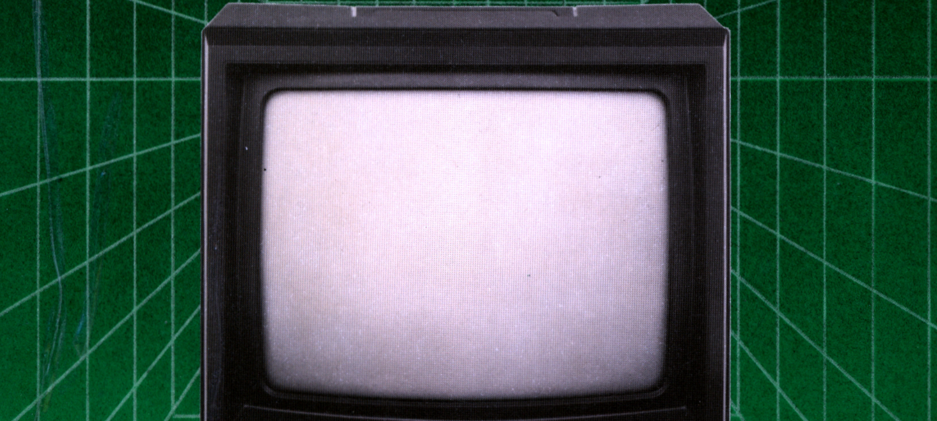
ADVANCED AMSTRAD CPC6128 COMPUTING

including CP/M, Printers and Business Applications



IAN SINCLAIR

Advanced Amstrad CPC6128 Computing

Ian Sinclair

COLLINS
8 Grafton Street, London W1

Collins Professional and Technical Books
William Collins Sons & Co. Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Collins Professional and Technical Books 1986

Distributed in the United States of America
by Sheridan House Inc.

British Library Cataloguing in Publication Data
Sinclair, Ian R.
Advanced Amstrad CPC6128 computing.
1. Amstrad CPC6128 (Computer)
I. Title
004.165 QA76.8.A4/

ISBN 0-00-383300-3

Typeset by V & M Graphics Ltd, Aylesbury, Bucks
Printed and bound in Great Britain by
Mackays of Chatham, Kent

Contents

Preface

At one time, it could be assumed that the buyer of a computer would know very little about a machine or its programming language. Now that small computers have evolved to the stage of the Amstrad CPC6128, however, the buyer can be expected to have some experience, and it is for such a buyer that this book is written. Though the manual for the CPC6128 is excellent as a reference book, the user, in my experience, always needs to be introduced to topics in a suitable order, and to know which are most likely to be important initially. There will always be a few first-time computer owners eager to know how they can get the best out of a new machine, and since the BASIC programming language of the CPC6128 has so much in common with that of the older CPC464, I urge any first-timer with a need to know about elementary programming in Amstrad BASIC to look at my introductory book, *Amstrad Computing with the CPC464*, published by Collins.

This book, by contrast, deals mainly with the differences between the earlier Amstrad models and the CPC6128, and with the more advanced commands that can now be used. There are many differences, and several of them are very significant indeed. One major difference is the use of an upgraded operating system, *CP/M Plus*, which is custom-built for this machine. This is very different from the older CP/M 2.2 that was used on the disc version of the CPC464, and is a very much better system for the not so dedicated CP/M user. It may help to stem the flood of Amstrad owners joining the CP/M user group in search of help! Another major change has been the provision of extra RAM, making a new total of 128K. This extra memory is *not* a way of squeezing in longer programs in BASIC (though it offers some more memory for CP/M programs), but it can be used both for graphics and as a form of random access filing system. Details of the use of the extra memory are provided in this book, along with illustrative programs. The graphics instructions of the previous machine have been retained, with several useful improvements which make for much easier programming of graphics from BASIC. The speed of the Locomotive BASIC is such that there is little point in going to machine code for many actions, but if you are curious, then my book *Introducing Amstrad Machine*

Code, written for the CPC464 but applicable also to the CPC6128, will form a useful introduction for you.

In this book, then, I have assumed that you know how to program in BASIC, and have a reasonable knowledge of computing terms. The topics in this book take over from there, introducing you to the use of the disc systems, both AMSDOS and CP/M Plus, to disc filing techniques, more advanced graphics and sound, the use of the extra memory, and to various hints and tips which will make your programming and use of the computer easier and more effective.

I owe a great deal to David Foster for many of the sections in this book. As a former owner of a CPC664 which he was fortunately able to exchange for a CPC6128, he has listed for me the many improvements in the new machine. Several of his comments and his listings have been incorporated here, and I am very glad to be able to draw upon his experience. I am also greatly indebted to Rhett Houghton and Mark Newton of Sudbury Microsystems, who were able to sell me a CPC6128 when it was not possible for me to borrow one from Amstrad. The listings in this book have therefore been checked on a production model of the computer rather than on a preproduction sample. Finally, I am very grateful to the team at Collins Professional and Technical Books, particularly to Richard Miles, Janet Murphy and Sue Moore, who turn my discs and paper into a real book.

Ian Sinclair

Chapter One
The Disc Systems

Note: RETURN and ENTER keys
On the CPC6128 both RETURN and ENTER keys are provided, and both normally perform the same action. If you have used any other machine, particularly the older Amstrad machines, then you will almost certainly keep striking the ENTER key when you try to reach the SHIFT key. To avoid this in BASIC, execute the command:

```
KEY 139,CHR$(0)
```

– and in CP/M, make a KEYIN.CCP file, using ED, which reads:

```
6 N S C"↑'#07'"
```

and use this with SETKEYS, preferably in a PROFILE.SUB file. The creation of these files will be dealt with later in this book.
The effect of these statements is to disable the ENTER key, so reducing the time wasted when you start using the machine.

What is a disc system

I am assuming right from the start of this book that you have used a computer before, and that the actions of setting up the CPC6128 will have been carried out. Since this book is intended for the more advanced user, we can't spare the space to give details of how to set up the machine, and if the CPC6128 is your first computer then I respectfully suggest that you consult a book which is devoted to the more elementary aspects of using the machine. The manual is useful, but mainly if you already have experience. The point at which we take up the reins is with the use of the disc systems, AMSDOS and CP/M Plus.

Disc system is the name given to a complicated combination of hardware and software. A disc system comprises the disc drive (or drives), the disc controlling circuits, both of which are hardware, and the disc operating system, which may be in software, firmware (ROM), or a combination of both. Different manufacturers approach the design of disc systems in different ways. The Amstrad approach for the CPC6128 has been to build one disc drive into the casing of the computer, and to make provision for another to be connected separately, using a connector at the back of the computer. The software which makes the disc system work is partly in the form of chips which hold some of the programs that are needed to control the action of the disc drive. This type of software cannot be altered except by plugging in other chips, and for that reason is often referred to as 'firmware'. The whole of the AMSDOS system is held in firmware in this way. Other parts of the control programs are held on the master discs (the system discs) which come along with the drive. The disc system is, in fact, a miniature computer in its own right, complete with its own memory.

The controlling circuits for the disc system are contained within the computer unit along with the disc filing system (DFS). A 'file' in this sense means any collection of data which can be stored on the disc. The DFS consists of a program, and most computers use a 'DOS disc' to hold this program. DOS is short for disc operating system. When this is the case, a lot of the RAM memory (the memory that is free for you to use) is needed to hold the DFS. The Amstrad CPC6128, however, uses chips within the computer to hold much of this information, and this leaves most of the memory of the CPC6128 free when you are using the normal AMSDOS system. Some memory has to be used, and this is taken from the main memory of the computer.

Tracks, sectors and density

The language of disc recording is another novelty that you may have encountered for the first time with the CPC6128. If your sole concern is to save and load programs in BASIC, you may possibly never need to know much about these disc terms. A working knowledge of how disc storage operates, however, is useful. At the simplest level, it makes it easier to understand how the disc stores data, so that you can understand why it should be that the disc has lots of unused space, but cannot accept any more data! At a more advanced level, it can allow you to extract information from damaged discs, and to make changes to the information that is stored on discs.

Unlike tape, which is pulled in a straight line past a recording/replay head, a disc spins around its centre. When you insert a disc into a drive, the protective shutter is rotated to expose part of the disc. When the drive is activated a hub engages the central hole of the disc, clamps it, and starts to spin it at a speed of about 300 revolutions per minute. The disc itself is a circular flat piece of plastic which has been coated with magnetic material. It is enclosed in a hard plastic case to reduce the chances of damage to the surface. The hub part of the disc is also built up in plastic to avoid damage to the disc surface when it is gripped by the drive. The surface of each disc is smooth and flat, and any physical damage, such as a fingerprint or a scratch, can cause loss of recorded data. The jacket has slots and holes cut into it so that the disc drive can touch the disc at the correct places. The slot and one hole (one of each on each side) are covered by a metal shutter when the disc is withdrawn from the drive. You can see the disc surface if, holding the 'A' or '1' side uppermost, you insert your thumbnail into the slot at the right-hand side of the disc casing, near the front. By sliding your nail back you engage the sliding peg which acts on the shutter, and you can turn the shutter until the disc surface is visible. Do *not* touch the disc surface, sneeze on it, or do anything which could leave any marks on the surface.

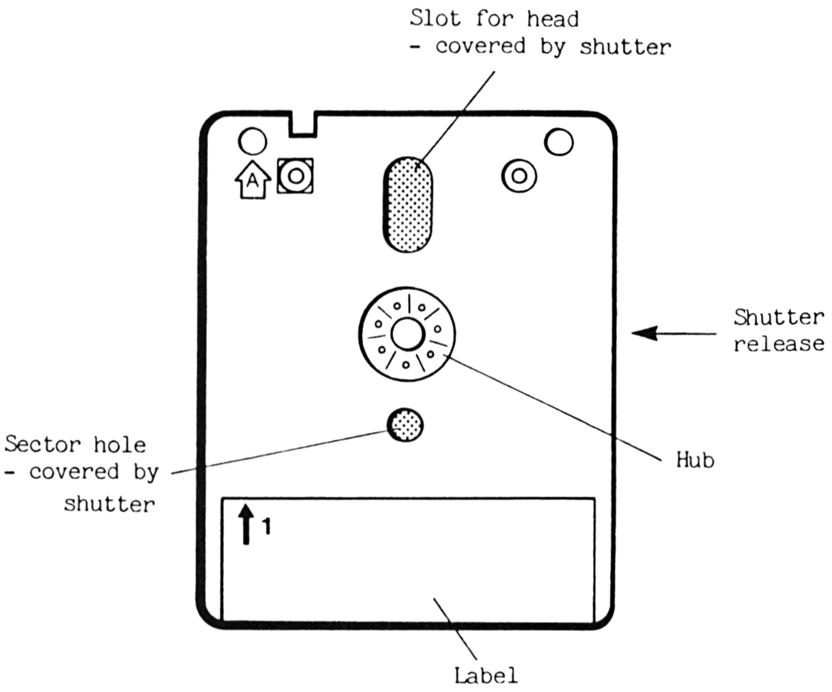


Figure 1.1. The slot in the casing of the disc allows the head of the drive to touch the disc surface when the shutter has been swung aside.

Through the slot that is cut in the casing (Figure 1.1), the head of the disc drive can touch the surface of the disc. This head is a tiny electromagnet, and it is used both for writing data and reading. When the head writes data, electrical signals through the coils of wire in the head cause changes of magnetism. These in turn magnetise the disc surface. When the head is used for reading, the changing magnetism of the disc as it turns causes electrical signals to be generated in the coils of wire. This recording and replaying action is very similar to that of a cassette recorder, with one important difference. Cassette recorders were never designed to record digital signals from computers, but the disc head is. The reliability of recording on a disc is therefore very much better than could ever be obtained from a cassette, which is why computers for serious use never feature ordinary audio cassettes.

Unlike the head of a cassette recorder, which does not move once it is in contact with the tape, the head of a disc drive moves quite a lot. If the head is held steady, the spinning disc will allow a circular strip of the magnetic material to be affected by the head. By moving the head in and out, to and from the centre of the disc, the drive can make contact with different circular strips of the disc. These strips are called *tracks*. Unlike the groove of a conventional record, these are circular, not spiral, and they are not grooves cut into the disc. The track is invisible, just as the recording on a tape is invisible. What creates the tracks is the movement of the recording/replay head of the disc drive. A rather similar situation is the choice of twin-track or four-track on cassette tapes. The same tape can be recorded with two or four tracks depending on the heads used by the cassette recorder. There is nothing on the tape which guides the heads, or which indicates to you how many tracks exist.

The number of tracks therefore depends on your disc drives. The vast majority of disc drives for other machines use larger discs with either 40 or 80 tracks. Forty track drives use 48 tracks per inch, and 80 track drives use 96 tracks per inch. The Amstrad CPC6128 disc drive uses a 3 inch disc with 40 tracks, but with 96 tracks per inch. This forces you to find a source for these special discs, however, which are by no means common even now. At the time of writing, in fact, only a few of the independent suppliers, such as Disking and Garwoods, were advertising them. Be very careful when you send for discs that you specify the 3 inch type. Several business computers, such as Apricot, have standardised on the Sony type of 3½ inch disc. This size is quite easy to find, and considerably cheaper. It is, however, *not* suitable for your CPC6128 disc drive. You can use either the Amsoft disc or the Maxell (also labelled 'Tatung') disc. They are practically identical except for the write-protect shutters, of which more later. At the time of writing, the price of a ten-pack was about £40, which compares with about £10 for a ten-pack of the normal 5¼ inch floppy discs for other machines.

Once you have accepted the idea of invisible tracks, it's not quite so difficult to accept also that each track can be invisibly divided. The reason for this is organisation – the data is divided into *blocks*,or *sectors*, each of 512 bytes. A *byte* is the unit of computer data; it's the amount of memory needed for storing one character, for example. Each track of the disc is divided into a number of sectors, and each of these sectors can store 512 bytes. Conventional 40 or 80 track discs use 10 sectors per track, but the Amstrad system uses 9 sectors per track, allowing $512 \times 9 = 4608$ bytes to be recorded on each track. Two tracks are reserved on a master disc (the *system* disc) for holding essential data, leaving 38 tracks for your use. Of these tracks, 4 sectors (2K) are reserved for the 'directory' entries, leaving the rest free. This corresponds to a total of 173056 bytes free, which is 169K. This applies mainly to a back-up system disc, however. If you are using only CP/M Plus or AMSDOS files, it is possible to make use of the reserved tracks by 'data formatting' the discs. In this way, 178K can then be stored on the disc, which will be the normal capacity that you will use for most of your discs.

The next thing we have to consider is how the sectors are marked. Once again, this is not a visible marking, but a magnetic one. The system is called *soft-sectoring*. Each disc has a small hole punched into it at a distance of about 14 mm from the centre. There is a hole cut also through the disc jacket, so that when the shutter is swung aside and the disc is turned round, it possible to see right through the hole when it comes round. When the disc is held in the disc drive and spun, this position can be detected using a beam of light. This is the 'marker', and the head can use this as a starting point, putting a signal on to the disc at this position, and at eight others, equally spaced, to form sectors (Figure 1.2). This sector marking has to be carried out on each track of the disc, which is part of the operation called *formatting*.

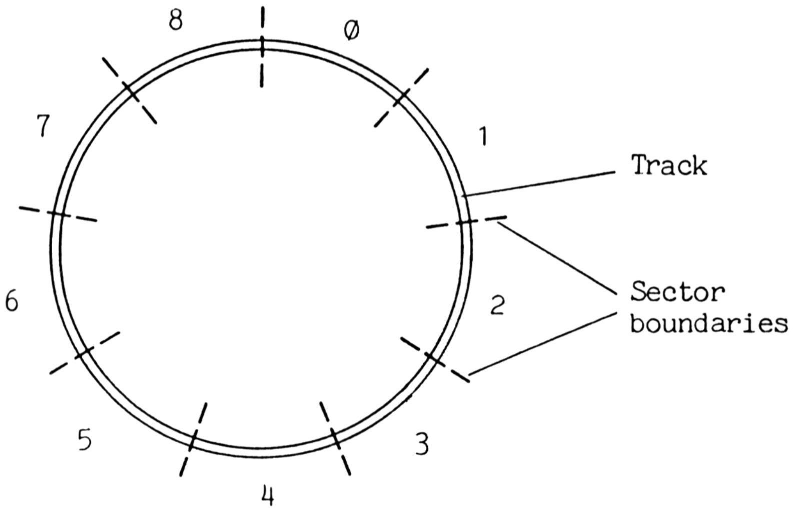


Figure 1.2. The arrangement of the disc sectors around a track. There are nine sectors in each track.

Formatting discs

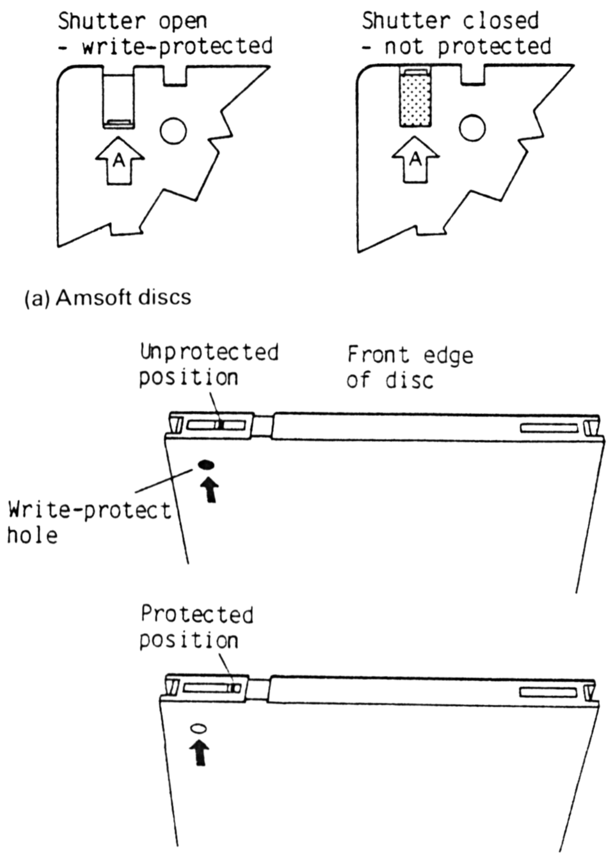
Formatting discs, as we have seen, consists partly of the action of 'marking' the sectors on a disc. The formatting action also tests the disc. This is done by writing a pattern to each sector, and checking that an identical pattern is read back later. Failure to do so indicates a faulty sector, and a disc with such a fault should be returned to the supplier with a request for a replacement. These small discs cost more than three times as much as a conventional floppy disc, and they *ought* to be perfect. The prices would probably come down from their present very high levels if more manufacturers used them, but there is little sign of this happening.

Formatting, then, consists of marking sectors and testing them. This takes about half a minute, and will normally end with a message which asks you if you want to format another disc. You *don't* have to format a new disc if you are going to use the DISCKIT3 program to make a copy of the system disc, because this particular copier (not available on other versions of CP/M) formats the disc as it goes. Any fault which is found at the formatting stage will be reported. This does not necessarily imply a *disc fault* however.

All 3 inch disks make use of a small sliding shutter (Figure 1.3) which exposes or covers a hole at the front left-hand side of the disc casing. If this hole is exposed, the disc is 'write-protected', which means that it cannot be formatted. There are small differences in this respect between the Amsoft and the Maxell discs. The Amsoft discs use a shutter which is operated from the flat surface of the disc. The Maxell type has a shutter which is moved by inserting the tip of a ball-pen into the small lever at the front edge of the disc. Now if the disc is protected by opening the hole that is normally covered by this shutter, it's probably because you wanted to preserve some information that has been recorded on it. Since formatting will wipe the disc clean, the message is a warning to you that you might want to think again. If you really want to format, then you have to slip the plastic cover over the write-protect hole. There are *no* plastic shutters on the system discs, to avoid any danger of erasing them by reformatting. This makes it impossible to carry out some actions, particularly setting keys, etc. on start-up, so you *must* make a copy of at least side 1 of the system discs if you want to make really effective use of the CP/M facilities. Note that if you have been used to ordinary floppy discs of 5¼ inch size, the protection action works in the *opposite* way. On the ordinary floppy, a slot has to be covered to protect the disc; on the 3 inch disc, the hole has to be *open* for protection. It's a particularly important point to remember if your second drive is a 5¼ inch type.

The formatting action is carried out when a set of instructions has been typed and entered. The CPC6128 disc system uses two controller programs, called AMSDOS and CP/M Plus respectively. Of the two, *AMSDOS* is intended for use with the Locomotive BASIC language of the CPC6128, and it's the system that you will use along with programs that are written in BASIC. *CP/M Plus* is a more general disc system, which handles some of the tasks that are common to all disc operations, like formatting. In other words, if you want to format a new disc or reformat an old one, you have to make use of the CP/M operating system.

First of all, you will need the correct system disc. The 6128 is supplied with two system (master) discs, which are referred to as *sides* 1 to 4, since each



(b) Maxell or Tatung discs

Figure 1.3. The write-protect system operates when the write-protect hole is open. (a) The shutter system used on discs which are marked 'Amsoft'. (b) The method used for Maxell and Tatung discs.

disc contains programs on each side. Sides 1 and 2 are the most important for you at this stage. These contain the CP/M Plus system itself, and programs of the type called 'utilities' that are designed to make your use of discs as trouble-free as possible. Side 1 is titled System Utilities; side 2 Programming Utilities. On the second disc, side 3 contains a program for a computing language called LOGO, along with the HELP program, and side 4 contains the older version of CP/M, called CP/M 2.2, which was used on the CPC464 and CPC664 (of brief life) just in case you should find yourself with a disc that was created on one of these machines and which cannot be used along with CP/M Plus. This isn't very likely to happen, but it's good to know that the eventuality is covered.

First of all, then, you need to select side 1 of the system discs. You cannot take risks with this disc – it is *valuable*, and if you don't believe me, then just try getting a replacement! Insert the system disc in the drive, making sure that it is the correct way round. The discs are double-sided, with the sides labelled 'A' or '1', and 'B' or '2' at the front left-hand side, next to the write-protect hole. The side number is at the right-hand side of the disc, on the Amsoft label next to the arrow. The actual recording and reading is carried out on the *under* side of the disc, but the labelling is placed so that side A or 1 is the one being used when this letter/number is *uppermost*. The flap on the front of the drive unit is opened when you push the disc into the slot. Hold your system disc with the A or 1 side facing upwards. Slide the disc into the slot. Don't use any force to do this, because you can jam the disc if you do so. Press the disc in firmly until it stays put and clicks into place. The system disc is now ready to control formatting. Make sure at this stage that you have a new blank disc ready to format, with the write-protection shutter over the hole.

You then have to type the command which will allow the CP/M Plus system to take control of the disc drive. This command is:

```
|CPM (or |cpm)
```

then press RETURN. As in Amstrad BASIC, commands can be in upper or in lower-case. To avoid confusion with other text, however, all command words will be printed in upper-case in this book. You will see the red 'drive working' light brighten, and hear the disc drive motor whirring. Soon the (colour) screen will clear to a bright pale blue colour, change to 80 column mode, and display the message in bright white letters on the blue background:

```
CP/M Plus Amstrad Consumer Electronics plc
```

followed on the next line by the information:

```
V1.0, 61K TPA, 1 disc drive
```

and on the next line by the prompt A> and the cursor. This distinctive screen layout is used to remind you that you are using CP/M and not AMSDOS. I, for one, find the letters hard to read in this format with the colour monitor, and numbers *very* difficult to distinguish. If you find the same difficulty, then be particularly careful when you are using this mode. A good alternative is to use the green screen monitor if you intend to make much use of CP/M Plus, because for a small extra sum you can buy a modulator to display colour pictures, if you need them, on a TV receiver.

To format a new disc you must now type the word DISKIT3, and press RETURN. This will bring up the message:

```
One drive found
```

unless you happen to have connected a second drive, in which case you will see the message:

```
Two drives found
```

This check can be very useful, as it will remind you if a second drive is incorrectly connected. At the bottom of the screen you will now see a menu of the choices open to you. These are:

Copy	7
Format	4
Verify	1
Exit from program	0

These numbers refer to the *function keys* which are on the number keypad on the right-hand side of the keyboard. If you press any of the ordinary number keys, you will get a loud honk from the built-in loudspeaker (unless you have turned down the volume control).

To format, then, you must press the key that is labelled *f4*, and you will see a new menu appear. This reads:

System format	9
Data format	6
Vendor format	3
Exit menu	

so that you can format the disc the way you want. For most purposes you will use the *data format*, but to start with you will probably want to copy the system discs, so you will need to select system format for this. Vendor format is not something that you are likely to need unless you are a professional programmer – in which case you don't need the information either!

At this stage, remove the system disc from the drive, and replace it with the disc that you want to format. Now select the type of formatting you want by pressing the function key of the correct number (the full-stop if you have changed your mind about formatting). If you are using twin disc drives, you will get another menu choice, asking in which drive the disc to be formatted is placed. This menu also allows you to escape by pressing the full-stop key in the number keypad. Making this last selection doesn't, by itself, start the formatting process, because you are given one last chance by a message that asks you to confirm that you want to format by pressing the Y key. This is done because formatting a disc wipes it *completely* clear of all data, whereas other processes do not. When you press this key the formatting will start, and the screen will show the track numbers, from 0 to 39, as the formatting proceeds. In the event of a track being impossible to format, you will be informed by a screen message, and the number of the faulty track will be displayed. Note this, so that you can quote the information when you return the disc. If the write-protect shutter has been slid aside to reveal the hole, then formatting will be impossible and you will get a message to that effect.

Once a side of the disc has been formatted, you are asked to remove the disc from the drive, and to press any key to continue. Any key in this sense should be taken as any ordinary key, number or letter, spacebar or RETURN. You can then format the other side of the disc by putting it into its drive, correct way round, and pressing the Y key again. Pressing any other key at this stage will return you to the first menu again. When formatting is completed you will probably want to return to the normal Amstrad operating system (AMSDOS). You can do this either by pressing CONTROL/SHIFT/ESC, which will restore the normal blue/yellow screen with the 'Amstrad BASIC 1.1' message. An alternative is to type AMSDOS with a system disc in the drive that you are using.

The formatting action reserves four sectors on the third track of the disc. This portion is reserved as a way of storing information about the contents of the disc. To put it crudely, the disc system reads the first few sectors of this track to find if the filename for a program is stored on the disc, and then to find at which sector the program starts. With this information, the head can then be moved to the start of the program, and loading can begin. This part of the track is known as the *directory*, which keeps a record of what tracks and sectors have been used, and which of them is free for further use. The directory entries consist of filenames and numbers which indicate which track and sector is used for the start of each program or other file that is stored on the disc. The filename for a disc consists of up to eight letters for the main name, and (optionally) three letters for the 'extension'. It's easy to forget the limitation on filename length if you have previously used the cassette system of the older machines, because the cassette system allowed filnames of up to sixteen characters. The 'extension' is also a novelty, and we'll deal with it later.

When you 'wipe' a program or some data from the disc, all you do is remove its directory entry – the data remains stored on the disc until it is replaced by new data. This can *sometimes* allow you to recover a program that you thought you had erased if you have a 'disc editing' program. Another important point about the directory is that it cannot take an unlimited number of program names – but we're coming to that!

Storage space

How much can you store on a disc? The Amstrad system uses 9 sectors on each track and a maximum of 40 tracks; 38 tracks on discs which contain the CP/M system program. This makes a maximum of 360 sectors if the system tracks are used as well. Of these, up to 4 sectors will be used for the directory entries, and this leaves 356 sectors free for you to use.

Each of these sectors will store 512 bytes, which is half a kilobyte. If you divide 356 by 2, giving 178, you end up with a figure of 178K on a single side of a 40 track drive. Not all of this will normally be usable, however, because data is not stored at every possible point on the disc. This is because the disc operating system works in complete sectors only.

Suppose you have a program that is 1027 bytes long. The disc operating system will split this into groups of 512 bytes, because it can record 512 bytes on one sector. When you divide 1027 by 512, you get 2 and a fraction – but the DFS does not deal with fractions of a sector. Three sectors will be used, even though the last sector has only 3 of its 512 bytes recorded. When the next program is saved it will start at the next clear sector, so the unused bytes are surrounded, and there is no simple way of making use of them. If you save many short programs on the disc, you will find that a lot of space may be wasted in this way. A set of short BASIC programs, for example, will use one sector each. There is another way in which space can be wasted if you keep a large number of very short programs on a disc. Each program will have a separate directory entry, and when the directory track is full no more entries can be accepted. The system, however, allows up to 64 directory entries on a disc, so you would have to be very fond of short programs to run out of directory space!

The large amount of storage space, theoretically up to 178K, on a disc contrasts with the 41K or so which you normally have available for BASIC programs on the CPC6128. For long programs, then, a disc system can be used as a form of extra memory. If a long program is split into sections, the sections can be recorded on a disc, and a master program entered into the computer. This master program can then call up different sections from the disc as needed, giving the impression that a very large program is, in fact, operating. The use of a disc system, therefore, allows you not only to load programs more quickly and store a lot of data, but also to use the computer as if it had a very much larger amount of memory. Note that you can't use the extra memory of the CPC6128 in any *simple* way for BASIC programs; and Chapter 9 deals with this problem.

Care of discs

1. Keep discs in their protective boxes when they are not inserted in the drive. If you drop a box, it will chip, but this is better than chipping the disc jacket.
2. Buy discs from a reputable source, such as Amsoft or one of the large disc suppliers. At these prices, you can't afford to take risks.
3. Never pull back the protective shutter unless you need to – which is normally never!
4. Never touch any part of the inner disc.
5. Keep your discs away from dust, liquids, smoke, heat and sunlight.
6. Avoid at all costs magnets and objects that contain magnets. These include electric motors, shavers, TV receivers and monitors, telephones, tape erasers, electric typewriters, and many other items which have surfaces that you might lay discs onto.
7. Label your discs well. If the label on the disc is not large enough, use self-adhesive labels in addition – but don't cover any of the shutters,
8. Remember that the disc is read and written from the *underside*.

Figure 1.4. Taking care of your discs. They are not as fragile as these points might suggest, but you have a lot to lose on each disc.

Figure 1.4 lists some precautions on the care of discs. These may look rather restrictive, but remember that a disc is precious. It can contain a lot of data, perhaps all of your programs. An accident to one disc, then, can wipe out all your work at the keyboard, or all the programs have you bought over the course of a year! Always make a back-up copy of each disc, and always take good care of your discs. If you leave a fingerprint on a piece of tape, you may cause some loading difficulties on that piece of tape, but it's unlikely that you will lose a whole program. A fingerprint on the surface of a disc could make the directory impossible to read, so that the whole disc becomes useless.

The metal shutter protects the disc against fingerprints, but not against being demagnetised by strong magnetic fields. These fields are present around loudspeakers, TV receivers or monitors, headphones, and electric motors. All of these should be regarded as potential disc-killers and avoided. You should make sure that the monitor is kept as far away from the disk drive(s) as the cable lengths permit. If you place your monitor close to the disc drive(s), you may find that discs will not format correctly, and that you continually get error messages trying to write or read discs. Even a small change of position may make a lot of difference, and it is a pity that the connecting cables between the monitor and the computer are not longer to give you more scope for locating the monitor in a good position. Incidently, if you are using the green screen monitor along with a computer stand, you may find that you need to prop up the back of the monitor. The screen is tilted so that the monitor can be used on the same desk as the keyboard, but this makes it rather difficult to use with a higher shelf unless you prop up the back by a couple of inches.

Loading and saving

We dealt with the formatting of a disc in the previous section. Once a disc has been formatted you can use it for storage, and if you want to use the discs for storing programs written in Locomotive BASIC, then you will use the ordinary AMSDOS system. For this purpose, it doesn't matter whether you have formatted the disc as a system format or a data format. First of all, type the simple program which is illustrated in Figure 1.5. It isn't exactly a masterpiece, but it's good enough for testing. Make sure that you have a disc in the drive which is formatted and not write-protected. Make the recording now by typing:

SAVE "TEST"   (or save "test")

and then press the RETURN key. The disc will spin briefly, with the red light indicating that it is in use, and then stop, with the test program recorded. In addition, the part of the disc that is used as a 'catalogue' or 'directory' will have a note of the filename 'TEST' and at what part of the disc this program is stored. If you record another program with this

```
10 REM
20 REM
30 REM
40 REM
```

Figure 1.5. A four line REM program to check saving and loading.

filename, the original version is not completely lost, but you will not be able to recover it in such a straightforward way, as you'll see later.

To load (or replay) a program that is on the disc, simply type LOAD"TEST" (or load"test") and press the RETURN key. The disc will spin, and if the correct side of the disc is uppermost, then the 'ready' message will reappear shortly to show that the program is loaded and ready to go. If you used the wrong filename, you will get either the wrong program or an error message, depending on whether or not there is a file with the name you used. If, for example, you asked for a program called TEMP and there was none on the disc, you would get the error message:

TEMP .not found

Not all programs can be loaded in this way, however. You will find, for example, that the ROINTIME demonstration program on side 4 of the system discs will not load in this way. That's because, like most games programs for the CPC6128, it is written in 'protected' machine code rather than in BASIC. A protected machine code program has to be loaded and run in one operation by typing RUN "ROINTIME.DEM" (for example). This loads and runs the program, but you can leave the program only by pressing CONTROL/SHIFT/ESC, which will also clear the memory.

The ordinary LOAD command fails also to load the LOGO program which is on side 3 of the system discs. This is because LOGO is also in machine code and has been saved by using CP/M. To load it, then, you must first switch to CP/M by inserting side 1 of the system discs and typing |CPM. When you see the A> prompt, you should insert side 3 of *a copy* of the system disc which is *not* write-protected and type SUBMIT LOGO3 (press RETURN), upon which you will see the LOGO copyright message. Note that this works *only* if the disc is not write-protected. The system discs are all permanently write-protected (they have no shutters), but if you want (at your own risk) to put more files on to each one, you can stick a small piece of white label over the write-protect hole.

The more normal way of loading a CP/M program is to type the name of the program and then press RETURN or ENTER. This is all you need to load normally when CP/M is in use. Unless you are likely to use some of the (very expensive) business programs which have been transferred on to 3 inch discs, however, LOGO might be the only program that you are likely to load from CP/M!

Loading is generally much faster than storing, because the disc system carries out a check on data when it records, but not when it replays. If you get any sort of error message when you are saving a program, then it's wise to assume that the program has not been saved, and to save it again.

When you have saved a program on a disc, it's time to take a look at the way the disc keeps track of your program. This is done by reading the directory of the disc. Make sure that you are using AMSDOS (dark blue background, orange/yellow print), then type CAT, and then press RETURN. If you happen to be using CP/M, then type DIR instead of CAT. The AMSDOS CAT command gives you a list, in alphabetical order, of the filenames, file types and size of each stored file. It also shows under this list how many kilobytes of storage remain unused and available on the disc. This is a more useful display than the one you get by using DIR in CP/M, because the DIR display does *not* show the file sizes, nor does it show the remaining space. On CP/M, however, you can find the size of all of your files by using a variation of DIR. Typing, for example, DIR [FULL] will produce the file size and arrangement on the disc of all the files. This is the action which was provided by STAT in CP/M 2.2.

You can also print out the DIR display, assuming that you have a printer connected. If you type DIR (or DIR[FULL]), and then follow it with CTRL P before pressing RETURN, the directory will be printed on paper as well as appearing on the screen. It's very convenient to keep printouts of your directories because you can then find what is on each disc without having to insert the disc and use CAT or DIR. Incidentally, if you save a blank file, meaning that there is no program in the memory, you can use the name to identify your disc. This has to be done when you first use the disc after formatting. Return to AMSDOS, and simply type save"-Sysdisc" (for example) and press RETURN. Starting the name with a dash ensures that the name always appears first in the catalogue listing, and the name can be used to remind you of what else is on the disc. There is an alternative way of 'labelling' a disc, using CP/M, but this must only be used on discs that are reserved for CP/M use and which will not be used with AMSDOS.

Remember that each disc has two sides, but only one side can be read by a drive at a time. You will have to turn the disc over to CAT or DIR the other side. If you have only one drive the commands that we have looked at are all you need, but with two drives you must specify which drive you want to use. The built-in drive is A, and this is the drive which is selected automatically at switch-on. Any actions that use drive A can be performed as before, but if you want to use drive B then you have to type B (if you are using AMSDOS) or B: if you are using CP/M. This will select drive B for all further use until you reselect drive A by using A or A:. As an alternative, you can keep one drive selected but make temporary use of another. This is done by making the drive letter *part of the filename*. If, for example, you had drive A selected and wanted to load a program called MYFIL from drive B, you could use either:

```
LOAD "B:MYFIL"  (from AMSDOS) or

A>B:MYFIL  (from CP/M)
```

Note that the B: has to be placed within the quotes in the AMSDOS version, but it is not counted as part of the filename as far as permitted filename length is concerned, nor does it appear in the directory.

You *must* keep a careful record of the filenames you use. This is because the disc drive will quite happily replace one program with another of the same name. If you have just completed a program and you want to save it, then always use CAT to read the directory to find if you have used a filename already. The old program is *not*, however, wiped from the disc. Instead, it is renamed with the extension label BAK. For example, suppose you want to save a BASIC program called EFFORTS. If there is another BASIC program of this name on the disc, it will appear in the catalogue as EFFORTS .BAS – the extension name of BAS having been placed there *automatically* by the action of the system. When you record your new program using the same name, it will receive the name EFFORTS .BAS, and the first program will be renamed EFFORTS .BAK. You can do this only once, though. If you save yet another program with the filename EFFORTS, then the first one will be wiped from the disc, the second one will be renamed EFFORTS .BAK, and the latest one will be labelled EFFORTS .BAS. You can load EFFORTS.BAK only by using the LOAD command with the full filename of "EFFORTS.BAK".

If you really want to prevent the replacing of a program, however, this can be done, and the method (using SET) is detailed in the Chapter 2. Though the protection is carried out by using CP/M, it is recognised by AMSDOS, and in an AMSDOS catalogue (using CAT) the filename is marked with an asterisk. Any attempt to save a file with the same name will then bring up an error message which shows the filename followed by the message 'is read only'. This useful protection should be applied to all valuable files. Note that there is nothing on the CP/M DIR display to mark these protected files. When you have a number of valuable files on a disc, you should write-protect the whole disc by flipping back the small shutter with the end of a ball-point pen or a nail-file. This will protect all of your files on that disc against being written over. It can't protect them from coffee or from stray magnetic fields, though!

MERGE and CHAIN

If you are fairly experienced in writing and using programs in BASIC, you will be able to make use of the other methods of loading a program, apart from LOAD and RUN, that the CPC6128 permits. One of these extra methods is MERGE, which allows you to add a BASIC program to one that is already held in the memory. The normal use of LOAD automatically erases any program in the memory, and loads the new program into the same piece of memory. By using MERGE correctly you can join a program which is on the disc to one that is in the memory to form a long program. There are, however, several rules for using MERGE that you have to obey rather carefully. First and most important is line numbering. The line numbers of the program on the disc must not be the same as the line numbers of the program in the memory. If any line numbers are identical, then the lines on the disc program will replace lines of the same number in the memory instead of being added to them. The second point is that merging two programs will not ensure that they can run as one program. If the first program has an END statement, for example, then that's where it will end; the other will not run! You cannot run two separate programs after a MERGE in this way; the two have to be combined into one program. Even if the two do combine correctly into one program, you have to be certain that the variable names are the same in both, and are used in the same way.

When you LOAD, RUN, or MERGE, any variables in the programs are automatically set to default values of zero for numbers and blank for strings. This normally means that you can transfer variable values only by saving variables on disc from one program and reading from another – but the CPC6128 allows another pair of options. The command CHAIN will load a program and run it, and CHAIN MERGE will do the same, but will allow variables from a running program to be preserved so that they can be used by another program. The CHAIN command requires a filename, and if this alone is specified, the new program will be loaded and will run from its first line. You can also force the CHAINed program to start at a specified line number, so that CHAIN"addit",200 would load the program 'addit' and run it starting at line 200. The CHAIN MERGE command allows variable values to be passed on, and the merged program to be run from a specified line (as would be normal in this case).

An additional facility, however, is to delete lines of the old program in order to economise on memory. For example, the command:

```
CHAIN MERGE "addit",200,DELETE 10-100
```

would delete lines 10 to 100 of the program that is running and then add program 'addit' and run it from line 200. The short listing of Figure 1.6 shows CHAIN MERGE in action. You should type and save program SECOND first (a), then the other program (b). When the main listing is run, it will print values, then the CHAIN MERGE action will load and run SECOND, printing the new values *which make use of the older ones*. You will see, when you list, that program SECOND has completely replaced program MAIN, because we used the same range of line numbers in each. Remember that protected programs cannot be loaded in this way.

(a)

```
10 REM SECOND
20 X=2*A
30 C$="NEW "+B$
40 PRINT C$;" ";X
50 END
```

(b)

```
10 REM PROGRAM MAIN
20 A=6:B$="EXAMPLE"
30 PRINT B$;" ";A
40 CHAIN MERGE"SECOND"
50 END
```

Figure 1.6. Using CHAIN MERGE. Type the program in (a) and save it. Now type the program in (b), save it, and run it. Note how the variables are passed over, and that program (a) replaces (b) in the memory.

Some other disc commands

The words SAVE and LOAD are CPC6128 command words, which operate on programs in BASIC. There is another set of commands, however, which affect anything that is stored on a disc. When these commands are used from AMSDOS, they are all distinguished by the use of the '|' sign before the command name (like | DISC), and by the way that filenames can be included. Many of these commands are also available in slightly different form when you are using CP/M. As your use of discs increases, you may find that you want to group files that are related in some way on to one disc. It would then be very helpful if you could give this disc a title which would remind you of what it contains. This is possible only in the indirect way that has already been suggested. You can, however, easily delete and rename individual files and groups of files. The AMSDOS commands for these actions all make use of the bar (|) prefix, and we have already come across some examples such as |CPM and |B.

Starting with deleting files, the system for doing this with the AMSDOS of the CPC6128 is much simpler than the method used on the old CPC464 disc drive. The disc must not be write-protected – if it is you will get an error message when you try to erase or rename a file. The command is |ERA followed by a comma, then the filename of the file that you want to delete. The command |ERA,OLDFILE, for example, will delete OLDFILE from the disc directory, though the actual bytes of the file remain on the disc until they are replaced by a new file. ERA is, however, one of the many commands that can make use of the 'wildcard' characters ? and *. The query mark can be used in place of any one letter, and means 'any character'. For example:

```
|ERA,"FILE?.BAS"
```

will be taken to refer to files with names such as FILE1, FILEA, FILE2, FILE9 and so on. The asterisk can be used to mean *any collection* of characters, so that if you use the command ERA,"E*" this would mean any name which starts with E. The asterisk can be used in various parts of a filename. For example, *.BAK would mean any 'old version' file, because it would refer to any filename that is followed by .BAK, like ERROR.BAK or PASSIT.BAK and so on. A 'name' such as *.* would mean *any* file. This 'wildcard' system can be useful but you have to be careful with it, especially when you are erasing files, because you are not asked to confirm that you want each of the files erased; it just goes right ahead and erases!

Renaming a file makes use of the REN command. The form of the command is |REN,"NEW.BAS","OLD.BAS" with the *complete* filename specified. No wildcard characters are permitted in this command. As usual, the disc must not be write-protected, and the name of 'new.bas' must not already exist on the disc.

In the following chapters we shall look at how these and many other commands can be used from the more versatile CP/M operating system. Before we start, however, you should be crystal clear about the differences between the systems. *AMSDOS* is a simple disc system which is needed so that you can save and load programs in BASIC and programs which have been written in machine code for the CPC6128. It is the system that you will use for any BASIC or machine code programs which you write for yourself, or buy. *CP/M* is a more general system which can be used for such actions as copying, erasing and renaming files, but also for a large selection of other actions. Programs which run from the CP/M systems (we talk of programs running 'under CP/M') are all machine code programs which have been designed to strict rules. You are not expected to write such programs for yourself, and you would normally buy them on disc. The problem is that though thousands of CP/M programs exist, and hundreds can be obtained virtually free, most of these are on large discs, not on the Amsoft type of 3 inch disc. In addition, the business types of programs that use CP/M are often very expensive – more costly than your complete computer. You have two problems, then; one of obtaining programs at reasonable cost, and the other of obtaining them on the 3 inch disc. The solution is to keep a very close eye on the advertisements in user magazines devoted to your computer. You will find that most of the reasonably priced and useful programs use AMSDOS rather than CP/M.

Chapter Two
Backing Up

One feature of a disc storage system which is less pleasant is that an accident to a disc can result in the loss of a lot of information as far as conventional LOAD commands are concerned. This does not mean that the information cannot be recovered from the disc, but this is a desperate measure, not to be undertaken lightly. It makes sense, then, if you have a disc full of valuable programs or data, to make a back-up copy as soon as possible. One sensible measure is to make a second copy of each program as you put it on disc. If you have bought programs on disc, however, you will still need to make a back-up copy, or two copies if the disc is a valuable one.

The CPC6128 system allows you to copy the whole of a disc surface. Note that I mean *surface*, not disc. The discs are two-sided, and any back-up method will copy only from one side. If you want to back up an entire disc, you will have to back up each side separately, turning the disc over at some stage to read from the other side. For many purposes, however, copying a file is enough, because you may have only one valuable program or data file on the disc. The operating system of the CPC6128 provides very well for copying a named file from one disc to another. If you use AMSDOS only, with programs in BASIC, this involves separate LOAD and SAVE steps. In other words, you will have to load the file into memory from one disc, and save it to another. This is straightforward enough when the files are BASIC programs, but the task is much more difficult when the files are machine code programs or data files. It is possible to back up machine code programs providing you know the essential information about the program. For example, if you have a machine code program from which you can break out into BASIC, then you can save this on disc. You need to know the address at which this program starts, and the length of the program in bytes. For example, suppose you have a machine code program which starts at address 1000 and which consists of 9060 bytes (denary). You can save this on disc by using

```
SAVE"MCODE",B,1000,9060  (RETURN)
```

but this is no great help if you don't know where the start and end addresses happen to be. The CAT command applied to disc does not help you very much with this, because it does not give a display of the disc header. This is the part of the disc which carries the information about where the program resides in the memory.

As it happens, the skilled machine code programmer has no problems finding this information, and by the time this book appears, there will probably be a rash of programs on disc which will help you. Fortunately, the utility programs which are part of the CP/M system are available for carrying out this essential copying task. We'll look later at the CP/M PIP utility which, among many other uses, allows files to be copied from one disc to another. A 'utility' is a program which aids you in some useful task like backing up a disc, printing what's on the screen, and so on.

Backing up is particularly easy when you have twin drives. With two drives in use, you can use a utility program to cause *everything* on one side of the disc in drive A to be copied to the disc in drive B, or the other way round. The process is accompanied by a lot of clicking and whirring, as one disc is read and the other written, but at least you don't have to attend to the process. You can make yourself a cup of coffee while it is all happening. Any important software should always be backed up on to another disc, and the original disc kept in a cool safe place well away from all the hazards to discs, such as loudspeakers, TV receivers, electric motors and anything else that uses magnets of any kind. Later in this chapter, we'll take a look at the sort of utility programs that are available for the CPC6128.

Making back-ups

For backing up programs which are not BASIC, and about which nothing is known, we have to turn to CP/M. The CP/M Plus package contains many utility programs which are stored on the CP/M system discs, and the ones which are of special interest to us at the moment are held on side 1 of the system discs. You should by this time have followed the instructions in the Amstrad manual about making a back-up copy of your system discs. If you have not, then now is the time to do it! DISCKIT3, which we previously used for formatting, is the utility that allows such a back-up copy to be made, and we shall therefore cover it first. The system discs are permanently write-protected, so that there is no risk of damage to them. If you are copying any disc which has a write-protection shutter, however, make certain that the shutter is open. If you don't, then some day you will find yourself copying the contents of a blank disc on to a disc that you have just bought!

To copy the whole of one side of a disc, as you would when making a back-up of the system disc, write-protect the disc and then insert it into the drive. If you are copying the system disc, the label should show side 1 uppermost, and you should then engage CP/M by typing|CPM. When you press RETURN, the disc will spin, and after a short time (when CP/M has been loaded) you will see the screen change to 80 character mode. This is when you have to be careful, because some numbers can be very hard to read in this mode, and a mistake can have unfortunate consequences. Fortunately, for the DISCKIT3 program, no numbers have to be read. You simply type DISCKIT3 and press RETURN. This is the universal CP/M method of loading a named program – just type the name (no quotes needed) and press the RETURN key. You then need to answer the menu selection, as you did when formatting. On the first menu, however, you will now select option 7, the Copy option. When this copy utility runs it formats the disc as it goes, so that you can use an unformatted disc if you like. If you have write-protection on the disc to which you want to make the copy, you will get an error message when the program starts copying.

For a single drive, you will get messages about which disc to insert at various times. The messages are 'Insert disc to read' and 'Insert disc to write', and when you have placed the correct disc in the drive you can press any key to continue the action. When you see the message to the effect that copying is complete, you can remove the disc from the drive and press any key. This will allow you the option of copying another disc or returning to the CP/M system. If you use two drives there is no need to swap discs around. All you need to do this time is specify which drive contains the write disc and which contains the read disc. The transfer is then completely automatic.

PIP copying

The utility called PIP is on side 1 of the system discs, and one of its many actions is to copy individual files from one disc to another. To do this, the PIP program has to be loaded into the computer, and this has to be done from CP/M. When you have entered CP/M, and the A> prompt is visible, type PIP and press RETURN. PIP then loads in, and when it is ready, the screen cursor changes to an asterisk, *. You can then remove the system disc, and type your PIP command, such as:

```
    B:=A:FILNAM
```

with the filename of the program that you want to copy, and press RETURN. Now this command looks as if it should work only with two drives. If you have two drives the transfer is straightforward, with the program FILNAM on the disc in drive A being copied on to the disc in drive B. If you have only one drive, however, the command will still work, and the screen will issue prompts to remind you which disc to put into the drive. This is very convenient, as it saves having to use different commands for a twin drive machine compared with a single drive machine.

Twin discs

You will have gathered from all that has gone before that the CPC6128 can be used quite easily with only the built-in drive. It is very much more convenient, however, to work with twin disc drives, particularly if you intend to make use of programs for business purposes. Most CP/M business programs, in fact, can be almost impossible to use with only a single drive. In addition, when you have to swap discs, you are always in danger of having the wrong disc in a drive, and from there it's one short step to writing on to a disc that you wanted to be kept for reading, or even wiping out wanted files. Unless computing is purely a hobby and time is of no great importance, then a second disc drive is a useful acquisition and, after a printer, should be high on your wanted list.

The difficult decision at this point is whether to go for the standard Amstrad 3 inch disc drive or to wait for other drives to become available. At the time of writing, few alternative suppliers had emerged, so that users of the CPC6128 did not have the huge choice of disc drives that the BBC Micro user enjoys. This situation is now changing, and you will soon have the choice of being able to add either a 3½ inch or a 5¼ inch drive as your second drive. My choice would be an 80 track 5¼ inch drive, because a great deal of useful software is available on discs of this format. This does *not* necessarily mean that such software, even if it is CP/M, will run on the CPC6128, but there is a better chance that you could adapt it by reconfiguring your CP/M system. In any case, once the hardware is available the software should follow. The difference in disc prices for, say, a hundred discs should pay for a fair bit of useful software, and there is a chance that you *might* be able to make use of the 'free' software from the CP/M user group. Notice I am carefully using words like 'might' and 'should'. It's difficult to tell what may become available in the computer world these days. Many suppliers are being very careful, particularly those who were about to announce new add-ons for the CPC664! The normal second drive for most users, however, will be the Amstrad second disc drive. This is not costly (less than £100 at the time of writing) and of course it is a straightforward plug in fit to the CPC6128.

Using CP/M Plus

So far, we have looked at the use of the AMSDOS operating system in more detail than the CP/M Plus system. This is because anyone who programs the CPC6128 in BASIC is more likely to use the AMSDOS system almost exclusively. CP/M is a system which is designed much more for the *user* of expensive business-biased software than the *writer* of programs in BASIC. The CPC6128 comes supplied with two versions of CP/M. Sides 1, 2 and 3 of the system discs make use of CP/M Plus, and side 4 uses CP/M 2.2, an older version which was used on the CPC464 and CPC664. The only reason for including CP/M 2.2 in this system disc was that, since it had been used on the older models, it should be available for the CPC6128 as well. This is a safeguard against the rather remote possibility that some CP/M 2.2 programs might not run correctly with CP/M Plus. If you have found that this happens, you could use side 4 as your CP/M system disc, and run under this operating system. This should never be necessary, and the use of CP/M Plus (sometimes referred to as CP/M 3.0) is a considerable improvement on CP/M 2.2.

CP/M Plus, then, is the latest version of a well-established system (first designed in 1973!) which is used mainly on computers which have no BASIC in ROM. When you use CP/M, *BASIC is switched out*, so you cannot expect to load and run a BASIC program when you use CP/M, unless you have bought a version of BASIC (like Mallard BASIC) which runs under CP/M. All of the programs which you use with CP/M will be in machine code, and unless you write machine code or make use of a language (like C or Pascal) which can be compiled to machine code, you are unlikely to write programs using CP/M. In any case, the aids to machine code programming which are included with the CP/M package are designed for the old 8080 chip rather than for the Z80 which is used in your CPC6128. This does not make these packages entirely useless, because 8080 code is compatible with Z80 code, but it's not quite so easy nowadays to find out about the 8080 codes, particularly as they are applied in the Amstrad machine.

In this section, we shall look mainly at the commands of CP/M Plus which do not require extensive knowledge of machine code. First of all, though, we need to look at the filename extensions. Remember that, though all the commands are shown here in upper-case (capital) letters for the sake of clarity, they can be typed in lower-case.

As you know from your use of AMSDOS, filenames can have no more than eight characters. In addition to the main name, however, you are permitted an 'extension' of three characters following the main name. There *must* be a dot (period) between the main name and the extension. These extensions are necessary for CP/M, and so they are used in the AMSDOS system as well. Their purpose is to identify a file type correctly, and if you do not specify an extension for yourself, the operating system will supply one. The most common extensions are listed in Figure 2.1. This is a small selection, and if you use CP/M programs a lot of the time, you could come across thirty or more of these extension names. You can make up three letter extensions for your own purposes, but the standard versions cater pretty well for everything you are likely to need. The programs on the system discs use a few other extensions, such as EMS (early morning start) for the CP/M itself, CCP for programs that affect the keyboard/screen/printer, and WP for word processing.

There are other extensions which can be used *preceding* the main filename, separated by a colon. The most important type of extension

.ASC	File of ASCII text
.ASM	Assembly language program file
.BAK	Backup file
.BAS	BASIC program file
.COM	CP/M transient program file
.DAT	Data file
.DOC	Document or text file
.HEX	Machine code file
.LIB	Library file
.OBJ	Machine code (object code) file
.PRN	Assembly language listing file
.REL	Relocatable machine code file
.SUB	SUBMIT file
.TEX	Text file
.TXT	Text file
.$$$	Temporary file. Also used for an unreadable file

Figure 2.1. Some of the most commonly used extension letters for CP/M files.

preceding a filename is the drive letter, A or B. You might, for example, have two programs, both called TESTIT.BAS, but on different drives (not something I would recommend!). One could then be called by typing A:TESTIT.BAS, and the other by typing B:TESTIT.BAS, using the drive letters preceding the filename. These preceding letters are not likely to be of much interest to you if you are using only one drive, however. The other 'extension' at the front of a filename is the user number, which will be mentioned briefly later.

The CP/M commands

CP/M commands are of two types, built-in and transient. The difference is important. The *built-in commands* are contained in the ROM, and when you make use of them, you don't replace anything that is in the memory of the computer. The *transient commands* are carried out by loading a program (a 'command' file) into the memory and running it. This will replace anything else that happens to be in that part of the memory at the time, so you have to be rather careful how you use these commands. In particular, you should not have a BASIC program in the memory and then switch to using a CP/M transient command unless you have a copy of the BASIC program on the disc. Since you will not normally use BASIC along with CP/M transient commands, this is not quite such a problem as it might seem. We'll look at the transient commands of CP/M later in this chapter, but for the moment we'll concentrate on the built-in commands, which are listed in Figure 2.2. Each of these will be obeyed at once when its name(and any other data that is needed) is typed, followed by RETURN. Though these commands are built-in, there are versions which are transient, and which we shall look at later).

DIR	Print directory
DIRS or DIRSYS	Print directory of SYS files
TYP(E)	Print out named ASCII file
ERA	Erase named file
REN	Rename file
USE(R)	Enter user number
n:	Select drive n

Figure 2.2. The built-in commands of CP/M Plus.

Of these commands you will, by this time, have used DIR to obtain the CP/M directory. You can obtain the same directory from AMSDOS by using |DIR, and rather more information from CAT. Staying with the CP/M DIR, however, you can also use this command to select directory entries. Suppose, for example, you want a listing of all the BASIC programs on a disc. You can type DIR*.BAS(ENTER), using the wildcard character to force the system to list any file which has the extension letters BAS. More likely, if you are working in CP/M, is that you will require a list of the .COM files, or possibly .ASC or .DAT files. Once again, you can obtain these selective directories by using the asterisk wildcard character. There is a variation of the built-in DIR, which is DIRS (or DIRSYS), which allows filenames to be displayed which have been excluded from the ordinary directory. We'll go into this later when we look at the SET command.

Of all the built-in commands, DIR is the one that you are likely to use most of all. If you have two drives you will find that you need to use A: and B: along with DIR. As you would expect, A: switches to drive A, and B: to drive B. One of the advantages of using CP/M Plus is that you do not encounter problems if you try to use a drive that is not connected. If you try to access a missing drive, the computer will simply make use of the drive that *is* connected instead. For example, if you type, from CP/M, the command B: with no second drive connected, you will get the message:

```
Please put the disc for B: into the drive then press any key
```

scrolling across the screen. When you insert a disc into the drive and press a key, you will see the prompt B> appearing, to indicate that drive B is selected. A message to this effect also appears at the bottom right-hand side of the screen. This was not possible when using CPM 2.2. Further changes between A: and B: do not cause any further prompts, and the disc drive message does not always change. This ability to use the computer equally easily with one drive or two has already been noted when we considered how to copy a file using PIP.

Using ERA and REN

CP/M uses these built-in commands in a simple way, even simpler than their use in AMSDOS. You must, however, be quite sure of what file, or files, you want to erase. The best method of proceeding is to use DIR first, and copy the filename of the file you want to delete. If there is no file of the name you have typed you will get the 'No file' message. Suppose, for example, that you want to delete MYPROG1.BAS. If you type ERA MYPROG, you will find that this is *not* sufficient. You will get the error message 'No file', which means that a more precise name is needed. If the file is write-protected, then you will get a reminder of the filename and the fact that it is protected. In this particular case (unless the file really *is* write-protected), you need to start again by pressing RETURN (or any other key), then ERA MYPROG1.BAS. When you specify the full filename with its extension, the erasure will be carried out. There must be at least one space between the command ERA and the filename.

You can use the wildcard character, *, but *not* to replace the whole of the extension. You can, for example use ERA *.BAS to erase all BASIC files, but you cannot specify ERA MYPROG.* to erase all files called MYPROG which might have different extensions. If you get a 'BDOS function' message in the course of using ERA, then you have done something wrong and you need to try again. It's very important to use DIR *after* an ERA command just to make sure that erasure has been carried out as you wished.

The REN command works rather as the |REN command in AMSDOS, but with no inverted commas around filenames. There must be at least one space between REN and the filename. The syntax of REN is REN NEWNAME.XXX=OLDNAME.XXX. For example, if you have a file called REMS.BAS, you can rename it TEST.BAS by using RENTEST. BAS=REMS.BAS. If the file REM.BAS is locked, then you will get an error message to the effect that the file has not been renamed because it is read-only, and its name cannot be changed until you have removed the write-protection. Unlike ERA, no wildcard characters are permitted *anywhere* in the REN filenames, because it would be ridiculous to rename a set of files to the same name. The transient program RENAME.COM does, however, permit use of wildcards. A point to watch is that you *can* rename a file with a name that is already in use on the disc. This was not permitted with earlier versions of CP/M. When you rename in this way, the old file is retained with the .BAK extension.

USER and TYPE commands

The USER command is much less likely to be used by the CPC6128 owner than the other CP/M built-in commands. USER is a way of identifying files which have been saved by one user of the system. The idea is useful when several users share a disc system, with each user having an identity number between 0 and 15. By typing USER 8, for example, you identify yourself as being entitled to load files which have been stored when this user code was in action. Since all the files on your disc have been stored using USER 0, there won't be any USER 8 files. If you type USER 8 (RETURN) and then DIR (RETURN), you will get the message 'No file', meaning that there are no files with this user number attached. Unless you are one of a group of disc users who write and save machine code files, this command is of little interest. If you want to keep your files secure, however, when strangers are around, typing USER 8 will prevent a DIR listing of the files. It might be useful when you are showing off the system on Club night! An alternative method is to use the number in the filename as a pre-extension. For example, saving with the filename 8A:MYPROG.BAS will alter the directory so that the entry shows only for the directory of user 8.

The TYPE command is one which is useful only for ASCII files. These might be BASIC programs which have been stored from AMSDOS by using the SAVE"Name",A command, or they might consist of data (names and addresses, for example?) which has been saved by using PRINT#9 in a database program. The reason for specifying ASCII files is that any other files are likely to cause odd effects. TYPE allows the contents of files to be displayed on the screen. Now if you send ASCII codes to the screen, you'll see normal characters appear, but codes in the range 0 to 31 can cause effects like turning off the cursor, setting a new mode, clearing the screen and so on. Machine code programs are likely to contain such characters, and so are BASIC programs which have been saved in the normal way without using the 'A' modifier. Try it for yourself – return to AMSDOS, and write a short BASIC program. Now save the program, using the filename "TESTXT",A. Return to CP/M and type: TYPE TESTXT (RETURN). You will see your program appear on the screen. If you use TYPE on any other BASIC program which has been saved in the ordinary way, you will probably see a line of text, a few graphics symbols, and nothing else. You may find a mode change or other disturbing effects which may require you to 'reboot' (load CP/M all over again). The moral is to keep TYPE for ASCII files only.

The transient commands

As we saw earlier, a transient command is one which is kept stored as a file on disc, and it will be loaded and run only when called. All transient programs are stored in the memory of your CPC6128 starting at address

#0100 (denary 256), and will wipe out anything else which has been stored starting at this address. Figure 2.3 gives a list of the commands which come under this heading. In this list you will have certainly used DISCKIT3, which carries out the formatting and disc copying actions. A fair number of these transient programs are highly specialised, and we shall deal with a few of them in Chapter 3. Figure 2.4 shows a shorter list of the transient programs that are likely to be used most. When you want to make use of these programs, you must have the appropriate side of the system discs in place in the current drive, and you need only type the name of the transient command, then press RETURN. All of these program names have the extension of COM but this does not need to be typed.

SETKEYS	PALETTE	DATE	ERASE	RENAME	SETDEF
SETSIO	DEVICE	GET	SHOW	AMSDOS	LANGUAGE
SETLST	DIR	PIP	TYPE	SUBMIT	SET24X80
DISCKIT3	ED	PUT	SET	HEXCOM	LINK
SAVE	SID	MAC	DUMP	INITDIR	PATCH
XREF	ASM	GENCOM	LIB	RMAC	

Figure 2.3. A full list of the transient commands on sides 1 and 2 of the system discs.

DISCKIT3	Essential copier, formatter and disc checker
PIP	The universal file copying program
AMSDOS	Back to BASIC!
DIR	An extended directory program
ERASE	An extended erase utility
RENAME	An extended rename utility
ED	The text editor
SETKEYS	Key redefinitions for CP/M
SETLST	Printer initialisation
SUBMIT	A way of programming programs

Figure 2.4. A selection of the most useful transient commands, in my own order of utility.

The AMSDOS.COM file is a special one, peculiar to Amstrad machines, which returns the operating system from CP/M Plus to AMSDOS so that you can, for example, make use of BASIC programs. This will, of course, wipe the memory clear of any CP/M files you have been using. The DISCKIT3 program is the one we have used for formatting and copying. It has a third use, which is to check for errors after copying a disc. This is never needed in the normal course of events, but the basis of some protection systems is to cause disc errors when a copy is being made, so that the 'verify' action can sometimes be useful if you suspect that a copy might be incorrect. It might seem odd to see RENAME, ERASE and DIR appearing also among the transients, but these are specially extended versions of the commands. For example, DIR[FULL] will give full details of the size of a file and all information on it. The command RENAME *.SAV=*.BAK will rename all files with the BAK entension into files of the same name but with the SAV extension. This allows all your 'old versions' to be preserved. An even more useful action is ERASE*.*[CONFIRM]. which will erase all files, but will ask for confirmation first. It's particularly useful if you want to delete most of the files on a disc surface, but preserve a precious few.

The PIP utility is one which has very wide applications for other machines that run CP/M, but for the CPC6128 its main purpose is copying individual files. PIP can be entered in two ways. One is to type PIP (RETURN), on which PIP is loaded, and you get an asterisk prompt. You are now expected to type commands, using (RETURN) to get back to CP/M. The other method is to type PIP followed by the actions you want. These will be carried out, and then the machine will automatically return to CP/M. One useful feature of PIP is that during a file copy the filename can be changed. If, for example, you use the command:

```
B:NEWNAME.*=A:OLDNAME.*
```

then any file on drive A which has the filename OLDNAME and any extension, will be copied to drive B with the filename NEWNAME and the same extension. If you have only one drive, then you will be reminded when to change discs.

A particularly handy feature of PIP used in this way is that a wildcard character can be used. The 'filename' part of PIP can also be used to specify the printer or the screen as a destination. For example, when you have run PIP, then by using:

```
LST:=NAME.BAS
```

you can send the file NAME.BAS to the printer. This is useful only if the file has been recorded as an ASCII file, with no special codes. Similarly, the command:

```
con:=name.com
```

will list the file name.com on the screen, once again only if the file is an ASCII one.

You can also use PIP as a form of text editor, by typing:

```
FILENAME.EXT=CON:
```

(then RETURN). This will make the cursor move down a line, on which you can now type something that you might want in a file (for a PROFILE, SETKEYS or SETLST file, see later). At the end of each line you need to press CONTROL-J after pressing RETURN. At the end of the text you press CONTROL-Z to put the file on to the disc. This editing use, however, is better carried out using ED, which we shall deal with in Chapter 3.

Finally in this collection of the more useful transient commands we have SET.COM. This program is concerned with what are called the *attributes* of files, meaning whether a file is read only, read/write, and appears in the directory or not. One particularly useful action of SET.COM is to protect files individually, irrespective of whether the write-protect tab has been used on the whole disc. When a file has been protected in this way, then it is specially marked when you use DIR or CAT to display the directory. Files which have been protected appear in the AMSDOS catalogue listing with an asterisk alongside, and you will not be allowed to erase, change the name, or record another file of the same name without first removing the protection. This applies to any files on the disc, whether they have been created using AMSDOS or are CP/M files. The syntax for making a file read-only is of the form:

```
SET *.BAS[RO]
```

which, in this example, will make all BASIC files read-only. Once again, the use of the wildcard makes this a particularly handy way of protecting all the files of a particular type. Of course, if you want to protect only one file then its full name will be used. To remove protection the syntax is almost identical, but with the letters RW used within the square brackets, so that:

```
SET *.BAS[RW]
```

would make all of your BASIC files read/write.

You can also use SET to conceal a file from the normal directory listing. This type of file is called a 'system file', and if you wanted to make a file called PRINTSET.BAS into such a file you would type SET PRINTSET. BAS[SYS]. When you now use DIR you will get a directory listing without this file, with a note that SYSTEM FILE(S) EXIST. To look at the names of the system files you type DIRS (or DIRSYS), which gives *only* these files, and reminds you that other files exist. You can remove the SYS attribute by using SET with [DIR]. The manual shows how you can also use SET to label a disc, put in password protection, and for date and time stamping. Unless you can be quite certain that you will never use AMSDOS files, however, it's better to omit these actions. If you do use these SET actions, you will also need to use SHOW at times to display the information. The most useful SHOW actions are SHOW used alone to indicate how much room remains on a disc, and SHOW[DIR] which tells you how many more directory entries you can use.

Chapter Three
Other System Utilities

Using ED

The ED.COM file in your CP/M system disc, side 1, is a text editor which can be used for creating, editing and saving ASCII text on to disc. There are no restrictions about how this text editor can be used, though its main use is to create files of assembly language commands which the assembler MAC can transform into machine code. You can also use ED to prepare files for other languages, however, such as Pascal and C, provided that these come in versions which can use the ED files. You can also use ED as a general form of text editor to prepare text which can later be incorporated into your own programs. It's particularly useful for making files for key definition, printer initialisation, and SUBMIT actions, of which more later. If you find that ED is likely to be useful to you, it's a good idea to make a copy of ED, along with the CP/M system, on several disc sides. As before, you can use DISCKIT3 and PIP to make whatever copies you need.

When you call up ED to create or change a text file, you must follow the name ED with the filename you want to use. If you have a text file on the disc and you want to edit it further, then you will use this filename following ED, and the file will be obtained and presented ready for use. Notice that this can be done only if you have ED on the same disc as the file, or if you have a second drive and can specify the second drive letter in the filename. You *can't* type ED, then a filename, and hope to change discs while the machine searches for the filename! You can, however, type ED B:FILE and swop discs when the screen message tells you to. The point is that if this filename is not found on the disc that is present in the drive, then the filename is used to make a new directory entry for a new file, one that you are about to create. This file will be given no extension unless you specify one. Another file, a temporary file, is also opened at this time, and will be erased when you finish using ED. The purpose of this second file is a form of notebook, so as not to take up too much machine space. You never see this second file in the directory because of the automatic erasure, but you have to be sure that there will be space for it on the disc. If, for example, you have only one directory entry left, then you can't be certain of being able to create an ED file. Equally obviously, you can't use ED along with a disc that is write-protected, so you need a copy which is not on the original system disc.

Suppose, then, that you place in the drive a disc which contains ED (*not* the system disc, nor any write-protected disc), and then type:

```
ED TRYOUT
```

and press RETURN. The disc spins, loading the ED program and then looking for this filename. If the filename is not on the disc, then the words NEW FILE appear as a reminder to you that you are now creating text, not editing old text. Under the NEW FILE message, you will see a prompt which consists of a colon and an asterisk. This type of prompt means that ED is waiting for a command letter. There is a huge range of command letters, which are listed in Figure 3.1. The list is rather intimidating, but you'll find that only a few of these command letters are used extensively, and some might never be useful to you. It's a good idea if you intend to use ED extensively to make a copy of the most useful command letters.

When you are trying to create a file for the first time, the most useful command letter is I (or i). This means 'insert new text', and allows you to use both upper and lower-case letters by the normal use of the SHIFT key. You can therefore type whatever text you want, using RETURN to take a new line. The line will be numbered automatically, but this can be turned off by using the command letter -V when you are in the command mode (colon and asterisk). The line numbers are not recorded with the file, and are a useful guide, so I generally prefer to keep them, particularly for an assembly language file.

It's one thing to type in text merrily and use RETURN to give a new line, but what happens when your fingers slip, and you make a typing error? If you use the normal DELETE key, then you will see a zero (0) appear in the *next* character position, and a copy of the character that has been deleted. The delete action *has* been carried out, but it's a lot better for your confidence if you can see it happen on the screen. If you use CONTROL-H for deleting you will see the action. This is a throwback to the days when CP/M was originally designed, and computers did not use DELETE keys. You will find the same use of CONTROL keys in other programs that were written in the early days of microcomputers – one well-known example is WordStar. The use of ED, incidentally, makes the ENTER key give the characters ↑A. When you have finished with a piece of sample text you can get back to the command level of ED by pressing CONTROL-Z, and you can then save your text permanently on disc by typing the command letter E, which saves the text, abandons ED, and restores CP/M Plus. If you want to save your text, but remain with ED for further activities, you can use the command letter H in place of E. In this case, though, it's useful to return to the CP/M level so that you can check the directory entry.

Having tried typing a piece of simple text (try Figure 3.2) and saving it, the

nA	Append 'n' lines from file into buffer
0A	Append from file until buffer is half full
B, −B	Move to start(B) or end(−B) of buffer
nC, −nC	Move 'n' characters forward(nC) or back(−nC)
nD, −nD	Delete 'n' characters forward(nD) or back(−nD)
E	Save file and return to CP/M Plus
Fstring↑Z	Find string in text
H	Save file, stay for editing
I	Enter insert mode
Istring↑Z	Insert string at current position
Jstring1↑Zstring2↑zstring3	Find string1, concatenate string2 on to it, then delete all characters up to start of string3
nK, −nK	Kill (delete) 'n' lines forward(nK) or back (−nK)
nL, −nL	Move 'n' lines forward(nL) or back(−nL)
0L	Move to start of current line
nMcommandlist	Execute command list 'n' times
n, −n	Move 'n' lines forward(n) or back(−n) and display that line
:ncommand	Execute from present line to line n
Nstring↑Z	Extend find string
O	Return to original file
nP, −nP	Move 23 lines forward(nP) or back(−nP) and display
Q	Abandon file, return to CP/M Plus; you will be asked to confirm Y/N
R↑Z	Read LIB file into buffer
Rfilename↑Z	Read named file into buffer
Sstring1↑Zstring2	Delete all string1, substitute string2
nT, −nT	Type 'n' lines forward or back
0T	Type current line
U, −U	Translate to upper-case
V, −V	Line numbering on/off
0V	Display buffer space figures
nW	Write 'n' lines to new file
0W	Write until buffer is half empty
nX	Write or append 'n' lines to LIB file
nXfilename↑Z	Write 'n' lines to named file; append lines if previous X command applied to this same file
0x↑Z	Delete LIB file
0xfilename↑Z	Delete named file
nZ	Wait for 'n' seconds

Figure 3.1. The command letters of ED. Where these are accompanied by a number, that number can be # to mean all, or number 'n'. Zero can sometimes be used to mean current line.

```
      1:    This is a demonstration file of
text
      2:    which has been created using the
      3:    text editor.
      4:
      5:    Using RETURN gives a space, as y
ou would
      6:    expect.
```

Figure 3.2. Some sample text for you to experiment with.

next step is to look at the disc directory entry. The filename TRYOUT should now appear on the disc, and it will have no extension unless you specified one when you called up ED. You can use TYPE either to see the file on the screen, or, by pressing CONTROL-P just before you press RETURN, on the printer. Having satisfied yourself that the file is as you remember it, you can now work with this file to see how the system treats a file that already exists. When you now type ED TRYOUT, you will hear the disc spin as ED loads, but this time there will be no NEW FILE message because the file already exists. All you will get this time is the prompt :* and the block cursor. Your file, however, *will not be shown*. If you use I at this stage, you will be able to type lines to which your file can be added. To get access to your file, you type #A (RETURN). The A command means append, and when this has been done the prompt will include the first line number of the file, but you still won't see the file listed on the screen. To see the file you need to type #T (RETURN), when all will be displayed. It's this sort of thing which will discourage you from using ED as a word processor, because you do need to go through a lot of steps to get anything done, and make use of a lot of commands. You can, however, put several commands into one line, so that you could have typed #A#T (RETURN) and achieved the same effect rather more quickly. Like any strange system, it takes some getting used to. Remember, too, that ED was devised in the days when only professionals used CP/M, and no-one ever considered the possibility of owner-drivers coming along.

Having got your file of text into place using #A and #T, you can now take a look at it. All of the text should be there, because the hashmark (#) is used to specify all of the text. If you had wanted just 2 lines of text, you could have specified 2A to append two lines, or 2T to type just two. Many of these ED commands will take number prefixes like this, with the hashmark always meaning 'all of it'. When your text is displayed, then, you'll see the cursor on a line numbered 1. If you want to add text you will have to move to the end of the file, using the command −B (B without the minus sign means 'go to the start of the file'). It's at this point that you can get thoroughly confused. When you see a line number *and* an asterisk then ED is awaiting a command with that line ready to be worked on if that's what you want. It's only when the asterisk disappears that you can type material.

Suppose, for example, you happen to be looking at the display:

```
1: *
```

which means command mode, line 1 ready. If you now type I for insert and type some text, like 'add this', then RETURN, you will see the words appear, and line 2 will then be presented for new text. You will *not* see what has originally been typed in these lines because you are creating a new line. If you escape again by pressing CONTROL-Z, you can use commands B#T to display the whole of the text. You'll see now that you have created a new text line, and that all the other text has been shifted down. Try again, but this time press CONTROL-Z instead of RETURN. Now you'll see when you list (with B#T) that words have been inserted into a line, rather than creating a new line. This is the action of the I command; insertion of text.

The most irritating feature of this, and all editing actions of ED, however, is that you can't see what you are doing at the time. Unlike modern word processor programs, where you see characters being deleted, shifted, corrected and what have you, ED requires you to list the text again before you can see what has been done. Let's face it, in 1964 I would have given a right arm for a facility like this, and we're getting a little bit spoilt now! You can see what I'm griping about, though, if you try to alter a line. For example, try this (loading in the text in Figure 3.2 again if you want to start with a clean sheet). With the prompt on line 1 (meaning that you see 1: *) type 4C to make the position pointer move along by four characters. All you can see when you press RETURN is that you have another prompt for the same line. Now type −4D which should delete four characters back from the pointer – the word 'This', if you typed the text as originally indicated. Once again, when you press RETURN you simply get another prompt. You can display your line now using T, and you'll see the effect of the change. Now insert a word, by pressing I, then RETURN. Type a word like WHAT, then press CONTROL-Z to insert the word into line 1 rather than occupy line 1 and move the rest of the text down. Now when you use T you will still see the line that reads:

```
is a demonstration file of text
```

– no WHAT in sight! That's because *you did not move the pointer back*. When you use T by itself it prints that line, but from the pointer position only. This is what made your insertion invisible. If you had moved the correct number of characters back before using T on the line, you would have seen your insertion. Try it again, this time using −4C or −5C (depending on whether you put in a space following WHAT), then use T to see the changed line. Each time you type a character you move the pointer, and any type command operates from that position. The character moving commands have the same pointer moving effect.

Once you appreciate what the character pointer of ED is doing, the action of ED becomes a lot easier to understand, if not easier to bear. The editing commands operate from the pointer position, either in characters or in lines. For example, C specifies character movement, like 4C (four forward) or −6C (six back). The L command acts in lines, so that you can have 3L or −2L, for example. You can delete characters using D with the usual numbers and signs, and you can delete (or kill) lines using K, also making use of numbers and a negative sign to go backwards. Unlike modern word processers, what you see may not be what you get, and what you do get you won't see right away. Despite all that, though, ED is packed with features, not all of which you get on some high-cost programs.

As you might expect, you can use ED to find the position of anything in text. Any search, of course, will start from the character pointer position, so if you want to search through the whole text, you have to start with the B command. For example, if you wanted to find the position of the word editor you could command BFeditor, with the B taking you to the start of text, the F meaning find, and the word we are looking for, 'editor'. You have to be careful to type the word just as you expect to find it – if you specify 'editor', then you won't find 'Editor'. When the command is executed you'll see the usual asterisk prompt, and the line number will be the number of the line that contains the word. Now the position of the character pointer is *immediately following the word* that you requested. Since this is the last word in this particular line, the command T will not produce anything apart from the full-stop at the end of the line, and to check that you have actually found the word, you need to type −6CT (which will shift the pointer six spaces back) and then type to the end of the line. You can, incidentally, limit the scope of T with commands such as 2:4T (sounds like a weed-killer!) which will type lines 2 to 4, with one extra line always thrown in for good measure and the pointer left at the start of the first of the lines.

Suppose you want to carry out a search and replace action. The command letter for this is S and it has to be followed by the word you want to replace, then a CONTROL-Z, and then the word you want as a replacement. For example, looking at the demonstration text in Figure 3.2 yet again, suppose you want to replace 'text' by 'words'. You would type:

```
BStext↑Zwords
```

but when you try this you'll see that only the first 'text' has been replaced. This is because S needs a specifier number. If you want only the first three occurrences of 'text' to be replaced, then you can use 3S; if you want to replace every occurrence of 'text' with 'words', then you need to use #S. When you do this, ED will inform you where the last position of 'text' was by a message such as:

```
BREAK "#" AT s
```

followed by the asterisk prompt in the line that holds the last word that was replaced. The character pointer is now placed following the 's' of the last 'words' that has been substituted.

Other features of ED

A text editor the size of ED is not mastered overnight, and in this chapter, I have concentrated on the features that you are most likely to encounter as a CP/M user rather than as a CP/M programmer. Though you *can* use ED as a word processor, it's not really ideal because of the difficulty of seeing what you are doing. Nevertheless, for short documents it can be quite useful, and you can always turn the line numbering off with the −V command if you want to. Remember that CONTROL-P will always bring your printer into action if you want to type text, and it will be typed just as it appears on the screen when you use T. You can also, of course, type direct from an ED file, using 'TYPE filename'.

Another convenience is paging. The editor will divide your text into pages, each of up to 23 lines. You can then display your text on the screen in pages by using B to get to the start of the text, then using P for page display, starting with 0P to show the first page. A command that is sometimes useful is 0V (zero, not 'oh'), which will show two figures. The first of these is the number of bytes that remain available, the second is the total that is free for use when no text exists – the maximum number of characters for ED. For some odd reason, even many costly word processors have no running word count, which makes them pretty useless for authors and journalists. At least ED allows you to estimate a word count by using these figures. Subtract the first figure from the second to find the number of characters used, then divide by 6 to get words – it's a reasonable approximation when you have a few thousand words of text, and is better than just guessing! Even on the short demonstration text it's not all that far out.

Setting keys and lists

One feature of CP/M is that the normal actions of the Amstrad keys are not all suitable for CP/M use. If you enter CP/M and try out the cursor shift keys you'll see what I mean. This can be sorted out by running a program called SETKEYS. The SETKEYS utility needs to make use of a file whose name follows the command, and the standard file that is provided is called KEYS.CCP. Make sure that you have this file on a *copy* of the system disc side 1. This file changes several key actions of the standard keyboard into a form that is needed by CP/M. To carry out all this, then, type SETKEYS KEYS.CCP and then press RETURN to have some of the key actions changed as detailed in the manual. The next step is to look at exactly how some of these actions have been carried out, and to do this we need to take a look at this file, KEYS.

The KEYS file is listed in Figure 3.3. It's not exactly like BASIC, or like any of the usual CP/M commands. With a little research, however, we can start to make sense of it. In the first 13 lines the first character is a number,

```
Ø N S C "^'#1F'" CCP cursor up
1 N S "^F" cursor right
1 C "^'#9F'"
2 N S C "^'#1E'" cursor down
8 N S "^A" cursor left
8 C "^'#9E'"
9 N S C "^W" copy
16 N S "^G" clr
16 C "^K"
18 C "^E" enter
66 N S "^'27'" esc
66 C "^C"
79 C "^X" del
E #8C "^R" ctrl enter
E #9E "^F^B"
E #9F "^F^B^B"
```

Figure 3.3. The KEYS file from the system disc. The first figure is the key number, using E for an expansion key.

and this number is the key *code* number. Not, you'll note, the ASCII code number, but the key number which is shown on the little reminder on top of the disc unit, and in the manual on page 7/23. The first line of the KEYS file, then, deals with key 0, which is the key marked with the up arrow in the numberpad on the right-hand side. The three letters that follow are N, S and C, meaning normal, shifted, and control respectively. By using all three in this specification, we mean that the key has to carry out the same action whether it is pressed by itself, pressed along with SHIFT, or pressed along with CONTROL. What follows is then the action. This must start with double quotes, and if we want to, we can put in a command such as DIR if that's what we want the key to produce. In that respect, the specification is rather like that of KEY in BASIC. In this example, however, the quotes are followed by the up arrow which on my printer appears as a ^, but which appears on its key (the same one as the £ sign) and on screen as an up arrow. All of these key definitions must start with this symbol. Next is an ASCII code, and it is written between single quotes. In this case it's #1F, which is denary 31. This does not produce the effect that you might expect from the manual (i.e. the start of a LOCATE command) but instead it gives the underline character when you are in CP/M. This is the shape that corresponds to ASCII code #5F.

The next line uses 1 N S "↑F" for the cursor right key. The ↑F part of this refers to the control codes which are shown on page 5/19 of the manual, and this particular code is a right shift by one character. Note that this will shift *only over a character*, it will not move across a blank screen. This definition applies only to the normal and shifted key, and for the control + key action we have code #9F. This causes the cursor to jump to the right-hand character in a set, the end of the line of characters. The code, #9F is defined later. The number 2 key, the cursor down key, is then defined as '#1E' in all positions. When you try this it gives two up arrows, so that code #1E gives the effect of the key whose ASCII code is #5E, the up arrow itself. Key 8, normal and shifted, is then defined to give one shift left, and with control, using code #9E, moves the cursor to the beginning of the line. The other keys are defined in the same way, and then finally we have definitions for the mysterious codes #9E and #9F, along with #8C. These numbers correspond to expansion key codes, and they are marked as such by the letter E preceding the key code number. Expansion code #8C, denary 140, is the control-ENTER key, and this is defined as 'retype command line'. Code #9E is defined as ↑F↑B, meaning move one character to the right, then to the beginning of the line. This corresponds with what we get from the number 8 key which was redefined to #9E. The #9F definition is ↑F↑B↑B, which means move one space to the right, then to the start, then to the end. This is possible because ↑B means move to the beginning if not at the beginning, but move to the opposite end if already at the beginning. There's nothing to stop you from altering this file using ED but it's better, if you want to modify keys again, to use a separate file. For example, you may want, as I did, to make sure that the ENTER key is completely disabled. The simplest way is to write a file, using ED with the filename KEYIN.CCP which reads:

```
6 N S C "↑'#07'"
```

so assigning this key with the code #07, which does nothing – it doesn't sound a bell when in CP/M. You can then carry out the disabling action from inside CP/M by typing the direct command SETKEYS/KEYIN.CCP.

Another transient program, SETLST, allows you to send control codes to a printer. This is a very important action if you have a printer, though it doesn't necessarily have to be carried out from CP/M. You can send these control codes to your printer by using BASIC, and then switch to CP/M with no effect on the printer. It may be, though, that you want to carry out all the printer initialisation actions from CP/M, and if you do, then SETLST is the method. Like SETKEYS, SETLST (the word LST is always used to mean a printer) requires a file of commands, and like the commands for SETKEYS, these are in a coded form. Each part of a printer command has to start with the ↑ symbol, and the ESC key, which is used for a lot of printer control codes, can be represented as 'ESC'. For example, to set an Epson printer into emphasised mode for the program listings in this book, I used the code line:

```
↑'ESC'↑'69'
```

which is the CP/M equivalent of the BASIC line:

```
PRINT#8,CHR$(27);CHR$(69)
```

Other codes are dealt with in the same way. Figure 3.4 show the codes in my own PRNT.CCP file, which sets the printer into emphasised mode, the right

```
^'ESC'^'69'
^'ESC'^'81'^'50'
^'ESC'^'108'^'10'
```

Figure 3.4. My own printer SETLST file.

margin at 50 (ESC "Q" 50), and the left margin at 10 (ESC "l" 10). The printer must be switched on when this file is used with SETLST and you will get the usual message about 'ignore, cancel or retry' if it is not.

Submit and profile

All versions of CP/M allow what is called a 'submit' file to be run. In fact, the files that we have used with SETLST and SETKEYS are really a form of submit file. A *submit* file contains a list of programs to be run or actions to be carried out. Suppose, for example, that you used ED to create a file called TEST.SUB which consisted of the two lines:

```
dir
ed prnt.ccp
```

and you recorded this file. From then on, by commanding SUBMIT TEST.SUB, you would make the machine produce a directory, then load in ED along with the PRNT.CCP file. SUBMIT is a very useful type of command when a number of actions are needed, or when several programs are needed in sequence. Even more useful, however, on a day-to-day basis, is the PROFILE.SUB file. This is a form of submit file, but with the difference that you never have to type SUBMIT. The initialisation of CP/M includes a search for a file called PROFILE.SUB, and if this is present on the disc, the commands in the file, which is a normal submit type of file, will be executed as part of the switch-on process. This is possible *only if the disc is write-enabled*, so no PROFILE.SUB file can be used with the system disc, only with write-enabled copies. From what has gone before, you might want a PROFILE.SUB file to contain a SETKEYS KEYS.CCP to initialise the CP/M keys correctly, perhaps another SETKEYS KEYIN.CCP to disable the ENTER key, and a SETLST PRNT.CCP to set up a printer. All of these can be put into a list, using ED, as Figure 3.5 illustrates. Once this file exists on a write-enabled copy of the system disc, then on the command CPM, the CP/M program itself will be loaded in, followed by the execution of the

```
A>type profile.sub
SETKEYS KEYS.CCP
SETKEYS KEYIN.CCP
SETLST PRNT.CCP
```

Figure 3.5. My own PROFILE file, initialising keys and printer each time CP/M is booted up.

commands in the PROFILE list. This takes quite a noticeable time, incidentally, but it can save a lot of keyboard use first thing each morning. The system disc contains PROFILE.ENG, which is a SETKEY statement and language change, as detailed in the manual. This cannot be provided as a SUB file because the system disc is write-protected. Unless you particularly need the £ sign, it's better not to use language 3, in which the £ sign replaces the # sign. Most computing languages make great use of the # sign so unless you are working with financial inputs, the US character set (the default language) is better.

This has not been an exhaustive tour of all the transients on side 1 of the system discs, but instead a close look at the most useful ones. Of the other programs, several are for more specialised use, and others are adequately covered in the manual. I must emphasise that CP/M Plus is a very comprehensive system, and most users will only ever need a tiny fraction of all that is on offer. It's not an adverse reflection on you if you don't happen to need every transient program that is provided. On the other hand, there will be few users who won't find some application for the programs that have been covered in these first three chapters.

Chapter Four
BASIC Filing Techniques

What is a file?

I have used the word 'file' many times in the course of this book to mean a collection of information which we can record on a disc. Programs in BASIC are one type of file (the only type, incidentally, which permits the use of LOAD and SAVE in a straightforward way, with no 'modifying' letters like A, B or P). As you will know by now, the most versatile type of file for BASIC programs is the ASCII file, since this is the one which can be used both by AMSDOS and by CP/M routines. BASIC programs saved as ASCII files do take up rather more space on the disc, and take marginally longer to save and to load, but these differences are hardly significant, even for fairly long programs.

In this chapter I shall use the word 'file' in a narrower sense. I'll take it to mean a collection of data that is *separate* from a program. For example, if you have a program that deals with your household accounts, you will need a file of items and money amounts. This file is the result of the data-gathering action of the program, and it preserves these amounts for the next time you use the program. Taking another example, suppose you devised a program which was intended to keep a note of your collection of vintage 78 rpm recordings. The program would require you to enter lots of information about these recordings, such as title, artists, catalogue number, recording company, date of recording, date of issue and so on. This information is a file, and at some stage in the program, you would have to record this file. Why? Because when you load a BASIC program and RUN it, it starts from scratch. All the information that you fed into it the last time you used it will have gone – unless you recorded that information separately. This is the topic that we're dealing with in this chapter, recording the information that a program uses. The shorter word is 'filing' the information. In this chapter, we will be dealing with filing programs that are in BASIC, so we shall not make any use of CP/M.

Knowing the names

We can't discuss filing without coming across some words which are always used in connection with filing. Most important are 'record' and 'field'. A *record* is a set of facts about one item in the file. For example, if you have a file about vintage steam locomotives, one record might be used for each locomotive type. Within that record, you might have wheel formation, designer's name, firebox area, working steam pressure, tractive force... and anything else that's relevant. Each of these items is a *field*, an item of the group that makes up a record. Your record might, for example, be the SCOTT class 4-4-0 locomotives. Every different bit of information about the SCOTT class is a field, the whole set of fields is a record, and the SCOTT class is just one record in a file that will include the Gresley Pacifics, the 4-6-0 general purpose locos, and so on. Take another example, the file 'British motorbikes'. In this file, BSA is one record, AJS is another, Norton is another. In each record you will have fields. These might be capacity, number of cylinders, bore and stroke, gear ratios, suspension system, top speed, acceleration ... and whatever else you want to take note of.

Filing is fun – if you like arranging things in the right order. The importance of filing is that all of the information can be recovered very quickly, and that it can be arranged in any order, or picked out as you choose. If you have a file on British motorbikes, for example, it's easy to get a list of machines in order of cylinder capacity, or in order of power output, or any other order you like. You can also ask for a list of all machines under 250 cc, which ones use four-speed gearboxes, which are vertical twins, which are two-strokes. Rearranging lists and picking out items is something which is much more difficult when the information exists only on paper.

Disk filing

In this book, because we are dealing with the CPC6128 and its built-in disc system, we'll ignore filing methods that are based on DATA lines in a BASIC program. Though you may be experienced in the use of filing with cassette systems on other machines, I'll explain filing from scratch in this chapter. If it's all familiar to you, please bear with me until I come to something that you haven't met before.

To start with, there are two types of files that we can use with a disc system; serial files and random access files. The differences are simple, but very important. A *serial* (or *sequential*) file places all the information on a disc in the order in which the information is received, just as it would be placed on a cassette. If you want to get at one item, you have to read all of the items from the beginning of the file into the computer, and then select. There is no way in which you can command the system to read just one record or one field. More important, with cassette files you can't change any part of a record, or add more records in the middle of such a file. Files of this type on disc are much more useful, because records can be read and checked more quickly, but adding or changing items still presents problems.

A *random access* file does what its name suggests – it allows you to get from the disc one selected record or field without reading every other one from the start of the file. You might imagine that, faced with this choice, no-one would want to use anything but random access files. It's not so simple as that, though, because the convenience of random access filing has to be paid for by greater complication. For one thing, because random access filing allows you to write data at any part of the disc, it would be very easy to wipe out valuable data, or even the directory, with a program that was badly designed.

We'll start, then, by looking at serial files, which are also easy to record on cassette. All of the AMSDOS commands for serial filing are identical to the commands of the cassette filing system. This makes the change very easy if you have been using filing on cassette and you then upgrade to disc. If you have never used cassette files, of course, it's all new.

Serial filing on disc

We'll start by supposing that we have a file to record, called CAMERAS. On this file we have records (such as Nikon, Pentax, Canon, Yashica and so on). For each record we have fields like Model, Film Size, Shutter speed range, Aperture range (standard lens), Manual or Automatic, and so on. How do we write these records? First of all, we need to arrange the program that has created the records so that it can output them in some order. Figure 4.1, for example, shows how we might arrange this part of a BASIC

```
100 RC$="":X%=0:DIM Field$(5)
110 CLS:PRINT TAB(15)"DATA ENTRY":PRINT:
PRINT"Type X to end entry.":PRINT
120 INPUT "Record name - ";RC$:IF RC$="X
" OR RC$="x" THEN 190
130 REM NEED TO RECORD THIS ON DISC
140 X%=X%+1:FOR N%=1 TO 5
150 PRINT"Field item ";N%;" ";:INPUT Fie
ld$(N%)
160 REM NEED TO RECORD THIS ALSO
170 NEXT N%
180 GOTO 120
190 REM END OF FILE
200 PRINT"There are ";X%;" records on th
e file."
```

Figure 4.1. Organising data for filing, in this example with five fields per record.

program in order to input a number of records, with five fields to each record. The number of fields is five, so the fields are input from the keyboard using a FOR N%=1 TO 5 loop. The number of records isn't fixed, so we use a GOTO loop, which keeps putting out records until it finds one called "X" or "x", which is the terminator. If we can make a test for the terminator at the start of a loop, then it's better to use a WHILE ... WEND loop. If, however, the test must be made at the end of the loop, or in the middle, then we have no alternative but to use GOTO in Locomotive BASIC, because there is no REPEAT ... UNTIL loop. Note that we haven't used an array for holding these items, because an array has to be dimensioned, and we don't know in advance how many items we will have. Instead of storing the items in an array for future use, they will be recorded on disc. The points where the disc recording routine would be fitted are shown in the REM lines 130 and 160. Each item, field or record, is treated as a string. This is because strings are easier to work with – you will not, for example, receive any error messages at the INPUT stage because of a mismatch of variable type. The other good reason for using strings is that a string is a set of ASCII characters, and these files are *always* recorded as ASCII files.

That deals with the organisation of the data for putting on to disc, but how do we actually put it on the disc? There are several stages, and the first is to open up the 'data stream', which is assigned with the number 9. This means assigning a filename which will be recorded on the disc, and sending data to the disc with the PRINT#9 command. The 9 is a number code that the machine will use to distinguish the disc drive from the screen windows (see Chapter 6) or the printer. Each time you want to make use of a file, then, you must have a filename, and this has to be used to prepare for recording on the disc by using the OPENOUT command.

The purpose of using the filename and the channel number 9 is to organise data. The disc stores all data in units of 512 bytes. It wouldn't make sense to spin the disc and find a place on the disc just to record one byte at a time, so when you record or read a disc, it's always one complete sector, or as much of a sector as possible, at a time. In fact, the operating system of the CPC6128 uses 2K blocks of data, which would fill four sectors. Some of the memory of the CPC6128 has to be used to hold data which is being gathered up for recording, or which is being replayed. The channel number 9 is an identifying number for the piece of memory being used, so that the machine can find the correct data in the correct part of the memory. Using this channel number avoids the need to allocate parts of the memory for use in this way as buffers. The memory which is used for this purpose lies at the top of the usable range. To see this in action, switch on from cold, and type ?HIMEM. On my machine, this gave the number 42619. If you now type OPENOUT "test" (ENTER), and then ?HIMEM, you will see that the number is now 38523. The difference is 4096, equal to 4K, or two buffers each of 2K. If you now type CLOSEOUT (ENTER), you will see that ?HIMEM gives the original figure of 42619 again. HIMEM means the top of usable memory; it is shifted down when files are to be read or written, and restored at the end of filing commands. This action prevents these memory addresses from being used for anything else when the program runs. These numbers, incidentally, are the same as you find on the CPC464 and the CPC664 machines, and the reasons for this are detailed in Chapter 9.

Opening the file

After that short diversion, back to our filing program. Before we start to gather the data together for filing, we need to 'open a stream' for the data. This is done using the OPENOUT command. OPENOUT has to be followed by a filename, and if you have used cassette files previously you will have to remember that disc filenames must use no more than eight characters. In addition, it helps if you can give the filename some useful extension label. The 'standard' extension for data is .DAT, so it makes sense to use this unless you have some pressing reason for using something else. When you are developing a program that will go through a large number of stages, it's a good idea to put the stage number into the main title as, for example, CAMERA01.DAT. To take another example, the line:

```
openout "aircraft.dat"
```

prepares to write a file called 'aircraft'. When this line is executed, the disc will spin for a short time, preparing for the file, *but the filename will not appear on the directory/catalogue* because no data has been put into the file. The buffer space will also be prepared in the memory. As always, you can place the drive letter ahead of a colon if you have more than one drive.

The use of the OPENOUT command opens a file – which means that we can make use of the file for writing data on to the disc. It also means that the disc is prepared for the file. Any file that exists on the disc already and has the same name of 'aircraft' will *not* prevent you from opening this file. This means that you have to be rather careful about how you use files, because one file will replace another of the same name, making the old file a .BAK type. This, however, makes it very easy to update and modify files, as we shall see. If you want to lock a file you will have to make use of the CP/M SET command *after* your BASIC data filing program has ended.

Printing to the file

It's at this stage that we need to make use of the loops in the writing program. Within these loops, we need to have a line something like:

```
PRINT#9,Field$(N%)
```

in which PRINT#9 means put the information out on stream 9, the stream for the disc system, so that PRINT#9 will *eventually* put out to the disc system the data that follows. In this example, it's Field$(N%). N% is the number in the FOR ... NEXT loop, so that as the loop goes round, we will put on to the disc Field(1), then Field(2), then Field(3) and so on. We also need to write the 'record' name, and this is done within the loop, by using a line such as:

PRINT#9,RC$

without using an array (because of the unknown amount of dimensioning).

```
10 OPENOUT"AIRCRAFT.DAT"
20 RC$="":X%=0
30 CLS:PRINT TAB(15)"DATA ENTRY":PRINT:P
RINT"Type X to end entry.":PRINT
40 WHILE UPPER$(RC$)<>"X":INPUT"Record n
ame - ";RC$
50 IF UPPER$(RC$)<>"X" THEN GOSUB 100
60 WEND
70 REM END OF FILE
80 PRINT"There are ";X%;" records on the
 file."
90 CLOSEOUT:END
100 PRINT#9,RC$
110 X%=X%+1:FOR N%=1 TO 5
120 PRINT"Field item ";N%;" ";:INPUT Fie
ld$(N%)
130 PRINT#9,Field$(N%)
140 NEXT:RETURN
```

Figure 4.2. The AIRCRAFT.DAT file-writing program. You can make the title suit whatever interests you.

Figure 4.2 shows an example of a very short and simple program of this type which has been adapted from the first example. The programming has been changed to make use of a WHILE ... WEND loop, with a subroutine used for the field inputs and the disc filing statement. You can enter anything you like into this, but it makes more sense to enter something that you can easily check. Since the file is called AIRCRAFT, you could make each record name the name of an aircraft type, and each field some feature of the aircraft, like wingspan, engine details, number of crew, and so on. You can, of course, easily change this program so that it has another title that suits the information you might want to use.

Before we move on, consider what this program has done. It has created a file called AIRCRAFT.DAT, and allocated a stream number of 9 to this file. It has then stored the data as it came along, in the sequence of record, then fields. Finally, the file has been recorded and closed by using CLOSEOUT. This last step is *very* important. For one thing, you don't actually record on the disc *any of the information* in this short program until the CLOSEOUT statement is executed. That's because it would be a very time consuming business to record each item of a file at a time. What the DFS does, remember, is gather the data together in memory. This is the 'buffer' piece of memory, placed just above HIMEM, and it will be written to the disc only under one of two possible circumstances. One is that the buffer is full, so that there are four sectors full of data (2048 bytes) to write. The other is that there is a CLOSEOUT type of statement in the program. For a large amount of data, the disc will spin and write data each time the buffer is full. The CLOSEOUT command then writes the last piece of data, the one which doesn't fill the buffer. It also writes a special code number, called the end of file code (EOF). This can be used when the file is read, as we'll see later. If you forget the CLOSEOUT statement, you'll leave the buffer unwritten, with no EOF – and cause a lot of problems both in your programs and possibly with your disc system. Forgetting the CLOSEOUT is called leaving your files open, and you wouldn't like to be seen like that, would you?

The biggest danger is when you are testing a program. If there is an error, such as a syntax error, which stops the program from running, there will be no CLOSEOUT carried out, and the files will be open. If you had typed a lot of data, you would lose it if you then went on to correct the program and run it again. The correct procedure is to close all of the open streams. In this example it's easy – you only have to type CLOSEOUT and press ENTER. For a large program you would probably find it better to write an ON ERROR GOTO line which, when an error occurred, would close files and end. This automatically ensures that files are never left open. The CLOSEOUT ensures that your data will be recorded. When you use an INPUT statement to gather up the data, you can find that with a lot of data you will hear the disc start and stop at intervals. That's an indication of the buffer transferring data to the disc. You can't use the keyboard while the transfer is taking place, but the time that's needed to write a sector is fairly short. In this example, there is nothing like enough data to fill a buffer. You will hear the disc spin when the OPENOUT command is executed, and again when the CLOSEOUT command is executed, but not at any time between these two unless you enter a huge amount of data.

Getting your own back

Having created a file on disc, we need to prove that it has actually happened by reading the file back. A program which reads a file must contain, early on, a command which opens the file for reading. This is OPENIN, and it must use the same filename as was used to write the file. If we recorded a file using the name AIRCRAFT, then we must not expect to be able to read it if we use CAMERAS – or any other name. Misspelling can haunt you here! Once the stream has been opened we can read data with INPUT#9, which will be followed by the variable name that we want to assign to each item. This command reads an item from the disc, and will allocate it to a variable name for printing the item or other use, according to what we have programmed.

The number of reads can be controlled by a FOR ... NEXT loop, if the number is known, or it can make use of the EOF marker if the number is unknown. By testing for EOF, then, we can make the program stop reading the file at the correct place. The example of Figure 4.3 shows both methods in use. The number of fields was five, so a FOR ... NEXT loop can be used to control the input of the fields. The number of records, however, cannot be settled by a FOR ... NEXT loop, so we have to keep reading the file until the EOF byte is found. This is done in line 120 by testing EOF in a WHILE ... WEND loop. If EOF is not zero, then the file is closed and the program ends. As you can see, this has been put into the WHILE ... WEND loop because EOF needs to be tested *before* another item is read. If you read again from a file like this, you will get the 'EOF found' error message, and the program will stop.

```
100 DIM FX(5)
110 OPENIN "AIRCRAFT.DAT"
120 WHILE EOF=0:CLS:PRINT TAB(12)"AIRCRA
FT DETAILS":INPUT#9,Name$
130 PRINT"Type is ";Name$:RESTORE
140 FOR N%=1 TO 5
150 INPUT#9,Gen$(N%)
160 READ Field$:PRINT Field$;" ";Gen$(N%
)
170 NEXT
180 PRINT"Press spacebar for next record
"
190 K$=INKEY$:IF K$=""THEN 190
200 WEND
210 CLOSEIN:PRINT"END":END
220 DATA Wingspan,Length,Crew No.,Engine
s,Range
```

Figure 4.3. A program which reads the serial file created by the listing in Figure 4.2.

Unless you have arranged for an ON ERROR GOTO line to close files for you, the files will still be open. Leaving a reading file open is not quite such a disaster as leaving a writing file open, but it's still very undesirable. Note that the disc does *not* spin each time you press a key to get another record. This is because a complete sector or set of four sectors is read each time, and if the information that you want is all in one buffer load, the disc need not be used.

Sorry if I seem to be labouring this point, but a newcomer to discs sometimes finds it difficult to remember.

Now this simple example shows a lot about serial filing that you need to know. When you use discs, the name that is used with OPEN (IN or OUT) is the filename for the file on the disc. Any other file that is later recorded with the same name will not overwrite this file, because the old file changes to a .BAK file. The system therefore provides for easy file replacement *and* reasonably good file security. In addition, a file is closed by writing the EOF character. How, then, can you update a file, particularly if you want to add more items to the end of the file?

Updating the file

There are two answers, if we stick to serial filing. One possibility, which is the simplest one for short files, is to load the whole file into the memory of the computer, make the alterations (your BASIC program will have to be written to provide for this), and then write the file again, wiping out the earlier version. The other possibility is to open two files, one for reading and the other for writing. You don't need to have dual disc drives for this, though it makes life much simpler if you do, using a disc drive letter (A or B) at the start of the filename. Working with two files open means that the computer will maintain two buffers. You read one record from the reading file and you can, if you wish, display it. If it's all right, it's then written (to the buffer initially). If the record has to be modified, you can do so. If extra records have to be added, this is equally simple. Each time a buffer empties, the disc will spin and a read or write will take place. This 'simultaneous' operation is possible because of the use of different OPEN commands, which control different buffers. In practice, it's a matter of writing your program to suit. Figure 4.4 shows a simple program which allows you to extend the file created by the program in Figure 4.3. Note, however, that the files use *the same* names, even though I have assumed that both files will be on the same disc. This is because the OPENOUT file and the OPENIN file are treated separately, using different buffers. This avoids any problems of deleting the old file and changing the name of the newly created file. It is a particularly useful feature of AMSDOS, which you don't find on many other disc filing systems. The operating system will see to it that the new file is recorded as AIRCRAFT.DAT, and the old file is renamed AIRCRAFT.BAK.

One point we have to be *very* careful about, however, is closing files. The CLOSEIN command is used whenever the program has finished reading the old file, and the CLOSEOUT command is used whenever the last of the new files has been added.

Looking at the program in detail, line 20 opens two files *with the same name*. One, however, is an input file, and the other is an output file. The input file will be used by INPUT#9, and the output file by PRINT#9, so

```
10 X%=0
20 OPENIN"AIRCRAFT.DAT":OPENOUT"AIRCRAFT
.DAT"
30 CLS:LOCATE 14,11:PRINT"PLEASE WAIT":W
HILE NOT EOF:INPUT#9,Name$:PRINT#9,Name$
40 FOR N%=1 TO 5
50 INPUT#9,Gen$:PRINT#9,Gen$
60 NEXT:WEND:CLOSEIN
70 CLS:PRINT TAB(15)"ADDITIONS":PRINT:PR
INT
80 WHILE UPPER$(Name$)<>"X":INPUT"Aircra
ft name - ";Name$
90 IF UPPER$(Name$)<>"X" THEN GOSUB 140
100 WEND
110 PRINT"You have added ";X%;" items."
120 CLOSEOUT
130 PRINT"END.":END
140 X%=X%+1:PRINT#9,Name$:FOR N%=1 TO 5
150 PRINT"Field item ";N%;" ";:INPUT Gen
$
160 PRINT#9,Gen$
170 NEXT:RETURN
```

Figure 4.4. How to extend a serial file by rewriting it, using the same filename.

there should be no conflict between them, since they use separate buffers. Line 30 clears the screen and issues a PLEASE WAIT notice while the reading and writing is done. If your files are long, it may take the disc some time to perform all of this reading and writing, and this notice is a reminder that it's all happening. Never leave a user with a blank screen, even if the user is always yourself! (When you have tackled Chapter 6, you might want to put this and any other notices into a 'window'.) In lines 30 to 60, data will be read from the old file and written to the new one until the EOF marker is found. When this happens the WEND in line 60 takes effect, and the next command is CLOSEIN, which shuts down the reading file. The writing file is still open, however, with its buffer containing data that has been read so far. You can now add more data, using the same lines as you used in the program in Figure 4.2. When an X or x is typed in response to the request for a record name, the program displays the number of added items, closes the write file (so recording the file) and stops. Quite easy, really, but in this program, no provision has been made for altering any of the records that are read from the old file. This is a routine which we can easily add – and that's the next thing to look at.

Changing a record

It's not difficult to find how to alter a record on a file. You read the item.

print it, and then change the item before re-recording the file. The main problem is finding a neat way of doing this. The program in Figure 4.5 shows one approach which I use in my own file programs. This is to read the whole of one record, display it on the screen, and give the user the chance to edit or leave it as need be. The editing is visual, rather than by the old fashioned method of numbering the entries and asking for a number to be entered. When the record is displayed, a flashing arrowhead points to the first field. If you want to leave the record as it is, press the COPY key and this will bring up the next record. If you *do* want to change a record, move the arrowhead to the record using the cursor up and cursor down keys, the arrowed keys on the right-hand side of the keyboard. When the arrowhead points to the field you want to change, press the spacebar. This wipes out the field name on the screen, but not in the memory. You can now type a new field name or number, and terminate it with the RETURN key. Only when you have pressed the RETURN key is this new field entered, and if you want to change your mind you can delete your entry and type the old entry again.

When a change has been made in this way, the arrowhead still points to the same field, and you can make another change to this or, by shifting the arrowhead, to any other field. When you have finished editing the record press the COPY key to bring up the next record. The process will continue for as long as there are records to read. The amended file will be recorded as AIRCRAFT.DAT, and the old file will be renamed to AIRCRAFT.BAK. Note, however, that the visual editing system is useful only if the fields are short – a field which spreads over more than one line will cause problems. If you work in mode 2 with the green screen monitor, however, you can use longer fields but you need to change the LOCATE numbers to make the messages appear central.

How it works

Lines 20 to 60 of Figure 4.5 follow the pattern which should be familiar to you by now. The files are opened, one for reading and the other for writing, and the WHILE EOF=0 loop will load each record until the end of the file. The record name is assigned to Gen$(0), however, to make it easier to work with the fields in one array. The new items start with the GOSUB 100 in line 60. This carries out the editing, and when editing of a record is complete this subroutine will return. The amended or unamended record is then put into the new file by line 70, and the WEND in line 80 brings up the next record.

In the subroutine the screen is cleared, and a message about the editing commands is printed at the bottom of the screen. Lines 120 to 150 then print the record name and each field on to the screen. Note that this works only in mode 1 if each field is less than 35 characters, because the whole method depends on using one line for each entry. In line 160, new X and Y positions for the cursor are assigned to PX% and PY%, and a loop starts in line 170. In

```
10 X%=0
20 OPENIN"AIRCRAFT.DAT":OPENOUT"AIRCRAFT
.DAT"
30 CLS:LOCATE 14,11:PRINT"PLEASE WAIT":W
HILE NOT EOF:INPUT#9,Gen$(0)
40 FOR N%=1 TO 5
50 INPUT #9,Gen$(N%)
60 NEXT:GOSUB 100
70 FOR N%=0 TO 5:PRINT#9,Gen$(N%):NEXT
80 WEND:CLS:LOCATE 14,11:PRINT"PLEASE WA
IT":CLOSEIN:CLOSEOUT
90 PRINT"END.":END
100 CLS
110 LOCATE 1,23:PRINT"Press arrow key to
 move cursor, space to alter item, COPY
to end edit."
120 PX%=5:PY%=4
130 FOR N%=0 TO 5
140 LOCATE PX%,PY%+N%
150 PRINT Gen$(N%):NEXT
160 PX%=1:PY%=4
170 WHILE INKEY(9)<>0:LOCATE PX%,PY%
180 CLEAR INPUT
190 PRINT">":FOR J=1 TO 50:NEXT
200 LOCATE PX%,PY%
210 PRINT" ";:FOR J=1 TO 50:NEXT
220 IF INKEY(47)=0 THEN GOSUB 280
230 IF INKEY(0)=0 THEN PY%=PY%-1
240 IF INKEY(2)=0 THEN PY%=PY%+1
250 IF PY%<4 THEN PY%=9
260 IF PY%>9 THEN PY%=4
270 WEND:RETURN
280 CLEAR INPUT:PX%=5:LOCATE PX%,PY%:PRI
NT SPC(34);
290 LOCATE PX%,PY%:INPUT Gen$(PY%-4)
300 PX%=1:RETURN
```

Figure 4.5. Altering a record, using a visual editing system.

the loop, the '>' character is printed at the cursor position, held for a short time, then removed. Four keys are then tested by using the INKEY command. INKEY(9) tests for the COPY key, and causes a return from the subroutine if this key is pressed. INKEY(47) tests the spacebar, and if this is pressed then GOSUB 280 calls up the replacement routine. The other two tests are for the cursor movement keys, and if one of these keys is pressed then the value of PY% is altered. Lines 250 and 260 then test the value of PY%, to ensure that it does not stray outside the line limits, and line 270 completes the loop.

When the spacebar is pressed the first action in line 280 is CLEAR

INPUT, a statement that is also used in line 180. This is a special statement that was introduced with the CPC6128, and its effect is to remove any codes from the keyboard buffer. This is essential because when any key is pressed its code is stored in memory until it is read. When the cursor movement keys are used, their codes accumulate in this keyboard buffer despite the fact that the program is using the keys differently. As a result, when the program leaves the loop, these codes are read and acted on. For example, if you have used the down cursor key twice to shift to the second field, there will be two 'shift down' codes in the buffer. There will also be a 'space' code because you pressed the spacebar. The effect would be to make the '?' prompt for the INPUT stage appear at the line you have selected, but the cursor would appear two lines down and one space across. This can be avoided if the buffer is emptied before the INPUT step, by using CLEAR INPUT.

With the buffer flushed out like this, the PX% number in line 280 is changed in order to locate the first character of the entry, and the SPC(34) clears most of the line (in Mode 1). The next LOCATE instruction places the cursor back at the start of the field name, and the INPUT then allows you to make the change. By using Gen$(PY%-4), you assign the new entry to its correct place in the array. Line 300 then restores the value of PX% to place the arrowhead correctly, and the routine returns. The effect is quite impressive, though the key actions in the loop are slightly 'sticky' because of the time delays. If longer fields were used, it would be possible to modify the routine in order to make use of, say, two lines per field, though it's easier to make use of mode 2 for longer fields.

In any case, this and the previous routine demonstrate how serial filing can be used to advantage on a disc system. In many cases, you will find this type of filing more useful than random access filing, which is never easy with any disc system. If you need random access filing, however, Chapter 9 indicates how the extra memory of the CPC6128 can be used like a disc with a simple form of random access filing.

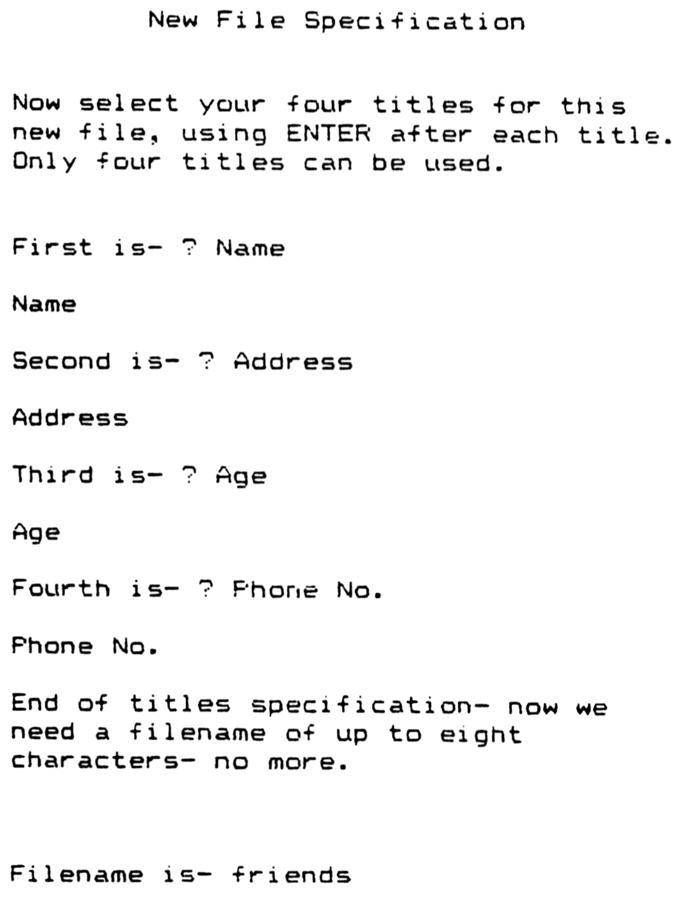
Chapter Five
A Database Example - Filing Cabinet

This chapter consists mainly of one long listing (Figure 5.2 at the end of this chapter) for a database type of program which uses serial filing. The program is called 'Filing Cabinet', and it allows you to specify four heading titles for the fields of your records. These field names are recorded on the disc, and will be used from then on. They might, for example, be Name, Address, Age, Telephone number. You can then enter information, add, delete or change information, read all of the data, or select items as you please. These are the normal actions of a simple database. Looking at the length of the program, you might wonder how long a complicated program would be, but this *is* a simple version. There is no facility, for example, for printing records in alphabetical order of any field. This is, you see, a skeleton database, which has been included to illustrate the use of discs for this type of program. Once you have this program up and running, and have completed reading this book, you should be able to add whatever extra trimmings you need. If, for example, you need random access, you will discover from Chapter 9 how to make use of the extra memory of the CPC6128 to implement this feature to a limited extent.

First principles

We shall start by looking at how the program works in outline. Two files are used, both of which are serial files. One short file, called HEADS.DAT, is used to keep a record of your four headings and of the filename for the main file. The purpose of this file is to make the action of the program automatic, so that you don't have to remember a filename or heading names. When you first use the program for a new variety of file, you will format a new disc side, record the program on it, run it so as to type these titles, and they stay with the file from then on unless you start another type of file on the same disc side. If you want to use more than one Filing Cabinet, you must keep them on separate discs, or different sides, with a copy of the program on each disc side. The main serial file will carry a filename that you will specify when you first start a file, and it can be of as large a size as will fill a disc. Each record uses either its position number or its first field as a 'key' to finding that record. In other words, you can locate a record by knowing that it is number 58, or by knowing that the first field is Surname, and you want to find Carruthers. This scheme is fairly flexible without being too difficult to implement. As I said, this is a skeleton program, and it's yours to trim to shape and pad out as you please. Because of the way the disc system operates, there is always a back-up copy of each file on the disc, with the .BAK extension, which makes the system quite safe to use. Also, because serial filing is used, you don't have to keep to fields of fixed size.

When the program runs, some set-up work is done and you are presented with the main menu. The first time you use the program on a disc you should go for the 'Start NEW type of file' option. This allows you to choose four titles for the fields of your records. These names will be recorded and used forever after, so you should plan them carefully. Figure 5.1 shows a typical display. After entering the fields and lengths, you will be prompted for a filename. You can choose anything you like, as long as it has eight characters



```
              New File Specification


Now select your four titles for this
new file, using ENTER after each title.
Only four titles can be used.


First is- ? Name

Name

Second is- ? Address

Address

Third is- ? Age

Age

Fourth is- ? Phone No.

Phone No.

End of titles specification- now we
need a filename of up to eight
characters- no more.



Filename is- friends
```

Figure 5.1. A typical screen display during the entry of titles.

or less, and is not HEADS.DAT. Perhaps you might like to add a line 495 which rejects this name as a filename, and asks again?

Once the filename has been typed and RETURN pressed, the HEADS file is opened and the headings, along with your filename for the main file, will be recorded. In addition, the main file is briefly opened and its own filename recorded. This is done to ensure that the file exists on the disc for later reading, though you will normally be recording from this point on. At this stage, and at several other places in the program, the disc will be busy and you may have to wait for it. When the file has been created, the program returns to the menu. You do *not* have to use this option again unless you decide to keep another different file of data on the same disc. There's nothing wrong with this if your data files are fairly short, but it avoids confusion if you can keep one disc side for each file.

You can now type the first lot of data into your file by choosing the 'Start ENTRY in file' option. You don't have to do so at once, of course, because the headings and filename are by now safely on the disc, and you can end the program and switch off if you like. When you go for the entry option, you will be prompted by the field names (such as Address, Name etc.) to type in data. The data will be restricted to 38 characters per entry by line 420. This has been done to make the mode 1 screen appearance slightly neater, though data can still spill from one line to the next if both titles and data items are long. Once again, this is something that you can change as you wish. You will find if you enter a lot of data that the disc spins at intervals while you enter data, and you have to wait until the screen cursor is visible again before you can continue typing. To end the entry type X in lower-case or upper-case. If this is inconvenient, change it (line 580)! The whole file will be recorded on the disc, using the filename you supplied originally. Once again, you can leave the program by selecting option 6 of the menu after you have recorded as many items as you like.

If you select the DELETE, CHANGE or ADD option from the main menu, you will be presented with another menu. This time, the choice is to add to the file, change an item, delete an item, or just return to the main menu. You must not use any of the first three options unless a file has already been created, which is why you have the 'cop-out' option. If you choose to add to the file, the disc drive will read the whole of the file, re-record it, and then stay open for additions to the file. The items will be correctly numbered, so that you know how many items were on the file, and the number of each item that you add. If you take the CHANGE option you will be asked to identify which item you want to change; the identification is by number only in this case (once again, this *can* be altered). When you enter a correct item number, the item is found by reading all of the file up to this point and rewriting it. The item whose number you have requested is then printed on the screen. This allows you to check whether you really want to change the record. There is, however, no drop-out option at this point, and you have to enter items for each field. Perhaps it might be useful to allow pressing RETURN to leave the original item unchanged? If that's what you want, then use an INPUT with a temporary assignment here, test it, and then assign to Q$, R$, S$ or U$ only if RETURN has not been pressed. When the change has been made the changed item, and all the rest of the file, is re-recorded. If you choose to delete an item, then once again the item is selected by number, and the file is read and re-recorded as far as the preceding item. The item to be deleted is then read, its title assigned to a variable name, and the rest of the file read and rewritten. The title of the deleted file is then displayed with the information that it has been deleted. Perhaps in this choice you might like to add a routine that lets you read the record and decide whether or not to delete it? If you don't want to delete, then it can be rewritten along with the rest of the file.

Getting back to the main menu, option 4 allows you to list the complete file. When you select this option you get another choice, of listing on screen or on the printer. This is done by pressing the 'S' or 'P' keys. For business purposes this might *always* be a printer option, but if you are using the program for hobby or household interests, you might use only the screen option. Obviously, if you have no printer you might want to delete the 'P' option. If you take the screen display option, each record is displayed and held for you to inspect. You can press the spacebar to get the next record until the end of the file. You might want to dispense with this, simply using the CPC6128 ESC key to prevent the listing running away from you. If the listing is to the printer there are no pauses and the whole listing is printed. Lines 100 to 120 of the program, incidentally, have been kept clear for you to insert any special printer set-up instructions.

Item 5 on the main menu allows you to pick one record for examination. You are then asked if you want to select by item number or by letters. If you choose item number, then the program reads the file to the correct place and prints the record whose number you have requested. If you choose to select a name, you are asked to type the name or the first letter or letters of the name. If you type the whole name, the file will be located only if your typed name agrees *exactly* with the name in the file. You can, however, type just the first letter, and this will result in a list of all the records whose first field starts with this letter. If you type more than one letter, you will get all names which start with these letters – it's rather like using a wildcard in a disc filename. There are no options here for using the printer, nor for looking at one record at a time, but you can add these facilities as you wish.

The program in detail

Now for the hard work. There are a lot of points in this program which are important. If you try to design your own database programs you will need to know what the disc drive does, and this listing reveals much that isn't exactly made clear by the manual, and which is not so easy to illustrate by short examples. No matter how much you may hate looking at other people's programs, then, it will be useful to study this one, so that you can appreciate reasons for some of the lines. Unless you do so, you can waste a lot of time in your own programming looking at inscrutable error messages and wondering why they arise.

The program is built round a core and a set of subroutines. Much of the programming is straightforward BASIC, and I have made no use of fancy colour or screen presentation effects – there's quite enough to type as it is. Programs for business purposes use the printer for anything important, and the screen is used only for messages to the operator like 'Try putting a disc in the drive'. I'll concentrate on the explanations that relate to the use of the disc drive, rather than explaining everything in detail. In other words, I'm assuming that you knew a reasonable amount of BASIC before you bought a machine of this class.

Lines 10-130 are concerned with initial values of constants, and setting up the system. The first two lines need more explanation than you will find in any of the manuals. Normally, when you open and close buffers, the CPC6128 creates space by shifting the top limit of memory. This can create some odd effects in your program, and one of the most puzzling is the corruption of filenames. In this program, the filename for the data is held as a string variable F$. Commands such as OPENIN F$ are then used, but if you follow the normal course that we have outlined so far, you run into trouble with this command. This crops up when you open a file some time after making a menu choice. You may, for example, have defined F$ as 'MYFILE.DAT', but if choice number 1 on the menu has been made, followed by OPENIN F$, you will find that the disc system has been asked to find a file called '1MYFILE.DA', and it can't find it. This is because unless buffers are allocated permanently, some of the memory may be used for storing string names when it is not being used for buffering, and when a buffer is used again, some of this memory is corrupted. The only remedy is to allocate the buffers permanently, using a set of steps mentioned in the manual. Lines 10 and 20 allow a dummy OPENOUT to shift the memory, and then allocate this limit permanently by using the MEMORY command. The dummy file is then closed again. This method of reserving space permanently for buffers solves all the curious problems that can arise when files are opened and closed, with some string storage between these operations.

Line 30 uses an ON ERROR GOTO statement to trap any errors in the BASIC, and ensure that an error will close all files and end the program. This includes syntax errors, so if the program ends when you start to run it, you will know you have made a typing error! It's better, in fact, to make this a REM line until the program is fully checked, along with any changes you want to make. Line 1430 will analyse the error for you before the shutdown. Always check your data files if you have found an error message, because if this happens in the middle of a change-item action, you will have a data file which is only half full. You will then have to recover the old data file, which will have the .BAK extension, by using PIP. However, once the program is up and running, with all mistypings removed, you should not find that the error trap ever operates. Lines 70 and 80 contain messages which are used in places, and which you might want to use considerably more. You might also want to use these messages in windows of different colour, to draw more attention to them. Line 90 reads the words which number the headings, and the program starts in earnest in line 140. This prints a title, centred by the GOSUB 290, and asks if you need instructions. The GOSUB 1440 in line 150 is a time delay routine which will give you a delay of as many seconds as you assign to the integer T%. You are then asked if you need instructions. I have not written very detailed instructions, because these just involve a great deal of typing. There is enough to remind you what to do if you have not used the program for some time. You can type your own instructions if you have modified the program for your own use. By allowing the choice of skipping the instructions, you can get into the program faster if you use it frequently.

Line 170 clears the screen and then prints the menu. You are asked to select by number in line 190, using the GOSUB 300 subroutine and converting to number form with VAL. This number is assigned to K% (an integer) and tested. If the range is acceptable, then lines 210 to 240 carry out the choice. This is not completely straightforward, because choice 1 is very different from the others. It is used *only when a new type of file is to be created*, and it opens the file HEADS.DAT. Because of this, it has to be treated separately. This is carried out in line 210, and because one choice has been removed from the list in this way, the value of K% then has to be reduced by one. If the choice is to be any other item, the program must then check that the data which is contained in file HEADS.DAT is present. This is performed in line 220 by testing for the filename F$. If this is a blank, then the file HEADS.DAT must be read, using GOSUB 1400. If you have been using other parts of the program and have returned to the menu, this file will already have been read, and it won't have to be read again. When you reach line 230 you are definitely choosing one of the actions that will need the data file to be opened, so the subroutine which opens the files is used. Line 240 then makes the choice of the other subroutines, and when the subroutine is finished, line 250 closes the files again. This ensures correct file use unless the program is stopped within a subroutine. Lines 260, 270 give you a chance to return to the menu unless you have picked the 'END program' option. The subroutines carry out all of the main actions. This is important, because it makes the program very easy to change. Practically all the subroutines that you might need for your own 'custom' version are listed, so if you know in detail what each subroutine does, making your own version is relatively easy.

The creation subroutine

The subroutine that starts in line 440 creates a completely new file. This will replace any other file that has been created with this program on the same disc side, which is why it's useful to have several copies of the program on different disc sides. The loop that starts in line 460 gets title names for each field. The INPUT stage for this is handled by a separate subroutine in line 420, which allows the length of title to be tested. You might want to use this subroutine also to test changes to the file. Each title and length is assigned to an array variable E$, with four items. As always, you can change this to suit yourself. If all is well, then you are asked for a filename in lines 480, 490. Once again, it might be useful to test this to make sure that the name HEADS was not used, and that no extension is placed with this name. Line 510 adds the extension of DAT, and assigns this also to E$(0). This allows the headings and the filename to be recorded by one loop in line 520. Line 530 then opens the main file, using the filename that you have supplied. Only the filename is then recorded before closing the file. This ensures that the file of that name exists on the disc, so that any subroutine which starts with an OPENIN will locate this file. Unless this is done, you are likely to find a 'No file' message at some point. When this has been carried out, the program returns in line 540 – in this case, the return will be to the GOTO 260 command in line 210. You can then either return to the menu or end the program. Your titles and filename are now recorded, and the program is now ready to make use of this new file. You will *not* make use of this menu option again until you come to choose another subject for filing.

Writing to the file

Selecting the 'Start ENTRY in file' option in the main menu leads to the subroutine which starts in line 550. If this option is selected, it will replace any existing file. It is used, therefore, just after a new file has been created by the use of option 1. If you want to add items to a file, then option 3 should be used. Line 560 gives brief instructions, and the titles of each field are printed from the array E$ as reminders. No attempt has been made to restrict the size of each entry, and you might want to do this for yourself. For a new file, the record number J% has been assigned with starting number 1 in line 80. The test at the end of line 580 checks for the entry of the letter X as the first field, because this terminates the entry. Line 640 will then correct the count number, and the subroutine returns. When it returns, the action of closing files will ensure that all data has been recorded. The write line is 620, and this calls a subroutine which uses WRITE#9 rather than PRINT#9. The reason is that WRITE#9 allows strings to be separated more easily when the output is in a form such as WRITE#9,R$,S$,U$. If PRINT#9 is used in such a case, then the read (using INPUT#9,R$,S$,U$) will *not* separate the items correctly. Our examples so far have used PRINT#9 in a loop, which has side-stepped this problem. Note that when X is used to terminate an entry, this letter is not recorded.

Once a file has been opened with menu choice 1 and written to with menu choice 2, it can be further used by choices 3, 4 and 5. You would not normally use choices 1 and 2 again, but the remaining choices will come in for heavy use from now on. We'll start with the 'heavyweight' item, choice 3. This allows addition to a file, alteration of a record, or deletion of a record. So that all of these actions can be catered for, this menu choice leads to another menu in line 660. This in turn leads to three more subroutines.

Choosing addition to a file leads to the subroutine in line 1080. Since the files are serial, adding to a file really means reading and rewriting the whole of the existing file, and then leaving the file open so that more items can be added. The reading and rewriting of the existing file is done by the WHILE ... WEND loop in lines 1080 to 1100. In this loop, the counter variable J% is used to count the records as they are read and rewritten. Line 1110 then increments J%, and calls the original entry subroutine so that more items can be added. As before, entering the letter X as the first field of a record will terminate the addition of records. The rest of the file is written, and the files are closed when the routine returns.

When you take the 'change' option of this extra menu, you have to specify which item is to be changed. The subroutine starts in line 1120 with its title, and then calls the subroutine at line 1360 to find the number of the item. This is a convenient method from the point of view of easy programming, but if you want to find the item by letter, the changes to the program are not too difficult. When the number is specified, Z% is set to a value of one less than this chosen number. The subroutine at line 1370 is then used to read and rewrite all records up to this item. The chosen record is then read, using the GOSUBs at the end of line 1130. Line 1140 then displays the record, and lines 1150 and 1160 are used to change each field of the record. You could use a more elaborate routine here, allowing the RETURN key with no entry to mean that the field should be unchanged. You might also want to add the option of leaving the whole record unchanged if you decided that you didn't want to change it after all. Line 1170 writes the changed record to the buffer, and lines 1180 to 1200 write the rest of the file.

Listing the file

The file listing option in the main menu leads to line 700, prints the title of the choice, and asks for the options of screen or printer. This choice is made by pressing the P or S keys, and the choice is tested in line 730. The result causes Z% to be assigned with 0 for screen listing, or 8 for printer listing. This allows the expression PRINT#Z% to be used to give either type of output. The items are printed one on each line, with the item number. This item number can be used also in picking individual items for changing or deleting, using option 3. In line 780, if Z%=0, meaning screen output, the listing stops after each item to give you time to read it. You could, if you liked, modify this so that you had the choice of paused or continuous listing. Listing to the printer is *always* continuous.

When the fifth option, to pick an item, is chosen, the subroutine which starts at line 800 is used. This prompts for a choice of number or letter selection, which is made in lines 840, 850. These are dealt with by separate subroutines, with the number choice routine starting in line 870. Taking this option first, the number is entered, and the two subroutines at 1330 and 1340 are used to input the fields, using a WHILE ... WEND loop rather than a FOR ... NEXT because of the ease of detecting an impossible record number. If the item is found, then line 900 will print it and break the loop. If it is not found, then the EOF marker operates the WEND and the message in line 920 is printed.

If the letter selection method is chosen, the subroutine starts at line 960 with brief instructions. The letter or group of letters is entered in line 980, and the number of characters is assigned to Y%. The 'marker' variable FD% is also set to zero. A loop starts in line 990 which will input each field name and compare a number of letters equal to Y% with the letters that you have entered. If these two are identical, then all of the record are printed and FD% is changed to −1. The loop continues over the whole file, because there may be more than one entry which uses the same letter or group of letters. Line 1020 will print the message only if no item has been found, using the value of FD% to indicate this.

The last menu option is simply to end the program. This is necessary, because it provides a way of ending without having to carry out any of the other menu actions which would open files. You should *never* have to end a program by pressing the ESC key twice, because this can result in the program leaving files open, and so leaving you with an incomplete data file on the disc. This has been avoided in this case by having ON BREAK GOSUB 280 in line 30. This ensures that pressing ESC twice will cause the program to end by carrying out line 280, rather than by breaking, and it therefore safeguards the files *to some extent*. The safeguard is not perfect, however, because it depends where you break. If you are deleting an item, for example, part of the file will have been read and passed to the output buffer. If you break at this point, the first part of the file will be saved, but the second part will not. You will then have to use the .BAK copy to restore your file. To do so, copy and rename the .BAK file, using PIP with the name of the file which should be the new file. You will then still have the .BAK copy as your back-up. Another option, of course, is to prevent a break altogether by using ON BREAK CONT.

That's all there is to it. Taken as a whole, it looks rather intimidating, but when you split it into core and subroutines, as it was when it was written, it looks a lot simpler. It's by no means a polished piece of programming. You'll find, for example, that more use could be made of subroutines in some sections. You'll certainly find that you will want to modify parts of the program to suit your own needs. It's yours now, so modify it as you wish, but please don't sell it or publish it as your own work!

```
10 OPENOUT"DUMMY":MEMTOP=HIMEM
20 MEMORY HIMEM-1:CLOSEOUT
30 ON ERROR GOTO 1430:ON BREAK GOSUB 280
40 NL$=CHR$(10)+CHR$(13)
50 REM FILING CABINET by Ian Sinclair 19
85
60 DIM H$(4),E$(4)
70 M$="Please make sure that the disc is
 in"+NL$+"the drive, correct way up"
80 J%=1:Y$="Please answer Y or N.":Z$="P
ress any key..."
90 RESTORE:FOR N%=1 TO 4:READ H$(N%):NEX
T
100 REM Place in lines 110 and 120 any p
rinter setup instructions.
110 REM
120 REM
130 DATA First, Second, Third, Fourth
140 CLS:T$="Filing Cabinet":GOSUB 290
150 T%=2:GOSUB 1440:PRINT:PRINT"Do you n
eed instructions? ";Y$
160 GOSUB 300:IF UPPER$(K$)="Y" THEN GOS
UB 310
170 CLS:T$="MENU":GOSUB 290
180 PRINT:PRINT"1. Start NEW type of fil
e.":PRINT"2. Start ENTRY in file.":PRINT
"3. DELETE, CHANGE, or ADD items.":PRINT
"4. LIST complete file.":PRINT"5. PICK o
ne item.":PRINT"6. END Program."
190 PRINT:PRINT"Please choose by number.
":GOSUB 300:K%=VAL(K$)
200 IF K%<1 OR K%>6 THEN PRINT"1 TO 6 ON
LY- PLEASE TRY AGAIN.":GOTO 190
210 IF K%=1 THEN GOSUB 440:GOTO 260
220 K%=K%-1:IF F$="" THEN GOSUB 1400
230 GOSUB 1310
240 ON K% GOSUB 550,660,700,800,280
250 GOSUB 1320
260 CLS:PRINT"Do you want to return to t
he menu?"
270 PRINT:GOSUB 300:IF UPPER$(K$)="Y" TH
EN 180
280 CLOSEIN:CLOSEOUT:MEMORY MEMTOP:PRINT
"END":END
290 PRINT TAB(20-(LEN(T$)/2));T$:RETURN
```

Figure 5.2. The Filing Cabinet program.

```
300 K$=INKEY$:IF K$=""THEN 300 ELSE RETU
RN
310 CLS:T$="INSTRUCTIONS":GOSUB 290:PRIN
T
320 PRINT TAB(2)"This program allows you
 to set up and":PRINT"use a serial file
database. You will":PRINT"be asked to pr
ovide four titles, which"
330 PRINT"will be recorded along with a
filename. You can then use the other opt
ions to":PRINT"put entries into the file
 with your":PRINT"headings appearing as
prompts. You can"
340 PRINT"add to the file, change or del
ete items":PRINT"and list the file as yo
u wish."
350 PRINT:PRINT"The main restriction is
that you must":PRINT"NOT enter anything
which contains a":PRINT"comma. You need
Menu Item 1. only when":PRINT"you start
a new file for the first"
360 PRINT"time. For the rest of the time
 that":PRINT"this file is in use, the ot
her options":PRINT"apply. Keep one disc
side for each":PRINT"different file - yo
u can keep a copy of":PRINT"this program
 on each disc side as well."
370 PRINT
380 PRINT M$
390 PRINT:PRINT Z$
400 GOSUB 300
410 RETURN
420 INPUT Q$:IF LEN(Q$)>38 THEN PRINT"To
o long- please change now.":GOTO 420
430 RETURN
440 CLS:T$="New File Specification":GOSU
B 290:T%=2:GOSUB 1440
450 PRINT:PRINT"Now select your four tit
les for this":PRINT"new file, using RETU
RN after each ":PRINT"title. Only four t
itles can be used.":PRINT
460 FOR N%=1 TO 4:PRINT H$(N%);" is- ";:
GOSUB 420:PRINT Q$
470 E$(N%)=Q$:NEXT
480 PRINT"End of titles specification -
now we":PRINT"need a filename of up to e
ight":PRINT"characters - no more.":PRINT
490 INPUT"Filename is - ",F$
500 IF LEN(F$)>8 THEN PRINT"Too long - e
```

Figure 5.2 (contd)

```
ight characters only.":PRINT"Please try
again.":GOTO 490
510 F$=F$+".DAT":E$(0)=F$
520 OPENOUT"HEADS.DAT":FOR N%=0 TO 4:WRI
TE#9,E$(N%):NEXT:CLOSEOUT
530 OPENOUT F$:WRITE#9,F$:CLOSEOUT
540 RETURN
550 CLS:T$="Entry of Items.":GOSUB 290:T
%=2:GOSUB 1440
560 PRINT"Items can now be entered until
 you":PRINT"enter X as the first of a se
t."
570 PRINT"Entry No. ";J%
580 PRINT E$(1):INPUT Q$:IF UPPER$(Q$)="
X" THEN 640
590 PRINT E$(2):INPUT R$
600 PRINT E$(3):INPUT S$
610 PRINT E$(4):INPUT U$
620 GOSUB 1350
630 J%=J%+1:GOTO 570
640 J%=J%-1:PRINT "END of entry.":T%=2:G
OSUB 1440
650 RETURN
660 CLS:PRINT:PRINT"Do you want to- ":PR
INT:PRINT"1. ADD to the file.":PRINT"2.
CHANGE an item.":PRINT"3. DELETE an item
.":PRINT"4. RETURN to the main menu."
670 PRINT:PRINT"Please select by number.
":GOSUB 300:K%=VAL(K$):IF K%<1 OR K%>4 T
HEN PRINT"1 to 4 only- please try again.
":GOTO 670
680 ON K% GOSUB 1080,1120,1220,1300
690 RETURN
700 CLS:T$="FILE LISTING":GOSUB 290:T%=2
:GOSUB 1440
710 PRINT:PRINT"Do you want to use the s
creen or the":PRINT"printer for your lis
ting?"
720 PRINT:PRINT"Please press P or S key.
"
730 GOSUB 300: IF UPPER$(K$)<>"P" AND UP
PER$(K$)<>"S" THEN PRINT"P or S only - p
lease try again.":GOTO 730
740 Z%=0:IF UPPER$(K$)="P" THEN Z%=8
750 C%=1:WHILE NOT EOF:GOSUB 1330
760 PRINT#Z%,"Item ";C%;"_"
770 GOSUB 1340:GOSUB 1450
780 C%=C%+1:IF Z%=0 THEN PRINT Z$:GOSUB
300
```

Figure 5.2 (contd)

```
790 WEND:RETURN
800 PRINT:T$="PICK AND ITEM":GOSUB 290
810 PRINT:PRINT"You can pick by number (
N) or by ":PRINT"letter (L)."
820 PRINT:PRINT"Please press N or L key
now."
830 GOSUB 300:IF UPPER$(K$)<>"N" AND UPP
ER$(K$)<>"L" THEN PRINT"N or L only - pl
ease try again.":GOTO 830
840 IF UPPER$(K$)="N"THEN GOSUB 870
850 IF UPPER$(K$)="L" THEN GOSUB 950
860 RETURN
870 PRINT"What number item do you want?"
:PRINT"Type number, then press RETURN ke
y."
880 INPUT X%:N%=1
890 WHILE NOT EOF:GOSUB 1330
900 GOSUB 1340:IF N%=X% THEN CLS:Z%=0:GO
SUB 1450:GOTO 930
910 N%=N%+1:WEND
920 PRINT"Item not found."
930 T%=1:GOSUB 1440:PRINT:PRINT"press an
y key to return.":GOSUB 300
940 RETURN
950 CLS
960 PRINT:PRINT"Type first few letters o
f the first":PRINT"entry. Don't forget c
apital letters.":PRINT"If you use one le
tter only you will":PRINT"get all entrie
s which start with that":PRINT"letter.":
PRINT
970 PRINT"Press RETURN key after typing
letters.
980 INPUT T$:Y%=LEN(T$):FD%=0
990 WHILE NOT EOF:INPUT#9,Q$
1000 INPUT#9,R$,S$,U$:IF LEFT$(Q$,Y%)=T$
 THEN Z%=0:GOSUB 1450:FD%=1
1010 WEND
1020 IF FD%=0 THEN PRINT"Item not found"
1030 T%=2:GOSUB 1440
1040 PRINT Z$;" to return.":GOSUB 300
1050 RETURN
1060 REM Sub-menu routines
1070 REM start with add to file
1080 J%=0:WHILE NOT EOF
1090 GOSUB 1330:J%=J%+1:GOSUB 1340
1100 GOSUB 1350:WEND
1110 J%=J%+1:GOSUB 560: RETURN
1120 CLS:T$="CHANGE ITEM":GOSUB 290
```

Figure 5.2 (contd)

```
1130 GOSUB 1360:GOSUB 1370:GOSUB 1330:GO
SUB 1340
1140 PRINT Q$+NL$+R$+NL$+S$+NL$+U$
1150 PRINT E$(1);:INPUT Q$:PRINT E$(2);:
INPUT R$
1160 PRINT E$(3);:INPUT S$:PRINT E$(4);:
INPUT U$
1170 WRITE#9,Q$,R$,S$,U$
1180 WHILE NOT EOF
1190 GOSUB 1330:GOSUB 1340
1200 GOSUB 1350:WEND
1210 RETURN
1220 CLS:T$="DELETE ITEM":GOSUB 290
1230 GOSUB 1360:GOSUB 1370
1240 PRINT:PRINT:T$="Please wait...":GOS
UB 290
1250 GOSUB 1330:GOSUB 1340
1260 D$=Q$
1270 WHILE NOT EOF:GOSUB 1330:GOSUB 1340
1280 GOSUB 1350:WEND
1290 PRINT:PRINT:T$=D$+" DELETED!":GOSUB
 290:T%=2:GOSUB 1440:RETURN
1300 RETURN
1310 OPENIN F$:OPENOUT F$:RETURN
1320 CLOSEIN:CLOSEOUT:RETURN
1330 INPUT#9,Q$:RETURN
1340 INPUT#9,R$,S$,U$:RETURN
1350 WRITE#9,Q$:WRITE#9,R$,S$,U$:RETURN
1360 PRINT"Please type number of item.";
:INPUT Z%:N%=Z%-1:RETURN
1370 FOR J%=1 TO N%:GOSUB 1330:GOSUB 134
0
1380 GOSUB 1350:NEXT:RETURN
1390 REM Get headings and filename
1400 OPENIN"HEADS.DAT"
1410 FOR N%=0 TO 4:INPUT#9,E$(N%):NEXT
1420 F$=E$(0):CLOSEIN:RETURN
1430 PRINT"ERROR ";ERR;" IN LINE ";ERL:G
OTO 280
1440 START=TIME:WHILE TIME<START+300*T%:
WEND:RETURN
1450 PRINT#Z%,E$(1);": ";Q$+NL$+E$(2)+":
 ";R$+NL$+E$(3);": ";S$+NL$+E$(4);": ";U
$+NL$
1460 RETURN
```

Figure 5.2 (contd)

Chapter Six
Mastering the Screen

Windowing

In this section we shall concentrate on the screen output, and how it can be controlled using the BASIC of the CPC6128. You need screen output for many purposes; for business programs the screen is used mainly for messages, but for entertainment programs, the very extensive graphics statements of the CPC6128 come in for a lot of use. In this chapter we'll look at both aspects of the screen output.

The picture that you see on the screen is the result of electrical signals that pass from the computer to the monitor. The whole picture that appears on the monitor has been put together within the computer, however, and a great deal can be done to it at that stage. Just as you can control the signal paths, called 'streams', to the disc drive and to the printer by making use of the stream numbers 8 and 9, imagine also that the screen is controlled in the same way. I say imagine, because all of the control happens inside the computer, and what is sent to the monitor is always just a set of signals that provides a complete screenful of display – even if the screen is blank.

Just to show what I mean, take a look at the short program in Figure 6.1. Line 10 asks you to enter a four-letter password. In line 20, PRINT CHR$(21) will 'disconnect' the stream that normally passes everything from the keyboard to the screen. The signals go normally to the monitor, but what you type is not part of these signals! You can now enter your password, using the INPUT in line 30, but without the password appearing. Line 40 links the keyboard to the screen again so that messages can be displayed. As usual, the entry is tested so that you can try again if necessary. If you want to check that you did enter something, then PRINT B$ will reveal it. The stream that links the keyboard and the main screen is numbered 0, and if this stream is cut, then nothing that you type is seen on the screen. The CPC6128 makes provision for ten stream numbers, 0 to 9, of which numbers 0, 8 and 9 are allocated to screen, printer and disc respectively. These stream numbers are distinguished by using a hashmark (#) before them. One of the special effects that is available through the use of these stream numbers is called 'windowing', and in this part of the chapter we'll take a look at this effect and

```
10 PRINT"ENTER YOUR 4-LETTER PASSWORD NO
W"
20 PRINT CHR$(21)
30 INPUT B$
40 PRINT CHR$(6)
50 IF LEN(B$)<>4 THEN PRINT"Incorrect- p
lease try agsin":GOTO 10
60 PRINT"Thank you- password acknowledge
d."
70 PRINT"Type PRINT B$ to see it!"
```

Figure 6.1. Concealing an input by shutting off the stream to the screen.

a few others that produce screen displays that are rather more interesting than plain text or figures. Windowing is a good place to start, because unless you have used one of the few machines that allows this effect to be programmed easily, you probably won't have tried this for yourself.

A 'window' simply means a section of screen which can be used independently of other sections and even independently of the screen as a whole. A window can have text printed on it, and can be scrolled or cleared irrespective of what is happening on other parts of the screen. Try this: fill the screen with a listing, or any other text. Now type as a direct command:

WINDOW 15,25,2,6 (then RETURN)

Now use CLS. You'll see that only a small piece of the screen, which is the window, will clear. Try typing a short program into this window. You will find that it behaves as if it were the only screen you have. It automatically selects a new line when it needs to and scrolls when it needs to, just like a full screen. Meanwhile, the rest of the screen remains unaffected. A CLS will clear only the window, not the rest of the screen. To get back quickly to normal, type MODE 1 (RETURN).

If that whetted your appetite, it's time to look at how we can control this window business for ourselves. WINDOW must be followed by four numbers. These numbers use the same value range as the LOCATE numbers in each possible screen mode. For mode 1, for instance, these numbers are 1 to 40 across the screen and 1 to 25 down the screen, and they specify, in order, the left, right, top and bottom positions of the window. When you use this type of WINDOW command, all of your printing will be to this window. It uses stream #0, which is the normal stream for PRINT and other commands of this type. You can, however, start the WINDOW specification with a stream number, like #2. When you do this, the window is used only when you have an instruction like PRINT #2,"Window".

Figure 6.2 shows this type of window in action. The whole screen is cleared, and then a window is connected to stream #2 by using the command:

```
10 CLS
20 WINDOW#2,10,30,1,5
30 WINDOW#3,12,28,10,15
40 PRINT#2,"This is window #2 in use"
50 GOSUB 150
60 PRINT#3,"Look at the text in window #
2 - this is window #3"
70 GOSUB 150
80 CLS #2
90 GOSUB 150
100 CLS#3
110 GOSUB 150
120 PRINT#2,"Window #2"
130 PRINT#3,"Window #3"
140 END
150 FOR N=1 TO 2000:NEXT:RETURN
```

Figure 6.2. Using streams #2 and #3 for windows.

WINDOW#2,10,30,1,5

Now what this means is that stream number 2 is being connected to a screen window. The numbers that follow #2 and the comma specify both the size and position of this window. Its left-hand side will be at column 10, and its right-hand side at column 30. Its top will be on line 1, and its bottom on line 5. The effect of this command, then, is to allow you to use commands like CLS#2 and PRINT#2 to clear this window or print to it. This action is selective. Unless you close the stream or allocate it to something else, the window will be controlled by using stream #2. Another big difference is that the whole screen can still be cleared and printed on. Clearing the whole screen will also clear this window, and printing on the whole screen will print over the window, though you can clear the window selectively again with a CLS#2.

In the example, then, the first window is created in line 20, and another window further down the screen is created in line 30. To prove that these windows exist, line 40 prints a phrase onto window #2. You can, incidentally, put LIST#2 into your program to list your program on this window, but if you do this, the machine will not execute any program lines that follow the LIST statement. Then there is a pause caused by the subroutine at line 150. After the pause, more text is printed to the other window, #3. The rest of the program then illustrates how the windows can be cleared and printed on separately. Notice, however, at the end of the program, how the 'Ready' message and the cursor appear at the top left of the screen. The main screen, whose stream number is 0, will always override any windows.

Some more window magic can be worked by using another window command, WINDOW SWAP. Suppose, for example, that you want to carry out some editing on a program. Normally, as you edit, the screen scrolls while you edit and fill in lines. If you have listed a number of lines that you want to edit, this can be very annoying because as you work on the first few lines, the rest of them scroll off the screen. The program in Figure 6.3 gets around this problem by creating two windows, using stream numbers #6 and #7 which split the screen into two parts. The effect of WINDOW

```
10000 CLS
10010 WINDOW#7,1,40,1,19
10020 PAPER#7,1:PEN#7,0:CLS#7
10030 WINDOW#6,1,40,21,25
10040 WINDOW SWAP 0,6
10050 PRINT"THIS IS AN EDITING AREA"
10060 PRINT:PRINT"To restore normal scre
en type"
10070 PRINT TAB(12)"MODE 1 [RETURN]"
10080 LIST#7
```

Figure 6.3. How to make use of windows during editing. Add this routine (a David Foster special) to the end of each program you want to edit.

SWAP 0,6 then has the effect of making window #6 act as the main stream, taking everything that would normally go to stream #0, the main screen stream number. This window is at the bottom of the screen. In the example, a message is printed in this window, and line 10080 causes a listing to appear in the upper part of the screen. The 'Ready' prompt, however, goes to the 'main screen', which is now the lower window. If you move the cursor now, you will see that only the lower part of the screen will scroll, leaving the listing in the upper part intact. You can now use commands such as EDIT 10070, see the edit line come up, change it, and relist on window #7, after using a CLS#7 if you want to clear the old listing. You can merge this routine into your own programs, so that when you want to edit you can type GOTO 10000 and then make use of the routine.

COPYCHR$ and CURSOR

These are two new statements (new, that is, to former owners of the CPC464) which are available on the CPC6128. The CURSOR statement allows you to turn the cursor on or off as you wish. The cursor is usually turned off by the operating system when a program is running, and appears only when INPUT is used, but not, for example, when INKEY$ is used. Using CURSOR 1,1 will ensure that the cursor is on, and CURSOR 0,0 ensures that it is off. The manual refers to the 'System switch' and the 'User switch' for these numbers. The first number is the System switch number, and means the control of the cursor by the operating system. The second number, the User switch is your own independent cursor control, and the cursor is on only when both numbers are 1, or when only the first number is provided and is 1. Normally, the use of the first number alone is enough for all purposes.

The other statement, COPYCHR$, is very useful, but its usefulness is not well illustrated in the manual. COPYCHR$ allows you to read whatever character is under the screen cursor, and its most obvious use is as another form of collision detector in games. It can, however, be used for more serious purposes as the listing in Figure 6.4 illustrates. In this program, a disc

```
10 ON ERROR GOTO 240
20 MODE 2:CAT
30 LOCATE 1,2:PRINT"Position cursor and
press RETURN to select."
40 ac%=1:do%=4
50 CURSOR 1,1
60 WHILE INKEY(18)<>0
70 LOCATE ac%,do%
80 WHILE INKEY$="":WEND
90 do%=do%+(INKEY(2))-(INKEY(0))
100 ac%=ac%+20*(INKEY(1))-20*(INKEY(8))
110 IF do%<4 THEN do%=4 ELSE IF do%>19 T
HEN do%=19
120 IF ac%<1 THEN ac%=1 ELSE IF ac%>61 T
HEN ac%=61
130 WEND
140 CURSOR 0,0
150 A$="":FOR N%=0 TO 11
160 LOCATE ac%+N%,do%
170 A$=A$+COPYCHR$(#0)
180 NEXT
190 LOCATE 1,20
200 IF MID$(A$,9,1)<>CHR$(46)THEN ERROR
200
210 PRINT"You have selected ";A$:CLEAR I
NPUT
220 PRINT"LOAD, RUN OR CANCEL? L/R/C ":W
HILE INKEY$="":WEND
230 IF INKEY(36)=0 THEN LOAD A$ ELSE IF
INKEY(50)=0 THEN RUN A$ ELSE RUN
240 IF ERR=200 THEN PRINT"Invalid select
ion":RESUME 40 ELSE RESUME
250 END
```

Figure 6.4. Using COPYCHR$ in a disc directory selection program. This is another David Foster special which deserves to be on your system disc copy.

catalogue is printed, and you are asked to select a file to load or run. The cursor is switched on while the directory is displayed, and the position numbers ac% and do% are assigned for the across and down positions for LOCATE. The cursor is turned on, and the WHILE loop will then continue until the RETURN key (key 18) is pressed. The next WHILE ... WEND loop in line 80 then holds everything until you press a key. If this is the RETURN key, then both loops terminate, but if you press the up or across shift keys, then lines 90 and 100 make changes to the values of ac% and do%.

To see how this works, consider what has happened if the loop in line 80 has been broken by pressing the down arrow key. Remember that INKEY gives −1 if the key is up, and 0 if the key is pressed. If you pressed the down arrow key, then INKEY(2) will give 0, and INKEY(0) will give −1. The expression in line 90 is then 0−(−1), which is +1, and so 1 is added to do%. If it had been the other way round, then you would have had the expression −1−0, which is −1, subtracting 1 from do%. If neither of these keys has been pressed, you get −1−(−1), which is 0; no change in do%. Line 100 deals similarly with the left and right keys, but using a 'multiply by 20' step so that the cursor jumps from the start of one filename to the start of the next, rather than by just one character. Lines 110 and 120 are then needed to ensure that the cursor cannot be moved off the screen. You cannot, however, prevent the cursor from being moved out of the directory display, because the program cannot predict how long the directory will be. This has to be dealt with by other methods.

When a selection has been made, then, the cursor is restored in line 140, and a string A$ is prepared in line 150. A loop starts which will read 12 characters of a filename, which is the length of any valid filename. Line 160 locates the start of the chosen filename, and in this loop, A$ is packed with characters taken from the screen display by COPYCHR$(#0) – note that you need to specify the stream here. The cursor is then (invisibly) relocated so as to show the selection, but first of all the string must be tested. The test is for the presence of a full-stop (period) in position 9 where it will be for a valid filename. This test is not infallible, because it's possible that another string might have a full-stop in this position, but at least it guards against placing the cursor over a blank space or over the message about the remaining disc space. If the full stop is not present, then ERROR 200 is generated. There *is* no ERROR 200 in the standard list of errors, and this is a 'user error number', used to trigger the ON ERROR GOTO line. This line, line 240, can detect error 200 and use it to print a message. For any other error, such as a syntax error, normal service is resumed with only the standard message displayed. If the selection has been valid, line 220 gives you the choice of load, run or cancel, with a WEND loop containing INKEY to detect the key that is pressed. The L and R keys produce the load and run effects respectively, but any other key is taken to be a cancel instruction, and it causes the program to run again. The whole program, which is by David Foster, is not only an excellent illustration of the use of CURSOR, COPYCHR$ and error trapping, but also a very useful routine in its own right. It deserves a place on each of your AMSDOS discs.

Write it in colour

The program in Figure 6.3 made some use of colour changes, and it's time now to pay some more attention to the colour instructions of CPC6128. There are four particularly important ones, which use the instruction words BORDER, PAPER, PEN and INK. We'll start with the simple one, BORDER. BORDER can be used with one number, or with two. If you use one colour number you will get a border round your main screen area, which is of one steady colour. The colour numbers are listed in Figure 6.5. If you use two colours with BORDER, the border will flash between these two colours.

Number	Colour
0	Black
1	Blue
2	Bright blue
3	Red
4	Magenta (red light + blue light)
5	Mauve
6	Bright red
7	Purple
8	Bright magenta
9	Green
10	Cyan (blue light + green light)
11	Sky blue
12	Yellow
13	White
14	Pastel blue
15	Orange
16	Pink
17	Pastel magenta
18	Bright green
19	Sea green
20	Bright cyan
21	Lime green
22	Pastel green
23	Pastel cyan
24	Bright yellow (gold)
25	Pastel yellow
26	Bright white

Figure 6.5. The Amstrad colour numbers, which are arranged in order of increasing brightness.

```
 10 REM LIST BEFORE RUNNING
 20 WINDOW#2,35,40,5,5
 30 FOR N=0 TO 26
 40 PRINT#2,N
 50 BORDER N
 60 GOSUB 150
 70 NEXT
 80 GOSUB 150
 90 FOR N=1 TO 26
100 PRINT#2,N;",";27-N
110 BORDER N,27-N
120 GOSUB 150
130 NEXT
140 END
150 START=TIME
160 WHILE TIME<START+600:WEND
170 RETURN
```

Figure 6.6. A program which demonstrates the BORDER colours, using TIME to create a delay.

Figure 6.6 gives you a taste of all this. The program first of all defines a window that is going to be used to display the colour numbers on the right-hand side of the main screen. It then starts a loop which will run through all of the available BORDER colours. Each colour is held on the screen by using a time delay. Unlike previous time delays, this is obtained by using the built-in timer of the machine. TIME is a number which starts from zero when you switch on the machine, and which is incremented 300 times per second. Now if you set up the subroutine that is shown in lines 150 to 170, you are setting a variable equal to the value of TIME at some instant. Each second later, TIME will have increased by 300, so a WHILE . . . WEND loop that waits for TIME to become 600 more than START is going to cause a delay of 2 seconds. It's simple and elegant, and easier to obtain precise times than a FOR . . . NEXT loop. Meanwhile, back at the program, after running through the single colours, the program goes through the flashing colours. The rate of flashing is set by another variable, and you can alter it for yourself. While the border continues flashing at the end of the program, try typing SPEED INK 10,10. When you press RETURN, you will see the flashing rate speed up. Then type SPEED INK 20,100, and watch the effect. The first number measures the time for one colour, the second number measures the time for the second colour. The numbers are fiftieths of a second (for the UK machine; sixtieths for the US version), so that a number 50 (UK) should give you a one second burst of one colour. Be careful how you use this command – some flashing rates of around 8 per second can be harmful to anyone who is subject to epilepsy. If you have ever tried to help an epileptic, you won't want to take risks with flashing borders.

Quite apart from being unpleasant, a flashing border can also be a nuisance. You will see that as the border flashes, the size of the picture on your monitor or TV changes. This is because these units are not designed to cope with flashing pictures – a monitor which could would be too expensive for most of us (allow something like £800 if you are thinking of saving for one!). To stop your border flashing, type BORDER 1 and press RETURN. A mode change does not affect the border.

The next items are PAPER, PEN and INK. The names tell some of the story but not all. In particular, if you have come to the CPC6128 after serving an apprenticeship on a Spectrum, you will find these terms slightly confusing. If you are completely new to the terms, you will still find them confusing – so stick with me! The one to start with is INK.

INK means a colour, and for each mode you have a limited range of INKs that you can use. Mode 0 allows you to use up to 16 different colours of INK on the screen. This means any 16 selected from the list of 27 possible colours in Figure 6.3. MODE 1, the normal text mode, allows you to use up to 4 INK colours. Once again, of course, you can choose which 4 of the 27 available colours you want to use. MODE 2, the high-resolution, 80 characters per line mode, allows only 2 INK colours, any 2 of the 27 that you care to use. A good way to think of INK is that you have a paint box with a limited number of pots. For MODE 1, for example, you have 4 pots. You can put any of your 27 colours in each pot – but you may paint only with the colours in the pots. If you want to use other colours you have to refill the pots!

Now this is where the CPC6128 starts to be very different from other machines. In MODE 1, your INK *numbers* are 0,1,2 and 3. I've used numbers, not colours, here, because these numbers are *not* the colour numbers. Coming back to our comparison with a paintbox, these numbers are just the numbers of the *pots*. We can use the INK command to decide which colours to put into the pots. The form of the command is, for example, INK 2, 7. This will fill pot 2 with ink whose colour is 7 (purple). When you start up in MODE 1 the pots are filled for you. Pot 0 is filled with blue, and this is the pot that decides the colour of your screen background. Pot 1 is filled with gold paint (actually called bright green), colour 18. This is the pot that is used for text colour. Pot 2 is filled with colour 20, bright cyan, and pot 3 is filled with colour 6, bright red. Figure 6.7 reminds you of these settings for each mode, which are useful to know.

How do you change these colours? The answer is by using the INK command. In MODE 1, you can use INK colours of 0 to 3 (you can use numbers greater than 3, but you will just get the same range used over again). To allocate INK pot number 1 to colour 15, for example, simply type: INK 1,15 – don't forget the space between the 'K' of INK and the number. If you type two colour numbers, separated by a comma, your pot will contain flashing ink! The colour will be flashing between the two colour numbers you have specified. For example, INK 1,3,7 will give you an ink that flashes between colours 3 and 7. You can alter the rate of flashing by using SPEED INK as before. It's rather like these old jokes about striped paint!

When you switch on, the computer puts you into MODE 1, and allocates the 'default' INK pot colours shown in Figure 6.7. It also decides which of

Mode 2

PAPER/PEN No.	*Colour No.*	*Colour*
0	1	Blue
1	24	Bright yellow

Mode 1

PAPER/PEN No.	*Colour No.*	*Colour*
0	1	Blue
1	24	Bright yellow
2	20	Bright cyan
3	6	Bright red

Mode 0

PAPER/PEN No.	*Colour No.*	*Colour*
0	1	Blue
1	24	Bright yellow
2	20	Bright cyan
3	6	Bright red
4	26	Bright white
5	0	Black
6	2	Bright blue
7	8	Bright magenta
8	10	Cyan
9	12	Yellow
10	14	Pastle blue
11	16	Pink
12	18	Bright green
13	22	Pastel green
14	1,24	Flashing blue/yellow
15	16,11	Flashing pink/blue

Figure 6.7. The default settings of INK in each mode.

the pots shall be used for background and which for text. The background, or PAPER, uses INK number 0, and the text, or PEN, uses INK number 1. Now I must emphasise yet again, particularly if you are a former Spectrum owner, that these INK numbers are *not colour numbers*, just the inkpot numbers. The colours that appear on the screen depend on which pot you use, and what colour of ink was put into it. By using the PAPER command, you can alter the number of the inkpot you use for background. By using PEN, you can alter the number of the inkpot you use for text.

Figure 6.8 illustrates PAPER and PEN in use. An instruction such as PAPER 2 does not, by itself, cause colour to appear. If we print on the screen following a PAPER 2 instruction, the background for our printing will appear in bright cyan, but only for the part on which we have printed.

```
10 BORDER 0
20 FOR N=0 TO 3
30 PAPER N:CLS
40 GOSUB 140
50 NEXT
55 PAPER 0:CLS
60 FOR N=0 TO 3
70 PEN N
80 GOSUB 170
90 GOSUB 140
100 NEXT
110 GOSUB 140
120 PEN 1
130 END
140 START=TIME
150 WHILE TIME < START+600:WEND
160 RETURN
170 PRINT"This is some text to show the
effect of colour of PEN";N
180 RETURN
```

Figure 6.8. Making use of PEN and PAPER commands.

To make PAPER 2 colour a complete screen, we have to follow it with a CLS instruction. That's illustrated in line 30. The second part of the program uses PAPER 0, which is dark blue because that's the colour of paint in pot number 0, and prints in different PEN colours. You can see from the results of this program that if you want to keep your text clear, you have to use contrasting colours. Colours whose brightness values are very close to each other will never give enough contrast to make a good display. My own preference is to have the PAPER colour dark, and the PEN colour light, because the opposite, a light screen with dark print, can appear to flicker very irritatingly. Don't expect the letters to appear in very good colours if you are using a TV receiver, because colour TV sets are not very good at displaying colour in small chunks. Add to that the fact that 90% of the male population is partially colour blind, and you'll see that the most impressive colour displays are the ones that use strong colours in big areas, displayed on a monitor.

When you run the program, you'll see that the text line for PEN 0 never appears. That's because inkpot 0 is the one that has been used for background, paper, colour. If you change the background colour before you write the text, however, this line will appear, and one of the others will not. I have used MODE 1 for illustrations here because the choice of four inkpots is a convenient one. Remember, however, that you have the choice of two inkpots, 0 and 1, in MODE 2, and of sixteen inkpots numbered 0 to 15 in MODE 0.

```
10 BORDER 0
20 WINDOW#1,1,40,3,5
30 WINDOW#2,1,40,7,10
40 WINDOW#3,1,19,11,24
50 WINDOW#4,20,40,11,24
60 A$="This is a test phrase to show off
 the windows."
70 PAPER#1,0
80 PAPER#2,1
90 PAPER#3,2
100 PAPER#4,3
110 PEN#1,3
120 PEN#2,2
130 PEN#3,1
140 PEN#4,0
150 CLS
160 FOR N=1 TO 4
170 PRINT#N,A$:NEXT
```

Figure 6.9. PAPER and PEN use in different windows.

The PEN and PAPER commands can be used to specify the colours that are used in windows. Figure 6.9 illustrates this, and shows that not all colour combinations are good ones! When you want to use different PEN and PAPER colours in the windows, take a good look at the colours first and check that they are really compatible. Sometimes the combination of two shades of one colour can be quite effective, but two light colours or two dark colours will never be very satisfactory, especially if you are using a colour TV for display. Some of these combinations are not satisfactory even on a monitor.

Some prettier printing

The best place to start on our next bit of exploration of special text effects is with INK again. Take a look for starters at the program in Figure 6.10 which illustrates flashing text. The flashing is obtained by using 'flashing ink' in

```
10 CLS
20 INK 2,6,8
30 INK 3,12,19
40 PEN 2
50 PRINT:PRINT"ATTENTION PLEASE!"
60 PRINT:PRINT:PRINT
70 PEN 3
80 PRINT"WARNING- DANGER!"
90 PEN 1
```

Figure 6.10. Creating flashing text by using flashing INK.

pots 2 and 3 in lines 20 and 30. When we use PEN to dip into these inks, the result is flashing text as you can see when lines 50 and 80 run. Line 90 switches back to PEN 1, the normal inkpot for text, so that the 'Ready' prompt doesn't flash as well. Note that the letters keep flashing for as long as they are on the screen. You can change the flashing colours instantly, or change to a non-flashing colour, by altering the INK. Figure 6.11, for example, shows how a title can be made to flash for a few seconds, and then revert to another steady colour. This is a good way of making sure that flashing achieves its effect of drawing attention to something, without boring the user.

```
10 INK 2,24,6
20 CLS
30 PRINT:PEN 2
40 PRINT TAB(14)"FLASHING TITLE"
50 FOR N=1 TO 3000:NEXT
60 INK 2,20
70 PEN 1
80 PRINT:PRINT"Now we can place text on
the screen":PRINT"without being distract
ed!"
```

Figure 6.11. Using an INK change to revert from flashing text to a steady display.

Another useful feature of the CPC6128 allows you to carry out effects like underlining, inserting foreign accents, and so on. This, and many other special effects that we shall look at later, is enabled by one of the 'non-printing' codes of the CPC6128, the codes 1 to 31. We have already used code 21 (turn off screen), 6 (turn on screen) and 8 (move cursor one step back). Your manual carries a full list of these effects starting at page 3 of Chapter 7. To underline text, we need to alter the way printing is carried out. Normally, when we print, the new print completely replaces the old. By using CHR$(22)+CHR$(1) we can alter this, superimposing new text on to old. In the program in Figure 6.12, then, the phrase which you input in line

```
10 CLS
20 INPUT"Enter a word or phrase. ",A$
30 LOCATE 1,10:PRINT A$;
40 GOSUB 1000
50 END
1000 REM Underline routine
1010 PRINT CHR$(22)+CHR$(1);
1020 S$=STRING$(LEN(A$),8)+STRING$(LEN(A
$),95)
1030 PRINT S$
1040 PRINT:PRINT CHR$(22)+CHR$(0)
1050 RETURN
```

Figure 6.12. Underlining with CHR$(22). This uses a subroutine which you can adapt for your own purposes.

20 is printed in line 30, and the underlining subroutine is called. The PRINT CHR$(22)+CHR$(1) causes the superimpose to be switched on, and then the string S$ is defined as the correct number of backspaces followed by the same number of underline characters. The number is found by using LEN(A$) to measure the string length. When S$ is printed, then, it will move the cursor back to the start of the word or phrase and print a set of underline characters. The subroutine then prints CHR$(22)+CHR$(0) to switch off the underlining.

Graphics

Any modern computer is expected to be able to produce dazzling displays of colour and other special effects. The CPC6128 is no exception, and in this section, we'll start to look at some of the more advanced graphics effects that are possible. I'm going to assume that you already know what is meant by terms such as pixels and resolution, because these are covered in the elementary books. In its lowest resolution mode, mode 0, the CPC6128 uses 32000 pixels, and in mode 2, the highest resolution mode, you have 128000 pixels. The CPC6128 does not steal your 43533 bytes (42.5K) of program memory for graphics use. Instead, it has a reserved (dedicated) piece of memory that is used for graphics. You'll find, then, that if you have a program which produces mode 0 graphics and you change over to mode 2 graphics, you don't lose any program memory, though you can't use the same colour range in mode 2.

In this first section we have been working with what is called the 'text screen', the normal arrangement of screen. This text screen can be used only for text, meaning characters that are put into place by the PRINT instruction. As we shall see, there are other instructions that can be used with 'graphics screens', and we'll be looking at these later. I shall assume, once again, that you already know about the keyboard graphics shapes and the low-resolution graphics instructions, using CHR$.

Character redefinition

Using the character redefinition facilities, the CPC6128 offers an interesting way of producing graphics on the ordinary text screen, using only the PRINT instruction to place the patterns. These could be classified as 'low-resolution' in the sense that they use the same limited number of PRINT positions on the screen, but they offer much more scope for useful effects in programs that make use mainly of the text screen.

We start, logically enough, with planning. The size of the shape we're talking about is one screen character, the size of the cursor block. Now this, and every other CPC6128 character, is made of up to 64 dots arranged on an 8 by 8 grid. Figure 6.13 shows the shape of this grid – you can redraw it for

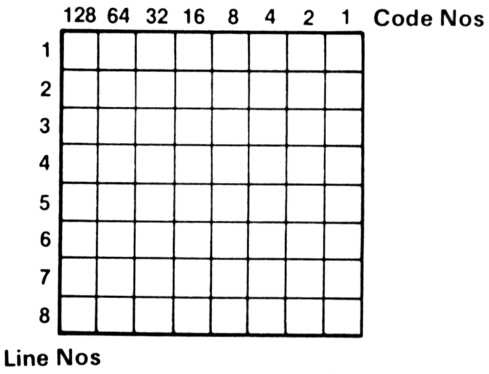


Figure 6.13. The grid for designing your own character shapes.

yourself on a sheet of graph paper if you want more copies. The important point is that the small squares of the grid represent dots on the screen that can be in either INK or PAPER colour, according to the value of code numbers that we use to instruct the computer. When a character is designed on this 8×8 grid, we normally use only 7 dots across and 7 down, leaving the right-hand column and the bottom row unused. This is because we want to have a space between any 2 characters, and between any 2 lines of text on the screen. If you are designing your own graphics shapes, however, you might want to make them fill the whole 8×8 block, and so join on to each other when you print them on the screen.

Now the CPC6128 manual shows you very briefly how to design these shapes, but unless you have done it before, you may be rather puzzled. The key to it is the numbers that are printed on top of each column of squares in Figure 6.13. Each number is a code for any square in the column underneath it. Use the number, and the square will be in INK colour. Use 0 instead, and the square will be in PAPER colour, which means invisible. An example will help to make this clearer, and it appears in Figure 6.14. I've used a simple shape of a 'spacewalker' to illustrate the principle.

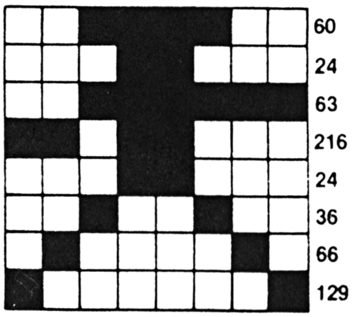


Figure 6.14. An example of creating a character shape.

The first line of squares has just four squares shaded in. I usually work on tracing paper clipped over the grid pattern, but in this example, I've shown what it will look like on the graph paper itself. The shaded squares in the top line are the ones that we want to appear in INK colour, and they are under the code numbers 32, 16, 8 and 4. There's nothing else shaded in this line, so we add the numbers 32, 16, 8 and 4, to get 60, which is the number we note at the side. Similarly, in the second line down, the squares in the 16 and 8 positions are shaded, so the number we use is the sum of these, 24. We continue in this way until all 8 lines have been dealt with.

There are a few points to note. One is that if none of the squares in a line is shaded, the code number is zero. The other point is that you can save a lot of arithmetic by remembering that a complete set of shaded squares in a line adds up to 255. This is the maximum size that you can use as a code number for a text character. You must always end up with 8 numbers, no matter what shape you are trying to produce.

The next matter is how we instruct the computer to produce the shape. What we have to do is store the code numbers in the CPC6128's memory, along with an ASCII code number that we will use to obtain the shape on the screen. This makes use of a new instruction, SYMBOL. SYMBOL has to be followed by *nine* numbers. The first number is the ASCII code we want to use. In this way, pressing a key or using this ASCII code will give our own pattern. The next eight numbers in SYMBOL are just the pattern numbers we have already found. All of the numbers are separated by commas, there must be a space between the 'L' of SYMBOL and the first number, and the pattern numbers are read from top to bottom of the shape. The effect of SYMBOL is therefore to place the code number into the memory.

Now that may seem all very well, but what ASCII numbers can we make use of? The answer is that you can normally use ASCII codes 240 to 255, a total of 16 codes for this purpose. Twelve of these codes are normally provided by the cursor keys at the top right-hand side of the keyboard – there are 12 because the SHIFT and CTRL keys can be pressed as well as the cursor keys. Using the codes for your own purposes, however, *does not* *affect the action* of these keys, so you do *not* get a spacewalker pattern when you press a cursor key! Let's suppose that we want to make our spacewalker pattern appear when CHR$(244) is used. Figure 6.15 shows what is needed – and it's not much. The ASCII code number is 244, and it's followed by the set of 8 numbers that map out the shape. We can then produce the shape wherever we like by using PRINT CHR$(244). You can, of course, use hex or binary for entering the 8 numbers. If you use hex, the numbers have to be prefixed with &, and a binary pattern needs to be prefixed with &x. Using the binary patterns makes entry very simple, because you only have to enter a 0 for a space and a 1 for a shaded block. This, however, requires a lot of typing, and can look rather untidy in a listing.

```
10 CLS
20 SYMBOL 244,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12)CHR$(244)
```

Figure 6.15. The listing for creating the spacewalker.

Now you aren't limited to creating characters for use by ASCII codes 240 to 255. You can use any character number you like from 0 to 255 onwards. This means that you can redefine *any* key code, and so make characters that will appear when you press the ordinary keys! Figure 6.16 shows what is needed to do this. In line 10, we have now added SYMBOL AFTER 90. This

```
10 CLS:SYMBOL AFTER 90
20 SYMBOL 92,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12);"\"
```

Figure 6.16. Redefining a key code to use the new character shape.

means that we can make our own shapes ('redefine') any character that has an ASCII code of 90 or more. We have chosen 92, which is the ASCII code for the backslash sign, the key next to the right-hand SHIFT. When you RUN this program, you'll see the spacewalker symbol printed in response to the PRINT TAB(12);"\" instruction this time. You can, of course, use CHR$(92) if you like. More important, though, you will see the spacewalker character appear each time you press the '\' key from now on. To get back to normal, save your program, and press SHIFT CTRL ESC to reset the machine.

There's an important point here, however. None of the other codes greater than 90 are affected by this change, only the one which you have redefined. Of course, if you want to, there's no reason why you shouldn't redefine each key to give a different symbol – but it's hard work. Just as a bit of fun, take a look at Figure 6.17 which redefines all the keys as spacewalkers. When this program has run, anything on the screen, apart from the cursor, will be a spacewalker! Even messages like 'Syntax error' and 'Ready' come out as a string of spacewalkers! It's a good way of ensuring that no-one messes about with your computer when you leave it!

```
10 CLS:SYMBOL AFTER 32
15 FOR N=32 TO 127
20 SYMBOL N,60,24,63,216,24,36,66,129
25 NEXT
```

Figure 6.17. Redefining all the keys – which can make typing very confusing!

You will need to use SHIFT CTRL ESC to restore normal service, and remember that this will remove your program from the memory.

High resolution graphics

The CPC6128 contains an excellent variety of graphics commands which allow you to draw patterns. These commands are more extensive than those of the CPC464, particularly with regard to the use of colour, and several extra commands have been provided along with additional parameters for existing commands. All of these commands make use of the X and Y coordinate system that we have already come across with LOCATE. The difference now is the numbers you can use. For all the graphics modes, the range of X numbers is 0 to 639, and the range of Y numbers is 0 to 399. This is very convenient, because it means that if you have developed a graphics program for one mode and you want to run it in another, you don't have to go through the program changing all the numbers. Figure 6.18 gives a set of instructions for making a graphics planning map for yourself, using inexpensive graph paper.

---

1. Take an A4 size sheet of graph paper, scaled in mm and cm. Chartwell and Guildhall make suitable graph papers.
2. Write scales, numbering the long side in steps of 25 up to 600. Number short side in steps of 25 up to 400.
3. Use tracing paper over this graph to draw your outlines and read co-ordinates.

Figure 6.18. How to construct a low-cost graphics map.

There are important differences between working with text and working with graphics, however. When you work with the graphics commands, you operate directly on the pixels, so that the distances that the units of X and Y numbers represent are much smaller. In addition, the origin of the graphics is different. The 'origin' is the place on the screen that is referred to by X=0 and Y=0. For the text screen you can't use zero, but LOCATE 1,1 means left-hand side, top of the screen. For graphics, the point X=0, Y=0 (usually referred to as the point 0,0) is at the left-hand side *bottom* of the screen. The Y distances are measured *upwards*, and the X distances *across* from left to right. This is the same place as is used as the 'origin' on most graphs. If you have been accustomed to drawing graphs, you will find graphics commands relatively simple to learn. Since we're on the topic of graphs, we'll start with a graph drawing command, PLOT.

Plotting it out

Drawing graphs is one very important aspect of graphics which is essential in any machine that can be used seriously for business or educational purposes. A graph is a set of points which can be joined and which should convey some information. The CPC6128 will plot graph points for you, using the PLOT or PLOTR commands. The colour of the point that is plotted will be whatever INK colour happens to be in use at the time. The difference between these two commands is that the PLOT command uses absolute coordinates and the PLOTR command uses coordinates measured *relative to the cursor position*. For example, if you use PLOT 0,0 the plot will be at point 0,0 which is the bottom left-hand corner of the screen. If you use PLOTR 0,0 the plot will be wherever the cursor happens to be. The cursor, throughout any graphics commands, is not the usual block you see when using text. The graphics cursor is invisible, so you don't see any evidence of it until you use a command that leaves pixels in a colour which is different from the background colour. The ordinary text cursor is restored when a graphics program ends.

Figure 6.19 shows the CPC6128 being used to plot three graphs at the same time, and doing so at a reasonable speed. The range of X in line 30 is set to cover the whole width of the screen, and for each value of X, three values of Y are calculated. One is obtained from the sine of the angle whose value is X/639, in radians. Another is obtained from the square of the sine of this angle, and the third is obtained from the cube of the sine of the angle. Each point is plotted by the PLOT commands in lines 50, 70 and 90, using different INK colours for the three different graphs. The reason for the

```
10 MODE 0
20 INK 0,0
30 FOR X=1 TO 639
40 Y=200+200*SIN(2*PI*X/639)
50 PLOT X,Y.1
60 Y=200+200*(SIN(2*PI*X/639)^2)
70 PLOT X,Y,2
80 Y=200+200*(SIN(2*PI*X/639)^3)
90 PLOT X,Y,3
100 NEXT
110 GOTO 110
```

Figure 6.19. A high-resolution graph drawing program, using three colours.

numbers used is that we want the graphs to fill the screen. The sine of an angle has a value that lies between −1 and +1. Now a value of 1 in the Y direction does not make much impression, so we multiply the value by 200. This makes the quantity vary between −200 and +200. We can't plot −200 on this screen, however, so we add 200 to each value, making the size vary between 0 and 400, the full range of Y values. The other point is that we use 2*PI*X/639 as the angle. This is to allow for radian measure. These angle functions go through a range of values as the angle goes from zero radians to 2*PI radians. When X=0, then 2*PI*X/639 is zero, which gives us zero radians at the left-hand side of the graph. When X=639, at the right-hand side of the graph, then the angle is 2*PI*639/639, which is 2*PI radians, just what we want. If you don't like working with angles in radians, just have the command DEG in a line somewhere before the angle functions are used. After using DEG, all angles will be in degrees. To reset to radians, reset the machine or use RAD.

Now, after running that program, take a look at the graphs. They show the pixel size of mode 0 very effectively. They also show that the CPC6128's 'low-resolution' looks better than some computers' high-resolution! Try altering line 10, first to MODE 1, and then to MODE 2, to see how small the pixels are in the other modes. It's less easy to see the colour in these very small dots, unless you are using a monitor, but the graph lines look smoother and more joined-up than in mode 0. You see only two graphs in mode 2. Why? Because you can have only two colours in this mode! For most purposes, mode 1 is the ideal compromise between number of colours and high-resolution.

As the listing of Figure 6.19 shows, the PLOT statement uses a third number following the X and Y coordinate numbers. This number is the pen colour that is to be used. A fourth number may be added which will determine how the ink being used will react with any ink already present in the same pixel. There are four possibities, given by numbers 0, 1, 2, and 3. These correspond to the default (obliterate existing colour), XOR colour numbers, AND colour numbers, and OR colour numbers. Appendix H gives details of the effects of these actions. The same two optional PEN and action numbers may be used along with the commands PLOT, PLOTR, DRAW, DRAWR, MOVE and MOVER. If the numbers for PEN and action are omitted they will remain as they were set. If you haven't made any effort to set them, they will remain at the default values.

We aren't quite finished with graph drawing yet, though. Try the slightly amended program in Figure 6.20. This starts with ORIGIN 0,200. Now the effect of this is to shift the origin of all graphs to the point 0,200 – the left-hand side, half way up the screen. The origin is always taken as being the point that corresponds to 0,0, however, so PLOT 0,0 will now put a point half way up the screen, left-hand side. Shifting the position of the origin like this allows us to make the graph drawing instructions simpler, because we don't have to use the '200+' part any more. We can now do things like

```
10 MODE 0:ORIGIN 0,200
20 INK 0,0
30 FOR X=1 TO 639
40 Y=200*SIN(2*PI*X/639)
50 PLOT X,Y,1
60 Y=200*(SIN(2*PI*X/639)^2)
70 PLOT X,Y,2
80 Y=200*(SIN(2*PI*X/639)^3)
90 PLOT X,Y,3
100 NEXT
110 GOTO 110
```

Figure 6.20. Shifting the origin of a graph.

PLOT 50,−50, because −50 just means 50 pixels down from the mid-point, just as +50 means 50 pixels up. The ORIGIN command can also be used to specify a window size, by adding another four numbers to the end of the statement. The numbers used *must be in terms of graphics dimensions* in the range 0 to 319 in the Y direction and 0 to 639 in the X direction. For example:

ORIGIN 0,200,0,639,300,100

would give a graphics window which is the full width of the screen, starting a quarter of the way down, and with the bottom edge a quarter of the way up from the bottom of the main screen. The origin from which any plotting and drawing would start would be on the left-hand side, half way down the graphics window.

Drawing the line

Plotting graphs is one very useful part of high-resolution graphics, but we can make use of commands which do much more than this. Two of these are MOVE and DRAW. The MOVE command, as you might expect from the name, moves the graphics cursor. Now since the graphics cursor is invisible, you won't see anything happen when you use MOVE. It's like moving a paint-brush – but with the brush kept clear of the paper. The DRAW command is for painting a line – the brush this time is definitely on the paper. Each of these commands uses the same system of X and Y coordinates already used in connection with PLOT. In addition, the DRAW command can use a third number, the inkpot number.

It's time to look at an example, in Figure 6.21. This is nothing elaborate, simply a program that draws a square. The colour of the square will be 'gold' against a blue background if you have just switched the machine on, but it will probably be red on black if you run this just after running the graph

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAW 150,300
50 DRAW 450,300
60 DRAW 450,100
70 DRAW 150,100
80 GOTO 80
1000 MODE 1
1010 LIST
```

Figure 6.21. Using the DRAW statement to produce a square.

program. The reason is that if you don't issue a paintpot number following a DRAW X,Y instruction, the colour used will be the colour used last time a DRAW or PLOT was carried out. They don't forget, these machines! You will also find that the square is drawn very quickly, faster than the eye can follow. Compare this with the time some other machines take on even this simple drawing.

You can also draw with dotted lines or dashed lines by using the MASK statement with a number such as 85 following it. MASK 85 gives a set of fine dots in place of a solid line. You can enter the number for MASK in binary form, if you like, prefixing it with &x. This will let you see more clearly the pattern as a series of 0s and 1s. For example, using MASK&X10101010 or MASK&AA or MASK170 will all result in an evenly spaced dotted line being drawn. Line 10 uses ON BREAK GOSUB 1000 because listings do not look good in mode 0. The 'subroutine' at line 1000 converts back to mode 1, then LISTS. Since a LIST command *always* comes back to 'Ready', there's no point in having a RETURN for this 'subroutine'. When you press the ESC key twice, then, to get the program out of its endless loop in line 80, the screen goes back to mode 1 and the program is listed for you. It's a very useful thing to have when you are getting a program running.

While we're on a simple drawing, we might as well see how DRAWR can be used. As I said earlier, this means DRAW with RELATIVE coordinates. DRAWR 10,100, for example, means DRAW a line 10 pixels to the right and 100 up. This is *not* the same as drawing a line to the point 10,100, which is what DRAW would do. Figure 6.22 illustrates our square drawing with DRAWR this time. If you want to move to the right or up, the distances are positive. If you want to move to the left or down, then the distances are negative. If you want one coordinate, X or Y, to stay the same, then the number to use is 0.

Just to rub in the difference between DRAW and DRAWR, Figure 6.23 shows a set of lines drawn from the centre of the screen to random positions, using DRAW (line 60). After a pause, a quite different effect is achieved using DRAWR. This time, each new line is attached to the end of the

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAWR 0,200,2
50 DRAWR 300,0,2
60 DRAWR 0,-200
70 DRAWR -300,0
80 GOTO 80
1000 MODE 1
1010 LIST
```

Figure 6.22. Using relative numbering in DRAWR. Use DRAWR in subroutines which would be needed to plot a pattern anywhere on the screen.

```
10 MODE 1
20 CLG
30 FOR N=1 TO 50
40 MOVE 320,200
60 DRAW RND (1)*639,RND(1)*399,RND(1)*4
70 NEXT
80 AFTER 200 GOSUB 1000
90 GOTO 90
1000 CLG
1010 MOVE 320,200
1020 FOR N=1 TO 50
1030 A=RND(1):IF A>0.5 THEN A=1 ELSE A=-
1
1040 DRAWR A*INT(RND(1)*100),A*INT(RND(1
)*100),RND(1)*4
1050 NEXT
1060 RETURN
```

Figure 6.23. Random line patterns which illustrate the difference between DRAW and DRAWR.

previous one. By using line 1030 to generate a value of A which is either −1 or +1, the direction of each line is made random, and its size is also made random by line 1040. This produces a pattern which is called a 'random walk' – you might like to think of it as the path of a demented fly. The pause has been organised by an AFTER command in line 80. AFTER 200 GOSUB 1000 means that after the number 200 has been counted by the timer, the program will go to the subroutine at line 1000. As before, the timer counts at a rate of 50 per second (UK), so that 200 represents a time delay of four seconds. For US readers, use 240 in place of 200. After this time the GOSUB 1000 carries out the DRAWR routine and then returns to the endless loop in line 90.

```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR N=1 TO 360
1020 PLOT R*SIN(N),R*COS(N),2
1030 NEXT
1040 RETURN
```

Figure 6.24. A rather slow circle drawing routine.

Making circles

Drawings that consist of nothing but straight lines are all very well, but for many types of drawing we need circles, ellipses and arcs. This is a weakness of the CPC6128, one of the very few, because it contains no CIRCLE command. We have to accept this, and make use of subroutines to carry out these tasks. The manual is very helpful in this respect. Figure 6.24 shows one circle drawing program. Line 10 sets the mode, just in case it had been changed, and this clears the screen. Line 20 then sets the centre of the circle as the origin, and sets the radius, R. The circle drawing subroutine is then called, and the program shifts the origin back, and loops endlessly until you press ESC twice. In the subroutine, DEG sets the angle to units of degrees. Line 1010 then starts a loop, plotting each point around the edge of the circle. If you did (and remember) coordinate geometry at school, you'll understand what is happening. If you didn't or don't, then take it on trust rather than getting mixed up in mathematics – the subroutine *paints* a circle.

It's not the only way of painting a circle, and it's very slow, but at least it's reasonably simple. Figure 6.25 shows a much faster alternative. This does

```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1005 MOVER 0,R
1010 FOR N=0 TO 360 STEP 10
1020 DRAW R*SIN(N),R*COS(N)
1030 NEXT
1040 RETURN
```

Figure 6.25. A much faster circle routine.

```
10 MODE 1
20 ORIGIN 320,200:R=100
30 GOSUB 1000
40 ORIGIN 320,200
50 FILL 2
60 END
1000 DEG
1010 MOVER 0;R
1020 FOR N=0 TO 360 STEP 10
1030 DRAW R*SIN(N),R*COS(N),2
1040 NEXT
1050 RETURN
```

Figure 6.26. Filling in a shape with colour, using FILL.

not paint a true circle, but instead draws a shape with a lot of straight sides. By choosing a large number of sides (36), however, the shape looks reasonably circular, and the speed makes up for a lot! This subroutine is the one to go for if you don't want to hang about waiting for a circle to be drawn.

While we are working with the CIRCLE command, it's a good time to take a quick look at how to fill in a circle. Figure 6.26 gives you a taste of this, with lines 10 and 20 setting up the origin and radius, and the subroutine that starts at 1000 drawing the circle. The idea is to draw a set of lines close together, with the ends of the lines drawing out the circle. The filling effect is achieved by using the FILL 2 statement, in which 2 is the colour required for the fill. The FILL statement was not available on the CPC464, and it makes colour filling very much easier to carry out. The fill ends when it encounters a boundary line in the same colour. No fill will start if the cursor is placed at an edge, so you have to be careful to place the cursor correctly before you specify a fill.

Detective work

Many interesting effects are possible if you can arrange for a program to detect, automatically, where the cursor is. The graphics cursor, remember, is invisible, so some method of detecting where it is or what it has just hit must rely on some sort of command. There are, in fact, several commands that report what the graphics cursor is up to. One, in the form of COPYCHR$, is specifically for the text screen, and we have already used it. Two others are XPOS and YPOS. Each comes up with a number. If you use J%=XPOS, then the value of J% will be the X coordinate number at the time when the test was made. If you use K%=YPOS, then, similarly, you get the value of the Y coordinate number. These commands carry out for the graphics cursor what POS and VPOS do for the text cursor. The difference is that XPOS and YPOS do not have any window number following the command word. What we need to look at now is how we make use of these actions.

```
10 CLS
20 X=320:Y=200
30 MOVE X,Y:RANDOMIZE TIME
40 WHILE X<>0
50 J=RND(1):IF J>0.5 THEN X=20*RND(1) EL
SE Y=20*RND(1)
60 DRAWR X,Y,1
70 IF XPOS>630 THEN MOVE 10,YPOS
80 IF XPOS<10 THEN MOVE 630,YPOS
90 IF YPOS>390 THEN MOVE XPOS,10
100 IF YPOS <10 THEN MOVE XPOS,390
110 WEND
```

Figure 6.27. Making use of XPOS and YPOS.

One very useful thing we can do is to ensure that the graphics cursor never goes off the screen. We can, for example, achieve 'wrap-around'. This means that if the cursor moves off the screen on one side, it appears on the other side. Figure 6.27 illustrates this, in a program that draws ragged lines. Line 20 sets a position in the middle of the screen, and line 30 puts the graphics cursor there. Line 30 also contains the instruction RANDOMIZE TIME. The reason for this is that RND does not generate truly random numbers, nothing like as random as the spin of a roulette wheel. The RND numbers are calculated by the computer, and like anything else that is calculated, they can't be truly random. The result is that the RND numbers, if you print out enough of them, will start to repeat their sequence. You can stop this by using RANDOMIZE, and the effect is even better if the number that follows RANDOMIZE is itself a number that won't repeat except by chance. TIME is a number of this sort, so RANDOMIZE TIME is a pretty cast-iron way of ensuring that you will never get two displays from this program that are exactly alike.

Line 40 then starts an endless loop, because X can never be 0. Line 50 decides on new X or Y values. One or the other is changed to a random number, according to the value of J. In line 60, the new values of X and Y are used to draw a line, but with relative values, not absolute. We then test the position of the cursor in lines 70 to 100. These tests make sure that if the cursor is getting to the edge of the screen, it will appear on the opposite side. Note the use of XPOS and YPOS in the MOVE parts of these lines. You can't use X or Y because these are *relative* position numbers, not absolute.

There's another way of checking where the invisible graphics cursor is. The command TEST (X,Y) gives an INK number, which is the INK number for the position X,Y. You can therefore put TEST (X,Y) into IF ... statements to find if you have background colour, or the colour of some obstacle. You can also use TESTR, which takes relative coordinates. Using TESTR (1,0), for example, will test the pixel just to the right of the cursor in mode 2. You can use TESTR (2,0) in mode 1, and TESTR (4,0) in mode 0 for the same purpose.

Tagging along

Our high-resolution graphics so far have been rather static, confined to drawing patterns and shapes. It doesn't have to be like this, because there's another very useful command, TAG. TAG connects the graphics cursor to the text cursor. Now the graphics cursor is small, just one pixel in size, and the text cursor is 8 × 8 pixels. The point of attachment is the top left-hand side of the graphics cursor. If you PRINT a character at the position that is given by the graphics cursor, then, the character will occupy a space which extends to some seven pixels to the right of the graphics cursor and seven down. I'm assuming a text character which is constructed of 7 × 7 pixels – you may be using a 'defined' character of 8 × 8 of course. The effect of TAG is that we can print shapes such as we get from CHR$ numbers on the screen *at the position of the graphics cursor.* Why should this be important? Well, if you think of the MODE 1 screen as an example, we have only 40 positions for a character shape on a line. If we want to move the shapes along the line we have to move them in 40 steps, which looks very jerky. The graphics cursor can move in much finer steps. For MODE 1, the graphics cursor can move in 320 steps, using X numbers of 0 to 639. This is a much smaller movement, and it can make animation look a whole lot better.

As a sample, take a look at Figure 6.28. The TAG command in line 30 performs this task of attaching the text cursor to the graphics cursor. Anyone who has programmed the BBC Micro will recognise this action – it's the equivalent of the BBC's VDU5 command. We can now print at the graphics cursor position. Line 40 sets up a simple loop, which will take the graphics cursor across the screen. Line 50 moves the graphics cursor to the position that has been set by the values of X and Y. Note that we use STEP 2

```
10 MODE 1
20 Y=200
30 TAG
40 FOR X=0 TO 639 STEP 2
50 MOVE X,Y
60 PRINT" ";
70 MOVER 2,0
80 PRINT CHR$(243);
90 FRAME
100 NEXT
110 TAGOFF
120 END
```

Figure 6.28. Animating characters (sprites) by using TAG.

in the loop. This is because the loop takes the usual values of 0 to 639 to control the cursor, but in MODE 1 there are only 320 positions. Each position of the graphics cursor can have two X numbers. X=0 and X=1 are the same position, the next one is X=2 or X=3 and so on. In MODE 2, each number from 0 to 639 is a separate position – try the program in this mode. In MODE 0, there are only 160 separate positions across the screen, so each one can use any of 4 numbers. X=0,1,2 or 3 is the first, X=4,5,6 or 7 the next and so on. In a loop, we could use STEP 4. These STEP numbers are important, because they eliminate another cause of jerkiness. If the shape spends (MODE 0) four passes of the loop in each position, then moves, it is going to move much more jerkily than if it moves on each pass.

At the cursor position, then, we print a blank. This is done to remove the previous drawing, and it has no effect first time round. Line 70 then moves the cursor two steps on, and line 80 prints the character shape. It's at this point that you need to be careful. Try running the program with these semicolons omitted and look at the strange results! When you are using the graphics screen *everything prints*, including the carriage return and line feed codes. When you use TAG, you can suppress these codes by using the semicolon, which does not print a shape of its own. The next mystery is in line 90. This invokes a bit of synchronisation. The best way to see why is to remove this line and run the program. You will see the shape move faster, but with some peculiar effects now and again. The reason for the odd distortions in the shape is that the TV picture is formed by drawing lines across the screen and down, and brightening the spot wherever there is something to be put on the screen. When the object on the screen is moving as well, there are times when the two movements conflict. To avoid this type of problem, we have to move the object at a speed which is synchronised with the rate at which the screen spot traces the pictures. This is 50 times per second in the UK, 60 times per second in the USA. A routine in the ROM of the CPC6128 can do this, and we call up this routine by using FRAME.

Of course, you might feel that slowing down the movement is the last thing you want to do. How do we speed it up? One way is to make X and Y both integers, X% and Y%. If this isn't fast enough for you, try STEP 4, or even STEP 8. Don't be tempted to omit the FRAME statement, however, because the effects are nothing like as good – the lack of synchronisation is much more noticeable when the object is moving fast. Note that in this program, the MOVER in line 70 is used to keep part of the shape on the screen, which also helps to reduce jerkiness. For other types of shape, you might want to omit this, or use a different number.

The CPC6128 has two commands which were not available for the CPC464 or CPC664: GRAPHICS PAPER and GRAPHICS PEN, both of which have to be followed by an INK number. For example:

```
GRAPHICS PEN 15
```

would cause any graphics line to appear in inkpot 15 colour, whatever that had been set to. The GRAPHICS PAPER effect is less obvious. If, for example, you use GRAPHICS PAPER 3, then the background will change completely only if you then use CLG. The paper colour will show through, however, when MASK is being used, or when TAG is printing characters on to the screen even if CLG has not been used. The GRAPHICS PEN statement can also take a second number to determine the state of the background. Using a 0 in this position causes a plotting or TAG statement to obliterate anything else that was previously drawn in the same place, but using a 1 will cause drawing in the current PEN colour, without affecting anything else that is already drawn in the same place.

INK animation!

A quite different, and very cunning, method of animation is possible by making use of the INK command. Suppose that what you want to animate is not a CHR$ shape (which, remember, can be a shape that you have designed), but a complete drawing which has been created using DRAW and MOVE. You could, of course, draw the shape, wait, erase the shape, wait, draw in a different position, and so on. The trouble with this is that it's slow, and the animation is jerky. A much better method is achieved if you make use of these incredible INK commands. Suppose you make a set of drawings, each in a different position, and each with ink taken from a different numbered pot. Suppose also that the ink in each pot is of the same colour as the background. The result, of course, is that all your drawings are invisible!

Now imagine a loop in your program. In each pass of the loop, you change one of the inkpot colours you have used to a new colour, a colour that is *not* background colour. This will have the effect of making one of your drawings visible. Wait a moment, and then make this inkpot contain background colour again. Your drawing disappears, and you can then change the ink in another pot to make another drawing appear, then disappear. In this way, you can make a set of drawings *that have been put on the screen beforehand* appear and disappear in turn, giving an illusion of animation. This, of course, is best done in MODE 0, in which you have many more inkpots to play with.

Figure 6.29 shows the kind of thing I have in mind. The first part of the program sets ink colours 2 to 15 to the background colour, which is colour number 1. Lines 40 to 90 then draw four rectangles on the screen. One rectangle is vertical, and the next three are each at 45 degrees to the one before. The whole set shows the successive positions of a rectangle as it turns from vertical to horizontal to vertical again, in a clockwise direction. If you want to examine what is going on in this part of the program, make the INK number in line 30 equal to 24 instead of 1, and put in a line 95 STOP.

Once the drawings are complete, with the screen still blank, you can make

```
10 MODE 0
20 FOR CL=2 TO 15
30 INK CL,1:NEXT
40 FOR NR=1 TO 4
50 READ X,Y:MOVE X,Y
60 FOR LN=1 TO 4
70 READ X,Y
80 DRAW X,Y,NR+1
90 NEXT:NEXT
100 WHILE INKEY(47)=-1
110 FOR NR=2 TO 5
120 FRAME
130 INK NR,24
140 FOR X=1 TO 30:NEXT
150 INK NR,1
160 NEXT
170 WEND
180 DATA 300,300,350,300,350,100,300,100
,300,300
190 DATA 385,288,415,252,270,113,235,148
,385,288
200 DATA 225,225,425,225,425,175,225,175
,225,225
210 DATA 230,255,275,293,410,148,377,110
,230,255
220 MODE 1:END
```

Figure 6.29. Animating a set of 'invisible' drawings.

them appear by changing INK colours. The loop that starts in line 100 will continue until the spacebar is held down. The FOR ... NEXT loop then changes each INK colour in turn for a short time and then returns it to background colour. The FRAME is added to make the animation smoother – try omitting it to see the effect. The overall impression is of a rectangle which rotates clockwise.

Nothing in graphics is achieved without a certain amount of sweat and toil. The coordinates for the successive drawings have to be found and put into the DATA lines, and this involves hard work – it's easier if you have a drawing board! It's very likely, however, that someone will bring out a program that will produce these coordinate numbers for you. You produce the original shape, and the program will then give you coordinates for each point at which the shape is turned through an angle which you specify. It's a piece of geometry that the machine can solve a lot quicker that you can. This method of animation, incidentally, is used also on the BBC Micro, though in a less simple way.

Chapter Seven
Sound Envelopes

The ability to produce sound is an essential feature of all modern computers. The sound of the CPC6128 comes from its built-in loudspeaker, which has a volume control sited next to it at the rear right-hand side of the casing of the computer. To increase volume, you pass your hand over the wheel, moving from right to left. In addition, a socket on the back of the CPC6128 allows you to connect to a hi-fi amplifier unit, so that you can hear the sound at greater volume, and with better bass (low notes). The difference is quite astounding if you have only ever heard the sound delivered from the tiny loudspeaker of the computer itself. In addition, the sound that can be obtained from the socket can be in stereo. If you have one of the popular makes of pocket stereo players you can plug in the stereo earphones to this socket, which is at the rear right-hand end of the CPC6128. The volume of sound that you get on these headphones is not very great, however, and a stereo amplifier is really needed to get the best from this effect. Amstrad will happily sell you their speech synthesiser and stereo amplifier unit (with loudspeakers), which will add a lot to your enjoyment of the sound capabilities of the CPC6128 for just under £30.

Now once again I'm going to assume that you already know the elementary aspects of using sound, such as the beep when you use PRINT CHR$(7), and the elements of the SOUND statement, and channel synchronisation. These topics are all covered in the manual, and in the more elementary books on the Amstrad machines. Books on the sound system of the CPC464, incidentally, apply also to the CPC6128, because the sound sections are identical. The main topic for any advanced book, then, is the use of what are called 'envelopes', both pitch and tone types, and the use of noise in sound programming.

A note from an instrument consists of many sound waves, often several hundred. However we set about creating a note on an instrument, it can't start instantaneously, and it never carries on unchanged. When you start a note, the first few waves form the build-up to the final amplitude. This part of a note is called the *attack*; it's the part in which the amplitude of each wave is considerably greater than the amplitude of the one that went before it. What happens after that depends a lot on what sort of musical instrument

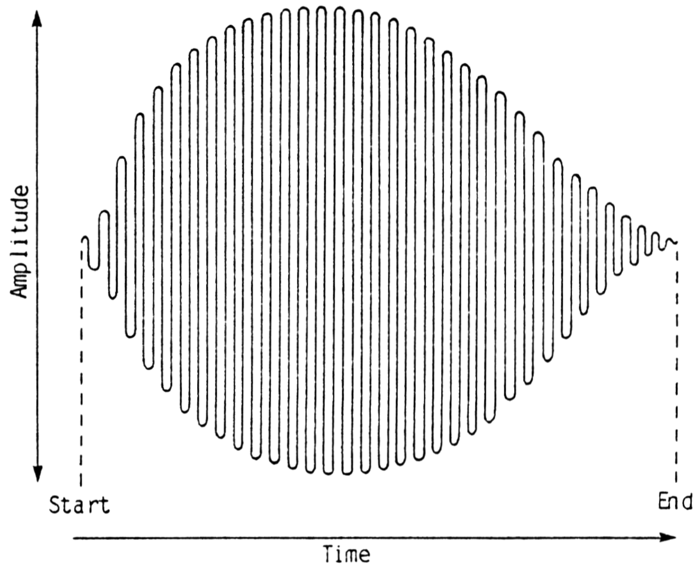


Figure 7.1. The amplitude envelope of a typical note from a plucked string. Normally when we draw envelopes, we draw only the top half, and omit the waves that make up envelope shape.

makes the note. If the instrument is one that is struck or plucked, like a drum, piano or guitar, then the note reaches its maximum amplitude just after the striking or plucking has stopped. From then on, the amplitude fades away again to zero. The shape, which we call the *amplitude envelope* of the note, is rather like the one in Figure 7.1. When we draw these envelope shapes, we show only one half, the top half, and we don't draw the waves. Figure 7.2 shows the rather different type of envelope that you can expect from instruments that are bowed or blown for the time that a note is to be sounded. This causes the envelope to have an attack, then a section in which the amplitude decreases, the *decay* section. When the amplitude of the note has settled down like this, it can be steady for a little while, and this part is called the *sustain* section. Finally the player stops blowing or bowing and the note dies away – this is the *release* section. Figure 7.3 shows the four sections for an imaginary envelope.

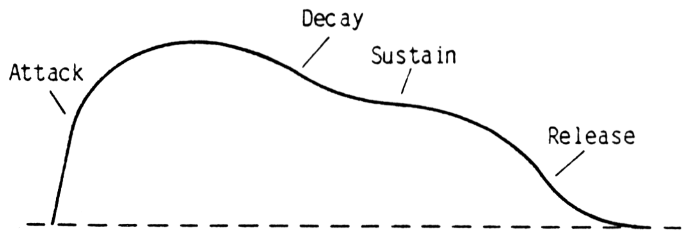


Figure 7.2. The type of envelope shape from a bowed or blown instrument.

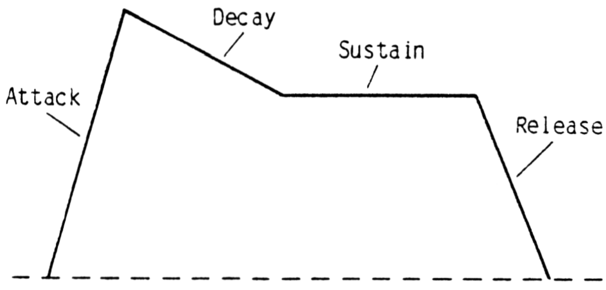


Figure 7.3. A idealised ADSR shape, using straight lines.

This type of envelope is called an amplitude envelope, because it deals with the changes of amplitude that occur while a note is being sounded. The way some instruments can be played, however, causes another type of envelope, a *pitch envelope*. If you watch a violin player at work, you'll notice that the hand whose fingers press on the strings is moved to and fro while a note is being played. This rolls the fingertips slightly over the strings being held down, and it makes the pitch of the note increase and decrease very slightly in time with the movement of the hand. This action is called *vibrato*, and it's a feature of instrumental playing that has developed over the last couple of hundred years. The aim is to make notes sound more interesting, richer, and more intense. Too much vibrato has just the opposite effect; it makes the notes sound wavering and undecided. Vibrato cannot be produced so easily on other instruments, apart from the slide trombone, and the corresponding effect for these other instruments is *tremolo*. Tremolo in wind instruments can be produced by variations in blowing, and it consists of amplitude variations rather than frequency variations.

If the computer is to be able to make a good job of producing sounds, then, it needs to be able to work with both an amplitude envelope and a pitch envelope. Both of these effects are produced by adding more data to the SOUND command of the CPC6128, so we'll take a look now at what is needed to specify an envelope. To start with, your SOUND instruction *must* have numbers for channel, tone, duration and volume. The duration number *must* be zero, and the volume number *must* have some value, normally zero. If you are trying to reproduce the sound of a musical instrument, the volume number should *always* be zero, because the note of any instrument will start from zero and increase in amplitude. Other values for the volume number (which specify the starting volume) should be used only for special effects. The range of volume numbers, which is normally 0 to 7, becomes 0 to 15 when an envelope is being used. This allows you a rather better choice of shape for the envelope than would be possible otherwise. The volume number is then followed by an amplitude envelope (or *volume* envelope) number, which will normally be in the range 1 to 15. If you use a zero here you will get a two second note of constant volume.

The envelope shape must then be specified. The SOUND instruction will only show the number of the envelope that is to be used, with the choice of 15 different types. Until you define what each of these envelopes looks like, the SOUND instruction cannot use the envelope. You don't of course, have to specify 15 different envelopes each time you program a sound that needs an envelope. You will very often want only one envelope to be used, and so you will pick one number, usually 1. For music, however, you might want to create one envelope for the time of a crochet, one for a minim, one for a quaver and so on. The amplitude envelope shape is then created using the ENV statement, specifying the number of this envelope right at the start. This ENV statement is one of the most complicated instructions in Locomotive BASIC, but one which is very rewarding to master.

The ENV statement

The division of an envelope into attack, decay, sustain and release sections is an approximation only. In addition, a real envelope outline is a complicated curve, which is very difficult to specify. The nearest we can get is the use of horizontal or vertical straight lines. This means that sloping lines have to be simulated by 'staircase' shapes, and a curve by a set of straight lines. Nevertheless, we can get sufficiently close to the true envelope shape by these methods to make a very great improvement to the sound of any note from the CPC6128. The approximations, however, make it very much easier to specify the shape we want.

To start with, there are three ways in which we can specify envelopes. The three methods are described as *hardware*, *software relative* and *software absolute*. The Amstrad manual describes in detail only the software relative method, which is the most complicated method. For many purposes, however, you might want to try the other two, and that's what we shall start with. The hardware envelope uses 'built-in' envelope shapes which are part of a ROM that belongs in the sound chip. You can put up to five of these 'hardware' envelope pieces into one single envelope if you wish, though for most purposes you will probably want only one. You can also mix a hardware section of envelope with a software section, and that's something that we'll look at later. The form of the ENV instruction for a hardware envelope is:

```
    ENV N%,=H%,P%
```

using just three numbers and the equals sign. Of these three numbers, the first is the number of the envelope, 1 to 15, the second number (range 8 to 15) decides which hardware envelope will be used, and the third number (range 0 to 65536) sets the timing of the envelope.

The hardware envelopes of the CPC6128 allow a considerable number of interesting sounds to be generated, and it's quite a task just to become acquainted with all the possible combinations. To start with, there are eight

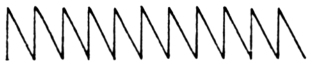
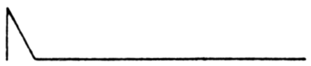
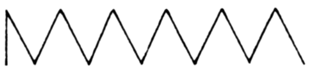
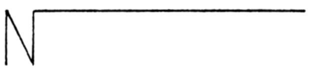
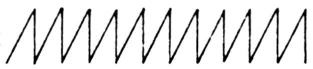
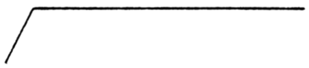
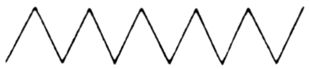
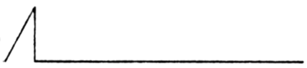
Number	Shape	Description
8		Fast up, slow down and repeat
9		Fast up, slow down, then hold at zero volume
10		Fast up, slow down then repeated slow up, slow down
11		Fast up, slow down, fast up and hold at maximum
12		Slow up, fast down, then repeat
13		Slow up then hold at maximum volume
14		Slow up, slow down and repeat
15		Slow up, fast down and hold at zero volume

Figure 7.4. The hardware envelope shapes that are available. Numbers 0 to 7 give the shapes 9 and 15.

possible hardware envelopes, whose shapes are sketched in Figure 7.4. The trouble with these sketches is that it's often very difficult for you to associate them with the sounds you hear. That's because the effect you hear depends critically on the time period of the envelope as well as on its shape. If you choose a very short time period, then any envelope will be played very quickly. That can mean that envelope shapes such as 9 and 15 are almost unheard, and you won't hear the start of envelopes 11 and 13. On the other hand, if you pick a time period that is too great, the time that is taken to change volume is so long that you hear only one section of the envelope. A time period figure of around 1000 is usually ideal to allow you to hear what is going on. So that you can judge this for yourself, the program in Figure 7.5 runs over all the possible waveform shapes, and plays each with time numbers of 10, 100, 1000, and 10000. With the time of 10, the effects are hardly noticeable. At 100, you hear a sharp click for waveforms like 9 and 15, and a rasping sound for the repeating waveforms like 8 and 12. This rasp is the result of mixing two frequencies, one being the sound tone that you specify in the SOUND instruction, the other being the fast repetition of the envelope. You can certainly create some interesting effects in this way.

At this point, it's desirable to clear up what is meant by the *period number* in these envelopes. This is a slightly misleading name, because it suggests that it might be setting the total time of the envelope. In fact, what it sets is

```
10 FOR W%=8 TO 15:RESTORE
20 PRINT"Waveform No. ";W%
30 FOR J%=1 TO 4
40 READ N%:PRINT" PERIOD ";N%
50 ENV 1,=W%,N%
60 SOUND 2,239,0,0,1
70 FOR X=1 TO 3000:NEXT
80 NEXT:NEXT
90 DATA 10,100,1000,10000
```

Figure 7.5. A program which demonstrates the envelope shapes, and the effect of the period number. Make sure that you have the volume control turned up!

the time that is needed for each step of the *changing* part of a waveform. Take, for example, waveform number 13. This consists of a sloping section, called the ramp, and a steady part. The sloping section is represented by 16 steps, and the time between steps is set by the 'period' number. According to the Amstrad hardware manual, the units of this number are steps of 128 *micro*seconds (millionths of a second). If, for example, we pick a period number of 100, it means that each of the 16 steps will require a time of 0.128 × 100 milliseconds, which is 12.8 milliseconds. This means that all 16 steps will be completed in 16 × 12.8 milliseconds, which is 204.8 milliseconds. In seconds, this is 0.204, less than a quarter of a second. This is why the low numbers produce so little effect to our ears. Tests with envelope 13 suggest that a value of around 5000 for the period number produces a ramp that lasts for one second, which corresponds to dividing the number by 5 to get the time in milliseconds. In any case, whatever number you choose will decide ramp time, but the overall length of time for the whole envelope is *always* about two seconds. This is because the envelope instruction automatically puts in a two second pause after any hardware envelope. You can get around this unwanted pause by adding a silence (software) section, since the ENV statement allows you to put in up to five sections in each complete ENV.

As an illustration of the use of a hardware envelope, take a look at the listing in Figure 7.6. This time, each envelope is made up from envelope 13, a ramp rise, followed by envelope 9, which is a ramp fall, then a short silence programmed using 1,0,40. The effect of combining the two hardware shapes is to give a waveform which has a triangular shape. If we programmed a short delay between them, of course, the result would be a rise, a flat top, then a fall.

The best way to regard the hardware envelopes is as a useful set of Lego parts for an envelope. Sometimes you can make use of one of these envelope parts directly, as I have demonstrated, but it's more likely that you will want to use the software control over envelopes that the CPC6128 provides. This consists of two types, the absolute software envelopes and the relative software envelopes. The difference between the two is that the *absolute*

```
10 ENV 2,=13,500,=9,500,1,0,40
20 ENV 4,=13,4000,=9,4000,1,0,40
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2.213,4
```

Figure 7.6. Using hardware envelopes which have been created by combining two of the basic shapes. Keep the DATA lines of this listing to use in other programs.

software envelope is a rather more crude one, with few changes of volume, whereas the *relative software envelope* can change volume more smoothly. The programming methods, however, are very similar, and we can take them together. We can specify a number of sections of the envelope, up to a maximum of five, just as we could for a hardware envelope. In each of these sections, you can have an outline which consists of a horizontal straight line, or (for the relative type) a set of steps that follow the angle of a sloping line. This makes it quite easy, for example, to simulate the classic attack, decay, sustain, release (ADSR) shape with straight lines, and still have one section in reserve. The absolute envelope allows you only one volume setting in each part of the envelope, so that the shape of the envelope is made out of horizontal straight lines with steps up or down only where one section meets another. You have to specify three numbers in each section; a step count, a step size, and a pause time. The step count can be any whole number in the range 0 to 127. If you use zero, then the envelope is an absolute one. The step size gives the amount by which the amplitude is changed in each step. This can be positive (rising amplitude) or negative (falling amplitude). Numbers of −128 to +127 can be used, but in practice, since the total range of amplitude is only 0 to 15 it makes little sense to use numbers of more than 2 or 3, unless you are looking for very special effects. The effect of using a number greater than 16 for volume is to produce the effect of the number MOD 16, the remainder after dividing by 16. In other words, if you use 18 you get the remainder of 2, if you use 37 you get the remainder of 5 and so on. Finally the period number is one that we have come across already; it can be in the range 0 to 255, with zero giving a pause of 2.56 seconds.

Absolute envelopes are useful for notes of constant volume, or notes which do not change volume much. One particularly good use for the absolute envelope is the simple creation of an echo effect. If you make the first section of the note have a volume which is large, follow this by a silence,

```
10 ENV 2,0,15,10,0,0,10,0,7,20,0,0,20
20 ENV 4,0,15,20,0,0,20,0,7,16,0,0,16
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4
```

Figure 7.7. Making an echo effect with absolute software envelope shapes.

and then by another section which gives a lower volume, you can create an echo effect which is very useful for some purposes. The timing of the notes should not be too short, and the echo should not be too quiet, otherwise it's easy to miss the effect. Figure 7.7 shows a melody with an echo on each note. Four sections have been used in each envelope, and the silence has been programmed as usual simply by using a section with zero volume.

The software relative ENV

When you specify a software relative envelope, you are creating an envelope shape which is considerably more complicated than any you can make up from the hardware shapes, or by the use of software absolute methods. To start with, we can specify a number of sections of the envelope, up to a maximum of five. In each of these sections you can have an outline which consists of a horizontal or vertical straight line, or a set of steps that follow the angle of a sloping line. This makes it quite easy, for example, to simulate the attack, decay, sustain, release (ADSR) shape with straight lines, and still have one section in reserve for a silence at the end of the note. Alternatively, you can simulate a curve which does *not* conform to the normal ADSR shape at all. The ENV instruction word is followed as usual by a number from 1 to 15 which is the reference number for the envelope. You then have three numbers for each section of the envelope. The first is the step count number. For an absolute envelope this number would be zero, but for a relative envelope it gives the number of steps of volume change which will be used in this section of the envelope. Remember that the starting volume will be decided by the number that is used in the SOUND statement. If you make the starting volume equal to zero, as is usual for envelope control, then the first section of the envelope will be used to specify the number of volume steps up to a volume level which will form the attack section of the note. The practical range here is 1 to 15, because if the volume changes by one unit in each step, then there is no point in having more than 15 steps unless you want repeating waveform effects. The second number in each section is the step size, as it was for the absolute envelope. Here, again, you are permitted numbers in the range −128 to +127, but since there are only volume steps of 0 to 15, you must choose sensible figures. The final number, as for the absolute envelope, is the pause time, in units of 1/100 of a second. The number range is 1 to 256, with 0 giving the effect of the number 256 (2.56 seconds).

What you need to watch is how these numbers interact with each other. The step count multiplied by the pause time, for example, gives the total time for the section. This should not be ridiculously short, otherwise your ear simply will not hear the note. It should not be too long either, otherwise each note will last for so long that it will be impossible to use for a melody unless a very slow tempo is wanted. Suitable values are something that you simply have to learn by trial and error, and the examples in this chapter should be a good starting point for you. The other interaction is that the step time multiplied by the step size gives the volume change. If you have ten steps, each of size 1 unit, for example, then your volume change in the section will be 10 volume units. If the volume started at zero (in the SOUND instruction) then it will end up at level 10. If the volume started at level 5, it will end up at 15, the maximum. You might, of course, want the volume to increase more sharply, such as by having 5 steps, each of size 3 which would give a volume change of 15. This, however, is a rapid change only if the pause time is short. If a long pause time has been chosen, all this would do would be to make a fairly large volume change 5 times over a comparatively long period. For a really rapid rise you would use 1 step of 15 volume units, with 1 time unit.

Since you are allowed 5 sections, the most obvious way of using the ENV instruction is by specifying A, D, S, and R sections, with a silence at the end specified by the fifth section. Let's hear how that sounds with an example, which will also show how we can plan these envelope shapes. Figure 7.8 shows an envelope designed on paper. Don't attempt to draw the steps, because this is time-wasting and difficult. Represent rises or falls by sloping lines, and show the total time of each section because this is important, particularly when you want to make any changes in the note. When you add the times for these sections, you will get the total time for the note. In this example, the attack consists of 5 steps of 3 units each to the total volume of 15. This is a comparatively slow attack, much slower that you would get from a plucked or struck instrument. The total time for this attack is 5 units, because the pause has been specified as 1 unit. The decay is then of 3 steps, each of −1 volume units and time 1 unit. This makes the decay decrease in volume from 15 to 12, and take 3 time units. The sustain section uses 5 steps of zero volume change, with pause 3. This makes the time equal to 15 units, with no change in the volume. The release is then slow, using 12 steps of size

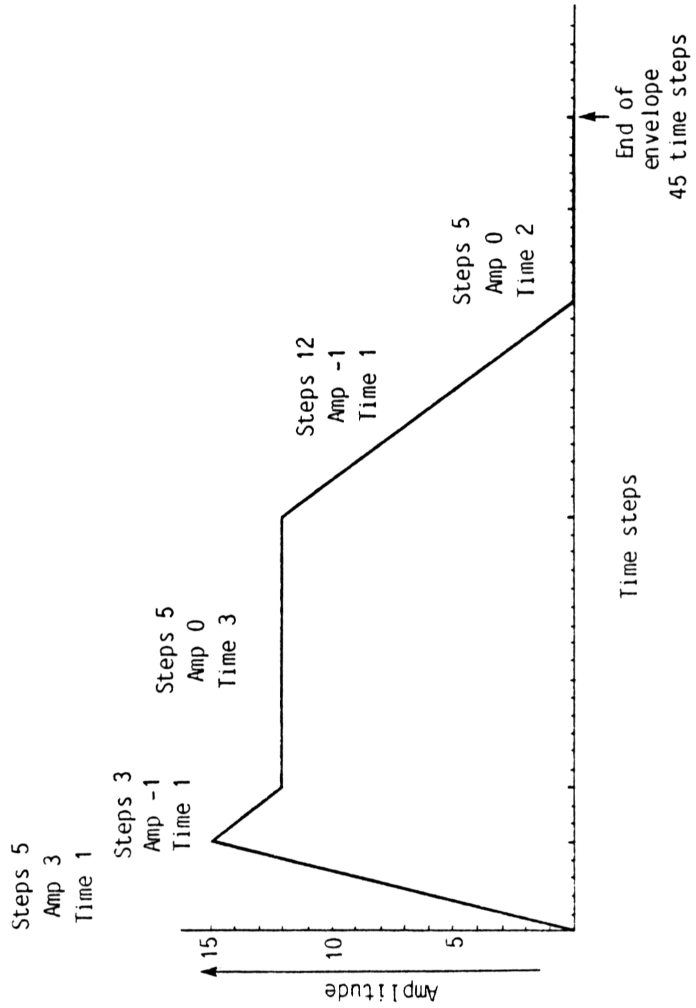


Figure 7.8. Designing a software relative envelope on paper. A grid for designing these envelopes is included in your manual.

−1 to get the volume back from level 12 to zero. The pause is 1, so that the time is 12 units. Finally, the volume is held steady (at zero) for 5 steps of pause 2, a total of 10 time units. The total time for the whole note is 42 units, corresponding to 0.42 seconds. The envelope is programmed by the lines of Figure 7.9, using longer times for the longer notes, and when you use this in the melody, you'll hear the typical sound of an envelope which has a comparatively slow attack and decay. Each note sounds as if it's afraid to come out! I have used the same DATA lines as in the two previous examples, so that you can concentrate on the effects. Sounds like this have their uses when you can also control the waveshape, but for the square wave pattern that the CPC6128 produces, such an envelope is not a particularly good one, though it can produce interesting piano accordian tunes. For

```
10 ENV 2,5,3,1,3,-1,1,5,0,3,12,-1,1,5,0,
2
20 ENV 4,5,3,2,3,-1,2,5,0,6,12,-1,2,5,0,
2
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4
```

Figure 7.9. A melody which uses the envelope shape of Figure 7.8. The DATA lines are from Figure 7.6

more interesting sounds we have to look at much sharper attack times in particular.

There are two ways in which we can produce steeper attacks. One is simply to use fewer steps with greater amplitude change, and minimum pause time. The other is to use some starting amplitude in the SOUND instruction. If the SOUND starts with a volume of 5, for example, it's easier to make a sharper attack. The ultimate is to start the volume at 15, and use the envelope to provide only the D, S and R sections. This provides the sharpest attack that you can get with this system. If, for example, you use the program in Figure 7.10 to start the SOUND volume at a figure of 7, and increase this to 15 in one step with a duration of 2 units, you will get an impressively sharp attack. The envelope then decays and releases in 2 further

```
10 ENV 2,1,8,2,5,-1,4,2,-5,2
20 ENV 4,1,8,2,5,-1,8,2,-5,4
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,7,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4
```

Figure 7.10. Providing much sharper attack by using a starting amplitude in the SOUND statement.

sections, with no steady hold portion. This gives a reasonably good 'plucked string' sound, with a sharpness to it that is good for picking out a melody. If you alter the envelope so that it consists of the attack, followed by a decay that is obtained from the numbers 15,−1,1 then you will get an even sharper 'plucked string' type of note. The tempo of the music will, however, be faster because of the decreased length of the whole note. When you are designing envelopes for musical effects, you have to be very careful about total length because, as we have seen, you need one envelope for each note length. Any alteration that you make to any section of an envelope is likely to affect the total length, and so throw out your timing. You can make up for this by altering the timing of a silent section, or by juggling steps between attack and decay sections. It's always a 'cut and fit' business, seldom the matter of precisely planning a good-looking envelope that you might expect.

Pitch envelopes

As well as the amplitude envelope, which decides the way amplitude changes during the time of a note, you can also alter the pitch of a note while it is being sounded. Now for musical notes, this is a way of obtaining vibrato, but it must conform to rules if it is to sound successful. The rate of vibrato has to be carefully chosen, and should be faster for high notes than for low ones. The amount of vibrato also has to be regulated. It is normally less than a semitone, because large amounts of pitch change sound ridiculous for music but they are useful for special effects. The use of a pitch envelope requires an extra number tacked on to the SOUND instructions, and the use of an ENT statement to define the pitch envelope.

The ENT statement follows the pattern of the ENV statement so closely that we can dispose of it quite quickly. As usual, following ENT, we have the envelope number with the usual range of 1 to 15. Choosing 0 will have the effect of leaving the note as a steady one. There is a difference here, however, because you can use a negative number as the envelope number. If you do so, then the pitch variation will be repeated until the note ends, and this is an option that you will normally want to use. The end of the note will be decided by its amplitude envelope or by the duration number in the SOUND statement. Following the envelope number, you can then define up to five sections of pitch change, each of which can be an *absolute* setting or a *relative* one. The difference is that the *absolute* setting decides a pitch number which is unchanged for a specified time, but the *relative* setting can specify a rate of change and an amount of change of pitch.

An absolute section of pitch envelope is obtained in the same way as a hardware amplitude envelope is obtained. Following the envelope number or the previous section there is a comma, then an equality sign, and then a tone period number, comma, and a pause time number. The tone period will be a number in the normal range (0 to 4095 permitted, 20 to 2000 more realistic), and the pause time a number in the range 0 to 255, with 0 providing the *longest* pause of 2.56 seconds. In such an absolute section, the tone period number gives the note that will be sounded, and the pause time gives the time in hundredths of a second for which it will be sounded. If the first section of a tone envelope is an absolute section, any tone period number in the SOUND instruction will be ignored. For example, using ENT 1,=239,50 will sound a C note for half a second. Try, as a quick way of getting acquainted with it, the lines:

```
10 ENT -1,=239,100,=190,100
20 SOUND 2,0,500,7,0,1
```

which will have the effect of sounding the notes C and E alternately. Now cut down the pause time in the ENT line to 1 from 100, and listen to the result. The absolute tone envelope is a very useful way of programming short phrases, particularly if they have to be repeated, and it can often be much simpler than the use of a SOUND instruction in a loop with data read in.

The relative type of pitch envelope uses three numbers in each section. The first number is the number of steps, the second is the step size, and the third is the pause time. The step size, however, is not the size of an amplitude step but of a pitch step, and it corresponds to a change in the note number. For example, if you are playing note 279, then a step of +1 means a change to 280, which is a *lower* pitch. These step sizes can be positive (lower pitch) or negative (higher pitch). The pause numbers are in the usual units of 1/100 second intervals. Figure 7.11 illustrates a note which starts with a little vibrato. The pitch changes in four steps, each of one unit positive, then eight negative, then four positive again, ending with the same note. In the SOUND portion of the program the usual numbers are specified, but with 0 for the amplitude envelope. This is because there must always be an amplitude envelope number preceding the ENT number, and if you have no amplitude envelope specified, then a zero must be used in this position.

As usual, it helps to some extent if we can draw the pitch envelope, and Figure 7.12 shows how ENT shapes can be planned. This is more useful for special effects, and for musical notes it's easier to follow the simplified rules. Since most musical notes that use vibrato tend to use the vibrato for as long as they are sounded, it's normal to use the negative envelope numbers. Notice that the number is negative *only in the ENT statement* – you must not use a negative ENT number in the SOUND statement.

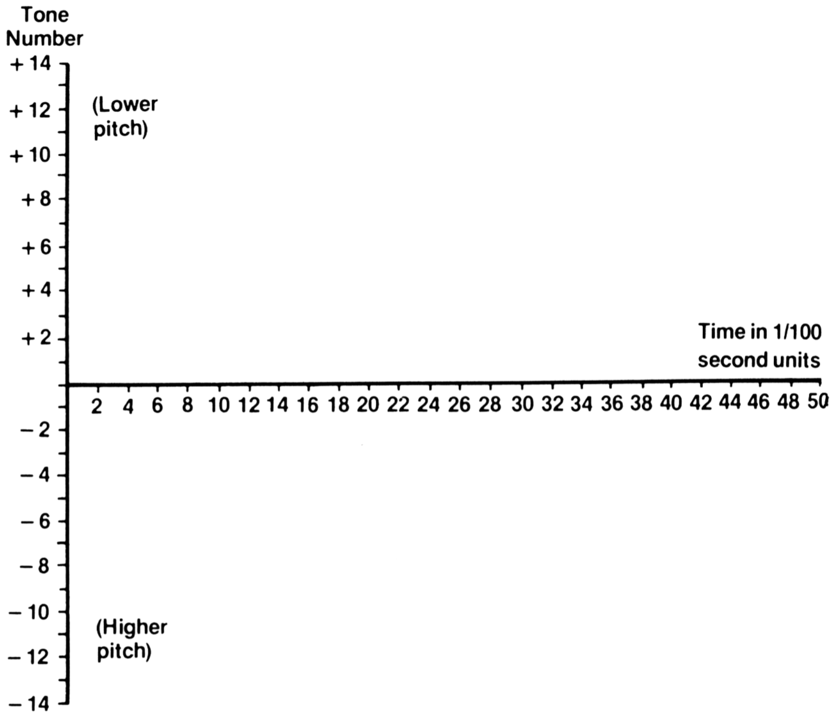
Using noise

Musical notes have definite values of pitch and amplitude, noise has not. We can generate electronically what is called 'white noise', meaning that the sound contains a mixture of all frequencies and a large range of amplitudes. Natural noises, however, aren't like this. The noise that you get from

```
10 ENT 1,4,1,1,8,-1,1,4,1,1
20 SOUND 2,239,100,15,0,1
```

Figure 7.11. A note that starts with a brief vibrato.

(a)



(b)

1. Keep the pitch change number small −1 for high notes, about 2 or 3 for low notes.
2. Keep the period number small also, using the same range of 1 for high notes, 2 or 3 for low notes.
3. Keep the step count small, using 1 or 2 in the first section.
4. Use three sections. The first and third sections are identical, with a small step count. The middle section should have a negative pitch change, and a step count equal to double that of the first/third section.

Figure 7.12. A planning grid for ENT (a), and some guidelines (b) for musical vibrato.

```
10 FOR N%=1 TO 31
15 PRINT"Noise No. ";N%
20 SOUND 2,0,50,15,0,0,N%
30 FOR J=1 TO 1000:NEXT
40 NEXT
```

Figure 7.13. A program which lets you hear the effects of the different noise numbers.

knocking two bits of wood together, for example, is not anything like white noise. It has an amplitude envelope with very fast attack and decay, and the range of frequencies is quite limited. The frequency range of the noise, in fact, is centred round a definite note which is related to the length of each piece of wood. Similarly, when you strike a drum skin the sound is closer to noise than to a musical note, but once again the frequencies that are produced are centred around one definite value, which depends on the size and tightness of the drum skin. All natural noises, from raindrops through surf on the shore to thunder are of this type.

The CPC6128 allows you to specify noise of 15 different 'centre tones'. This is done by adding a noise number, range 0 to 15, following the tone envelope number. Using zero as the noise number will, as usual, produce no noise. Noise can be produced alone or in conjunction with other sounds. If you specify a pitch number of zero for the sound channel, then no musical note will be produced, allowing the noise only in that channel. Note, however, that there is only one noise generator, and you cannot have different noises in each channel. The best place to start is by listening to straightforward noises to hear the effect of the noise 'pitch' numbers 1 to 31. Figure 7.13 illustrates this by cycling through the effects of these numbers from 1 to 31. The noise is rather like a release of steam, and the 'pitch' starts high and ends low. Even with this comparatively simple type of statement, you can produce many interesting effects. By cycling through these noise pitches more rapidly, for example, you can produce a 'raging surf' sound, ideal for tropical beaches. A simple use of noise is illustrated in Figure 7.14, which produces the much-loved steam locomotive note. This is not such a good loco simulation as would be produced by using an envelope, but for a simple routine it is quite effective. The noise pitch is fixed at 5 and duration at 10, with the volume of 1 in each 4 sounds being boosted. This gives the correct rhythm to an effect which would otherwise be rather unconvincing.

Noise, however, really comes into its own when you can make use of an

```
10 FOR J%=0 TO 79
20 IF J% MOD 4=0 THEN V%=15 ELSE V%=9
30 SOUND 2,0,10,V%,0,0,5
40 FOR X=1 TO 150:NEXT
50 NEXT
```

Figure 7.14. A program which uses noise to produce a steam locomotive sound.

```
10 ENV 1,=8,100
20 FOR N%=1 TO 12
30 SOUND 2,0,0,0,1,0,10
40 NEXT
```

Figure 7.15. Engine noise for a World War I biplane!

amplitude envelope along with the noise. This does not need to be an elaborate relative software envelope, because impressive effects can be obtained from the range of hardware envelopes. The hardware envelope 9 is particularly useful for a variety of sounds that have a steep attack. The repeating hardware envelopes are much more useful for sound effects than for music. One particularly useful application is to create sound effects which consist of fast rhythmic noises, like old-style aero engines with pistons. By using a fairly small time number in the hardware envelope, along with shape 9, you can create quite a convincing noise for a World War I Sopwith Pup warming up, as Figure 7.15 illustrates. Just to show how versatile this system is, try changing to hardware shape 14 in this program. You'll now hear the characteristic swishing of helicopter blades. Using ENV 1,=12,200 is also a very good helicopter blade sound. The combination of hardware shapes can also be usefully employed, particularly shapes 13 and 9. Used with short times, these give gunshot sounds, but with long times you can get a sort of steam hammer effect.

Obviously, you can make use of software relative envelopes to control a noise, and once we get into this the sky's the limit! It's because there are so many parameters that can be altered that you find yourself often spoiled for choice. Another dimension can be introduced by adding pitch envelopes to noise. The effect of this is *not* to alter the predominant pitch of the noise, because this is set by the last number in the SOUND instruction, but to *add* a changing musical note. Using an amplitude envelope which has fast attack and slower decay, along with a rising pitch envelope, gives the traditional 'boing' sound which is well illustrated in Figure 7.16. At lower pitches, this is good for 'giants' footsteps' (try it with an echo, too), and at higher pitches gives quite a reasonable plucked string note. This is only because the time of the amplitude envelope does not give time for more than a fraction of the pitch variation, however. You always have to remember how these envelopes are linked, with the amplitude envelope deciding how long the note plays, irrespective of whether or not the pitch envelope has finished. If you find that a set of envelopes does not do what you expect, it's often a

```
10 FOR N%=1 TO 10
20 ENV 1,1,15,1,15,-1,3
30 ENT 1,10,-1,2
40 SOUND 2,1000,0,0,1,1
50 NEXT
```

Figure 7.16. The traditional 'bouncing' sound.

good idea to look at the total times of the envelopes. Remember that if you have a slow changing amplitude envelope and a fast pitch envelope, all of the pitch changes could be over before the amplitude is loud enough to hear. If you get it the other way round, the note could be too short for you to notice any pitch change. As always, this is something that you have to work at, but remember that you can always make the pitch envelope repeat so as to fill in time in the amplitude envelope. All in all, the effective use of sound demands a good ear, some planning, the willingness to experiment and above all, the keeping of good notes. One of the most annoying things that can happen is knowing that you once came across just the sound you now want, but you can't remember how you achieved it!

Chapter Eight
Printers

Whenever your use of a disc-equipped computer extends beyond playing games that other people have written, there is one addition to your computer equipment that you will urgently want; a printer. In many cases, particularly when you are developing your own programs, the printer has a very high priority indeed. The reasons for needing a printer are obvious if you use the machine for business purposes. You can hardly expect your accountants or your income tax inspector to look at accounts that can be shown only on the screen. It would be a total waste of time if you kept your stock records with a computer, and then had to write down each change on a piece of paper, copying from the display on the screen. For all of these purposes, and particularly for word processing, the printer is an essential part of the computer system. Output on paper is referred to as 'hard copy', and this hard copy is essential if the computer is be of any use in business applications. For word processing uses, it's not enough just to have a printer; you need a printer with a high-quality output with characters as clear as those of a first-class electric typewriter.

Even if your computer is never used for any kind of business purpose, however, you can run up against the need for a printer. If you use, modify or write programs, the printer can pay for itself in terms of your time. Trying to trace what a program does from a listing that you can see only on the screen a few lines at a time is totally frustrating. Quite apart from anything else, if your use of BASIC on the CPC6128 relies a lot on the use of GOTO for loops, you might have to list a dozen different pieces of a program just to find where one GOTO might lead you to. The answer is to avoid the use of GOTO, but there are times when FOR ... NEXT and WHILE ... WEND loops are not completely satisfactory substitutes. The problem is even worse if you write your own programs. Even a very modest program may need a hundred lines of BASIC, some of which may be long lines. Trying to check a program of a hundred lines when you may be able to see only a dozen or so at a time on the screen is like bailing out a leaky boat with a teaspoon. With a printer attached to your CPC6128 you can print out the whole listing, and then examine it at your leisure. If you design your programs the way you ought to, using a 'core' and subroutines, then you can print each subroutine on a separate piece of paper. In this way, you can keep a note of each different subroutine, with variable names noted. On each sheet you can note what the subroutine does, what quantities are represented by the variable names, and how it is used. If you have a utility program that allows you to merge subroutines, you can then construct programs painlessly using your library of tested subroutines. Printing is even more important if you use one of the more advanced languages that can run in the CPC6128, like C or Pascal.

Printer types

Granted, then, that a printer is a high priority for the really serious computer user, what sort of printers are available? The CPC6128 uses the almost universal Centronics connection for printers, including the Amstrad printers which are made by Seikosha. It's difficult to imagine any 'serious' computer without a Centronics interface, for this is the connection method that is used by all the famous name printers which are available. This means that you can attach almost any good quality printer you like to the CPC6128.

This opens up the way for any of the printers which are offered at such attractive prices in the magazines. In particular, it allows you to use printers which are universally recognised such as the Epson and Juki range, as well as the excellent Amstrad DMP2000.

Printers that are used with small computers will employ one of the mechanisms listed in Figure 8.1. Of these, the impact dot matrix type is the most common. A dot matrix printer creates each character out of a set of dots, and when you look at the print closely, you can see the dot structure. The printhead of the dot matrix printer consists of a set of tiny electromagnets, each of which acts on a set of needles arranged in a vertical line (Figure 8.2). By firing these needles at an inked ribbon which is placed between the head and the paper, dots can be marked on the paper. Each character is printed by firing some needles, moving the head slightly, then firing another set of needles, and so on until the character shape is completely drawn (Figure 8.3). The most common pattern of dots for low-cost printers is the 7×5, meaning that the characters can be made of up to seven dots in height and up to five in width. This implies that the head moves across the paper in five steps to print each character, and that up to seven needles can be fired. Using a 7×5 structure gives characters which are just about readable, but not good-looking. The dots are very evident, and some of the letters are misshapen. You will find, for example, that lower-case letters lack 'descenders'. This means the tails on letters y,g,p,q will appear on the same level as the foot of other letters. When this print is used for listings which are in upper-case only, there is no problem. You would not, however, use a printer of this class to print letters or other documents that anyone else would have to read.

Dot matrix
- impact
- thermal
- electrostatic

Type impact
- type stalk
- daisywheel
- thimble
- type band

Plotters
- graphics printers
- X-Y plotters

Ink jet
- single colour
- multicolour

Laser
- laser fast printer

Figure 8.1. A list of printer mechanism types.

Rather better results can be obtained if the number of needles in the printhead is increased. Using 9 × 9 (nine needles, nine steps across) or 15 × 9 heads can create much better-looking characters, lower-case or upper-case. Another advantage of these printheads is that the characters are not limited to the ordinary letters of the alphabet and numbers. Foreign characters can usually be printed, and it is possible to print Arabic script, or to make up your own character set for example. Most of the dot matrix printers are impact types. This means what is says, that the paper is marked by the impact of a needle on an inked ribbon which hits the paper. There are also thermal and electrostatic dot matrix printers. These use needles, but the needles do not move. Instead the needles are used to affect a special type of

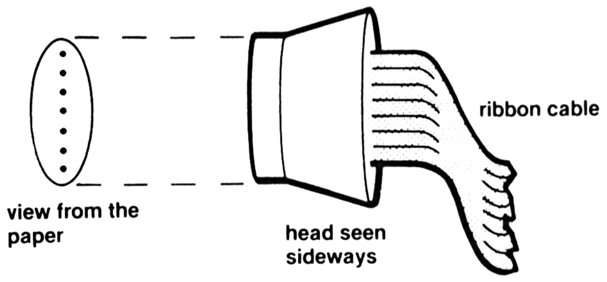


Figure 8.2. The form of a dot matrix printhead.

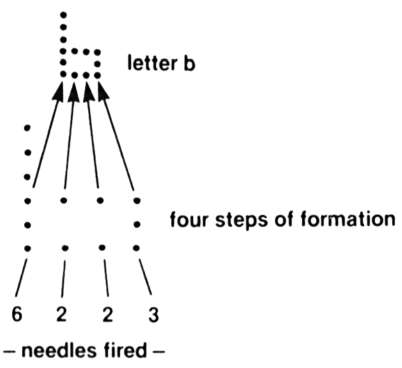


Figure 8.3. Creating a character with a 7 × 5 matrix printhead. Some needles are fired in each of the four positions illustrated.

paper. In the electrostatic printer (such as the old ZX printer), the needles are used to pass sparks to the paper, removing a thin coating of metal from the black backing paper. The thermal type of printer uses hot needles to make marks on heat-sensitive paper. Both of these printers require expensive special paper, and are unsuitable for serious business purposes, so we won't spend any time on them here. If you want a cheap printer for listings, there are better methods.

The ultimate in low-cost print quality at the moment is provided by the daisywheel printer. This uses a typewriter approach, with the letters and other characters placed on stalks round a wheel. The principle is that the wheel spins to bring the letter you want to the top, and then a small hammer hits the back of the letter, pressing it against the ribbon and on to the paper. Because this is exactly the same way as a typewriter produces text, the quality of print is very high. It's also possible now to buy a combination of typewriter and daisywheel printer. This looks like a typewriter, with a normal typewriter keyboard, but has an interface connection for a computer. You can use it as a typewriter, and then connect it to the computer and use it as a printer. Machines of this sort are made by leading typewriter manufacturers such as Silver Reed, Brother, Triumph Adler, Smith Corona, and others. If you need a typewriter as well as a printer, then this type of machine is an obvious choice. A point to watch out for, however, is ribbon costs. Just to give you an example, I have an electronic typewriter which uses ribbons that cost twice as much as the ribbons for my Juki daisywheel, and last for one fifth of the number of characters! One of the great advantages of using machines in the Epson/Juki class is that ribbon costs are as low as you are likely to find.

The third kind of mechanism that we shall look at here is the graphics printer/plotter. This is a remarkable mechanism which uses four miniature ball pens to mark the paper directly, with no ribbon. It can be used for graphics work, and when it is used as a printer, the letters are *drawn* rather than printed. Because four pens are used, the markings can be in four different colours. Printers of this type are not expensive (as printers go) and can be very useful, particularly if you want graphics output in colour.

Another type of printer that is now becoming available is the ink jet printer, which operates by shooting fine jets of ink at the paper. This one shares the disadvantage of thermal and electrostatic types in that you get only one copy. Impact printers have the great advantage that you can obtain an extra copy by using a sheet of carbon paper and another sheet of plain paper. You can also buy listing paper which has a built-in carbon, or which uses the NCR (no carbon required) principle to produce two copies. Looking further ahead, laser printers are now available at very high prices which provide superb quality of printing at very high speeds. Their prices at present rule them out for ordinary home use, but for even a comparatively small office they have a lot to offer when connected to a CPC6128 used as a word processor.

Interfaces

The printer has to be connected by a cable to the computer, so that signals can be passed in each direction. The computer will pass to the printer the signals that make the printer produce characters on the paper, but the printer must also be able to pass signals to the computer. This is because the printer operates much more slowly than the computer. Unless the printer contains a large memory 'buffer', so that it can store all the signals from the computer and then get to work on them at its own pace, some sort of 'handshaking' is needed. This means that the printer will accept as many signals as its memory will take, and then will send out a signal to the computer which makes the computer hang up. When the printer has completed a number of characters (one line, one thousand, or possibly just one character) it changes the 'handshake' signal, and the computer sends another batch. This continues until all of the text has been printed. This can mean that you don't have the use of the computer until the printer has finished. Printers can be very slow, particularly daisywheel and plotter types. Even the fastest dot matrix printers can make you wait a minute or more for a listing.

Two types of interface are used by practically all printers. These are classed as serial or parallel. A *parallel* printer is connected to the computer by a cable which uses a large number of separate strands. Since each character in ASCII code uses seven signals, the parallel printer sends these along seven separate strands – many printers can use an eighth signal and this is often sent as well, but not by the standard Amstrad printer port. In addition, there are cable strands for the 'handshake' signals. The best-known, and most used variety of parallel connection is called 'Centronics', after the printer manufacturer which first used it. Practically all of the popular printers use this type of parallel interface, and the connections to the CPC6128 Centronics socket are illustrated in Figure 8.4.

Pin No	CPC6128	Epson RX80	Juki 6100
1	STROBE	STROBE	STROBE
2	DATA	DATA 1	DATA 1
3	DATA 1	DATA 2	DATA 2
4	DATA 2	DATA 3	DATA 3
5	DATA 3	DATA 4	DATA 4
6	DATA 4	DATA 5	DATA 5
7	DATA 5	DATA 6	DATA 6
8	DATA 6	DATA 7	DATA 7
9	EARTH (GND)	DATA 8	DATA 8
10	No connection (N/C)	ACKNLG	ACKNLG
11	BUSY	BUSY	BUSY
12	N/C	PE(end of paper)	PE
13	N/C	+5V	SLCT
14	EARTH	AUTO FEED	EARTH
15	N/C	N/C	N/C
16	EARTH	EARTH (LOGIC 0)	EARTH
17	N/C	CHASSIS EARTH	CHASSIS EARTH
18	N/C	N/C	+5V
19	EARTH	EARTH	EARTH
20			
21			
22			
23	ALL		
24	EARTH	ALL	ALL
25		EARTH	EARTH
26			
27			
28	EARTH		
29	N/C		
30		EARTH	EARTH
31		INIT	PRIME
32	N/C	ERROR	ERROR
33	EARTH	EARTH	EARTH
34	N/C	N/C	N/C
35	N/C	+5V	N/C
36	N/C	SLCT IN	N/C

Figure 8.4. The Centronics connections for the CPC6128, the Epson RX80, and the Juki 6100. This illustrates how different manufacturers treat a 'standard'!

The *serial* interface sends the signals out one at a time. This means that at least seven signals have to be sent for each character, and in practice the total must be ten or eleven, to allow for 'start and stop' signals which are used to mark where the signals for each character begin and end. This system uses less cabling, because only two strands need to be used for signals, and the cables can be longer because there's no risk of one signal interfering with another. The standard system is called 'RS-232'. Printers can be obtained with RS-232, but seldom as standard, and often only as an extra, costing up to £50 more. The CPC6128 uses a parallel system, so that practically all of the popular printers can be connected in addition to the Amstrad printers. The Amstrad DMP-1 printer is a simple design which produces acceptable results for listings. For business use, however, it's unlikely that this standard of print would be acceptable. The DMP-1 printer, however, can reproduce the graphics shapes of the CPC6128, and this might be important to you. If, however, I make the assumption that you wouldn't buy a machine with a built-in disc drive and CP/M unless you were interested in business applications of some sort, then it makes sense if I confine the descriptions here to printers with high-quality output, of which the later Amstrad DMP2000 is one.

In this book, for example, the listings have been reproduced on an Epson RX80 printer. This uses a 9×9 matrix for characters, so that the appearance of the characters is better. In addition, the Epson can operate in 'emphasised' mode. In this printing mode, each dot is struck twice, but the head is shifted slightly between dots. This causes the dots to look almost joined up, and makes the appearance of the print much more acceptable. The most recent Epson model, the LX80, also has an 'NLQ' (near letter quality) option, which produces a print style which is very similar to that of a good electric typewriter (which, incidentally, would cost more!).

A problem that you are bound to run up against when you use any non-Amstrad printer is that of line feed and carriage return. Many computers send out only one code number, the carriage return code (13) at the end of a line. Other machines send both the line feed (code 10) and carriage return codes. Printers are arranged, therefore, so that either possibility can be catered for by a switch. If you connect your printer and find that everything is printed on one line, then don't return the printer. Just look in the manual and find out the details of the switch that alters the line feed setting. If, on the other hand, you find that each line is double-spaced, then this switch will have to be set to the opposite position. Note that the printer must be switched off when you make this adjustment. Apart from any other considerations, the changes do not take effect until the printer has been switched off and on again. This is a safety precaution to prevent a piece of printing from being ruined by a thoughtless piece of switching! My old CPC464, when connected to an Epson MX80, performed two line feeds, no matter how the selector switch was set, and this had to be corrected by a method that sounds rather drastic. The problem centres on the way that Epson printers make use of the connection to pin 14 of their Centronics socket. On the MX80 and RX80 machines, this pin is used to control line spacing, and because the Amstrad connection is used for a different purposes, this causes trouble. One way out is to remove the cable strand from this pin. If you lever open the top of the connector at the printer end, you will see that the flat cable is simply pressed down against a set of spikes which pierce the insulation of the wires. Locate the strands which are over pin 14 and mark this point. Check that you have the correct pin before going any further. Now take a Stanley knife or something similar, and carefully cut the insulation on each side of this part of the cable (Figure 8.5). The idea is to cut the plastic so that the insulated wire can be peeled back. When you have done this, cutting as far as the other row of pins, pull the wire off pin 14, and bend it back so that it cannot make any contact with any pins of the connector. Now clamp the top on to the connector again, and the

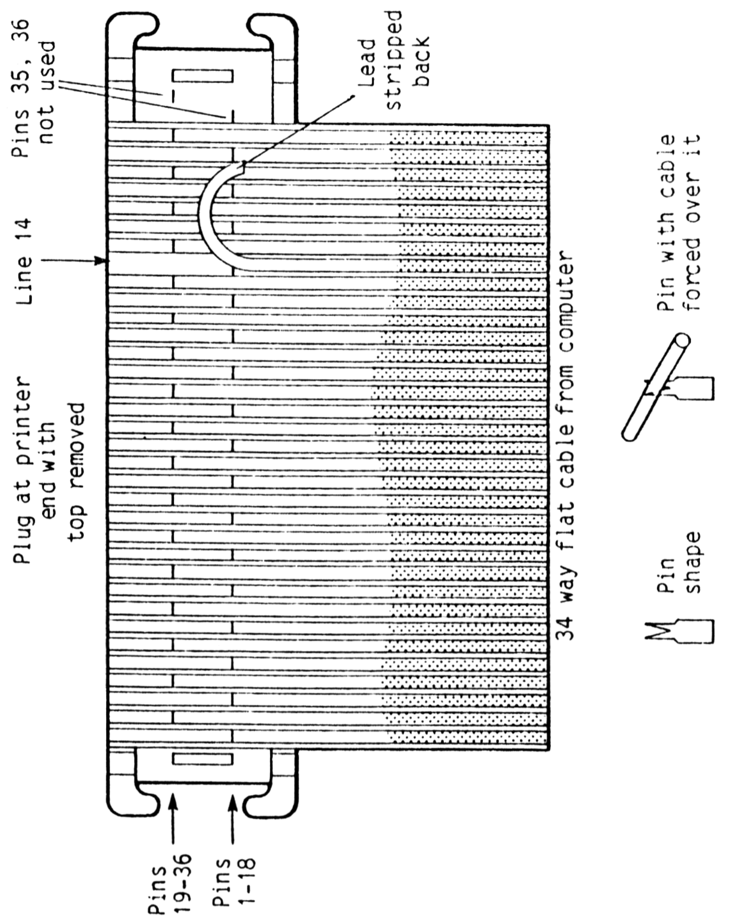

Pins 35, 36 not used
Lead stripped back
Line 14
Plug at printer end with top removed
34 way flat cable from computer
Pin with cable forced over it
Pin shape
Pins 19-36
Pins 1-18

Figure 8.5. Removing the connection to pin 14 for an Epson printer. Some firms can sell you 'Epson-compatible' cables on which this has been already done.

modification is complete. A less drastic alternative is to cut a piece of sticky tape so that it can be fitted over the copper strip that corresponds to pin 14 on the CPC6128 printer port. This, however, is not so useful if you are likely to remove the plug and reconnect several times. I should emphasise that the modifications are needed only for a few printers, and some suppliers now provide 'Epson' printer cables for the CPC6128 which have no pin 14 connection to worry about.

The Epson MX80, FX80 and RX80

The Epson range of printers has for a long time been the most popular range of moderately-priced printers, offering excellent print quality at reasonable prices. The RX80, FX80, and LX80 are the latest in this line, but if you are offered a second-hand MX80, then this also is a good buy. A particular feature of the Epson range is that the printheads plug into place, and can

```
10 REM USING RX80 IN NORMAL MODE
20 REM WHICH PRINTS AT MAXIMUM SPEED
```

Figure 8.6. The normal upper-case letters of the RX80, as you would use for listings.

easily be replaced when they wear out. My old Epson MX80 was just beginning to show signs of head wear after printing half-a-million words, so it might not be a problem for you!

The standard version of the RX80 uses pin feed, but the RX80F/T can take any form or paper, including rolls. You have to pay extra for a paper roll holder, but if you are handy with wood and piano-wire, this is something that you could easily make for yourself. The advantage of using the F/T version is that plain unperforated paper rolls are *very* much cheaper to buy, and it also means that you can use plain paper sheets if you want to. When you use a lot of paper for listings, this can be a great saving. Paper width of 4 inch to 10 inch in pin feed or plain form can be used, so you can buy whatever paper size is on offer. If you use the F/T option you can then buy the teletype rolls, which are 8½ inches wide. These can also be obtained in two-ply form, with carbon paper, so that you can make a copy as you print. The drawback here is that the two sheets tend to slide apart sideways, and you should use a one inch margin on each side if you intend to make use of this type of paper with friction feed.

The RX80 offers a full set of upper or lower-case letters, and you don't have to go through any elaborate antics to select which one you want. Figure 8.6 shows the normal upper-case letters of the RX80, as you would use them for a listing. The print speed is very fast, and most listings will be completed in under a minute. Figure 8.7 shows the lower-case letters, which are much better formed than those of cheaper printers. Figure 8.8 shows the 'emphasised' type of the RX80. This is achieved by typing PRINT#8, CHR$(27)CHR$(69) (press ENTER) before listing. The emphasised print can be cancelled by using PRINT#8,CHR$(27)CHR$(70). These commands can be used in programs, so that you can print normal, condensed, emphasised, double width, and all of the other varieties, under program control. This makes it very easy to produce good headings, produce words in bold type or italics, and to underline. We have already noted how to obtain printer set-up under CP/M in Chapter 3. For many

```
10 rem lower case on the screen
20 rem can also be produced on the printer.
```

Figure 8.7. The lower-case letters of the RX80.

```
10 REM THIS SHOWS THE EMPHASISED
20 REM STYLE OF PRINT OF THE RX80
```

Figure 8.8. The emphasised print of the RX80, as used for the listings in this book.

Switch 1

Position	ON	OFF
1	Condensed	Pica (print size)
2	Graphics	Control code
3	No buzzer	Buzzer on (end of paper)
4	12 inch	11 inch (form length)
5	Not detected	Detected (paper end)
6, 7, 8	Selects from international character set of eight languages	

Switch 2

Position	ON	OFF
1	Slashed	Non-slashed (zero)
2	Control pin	Not fixed
3	Line feed	No line feed (with C/R)
4	Skip	Don't skip (perforation)

Figure 8.9. The settings of the two DIP (dual in line package) switches of the RX80.

word processing actions, the RX80 can be a very satisfactory low-cost alternative to a daisywheel. International character sets (USA, France, Germany, England, Denmark, Sweden, Italy, Spain, Japan, Norway) can be printed, and are under software control. This means that selection is made by printing CHR$ numbers rather than by altering switches on the printer itself. The only switches that you have to alter are for such items as are listed in Figure 8.9. For most purposes, you would probably never need to alter the factory settings of these switches. Figure 8.10 shows the options that can be selected by sending CHR$(27)CHR$(N) codes to the printer.

Each of the letter codes will be preceded by CHR$(27), the ESC code. Some of the CHR$(number) codes can be used alone – consult the manual for details.

Code	*Effect*
J	Adjust line spacing in 1/216 inch units.
M	Elite size characters.
P	Pica size characters.
CHR$(14)	Enlarged print.
CHR$(20)	Cancel enlarged print.
W	Second enlarged print mode.
CHR$(15)	Condensed print.
CHR$(18)	Cancel condensed print.

Figure 8.10. The software selections available on the RX80.

_	Underline on/off switch.
E	Set emphasised mode.
F	Cancel emphasised mode.
G	Double strike mode.
H	Cancel double strike mode.
S	Superscript/subscript switch.
T	Cancel superscript/subscript.
CHR$(8)	Backspace.
CHR$(4)	Alternate character set.
CHR$(5)	Cancel alternate character set.
m	Choose graphics or control characters.
0	1/8 inch line spacing.
1	7/72 inch line spacing.
2	1/6 inch line spacing.
3	Set spacing in 1/216 inch units.
A	Set line spacing in 1/72 inch units.
CHR$(9)	Horizontal tab.
CHR$(11)	Vertical tab.
e	Tab unit setting.
f	Skip position setting.
C	Form length setting.
N	Skip over perforation setting.
O	Skip over perforation cancel.
Q	Right margin set.
I	Left margin set.
8	Ignore paper end detector.
9	Enable paper end detector.
<	One line unidirectional printing.
@	Restore normal settings.
U	Unidirectional printing.
S	Half speed (quiet!) printing.

Other codes can be used to control each pin in the head so that graphics can be printed. This allows 'screen dump' programs which place a copy of the screen graphics on to the paper to be written for this printer.

Figure 8.10 (contd)

The most recent model, the LX80, can use all of the print options of the RX80, and in addition has italics and the NLQ facility. Many of the print styles can now be selected without the need to alter internal switches or send control codes, simply by using a sequence of key presses on the printer. The standard feed is friction, but tractor feed is available as an extra at a very reasonable price. The printer itself is quoted at prices in the £275 range, which is very reasonable for a printer of this class at the moment.

The Juki 6100 daisywheel

The Juki was one of the first low-cost daisywheel printers to become available. Like most printers, it comes with a Centronics parallel interface, though an RS232 serial interface is available at extra cost. The Juki is a large and very heavy machine which can accept paper up to 13 inches wide. The daisywheel is of the same type as is used on Triumph Adler printers, and in my experience its life is very long. The ribbon cartridge is an IBM Selectric 82/C type. The ribbon that was supplied with my Juki was of the 'single-strike' variety, and this had a very short life (about three chapters of this book!). A 'multistrike' type of ribbon is much better. With this type of ribbon, the whole width of the ribbon is used by moving the cartridge up and down as well as by moving the ribbon itself. These ribbons are very easy to obtain from a variety of suppliers, but the best prices I have seen have been in the Inmac catalogue. The ribbons are carbon film rather than inked nylon, and are thrown away after use. This always seem a pity, because the cartridge contains a lot of mechanism that looks as if it could easily be used again. Some day, I'll try reloading one of these cartridges.

The printhead of the Juki will print in either direction, and there is a 2K buffer. This means that short pieces of text can be transferred to the printer buffer almost instantly, and the computer can be used for other purposes while the printer gets on with the printing actions. Printing is much slower than the normal rate of the Epson, but not so much slower than the emphasised mode as to make the daisywheel seem irritatingly slow. Its enormous advantage is the quality of the type. This is exceptionally clear on the top copy, and even three carbons later it is still very legible. For any letter work, or for the manuscript of a book, the Juki is ideal. If someone comes up with a program for using the extra 64K of the CPC6128 as a printer buffer, this will allow you to perform all your printing from the buffer, leaving the computer free for other tasks.

As you would expect of any modern design of printer, the Juki permits a large range of character sets, but you need to have the appropriate daisywheels fitted for each language. You cannot, for example, use words in alternate character sets without changing wheels in between. Changing wheels is particularly simple, but this is something that you don't have to worry about with dot matrix printers, because the same dot matrix head can produce any character under software control. The Juki allows underlining, bold type and shadow type, in addition to the normal printing style, and you can select your print style from a range of at least fourteen daisywheels. The daisywheels are expensive in comparison with others on the market, but ribbons are cheap. Figure 8.11 shows a printout from the Juki with the standard Courier daisywheel fitted. By removing the top cover, you can gain access to a set of miniature switches. Switch No. 1 controls auto line feed, and for use with the CPC6128, this must be set to the OFF position. This will give correct line-spacing – the ON position causes each line to be double-

```
10 REM DEMONSTRATION OF JUKI
20 PRINT#8,"This is JUKI normal print"
30 PRINT#8,CHR$(27);"E";"This is underli
ned";CHR$(27);"R"
40 PRINT#8,"We can change";CHR$(27);"O";
" to bold print."
50 PRINT#8,"We can change ";CHR$(27);"W"
;"to shadow print."
60 REM The C/R clears these effects
70 PRINT#8,CHR$(27);"Y";CHR$(27);"Z";CHR
$(27)"H";CHR$(27);"I";CHR$(27);"J";CHR$(
27);"K"
```

```
This is JUKI normal print
This is underlined
We can change to bold print.
We can change to shadow print.
```

Figure 8.11. Demonstration of Juki daisywheel output, using the Courier daisywheel which is supplied.

spaced. The switch change must be done with the machine switched off. This is not so much because of risk but because these switch settings have *no effect* until the machine is switched off and then on again.

Like the Epson, the Juki permits a number of changes to be made simply by sending control codes to the printer. These use the ESC character, CHR$(27) followed by one more character, so that whatever immediately follows CHR$(27) is never printed. The options include graphics mode, left and right margins, lines per page, half-line feeds in either direction (for printing subscripts and superscripts), top and bottom page margins, and some special characters, including the English pound sign. Even more usefully, the print can be changed to bold or shadow by sending such codes, and text can be underlined. Figure 8.12 lists these actions.

Each of these codes will be preceded by CHR$(27).

Code	*Effect*
1	Set horizontal tab (HT) at present position.
2	Clear all tabs.
3	Graphics mode on (C/R clears).
4	Graphics mode off.
5	Forward print on (C/R clears).
6	Backward print on (C/R clears).
7	Print suppress on (C/R clears).
8	Clear present HT stop.

Figure 8.12. The different selections which may be made for the Juki by software signals.

9	Set left margin at present position.
0	Set right margin at present position.
CHR$(9)	Set HT (tab number follows).
CHR$(10)	Set lines per page (number follows).
CHR$(11)	Vertical tab (VT) set (number follows).
CHR$(12)	Set lines per page (number follows).
_	Sets VT at present position.
CHR$(13)P	Remote reset.
CHR$(30)	Sets line spacing (number follows).
CHR$(31)	Sets character spacing.
C	Clears top/bottom margins.
D	Reverse half-line feed.
U	Normal half-line feed.
L	Sets bottom margin at present position.
T	Sets top margin at present position.
Y	Special character.
Z	Special character.
H	Special character (new paragraph symbol).
I	English pound sign.
J	Diaeresis mark.
K	Spanish c with cedilla.
/	Automatic backward print.
\	Disable backward print.
S	Set character spacing.
CHR$(26)A	Remote error reset.
CHR$(26)I	Initialise printer.
CHR$(26)1	Status (serial interface only).
P	Proportional spacing on.
Q	Proportional spacing off.
CHR$(17)	Offset selection.
E	Underline on.
R	Underline off.
O	Bold print on (C/R clears).
W	Shadow print on (C/R clears).
&	Bold or shadow print off.
%	Carriage settling time.
N	Clear carriage settling time.
CHR$(8)	1/120 inch back space.
X	Cancels all word processing modes except proportional spacing.

Figure 8.12 (contd)

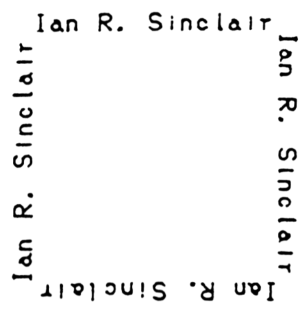
The same quality of print can be obtained from a large number of daisywheel typewriters, and many of these can now be obtained with a Centronics parallel interface. This type of machine offers many advantages, because it can be used as a typewriter for small items that do not justify the use of the computer, yet is available for word processing use along with the CPC6128 and such programs as AMSWORD. These machines can now be bought in high street stores as well as from office supply shops. The only thing to watch is that replacement ribbons and daisywheels are obtainable from several different sources. There's nothing worse than being stuck with a machine for which you can get spares from only one supplier.

The CGP-115 four colour graphics printer

One of the most popular small graphics printer mechanisms is made under the trademark of ALPS. It's Japanese, and in place of the mechanisms that are used by most printers, it actually *draws* its characters with a set of four miniature ball pens. The reason for the set of four is that this allows printing in four different colours – black, blue, red and green. The mechanism is made into boxed units by many manufacturers and sold under a wide variety of names, but it is most easily obtained from Tandy stores under the Tandy code number of CGP-115. This version includes both a Centronics and a serial interface, which makes the printer usable on practically any microcomputer which uses reasonably standard interfaces. Since the Tandy stores offer a good service on both spares (pens, paper etc.) and trouble-shooting, it makes sense to buy the Tandy version as there is a Tandy store in most large towns. In addition to being used as a printer, however, this machine acts as a graphics plotter, and you can draw diagrams and other pictures by means of instructions sent from *any* computer. This applies even if the computer has no graphics capabilities of its own.

The CGP-115 in detail

The printer uses a plain paper roll which is 4½ inches wide. Tandy stores sell three rolls, each about 145 to 150 feet long, for just under £5. These paper rolls are also used by a wide variety of adding machines, so if you haunt your local office supply stores, you may find alternative sources at lower prices. The paper is tightly gripped by the printer, because it is moved around a lot in the course of printing. The printing carriage consists of a holder which is loaded with four miniature ball pens. This holder can be rotated so that one pen is touching the paper. Printing is achieved by moving the pen holder from side to side and the paper up and down, and is such a fascinating sight that you'll probably print listings over and over again just for the pleasure of watching the mechanism! I know I did. When the printer is switched on, it



```
10 REM DIRECTIONS
20 PRINT#8,CHR$(18)
30 PRINT#8,"M50,0"
40 INPUT"Your name, please ";NM$
50 PRINT#8,"P";NM$
60 PRINT#8,"Q1"
70 PRINT#8,"P";NM$
80 PRINT#8,"Q2"
90 PRINT#8,"P";NM$
100 PRINT#8,"Q3"
110 PRINT#8,"P";NM$
120 PRINT#8,"Q0"
130 PRINT#8,"A"
140 END
```

Figure 8.13. A short listing and its result, to illustrate the use of the Tandy CGP-115 graphics printer.

goes into a 'pen test' routine, slowly drawing a square in each colour so that you can check that none of the pens has run dry. They have a surprisingly long life, and each pack of three pens costs around £1.99 from Tandy stores. You won't find alternative supplies quite so easily in this case!

Normally, the CGP-115 acts as a printer, and you can use it to print listings. It is not a fast printer by any stretch of the imagination, even compared with a daisywheel but the results are much easier to read than some dot matrix outputs. The enormous advantage of using the Tandy printer, however, is that it can be used as a graphics plotter. This means that if you send suitable instructions to the printer, it will draw diagrams. The instructions are not the same as the graphics instructions of the CPC6128 (or any other computer), but this is not a disadvantage. If at some stage you change to another computer, the Tandy printer will still be useful, and the graphics programs that you have used with the CPC6128 can easily be adapted to another computer. This is very useful to know if your household is on the verge of becoming a two-computer family. The CGP-115 has a small set of four switches at the back which can be used for setting up the printer. For the CPC6128, the settings of the switches are: 1.OFF, 2. ON, 3. OFF, 4. ON. This gives the correct line feed and the normal size of print, with the parallel interface in use.

The Tandy CGP-115 commands

Because this book is mainly concerned with the use of the Amstrad CPC6128 and several different printers, I have had to resist the temptation to add several chapters on the Tandy graphics printer. Many CPC6128 owners, however, will probably want to make use of this type of printer mechanism, which is sold under a variety of other brand names. For business applications, for example, the ability of the CGP-115 to produce graphs in four colours is extremely useful for such a modestly-priced unit. The following is a list of the commands which are available when the Tandy version is used. The commands are shown in their CPC6128 form. Figure 8.13 demonstrates the use of these commands in printing a name in four different directions.

PRINT#8,CHR$ (8)	Move one space left (backspace); used in text mode
PRINT#8,CHR$ (11)	Reverse line feed – move paper down by one line in text mode
PRINT#8,CHR$ (17)	Select text mode from graphics mode
PRINT#8,CHR$ (18)	Select graphics mode from text mode
PRINT#8,CHR$ (29)	Change colour in text mode

Graphics commands

The following letters can be sent when the printer is in *graphics* mode. The letters are *not* printed; instead, they are used as commands. Several of these commands must be followed by numbers, such as X, Y coordinate numbers, to specify positions. All of these letters would be sent to the printer by using PRINT#8, after executing PRINT#8,CHR$(18).

A	Reset pen to left margin, no line drawn, return to text mode.
Cn	Change colour of pen. n is colour number, 0 to 8.
Dx,y	Draw from present position to point x,y. Can be extended to more than one point.
H	Move pen to origin without drawing a line. The origin is a specified starting point.
I	Set new origin at current pen position. If you want a new origin at point 5,10, then place the pen there, and PRINT#8, "I"

Jx,y	Jump, or draw-relative. Draws a line from present position to one x steps to the right and y steps up. Do not confuse this with D, which draws to the *absolute* point x,y.
Ln	Change line type. If n=0, the line is solid, but using numbers 1 to 15 will draw various dotted lines.
Mx,y	Move to point x,y without drawing a line.
Pchars	Print the following characters while the printer is in graphics mode. The size of the characters can be controlled, and characters can be printed vertically or backwards.
Qdir	Change print direction. The number dir, can be in the range 0 to 8. 0 gives normal printing, 1 gives top to bottom, 2 gives upside down, 8 gives bottom to top.
Rx,y	Relative move. Move pen, without drawing, to a point x steps to the right and y steps up. Using −x moves pen left, using −y moves pen down.
Sn	Selects size of characters to be printed. n must be between 0 and 8.
Xa,b,c	Draw graph axis. a is 0 for a Y axis, 1 for X axis. The distance between marks on the axis is specified by b, which must be between −999 and +999. The number of steps is c, between 1 and 255.

Chapter Nine
The Extra Memory

The address in memory for the start of BASIC in the older CPC464 and CPC664 computers was 368 denary, &170 in hex, and the CPC6128 uses the same address. The upper limit of memory is the address called HIMEM, which can be found by typing PRINT HIMEM as a direct command. On the old models, this gives 42619 denary (&A67B hex) and the same figure is once again obtained for the CPC6128. In other words, the space which is available for BASIC programs is exactly the same as it was in the older machines, and you may well wonder where the extra 64K of the CPC6128 has gone. The reason for the unaltered value of HIMEM is that the Z80 microprocessor which is used in of the Amstrad machines (to date) can work with a maximum of 64K of memory at a time. As it happens, even the smaller CPC464 and CPC664 machines use more than a total of 64K of memory, because they use 64K of RAM, and 32K of ROM. Of the 64K of RAM, the top 16K is used for the screen and about 7K for the operating system (I'm giving figures for the disc-equipped machines here), with the rest available for your programs. To do this, the operating system had to be able to switch between addressing RAM and addressing ROM, a method that is called 'bank-switching'. On the earlier machines, the memory layout was as shown in Figure 9.1, with the top 16K and the bottom 16K of memory switched between ROM and RAM according to the needs of the machine.

On the CPC6128, this scheme has been extended to switch in another bank of 64K of RAM. Once again, this has to be done by the operating system, but unless provision is also made for controlling the switching from BASIC, the extra memory would be useful only to machine code programmers. If you program in machine code, then the Amstrad firmware manual for the CPC6128 gives details of the calls to the operating system. For programmers in BASIC, however, a program called 'Bank Manager', is provided on side 1 of the system discs. This loads in as a resident system extension (RSX), meaning a control program in a reserved part of the RAM rather than in ROM. Once Bank Manager has been loaded, you can gain some access to the extra memory from BASIC programs. In this chapter, we shall look briefly at how the additional memory can be used by way of the extra commands gained through loading Bank Manager.

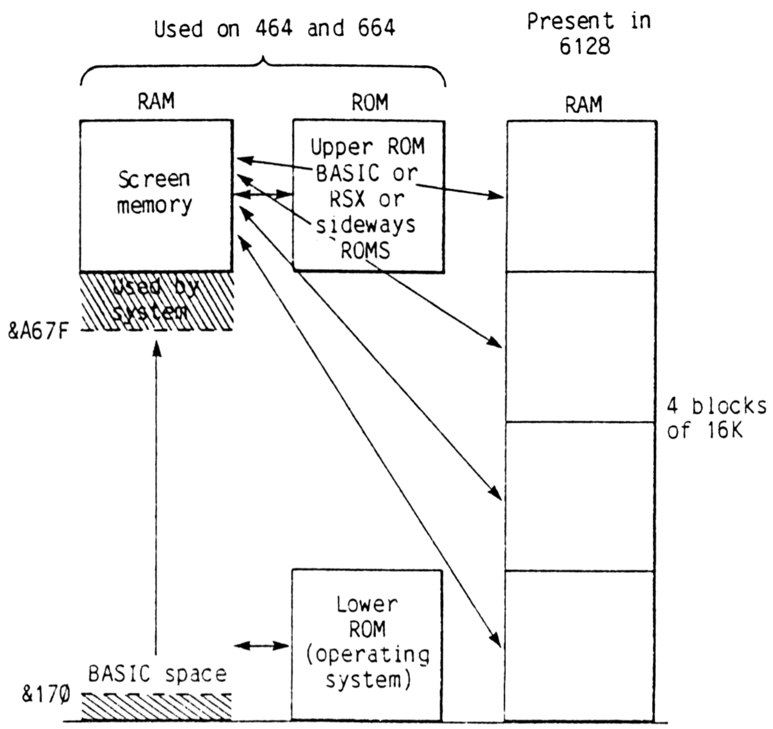


Figure 9.1. The memory layout of the older Amstrad machines, and how the extra memory of the CPC6128 is located.

The essential first step to using the extra memory is to load and run the program on side 1 of the system discs by using the command RUN"BANKMAN". This loads and runs a protected BASIC program, BANKMAN.BAS, which in turn reads in machine code which is poked into memory in the form of an RSX. This machine code is held on the disc as BANKMAN.BIN. As an alternative, you can make use of the short routine which is illustrated in Figure 9.2. This is intended to be a loader for each BASIC program that you want to be able to use extra memory. The routine reserves 1317 bytes of memory immediately below the default HIMEM address, and then puts the BANKMAN.BIN codes into this space. The

```
10 SYMBOL AFTER 256
20 ADDRESS=HIMEM-1317
30 MEMORY ADDRESS-1
40 LOAD"BANKMAN.BIN",ADDRESS
50 CALL ADDRESS
60 SYMBOL AFTER 240
70 REM LOAD PROGRAM HERE
```

Figure 9.2. A BASIC loader for the BANKMAN.BIN machine code program. This loader should be used for every BASIC program that will access the extra memory.

machine code is then run by using 'CALL address'. Your own BASIC program is then loaded by the statement in line 70, wiping out the loader portion. To make use of this, you have to ensure that a copy of BANKMAN.BIN is included with each disc of BASIC programs which use the additional memory. The copy of BANKMAN.BIN can be transferred using PIP in the usual way. The loader program itself must be saved separately, under a filename which will help you to remember which program it is associated with. For example, if you use the loader to gain extra memory for a program called DATMAS, then a name such as LODDMAS.BAS could be used. Once the main BASIC program has been thoroughly debugged, you can change the LOAD in line 70 of the loader into RUN "filename". Once this has been done, then using the loader will load in Bank Manager, and load and run your BASIC program with no further effort on your part.

The BANKMAN commands

Once BANKMAN has been run, a new set of commands is made available to extend the BASIC of your machine, hence the name resident system extension. These commands allow you to to use the additional RAM for storing the contents of four blocks of screen memory (16K each, remember), or to store data as if the extra RAM were a disc. What you *cannot* do is to write BASIC programs that extend into the extra RAM space, other than storing their data in it. It's important to point this out, because at the time of writing there are two machines which *can* make use of bank-switched RAM so as to allow longer BASIC programs. There are, however, very few programs in BASIC which are likely to need such a lot of space, and the problem can better be solved by using the CHAIN and CHAIN MERGE facilities of the CPC6128. The BANKMAN commands, like some of the AMSDOS commands, are used with the | prefix.

The first group of BANKMAN commands is concerned with switching the extra memory in order to store screen data. The memory allocated for serving the screen display is 16K, which is normally taken from the top end of the first block of RAM, and is also shared with the upper ROM which contains the BASIC interpreter. By using all of the extra 64K for screens, we have a total of 5 different screen displays which can be stored and switched in and out as needed. These are numbered 1 to 5, with screen 1 being the default one at the top of the main RAM bank. This screen memory is *always* the one that is used to provide the picture to the monitor. If you want to display the contents of any one of the other stored screens, you must transfer *all* the bytes of that screen into the memory addresses of the standard screen, &C000 to &FFFF (denary 49152 to 65535). The new commands allow this to be done quickly and efficiently, with no knowledge of machine code required. The two commands for this purpose are |SCREENSWAP and |SCREENCOPY.

|SCREENSWAP is used with two or three number parameters following it. The first parameter is an optional one, and if used must be a number between 0 and 63. This number represents the *fraction* of the screen that is to be swapped, in 64ths. For example, if you wanted to swap half a screen, you would use the number 31 – not 32, because the count is 0 to 63, not 1 to 64. You might think that this would allow you to perform such tricks as swapping windows without affecting the rest of the screen, but it's not so simple as that, because unless you know how the screen addresses are used, you can't be sure which fraction of the screen memory contains the data you want. If this parameter number is not used all of the screen will be swapped, and there must be a comma between the SCREENSWAP command word and the next number. The other two numbers, which you *must* supply, are the numbers of the screens you want to swap. For example:

```
|SCREENSWAP,1,3
```

would exchange the contents of screen 1 and screen 3. In other words, you would now be looking at the screen picture from screen 3, with the old picture from screen 1 now stored in screen 3. Obviously, repeating the command would restore the original screen picture. You can also swap contents of screen memory blocks which are not visible (that is, not including screen 1), but this action is a slower one.

|SCREENCOPY is the complementary command which copies rather than swaps screens. It can take the same three numbers, of which once again the first one, the screen fraction, is optional. This command, however, requires a lot more care than |SCREENSWAP. The reason is that if you get the screen numbers the wrong way round, you could copy a blank screen memory block on to a full block, effectively clearing a screen. The first of the screen numbers is the destination block, the second is the source, so that:

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

would make a copy of whatever is being viewed (screen 1) on to screen block 2. This does not affect what is on screen 1, so the monitor picture is unchanged. The demonstration program in Figure 9.3 illustrates how |SCREENCOPY can be used to store screen memory blocks by placing (in this example) text on the viewed screen and transferring it to the others. The first four screen displays are transferred into the extra memory using |SCREENCOPY, with MODE 1 being used to clear the screen each time and ensure that the screen has not been scrolled. This is very important, because the Amstrad machines perform scrolling by changing the screen addresses. This means that if the main screen scrolls between the time a picture is stored and when it is recovered, the result will be a very jumbled picture. This might be an interesting basis for a 'name that picture' game, but it's not an effect that we normally want. In addition, you must be in the same

```
10 MODE 1:LOCATE 10,1:PRINT"This is now
stored in 2"
20 |SCREENCOPY,2,1
30 GOSUB 1000
40 MODE 1:LOCATE 10,6:PRINT"This is now
stored in 3"
50 |SCREENCOPY,3,1
60 GOSUB 1000
70 MODE 1:LOCATE 10,11:PRINT"and this is
 in 4"
80 |SCREENCOPY,4,1
90 GOSUB 1000
100 MODE 1:LOCATE 10,16:PRINT"This is in
 the 5th."
110 |SCREENCOPY,5,1
120 GOSUB 1000
130 MODE 1:LOCATE 10,21:PRINT"This is th
e current screen"
140 LOCATE 1,1:END
1000 LOCATE 10,25:PRINT"Press any key to
 store"
1010 WHILE INKEY$="":WEND
1020 RETURN
```

Figure 9.3. How to use|SCREENCOPY to store screen images in the extra memory bank.

screen mode when you recall a screen as you were when it was stored. You cannot expect to store a screen from mode 1 and recall it to mode 2, for example. The simplest way of ensuring that these requirements are met is to perform a mode command before a screen is created, avoid scrolling when text or graphics are put on the screen, and perform another identical mode change when the screen is recalled.

When you have run the demonstration program of Figure 9.3 you will have four screens stored in the memory and a message remaining on the visible screen, screen 1. You can now switch between screens by using |SCREENSWAP, using MODE 1 to clear the screen before each direct |SCREENSWAP command. Try, for example, |SCREENSWAP,1,2. Performing this twice will restore the original screen, and you can also try the effect of |SCREENSWAP,3,4: |SCREENSWAP,1,3 to the speed of swapping the screens that are not on display. Try also moving the cursor to the bottom of the screen to cause scrolling, and then use another |SCREENSWAP.

You'll see from all of this that the screenswap action for a complete screen is rather slow, which rather limits the use of the action. It's slow because so much memory is being juggled at a time, and you can't get around this simply by swapping fractions of a screen, because you can't be sure which fraction will contain your screen image. For example, just to store the data

2 pixels in mode Ø
4 pixels in mode 1
8 pixels in mode 2

Left							Right
	CØØØ	CØØ1	CØØ2	- - - - - - - - - - -	CØ4D	CØ4E	CØ4F
1st	C8ØØ	C8Ø1	C8Ø2	- -	C84D	C84E	C84F
row of	DØØØ	DØØ1	DØØ2	- - - - - - - - - - -	DØ4D	DØ4E	DØ4F
char-	D8ØØ	D8Ø1	D8Ø2	- - - - - - - - - - -	D84D	D84E	D84F
acters	EØØØ	EØØ1	EØØ2	- - - - - - - - - - -	EØ4D	EØ4E	EØ4F
8 lines	E8ØØ	E8Ø1	E8Ø2	- - - - - - - - - - -	E84D	E8HE	E84E
deep	FØØØ	FØØ1	FØØ2	- - - - - - - - - - -	FØ4D	FØ4E	FØ4E
	F8ØØ	F8Ø1	F8Ø2	- - - - - - - - - - -	F84D	F84E	F84F

(The next line now starts with CØ5Ø)

	CØ5Ø	CØ51	CØ52	- - - - - - - - - - -	CØ9D	CØ9E	CØ9F
2nd	C85Ø	C851	C852	- - - - - - - - - - -	C89D	C89E	C89F
row of	DØ5Ø	DØ51	DØ52	- - - - - - - - - - -	DØ9D	DØ9E	DØ9F
char-	D85Ø	D851	D852	- - - - - - - - - - -	D89D	D89E	D89F
acters	EØ5Ø	EØ51	EØ52	- - - - - - - - - - -	EØ9D	EØ9E	EØ9F
	E85Ø	E851	E852	- - - - - - - - - - -	E89D	E89E	E89F
	FØ5Ø	FØ51	FØ52	- - - - - - - - - - -	FØ9D	FØ9E	FØ9F
	F85Ø	F851	F852	- - - - - - - - - - -	F89D	F89E	F89F

(The next line now starts with CØAØ)

Difference between lines in a character = #8ØØ
Difference between rows in a character = #5Ø

Last row ends with FF8Ø FF81 ... FFCF. Bytes C7DØ to C7FF, up to CFDØ to CFFF are *unused.*

Figure 9.4. The layout of the screen map. This shows that the pixels that make up a character are stored at a range of addresses. Scrolling is carried out by altering the start of display address.

for the top line means that you need to work with 6223 bytes of screen memory, and these bytes will also contain data for other parts of the screen. This is because of the way that the screen memory is used, and Figure 9.4 illustrates what I mean. The provision for swapping or copying one 64th at a time is not intended to allow you to swap pieces of screen so much as to interleave the screen swapping with other actions. Take a look at the action of Figure 9.5, for example. This copies in one screen, and then swaps back one section at a time. By watching this slow motion swap, you can see how the line of text is built up as the screen message is obliterated. The delay loop in this example represents other actions which the computer could be carrying out during this time.

```
10 MODE 1:LOCATE 10,1:PRINT"This is now
stored in 2"
20 |SCREENCOPY,2,1
30 GOSUB 100
40 MODE 1:LOCATE 10,21:PRINT"This is the
 current screen"
50 FOR N%=0 TO 63
60 |SCREENSWAP,N%,1,2
70 FOR J=1 TO 1000:NEXT
80 NEXT
90 END
100 LOCATE 10,25:PRINT"Press any key to
store"
110 WHILE INKEY$="":WEND
120 RETURN
```

Figure 9.5. Screen swapping one 64th at a time!

RAM disc use

The alternative use for the extra memory is as a form of 'RAM disc'. This is a form of memory store that has been available for some time on a number of other machines, notably the BBC Micro, and which can be very useful. A RAM disc is a section of memory that can be used for storing data just as if it were a disc. Unlike the disc access of the Amstrad disc system, however, the RAM disc can be used with random access. The normal disc drive commands allow only serial access to data files, meaning that if you want to read just the 105th item in a file, you must read items 0 to 104 first of all. Random access implies that just one record, the one that you need, can be read *and* written. If you are a little rusty on this subject, I suggest you re-read Chapter 4 of this book. The BASIC of the CPC6128 does not permit random access files on a disc, though this can be accomplished using machine code. In addition, of course, because no mechanical action is needed, the use of a RAM disc is very fast. If you need to retain the data permanently, of course, you will still need to make use of the disc system for storing each file. With a RAM disc, however, you can load a great deal of data in the RAM, and from then on dip into it as you need to with no further need to use the disc drive until you want to record the whole memory content back on the disc again.

One new concept that has to be mastered before you can profit from this RAM disc use is the 'current record'. The current record is simply the record in a data file that can be accessed. The default current record is the one that is the first record in the memory bank, number 0. If you use this record, either reading it or writing a replacement, then the current record number changes to the next record along. The whole scheme is based on the idea of keeping a 'pointer number', which is set to the number of a record, and which automatically increments unless you instruct otherwise. Machine code

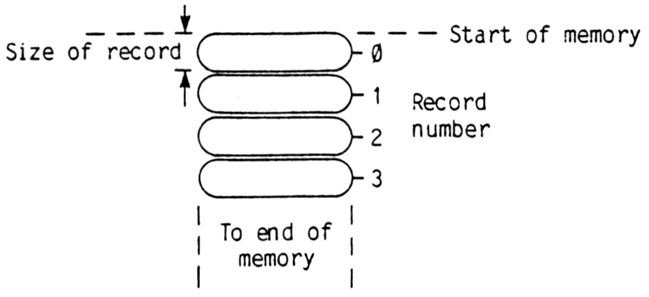


Figure 9.6. Illustrating how the records are stacked in the memory.

programmers, and those who have used random access filing on other computers will recognise the principle. The commands for using random access also allow you to set this pointer number to any (legal) value that you like, so that any record can be read or written. If you think of the records as being arranged in a pile, with a reference number attached to each (Figure 9.6), then you can see how comparatively simple it all is.

Like anything that is basically simple, however, there are complications. The main complication is that this simple method of gaining access to any record is possible only if all of the records are of *exactly the same length*. The reason is not difficult to see. If each record is, for example, 100 bytes long, then the records can be stored end to end in the memory. If you want item number 10, then you only have to look 10 × 100 bytes along from the start. Many random access systems work just like this, in fact, with a number that can be made to point to each individual byte. The second, rather smaller, restriction is that all data must be stored in string form, just as it would be on an ASCII coded disc file. This implies that any numbers which are to be contained in a data file must be converted to string form by the use of STR$, and when such a file is read back the string representation of the number may need to be reconverted by using VAL. This, of course, applies only if you want to perform arithmetic. For other numbers, such as reference numbers, ages, numbers in addresses and so on, you could use strings from start to finish, entering the number as a string from the keyboard, and displaying it also as a string. The number form is needed only for items like money which will be added or subtracted. When you have decided on the form of the string that you will use, and the number of characters in it, you set this up by use of the |BANKOPEN N% command, in which N% is the fixed number of characters in each string.

The command that is used for storing each string of a record into memory is |BANKWRITE. The syntax is of the form:

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

where r% is an integer that will return the *next* current record number, a$ is the string that will be stored in the memory, and n% is the record number for the string that is being stored. This last number is optional, and if you omit it the current record number will be used. Normally, you would use n% only when you wanted to change a record from an existing list. The number r% should be tested after a |BANKWRITE action, because if it is negative something has gone wrong. If the value of r% is −1 you are out of memory, and if the value is −2, there has been a failure of the bank switching action for some reason. To read a record from the memory bank you need to use |BANKREAD. The syntax for |BANKREAD is exactly the same as that of |BANKWRITE, but with the difference that the string a$ will have been assigned with the content of the current record after the reading action. This is not quite so straightforward as it sounds, because |BANKREAD cannot create a string in the way that a statement like a$="THIS IS A STRING" can. Any string name that is being used in |BANKREAD must have been assigned earlier with a string of the correct length, even if this is just a string of blank spaces. For example, you might use:

```
a$=SPACES$(40):|BANKREAD,@r%,a$
```

to make use of 40 character strings read from a file. In this example, if the record consists of a string of more than 40 characters, the excess characters are simply lost. If the string in the memory is less than 40 characters, the remainder consists of blanks as you might expect.

String length is important, both in reading and in writing records. If you decide, for example, to write 40 character records, you must make sure that each string you use is padded with blanks to a length of 40 characters. If you do not do this you can encounter problems. There will be no problems if you simply write and read the file, but you will have trouble if you replace records. The point is that if you have initially recorded a record of, say, 38 characters, and you then replace it with a record of 25 characters, then only 25 characters of the file will be replaced. This means that when you read the record back, you will find the 25 character record that you used for replacement, plus the additional characters, 13 of them, left over from the old record. If all records are padded to a standard size in the program that uses the records, no such problems will arise.

Once a random access file has been established with |BANKOPEN and |BANKWRITE, another useful action is to search through the file for a particular record. This is done by using |BANKFIND. The syntax is:

```
|BANKFIND,@r%,a$,st%,nd%
```

where r% is the integer variable which contains the next record number. Once again, if this number is negative a problem is indicated. Numbers of −1 and −2 have the same meanings as with |BANKWRITE, and a value of r% equal to −3 means that no match has been found; the record cannot be located. Your program should test for this eventuality, and the action can also be used to test for reaching the end of a file. As before, a$ is the string that is searched for in the record, which should be prepared in advance, and the numbers st% and nd% represent the starting and ending record numbers between which the search is conducted. If you omit these numbers, the whole of the bank will be searched. A form of 'wildcard' can be used when the search string a$ is specified. For example, if you specify:

a$=STRING$(12,0)+"file"

then this string would match any 16 character string which ended with 'file'.

Figure 9.7 is a listing for a short demonstration of the RAM disc commands. It has been written as a BASIC program which will need to be

```
10 MODE 1
20 R%=0:|BANKOPEN,15
30 FOR N%=1 TO 10
40 LOCATE 1,5:PRINT"Enter word number ";
N%;SPACE$(16)
50 LOCATE 22,5:INPUT INWORD$:INWORD$=INW
ORD$+SPACE$(15)
60 |BANKWRITE,@R%,INWORD$
70 NEXT
80 CLS:PRINT"Words stored are:- ":PRINT
90 R%=0:|BANKOPEN,15
100 WHILE R%<=9
110 OUTWORD$=SPACE$(15)
120 |BANKREAD,@R%,OUTWORD$
130 PRINT OUTWORD$
140 WEND
150 N%=1:WHILE N%
160 PRINT:PRINT"Select a word to search
for."
170 PRINT"Use RETURN alone to end"
180 INPUT WORD$
190 IF WORD$="" THEN 300
200 |BANKFIND,@R%,WORD$,1,11
210 GOSUB 310
220 IF R%>=0 THEN PRINT WORD$;" is entry
 number ";R%
230 PRINT:PRINT"Select a record number t
o find"
240 INPUT RECNUM
250 WORD$=SPACE$(15)
260 |BANKREAD,@R%,WORD$,RECNUM
270 GOSUB 310
280 IF R%>=0 THEN PRINT"Record No. ";R%;
" is ";WORD$
290 WEND
300 END
310 IF R%<0 THEN PRINT"NO SUCH RECORD"
320 RETURN
```

Figure 9.7. A demonstration of the use of the RAM disc.

loaded by its own loader section, as illustrated in Figure 9.2, so that you can use it directly. Line 20 allocates a record number of zero, and opens the banked memory for strings of 15 characters length, and lines 30 to 70 use a loop to place strings into the memory. You are asked to type words, and in line 50 each word has a string of 15 spaces added to it. This ensures that the word is at least 15 spaces long even if you enter nothing. There is no need to use LEFT$ to cut each entry to a total length of 15 characters, because the use of |BANKOPEN,15 will do this for you. Each use of|BANKWRITE then produces a value of R% for the next writing action, so that the incrementing of R% is completely automatic. Lines 80 to 140 then replay the strings to check the action. Line 90 resets the value of R% to zero so that the whole range of records can be used, and|BANKOPEN,15 prepares again for use – note that there is no corresponding 'BANKCLOSE' command. The WHILE loop then runs through the known values of R%, preparing a string of blanks to receive each record, and then reads the records using |BANKREAD. The recovered word is then printed. Note that you can't assign OUTWORD$=SPACE$(15) outside the loop, because the string must be filled anew with blanks each time. The program ends by demonstrating finding a word, using|BANKFIND, and finding a record of a particular number using the final parameter number in |BANKREAD. Notice that the reading section of the program has used Record No. 0, but this name is concealed from the name search action by using the range 1,11 in line 200. You can, however, specify record 0 in line 240 and get this item. The ability to select from a range can be very useful if, for example, you want to reserve some places for words that the user cannot see (passwords, for example).

Before I leave you with this new-found memory use, there are a few useful points and examples to look at. One is the use of string padding in the conventional way. In the previous example, the string was padded by using the string length parameter in |BANKOPEN, but you will sometimes want to show the padded string before using the RAM disc, and so the string has to be cut to length by using LEFT$. The short routine in Figure 9.8 shows the conventional padding action, adding spaces and then chopping with LEFT$. Another important action is packing and unpacking strings. A considerable amount of |BANKWRITE actions can be eliminated if a

```
10 REM STRING PAD
20 length%=20
30 FOR N%=1 TO 5
40 INPUT "Name- ";Name$
60 Name$=LEFT$(Name$+SPACE$(length%),len
gth%)
70 PRINT Name$+"";LEN(Name$)
80 NEXT
90 END
```

Figure 9.8. A routine for padding a string to length.

```
10 REM STRING PACK/UNPACK
20 length%=15
30 A$=""
40 FOR N%=1 TO 5
50 INPUT "Name- ";Name$
60 Name$=LEFT$(Name$+SPACE$(length%),len
gth%)
70 A$=A$+Name$
80 NEXT
90 PRINT A$
100 PRINT"Press any key...."
110 K$=INKEY$:IF K$="" THEN 110
120 FOR N=1 TO 5
130 PRINT MID$(A$,length%*(N-1)+1,length
%)
140 NEXT
150 END
```

Figure 9.9. Packing and unpacking a string to use as a record.

complete record is written as a single string. This assumes, of course, that the record is short enough to be made into a single string. As it happens, most records are and those which aren't can certainly be packed into two strings or possibly more. Figure 9.9 shows the principle, in which five strings, each of length fifteen characters are packed into one single string. This could be stored in RAM disc (not shown here), and then recovered and unpacked by the second part of the routine. This can still be done even if the fields are of unequal lengths, but the action is slightly more complicated. A variation on this is to pack and pad to a long string, perhaps of 200 characters, so that the fields of each record can be of almost any reasonable length. This normally works well on the 'swings and roundabouts' principle that one long string is often accompanied by several short ones, and the maximum size is not often exceeded.

The last topic is how to load the extra memory bank from disc, and save its contents on to disc. Figure 9.10 illustrates the essential techniques that you will need for this. Lines 20 to 70 simply create a file on the disc, called TESTFIL, which consists of the word TESTWORD and a number for each record. The next section of the program shows how this disc file can be read into the RAM disc. The procedure is completely straightforward, reading a string from the disc, making its length the required size, and then placing it into the RAM disc by using|BANKWRITE. The last section is not quite so simple. You could, of course, if you always knew how many records might be stored, use a FOR ... NEXT loop to read back from the RAM disc and write onto the serial file. It's usually better, however, not to have to rely on a number. The trouble here is that the RAM disc does not use the type of EOF that the serial disc uses, and it's not quite so easy to detect the end of the file. You can't find the end by looking for R% to become negative, because this

```
10 REM MAKE DISC FILE
20 OPENOUT "TESTFIL"
30 FOR N%=1 TO 50
40 A$="TESTWORD "+STR$(N%)
50 PRINT#9,A$
60 NEXT
70 CLOSEOUT
80 REM NOW READ INTO RAM-DISC
90 PRINT"Press any key to input data"
100 WHILE INKEY$="":WEND
110 |BANKOPEN,20
120 R%=0
130 OPENIN "TESTFIL"
140 WHILE NOT EOF
150 INPUT#9,A$
160 A$=LEFT$(A$+SPACE$(20),20)
170 |BANKWRITE,@R%,A$
180 WEND
190 CLOSEIN
200 REM NOW PUT IT BACK IN ANOTHER
210 PRINT"Press any key to return file"
220 WHILE INKEY$="":WEND
230 R%=0
240 OPENOUT "NEWFIL"
250 |BANKOPEN,20
260 WHILE R%>=0 AND A$<>STRING$(20,0)
270 A$=SPACE$(20)
280 |BANKREAD,@R%,A$
290 PRINT#9,A$
300 WEND
310 CLOSEOUT
```

Figure 9.10. Passing data between the RAM disc and the disc system files.

happens only at the end of the 64K block of memory. The obvious thing to test for is A$=SPACE$(20), but this does not work either. When you test to find out just what A$ consists of after all the strings have been read, you will find that it is 20 characters of ASCII 0. If we test for this, then, all is well, and the program as shown works. You have to remember, though, that this end test depends on the 64K of memory being clear at the end of the file. If the bank has been used for a longer file previously, and the computer has not been switched off to clear this, then the end test will not work. One minor mystery, also, is that TESTFIL appears in the catalogue as a 1K file, and NEWFIL appears as a 2K file. The reason is that the words which have been read from the RAM disc have been padded to 20 characters, whereas when they were originally recorded, they were only as long as they needed to be. If this could be troublesome, you would need to add a routine to remove spaces, such as that in Figure 9.11.

```
10 REM remove spaces
20 A$="TEST"+SPACE$(16)
30 K%=20
40 WHILE MID$(A$,K%,1)=CHR$(32)
50 K%=K%-1:WEND
60 A$=LEFT$(A$,K%)
70 PRINT A$;" ";LEN(A$)
80 END
```

Figure 9.11. Removing spaces from records that have been stored in the RAM disc.

Appendix A
Using a TV Receiver

For some purposes, the use of a TV receiver along with your CPC6128 can be very useful. If, for example, you have bought the CPC6128 for business purposes or for running any CP/M software, you will find the green screen monitor much easier on your eyes for the 80 column screen display than the colour monitor. If you sometimes want to make use of low-resolution colour displays, then a TV receiver is a low-cost alternative to having both a green screen and a colour monitor. To make use of a TV receiver, however, you need a TV modulator, which will provide power for the CPC6128 and convert the colour signals from the computer into the form that a TV receiver can use. Note, however, that though it's convenient to be able to use a TV receiver (and cheap as well), the picture that you see is of a poor standard. This is because a TV receiver was never designed to accept computer signals, and because of the need to convert the computer's signals into the same form as transmitted signals. The result is that the letters and other shapes on the screen look fuzzy, and the colours look streaky.

To connect the CPC6128 to a TV receiver you will need to plug the aerial lead from the modulator pack to the TV. Unless you can keep a TV receiver specially for use with the CPC6128, you will find that you have to keep plugging and unplugging the aerial cable and the CPC6128 cable. This is never a good thing to have to do because it loosens the contacts of the socket on the TV, and an alternative is to use the type of adaptor that is illustrated in Figure A.1. This allows you to plug a lead into the aerial socket of the TV so that the TV can be used both for the CPC6128 and for *Dynasty* without having to pull plugs out. The two-way TV adaptor that I use is sold in TV stores under the name Panda. You need to connect the CPC6128 to the TV or to the adaptor using the special cable provided with the CPC6128 modulator. If this lead is too short for you, you can buy extension pieces of cable and cable joiners. If you have the adaptor, then all you have to do to change between computing and TV watching is change channels.

The next step is to switch on the TV receiver and the CPC6128. Unless you are exceptionally lucky you will probably see nothing appear on the screen. This is because a TV receiver has to be tuned to the signal from the CPC6128. Unless you have been using a video cassette recorder, and the TV

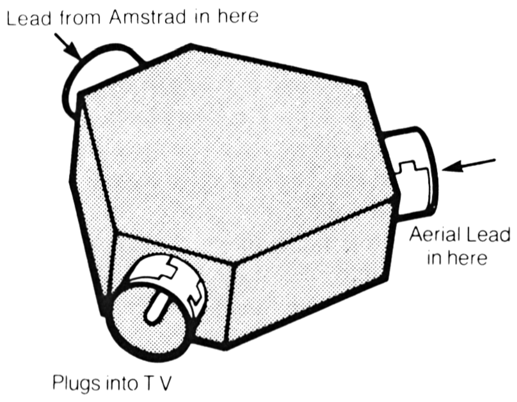


Figure A.1. A useful TV aerial cable adaptor.

has a tuning button that is marked VCR, it's unlikely that you will be able to get the CPC6128 tuning signal to appear on the screen of the TV simply by pressing tuning buttons. The next step, then, is to tune the TV to the CPC6128 signals.

Figure A.2 shows the main method that is used for tuning modern TV receivers in this country, using touch pads or very small push-buttons for selecting transmissions. These are used for selection only, not for tuning. The tuning is carried out by a set of miniature knobs or wheels located behind a panel which may be at the side or at the front of the TV receiver (Figure A.2), or sometimes in a miniature drawer. The buttons or touch pads are usually numbered, and corresponding numbers are marked on the tuning wheels or knobs. Use the highest number available (usually 6 or 12), press the pad or button for this number, and then find the knob or wheel

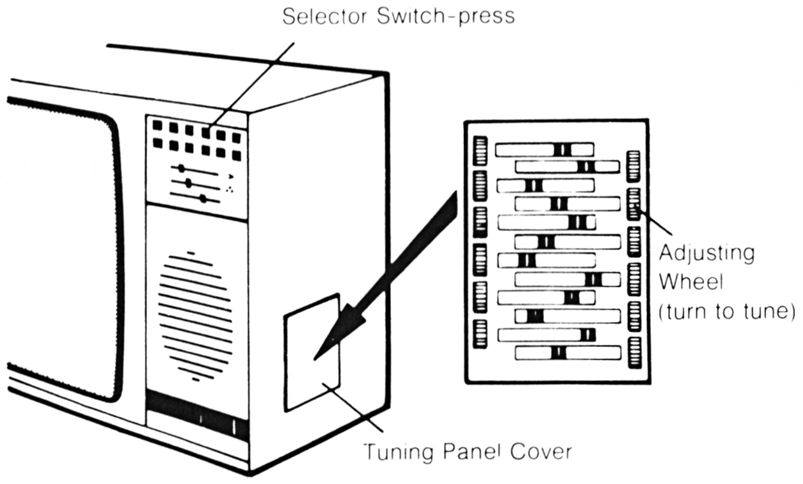


Figure A.2. The tuning system which is used on modern TV receivers.

which also carries this number. Tuning is carried out by turning this knob or wheel. What you are looking for is a clear picture on the screen and silence from the loudspeaker. On this type of receiver, the picture is usually 'fine-tuned' automatically when you put the cover back on the tuning panel, so don't leave it off. If you do, the receiver's circuits that keep it in tune can't operate, and you will find that the tuning alters and you will have to keep retuning. The CPC6128 should give a good picture on practically any TV receiver. If your TV exhibits faults like a shaking picture or very blurred colours, then check the tuning carefully. If the faults persist and the TV is correctly tuned, you will have to contact the service agents for the TV – or use a different model in future!

Appendix B
CP/M Plus CONTROL Codes

The CONTROL key is used to a very large extent in CP/M, and in programs which run under CP/M. The main actions are as listed here. The use of the CONTROL key shows on the screen as an up arrow, but most printers print this characters as a circumflex (^). When the CONTROL key is being used correctly in CP/M, however, the symbol does not appear on the screen.

↑A	Move cursor one character left.
↑B	Move cursor from one end of line to the other.
↑C	Stop program when prompt shows, or after ↑S.
↑E	Take a new line on the screen, but not into memory buffer.
↑F	Move cursor one character right.
↑G	Delete character under cursor.
↑H	Backspace and delete.
↑I	Tab along 8 spaces.
↑J	Line feed.
↑K	Delete from cursor to end of line.
↑M	RETURN action.
↑P	Send output to printer.
↑Q	Restore scrolling (after ↑S).
↑R	Put characters to the left of cursor on a new line.
↑S	Stop screen scrolling during a program.
↑U	Change buffer to show characters to the left of cursor.
↑W	Recall previous command line.
↑X	Delete from cursor to beginning of line.

Appendix C
Editing and Debugging

Editing means changing something that has already appeared on the screen. You can, of course, delete a character by using the DEL key, or you can retype a faulty line. You can also delete a range of lines by using the DELETE command (such as DELETE 10-100). Editing, however, means changing one feature of a line without having to change anything else. Any feature of a line, including its line number, can be changed by editing. The editing process can be carried out:

(a) While a line is being entered, before RETURN has been pressed.
(b) At a later stage, after a line has been entered, but before the program is run.
(c) When an error is signalled during running.
(d) When you see an asterisk appear against a line number during entry, using AUTO, to signal that this line contains a statement already.

Dealing with these in order:

(a) While a line is being entered, all of the line editing commands below can be used. Editing is completed by pressing the RETURN key.
(b) When the line has been entered, but the program has not been run, you can use either of the editing methods detailed below.
(c) When the program stops with an error message, the line in which an error has been traced may appear on the screen, with the cursor on its line number. You can edit the mistake using the line editing method.
(d) When a line number selected by AUTO corresponds to an existing line, the computer enters edit mode automatically.

Editing methods

A unique feature of the CPC6128 is that editing can be carried out in two different ways. Most computers use either line editing or screen editing. *Line editing* means that you have to call up the line you want to change, using a command like EDIT 200 (to edit line 200). The cursor can then be moved over the line to locate and correct the mistake. In *screen editing*, any line that you see on the screen can be edited simply by moving the cursor to it and correcting the mistake. Most people have strong likes and dislikes about editing methods, and will accept only one of these systems. By using both, the CPC6128 pleases everybody!

Editing commands

1. Line editing

If you don't already have the line on the screen with the cursor on it (as when an error is signalled), then use the EDIT command to get the line in place. Remember that there must be a space between the T of EDIT and the first digit of the line number. The result will be to place the line on the screen with the cursor over the first digit of the line number. You can now move the cursor, using the arrowed keys. You can delete a character which the cursor is over by pressing the CLR key. By holding down this key, you can clear everything that is to the right of the cursor. Alternatively, you can delete whatever is *to the left of the cursor* by using the DEL key. Typing a character will insert that character *at the cursor position.* Press RETURN to complete editing a line. Remember that you can edit a line in this way while you are entering it. If you have typed:

```
PRINT THIS IS THE END"
```

and you suddenly notice that you have left out the first quotes, then use the left arrow cursor key to position the cursor over the 'T' of THIS, press the quotes key, and then RETURN. It's important to note about line editing like this that *you can press RETURN whenever you have corrected the mistake*; you don't have to shift the cursor to the end of the line.

A useful variation, when you have a number of consecutive lines to edit, is to type AUTO, followed by the line number of the first line you want to edit. This will bring the line to the screen for editing, and when you have finished editing it, the next line will be presented. You can stop the process by pressing ESC twice. You can also use this to renumber a block of lines, but you must remember to delete the old numbers. In addition, GOTO and GOSUB destinations are not automatically renumbered as they are when you renumber normally.

2. Copy editing

Copy editing is the CPC6128 version of screen editing. This method makes a copy of a line, but enables you to omit or replace parts of it. Any line you can see on the screen, and even direct commands, can be copy edited. Press the SHIFT key and hold it down. Now press the cursor keys so that the cursor moves to the line you want to edit. If you now release the cursor keys and press the COPY key you can make a copy of the line at the bottom of the screen. The copying will start from where the cursor was positioned. If you want to skip a part of the line, use the cursor key instead of the COPY key. Press RETURN when you have copied as much as you want. If you want to copy the rest of the line *you must complete the copy* before you press RETURN. If you press RETURN midway along a line, only part of the line will be copied. This is a very important difference between copy editing and line editing.

This is a very versatile editing method. You can, for example, change a line number. Suppose you have line 200 on the screen, and you want to make it 1000. Type 1000, then use SHIFT and the cursor keys to send the cursor to the first command in line 200, not to the line number itself. Now copy the line, using the COPY key. Press RETURN and you have a line 1000, identical to line 200. Type 200, and RETURN, and line 200 disappears leaving its new copy.

You can even copy something which never had a line number. Suppose that in the middle of entering a program you typed FOR N=1 TO 200:NEXT, forgetting the line number. You don't have to type it all over again. Just type the correct line number, then use the cursor keys to place the cursor over the F of FOR, and COPY the line. Press RETURN at the end of the line, and it's in place! Another excellent feature of this method, which is also used by the BBC Micro, is that you can copy part of one line into another. If you decide that the time delay you have used in line 400 is also needed in line 700, then it's simple. Ensure that both lines are on the screen. Copy line 700 until the place where you want the time delay. Shift the cursor to the place in line 400 where the routine exists, and copy it. Then go back to copying the rest of line 700, if there is any. Press RETURN when finished. This is particularly useful if you decide to make a few lines into a subroutine.

Digging out the bugs

In computing language, a fault in a program is called a 'bug', and someone who puts the faults there is called, of course, a programmer. Your programs can exhibit many kinds of bugs, and these are indicated by the error messages you get when you try to run a program. Some of these messages are pretty obvious. 'Line does not exist', for example, means that you have used a command like GOTO 1000 or GOSUB 1000 and forgotten to write line 1000. It can also appear if you have tried to DELETE 1000 with no line 1000, or if you have a THEN 1000 ELSE 2000 following an IF somewhere.

The most common fault message is 'Syntax error'. This means that you have wrongly used some of the reserved words of BASIC. You might have spelled a word incorrectly, like PRIBT instead of PRINT. You might have missed out a bracket, a comma, a semicolon, or put a semicolon in place of a colon. The most common mistake, however, is to miss out a space before or after a command word. Any spelling errors of command words, or missing spaces which allow command words to be joined to other words, can be found easily if you always enter your programs *using lower-case*. Since the computer always converts all reserved words to upper-case, you will find that mistakes are highlighted by being still in lower-case. Machines can't find out what you meant to do, they can only slavishly do exactly what you tell them. If you haven't used BASIC in exactly the way the machine expects, you'll find a syntax error being reported. Another common error is 'Improper argument'. This usually means that something silly has happened involving a number. You might, for example, have used mode 3, which is an obvious mistake, but it's not so obvious if the command happens to be MODE N%, and N% has been incremented to a value of 3. Anything that makes use of numbers, like MID$, LEFT$, RIGHT$, INSTR, STRING$, and others can use an incorrect number – and this will cause the 'Improper argument' error. You will also find that using a negative number in SQR(N), a negative or zero value in LOG(N), and other mathematical impossibilties will cause this error message. The cause shouldn't be hard to trace, because the machine tells you which line caused the trouble.

Even when you have eliminated all of the syntax errors and improper arguments, you may still find that your program does not do what it should. The CPC6128 does what any machine of the '80s should do – it gives you a lot of ways of finding out exactly what has gone wrong. One of the most powerful of these is the ESC key. As you know, this stops the action of the machine when you press it, and restarts it when you press any other key. This, as we'll see later, can be very useful for graphics bug-hunting, but for other programs, pressing ESC twice is more useful. This stops the program and prints the line number in which the program stopped. What you probably don't know, however, is that you can print out the values of variables, and even alter values while the program is stopped, and then you can make the program resume by using a CONT or a GOTO. Of the two, CONT is preferable, because it will continue from where the program left off. You cannot, however, make use of CONT if you have called up an edit or changed the program in any way. This will give a message such as 'Cannot CONTinue' or possibly 'Unexpected NEXT' (if you stopped in a loop).

Suppose, for example, you are running a program which uses a slow count, and you press ESC twice at some early stage. The program stops, and you get a message like 'Break in 40'. That line number, 40, is important, because you can start the program again at that line *providing* you don't edit, delete or add to any of the lines of the program. Suppose the counter variable is N. Try typing ?N, and RETURN. This will give you the value of N. Now try N=998 (say) and press RETURN. Type GOTO 40 (or whatever line number you stopped in). You will then see the count start again – but at 998! This is an excellent way of testing what will happen at the end of a long loop. Testing would be a rather time-consuming business if you had to wait until the count got there by itself. You can even make this testing process automatic! We have already looked at the command ON BREAK GOSUB. This will make sure that a subroutine is run whenever the ESC key is pressed twice. In this case, you can make the subroutine print the values of whatever variable you want to see, allow you to input others, and then return!

ESC used alone can be most useful when you have a graphics program that has gone wrong. When you press ESC, the graphics action (apart from flashing characters) will be frozen, and you can use this to see in what order things happen. Pressing another key then resumes the action, and there is no limit to the number of times you can press the ESC key to check on how the picture is changing. If this alone isn't enough, add a delay loop temporarily to your graphics program and run it in slow motion, using ESC to check the tricky parts.

Tracing the loops

One way in which a program can be baffling is when it runs without producing any error messages – but doesn't run correctly. This is really a sign of faulty planning, but sometimes it's an oversight. The ON BREAK GOSUB method of tracing a fault can be very useful because it allows you to print out the state of the variables at any stage in the program, and then carry on. Sometimes you will want a simpler form of tracing, though. If your program contains a lot of IF ... THEN ... ELSE lines, it often happens that one of these will not do what you expect. In such a case, the CPC6128 provides help for you in the form of two commands, TRON and TROFF.

TRON (how did you think the film got the name?) means trace on, and its effect is to print on the screen the line number of each line as it is executed. The line numbers are put between square brackets, and they are printed at the start of a screen line, in front of anything the program prints. Try typing TRON and then running a slow-acting program. TRON is particularly useful if you aren't sure what a program is doing, and it can be very handy in pointing out when something goes wrong with a loop. Remember that you can combine TRON with other debugging commands. You can, for example, stop the program, alter the variables and then continue, with TRON showing you which lines are being executed. Typing TROFF (then RETURN) switches off this tracing process.

Appendix D

Contents of Sides 1–3 and HELP File

Side 1

```
A: C10CPM3  EMS : BANKMAN  BAS : PROFILE  ENG : SUBMIT   COM
A: SETKEYS  COM : KEYS     CCP : LANGUAGE COM : SET24X80 COM
A: PALETTE  COM : SETSIO   COM : SETLST   COM : DISCKIT3 COM
A: DATE     COM : DEVICE   COM : DIR      COM : ED       COM
A: ERASE    COM : GET      COM : PIP      COM : PUT      COM
A: RENAME   COM : SHOW     COM : TYPE     COM : SET      COM
A: SETDEF   COM : AMSDOS   COM : BANKMAN  BIN : KEYS     WP
```

Side 2

```
A: AMSDOS   COM : SID      COM : DUMP     COM : GENCOM   COM
A: HEXCOM   COM : HIST     UTL : INITDIR  COM : LIB      COM
A: LINK     COM : MAC      COM : PATCH    COM : RMAC     COM
A: SAVE     COM : TRACE    UTL : XREF     COM : DD-DMP1  PRL
A: DDSHINWA PRL : DDHP7470 PRL : ASM      COM
```

Side 3

```
A: AMSDOS   COM : HELP     COM : HELP     HLP : SUBMIT   COM
A: SETKEYS  COM : KEYS     CCP : KEYS     DRL : LOGO3    SUB
A: GSX      SYS : GENGRAF  COM : DDMODE2  PRL : DDFXLR7  PRL
A: DDMODE1  PRL : DDMODE0  PRL : ASSIGN   SYS : DRIVERS  GSX
A: LOGO3    COM
```

HELP file

```
Topics available:

COMMANDS  CNTRLCHARS  COPYSYS  DATE     DEVICE    DIR
DISCKIT3  DUMP        ED       ERASE    FILESPEC  GENCOM
GET       GSX         HELP     HEXCOM   INITDIR   LANGUAGE
LIB       LINK        MAC      PALETTE  PATCH     PIP (COPY)
PUT       RENAME      RMAC     SAVE     SET       SET24X80
SETDEF    SETKEYS     SETLST   SETSIO   SHOW      SID
SUBMIT    TYPE        USER     XREF
```

Appendix E
The ASCII Codes in Hex

No.	Hex.	Char.	No.	Hex.	Char.
32	20		80	50	P
33	21	!	81	51	Q
34	22	"	82	52	R
35	23	#	83	53	S
36	24	$	84	54	T
37	25	%	85	55	U
38	26	&	86	56	V
39	27	'	87	57	W
40	28	(	88	58	X
41	29	)	89	59	Y
42	2A	*	90	5A	Z
43	2B	+	91	5B	[
44	2C	,	92	5C	\
45	2D	-	93	5D	]
46	2E	.	94	5E	^
47	2F	/	95	5F	_
48	30	0	96	60	`
49	31	1	97	61	a
50	32	2	98	62	b
51	33	3	99	63	c
52	34	4	100	64	d
53	35	5	101	65	e
54	36	6	102	66	f
55	37	7	103	67	g
56	38	8	104	68	h
57	39	9	105	69	i
58	3A	:	106	6A	j
59	3B	;	107	6B	k
60	3C	<	108	6C	l

61	3D	=	109	6D	m
62	3E	>	110	6E	n
63	3F	?	111	6F	o
64	40	@	112	70	p
65	41	A	113	71	q
66	42	B	114	72	r
67	43	C	115	73	s
68	44	D	116	74	t
69	45	E	117	75	u
70	46	F	118	76	v
71	47	G	119	77	w
72	48	H	120	78	x
73	49	I	121	79	y
74	4A	J	122	7A	z
75	4B	K	123	7B	{
76	4C	L	124	7C	¦
77	4D	M	125	7D	}
78	4E	N	126	7E	~
79	4F	O	127	7F	

Appendix F

Programmable Keys

An essential feature of a modern computer is the provision of 'soft keys'. This means that a range of the keys can be programmed to carry out actions, so that simply pressing the key provides the action. Instead of typing LIST (RETURN) each time you want a listing, for example, you can have one key do this job, or you can have a key to provide PRINT TAB(2)" each time you need this when you are writing text.

Though a number of computers provide special keys for this purpose, very few provide any simple way of programming them – and some users of other machines may well suspect that their 'programmable keys' are little more than ornamental. The CPC6128 allows you to define an action for up to 32 keys, using simple BASIC commands. There is, however, one practical restriction. The total of these characters must not exceed 120. You could use all 120 characters for one key, or make use of several keys up to the maximum of 32. The amount of memory available for these keys can be increased using machine code, but not by using BASIC.

Suppose, then that you want to redefine a key. It makes sense if you redefine a spare one that you are not going to use otherwise. You can make use of ASCII codes 128 to 159 for this purpose; the first thirteen of these codes are already allocated to keys, the keys on the numberpad at the right-hand side of the keyboard. Chapter 7, page 22 of your CPC6128 manual shows the codes for these keys, and you will see that the f0 key is allocated the code 128. To make this produce LIST (RETURN) you have to program as follows:

```
KEY 128,"LIST"+CHR$(13)
```

and press RETURN. The CHR$(13) in the definition provides a RETURN at the end of the command. When you press the f0 key on the numberpad, you will now see a listing!

Suppose you pick an ASCII code, such as 156, which has no key allocated to it? Try it – type:

```
KEY 156,"PRINT:PRINT"+CHR$(13)
```

You can now make *another key* provide ASCII 156. Take the square bracket key, which is the upper one next to the RETURN key. Type:

```
KEY DEF 17,1,156
```

and press ENTER. Now when you press this key, your PRINT:PRINT appears and is obeyed. The 17 in this command is the INKEY number for the key, so any key on the board can be redefined in this way. All of these definitions will disappear when the machine is reset with SHIFT CTRL ESC, or when you enter CP/M. If you don't want the command that is entered by a key in this way to be executed at once, just omit the CHR$(13). When you use PRINT TAB(, for example, you will always want to add the tab number, then the closing bracket, then some text.

To save on the character count you can use ? in place of PRINT in any key definition. If you need to have quote marks in a key string, you need to use CHR$(34), because the quote marks typed from the keyboard mark the end of the string. Remember too, that you can include an INPUT in a key string. If you type:

```
KEY 0,"INPUT"+CHR$(34)+"Name of file"+CHR$(34)+"; name$:
SAVE name$"+CHR$(13)
```

or incorporate this in a program, then pressing function key f0 will ask you for a filename and then save the program with that name on a disc.

Appendix G
Error Trapping

Earlier in this book we came across the idea of mugtrapping. This is a way of checking data that has been entered at the keyboard to see if it makes sense or not. The mugtrapping is carried out by using lines such as:

```
60 IF LEN(A$)=0 THEN GOTO 1000:GOTO 50
```

and you need a separate type of mugtrap for each possible error. This can be fairly tedious, and it usually turns out that there is one other error that you haven't spotted. The CPC464 is one of the exclusive few machines that offers you another mugtrapping command, ON ERROR GOTO. Figure G.1 gives a very artificial example – a real-life example would involve too much typing. In this example, the length of a word is measured and the number inverted (divided into 1). This is impossible if the length is zero, and the ON ERROR GOTO is designed to trap this. You could get a zero entry, for example, by pressing RETURN without having pressed any other keys. Now normally when this happens, you would get an error message and the program would stop. The great value of using ON ERROR GOTO is that the program does not stop when an error is found; instead it goes to the subroutine. In this example, the subroutine prints a message, then resumes on line 20. It's delightfully simple, but it's something that calls for experience. You see, if your program still contains things like syntax errors, these also will cause the subroutine to run, and this can make the program look rather baffling as it suddenly goes to another line.

```
10 ON ERROR GOTO 1000
20 PRINT"Type a word please"
30 INPUT A$
40 L=LEN(A$).
50 PRINT 1/L
60 END
1000 PRINT"Word has no letters!"
1010 RESUME 20
```

Figure G.1. Using the ON ERROR GOTO statement.

```
10 CLS
20 ON ERROR GOTO 1000
30 N%=0:LOCATE N%,1
40 PRINT"The program has continued from
the next line"
50 END
1000 IF ERR=5 THEN PRINT"There is an err
or in line ";ERL
1010 RESUME NEXT
```

Figure G.2. Analysing the error, and forcing a resumption of a program.

RESUME can be used in any of three ways. Firstly, used on its own, it will cause the whole program to run from the start when RESUME is executed. Secondly, as shown, when used with a line number it will cause the program to resume at that line. The third method is using RESUME NEXT, in which case the program will resume at the line *following the line in which the error was trapped.*

Figure G.2 shows another example. Line 20 sets up the ON ERROR GOTO command, while line 30 generates an error by using an invalid number in the LOCATE statement. This would normally cause an 'Improper argument' error message to be displayed, stopping the program. Because of the use of ON ERROR GOTO 1000, however, the detection of the error causes this line to be run, and in line 1000 the type of error is checked. If it is indeed an 'Improper argument' error, then the error message will be printed, showing the error line by use of the variable ERL. Irrespective of what type of error occurred, the RESUME NEXT in line 1010 will make the program go to line 30.

There is another command word, ERROR, which can be used to make an ON ERROR GOTO execute, and which is very useful for testing the effect of error trapping. ERROR can be used in either of two ways; with a number between 1 and 32 (the error code numbers in the manual), or using numbers 33 to 255. If ERROR is used with the standard error codes, it will make the

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR N%=1 TO 10
40 ON ERROR GOTO 0
50 PRINT"The next message has been force
d by"
60 PRINT"the ERROR command."
70 ERROR 17
80 END
1000 IF ERR=26 THEN PRINT"NEXT missing,
but program continues"
1010 RESUME 40
```

Figure G.3. Creating an error for test purposes. ERROR can also create 'user-defined' errors.

program's error trapping respond as if one of the standard errors had occurred. Using ERROR with a number between 33 and 255 allows you to create your own error messages. The standard use of ERROR is indicated in Figure G.3, in which the routine at line 1000 detects a missing NEXT in a loop – not something that you would normally leave out of a program! Line 40 then turns off the error trapping, so that when ERROR 17 is invoked, you can see the normal error message for this error appear.

Appendix H
XOR, AND, OR

Colour number	Binary form
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
10	01010
11	01011
12	01100
13	01101
14	01110
15	01111
16	10000
17	10001
18	10010
19	10011
20	10100
21	10101
22	10110
23	10111
24	11000
25	11001
26	11010

Figure H.1.

The fourth number in a drawing command has the effect of specifying the *type* of drawing action. The names of these actions (strictly speaking, they are *logic* actions) are XOR, AND and OR, and they are specified by the numbers 1,2, and 3 respectively. The first thing you need to know about these is that the commands work on the colour numbers. To see what they do, however, you have to write the colour numbers in a different form, called binary. Figure H.1 shows all 27 colour numbers written in this form, which consists of 1s and 0s. Each colour number is written as a set of five digits.

Now when we use AND, we compare each digit in one number with the digit in the same place of another number. If both digits are 1, then the result is 1, otherwise the result is 0. Figure H.2 shows how this, along with the OR and XOR rules, applies. On the screen we will be comparing the foreground colour of the INK with the colour which was there previously. When this is done with AND or OR, the results will usually create a different colour. When the OR command is used, the rule is that if one or both of the binary numbers

---

AND

The result is 1 if two 1s are present, so that 1 AND 1 is 1, 1 AND 0 is 0, 0 AND 0 is 0. For numbers with several digits, each digit in one number is compared with the corresponding digit in the other number.
For example, if we AND 1011 with 1001:

```
    1  0  1  1
    1  0  0  1

    1  0  0  1
```

then the result is 1001.

OR

If either or both digits is a 1, then the result will be a 1. 1 OR 0 is 1, only 0 OR 0 is 0. For example, if we OR 1011 with 1100, we get:

```
    1  0  1  1
    1  1  0  0

    1  1  1  1
```

and the result is 1111.

XOR

If one digit is 1, then the result is 1, but if both digits are 1 (or 0) then the result is 0. For example, if we XOR 1011 with 1100, we get:

```
    1  0  1  1
    1  1  0  0

    0  1  1  1
```

and the result is 0111.

Figure H.2. A summary of the effects of AND, OR and XOR.

has a 1, then the result will have a 1 in the same position. The XOR rule is that the result is 1 only if there is a 1 in one of the numbers, but not in both in the same position. If you have not come across these 'logic rules' before, you may well find this baffling.

Index

The Amstrad CPC6128 is equipped with ample memory, a built-in disc drive, and the latest version of the well-established CP/M operating system, all at a very competitive price. The serious user, particularly the business user, needs to know how to master the disc system and to make use of this machine with the many excellent printers that can be used with it.

The principles of CP/M, and how to use the available facilities are set out, including how to connect a second disc drive and make effective use of it.

The more advanced programmer will find details of the more advanced methods of BASIC and AMSDOS, covering such items as opening and closing disc files, file maintenance, creating, searching, sorting, amending and deleting files and making use of the additional memory for filing. An outline database program in BASIC is listed, allowing you to make your own personal database system. The graphics capabilities of the CPC6128 are covered in detail, including the CPC6128's new graphics commands, and using the additional memory for graphics. Useful information, which is difficult to find elsewhere, is provided on the sound system, such as setting up hardware envelopes and software relative envelopes.

The Author
Ian Sinclair has regularly contributed to journals such as *Personal Computer World, Computing Today, Electronics and Computing, Hobby Electronics* and *Electronics Today International.* He has written over fifty books on aspects of electronics and computing.

ISBN 0-00-383300-3

Front cover designed by Barry Morley

COLLINS
Printed in Great Britain

£9.95 net

9 780003 833003

SINCLAIR ADVANCED AMSTRAD CPC6128 COMPUTING

COLLINS