (i.e disc units, printers etc.) and in the form of additional, so-called **co-processors**. The aim here is to provide basic information – it is important to read and follow any additional instructions provided with any expansion unit.

The illustrations below show the various sockets on the back and front underside of the computer.





Connecting a colour monitor

Whilst the use of an ordinary domestic colour television is adequate for many users, those with specialist requirements, such as a high volume of word-processing (see note below), or extensive use of colour graphics, will benefit from the addition of a colour (RGB) monitor. (RGB stands for Red-Green-Blue, the three basic colours which, in conjunction with the normal black screen background, provide the full range of eight basic and eight flashing colours.)

Colour monitors provide better definition and have a further positive benefit – they release the household television for use by the remainder of the family!

Connection to the computer is by means of a 6-pin DIN lead which is plugged into the socket marked 'RGB' on the back of the computer.

No special commands are required to drive a monitor although, as with a domestic television, you may also need to use the Control Panel utility to reset the vertical screen alignment (see page 24).

Note

Serious users of VIEW or ViewSheet may also consider the connection of a monochrome monitor (typically green-on-black or amber-on-black), which offers better resolution at less cost than a full-colour, RGB monitor. Such units are normally connected to the computer via the 'Video out' connector on the back of the computer.

However, better quality medium- or high-resolution RGB monitors can also cope more satisfactorily with the display of small characters.

Both RGB and monochrome monitors are powered directly from the mains.

Connecting a disc unit

By far the most useful expansion to the basic system is the connection of a disc unit which provides rapid access to files and which also removes much of the manual intervention associated with cassette tape storage.

5.25", 3.5" and 3" disc units

Single or multiple drives of this type are connected to the 'disc drive' connector located on the front underside of the computer. (Your supplier will be able to ensure that the disc drive has the appropriate plug on the end of the connecting cable). Some disc units are powered directly from the mains; others are able to take their power directly from the computer using a connecting cable and a special plug which is plugged into the 'auxiliary power output' socket.

If you have not already done so, you will probably want to re-configure your

computer (using *CONFIGURE or the Control Panel utility) so that it selects either the Disc Filing System or the Advanced Disc Filing System on power-up.

Winchester Disc units

The Acorn (and other) Winchester hard disc units are mains powered and connected via the '1 MHz bus' connector, located on the front underside of the computer.

Winchester disc drives must be operated under the control of the Advanced Disc Filing System.

Connecting a printer

A printer is almost essential if you are going to use the VIEW word-processor or ViewSheet; it is a convenience if you are writing programs of your own.

Your computer can be used with the vast majority of currently-available printers, which are of two types:

- **parallel printers**, incorporating a 'Centronics (r) interface';
- **serial printers**, incorporating a 'RS232-C (or V24) interface'.

All printers are mains powered; parallel printers are connected to the 'printer' socket on the front underside of the computer; serial printers are connected (using a 5-pin (Domino) DIN plug) to the RS423 socket on the back of the computer. (RS423 is a later, but compatible standard to RS232C).

You must tell the computer which type of printer you are using, and this is done using the *FX5 command:

- *FX5,1 RETURN tells the computer that you are using a parallel printer;
- *FX5,2 RETURN tells the computer that you are using a serial printer; and you must also tell the computer the speed (referred to as the **baud rate**) at which the printer operates using the *FX8 command:

*FX8,1 RETURN	–	75 baud
*FX8,2 RETURN	–	150 baud
*FX8,3 RETURN	–	300 baud
*FX8,4 RETURN	–	1200 baud
*FX8,5 RETURN	–	2400 baud
*FX8,6 RETURN	–	4800 baud
*FX8,7 RETURN	–	9600 baud
*FX8,8 RETURN	–	19200 baud (not guaranteed)

Many printers have an automatic line-feed facility and in such cases, it is necessary to tell the computer not to send additional line-feed characters to the printer. This is done using the command:

```
*FX6,10 RETURN
```

*FX6 tells the computer not to send a particular character to the printer; 10 is the character code for a line-feed.

It is possible to make the computer carry out each of the necessary commands on power-up using the *CONFIGURE command or the Control Panel utility.

Connecting Joystick(s)

Many computer games and some rather more serious applications can be enhanced by the use of one (or a pair of) **joystick(s)**, which can be used to position objects displayed on the television or monitor screen.

Joystick(s) are connected via the 'analogue in' socket which is located on the computer's back panel.

Joysticks do not require a power supply.

The 'analogue in' socket may also be used for the input of other analogue signals, such as might be produced from equipment monitoring scientific experiments.

Connecting a Teletext Adapter

Using a Teletext Adapter, your computer can be made to receive (and store) pages of teletext information broadcast by both the BBC's CEEFAX and the IBA's Oracle services.

The Teletext Adapter is a mains powered external expansion unit incorporating a conventional television aerial connection and a tuner unit enabling each of the four currently-available UK television channels to be selected (i.e. BBC1, BBC2, ITV and Channel 4). The unit itself is connected via the '1MHz bus' connector and operates under the control of a special Telesoft Filing System (TFS), which is supplied (as a ROM) with the unit.

The Teletext Adapter may also be used to access programs transmitted by the BBC's Telesoftware service (see CEEFAX pages 700-).

Connecting a Prestel Adapter

Whilst the Terminal software in the computer (and a suitable modem) provide the facilities necessary to connect your computer to a variety of other computer systems, The Acorn Prestel Adapter provides you with a convenient means of accessing, and interacting with the British Telecom **Prestel** Service (and other similar Viewdata services).

The Prestel Adapter is a mains-powered external expansion unit which connects to the RS423 socket on the back of the computer. It contains a self-dialling modem and it is connected directly to your normal telephone socket using a lead and plug supplied. The Adapter also comes with a ROM containing the software necessary to drive the adapter.

The user port

As mentioned above, the 'analogue in' socket can be used for the input of a variety of analogue signals but some devices produce digital signals and these can be read from the 8-bit **User Port**, located on the front underside of the computer.

It is therefore possible to connect a Mouse or a Trackerball and use it to carry out functions such as controlling the movement of the cursor and identifying objects on the screen (using one or more of the available push buttons). However, the real power of the User Port lies in its ability to provide both input and output making it possible to transmit signals to control the operation of external devices such as robot arms and machine tools.

Connecting an IEEE interface

Whilst the 8-bit User Port is often adequate for small control applications, serious scientific applications will require your computer to adhere to the international standard IEEE 488 interface specification and this is provided by means of the Acorn IEEE Interface expansion unit.

The unit is mains powered and is connected to the computer via the '1Mhz bus'. In turn the unit can be connected to a network of up to 14 separate devices, such as oscilloscopes, voltmeters, spectrum analysers and frequency meters.

Connecting a co-processor

In its standard form, the computer is driven by a 65C12 microprocessor but it is possible to fit additional co-processors, some of which are designed merely to expand the power and versatility of the basic machine, others to actually change the nature of the machine so that it can be used for applications requiring specialist facilities.

The 65C102 co-processor

The 65C102 is an internal expansion unit which plugs directly into connectors provided in the computer's printed circuit board.

It contains its own 64K of user memory and, in use, produces significant increases in processing speed, vital for serious computer applications, particularly those involving large-scale use of Assembly Language.

The 6502 second processor

The 6502 second processor is a mains powered external expansion unit which is connected via the 'Tube' connector located on the front underside of the computer.

Like the 65C102 co-processor, it contains its own user memory and increases your computers processing speed and capability.

The 32016 second processor

The 32016 second processor is a mains powered external expansion unit which is connected via the 'Tube'. It contains a NS32016 32-bit processor and up to 1Mb of user RAM, thereby producing a 32-bit microcomputer system suitable for the efficient development and execution of software requiring a large amount of processor power or 32-bit arithmetic. In addition to BBC BASIC, the 32016 second processor comes with four languages, C, FORTRAN 77, Cambridge Lisp and Pascal. An internal, co-processor version is planned for later availability.

Operation of the 32016 second processor is controlled by the PANOS operating system supplied with the unit.

The Z80 second processor

The Z80 second processor is a further mains powered external expansion unit which may be connected to the computer via the 'Tube'. It contains a Z80 processor and its own 64K of user memory which allows you to load the CP/M operating system – probably the most common business-oriented operating system in use – and hence gives access to what is one of the largest business software libraries available.

The usual CP/M utilities are provided with the unit together with a series of software packages:

- BBC BASIC;
- MemoPlan, a CP/M based word-processor;
- FilePlan, a personal database package;
- GraphPlan, a spreadsheet modelling program;
- Accountant, an integrated accounting system;
- CIS COBOL (incorporating the ANIMATOR debugging tool and FORMS2, an aid to the development of interactive CIS COBOL programs);
- Nucleus a system generator which will help you to develop your own CP/M based software.

Further co-processors based on other microprocessors and operating systems are both possible and likely to be made available for your computer. Contact your dealer or supplier for details.

Appendix A

Mode characteristics

Table 1 below gives the text, character set, graphics and colour capability of each of the eight standard screen modes and their corresponding 'shadow' screen modes.

Information relating to the default colour assignments for screen modes 0 – 6 and their corresponding 'shadow' screen modes is given in Table 2. A change of mode always results in the selection of a white foreground and a black background for both text and graphics (if available).

Table 1

Mode	Text rows	Text columns	Character set	Graphics Pixels	Colours
0 (128)	32	80	ASCII	640 × 256	2
1 (129)	32	40	ASCII	320 × 256	4
2 (130)	32	20	ASCII	160 × 256	16
3 (131)	25	80	ASCII	–	2
4 (132)	32	40	ASCII	320 × 256	2
5 (133)	32	20	ASCII	160 × 256	4
6 (134)	25	40	ASCII	–	2
7 (135)	25	40	TELETEXT	(see Appendix B)	

Table 2

Mode	Foreground	Background	Colour
0 (128) 3 (131) 4 (132) 6 (134)	0 1	128 129	Black White
1 (129) 5 (133)	0 1 2 3	128 129 130 131	Black Red Yellow White
2 (130)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143	Black Red Green Yellow Blue Magenta Cyan White * Black/White * Red/Cyan * Green/Magenta * Yellow/Blue * Blue/Yellow * Magenta/Green * Cyan/Red * White/Black

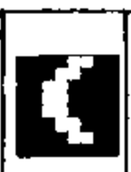
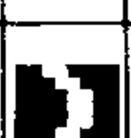
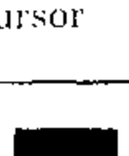
* denotes a flashing colour pair

Appendix B

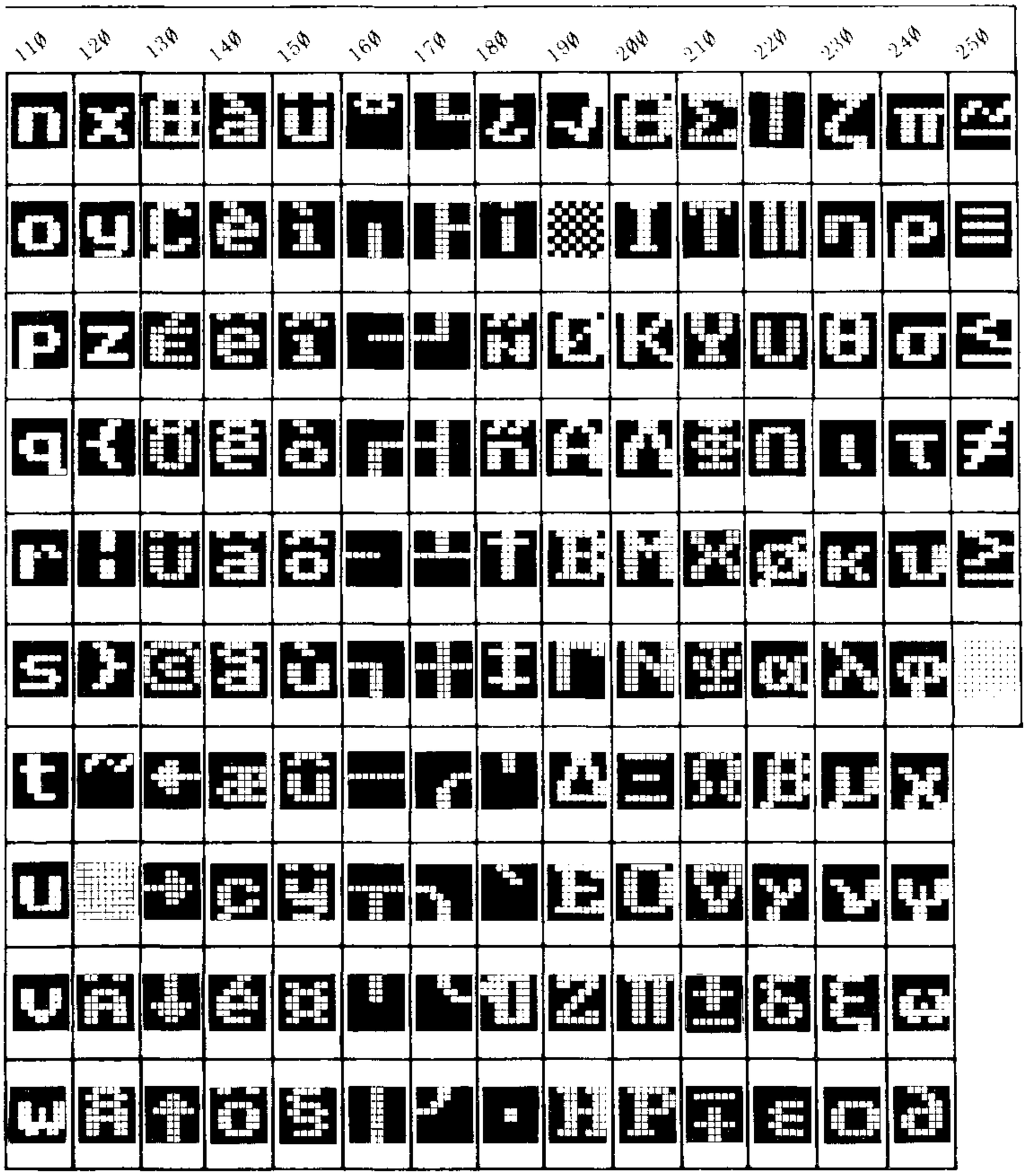
Character Sets

ASCII displayed character set (modes 0 to 6 and 128 to 134)

ASCII codes in the range 0 to 31 are control codes which correspond to the VDU codes described in Appendix G.

	Ø	1Ø	2Ø	3Ø	4Ø	5Ø	6Ø	7Ø	8Ø	9Ø	1ØØ
Ø	Nothing	Down	Default logical colors	Move text cursor to ØØ							
1	Next to printer	Up	Disable VDU	Move text cursor							
2	Start printer	Clear text	Select mode								
3	Stop printer	Start of line	Reprogram characters								
4	Separate cursors	Paged mode	Define graphics area								
5	Join cursors	Scroll mode	Plot								
6	Enable VDU	Clear graphics	Default text/ graphics areas								
7	Beep	Define text color	Nothing								
8	Back	Define graphics color	Define text area								
9	Forward	Define logical colors	Define graphics origin								

Each displayed character consists of eight rows of eight dots.


110
120
130
140
150
160
170
180
190
200
210
220
230
240
250

Teletext displayed alphanumeric character set (modes 7 and 135)

Codes in the range 0 to 31 are control codes which correspond to the VDU codes described in Appendix G.

Codes in the range 128 to 159 are the Teletext control codes which affect subsequent characters on the same line (see page 95). The characters shown in the table below are those displayed under the effect of an alphanumeric control code.

	Ø	1Ø	2Ø	3Ø	4Ø	5Ø	6Ø	7Ø	8Ø	9Ø	1ØØ	11Ø	12Ø
Ø	Nothing	Down	Nothing	Move cursor to ØØ									
1	Next to printer	Up	Disable VDU	Move cursor									
2	Start printer	Clear screen	Select mode										
3	Stop printer	Start of line	Reprogram characters										
4	Nothing	Paged mode	Nothing										
5	Nothing	Scroll mode	Nothing										
6	Enable VDU	Nothing	Nothing										
7	Beep	Nothing	Nothing										Back space and delete
8	Back	Nothing	Nothing										Nothing
9	Forward	Nothing	Nothing										Alpha red

Each code produces a unique character. Thus VDU 78 or PRINT CHR$(78) would display an N since column 7Ø, row 8 shows an N.

130	140	150	160	170	180	190	200	210	220	230	240	250
Alpha green	Normal * height	Graphic cyan										
Alpha yellow	Double * height	Graphic white										
Alpha blue	Nothing	Conceal display										
Alpha magenta	Nothing	Contiguous graphics *										
Alpha cyan	Nothing	Separated graphics										
Alpha * white	Graphic red	Nothing										
Flash	Graphic green	Black * background										
Steady *	Graphic yellow	New background										
Nothing	Graphic blue	Hold graphics										
Nothing	Graphic magenta	Release * graphics										

* every line starts with these options

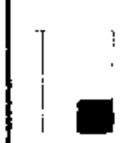
Teletext displayed graphics characters

Codes in the range 0 to 31 are control codes which correspond to the VDU codes described in Appendix G.

Codes in the range 128 to 159 are the Teletext control codes which affect subsequent characters in the same line (see page 95). The characters shown in the table below are those displayed under the effect of a graphics control code.

	Ø	1Ø	2Ø	3Ø	4Ø	5Ø	6Ø	7Ø	8Ø	9Ø	1ØØ	11Ø	12Ø
Ø	Nothing	Down	Nothing	Move cursor to ØØ									
1	Next to printer	Up	Disable VDU	Move cursor									
2	Start printer	Clear screen	Select mode										
3	Stop printer	Start of line	Reprogram characters										
4	Nothing	Paged mode	Nothing										
5	Nothing	Scroll mode	Nothing										
6	Enable VDU	Nothing	Nothing										
7	Beep	Nothing	Nothing										Back space and delete
8	Back	Nothing	Nothing										Nothing
9	Forward	Nothing	Nothing										Alpha red

Each character has a code. Thus H is code 72 since it is in column 7Ø row 2.

130	140	150	160	170	180	190	200	210	220	230	240	250
Alpha green	Normal * height	Graphic cyan										
Alpha yellow	Double * height	Graphic white										
Alpha blue	Nothing	Conceal display										
Alpha magenta	Nothing	Contiguous graphics *										
Alpha cyan	Nothing	Separated graphics										
Alpha * white	Graphic red	Nothing										Back space and delete
Flash	Graphic green	Black * background										
Steady *	Graphic yellow	New background										
Nothing	Graphic blue	Hold graphics										
Nothing	Graphic magenta	Release graphics *										

* every line starts with these options

Keyboard codes

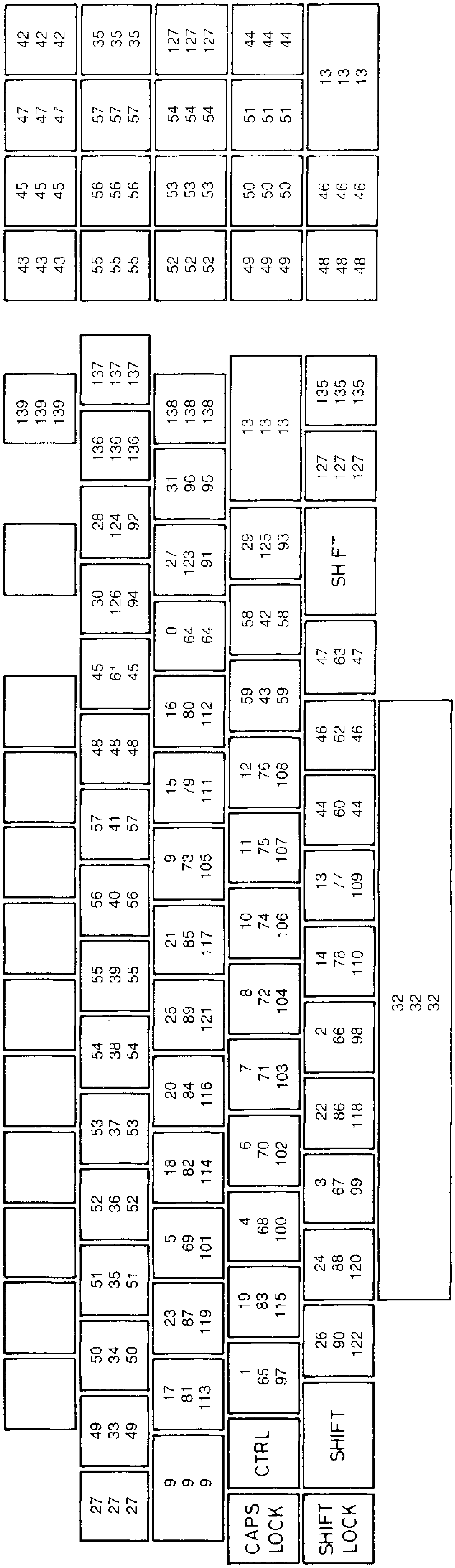
The code produced by each key on the keyboard depends upon the settings of *caps lock*, *shift lock* and the depression of [SHIFT] or [CTRL]. For each key in the diagram below:

- the lower number is the lower case code;
- the middle number is the upper case code;
- the top number is the code generated if the key is depressed in conjunction with [CTRL].

The codes generated by the 10 function keys can be specified by the user (see Appendix D).

The cursor editing keys produce codes only if enabled with *FX4 (see Appendix D).

All numbers are given in decimal.


139
27 27 27
49 33 49
50 34 50
51 35 51
52 36 52
53 37 53
54 38 54
55 39 55
56 40 56
57 41 57
48 48 48
45 61 45
30 126 94
28 124 92
136 136 136
137 137 137
9 9 9
17 81 113
23 87 119
5 69 101
18 82 114
20 84 116
25 89 121
21 85 117
9 73 105
15 79 111
16 80 112
0 64 64
27 123 91
31 96 95
138 138 138
CAPS LOCK
CTRL
1 65 97
19 83 115
4 68 100
6 70 102
7 71 103
8 72 104
10 74 106
11 75 107
12 76 108
59 43 59
58 42 58
29 125 93
13 13 13
SHIFT LOCK
SHIFT
26 90 122
24 88 120
3 67 99
22 86 118
2 66 98
14 78 110
13 77 109
44 60 44
46 62 46
47 63 47
SHIFT
127 127 127
135 135 135
32 32 32
43 43 43
45 45 45
47 47 47
42 42 42
55 55 55
56 56 56
57 57 57
35 35 35
52 52 52
53 53 53
54 54 54
127 127 127
49 49 49
50 50 50
51 51 51
44 44 44
48 48 48
46 46 46
13 13 13

Appendix C

Operating system commands

Operating system commands provide a means of communicating your requirements to the MOS. The commands described below are of a general nature; summaries of the commands applicable to the various filing systems are contained in Appendix E.

Operating system commands may be issued directly from the keyboard (in which case they are terminated by depression of RETURN) or incorporated in programs.

Most commands may be abbreviated to their first few characters terminated by a . – where applicable, the minimum abbreviation for each command is given in brackets after each command name.

Commands marked □ apply to facilities required only by advanced users and full details can be found in the Reference Manual.

*CODE

□ Provide a means of executing machine code routines which are already in memory as if they were part of the MOS.

*CONFIGURE (*CO.)

Provide a direct means of altering the settings held in the CMOS RAM (i.e. without using the Control Panel utility described on page 23). *CONFIGURE takes one or two parameters, the first being the name of the setting to be changed; the second (if necessary) being the value to be stored in the CMOS RAM. In the parameter list below, *n* and *m* denote decimal numbers; *x* denotes a number in hexadecimal notation.

□ BAUD *n* — Change the RS423 transmit/receive rate setting according to *n*.

BOOT — Reverses the actions of BREAK and SHIFT+BREAK.

CAPS — Set CAPS LOCK option.

□ DATA *n* — Change the RS423 data format setting according to *n*.

DELAY *n* — Change the keyboard auto-repeat delay setting to *n* hundredths of a second.

□	DIR	Initialise ADFS with selected directory.
	EXTUBE	Use an external second processor (if fitted).
□	FDRIVE *n*	Configure the disc controller for different types of disc unit according to *n*.
	FILE *x*	Change the default filing system setting to that contained in ROM socket *x*.
	FLOPPY	Cause ADFS to access the floppy disc unit when initialised.
	HARD	Cause ADFS to access the Winchester disc unit when initialised.
	IGNORE *n*	Change the 'printer ignore character' to ASCII *n*. If *n* is omitted, all characters are sent to the printer.
	INTUBE	Use an internal co-processor (if fitted).
	LANG *x*	Change the default language setting to that contained in ROM socket *x*.
	LOUD	Change the volume setting for the BELL sound to full.
	MODE *n*	Change the display mode setting to *n* (0–7 or 128–135).
	NOBOOT	Assigns normal function to [BREAK] and [SHIFT]+[BREAK].
	NOCAPS	Reset the CAPS LOCK option.
□	NODIR	Initialise ADFS without a directory selected.
	NOSCROLL	Enables the scroll protect option.
	NOTUBE	Ignore both internal and external co-processors.
	PRINT *n*	Change the printer type setting according to *n* (see page 177).
	QUIET	Change the volume setting for the BELL sound to half volume.
	REPEAT *n*	Change the keyboard auto-repeat rate setting to *n* hundredths of a second.
	SCROLL	Disables the scroll protect option.
	SHCAPS	Set the [SHIFT]+[CAPS LOCK] option.

TUBE — Use a second (co-) processor (if available).

TV *n*,*m* — Set the vertical screen alignment and interlace option settings (as *TV below).

When used without a parameter, *CONFIGURE displays a list of the above options.

*GO — □ Execute a machine code program in a single processor system or in the language processor of a system equipped with an internal or external second processor.

*GOIO — □ Execute a machine code program in the I/O processor of a system equiped with a second processor (i.e. across the Tube).

*HELP (*H.) — Display brief information about the MOS, the languages and the filing systems currently resident (but not necessarily selected) in the machine.

*IGNORE (*IG.) — Set the 'printer ignore character' (see page177).For example:

*IGNORE 10 causes line feed characters (ASCII 10) to be ignored by the printer.

*INSERT (*INS.) — □ It is possible to make the MOS ignore the presence of a given ROM using the *UNPLUG command described below. *INSERT returns the ROM in a specified socket to its normal status, but only after a hard break or subsequent power-on.

*KEY (*K.) — Assign a sequence of characters to a specified function key (see page 14). For example:

*KEY 0 LIST¦M assigns the BASIC LIST command to function key 0.

*LINE — □ Acts in a similar manner to *CODE (above) except that it provides a means of passing a line of text (i.e. the remainder of the command) to the routine.

*MOTOR (*M.)

Controls the cassette recorder motor:

*MOTOR Ø switches the cassette motor off;
*MOTOR 1 switches the cassette motor on.

*MOVE □

Copy files from one filing system to another, for example:

```
*MOVE -DISC-ADDRESS -TAPE-BACKUP
```

copies file ADDRESS from (DFS) disc to cassette, using the name BACKUP.

*ROMS

Display a list of the ROMs currently present in each of the 16 ROM sockets. The listing gives the socket number (in hexadecimal), the ROM name and an indication of whether the ROM is currently available (see *UNPLUG below).

*SHADOW (*SH.)

Switch to either the main memory or the shadow memory:

*SHADOW Ø causes the shadow memory to be selected on subsequent mode changes (even if the mode selected is in the range 0 – 7);
*SHADOW 1 causes main or shadow memory to be selected according to the mode number (i.e. modes 0 – 7 select main memory, modes 128 – 135 select shadow memory).

*SHOW

Display the sequence of characters currently assigned to a specified function key. For example:

*SHOW Ø displays the sequence of characters currently assigned to function key 0.

*SHUT

Close all currently open files known to the MOS, regardless of the current filing system.

*SRDATA □

Reserve a designated area of sideways RAM for use with data.

*SRREAD □

Copy a designated area of sideways RAM to normal RAM.

*SRROM □ Reserve a designated area of sideways RAM for use with absolute addressing.

*SRWRITE □ Copy a designated area of normal RAM to sideways RAM

*STATUS (*ST.) Display the current content of the various settings held in the CMOS RAM. *STATUS may also be used with one of the parameters described under *CONFIGURE (above), in which case only the specified setting is displayed. For example:

*STATUS	displays the setting of the keyboard
DELAY	auto-repeat delay held in the CMOS RAM;
*STATUS	displays all the settings held by the CMOS RAM.

*TIME (*TI.) Display the current day, date and time from the CMOS RAM.

*TV Two parameters are used, the first indicating an adjustment to the vertical screen alignment; the second defining the setting of the interlace option. For example:

*TV0,1	causes no vertical screen adjustment but sets the interlace off;
*TV1,0	causes the screen to be shifted up one line and sets the interlace on;
*TV255,1	causes the screen to be shifted down one line and sets the interlace off.

*UNPLUG (*UNP.) □ Cause the MOS to ignore the ROM in a specified socket after the next hard break or power-on.

Appendix D

*FX commands

A proportion of the memory reserved for use by the MOS is given over to the storage of information relating to the current state of the machine and how it is to react in various circumstances. This information is directly accessible to, and may be changed by the user by means of special operating system calls (often referred to simply as **OSBYTE** calls).

Some OSBYTE calls have an equivalent *FX command which may be issued directly from the keyboard or included in, say, a BASIC program and these commands are summarised below. Commands marked □ provide access to facilities required only by advanced users and full details can be found in the Reference Manual.

Apparent gaps in the sequence relate to OSBYTE calls which do not have an equivalent *FX command and which must therefore be implemented by means of techniques beyond the scope of this guide.

Parameters in *FX commands may be separated by a comma (as in the examples below) or a sequence of one or more spaces.

If a parameter is omitted it is assumed to be zero.

`*FX0` Display MOS version.

`*FX1` □ Reserved for application programs.

`*FX2` Select input stream:

`*FX2,0` keyboard only (disables RS423input);
`*FX2,1` RS423 input only;
`*FX2,2` keyboard input and buffered RS423 input.

`*FX3` Select output stream:

`*FX3,0` Printer and screen only;
`*FX3,1` Printer, screen and RS423;
`*FX3,2` Printer only;
`*FX3,3` Printer and RS423;
`*FX3,4` Screen only;
`*FX3,5` Screen and RS423;
`*FX3,6` none;
`*FX3,7` RS423 only.

□ Other values may also be used.

*FX4 Enable/disable cursor editing:

*FX4,0 enable cursor editing;
*FX4,1 disable cursor editing and assign ASCII codes:

COPY	135
←	136
→	137
↓	138
↑	139

*FX4,2 disable cursor editing and assign soft key numbers:

COPY	11
←	12
→	13
↓	14
↑	15

*FX5 Select printer type (see page 177):

*FX5,0 selects printer sink (no printing);
*FX5,1 selects parallel printer;
*FX5,2 selects serial printer;
□ *FX5,3 selects user printer routine;
*FX5,4 selects network printer server.

Printer types higher than 4 should not be used.

The default setting can be set using *CONFIGURE PRINT (see page 192).

*FX6 Select printer ignore character (equivalent to *IGNORE). For example:

*FX6,10 prevents line feeds (ASCII 10) being sent to the printer.

*FX7 □ Select RS423 receive rate.

*FX8 □ Select RS423 transmit rate (see page 177).

*FX9 Set flash rate of first colour in fiftieths of a second (default setting 25).

*FX9,0 disables flashing and forces the first colour on the screen;
*FX9,10 sets the flash rate to one fifth of a second.

*FX10 Set flash rate of second colour in fiftieths of a second (default setting 25).

*FX10,0 disables flashing and forces the second colour on the screen;
*FX10,5 sets the flash rate to one tenth of a second.

*FX11 Set keyboard auto-repeat delay in hundredths of a second (default setting 32 or as set by *CONFIGURE DELAY).

*FX11,0 disables auto-repeat;
*FX11,10 sets auto-repeat delay to one tenth of a second.

*FX12 Set keyboard auto-repeat period in hundredths of a second (default setting 8 or as set by *CONFIGURE REPEAT).

*FX12,0 restores default settings of auto-repeat delay and auto-repeat period.
*FX12,3 sets auto-repeat period to three hundredths of a second.

*FX13 □ Disable various events.

*FX14 □ Enable various events.

*FX15 Flush buffers:

*FX15,0 flushes all buffers;
*FX15,1 flushes current input buffer.

*FX16 □ Select number of ADC channels.

*FX17 □ Select next ADC channel to be sampled.

*FX18 Clear user-defined function key definitions.

*FX19 □ Wait for vertical synchronisation.

*FX20 Restore default font definitions i.e. reset the characters corresponding to ASCII codes 32 to 126 to normal.

*FX21 Flush selected buffer:

*FX21,0 keyboard buffer;
*FX21,1 RS423 input buffer;
*FX21,2 RS423 output buffer;
*FX21,3 printer buffer;
*FX21,4 sound channel 0;
*FX21,5 sound channel 1;
AK9 *FX21,6 sound channel 2;
*FX21,7 sound channel 3.

*FX22 [ ] Increment ROM polling semaphore.

*FX23 [ ] Decrement ROM Polling semaphore.

*FX25 Restore a group of font definitions:

*FX25,0 restore character codes between 32 and 255;
*FX25,1 restore character codes between 32 and 63;
*FX25,2 restore character codes between 64 and 95;
*FX25,3 restore character codes between 96 and 127;
*FX25,4 restore character codes between 128 and159;
*FX25,5 restore character codes between 160 and 191;
*FX25,6 restore character codes between 192 and 223;
*FX25,7 restore character codes between 224 and 255.

*FX107 Select internal or external 1MHz bus:

*FX107,0 selects external bus;
*FX107,1 selects internal bus.

*FX108 Switch main / shadow memory into main map:

*FX108,0 switches main memory into main map (immediate);
*FX108,1 switches shadow memory into main map (immediate).

*FX109 [ ] Make temporary filing system permanent.

*FX112 Select memory to which characters will be written until the next mode change:

*FX112,0 writes to memory specified by the mode change;
*FX112,1 writes to main memory (immediate);
*FX112,2 writes to shadow memory (immediate).

*FX113 Select memory to be displayed until the next mode change:

*FX113,0 displays the memory specified by the mode change;
*FX113,1 displays main memory (immediate);
*FX113,2 displays shadow memory (immediate).

*FX114 Select main / shadow memory in subsequent mode changes (equivalent to *SHADOW):

*FX114,0 forces selection of shadow memory;
*FX114,1 selects main/shadow memory according to mode number.

*FX118 [ ] Reflect keyboard status in keyboard LEDs.

*FX119 [ ] Close any *SPOOL / *SPOOLON / *EXEC files.

*FX120 □ Write 'keys pressed' information.

*FX124 □ Acknowledge 'escape condition' without side effects.

*FX125 □ Set 'escape condition'.

*FX126 □ Acknowledge 'escape condition' with side effects.

*FX136 □ Define entry point for user MOS routine (equivalent to *CODE).

*FX137 Switch cassette relay (equivalent to *MOTOR):

*FX137,0 switch relay OFF;
*FX137,1 switch relay ON.

*FX138 Insert character code into buffer. (See *FX21 for a list of buffer numbers.) For example:

*FX138,0,65 places ASCII 65 (A) into the keyboard buffer.

*FX139 □ Select option value (equivalent to *OPT).

*FX140 Select cassette filing system and baud rate (equivalent to *TAPE}):

*FX140,3 sets the transfer rate to 300 baud;
*FX140,12 sets the transfer rate to 1200 baud.

*FX141 Select ROM filing system (equivalent to *ROM).

*FX142 □ Enter language ROM.

*FX143 □ Issue paged ROM service request.

*FX144 Set vertical screen shift and interlace option for next mode change or break (equivalent to *TV). For example:

*FX144,0,1 gives no screen shift and turns the interlace off;
*FX144,1,0 shifts the screen up by one line and turns the interlace on;
*FX144,255 shifts the screen down by one line (and turns the interlace on).

*FX146–*FX151 □ Access memory-mapped I/O areas.

*FX153 □ Insert character code into buffer, checking for ESCAPE.

*FX154 □ Write to Video ULA control register.

*FX155 □ Write to Video ULA palette register.

*FX156 □ Write to 6850 ACIA control register.

*FX157 □ Write byte across Tube.

*FX162 □ Write to CMOS RAM.

*FX178 □ Disable keyboard scanning.

*FX179 □ Write ROM polling semaphore.

*FX180 □ Write Operating System High Water Mark (OSHWM).

*FX181 □ Write RS423 mode.

*FX183 □ Write cassette/ROM filing system switch.

*FX190 □ Set ADC resolution.

*FX191 □ Write RS423 use flag.

*FX193 □ Write flash counter.

*FX194 □ Write mark period count.

*FX195 □ Write space period count.

*FX196 □ Write keyboard auto-repeat delay.

*FX197 □ Write keyboard auto-repeat period.

*FX198 □ Write *EXEC file handle.

*FX199 □ Write *SPOOL file handle.

*FX200 Set BREAK and ESCAPE effect according to *n*:

*FX200,0 set normal BREAK and ESCAPE action;
*FX200,1 set normal BREAK and disable ESCAPE;
*FX200,2 clear memory on BREAK and set normal ESCAPE action;
*FX200,3 clear memory on BREAK and disable ESCAPE.

*FX201 □ Write keyboard disable.

*FX202 □ Write keyboard status byte.

*FX203 □ Write RS423 handshake extent.

*FX204 □ Write RS423 input suppression flag.

*FX205 □ Write cassette/RS423 selection flag.

*FX206- □ Used by Econet.
*FX208

*FX210 Write sound suppression status:

*FX210,0 enables sound output;
*FX210,1 disables sound output.

*FX211 Write BELL (CTRL+G) channel (default setting 3). For example:

*FX211,0 selects channel 0.

*FX212 Write BELL (CTRL+G) sound information (default setting 144). For example:

*FX212,200 produces a 'softer' BELL sound.

*FX213 Write BELL (CTRL+G) frequency (default setting 101). For example:

*FX213,200 produces a high-pitched BELL sound.

*FX214 Write BELL (CTRL+G) duration (default setting 7). For example:

*FX214,1 produces a very short BELL sound;
*FX214,255 produces an indefinite BELL sound.

*FX215 □ Write start-up message suppression and !BOOT option status.

*FX216 □ Write length of soft key string.

*FX217 □ Write number of lines printed since last page halt.

*FX218 □ Write number of items in VDU queue.

*FX219 Write character value returned by TAB (default setting 9, i.e. cursor right). For example:

*FX219,127 makes TAB equivalent to DELETE.

*FX220 Write ESCAPE character (default setting 27). For example:

*FX220,32 makes [SPACE BAR] the ESCAPE key.

*FX221-*FX224 □ Write input buffer code interpretation status.

*FX225 Write function key status:

*FX225,0 disables the function keys;
*FX225,1 gives the keys their normal function of generating strings;

The function keys may also be set to generate a single ASCII code using *FX225,*n* (where *n* is the base number in the range 2 – 255). This has the effect of assigning ASCII *n* to f0, ASCII *n*+1 to f1, ASCII *n*+2 to f2 etc. So, for example:

*FX225,65 causes f0 to generate ASCII 65 (A), f1 to generate ASCII 66 (B), f2 to generate ASCII 67 (C) etc.

*FX226 Set base number for [SHIFT]+function key depressions (default setting 128). For example:

*FX226,97 causes [SHIFT]+[f0] to generate ASCII 97 (a), [SHIFT]+[f1] to generate ASCII 98 (b) etc.

*FX227 Set base number for [CTRL]+function key depressions (default setting 144). For example:

*FX227,48 causes [CTRL]+[f0] to generate ASCII 48 (0), [CTRL]+[f1] to generate ASCII 49 (1) etc.

*FX228 Set base number for [SHIFT]+[CTRL]+function key depressions (default setting: no effect). For example:

*FX228,200 causes [SHIFT]+[CTRL]+[f0] to generate ASCII 200, [SHIFT]+[CTRL]+[f1] to generate ASCII 201 etc.

*FX229 Write ESCAPE key status:

*FX229,0 gives [ESCAPE] its normal function;
*FX229,1 causes [ESCAPE] (or the key selected by *FX220) to generate its ASCII code.

*FX230 □ Write flags determining ESCAPE effects.

*FX231 □ Write IRQ bit mask for user 6522.

*FX232 □ Write IRQ bit mask for 6850 (RS423).

*FX233 □ Write interrupt bit mask for system 6522.

*FX236 □ Write character destination status.

*FX237 □ Write cursor editing status.

*FX238 Set base number for numeric keypad (default setting 48 for keypad 0).

*FX241 Not used

*FX244 □ Write soft key consistency flag.

*FX245 □ Write printer destination flag.

*FX246 □ Write printer ignore character.

*FX247- □ Intercept BREAK vector.
*FX249

*FX254 Set effect of [SHIFT] on numeric keypad:

*FX254,0 causes [SHIFT] to operate (i.e. [SHIFT]+keypad 0 generates !);
*FX254,1 makes [SHIFT] have no effect.

*FX255 □ Write start-up options.

Appendix E

Filing system commands

Listed below are the commands available under the various filing systems. In reality, many of the commands are handled by the MOS but, for the sake of completeness, such commands are listed (and duplicated) under each filing system heading.

Most commands may be abbreviated to their first few characters terminated by a **.** - where applicable, the minimum abbreviation for each command is given in brackets after each command name.

Commands marked □ apply to facilities required only by advanced users; full details can be found in the Reference Manual.

The Cassette Filing System

***BUILD (*BU.)** Create a cassette file containing lines of text typed in by the user. Each line must be terminated by RETURN; the end of input is indicated by pressing ESCAPE. For example:

`*BUILD START` creates a file called START containing lines subsequently typed by the user.

***CAT (*.)** Display a catalogue (i.e a list of filenames plus other information) of the current cassette.

***CLOSE (*CL.)** Close all currently open cassette files.

***DUMP (*D.)** Produce a hexadecimal dump of the named cassette file. For example:

`*DUMP MYFILE` produces a dump of file MYFILE.

***EX** □ Similar to *CAT (above) except that it provides additional information about each file.

***EXEC (*E.)** Cause the MOS to take input from the named cassette file rather that the keyboard. For example:

`*EXEC START` causes the MOS to take input from file START.

*LIST (*LI.) □ Display the content of the named cassette file in GSREAD format with line numbers.

*LOAD (*L.) □ Load the named cassette file into memory.

*OPT1 (*O.1) Adjust the level of output during file operations:

*OPT1,Ø suppresses output of all information;
*OPT1,1 allows output of the filename, block count and length;
*OPT1,2 allows output of the filename, block count, length, load address and execution address.

*OPT2 (*O.2) Set the action to be taken in the event of an error during loading:

*OPT2,Ø ignores all errors;
*OPT2,1 prompts the user to rewind the tape;
*OPT2,2 aborts the load and issues an error message.

*OPT3 (*Ø.3) □ Set the length of the inter-block gap when writing sequential files.

*PRINT (*P.) □ Display a pure ASCII dump of the named cassette file.

*RUN Load and execute the named machine code program from cassette. For example:

```
*RUN PANEL
```

*SAVE (*S.) □ Save a block of memory to a named cassette file.

*SPOOL (*SP.) When used with a filename, *SPOOL causes all subsequent output to the screen to be written to the named file; *SPOOL on its own closes the current *SPOOL file. For example:

*SPOOL LISTING opens file LISTING and directs all subsequent output to it;
*SPOOL closes the current *SPOOL file.

*SRLOAD □ Load the specified file into a designated area of sideways RAM.

*SRSAVE □ Save a designated area of sideways RAM to the specified file.

*TYPE (*TY.) ␣ Display the content of the named cassette file in GSREAD format without line numbers.

The ROM Filing System

The commands listed below operate in exactly the same manner as described under the Cassette Filing System.

CAT (.)
*CLOSE (*CL.)
*DUMP (*D.)
*EX
*EXEC (*E.)
*LIST (*LI.)
*LOAD (*L.)
*OPT1 (*O.1)
*PRINT (*P.)
*SRLOAD
*RUN
*TYPE (*TY.)

The Disc Filing System

Unless stated otherwise, all commands operate on the current drive and the current directory (see page 155).

Certain commands allow the use of a wildcard facility, in which the character * may be used to denote a sequence of any characters and the character # may be used to note any single character.

*ACCESS (*AC.) Disc files may be 'locked' to prevent accidental erasure. *ACCESS locks or unlocks the named file, for example:

*ACCESS MYPROG L locks file MYPROG;
*ACCESS MYPROG unlocks file MYPROG.

*APPEND (*AP.) Extend files created using *BUILD by the addition of extra lines. For example:

*APPEND START adds subsequent lines of input to file START.

*BACKUP (*BAC.) Make a copy of a complete disc surface.

*BACKUP takes two parameters, the first being the 'source drive'; the second the 'destination drive'. For example:

*BACKUP Ø 1 copies the contents of drive 0 onto the disc in drive 1 (assuming two drives are available);

*BACKUP Ø Ø copies the contents of a disc onto another using a single drive. The user is prompted to exchange the source and destination discs.

*BUILD (*BU.) Create a disc file using subsequent lines of input as described under the Cassette Filing System.

CAT (.) Display a catalogue of the current drive.

*CLOSE (*CL.) Close all currently-open disc files.

*COMPACT (*COM.) Reorganise the files stored on the specified drive so that spaces created by file deletions (see *DELETE) are amalgamated into one block. For example:

*COMPACT Ø compacts drive 0.

*COPY Copy a file from one drive to another. For example:

*COPY Ø 1 LETTER copies file LETTER on drive 0 to drive 1.

*CREATE □ Reserve space for a file.

*DELETE (*DE.) Delete the name of the specifed file from the current drive's catalogue. (The space occupied by the file contents is not reallocated until the disc is compacted using *COMPACT). For example:

*DELETE OLDPROG deletes the name OLDPROG from the current drive's catalogue.

*DESTROY (*DES.) Delete a group of files in a single operation using the 'wildcard' facility. For example:

DESTROY OLD deletes all files whose names begin with the characters OLD.

*DIR Set the current directory to the character specified, For example:

*DIR W sets the current directory to W;
*DIR :1.B selects drive 1 and assigns directory B.

*DRIVE (*DR.)

Set the current drive to the number specified, for example:

*DRIVE 1 selects drive number 1.

*DUMP (*DU.)

Display a hexadecimal dump of the named disc file as described under the Cassette Filing System.

*ENABLE (*EN.)

As a security measure, the *BACKUP, *DESTROY and *WIPE commands normally produce the prompt:

Go ? (Y/N)

before any action is taken. Issuing a *ENABLE command immediately prior to these commands suppresses the prompt.

*EX □

Display information about the files held in the specified directory.

*EXEC (*E.)

Cause the MOS to take subsequent input from the named disc file rather than from the keyboard, as described under the Cassette Filing System.

*FORM (*FO.)

Format a blank disc for use with the Disc Filing System. The first parameter determines whether the disc is to be formatted as a 40-track or an 80-track disc; the remaining parameters are a list of drive numbers. For example:

*FORM 40 0 formats the disc in drive 0 as a 40-track disc;
*FORM 80 0 2 formats the discs in drives 0 and 2 as 80-track discs.

Each track is verified after formatting (see *VERIFY below).

*FREE (*FR.)

Display information about the free space available on the specified drive. For example:

*FREE 1 displays free space information for drive 1;
*FREE displays free space information for the current drive.

*INFO (*I.) □

As for *EX except that information may be obtained for single files (or groups of files, using the 'wildcard' option).

*LIST (*LI.) □ Display the content of the named file in GSREAD format with line numbers.

*LIB Set the library directory (see page 158).

*LOAD (*L.) □ Load the named disc file into memory.

*MAP Display the free-space map for the specified drive, for example:

*MAP 3 displays the free space map for drive 3;
*MAP displays the free-space map for the current drive.

*OPT1 (*O.1) Set the level of reporting during file operations:

*OPT1,0 supresses all messages;
*OPT1,1 allows output of the filename, load address, execution address, lengthand track/sector location on the disc.

*OPT4 (*O.4) Set the effect of the auto-boot option:

*OPT4,0 switches the auto-boot option off;
*OPT4,1 *LOADs !BOOT into memory;
*OPT4,2 *RUNs !BOOT;
*OPT4,3 *EXECs !BOOT.

*PRINT (*P.) □ Display a pure ASCII dump of the named disc file.

*REMOVE (*RE.) Equivalent in effect to *DELETE, except that the Not found message is suppressed if the named file cannot be located.

*RENAME (*REN.) Change the name of a disc file. For example:

```
*RENAME NEWPROG OLDPROG
```

Changes the name of file NEWPROG to OLDPROG.

*RUN Load and Execute the named machine code program. For example:

```
*RUN PANEL
```

Under the Disc Filing System, a command like that above may be abbreviated to:

```
*PANEL
```

*SAVE (*S.) □ Save a block of memory to the named disc file.

*SPOOL (*SP.)

Cause all subsequent output to the screen to be written to the named disc file, as described under the Cassette Filing System.

*SPOOLON

Extend an existing *SPOOLfile. For example:

*SPOOLON LISTING appends all subsequent output to the screen to file LISTING.

The file is closed with *SPOOL as described under the Cassette Filing System.

*SRLOAD □ Load the specified file into a designated area of sideways RAM.

*SRSAVE □ Save a designated area of sideeways RAM to the specified file.

*TITLE (*TIT.)

Set the disc title for the current drive to the specified sequence of characters. For example:

*TITLE BASIC-PROGS sets the disc title to BASIC-PROGS.

*TYPE (*TY.) □ Display the content of the named disc file in GSREAD format without line numbers.

*VERIFY (*V.)

Check the formatting of each sector on the specified drive. For example:

*VERIFY Ø verifies the formatting of the disc in drive 0

*WIPE (*W.)

*DELETE files specified using the 'wildcard' facility – the user must respond with Y (to delete) or N (to retain) each file.

The Advanced Disc Filing System

Unless stated otherwise all commands operate on the currently selected directory.

Many Advanced Disc Filing System commands allow the use of the wildcard facility described above.

*ACCESS (*AC.)

Set the attributes associated with files. The attributes are:

E for 'execute-only' access (for machine-code program files only);
L for locking a file;
W for write access;
R for read access.

For example:

*ACCESS MEMO L — locks file MEMO;
*ACCESS DADSPROG WR — assigns read/write access to file DADSPROG.

*APPEND (*AP.)

Extend files created using *BUILD (below) as described under the Disc Filing System.

*BACK

Instruct the Advanced Disc Filing System to select the previously-accessed directory and make it the current directory.

*BUILD (*BU.)

Create a disc file containing subsequent lines of input as described under the Cassette Filing System.

*BYE

Ensure that all currently open files are closed at the end of a session (similar in effect to *CLOSE).

CAT (.)

Display the filenames in the current or a specified directory. For example:

*CAT — displays the catalogue for the current directory;
*CAT $.DAVE — displays the catalogue for directory DAVE (which is subordinate to the root directory.

*CDIR (*CD.)

Create a subordinate directory with the specified name in the current diectory. For example:

*CDIR MARY — creates a directory called MARY in the current directory.

*CLOSE (*CL.)

Close all currently-open disc files.

*COMPACT (*COM.)

Reorganise the files in the directory heirarchy so that spaces created by file deletions are amalgamated into larger blocks.

□ It is possible to define the area of memory to be used during a *COMPACT.

*COPY

Copy a file from one directory to another. For example:

```
*COPY $.TEXT $.BACKUP.VIEW
```

copies the file TEXT (in the root directory) into directory $.BACKUP.VIEW i.e. it creates a file whose full pathname is $.BACKUP.VIEW.TEXT

*CREATE □ Reserve space for a file.

*DELETE (*DE.)

Delete the name of the specified file. For example:

*DELETE IDEA	deletes file IDEA from the current directory;
*DELETE $.DAVE.B1	deletes file B1 from directory DAVE (itself in the root directory).

A directory may be deleted only if it is empty.

*DESTROY (*DES.) Delete a group of files(using the wildcard facility).

*DIR Set the current directory (see page 161).

*DISMOUNT (*DISM.) Ensure that all currently-open files are closed prior to changing a disc.

*DUMP (*DU.) □ Display a hexadecimal dump of the named file, as described under the Cassette Filing System.

*EX □ Display information about the files contained in the specified directory.

*EXEC (*E.) Cause the MOS to take subsequent input from the named disc file rather than the keyboard as described under the Cassette Filing System.

*FREE (*FR.) Display the free space map.

*INFO (*I.) □ Display information about a single file (or a group of files) using the 'wildcard' option.

*LCAT (*LC.) Display a catalogue of the library directory.

*LEX □ Display information about the files held in the library directory.

*LIB		Set the library to the specified drive and directory. For example: *LIB $.UTILITIES
*LIST (*LI.)	□	Display the named disc file in GSREAD format with line numbers.
*LOAD (*L.)	□	Load the specified file into memory.
*MAP		Display the free-space map.
*MOUNT (*MOU.)		Initialise a disc drive – commonly used to switch between more than one drive. *MOUNT Ø initialises drive 0. Note that *MOUNT 0 is equivalent to *DIR :0
*OPT1 (*O.1)		Set the reporting level during file operations as described under the Disc Filing System.
*OPT4 (*O.4)		Set the operation of the auto-boot option, as described under the Disc Filing System.
*PRINT (*P.)		Display a pure ASCII dump of the named file.
*REMOVE (*RE.)		Equivalent in effect to *DELETE, except that the Not found message is suppressed if the file cannot be located.
*RENAME (*REN.)		Change the name of a disc file. For example: *RENAME PROG1 PROG2 changes the name of file PROG1 to PROG2 (in the current directory). *RENAME can also be used to physically move (rather than copy) a file from one directory to another. For example: *RENAME $.BASIC.THIS $.GARBAGE.THAT moves file THIS from directory $.BASIC to directory $.GARBAGE and renames it THAT.
*RUN		Load and Execute a machine code program as described under the Disc Filing System.
*SAVE (*S.)	□	Save a block of memory to the named disc file.

*SPOOL(*SP.)		Cause all subsequent output to the screen to be written to the named disc file as described under the Cassette Filing System.
*SPOOLON		Append all subsequent output to the screen to the named disc file as described under the Disc Filing System.
*SRLOAD	□	Load the specified file into a designated area of sideways RAM.
*SRSAVE	□	Save a designated area of sideways RAM to the specified file.
*TITLE (*TIT.)		Set the title for the current directory. For example: *TITLE WOBLETS Note that a directory title is distinct from a directory name.
*TYPE (*TY.)	□	Display the content of the named disc file in GSREAD format without line numbers.

The following Advanced Disc Filing System utilities are provided on the Welcome disc. They may be accessed via the menu system, by typing:

```
*ADFS RETURN
CHAIN"UTILITIES" RETURN
```

and selecting the ADFS utilities option. The menu system also provides help regarding both the syntax and operation of each utility.

Alternatively, each utility may be executed individually as required. Those utilities whose names begin with * are machine code programs; the remainder are BASIC programs which must be executed using either LOAD and RUN or CHAIN.

*AFORM	Formats a floppy disc in ADFS format.
*BACKUP	Copies the contents of one disc onto another.
CATALL	A BASIC program which produces a listing of the contents of all the directories on a disc.
COPYFILES	A BASIC program which copies files from one filing system to another.
DIRCOPY	A BASIC program which copies the entire contents of a specified directory (and all its subordinate directories) to another directory.

EXALL	A BASIC program which displays information similar to that produced by *EX for all the directories on a disc.
HARDERROR	A BASIC program which enables permanent floppy disc errors to be ignored by the filing system.
RECOVER	A BASIC program which enables a file, or part of a file to be recovered in the event of accidental erasure.
*VERIFY	Verifies the formatting on a disc.

Appendix F

BASIC keywords

Each BASIC keyword is described briefly below. If an abbreviated form of the keyword is allowed, it is given in brackets after the full version. Note that the abbreviation for some keywords includes an opening bracket; for example LE. is equivalent to LEFT$(and not just LEFT$.

Many of the keywords are explained in more detail in Chapter 2. Keywords marked □ provide access to facilities beyond the scope of this guide and users should consult the Reference Manual for further information.

ABS		Function giving the positive value of any number.
ACS		Function giving the arc-cosine, in radians, of any number from −1 to 1 inclusive.
ADVAL (AD.)	□	Read data from the analogue port or buffers.
AND (A.)		Used as a logical or bitwise operator.
ASC		Function producing the ASCII code of the first character in a string.
ASN		Function giving the arc-sine, in radians, of any number from −1 to 1 inclusive.
ATN		Function giving the arc-tangent, in radians, of any number.
AUTO (AU.)		Command to give automatic line-numbering.
BGET# (B.#)	□	Give the code of the next character in a file.
BPUT# (BP.#)	□	Write the code of a character to a file.
CALL (CA.)		Execute a machine-code routine.
CHAIN (CH.)		Load and run a BASIC program.
CHR$		Function producing the character with the given ASCII code.
CLEAR (CL.)		Clear the memory of all program variables, except the resident integer variables.

CLG	Clear the graphics window to the current graphics background colour.
CLOSE# (CLO.#)	Close an open file.
CLS	Clear the text window to the current text background colour.
COLOR or COLOUR (C.)	Set the text foreground or background colours.
COS	Function giving the cosine of any angle, the angle being in radians.
COUNT	Variable containing the number of characters printed since the last new line.
DATA (D.)	Used in conjunction with READ to specify data items to be used in a program.
DEF	Define a function or procedure.
DEG	Function which converts from radians to degrees.
DELETE (DEL.)	Delete a number of lines from a program.
DIM	Reserve memory space for an array of given size.
DIV	Carry out integer division, any remainder being discarded.
DRAW (DR.)	Draw a line from the last graphics point specified to the given point.
EDIT	Call the text editor and convert the BASIC program into a text file if there is sufficient room in memory.
ELSE	Part of the extended IF...THEN statement used when an alternative decision may be required.
END	The computer executes no further statements after it reaches the END statement. Its use is optional if the END statement is physically the last statement in the program.
ENDPROC (E.)	Indicate the end of a procedure definition.
ENVELOPE (ENV.) □	Define a sound envelope.
EOF#	Function indicating whether the end of a file has been reached.

EOR	Used as a logical or bitwise exclusive OR.
ERL	Give the line number where the last error occurred.
ERR	Give the error number of the last error.
ERROR (ERR.)	Part of the ON...ERROR statement.
EVAL (EV.)	Function which evaluates a string as if it were a BASIC calculation.
EXP	Function which calculates e (which is 2.7183..) raised to the given power.
EXT#	Function which controls the length (extent) of an open file – note that this has no relevance for cassette files.
FALSE (FA.)	Function returning the value 0. Used in logical expressions.
FN	Used in the definition of a function or a call to that function.
FOR (F.)	Start of the FOR...NEXT loop which causes the computer to repeatedly execute the statements between the FOR and the NEXT..
GCOL (GC.)	Set the graphics colour to be used by future graphics commands, and determine the way the colour interacts with the colour of any point in the same position on the screen.
GET	Wait for a key to be pressed and produce the ASCII code for that key.
GET$	Wait for a key to be pressed and produce the character for that key.
GOSUB (GOS.)	Execute a subroutine then return control to the statement following the GOSUB call. GOSUB is a more limited predecessor of DEFPROC, and does not allow the passing of parameters. It is includeed for compatability with the BASIC language on other computers.
GOTO (G.)	Jump to the given line number.

HIMEM (H.)	□	Variable used to indicate the highest free memory location which can be used by the current program. HIMEM can be reset by the user so as to protect a portion of memory above HIMEM where data has been stored.
IF		Part of the IF...THEN statement. The computer only executes the instruction after THEN if the condition following IF is true.
INKEY	□	Wait for a given time for a key depression, and produce the ASCII code for that key. The time is expressed in hundredths of a second.
INKEY$ (INK.)	□	Wait for a given time for a key depression, amd produce the charActer for that key. The time is expressed in hundredths of a second.
INPUT (I.)		Wait for an input or inputs from the keyboard terminated by RETURN.
INPUT LINE		Accepts, from the keyboard, a single input containing leading or trailing spaces or commas, terminated by RETURN.
INPUT# (I.#)		Input data from an open file and store the data in the variables following the INPUT# statement.
INSTR (INS.)		Search one string for occurrences of another string, and give the character position where the matching string begins.
INT		Function which converts a decimal number into the nearest integer smaller than the original number.
LEFT$((LE.)		Extract the left part of a string.
LEN		Function which gives the length of a string.
LET		Set a variable to a given value. The use of LET is optional in BBC BASIC.
LIST (L.)		List the current program. LISTO sets the indentation options to make the program easier toread. LIST IF is used to list all lines containing a particular character sequence.
LN		Function which gives the natural logarithm of a number.
LOAD (LO.)		Load a BASIC program.

LOCAL (LOC.)	Declare the variables that follow as local only to that procedure or function. Thus they will not interfere with similarly named variables elsewhere in the program. Parameters passed to a procedure are automatically local.
LOG	Function which gives the logarithm of a number to base 10.
LOMEM (LOM.) □	Variable used to indicate the lowest free memory location which can be used to store the value of variables used by the program. LOMEM can be reset by the user.
MID$((M.)	Extract a substring from a longer string.
MOD	Give the integer remainder after a division.
MODE (MO.)	Change the display mode. Mode cannot be changed within a procedure or function.
MOVE	Move the graphics cursor invisibly to the given position.
NEW	Remove the current program. It can be retrieved using OLD.
NEXT (N.)	Part of the FOR...NEXT loop indicating the end of the statements which are to be repeatedly executed.
NOT	Used as a logical or bitwise operator.
OFF	Part of the ON ERROR OFF statement which switches error trapping off and enables the computer to again print its standard error messages and halt the program.
OLD (O.)	Retrieve a program after a NEW or after BREAK has been pressed. If typing OLD gives a 'Bad Program' message after the BREAK key has been pressed, the program has been corrupted and must be loaded again from tape or disc.
ON ERROR	Used to control the action taken by the computer if it encounters an error in the program.
ON...GOTO or ON...GOSUB	The value of the variable following ON is found. If its value is 1, the computer jumps to the first line number in the list following the GOTO/GOSUB statement; if the value is 2, it jumps to the second line number, and so on.

ON...PROC		Used to give a multi-branching facility and enable one of a series of procedures to be executed.
OPENIN (OP.)		Open a file for input only.
OPENOUT (OPENO.)		Open a file for output only.
OPENUP		Open a file for updating (input and output). This is not possible with cassette files.
OR		Used as a logical or bitwise operator.
OSCLI	□	Used to pass a string to the operating system.
PAGE (PA.)	□	Variable used by the computer to indicate the memory location at which storage of the program begins. PAGE can be reset by the user, so with care it is possible to have several programs in the computer memory at the same time.
PI		Give the value of *pi* (3.141592653) for use in calculations.
PLOT (PL.)		Carry out a plotting function according to the parameters following the PLOT command (see the full list of PLOT codes in Appendix H).
POINT((PO.)		Give the logical colour number at a particular graphics point.
POS		Give the current x coordinate of the text cursor.
PRINT (P.)		Print characters to the screen. The format of printing is affected by the use of ; , ' and the printing of numbers is controlled by the value of the integer variable @%.
PRINT# (P.#)		Print the variable values following PRINT# to an open file.
PROC		Define or call a procedure.
PTR#		Function which gives the position within a file where the next characters will be read or written. The user can change the value of PTR# and can thus read or write anywhere within the file, allowing random access to records. This is only possible on disc, and has no relevance for cassette files.
RAD		Function which converts an angle from degrees to radians.

READ		Read items from a DATA statement.
REM		A remark to help document the program. REMs are ignored by the computer on execution of the program.
RENUMBER (REN.)		Assign default (or specified) line numbers to a BASIC program.
REPEAT (REP.)		Part of the REPEAT...UNTIL loop which executes all statements between REPEAT and UNTIL until a condition or conditions are satisfied. Note that such a loop is always executed at least once, even if the terminating conditons are met immediately, as the test for the conditions comes at the end of the loop.
REPORT (REPO.)		Print an error message for the most recent error found.
RESTORE (RES.)		Read further data beginning at the line number following the RESTORE.
RETURN (R.)		Indicate the end of a subroutine which has been called using GOSUB. The computer returns to the statement in the program which is immediately after the GOSUB which called the routine.
RIGHT$((RI.)		Extract the right-hand part of a string from a longer string.
RND	□	Function which produces a random number. RND(1) gives a random decimal from 0 to 0.99999. RND(N) gives a random integer from 1 to N inclusive.
RUN		Execute the program in memory.
SAVE (SA.)		Save a program in the computer's memory to cassette or disc.
SGN		Function which gives the sign of the number following, producing −1 for minus numbers, 0 for zero and +1 for positive numbers.
SIN		Function which gives the sine of any angle, the angle being in radians.
SOUND (SO.)		Produce a sound through the internal loudspeaker.
SPC		Used only with PRINT or INPUT to print multiple spaces.

SQR		Function which finds the square root of the number that follows.
STEP		Part of the FOR...TO..STEP statement which allows a FOR...NEXT loop with steps other than 1.
STOP		Interrupt a program with the untrappable error message STOP.
STR$		Converts a number into its equivalent string representation.
STRING$ (STRI.)		Produce multiple copies of a string up to a maximum length of 255 characters.
TAB		Used only with PRINT or INPUT to position the text cursor on the screen.
TAN (T.)		Function which gives the tangent of any angle, the angle being in radians.
THEN		Part of the IF...THEN statement.
TIME (TI.)		Set or read the value of one of the internal clocks in hundredths of a second.
TIME$		Set or read the real-time clock.
TO		Part of the FOR...TO...NEXT statement.
TOP	□	Variable giving the first free memory location after the end of the BASIC program. TOP is usually the same as LOMEM, but unlike LOMEM it cannot be reset by the user.
TRACE (TR.)		Display the line number of each line executed. Used for tracing errors. TRACE OFF switches trace off, TRACE ON switches it on.
TRUE		Function producing the value −1, used in logical expressions.
UNTIL (U.)		Part of the REPEAT...UNTIL loop, signalling the end of the loop. Statements between REPEAT and UNTIL are executed repeatedly until certain conditions are met.
USR	□	Function providing a means of calling a machine code routine designed to produce one value.

VAL	Function which converts a string into its numeric equivalent. The string is examined up to the first non-numeric character, so a string not beginning with a number is given a value of 0.
VDU (V.)	A general purpose command producing various effects on the screen display.
VPOS (VP.)	Give the current y coordinate of the text cursor.
WIDTH (W.)	Set the width of all subsequent lines of output.

Appendix G

VDU codes

The output of text and graphics is controlled by a complex set of MOS routines referred to as the **VDU driver**. The VDU driver is active unless the display screen has been disabled using *FX3 (see page 196) or VDU 21 (see below).

The codes described below alter the the behaviour of the VDU driver and may be used to produce a variety of effects. The most common implementation is through the BASIC language's VDU statement although commands to the VDU driver may also be issued directly from the keyboard by means of *control key depressions* (i.e. simultaneous depression of CTRL with another key).

Some VDU codes consist of a sequence of values. Where necessary, these extra values must be specified for the code to take effect.

Code	CTRL key	Extra values	Effect
VDU 0	@	0	Does nothing.
VDU 1	A	1	Send the next character to the printer only. For example: VDU 1,65 prints but does not display the character A.
VDU 2	B	0	Enable the printer.
VDU 3	C	0	Disable the printer.
VDU 4	D	0	Write text at text cursor (i.e. restore the text cursor and display subsequent text in normal character positions).
VDU 5	E	0	Write text at graphics cursor (i.e. remove the text cursor and display subsequent text at graphics co-ordinates). The position of the text cursor remains unaltered.
VDU 6	F	0	Re-enable screen output (i.e. enable the VDU driver).
VDU 7	G	0	Emit a bleep from the speaker.

VDU 8	H	0	Move the text cursor one character position to the left.
VDU 9	I	0	Move the text cursor one character position to the right.
VDU 10	J	0	Move the text cursor down one line.
VDU 11	K	0	Move the text cursor up one line.
VDU 12	L	0	Clear the screen and restore the text cursor to position (0,0). (Equivalent to CLS).
VDU 13	M	0	Move the text cursor to the start of the current line.
VDU 14	N	0	Set page mode on (i.e. suspend output at the end of each full screen of output and wait for the user to depress SHIFT).
VDU 15	O	0	Set page mode off (i.e. allow unrestricted output).
VDU 16	P	0	Clear the current graphics area to the current graphics background colour. (Equivalent to CLG).
VDU 17	Q	1	Change the foreground or background colour for subsequent text output (equivalent to COLOUR). In mode 5 (133), for example: VDU 17,2 sets the text foreground colour to Yellow. VDU 17,129 sets the text background colour to Red.
VDU 18	R	2	Change the foreground or background colours for subsequent graphics output and define the way in which it is to be placed on the screen (equivalent to GCOL). In mode 2 (130), for example: VDU 18,0,4 changes the graphics foreground colour to Blue. VDU 18,0,134 changes the graphics background colour to Cyan.

VDU 19	S	5	Change the colour palette. VDU 19 allows any of the 16 available colours to be assigned to the colour numbers available in a particular mode. In mode 0 (128) for example: `VDU 19,1,2,0,0,0` changes colour 1 (normally White) to Green. `VDU 19,0,7,0,0,0` changes colour 0 (normally Black) to White. The three items at the end of this sequence should always be 0.
VDU 20	T	0	Restore default colours (i.e. revert to white text / graphics on a black background) and reset the palette to its default colour assignments.
VDU 21	U	0	Disable the VDU driver (i.e stop subsequent output to the screen). Note that CTRL + U issued from the keyboard has the effect of deleting the current line.
VDU 22	V	1	Select screen mode. This sequence should not be used from the keyboard in languages such as BASIC or from the command screens of either View or ViewSheet. See the Reference Manual for further information.
VDU 23	W	9	Miscellaneous functions. VDU 23 provides a great many functions, most of which are beyond the scope of this guide. The functions are listed below – details of the remaining parameters are given in the Reference Manual. `VDU 23,0` control 6845 directly `VDU 23,1` change cursor `VDU 23,2`, `VDU 23,3`, `VDU 23,4`, `VDU 23,5` set full pattern-fill patterns `VDU 23,6` set dotted line pattern `VDU 23,7` scroll window directly `VDU 23,8` clear block in text window

			VDU 23,9 } set flash rate VDU 23,10 }
			VDU 23,11 restore default pattern-fills
			VDU 23,12 } VDU 23,13 } set simple pattern-fill VDU 23,14 } pattern VDU 23,15 }
			VDU 23,16 control cursor movement
			Functions 17 to 31 are reserved.
			Any value greater than 31 following VDU 23 is taken as a reference to a character which is to be redefined (see page 91).
VDU 24	X	8	Define graphics window (see page 58).
VDU 25	Y	5	Equivalent to the BASIC PLOT statement (See Appendix H.)
VDU 26	Z	0	Restore text and graphics windows.
VDU 27	[	0	Does nothing. Note that [CTRL]+[is equivalent to [ESCAPE].
VDU 28	\	4	Define text window (see page 57).
VDU 29	|	4	Define graphics origin (i.e. the position on the screen with graphics co-ordinates (0,0). For example: `VDU 29,640;512;` makes subsequent graphics co-ordinates relative to (640,512) –a point roughly in the centre of the screen. Note the (mandatory) use of semi-colons.
VDU 30	^	0	Move text cursor to (0,0).
VDU 31	_	2	Move text cursor to a specified position (equivalent to PRINT TAB): For example: `VDU 31,20,10` moves the text cursor to character position 20 on line 10 (the first character position and line being 0).
VDU127		0	Backspace and delete (i.e. the normal action of [DELETE]).

Appendix H

PLOT codes

The BASIC PLOT statement can be summarised as:

PLOT *code*,x,y

and its effect is to plot to the point (x,y) in a manner determined by the value of *code*. An identical effect can be produced using:

`VDU 25,` *code*`,x;y;` (note the use of semi-colons).

The permissible PLOT codes and their effects are given (in groups of eight codes) in Table 1. The codes within each group are obtained by adding an 'offset' value to the first code in the group. The offset values are as follows:

0	move relative to the previous point;
1	plot relative to the previous point in the current graphics foreground colour;
2	plot relative to the previous point in the logical inverse colour;
3	plot relative to the previous point in the current graphics background colour;
4	move to absolute position;
5	plot to absolute position in the current graphics foreground colour
6	plot to absolute position in thhe logical inverse colour;
7	plot to absolute position in the current graphics background colour.

The column headed *Previous points* contains the number of points which must have been 'visited' before the corresponding PLOT statement is executed. For example, in order to plot a rectangle, one corner must be first be visited (perhaps using MOVE or DRAW) – the co-ordinates of the diametrically opposite corner are specified in the PLOT statement.

Examples of various PLOT commands are given in Chapter 2, and detailed information can be found in the Reference Manual.

Table 1

Plot code	Effect	Previous points
0 – 7	Solid line, includes both ends	1
8 – 15	Solid line, final point omitted	1
16 – 23	Dot–dash line, includes both ends, pattern restarted	1
24 – 31	Dot–dash line, final point omitted, pattern restarted	1
32 – 39	Solid line, first point omitted	1
40 – 47	Solid line, both points omitted	1
48 – 55	Dot–dash line, first point omitted, pattern continued	1
56 – 63	Dot–dash line, both ends omitted, pattern continued	1
64 – 71	Point plot	
72 – 79	Line fill left and right to non–background	
80 – 87	Triangle fill 2	
88 – 95	Line fill right to background	
96 – 103	Rectangle fill	1
104 – 111	Line fill left and right to foreground	
112 – 119	Parallelogram fill	2
120 – 127	Line fill right to non–foreground	
128 – 135	Flood until non–background	
136 – 143	Flood until foreground	
144 – 151	Circle outline	1
152 – 159	Circle fill	1
160 – 167	Circular arc	2
168 – 175	Circular segment	2
176 – 183	Circular sector	2
184 – 191	Rectangle copy/move:	
	184 Move relative	2
	185 Relative rectangle move	2
	186/187 Relative rectangle copy	2
	188 Move absolute	2
	189 Absolute rectangle move	2
	190/191 Absolute rectangle copy	2
192 – 199	Ellipse outline	2
200 – 207	Ellipse fill	2
208 – 255	Reserved	

Appendix I

VIEW Commands

Command screen commands

Most commands may be abbreviated to their first few characters. Where applicable, the minimum abbreviation is given in brackets after each command name.

Commands marked □ apply to facilities for which detailed descriptions are outside the scope of this guide. Full details may be found in the VIEW User Guide.

Note that operating system and filing system commands can be issued from the VIEW command screen.

CHANGE (C)		Find all occurrences of one target string and change it for another. For example: `CHANGE WATER WINE`
CLEAR (CL)		Remove all markers from the text.
COUNT (CO)		Count the number of words in memory or between markers (if specified).
EDIT (E)	□	Start editing a file which is too large to fit into available memory (disc only).
FINISH (F)	□	Finish an EDIT session.
FIELD *n* (FI)	□	Assign the tab function to the key with ASCII value *n*. (Default setting FIELD 9.)
FOLD	□	Turn the facility to ignore case on and off with SEARCH, CHANGE and REPLACE. With no parameters, FOLD tells you the current status.
FORMAT (FOR)		Format the whole document in memory.
LOAD (L)		Load the specified file into memory replacing what was there previously (disc only). For example: `LOAD RESIGN`
MICROSPACE (MI)	□	Enable microspacing.

MODE (M)

Switch the computer into the specified screen mode. For example:

```
MODE 132
```

MORE (MO) □

Continue an editing session.

NAME (N)

Name (or rename) the file in memory. For example:

```
NAME JULY12
```

NEW

Clear the text from memory.

PRINT (P)

Print text onto continuous stationery. PRINT (P) by itself prints the text in memory; with filenames it prints the contents of those files.

PRINTER (PRINTE) □

Load the specified printer driver into memory. For example:

```
PRINTER EPSON
```

QUIT □

Abandon an EDIT session.

READ (RE)

Read a file onto the end of the document in memory. May be used to read a file into a document at a point indicated by a marker. For example:

`READ INDEX 1` reads file INDEX into the current document at the point indicated by marker 1.

REPLACE (R)

Find all occurrences of one string and request the user to confirm replacement with another. For example:

```
REPLACE FAT ROTUND
```

SAVE (SA)

Save the text in memory with the specified name. For example:

`SAVE MY-CV` saves the current file with the name MY-CV;
`SAVE` saves the current file with its current name.

SCREEN (SC)

Display the text on the screen as it will appear when printed. For example:

`SCREEN LETTER` displays the file LETTER;
`SCREEN` displays the file in memory.

SEARCH (S)

Search the text for the specified string. For example:

SEARCH dog

[CTRL]+[f1] (NEXT MATCH) to find subsequent occurrences.

SHEETS (SH)

Print the text pausing between pages for the user to feed in the next sheet of paper. For example:

SHEETS BOOK prints file BOOK;
SHEETS prints the file in memory.

SETUP (SET)

Set any or all of the text screen flags, for example:

SETUP FI selects formatting and insertion, but not justification.

WRITE (W)

Write text to disc or cassette using the specified filename. This is slower than SAVE but can be used with markers, for example:

WRITE PORTION 1 2 saves the section of the current document between markers 1 and 2.

Stored commands

These commands are used in the text screen and are placed in the command margin by using [SHIFT]+[f8] (EDIT COMMAND).

CE *text* Centre *text* between the left and right margins.

RJ *text* Right justify *text*, i.e. aligns it to the right margin.

LJ *text* Left justify *text*, i.e. aligns it to the left margin.

DH □ Define page header.

DF □ Define page footer.

HE ON/OFF □ Switch printing of page headings on or off.

FO ON/OFF □ Switch printing of page footers on or off.

DM *m* □ Define the start of macro *m*.

EM □ End macro definition.

SR *l v* □ Set register *l* to value *v*.

PB ON/OFF		Switch page breaks on or off (default ON).
PL *n*		Set page length to *n* lines (default 66).
TM *n*		Set top margin to *n* lines (default 4).
HM *n*		Set header margin to *n* lines (default 4).
FM *n*		Set footer margin to *n* lines (default 4).
BM *n*		Set bottom margin to *n* lines (default 4).
LM *n*		Set left margin for printed output to *n* spaces (default 0).
LS *n*		Set line spacing – causes *n* blank lines to be printed between each line of text.
TS ON/OFF	□	Switch two-sided printing on or off.
PE		Page eject. PE n may be used to perform a page eject if n is greater than the number of lines remaining on the current page.
OP	□	Odd page i.e. give one page eject if on an even numbered page, two if on an odd numbered page.
EP	□	Even page i.e. give one page eject if on an odd numbered page, two if on an even numbered page.
HT -/* *n*	□	Set highlight character to *n*.

Appendix J

ViewSheet Commands

Most commands may be abbreviated to their first few characters. Where applicable, the minimum abbreviation is given in brackets after each command name.

Commands marked □ apply to facilities for which detailed descriptions are outside the scope of this guide. Full details may be found in the ViewSheet User Guide.

Note that operating system and filing system commands can be issued from the ViewSheet command screen.

`CREATE (CR)` □ Create a disc file for use by READ and WRITE within the sheet.

`HEADINGS (H)` □ Indicate if user-defined headings are set.

`HEADINGS OFF` `(H OFF)` □ Switch off user-defined headings.

`HEADINGS ON` `(H ON)` □ Switch on user-defined headings.

`LOAD (L)` Load the specified file into memory, replacing what was there previously. For example:

```
LOAD BUDGET
```

`LW` Load the specified file of window definitions.

`MODE` *n* `(M)` Set the screen mode specified in n. For example:

```
MODE 131
```

`NAME (NA)` Assign the specified name to the sheet in memory. For example:

```
NAME WAGES2
```

`NEW` Create a blank worksheet.

`PC` Print out the contents of every occupied slot, with coordinates.

`PRINT (P)` Print out the sheet in memory.

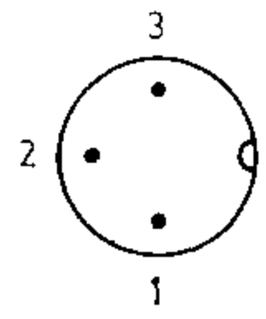
PRINTER (PRINTE)	Load the specified printer driver into memory. For example: PRINTER EPSON
PROTECT (PRO)	Indicate if protection is enabled or disabled.
PROTECT OFF (PRO OF)	Disable protection.
PROTECT ON (PRO ON)	Enable protection.
SAVE (SA)	Save the current sheet under the specified filename. For example: SAVE SUMS saves the current sheet with the name SUMS; SAVE saves the current sheet with its current name.
SCREEN (SC)	Display the sheet in memory with current print windows.
SW	Save a file of window definitions.

Appendix K

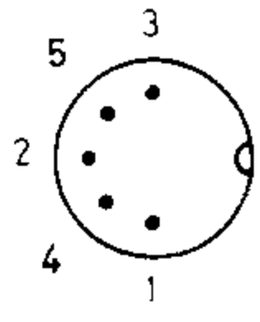
Technical information

Tape recorder leads

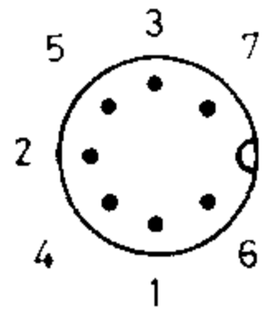
A variety of different leads may be used to connect a tape recorder to the computer. In the diagrams below, the numbered connections refer to DIN plug pins, viewed towards the plug's solder terminals:



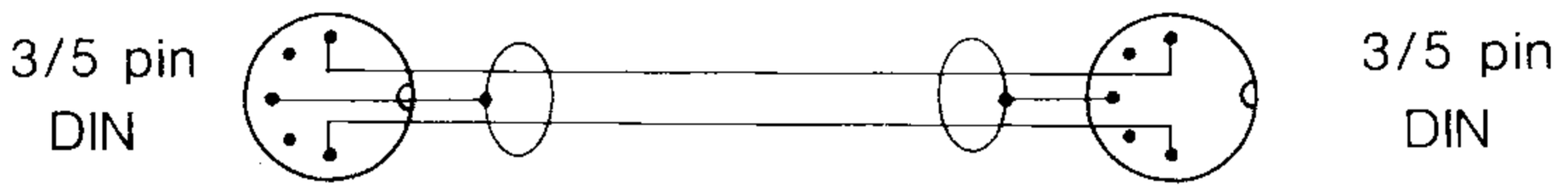
3 pin DIN

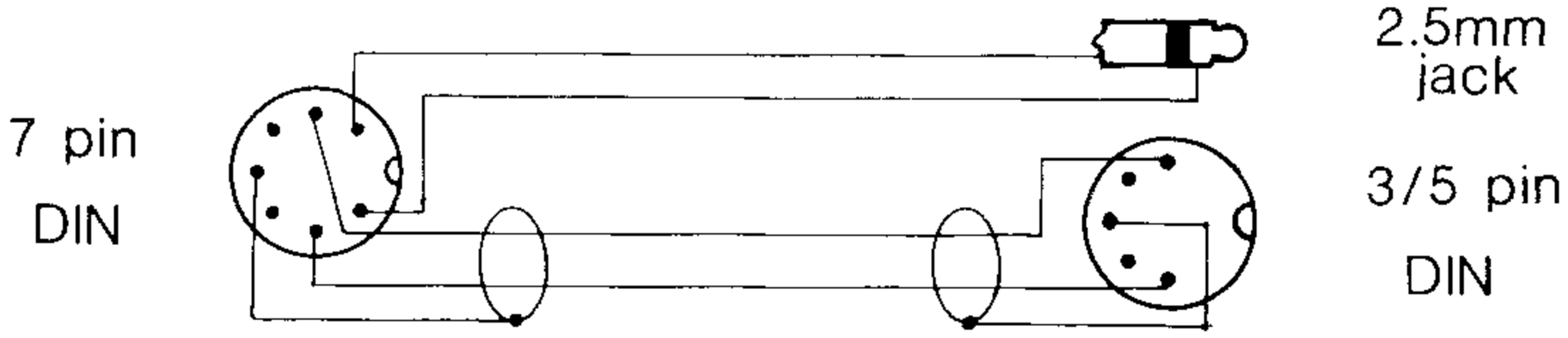


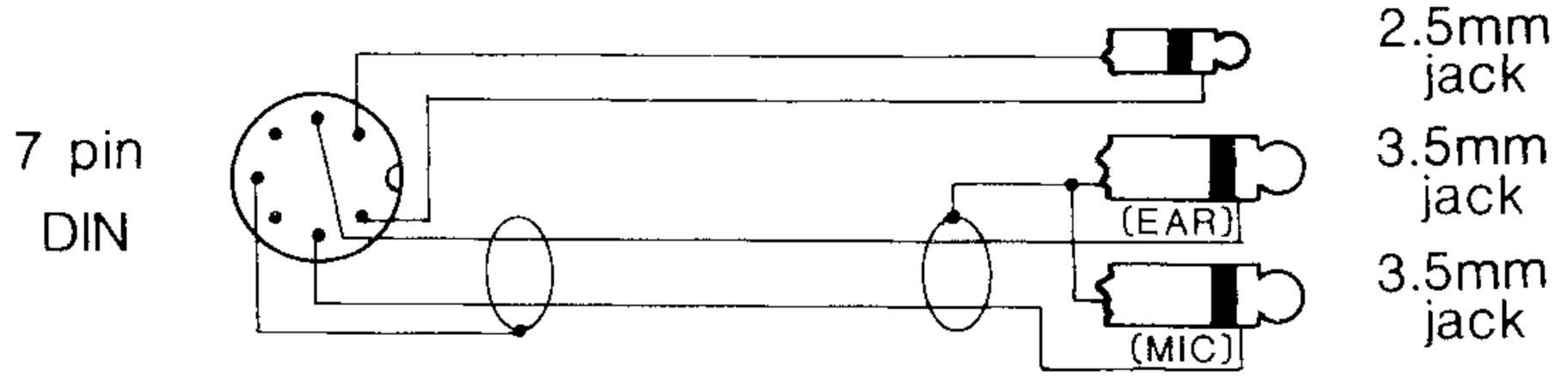
5 pin DIN



7 pin DIN



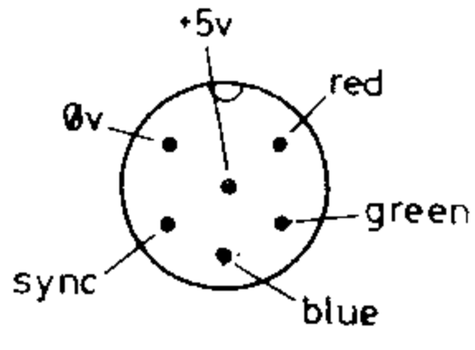


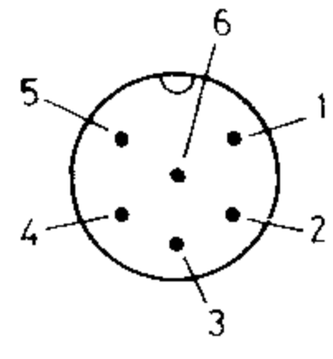


Connector pin assignments

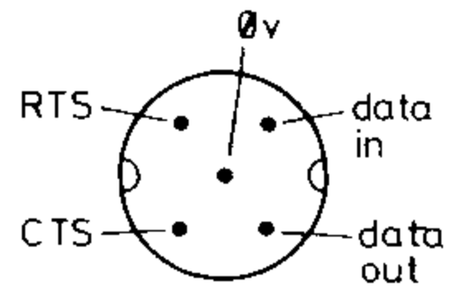
The pin assignments for the connections on the rear of the computer are shown in the diagrams below. Each view is towards the socket, from outside the computer's case.

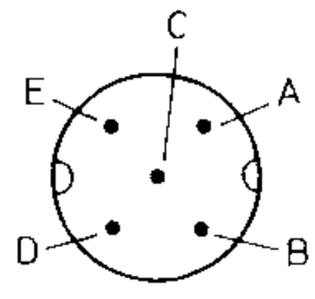
RGB



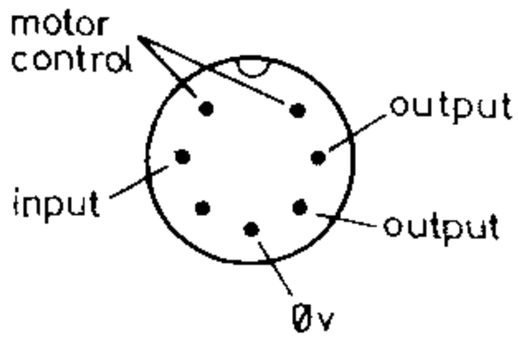


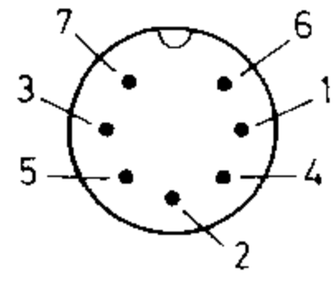
RS423



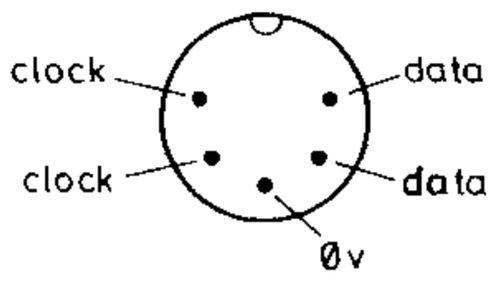


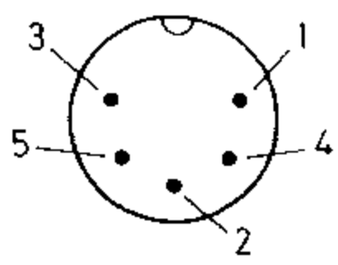
CASSETTE



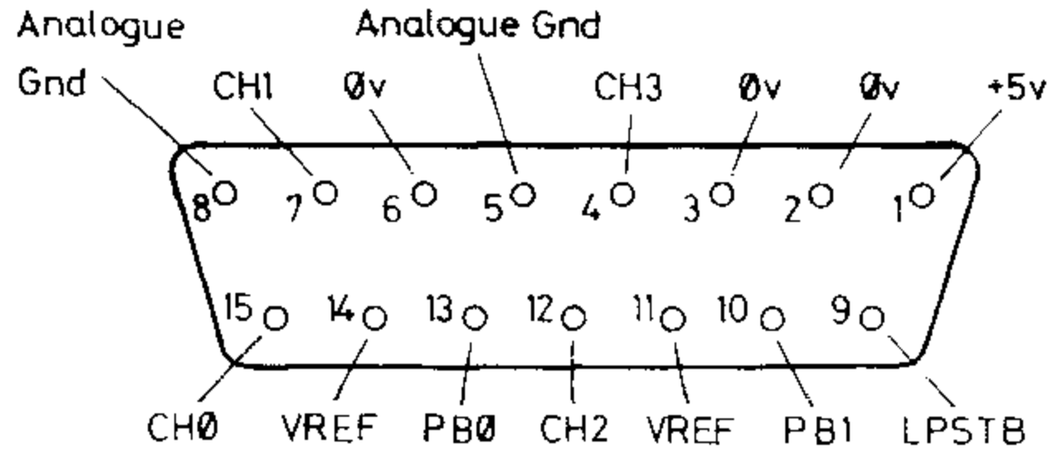


ECONET

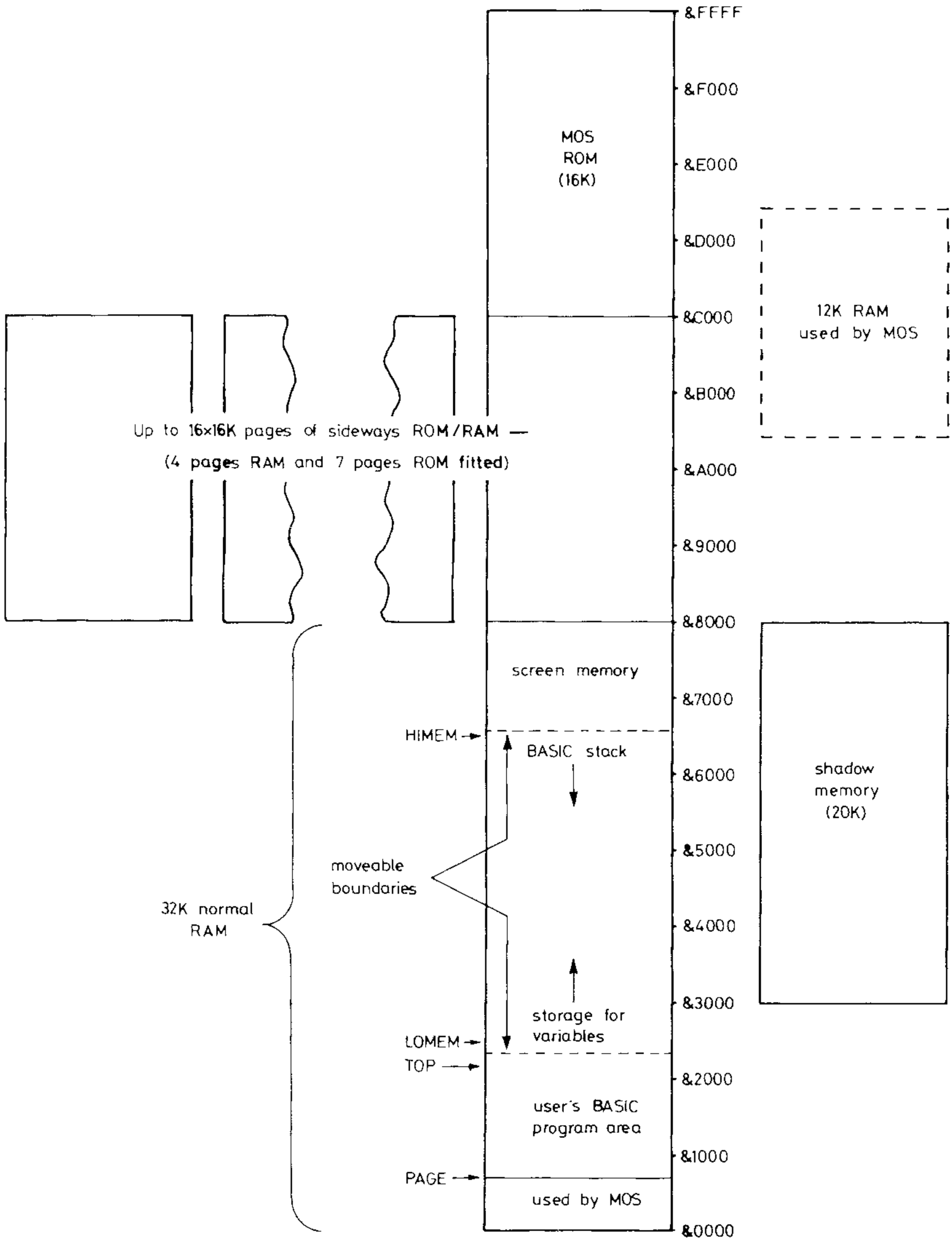




ANALOGUE IN



Memory map


&FFFF
&F000
&E000
&D000
&C000
&B000
&A000
&9000
&8000
&7000
&6000
&5000
&4000
&3000
&2000
&1000
&0000
MOS ROM (16K)
12K RAM used by MOS
Up to 16x16K pages of sideways ROM/RAM —
(4 pages RAM and 7 pages ROM fitted)
screen memory
shadow memory (20K)
HIMEM
BASIC stack
moveable boundaries
32K normal RAM
storage for variables
LOMEM
TOP
user's BASIC program area
PAGE
used by MOS

Replacing the internal battery

REFER TO THE HEALTH AND SAFETY INSTRUCTIONS AT THE FRONT OF THIS GUIDE BEFORE ATTEMPTING TO REMOVE OR REPLACE THE INTERNAL BATTERY.

The Lithium cell fitted in your computer is used to maintain the content of the CMOS RAM at all times when the computer is disconnected from the mains power supply.

Under normal operating conditions, the life of the cell can be expected to be well in excess of one year, but annual replacement is recommended to ensure absolute reliability. Replacement cells can be obtained from your supplier – note that standard alkaline batteries are **not** suitable for this application.

Removing or replacing the battery will corrupt the current content of the CMOS RAM and it is recommended that a copy of the settings is first written to either tape or disc. This can be achieved in the following manner:

– Select the appropriate filing system and load a cassette or appropriately formatted disc;
– Type:

```
MODEØ RETURN
*SPOOL CONFIG RETURN
*STATUS RETURN
```

The computer will display (and store) a list of the current CMOS RAM settings.

Close the spool file by typing:

```
*SPOOL RETURN
```

To replace the internal battery, proceed as follows:

– disconnect the computer from the mains supply;
– remove the computer's top cover by unscrewing the four screws labelled FIX (located on the computer's underside);
– locate the *battery nest* (shown in the illustration below);
– remove the existing cell and replace it with the new cell, ensuring correct polarity by reference to the + and – markings on the battery's casing – dispose of the old cell sensibly;
– replace the computer's top cover;

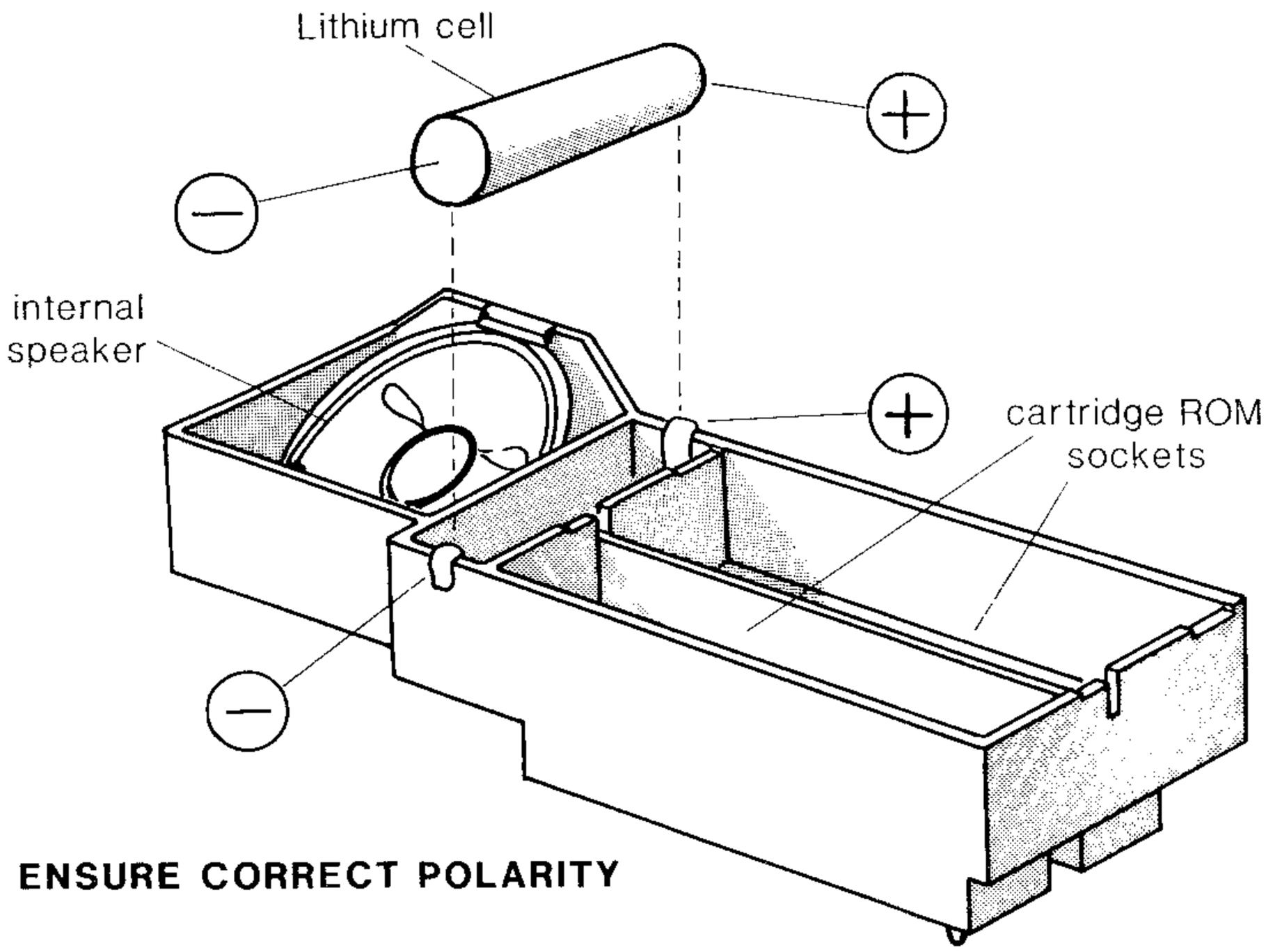
The CMOS RAM may then be restored to its former state in the following manner:

– connect the computer to the mains supply and execute a power-on reset, i.e. switch the computer on whilst holding down the R key. Keep the R key depressed until the message:

CMOS RAM reset

press BREAK to continue

appears on the screen.



– press [BREAK] and manually reselect the filing system used to store file CONFIG (as above):

  For CFS : press [CTRL] + T + [BREAK]
  For DFS : press [CTRL] + D + [BREAK]
  For ADFS : press [CTRL] + F + [BREAK]

– select the BASIC language, select MODE 0 and then display the content of file CONFIG, i.e. type:

```
*BASIC [RETURN]
MODEØ [RETURN]
*PRINT CONFIG [RETURN]
```

– use the information from the screen display as parameters to a sequence of *CONFIGURE commands (see Appendix C). You will, of course, have to reset the date and time using either TIME$ (as described on page 98) or the Control Panel utility.

Index

Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
England