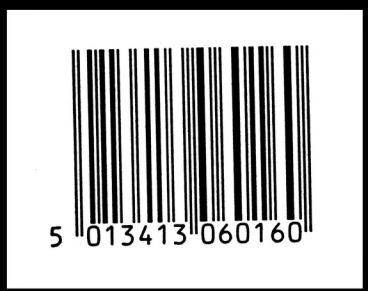
Amsoft®

**Operating Amstrad**
**CP/M 2.2 Soft 06016**



# Operating Amstrad CP/M 2.2

# Soft 06016

**AMSOFT**
A division of
**Amstrad Consumer Electronics plc**
Brentwood House
169 Kings Road
Brentwood
Essex


First edition 1985

Written by Boris Allan
Edited by Sally Tyler

Technical assistance: Cliff Lawson and Ken Clark

Published by AMSTRAD
Typeset by KAMSET typesetting graphics (Brentwood)

# Contents

## 5 Files and things



## 6 Changing the system



## 7 The Editor



## 8 Assemble and submit



## Appendix

# Chapter 1
# Introduction

This Guide assumes no knowledge of CP/M, and no real knowledge of computers, and is intended to provide sufficient background information to help the novice user to understand the nature of an operating system, and show how such a system can be used to advantage.

At the same time sufficient information is given to enable the more experienced user to see how CPC CP/M fits in with other CP/M systems, and to enable such a user to program CP/M. Programming CP/M will involve the user referring to technical documentation provided by both Amsoft and Digital Research, and will also involve the user learning 8080 and Z80 machine code.

There are two ways in which it is possible to program with CP/M: first, there is the creation of programs (say, batch files using `SUBMIT`) which only use CP/M as a working environment; and, second, there is the modification of CP/M itself to create new environments. The bias of this Guide is towards those in the first category.

The Guide starts by explaining what CP/M is so that the user can adapt to a way of thinking which may be new to those who have no experience of more complex computer systems. Though CP/M is distributed on disc and needs a disc drive, there is far more to CP/M than its file handling capabilities. CP/M is designed to help you, the user, to do things, and so the main portion of the Guide is then directed towards showing how each specific CP/M program operates.

The appendix (the Inventory) ties it all together. The Inventory is intended to be used as a reference tool for many topics which may or may not have been considered in the previous text. There are, intentionally, many cross references to other topics - to show how the system hangs together. The Inventory functions as an index, in that topics are explained there and then and so there is no need to refer to the main body of text. The index, in fact, only contains references to CP/M commands and programs.

# Chapter 2
# What is CP/M?

This chapter could be subtitled 'What is an operating system?', and to understand the nature of CP/M it helps to start by considering a microcomputer with a cassette recorder.

We will start with a very primitive set up. To load a program from the cassette recorder we enter

```
LOAD "<PROGNAME>"
```

and then have to remember to switch on the recorder. The system does not remind us to switch on, and the control of the recorder is not by the micro, but by the user. The CPC 464, for example, has a built in cassette recorder which is controlled by the computer (once the recorder is switched on), but this is not true for many other home micros.

When the program has finished loading, the user has to switch off the recorder because the micro does not control this part of the operation. If the program is loaded in parts, for example, a BASIC program loading a machine code program, then sufficient time has to be left between the parts to allow the loading to be successful. If the operation of the recorder can be controlled by the computer, it is possible to stop and start the tape between programs (as with the CPC 464).

The difficulties are greater where the saving of programs is concerned: first, there is the perennial problem of accidentally saving a program and overwriting an existing program on the tape, combined with the problem of finding what program files are on a tape; and, second, there is the realization that you have switched off, and had forgotten to press the RECORD button.

Concerning the first point, many present micros have a facility to catalogue files, but even so it is remarkable how many have no such facility. The lack of this facility means that to catalogue files on some computers, you have to attempt to 'load' a nonexistent program, and then watch the result of the search. Even with an accurate cataloguing facility, it is still possible to record over existing programs by accident. As far as I am aware, there is no tape system on any computer which can protect the user from such errors.

There may be tape systems which can protect the unwary from forgetting to press the RECORD button, but I do not know of any. I do not count a reminder on screen to PRESS REC AND PLAY as sufficient, given my ineptitude.

With the addition of a printer and other devices the need to organize what is happening becomes more necessary than ever. As the complexity of the set up increases, the problems inherent in having one system to deal with the printer, another system for cassettes, and another for discs (when added), multiply the problems of the smooth organization of work.

## 2.1 The purpose of CP/M

CP/M stands for the 'Control Program for Microcomputers', and is an operating system. An operating system may best be described as an environment for controlling the behaviour of a computer, and (most important) the devices and peripherals which are connected to that computer.

Simple computers tend to have simple operating systems, in which the user has to perform most of the control functions, and only allow simple devices to be connected. Sometimes it is not clear where the operating system starts and BASIC finishes - or if there is any real operating system. Such systems tend to have special commands and procedures for each new form of device, and there is no coherent method of dealing with devices.

Operating systems for simple computers tend to grow from the bottom upwards, and there is no real coherent philosophy of operation. CP/M is designed from the top downwards, in that CP/M is intended to provide an operating system for a wide variety of computers, and not just an ad hoc system for one particular type of computer. The CPC computers with CP/M are thus part of a large family of serious computers with access to vast resources of software.

CP/M is a system designed to control five main aspects of any sophisticated system:

1. The use of discs and disc files, controlled by the disc operating system.

2. The input from the keyboard, and the output of information on a display (normally a video display with recent computers), the CONSOLE function.

3. The input of information from some device connected to the computer, the READER function.

4. The output of information to some device connected to the computer, the PUNCH function.

5. The output of information to a listing device (eg a printer) connected to a computer, the LIST function.

The CP/M BDOS (Basic Disc Operating System) controls all the standard features of CP/M, that is, those parts of CP/M which do not have to be altered to suit any specific type of computer ('Basic' should not be confused with the BASIC programming language). Those parts of CP/M which have to be altered to fit a specific computer (the actual type of disc drive, the actual type of listing device, for example) are under the control of the BIOS (Basic Input/Output System), the CPC BIOS is specially written by Amsoft to fit the CPC configuration.

The ways in which the above aspects are implemented for CPC CP/M are:

1. The disc drives are Hitachi 3″ drives, which are driven by an NEC μPD765A disc controller chip. The discs are single sided soft sectored, though reversible, with a formatted capacity of 180K, of which 9K is taken up by CP/M information (in normal use).

2. The CONSOLE is normally set to be the keyboard and screen.

3. The READER is normally set to be the special CPC I/O device 0.

4. The PUNCH is normally set to be the special CPC I/O device 0.

5. The LIST is normally set to the printer connected to the Centronics port. and because CP/M is a sophisticated system, it is possible to alter the standard assignments of CP/M logical device functions to actual physical devices. It is very important to realize that there is a crucial distinction between the names used in the construction of the system (such as LIST) and the physical devices (such as a printer) that embody the actual ways in which the system communicates with the outside world.

## 2.2 Logical and actual devices

The CONSOLE READER PUNCH and LIST functions are effectively logical names only, and they bear no real relationship to the actual (physical) devices connected to the computer. The logical devices behave like variables in algebra, in that they are tokens which can stand for many different values (or many actual devices) within defined limits. The actual values which the logical devices may represent (or to which they may be 'assigned') are restricted to the types of physical device which conform to the theoretical behaviour of the logical devices (for example, it is impossible to read from a printer).

The names of the logical devices are frequently shortened to a standard three letter mnemomic (ending with :) which can be used in an almost algebraic form by certain CP/M programs. The algebraic logical device names are CON: RDR: PUN: LST: and though the labels correspond to names such as CONSOLE READER PUNCH LIST, it is important to realize that READER itself (say) is only a short form for 'the logical device name which corresponds to the input of information from some device connected to the computer'.

The names of the logical devices correspond to the actual devices connected to the Intel MDS 800 system on which CP/M was first implemented. The MDS 800 had a console, paper tape reader, paper tape punch, and a line printer, and these reflect the names CON: RDR: PUN: and LST:.

The MDS 800 used an Intel 8080 processor, so that is why CP/M still uses 8080 machine code for distribution, and provides 8080 machine code tools. The physical names used in standard CP/M also correspond to devices popular in an earlier era of microcomputing.

To begin to understand how CP/M on the CPC works, or how any other sophisticated operating system functions, it is worth studying how the system attempts to organize the various devices to which the computer may be connected. There are two aspects to this organization: one, the manner in which CP/M designates devices and, second, what these designations mean as far as the CPC implementation is concerned.

The standard names for physical devices in CP/M, the shortened form of the name, and the way in which the device is implemented in CPC CP/M do not always agree.

## Batch processing device (BAT:)

Is equivalent to input from RDR: and output to LST: in CPC CP/M (can be used by CON:).

## Cathode ray tube (CRT:)

In CPC CP/M this device is taken to be the keyboard and screen, and is the usual actual device to which the logical device CON: is assigned. If LST: is assigned to the CRT:, only the screen is relevant (it is not possible to input from a listing device).

## Line printer device (LPT:)

Refers to the Centronics (parallel) printer port, which itself can be connected to a variety of printers (is the usual device to which LST: is assigned).

## Paper tape punch (PTP:)

This is the 'device' known as the null output device in CPC CP/M which means that, in testing software, if output is sent to the PTP: then nothing happens - later the software can be modified to output to a real device (can be used by PUN:).

## Paper tape reader (PTR)

In CPC CP/M sends an end of file input message (can be used by RDR:).

## Teletype device (TTY:)

CPC has two special input/output devices, and TTY: refers to the first of the two devices, that is, special I/O device 0 (can be used by CON: LST: and is the default for RDR: PUN:).

## User-defined console (UC1:)

In CPC CP/M there are two special input/output devices, and UC1: refers to the second of the two devices, that is, special I/O device 1 (can be used by CON:).

## User-defined list device 1 (UL1:)

The CPC has two special CP/M input/output devices, and UL1: refers to the second of the two devices, that is, special I/O device 1 (can be used by LST:).

## User-defined punch 1 (UP1:)

CPC CP/M distinguishes between two special input/output devices, and UP1: refers to the second of the two devices, that is, special I/O device 1 (can be used by PUN:).

## User-defined punch 2 (UP2:)

In CPC CP/M, UP2: stands for the screen when considered by itself, that is, as distinct from the keyboard (can be used by PUN:).

## User-defined reader 1 (UR1:)

The CPC has two special input/output devices designated for use in CP/M, and UR1: refers to the second of the two devices, that is, special I/O device 1 (can be used by RDR:).

## User-defined reader 2 (UR2:)

This device can be used by RDR:, and in CPC CP/M stands for the keyboard considered as a separate entity.

Though there are potentially sixteen differently named actual devices in CP/M (and four logical devices), in CPC CP/M the number of actual devices implemented is rather less (and the same is true of most versions of CP/M). There are 5 different actual devices in CPC CP/M, not including the null and end of file 'devices'.

| | |
|---|---|
| Centronics port | LPT: |
| I/O device 0 | TTY: |
| I/O device 1 | UC1: UL1: UP1: UR1: |
| Keyboard | CRT: UR2: |
| Screen | CRT: UP2: |

With the standard CPC defaults, only special I/O device 1 is unused, but it is difficult to arrange for two special devices for input and output to be assigned at the same time, with the screen, keyboard, and printer devices also in concurrent use.

For reference (though they are repeated in the Inventory) here are the possible assignments of logical devices to physical devices, with the standard assignment emphasized:

| | |
|---|---|
| CON: | TTY: **CRT:** BAT: UC1: |
| RDR: | **TTY:** PTR: UR1: UR2: |
| PUN: | **TTY:** PTP: UP1: UP2: |
| LST: | TTY: CRT: **LPT:** UL1: |

CP/M is basically a system for organizing the use of devices, and the most important device is the disc drive. There has been no real mention of discs and disc drives so far, because it is worth emphasizing that CP/M is concerned with the total system, and is not just another way of organizing disc files. In the next chapter we will begin to see how the disc handling in CPC CP/M fits in with the system organization aspects.

# Chapter 3
# Discs and drives

**DO NOT SWITCH ON THE COMPUTER.**

**DO NOT PUT A DISC IN THE DRIVE.**

As I noted in the previous chapter, CP/M tries to eliminate as many as possible of the tedious tasks in programming, but even so the user still has to perform certain actions. Until a disc operating system is available with robot control, you will still have to change your own discs, or switch on the printer.

One of the biggest problems with CP/M is you (and I include 'me' in with 'you'). Humans are not as well organized as CP/M, and humans do silly things such as leaving discs in drives when they switch off, and then when they switch on the disc becomes corrupted (it has happened to me).

We will start, therefore, with one disc, that which comes with CPC CP/M.

## 3.1 Protecting a disc

When you buy a blank audio cassette, it is possible to record on the cassette, and then to listen to your recording. It is also possible to record over an earlier attempt. To stop inadvertent rerecording, which would erase some treasured item, you can knock out the recording tabs so that a cassette recorder cannot record on that side of the cassette.

The disc you get from Amsoft to accompany the drive has similar 'write protect' tabs, to stop you from erasing the information saved on the disc. The disc has two sides (like a cassette) and a write protect tab on each side. Close both tabs, but do not put the disc in the drive.

If you have a CPC 464, switch on the disc drive, and then switch on the computer; if you have a CPC 664, switch on the computer. Only when the computer is switched on should you insert the disc. Insert the disc, with the side which says CP/M uppermost.

You will now be in BASIC, and to enter CP/M you have to 'boot' the system, which means that you have to start up CP/M. To boot CP/M you enter a bar command, that is,

```
Ready
|CPM
```

(Even though commands are usually entered in lower case I use upper case for emphasis).

The screen changes colour, the drive operates, and the heading reads

```
CP/M 2.2 - Amstrad Consumer Electronics plc

A>
```

to indicate that you are now at command level in CP/M version 2.2 as modified for the CPC computers by Amstrad. The **A>** symbol is a prompt, which tells you that CP/M assumes that all commands are intended for disc drive A:, unless otherwise specified. At this stage CP/M assumes drive A: whether one drive is connected or two.

If by chance the screen gives a welcome to Amstrad LOGO V1.1, then the disc is upside down. Hit **[CTRL] [SHIFT]** and **[ESC]** to return to BASIC, turn over the disc, and repeat the exercise of booting CP/M.

Check the information available on the CP/M system disc by issuing the command

```
A>DIR
```

to be given a list (a directory) of all the files on that disc. Turn over the disc, and enter

```
A>DIR
```

to discover the files on the Dr LOGO side of the disc. Remove the disc from the drive, and turn it over.

Issue the directory command again, and study the list of disc files (for that is what they are). Two of the members of the list are

```
COPYDISC COM
DISCCOPY COM
```

and these are special files. The primary filenames are **COPYDISC** and **DISCCOPY**, and both have the same filetype **COM**, which means that **COPYDISC.COM** and **DISCCOPY.COM** are files that contain program instructions (they are executable files).

The purpose of the programs contained in these files is to copy the content of one disc onto another disc: the **DISCCOPY** program assumes that only one drive is available; the **COPYDISC** program assumes that two drives are available. The most important thing, however, is to make the copy: until the copy is made, we are at the mercy of our mistakes. Even though we have protected the disc from accidental erasure (by use of the write protect tabs), there is still the possibility of the disc being corrupted.

It is for this reason that we will not examine the detail of the commands at the moment, but concentrate on making the copy (using the simplest and most generally available set up, that for one disc).

## 3.2 Making a DISCCOPY

Insert the CP/M system disc, and enter

```
A>DISCCOPY
```

to which the response of CP/M is

```
DISCCOPY V2.Ø
Please insert source disc into drive A
then press any key:_
```

The source disc is the CP/M disc that is already in drive A, so we just press any key, and read that copying has started and that tracks 0 to 7 are being read. When track 7 has been read, the system requests

```
Please insert destination disc into Drive A
then press any key:_
```

which means that the disc onto which you are to make the copy should now be inserted in the same drive (after removing the original disc). If the new disc is blank, then you will be told that the system is

```
Formatting whilst copying
```

and then writing tracks 0 to 7. When the eight tracks have been copied over you are asked to replace the source disc, and the system continues to repeat the sequence (eight tracks at a time) until the last track (39) has been copied to the new disc. You are told that copying is complete, and asked if you want to make another copy you answer **[N]**. (This allows a test of the copied disc.)

```
Please insert a CP/M system disc into drive A
then press any key_
```

Your copy is now a CP/M system disc, and thus it need not be removed, and if you ask for a directory you will discover that the content is the same as the original disc.

As the content is the same, you can load the `DISCCOPY` program from the new disc. The task is now to copy the Dr LOGO disc (the other side of the CP/M system disc) onto the reverse of your new copy of CP/M. You enter

```
A>DISCCOPY
```

and when asked for the source disc to be inserted in drive A you put in the Dr LOGO disc, and copy it onto the other side of the new CP/M disc. When the copying is complete, and you are asked to insert a CP/M system disc, you can leave the Dr LOGO copy in the drive, because the Dr LOGO disc has the CP/M system tracks (though none of the CP/M system programs).

Now we have copies of both the CP/M system programs and the Dr LOGO program, we can investigate CP/M with somewhat more security. At this point it might be worth using the **DISCCHK** to see if the two discs are identical, but the use of **DISCCHK** (one drive) and **CHKDISC** (two drives) is so simple that I will not give any details other than the technical specifications later in this chapter.

As soon as you have finished checking, put the original disc somewhere safe, well away from the computer, so that you do not get tempted to use the disc because 'it happens to be handy'.

The full details on the use of **COPYDISC/DISCCOPY**, and **CHKDISC/DISCCHK**, are given here, and in the next chapter we will examine what happens when CP/M is initialized (that is, 'booted'). If any of the terms are unfamiliar, check with the Inventory.

# 3.3 COPYDISC

**COPYDISC** is an Amstrad utility transient program located in a file **COPYDISC.COM**, which copies the content of a disc in drive A to a disc in drive B. The memory needed for the program is 39K (that is, the TPA has to be a minimum of 39K).

The program is activated by entering the command line **COPYDISC**, and - if the file **COPYDISC.COM** is on the current disc - the user is asked to

```
Please insert source disc into drive A and destination
disc into drive B then press any key:_
```

The original disc is placed in drive A and the disc to take the copy is placed in drive B. The format of the original disc is checked, and this format is checked against the format of the destination disc. If the formats are not the same, or the destination disc is unformatted, the destination disc will be formatted at the same time as copying occurs.

The copying takes place eight disc tracks at a time, and when the copying is complete, the user is asked

```
Do you want to copy another disc (Y/N):_
```

and when the response is **[N]**, the user is asked

```
Please insert a CP/M system disc into drive A
then press any key:_
```

where a CP/M system disc is one which conforms to the System format (that is, with two system tracks).

There are the following messages which may be output by **COPYDISC**

```
Please insert source disc into drive A and destination
disc into drive B then press any key:-
```

- Output at the start of a copy.

```
Please insert a CP/M system disc into drive A
then press any key:_
```

- Output when **COPYDISC** is ready to return to CP/M.

```
You must insert the source disc into drive A
```

- Output when there is no disc placed in drive A, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
You must insert a CP/M system disc into drive A
```

- Output when there is no disc in drive A, or the disc is of an inadmissable format (for example, Vendor or Data Only formats), the user is prompted for the insertion of the appropriate disc.

```
You must insert the destination disc into drive B
```

- Output if there is no disc in drive B, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
The destination disc in drive B must be write-enabled
```

- Output if the disc in drive is write protected, the user is prompted for the insertion of the appropriate disc.

```
Formatting whilst copying
```

- Output if the destination disc is unformatted, or does not have the same format as the source disc.

```
Copying started
```

- Output at the beginning of a disc copy.

```
Copying complete
```

- Output when the disc copy is complete.

```
WARNING: Failed to copy disc correctly
The destination disc should not be used until it
is successfully copied on to
```

- Output if **COPYDISC** is abandoned whilst copying.

```
The source disc has an unknown format
```

- Output if **COPYDISC** does not recognize the format of the source disc. **COPYDISC** is abandoned.

```
Reading track: n
```

- n is the current track number, message is output during the reading of information from the source disc.

```
Writing track: n
```

- n is the current track number, message is output during the writing of information to the destination disc.

```
Failed to read source disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the source disc. **COPYDISC** is abandoned.

```
Failed to write destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when writing to the disc. **COPYDISC** is abandoned.

```
Failed to read destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when reading back from the disc. **COPYDISC** is abandoned.

```
Failed to verify destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when the data reading back from the disc does not agree with the data sent. COPYDISC is abandoned.

```
Failed to format destination disc correctly: track n
```

- n is the current track number. The message is output if COPYDISC fails to format correctly a track on the destination disc. COPYDISC is abandoned.

```
Do you want to copy another disc (Y/N):_
```

- Output at the end of a successful copy to allow the user to copy another disc or return to CP/M.

```
↑C...aborted
```

- Output if the user types **[CTRL]C** when COPYDISC is waiting for user input. COPYDISC is abandoned.

```
Illegal message number: program aborted
```

- Should not be output, as it indicates an error in the execution of COPYDISC.

```
Insufficient space in TPA
```

- There is not enough room for the COPYDISC program, and the buffers in which it stores information. Should not normally be produced, but might possibly be due to an inadvertant use of MOVCPM. Reboot CP/M with a standard system disc, which has a TPA of greater than 39K.

# 3.4 DISCCOPY

DISCCOPY is an Amstrad utility transient program located in a file DISCCOPY.COM, which copies the content of a disc in drive A to another disc which is placed alternately into drive A. The memory needed for the program is 39K (that is, the TPA has to be a minimum of 39K).

The program is activated by entering the command line DISCCOPY, and - if the file DISCCOPY.COM is on the current disc - the user is asked to

```
Please insert source disc into drive A
then press any key:_
```

The original disc is placed in drive A, the format of the source disc is checked, and later this format is checked against the format of the destination disc. If the formats are not the same, or the destination disc is unformatted, the destination disc will be formatted at the same time as copying occurs.

The copying from the source disc takes place eight disc tracks at a time, and when the copying of the eight tracks is over, the user is told

```
Please insert destination disc into drive A
then press any key:_
```

and the user has to put the correct disc into drive A. The format of this disc is checked against that of the source disc, and if they differ the destination is formatted in accordance with the source format. When the copy of the disc is complete, the user is asked

```
Do you want to copy another disc (Y/N):_
```

and when the response is **[N]**, the user is asked

```
Please insert a CP/M system disc into drive A
then press any key:_
```

where a CP/M system disc is one which conforms to the System format (that is, with two system tracks).

```
Please insert source disc into drive A
then press any key:_
```

- Output at the start of each copying of eight tracks from the source disc.

```
Please insert destination disc into drive A
then press any key:_
```

- Output at the start of each copying of eight tracks to the destination disc.

```
Please insert a CP/M system disc into drive A
then press any key:_
```

- Output when **DISCCOPY** is ready to return to CP/M.

```
You must insert the source disc into drive A
```

- Output when there is no disc placed in drive A, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
You must insert a CP/M system disc into drive A
```

- Output when there is no disc in drive A, or the disc is of an inadmissable format (for example, Vendor or Data Only formats), the user is prompted for the insertion of the appropriate disc.

```
You must insert the destination disc into drive A
```

- Output if there is no disc in drive A, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
The destination disc in drive A must be write-enabled
```

- Output if the disc in drive is write protected, the user is prompted for the insertion of the appropriate disc.

```
Formatting whilst copying
```

- Output if the destination disc is unformatted, or does not have the same format as the source disc.

```
Copying started
```

- Output at the beginning of a disc copy.

```
Copying complete
```

- Output when the disc copy is complete.

```
WARNING: Failed to copy disc correctly
The destination disc should not be used until it is
successfully copied on to
```

Output if **DISCCOPY** is abandoned whilst copying.

```
The source disc has an unknown format
```

- Output if **DISCCOPY** does not recognize the format of the source disc. **DISCCOPY** is abandoned.

```
Reading track: n
```

- n is the current track number, message is output during the reading of information from the source disc.

```
Writing track: n
```

- n is the current track number, message is output during the writing of information to the destination disc.

```
Failed to read source disc correctly: track n sector m
```

- n and m are the track and sector numbers of a bad sector on the source disc. DISCCOPY is abandoned.

```
Failed to write destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when writing to the disc. DISCCOPY is abandoned.

```
Failed to read destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when reading back from the disc. DISCCOPY is abandoned.

```
Failed to verify destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when the data reading back from the disc does not agree with the data sent. DISCCOPY is abandoned.

```
Failed to format destination disc correctly: track n
```

- n is the current track number. The message is output if DISCCOPY fails to format correctly a track on the destination disc. DISCCOPY is abandoned.

```
Do you want to copy another disc (Y/N):_
```

- Output at the end of a successful copy to allow the user to copy another disc or return to CP/M.

```
↑C...aborted
```

- Output if the user types **[CTRL]C** when DISCCOPY is waiting for user input. DISCCOPY is abandoned.

```
Illegal message number: program aborted
```

- Should not be output, as it indicates an error in the execution of **DISCCOPY**.

```
Insufficient space in TPA
```

- There is not enough room for the **DISCCOPY** program, and the buffers in which it stores information. Should not normally be produced, but might possibly be due to an inadvertant use of **MOVCPM**. Reboot CP/M with a standard system disc, which has a TPA of greater than 39K.

# 3.5 CHKDISC

**CHKDISC** is an Amstrad utility transient program located in a file **CHKDISC.COM**, which checks the content of a disc in drive A to a disc in drive B. The memory needed for the program is 39K (that is, the TPA has to be a minimum of 39K).

When the program is executed, the system asks

```
Please insert source disc into drive A and destination
disc into drive B then press any key:_
```

The user places the original disc into drive A and the new copy into drive B, and then presses any key. If the discs are of differing format, the system informs the user that the discs are of different formats and abandons the program.

The two discs are compared eight tracks at a time, starting at track 0. Each block of eight tracks is read from the source disc then compared with the data on the destination disc. If there is a mismatch, then the mismatch is signalled, otherwise when the checking is successful the user is asked if another check is to be made. Finally there is a request to insert a CP/M system disc.

There are the following messages which may be output by **CHKDISC**

```
Please insert source disc into drive A and destination
disc into drive B then press any key:_
```

- Output at the start of a check.

```
Please insert a CP/M system disc into drive A
then press any key:_
```

- Output when **CHKDISC** is ready to return to CP/M.

```
You must insert the source disc into drive A
```

- Output when there is no disc placed in drive A, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
You must insert a CP/M system disc into drive A
```

- Output when there is no disc in drive A, or the disc is of an inadmissable format (for example, Vendor or Data Only formats), the user is prompted for the insertion of the appropriate disc.

```
You must insert the destination disc into drive B
```

- Output if there is no disc in drive B, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
Copy checking started
```

- Output at the beginning of a disc check.

```
Copy checking complete
```

- Output when the disc check is complete.

```
WARNING: Failed to compare disc correctly
```

- Output if CHKDISC is abandoned whilst checking.

```
The source disc has an unknown format
```

- Output if CHKDISC does not recognize the format of the source disc. CHKDISC is abandoned.

```
Source and destination discs have different formats
```

- Output if the formats differ. CHKDISC is abandoned.

```
Reading track: n
```

- n is the current track number, message is output during the reading of information from the source disc.

```
Checking track: n
```

- n is the current track number, message is output during the checking of information on the destination disc.

```
Failed to read source disc correctly: track n sector m
```

- n and m are the track and sector numbers of a bad sector on the source disc. CHKDISC is abandoned.

```
Failed to read destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when reading back from the disc. CHKDISC is abandoned.

```
Failed to verify destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when the data reading back from the disc does not agree with the data sent from the source disc. CHKDISC is abandoned.

```
Do you want to check another disc (Y/N):_
```

- Output at the end of a successful check to allow the user to check another disc or return to CP/M.

```
↑C...aborted
```

- Output if the user types **[CTRL]C** when CHKDISC is waiting for user input. CHKDISC is abandoned.

```
Illegal message number: program aborted
```

- Should not be output, as it indicates an error in the execution of CHKDISC.

```
Insufficient space in TPA
```

- There is not enough room for the CHKDISC program, and the buffers in which it stores information. Should not normally be produced, but might possibly be due to an inadvertant use of MOVCPM. Reboot CP/M with a standard system disc, which has a TPA of greater than 39K.

# 3.6 DISCCHK

DISCCHK is an Amstrad utility transient program located in a file DISCCHK.COM, which checks the content of a disc in drive A to a disc alternately in drive B. The memory needed for the program is 39K (that is, the TPA has to be a minimum of 39K).

When the program is executed, the system asks

```
Please insert source disc into drive A
then press any key:_
```

The user places the original disc into drive A, and then presses any key. If the discs are of differing format, the system informs the user that the discs are of different formats and abandons the program.

The two discs are compared eight tracks at a time, starting at track 0. Each block of eight tracks is read from the source disc, which is then removed from the drive. The destination disc is then inserted and compared with the data on the source disc. If there is a mismatch, then the mismatch is signalled, otherwise when the checking is successful the user is asked if another check is to be made. Finally there is a request to insert a CP/M system disc.

There are the following messages which may be output by **DISCCHK**

```
Please insert source disc into drive A
then press any key:_
```

- Output at the start of a check, and after every check of eight sectors.

```
Please insert destination disc into drive A
then press any key:_
```

- Output when eight sectors on the source disc (stored in memory) are to be compared with the corresponding eight sectors on the destination disc.

```
Please insert a CP/M system disc into drive A
then press any key:_
```

- Output when **DISCCHK** is ready to return to CP/M.

```
You must insert the source disc into drive A
```

- Output when there is no disc placed in drive A, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
You must insert a CP/M system disc into drive A
```

- Output when there is no disc in drive A, or the disc is of an inadmissable format (for example, Vendor or Data Only formats), the user is prompted for the insertion of the appropriate disc.

```
You must insert the destination disc into drive A
```

- Output if there is no disc in drive A, when one is expected, the user is prompted for the insertion of the appropriate disc.

```
Copy checking started
```

- Output at the beginning of a disc check.

```
Copy checking complete
```

- Output when the disc check is complete.

```
WARNING: Failed to compare disc correctly
```

- Output if **DISCCHK** is abandoned whilst checking.

```
The source disc has an unknown format
```

- Output if **DISCCHK** does not recognize the format of the source disc. **DISCCHK** is abandoned.

```
Source and destination discs have different formats
```

- Output if the formats differ. **DISCCHK** is abandoned.

```
Reading track: n
```

- n is the current track number, message is output during the reading of information from the source disc.

```
Checking track: n
```

- n is the current track number, message is output during the checking of information on the destination disc.

```
Failed to read source disc correctly: track n sector m
```

- n and m are the track and sector numbers of a bad sector on the source disc. **DISCCHK** is abandoned.

```
Failed to read destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when reading back from the disc. **DISCCHK** is abandoned.

```
Failed to verify destination disc correctly:
track n sector m
```

- n and m are the track and sector numbers of a bad sector on the destination disc, discovered when the data reading back from the disc does not agree with the data sent from the source disc. **DISCCHK** is abandoned.

```
Do you want to check another disc (Y/N):_
```

- Output at the end of a successful check to allow the user to check another disc or return to CP/M.

```
↑C...aborted
```

- Output if the user types **[CTRL]C** when **DISCCHK** is waiting for user input. **DISCCHK** is abandoned.

```
Illegal message number: program aborted
```

- Should not be output, as it indicates an error in the execution of **DISCCHK**.

```
Insufficient space in TPA
```

- There is not enough room for the **DISCCHK** program, and the buffers in which it stores information. Should not normally be produced, but might possibly be due to an inadvertent use of **MOVCPM**. Reboot CP/M with a standard system disc, which has a TPA of greater than 39K.

# Chapter 4
# The parts of CP/M

The two programs we have encountered thus far (DISCCOPY/COPYDISC and DISCCHK/CHKDISC) were, as we saw, stored as programs on the CP/M disc, with a common filetype (.COM). The filetype enables CP/M and us to distinguish between various types of file, where each filetype has a distinct purpose and a distinct action.

Take your copy of the system master, and enter

```
A>DIR *.COM
```

and you will be presented with a list of files, all of the same type: that is, all the files listed have a .COM filetype (though note that the 'dot' of the dot-COM files is not shown). The asterisk (*) is a wildcard, that is, any file which matches the .COM filetype is listed in the directory. This is the means by which all the executable program files can be isolated.

If we want to isolate all files with the same primary file name, the filetype can be turned into a wildcard. For example

```
A>DIR DUMP.*
A: DUMP     COM : DUMP     ASM
A>
```

will list all files which have the primary file name DUMP. The asterisk is a global wildcard, whereas the query (?) is rather more specific.

```
A>DIR DISC????.*
A: DISCCOPY COM : DISCCHK COM
A>
```

Now try to enter

```
A>DIR DIR?????.*
NO FILE
A>
```

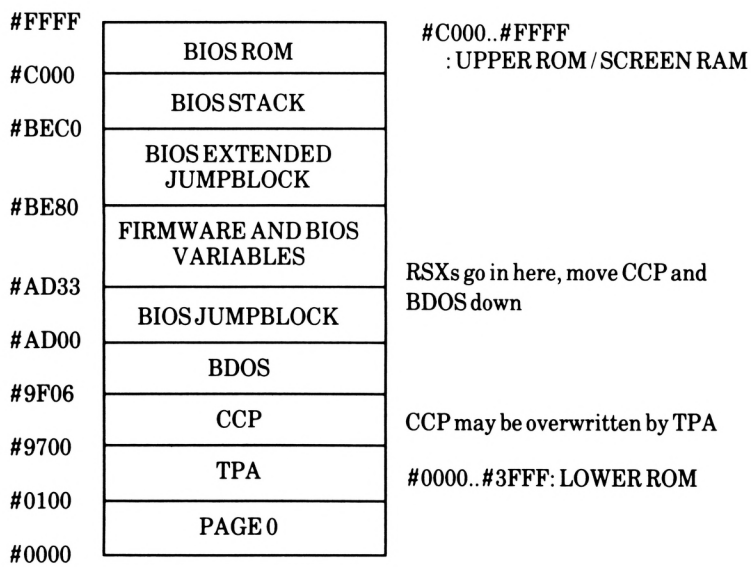
and thus there is no file which looks remotely like DIR, and a check of the complete directory will confirm this discovery. DIR acts very much like a .COM (executable) file, but it is not a file of any form. DIR is a CP/M built-in command, and when we begin to investigate built-in commands we can start to understand more about CP/M.

# 4.1 Built-in commands and the CCP

Start with the initialization of CP/M, or as it is usually termed 'booting' CP/M. We have already noted the distinction between the BDOS and the BIOS when examining devices, and the distinction is relevant here. We noted that the BDOS contained those parts of CP/M which were fixed, and did not vary from system to system, and the BIOS was that aspect of CP/M which was modified to accomodate different computers.

There are certain CP/M comands, such as `DIR`, which are built into the CP/M BDOS, and these are effectively machine code routines similar to the standard commands used in BASIC. These routines are not on the disc, because they are built-in to the CP/M BDOS. The nature of these commands is such that they are always stored in a constant place in memory, but to access these routines requires some system to decide what to do with the information you enter. The least such a system needs to perform is the decision about whether `DIR` is a built-in command, or not, and if not built-in how to find if it is a `.COM` file on the current disc.

Such decisions are the perogative of the CCP (the console command processor), which is the third aspect to CP/M (after the BDOS and BIOS). All these various parts of CP/M have to be stored in memory (there is nowhere else to go), and they are stored with the BIOS and associated elements in high memory (including part of the BIOS in ROM), followed by the BDOS, then the CCP, and finally the TPA (transient program area) which always starts at location 256 in denary (or #0100).



The CCP, therefore, is the part of CP/M which decides what is to be done with your output, and the TPA (the fourth aspect) is that portion of memory into which files are loaded. A long program will extend over the lower boundary of the CCP, and so the CCP can be erased by a large program. Such a large program is `DISCCOPY`, and at the end of the program there is no CCP left to control your input. At the end of the program, therefore, you have to reload in part of the CP/M system from a CP/M disc, to reconstitute the CCP and the BDOS. This is called a warm boot, and a warm boot can always be activated at command level by issuing a **[CTRL]C** instruction.

A warm boot only loads in the BDOS and CCP because no program can, or should, overwrite the BIOS: if the BIOS is modified then the system as a whole cannot function. Input and output, for example, are impossible if changed in an unpredictable manner, if portions of the BIOS are corrupted. It is possible to change the size of the TPA (which includes the CCP), by `MOVCPM` but its use will be left to end of the next but one chapter. A cold boot resets the whole system.

The built-in commands, BDOS, and the CCP are thus one element, and when the size of the TPA is altered for some reason (possibly to incorporate extra RSXs in CP/M or machine code routines in LOGO), the BDOS and CCP are moved together.

## 4.2 The built-in commands

The built-in commands are given below, where `fs` represents a file specification, and `wfs` represents a file specification which allows wildcards.

```
DIR wfs
```

- Produces a list of specified files. If no file is specified, all the files in the current user area are listed (normally this means the entire disc).

```
ERA wfs
```

- Erases specified files. To erase all files use `ERA *.*`, though CP/M will ask you to confirm this action.

```
REN fs2=fs1
```

- Renames specified file 1 to a specified file 2. If the file to be renamed is on the drive which is not currently logged, then the drive designation is used for the second file specification alone. A drive designation for the first file specification is illegal.

```
SAVE n fs
```

- n is a number which indicates how many pages (256 byte sections) of memory should be saved from the TPA onto disc, with the specified file name. SAVE is often used in conjunction with the executable program DDT to load in a program, change the USER area, and then SAVE a program. As it is not possible to access files without the USER logical area, this process is the only means by which such transfers can be made. DDT gives an indication of the size of a program which can be used by SAVE.

```
TYPE fs
```

- Displays the content of the specified (ASCII) file on the the CONSOLE (CON:) device. If the specified file is not ASCII, strange effects are likely, and the system may crash.

```
USER n
```

- n is an integer from 0 to 15, and specifies a logical area on the disc. Only those files which are contained in that logical area can be used, though the built-in commands are always accessible.

A file specification is in three parts: a drive designation (that is, A: or B: or nothing for the logged on disc), followed immediately by the primary file name, with the filetype separated by a fullstop. A wildcard file specification is also in three parts: the drive designation as before, with the primary file name and /or filetype incorporating wildcards. The only slight modification to this is the REN command, which only allows the drive designation in the second file specification.

To rename a file TEXT.TXT to NEW.TXT on drive B:, where the current drive is A:, therefore, we enter

```
A>REN B:NEW.TXT=TEXT.TXT
A>
```

To enter the slightly different

```
A>REN NEW.TXT=B:TEXT.TXT
A:TEXT.TXT?

A>
```

indicates that the command does not treat the first file specification as a three part specification, because it is confused by the drive designation.

# 4.3 Booting CP/M

When in BASIC, you enter `|CPM` to start a process which loads the CP/M system, or you 'cold boot' CP/M. Return, therefore, to BASIC by issuing the command

```
A>AMSDOS
```

then take the disc out of the drive, and enter

```
Ready
|CPM
```

which sets the drive a-whirring. You will be told

```
Drive A: disc missing

Retry, Ignore or Cancel? [C]

Failed to load boot sector

Retry, Ignore or Cancel? [C]

Drive A: disc missing

Retry, Ignore or Cancel? [C]
```

and so you can continue. Replace the disc, and all will be well.

## What is the 'boot sector'?

Every disc used on the CPC has 40 tracks, where the tracks are concentric rings of magnetic information. In one sense, think of the 3″ disc in the drive as being equivalent to forty very high speed cassette recorders, where each cassette recorder uses an endless loop of tape. The forty recorders are called 'tracks', and are numbered from 0 to 39. Each recorder is distinguished by that number, and so to refer to the third recorder the system uses the designation 'track 2'. (We have already encountered the tracks when copying or checking a disc.)

To read in information from the recorders, we first need to know the name of the file, and once this is established, CP/M looks at a copy of the directory of files. The copy is stored in memory. The directory tells CP/M where the file starts, and how long it is, so information is read off the appropriate track. Each track is divided into portions called 'sectors', and (as the loop is endless) each sector on the track has to have some identification so that we know when to start reading information.

If a file is greater than a certain size, then it will extend over more than one track or (in our analogy) one recorder. The system knows that when it reaches the end of one track it has to change to the next track (read from the next recorder), and thus, every so often, there is a change from one track to the next.

The way in which the sectors within the tracks are numbered depends on the nature of the organization of the tracks on a particular disc, that is, the 'format' of the disc. For normal CP/M, the sector numbers for each track start at sector 65 (#41) and extend to sector 73 (#49). If you were to write machine code routines to directly access these sectors you would use the hexadecimal form (#41..#49), but if an error is reported in `DISCCOPY`, say, the error report gives the sector number in decimal (65..73).

The 'boot' sector, which was unable to be loaded (as the disc was not in the drive), is track 0 sector 65 (#41). When the `|CPM` command is issued from BASIC, this activates certain machine code instructions permanently stored in ROM. The machine code instructions include certain primitive CP/M utilities such as the disc missing warning, and also directions to load from disc certain information concerning the initial set up of the CP/M system. This information comes from the boot sector.

The boot sector then loads the next sector from track 0 (sector 66, #42). This is known as the configuration sector, and is responsible for the initialization of various system parameters for the BIOS. Once the initialization is complete, a warm boot occurs and the CCP and BDOS are loaded from track 0 sectors 72 and 73 (#48 and #49), and track 1 sectors 65..73 (·41..·49). Every time a warm boot is activated these sectors are loaded.

This is the standard CP/M system format (see the Inventory for more details), and to produce a disc according to this format, you use the `FORMAT` program. To change the content of the configuration sector, you use `SETUP`, and to alter the sector concerned with the CCP and BDOS you use `SYSGEN`. The `FORMAT` and `SETUP` programs are discussed in this chapter, and `SYSGEN` in the next chapter but one.

# 4.4 FORMAT

The `FORMAT` program is held in a file named `FORMAT.COM`, and is activated by either of

```
A>FORMAT

A>FORMAT S
```

to produce an ordinary CP/M system disc, and by

```
A>FORMAT V

A>FORMAT D

A>FORMAT I
```

for the Vendor, Data only, and IBM formats (see Inventory). Usually you will only ever need

```
A>FORMAT
```

One point to remember when using **FORMAT** is that the program itself has to reside on a system disc (that is, a disc with the above arrangement of sectors on tracks 0 and 1). The program goes

```
A>FORMAT

FORMAT V2.Ø

Please insert disc to be formatted into drive A
then press any key:_
```

and once the formatting is successfully complete, and the usual request concerning repetition is made, and finally a system disc has to be inserted into drive A. If the disc which has just been formatted has been processed normally, it will be a system disc itself so just press a key.

There are several messages which may be produced by **FORMAT**:

```
Please insert disc to be formatted into drive A
then press any key:_
```

- Output at the commencement of formatting.

```
Please insert system disc into drive A
then press any key:_
```

- Output when a return is to be made to CP/M, needed so that a warm boot may be performed.

```
You must insert a CP/M disc into drive A
```

- Output if there is no disc in drive A, when a disc is expected. The user is reprompted to insert a system disc.

```
You must insert the disc to be formatted into drive A
```

- Output when the program expects to find a disc in drive A. Reprompts for a disc.

```
The disc to be formatted in drive A
must be write-enabled
```

- Output if the disc in drive A is write-protected. Reprompts for a disc.

```
Formatting started
```

- Output when formatting starts.

```
Formatting complete
```

- Output when formatting is complete.

```
WARNING: Failed to format destination disc correctly
The destination disc should not be used until it is
successfully formatted
```

- Output when **FORMAT** is abandoned whilst formatting.

```
The disc in drive A is not a CP/M system disc
```

- Output when CP/M tries to read the two reserved tracks from a non CP/M system disc. **FORMAT** is abandoned. Try again with a CP/M system disc.

```
Formatting track: n
```

- n is the current track number. Output when **FORMAT** is in operation of formatting.

```
Failed to read source disc correctly: track n sector m
```

- n is the track number, and m the sector number of the bad sector. Output if **FORMAT** fails to read a sector from the reserved system tracks correctly. **FORMAT** is abandoned.

```
Failed to read destination disc correctly:
track n sector m
```

- n is the track number, and m the sector number of the bad sector. Output if **FORMAT** fails to read a sector from the destination disc correctly. **FORMAT** is abandoned.

```
Failed to format destination disc correctly: track n
```

- n is the current track number. Output if FORMAT fails to format a track on the destination disc correctly. FORMAT is abandoned.

```
Do you want to format another disc (Y/N)
```

- Output at the end of a disc format, to allow user to format another disc.

```
↑C...aborted
```

- Output if user types **[CTRL]C** when FORMAT is waiting for user input. FORMAT is abandoned.

```
Illegal message number: Program aborted
```

- Indicates an error in the execution of the FORMAT. Should not normally be produced.

# 4.5 SETUP

The SETUP program allows the user to change certain of the system parameters in the configuration sector. The parameters which may be altered are

1. Initial command buffer.

2. Sign on string (the greeting).

3. Printer power up string.

4. Keyboard translations.

5. Keyboard expansion strings.

6. IOBYTE settings (that is, which actual devices are used for which logical devices).

7. Saving alternate and IY registers (enable/disable).**

8. BIOS message (enable/disable).**

9. Initial command buffer clearing.**

10. Motor on delay.**

11. Motor off delay.**

12. Drive stepping rate.**

13. Z80 SIO channel A baud rate, data bits, parity, and stop bits.

14. Z80 SIO channel B baud rate, data bits, parity, and stop bits.**

Those items marked ** should seldom require changing.

The program is activated by SETUP, and as long as the configuration sector on the current disc has a valid configuration sector, the program will execute correctly. Certain of the SETUP options require text input, and sometimes need the entry of control codes, which is accomplished by the use of the up arrow key ↑ and the corresponding character. For the standard CP/M codes see the Inventory, and for the standard CPC codes see the CPC manual.

The user is taken through each of the possible options one at a time by SETUP, and this is the order in which they will be discussed herein. At the end of SETUP you are asked if you wish to update your system disc (by changing the configuration sector), and then (if you answered in the affirmative) if you want to reboot the system (or restart CP/M).

There are many messages output by SETUP, but most are self explanatory if considered in conjunction with the following account.

## 4.5.1 Initial command buffer

The default is that this buffer is empty. The initial command buffer is used on the LOGO disc in this way

```
LOGO↑M
```

which indicates that a command is to be sent to the system when the Dr LOGO disc is booted. The command is LOGO, followed by a carriage return ( ↑ M), so that the LOGO system is entered automatically. As with all options you are asked if the content is correct, and if it needs modification, then you can enter the modifications. Consider this option to be your way of controlling the user's way into the system.

## 4.5.2 Sign on string

The sign on string for the CP/M system disc is

```
↑\@ww↑\a@@↑]wwCP/M 2.2 - Amstrad Consumer
Electronics plc↑J↑M
```

where, for example, ↑ \ is a control character (FS) which is used to set Ink to a pair of colours. @ (ASCII 0) sets the ink number, and ww (ASCII 119,119) set the pair of colours MOD 32, which sets the colour to number 23, pastel cyan: so the background is set to pastel cyan. The next sequence ↑ \a@@ sets Ink 1 to 0 (black), thus text is in black. ↑ ]ww sets the border to pastel cyan.

The greeting is then printed out on the screen, followed by a line feed ( ↑ J ) and carriage return ( ↑ M ).

### 4.5.3 Printer power up string

This enables the printer (or whatever actual device is used by logical device LST:) to be initialized with a special font, or whatever. Uses the same method of entering information as the sign on string.

### 4.5.4 Keyboard translation table

This is a table, for which the default is a nil table, which has four headings: the key code, which is the key number as given in the CPC manual; the normal value, which is the ASCII code for the desired character; the shift value, which is the ASCII code for the desired character when the shift key is depressed; and the control value, which is the ASCII code for the desired character when the Control key is depressed.

A good example of these translations come with the Dr LOGO system. For example, the **[ESC]** key has code 66 and, in the LOGO configuration, this is assigned to the ASCII value 7. The ASCII value 7 corresponds to **[CTRL]G** which is used in Dr LOGO to stop execution. In this manner, therefore, we have made the operation of **[ESC]** identical to that of **[CTRL]G**.

If translations are to be effected, then the system gives four suboptions: key translations may be added to the table; key translations may be deleted; the translation table may be cleared; and the translations may be finished. The commands are entered in the form

```
A - Add key translation (key number, normal, shift, control)

D - Delete key translation (key number)

C - Clear all translations

F - Finish translations
```

## 4.5.5 Keyboard expansion table

There are thirty two keyboard expansion characters, and the expansion table gives the expansion token (as defined in the CPC manual) and the corresponding expansion string. For example,

| Expansion Token | Expansion string |
|---|---|
| 11 | dir↑M |
| Ø | stat *.*↑M |

will set the small **[ENTER]** key to produce a directory request, and set the Ø key on the numeric pad to produce the statistics of all files on the disc.

## 4.5.6 Setting IOBYTE

The CP/M IOBYTE function is the means by which assignments of logical devices to actual devices can be changed. These assignments are shown by SETUP as

```
Default IO byte settings are:
   CON: is assigned to CRT: (keyboard and VDU)
   RDR: is assigned to TTY: (special IO device Ø)
   PUN: is assigned to TTY: (special IO device Ø)
   LST: is assigned to LPT: (centronics printer)
```

and the assignments are changed by entering, for example,

```
Enter required IO byte setting:_PUN:, LPT:

PUN: may only be assigned to TTY:, PTP:, UP1: or UP2:

Enter required IO byte setting:_
```

and so forth. See Chapter 2 and the Inventory for the extent of possible IOBYTE reassignments.

## 4.5.7 Register saving

This option is not normally a good one to change. The default value is what is termed SLOW mode, and the reason why it is called slow is that every time CP/M accesses the CPC firmware the alternate and IY registers are saved. This is extra security, and takes slightly more time than saving just the main registers and the IX register. The time difference is not crucial, and so it is only in rare circumstances that this option is changed.

### 4.5.8 BIOS messages

Not a good option to change, because it is only in rare circumstances that the ordinary programmer wishes to produce his own interface to CP/M, and without the messages there is the potential for great confusion.

### 4.5.9 Command buffer clear

Not worth changing from the default of clear the buffer, unless you are a CP/M programmer who can then retrieve the command from the buffer.

### 4.5.10 Drive motor on delay

This option is only provided for those who wish to attach their own drive to the CPC.

### 4.5.11 Drive motor off delay

This option is only provided for those who wish to attach their own drive to the CPC.

### 4.5.12 Stepping rate

This option is only provided for those who wish to attach their own drive to the CPC.

### 4.5.13 Serial Channel A Configuration

The default values for the Z80 SIO Channel 0 (otherwise known as the special I/O device 0 are:

9600 transmit baud rate

9600 receive baud rate

8 data bits

NO parity bits

1 stop bits

and these may be altered to fit with specific Modems or other devices. The parameters are changed by answering **[N]** to the question

```
Is this correct (Y/N):_
```

and then entering the appropriate values separated by commas. If any parameter value is incorrect, then the mistake is flagged, and you are prompted to reinput.

The Amstrad serial interface can be addressed as serial Channel A and thus can be configured to the users own requirements. The device is accessed by the logical name `PUN:`, which is, by default connected to Special I/O device O.

### 4.5.14 Serial Channel B Configuration

This device is similar to the other serial device, but in this case both the send and receive rate are the same. As effectively only one serial device can be operative at one time, it is sensible to stick with the more flexible Channel A/device 0.

# Chapter 5
# Files and things

We know that the CP/M program **FORMAT** (say) is stored as the file **FORMAT.COM** on disc, and that **FORMAT** is the primary file name with **.COM** as the filetype. This information is revealed by use of a **DIR** command

```
A>DIR FORMAT.COM
A: FORMAT   COM
A>
```

where **A:** is the drive designation. If we were using drive **B:** as the default (or 'logged') drive, the sequence would be

```
B>DIR A:FORMAT.COM
A: FORMAT   COM
B>
```

In this chapter we will examine the ways in which we can manipulate files, and modify certain aspects of the working system.

## 5.1 Status and statistics

Rather than ask for a directory of all files, enter

```
A>STAT *.*
```

at which a long list of names is displayed on the screen with (at the end of the list) an indication of the number of bytes remaining on **A:**. By using the double * wildcard we have produced a set of statistics for all the files on the disc. Enter

```
A>STAT FORMAT.COM

 Recs  Bytes  Ext Acc
   21     3k    1 R/W A:FORMAT.COM
Bytes Remaining On A: 50k

A>
```

and we are informed that the file **A:FORMAT.COM** is described by the following characteristics:

- The actual information stored in the file is equal to 21 records, where each record is 256 bytes, or 1/8 K bytes.

- The length of the file, as stored on the disc, is 3K because files are stored in units of 1K on the disc. 3K is equal to 24 records and thus 3K is needed to store 21 records of information (2K is equal to 16 records).

- There are certain characteristics of files called 'extents', each of which is equal to 16K, and the Ext for this file is 1 (it is less than 16K in length). The notion of extents need not bother the user, as it is related to the way in which information about the files are stored in FPBs (file parameter blocks).

- The access status of the file is set to R/W, which means that it is possible both to read information from the file, and to change the content of the file (that is, write to the file).

If we enter

```
A>STAT *.DEM

 Recs  Bytes  Ext Acc
  208    26k    2 R/W A:ROINTIME.DEM
Bytes Remaining On A: 50k

A>
```

we can see that this is a very long file, approximately ten times as long as **FORMAT.COM** (208 records compared to 21 records), but with a shorter relative file length (26K compared to 3K), and only twice the extent. Larger files are saved more economically, in that there is less relative wasted space.

When we started, the first thing we did was to close the write protect tabs on the discs, what we would like to be able to do is protect certain files (the **.COM** files at the very least) from accidental alteration. The **STAT** program (itself saved on the file **STAT.COM**) enables us to make that change in Access status. To change access status on all the **.COM** files, therefore, we enter

```
A>STAT *.COM $R/O
```

and there is a long sequence of lines all of which say more or less the same thing: for example,

```
STAT.COM set to R/O
DUMP.COM set to R/O
```

and so forth. A quick check of the status of the **.COM** files by

```
A>STAT *.COM
```

shows (inter alia)

```
41    8k    1 R/O A:STAT.COM
10    2k    1 R/O A:SUBMIT.COM
```

and a simple check of the status of all files (using `STAT *.*`) shows that some files are still `R/W` (all except the `.COM` files). To set the complete disc to read only status, we assign `R/O` to the appropriate drive designation by

```
A>STAT A:=R/O

A>STAT *.*
```

and there does not appear to be any change, but, however,

```
A>ERA ROINTIME.DEM
Bdos Err On A: R/O
A>
```

shows that all files seem to be protected from accidental erasure. You may verify that the file has not been erased by

```
A>DIR ROINTIME.DEM
A: ROINTIME.DEM
A>
```

However, as CP/M was warm booted the R/O status of the disc will have been removed, typing `A>ERA ROINTIME.DEM` a second time will erase the file.

(shortly we will see how to return `ROINTIME.DEM` to its place on your disc).

Another feature of `STAT` worth noting is

```
A>STAT *.COM $SYS
```

The output which is produced by this command line contains

```
STAT.COM set to SYS
DUMP.COM set to SYS
```

and a listing of the status of all files (by `STAT *.*`) shows that

```
41    8k    1 R/O A:(STAT.COM)
10    2k    1 R/O A:(SUBMIT.COM)
```

and

```
A>DIR
```

reveals an absence of .COM files. The $SYS parameter 'hides' files from the directory request, and so can be used to accentuate important programs (which may of course call hidden programs). The files may be revealed by a $DIR parameter to STAT,

```
A>STAT *.COM $DIR
```

which reverses the effect of $SYS.

STAT provides more environmental enquiries about the status of aspects of the system (see later), but before we reinstate ROINTIME.DEM,

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:

A>
```

which are two useful features. The first, VAL:, gives a list of the possible types of assignment and enquiry using STAT, and the second, DEV:, gives the present assignment of logical devices (on the left) to actual devices (on the right). If a printer is attached (an LPT:) then to hit **[CTRL]P** will repeat all that is displayed on the screen onto the printer (hitting **[CTRL]P** again will switch off the printer). If you have no printer attached and hit **[CTRL]P**, CP/M hangs, waiting for the printer to recognize the signals CP/M is sending. Hit **[CTRL]C**, and all is well.
It seems a pity to not be able to use **[CTRL]P**, just because there is no printer, besides which what we are going to do now is somewhat amusing. When assigning logical to physical devices, it is possible to perform more than one assignment on the same line: each assignment is separated from the next assignment by a comma. In this case it is not needed.

```
A>STAT LST:=CRT:

A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is CRT:

A>[CTRL]P

AA>>
```

and whatever you now enter appears double on the screen. For example, if you enter `DIR *.*` then it appears as

```
AA>>DDIIRR **..**
```

and all the output appears as double characters, for example, the file

```
A : DUMP     ASM :
```

appears as

```
AA::  DDUUMMPP          AASSMM  ::
```

The strange output can be stopped, and things returned to normal, by hitting **[CTRL]P** again. The reason why the characters have been repeated is that the output from the listing device (`LST:`) has been assigned to the screen, and so when the information is copied from the screen to the listing device, it copies itself, a character at a time.

Now to copying files and things.

# 5.2 Copying files

The first thing we want to do is resuscitate the `ROINTIME.DEM` file. There are various ways in which this life giving operation can be performed, and the first way is by use of `FILECOPY`. `FILECOPY` is one of three special Amstrad utilities to copy files and `FILECOPY` is designed for users with one drive (the other two special utilities, `CLOAD` and `CSAVE`, are concerned with cassette operation).

If there is more than one drive, `PIP` (a standard CP/M program) is more effective for copying from one disc to another, but `PIP` cannot copy from one user area to another, whereas `FILECOPY` is specially designed for such inter user area copying.

First, to copy ROINTIME.DEM using FILECOPY: the file we want (ROINTIME.DEM) is stored on the original CP/M master (safely stored away), so we have to look it out. As it happens, the file FILECOPY.COM is stored on both the original and the copy, and so, because ROINTIME.DEM is also on the original, place the CP/M original in the drive and enter

```
A>FILECOPY ROINTIME.DEM
FILECOPY V2.1

Please insert SOURCE disc into drive A
then press any key:_
```

at this point, you hit **[ENTER]** and see

```
Copying started.  .  .  .

Please insert DESTINATION disc into drive A
then press any key:_
```

Remove the original disc from the drive, and insert the copy: hit **[ENTER]** and the last line disappears to be replaced by

```
ROINTIME.DEM Copied.

Copying complete
Please insert a CP/M system disc into drive A
then press any key:_
```

Hit **[ENTER]** to obtain

```
FILECOPY V2.1 finished

A>
```

Try to copy FILECOPY.COM

```
A>FILECOPY FILECOPY.COM

FILECOPY V2.1

Please insert SOURCE disc into drive A
then press any key:_
```

Press **[ENTER]** then

```
No SOURCE file present on disc
Please insert a CP/M system disc into drive A
then press any key:_
```

Press **[ENTER]** then

```
FILECOPY V2.1 abandoned

A>
```

Though the file FILECOPY.COM is on the disc, and can be used as an executable file, FILECOPY.COM is hidden from other programs, in fact FILECOPY cannot find itself (unless the STAT was changed to $DIR).

If you have two drives the copying could have been performed by the CP/M standard program PIP (Peripheral Interface Program). Assume that the copied disc (without ROINTIME.DEM) is in drive A, and that the original disc is in drive B

```
A>PIP A:ROINTIME.DEM=B:ROINTIME.DEM

A>
```

or, equivalently,

```
A>PIP A:=B:ROINTIME.DEM

A>
```

FILECOPY does not allow the name of the file to be changed, so that the name of the source file has to be the same as the name of the destination file, PIP, however, allows you to make two copies of a file, each with a different name. The DUMP program (stored on the file DUMP.COM) can be created from the file DUMP.ASM: later we will examine the means by which the file DUMP.COM is produced, but for now we will make a copy of the file DUMP.ASM and call the copy DOMP.ASM

```
A>PIP DOMP.ASM = DUMP.ASM

A>DIR *.ASM
A: DUMP      ASM : DOMP      ASM
A>
```

PIP cannot transfer files between user areas unless the file PIP.COM is in the source user area, but we can use FILECOPY to copy between any two areas.

```
A>STAT USR:

Active User : Ø
Active Files: Ø
A>USER 1
A>DIR
NO FILE
A>USER Ø
A>FILECOPY DUMP.ASM /SØD1

FILECOPY V2.1

Copying will be from user Ø to user 1

Please insert SOURCE disc into drive A
then press any key:_
```

and so forth until we reach

```
FILECOPY V2.1 finished

A>STAT USR:

Active User : Ø
Active files: Ø 1
A>USER 1
A>DIR
A: DUMP    ASM
A>STAT USR:
STAT?

A>
```

The file **DUMP.ASM** is copied from user area **Ø** (the standard) to user area **1**. The source area is **Ø**, thus the **SØ** after the **/** demarcation symbol, and the destination area is **1**, thus the **D1** following the **SØ**. Set things to rights, and enter

```
A>ERA *.ASM
A>DIR
NO FILE
A>USER Ø
A>
```

There is more to PIP than copying files, because the left hand side of the assignment may be a logical or actual device, whilst the right hand side of the assignment may also be logical or actual device. We can, for example, view the content of the file DUMP.ASM by use of the TYPE built-in command, or use

```
A>PIP CON:=DUMP.ASM
```

where CON: is a logical device, or use

```
A>PIP CRT:=DUMP.ASM
```

where CRT: is an actual device. Both CON: and CRT: refer to the screen, and thus send the content of the file DUMP.ASM to the screen.

A neat way of creating a file is the command

```
A>PIP TEST.TXT=CON:
test file [CTRL]Z
A>TYPE TEST.TXT
test file
A>
```

where 'test file' is input typed in by the user, which is ended by a **[CTRL]Z**. An ASCII file named TEST.TXT is produced, which is then successfully output by use of TYPE. We can again confuse ourselves, though not CP/M:

```
A>PIP CON:=CON:
tteesstt [CTRL]Z
A>
```

where the repetition of characters is due to the screen being used both as an input and output device. PIP is a command which is well worth investigating, and in the more complete description (below) there are several special PIP parameters, parameters which follow the rest of the command line and are enclosed in square brackets []. Try this

```
A>PIP CON:=CON:[u]
tTeEsStT  fFiIlLeE [CTRL]Z
A>
```

which shows the operation of the uppercase parameter. The input (in lowercase) is converted to uppercase. More than one element may be placed on the right hand side of the assignment, for example

```
A>PIP CON:=CON:, DUMP.ASM
```

will expect input from the console, which (terminated by a **[CTRL]Z**) will be followed by the listing of the file DUMP.ASM. The comma is a concatenation operator in this context, it allows the series of inputs on the right hand side to be sent to the output (on the left) in succession.

Parts of a file may be selected, by use of Q (for Quit copying) and S (for Start copying):

```
A>PIP CON:=DUMP.ASM[SOPNMSG: [CTRL]Z ]
OPNMSG: DB      CR,LF,NO INPUT FILE PRESENT ON DISK$'
```

until the end of the file is reached. A common parameter is the V (for Verify option

```
A>ERA DOMP.*
A>PIP DOMP.ASM=DUMP.ASM[v]

A>
```

which not only copies the file to the new name but also checks on the accuracy of the copy.

As a final point, enter

```
A>PIP
*CON:=DUMP.ASM
```

and the file is listed, with the * prompt at the end

```
*con:=con:[u]
pPiIpP [CTRL]M [CTRL]J

[CTRL]Z * [CTRL]M

A>
```

and in this last example, console copies to console (with uppercase conversion) with pip being the text entered. The line of text is completed by the carriage return line feed pair (**[CTRL]M [CTRL]J**), and the input is ended by **[CTRL]Z**. The * prompt is reasserted, and carriage return (shown as **[CTRL]M**) hit. The PIP session ends.

This rest of this chapter contains technical details of the operation of the file handling commands CLOAD, CSAVE, FILECOPY, PIP, and STAT.

# 5.3 CLOAD

CLOAD is a special Amstrad utility stored on file CLOAD.COM, which reads in a cassette file and writes it to a disc file. AMSDOS creates a special form of header record when writing to disc, but CLOAD does not replicate that header. This means that there may be problems of compatibility when using AMSDOS to access a file created by CLOAD.

A program on cassette named SNOW is copied on to disc by

```
A>CLOAD "SNOW",SNOWDISC

CLOAD V2.Ø

Press PLAY then any key:_
```

and the cassette file is loaded, CLOAD finishes, and if we

```
A>DIR SNOW????
A: SNOWDISC
A>
```

This shows that the cassette file SNOW has been copied to disc under the name SNOWDISC. If the name was to remain the same then the disc file name need not be given.

There are certain messages which can be produced during a CLOAD session.

```
Invalid cassette filename
```

- Output when a cassette name is specified which contains more than sixteen characters.

```
Invalid CP/M filename
```

- Output when an illegal CP/M file name is specified, either the name is invalid or (if none given) the cassette file name is not a valid CP/M name.

```
Cannot read protected cassette files
```

- Output when a cassette file is protected (cannot be read).

```
Program error: Cassette stream in use
```

- This message should not be output in normal operation.

```
Program error: Cassette stream not in use
```

- This message should not be output in normal operation.

```
** BREAK **
```

- Output if **[ESC]** is pressed whilst reading from cassette.

```
Disc directory full
```

- Output if the directory of the CP/M disc is full.

```
Failed to rename temporary file
```

- Should not normally be output. A temporary file (with filetype **.$$$**) is created when the cassette file is being transferred, and, on successful completion, this file is renamed with the correct filetype. If a file already exists with that name the first file is renamed with a **.BAK** filetype.

```
Disc or directory full
```

- Output if **CLOAD** fails to write a record to disc due to either the disc or the directory being full.

```
Failed to close CP/M file correctly
```

- Output if an error is produced when **CLOAD** closes the CP/M file.

# 5.4 CSAVE

An Amstrad utility program stored on the file **CSAVE.COM**, which copies a file from disc on to cassette. **CSAVE** assumes the cassette file is an ASCII type 1 file.

A file is transferred by

```
A>CSAVE SNOW.TXT, "SNOW", 1
```

where the name of the disc file is **SNOW.TXT**, the cassette file is to be called **SNOW**, and the cassette file is to be saved at the fast rate of 2000 bits per second. If the cassette file is to have the same name as the disc file, then the second name can be omitted, and if the normal (or slow) rate of saving is to be used, then that parameter may also be omitted (or set to zero).

There only a few messages which accompany this program.

```
Invalid cassette filename
```

- Output when a cassette name is specified which is more than sixteen characters.

```
Invalid CP/M file
```

- Output if an illegal (or no) CP/M file name has been specified.

```
Program error: Cassette stream in use
```

- Should not be output.

```
Program error: Cassette stream not in use
```

- Should not be output.

```
Invalid speed setting (you may only specify 0 or 1)
```

- Output if an illegal speed setting is entered.

```
** BREAK **
```

- Output if the **[ESC]** key has been pressed whilst saving to cassette.

```
CP/M file does not exist
```

- Output if the CP/M file does not exist.

# 5.5 FILECOPY

A special Amstrad utility stored on a file `FILECOPY.COM`, which copies files on a single drive, with the optional use of wildcards. All disc formats are supported, and any R/O source files will be copied over to R/W destination files.

Examples of the use of `FILECOPY` have been given previously, but the general form of the command is

```
A>FILECOPY filename.typ /SnDm
```

where `filename.typ` is the name of the file to be copied, and the (optional) parameter `/SnDm` sets the source user area as n and the destination user area as `m` (both user areas have a value with the range 0..15). If the parameter is not given, or any part of the parameter is missing, then the default value is the present user area. If no parameters are given, then an error is flagged and `FILECOPY` is abandoned.

If there is a wildcard designation in the file name, then FILECOPY reminds the user that there is an ambiguous file name, and if each individual file name is to be confirmed. The confirmation request allows you to select from a disc, because, for example, not all the .COM files need be copied by

```
A>FILECOPY *.COM

FILECOPY V2.1

Please insert SOURCE disc into drive A
then press any key:_ [ENTER]
Ambiguous file name: Confirm individual files (Y/N) ?
```

Press **[Y]** then

```
AMSDOS  .COM  Copy (Y/N) ?
```

and thus the process continues, until there are no more .COM files to be copied. If you enter that you do not wish individual files to be confirmed, then all .COM files are copied (with much manipulation of discs in the drive A).

Certain error messages are produced by FILECOPY.

```
↑C...aborted
```

- Output when a **[CTRL]C** has been typed whilst FILECOPY was expecting keyboard input. The program is abandoned.

```
Failed to open SOURCE file correctly
```

- Output when FILECOPY cannot open the source file. This may be the result of a read fail, or the insertion of an incorrect source disc. The program is abandoned.

```
Failed to close DESTINATION disc correctly
```

- Output when destination file cannot be closed. This may be the result of the directory being full. The program is abandoned.

```
DESTINATION disc directory full
```

- Output when there is no room in the directory to create a new entry. The program is abandoned.

```
DESTINATION disc full
```

- Output when there is no more memory available on the destination disc. The program is abandoned.

```
The DESTINATION disc has an unknown format
```

- Output when a disc does not have one of the standard Amstrad disc formats, or if the disc is unformatted, or if the disc is corrupt. There is a reprompt for the destination disc.

```
The SOURCE disc has an unknown format
```

- Output when the source disc does not have one of the standard Amstrad disc formats, or if the disc is unformatted, or if the disc is corrupt. The source disc prompt is repeated.

```
SOURCE disc missing
```

- Output when there is no disc in the drive. The source disc prompt is repeated.

```
DESTINATION disc missing
```

- Output when there is no disc in the drive. The destination disc prompt is repeated.

```
DESTINATION disc is write protected
```

- Output when FILECOPY cannot write to the destination disc because the write protect tabs are closed. The destination disc prompt is repeated.

```
Incorrect DESTINATION disc
```

- Output when user has inserted the wrong disc in the drive, that is, in copying a long file (with multiple transfers) the wrong disc was inserted part way through the copying. This message is only displayed when the file is longer than one data buffer.

```
Failed to read SOURCE disc correctly
```

- Output if source file is incomplete, or has zero length, or a read fail has occurred. The program is abandoned.

```
Failed to write DESTINATION disc correctly
```

- Output if a write fail occurs whilst copying to the destination disc. The program is abandoned.

```
WARNING: DESTINATION file <filename.typ> is incomplete
```

- Output if FILECOPY cannot successfully write the destination file <filename.type>.

# 5.6 PIP

Is a standard CP/M utility program stored on the file PIP.COM, and is a generalized copying program (PIP stands for the Peripheral Interface Program), which allows files to be treated like devices, and vice versa.

The standard form of PIP is either

```
A>PIP DESTINATION=SOURCE[PARAM]
```

or

```
A>PIP
*DESTINATION=SOURCE[PARAM]
```

and the DESTINATION and SOURCE parts of the assignment can be either disc files or devices, in almost any combination.

## DESTINATION

Can be a file designation of the form d:filename.type. If the designation only consists of a drive specification (A: or B:) then the filename.typ is taken from the file on the SOURCE.

Can be a device name (logical or physical) that is, those given by STAT VAL: and output is sent to that device (taking into account the nature of the devices, and the form of device assignments). Do not experiment with unusual devices (eg PIP UC1:=CON:) unless you are willing to reboot the system.

## SOURCE

Can be a file designation of the form d:filename.type. If the file is on the currently logged drive, then the drive designation need not be given.

Can be a device name (logical or physical) that is, those given by STAT VAL: and information is taken from that device (taking into account the nature of the devices, and the form of device assignments). Do not experiment with unusual devices (eg PIP CON:=UR1:) unless you are willing to reboot the system.

Can be a sequence of files and/or devices connected by commas (concatenated), for example, =FIRST.TXT,SECOND.TXT,THIRD.TXT will take information from the file FIRST.TXT, then information from SECOND.TXT, and finally THIRD.TXT, assigning that information to the disc or device on the DESTINATION. If any parameters are to be included, they accompany the appropriate SOURCE element.

## PARAM

These are optional parameters which control the form of the data transfer between SOURCE and DESTINATION. The parameters (if present) are enclosed in square brackets. The PIP parameters are

B

- Block mode transfer. Data is buffered until a **[CTRL]S** signal is received from the SOURCE device. This allows transfer of data from a continuous reading device (such as a serial I/O device). On receipt of **[CTRL]S** the buffers are cleared and more input is accepted. If the buffers overflow, then PIP issues an error message.

Dn

- Deletes all characters that extend past column n in the transfer of data (used to truncate long lines).

E

- Echoes all data transfers on the console as they occur, so that PIP CON:=CON:(E) will echo on the display three examples of each character (and confuse the screen display).

F

- Filters out all form feeds (**[CTRL]L**). The P parameter can be used to insert new form feeds.

Gn

- Gets a file from user area n (0..15), though PIP has to be resident in the DESTINATION user area.

H

- Transfers HEX data. All data are checked for proper Intel hex file format. Nonessential characters are removed. The user is prompted for corrective action if errors occur.

I

- Ignores :00 records in the transfer of Intel hex format files. This parameter automatically sets the H parameter.

L

- Translates upper case to lower case characters.

N

- Adds line numbers to each line transferred to the DESTINATION, starting at 1 (in increments of 1). Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified leading zeroes are included, and a tab is inserted after the number. The tab is expanded if parameter T is set.

O

- Transfers non-ASCII files. The normal CP/M end of file marker is ignored.

Pn

- Includes page ejects at every n lines with an initial page eject. If n = 1 (or it is omitted) page ejects are every 60 lines. If the F parameter has been set, then the F operates before P.

Qstring↑Z

- Quits copying from the SOURCE when the string (terminated by **[CTRL]Z**) is encountered. If the SOURCE and DESTINATION follow on the same command line as PIP, the string is translated to upper case, but if they follow a * prompt, then there is no conversion.

R

- Reads system files.

Sstring↑Z

- Starts copying from the SOURCE when the string (terminated by **[CTRL]Z**) is encountered. The string commences the output to DESTINATION. If the SOURCE and DESTINATION follow on the same command line as PIP, the string is translated to upper case, but if they follow a * prompt, then there is no conversion.

```
Tn
```

- Expands tabs (**[CTRL]I**) to every nth column during the transfer from SOURCE to DESTINATION.

```
U
```

- Translates lower case to upper case characters.

```
V
```

Verifies that data has been copied correctly by rereading the DESTINATION.

```
W
```

- Over writes R/O files without any checking on the console.

```
Z
```

- Zeroes the parity bit on input for each ASCII character (makes all byte values range from 0 to 127 by subtracting 128 from values of 128 and greater).

The following are possible error messages when using PIP.

```
ABORTED
```

- Output when PIP was stopped by pressing a key.

```
CANNOT CLOSE DESTINATION FILE
```

- The destination file cannot be closed, check that the correct disc is in the drive, and that the disc is not write-protected.

```
CANNOT READ
```

- Output when PIP cannot read SOURCE, for example, device might be input only, or not implemented.

```
Checksum error
```

- A HEX checksum error in transferring a file was encountered. Examine the file, and take corrective action.

```
CORRECT ERROR, TYPE RETURN OR CTRL-Z
```

- A HEX record checksum error was encountered during the transfer of a HEX file. Correct the error.

```
DESTINATION IS R/O, DELETE (Y/N) ?
```

- Output when the DESTINATION is a R/O file, if [Y] is entered, the file is deleted before the copy is made.

```
DISK READ ERROR
```

- Output if the SOURCE cannot be read correctly, could be an unexpected end of file marker. Correct the file.

```
DISK WRITE ERROR
```

- Output when a disc write operation to the DESTINATION cannot be performed, possibly due to a full disc. Take corrective action.

```
ERROR: BAD PARAMETER
```

- Output when an illegal parameter was entered in the command line.

```
FILE NOT FOUND
```

- Output when a SOURCE file does not exist.

```
INVALID FORMAT
```

- Output when the PIP command line is incorrect.

```
INVALID SEPARATOR
```

- Output when elements of the SOURCE have an incorrect character as separator.

```
INVALID USER NUMBER
```

- Output if a user number is outside the range 0..15.

```
NO DIRECTORY SPACE
```

- Output if there is not sufficient directory space. Erase some entries, or use a new disc.

```
NO FILE
```

- Output if the specified file cannot be found.

```
NOT A CHARACTER SOURCE
```

- Output if the SOURCE specified is illegal, possibly an output device has been specified by mistake.

```
QUIT NOT FOUND
```

- Output if a string argument to a Q parameter was not found in the specified file.

```
UNEXPECTED END OF HEX FILE
```

- Output if an end of file marker was encountered prior to a termination HEX record. Take corrective action.

```
Unrecognized Destination
```

- Output if DESTINATION is incorrect.

```
VERIFY ERROR
```

- Output when PIP has the V parameter specified, and there was a problem with the verification. Indicates a failure in either the disc or the drive.

# 5.7 STAT

Is a standard CP/M utility program stored on the file PIP.COM, and is a program designed to enable the user to investigate the present characteristics of the system, the STATus and STATistics.

STAT has two main functions: one, the examination of status and statistics, and, two, the alteration of status. There are many different parameters to STAT which can be used in the examination of the system, and all are of the basic form STAT PARAM.

```
STAT d:
```

Gives details of the drive status, for example,

```
A>STAT
A: R/W, Space: 31k

A>
```

that is, drive A is a R/W disc and there is 31K left on the disc. If the drive designation is given, then the status of the disc in that drive is given.

```
STAT wfn
```

Is used to examine the status of the files specified, where a wildcard specification can be used to select more than one file for examination. Each file is characterized by certain items of information:

```
A>STAT AMSDOS.COM

 Recs  Bytes  Ext Acc
    2    1k    1 R/O A:AMSDOS.COM
Bytes Remaining On A: 31k

A>
```

That is, 2 records, 1K file length, 1 extent, R/O access status: if the command is slightly altered to

```
A>STAT AMSDOS.COM $S

 Size  Recs  Bytes  Ext Acc
    2    2    1k    1 R/O A:AMSDOS.COM
Bytes Remaining On A: 31k

A>
```

That is, as before, but also we now know that the virtual size of the file is 2 records (for ordinary sequential files the Size and Recs values are the same, it is only for random access files that the two differ).

```
STAT VAL:
```

This produces a summary of the available status commnds (that is, the status of status commands)

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

The intention of this version of the STAT command is to give a reminder of other uses of the command.

```
STAT DEV:
```

This parameter gives the present status of the logical to physical device assignments.

```
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:

A>
```

```
STAT DSK:
```

This parameter produces information on the current status of the disc and drive.

```
A>STAT DSK:

    A: Drive Characteristics
 1368: 128 Byte Record Capacity
  171: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block    36: Sectors/ Track
    2: Reserved Tracks

A>
```

and this information can be compared with the explanations given in the Inventory.

```
STAT USR:
```

Gives a display of the user areas that have files on the current disc.

```
A>STAT USR:

Active User : Ø
Active files: Ø 1
A>
```

That is, in this case, there are only two user areas (Ø and 1) which have any files and user area Ø is the active area at present.

All the above enquiries to be made of the system, all the following alter aspects of the system. Note that 'wfn' is a filename with possible wildcards.

```
STAT d:=R/O
```

- Sets the disc to temporary R/O status.

```
STAT wfn $R/O
```

- Sets the specified file(s) to R/O status.

```
STAT wfn $R/W
```

- Sets the specified file(s) to R/W status.

```
STAT wfn $SYS
```

- Sets the specified file(s) to SYS status, that is, the files are not disclosed by a DIR command. $SYS files are shown with their names enclosed in parentheses when a STAT wfn command is issued.

```
STAT wfn $DIR
```

- Sets the specified file(s) to DIR status, that is, the files are disclosed by a DIR command (counteracts a $SYS).

```
STAT ldev1=pdev1, ldev2=pdev2,...
```

Assigns logical devices (ldevs) to physical devices (pdevs): these assignments alter the IOBYTE settings as described by SETUP and shown in STAT DEV:. There are certain error messages which can be produced as a result of the operation of STAT.

```
BAD DELIMITER
```

- Output if command line is incorrect.

```
File Not Found
```

- Output if file is not on the disc in the specified drive.

```
Invalid assignment
```

- Output if invalid drive or file, or misspelt device name, was specified. Sometimes this error can be followed by a list of valid file assignments.

```
Invalid disc assignment
```

- Output if the drive specification is followed by anything except `=R/O`.

```
Invalid file indicator
```

- Output if the parameter was not `$R/O $R/W $SYS $DIR`.

```
** TOO MANY FILES **
```

- Output if there is not enough memory for `STAT` to operate.

# Chapter 6
# Changing the system

Take a CP/M system disc or any other disc which contains the file FORMAT.COM, and enter

```
A>FORMAT V
```

and obey the instructions, placing a blank disc to be formatted in Drive A. When formatting is complete, and you are asked if you wish to format another disc, answer **[N]**. Leave the newly formatted disc in the drive when a request is issued to

```
Please insert a CP/M system disc into drive A
then press any key:_
```

Press any key, the disc will format, then

```
FORMAT V2.Ø finished
Failed to load CP/M
Retry, Ignore or Cancel?
```

and whatever your response the CPC repeats

```
Failed to load CP/M
Retry, Ignore or Cancel?
```

The only way to rectify matters is to take out the disc you have just formatted, and insert a CP/M system disc, hit **[R]**, and all is well.

Until now, whenever a disc has been formatted, the disc has always been a CP/M system disc. The V parameter to FORMAT indicates to the program that the formatting is to be of a Vendor disc. The difference between a vendor and an ordinary system disc is important. Hit **[CTRL]C** with the vendor disc in the drive, and you find

```
A>↑C
Failed to load CP/M

Retry, ignore or Cancel?
```

or, if you enter BASIC (either by the command AMSDOS, assuming the file AMSDOS.COM is on the disc, or by **[CTRL]** **[SHIFT]** and **[ESC]**, when you type:

```
Ready
ICPM

Failed to load boot sector

Retry, Ignore or Cancel?
```

you discover the basic problem. The vendor disc does not have a boot sector: even though it does have a sector in the right place, that sector is empty.

# 6.1 Putting in the boot

We have noted earlier that a CP/M system disc has a boot sector, used for starting CP/M, and a configuration sector, used to define the form of set up for CP/M. There are other sectors on the first two disc tracks which contain the CCP and BDOS. The complete set of CP/M information is thus stored on the first two tracks of a CP/M system disc. In the case of a Vendor format disc, all is exactly the same as a CP/M system disc, except that those two tracks are blank, that is, they contain no information.

If you have written software for the CPC, and the program runs under CP/M, you cannot distribute that software on ordinary CP/M system discs, because you are breaking the terms of your licencing agreement with Digital Research. If you distribute software with the CP/M system intact, then you are giving away free copies of CP/M, so that if you do distribute software you cannot have any information on the first two (system) tracks.

Vendor format enables the software developer to distribute software without the two system tracks. The recipient of the software on a vendor disc cannot use the disc without himself having a CP/M system, because it is impossible to boot from the vendor disc. To have a vendor disc as such is somewhat restrictive for the user, and so there is provision within CPC CP/M for the filling of the system tracks from one's own system. That is, there are two programs **BOOTGEN** and **SYSGEN** which load CP/M system information into those two tracks. Once the tracks have been filled, then the vendor disc has become a system disc. Before filling the tracks, it is best to copy the disc by **COPYDISC** or **DISCCOPY** (for safety's sake).

The boot and configuration sectors are copied from a CP/M system disc by use of the special program **BOOTGEN**, where we assume the file **BOOTGEN.COM** is on a CP/M system disc in drive A.

```
A>BOOTGEN

BOOTGEN V2.Ø

Please insert SOURCE disc in drive A
then press any key:_
```

Press any key to continue to

```
Please insert DESTINATION disc in drive A
then press any key:_
```

At this point change the disc in drive A to the newly formatted vendor disc, press any key and after copying there is the option

```
Do you wish to reconfigure another disc (Y/N) ? :_ [N]
Please insert a CP/M system disc into drive A
then press any key:_
```

Leave the vendor disc in the drive, and press any key, so that

```
BOOTGEN V2.Ø finished

Failed to load CP/M

Retry, Ignore or Cancel?
```

and we still do not have a disc which is completely acceptable as a CP/M system disc, so that we have to replace the disc with a CP/M system disc, such as that on which is stored **BOOTGEN.COM** and **SYSGEN.COM**. **BOOTGEN** only adds the two sectors (boot and configuration) to the disc, and for a proper system disc we also need the information about the BDOS and CCP. We add this information by use of **SYSGEN**, and the pattern of operation is similar to that of **BOOTGEN**, until the end. When we are asked to insert a CP/M system disc, and leave the vendor disc in the drive, CP/M is quite happy to accept the former vendor disc as a present system disc.

The outline sequence is as follows:

```
A>SYSGEN

SYSGEN V2.Ø

Please insert SOURCE disc in drive A
then press any key:_
```

Press any key to obtain

```
Please insert DESTINATION disc in drive A
then press any key:_
```

At this point change the disc in drive A to the newly formatted vendor disc, press any key and after copying there is the option

```
Do you wish to reconfigure another disc (Y/N) ? :_ [N]

Please insert a CP/M system disc into drive A
then press any key:_
```

Leave the vendor disc in the drive, and press any key, so that

```
SYSGEN V2.Ø finished

A>
```

The difference comes with the line after **SYSGEN V2.Ø finished**, because in this case the vendor disc is treated as if it were a genuine system disc. It is a genuine system disc, it is no longer a vendor format disc. It is possible to alter the sequence of the two programs (**SYSGEN** preceding **BOOTGEN**), but the result is the same. Only after both programs have executed, will the vendor disc be accepted as a system format disc. Though it is possible to warm boot after a **SYSGEN** (because the CCP and BDOS are on the disc) until **BOOTGEN** is executed there can be no cold boot.

## 6.2 Changing the size of CP/M

The **SYSGEN** command can take parameters, and in this case can not only store system tracks on disc from the current TPA image (CCP plus BDOS) but also can store system tracks on disc from a disc file image of the TPA information. It is for these extra facilities that the program **MOVCPM** (move CP/M) is used in conjunction with **SYSGEN**.

**MOVCPM** changes the size of the TPA so that the top of the BDOS is lowered in memory, to allow the insertion of RSXs or machine code routines for LOGO and other languages. The memory available for CP/M on the CPC is a maximum of 179 pages, where a page is equivalent to 256 bytes (1/4 K bytes). 179 pages is equivalent to 44.75K bytes, and the size of the TPA which corresponds to this size of CP/M is 39.5K bytes (or 158 pages). The BIOS and other aspects of the CP/M system not included in the TPA area, therefore occupy 5.25K bytes (or 21 pages).

To reduce the size of the TPA, we have to reduce the total size of CP/M, but there are certain restrictions on the maximum and minimum sizes of CP/M on the CPC. The maximum size is obviously 179 pages, and the minimum size of CP/M is 64 pages. The lower limit (16K) is fixed by the design of CP/M itself, but as it only leaves 43 pages (10.25K bytes) for programs it seems pointless to reduce the system to that size, and in fact the CCP and BDOS require about half that much memory.

If the CP/M system is set up as standard there is no memory available for special RSX routines. If, however, the TPA is relocated (including the BIOS jumpblock) then RSXs can be inserted in memory from location #AD32 downwards. The amount of space available for these routines is given by the difference between 179 and the size of CP/M set by **MOVCPM**. Thus a 2K (8 page) portion of memory for RSXs is given by a command to create a CP/M system of 171. The start of memory for RSXs is thus #AD33 -#0800 = #A633.

To save any possible problems with your main system copy, take the recently altered vendor format disc (now system format) and copy over the files **MOVCPM.COM** and **SYSGEN.COM** from the standard CP/M system disc. The copying is purely for purposes of safety, because if the parameters to **MOVCPM** are entered incorrectly there is a tendency for the system to hang in mid air, and the only way to recover is to switch off. The second parameter (the *) must be present and separated from the first parameter (the size in pages) by a space.

```
A>MOVCPM 171 *

CONSTRUCTING 42k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM42.COM"
A>SYSGEN *

SYSGEN V2.Ø

Please insert DESTINATION disc into drive A
then press any key:_
```

Press any key to obtain

```
Do you wish to reconfigure another disc (Y/N) ?:_ [N]

Please insert a CP/M system disc into drive A
then press any key:_
```

Press any key, then

```
SYSGEN V2.Ø finished

A>
```

With this version of the **SYSGEN** command (that is, **SYSGEN *****) the contents of specific memory locations are loaded directly onto the appropriate disc tracks and sectors of the disc in the drive. Normally it is unnecessary to keep **MOVCPM.COM** and **SYSGEN.COM** on the disc being modified because a second disc can be inserted to store the new track information. Copy over the file **LOGO.COM** from the Dr LOGO disc, to the disc which now has the smaller TPA.

Insert the ordinary Dr LOGO disc, and warm boot to revert to a normal size TPA. Enter LOGO from the Dr LOGO disc by

```
A>LOGO
```

and, within LOGO, enter

```
?recycle nodes
2104
?
```

which indicates that the memory normally available to LOGO is equivalent to 2104 nodes. Using the same sequence with the 42K CP/M disc (onto which you have copied **LOGO.COM**) after a warm boot by use of **C**, that is,

```
A>↑C
A>LOGO

?recycle nodes
1608
?
```

so that by reducing the size of the TPA by 2K, the number of nodes has been diminished by 496. Now enter (keeping the same disc in the drive throughout).

```
A>MOVCPM 163 *

CONSTRUCTING 40k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM40.COM"
A>SAVE 34 CPM40.TPA
A>SYSGEN CPM40.TPA

SYSGEN V2.0

Please insert SOURCE disc into drive A
then press any key:_
```

Press any key to obtain

```
CPM40    .TPA     Loading
Please insert DESTINATION disc into drive A
then press any key:_
```

Press any key then

```
Do you wish to reconfigure another disc (Y/N) ? [N]

Please insert a CP/M system disc into drive A
then press any key:_ [ENTER]

SYSGEN V2.0 finished

A>DIR *.TPA
A: CPM40      TPA
A>LOGO
```

and when in LOGO you will find there are 1111 - a fall of another 497 nodes.

The file you have created for a CP/M system with 4K for machine code routines (that is, `CPM40.TPA`) can now be used to reconfigure other discs (by use of `SYSGEN`). Note that I did not use the filetype `.COM` as suggested in `MOVCPM` because `CPM40.COM` might have been confused with an executable file. In fact, if you try to execute a file `CPM40.COM`, you will cause the system to hang, or to produce completely unexpected results.

As a final warning about the pitfalls of `MOVCPM`, take a copy of the system disc with the file `FORMAT.COM`. By use of `MOVCPM 70 *` and `SYSGEN *` reconfigure the system. Now enter

```
A>FORMAT

FORMAT V2.0
```

and the system hangs, and only **[CTRL] [SHIFT]** and **[ESC]** can retrieve the situation. The file `FORMAT.COM` is too large to load into the memory available for that size of CP/M, and parts of memory are overwritten, parts which should not have been overwritten. Even `C` has no effect.

# 6.3 BOOTGEN

This is an Amstrad utility stored in the file `BOOTGEN.COM`, and is used to load a copy of the boot and configuration sectors from a CP/M system disc, and copy these sectors to another disc (usually a vendor disc which has neither boot nor configuration sectors).

The program is executed by

```
A>BOOTGEN

BOOTGEN V2.Ø

Please insert SOURCE disc in drive A
then press any key:_
```

Press any key then

```
Please insert DESTINATION disc in drive A
then press any key:_
```

At this point change the disc in drive A to the one which needs the sectors loading

```
Do you wish to reconfigure another disc (Y/N) ? :_ [N]

Please insert a CP/M system disc into drive A
then press any key:_
```

Insert the CP/M system disc

```
BOOTGEN V2.Ø finished

A>
```

**BOOTGEN** will not operate if the source disc does not possess system tracks with intact boot and configuration sectors (for example, they may be empty or the disc may be of the wrong format), or if the destination disc does not have tracks available for loading the sectors. If either disc is of an unknown format, then obviously the program is also abandoned.

All these possible errors are clearly flagged.

# 6.4 MOVCPM

A standard CP/M utility, slightly modified by Amstrad, stored as a program on the file **MOVCPM.COM**: **MOVCPM** is used to create a CP/M system of different sizes. The program is executed by

```
A>MOVCPM xxx *

CONSTRUCTING vvk CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPMvv.COM"
A>
```

where xxx is the new size of the CP/M system (in pages, that is, 256 bytes or 1/4 K bytes), and vv is the equivalent size of the CP/M system in K bytes. The relationship between xxx and vv is that vv = int(xxx/4).

The program moves the locations of the CCP, BDOS, and BIOS jumpblock down in memory, altering all relevant pointers. The outcome of the program is to leave in memory a copy of the relevant disc sectors (for the new version of CP/M). The extent of the information stored in memory is 34 pages (that is, 17 disc sectors). All the information on a CP/M system disc is contained in the first two tracks (18 sectors) but the boot and configurations sectors leave 16 sectors to be filled. The extra sector contains some check information.

These 17 sectors can either be stored on a disc file by use of the built in command **SAVE** or they can be directly copied onto the relevant tracks by use of **SYSGEN *** (the * indicates that the information to be copied onto the system tracks comes directly from memory).

There is only one message which you are likely to encounter with **MOVCPM**, that is, **INVALID MEMORY SIZE** which means that you went outside the bounds of memory for CP/M. You probably asked for 256 pages or greater, 63 pages or less. Note that the limits for CPC CP/M are 64..179 and so it is possible to create a CP/M system larger than can fit in the available memory. In such cases, using **SYSGEN** with this image will create havoc.

# 6.5 SYSGEN

**SYSGEN** is a special Amstrad utility program stored on the file **SYSGEN.COM**, the function of which is to copy information onto the system tracks of a standard format disc. The boot and configuration sectors are not copied, but all other sectors on the two tracks are filled.

**SYSGEN** can copy tracks from a system disc by

```
A>SYSGEN
```

or it can copy from a memory image of the relevant sectors (left by **MOVCPM**) by

```
A>SYSGEN *
```

Most errors are clearly flagged. **SYSGEN** will not work if the source disc does not possess system tracks with boot and configuration sectors (they may be empty or of the wrong format), the destination disc has to have tracks available for loading in the copied sectors. **SYSGEN** will not work if the source or destination disc are of the wrong format.

It is possible that the image left by **MOVCPM** has been contaminated or the file saved after **MOVCPM** has been corrupted in some way, and so

```
MOVCPM memory image not present in TPA
```

or

```
filename.typ has incorrect format
```

In most cases there will be no problems of understanding.

# Chapter 7
# The Editor

Learning to use the CP/M context editor is a vast task, but fortunately many systems which involve the preparation of ASCII files provide their own editor. If you want to prepare short ASCII files the `PIP FILE.TXT=CON:` system is reasonably adequate, as long as you remember to hit **[CTRL]M [CTRL]J** at the end of each line of text. If a mistake is made, it is often easier to redo the file completely rather than bother entering the editor.

The only way to come to terms with the editor is through use: what I will give is a few example sessions.

## 7.1 Creating a file

Suppose that we wish to create a LOGO procedure file outside LOGO. We can either use `PIP`, or the editor, and as the editor allows us to modify our mistakes, we will use that.

```
A> ED PROCDEMO.LOG

NEW FILE
     : *i
    1:  to square :side
    2:  repeat 4 [fd ;side rt 90]
    3:  end
    4   [CTRL]Z
     : *b
    1: *#t
    1:  to square :side
    2:  repeat 4 [fd ;side rt 90]
    3:  end
    1: *f;
    2: *t
side rt 90]
    2: *-1d
    2: *i:Z-t

    1:  to square :side
    2:  repeat 4 [fd :*+b
```

continued overleaf

```
BREAK "?" AT +
     2: *#b
     1: *#t
     1:  to square :side
     2:  repeat 4 [fd :side rt 90]
     3:  end
     1: *e

A>DIR PROCDEMO.LOG
A: PROCDEMO.LOG
A>
```

In this session a procedure was entered, then modified, and saved. Note that the sign # is short for 65535, or 'the biggest number in ED'.

A further session:

```
A>ED EX1.BAS
      : *0v
25014/25015
      : *#t
      : *#a
     1: *#t
     1:  10 DIM colour(2)
```

....[A program listing ending with]

```
    40: 400 DATA 2,26,6
     1: *0v
23945/25015
     1: *t
     1:  10 DIM colour(2)
     1: *i
     1:  1 REM new line
     2:  [CTRL]Z
     2: *b
     1: *3t
     1:  1 REM new line
     2:  10 DIM colour(2)
     3:  20 MODE 1:ORIGIN 0,0,0,640,0,400 : REM reset
         screen
     1: *q

Q-(Y/N)? [Y]

A>
```

In this sequence we give the name of a pre-existing file (EX1.BAS) as a parameter to ED, but this file is not automatically read into the editing buffer. The request 0v asks for an indication of memory currently available (25014 bytes) and total possible memory for the edit buffer (25015 bytes).

We have to instruct ED to read in parts of the file we have specified, and we add 65535 lines by the command #a (that is, all the file). That all the file has been read into memory is shown by the injunction to list the 65535 lines of the file (it comes to 40, and then stops). The memory now available in the edit buffer is 23945 bytes, and so it can be seen that very large files can be edited in memory.

A line is inserted before the existing line 1, the 'character pointer' (CP) is moved back two lines, and we list the first three lines. The first three lines are the previous first two lines, plus our extra new line. We quit (without saving the modifications).

# 7.2 ED

The editor works on the notion of a memory buffer which acts as an intermediary between the source file (that given as a parameter to the command) and the destination file (which will have the same name at the end of the session). As the editing progresses, parts of the source file are entered into the buffer (or all, for most files) and parts of the edited file are written to a temporary destination file.

The temporary destination file is stored with the same primary file name as the source file, but with the filetype part being altered to .$$$. When the edit is complete, the original source file is renamed, in that the filetype is altered to .BAK. The destination file is renamed so that the .$$$ filetype is changed to the original filetype of the source file - in fact the destination file has the same designation as the original source file.

In CP/M the transfer of information from and to the files is accomplished by what are called the text transfer commands, and these commands use certain special terms: the source pointer (SP) is the current line in the source file, all preceding source lines have been read into the memory buffer; the memory pointer (MP) indicates the point in memory (the line number of the file) where additions can be made, that is, the extent of the present information in the buffer; and, the temporary pointer (TP) indicates the end of information currently sent to the temporary (.$$$) file.

nA

Adds the next n unprocessed lines from the source file at SP, to the memory buffer, starting at MP. The SP and MP are incremented accordingly. If upper case translation is set (by U, see later), and A is typed in upper case, then all input lines are translated into upper case.

nW

Writes the first n lines of the memory buffer to the temporary file free space. Shift lines in the memory buffer to delete these saved lines from the buffer, and alter the TP accordingly.

E

End edit. Copy all the text in the memory buffer to the temporary file, and copy all unprocessed source lines to the temporary file. Rename the files.

H

Move the head of a new file by performing an automatic E command. The temporary (destination) file becomes the new source file, and a new temporary file is created. If the original file was edited by ED FILE.TXT, then H is the same as exiting by E, and then entering another ED FILE.TXT.

O

Return to the original state, all editing is nullified.

Q

Quit editing returning to CP/M, but do not save any changes. There is a confirmation request.

You edit the information in the memory buffer, which can be considered as a sequence of source lines. As with the block storage devices, there are source, memory, and temporary pointers to lines, within the memory buffer there is a character pointer (CP) which points to the current position of an imaginary cursor, insertion and deletion occur at the CP.

Reference can also be made to lines within the memory buffer, and (for example) the instruction sequence

```
  : *#a
 1: *35:
35: *2t
35:  350 ORIGIN 0,0,260,372,0,400:CLG 1
36:  360 ORIGIN 0,0,0,640,150,250:CLG 1
35: *
```

will first add all (# means 65535 lines) the file to the (empty) memory buffer, and then go to line 35 (that is, 35:). The next two lines will then be typed out for inspection, but the CP will not move.

We will now return to line 1:, try to find the string SAVE, and type out the rest of the line containing SAVE.

```
   35: *1:fSAVE [CTRL]Z t
dumpfile$,b,&C000,&4000
   39: *
```

Note the use of the **[CTRL]Z** to terminate the string `SAVE`, when within a long line. Normally, if there is no other command to follow, simple **[ENTER]** is sufficient. The type command which follows **[CTRL]Z**, only types the line from the CP, and thus the first part of the line is omitted from the output.

The moving and changing commands are

`:n`

Move from the current line through to the line number `n`. `:nT` (for example) will type all lines from the current line up to and including `n`, and then return to the current line. Can be used in conjunction with `n:`.

`n:`

Move to the line number given. The sequence `n:T` will cause the CP to move to line `n` and type out the content of the line. Can be used in conjunction with `:n`.

`B, 0B`

If `B`, then move to just before the first line (beginning of memory buffer), or if `0B` then move to just after the last line (end of memory buffer).

`nC`

Move CP that number of characters (plus or minus): the end of line carriage return and line feed (**[CTRL]M [CTRL]J**) is counted as two characters.

`nD`

Delete that number of counters from CP (plus or minus).

`nFstring` **[CTRL]Z**

Find `n`th example of string in text in memory buffer, if `n` is omitted then first occurrence.

`I`

Insert text until a **[CTRL]Z** is entered

`Istring`

Insert string given directly at the CP, and then wait in insert (as in the case of `I`).

`Istring` **[CTRL]Z**

Insert string given directly at the CP, and then exit insert mode.

`nJstring` **[CTRL]Z** `ins` **[CTRL]Z** `del` **[CTRL]Z**

Search for string, insert `ins` next to string (juxtapose), and then delete text until `del` is found. Perform operation `n` times.

`nK`

Kill (delete) `n` lines after (`n` plus) or before (`n` minus) the CP position.

`nL`

Move CP `n` lines (plus or minus).

`nMinstructions`

Repeat the instructions `n` times.

`nNstring` **[CTRL]Z**

Works as for `F`, with the extra characteristic that the source file is automatically appended if the match has not been made.

`nP`

Move `n` pages, (plus or minus) and display page of text after CP.

`R`

Reads the temporary library file (created by `nX`) into the memory buffer.

`Rfilename.LIB`

Read the content of the library file (filetype `.LIB`) into the memory buffer.

`nSstring` **[CTRL]Z** `repl` **[CTRL]Z**

Search for string and replace with `repl`. Perform operation `n` times.

`T, ØT, ØTT, nT`

If `T`, display text on current line after the CP; if `ØT`, current line before CP; if `ØTT`, complete line (that is, before and after CP); and, if `nT`, display `n` lines (plus or minus) from the CP position.

`U, ØU`

If `U` enabled, then conversion of source file to upper case in memory buffer, if `ØU` disable conversion to upper case.

V, -V, ØV

If V, display line numbers; if -V, suppress line numbers; and, if ØV, verify total memory buffer usage, and amount of free space.

nW

Write n lines of text from memory buffer to temporary file. If n is minus then lines before, and if plus then lines after. If ØW, write all lines until buffer is half empty, or, if #W, then all lines until buffer is empty.

nX

Write n lines of text from the memory buffer to a temporary file.

The library files are files with the filetype .LIB which contain data which is to be directly loaded into the file, the library file cannot be edited as such. The temporary files (produced by nX and loaded by R) are used for the transfer of text from one point to another in a text file, or for repeating the same portion of text at different points.

When errors occur, ED outputs

BREAK "x" AT c

where c is the character, and x is one of the following:

?

Unrecognized command.

>

Memory buffer full (save some lines), or F N or S strings too long.

#

Cannot apply the command the number of time specified.

0

Cannot open LIB file in R command.

If the command ED is issued without a file name, then you are informed that:

DISK OR DIRECTORY FULL

and during a session you might find that either the disc or the directory is full. In this case it is best to issue the command Q, and try to erase files.

# Chapter 8
# Assemble and submit

The commands in this final chapter are not those for the inexpert. There are examples of the use of these commands throughout the Inventory, but in the case of SUBMIT and XSUB one should experiment and in the case of the 8080 commands, quite often specific Z80 assemblers are far superior. The outline details of these commands are given below.

## 8.1 ASM

A CP/M utility stored on the file ASM.COM to assemble a text file (filetype .ASM) into a machine code hex file (filetype .HEX). The program also generates a listing file (filetype .PRN) which contains assembly language source lines, together with error flags, and the hexadecimal values for the machine code generated by ASM.

The form of the command is

```
ASM filename
ASM filename.abc
```

where in the first case all defaults are operative, and in the second case there three parameters a b c.

The default is that the filename is short for filename.ASM, and files filename.PRN and filename.HEX will be produced as a result of the assembly. The .HEX file contains the machine code corresponding to the original program in Intel hex format, and the PRN file contains an annotated listing showing generated machine code, error flags, and source lines. If errors occur during translation, they are listed in the .PRN file and at the console.

The three parameters are used to redirect input and output files from their defaults. a represents the origin of the source file, b the destination of the hex file, and c the destination of the print file. The parameters can be either A or B, depending on the drive, and in the case of b and c the parameter can be Z which skips a generation of that file. In addition if parameter c is X then the full listing is to the console.

The error messages which acompany the assembler are too detailed and require too great a knowledge of 8080 assembler to be of use in this Guide.

## 8.2 DDT

The dynamic debugging tool (DDT) is stored on the file DDT.COM, and is designed to allow dynamic interactive testing and debugging of programs generated in the CP/M environment. The program is invoked in either of two ways

```
DDT
DDT filename.typ
```

where .typ will be either .COM or .HEX. For most purposes, the ordinary user will execute DDT for the purposes of working out how compiled programs work. DDT is used by more experienced users to patch CP/M programs.

Proper use of DDT needs the appropriate Digital Research documentation, but for reference purposes, here are the main commands.

A

Enters assembly language mnemonics with operands.

Ds,f

If D is without any parameters, then view the next 16 display lines of information, where each line is of the form: hexadecimal display address; the content of sixteen locations (in hexadecimal); and sixteen ASCII characters corresponding to the sixteen hexadecimal codes. If the parameters are used s is the start address, and f the finish address.

Fs,f,c

s is the starting address, f is the final address, and c is a hexadecimal byte constant, and this command fills the specified memory range with the specified character.

Gs,b,c

Execute a program with up to two optional breakpoint addresses. All parameters are optional, and s is the start location, b is the address at which execution stops, and c is a second address at which execution stops. GØ executes a warm boot.

Ifilename.typ

Insert a filename into the FCB. If the filetype specified is .COM or .HEX, then the file can be read into memory by use of the R command.

`Ls,f`

List assembly language mnemonics, with parameters being optional. If there are no parameters then twelve lines of assembled machine code are listed, from the current list address (which is then incremented by twelve). `s` is the start address, and `f` the finish address (if this option is used the start address has to be specified).

`Ms,f,d`

Produces a block movement of program or data from one location to another in memory. `s` is the start address, `f` the finish address, and `d` the start of the destination block.

`Rb`

Read in a file whose name has been stored in the FCB by the `I` option. If the `b` parameter is specified then it gives the bias address for the loading.

`Tn`

Trace execution for `n` steps, where if `n` is not specified, the next step is executed.

`U`

Switches off trace mode, and allows execution without tracing.

`Xa`

Allows the status of registers to be examined, and if the parameter is specified, the corresponding register or flag can be altered. The parameters are given as a result of `X` command.

**[CTRL]C**

Exits `DDT`.

The result of modifying a program using `DDT` can be saved onto disc by the command `SAVE n filename.typ`, where `n` is the number of pages occupied by the program (256 byte or #100 bytes). This information can be discovered at the outset when the program `filename.typ` was loaded into `DDT`. One subtracts PC (which is given in hexadecimal) from `NEXT` (also given in hexadecimal).

## 8.3 DUMP

A standard CP/M utility stored on the file **DUMP.COM**, which lists the contents of a file in hexadecimal.

Is activated by

```
DUMP filename.typ
```

## 8.4 LOAD

Is a standard CP/M utility (stored on the file **LOAD.COM**) which takes a **.HEX** file and converts it into an executable (**.COM**) file.

Is activated by either of

```
LOAD filename
LOAD filename.HEX
```

and produces an executable file **filename.COM**.

## 8.5 SUBMIT

This program is a standard CP/M utility (stored on the file **SUBMIT.COM**), which takes an ASCII file with the filetype **.SUB**, and treats the contents of the file as if they were instructions from the console.

For example, the file **IN.SUB** contains the one line **DUMP $1**

```
A>SUBMIT IN DDT.COM

DUMP DDT.COM

0000 .....
```

The form of the command is

```
SUBMIT filename param1 param2 ..
```

where in the case above **IN** is the filename and **DDT.COM** is **param1**. All parameters are optional, and if they are used within the **.SUB** file then they are shown as **$1 $2 $3** and so forth. For commands within commands see **XSUB**.

## 8.6 XSUB

Is a standard CP/M 2.2 utility program (on the file XSUB.COM) which allows instructions to be entered within programs executing under control of a SUBMIT file (.SUB). For example, consider the file IN.SUB whose contents are

```
XSUB
DDT $1.COM
GØ
DIR
```

which is executed by

```
A>SUBMIT IN DUMP

A>XSUB

A>DDT DUMP.COM
DDT VERS 2.2
NEXT  PC
Ø3ØØ Ø1ØØ
-GØ

(xsub active)
A>DIR .......
A : $$$      SUB
A>DIR
```

and there is no file $$$.SUB. The $$$.SUB file is a temporary file constructed, whilst SUBMIT is in action, and which contains the instructions treated by CP/M as equivalent to the console. After SUBMIT has ended, the file is deleted.

# Appendix 1
# An Amstrad CP/M Inventory

This appendix is intended to help users find their way around CP/M on Amstrad CPC computers. In fact, the contents of this inventory extend beyond CP/M as such, because CP/M has to be seen in relation to the CPC computer, on which it is implemented.

The inventory includes details of special Amstrad related features such as the AMSDOS disc system used in the BASIC programming environment. Only certain specific AMSDOS commands (such as `|CPM`) are given herein, as many such commands are not relevant to CP/M. Details are given of the special Amstrad utility commands used in CP/M.

The entries are arranged in ASCII order, so that (in the main) the entries are in alphabetical order. The special characters (such as `*`) are in the order given by the ASCII code for those characters. Commands are always shown in upper case when they appear in the inventory, but usually such commands are entered in lower case for both CP/M and AMSDOS.

## THE INVENTORY

### $DIR

A parameter used by the transient command `STAT` to reveal hidden files when the `DIR` command is issued. Balances the `$SYS` parameter, which hides files from `DIR`.

### $R/O

A parameter used by the transient command `STAT` to set the access status of a file to the status of `R/O`, that is, read only. A file with the access state `R/O` cannot be altered or erased. Compare with `$R/W`.

### $R/W

A parameter used by the transient command `STAT` to set the access status of a file to the (default) status of `R/W`, that is, read and write. A file with the access state `R/W` can be altered, and even erased. Compare with `$R/O`.

## $S

A parameter used by the transient command STAT to find the 'size' of files on a disc. Generally, this command is unnecessary because the result is always the same as the number of records, unless the file is random access.

## $SYS

A parameter used by the transient command STAT to hide files when the DIR command is issued. The $DIR parameter reveals hidden files.

## * (command line)

Used by some transient programs (for example, PIP) to allow command lines to be entered as part of the program: for example, rather than entering PIP LST:=T.TXT and then the command PIP LST:=Y.TXT, it is possible to enter PIP without parameters. The program then prompts for input with a *, and one can enter *LST:=T.TXT followed by **[ENTER]**, plus *LST:=Y.TXT followed by **[ENTER]** (the * is output by the CPC). The sequence of commands is terminated by a single **[ENTER]**.

## * (wildcard)

A wildcard is a symbol which replaces all or some examples of a certain part of a file name.

In the case of *, all instances are replaced. For example, a file designation *.BAS will be interpreted by CP/M as refering to all files with the filetype .BAS (that is, all BASIC program files). The file designation HILO.* will be interpreted to mean all files on the disc with the main file name HILO, and any legal filetype.

Wildcards are used extensively by commands which manipulate files, and (for example) to erase all files one can enter ERA *.* or ERA ????????.??? (see ? for the other wildcard).

## (and also)

Used in the transient command STAT to indicate that there is more than one assignment: for example, to make two assignments on one line with STAT we can use STAT PUN:=TTY, LST:=LPT: (instead of the two lines STAT PUN:=TTY: and also STAT LST:=LPT:).

## , (concatenate)

Used in the transient command PIP to indicate that two files are to be concatenated: for example, to join two files A.TX1 and A.TX2 (to produce a new file A.TXT) we issue the command PIP A.TXT = A.TX1,A.TX2.

## 8080

The Intel 8080 microprocessor, see Microprocessors.

## 8085

The Intel 8085 microprocessor, see Microprocessors.

## = (assign)

Is used in various CP/M commands to indicate that the input(s) named on the right of an assignment is/are to be sent to the output named on the left of the assignment, after modification as indicated by the nature of the command.

## ? (wildcard)

A wildcard is a symbol which replaces all or some examples of a certain part of a file name.

This wildcard only matches one character: for example, HILO?.BAS will match HILO1.BAS and HILO2.BAS but not HILO.BAS. More than one example of the wildcard will match up to that number of characters in the file name.

Wildcards are used extensively by commands which manipulate files, and (for example) to erase all files one can enter ERA ????????.??? or ERA *.* (see * for the other wildcard).

## A:

In a file designation indicates that the file to which reference is being made is situated on the disc in drive A. This symbol is also used to designate the complete disc on drive A.

## A>

When in command level CP/M indicates that the disc in drive A is the default drive. Drive A is the 'logged' drive.

## Absolute code file

A disc file which contains instructions for a compiled program but where the program will only successfully execute when placed in one position in memory, can be contrasted with a relocatable code file.

## Acc

Occurs as a heading in the output from the transient program command `STAT`, and is short for Access status.

## Access

A file can either be read only (`R/O`) or read and write (`R/W`) and these are the two forms of access. In a listing displaying the status of files - issued by a `STAT` command - there is a heading `Acc` which gives the access state for each specified file.

The access state can be altered by use of the `$R/O` and `$R/W` parameters to the transient program `STAT`. The access status of the complete disc can be altered by the assignment of the requisite status (`R/O` or `R/W`) to the drive designation `A:` or `B:`.

## AMSDOS

Is the name of the disc operating system used when in Locomotive BASIC, and is the default environment when the CPC 664 (or 464 with the DDI-1) is switched on. To reach the CP/M disc operating system from within AMSDOS, one enters `|CPM` to cold boot CP/M.

If in CP/M one enters the word `AMSDOS`, then one re-enters Locomotive BASIC. There is a special file `AMSDOS.COM` in the Amstrad implementation of CP/M which contains a transient program to restart BASIC (rather than the reset combination **[CTRL][SHIFT][ESC]**).

## ASCII

This is an acronym that stands for the American Standard Code for Information Interchange, but the implementation of the code is not so standard as might be hoped. Each character is represented by a number from 0 to 127 (that character's ASCII code), and it is intended that these numbers be used by electronic devices to store and transmit information between devices (for example, computers).

Generally the codes 32 to 126 are fairly standard, being constant across computers and other devices with which computers need to communicate. The characters corresponding to codes from 0 to 31, however, are not so standard - these characters are usually known as control characters.

On the CPC, the codes from 0 to 31 can represent: special characters (the machine specific firmware set); or special control character commands (used, for example, in SETUP); or special CP/M control characters (shown as **[CTRL]C** to **[CTRL]Z**), Even within CP/M the same control characters can have different interpretations depending upon which particular transient program is being executed. This is especially true of the ED program.

The original ASCII codes only ran from 0 to 127, but these codes have been extended from 128 to 255 for many devices. The CPC firmware set uses these extra codes for special characters, and some printers (for example) use these codes for italic versions of the characters from 0 to 127 - the character A has an ASCII code value of 65, and italic A has an ASCII code of 65 + 128 = 193.

The difference between the interpretation of code values between devices is often the reason why printers (say) do not accurately reflect information on screen.

## ASCII (text) file

An ASCII (or 'text') file is a disc file in which the information is stored in character form. Information is still stored as numbers, but the interpretation of the numbers is that of the corresponding ASCII codes - normally only code values from 0 to 127 are stored in the file. ASCII files can be listed by the use of the CP/M command TYPE, whereas other forms of file (for example, binary files) have numbers which correspond to control characters, and thus produce unexpected results when listed on the screen.

## ASM

This is a transient program that allows a source code program in an ASCII file to be assembled (see Assembler) into a 8080 hexadecimal code file. The source code file has the filetype .ASM, the hexadecimal code file has the filetype .HEX, and a listing of the assembled program has the filetype .PRN. The .HEX file can turned into executable code file by use of the transient program LOAD.

The format of the command is ASM PROG (where in this case we will suppose there is a source code file PROG.ASM): the assembler produces a file PROG.PRN which contains a listing of the source program, together with the machine code generated by each statement of the program. If there are errors in the program, then diagnostic error messages are given. The .PRN file contains the original source listing plus extra assembler information in the leftmost sixteen columns (for example, memory locations, and hexadecimal machine code). .PRN is thus a backup file because the leftmost sixteen columns can be edited out, and the remaining content of the file is the same as the original source file.

The file PROG.HEX is produced which contains 8080 machine code in a special form known as the 'Intel HEX format', and this file can be loaded for execution by use of other programs. As the CPC CPU is a Z80 microprocessor, and this program cannot cope with Z80 specific assembler commands, other assemblers for the Z80 are in more common use.

## Assembler

An assembler is a transient program which takes a source code file written in a simple programming language called assembly language, and converts it into machine code, or into a form suitable for conversion into machine code. Assembly language is very close to the workings of the microprocessor itself.

The ASM assembler provided with CP/M does not assemble into the Z80 machine code used by the CPC computer, though there are many Z80 assemblers written specially for use in CP/M - the CPC computers also have specially written assemblers available.

## Asynchronous device

See serial device.

## B:

In a file designation indicates that the file to which reference is being made is situated on the disc in drive B. This symbol is also used to designate the complete disc on drive B.

## B>

When in command level CP/M indicates that the disc in drive B is the default drive. Drive B is the 'logged' drive.

## Backup file

A general name given to a file (often with the filetype .BAK) which is a copy of the original file made before that file is subjected to modification (particularly in an file editor or word processor). AMSDOS creates backup files if a BASIC program is saved with the same name as an existing file.

## Basic Disc Operating System (BDOS)

See BDOS.

## Basic Input/Output System (BIOS)

See BIOS

## BASIC programming language

An acronym for Beginners All-purpose Symbolic Instruction Code, BASIC is the most popular programming language available on microcomputers. BASIC takes many forms, and the version available on the CPC is that called Locomotive BASIC. This BASIC is specially written for CPC computers, is fast to execute, and has many advanced features.

Locomotive BASIC is only one version of the language, and there are a multitude of different implementations of BASIC available on CP/M, including BASIC compilers. The versions of BASIC which are available under the CP/M operating system differ in many ways from each other, but all these BASICs share one characteristic: all are transient programs. Locomotive BASIC is not a transient program to be loaded from disc into memory (RAM), because Locomotive BASIC is permanently resident in ROM.

With the CPC, therefore, you have a choice between the resident Locomotive BASIC (plus AMSDOS), and other versions of BASIC which will have to be loaded from disc - in fact, any BASIC compiler is almost certain to be a disc based transient program, as with any other compiler.

## BAT:

Is a physical device which can be assigned to the logical device CON:. Is known as the batch processing device, but the name does not mean anything as such. The BAT: device is defined as being equivalent to input from the RDR: logical device (READER), and output to the LST: logical device (LIST). (Normally RDR: is equivalent to the TTY: physical device, and LST: is equivalent to the LPT: physical device).

Like most of the names of the logical and physical devices this device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## Batch file

A batch file is a file with the filetype .SUB, upon which the transient program SUBMIT operates. The file is a sequential list of commands which are interpreted by SUBMIT as if they were commands entered at the keyboard. The power of the batch program can be extended, using XSUB, so that it is then possible to feed commands to other transient programs. By use of parameters it is possible to extend the applicability of such a program.

## Baud rate

This is the rate at which information is transmitted between asynchronous/serial devices. The baud rate is defined as the number of 'data significant' line transitions made per second. There are only two possible states (OFF/ON, HIGH/LOW), and thus only two possible data significant states, these two states can be described by binary arithmetic. In this case the baud rate is equal to the number of bits per second which can be transmitted, because of the fact that there are only two possible data significant line transitions (HIGH/LOW).

For computers, therefore, there are these two states only, so that the baud rate is in fact equivalent to the number of bits transmitted per second. For most purposes, it is possible to assume that the baud rate is equivalent to the number of bits per second.

CPC computers offer a choice of baud rates through modification of the parameters in the transient program SETUP. To use a modem, for example, one needs to set the computer to send and receive information at the correct baud rate. The setting of the baud rate is necessary so that the device at the other end can understand information you send, and so that you can understand the information being sent down the line to you.

## BDOS

Acronym which stands for the Basic Disc Operating System, and is one of four logically distinct parts of CP/M (the other parts are the BIOS, CCP and TPA). The BDOS provides disc management facilities to control two disc drives (A: and B:) which include special dynamic methods of controlling disc allocation and usage.

When a warm boot is performed, the BDOS is loaded into memory (together with the CCP), from the system tracks on a CP/M disc. Both BDOS and the CCP are loaded into RAM. The BDOS has certain primitive functions to SEARCH for files by name, OPEN and CLOSE files, RENAME files, READ from and WRITE to files, and SELECT a drive.

BDOS is not dependent upon the particular computer hardware on which it is implemented, and only the BIOS is dependent upon the computer (and thus the CPC BIOS is specific to these computers). When the BIOS and BDOS are considered as one unit, they are known as FDOS. The FDOS sits in memory above the CCP, which is itself higher in memory than the TPA. On the CPC computers between the BIOS ROM and BDOS there is a BIOS work space, and RSXs can be introduced in this workspace if the BDOS, CCP, and TPA are located lower in memory (by use of `MOVCPM`).

## Binary file

A binary file is an image of a portion of memory stored as a disc file. The image can be that of an actual portion of memory, for example, it could be an executable code file, or an extension to memory on disc (a virtual memory file).

If the file is that of a program, the file may be relocatable or absolute: a relocatable file does not have to occupy any fixed memory position, whereas an absolute file must always occupy certain specific locations, and the program cannot be moved around in memory. Binary program files may be confused with hex program files, and sometimes they are equivalent, but normally a hex file (filetype `.HEX`,) is not a memory image. A hex file usually contains data organized in a special way which conforms to an old format for storing information on paper tape.

A program stored as a binary file takes less room than a source file stored in an ASCII format, because the binary file is more compact. The differential in size is illustrated by comparing a program in Locomotive BASIC saved as a binary file (the default), with the same program saved as an ASCII file (by use of the `A` parameter).

## Binary numbers

Numbers based on the binary system, where numbers are formed by using binary digits known as bits. On the CPC (and many other computers) binary numbers are constructed from groups of eight bits known collectively as a byte. A binary number formed from eight bits can take values which extend from 00000000 to 11111111, which - in denary form - are 0 to 255 (256 different values). The importance of the byte as a basic unit explains why the numbers 255 and 256 are so frequent in computing.

Ten bits extend from 0000000000 to 1111111111, or (denary) 0 to 1023, which is a total of 1024 different values. 1024 is equivalent to 1K, and two bytes (that is, sixteen bits) can take 65336 different values, equivalent to 64K. This is why 64K is such a common unit of memory: using two bytes we can point to 64K different locations.

## BIOS

An acronym which stands for Basic Input/Output System, and is one of four logically distinct parts of CP/M (the other parts are the BDOS, CCP, and TPA). The BIOS provides the primitive operations necessary to control devices such as disc drives and printer, as well as the essential console VDU. As the CP/M system is divided into these distinct modules the standard system is easily modified to produce customized versions. Amstrad have customized CP/M to run on the CPC, and it is possible for the user to recustomize CP/M according to new criteria - such customization is beyond this Guide's remit.

The BIOS is implemented in ROM on the CPC computers (in the case of the 464 via the DDI-1 interface) and the BIOS ROM is overlayed by the screen RAM.

## Bit

Short for binary digit, this is the most primitive unit of information - conventionally, a bit can take either of the values 0 or 1 (equivalent, say, to ON/OFF or HIGH/LOW). As the bit is the primitive unit of information, when a standard asynchronous/serial device is used, the rate at which information is transmitted is given as the number of bits per second. It is for this reason that the baud rate is equivalent to the number of bits per second for such devices.

## Block size

The smallest unit of file storage in CP/M, which in the case of the CPC computers is equal to 1K bytes. Each 1K block is divided into two sectors (of 512 bytes), and each sector is composed of four (128 byte) records. The physical length of a file will be less than an exact number of blocks, but a file has to occupy an exact number of blocks, and thus a short file of one record (that is, up to 128 bytes) will still occupy 1K bytes storage on the disc.

## Boot

This is the general term for starting and initializing a computer system. The term 'boot' comes from 'bootstrap', so named because computers seem to pick themselves up by their own bootstraps. Starting up the CPC is a process of small programs loading bigger programs, loading other programs.

When the CPC is switched on, the CPU automatically jumps to a location in ROM, executes a short machine code program which initializes the system and starts Locomotive BASIC. CP/M is cold booted from BASIC by use of the command `|CPM`: CP/M does not boot automatically when one switches on the CPC. CP/M can be automatically booted by altering an option link in the disc drive interface from the preset value #7 to the value #1.

## Boot sector

When the BIOS cold boot routine is activated, a 'boot' program is loaded from the first sector on the first track of a CP/M system format disc. If the loading is successful then the configuration sector is loaded to complete set up and initialization. (On a CP/M system disc, the first sector is given the number #41, and the configuration sector is known as #42).

If the contents of the boot sector are modified, it is possible to produce a completely new operating system (with modifications to other aspects). There are two tracks which contain the parts of the CP/M system (BIOS loader, BDOS, and CCP) which are not held in ROM. If the size of the TPA has been altered by `MOVCPM`, say, these system tracks can be reconfigured by use of `BOOTGEN` and/or `SYSGEN`.

## BOOTGEN

An Amstrad utility transient program specially written for CPC computers, this program copies information from the boot sector on a CP/M system disc to blank sectors on a vendor disc. All CP/M system discs have the boot sector, and the sector does not vary with alterations in the configuration of CP/M. The use of `BOOTGEN` should be compared with the use of `SYSGEN`, because `SYSGEN` adds the BDOS and CCP sectors to the disc.

## Buffer

When you type information onto the screen, though it may seem as if you are in direct communication with the screen, you are not in such communication. The keyboard has a buffer which is a queue of characters formed when you type at the keyboard. The computer takes information from this keyboard buffer, not directly from the keyboard. If the system you are using is so directed, then the information entered into the buffer need not necessarily appear on the screen. For example, in the case of a password, the letters need not appear on screen (they do not 'echo').

Communication between devices is usually through one buffer or another (or a microprocessor register), so this means it does not matter from where the buffer is filled or to where the information is sent. Direct communication between devices rarely occurs, and this enables the reassignment of devices by `STAT`. The use of buffers for communication between devices means that (as long as the information is of the correct nature) the same information might be sent to either a disc file or a printer or the screen.

One of the clearest examples of the use of buffers is in the saving of information onto cassettes, where there are stops and starts as buffers are filled and emptied.

## Built-in commands

Built-in commands are a part of the CCP program, while transient commands are loaded into the TPA from disc. The built-in commands are

| | |
|---|---|
| ERA | erase a file |
| DIR | list files in the directory |
| REN | rename a file |
| SAVE | save memory contents in file |
| TYPE | type the contents of a file onto the screen |
| USER | move to a new logical area within a disc directory |

## Byte

Used as the basic unit of memory allocation, and is equivalent to eight bits.

## C programming language

A powerful computer programming language usually a compiler. Versions of C abound for CP/M, and a special AMSDOS version of C has been written for the CPC by Hisoft.

## CAT

Within the AMSDOS environment in Locomotive BASIC is equivalent to the command |DIR.

## CCP

The console command processor which provides an interface between the console and the rest of the CP/M system. Is one of the four main parts of CP/M (the other parts are BDOS, BIOS, and TPA). The CCP uses the BDOS and is effectively computer independent: the BDOS and CCP are loaded into memory in a warm boot.

The CCP can be overwritten by transient programs, and in such cases CP/M has to be rebooted (a warm boot).

## Central processor unit (CPU)

See CPU.

## Character

There are two main types of character where input to the CPC is concerned. The first type of character consists of those it is possible to enter from the keyboard. The second type of character consists of those it is not possible to enter from the keyboard. Generally speaking, characters with ASCII codes from 32 to 127 can be entered via the keyboard, but other characters are less easily defined.

In some cases there are two alternative characters for each ASCII value. There is the 'true' (ASCII related) character, and there is a user defined character. This is particularly true for those known as 'control' characters. The means by which control codes are entered depends upon whether one is at the command level in CP/M, or in a transient program in which entry of such information is modified.

## CHKDISC

This is a CPC specific transient program which checks to see if the disc in drive `A:` is identical in format to the disc in drive `B:`. The program provides a check on the results of the `COPYDISC` program. The `CHKDISC.COM` file is on the CP/M master disc, and so after running `CHKDISC` but before the checking takes place the master disc has to be replaced by the disc to be checked. For one drive use `DISCCHK`.

## CLOAD

This is a CPC specific transient program which reads a cassette file, writing the result to a disc file. The command takes two parameters, the name of the cassette file followed by the name to be given to the disc file.

The program contructs a temporary file on disc with the filetype `.$$$` and after completion renames the file to that specified. If the file name already exists, the old file is altered to produce the filetype `.BAK`.

## COBOL programming language

The standard 'business' computer programming language: the initials represent the name COmmon Business Oriented Language, and the language was originally developed for the US Department of Defence. Is normally a compiler, and versions are available on CP/M.

## Cold boot

A complete initialization of CP/M, during which no information in memory is retained, and the CPC firmware is reset by a call to the MC START PROGRAM entry point. To save information from one cold boot to another, either the information has to be permanently stored in ROM, or the information has to be available on disc. The boot sector is loaded into memory at #0100 and is then entered. The standard cold boot program loads the configuration from the disc and performs some further initialisation. When this initialisation is complete a warm boot is performed which loads the CCP and BDOS.

Should be distinguished from a warm boot, because a warm boot merely loads the BIOS jumblock into RAM, immediately after the BDOS, and makes a jump to the BIOS jumpblock entry point known as WBOOT. The warm boot loads the CCP and BDOS.

## Command

A CP/M command line, where a command line has three parts: the command keyword, the command tail, and a carriage return. A command is executed when a command line is entered after the prompt (`A>` or `B>`). The command keyword is either a built-in command, a transient program, or an executable file. The command tail and the command keyword may include a drive designation, a file designation, and various options or parameters.

## Command level

The state in which one can issue commands, normally when the prompt is present. Some programs, such as Wordstar running under CP/M, allow the user to leave the program to run another (second) program at command level - when the second program is complete, control returns to Wordstar.

## Command processor, console (CCP)

See CCP.

## Compilers and interpreters

When a program is written, the lines of program produce the source code program. For example, when you enter a BASIC program, and then `LIST`, you list the source program. Though the source program deals with activities and entities the user can understand, the source program cannot be immediately understood by the CPC's CPU .

The source program has to be converted into instructions the CPU can understand (that is, it has to be changed into machine code). Generally, the conversion (or translation) of the source program into machine instructions is performed by one of two main methods.

The simpler method uses an 'interpreter': when each line of program is encountered as the program is executed, that line is interpreted into machine code, and the machine code instructions are then executed. After that line of program has executed, the translated form is forgotten, and the next line is translated. Every time the same line is executed there has to be a new translation, because the previous translation of the line has been lost. Locomotive BASIC is an interpreted language.

A more complex method uses a 'compiler': with a compiler, the entire source program is translated into machine code to produce an object code program. The object code program is composed of machine code instructions, which are then loaded into memory (usually RAM). An object code program stored on a disc file is also known as an executable file.

Compiled programs run more speedily than interpreted programs - as a general rule - because there is very little translation left to be performed in the case of compiled programs.

Generally speaking, assembler programs run even more swiftly, because the assembler source code is closer to the machine code in the first place, and thus one is able to control more exactly the way in which any routine is to be coded. With 'optimizing' compilers, however, this difference in order of speed can be reversed, because a good optimizing compiler can achieve certain global efficiencies which are difficult to program in assembler.

## CON:

Is the console logical device to which various physical devices can be assigned, normally the physical device is CRT: the cathode ray tube device (the VDU). The physical devices which can be assigned to CON: are TTY: CRT: BAT: and UC1:, and the possible assignments can be checked by use of the transient program `STAT` by use of the command `STAT VAL:`. The present assignment can be found by `STAT DEV:`. Output from files can be sent to the console by use of `PIP`, so that (for example) `PIP CON: = TEST.TXT` will list the contents of the file `TEST.TXT` on the console.

The console is the primary input and output device, and like most of the names of the logical and physical devices this device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## Concatenating files

This operation is performed by the PIP transient program, using the , (comma) to join files.

## Configuration sector

The configuration sector is sector #42 on track 0 of a CP/M system disc, and is loaded by the boot program. The information contained in this sector refers to various system parameters which can be altered by use of the transient program SETUP. This is the second stage of a cold boot and is part of the warm boot process.

## Configuring system

A system can be configured (that is, arranged) in a physical sense (which printers, and which make of discs, for example) and also in an operational sense (how tasks are performed, and in what way). The physical configuration is mainly a case of choosing the correct interfaces between devices, and the SETUP transient program has facilities to change the form of the interfaces.

The operational configuration is set by use of the STAT transient program, as well as by use of SETUP. With CP/M input and output of information is generally by use of buffers, which is why it is a simple operation to change from one device to another. Changing assignments is merely a case of changing the device which fills the buffer, and the device which empties the buffer.

## CONSOLE

Part of the IOBYTE function, and is the name given to that part of the IOBYTE devoted to information about the console logical device (CON:). The CONSOLE field is given by the two least significant bits of the IOBYTE, where the values correspond to physical device assignments to the logical device CON:.

0 Console is assigned to the console printer device (TTY:)
1 Console is assigned to the cathode ray tube device (CRT:)
2 Batch mode: use the READER as the CONSOLE input, and the LIST as the CONSOLE output (BAT:)
3 User-defined console device (UC1:)

Normally this part of the IOBYTE is set to 01 binary, which means that the default physical device assigned to the CON: logical device is the CRT:.

Like most of the names of the logical and physical devices this device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## Console command processor (CCP)

See CCP.

## Control characters

These are characters which correspond to ASCII code 0 to 31. Sometimes the characters can be entered via the keyboard by use of the **[CTRL]** key, and sometimes they are entered in more devious ways. Depending upon the circumstance, the interpretations of control characters differ.

1. The control characters are much used by `SETUP` in setting the screen characteristics (colour, border, and so forth): the control characters - as used in this context - are given in CPC manuals as 'Additional control character commands'.

2. Certain other control characters are used by CP/M for specific purposes, and these are listed in this inventory in the form **[CTRL]X**. The control characters are thus situated near the end of this inventory.

3. Control characters may have different meanings within any CP/M program and so, for example, **[CTRL]C** will not necessarily institute a warm boot of the system when issued within a program. Wordstar, for example, uses control characters for many different purposes, even within the same program.

## COPYDISC

An Amstrad utility transient program used to copy the contents of one disc onto another. This program can only be used if you have two disc drives, if you have only one disc drive then use `DISCCOPY`. The disc onto which the copy is made does not have to have been previously formatted (see `FORMAT`) because in such a case the new disc is automatically formatted.

## CP/M master disc

The disc provided with the CPC disc system has the Dr LOGO program on one side, and on the reverse is the CP/M system master.

The master contains all the CP/M transient programs, plus the special Amstrad utilities. Both sides of the disc should be immediately set to read only by setting the special read only tabs. The system files should be copied to one side of a new disc by `DISCCOPY` and the LOGO system placed on the reverse side of the new disc. Keep the original disc safe, and away from the computer: it may be impossible to overwrite information on the disc, but it is still possible to corrupt it by accident, so take no chances.

## CPU

The central processor unit is called the CPU, and in the case of the CPC is a variant of the Zilog Z80 chip. The CPU is the engine of the computer, in that it drives everything else, though certain devices might also have their own CPU (for example, some printers). The CPU decodes instructions in machine code and executes the content of those instructions.

Any CPU has to have a control unit, an arithmetic unit (the calculating engine), and some storage for intermediate results (normally known as registers). The decoding of instructions is performed by a special type of program built into the processor, and the set of instructions available, the means for controlling the computer, the way in which calculations are performed, and the availability of registers all distinguish the type of processor.

CP/M was originally written for the Intel 8080 CPUs, in a language known as PL/M, and in 8080 assembly language. The Z80 assembly languge contains all the commands of the 8080, and so CP/M written in 8080 code will execute on the Z80, but none of the more powerful facilities of the Z80 processor will be utilized. Those parts of CP/M (such as the BIOS) which are customized for the CPC computers are, of course, able to use Z80 code.

## CRT:

Is a physical device which can be assigned to the logical devices CON: and LST:. Is known as the cathode ray tube device and is the device normally assigned to CON:. The CRT: is both an input and output device and on the CPC is equivalent to the keyboard and screen.

The term CRT is less common now than the term VDU, and like most of the names of the logical and physical devices this device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## CSAVE

An Amstrad utility transient program which copies a CP/M file onto cassette. As AMSDOS adds a special header to disc files, and this header is not used when the cassette copy is made, there may be some compatibility problems when using `CSAVE` with AMSDOS disc files.

## CTRL

When used in conjunction with other keys can produce control characters. Control characters are used at command level in CP/M, and within some transient programs.

## Data only disc format

Is one of three Amstrad disc formats (the other two are IBM PC format and the System/Vendor formats). If it is only intended to use AMSDOS files this format provides a little more space for storage, because there are no reserved tracks. When this format is used in CP/M it is not possible to warm boot (by **[CTRL]C**), because there is no CP/M information stored on the disc.

With the data only disc format all 40 tracks are available for storage of data (with 9 sectors per track numbered #C1..#C9). This format should not be confused with vendor format (which also cannot be warm booted), because with vendor format there are two tracks set aside for later loading of the CP/M system tracks (by use of the Amstrad utility transient program `BOOTGEN` and `SYSGEN`).

See also Disc Format.

## DDI-1

The disc interface which converts a CPC 464 to a CP/M computer.

## DDT

Stands for Dynamic Debugging Tool and is one of the standard CP/M transient programs supplied with all CP/M systems. `DDT` has many useful features for the more advanced programmer, including

- Examining machine code programs;
- Examining text files of any type;
- Allowing executing programs to be examined step by step;
- Allowing alterations to be made to the content of files ('patches');
- Disassembling programs.

Successful use of `DDT` requires a knowledge of machine code, though the assembler mnemonics are those of the Intel 8080 .

## Denary (decimal) arithmetic

Standard arithmetic, to the base ten.

## DEV:

Is used as a parameter with the CP/M transient program `STAT`, to give a list of assignments of physical devices to actual devices: the listing of standard assignments for CPC CP/M produced by `STAT DEV` is:

```
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:
```

These standard assignments are set by the IOBYTE function, whose original state can be modified by use of the Amstrad utility `SETUP`. Compare `STAT VAL:`.

## Device driver

A device program is a special routine which enables the CPC to use a specific physical device. Usually implemented as an RSX, and patched through special calls in the BIOS extended jumpblock, the driver controls the form of input and output for the device.

## DIR

The CP/M built-in command which gives a listing of the files as given in the directory on the logged disc, or - if a drive is specified (that is, `A:` or `B:`) - the files on the specified disc are listed. If a file specification is given, with or without wildcards, details of those files are given. The command `DIR` is in fact equivalent to `DIR *.*`.

## Directory

This is a list of files contained on the disc in question. The directory also contains a list of blocks allocated to each of the files.

The directory entry for a file is in the form of a 36 byte FCB (file control block), though a file may have more than one FCB. CPC CP/M allows each directory to contain 64 entries, information which can be discovered by the command `STAT DSK:`, though it is possible to modify the number of directory entries allowed per disc by modifying the DPB (the disc parameter block).

## Disc format

To `FORMAT` a disc is to arrange the magnetic media in such a way that the disc is ready for the storage of information according to specified systems. Certain aspects are common to all CPC disc formats:

- Only one side of the disc is used at any one time, though the disc may be reversed, and thus be treated as being two separate discs. One side cannot communicate with the other side.

- The disc is divided into 40 tracks, that is, 40 concentric rings of information -the tracks are numbered from 0 to 39. For certain formats specific tracks are reserved for system functions.

- Each track is divided into sectors, where each sector is equivalent to 512 bytes of information (1/2 K), thus if a format has nine sectors per track then this means that a disc can contain 180K bytes of information (1/2 * 9 * 40). If there are eight sectors per track then the disc contains 160K.

- The minimum size recognized for file storage is one block, which is equal to 1024 bytes (or 1K).

- The maximum number of file entries in the directory is 64.

The principal format is the System format, which is the only format which allows CP/M to be cold booted: there are nine sectors per track, two reserved system tracks, and a 2-to-1 sector interleave (see the entry for File). The Vendor format is identical to the System format apart from the fact that the reserved system tracks are empty in the case of a Vendor disc.

The data only format has no reserved tracks so that all tracks are available for data: the number of sectors per track and the interleaving are the same for the previous two formats.

The IBM format is a specialist format based on the format used by CP/M-86 on the IBM PC. There are only eight sectors per track, and only one reserved track.

## Disc parameter block

See DPB.

## Disc parameter block, extended

See XPB.

## Disc parameter header

See DPH.

## Disc sector

See Disc Format.

## Disc track

See Disc Format.

## DISCCHK

An Amstrad utility transient program which checks to see if two discs are identical in format. The program assumes one disc drive, and supports all Amstrad disc formats. The `DISCCHK.COM` file is on the CP/M master disc, and the program is used to check the performance of the `DISCCOPY` utility. For two drives use `CHKDISC`.

## DISCCOPY

An Amstrad utility program which copies the content of one disc on to a new disc. If the new disc is not formatted, then the program will also format the new disc before copying. Assumes one disc drive, see also `COPYDISC`.

## DPB (Disc parameter block)

This is a series of bytes which contain information about the disc format, including (inter alia) the number of sectors per track, the total storage capacity of the disc, the maximum number of directory entries, and the number of reserved tracks on the disc. (See appropriate entries for further details.) Some of this information is revealed by use of the command `STAT DSK:`.

In the CPC version of CP/M, the DPB is part of an extended disc parameter block (the XPB) and it is possible to find the starting address of the XPB by using the BIOS jumpblock function `SELDSK`, which returns the address of the disc parameter header. The address of the XPB is given as part of the DPH.

The DPB should not be confused with the parameter block specified in the SETUP DISC routine of the BIOS extended jumpblock which controls the physical attributes of the disc controller performance.

## DPH (Disc parameter header)

The disc parameter header is a sixteen byte portion of memory that contains information about the disc drive and which also provides a scratch area for certain BDOS operations. The principal characteristic of the DPH is that the address of the DPB is held in the two bytes at offset 10, 11.

## Dr LOGO programming language

A very powerful programming language provided with the CP/M system. The LOGO program is on the reverse of the CP/M master disc, and the language is an interpreter. A Guide to LOGO is available from Amsoft.

## Drive designation

In theory any of the letters A through to P followed by a colon, but on the CPC restricted to the two designations `A:` and `B:`.

## DSK:

When used as a parameter to the `STAT` transient program, that is, `STAT DSK:` gives a listing of the characteristics of discs in the drives known to CP/M. The information includes details of how data is to be stored on that disc (which can then be compared to the entry for Disc Format). The information on which the output is based is stored in the DPB (disc parameter block).

## DUMP

A CP/M transient program which allows the contents of a binary file to be displayed in hexadecimal form, sixteen bytes to a line, together with a byte count. Is effectively the same as one of the DDT options. To examine the content of the `DUMP.COM` file, for example, enter `DUMP DUMP.COM`.

## ED

This is the CP/M context editor. `ED` reads segments of the named file into memory, where it is available for inspection and/or change: if the content is changed, the corrections are written back to disc. At the end of the session the original file is retained but the filetype is changed to `.BAK`, and the modified file is given the same name as the original file.

It is worth remembering that not all of the file is necessarily in memory at a particular time, and thus the editor has to be instructed to read in lines of data. For many files, however, it is posssible to fit the file in memory, and thus ease the editing.

## EOF:

When used as an assignment to a file in `PIP`, sends the CP/M end of file marker.

## ERA

A built-in command which erases a file (or set of files if wildcards are used).

## Executable code file

A binary file, with the filetype `.COM` which contains machine code instructions to execute a program (an object code file).

See also Compilers.

## Ext

One of the headings used in the standard output from `STAT`, for example, `STAT *.*`. Is short for Extent (see File and FCB).

## Extent

See File, and FCB.

## FCB

The file control block is a series of 36 bytes of which 33 bytes are relevant in the case of a normal sequential access file, with all 36 bytes being used in the case of a random access file. The FCB contains information, which (inter alia) gives the drive in use, the name, filetype, the extent number, the number of records in the currently accessed extent, the current record for sequential files, and an (optional) random record number for random access files. (See appropriate references for these terms.)

The FCB is stored in the dictionary area of the disc, and is loaded into memory before one can engage in file operations. The memory copy of the FCB is updated as file operations take place, and modifications are stored permanently when the file is closed. There is one FCB per extent for a file.

## FDOS

The name given to the BIOS and BDOS, when considered together.

## File

This is a set of information stored on disc which can be referenced by a unique designation, the file name plus filetype. A CP/M file is identified by a file designation, and may in theory contain up to 65536 records. Each record is 128 bytes in length (which is 1/8 K) so, therefore, the maximum size of file is in theory 8M, rather more than the total size of a CPC CP/M disc: the maximum size of file is very much theory only.

There are two main types of file: binary files which contain bytes of information without any particular interpretation; and ASCII files which contain bytes of information stored according to the ASCII format, where the end of each line of text is delineated by the ASCII value corresponding to the carriage return (that is, value 14 equivalent to **[CTRL]M**). Binary values range from 0 to 255, and ASCII values normally range from 0 to 127.

The end of a binary file is indicated by a special CP/M marker, whereas ASCII files are terminated by a **[CTRL]Z**) or the special CP/M marker which is produced by the CP/M BDOS read operation. **[CTRL]Z** characters in binary files are not treated as end of file markers.

Files are stored in units of a sector a time (512 bytes, equivalent to two records), but though successive sectors can be regarded as being logically contiguous, the successive sectors are never physically contiguous. The reading of sectors is controlled by the FCB for each file in conjunction with a special block allocation map. There may be more than one FCB for each file.

The Amstrad has a disc access system which is called 2-to-1 sector interleave for its main disc formats, which means that the system only reads every other sector, so for a nine sector track the order of consecutive reading is 1, 3, 5, 7, 9, 2, 4, 6, 8. The alternation of sectors allows a smoother reading and writing operation, because - while the intervening sector is being skipped - other tasks can be performed. The IBM PC format does not use sector interleave.

Internally, all files are divided into 16K byte segments known as extents, so that all access can be made via a one byte pointer. One byte can point to 256 different sectors, thus one byte can point to 256/8K (= 32K), but the actual pointer used in the FCB to determine the record within the extent only takes values from 0 to 127.

## File control block (FCB)

See FCB.

## File designation

A file designation is in three parts, an (optional) drive designation, a file name, and a filetype. These are used to search through FCBs on disc to find a file and the locations of its contents.

## Filetype

A file name has a main root designation (the primary filename), which can be of up to eight characters, and an optional extra type designation, which can be up to three characters (the filetype). The filetype is used principally to differentiate between types of files, and/or the type of action to be taken by CP/M.

In some cases filetypes are optional in their meaning, but in other cases the filetype is mandatory (such as `.SUB`, or `.COM`): be careful to establish when programming whether a filetype is mandatory or only helpful. The use of filetypes to clarify the nature of files has much in its favour.

The filetype is called the 'type part' in Amstrad documentation, but 'filetype' is the customary term in CP/M.

The more common filetypes are:

`.$$$`

A temporary file, used - for example - by AMSDOS when writing to a file. When a file is opened, the file is given the filetype `.$$$` and any previous version of the file will be changed to take the filetype `.BAK` (that is, 'backup'). When the file is successfully closed the proper filetype is then appended, and the `.$$$` filetype is deleted.

.ASM

An assembler source file, frequently a file with this filetype has been produced as a result of the ASM command.

.BAK

A backup file. Used by commands such as ED, by word processors, and by AMSDOS. A .BAK file is a copy of an original file before modification by some system.

.BAS

A BASIC language program file, which in the case of Locomotive BASIC is either an ordinary program file, or a protected file: programs saved as ASCII files have no default filetype. A program stored by SAVE "A" in AMSDOS will be filed as A.BAS.

.BIN

A binary file, that is, a file which is a byte image of a compiled program, or data stored in byte form.

.C

The source file for a C language program (usually an ASCII file).

.COB

The source file for a COBOL language program (usually an ASCII file).

.COM

A CP/M transient program, either a standard CP/M 'command', or a specially produced compiled program (as may be produced as a result of a compiler such as Pascal, C, or FORTH).

.CRF

A relocatable asssembler cross reference file.

.DAT

A data file (normally taken to be in ASCII format).

.DEF

A definitions file.

.DOC

A document file (normally taken to be in ASCII format).

.FOR

The source file for a FORTRAN program (usually an ASCII file), or a FORTH file which contains FORTH screens (source definitions) - often, but not always, FORTH screens are stored in an ASCII format.

.HEX

A hexadecimal file created by (for example) the ASM command. Is used by the LOAD command to produce a .COM file.

.HLP

A help file. Such a file contains information used by systems when a request is made for assistance.

.INT

An intermediate program file. The file usually contains program object code, produced by a compiler (for example, a BASIC compiler).

.LIB

A library source file (usually an ASCII file). Is used in ED when writing a temporary file from the edit buffer: the file is given the name X$$$$$$$$.LIB, and is deleted automatically at the end of the editing session (unless there is a system failure).

.LOG

An ASCII file which contains the saved workspace from a Dr LOGO session. The file cannot be erased from within LOGO, but can be edited from within CP/M.

.LST

A list file (ASCII) created by a compiler (for example, C) which has the program source code, plus other relevant information. Sometimes .LST is used for an ASCII file produced as output from a program.

.MAC

A relocatable assembler source code file.

.MAP

A linker map file.

.MSG

An ASCII message and/or help file.

.OBJ

An intermediate object code program file, and is distinguished from .INT in that an .OBJ file always contains object code.

.OVL

An overlay file.

.OVR

An overlay file.

.PAS

The source file for a Pascal language program (usually an ASCII file).

.PRN

An ASCII file created by some text formatters, some compilers, by ASM, and similar types of program.

.REL

A relocatable object code file produced by some assemblers and by some programming language compilers.

.SUB

A CP/M SUBMIT command file. The .SUB filetype is mandatory for such purposes. SUBMIT is a transient program.

.SWP

A swap file, used by some word processors and similar systems to allow larger files to be 'edited' by only storing part of the information in memory, and then swapping portions of the file in and out of memory from a swap file. This is a simple example of a virtual memory system.

.SYM

An ASCII file containing a symbol table produced by some macro assemblers.

.TEX

An ASCII text file.

.TMP

A temporary file.

.TXT

An ASCII text file

## File name

Consists of up to eight characters, and should not contain any control characters or special CP/M symbols.

## File, relocatable

See relocatable file.

## FILECOPY

An Amstrad utility transient program to copy a file from one disc to another, where (by use of wildcards) groups of files can be copied (drive `A:` is assumed). It is possible to copy files from one user area to another.

## FORMAT

An Amstrad utility transient program to format a disc (see Disc Format), into one of the four possible styles. The program is activated by `FORMAT X` where `X` is one of the following:

```
S    System format
V    Vendor format
D    Data only format
I    IBM PC format (CP/M-86)
     System format
```

For details of these formats see under the appropriate entries (plus Disc Format).

## FORTH programming language

A flexible and fast programming language originally developed for controlling devices (such as radio telescopes). FORTH is neither a compiler nor an interpreter, but what is known as a 'threaded interpreted language'. In terms of speed of execution FORTH is as fast as most compilers, almost as fast as machine code, but also allows an interactive form of programming superior to BASIC.

Amsoft produce a version of FORTH by Abersoft, and Abersoft FORTH is a great way of developing applications which require speed of execution (such as video games). Using this version an arcade game can be developed more easily than in machine code, yet be almost as fast as one written in machine code. Amsoft FORTH has a special facility, called ZAP, which enables the whole system to be compiled in a `.COM` file, and sold as a freestanding compiled transient program.

## FORTRAN programming language

A programming language much used in scientific applications, normally a compiler. The two most common standards are FORTRAN IV (sometimes known as FORTRAN 66, after the date of the standard), and FORTRAN 77. The FORTRANs available on CP/M tend to be versions of the FORTRAN 77 standard, which has several structured facilities which were lacking in FORTRAN IV.

COBOL and FORTRAN are the two most common languages used on mainframes.

## Hex file

A file with the filetype `.HEX` in which information is stored in a special format, designed to be used with the CP/M transient program `LOAD`. The format of the hex file was designed for use with punched paper tape (due to the nature of the MDS 800 system). If, for example, the file `DUMP.ASM` which comes with the CP/M disc is assembled by the `ASM` transient program, by entering

```
ASM DUMP
```

then a file DUMP.HEX is produced. Issuing the command line TYPE DUMP.HEX lists the contents of the file as

```
:10010000210000392215702CDC101FEFFC284
```

plus many other lines, the last two of which are

```
:03021000534B2429
:0000000000
```

The original intention was to let the paper tape reader step through the tape, under control of a 'loader'. The reading continued until a colon was read, where the colon indicated the start of a record.

The first two hexadecimal digits (see hexadecimal arithmetic) denote the length of the record, and in this case the length is #10 or 16 in decimal. (Note that a record used in this sense is not the same as a record in the sense used for file storage.) The next four hexadecimal digits indicate where this record is to start in memory: #0100 is the normal starting point for most transient programs. The next two digits, #00, are a code to denote record type.

A byte is equivalent to two hexadecimal digits, and the next four bytes are #21 #00 #00 #39, which are equivalent to the assembly language instructions

```
LXI     H,0
DAD     SP
```

which - if the file DUMP.PRN is listed - can be checked with the original assembly language program.

The hex file is turned into a transient program by issuing the command LOAD DUMP which produces the file DUMP.COM.

The final line of the hex file, containing a colon and a series of zeroes, was used by the tape reader to establish the end of the data.

## Hexadecimal arithmetic

Arithmetic to base sixteen: the sixteen hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Usually the digits are prefaced by a hash # (or, in American terminology, the pound sign) or have a trailing H after the number. One byte has 256 values, and in hex the values extend from #00 to #FF (00H to FFH). Two bytes are used to point to locations in memory, and in hex two bytes are #0000 to #FFFF (0000H to FFFFH).

## IBM PC disc format

Is one of three Amstrad disc formats (the other two are the Data Only format and the System/Vendor formats). This is a specialist format, and not intended for general use.

There are 8 sectors per track, numbered 1..8, with one reserved track, and no sector interleave (see appropriate references). This format is effectively the same as the single-sided format used by CP/M-86 on the IBM PC. It is not possible to warm boot (using **[CTRL]C**) from this disc format.

## INP:

This is a special `PIP` device which can be patched into the `PIP` program by use of `DDT`. If `INP:` is used, in the form `PIP EXAMPLE.HEX=INP:`, a call is made to a machine code routine at #0103. Before the program is patched, this location contains a RETurn instruction: thus nothing happens. The locations #0103 to #0105 can then be used to call a subroutine at a specified address (an instruction which takes three bytes), data is returned a byte at a time in location #0109.

The memory locations #0110 to #01FF of `PIP` are reserved for user defined routines, and is known as the '(INP:/OUT:SPACE)'. This space can be examined by `DDT PIP.COM`, and then issuing the command

```
-d
```

where the '`-`' is the `DDT` prompt. A second use of `d` will also show the Digital Research copyright declarations, plus some of the keywords. Exit `DDT` by use of **[CTRL]C**.

## Interpreters and compilers

See compilers and interpreters.

## IOBYTE

The four logical (imaginary) devices available to CP/M (CON: RDR: PUN: and LST:) communicate to different physical (real) devices through buffers so that each logical device may take input from, or send output to, different physical devices. For some CP/M systems there is a fixed assignment of logical and physical devices, but CPC CP/M allows the assignments to be modified.

The assignments are controlled via the IOBYTE function. The assignments may be altered either by using the STAT transient program during a CP/M session, or by SETUP so that the initial assignments are automatically configured in the correct manner on a cold boot. If the SETUP program is run, part way through it is found that the default IO byte (that is, IOBYTE) settings are

```
CON: is assigned to CRT: (keyboard and VDU)
RDR: is assigned to TTY: (special IO device 0)
PUN: is assigned to TTY: (special IO device 0)
LST: is assigned to LPT: (centronics printer)
```

The physical location of the IOBYTE is at #0003, and its value can be examined by use of DDT without a file specification, and then d0 at the '-' prompt. The value revealed is #81, which is 10 00 00 01 in binary (or 2 0 0 1 expressed as two bit pair denary equivalents). The left most bits (7,6) correspond to the LIST field, bits 5,4 correspond to the PUNCH field, bits 3,2 correspond to the READER field, and bits 1,0 correspond to the CONSOLE field. (See appropriate references for further details.)

The default value for the CONSOLE is thus 1 decimal, or CON: is assigned to CRT:, READER is 0 decimal, or RDR: is assigned to TTY:, PUNCH is 0 decimal, or PUN: is assigned to TTY:, and LIST is 2 decimal, or LST: is assigned to LPT:. The IOBYTE value can be altered directly from DDT, and if the IOBYTE value is changed to (say) #A9 (10 10 10 10 01) then the assignments for READER and PUNCH are altered so that the new assignments are

```
CON: is CRT:
RDR: is UR1:
PUN: is UP1:
LST: is LPT:
```

## Jumpblock, BIOS

The BIOS jumpblock is a series of 17 machine code jumps to special locations in memory, where the ordering is standard throughout CP/M 2.2 systems. The physical locations of the jumps differ from system to system, and for the CPC implementation the jumps commence at #AD00. Certain of the functions of the standard BIOS jumpblock are replaced by the BIOS extended jumpblock.

```
AD00  JMP BOOT      ;Cold start load
AD03  JMP WBOOT     ;Warm start load
AD06  JMP CONST     ;Console character ready
AD09  JMP CONIN     ;Console character input
AD0C  JMP CONOUT    ;Console character output
AD0F  JMP LIST      ;Listing character output
AD12  JMP PUNCH     ;Punch character output
AD15  JMP READER    ;Reader from Reader device
AD18  JMP HOME      ;Go to track 00 on disc
AD1B  JMP SELDSK    ;Select disc drive
AD1E  JMP SETTRK    ;Set track number
AD21  JMP SETSEC    ;Set sector number
AD24  JMP SETDMA    ;Set DMA address
AD27  JMP READ      ;Read selected sector
AD2A  JMP WRITE     ;Write selected sector
AD2D  JMP LISTST    ;Return list status
AD30  JMP SECTRAN   ;Sector translate
```

The meanings of these mnemonics, and the action of each of the routines, are as follows:

BOOT

Control comes from the cold start loader, and this routine is responsible for basic system initialization. IOBYTE is set, and register C is set to zero to select drive A.

WBOOT

Gets control from a warm boot. There is a JMP AD03 instruction in the first three locations, location #0003 has the IOBYTE, #0004 user number and drive, #0005 to #0007 have a JMP 8F00, which is a jump to BDOS (the primary entry point for transient programs).

CONST

Samples the status of the currently assigned console device, and returns #0FF in register A if a character is ready for reading, and #000 otherwise.

CONIN

Reads next character from console into register A (setting bit 7 to zero), if no character has been input then wait.

CONOUT

Sends a character stored in register C to the console output device. The character is treated as ASCII with bit 7 set to zero.

LIST

The character is sent from register C to the current listing device. The character is treated as ASCII with bit 7 set to zero.

PUNCH

The character is sent from register C to the current punch device. The character is treated as ASCII with bit 7 set to zero.

READER

The next character is read from the current reader device into register A, with bit 7 set to zero. An end of file condition is signalled by an ASCII ↑ Z (#1A).

HOME

The disc head for the current disc (initially A) is moved to track 00. Depending on the controller either calls SETTRK or the controller emulates SETTRK with parameter 0.

SELDSK

The drive given by register C is selected for further operations, (0 for A through 15 for P). On a disc select there must be returned in HL the base address of a 16 byte area, called the Disc Parameter Header (DPH) which describes the characteristics of the disc being used. If the value #0000 is returned then a nonexistent drive has been selected.

The CPC modification to this routine means that the E register contains the status of the disc. If the value is #00 then this is the first time that the disc has been accessed, and so the BIOS will attempt to determine the format of the disc by reading an ID header. If the reading is successful then the DPB is modified to conform.

SETTRK

Register BC contains the track number for subsequent disc accesses on the currently selected drive, the sector number in BC is the same as the number returned from the SECTRAN entry point. Register BC can take on values in the range from 0..39 corresponding to valid track numbers.

SETSEC

Register BC contains the sector number, which will vary according to the disc format in use. The sector number returned in BC is the same as the number returned from the SECTRAN entry point.

SETDMA

Register BC contains the DMA (Disc memory access) address for subsequent read or write operations, that is, the location in which the 128 byte data record resides before a disc write and after a disc read. The record is normally stored from #0080 to #00FF, thus register BC will contain #0080.

READ

Assuming that a drive has been specified, the track has been set, and the DMA has been specified, the READ routine attempts to read one sector as given by these parameters. Errors codes are returned in register A, zero for no error, and 1 for nonrecoverable error. If there is an error, then CP/M should make ten retries to see if the error is recoverable. When an error is reported, BDOS reports `DOS ERR ON x : BAD SECTOR`, the user can ignore the warning or warm boot (**[CTRL]C**) to abort.

The CPC modification to this routine means that the HL register contains the address of a buffer of results from the disc controller operation.

WRITE

Data is written from the current DMA address (that is, one sector of 128 bytes) to the currently selected drive, track, and sector. Error codes from the WRITE are returned in the A register, and recovery attempts are similar to those with READ.

The CPC modification to this routine means that the HL register contains the address of a buffer of results from the disc controller operation.

LISTST

Is only used by DESPOOL, the printer spooling facility. Returns the ready status of the list device in register A. If the device is ready then #FF is returned, else #00.

SECTRAN

Sets the logical to physical sector translation of discs. The number of sectors skipped in reading is known as the 'skew factor', and for CPC CP/M the skew factor is 1 for all formats except the IBM PC format, where the skew factor is 0.

## Jumpblock, BIOS extended

The BIOS extended jumpblock consists of special CPC routines made available to the programmer to assist in portability of programs between the 464 and 664. Though the contents of ROM may alter, the jumpblock locations are fixed, and so programs which use these calls are transferable with no change to the content. (This is why the standard jumpblock calls should also be used.)

The extended jumpblock can be conveniently split into two main sections: a conventional jumpblock covering similar tasks to those of the standard BIOS jumpblock, and a set of eight routines for special I/O devices. These latter jumps can be altered (patched) to use special RSX device driver routines. If not patched they default to the standard serial interface driver (see Serial Device).

```
BE80  JMP SET MESSAGE        ;Enable/disable disc error messages
                             ([CTRL]A)
BE83  JMP SETUP DISC         ;Initialize drive parameters
                             ([CTRL]B)
BE86  JMP SELECT FORMAT      ;Select a standard format ([CTRL]C)
BE89  JMP READ SECTOR        ;Read a disc sector ([CTRL]D)
BE8C  JMP WRITE SECTOR       ;Write a disc sector ([CTRL]E)
BE8F  JMP FORMAT TRACK       ;Format a disc track ([CTRL]F)
BE92  JMP MOVE TRACK         ;Move to disc track ([CTRL]G)
BE95  JMP GET DR STATUS      ;Get drive status (PD765A reg 3)
                             ([CTRL]H)
BE98  JMP SET RETRY COUNT    ;Set number of error retries
                             ([CTRL]I)
BE9B  JMP ENTER FIRMWARE     ;Invoke a firmware routine
BE9E  JMP SET REG SAVE       ;Set/clear alternate reg saving
BEA1  JMP SET SIO            ;Initialize serial interface
BEA4  JMP ST CMND BUFFER     ;Initialize command buffer
BEA7  JMP D0 IN STATUS       ;Input status, device 0
BEAA  JMP D0 IN              ;Input from device 0
BEAD  JMP D0 OUT STATUS      ;Output status, device 0
BEBfl JMP D0 OUT             ;Output to device 0
BEB3  JMP D1 IN STATUS       ;Input status, device 1
BEB6  JMP D1 IN              ;Input from device 1
BEB9  JMP D1 OUT STATUS      ;Output status, device 1
BEBC  JMP D1 OUT             ;Output to device 1
```

Those commands with a control character as part of the description (for example, **[CTRL]I** for the SET RETRY COUNT routine) can be used from within AMSDOS by means of the KL FIND COMMAND at location #BCD4.

The functions of many of these routines are initialized as part of the SETUP transient program.

The meanings of these mnemonics, and the action of each of the routines, are as follows:

SET MESSAGE

The A register contains #00 (enable disc error messages) or #FF (disable disc error messages) and when activated this return sets the status. Normally (enabled) the BIOS will display error messages, otherwise (disabled) no messages are displayed.

SETUP DISC

The address of the disc controller parameter block is stored in register HL, and to call this routine sets values for various disc parameters including motor on/off, write current off, and head settle times. Sends a SPECIFY command to the PD765A disc controller.

SELECT FORMAT

Initializes the XPB for a specified disc format. Is mainly of use for programs such as formatters, because the SELDSK function of the BIOS jumpblock automatically detects the format of a disc. Register A should contain the first sector number of the appropriate format (see Disc Format), that is, #41 for System/Vendor, #C1 for Data Only, and #01 for IBM. The drive number is given by the content of register E (#00 for A, and #01 for B).

READ SECTOR

Reads the specified sector into the sector buffer. The HL register should contain the address of the sector buffer, register E the drive number, register D the track number, and register C the sector number. If the carry flag is set to true after the read operation then all is well, but if the carry is false then an error occured.

WRITE SECTOR

Writes to the specified sector from the sector buffer. The HL register should contain the address of the sector buffer, register E the drive number, register D the track number, and register C the sector number. If the carry flag is set to true after the write operation then all is well, but if the carry is false then an error occured.

FORMAT TRACK

Formats an entire disc track. The HL register should contain the address of the disc header information buffer, register E the drive number, and register D the track number. If the carry flag is set to true after the formatting then all is well, but if the carry is false then an error occured. The XPB must have already been set to the correct format, as must have the information buffer.

MOVE TRACK

Move read/write head to a specified track. Register E contains the drive number, and register D the track number. If the carry flag is set to true after the move attempt then all is well, but if the carry is false then an error occured. This routine is intended as a diagnostic aid.

GET DR STATUS

This routine returns the content of status register 3 of the PD765A floppy disc controller. On entry to the routine register A contains the drive number (#00 for A, #01 for B). If the carry is true after the operation, then register A then contains the drive status byte, otherwise register A is corrupt and register HL contains the address of the error buffer plus the status byte.

SET RETRY COUNT

Register A contains the number of times a disc operation is retried before being abandoned in the event of an error.

ENTER FIRMWARE

Calls a firmware routine where the address of the firmware routine is given as an in-line parameter in the machine code routine. The ENTER FIRMWARE routine must never call itself (this means that an endless loop will be initiated). Another routine which cannot be used is SET REG SAVE because that routine will interfere with the saving of registers by ENTER FIRMWARE.

SET REG SAVE

This routine enables or disables the saving of the alternate and IY registers. This routine should never be called by using the ENTER FIRMWARE routine. The A register sets the state of enabling: if A contains #00 then the alternate and IY registers are saved, but if A contains #FF then those registers are not saved.

The SETUP transient program can be used to set the state of register saving: if set to 'slow' then registers are saved, and if set to 'fast' then registers are not saved.

SET SIO

Resets and initilizes both channels of the Z80-SIO (see SIO, Z80). Also initializes the Intel 8253 used as timer for setting the baud rate for the transfer of serial data.

SET CMND BUFFER

Initializes the initial command buffer (see Buffer): when the BIOS routine CONIN is called for the first time, characters are fetched from the command buffer. When the command buffer is empty, characters are then fetched from the console. If the content of register A is #00 then, if a key is pressed, the buffer is emptied; if register A contains #FF the command buffer is retained intact, even if a key is pressed. Register HL contains the address of the buffer.

The above are the extended equivalents of the standard CP/M jumpblock routines, the following are special routines for input and output on the two special serial devices. The following features are general to all these routines:

1. All the jumps may be patched to RSXs.
2. The default device 0 is channel A of the serial interface, and the default device 1 is channel B.
3. Register BC DE HL are always corrupted by the routines.

D0 IN STATUS

Tests to see if device 0 has a character available for input. If a character is available then #FF is left in register A, else #00.

DO IN

Input a character from device 0. The content of register gives the character input.

D0 OUT STATUS

Tests to see if device 0 is ready to output a character. If the device is ready then register A contains #FF, else #00 means that the device is busy.

D0 OUT

Output a character to device 0. If device is busy, then wait. Register C contains the character to output.

D1 IN STATUS

Tests to see if device 1 has a character available for input. If a character is available then #FF is left in register A, else #00.

D1 IN

Input a character from device 1. The content of register gives the character input.

D1 OUT STATUS

Tests to see if device 1 is ready to output a character. If the device is ready then register A contains #FF, else #00 means that the device is busy.

D1 OUT

Output a character to device 1. If device is busy, then wait. Register C contains the character to output.

## Linker

A linker is a transient program which takes relocatable object code programs (.REL) from disc, and links them together to form an executable object code program (.COM). Is used by many compilers to incorporate libraries of routines stored as relocatable files. Has similarities to the LOAD transient program used for nonrelocatable files. Is not provided as standard with CP/M but many versions of linkers are available for CP/M.

## LIST

Part of the IOBYTE function, and is the name given to that part of the IOBYTE devoted to information about the listing logical device (LST:). The LIST field is given by the two most significant bits of the IOBYTE, where the values correspond to physical device assignments to the logical device LST:.

0 LIST is the teletype device (TTY:)
1 LIST is the CRT device (CRT:)
2 LIST is the line printer device (LPT:)
3 user-defined list device (UL1:)

Normally this part of the IOBYTE is set to 10 binary, which means that the default physical device assigned to the LST: logical device is the LPT:.

Like most of the names of the logical and physical devices this device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## LOAD

Is a CP/M transient program which takes a hex file (.HEX) and produces an executable code file (.COM). Is frequently used in conjunction with an assembler, for example, ASM.

## Logical device name

CP/M has two main classes of devices: the logical devices (which are known as CON: RDR: PUN: LST:) and the physical devices (which are known as TTY: CRT: BAT: UC1: PTR: UR1: UR2: PTP: UP1: UP2: LPT: UL1:).

The logical devices are used by CP/M to provide standard ways of talking to the world, through the medium of buffers or registers (see Microprocessors). The routines to control the behaviour of logical devices are implemented in the BDOS jumpblock (that is, the JMPs to CONST CONIN CONOUT LIST PUNCH READER), and thus their behaviour is standard across CP/M systems. Real devices are not, however, constant.

The physical devices are convenient names (based on designations for the MDS 800 system) which can be used to represent real devices. The physical devices take information from, or send information to, the buffers or registers which are used by the standard logical devices. What a physical device does with that information is under the control of special device drivers, patched in the BIOS and possibly also implemented as RSX calls. These special routines are adjusted to the physical characteristics of the real world devices.

## LOGO programming language

See Dr LOGO.

## LPT:

Is a physical device which can be assigned to the logical device LST:. Is the physical device known in CP/M as the line printer device, which has no meaning for the CPC. In CPC CP/M, LPT: is the printer attached to the parallel (Centronics) printer port, and this is the default assignment for the logical device LST: (see IOBYTE).

Like most of the names of the logical and physical devices this device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## LST:

Is the listing logical device to which various physical devices can be assigned, normally the physical device is LPT: the parallel printer device. The physical devices which can be assigned to LST: are TTY: CRT: LPT: and UL1:, and the possible assignments can be checked by use of the transient program `STAT` by use of the command `STAT VAL:`. The present assignment can be found by `STAT DEV:`. Output from files can be sent to the listing device by use of `PIP`, so that (for example) `PIP LST: = TEST.TXT` will list the contents of the ASCII file `TEST.TXT` on the physical device assigned to the logical listing device.

It is possible for LST: to be assigned the CRT: physical device, by use of STAT, that is, STAT LST:=CRT:. Thus, when the command line PIP LST: = TEST.TXT is entered the output is sent to the screen, in the same manner as entering TYPE TEST.TXT. The listing device can be activated to copy everything on the screen (actually everything for the logical device CON:) by entering the control character **[CTRL]P**. If a printer is attached, and normal assignments hold, then everything input to the screen (command lines for example) or output to the screen (a listing of a file for example), is copied on the printer.

If the STAT LST:=CRT: assignment is made, and **[CTRL]P** plus carriage return is issued, then to input TYPE TEST.TXT produces the following effect

```
A>[CTRL]P
AA>>TTYYPPEE  TTEESSTT..TTXXTT
```

and the listing of the file has every character repeated. The first character is that of the CON: logical device which is assigned to the CRT: physical device, and the second (identical) character is that of the LST: logical device which is assigned to the CRT: physical device. Check by

```
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is CRT:
```

Like most of the names of the logical and physical devices the LST: device name is a hangover from the days of teletypes and line printers, with paper tape readers and paper tape punches. In the standard distribution version of CP/M (before modification to conform to the CPC architecture) the devices correspond to the names on the MDS 800 development system.

## Machine code

The instructions recognized by a microprocessor. CP/M was originally developed on the Intel MDS 800 development system as a disc operating system for the Intel 8080 processor. The machine instructions understood by the 8080 were the same as those of the enhanced Intel chip, the 8085, and so machine code programs written for the 8080 were able to run without change on the 8085 (though the 8085 had a few extra instructions). As the 8085 is somewhat simpler in configuration than the 8080, the 8085 is used in preference to the 8080 for many purposes. CP/M was not changed because it ran on the 8085 without alteration to the basic code.

The Zilog Z80 was designed as an enhanced version of the 8080, and so incorporated all the instruction set of the earlier processor. The Z80 will run 8080 machine code directly, and thus CP/M works directly on the Z80 processor. The Z80 has a much larger and more versatile instruction set than the 8080, but standard CP/M does not use this extra power. When customizing CP/M to the CPC, however, the extra power of the Z80 has been utilized.

## Macro assembler

An assembler which uses macros. A macro is a section of code which is inserted into a program code when the macro is called by name in the program code. The `SUBMIT` transient program is an example of a macro facility (though not an assembler), because, when the name of the `.SUB` file is encountered after a `SUBMIT`, the content of the `.SUB` file is treated as if it were a sequence of commands. The content of the `.SUB` file is inserted at command level. The macro facility has similarities to a subroutine facility, though, as a jump does not have to be made to a specific location and back, generally macros lead to faster execution - if lengthier code.

## MDS 800

MDS is an acronym for Microcomputer Development System, and the MDS 800 was produced by Intel to help system developers produce software for the 8080/8085 microprocessors. The MDS 800 consisted of a console, paper tape punch, paper tape reader, and line printer, connected to an 8" single density IBM 3740 format disc. This disc is still the normal format for distributing CP/M software.

If this standard configuration is examined, the names for the various logical and physical devices become understandable.

## Microprocessors

A microprocessor controls the operation of a computer, and runs under control of instructions given in machine code, together with a set of internal commands which control the internal workings.

Any microprocessor has to have a control unit, an arithmetic unit (the calculating engine), and some storage for intermediate results (normally known as registers). The decoding of instructions is performed by a special type of program built into the processor. The set of instructions available, the means for controlling the computer, the way in which calculations are performed, and the availability of registers all distinguish the type of processor.

CP/M was originally developed for the Intel 8080 microprocessor, which is the first of a series of three chips (Intel 8085, Zilog Z80) where each successive chip was compatible with the earlier chips (in that programs written for an earlier processor would work on a later processor). Each successive member extended the facilities available: the progression can be seen as:-

Intel 8080 : Has a bank of six general purpose registers (8 bit), which can also be used in some circumstances as three 16 bit registers. There are certain other registers which have specific functions (inter alia: an accumulator, stack pointer, and program counter). Has a set of about eighty instructions, and needs several other chips to form a working system.

Intel 8085 : Is similar to the 8080 with the addition of serial input/output from the accumulator, a few more instructions to cope with input and output, and certain facilities (such as a timing clock) are on chip.

Zilog Z80 : Has two banks of two special 8 bit registers, and six general purpose 8 bit registers (or three 16 bit registers), only one bank is active at any one time, and one of the banks is known as the alternative set of registers. In addition to the stack and program pointers of the 8080/8085, there are two index registers which are not switchable. There are about 160 different instructions, and several new ways of addressing memory, compared to other chips.

The CPC uses a Z80, but CP/M is still largely written in the original 8080 instruction set, though with Amstrad modifications which use the improved power of the Z80.

## Modem

Short for MOdulator-DEModulator, a modem is a device which is used to convert information output by a computer to modulated wave suitable for transmission down a telegraph line. The modem also takes modulated waves transmitted to the device and converts into data suitable for input to a computer. A modem is necessary for any communication via telegraph lines (though it is possible to use an acoustic coupler for the same purposes), and a modem is an asychronous device, which operates at specified baud rates.

## MOVCPM

Pronounced MOVE CPM, this is a CP/M transient program which alters the memory requirements of CP/M, and - for example - can alter the memory occupied by CP/M so that the user can introduce RSXs without any danger of CP/M overwriting the memory locations occupied by the routines.

## NUL:

Used with the PIP transient program sends forty nulls (ASCII zeroes) to the appropriate device or file. For example,

```
A>PIP T.TXT=NUL:

A>DUMP T.TXT

0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 1A 1A 1A 1A 1A 1A 1A 1A
0030 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0040 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0050 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

A>
```

There are forty examples of ASCII #00, followed by eighty eight examples of ASCII #1A. The zeroes were sent by the NUL:, and the remainder of the file (that is, one record of 128 bytes) was filled with end of file characters (that is, **[CTRL]Z**).

## Object code

Binary instructions for a compiled program stored as either an executable file (.COM) or a relocatable file (.REL), otherwise known as machine code.

## OUT:

This is a special PIP device which can be patched into the PIP program by use of DDT. If OUT: is used, in the form PIP OUT:=EXAMPLE.HEX, a call is made to a machine code routine at #0106. Before the program is patched, this location contains a RETurn instruction: thus nothing happens. The locations #0106 to #0108 can then be used to call a subroutine at a specified address (an instruction which takes three bytes). PIP loads register C with the character to be output, and then calls the routine at #0106.

The memory locations #0110 to #01FF of PIP are reserved for user defined routines, and is known as the '(INP:/OUT:SPACE)'. This space can be examined by DDT PIP.COM, and then issuing the command

```
-d
```

where the '-' is the DDT prompt. A second use of d will also show the Digital Research copyright declarations, plus some of the keywords. Exit DDT by use of **[CTRL]C**.

## Overlay

A part of a program which is stored on disc until required, and then is loaded into a specific portion of memory to be later over written when no longer needed. In this way large programs can be tailored to fit into less memory (RAM). Many word processors, and interactive programming languages, use this form of memory management.

## Parallel device

A device for which information on (say) the bits corresponding to the ASCII code for a character are transmitted simultaneously. For example, if one imagines each bit of an eight bit code to be connected to a distinct line, communication to a device is by eight lines in parallel.

If only one line is available, then the transmission is serial. Communication between the Z80 and memory is in parallel, in that eight bits are transmitted simultaneously (by what is known as the data bus). For communication between the elements of a computer, and communication to a printer, parallel operation is preferred because (normally) it is speedier.

For purposes of communication between computers, serial operation is necessary, because only one line is available. As serial operation is necessary for telegraphic and similar communication, some computers only provide serial communication to devices - to save money.

## Parallel printer

A printer which is a parallel device.

## Parameter

To produce a directory of the current disc, one needs only only to enter

```
A>DIR
```

but, to produce a directory of all files with the filetype .COM, the command is modified by entering

```
A>DIR *.COM
```

so that an explicit request is made. The extra refinement to the command, the extra directive to define the type of file, is known as a parameter. A parameter is an adjunct which enables the user to specify more exactly what is required from the command.

## Pascal programming language

A popular programming language, usually a compiler, which was designed to assist in structured programming. There is a version specifically designed for CPC computers, Amsoft Pascal from Hisoft. There are many other versions of Pascal available for CP/M.

## Patch

The name given to a portion of machine coded inserted into an existing machine code to alter its operation in some way. A patch 'mends' a program. Digital Research distribute information on patches to CP/M, where such patches are usually inserted by means of DDT.

## $\mu$PD765A

The floppy disc controller used by the CPC. The controller is manufactured by NEC, and is intended for soft sector dual density single or double sided discs. Can control up to two disc drives as used on CPC machines.

## Physical device assignment

CP/M has two main classes of devices: the logical devices (which are known as CON: RDR: PUN: LST:) and the physical devices (which are known as TTY: CRT: BAT: UC1: PTR: UR1: UR2: PTP: UP1: UP2: LPT: UL1:).

The logical devices are used by CP/M to provide standard ways of talking to the world, through the medium of buffers or registers (see Microprocessors). The routines to control the behaviour of logical devices are implemented in the BDOS jumpblock (that is, the JMPs to CONST CONIN CONOUT LIST PUNCH READER), and thus their behaviour is standard across CP/M systems. Real devices are not, however, constant.

The physical devices are convenient names (based on designations for the MDS 800 system) which can be used to represent real devices. The physical devices take information from, or send information to, the buffers or registers which are used by the standard logical devices. What a physical device does with that information is under the control of special device drivers, patched in the BIOS and possibly also implemented as RSX calls. These special routines are adjusted to the physical characteristics of the real world devices.

## PIP

A CP/M transient program - the Peripheral Interface Program, used for sending information from one device or disc to another device or disc. Can be considered an elaborate copying program.

## PROLOG programming language

An interpreted language, PROgramming in LOGic, which is based on an implementation of simple logical assertions. PROLOG is increasingly used for the construction of databases, and for designing inferential networks in Expert Systems. Logic Programming Associates have a version of PROLOG running under CP/M.

## Prompt

At command level the prompt is `A>` or `B>`, and within programs the prompt can be of different forms. A prompt is a request for input from the user.

## PTP:

Is a physical device which can be assigned by the logical device PUN:, by use of the transient program `STAT`. On the CPC, the PTP: is the label given to the null output device.

## PTR:

Is a physical device which can be assigned by the logical device RDR:, by use of the transient program `STAT`. On the CPC, the PTR: is the label given to the end of file input marker.

## PUN:

Is the logical device which can be assigned to TTY: PTP: UP1: and UP2:, by use of the transient program `STAT`, and which corresponds to the IOBYTE field PUNCH. The possible assignments can be checked by `STAT VAL:`.

## PUNCH

Part of the IOBYTE function, and is the name given to that part of the IOBYTE devoted to information about the principal tape punching device (PUN:). The PUNCH field is given by bits 5,4 of the IOBYTE, where the values correspond to physical device assignments to the logical device PUN:.

0 PUNCH is assigned to the console printer device (TTY:)
1 PUNCH is assigned to the high speed punch device (PTP:)
2 user-defined punch 1 (UP1:)
3 user-defined punch 2 (UP2:)

Normally this part of the IOBYTE is set to 00 binary, which means that the default physical device assigned to the PUN: logical device is the TTY:.

## R/O

Appears in many contexts, and means that the file (or a complete disc) has been set to a read only status. (See `STAT filename.typ $R/O`, and `STAT d:=R/O`) When a file is set to read only, it may not be altered in any way, and may not be erased.

## R/O (temporary)

A disc can be set to temporary read only status (that is, for the length of a session) by

```
A>STAT A:=R/O
```

which means that it is only possible to read from files on the disc, and one is unable to modify. For example,

```
A>ERA T.TXT
Bdos Err On A: R/O
A>
```

Thus it was not possible to erase `T.TXT`. (See `$R/W`, `R/W`)

## R/W

Means that a disc or file has been set to read and write status (the normal state of affairs). (See `$R/W`, `$R/O`, `R/O`)

## RAM (random access memory)

Computer memory which can be altered by the system, in that information can be entered into memory locations (as bytes), and information can be read from those. RAM should be compared to ROM. It is called random access memory because - in theory - a piece of information can be retrieved or deposited as easily in one location as any other.

## Random access memory (RAM)

See RAM.

## RDR:

Is the logical device which can be assigned to TTY: PTR: UR1: and UR2:, by use of the transient program `STAT`, and which corresponds to the IOBYTE field READER. The possible assignments can be checked by `STAT VAL:`.

## Read only memory (ROM)

See ROM.

## READER

Part of the IOBYTE function, and is the name given to that part of the IOBYTE devoted to information about the principal tape reader device (RDR:). The READER field is given by bits 3,2 of the IOBYTE, where the values correspond to physical device assignments to the logical device RDR:.

0 READER is assigned to the teletype device (TTY:)
1 PUNCH is assigned to the high speed reader device (PTR:)
2 user-defined reader 1 (UR1:)
3 user-defined reader 2 (UR2:)

Normally this part of the IOBYTE is set to 00 binary, which means that the default physical device assigned to the RDR: logical device is the TTY:.

## Record

A 128 byte portion of memory on disc. See Disc Format.

## Recs

A heading output by a `STAT` enquiry, gives the number of records for a file.

## Register

A special portion of memory on a microprocessor which can either be dedicated to a special purpose, or be of general utility for storing the intermediate results of microprocessor operations.

## Relocatable object code file

A disc file containing program object code which can be executed from different positions in memory. Relocatable code files are often linked to form larger programs. Can be contrasted with an absolute code file.

## REN

A built-in CP/M command which is used to rename files.

## Resident System Extensions (RSXs)

See RSX.

## ROM (Read only memory)

A form of computer memory in which information is permanently stored, information which can only be 'read' and not written. Should be compared to RAM. ROM is mainly used for holding routines and programs for booting CP/M, and for language interpreters such as BASIC.

## RSXs (Resident system extensions)

Resident system extensions are special machine code routines which can be patched to provide a system extension such as a printer driver (see device driver). An RSX is treated effectively as if it were a special ROM routine rather than a RAM routine (facilities exist in the CPC for background ROMs). In the CP/M environment, RSXs can be inserted above the BDOS, below the BIOS jumpblock, but the size of the CP/M system has to be reduced by use of `MOVCPM`.

## SAVE

A CP/M transient program which saves the contents of memory to a disc file. Is used after `MOVCPM` to save the new CP/M system.

## Serial device (asynchronous device)

A device sends and receives information, and a serial (asynchronous) device sends or receives information one item at a time. The rate at which the information is transmitted or accepted is called the baud rate, and the timing is such that the completion of one event is used to initiate the next event. There is a facility for two serial devices in the CPC implementation of CP/M, and the characteristics of the serial devices are set as part of the special `SETUP` system provided with CPC CP/M.

The two serial devices are known as SIO Channel A and Channel B, and to use these two channels requires special hardware interfaces. These two channels are also known as special I/O devices: special I/O channel 0 is associated with the TTY: logical device, and special I/O channel 1 may be associated with the UC1: UR1: UP1: UL1: logical devices.

Transmission of information over a telephone line has to be asynchronous, because only one item of information can be sent a time -though many items can be sent, one after the other. To connect a CPC computer to a data network, therefore, one needs a modem which transmits at the appropriate baud rate - in a serial manner.

## SETUP

A special Amstrad utility transient program which is used to change the operational configuration of CP/M.

## SIO, Z80

SIO is an acronym for Serial Input/Output. A Zilog Z80A SIO/0 or a Z80A DART can be used, in conjunction with a timer, to organize two serial input and output channels (see serial device). This is a task for a specialist, however the Amstrad serial interface conforms to this specification.

## Source code

The program instructions as written by the user, before modification to machine code instructions (see also compilers and interpreters).

## STAT

A CP/M transient program used to change the status of the system (the configuration), and to provide information on the present status.

## SUBMIT

A CP/M transient program which takes the name of a file (filetype .SUB) and executes the instructions in the file as if the instructions were entered at command level. See Batch File.

## SYSGEN

An Amstrad utility transient program which copies the CCP and BDOS onto a disc. SYSGEN will not the copy the boot and configuration sectors, that is performed by BOOTGEN. Effectively allows the CP/M system to be reconfigurated on an existing CP/M system disc.

## System disc format

Is one of three Amstrad disc formats (the other two are IBM PC format and the Data Only formats). This is the normal disc format, and is closely allied to vendor format.

With the system disc format 38 tracks are available for storage of data (with 9 sectors per track numbered #41..#49). Two tracks are reserved for CP/M system use. This format should not be confused with vendor format (which cannot be warm booted), because with vendor format there are two tracks set aside for later loading of the CP/M system tracks (by use of the Amstrad utility transient program BOOTGEN and SYSGEN).

See also Disc Format.

## System disc

A disc with both the standard CP/M boot sector, and configuration sectors.

## Text (ASCII) file

See ASCII file.

## TPA (Transient program area)

Area in memory where user programs run and store data. The TPA usually starts at #0100 and extends to the top of the CCP. Large programs can, therefore, overwrite the CCP and in cases such as this the system has to be warm booted by a **[CTRL]C**.

## Transient programs

Transient programs are loaded into the TPA from disc, while built-in commands are a part of the CCP program. The standard CP/M transient programs are:

`ASM`

8080 machine code assembler.

`DUMP`

Display contents of a file in hexadecimal form.

`DDT`

Dynamic debugging tool.

`ED`

Text editor.

`LOAD`

Convert hex files into executable files.

`MOVCPM`

Move CP/M, change size of TPA.

`PIP`

Peripheral interface program.

`STAT`

Status (or statistics) program.

SUBMIT

Batch processing program.

SYSGEN

System generation.

XSUB

Extended SUBMIT program.

Certain extra transient programs have been written by Amstrad, and are:

AMSDOS

Return to BASIC.

BOOTGEN

Copy boot and configuration sectors.

CLOAD

Copy a file from cassette to disc.

CSAVE

Copy file from disc to cassette.

CHKDISC

Compare two discs (two drives).

COPYDISC

Copy one disc to another (two drives).

DISCCHK

Compare two discs (one drive).

DISCCOPY

Copy one disc to another (one drive).

FILECOPY

Copy files from one disc to another (one drive).

FORMAT

Format a disc.

SETUP

Changes configuration sectors.

All transient program files are executable programs and thus have the filetype .COM.

Certain programs (such as MOVCPM and SYSGEN) have been extensively modified by Amstrad for the CPCs.

## Transient program area (TPA)

See TPA.

## TTY:

The physical device known in standard CP/M terminology as the teletype device (slow speed console), to which can be assigned the CON: RDR: PUN: and LST: logical devices. In CPC CP/M the TTY: represents the special I/O device 0.

## TYPE

A CP/M built-in command which displays the content of an ASCII file on the console device (that is, CONSOLE or CON:). If the file is not in an ASCII form then the output can be rather strange. It is possible (by using **[CTRL]P**) to copy the output to the printer, as well as using PIP. The TYPE FILE.TXT command is similar in operation to PIP CON:=FILE.TXT (see PIP).

## UC1:

A physical device name which means user-defined console in standard CP/M, a device to which the CON: logical device may be assigned. In CPC CP/M the name means the special I/O device 1.

## UL1:

A physical device name which means user-defined list device 1 in standard CP/M, a device to which the LST: logical device may be assigned. In CPC CP/M the name means the special I/O device 1.

## UP1:

A physical device name which means user-defined punch 1 in standard CP/M, a device to which the PUN: logical device may be assigned. In CPC CP/M the name means the special I/O device 1.

## UP2:

A physical device name which means user-defined punch 2 in standard CP/M, a device to which the CON: logical device may be assigned. In CPC CP/M the name means the screen.

## UR1:

A physical device name which means user-defined reader 1 in standard CP/M, a device to which the RDR: logical device may be assigned. In CPC CP/M the name means the special I/O device 1.

## UR2:

A physical device name which means user-defined reader 2 in standard CP/M, a device to which the RDR: logical device may be assigned. In CPC CP/M the name means the keyboard.

## USER

This is a built-in command to allow the disc files to be organized into distinct and discrete sections, where each section is given a number from 0 to 15. On a cold boot the user is automatically in user area 0. If a change is made to the user area in operation, then this is maintained until the next `USER` command or until the session is complete - the CPC is switched off, CP/M is exited, or there is a cold boot. A warm boot does not alter the current user area. See USR:

## USR:

In conjunction with the transient program STAT, gives the number of the currently active USER area, and the number of currently active files. The default at cold boot is

```
A>STAT USR:

Active User : Ø
Active Files: Ø
A>USER 1
A>STAT USR:
STAT?

A>DIR
NO FILE
A>
```

Which indicates that when a change is made to the USER area, none of the old files are available for use.

## VAL:

Used as a parameter with the transient program STAT, gives a list of the possible alterations to the status of devices and the disc, which can be modified by use of STAT with other parameters.

## Vendor disc format

Is one of three Amstrad disc formats (the other two are IBM PC format and the Data Only formats). This is closely allied to system disc format.

With the system disc format 38 tracks are available for storage of data (with 9 sectors per track numbered #41..#49). Two tracks are reserved for CP/M system use. This format is slightly altered for vendor format (which cannot be warm booted), because with vendor format these two tracks are set aside for later loading of the CP/M system tracks (by use of the Amstrad utility transient program BOOTGEN and SYSGEN).

See also Disc Format.

## Virtual memory

An arrangement where not all a program or data set is kept in memory, but where parts of the program/data are kept on a disc file so that when a portion is needed, this portion is loaded from disc (and possibly the old portion written back to disc). In this way, the memory available to the user is not restricted to the available RAM.

Virtual memory techniques have been implemented for large programs (Locomotive BASIC has a modest virtual facility in that programs can be chained, and old values of variables not be lost); virtual memory techniques have also been used to allow larger data files to be used in (say) word processing (see swap file, .SWP).

## Warm boot

A warm boot is occasioned by entering **[CTRL]C**, and results in a jump to the BIOS jumblock routine WBOOT. A warm boot reloads the BDOS and CCP. See cold boot.

## Wildcards (in file designations)

A wildcard is a symbol which replaces all or some examples of a certain part of a file name. Wildcards are * and ?.

## XPB (Extended disc parameter block)

The XPB contains the standard CP/M DPB, with additions to optimize to the discs used by the CPC, and to the different formats available. There is one XPB per drive, stored in RAM.

## XSUB

The `XSUB` command must be included as the first command of a `.SUB` batch file (to run under the transient program `SUBMIT`). The use of `XSUB` allows command lines within programs to be given within the batch file.

For example, the sequence of commands

```
DDT
GØ
```

in a file TEXT.SUB, when called by

```
A>SUBMIT TEXT

A>DDT
DDT VERS 2.2
-
```

and the system hangs around and waits, because the line GØ is not recognized within the DDT program (exit by entering GØ or **[CTRL]C**), then the extra command GØ is misinterpreted by CP/M. The addition of a command line within the batch program which involves the XSUB command remedies the situation

```
XSUB
DDT
GØ
```

produces the outcome

```
A>SUBMIT TEXT

A>XSUB

A>DDT
DDT VERS 2.2
-GØ

(xsub active)
A>
```

In this case the line GØ is interpreted by DDT as a command, and thus the program ends.

## Z80 processor

The Zilog Z80 microprocessor, see Microprocessors.

## [CTRL]C

Warm boot of CP/M, when pressed at the beginning of a line.

## [CTRL]E

Physical end of line: carriage is returned, but the line is not sent to the system until **[ENTER]** is pressed.

## [CTRL]H

Erase previous character.

## [CTRL]I

Tab, that is, move eight characters.

## [CTRL]J

Line feed.

## [CTRL]M

Carriage return.

## [CTRL]P

Copies output on the CON: device (the screen) to the current list device (LST:), output is sent to both devices until the next **[CTRL]P** is entered.

## [CTRL]R

Retypes current line (a useful facility on teletypes when there was a good deal of alteration, not a great deal of use on the CPC).

## [CTRL]S

Stops output on the screen (that is, stops scrolling). Stops execution of the program. Execution and scrolling commence when another key is pressed.

## [CTRL]U

Deletes line, that is, puts a hash against the characters on that line, and moves to a new line.

## [CTRL]X

Deletes line, that is, line is erased.

## [CTRL]Z

If used within command level CP/M, selects a window (see Amstrad documentation concerning control codes), but within certain programs (such as `PIP` and `ED`), indicates the end of a line or file (depending on context).

## |CPM

To cold boot CP/M from within BASIC, one enters this command.

## |DIR

Provides a directory of the disc from within the BASIC environment (equivalent to `CAT` within BASIC).

## |ERA

Allows erasure of files from within the BASIC environment (compare with, `ERA` command in CP/M).

## |REN

Allows files to be renamed from within the BASIC environment (compare with, `REN` command in CP/M). An enhancement to the normal `REN`, this allows user numbers to be specified.