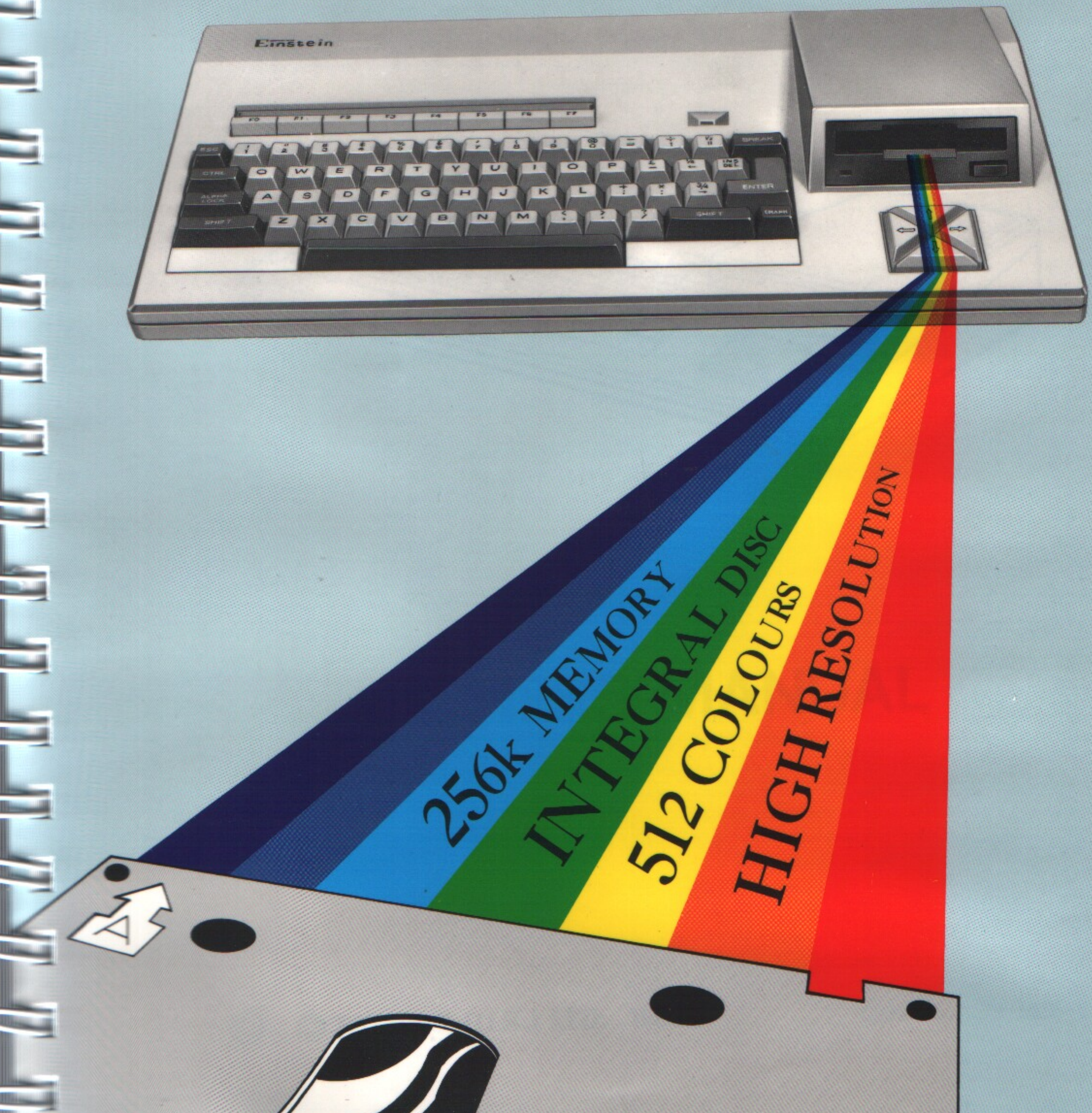
TATUNG

Einstein 256

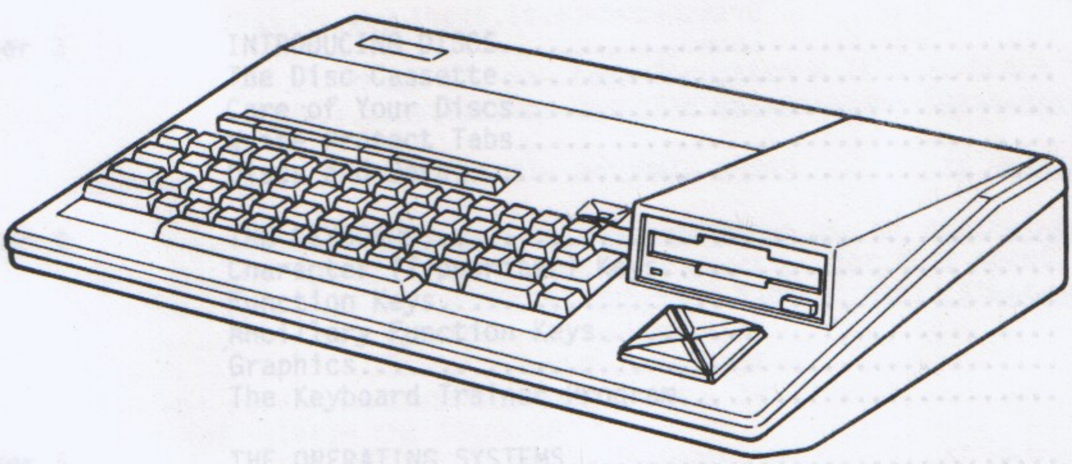
512 COLOUR MICRO - COMPUTER

OWNERS MANUAL



TATUNG

Einstein 256



OWNERS MANUAL

© TATUNG (U.K.) LTD. 1986

Code No. 79-1056-8

CONTENTS

INTRODUCTION

Please READ ME

This handbook is intended to cover the setting up of the EINSTEIN 256 computer, and to introduce you to its various features.

It is divided into sections and chapters, at the end of each chapter is a summary, which will remind you of the key items.

No technical handbook ever made very entertaining reading, but we hope and suggest that you will at least read some of it whilst drinking a cup of tea, and getting used to the idea of owning this computer.

There is a lot of information within these covers, some you may know, some you may not. It is a good idea to read each section and make a mental note of the key points (that is why there is a summary).

For the real beginner, and those who may have forgotten a little, there is a section on what is inside a computer, by way of simple explanations.

NOTE: Conventions.

1. You will often see some instructions requiring the use of special keys, like CTRL & BREAK. In case you do not know exactly what that means, and to save you the time and effort of typing C T R L, etc., it is done thus :-

   a. Press and hold down the CTRL key
   b. Press, and release the BREAK key
   c. Release the CTRL key

   This procedure also holds good for "SHIFT-BREAK", but this time you can use either SHIFT key. (They are identical).

2. Upper case functions, like the capital letters, are engaged by pressing the SHIFT and whatever letter you want. You can stay in 'Upper Case' letters by pressing the 'Alpha Lock' key. On the numeric keys (1-0) and several others, you will see two sets of symbols. The upper one is obtained by pressing the shift key and the symbol you need.

3. In this book, lots of examples are given. You are advised to try them, experiment with them, invent your own variations: in short, get to know your Machine. Every time you press a key, the data goes into what is called a 'buffer'. There it will stay until ɟou do something else, like pressing "ENTER".

4. For the rest of this book, we will assume that you will press the ENTER key. (We will give you a prompt or two, first). Similarly, in BASIC you will need to clear an old program out of memory. This is done by typing **N E W** and then ENTER. We will leave that as "NEW". Clearing the screen can be done two ways; typing CLS followed by ENTER, (or CTRL-L). From now on we will just say clear the screen, or CLS.

For The Impatient:-

1. Connect up as illustrated in Fig. 0.1 below according to your system components. Switch on at the mains plug. (Important Note: Refer to the manufacturers instructions before connecting any equipment to the mains supply).

2. Insert your Master Disc. (See Fig 0.2).

3. Press CTRL - BREAK

4. Select the program from the menu on the screen. (Or key DIR ENTER to view what is on the disc, and then select your program (On Basic).

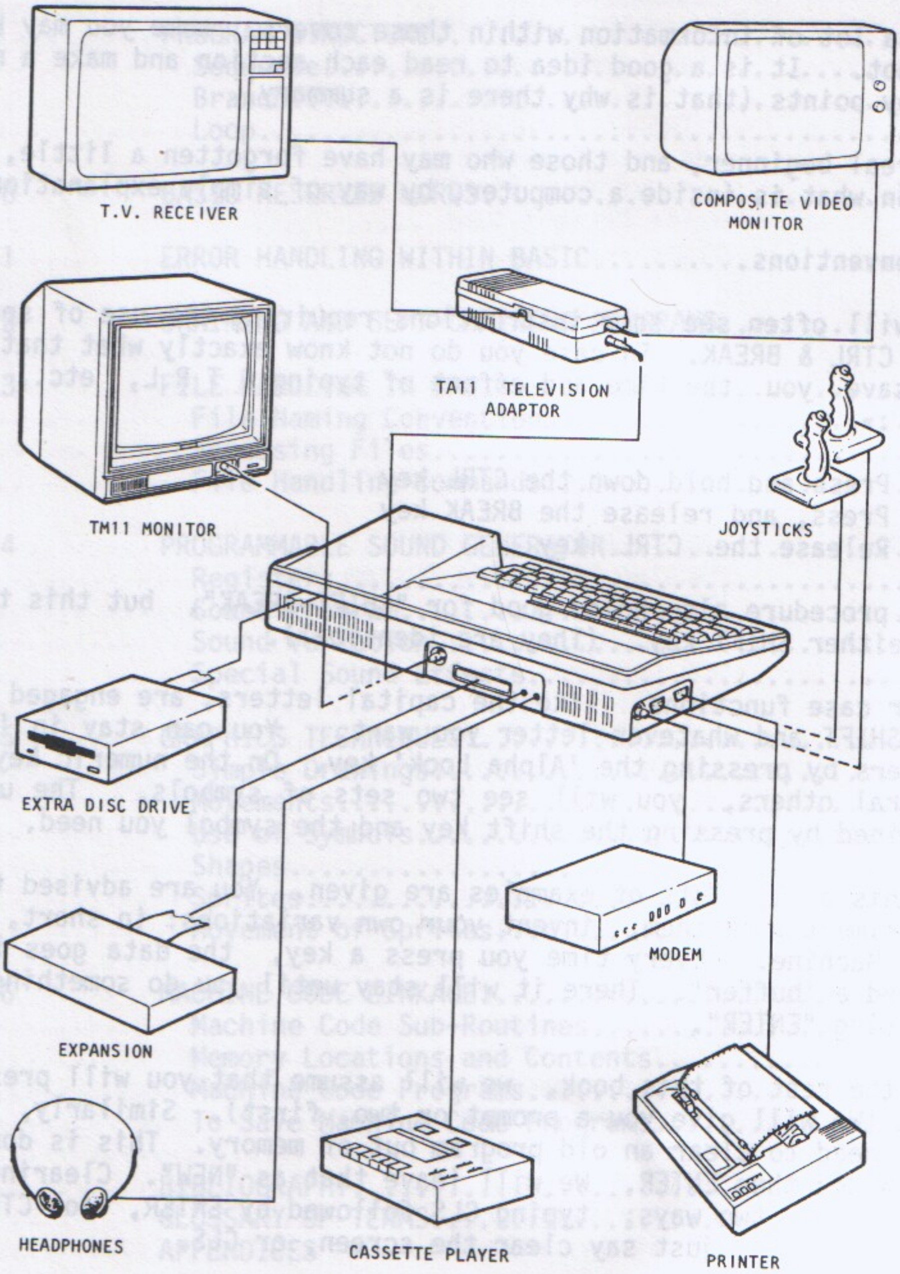


Fig 0.1. System component connections

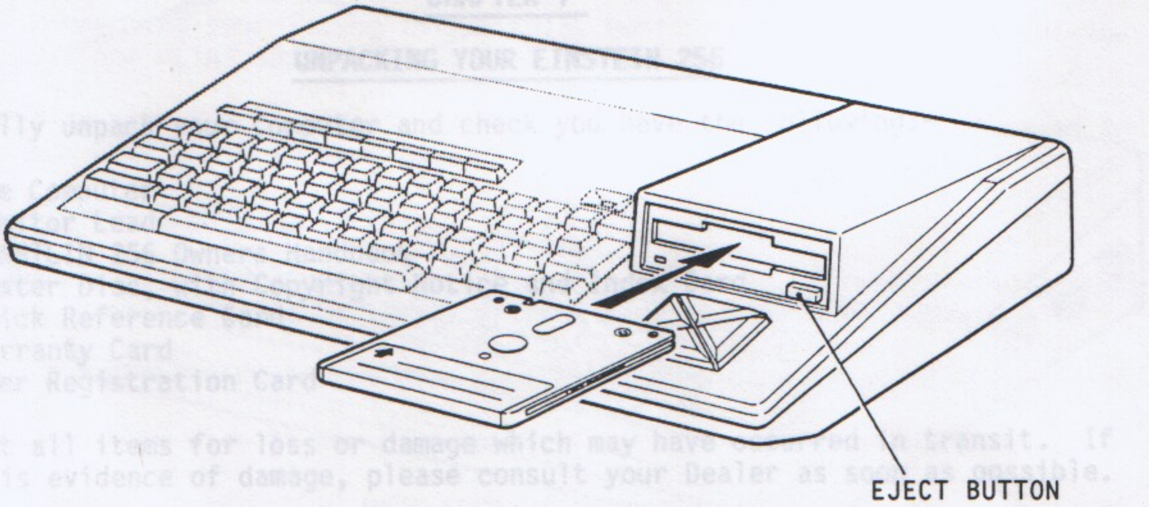


Fig 0.2. Loading a disc.

CHAPTER 1

UNPACKING YOUR EINSTEIN 256

Carefully unpack your Computer and check you have the following:-

1. The Computer
2. Monitor Lead
3. EINSTEIN 256 Owners Handbook
4. Master Disc, with Copyright Notice and Index Card
5. Quick Reference Card
6. Warranty Card
7. User Registration Card

Inspect all items for loss or damage which may have occurred in transit. If there is evidence of damage, please consult your Dealer as soon as possible.

There are several options available for your EINSTEIN 256:-

1. TM11 Colour Monitor,
2. TA11 Television Adaptor,
3. Second Disc Drive (available from you dealer).

CONNECTIONS AND FEATURES ON YOUR EINSTEIN 256 (Fig 1.1)

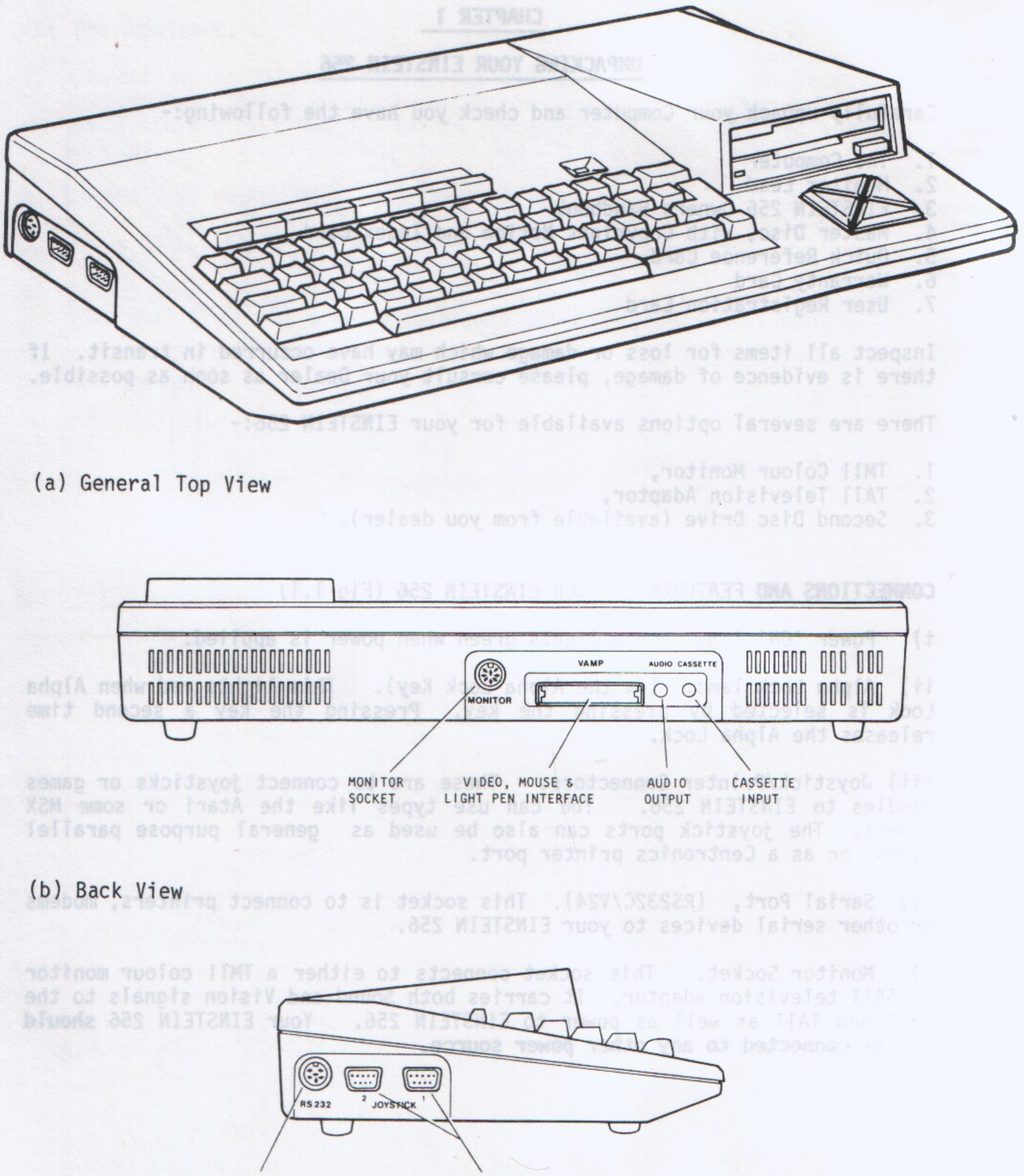
i) Power 'ON' lamp. This lights green when power is applied.

ii) Alpha Lock lamp, (on the Alpha Lock Key). This lights red when Alpha Lock is selected by pressing the key. Pressing the key a second time releases the Alpha Lock.

iii) Joystick/Printer Connectors. These are to connect joysticks or games paddles to EINSTEIN 256. You can use types like the Atari or some MSX models. The joystick ports can also be used as general purpose parallel ports, or as a Centronics printer port.

iv) Serial Port, (RS232C/V24). This socket is to connect printers, modems or other serial devices to your EINSTEIN 256.

v) Monitor Socket. This socket connects to either a TM11 colour monitor or TA11 television adaptor. It carries both Sound and Vision signals to the TM11 and TA11 as well as power to EINSTEIN 256. Your EINSTEIN 256 **should not be connected to any other power source.**



(a) General Top View

(b) Back View

(c) Side View

Fig 1.1 Looking at your machine

ELECTRICAL SAFETY

Before connecting your Tatung TM11 Display Monitor or Tatung TA11 television adaptor to the mains supply please read the following:

You will need to fit a suitable plug to the mains lead which is coloured in accordance with the following code:

BLUE = NEUTRAL ; BROWN = LIVE

As the colours of the flexible mains lead may not correspond with the markings identifying the terminals of your plug, CONNECT AS FOLLOWS:

Connect the BROWN wire to the plug terminal marked L or coloured RED.

Connect the BLUE wire to the plug terminal marked N or coloured BLACK.

If a 13 Amp plug is fitted make sure it is protected by a 5 amp fuse. If this fuse needs replacing use only one of the same amp rating and ASTA approved to BS1362.

If any other type of plug is used protect with a 5 amp fuse, or with a 5 amp fuse wire in the adaptor or distribution board.

IMPORTANT NOTE: when using a TV receiver or composite video monitor in conjunction with the TA11 television adaptor refer to the manufacturers instructions for connecting to the mains. EINSTEIN 256 must not be connected directly to the mains. It must only be used with its matching TM11 monitor or TA11 television adaptor.

PRECAUTIONS

It is important that all ventilation slots are clear of any obstruction, so that a free flow of air is maintained. Do not place the equipment on or near a source of heat, (even the window on a hot day). Always stand the equipment on a firm flat surface. Do not stand it upon upholstered surfaces or soft furnishings as these may obstruct the air flow.

Warning: Do not expose to any rain or moisture. If this happens, remove the supply plugs, and have the unit checked by your dealer.

Should any unit fail after switching on, switch off at the mains point. If you are using a TV, plug in the normal aerial lead, and tune to a TV station. If you still have a problem, talk to your TV repair man. If the TV works, take your unit to the dealer. Do not operate any faulty unit, it could cause more damage. If you are in any doubt, talk to your dealer.

Cleaning: The computer, TA11 and monitor cabinets may be periodically cleaned with a soft, damp cloth. Do not use proprietory cleaning agents which could damage the surface of the unit. The screen of the monitor should be cleaned in a like manner.

Connecting Up Your EINSTEIN 256

1. Using a Tatung TM11 Colour Monitor

Ensure that the TM11 is switched off, and connect up the monitor cable between the socket on the TM11 and the monitor socket on the Einstein 256, as shown in Fig 1.2. Connect the mains lead of the TM11 to the standard domestic mains supply.

The TM11 provides all the necessary power for Einstein 256. Switching on the TM11, also switches on the computer.

Important Note: On no account connect Einstein 256 directly to the mains supply.

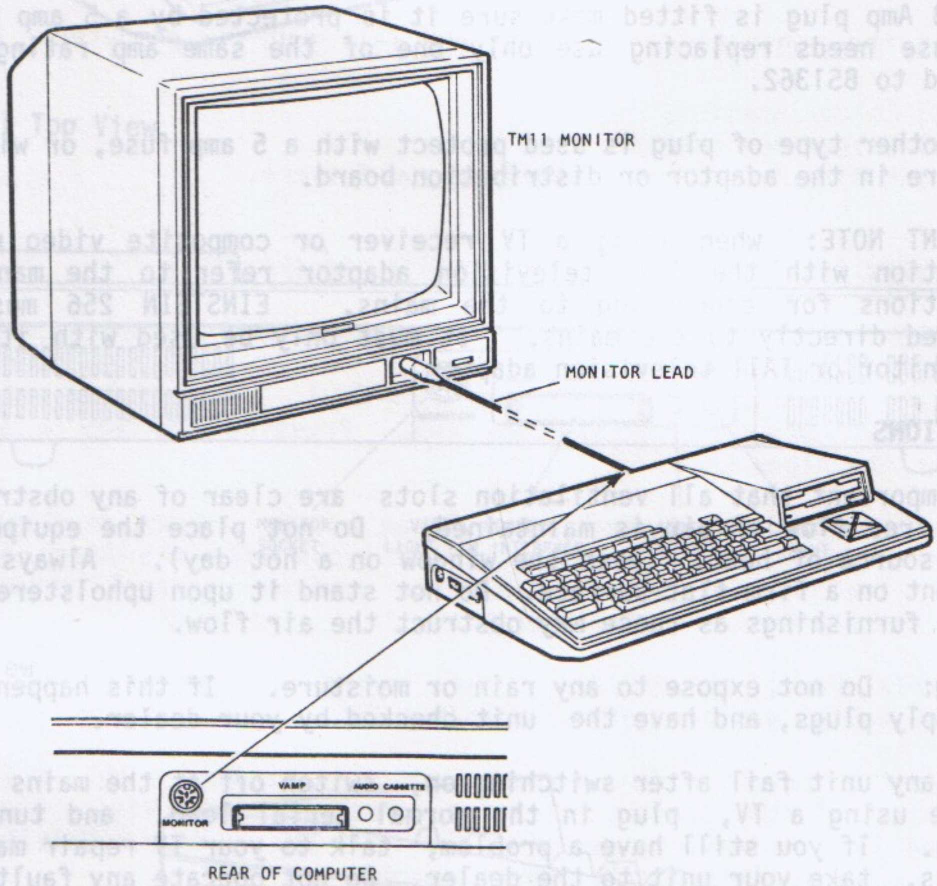


Fig 1.2. Using a Tatung TM11 Monitor

Adjusting the TM11

The TM11 has two controls, brightness and volume. Once the TM11 has been swtiched on, allow a few seconds for the crt to warm up and adjust the brightness control for an adequate display. For best results, the display brightness should be kept as low as possible, consistent with ambient lighting. The volume control should be adjusted for a comfortable listening level.

2. Using a Television Adaptor, TA11: (See Fig 1.3)

a) With a television receiver:
The sound and vision signals generated by the computer are converted to television signals, suitable for use on a domestic television receiver. In turn, the TA11 provides the power needed for the computer.

There are three versions of the TA11, suitable for use in various countries. These are:-

TA11 240V, PAL system I, suitable for use in United Kingdom (UHF, channel 36).

TA11A, 240V, PAL system B/G, suitable for Europe (UHF channel 36).

TA11X 110V, NTSC, suitable for far Eastern countries (VHF channel 13).

Connect the monitor lead between the TA11 and the computer, as shown in Fig 1.3. Connect the mains plug to the domestic mains supply.

Connect the antenna (aerial) socket of your TV to the TV outlet on the TA11.

Switch on both TA11 and your T.V. Tune the receiver to the appropriate channel (UHF ch 36 for TA11 and TA11A; or VHF ch 13 for TA11X). Adjust the tuning until the Einstein 256 logo appears, in colour, on the screen. (You may need to refer to the operating instructions for your particular television receiver).

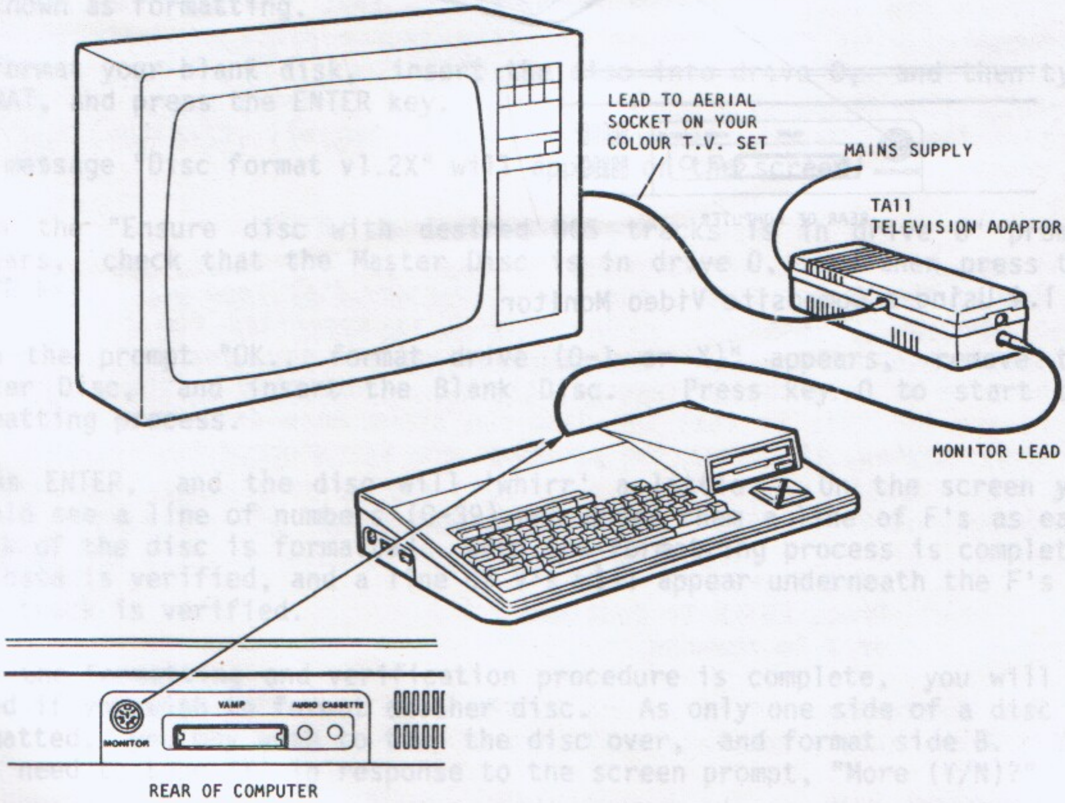


Fig 1.3 Using a Television Receiver.

b) <u>With a composite video monitor:</u>

The TA11 television adaptor provides a standard composite video output, suitable for connecting to a monochrome or NTSC colour monitor. The sound output is available from the STEREO socket, should your monitor be equipped with sound. You may prefer to listen to the sound with a pair of stereo headphones. The headphones should be fitted with a standard 3.5mm miniature stereo jack plug. Connect up as shown in Fig 1.4.

For details on how to connect and adjust your composite video monitor, refer to the manufacturers operating instructions.

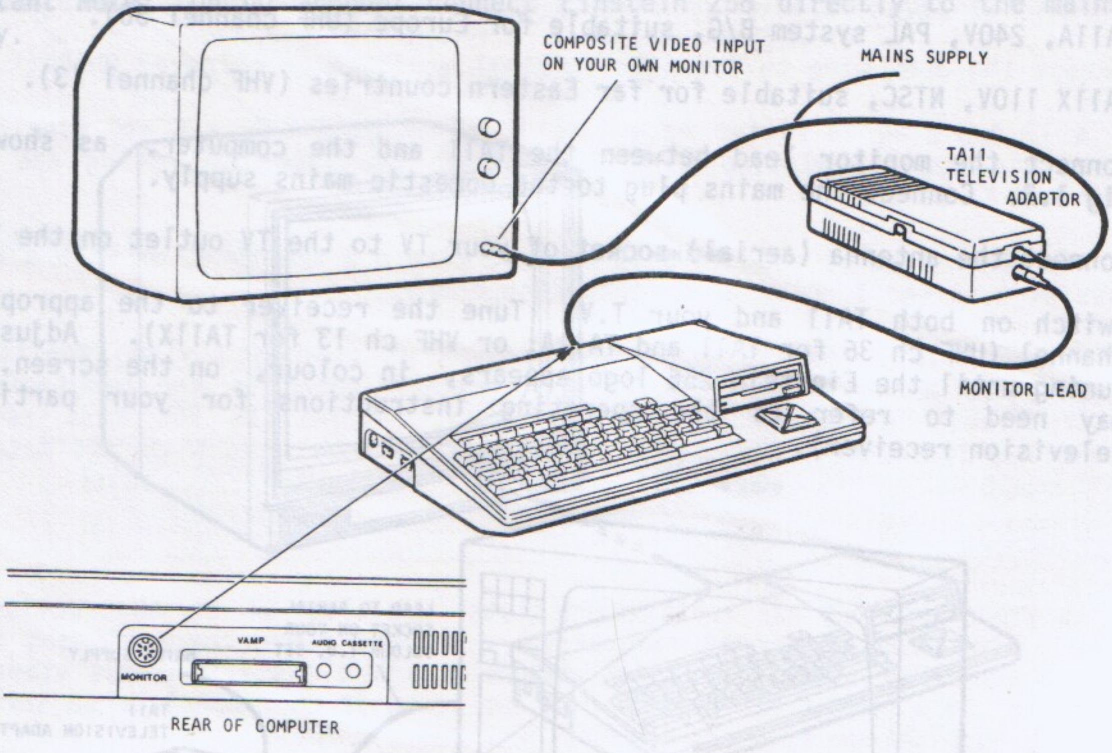


Fig 1.4 Using a Composite Video Monitor

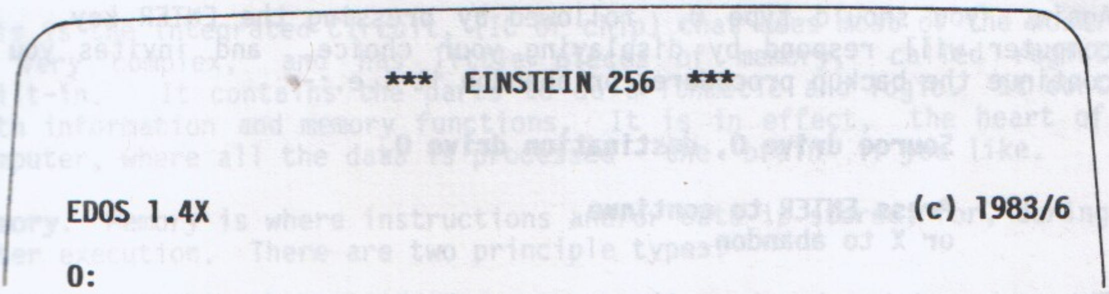
MAKING A BACKUP, OR SECURITY COPY.

You are advised to make a security copy of the Master Disc. (You will need a spare, unused, disc).

a) With no disc inserted, connect up your system and switch on. Ensure that the "Write Protect" tab on the System Master Disc (See Fig.3.1) is set to the "protected" position.

b) Insert the Master Disc. Press and hold the CTRL key; press and release the BREAK key; release the CTRL key. The disc light will illuminate briefly.

The screen should clear and a message appear on the screen:-



```
              *** EINSTEIN 256 ***

EDOS 1.4X                              (c) 1983/6

0:
```

c) You may need to intialise a blank disc. The surface of the disc needs to be prepared to accept data. The process of preparation / initialisation is known as formatting.

To format your blank disk, insert the disc into drive 0, and then type FORMAT, and press the ENTER key.

The message "Disc format v1.2X" will appear on the screen.

d) When the "Ensure disc with desired DOS tracks is in drive 0" prompt appears, check that the Master Disc is in drive 0, and then press the ENTER key.

When the prompt "OK....format drive (0-1 or X)" appears, remove the Master Disc, and insert the Blank Disc. Press key 0 to start the formatting process.

e) Press ENTER, and the disc will 'whirr' a little. On the screen you should see a line of numbers (0-39) and under them a line of F's as each track of the disc is formatted. When the formatting process is complete, the data is verified, and a line of V's will appear underneath the F's as each track is verified.

f) When the formatting and verification procedure is complete, you will be asked if you wish to format another disc. As only one side of a disc is formatted, you may wish to turn the disc over, and format side B. You will need to type 'Y' in response to the screen prompt, "More (Y/N)?"

g) When you have formatted both sides of the blank disc, type 'N' in response to the screen prompt. The system will be returned to the control of the disc operating system (DOS).

h) Before proceeding any further, ensure that your Master Disc is write protected. Refer to Chapter 3.

Type BACKUP, and then press the ENTER key. The screen will then display the message:-

Source Drive (0-1 or X)?

You should type 0, then press the ENTER key. Another message will be displayed on the screen:-

Destination Drive (0-1 or X)?

i) Again, you should type 0, followed by pressing the ENTER key. The computer will respond by displaying your choice, and invites you to continue the backup procedure, or abandon it i.e.:-

Source drive 0, destination drive 0,

Press ENTER to continue
or X to abandon.

The source drive is where the programs to be transferred are, and the destination drive is where the programs will be transferred to.

Before pressing the ENTER key, ensure that the original master disc supplied with your Einstein 256 (the source disc) is inserted into the drive (drive 0). Then press the ENTER key. The drive 'in use' light will illuminate, indicating that the disc is being read. After a short time the light will go out, and the screen will display the following message:-

Put In Destination Disc And Press ENTER:

j) Remove the Master disc, and insert the new disc, and proceed as directed by the BACKUP utility. As drive 0 is specified for both source and destination, the master (source) disc and the new (destination) disc will have to be interchanged several times, as directed by the program. The reason for this is that the disc can store more data than the memory in the computer, therefore the programs are transferred in "chunks".

When the backup process is complete, the screen will display:-

Disc Backed up -- OK
Press ENTER to continue
or X to abandon

You should key "X" to return to the disc operating system (DOS). You will now have a complete duplicate of your master disc.

IMPORTANT NOTE: The Backup utility will destroy any existing data or programs which may be on the destination disc, so please ensure that the destination disc does not contain anything you wish to keep!

CHAPTER 2

WHAT IS A COMPUTER?

If you have had a computer, or if you know something about them, then skip this chapter, for it is intended as a short, (and simple), guide to introduce you to some jargon.

Fact: Computers cannot of themselves think.
They are, therefore, only a way of doing a lot of tasks in a little time.

Stripped to the bare essentials they usually comprise:-

The **CPU** (Central Processing Unit)

This is the Integrated Circuit, (IC or chip) that does most of the work. It is very complex, and has little pieces of memory, called registers, built-in. It contains the parts to do arithmetic and logic. It controls both information and memory functions. It is in effect, the heart of the computer, where all the data is processed - the 'brain' if you like.

Memory. Memory is where instructions and/or data is stored, for, during and after execution. There are two principle types:-

ROM (Read only memory). Instructions/data of a permanent nature are stored in ROM. As the name implies, you cannot "write" to it. It is "non-volatile", i.e. the data is retained when the computer is switched off.

RAM (Random Access Memory). Data can be entered (WRITTEN) and retrieved (READ) from any location in the RAM. Most RAM, at least in the home computer, is "volatile", so data can only be handled whilst the supply is maintained; when power is lost, so is the data, irretrievably.

Input/Output.

This vital, and often ignored, part of the computer allows commands to be entered, and data sent to be displayed. It is the means whereby the CPU communicates with the outside world, other (peripheral) devices etc. Everything, except memory, is sent through the I/O Port. A large number of devices may often be handled through the I/O Port.

When something is typed in at the keyboard, the "Input Port" takes the 'offering' and presents it to the CPU. The output "port" takes data from the CPU and routes it to the chosen destination, which is often the screen, but may be the disc, or even a "Port" like the RS232C. - See Fig 1.1 (c).

Backing Stores.

It would be inconvenient , if not catastrophic, to lose all the Data in RAM in the event of a loss of power to the computer, so some form of storage system is needed. In some machines, a compact cassette is used, but EINSTEIN 256, uses a 3 inch compact floppy disc, which is protected by a plastic case. The physical operations of insertion and removal are very similar to those of a front-loading car cassette player. Data may be "saved" and "loaded" in exactly the same way, but much quicker, so data may be stored on the disc temporarily or more permanently. Further details of discs are given in Chapter 3.

Hardware and Software

The items in a computer that you can pick up are called "Hardware". System hardware is the computer, printer, display, external drive, etc. Inside the computer itself, the various integrated circuits are also called Hardware.

"Software" is a set of instructions which the CPU will execute. Software is often contained on disc, but some essentials, like the "Einstein" logo seen on the screen when you switch on, are stored in "Firmware", which is a set of instructions in ROM.

Peripherals and the System.

The Einstein 256 may be expanded to suit personal needs, requirements or taste. As can be seen in Fig 0.1, (Introduction), there are a number of options, including joysticks, printer, external disc, etc., as well as future expansion via the VAMP interface. There are input devices (joysticks for example), and output devices (e.g. a printer or the RS232C Serial Port).

The Display

The screen of your monitor/tv can display information in two ways:

i) As text (numbers, letters and symbols).
ii) As graphics.

1. When displaying text, the screen is organised into horizontal rows, and vertical columns, (See Fig 2.1). Einstein 256 can display text in 24 rows and 32, 40, 64, or 80 columns, according to the display mode. In text, the display origin (0,0) is at the top left hand of the screen.

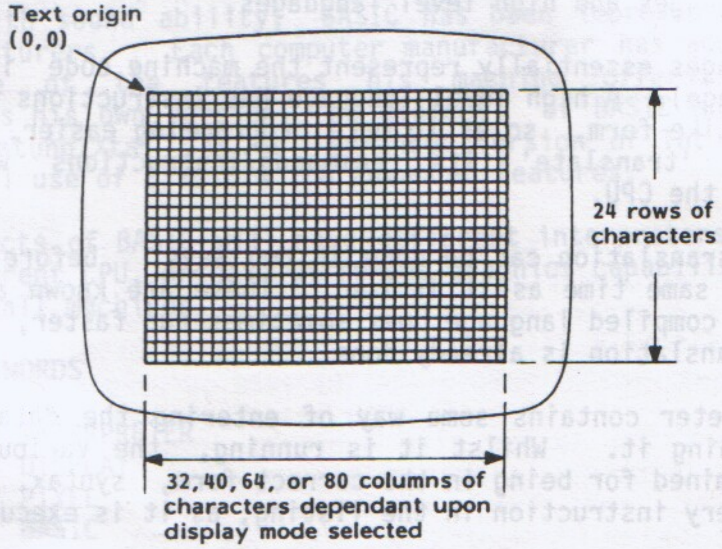

Text origin
(0,0)
24 rows of characters
32,40,64, or 80 columns of characters dependant upon display mode selected

Fig. 2.1 Screen - text modes

2. When displaying graphics, the screen is organised into 256 or 512 pixels horizontally, and up to 424 pixels vertically. (Note BASIC only supports 192 pixels vertically.) A pixel is the smallest picture element the computer can generate. The origin for graphics (0,0) is at the bottom left hand of the screen.

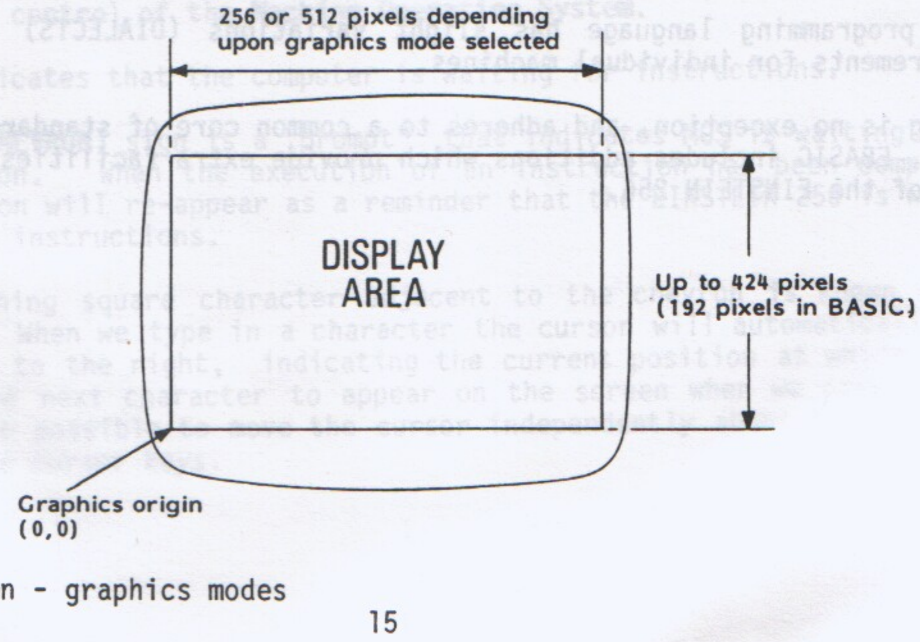


Fig. 2.2 Screen - graphics modes

COMMUNICATION

Until now, we have not asked the computer to execute any instructions for us. To do this, we need to communicate with it. The CPU in the computer only understands data in the form of binary digits, '1's and '0's. People communicate in a language, such as English or French. In order to allow people to communicate effectively with a computer, some means of "translating" to binary digits (machine code) is necessary.

There are two principal ways, or 'languages' used in computer communications - low level languages and high level languages.

Low level languages essentially represent the machine code in mnemonic form (assembly language). A high level language has instructions and commands in a near English like form, so as to make programming easier. A high level language will 'translate' its commands/instructions which can be 'understood' by the CPU.

The job of the translation can be done in two ways: before the program is run, or at the same time as it is run. These are known as Compiled or Interpreted. A compiled language does sometimes run faster, because all the hard work of translation is already done.

A BASIC Interpreter contains some way of entering the data, editing any errors and running it. Whilst it is running, the various words in the listing are examined for being in the correct form, syntax. This process is repeated for every instruction in the listing, as it is executed.

Here are a few examples:-

HIGH LEVEL	LOW LEVEL
BASIC	MACHINE CODE
COBOL	ASSEMBLER
FORTRAN	
PASCAL	
FORTH	

Each programming language has slight variations (DIALECTS) according to requirements for individual machines.

Tatung is no exception, and adheres to a common core of standard (DARTMOUTH) BASIC. EBASIC includes additions which provide extra facilities for you as a user of the EINSTEIN 256.

A Note about BASIC

BASIC is an acronym for Beginners All-purpose Symbolic Instruction Code. BASIC is the most popular computer language. It is easy to learn, but yet powerful enough to handle many applications.

It was developed in 1964 at the Dartmouth Naval College in the U.S.A., where it was used as an introductory language for programmers using a time-shared computer. These programmers never actually saw the computer, only a tele-type (Telex) machine. Displays such as EINSTEIN 256 can provide were only a pipe-dream to most programmers at that time.

With the advent of sophisticated computers, with equally sophisticated displays and with sound ability, BASIC has been improved and expanded by several manufacturers. Each computer manufacturer has added to BASIC to take advantage of the features his machine offers. Thus each manufacturer has his own version, or dialect, of BASIC and Tatung are no exception. Tatung/Xtal is an enhanced version of Tatung/Xtal BASIC 4 which makes full use of EINSTEIN 256 advanced features.

Different dialects of BASIC will have different interpreters, because they may use a different CPU, or have different graphics capabilities, so not all BASIC's fit on all machines.

SUMMARY OF KEY WORDS

CPU
RAM
ROM
INTERPRETER
DISPLAY
PIXEL
BASIC

Having set up the computer and display as described earlier, the initial message displayed on the screen is explained below.

THE MESSAGE

a) The title at the top of the screen is the computer's own name followed by the instructions to load the system disc.

b) The message, TATUNG/XTAL MOS 2.XX indicates that the computer is working under the control of the **Machine Operating System.**

c) **READY** indicates that the computer is waiting for instructions.

d) The **> (CHEVRON)** sign is a "prompt" that indicates MOS is waiting for an instruction. When the execution of an instruction has been completed, the chevron will re-appear as a reminder that the EINSTEIN 256 is waiting for fresh instructions.

e) The flashing square character adjacent to the chevron is known as the **CURSOR.** When we type in a character the cursor will automatically move one space to the right, indicating the current position at which we can expect the next character to appear on the screen when we press a key. It is also possible to move the cursor independently about the screen by use of the cursor keys.

THE OPERATING SYSTEMS

There are **three** different systems available with the EINSTEIN 256, these are:-

1. The MACHINE OPERATING SYSTEM - MOS

2. The DISC OPERATING SYSTEM - DOS

3. High level languages

EBASIC is the high level language supplied with the EINSTEIN 256. Other languages may be available from your dealer.

Various features exist within these systems, depending upon what is required.

MOS:

Programmed into EINSTEIN 256's read-only memory (ROM) is a control program called the machine operating system. The machine operating system is a machine language program and acts as a link between the various low-level functions that EINSTEIN 256 performs, such as printing a character to the screen, fetching a character from the keyboard, or plotting lines.

The machine operating system also supports some utilities which can be accessed via the keyboard. These utilities allow to examine, and alter the contents of memory, read from, and write to the disc, perform number base conversions and so on. Most of the time you will not need to use this particular feature, but programmers will find it useful to debug programs, and to write machine code programs.

EINSTEIN 256 automatically comes up in MOS when switched on without a disc in the drive. When in DOS or BASIC, typing MOS will return control back to MOS.

The prompt message displays "Ready" with a chevron below when in MOS, i.e:-

```
TATUNG/Xtal MOS 2.XX
Ready
>
```

DOS:

The DOS concerns itself with accessing programs and data on the Discs. There are limited facilities available within this mode but more often it is used in conjunction with the particular language or program in operation within the machine. It can be thought of as a file manager.

Inexperienced users need not concern themselves too greatly with this at the moment as more use can be made of the DOS at a later stage when familiarisation of the machine is more complete. (More details of DOS are given in Chapter 5).

The prompt message displays the current drive number and a colon when in DOS:-

```
EDOS 1.xx
0:
```

HIGH LEVEL LANGUAGES:

i) Insert the MASTER DISC into Drive 0, and press CTRL-BREAK followed by "Enter" (The disc should be pressed firmly home in the same manner that you would with a front loading cassette tape player).

ii) The indicator light on the disc drive facia will illuminate.

(Pressing CTRL-BREAK causes the Disc Operating System to be loaded into the computer from the disc).

The computer should now be operating in DOS and the following heading should appear on the screen.

```
            *** EINSTEIN 256 ***

      EDOS V1.4X      (c)  1983/6

   0:
```

If this is not the case then check through the "loading" procedure again.

Now type EBAS and then press the ENTER key. This causes BASIC to be loaded into the computer from the disc.

The computer should now be operating in BASIC and the following heading should appear on the screen.

```
EBASIC V4.5X    1983/6
Size 41852
Ready
```

If this is not the case then check through the 'loading' procedure again.

The heading only appears when the language is first loaded and the figure given after "Size" will vary according to the version of BASIC installed. The screen format changes by necessity in accordance with the operations in hand. The heading will disappear at the first change of screen, leaving only the Ready prompt and cursor to indicate that the computer is working in BASIC.

SUMMARY

There are three modes of operation:
a) MACHINE OPERATING SYSTEM (MOS)
b) DISC OPERATING SYSTEM (DOS)
c) HIGH LEVEL LANGUAGE

CHAPTER 3

INTRODUCING DISCS

THE DISC CASSETTE

The disc is accessed by means of the window when loaded in the disc drive unit. This window has a protective metallic shutter which opens as the cassette is loaded, thereby exposing that area of the disc to the read/write head within the drive unit. Both sides of the disc can be used.

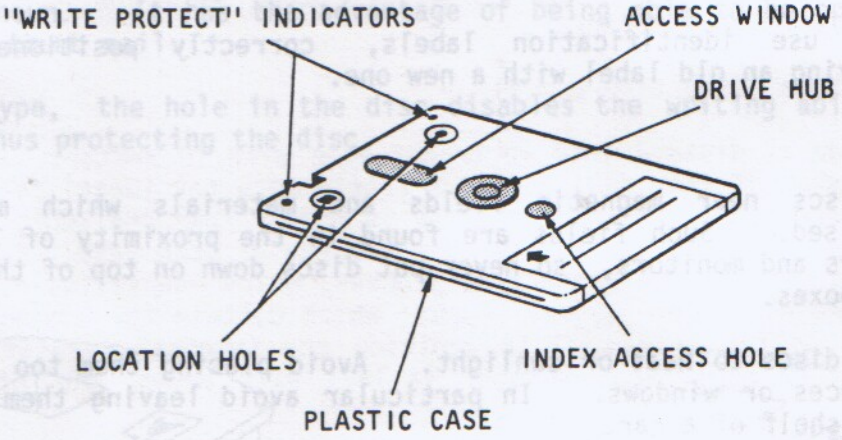


Fig.3.1 The important features of a disc.

NOTE:

When purchasing disc cassettes (often known as compact floppy discs), or extra disc drives, make sure they are marked with the symbol illustrated in Fig.3.2.

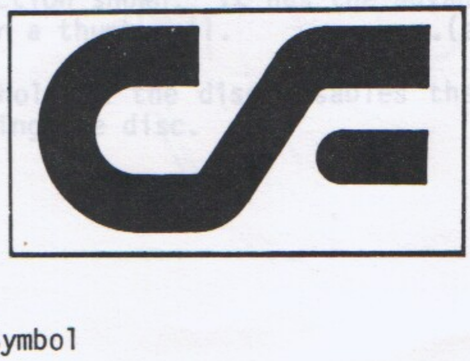


Fig 3.2 Compatibility Symbol

This logo indicates that any disc related materials bearing it are compatible with each other and with EINSTEIN 256. Beware of any attempt to supply disc related materials which do not bear this logo, as they may not function correctly (if at all) when used with your EINSTEIN 256.

CARE OF YOUR DISCS

Rough handling can cause damage to disc cassettes. The following precautions, DO's and DON'Ts, should be observed at all times.

DO.....

1...store discs in their boxes if possible, or in a polythene bag, when not in use.

2...always use identification labels, correctly positioned. Avoid overlaying an old label with a new one.

DON'T...

1...put discs near magnetic fields and materials which might become magnetised. Such fields are found in the proximity of TV sets, TV adaptors and monitors, so never put discs down on top of them, even in their boxes.

2...expose discs to heat or sunlight. Avoid placing them too near heating appliances or windows. In particular avoid leaving them on the rear parcel shelf of a car.

3...expose discs to moisture. As well as rain, high humidity can cause problems if a disc is brought from a cold atmosphere to a warmer one. If this is impossible, e.g. if the disc is brought into the house on a cold day, the disc must be allowed to attain the ambient temperature before use.

4...expose discs to excessive dust which might contaminate and cause damage. Cigarette ash is an obvious example.

5...expose the access area of the disc by opening the "window shutter" manually.

6...use erasers (rubbers).

WRITE PROTECT TABS

There are two write protect tabs labelled A and B, one for each side of a disc. When activated they protect the data on the disc from being overwritten, similar to the way that recordings on cassette tape can be protected. It allows information to be read from the disc but not written onto it, thus preventing accidental corruption or deletion of data.

The write protect tabs slide in a small housing at each corner of the cassette case as illustrated in Fig.3.3(a). If the tab obscures the hole, the write protect is NOT in operation. To activate the write protect, slide the red tab (using a small pointer, e.g. pencil) to the opposite end of the housing leaving the hole clear.

Another type of tab is illustrated. This is coloured white and slides in the direction shown. It has the advantage of being able to be operated by a finger or a thumb nail.

In either type, the hole in the disc disables the writing ability of the disc unit thus protecting the disc.

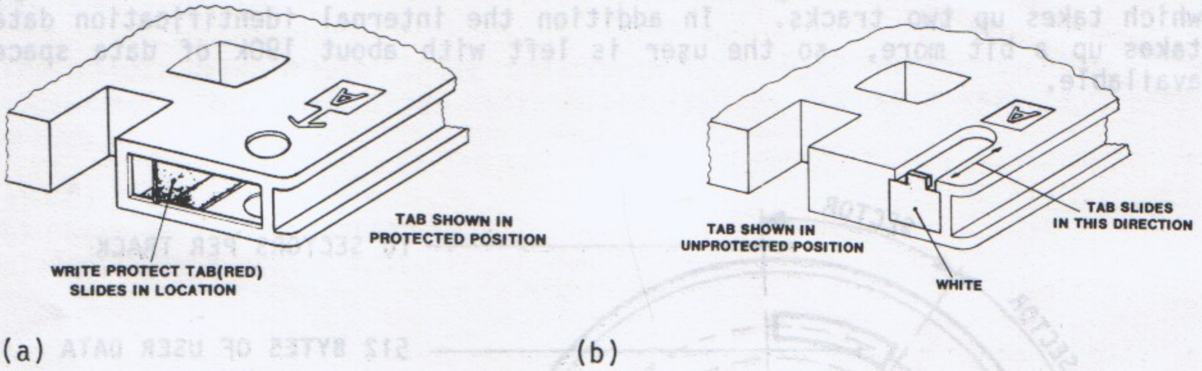


Fig 3.3 Alternative Write Protect Tabs

Another type of tab is illustrated in Fig 3.3(b). This is coloured white and slides in the direction shown. It has the advantage of being able to be operated by a finger or a thumb nail.

In either type, the hole in the disc disables the writing ability of the disc unit thus protecting the disc.

DISCS AND DATA

When data, (files, programs, or tables , for example), are 'saved' to a disc, the recording head moves to an appropriate place and leaves at that place a series of magnetic patterns on the surface of the disc, rather in the same way as audio signals are left on the tape. These patterns cause the data to be reproduced in the head when the disc is being read, so the process is two way.

The Tape storage system suffers from a lack of speed, due to two things. The effective speed of the tape against the head is slow, as is the transfer speed of data. In addition, data is stored on the tape sequentially, so you have to search from end to end to find a particular piece, or file, rather in the same way as you would look for a favourite track on an LP tape. A disk based system is much faster each file has a unique address (track X sector Y), in a similar way as a particular location in RAM, and the transfer rate is much faster.

Storage Of Data (See figure 3.4)

The disc side is divided into 40 tracks each with 10 sectors. Each sector holds 512 bytes of user data, plus a little more for identification, which is used internally. A disc's capacity is 250k - 'UNFORMATTED', but you will see that there is 190k is free for the storage of your data. The difference (60k) is taken up by the formatting process, which identifies each track and sector on the disc, the 'system tracks' which contain the DOS and directory, which takes up two tracks. In addition the internal identification data takes up a bit more, so the user is left with about 190k of data space available.

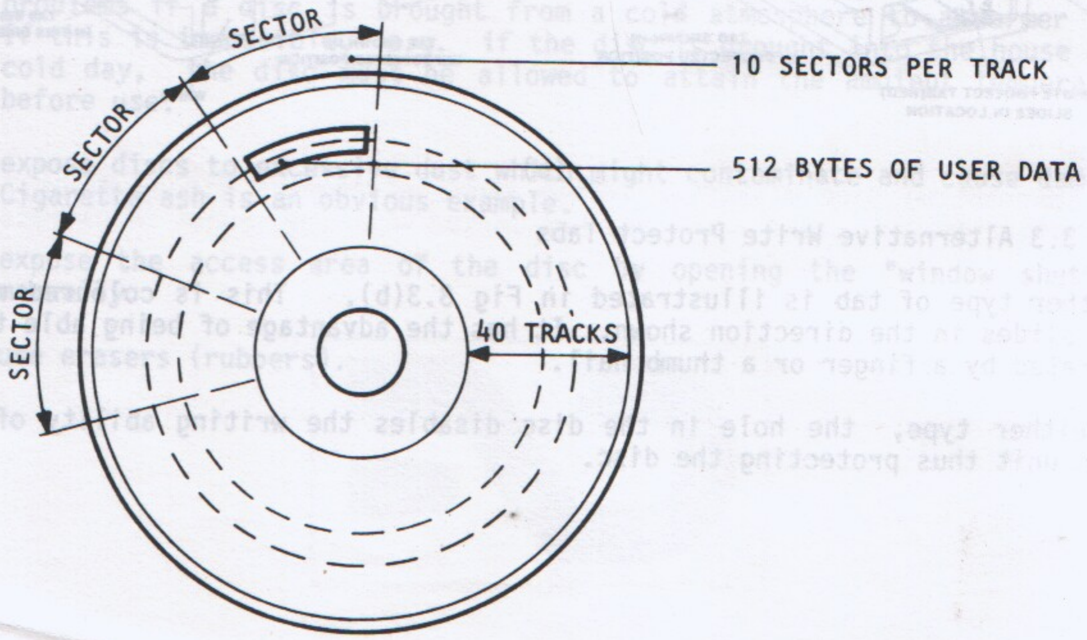


Fig 3.4 Disc Cassette Data Storage

SUMMARY

Characteristics of Discs
Precautions for use with Discs
WRITE-PROTECT TABS - Use
Storing data on Disc

CHAPTER 4

THE KEYBOARD

OVERALL LAYOUT

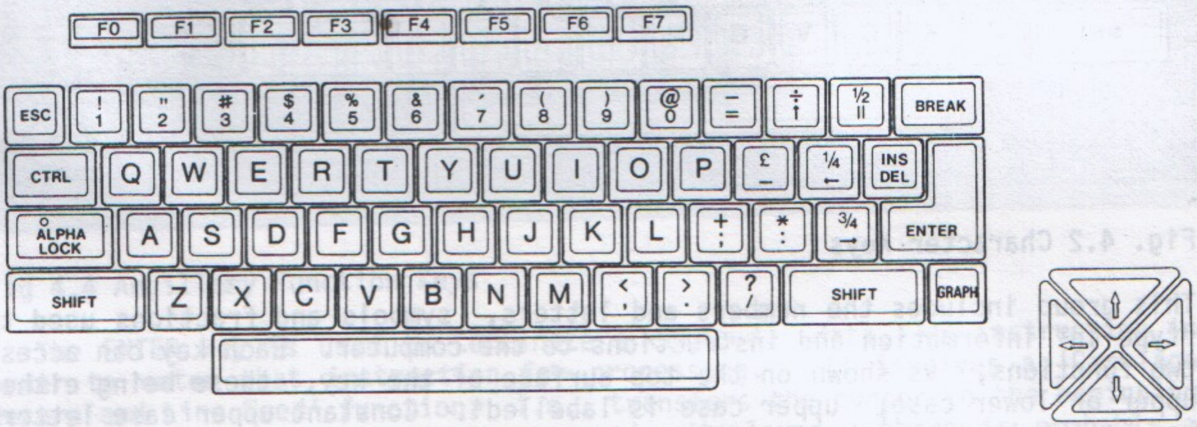

F0 F1 F2 F3 F4 F5 F6 F7
ESC ! 1 " 2 # 3 $ 4 % 5 & 6 ´ 7 (8) 9 @ 0 – = ÷ ↑ ½ ‖ BREAK
CTRL Q W E R T Y U I O P £ _ ¼ ← INS DEL
ALPHA LOCK A S D F G H J K L + ; * : ¾ → ENTER
SHIFT Z X C V B N M < , > . ? / SHIFT GRAPH

Fig 4.1 The Keyboard

The illustration above shows the keyboard, which resembles that of a typewriter, but with a few extra extra keys. The keys fall into three groups:-

a) Character keys.
b) "User defined" function keys.
c) Ancillary function keys.

CHARACTER (TYPEWRITER) KEYS

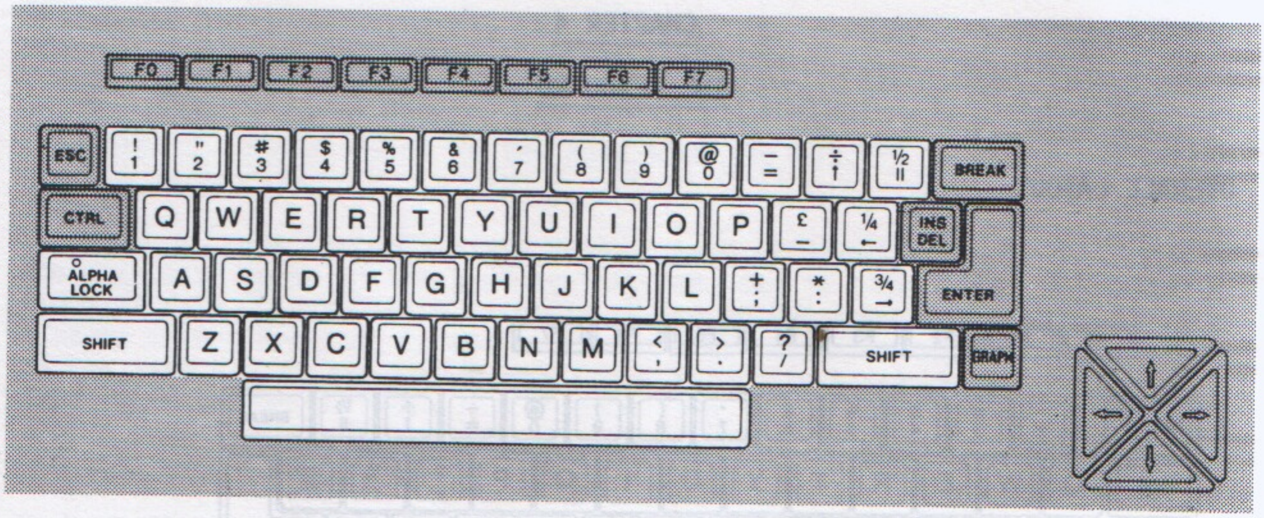

F0 F1 F2 F3 F4 F5 F6 F7
ESC 1 2 3 4 5 6 7 8 9 0 = ↑ ½ BREAK
CTRL Q W E R T Y U I O P £ ¼ INS DEL
ALPHA LOCK A S D F G H J K L ; : ¾ ENTER
SHIFT Z X C V B N M , . / SHIFT GRAPH

Fig. 4.2 Character Keys

This group includes the numbers and letters, symbols and fractions used to "type-in" information and instructions to the computer. Each key can access two functions, as shown on the top surface of the key, these being either upper or lower case; upper case is labelled*. Constant upper case letters may be obtained by using the "ALPHALOCK" key, but symbols are still selected by using the "SHIFT" key. The "ALPHALOCK" function is indicated by the built-in LED indicator; it is cancelled by pressing it a second time. A few moments practice will soon demonstrate.

* The selected case is accessed by first pressing, and holding, the "SHIFT" key, followed by the required key and then releasing both.

FUNCTION KEYS.

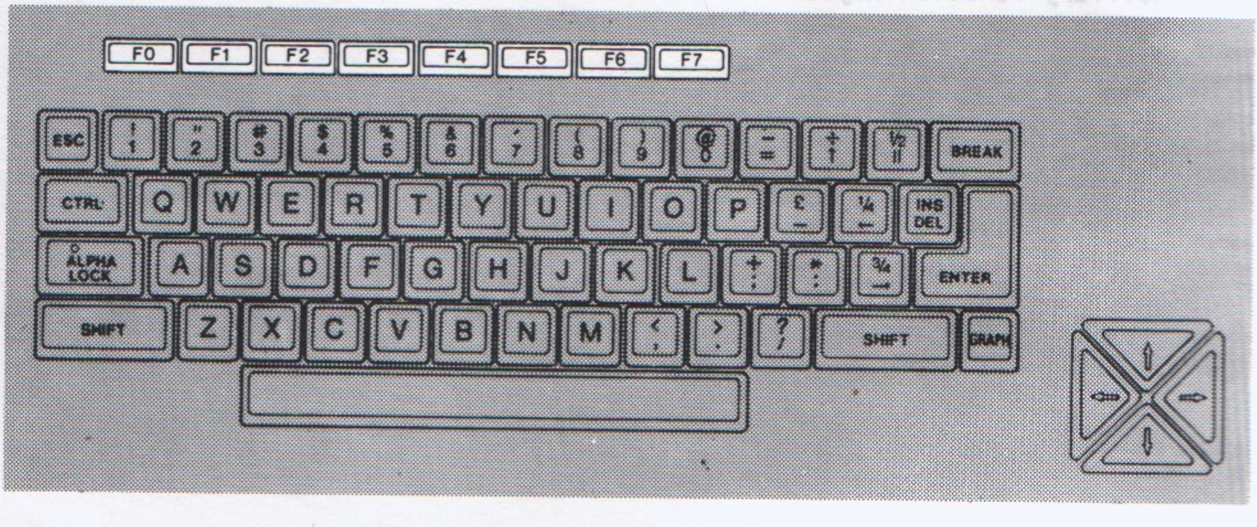

F0 F1 F2 F3 F4 F5 F6 F7
ESC 1 2 3 4 5 6 7 8 9 0 = ↑ ½ BREAK
CTRL Q W E R T Y U I O P £ ¼ INS DEL
ALPHA LOCK A S D F G H J K L ; : ¾ ENTER
SHIFT Z X C V B N M , . / SHIFT GRAPH

Fig 4.3 Function Keys - User Defined

The top row of eight keys, (F0 to F7), may be re-defined by the user. Details of reprogramming these keys is described later on in the section on EBASIC.

ANCILLARY FUNCTION KEYS

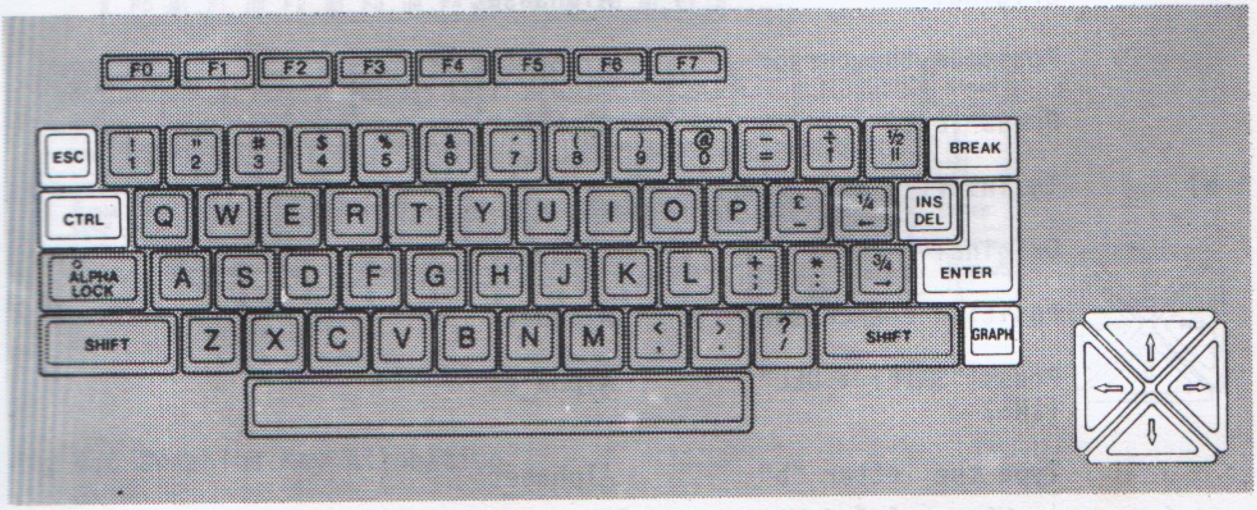


Fig 4.4 Ancillary Function Keys

1. The ENTER key is used to terminate a command, data, or instruction in order to enter that instruction for processing. It involves a 'Carriage Return and Line Feed' function, i.e., transfers the cursor to the beginning of a new line. If an error message is displayed, there is probably a mistake in the format or syntax. Later chapters will tell you what error messages exist and what to do with them.

2. The BREAK key halts execution of whatever process is executed, but only while it is pressed. Halting a BASIC program is accomplished by pressing 'SHIFT and BREAK' together, (SHIFT-BREAK). The program may be re-started by typing 'CONT' and keying ENTER. All variables are preserved, so you lose nothing.

3. Cursor Control. These four keys enable the movement of the cursor around the screen, for editing or games purposes. The cursor is moved in the direction of the arrows.

4. INS/DEL. This key provides the facility of inserting or deleting characters on the screen, relative to the cursor. The 'DEL' key is used 'unshifted'. The INS function is obtained by using DEL and SHIFT keys.

a) To delete a character:-

i) Use the cursor keys to position the cursor one place to the right of the character to be deleted.

ii) Press the DEL key.

iii) The character will be deleted, and text to the right will "Close up" from the right.

iv) Under some circumstances (e.g. in editing a BASIC listing), you must terminate the process by pressing the "ENTER" key, see the section on Editing in BASIC).

Example:-

	Original	Alphabset
	Position the cursor	Alphabset
	Press (DEL)	Alphabet
b)	To insert a character:-	
	Original	Alphaet
	Position the cursor	Alphaet
	Press "SHIFT + DEL" (INS)	Alpha_et
	Type the letter "b"	Alphabet
c)	To change a character:-	
	Original	Alphapet
	Position the Cursor	Alphapet
	Type new letter"b"	Alphabet

The character has been "overwritten".

5. CTRL

The control key is used in conjunction with other keys providing various useful facilities. Like the SHIFT key, it performs no direct function as an individual key, but creates a series of functions.

i) Press and hold the CTRL key.

ii) Press the desired second key.

iii) Release both.

6. ESC

The escape key provides a special abort function which stops listing or tabulation. Some software uses it to reset to a given state of the program.

7. Scrolling.

When a screen full of text has extra information typed in, the displayed text seems to vanish at the top of the screen, allowing more text to be entered at the bottom. This is called "scrolling".

GRAPHICS

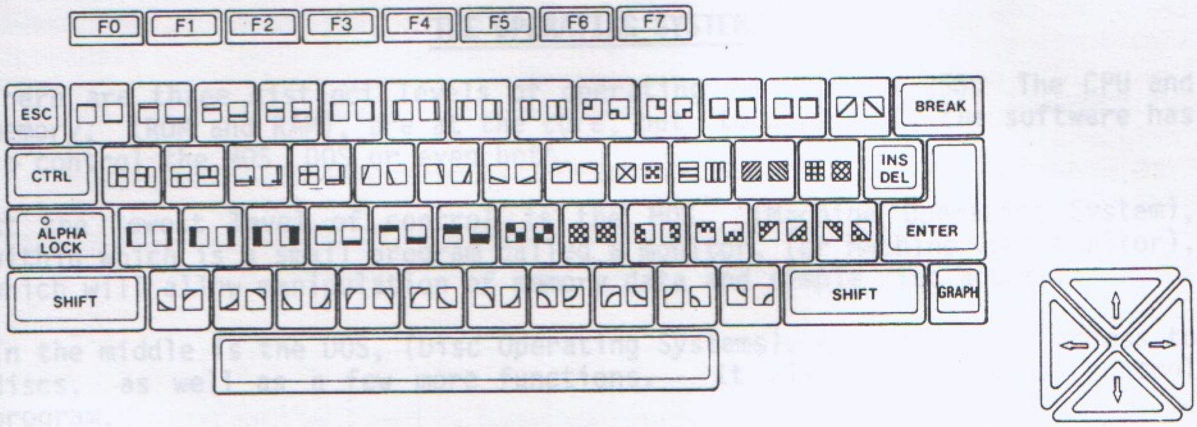


Fig 4.5 Graphics Key Allocation

The various character keys (numbers, letters, symbols) provide a secondary function incorporating graphic characters. Each key provides two different graphic symbols as shown in fig 4.5 (Note that the symbols are not actually shown on the keys). Each symbol occupies one character space.

To access the Graphic Characters, press the GRAPH key and hold it down whilst operating the particular character keys required. Under normal conditions the left-hand character will be accessed. By using the SHIFT key in conjunction with the GRAPH key the right-hand character will be accessed.

Try a few examples of graphic symbols. It is enjoyable building up patterns and shapes using the symbols. The only limitation is the imagination.

The Keyboard Trainer Program

The various character keys (numbers, letters, symbols) provide a secondary function incorporating graphic characters. Each key provides two different graphic symbols as shown in Fig 4.5. (Note that the symbols are not actually shown on the keys.) Each symbol occupies one character space.

To access the Graphic Characters, press the GRAPH key and hold it down whilst you operate the particular character keys required, Under normal conditions the left hand character will be accessed. By using the SHIFT key in conjunction with the graph key the right hand character will be accessed.

When the graph key is released the keys return to "text" mode.

CHAPTER 5

THE OPERATING SYSTEMS

There are three distinct levels of operating in EINSTEIN 256. The CPU and memory, (ROM and RAM), are at the core, but to access it, the software has to control the MOS, DOS or even both.

At the lowest level of control is the MOS, (Machine Operating System), within which is a small program called a monitor, (or Machine Code Monitor), which will allow manipulation of memory data and simple disc access.

In the middle is the DOS, (Disc Operating Systems), which provides access to discs, as well as a few more functions. It also has a monitor control program.

At the top, or outer layer, is the program to be run, whether it be a language or some sort of "applications" software.

MACHINE OPERATING SYSTEM (MOS)

MOS is lowest level of operation with the computer. We shall use MOS from now on when we wish to refer to the Machine Operating System. It allows direct access to the memory and various facilities are available for the manipulation and display of data contained therein.

The method of access to the MOS "monitor" depends on the current "state" of the computer. Access can be made as follows:-

a) From BASIC - Type the command **MOS**, (followed by ENTER).

b) From DOS (Disc Operating System) - Type the command MOS.

c) From power up, or reset, with a disc in the drive unit the computer will come up in DOS - therefore type MOS and key ENTER.

d) From power up, or reset, without a disc in the drive unit - the computer will come up in MOS.

In the EINSTEIN 256's ROM is a control program, which forms a link between the various functions required by the user, such as printing a character to the screen, getting a character from the keyboard or even plotting lines. The monitor also allows access to utilities which allow the alteration or examination of the contents of memory, as well as writing to the disc and perform number base conversion. This sort of low level access is mostly used by programmers and enthusiasts. For much of the time, the new user will not need them, but the software he loads undoubtedly will.

NOTES:-

1. Editing
Normal "Screen" editing is available in MOS, which means that the cursor may be moved around the screen by the cursor keys. Insertion and Deletion are as previously described.

2. Commands
Each command consists of a single letter, (upper or lower case), which in some instances may be followed by either 1,2,3,or 4 hexadecimal numbers. There is one exception: the H command requires a single decimal number input.

MOS COMMANDS

Each command consists of a single letter (either upper **or** lower case may be used) which in some cases is followed by either, 1,2,3, or 4 hexadecimal numbers. These hexadecimal numbers may contain up to four HEX digits. The exception is the H command which requires a single decimal number.

ARITHMETIC

Syntax:A xxxx yyyy

Where 'xxxx' and 'yyyy' are given as two hexadecimal numbers.

Purpose: This command calculates the sum, difference, and the necessary "offset" for a relative jump for the two numbers given.

The results are displayed as follows:-

```
   AAAA      BBBB      CC

   SUM     DIFFERENCE  OFFSET
```

In cases where a "relative jump" would not be possible for the two given values then '--' is displayed.

BAUD RATE

Syntax: B xy wwzz

Purpose: This command sets up the baud rate for the RS232-C Port according to the values specified by x and y. x is the "Receive rate" and y is the "Transmit rate" and can be a number in the range 0 to 8 representing baud rate values as given in the table below.

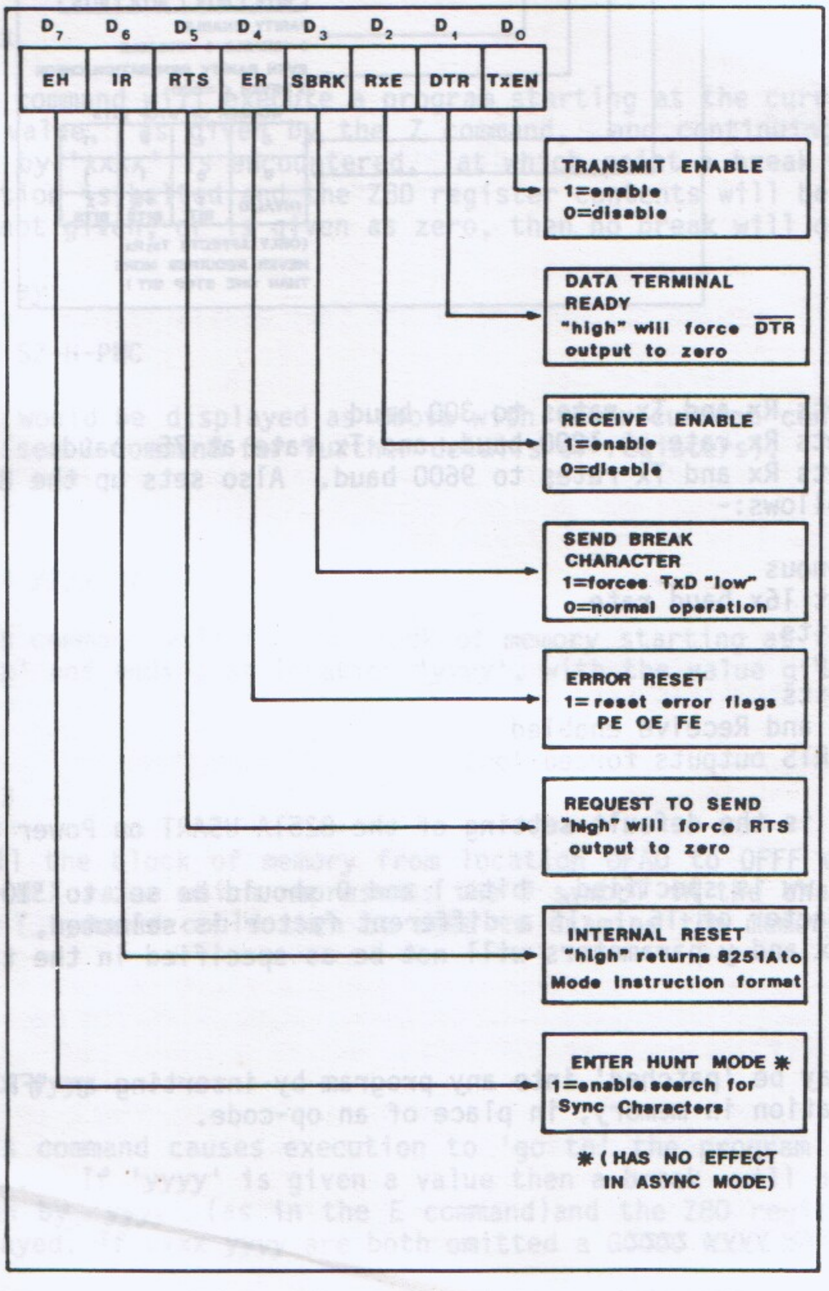
0 -	75 baud	5 -	1256 baud
1 -	110 baud	6 -	2400 baud
2 -	150 baud	7 -	4800 baud
3 -	300 baud	8 -	9600 baud
4 -	600 baud		

If only one of x or y is specified then both will be set to the same baud rate.

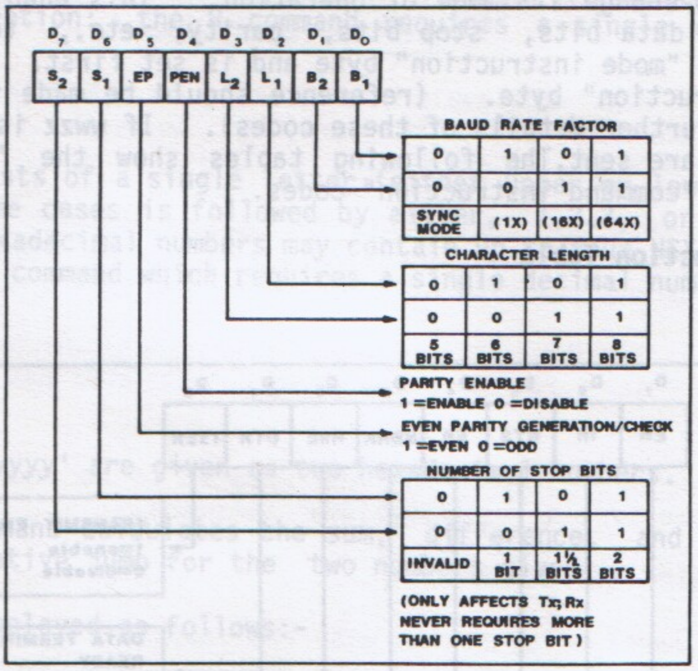
Any mix of baud rates for x and y is permissible with the following exception - 75 baud can only be set to receive if 75 baud is also set to transmit.

The wwzz is optional but if given it sends the specified values to the USART as commands to change its mode of operation. This enables such things as the number of data bits, stop bits, parity, etc., to be set up. ww represents the "mode instruction" byte and is set first. zz represents the "command instruction" byte. (reference should be made to the 8251A USART handbook for further details of these codes). If wwzz is not specified, no control codes are sent.The following tables show the "mode instruction" codes and the "command instruction" codes.

Command Instruction Table


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
EH IR RTS ER SBRK RxE DTR TxEN
TRANSMIT ENABLE
1=enable
0=disable
DATA TERMINAL READY
"high" will force $\overline{DTR}$ output to zero
RECEIVE ENABLE
1=enable
0=disable
SEND BREAK CHARACTER
1=forces TxD "low"
0=normal operation
ERROR RESET
1=reset error flags
PE OE FE
REQUEST TO SEND
"high" will force $\overline{RTS}$ output to zero
INTERNAL RESET
"high" returns 8251A to Mode Instruction format
ENTER HUNT MODE *
1=enable search for Sync Characters
* (HAS NO EFFECT IN ASYNC MODE)

Mode Instruction Table



D7	D6	D5	D4	D3	D2	D1	D0
S_2	S_1	EP	PEN	L_2	L_1	B_2	B_1

BAUD RATE FACTOR

0	1	0	1
0	0	1	1
SYNC MODE	(1X)	(16X)	(64X)

CHARACTER LENGTH

0	1	0	1
0	0	1	1
5 BITS	6 BITS	7 BITS	8 BITS

PARITY ENABLE
1 =ENABLE 0 =DISABLE

EVEN PARITY GENERATION/CHECK
1 =EVEN 0 =ODD

NUMBER OF STOP BITS

0	1	0	1
0	0	1	1
INVALID	1 BIT	1½ BITS	2 BITS

(ONLY AFFECTS Tx; Rx NEVER REQUIRES MORE THAN ONE STOP BIT)

Examples:

B3	Sets Rx and Tx rates to 300 baud
B50	Sets Rx rate at 1200 baud, and Tx rate at 75 baud.
B8 CE27	Sets Rx and Tx rates to 9600 baud. Also sets up the 8251A USART as follows:-

Asynchronous
clocks at 16x baud rate
8 data bits
no parity
2 stop bits
Transmit and Receive enabled
DTR and RTS outputs forced low.

(B8 CE27 is the default setting of the 8251A USART on Power Up)

NOTE: When ww is specified, bits 1 and 0 should be set to '10' to select a baud rate factor of 16. If a different factor is selected, the baud rate set by the x and y parameters will not be as specified in the table above.

BREAK

A "break" may be 'patched' into any program by inserting an "FF" code at the desired location in memory, in place of an op-code.

COPY

Syntax: C xxxx yyyy zzzz

Purpose: This command is used to copy a block of memory which starts at address 'xxxx' and finishes at address 'yyyy'. The block is copied into a new position in memory beginning at the address given by 'zzzz'.

DECIMAL

Syntax: D xxxx

Purpose: This command converts a hexadecimal number given in 'xxxx' into a decimal number in the range 0 to 65535, and then displays the result on the next line.

Example:

D 003E will give a result of 62 decimal.

EXECUTE

Syntax: E xxxx

Purpose: This command will execute a program starting at the current program counter (PC) value, as given by the Z command, and continuing until the address given by 'xxxx' is encountered, at which point a break will occur, program execution is halted and the Z80 register contents will be displayed. If 'xxxx' is not given, or is given as zero, then no break will occur.

Register Display:

A BC DE HL PC SZ-H-PNC

The registers would be displayed as above with their current contents shown below them. (see Z command for further details of registers).

FILL

Syntax: F xxxx yyyy zz

Purpose: This command will fill a block of memory starting at the location given by 'xxxx' and ending at location 'yyyy', with the value given by 'zz'.

Example:

F 0FA0 0FFF 23

This will fill the block of memory from location 0FA0 to 0FFF with 23 HEX (23 being the HEX value which represents the # symbol in the character code table. The T command could then be used to examine the memory locations given above).

GOTO

Syntax: G xxxx yyyy

Purpose: This command causes execution to 'go to' the program starting at address 'xxxx'. If 'yyyy' is given a value then a break will occur at the location given by 'yyyy' (as in the E command)and the Z80 registers status will be displayed. If xxxx yyyy are both omitted a G0000 will be performed

Example:

G B002 B0FF
This instructs execution to transfer to location B002 and carry out the routines from there onwards. When location B0FF is reached a break point occurs and the Z80 registers will be displayed with their current contents.

HEXADECIMAL

Syntax: H ddddd

Purpose: This command converts a decimal number given by 'ddddd' (in the range 0 to 65535) to a hexadecimal number and displays the result on the next line.

Example:

H 246 will give a result of $00F6_H$

LANGUAGE

Syntax: LN

Purpose: This command selects one of the four available character sets.

N	Character Set	N	Character Set
0	= Einstein (ISO646)	1	= ASCII
2	= German	3	= Spanish

MODIFY

Syntax: M xxxx

Purpose: This command is used to alter an area of memory starting from the address given by 'xxxx' and the procedure is as follows.

a) Type in the command, followed by the starting address.

b) The address and its contents are displayed on the screen with the cursor positioned at the first digit of the contents.

c) To modify the contents, type in the new values as a two digit HEX number, overtyping the existing contents, and press ENTER.

d) The next address and data then appears, again with the cursor positioned at the first digit of the contents.

e) If no modification is to be made then simply press ENTER so as to examine the next address, and so on.

f) To EXIT the modify command type a full stop (.) at cursor position and press ENTER.

It is possible to modify the contents of consecutive addresses by typing the successive bytes on one line. When ENTER is pressed the next unmodified address is displayed.

Example:

G B002 BOFF
This instructs execution to transfer to location B002 and carry out the routines from there onwards. When location BOFF is reached a break point occurs and the Z80 registers will be displayed with their current contents.

HEXADECIMAL

Syntax: H ddddd

Purpose: This command converts a decimal number given by 'ddddd' (in the range 0 to 65535) to a hexadecimal number and displays the result on the next line.

Example:

H 246 will give a result of 00F6$_H$

LANGUAGE

Syntax: LN

Purpose: This command selects one of the four available character sets.

N	Character Set	N	Character Set
0	= Einstein (ISO646)	1	= ASCII
2	= German	3	= Spanish

MODIFY

Syntax: M xxxx

Purpose: This command is used to alter an area of memory starting from the address given by 'xxxx' and the procedure is as follows.

a) Type in the command, followed by the starting address.

b) The address and its contents are displayed on the screen with the cursor positioned at the first digit of the contents.

c) To modify the contents, type in the new values as a two digit HEX number, overtyping the existing contents, and press ENTER.

d) The next address and data then appears, again with the cursor positioned at the first digit of the contents.

e) If no modification is to be made then simply press ENTER so as to examine the next address, and so on.

f) To EXIT the modify command type a full stop (.) at cursor position and press ENTER.

It is possible to modify the contents of consecutive addresses by typing the successive bytes on one line. When ENTER is pressed the next unmodified address is displayed.

To examine a different address when using the modify command it is possible to delete to the start of a line and type in the new address followed by ENTER. The contents of that address then appear on the display as usual.

Example:

```
M OB2C
```

This produces a display commencing from the address given (i.e. OB2C) and then continues from there as modifications are made.

Example Display:

```
OB2C 22          (The contents of the addresses
OB2D 41           are given as examples only)
OB2E 3A
OB2F CD
OB30 23
OB31 4F
OB32  F
```

The contents of address OB2C to OB31 have been examined in turn. The cursor is shown positioned at the first digit of the contents of the location currently being examined (OB32). After each of the two digits have been examined and the ENTER key pressed, the next location will appear on the display.

NOTE: Mistakes in previous lines may be corrected by using the cursor control keys to transfer back to a particular line and then type in the correct values over the incorrect values. Press ENTER whilst still on the corrected line in order to execute the modification and then continue. Use a full stop, (.), to terminate the routine.

READ

Syntax: **R** xxxx yyyy sstt d

Purpose: This command fills a block of memory starting at address "xxxx" and finishing at address "yyyy". The data to be placed in memory is read from the floppy-disc starting at the sector "ss" on track "tt" of drive "d".

The sector number is "ss" must be in the range 0 to 9, track number "tt" must be in the range 0 to 27 (hexadecimal) and drive number must be 0 or 1.

Using this facility information may be transferred from any specified area of the disc into memory, as selected by the user.

SCREEN

Syntax: SN

Purpose:

This command selects either text mode 2, graphics mode 2, graphics mode 3 or graphics mode 6 for the enhanced video display processor (EVDP) in screen widths of 32,40,64 or 80 columns.

N	EDVP mode	screen width, columns
0	Graphics mode 2	32
1	Graphics mode 2	40 (default)
2	Text mode 2	80
3	Graphics mode 6	32
4	Graphics mode 6	40
5	Graphics mode 6	64
6	Graphics mode 6	80

TABULATE

Syntax: T xxxx yyyy zz

Purpose: This command will tabulate the contents of memory starting from the address given by 'xxxx' and ending at address 'yyyy'. If zz is **not** specified the memory will be displayed in blocks of 8 bytes, otherwise in blocks of zz bytes. The command can be abandoned by pressing the 'ESC' key.

Example:

T 0100 0120

This will produce a display similar to the following (the contents of locations being givcn as examples only).

```
Address          Data                  ASCII character represented

  0100    C3 66 2B C3 79 07 01 3D          Cf+Cy..=
  0108    BF 39 78 3B C4 38 3D 39          ?9x;D8=9
  0110    1A 3D A2 3D E8 3B D2 06          .="=h;R.
  0118    EA 2B 0F 00 80 00 7A 01          j+....z.
```

Each line of the tabulation begins from the left with a memory address. This is followed by the current value in that location and then each value of the next 7 consecutive locations. The right hand end of the line gives the corresponding characters represented by the values in the same consecutive sequence. When there is no ASCII character for the hex code tabulated, a '.' (full stop) is displayed.

Looking at the first line of the listing, 0100 is the first address and C3 is the value contained therein. Then 66 is the value in 0101, 2B is the value in 0102, and so on up to location 0107 which contains the value 3E. The number of consecutive locations given on one line can be varied by use of the zz parameter which specifies the actual number to be displayed.

NOTE:

1. All memory locations (addresses) and contents are given in Hexadecimal.

2. The characters displayed in the right hand section of each line ignore the most significant bit of the corresponding code. An example of this appears in the first line of the display given above. C3 is given as the value representing the character C.

   Thus:-

   C3-80 = 43 (i.e. ASCII value for character C).

The sector number "ss" must be in the range 0 to 9, track number "tt" must be 0 to 27 (hexadecimal) and the drive specified must be 0 to 1.

WRITE

Syntax: **W** xxxx yyyy sstt d

Purpose: This command will write a block of data to the disc currently in the drive specified by d. The 'write' will start from track tt, sector ss and the block will be transferred from memory starting at address 'xxxx' and ending at address 'yyyy'.

The sector number "ss" must be in the range 0 to 9, track number "tt" must be 0 to 27 (hexadecimal) and the drive specified must be 0 or 1.

Thus using this facility data may be transferred to any specified area of a disc as selected by the user.

COLD START

Syntax: X

Purpose: This command will cause a "cold start" transfer to the current program from MOS. (Always provided the execution vector has been passed into MOS by the program loaded, otherwise performs a G 0100)

When using this command to return to BASIC from MOS all programs and variables are lost and the transfer is as if BASIC had just been loaded. Hence the term "cold start".

WARM START

Syntax: Y

Purpose: This command will cause a "warm start" transfer to the current program from MOS.

When using this command to return to BASIC all programs and variables are preserved. Hence the term "warm start".

The "warm start" vector is equivalent to a G0103.

DISPLAY REGISTERS

Syntax: Zx

Purpose: This command will display the Z80 register contents according to the value specified by x as indicated below.

Z0 - Displays normal registers.
A,BC,DE,HL,PC and Flags.

Z1 - Displays alternate registers.
A, BC, DE , HL, PC and Flags'.

Z2 - Displays special registers.
I,IX,IY, SP and PC.

Z80 Registers and Flags

MAIN REGISTER SET

A	Accumulator	F	Flag Register
B	General Purpose	C	General Purpose
D	General Purpose	E	General Purpose
H	General Purpose	L	General Purpose

ALTERNATE REGISTER SET

A	Accumulator	F	Flag Register
B	General Purpose	C	General Purpose
D	General Purpose	E	General Purpose
H	General Purpose	L	General Purpose

	S	-	Sign Flag
	Z	-	Zero Flag
	-	-	Not Used
F (flag)	H	-	Half Carry Flag
Register	-	-	Not Used
	P	-	Parity Flag
	N	-	Subtract Flag
	C	-	Carry Flag

SPECIAL REGISTERS

I	-	Interrupt Register	
IX	-	Index Register	X
IY	-	Index Register	Y
SP	-	Stack Pointer	
PC	-	Program Counter	

Display:
The registers are normally displayed as shown in the examples below (the contents shown are examples only)

Examples
Normal Registers

```
  A     BC     DE     HL     PC    SZ-H-PNC
 00    0000   0000   0000   B002   00010100
```

```
Alternate Registers
   A     BC     DE     HL     PC     SZ-H-PNC
  00   0000   0000   0000   B002   00010100

Special Registers
   I    IX    IY    SP    PC
  FB   0000  0000  FCFF  B004
```

NOTE:

a) PC is the same for all three displays.

b) The R (Refresh) register is not given since its value is constantly changing.

c) An automatic ZO occurs after a break from a program.

FILE NAMING CONVENTIONS

Filenames

The file naming convention is as follows:-

drive: filename . extension

where:-
drive - is the drive number where the file will be. If omitted, the default drive is selected.

file name - is the title of the file selected by the user and can be up to 8 characters in length.

extension- specifies the type of file and can be up to 3 characters in length.

Both filename and extension may consist of any combination of ASCII characters with the following exceptions:-

a) characters with ASCII codes less than 32 (control codes).
b) characters with ASCII codes greater than 127
c) the characters . , " ; : = ? *

NOTE

It is not always necessary to specify all the parameters. In some cases the extension can be omitted, or wild cards substituted for any parameters; e.g. the .COM extension may be omitted when executing command files in DOS, and the .XBS extension may be omitted when executing BASIC files under EBASIC. In all cases the drive: parameter may be omitted when files reside in the current drive.

In order to denote when wild cards can be used, or extensions omitted, we shall use the following notation when referring to file syntax:

<ufn> unambiguous file name - all parameters must be specified.
<afn> ambiguous file name extensions may be omitted and wild cards used.

Example: MOONLAND .XBS

This specifies the file called MOONLAND, which is a BASIC file (as denoted by the extension .XBS).

Common Extensions

The following are commonly used extensions to denote specific types of file.

1. .XBS - This is an EBASIC source file.

2. .ASC - This indicates an ASCII file.

3. .OBJ - This is an object, or machine code file.

4. .COM - This is a command file. These files can be loaded and run automatically under DOS by simply specifying their name without the extension. The files load from 0100H upwards and then execute.

Users may select their own combinations for files and the following are given as examples:-

.DAT - data file
.DOC - document files
.BAK - backup file
.GRA - graphics file

NOTE:

1. The standard extension .XBS, .ASC, and .OBJ are only distinguished under BASIC.

2. .ASC files can be listed to the screen by use of the DISP command. This produces a similar effect to LIST for a .XBS file in BASIC.

3. Other extensions are often assigned to files created by a program. In the same way, a Data Base program may store as .DAT any number of files, so the index might be stored .IND, or a Word Processor might store a text as .TXT, .UFT, or DOC.

4. Wild Cards - In some disc applications it is necessary to specify more than one file, for example when specifying the files for a DIRECTORY list. If one of the characters in a file name is replaced by a ?, then it will match any character in that position in the file found.

Example:

```
X?Z.ASC - Matches XYZ.ASC,XAZ.ASC,X9Z.ASC,etc.
?X.D?T  - Matches AX.DAT,BX.D7T,4X.DZT,etc
```

5. If an * is inserted in place of a character then it will match all the characters at and after that particular position in the file name or file type in which it appears.

Example:

```
*.XBS - matches any XBS file.
*.* - matches any file, and is the same as ????????.???
PROG*.ASC matches PROG1.ASC,PROG10.ASC,PROGABC,ASC,etc.
```

Loading Command (.COM) files

To load command file (.COM files) simply key in <afn> i.e. omit the .COM extension.

When a command file has related files, these too can be entered in the DOS command line i.e. the PSG.XBS file can be automatically loaded and run from DOS at the same time as loading EBASIC. To do this key:

```
            EBAS PSG, then press ENTER
```

DOS will load and excute EBASIC which will, in turn, load and execute PSG.XBS. In general the format is:

```
 command file   space parameter 1   space  .....  parameter n    ENTER
```

The number of related files or parameters that can be used depends upon the particular applications program. For EBASIC, the limit is one .XBS file.

DOS COMMANDS

DIR (Directory)

Syntax: DIR afn

Purpose: This command will display the directory of the disc currently in use, showing the files specified by filename . If filename is omitted then the whole directory will be displayed.

The DIR command will display up to the first 12 lines (24 entries) of a disc directory on the screen. If the directory is less than 12 lines it will be displayed in its entirety on the first entry of the command and the system prompt (drive number and colon) will then appear at the end of the display If, however, the directory extends beyond 12 lines then the cursor only will appear at the end of the initial display, indicating there is more to follow. To examine the remainder of the directory, press any key. This will cause the next 12 lines of the directory to scroll up onto the display. The process is repeated until the system prompt (drive number and colon) appears on the display, indicating the end of the directory.

The total size of the files displayed is given at the bottom of the directory listing, together with the free space remaining and the total capacity of the drive.

Example: 0:DIR

This will give a display of all the directory for the disc in drive 0, similar in format to the one given below.

```
0:*EBAS     .COM : MAIL     .XBS
0: FORMAT   .COM : FKEY     .COM
0: XYZ,     .ABC :*TESTPROG.ASC
56k Size,  132k Free, 190k Total
```

This shows that the disc capacity is 190k, the total size of files shown is 56k, and 132k is unused. Note the asterisk on EBAS.COM and test prog.ASC, indicating locked files which cannot be written to or RENamed.

Example:
0:DIR *.COM

```
0:*EBAS     .COM : FORMAT   .COM
0: FKEY     .COM
39k size, 132k free, 190k Total
```

Example:

```
0:DIR EBAS .COM
0:*EBAS    .COM
16k Size,  162k Free, 190k Total
```

This last example shows how the size of any one file may be obtained by just performing a DIR with a single-file specification. If DIR of a non-existent file is requested, it will be returned as a 0k size preceded by 'No File'.

DISP (Display Files)

Syntax: DISP <ufn>

Purpose: This command should be used with files which only contain ASCII characters. It causes the contents of a file to be displayed on the screen (i.e. lists it).

Example:

DISP TEXT.ASC

This will display the text of the file named TEXT.ASC on the screen.

NOTE: Using DISP may show some rather surprising results on the screen. Don't forget that some languages store their data as "tokens", (a sort of short-hand), so you won't make a lot of sense out of an .XBS file, for example. Some machine code files contain no printable characters at all, but the DISP function is very useful for inspecting those which are.

Some discs have a file called "READ.ME" which is often a late addition to supplied documentation. It is designed to be displayed. If you want a hard copy and have a printer, use CTRL-R prior to pressing ENTER so as to dump the displayed file to the printer.

DRIVE (Disc Drive)

Syntax: d:

d is specified as a number 0 or 1 depending on the drive units available within the system.

If the drive specified in d is not available on the system a

"No Drive d:"
"No (0-1)?" message will be displayed. You should enter the correct drive number.

Purpose: This command sets up the default disc drive as specified by d for any subsequent access to a disc.

Example:

1:

This selects drive 1 as the default drive. The new default drive number will appear as the DOS prompt until subsequently changed.

ERASE

Syntax: ERA <afn>

Purpose: This Command will erase the file, or a group of files, given by file-name.

The first file conforming to the given specification will appear on the screen, together with a "?" symbol. The user must then type in one of the following single letters:-

Y - "Yes", erase this file. The word "Erased" will appear on the same line when that operation has been completed.

N - "No", do NOT erase this file. The next file to be erased will then be displayed, if one exists.

A - "All", erase this, and all subsequent files without further prompting. The word "Erased" is displayed as each file is erased.

F - "Finish", abandon the ERA command, without erasing any more files (those previously shown as erased will remain erased).

If the file does not exist, a "No File" error will appear on the screen.

If the file is a locked file then the following message appears:-

Unlock (Y/N)?

The user must then type in one of the following single letters:-

Y - "Yes", unlock the file and erase. The word "Erased" will appear on the same line when that operation has been completed.

N - "No", do not unlock the file. This then abandons the current file. The next file to be erased will then be displayed, if one exists.

GO

Syntax: GO

Purpose: This command will cause a jump to be made to location 0100 (HEX). The program at location 100_H, - memory will then be executed.

NOTE: If, for some reason, you have left a program (e.g. Backup or Copy) and you want to return to it for another operation, (say formatting, or back-up of a file), then GO is the right command. If, however, you jumped out of a language, and you want to keep the variables, then type MOS, and then Y, or G103 which will take you to the start of the program, but "warm started". (This is mainly applicable to a language, like BASIC.)

LOAD

Syntax: LOAD <ufn>

Purpose: This command is used to load a file from the disc into memory. The file will be loaded starting at memory location 0100 (HEX). The size of the file, specified as a number of 512 byte BLOCKS, is then displayed on the screen.

Example:

```
  LOAD EBAS.COM
```

When loading is complete the following would be displayed on the screen:-

```
N BLOCK(S)
```

Where N is given as a decimal number

The file can be patched by returning to the "machine operating system" for modification etc., or run by means of the GO command.

Command Files:

Command files (.COM) will auto-execute by simply entering the <filename> i.e. they will load and run automatically without requiring further action from the user.

Example:

To load and execute BASIC from DOS,the BASIC file is EBAS.COM. Simply type EBAS, and then press ENTER. There is no need to type the file extension .COM when in DOS.

If the command (.COM) file can itself call other files on disc, (i.e. BASIC can call .XBS files) then these, too, can be entered in the DOS command.

For example, there is a demonstration file on your master disc, which runs under BASIC. This appears on the directory as DEMO.XBS To execute the file DEMO.XBS, under BASIC directly from DOS, you should type:

```
EBAS DEMO
```

DOS will load and execute BASIC, and in turn BASIC will load and execute the DEMO program.

NOTE: The file following the command (.COM) file must be of a matching type (e.g. .XBS for BASIC; .LOG for LOGO)

LOCK

Syntax: LOCK ufn S

Purpose: This command is used to lock a given file so that it cannot be rewritten or altered without first being UNLOCKED.

In addition LOCK may be used to turn the file into a 'SYSTEM' for hidden file, by including the letter S after the command. System files do not appear in the directory list but may be read or executed as required.

Example:

```
LOCK MAIL.XBS    - locks the file MAIL.XBS

LOCK EBAS.COM S - locks the file EBAS.COM and makes
                  it into a system file.
```

NOTE: The lock marker is a small star *. When looking at the list of files under the DIR command, the list would have the star at the beginning of the filename:-

e.g:-

```
*MAIL.XBS  -  would indicate that the file MAIL.XBS
              is locked.
```

A system file is not displayed on the DIR listing.

UNLOCK

Syntax: UNLOCK <ufn>

Purpose: This command is used to unlock a previously locked file so that it may be rewritten or altered. If the file was also a system file, that attribute will automatically be removed by the UNLOCK command and it will then display normally in the directory list.

Example:

```
UNLOCK MAIL.XBS  - unlocks the file MAIL.XBS
                   (assuming it was previously locked).
```

NOTE: Unlock will also expose a "System File".

MOS (Machine Operating System)

Syntax: MOS

Purpose: This command is used to return to MOS. This is useful if patching and debugging are required for the currently loaded program/file. The modifications are made in MOS, the G command is used to return to DOS, and finally the program/file may be saved using the SAVE command within DOS.

PSW (Password)

Syntax: PSW password

Where password is an 8 character name selected by the user and may contain any characters other than control characters. Note: all 8 characters must be entered.

Purpose: This command sets up the password protection facility which can be used for security purposes to limit the access to any given file to authorised personnel only.

Once a password has been invoked any files saved can then only be loaded back under the same password. Any files which exist on the disc either without a password, or under a different password, cannot be loaded whilst the current password is in operation.

To change the password use PSW again with a different password. To turn off the password (or make sure that no password is in force!) use PSW by itself.

NOTES:

1. An unprotected file must be read back without a password being in force.
2. The password itself is not stored anywhere, therefore the user must know it or record it elsewhere.

3. There is no indication given in the directory that a file has been protected; the file can apparently be read, but appears as complete rubbish.

4. The directory itself is unaffected by the password so that it is perfectly acceptable to mix unprotected files and files saved under various passwords stored on the same drive (as long as you know which are which!).

Example:

```
PSW IXZ247YT
SAVE 20 MAIL.XBS
```

Having saved the file MAIL.XBS under the password IXZ247YT it can only be read or loaded back if that password is in force. Likewise other files not saved under this password cannot be read or loaded while it is in force.

REN (Rename)

Syntax: REN old <ufn> TO new <ufn>

Purpose: This command is used to rename an existing file, giving it a different name or extension, or both.

Example - assume that HATS is an existing BASIC file.

```
REN HATS.XBS  TO  BUTS.XBS
```

This will rename the existing BASIC file HATS giving it the new name BUTS.

Example:

```
REN HATS.XBS TO HATS.BAK
```

This will rename the extension of the file HATS from .XBS (BASIC) to .BAK (say backup file)

If <filename> is a locked file then the following message appears:-

```
File Lock, d
Unlock (Y/N)?
```

The user must then type in the following single letters:-

Y - "Yes", unlock the file and rename.

N - "NO", do not unlock the file. (This then abandons the REN command, without renaming the file).

SAVE

Syntax: SAVE N ufn

Where N = BLOCK size of file

Purpose: This command is used to save the file in memory, on to the disc.

Example

Assume that a machine code game program exists in memory, starting at 100H. It occupies 2 BLOCKS. (1 k bytes)
You wish to save the program onto disc, and call it "GAME". Assuming it is a command file, the extension will be .COM. To save such a file, type:-

```
SAVE 2 GAME.COM
```

NOTE: See also LOAD.

DOS UTILITIES

There are five utilities which allow the user to prepare discs and transfer files. The utilities fall into two categories; resident and disc based.

RESIDENT UTILITIES

There are two utilities which will allow the user to prepare discs and transfer files. These are stored in ROM, and are available for use whenever Einstein 256 is in DOS.

a) **BACKUP**
This utility transfers all the data on all the tracks to a new disc. This process will include erased files as there is no re-organisation of the data. The disc must first be formatted.

b) **COPY**
This versatile utility will copy files to any logical device or vice versa, on a file-by-file basis, and will ignore erased files. Files are transferred from one logical or peripheral device to another. Note that not all files will copy, as various methods are employed to combat piracy of software etc.

To call these utilities, start from DOS and follow the instructions on each utility.

NOTE: Although Einstein 256 may be used with earlier versions of Tatung/Xtal DOS, in order to use the resident utilities, you should use the EDOS version 3 supplied with Einstein 256.

Should you wish to use discs programmed for the Einstein computer, you are advised to use the DOSCOPY program to transfer the Einstein 256 DOS to your existing Einstein disc.

NOTE: Not all discs need DOS, many games don't for example. In general, it will be unwise to transfer DOS to discs which do not display a DOS sign-on message when CTRL-BREAK is pressed.

a) BACKUP

Two disc drives must be declared when using BACKUP. The source drive, which contains the disc to be duplicated, and the destination drive, which contains the disc which will receive the copy. When Einstein 256 is used without extra disc drives, both the source and destination drives will be the same, drive 0, so the source and destination discs will need to be interchanged during the backup procedure. The program prompts the user to do this.

Typing BACKUP selects the backup utility. (There is no need to use your Master Disc, as with other computers, as the backup program is built into the computer itself).

The screen will display the following message:-

```
BACKUP V1.XX

Source Drive(0-1 or X)?
```

You should now type in the drive number which contains the original (source) disc, which is to be duplicated, This will usually be drive 0. If you type "X" in response to the question, the program will return control to the disc operating system.

When the source drive number has been entered, you will be asked to specify the destination drive number, For most Einstein 256 users, this will be the same as the source drive. The screen will display the message

```
Destination Drive (0-1 or X)?
```

You should now key in the chosen drive number.

The screen will now display:-

```
Source drive 0, Destination drive 0,

Press ENTER to continue
or "X" to abandon.
```

The program will prompt the user as to what to do, and when the duplication process is completed, the screen will display:-

```
Disc Backed up - - OK

press ENTER to continue
or "X" to abandon.
```

Should you wish to duplicate another disc, press ENTER, and the process will be repeated, otherwise press "X" to return to DOS.

NOTE: The backup program copies the whole side of a disc, including the first two tracks, which contain the disc operating system. BACKUP overwrites all the data that may exist on the destination disc with that which is on the source disc, including "blank" areas of the disc.

Should the destination disc contain any wanted programs, or data, then these will be destroyed during the backup procedure, so take care!

b) COPY

This is a general-purpose file-transfer program operating from within the disc operating system and provides facilities for the transfer of data between peripheral devices. The use of COPY will normally involve:-

1. Copying of files from disc to disc using a single drive.

2. Copying of files from disc to disc, using two drives.

3. Copying of files between peripheral devices connected to the computer.

Copying Files:

The facility can be used to copy individual files from one disc to another, or multiple files can be copied one after the other. During the copying process the file name may be changed if so desired.

1. Type **COPY**<input-afn> **TO** <output-afn>and press ENTER (Where <input-afn> and <output-afn> include the respective drive numbers e.g. 0:EBAS.COM to 1:EBAS.COM)

In this form a single file will be copied from one disc to another in the drives specified and the command then terminates. When a drive is not specified the current default drive number is assumed.

NOTE: If <input-afn> and <output-afn> are identical (i.e. same name and same drive number) a prompt is displayed on the screen so the user may swap the discs when using a single drive for copying. If the names of the files are different, then it is possible to make a copy of a file on the same, or different disc under the new name.

Examples:

```
COPY 0:OLD.ABC TO 0:
```

This will copy OLD.ABC from the source disc, and display the message.

Insert Destination Disk

When you press ENTER, the file will be copied onto the new (destination) disc with no change of name.

```
COPY 0:OLD.ABC TO 1:NEW.XYZ
```

This copies the file OLD.ABC from drive 0 to drive 1 under the new name NEW.XYZ.

```
COPY 0:OLD.ABC TO 1:
```

This copies the file OLD.ABC from drive 0 to drive 1 under the same name.

```
COPY OLD.ABC TO 1:NEW.XYZ
```

This copies the file OLD.ABC from the current default drive to drive 1 under the new name NEW.XYZ.

```
COPY 1:*.XBS TO 0:
```

This copies all .XBS files from drive 1 to drive 0.

2. Type **COPY** and press ENTER

In this form the command is used by itself and a "*" prompt appears on the screen after ENTER has been keyed. The input-file TO output-file statement (in any of the forms given above) is then typed in following the "*" prompt and ENTER is keyed. The copying process takes place and the "*" appears on the next line for another entry.
In this way a series of files may be copied one at a time under the single command.
To terminate COPY simply key **ENTER** immediately following the "*" prompt.
COPY may be terminated by CTRL-C;

Copying Using Multiple File Specifications:

Multiple file specifications can be used with COPY and several files may then be copied at one go (even an entire disc). In this instance each file name is displayed on the screen followed by a question-mark prompt, before being copied. To continue the user must type a single letter response as follows:-

Y - Yes, copy this file.
N - No, do not copy this file, go to the next file.
A - All, copy this and all other files without further prompting.
F - Finish, terminate the COPY command.

Each file successfully copied displays the word "copied" to the right of the name on completion. The following shows an example of the type of display to be expected when using multiple file specifications:

```
0:COPY *.XBS TO 1:
CHESS   .XBS?Y copied
MUSIC   .XBS?N
CIRCLES .XBS?Y copied
ELLIPSES.XBS?A copied
HANGMAN .XBS   copied

0:
```

On a one drive system:-

```
COPY 0:*.XBS TO 0:
```

Would copy all .XBS files and give prompts to swap discs.

Copy Using Peripherals/Devices:

The COPY utility may be used in conjunction with peripheral devices in place of file names as shown in the format below.

Type COPY <Input-device> TO <Output-device>and press ENTER.

COPY can also be used in combination with files and devices as shown below.

Type COPY <Input-device> TO <Output-afn>and press ENTER.

Type COPY <Input-afn> TO <Output-file>and press ENTER.

The peripheral device names which are allowed in the COPY command are as follows:-

Logical Devices -

CON: console, input or output
AUX: auxiliary, input or output
LST: list, output only. An error message will result if LST: is used as an input device.

Physical Devices -

VDU: VDU, output only
KBD: keyboard, input only
SRL: serial RS232, input or output
LPT: parallel line printer, output only.

Examples:

COPY CON: TO LST:

This copies data input at the keyboard to the printer.

COPY SRL: TO 0:DAT.XBS

This copies the input from the RS232 to the disc in drive 0 under the name DAT.XBS

Options:

In addition to specifying the input and output within the COPY command, the user may also give a list of optional parameters. The parameter list is used in the output specification only, immediately following the particular file or device it relates to, and is contained within chevrons ( ). Some of the parameters may be followed by an optional decimal number "n", while others may require a string "s" terminated by a ctrl-Z (shown as Z on the screen).

Parameters may be selected from the following list:

Dn - Delete any characters extending beyond "n" in a given line. When copying text files, COPY counts the number of characters copied after a carriage-return (i.e. the length of a line of text). Upon receipt of the nth character in the line, COPY will ignore all characters until another carriage-return is received. This feature is useful when dumping text files to narrow printers, and when only a few lines are longer than the printer width.

E - Echo all copied characters to the console device (i.e. the screen)

F - Ignore any form-feed/clear-screen characters (OCH) received

L - Translate any upper-case (A to Z) characters to lower-case.

N - Add a line number followed by a ":" to each line of a text file transferred, starting at 1 and incrementing by 1. Leading zeroes are suppressed unless N2 is given, in which case leading zeros are included, together with a "TAB" character (09H) following the number.

O - Object code transfer; Treat any end-of-file (EOF) characters as if they were normal received characters. This only applies to devices- the EOF code would normally be the only way to indicate termination of a transfer from a peripheral device. With this option, however, object code may be received from a device, and termination effected by pressing CTRL-S on the keyboard.

Pn - Include form-feed characters (OCH) at every n lines, with one at the start. If n is excluded or set to 1, a default value of 60 will be assumed. If the F parameter is also used, the old form-feeds are removed and new ones added at the appropriate places.

Qs Z - "Quit" - Terminate copying after the string s has been received. The string s will be included in the copied data.

R - Read system files - System files would normally be ignored on file copying, but will be included if this parameter is given (see - LOCK). The R parameter is only valid when using wild cards and should not be used when specifying individual files.

Ss Z - Ignore all input until string s is received, then start copying. The string s will be included in the copied data. The Q and S parameters may be used to "abstract" portions from a file. Note the use of the Z (CTRL-Z) code as the delimiter in both cases.

Tn - Expand "TAB" characters (09H) with spaces to every nth column during the copy. (Normally "TAB" characters would be transferred as received, so for example, a printer could be set up with its own tab points rather than expanding with spaces).

U - Translate any lower-case (a to z) characters into upper-case.

V - Verify that files have been copied correctly, by re-reading after the write operation. (This only applies if the output is to a file).

W - Write over "locked" files without prompting the user. Normally, COPY will prompt the user before attempting to over-write a file that has been "locked".

Z - Zeros the parity bit (bit 7) of each character received.

Examples:

```
COPY XMPL.ASM  TO SILLY.LIB   SSUB1: QsJP  LABL2 Z
```

This copies from the file XMPL.ASM to the file SILLY.LIB on the same (default) drive, the section starting with string "SUB1:" up to the string "JP LABL2"

```
COPY TEST.ASC   TO LST:  NT8U
```

This copies the file TEST.ASC on drive 0 to the current "list" device, with each line numbered, tabs expanded with spaces to every 8th column, and lower-case letters converted to upper-case.

```
COPY *.* TO 1:  V
```

This copies all files from the current default drive onto drive 1, with verification of all files after copying. The V can also appear on the "input" side, with the same effect.

DISC BASED UTILITIES

There are four DOS utilities which are on the master disc, these are:-

a) FORMAT.COM

b) FKEYS.COM

c) DOSCOPY.COM

d) TAPE.COM

These utilities are used to initialise blank discs, to set up function keys, to copy system (DOS) tracks from one disc to another and to read cassette tapes respectively.

a) FORMAT

Insert backup master disc, (if you have not made a backup, see Chapter 1).

From DOS, type FORMAT, and press "ENTER"

You will be asked to insert a disc with the required system tracks. You are strongly advised to ensure that the 'Write Protect' tab or the master disc is in the 'Protect' position (refer to Chapter 3) before going further.

You will now see another message:-

```
OK -- Format Drive (0-1 or X)?
```

Now remove the reference (master) disc and insert the disc to be formatted.

Select 0 or 1 (0 is the internal drive), and press ENTER.

During the process of formatting the "track numbers" are presented on the screen as below

```
0         1         2         3
0123456789012345678901234567890123456789
```

(i.e. 40 tracks in four groups of 10)
As each track is formatted in turn a letter F (for format) will appear below the track number.

```
0         1         2         3
0123456789012345678901234567890123456789
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

When every track has been formatted, each one will then be verified and again a letter V (for verify) will appear below each letter F on the screen in turn as verification takes place.

```
0         1         2         3
0123456789012345678901234567890123456789
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
```

When formatting and verification are complete the following message will appear on the screen.

```
Disc Formatted and Verified -- OK
Press ENTER to continue
or 'X' to exit
```

At this stage the destination disc is formatted. The user may format another side, or another disc or return to DOS.

NOTE: If you get failure of the format process, try again! If you still have a failure, you may have a faulty disc. Try formatting a different disc.

b) FKEY

The purpose of this utility is to allow users to program the function keys, without having to load EBASIC. The user can either program the keys with pre-defined functions, or program the keys with their own data. Up to 128 bytes of storage is available for the function keys. This can all be allocated to a single key, or shared between all 16.

Non printable characters, such as ENTER, or control codes can be entered by pressing the GRAPH key with the ENTER or CTRL-code keys.

Insert the (backup) master disc.

From DOS, type FKEY, and press "ENTER"

The screen will display the function key menu:-

```
   FUNCTION KEY UTILITY V1.X

   1. Display function keys

   2. Program function keys

   3. Set standard defaults

   4. Return to DOS
   select option....
```

The program is self prompting, and the user is guided through the program by a series of prompts.

When programming the function keys (option 2), the string of new data is terminated by pressing "ENTER". Existing data for a particular key may be retained by pressing the "ENTER" key without entering any data. The existing key data will then be copied into the "New Data:-" column.

c) DOSCOPY

The purpose of this utility is to allow you to easily transfer DOS tracks from one disc to another. This is particularly useful when using discs which contain programs written for the original Einstein (TC01) computer. Einstein 256 allows you to run programs written for Einstein, but the DOS is slightly different, in that Einstein has its BACKUP and COPY programs on disc, whilst Einstein 256 has them in ROM.

Because of this, Einstein's DOS 1.31 won't allow you to access the ROM based utilities. However, you can transfer Einstein 256's DOS 1.4X to your Einstein disc with DOSCOPY. When using "Einstein Discs", we strongly recommend that you do this! Proceed as follows:

Insert the master (backup disc).

From DOS, type DOSCOPY and press "ENTER"

The screen will then display the message:-

```
DOSCOPYV1.XX

Source Drive (0-1 or X)?
```

You should then type in the number of the drive where your master disc will be, usually drive 0, and press ENTER. DOSCOPY will then ask you to specify the destination drive. You should then type in the number of the drive where the disc onto which the DOS is to be copied will be, usually drive 0. Press the ENTER key. The disc "in use" lamp will light briefly, and DOSCOPY will then ask you to "put in DESTINATION disc, and press enter". On pressing ENTER, the DOS will be copied onto the disc.

d) TAPE.COM

The purpose of this utility is to allow the user to read program, or data cassette tapes, and load them into RAM. By using the SAVE command in DOS, the program/data can be transferred to disc.

Loading Tape Based Programs/Data

Insert the cassette into the cassette recorder, and ensure that the tape is at the beginning. Connect your tape recorder and (refer to the instructions supplied with the recorder) to the CASSETTE socket on the rear of Einstein 256. You should use a stereo lead, fitted with 3.5mm short reach stereo jacks, even when using a mono recorder. Set the playback volume, initially, to half-way.

Insert your Master Disc and from DOS type TAPE, and press ENTER. After a short time, the screen will display the following message:-

```
CASSETTE TAPE READ PROGRAM V1.X (c) 1986

START CASSETTE AND PRESS ENTER
```

Press PLAY on your recorder and then press the ENTER key on the computer.

If the tape loading is successful, the screen will flash the message LOADING. If after about 30 seconds the LOADING message does not appear, it probably means that the playback volume setting is incorrect. Start again!

To exit the TAPE program, press the BREAK key. Restart the program by typing TAPE, and press ENTER.

Rewind your tape, and repeat the above procedure with a different volume setting. (Hint, start at, say ⅓ volume setting, and gradually increase at each try).

When the tape has been successfully loaded, the message

```
PROGRAM LOADED: X BLOCKS
```

will be displayed. To save the program on disc, type SAVE X <filename.extension> , and then press ENTER. The X after SAVE is the number of blocks shown by the TAPE program. The <filenam.ext> should be exactly the same as that shown in the instructions that come with the program on cassette.

NOTE: Only use tape based programs intended for use with Einstein 256. Programs for other machines will not work.

CHAPTER 6

INTRODUCING EBASIC

Einstein 256 differs from most low cost computers in that it does not have BASIC resident within the system. This is possible since your Einstein 256 has a built-in disc drive, and BASIC can be loaded in a few seconds. Such a technique would take many minutes on a tape based system. This approach makes your Einstein 256 much more flexible when using other languages or other versions of BASIC.

The language used on Einstein 256 is an enhanced version of the popular Tatung/Xtal BASIC 4, used on the Einstein computer. It is known as EBASIC, and you will find this as EBAS.COM on the master disc supplied with the computer.

In order to use EBASIC, it must first be loaded into the computer. The examples in the tutorial which follows assume that EBASIC is already loaded. To load EBASIC, from your backup master disc, follow the procedure outlined below:-

i) Insert the master disc. (You haven't forgotten to make a backup, have you? If so, refer to Chapter 1.)

ii) Switch on.

iii) Wait for the screen to display:

```
        *** Einstein 256 ***

  EDOS 3.X         (c) 1983/6
  0:
```

iv) Type EBAS, then press the ENTER key.

v) When BASIC is loaded, the screen will display:

```
        *** Einstein 256 ***

  EBASIC 4.X       (c) 1983/6
  Size 41852
  Ready
```

BASIC or other programs can only be stored in memory as long as the computer remains powered.

If you wish to store them on a more permanent basis, they must be stored on disc. In EBASIC, this is accomplished by using the SAVE command, explained overleaf.

Suppose you wish to SAVE a program called EXAMPLE.

1. Load BASIC and key in your program.
2. Ensure that the write protect tabs are in the "write" position.
3. Type SAVE "EXAMPLE" then press ENTER (The "" marks are important).
4. The activity lamp on the disc will briefly illuminate, indicating that your program is being saved.
5. When the "Ready" prompt appears, key DIR (directory) then "ENTER" and you will see that your program has been entered in the directory as "EXAMPLE".XBS. The .XBS indicates that your program is a BASIC file.

In general saving BASIC programs takes the general form:-
SAVE " filename " "ENTER"

NOTE:
i) The program name can be up to 8 characters in length, and is devised by the user. The program name can be any combination of ASCII characters, except those greater than 127, and *:; .,"=?

ii) Further details of other facilities available with the SAVE command are given in a later chapter.

iii) Any blank disc must be FORMATTED before it can be used to store data (i.e., save programs).

Loading BASIC programs

1. To load a BASIC program from disc into memory, without executing it, type:-
LOAD " filename " "ENTER"
e.g. LOAD "EXAMPLE" "ENTER"

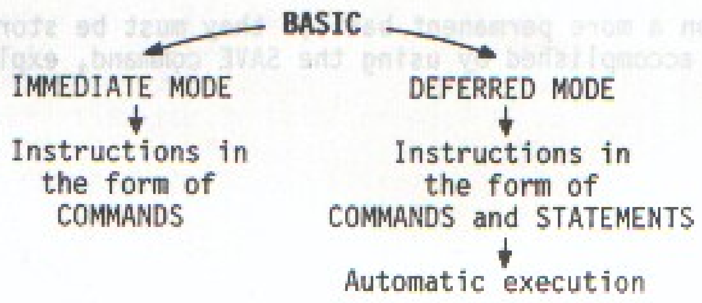
2. To load, and execute a BASIC program from disc, type:-
RUN " filename " ENTER

If the computer has been switched off or re-set since loading BASIC then it will be necessary to re-load it as before.

THE LANGUAGE

BASIC consists of a series of COMMANDS and STATEMENTS. The commands/statements are "English Like" words which represent specific functions. These are known as KEY, or RESERVED, words.

The first principle to appreciate is that BASIC operates in two quite separate MODES, these being known as IMMEDIATE MODE and DEFERRED MODE.



BASIC

IMMEDIATE MODE

Instructions in the form of COMMANDS

DEFERRED MODE

Instructions in the form of COMMANDS and STATEMENTS

Automatic execution

IMMEDIATE MODE

Immediate mode (sometimes known as Command or Direct mode) is such that the computer executes commands DIRECTLY they are entered; in other words an immediate reaction to the instructions given.

The following example illustrates a direct mode operation and uses the computer like a calculator.

Example - type in the following exactly as shown:

```
          PRINT 7*9
```

Now press the ENTER key

The * is used as a multiplication sign and the result should immediately appear on the screen, on the next line of text below, as 63.

NOTE: Pressing the ENTER key informs the computer that you have finished typing, and the instruction is ready for execution. The computer then acts accordingly.

IMPORTANT
You are advised to try the "keyboard trainer" program (see Chapter 4) before typing in a program. You will then be more familiar with the keys and their functions and effects.

Care must be taken to ensure all the example exercises are typed in as printed. Any slight mistake or deviation from the given format may result in the computer displaying an error message, or a different result. If a mistake occurs check that the example has been typed correctly; a punctuation mark, character, or space out of position or context may cause a deviation from the required result.

If a mistake is made, use the "cursor control" and INS/DEL keys to correct, as shown before (i.e. edit the mistake).

There are various error messages given by the computer to the user and more information concerning them is given later.

Immediate Mode Examples (Calculating)

Normal calculating operations can be conducted using the following mathematical and relational operators:

MATHEMATICAL		RELATIONAL
	() Parenthesis	
Descending order of precedence	Raise to power (exponentiation)	
	* Multiply	= Equal to
	/ Divide	
	+ Add	
	- Subtract	

The mathematical operators are listed in order of precedence for multiple operator calculations. Further details of OPERATORS are given in a later chapter.

Type the following examples **EXACTLY** as printed and press the ENTER key at the end of each one.

```
Ex. 1     PRINT 8*7     The result should be 56
Ex. 2     PRINT 9/3     The result should be 3
Ex. 3     PRINT 7+4-2   The result should be 9
Ex. 4     PRINT 33-4*6
```

The order of precedence is:
i) 4*6 = 24 (multiplication first)
ii) 33-24 (subtraction next)
iii) Answer = 9 (result)

Ex. 5 **PRINT (6+8)-6/2** ENTER

The order of precedence here will be:

i) (6+8) = 14 (brackets first)
ii) 6÷2 = 3 (division next)
iii) 14-3 (then subtract)
iv) Answer = 11 (result)

Before continuing, type **CLS** and then key **ENTER.** This is a command used in BASIC to clear the screen and HOME the cursor. (The abbreviation **CLS ENTER** is adopted to represent this process in the following text).

DEFERRED MODE

Deferred mode is such that instructions in the form of numbered lines containing commands are typed in but no action is taken on them until the RUN, CHAIN or GOTO commands are entered. Instructions with related line numbers are known as STATEMENTS and are then executed automatically when the RUN command is entered. A list of numbered statements (instructions) in sequence is known as a program. Try the following program. First type NEW, then press ENTER, to clear out old programs.

Example

```
                  10 PRINT 8+7
                  20 PRINT 5-4
 LINE NUMBERS     30 PRINT 6*3
                  40 END
```

Type in the above program, exactly as shown. After each line, press the ENTER key.

You will observe that the computer does not act upon these instructions and therefore no results appear.

Type in LIST and press ENTER. This command lists the instructions stored in memory on the display.

Now type in RUN, followed by ENTER to execute the program.

The computer now automatically executes the instructions given and produces the three results of 15, 1, 18, at the beginning of consecutive lines.

If you have loaded, or RUN a BASIC program previously, clear out the old program by typing NEW, then press ENTER, before typing the example.

To summarise:-

a) IMMEDIATE MODE is executed immediately after ENTER (manual execution)

b) DEFERRED MODE allows for automatic execution of a sequence of instructions known as a PROGRAM which is activated by RUN.

c) Editing and corrections are done in Immediate Mode.

GRAPHICS

Here we illustrate some of the graphics capabilities other than the "key" based graphics characters. Type each line followed by the ENTER key. Before you start, clear the screen by typing CLS, followed by ENTER.

First type RST ENTER.

Example 1 DRAW 50,50 TO 150,50,

Example 2 DRAW 150,50 TO 75,100,2

Example 3 DRAW 75,100 TO 50,50,2

This should now have produced a triangle on the screen display with two sides dotted. Each pair of numbers represents the co-ordinate of individual pixels (points) on the screen.

Clear the screen using CLS ENTER

Example 4 ELLIPSE 175,100,65 (for 525 line ELLIPSE 175,100,65,1.167)

This should have drawn a circle on the screen (a circle is a special case of an ellipse).

Example 5 Now clear the screen and try:-

POLY 7, 175, 100, 50

(For 525 line standard POLY 7,175,100,50,1.167)

This should put a seven sided polygon at a point 175,100 with a radius of 50 pixels. See also "POLY" in the reserved words definitions in Chapter 12.

NOTE: For 625 lines the commands should be typea in as shown. When using 525 lines, NTSC standard, as in Eastern countries and America, the ELLIPSE and POLY commands should be keyed in as shown in brackets in order to obtain a regular figure. If this is not done, an ellipse, or irregular polygon will result. - See ELLIPSE and POLY commands for further details.

COLOUR

An image has three properties:-

1. **Hue**
2. **Saturation**
3. **Luminance**

1. Hue is the actual colour itself red, green, blue etc.

1.1 There are 512 colour options available in your Einstein 256. Colour is selected from a pallette of 16 colours. (Except for the high resolution graphics mode) when 256 colours are available for each pixel. Each colour in the pallette is allocated a code from 0 to 16.The colours available at switch on are shown in table 1 .)

2. Saturation is the amount of colour of a particular hue e.g. red and pink may have the same hue, but will have different saturation.

3. Luminance is the brightness of the image.

A colour monitor, or colour TV responds to all 3 parameters. A monochrome monitor, or monochrome TV (a green, or amber screen monitor is a monochrome device) responds only to the brightness of the image.

It is possible for two different colours to have the same brightness values. Two such colours may be easily distinguished on a colour display, but will be completely indistinguishable on a monochrome display.

The colours should be chosen to give the best luminance contrast. i.e. choose colours which have high luminance values to contrast those with low ones.
The following BASIC program will illustrate the point. Just key in the program and run it.

```
  5 REM COLOUR BARS
 10 SCREEN 1
 20 BCOL1
 30 TCOL6
 40 GOSUB220
 50 TCOL12,0
 60 GOSUB220
 70 TCOL9,1
 80 GOSUB220
 90 TCOL7,0
100 GOSUB220
110 TCOL3
120 GOSUB220
130 TCOL8,1
140 GOSUB220
150 TCOL5
160 GOSUB220
170 TCOL13
180 GOSUB220
190 TCOL15,1
200 GOSUB220
210 END
220 PRINT MUL$(CHR$(&FF),40);
230 RETURN
```

The **BCOL** command is used to change the Backdrop COLour on the screen depending on the colour code number specified from the following table.

TABLE 1 - Default Colour Pallette

0 - Transparent	8 - Medium Red
1 - Black	9 - Light Red
2 - Medium Green	10 - Dark Yellow
3 - Light Green	11 - Light Yellow
4 - Dark Blue	12 - Dark Green
5 - Light Blue	13 - Magenta
6 - Dark Red	14 - Grey
7 - Cyan	15 - White

Type in the following examples using the ENTER key after each one. Observe the effect on the screen.

```
BCOL 3
BCOL 6
BCOL 10
BCOL 13
BCOL 4
```

Each example changes the colour of the screen in accordance with the colour code number specified. Example 5 returns to the original background colour.

Now clear the screen again (CLS:ENTER).

The **TCOL** command allows the colour of characters and background colour of the character cells to be specified independently of the screen backdrop colour.

Type in the following examples, keying ENTER after each line.

Example 1 TCOL 1,15 displays BLACK character on a white cell background

Example 2 TCOL 11,6 displays YELLOW characters on a RED cell background

Example 3 TCOL 3,13 displays GREEN characters on a MAGENTA cell background

Example 4 TCOL 15,0 - returns to normal display setting (WHITE characters on TRANSPARENT cell background).

SOUND

Various options are available for sound and music; details will be found in Chapter 14. For the moment type in the example given below followed by the ENTER key and observe what happens.

Example **BEEP**

Now clear the screen.

Examples

NOTE: After trying out each of the examples, below, clear the program from memory by typing NEW, followed by pressing the ENTER key.

N E W : ENTER

In the following examples type each line followed by ENTER key. Observe the results in each case.

Example 1

```
10  PRINT 7-3
20  PRINT 9-6/2
30  PRINT (10+2)-4*2
40  END
RUN
```

NOTE: To save time you can type ? instead of PRINT. ? is an abbreviation for the PRINT command.

The results of this program should appear as 4,6,4 on consecutive lines. When execution is complete type in NEW and key ENTER. This will clear the existing program from memory.

Example 2

```
5   REM SIGHT
10  CLS
20  BCOL 12
30  GCOL 6,0
40  DRAW 70,100 TO 170,100,3
50  DRAW 120,35 TO 120,165,3
60  ELLIPSE 120,100,50
70  END
RUN
```

The results of this program should display a red circle with the horizontal and vertical diameters shown as red dotted lines. On completion key NEW ENTER.

```
Example 3     10  CLS
              20  BCOL 3
              30  BEEP
              40  DRAW 70,100 TO 170,100,
              50  VOICE 1,0,-12,0,1,20
              60  A$= "G2AGFE5CG3AGFE5C"
              65  TEMPO 5
              70  MUSIC "V1"+A$
              80  BCOL 13
              90  BEEP
             100  DRAW 120,34 TO 120,166,0
             110  MUSIC "V1"+A$
             120  BCOL 10
             130  BEEP
             140  ELLIPSE 120,100,50
             150  MUSIC "V1"+A$
             160  BCOL 4
             170  BEEP
             180  END
             RUN
```

The following examples will illustrate some of the spectacular effects which can be produced using the graphics capabilities. It is not intended that the inexperienced user should understand how these work at this particular time, but rather they are a demonstration of possibilities available once a fuller appreciation of the BASIC language has been obtained.

Follow the instructions carefully:-

1. Clear the screen.

2. Clear the previous program from RAM. (NEW)

3. Type each example program as before.

```
5 REM CIRCLES
10 SCREEN 6
20 GCOL RND (14+2)
30 ELLIPSE RND (511), RND (191), RND (50), 0.59
40 GOTO 20
```

This program should display circles of random size, colour, and position on a black background. To stop the program, press the SHIFT and BREAK keys simultaneously. Key NEW ENTER to clear the memory.

```
5 REM POLYGONS
10 SCREEN 6
20 GCOL RND (14+2)
30 POLY RND (5+3),RND(511),RND(191,RND(50),0.59
40 GO TO 20
```

This program should display polygons of random size, colour and position on a black background. To stop execution, key SHIFT BREAK. Then KEY NEW, ENTER.

SUMMARY

a) BASIC - Consists of the following:
  i) COMMANDS (NEW, CLS) etc
  ii) STATEMENTS
  iii) RESERVED WORDS

b) TWO MODES -
  i) IMMEDIATE - Immediate response
  ii) DEFERRED - Automatic execution,
    delayed until the run command is used

c) EXAMPLES, illustrating:
  i) CALCULATIONS
  ii) SIMPLE GRAPHICS
  iii) SIMPLE COLOUR
  iv) SIMPLE SOUND
  v) MORE COMPLEX GRAPHICS/COLOUR

d) SAVING PROGRAMS

PROGRAMS

You should by now be familiar with some of the capabilities of the EINSTEIN 256. The facilities are accessed in either direct mode or deferred mode through the use of programs.

REMEMBER - a PROGRAM is a sequence of instructions which direct the computer to perform a particular task.

PROGRAM FORMAT

Each line of a program is numbered. It is common practice to begin at 10 and progress in increments of 10 (consecutive line numbers being 10, 20, 30, 40 etc.)

This is not absolutely necessary, as any numbers from 1 to 65,535 may be used, provided they are in some form of ascending sequence (e.g. increments of 10 or 100).

The purpose of incrementing in this manner is to allow space for the insertion of extra lines. This may be necessary either to improve or alter a program (or if a line has accidentally been omitted!). Therefore there is the flexibility of another 9 optional lines between each existing line of a program. Lines of a program are referred to as statements.

PROGRAM STORAGE IN RAM

When a program is typed in it is stored in RAM until either it is cleared or the power is turned off. To input another program a special command must be used which clears the existing program from memory. If this is not done, the new program overlaps the old program and sections of each will intermingle depending on line numbers.
The NEW command, as used earlier, provides this facility. NEW is entered immediately prior to commencing another program, thus clearing the memory of any existing program.

RESERVED WORDS

The following RESERVED WORDS are very common in programs:-

REM
END
RUN
PRINT

REM - This is an abbreviation for **REM**ARK. Any line preceded by REM is not part of the program, but simply an aid to the programmer as a title, or comment.

REM is optional and can be omitted to save memory space.

END - This terminates the program involved. In complex programs the END statement is not necessarily the final statement in the listing. However, when the end of the program corresponds to the last statement of the program the END command may be omitted.

RUN - This is the "magic word" which makes everything work. RUN instructs the computer to commence automatic execution of a program.

PRINT - This is an OUTPUT command and causes information/data to be transferred to a particular output device such as screen or printer, etc. For the moment we are only concerned with output to the screen.

Some of the previous examples have used the PRINT command but we will now examine the format in a little more detail.

i) If the material is in the form of an arithmetic expression (e.g. 2+7 or 3*5 etc.) the computer will evaluate the expression and then PRINT the result only. This has been seen in previous examples.

ii) If the material is "TEXT" to be printed to the output device it must be enclosed by double quote marks (""). Any displayable character within the quotes will be output to the particular device in use at the time, e.g. the screen.

Example: PRINT "COMPUTERS ARE QUICK"

This will cause COMPUTERS ARE QUICK to appear on the screen.

Look at the following program:-

```
10  REM A PROGRAM TO PRINT A NAME AND ADDRESS
20  PRINT "THE PRIME MINISTER"
30  PRINT "10 DOWNING STREET"
40  PRINT "LONDON"
50  PRINT "ENGLAND"
60  END
TYPE RUN
```

As before, use NEW: ENTER to clear out the previous program, and CLS ENTER to clear the screen.

iii) The spacing in PRINT expressions can also be controlled using a selection of SEPARATORS. SEPARATORS are symbols which are positioned after print expressions to indicate specified output formats.

a) The ; (semi-colon) separating two expressions indicates that the second expression will be printed immediately following the first, without a space.

Example: PRINT "COMPUTER" ; "JARGON"

This will appear as COMPUTERJARGON

c) If spaces are left within the quote marks they will be considered to be character positions when the print out is executed.

Example: PRINT "COMPUTER " ; "JARGON"

This will now appear as COMPUTER JARGON

b) The , (comma) at the end of one expression indicates that printing will re-start at the next TAB POINT.

Tab positions are simply pre-set positions on each line of the screen grid and are 10 columns apart as shown in Fig.6.1. The preset positions can be altered by using the **ZONE** command.

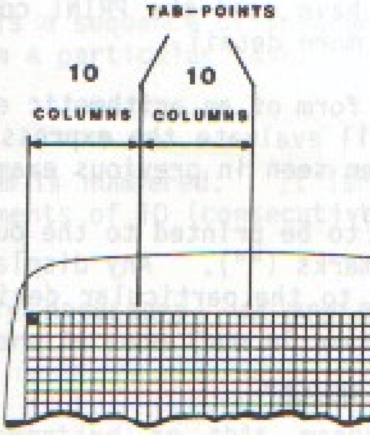


Fig 6.1 Tab Positions

c) When no separator is placed after a PRINT expression, then EINSTEIN 256 generates an automatic carriage return and line feed, so that subsequent output will appear on the next line.

Example:

```
10 PRINT "THIS IS THE YEAR"
20 PRINT "OF HALLEY'S COMET"
30 PRINT "WILL NEXT APPEAR IN 2060"
```

Before typing in the above program, clear any old program by typing NEW, followed by "ENTER". When you RUN the program, each print expression will appear on a new line.

ERRORS/BUGS

All too often a program may be typed into the computer but when the RUN command is issued nothing happens. This usually means there is a fault somewhere in the program listing and under normal circumstances the computer will display an ERROR MESSAGE. This message gives some indication of the error involved and which area of the program it affects.

When a program will not run correctly because of errors, we say it has a BUG in it. We use the term **DE-BUGGING** to indicate tracing and correcting errors.

ERRORS/SYNTAX

The most common category of errors in program listings are due to syntax. This means that a particular statement or command has not been written in its correct format. BASIC must always be formatted precisely as indicated in the handbook. Some of the more common syntax errors include:-

a) Incorrect use of punctuation marks.
b) Typing errors, ie. spelling mistakes.
c) Additions or exclusions to the format of a particular statement/command

A single punctuation mark out of place could prevent a whole program from running, therefore accuracy of format is absolutely essential in the listing.

A full listing of error messages, can be found in appendix A.

To exit a BASIC program whilst it is running, the BREAK key is used in conjunction with SHIFT (i.e. SHIFT-BREAK)

This may be required for various reasons, but perhaps the most common use is as follows:-

Occasionally, bad logic and structuring can cause a program to go into a continuous cycle of processing which does not end. We say that the program has gone into a loop.

The SHIFT-BREAK function is used to stop program execution and discover the cause of the unwanted loop. BASIC will generate a message indicating the line number when the BREAK occurred, This will usually be within the loop, so pointing to the problem area, and allowing the programmer to correct it.

The program HALTS at whatever line it is on at the particular moment when SHIFT-BREAK is used.

EDITING

EBASIC has extensive editing facilities. It has been specifically designed to make the task of program entry, and debugging more of a pleasure. Rather than the tiresome task it can be with other BASIC interpreters. Input lines can be up to 254 characters long. Full screen editing is supported, which means that any line, which is visible on the screen, can be corrected simply by positioning the cursor over the appropriate line, and making modifications to it, even if it occupies two, or more, rows on the screen. If modifying the line extends it so that it would oveflow into the next line, then tell the lines below move down a line, so as to create extra space for the modified line. When the line is suitably altered, the changes are entered into the program by pressing the ENTER key.

The functions available within the BASIC editor are the INS/DEL (insert/delete) key, the cursor control keys and the screen control codes. Ths screen control codes are fully explained in appendix.

Sometimes it is necessary to delete a character, or even to insert spaces to allow for extra characters. To delete a character, position the cursor, using the cursor control keys, to the character immediately to the right of the unwanted character, and press the DEL key. The offending character will be deleted. The action of the DEL key is to delete the character which is to the left of the cursor.

In order to create space for extra characters, position the cursor at the point where the insertion is to take place. Press the SHIFT and INS/DEL key together, and a space will be inserted. The extra character (s) can then be typed in without overwriting existing data.

Don't forget to press the ENTER key to add your modifications to the program.

The Line Editor

The "line edit" mode is available primarily as an alternative for use with programs where the "screen editor" might not be quite so convenient.

In "line edit" mode the only editing functions which operate are as follows:-

a) The "cursor left" function on the cursor control key acts as a "back space" and deletes characters as it moves. All other functions of cursor control keys are non-operational.

b) CRTL-A transfers the screen contents to a printer.

c) Re-typing of a program line so as to overwrite the original line.

All other control functions and cursor movements as described in "SCREEN CONTROL CODES" and "BASIC EDITOR" are non-operational under LINE EDIT mode.

To use the LINE EDITOR carry out the following procedure:-

1) Type IOM 2,0

This activates an internal "switch" which then allows the option of selecting either LINE EDIT or SCREEN EDIT.

2) Type IOM 0,0

This now sets the "switch" to LINE EDIT mode (i.e. goes into line edit)

3) To return to SCREEN EDIT type IOM 0,1.

When line edit is no longer required IOM 2,1 is used to de-activate the internal "switch".

NOTE:

i) During "line edit" the prompt in front of the cursor becomes a horizontal arrow (—►) in direct mode.

ii) The arrow changes to a question mark (?) if an INPUT statement is used without a specified "prompt string".

SUMMARY

a) A PROGRAM is a list of instructions with:-
  i) Each line numbered
  ii) Increments of 10 commonly used

b) Programs are stored temporarily in RAM

c) Programs cleared from RAM by "NEW" command

d) Permanent storage of programs is on disc (backing store)

e) Introduction to:
  i) REM
  ii) END
  iii) RUN
  iv) PRINT and use of separators

f) Introduction to errors:
  i) BUG
  ii) SYNTAX ERROS
  iii) DEBUGGING

CHAPTER 7

SYSTEM "COMMAND KEYS"

CTRL-BREAK

Simultaneously pressing CONTROL and BREAK keys causes a transfer to the disc operating system from BASIC.

SHIFT-BREAK

Holding down the SHIFT key and then pressing BREAK halts program execution, preserving all variables, The CONT command can then be used to allow continuation of program execution if required, The message "Break in line..." is displayed on the screen when SHIFT-BREAK is used.

BREAK

The BREAK key halts program execution whilst it is held down, but execution continues when the key is released, (variables are preserved).

ESC

Pressing the ESCAPE key causes listing and tabulations to be aborted.

HARDWARE RESET

Pressing the CRTL;ALPHA LOCK, and GRAPH keys simultaneously causes a hardware reset, just as if the computer had been switched off, and on again. All program data and variables are lost. It should not normally be necessary to use this facility, except as a means of exiting some 'autobooting' programs, particularly games.

CHAPTER 8

NUMBERS AND STRINGS

There are two types of data used in EBASIC 4:-

NUMERIC DATA
STRING DATA

NUMERIC DATA

These can be whole numbers (integers), or floating point numbers (reals).

Integers

Integers are whole numbers without fractions or decimal points.

The number can be positive or negative and must fall within the range -32768 to +32767. (BASIC supports 16 bit integers).

Examples: 44, 6, -17, 32500, -29076

Numbers in the ranges -65535 to -32769, and 32768 to 65535, may be accepted by integer variables. In these cases the values are internally converted to fall in the range 1 to 32767 and -32768 to -1 respectively (otherwise 17 bits would be needed to store each number). Integers, in BASIC, are shown by the % sign after the variable name

Floating-Point Numbers

Floating point numbers can be whole numbers, or decimal numbers.

They can be positive or negative and if no sign is given the number is assumed to be positive.

The following are examples of floating point numbers which are also integers.

6 -14 2650 -21

The following examples include decimal points.

7530.23 0.7 7.4 -0.0008 -27.029

Commas must NOT be used in numbers otherwise a syntax error may result.

For storage purposes, floating point numbers are internally converted to scientific notation.

Scientific Notation

A number in scientific notation is expressed as a base number (mantissa) multiplied by 10 raised to a particular power of 10 (EXPONENT)

NUMBER = MANTISSA x 10^{EXPONENT} (Power of 10)

Example: 3113 = 3.113×10^3

In BASIC this "scientific notation" is as follows:-

3.265×10^4 is entered as 3.265E4 in BASIC

For a negative power:-

3.265×10^{-4} is entered as 3.265E-4 in BASIC
By using Scientific Notation very large and very small numbers are easily handled by the computer.

The exponent range is given below:-

MAXIMUM - 1.701411E38
MINIMUM - 0.940396E-38

Any computations yielding a result above or below these values will display a quantity error.

Four bytes are used to store numbers internally, one of which represent the "signed Exponent" and the other three the "signed mantissa".

The "exponent" can range from -38 to +38 whilst the signed mantissa can be up to seven digits, (any value outside this range will cause a quantity error to be displayed).

The full seven digits of a mantissa are used internally for calculations but are then rounded off to six significant figures for output. (Always use a seventh figure if known for accuracy even though only six are displayed).

Leading and trailing zeros are always suppressed on output. This avoids long trails of zeros either preceding or following a number.

In practice, very large numbers or very small numbers can be entered in "decimal" or "scientific notation" as required, but may be automatically output in scientific notation only.

Hexadecimal (HEX) Numbers
Hexadecimal numbers are to base 16, and are expressed using a combination of numbers in the range 0 to 9 and letters in the range A to F. The following table gives the Hexadecimal characters with decimal number equivalents.

HEXADECIMAL		DECIMAL	HEXADECIMAL		DECIMAL
0	-	0	8	-	8
1	-	1	9	-	9
2	-	2	A	-	10
3	-	3	B	-	11
4	-	4	C	-	12
5	-	5	D	-	13
6	-	6	E	-	14
7	-	7	F	-	15

Example: HEX B3 is equivalent to 179 decimal.

Evaluation $3 \times 16^0 = 3\ (16^0 = 1)$

$B \times 16^1 = 176$ (ie 11×16)

Total 179

Using HEX numbers:

1) An ampersand symbol (&) is used as a prefix to indicate Hexadecimal numbers. e.g: &1298 &A7 &1F34

2) When hexadecimal numbers are used in numeric expressions they are internally converted to a decimal number. The hexadecimal number must not exceed four digits. If more than four digits are entered only the LAST FOUR will be used.

Example:

&1F34 equivalent to 7988 decimal
&71F34 still equivalent to 7988 decimal because only the LAST FOUR digits are used. (i.e. the 7 is ignored).

STRING DATA

Strings are combinations of ASCII characters representing letters, numbers, and symbols.

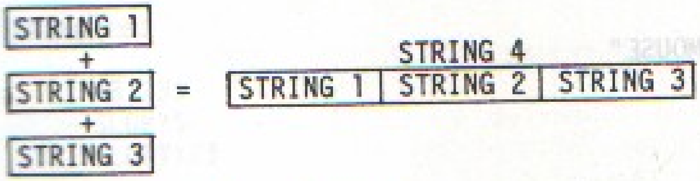
They are useful for storing names, titles, and text, but can also be used to hold numeric values.

A string can be any combination of up to 255 characters, usually shown in quotes.

Example: - "HELPLESS"
"*!ZK"
"1379.76" etc.

Concatenation of Strings

Strings can be CONCATENATED (i.e. strung together consecutively) using the "+" sign.



Example:
```
10 A$  = "MOUSE"
20 B$  = "TRAP"
30 C$  = A$ + B$
40 PRINT C$
RUN
```

This will give a display of MOUSETRAP.

Strings and Relational Operators

Strings can be compared using relational operators:-

```
   =,  ,=, =, ,
```

```
Examples:          "A"  "B"
                "SMIT"  "SMITE"
           "MOUSETRAP" = "MOUSETRAP"
```

The comparison is done character by character until a position is found in which the two differ. The "greater" string is the one whose differing character has the greater ASCII code. If no differences are found, but one string is longer than the other, the longer string is considered to be the greater.

Example 1

Character difference on comparison.

```
"STEPHEN"    "DAVID"
 ↑             ↑
ASCII 83    ASCII 68

.'."STEPHEN" is greater than "DAVID"
           "STEPHEN" > "DAVID"
```

Example 2

Character difference on comparison.

```
"ANDY"      "ANDREW"
   ↑             ↑
ASCII 89   ASCII 82

.'. "ANDY" is greater than "ANDREW"
            "ANDY" > "ANDREW"
```

Example 3

No character difference detected but one string longer.

```
"MOUSE"     "MOUSETRAP"

.'. "MOUSETRAP" is greater than "MOUSE"

           MOUSETRAP > MOUSE
```

VARIABLES

Variables are "names" used to represent values. The values are either assigned by the programmer, or as a result of calculations/operations within the program. Prior to being assigned a value, a variable is assumed to be zero.

A variable "name" can be a combination of letters or letters and numbers but the first character must always be a letter.

Examples:

VALID NAMES	INVALID NAMES
ZFBC	6BLF
SAP	*YXL
L93A7	!79B

Variables may be of the following types:-

NUMERIC - { FLOATING POINT - holds numbers
{ INTEGER - holds whole numbers.
STRING - holds strings.

Variable "types" are indicated by characters which suffix the variable "name".

i) Floating point variables are the default type, having no suffix after the variable name.

Example: ATC

ii) Integer variables are indicated by the presence of a % sign after the variable name.

Example: BX7%

iii) A string variable is indicated by the presence of a $ sign immediately following the variable name.

Example: ZUP$

EBASIC uses the first five characters only of a variable name. More can be entered (within the limits of the maximum line length) but characters after the 5th will be ignored by BASIC.

Examples:

AB123$
SLOPY$
SLOPYXZ$

All three are valid variable names but the BASIC would not distinguish between the 2nd and 3rd example because the first five characters of each are identical.

Care must be taken to ensure that variable names do not contain reserved words!

EXAMPLES:

TONE,LETTERS,PINCH,TERROR

All the above examples contain reserved words, as indicated below, and could cause problems!

TOne,LETter,pINCH,tERROR

It is advisable to keep variable names to two characters to avoid this problem, and commonly only one letter is used. (the only two character reserved words are IF,OR,LN,TO,ON,FN,PI)

ARRAYS

An ARRAY is in effect a list or table full of variables. There are NUMERIC arrays and STRING arrays.

In the case of a list each element of the list is numbered in order of appearance. The numbers are then used to refer to individual elements by inserting them immediately after the variable name.

Example: A(2)

This refers to the 3rd element of the array variable A. (Array subscripts start at 0)

This is known as SUBSCRIPTING the array, the numbers, or variables, within the brackets being the subscripts.

Lists have a single subscript and are known as one dimension arrays.

In the case of tables there would be two subscripts referencing individual elements in a similar manner to map or graph co-ordinates. These are known as two dimensional arrays.

Example: B(2,3)

The numbers indicate vertical column and horizontal row numbers relative to the element required.

EBASIC also caters for 3 dimensional arrays thereby giving 3 subscripts after the array name.

Example: B(2,0,1)

Arrays in higher dimensions can be supported by EBASIC, the limit being determined by the amount of memory available at any one time.

NOTE:
i) The array subscripts always number from zero.
ii) Arrays containing more than 10 elements must be dimensioned with a DIM statement to inform the BASIC how much space to allocate for it. DIM statements are explained in detail in a later section of this handbook.
iii)If you are unfamiliar with arrays refer to Chapter 8.

EXPRESSIONS AND OPERATORS

EXPRESSIONS

Expressions consist of:-

i) Numeric Variables
ii) Numeric Data
iii) String Variables
iv) String Data

These can be combined using arithmetic, logical and relational operators.

ARITHMETIC OPERATORS

The arithmetic operators in order of precedence are as follows:-

OPERATOR	OPERATION	EXAMPLE	MATHEMATICAL EXPRESSION
( )	Parenthesis	3*(2+6)	3(2+6)
	Exponentiation (raise to power)	5 2	5^2
*	Multiply	3*4	3x4
/	Divide	4/2	4÷2
MOD	Remainder	8MOD3	8÷3=2 rem.2
+	Addition	3+2	3+2
-	Subtraction	2-1	2-1

Example:

```
    BASIC
6*(2+6)/2 3-5
```

MATHS

$$\frac{6(2+6)}{2^3}-5$$

The order of operation in this expression is as follows:-

i) Brackets evaluation
ii) Powers evaluation
iii) Multiplication and Division evaluation
iv) Subtraction evaluation

MOD accounts for the remainder after Division -

Example:

5MOD3 would return a value of 2 (i.e. 5÷3=1 remainder 2)
7MOD2 would return a value of 1 (ie. 7÷2=3 remainder 1)
8MOD3 would return a value of 2 (ie. 8÷3=2 remainder 2)

The actual process undertaken to obtain these values is a follows:-

```
FOR ANY TWO VALUES 'X' and 'Y'
THEN XMODY = X-Y*INT(X/Y)
                       ↑
                 This section returns the largest integer (whole number)
                 less than or equal to the result of the division
                 computation.
```

Examples of MOD:

1) For a value of X=5 and Y=3

```
5MOD3 = 5-3*INT(5/3)
      = 5-3*INT(1.6666)
      = 5-3*1
           ∴    5MOD3 = 2
```

2) For a value of X=7 and Y=2

```
3) 7MOD2 = 7-2*INT(7/2)
         = 7-2*INT(3.5)
         = 7-2*3
           ∴    7MOD2 = 1
```

If the values of 'X' are negative then because of the INT function the results are sometimes unexpected. Using the same examples as above but with X=-5, and -7 respectively we would see the following:-

```
1) -5MOD3 = -5-3*INT(-5/3)
          = -5-3*INT(-1.6666)
          = -5-3*(-2)
              = -5-(-6)
           ∴    -5MOD3 = 1

2) -7MOD2 = -7-2*INT(-7/2)
          = -7-2*INT(-3.5)
          = -7-2*(-4)
          = -7-(-8)
           ∴    -7MOD2 = 1
```

Thus we can see the following comparison:-

5MOD3 = 2 BUT -5MOD3 = 1
7MOD2 = 1 AND -7MOD2 = 1

RELATIONAL OPERATORS

Relational operators are commonly used with IF statements for comparisons and the evaluation of conditions. EBASIC uses the following:-

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

LOGICAL OPERATORS

Again logical operators are commonly used with IF statements in conjunction with relational operators. EBASIC contains the following (listed in descending order of precedence).

NOT, AND, OR, XOR (Exclusive-OR)

Example: `10 IF(X+Y-Z) 3 AND Y =20 THEN 100`

This performs a bit-by-bit logical AND of each numeric expression. If both bits are set to a '1' then the result bit is set to a '1'. If either bit is a '0' the result bit is set to '0'.

NOTE: Although expressions involving relational and logical operators are normally used within IF statements, they can also be used within arithmetic expressions (a relational expression returns a value of -1 if it is TRUE, and 0 if FALSE).

In some cases, quite a lot of space can be saved:-

Example: `IF X  15 THEN A=0:ELSE A=1`

This could be replaced by:- `A = -(X 15)`

CHAPTER 9

PROGRAM STRUCTURE

There are three fundamental facilities within the structure of a program:

1. SEQUENCE
2. BRANCH
3. LOOP

1. SEQUENCE

This is the ability to execute instructions one after another in a continuous sequence. All the example programs used previously exhibit this facility.

2. BRANCH

This is the ability to execute one of several sets of instructions within a program as a result of decision making.

Branching is controlled three ways in BASIC:

i) IF-THEN command
ii) GOTO command
iii) Subroutines (GOSUB command)

Further details of these three commands can be obtained from Chapter 10.

i) The **IF-THEN** command is a conditional branch and has the following format:

IF "a condition exists" **THEN** "execute a given set of instructions". Fig 9.1 depicts a practical example of where a decision is called for.

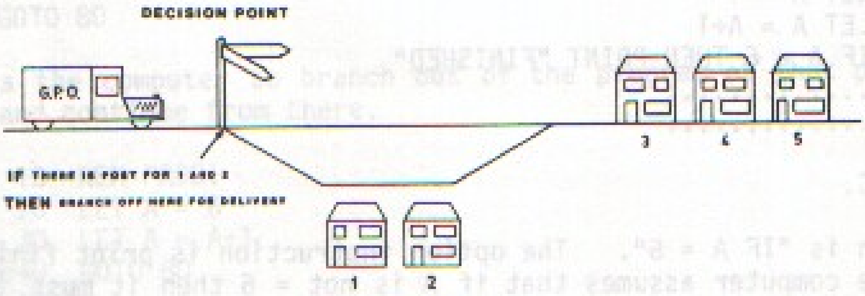

DECISION POINT
G.P.O.
IF THERE IS POST FOR 1 AND 2
THEN BRANCH OFF HERE FOR DELIVERY
1
2
3
4
5

Fig 9.1

This can be compared with decisions we make in our everyday lives. For example:-

If "it is raining" THEN "wear a raincoat"

CONDITION OPTION

If the condition is true, the option instruction is executed before continuing. If the condition is not true, the option instruction is ignored and you continue. The assumption is made therefore that if it is not raining you do not wear a raincoat.

The flowchart format would appear as shown in Fig 9.2. The "diamond" box is the standard method of indicating a decision point in a flowchart, thereby creating a branch (i.e. a question box).

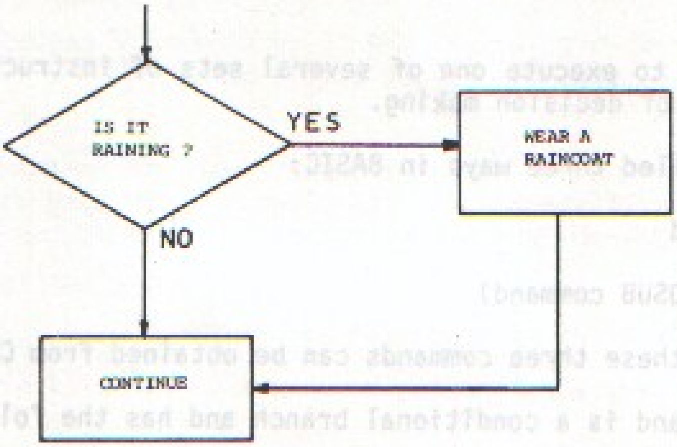


Fig 9.2

Example

```
10  REM EXAMPLE
20  LET A = 1
30  LET A = A+1
40  IF A = 6 THEN PRINT "FINISHED"
50  ..............
60  ..............

  ETC.
```

The condition is "IF A = 6". The option instruction is print finished (see Fig 9.3) The computer assumes that if A is not = 6 then it must ignore the PRINT instruction and carry on to the next listed instruction in the normal sequence.

iii) SUBROUTINES

Quite often in a program, a particular sequence of instructions may occur several times. To avoid reproducing the sequence each time, the instructions are grouped separately as a SUBROUTINE. The computer can simply go to the subroutine from any line in the program and return to the same place to continue the main program. Subroutines are in fact a mini-program and could operate in isolation if required.

The GOSUB statement

The "GOSUB" command followed by a line number is used in a program to inform the computer that it should access a subroutine at that point, e.g. GOSUB 90 accesses a subroutine which starts at line 90.

The final line of any subroutine must contain the RETURN instruction. This sends the computer back into the main stream program at the line immediately following the GOSUB instruction. Thus, if line 60 was the GOSUB instructions, the computer returns to line 70.

There can be any number of subroutines attached to a program and each individual subroutine can be accessed any number of times.

Example 1

We can make a comparison here with a touring holiday which involves the use of two different types of accommodation according to location.

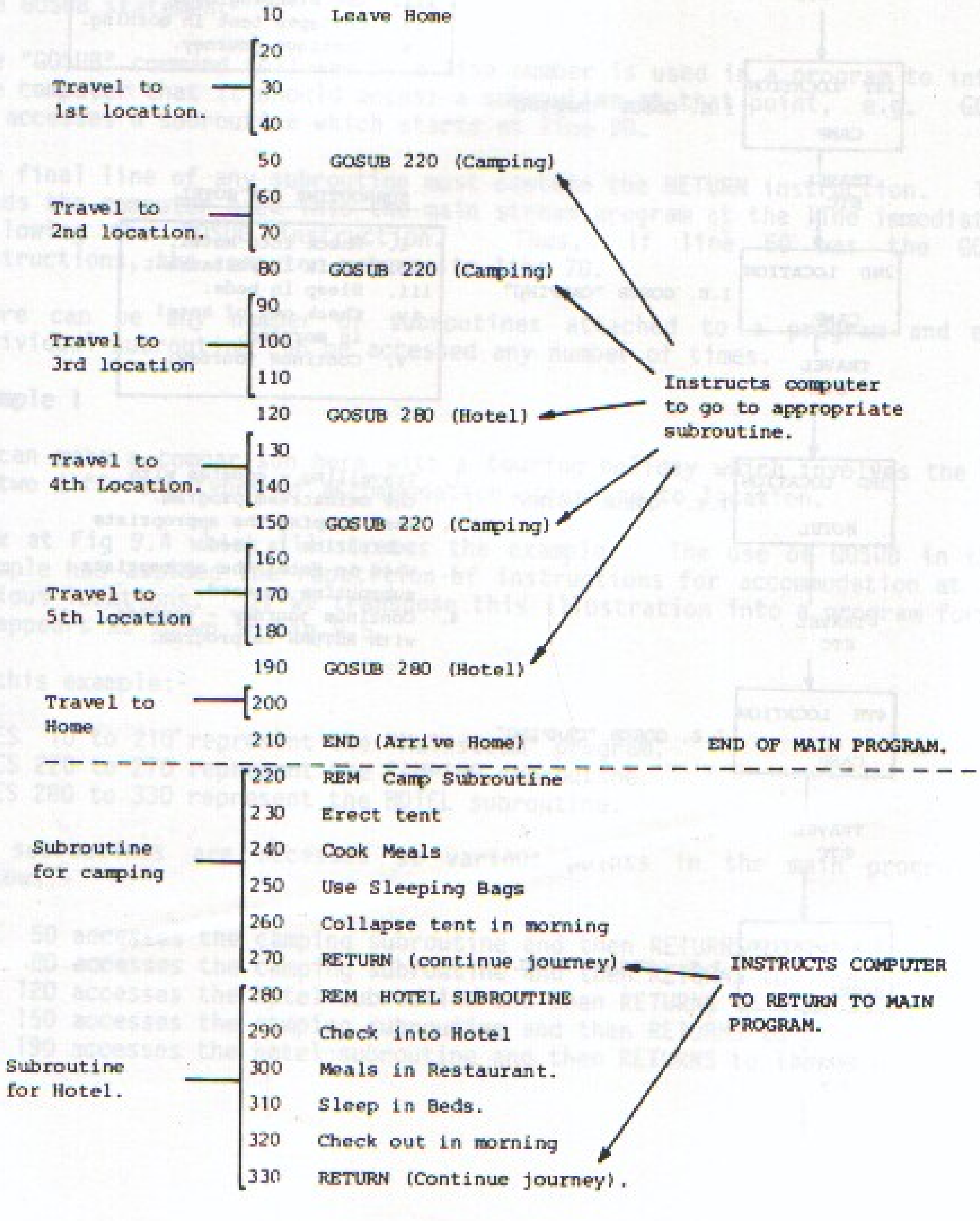
Look at Fig 9.4 which illustrates the example. The use of GOSUB in this example has avoided the repetition of instructions for accommodation at the various locations. If we transpose this illustration into a program format it appears as shown in Fig 9.5.

In this example:-

LINES 10 to 210 represent the "Mainsteam" program.
LINES 220 to 270 represent the CAMPING subroutine.
LINES 280 to 330 represent the HOTEL subroutine.

The subroutines are accessed at various points in the main program as follows:-

LINE 50 accesses the camping subroutine and then RETURNS to line 60.
LINE 80 accesses the camping subroutine and then RETURNS to line 90.
LINE 120 accesses the hotel subroutine and then RETURNS to line 130.
LINE 150 accesses the camping subroutine and then RETURNS to line 160.
LINE 190 accesses the hotel subroutine and then RETURNS to line 200.



```
10   Leave Home
20
30
40
50   GOSUB 220 (Camping)
60
70
80   GOSUB 220 (Camping)
90
100
110
120  GOSUB 280 (Hotel)
130
140
150  GOSUB 220 (Camping)
160
170
180
190  GOSUB 280 (Hotel)
200
210  END (Arrive Home)
220  REM  Camp Subroutine
230  Erect tent
240  Cook Meals
250  Use Sleeping Bags
260  Collapse tent in morning
270  RETURN (continue journey)
280  REM  HOTEL SUBROUTINE
290  Check into Hotel
300  Meals in Restaurant.
310  Sleep in Beds.
320  Check out in morning
330  RETURN (Continue journey).
```

Fig 9.5

Example 2

Type in the following program and observe the results on the screen.

```
 10  REM MAGICIANS POEM
 20  CLS
 30  PRINT
 40  PRINT " ", "MAGICIANS POEM"
 50  BCOL2
 60  PRINT
 70  GOSUB 280
 80  PRINT
 90  PRINT " ", "ABRACADABRA"
100  PRINT
110  GOSUB 280
120  PRINT
130  BCOL10
140  PRINT "WHITE RABBITS AND MICE"
150  PRINT
160  GOSUB 280
170  PRINT
180  BCOL13
190  PRINT "ALL MIXED TOGETHER"
200  PRINT
210  GOSUB 280
220  PRINT
230  BCOL7
240  PRINT "IN A BUCKET OF RICE"
250  PRINT
260  GOSUB 280
270  END
280  REM SUBROUTINE
290  FOR I = 1 TO 30
300  BCOL8
310  PRINT"*";
320  NEXT I
330  BEEP
340  RETURN
```

In this example the "mainstream" program is 10 to 270, and the subroutine is 280 to 340.

The action of the subroutine is to print a line of "*" accompanied by colour change and a noise, following each line of print on the screen. This occurs five times during the program, thus illustrating the advantage of using a subroutine for this purpose.

3. LOOP

This is the ability to loop back and repeat certain sequences of instructions as required.

Fig. 9.6 illustrates a practical application of the loop principle

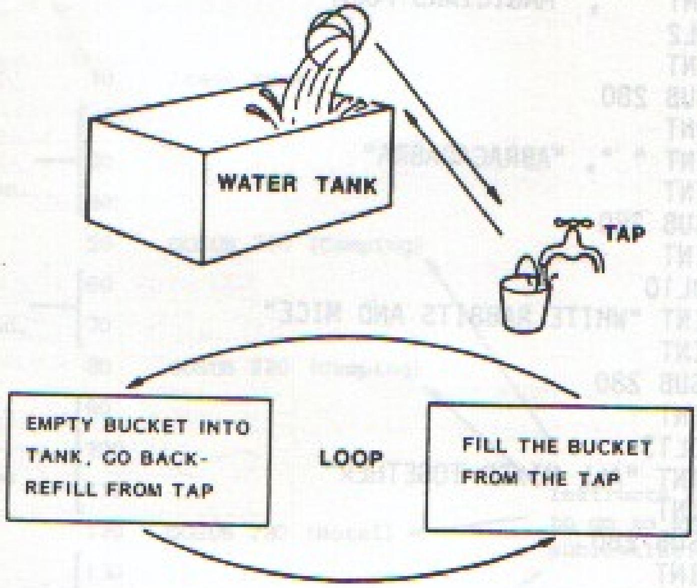

WATER TANK
TAP
EMPTY BUCKET INTO TANK, GO BACK-REFILL FROM TAP
LOOP
FILL THE BUCKET FROM THE TAP

Fig. 9.6

Loops in programs can be set up in various ways. The two most commonly used methods involve:

i) Combinations of IF-THEN and GOTO statements.
ii) FOR-TO-NEXT statement.

i) The following program example and flowchart (Fig 9.7) illustrates the IF/THEN/GOTO combination:-

```
 10   LET I = 0
 20   PRINT "?"
 30   LET I = I+1
 40   IF I <256 THEN GOTO 20
 50   END
```

LINE 40 is the operative line of the loop. It asks if I is less than 256 and if so instructs the computer to go back to line 20 and repeat the sequence of instructions again.

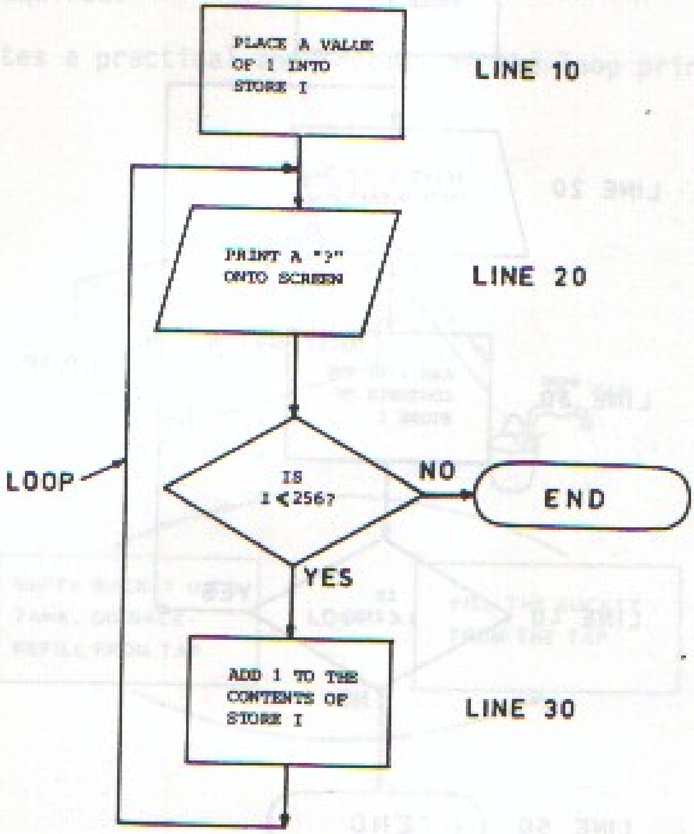


Fig. 9.8

LINE 10 Sets the lower and upper limits for the values to be stored in I.
LINE 20 Instructs the computer to print a "?".
LINE 30 Instructs the computer to increment I and repeats the previous instructions. (Incrementing will be one unit at a time unless otherwise stated in the program.)

The loop will continue until the upper limit of I set in the first statement is reached; at that point the program will end. Note that on this occasion the question to determine the value of I and compare it with the maximum value is inherent and assumed as a result of the FOR-TO-NEXT statement. Greater efficiency is therefore achieved.

The following analogy (fig 9.9) may help to explain the FOR-TO-NEXT loop further. Imagine a "window cleaner" is to clean the windows of each house in a street of 10 houses.

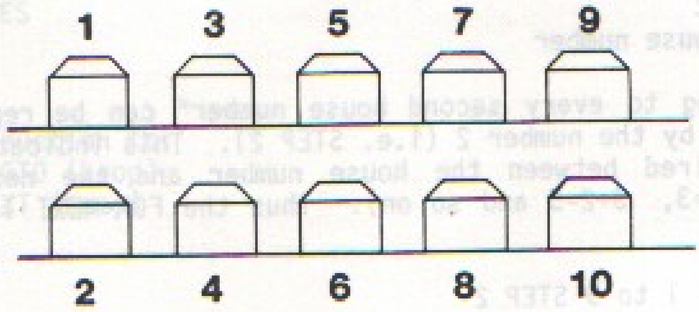


Fig 9.9

For each house he will carry out the same process.

a) Clean lower windows)
b) Clean upper windows) "cleaning process"
c) Collect payment)

Thus, for house number 1 he will carry out the "cleaning process", then for house number 2 he will REPEAT the "cleaning process", then for house number 3 and so on. In other words he repeats the "cleaning process" for each house number in turn, working his way along the street.

If we write this down as a sequence of instructions, it appears as:

1. FOR each house number from 1 to 10.
2. clean the lower windows
3. clean the upper windows
4. collect the payment
5. move to the NEXT house number

We can see the formation of a FOR-TO-NEXT loop the program for which appears below with H representing the house numbers.

```
10  FOR H = 1 to 10
20  clean lower windows
30  clean upper windows
40  collect payment
50  NEXT H
```

If the window cleaner had only to clean the houses on one side of the street, say the odd numbers (1,3,5,7,9) then he would need another instruction to inform him of this. The instructions would now appear as:-

1. FOR each house number from 1 to 9, moving to every second house number
2. clean lower windows
3. clean upper windows
4. collect payment
5. move to NEXT house number

The section "moving to every second house number" can be replaced by the word STEP followed by the number 2 (i.e. STEP 2). This indicates the amount of increment required between the house number and the next one to be cleaned (i.e. 1+2=3, 3+2=5 and so on). Thus the FOR-NEXT loop would now appear as:-

```
10  FOR H = 1 to 9 STEP 2
20  clean lower windows
30  clean upper windows
40  collect payment
50  NEXT H
```

Therefore the processing is unchanged but the inclusion of STEP 2 indicates that the house numbers will be incremented by two as opposed to one each time. STEP 3 would give an increment of 3 (i.e. house numbers 1,4,7). STEP 4 would give an increment of 4 (i.e. house numbers 1,5,9) and so on.

Examples

1.

```
10  FOR I = 1 TO 10 STEP 3
20  PRINT I
30  NEXT I
```

This would take each value of I from 1 to 10 in increments of 3 and print it on the output device as follows:

```
1
4
7
10
```

2.

```
10  FOR I = 1 TO 20
20  PRINT "*";
30  NEXT I
40  END
```

This would cause a print on the output device of a row of 20 *

More information and details relating to the FOR-TO-NEXT statement will be found in Chapter 10.

SUMMARY

a) SEQUENCE

b) BRANCH - Introduction to:
  i) IF-THEN (conditional)
  ii) GOTO (unconditional)
  iii) SUBROUTINES
    a) GOSUB
    b) RETURN

c) LOOPS - Introduction to:
  i) IF/THEN/GOTO (Loop)
  ii) FOR-TO-NEXT (Loop)

CHAPTER 10

BASIC RESERVED WORDS

ABS (Absolute value)

Syntax: ABS(N)

Where N is any numeric expression.

Purpose: The ABS function returns the absolute value of the numeric expression, i.e. ignores signs, always returning positive values.

Example:

```
X=ABS(-3.14159)
```

This will return a value of 3.14159 in X.

Related Keyword : SGN

ADC (Pseudo Analogue to Digital Converter)

Syntax: ADC(J)

Where J is a "channel number" given as 0,1,2, or 3.

Purpose: This is a function which reads the value at the Joystick port for the channel specified in J.

Channel	Function	Joystick Port
0	left/right	2
1	up/down	2
2	left/right	1
3	up/down	1

For each channel a value of 0 represents left, or down, as appropriate; 127 the centre position; and 255 right, or up, as appropriate.

The function allows incorporation of joystick control into a program.

Examples:

```
A = ADC(0) PRINT ADC(0)
PRINT A
```

This will read the current value at the analogue port for channel 0, and display its value. With no joystick connected, the result should be 127, corresponding to the centre position.

Related Keywords: BTN,JOY

AND

AND

Syntax: statement AND **statement**

Purpose: This is a LOGICAL OPERATOR used in the evaluation/comparison of statements and/or numeric expressions.

Examples:

```
10 IF (x + y)<>3  AND y = 20 THEN 100
```

This will transfer program execution to line 100 if (x + y) is not equal to 3 and at the same time y = 20

```
20 A = 15 AND 7
```

This returns a value A = 7.

```
30 K = INCH AND &DF
```

This returns upper case ASCII whether upper or lower case entered.

Related Keywords: ELSE IF NOT OR THEN XOR

APPEND

APPEND

Syntax: APPEND <ufn>, SV

<ufn> must be a legal "file name". SV is a string variable (but not a string array element) and is called the file descriptor.

Purpose: This command is used to write extra information at the end of a sequential file when to OPEN the file and read to the end would be most inefficient.

The operation is similar to OPEN with the following differences:-

a) no record length is supplied.
b) the internal file pointer moves to the end of the file not the start of the file.

Example:

APPEND "0:SILLY.DAT",FD$

This will perform the following:-

a) opens the file SILLY.DAT on the disc currently in drive 0, and moves the pointer to the end of the file so data may be added.

b) assigns FD$ has been assigned as the file descriptor.

NOTE: If the file specified by <file> does not exist on the disc then a NO FILE ERROR will be given.

Related Keywords : CLOSE CREATE OPEN

ASC (American Standard Code For Information Interchange - ASCII)

Syntax: ASC(<string expression>)

Purpose: This is a Standard String Function which returns the ASCII value (in decimal) of the first character of the string given in the function.

To display the values given by this function use the PRINT command as a prefix.

Example:

```
X=ASC("ABC")
```

This will return a value of 65 in X. X contains the ASCII value of A (i.e. the first character of the string).

65 is the decimal code for A.

Related Keywords : CHR$ STR$

ATN(Arctangent)

Syntax: ATN(N)

Where N is a number or a numeric expression.

Purpose: This is a Standard Function which returns the arctangent of N, in radians, within the range - PI/2 to +PI/2

Example:

```
X=ATN(1)
```

Returns a value of 0.785398 radians in X. (i.e. PI/4 radians or 45°)

Other Transcendental Functions:

```
ASN(X)=ATN(X/SQR(1-X*X))   arcsin(x)
ACS(X)=(PI/2)-ASN(X)       arccos(x)
HCS(X)=(EXP(X)+EXP(-x))/2  cosh(x)
HSN(X)=(EXP(X)-EXP(-X))/2  sinh(x)
HTN(X)=1-2/(1+EXP(X*2))    tanh(x)
```

Related Keywords : COS SIN TAN

AUTO

AUTO (Automatic Line Numbering)

Syntax: AUTO L1,L2

L1 is the line number from which automatic numbering is to commence.

L2 is an increment value for the numbers to be used in the automatic numbering sequence.

Both L1 and L2 will default to a value of 10 if they are not stated.

Purpose: This is a System Command which gives automatic line numbering while entering a program.

Examples:

AUTO 100,5	Starts at line 100 and continues 105,110,115,etc.
AUTO 100	Starts at line 100 and continues 110,120,130,etc.
AUTO,20	Starts at line 10 and continues 30,50,70,etc.
AUTO	Starts at line 10 and continues 20,30,40,50,etc.

When AUTO has been invoked, the next line number automatically appears for the user to continue after the ENTER key has been pressed.

Each number is displayed just as if it had been typed from the keyboard. The automatic line numbering may be abandoned keying "SHIFT-BREAK".

The "editing mode" is not affected by the use of AUTO.

Related Keyword: RENUM

BAUD (Baud Rate)

Syntax:BAUD R,T.

Purpose: This is a command to set the receive and transmit rates of the serial port R represents the receive channel, and T the transmit channel. R and T can have values in the range 0 to 8, and represent the receive and transmit rates as defined in the table below.

Value	Baud Rate	Value	Baud Rate
0	75	5	1200
1	110	6	2400
2	150	7	4800
3	300	8	9600
4	600		

Any mix of receive and transmit rates is possible with the exception that a receive rate of 75 baud can only be set if the transmit rate is also 75 baud.

The default value is 9600 baud for both transmit and receive.

Example:

```
BAUD 5,0
```

Sets up the serial port to 1200 baud receive, 75 baud transmit. (Prestel speeds).

Related Keyword: MODE

BCOL (Backdrop colour)

Syntax: BCOL N

N can be a value from 0 to 15, each number representing a particular colour as defined by the **PCOL** command. The default pallette settings are given in appendix D.

Purpose: This is a Display Command which sets the backdrop colour according to the value of N.

Example:

BCOL 6

This would set the backdrop colour on the screen to Dark Red.

When BASIC is loaded the backdrop colour is set to 4 (Dark Blue)

The Backdrop colour is maintained with SCREEN mode changes.

The Backdrop command is operative in all SCREEN modes.

Related Keywords : GCOL, PCOL, TCOL

BEEP

Syntax: BEEP J

Purpose: This is a Sound Command which causes an 880Hz tone to be sounded for a length of time indicated by the value of J which must be in the range 1 to 255.

J-VALUE	TIME
1	100ms
2	200ms
3	300ms
4	400ms
5	500ms
.	.
.	.
.	.
255	25,500ms

Example:

BEEP 20

This will cause the tone to be sounded for a length of time equivalent to 2000ms (2 seconds).

It is useful as an audible prompt that some process has been accomplished, or a mistake has occurred, usually in programs requiring input from the keyboard.

Related Keywords : MUSIC PSG

BIN$ (Binary String)

Syntax: BIN$(I,J)

Purpose: This is a Machine Code related command which returns the Binary number which corresponds to the decimal number given by I.

J indicates the number of binary digits to be returned in the result and must evaluate to an integer which is less than, or equal to, 16. If J is omitted then 16 binary digits are returned (the number being "padded" with leading zeros if necessary).

If the value of J is too small for the binary number to be returned, then only the J least significant digits will be returned.

Example:

```
X$=BIN$(86)
```

This will return a result of 0000000001010110 in X$

```
X$=BIN$(42,8)
```

This will return a result of 00101010 in X$

Related Keyword : HEX$

BTN (Press Button)

Syntax: BTN(J)

Where J is given as 0 or 1.

Purpose: This function returns a value of 0 or 1 for a press button connected to the Analogue 1 and Analogue 2 ports.
For the Analogue 1 port J=0
For the Analogue 2 port J=1

When the button is pressed a value of 0 is returned.
When the button is not pressed a value of 1 is returned.

A common application of this function relates to the firing button incorporated with joystick controls, but it can also be used to signal the program of some external stimulus.

Examples:

```
1) B = BTN(0)

   PRINT B
```

This will return a value (either 0 or 1) in B according to the status of the Analogue 1 port. The value can be output using the PRINT statement.

```
2) 30 IF BTN(0) = 0 THEN CLS
```

Related Keyword: ADC

CALL

Syntax: CALL I

Purpose: This is a Machine Code related command which calls a machine code subroutine which starts from the address given by I. The related machine code subroutine must be terminated with a &C9 (return) code. This will automatically return control to BASIC.

Example: CALL 3840

This causes execution of machine code from location &0F00 (i.e. HEX equivalent of 3840 decimal).

NOTE: The pointer to the current position in the program text will be available at the top of stack, if required.

Syntax: CALL(E)

Purpose: E is passed to the floating point accumulator within the BASIC interpreter. Machine code is then executed from the address set by means of the PTR 9,I command. This defines the location for the machine code routine. Again a &C9 (return) code returns control to BASIC, and the contents of the floating point accumulator form the argument of the CALL function.

Example: A = CALL(B)

The value of B is loaded into the floating point accumulator. On return from the machine code routine, the contents of the floating point accumulator are passed into variable A.

Related Keywords: PTR, CLEAR

CHAIN

Syntax: CHAIN L

Where L, if given, is a line number.

Purpose: In this case CHAIN is similar to the RUN command except that all variables are preserved and can be passed from one program to another.

Examples:

```
  CHAIN    - Executes the program currently in memory.

  CHAIN 50 - Begins execution at line number 50.
```

Syntax: CHAIN <file>

Where file must be a legal "file name" as described in Chapter 13, File Handling in BASIC.

Purpose: In this case the file specified by file is loaded from the disc and executed. This is similar to RUN file except that all variables are preserved.

Example:

```
  CHAIN "PROG" - This loads and executes a program called "PROG"
                 preserving any variables.
```

NOTE: Programs can be combined using a combination of CHAIN, HOLD, and MGE commands. this could be useful in large applications which may be divided into smaller programs, all using the same variables.

Related Keyword : RUN

CHAR

CHAR (Character Set)

Syntax: CHAR N

Purpose: This is a command to change the character set. It has the effect of changing the characters generated by certain codes. N can have any value from 0 to 3.

N	Character Set
0	Einstein (ISO646)
1	ASCII
2	German
3	Spanish

The table, below, shows the keys affected

Key	N=0	N=1	N=2	N=3
÷ ↑	÷ ↑	~ ↑	ß ^	~ ↑
½ ‖	½ ‖	\\ ¦	Ö ö	+ ;
¼ ←	¼ ←	{ [	Ä ä	{ ¡
¾ →	¾ →	}]	Ü ü	} ¿
+ ;	+ ;	+ ;	+ ;	Ñ ñ
£ -	£ -	£ -	\\ -	\\ -

The default character set is determined by the setting of DIP switches inside Einstein 256, and is normally Einstein (ISO646). Should you wish to change the power-up default character set, refer to the hardware section, Appendix P, under Technical Specification: TSC256.For complete code tables refer to appendix B.

Related Keywords:

CHR$

CHR$ (Character String)

Syntax: CHR$(J)

Purpose: This is a String Function which returns the single character string whose ASCII value is given by J.

Example:

X$=CHR$(65)

The decimal ASCII value 65 is that of the character A. Therefore the string A will be stored in X$.

Related Keywords : ASC STR$

CLEAR

Syntax: CLEAR I1,I2

I1 when specified, sets up the topmost location of memory available to BASIC. I2, when specified, allocates the size of the "stack".

Purpose: This Command clears all variables, arrays and strings from the system. The top of memory would be set in order to leave space for object (.OBJ) files (i.e. machine code routines/data). Normally, no space is reserved, and the stack size is left unchanged if I2 is omitted.

If I1 is set above the top of RAM, or set too low, or too large a stack size (I2) is set, then a MEM FULL ERROR will occur.

The stack is normally 256 bytes and cannot be smaller. Normally it would not be necessary to increase this size unless large numbers of nested FOR loops, subroutines, and expressions are used. (If a STACK FULL ERROR is encountered it is usually because subroutines are being ENTERED but not RETURNED from!).

Examples:

CLEAR,500 - sets 500 bytes of stack space.

CLEAR&7FFF - sets the top of RAM for BASIC programs and variables to $7FFF_H$. Thus machine code programs can be placed in the area from 8000_H up.

CLEAR &AFFF,300 - sets 300 bytes of stack space, and the top location to &AFFF.

Related Keyword : PTR

CLOSE

CLOSE

Syntax: CLOSE SV1,SV2,..,SVn

SV1,SV2, etc. must be string variable names (but not string array elements) and are the file descriptor.

Purpose: This is a File-Handling command which performs the following:-

a) writes the remaining contents of the appropriate buffers to their files.

b) stores directory information.

c) closes the files given by the file descriptors in SV1 to SVn, having previously been opened using OPEN, CREATE, or APPEND.

A buffer will only be written out if the last operation performed on it was a write. The file descriptors are then set to null strings which makes the space available for use by variables or other files.

If any of the file descriptors specified is not active (or is an ordinary string) then a FILE ERROR will be given. (File descriptors are internally marked to BASIC can distinguish them from normal strings).

If no file descriptors are specified then all files currently open will be closed. (No error is given if there are no files open).

NOTE: In addition to the above processing, CLOSE induces an automatic PRINT#O:INPUT#O. This will cause all output and input to go through the screen and keyboard and the CLOSE command can be used at any time when these two statements are required (it is shorter.)

Related Keywords: APPEND CREATE OPEN

CLS

CLS (Clear Screen)

Syntax: CLS N

Where N has a value of 32 or 40

Purpose: To clear the display screen or send a form feed character to any other selected output device.

If current output is to the screen then:-

If N = 40 (i.e.CLS40) this will clear the screen and set up the 40 column Display.

If N = 32 (i.e.CLS32) this will clear the screen and set up the 32 column Display.

If N is omitted, the screen will clear, leaving the current Display unchanged.

For any other output device:-

A "form feed" character, (OC), is sent to the selected device. See the section dealing with external devices.

NOTE: The N parameter is retained for compatibility with EBASIC, so as to allow programs written in EBASIC to run. When writing new programs the SCREEN command should be used to change display formats and modes.

Related Keyword: SCREEN

CONT (Continue)

Syntax: CONT

Purpose: Causes an interrupted program to resume without clearing the variables.

May be used after a program has been terminated with a STOP command. During the "stopped" period, the user may look at or alter variables without causing any problems, but any attempt to alter the program itself will result in a CONT ERROR.

CONT may be used to re-start a program which has been halted by SHIFT-BREAK. This is quite a useful aid when debugging a program.

Related Keywords: END RUN STOP

COS (Cosine)

Syntax: COS(N)

Where N is an angle, or a numerical expression returning an angle, given in radians.

Purpose: This is a Standard Function which returns the COSINE value of N.

Examples:

i) X = COS(1.0472)

This gives a value of 0.499998 in X. (1.0472 radians is 60°)

ii) PRINT COS(0.5236)

The value 0.866025 will appear on the screen. (0.5236 radians is 30°)

iii) This example shows an entry made in degrees using the RAD function to convert the angle within the COS function.

PRINT COS(RAD(30))

The value 0.866025 will appear on the screen.

Related Keywords: ATN DEG RAD SIN TAN

CREATE

CREATE

Syntax: CREATE <ufn> ,SV,I

SV is a string variable name (but not a string array element) and is the file descriptor.

I is the random record size (length) and is given as a value in the range 0 to 65535, indicating the number of characters involved.

Purpose: This is a File-Handling Command which creates and opens a new serial data file as follows:-

a) deletes any existing file with the same name as given in the command.

b) creates and opens a new empty serial data file having the name given in the command, and identified by the string variable SV, to be structured into data records of length I bytes. I is only specified for "random access", if "sequential access" is to be applied then I is omitted (in fact a random record length of 0 indicates that sequential access is to be performed).

Example:

```
CREATE "0:SILLY.DAT",FD$,15
```

This will perform the following:-

a) creates and opens the file SILLY.DAT on the disc currently in drive 0 (if any file of the same name already exists on the disc it will be deleted prior to the new empty file being created).

b) assigns FD$ as the file descriptor.

c) sets up for "random access" using a 15 character length record size.

Related Keywords: APPEND CLOSE OPEN

DATA

Syntax: DATA data 1 , data 2 , ... , data n

The items of data (data 1, data 2, etc) may be any of the following types.
a) numeric
b) strings in quotes.
c) strings without quotes, providing there are no leading spaces or separators.

Purpose: This statement holds items of data required within a program. It is used in conjunction with the READ statement.

Any number of DATA statements can be used within a program, each containing as many or as few items as are convenient

DATA statements may appear at any position in a program but will be read as though they were all in one block.

They are ignored when encountered during the running of a program in the same manner as REM statements.

The SEPARATOR between items of data is normally a comma (,) but this may be modified by use of the SEP command. (This will also affect INPUT and READ statements within the same program).

Example:

```
DATA 6, "NO", YES
```

Related Keywords: READ RESTORE

DEEK Double Peek

Syntax: DEEK(I)

I must evaluate to a number in the range 0 to 65535.

Purpose: This is a Machine Code related command which returns an integer in the range (0 to 65536) which represents the contents of the memory locations given by I and I+1. In effect it is a two byte memory PEEK. (The byte I+1 is taken to be the most significant byte).

Example:

Suppose location &4000 contains &C4 (least significant), and &4001 contains &06 (most significant).

```
PRINT DEEK (&4000)
```

This would display an integer of 1732 (i.e. &06C4) on the screen.
contents of location &4001
contents of location &4000

Related Keywords: DOKE PEEK POKE

DEF FN

DEF FN (Define Function)

Syntax: DEF FN V1(V2) = E

V1 is the name given to the function and can be any legal variable name (numeric or string) and is usually known as the identifier. The value of V1 must be of the same "type" as the result of expression E.

V2 is a "dummy variable" which corresponds to the variable used within the function and may be numeric or string.

E is a string or numeric expression which contains the variable V2.

Purpose: DEF FN defines a function in a program which is to be used in that program. The expression E contains the function definitions and must contain the dummy variable V2. The variable V1 identifies the functions.

On encountering FN V1(N) in the program the value N is passed to the defined function as the dummy variable. The resultant value of the expression E is returned into FN V1(N).

Having defined a particular function with DEF FN it may be called up at any future point in a program as FN V1(N) see related word FN.

Example:

```
  10 DEF FN B(X) = X+3
 100 Y = FN B(6)
```

The function B(X) is defined in line 10.
In line 100, 6 is passed to the defined expression X+3 as the dummy variable X and hence Y takes the value 9.

If a function call is made before the appropriate DEF FN statement has been made a FN DEFN ERROR will occur.

Related Keyword: FN

DEG (Degrees)

Syntax: DEG(N)

Where N is given in radians.

Purpose: This is a Standard Function which converts the number expressed in radians given by N, to degrees.

Example:

```
DEG(0.523604) - returns a value of 30 degrees.
```

Example:

```
PRINT DEG(0.523604)
```

This will cause the value 30 to appear on the screen.

Related Keywords: ATN COS RAD SIN TAN

DEL (Delete)

Syntax: DEL L1,L2

Where L1 and L2 are given as line numbers of a program.

L1 will default to 0 if not specified.

If L1 is larger than L2, or if L1 is larger than the largest line number of the program, a RANGE ERROR will occur.

Purpose: This is a System Command which deletes all lines from a program in the range of L1 to L2.

Examples:

```
DEL 100,199
```

This will delete all program lines with numbers from 100 to 199 inclusive.

```
DEL,155
```

This will delete all lines up to 155 inclusive (value of L1 omitted therefore defaulting to 0).

Related Keyword: LIST RENUM

DIM

DIM (Dimension)

Syntax: DIM <array name> (I1,I2,I3..,In)

The array name can be either a numeric or string variable.

I1,I2,I3 are numeric expressions, known as subscripts, in the range 0-65535 and represent the number of elements in an array.

If an array is not dimensioned it is assumed to have a maximum value of 10 for each subscript (dimension) which is referenced. Therefore if subscripts are less than 10 the DIM statement may be omitted.

An array must only be dimensioned once in a program. If dimensioned more than once a DIMENSION ERROR will occur.

Purpose: This is used to reserve storage space for numeric or string arrays.

An array with only one subscript is known as one dimensional.

A(I)

An array with two subscripts is known as two dimensional.

A(I1,I2)

An array with three subscripts is known as three dimensional.

A(I1,I2,I3)

Each subscript represents the maximum number of elements of each dimension in the array. EBASIC will support multi dimensional arrays, the limit being determined by the amount of memory available at any one time.

Several arrays may be dimensioned in one DIM statement using a comma as a separator.

DIM A(I1,I2), B(I), C(I1,I2,I3) etc.

Examples:

```
DIM A(50)
```

Defines array A as having one dimension with storage for 51 elements (0 to 50).

DOKE Double Poke

Syntax: DOKE I,I1,I2,...,In

Purpose: This is a Machine Code related command which places the values of the expressions I1,I2 etc. into memory, starting at the location given by I. Each expression will be placed into TWO bytes of RAM, the first byte being the least significant byte, the second being the most significant byte.

Example:

```
DOKE 16384,5764
     └──────address of least significant byte.
```

5764 is equal to &1684 (HEX) therefore &84 (least significant byte) is placed into location &4000 (HEX value of 16384), the first byte of memory, and &16 (most significant byte) is placed into location &4001 (second byte of memory).

Example:

```
DOKE &5100,&77,&1234
```

This places &77 into location &5100,&00 into &5101 (i.e. the 00 preceding the 77 in HEX format), &34 into &5102, &12 into &5103.

Related Keywords: DEEK PEEK POKE

DOS (Disc Operating System)

Syntax: DOS

Purpose: This is a System Command and is used to transfer control to the "Disc Operating System".

Usually used in "Direct Mode".

Related Keyword: MOS

DRIVE (Disc Drive)

Syntax: DRIVE N

N is specified as a number (0 or 1) depending on the number of drive units available within individual systems.

If the drive specified in J is not available on the system a DRIVE SELECT ERROR will be given.

Purpose: This command sets up the default disc drive as specified by drive name for any subsequent access to a disc.

Example:

```
DRIVE 1
```

This selects drive 1 as the default drive.

Related Keywords: DIR ERA LOAD REN SAVE

ELLIPSE

Syntax: ELLIPSE x,y,R,T,z,a,b

Purpose: This command draws an ellipse or a circle depending on the values of the parameters given.

x,y are the co-ordinates of the centre for the ellipse and can have values in the range -32768 to +32767.

R is the value of half the horizontal axis of the required ellipse.

T is a qualifier and is given by the ratio of the two axes as follows:-

$$T = \frac{\text{VERTICAL AXIS}}{\text{HORIZONTAL AXIS}}$$

If T is omitted it defaults to 4/3 and a circle of radius R is drawn (owing to the aspect ratio of the VDU screen being 4:3).

NOTE: The aspect ratio of 4/3 is true only in screen modes 0 and 1, and in 625 lines. The default value is retained for compatibility with Einstein programs. For other screen modes, and line standards, the value for T is shown in the table below.

	Values for T to give true circles.	
Screen mode	T (525 lines)	T (625 lines)
0	1.167	4/3 (default)
1	1.167	4/4 (default)
2*	not applicable	not applicable
3	0.58	0.67
4	0.58	0.67
5	0.58	0.67
6	0.58	0.67

* Ellipse has no meaning in screen mode 2. (80 column/text only).

The value of **z** is a number in the range 0 to 5 which indicates the type of line in accordance with the table given below. (If omitted z defaults to 0).

0 - Continuous line.
1 - Continuous unplot (i.e. line drawn in background colour).
2 - Dotted line, 2 dots on 2 dots off.
3 - Dashed line, 4 dots on, 4 dots off.
4 - Dotted-dashed line, 10 dots on, 2 dots off, 2 dots on, 2 dots off
5 - Dashed-dotted line, 2 dots on, 2 dots off, 2 dots on, 10 dots off.

a and b are start and end angles which indicate where the drawing of the ellipse should begin and end. These are optional. The start and end angles are specified in radians and are numbered in an anticlockwise direction from 0 to the right hand horizontal axis up to 2 pi radians for one complete revolution, the values may be specified as numbers or an expression in terms of pi.

Examples:

```
ELLIPSE 100,100,60,1,0,1,0.25
```

If a or b are given as negative values the start and end points will be joined to the 'x,y' point (centre) with a line.

```
ELLIPSE 100,100,60,1,0,-1,0.25
```

Example:

```
ELLIPSE 100,100,50
```

A circle of radius 50 is drawn with its centre at co-ordinates 100,100 (T defaults to 4/3 and z to a continuous line, a and b being omitted to produce a full circle).

Related Keywords: DRAW ORIGIN PLOT POLY UNPLOT

ELSE

Syntax: IF <condition> THEN <statement> ELSE <statement>

Purpose: This command is used in conjunction with the IF - THEN statement to provide an alternative course of action.

Example:

```
IF X = 10 THEN 100:ELSE 50
```

If x equals 10 then execution is transferred to line 100. If x does not equal 10 then the program branches to line 50.

Refer to the IF statement for further information on the use of ELSE.

Related Keywords: GOTO IF THEN

END End of Program

Syntax: END

Purpose: This command is used in deferred mode i.e. as a line of a program.

This command terminates the execution of a program.

It is not strictly necessary when the end of a program coincides with the highest line number and in such cases can be omitted.

Related Keywords: CONT STOP

EOF

EOF (End of File)

Syntax: <statement> EOF <statement>

Purpose: This command is used in relation to File Handling and invokes a specific action following detection of the end of a file during processing.

EOF is used within the following statements.

ON EOF GOTO "Line No"
ON EOF GOSUB "Line No"

Either of these two statements could be used in a program involving file handling. If an "end-of-file" is detected then a GOTO/GOSUB is made to a particular routine which would carry out a predetermined course of action. The last statement of this routine should direct execution back into the main program. Execution would then continue from the statement immediately following the one which detected the "end of file".

When either of the two statements are used, an internal flag is set in order to activate the above procedure.

OFF EOF is used to turn off the ON EOF mode. Any subsequent end-of-file encountered will then cause an END OF TEXT ERROR to be displayed. However, when a program "ends" in the normal way, the ON EOF is automatically turned off.

Example:

```
10 - - - - - - -
20 - - - - - - -
30 - - - - - - -
40 ON EOF GOTO 120
50 - - - - - - -
etc.
```

In this example any "end-of-file" detected after line 40 would cause a branch to a routine starting at line 120. This routine would then carry out a predetermined course of action in respect of the "end-of-file" before returning execution to the main program.

Related Keywords: GOTO GOSUB OFF ON ERR

ERA (Erase)

Syntax: ERA "<ufn>"

Purpose: This is a Disc Command which will erase the file, given by filename from a disc in the specified drive (See Chapter 13 for file handling conventions).

If <filename> does not exist then a NO FILE error will be given.

If <filename> is a "locked" file then a FILE LOCKED error will be given.

If the write-protect of the disc is in operation a DISC LOCKED error will be given.

The default drive may be changed or re-selected by use of the DRIVE command.

NOTE: Wildcards (? and *) are not allowed in BASIC.

Related Keywords: DIR DRIVE

ERL (Error line number)

Syntax: ERL

Purpose: This function is used in error handling routines and returns the line number at which the last error occurred (see Chapter 11 for further information on error handling). 0 is returned if no error has occurred.

Example:

```
10 PRTNT "DOS"
```

The spelling mistake in line 10 would generate a SYNTAX ERROR. ERL would now contain 10 and PRINT ERL would then display 10

Related Keywords: ERR ERR$OFFON

ERR (Error)

Syntax: ERR

Purpose: This command returns the value of the last error generated.

Example:

```
  PRINT ERR
```

This will display the value of the last error generated.

ERR can also be used in conjunction with the ON command as follows:-

```
ON ERR GOTO L
ON ERR GOSUB L
```

Where L is a line number.

For further information relating to ON ERR refer to Chapter 11, Error Handling Within BASIC.

Related Keywords: ERL ERR$ GOTO GOSUB OFF ON

ERR$ (Error String)

Syntax: ERR$

Purposes: This function is used in error handling routines and returns the error string message, without the word "ERROR", corresponding to the last error which occurred. Refer to Chapter 11 for further information on error handling.

Example:

```
  10 PRJNT "ONE"
```

A SYNTAX ERROR is generated in line 10 due to the incorrect spelling of PRINT.

PRINT ERR$ will now display 'syntax'

Related Keywords: ERL ERR ON ERR

EVAL (Evaluate)

Syntax: EVAL(<string expression>)

Purpose: This is a Standard Function where the string expression is calculated as if it were a numeric expression and returned as a numeric value.

The syntax of string expressions must be correct as a numeric expression otherwise a SYNTAX ERROR will occur.

Example:

```
A = EVAL("10*3+4")
```

10*3+4 is a string expression but is treated as if it were a numeric expression and the value 34 is returned into A.

Related Keyword: VAL

EXP (Exponent)

Syntax: EXP(N)

Purpose: This is a Standard Function which raises the exponential, e, to the power given by N.

If N exceeds 87,34 an OVFL ERROR (overflow error) will occur.

e has the value of 2.71828.

Example:

```
X=EXP(12)
```

This will raise e to the power 12 (i.e. e^{12}) and store the result 162755 in X

Related Keywords: ATN LN LOG

FILL

Syntax: FILL x,y,J

x and y are the co-ordinates of a required point and can be in the range -32767 to +32767.

J can be a value in the range 0 to 15, each number representing a colour as defined by the palette command. The default palette is shown in appendix C.

Purpose: This is a Graphics Command which will fill in colour J an area of the screen which has its perimeter drawn in a foreground colour and which encloses the point x,y.

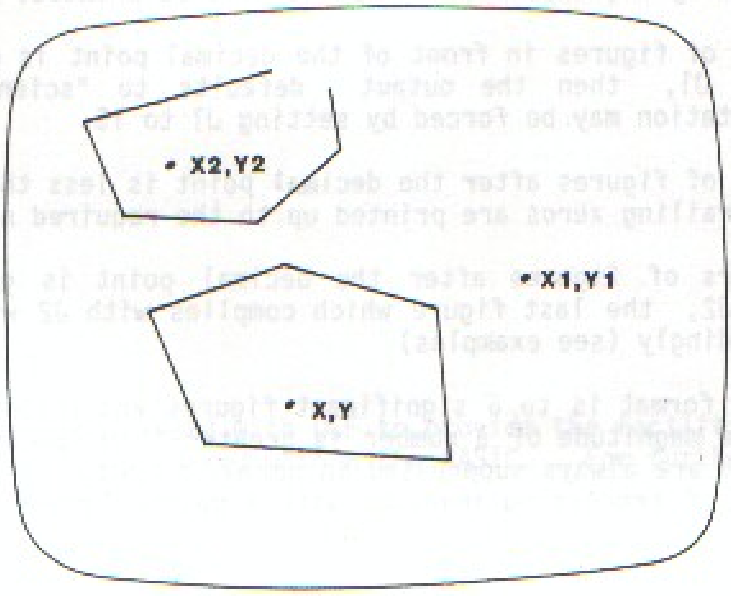
This command will fill in foreground if point x,y is background or in background if the point is foreground. If J is omitted the colour of the fill will be the current graphics colour for the fill type (i.e. background or foreground) unless J is declared.

Example:

```
10 REM FILL
20 BCOL 0
30 SCREEN1:GCOL 7,0:ORIGIN 128,96
40 ELLIPSE 0,0,60:FILL 0,0
50 GCOL,8
60 POLY 6,0,0,45,,1:FILL 0.0
```

This example draws a cyan-coloured disc, and then draws a red hexagon inside it.

Example:



In the example illustrated:

If point X,Y is specified then the command will fill the given polygon to its boundary.

If point X1,Y1 is specified then the command will fill the screen outside the polygon.

If X2,Y2 is specified then the command will fill the associated polygon and then spill out, because the shape is not fully enclosed, and fill the remainder of the screen.

NOTE: A 'STACK FULL' error may occur when FILLing around text. This is caused by overflow of the 'FILL' stack.

Related Keywords: DRAW ELLIPSE GCOL ORIGIN PLOT POLY TCOL UNPLOT

FMT (Numeric Output Format)

Syntax: FMT J1,J2

Purpose: This command is used to format numeric output for PRINT statements and STR$ functions.

J1 gives the number of figures to be printed in front of the decimal point, and J2 the number of figures to be printed after the decimal point. The sum of J1 and J2 must not be greater than 8 (maximum precision of the system is only 7 significant figures) otherwise a QTY ERROR occurs.

If the actual number of figures in front of the decimal point is less than that specified by J1, then leading spaces will be printed.

If the number of figures in front of the decimal point is greater than that specified by J1, then the output defaults to "scientific notation". Scientific notation may be forced by setting J1 to 15.

If the number of figures after the decimal point is less than that specified by J2, then trailing zeros are printed up to the required number of figures.

If the numbers of figures after the decimal point is greater than that specified by J2, the last figure which complies with J2 will be rounded up or down accordingly (see examples)

Normal output format is to 6 significant figures and scientific notation is invoked if the magnitude of a number is greater than 1E6 or less than 1E-2. Trailing zeros are always suppressed in normal output.
On conclusion of processing involved with a particular FMT command, normal output can be restored by use of FMTO,0.

Examples:

```
FMT4,2:PRINT5692.347 - displays as 5692.35

FMT4,2:PRINT347.6932 - displays as 347.69

FMT3,2:PRINT 5678.346 - defaults to a display of 5.67835E+03

FMT3,4:PRINT543.45 - displays as 543.4500

FMT15,2:PRINT 567.9876 - displays as 5.68E+02
```

If the sign of a number is given, it is not counted with the number of figures but appears in the leading space at the start of the entire number.

Examples:

```
FMT3,3:PRINT-1.73205 - displays as -  1.732

FMT3,3:PRINT-42.763  - displays as - 42.763
```

Related Keywords:

FN (Function)

Syntax: FN V1(V2)=E

Purpose: This command is used with DEF to provide the facility for creating functions not normally contained within the BASIC. For further information see DEF command.

Related Keyword: DEF

FOR

FOR

Syntax: V = N1 TO N2 STEP N3

Purpose: This statement sets up a loop within a program to repeat a sequence of operations. It is used in conjunction with the TO, STEP and NEXT statements.

V is the CONTROL VARIABLE which must be a numeric variable.

The values, or numeric expression string values, for N1,N2, and N3 have the following functions.

N1 is the INITIAL VALUE from which V starts the loop.

N2 is the LIMIT VALUE of V, which, when passed, ends the loop.

N3 is an optional STEP VALUE which is the amount by which V is incremented on each cycle of the loop. If "STEP N3" is omitted then a step value of +1 is assumed.

The FOR statement indicates the beginning of the loop. The NEXT statement indicates the end of the loop. The NEXT statement is followed by the control variable to link it with the relevant FOR statement.

The program loops between these FOR and NEXT statements. On reaching the NEXT statement V is incremented to the next value and program execution jumps back to the FOR statement. The loop is repeated until V has incremented past the value of N2

Any program lines between the FOR and NEXT statements are executed on each cycle of the loop.

Example:

```
10 FOR I = 1 TO 10 STEP 3
20 NEXT I
```

The control variable, I, starts at 1 (initial value) for the first loop and then increments by 3 (step value) for each repetition of the loop until the limit value of 10 is exceeded.

```
10 FOR I = 1 TO 10
20 NEXT I
```

In this case the step value has been omitted and therefore a value of 1 will be assumed. Thus I will start at 1 and increment by 1 until it exceeds 10.

The necessary operation statements of the loop immediately follow the FOR statement.

Example:

```
10 FOR I = 1 TO 10
20 A = I + 3
30 PRINT A
40 NEXT I
```

In this instance the expression A=I+3 is evaluated and displayed for each value of I from 1 to 10 (I increments by 1 after each evaluation).

If the control variable is missing off the NEXT statement, then it is assumed to be linked with the previous FOR statement.

Example:

```
┌►10 FOR I = 1 TO 10
│ 20 PRINT "RED"
└─30 NEXT
```

I is not needed in line 30 to complete this.

Nested Loops:

FOR-TO-NEXT statements can be nested (i.e. one loop contained within another).

In "nested" loops the NEXT statement takes on the following format.

NEXT V1,V2,..,Vn

This is the equivalent of:-

NEXT V1:NEXT V2;..;NEXT Vn

Example:

```
10 FOR I = 0 TO 7
20 FOR J = 5 TO 19 STEP 2
30 K = I+J
40 PRINT K
50 NEXT J,I
```

This example will print out all the values of I+J using values of I from 0 to 7 and values of J from 5 to 19. The processing will be executed in the following manner.

```
FOR FIRST VALUE OF I=0

I +  J =  K
0 +  5 =  5
0 +  7 =  7
0 +  9 =  9
0 + 11 = 11
0 + 13 = 13
0 + 15 = 15
0 + 17 + 17
0 + 19 = 19

FOR SECOND VALUE OF I=1

I +  J = K
1 +  5 =  6
1 +  7 =  8
1 +  9 = 10
1 + 11 = 12
1 + 13 = 14
1 + 15 = 16
1 + 17 = 18
1 + 19 = 20
```

This process is repeated for each value of I until all the required results have been printed.

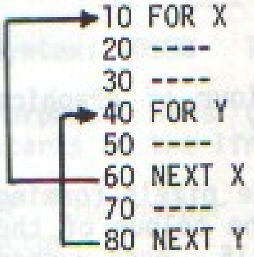
Example:

The following example illustrates how documentation often presents the listing in a format which indicates the nested loops.

```
10 FOR Z = 1 TO 10
20    FOR X = 0 TO 11
30       FOR Y = 1 TO X+13
40          PRINT CHR$(170)+MUL$(CHR$(203),16)
                 +CHR$(170)
50       NEXT Y
60       PRINT MUL$(CHR$(32),18)
70       FOR Y = X+15 TO 40
80          PRINT CHR$(170)+MUL$(CHR$(203),16)
                   +CHR$(170)
90       NEXT Y
100   NEXT X
110 NEXT Z
120 END
```

Crossing

Although loops can be nested they must not be allowed to cross over each other.



```
10 FOR X
20 ----
30 ----
40 FOR Y
50 ----
60 NEXT X
70 ----
80 NEXT Y
```

The format shown above would not work and is an example of bad logic in setting up the loops.

NOTE: If the value of the control variable V in the NEXT statement does not correspond to an active FOR loop, then a NEXT ERROR will be given.
If the control variable V is omitted from the NEXT statement, then the last FOR statement is assumed to be the one required.

There is no limit to the amount of nesting allowed with loops other than the capacity of the memory to deal with the necessary program operation.

Related Keywords: TO STEP NEXT

GCOL

GCOL (Graphics Colour)

Syntax: GCOL J1,J2

Purpose: This is a Display Command which selects the colour of graphics displayed on the screen according to the values of J1 and J2.

J1 represents the foreground colour (i.e. the colour of the pixels forming the character, or shape) and J2 the background colour (the colour of the surrounding pixels). J1 and J2 must lie in the range 0 to 15, each number corresponding to a particular colour as defined by the palette command. (The default palette is given in appendix D). If no previous GCOL command has been given, then J1 will default to 15, and J2 to 4. If the palette has not been re-defined, these will correspond to white and dark blue respectively.

The command operates in all screen modes, ecept screen 2 (which is purely text mode). In screen modes 3 to 6, background colour (J2) has no meaning, and produces no result, as pixels are plotted in foreground only. However, J2, can be specified, for convenience, if required.

Screen Mode	Foreground	Background	Comments
0			Graphics2 32col. Einstein compatible
1			Graphics2 40 col. Einstein compatible
2	No graphics in this mode - text (80 column) mode 2 only		
3		X X	Graphics 6, 32 col. text
4		X X	Graphics 6, 40 col. text
5			Graphics 6, 64 col. text
6			Graphics 6, 80 col. text

Example:

GCOL 10,6

Any graphics, in screen modes 0 or 1, with the default palette will appear in Dark Yellow foreground on a Dark Red background. In screen modes 3 to 6, only the Dark Yellow foreground will appear, while screen mode 2 will produce no result (text mode only)!

NOTE:

This command only directly affects the graphics pixels as they are plotted on the screen and/does not affect the overall backdrop colour of the screen.

Related keywords: BCOL SCREEN TCOL

GOSUB

GOSUB (Go to Subroutine)

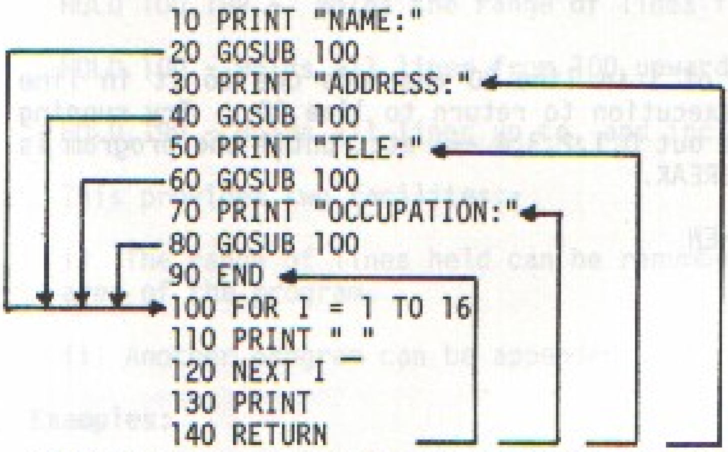
Syntax: GOSUB line number

Purpose: This transfers execution of the program to a subroutine which starts at the line number specified

If line number is not specified, or does not exist, a BRANCH ERROR will occur.

Execution continues from the specified line number onwards until a RETURN statement is encountered, whereupon execution is returned to the line immediately following the original GOSUB statement.

Example:



```
10 PRINT "NAME:"
20 GOSUB 100
30 PRINT "ADDRESS:"
40 GOSUB 100
50 PRINT "TELE:"
60 GOSUB 100
70 PRINT "OCCUPATION:"
80 GOSUB 100
90 END
100 FOR I = 1 TO 16
110 PRINT " "
120 NEXT I
130 PRINT
140 RETURN
```

This program prints out the following format:-

```
NAME:
----------------
ADDRESS:
----------------
TELE:
----------------
OCCUPATION:
----------------
```

Lines 20, 40, 60, and 80 cause a branch to the subroutine contained within lines 100 to 140. This subroutine carries out the repetitive process of printing the broken lines between each of the titles. Line 140 returns program execution each time to the lines immediately following the last executed GOSUB.

Related keywords: GOTO POP RETURN

GOTO

Syntax: GOTO line number

Purpose: This transfers program execution directly to a specified line. (i.e. creates a BRANCH).

If the line number is not specified or, does not exist, then a BRANCH ERROR will occur.

Example:

```
 10 I = 0
 20 PRINT I
 30 I = I+1
 40 GOTO 20
 50 END
```

This program prints the value of I in line 20 and adds one to it in line 30. Line 40 causes program execution to return to line 20. Try running the program. It should print out 0,1,2,3,4 --- etc. until the program is terminated by pressing SHIFT-BREAK.

Related Keywords: GOSUB IF ON THEN

HEX$ (Hexadecimal String)

Syntax: HEX$(I,J)

Purpose: This command returns a Hexadecimal string which corresponds to the number given by I (in decimal).

J dictates the number of characters to be returned in the Hexadecimal string and must be in the range 1 to 4. If J is omitted a value of 4 is assumed.

The Hexadecimal number will be "padded" with leading zeros if necessary.

If the value of J is too small for the HEX number to be returned, then only the J least significant digits will be returned.

Examples:

```
 X$=HEX$ (1234)     -  returns the string "04D2"
 X$=HEX$ (100,2)    -  returns the string "64"
 X$=HEX$ (4708,2)   -  also returns the string "64"
```

(only the two least significant characters are displayed as dictated by the value of J being 2).

Related Keyword: BIN$

The MGE (merge) command is used to replace the section in the program and bring the "non-held" part back "in view".

The final result when listed then appears as below with the original held section now in its new position.

```
10 FOR I = 1 TO 5
20 PRINT "*";
30 NEXT I
70 FOR I = 1 TO 5
80 PRINT "#"
90 NEXT I
100 FOR B = 6 TO 10
110 PRINT B
120 NEXT B
```

NOTE: It is important that the section renumbered is not renumbered to lines that exist within the non-held sections. If in the above example the held section was renumbered as RENUM 70,10 then two lines could be created for each line number 70, 80 and 90!

2. To append another program.

Starting with the following initial program.

```
10 FOR I = 1 TO 5
20 PRINT I
30 NEXT I
```

The whole program is held using HOLD,30.

The second program can then be loaded without affecting the held program.

```
10 FOR I = 6 TO 10
20 PRINT I
30 NEXT I
```

The second program is then renumbered using RENUM such that its line numbers are greater than the last line number of the held program.

```
40 FOR I = 6 TO 10
50 PRINT I
60 NEXT I
```

The MGE command is then used to bring the two programs together giving the following result.

```
10 FOR I = 1 TO 5
20 PRINT I
30 NEXT I
40 FOR I = 6 TO 10
50 PRINT I
60 NEXT I
```

CAUTIONARY NOTES:

When renumbering, care must be taken in the selection of the new line numbers. Any duplication of line numbers will result in BOTH lines being listed in the final program.

The line numbers of the "non-held" part of the program are not affected by the renumbering process, but any line number references following GOTO, GOSUB, RUN, THEN, ELSE, RESTORE, contained within the non-held lines will be altered accordingly. (This is obviously a most necessary and useful facility when moving sections about within one program but care should be exercised in respect of this when adding a second program).

HOLD is also used in conjunction with CHAINING and SEMI-CHAINING of programs. Note also that RUN and CHAIN will restore a HELD program.

Related Keywords: CHAIN MGE RENUM

IF

Syntax: IF <condition> THEN <statement> ELSE <statement>

Purpose: This allows the evaluation of conditions so that a choice of execution can be made, depending on whether a condition is TRUE or FALSE (i.e. it is a Conditional Branch instruction). The IF command is used in conjunction with the THEN and ELSE commands.

THEN statement is the statement executed if the condition is TRUE. The ELSE options being ignored in this case.

ELSE statement is the statement of execution if the condition is FALSE. In this case the THEN options are ignored. ELSE is optional and quite often becomes a redundant part of the statement, in which case it is omitted.

Example:

```
IF A=3 THEN PRINT "YES": ELSE PRINT "NO"
```

If A does equal 3 the PRINT "YES" becomes operative (i.e. the TRUE statement).

If A does not equal 3 the PRINT "NO" becomes operative (i.e. the FALSE statement).

ELSE's must not be nested but the following does however work.

```
IF...THEN...ELSE IF...THEN...ELSE...
```

Variations:

IF condition THEN L1 ELSE L2

L1 and L2 are line numbers to which execution will transfer according to TRUE or FALSE conditions. The following format will also produce the same result.

IF condition GOTO L1 ELSE L2

Examples:

```
IF B=7 THEN 70 ELSE 120

IF B=9 GOTO 90 ELSE 50
```

ELSE is optional in both the above cases and if omitted execution will transfer to the next line of the program if the condition is FALSE.

NOTE: The formats may be mixed, replacing either of the lines, L1 and L2, with a statement but the following points must be observed:

i) If L1 is replaced by statements there must be a separator (:) between the last statement and the ELSE.

ii) A line number must always follow the GOTO if that format is used.

Examples:

```
  IF C=5 THEN PRINT "YES":ELSE 90
  IF D=9 GOTO 120 ELSE PRINT "NO"
  IF A=20 THEN 70 ELSE PRINT "WHITE"
```

Related Keywords: ELSE GOSUB GOTO THEN

INCH (Input Character)

Syntax: INCH

Purpose: This is a Standard Function which waits for the next input character and then returns the ASCII value of that character.

Example:

```
  PRINT INCH
```

This will cause the machine to await the next input character and then display its ASCII value on the screen.

Related Keywords: INCH$ INPUT KBD KBD$

INP (Input)

Syntax: INP(J)

Where J represents the address of an INPUT/OUTPUT port.

Purpose: This is a command relating to direct input from any I/O device.

The value returned by this command will be a number in the range 0 to 255
For the addresses of the various ports, refer to appendix L.

Example:

```
A% = INP(&32)
```

This will read the current value at the user INPUT/OUTPUT port and place it in A%.

Related Keywords: OUT WAIT

INPUT

Syntax: INPUT "Prompt" ;V1,V2,...,Vn

The Prompt is optional, but if used must be a string in quotes followed by a semi-colon(;).

V1,V2, etc are numeric or string variable names, strings, or strings within quotes. When more than one variable is used they are normally separated by a comma, but this can be changed by previous use of the SEP command (page 213).

Purpose: This statement is used to input data from the keyboard during the execution of a program.

If the "Prompt" is omitted then BASIC will introduce its own prompt in the form of a question mark (?), unless the system is in "immediate" mode, in which case no question mark will appear. This facilitates the input of a line without any unwanted characters appearing before it. Alternatively, if the prompt is declared as an empty string the "?" will not appear on the screen.

If the number of entries typed in exceeds the number of variables listed in the INPUT statement, then only the first values entered will be used and the message EXTRA IGNORED is displayed.

If the number of entries typed in is less than the number of variables listed in the INPUT statement, then a further prompt (?) will appear.

If a string is entered when numeric data is expected, then the non-numeric data will be ignored; 0 will be assumed if the first character of the string is non-numeric.

Example:

```
  10 INPUT "NAME,STREET,HOUSE NUMBER ";NAME$,STREET$,N
```

The INPUT statement causes execution of a BASIC program to be interrupted and then it waits until the required data is input from the keyboard.

Example:

```
  10 REM PROGRAM TO CALCULATE COST OF ITEMS.
  20 INPUT "COST OF ONE ITEM IN PENCE";I
  30 INPUT "NUMBER OF ITEMS ";N
  40 PRINT "TOTAL COST IS ";I*N
  50 GOTO 20
```

This simple program calculates the cost of a quantity of items, knowing the cost per item.

It illustrates a use of INPUT. The program does not commence execution until variables I (cost of one item) and N (number of items) are typed in from the keyboard; a result then appears on the screen.

Related Keywords: INCH INCH$ INPUT# KBD KBD$ PRINT PRINT#

INPUT# (<u>Input</u> Device Number)

Syntax: INPUT#J

Where J gives a "device number" assigned to a device and in the range 0 to 254.

Purpose: This statement assigns a new input device (eg. serial port) indicated by the value of J.

All input statements, such as INPUT and INCH$, will be received from the new device selected by J until another INPUT# statement is encountered to change the device selection.

When a program ends or aborts, either through an error or as directed from the keyboard, the input device reverts back to the keyboard (i.e. device 0).

If INPUT# is used in direct mode the corresponding input statement must appear in the same line.

The following two input devices are currently assigned within the BASIC language provided.

DEVICE	DEVICE NUMBER
KEYBOARD/VDU	0
PRINTER	1
SERIAL PORT (RS232)	2

Example:

```
INPUT#2
```

All input following this statement in a program will be received from the serial port.

INPUT# can also be used in the same manner as a normal INPUT statement but the "device number" must be followed by a semi-colon (;) so as to distinguish the remainder of the statement. INPUT# may NOT contain prompts.

Example:

```
INPUT#2;X
```

Example:

```
10 PRINT#1
20 INPUT#2;X
30 IF X= 0 THEN END
40 FORI = 1 TO X
50 INPUT A$: PRINT A$
60 NEXT I
70 INPUT#0
80 PRINT#0;"DO YOU WISH TO CONTINUE ";:Y$=INCH$
90 IF Y$="Y" THEN 10 ELSE END
```

This program illustrates the use of INPUT# and also PRINT#. Data must be provided externally to the RS232 serial port in order to run this program successfully.

X represents the number of items of data to be read. First X is read, then data is accepted from input device 2. This data is printed, as it is read, on the printer (output device 1).

After X items have been read both input and output revert back to device 0 (keyboard and VDU screen) for further instructions from the user.

USE WITH FILES

Syntax: **INPUT#SV,R;V** variable list

Purpose: This command takes input from the file given by the file-descriptor SV, starting at the first character of the record number given by R in the file.

The variable list given by V is as for the normal INPUT command. Items are assigned to the variable names given in the same way.

If R is omitted the file will be read from the last point reached, or from the beginning if it has just been opened (same application as in PRINT#). The format of the command is then:

```
INPUT#SV;
```

Following this command all subsequent statements relating to input will attempt to access the file specified by SV (i.e. INPUT, INCH, INCH$, INCH$N) until another INPUT# OR CLOSE is encountered.

As before, the V can be omitted giving the following format(s).

```
INPUT#SV,R
or
INPUT#SV
```

Both these versions will set up the file specified by SV for input, and all subsequent normal INPUT statements will access that particular file.

Related Keywords: INCH INCH$ PRINT PRINT#

INT (Integer)

Syntax: **INT(N)**

Purpose: This is a Function which returns the largest INTEGER value less than or equal to the value given by N.

Example:

```
  X=INT(4.62132)  -  returns a value of 4
  X=INT(-4.62132) -  returns a value of -5
```

(Note the effect on negative numbers).

It is often useful to have a whole number, "rounded up" for some calculations. It is done by adding .5:-

```
  X+INT(Z+.5), which will leave a whole number in X.
```

Example:

```
  PRINT INT(9.72513)
```

The value 9 will appear on the screen.

Related Keyword: MOD

IOM

IOM (Input Output Mode)

Syntax: IOM J1,J2

Where J1 is in the range 0 to 15 and J2 can take the value 0 or 1.

Purpose: This command is used to set up various input and output conditions. J1 declares which condition (i.e. which bit of the 2 IOM bytes) is to be set to the value declared in J2.

12 of the 16 IOM bits have been allocated to particular input and output options. The remaining 4 bits are left available for future expansion.

Until this command is used all the IOM BITS are set to 1. Thus all the conditions are operative as specified for when the BITS are set to ON.

IOM Bit Assignment:

Bit 0 (Edit Mode):

When = 1, this gives SCREEN EDIT mode.

When = 0, this gives LINE EDIT mode.

Example: IOM 0,0

This invokes LINE EDIT mode only.

Bit 1 (Echo Mode):

When = 1, all input characters are echoed to the output device (eg. input from keyboard is echoed to appear on output screen)
When = 0, there is no echo of characters and LINE EDIT mode is automatically selected regardless of the setting of BIT 0 (otherwise the whole system locks up).

Example: IOM 1,0

This would prevent echo of characters eg. from keyboard to screen.

Bit 2 (Switch Mode):

When = 1, the system automatically reverts to LINE EDIT mode when a program runs, and back again to SCREEN EDIT mode when the program ends or is abandoned.

When = 0, the automatic change of EDIT mode is prevented (i.e. the current EDIT mode will be maintained before, during, and after a program is run).

Bit 3 (Break Mode):

When = 1, this allows the use of the SHIFT-BREAK keys to interrupt a program, and EOF to indicate an "end-of-file".

When = 0, program interruption is not available from the SHIFT-BREAK keys, and an "end-of-file" will only be indicated by the last block in a file being detected. (This is useful for reading files that may contain ANY characters as part of the data eg. "program files".

Bit 4 (Trailing Space Mode):

When = 1, trailing spaces will be printed after any NUMERIC output.

When = 0, no trailing spaces are printed and numbers will run into one another (in the same way that strings do normally).

Bit 5 (Leading Space Mode):

When = 1, a leading space is printed for numeric output of positive numbers, or alternatively a negative sign for the output of negative numbers.

When = 0, no leading spaces are printed. The examples below show a comparision of this effect.

```
IOM 5,1                 IOM 5,0
10 A=456                10 A=456
20 B=765                20 B=765
30 PRINT A;B            30 PRINT A;B

This displays           This displays
456 765                 456765
```

Bit 6 (Automatic LINE FEED Mode);

When = 1, this outputs a line feed character whenever a new line is output. In other words the BASIC thinks it is sending a CARRIAGE RETURN/LINE FEED at the end of each line. (The Print Head moves to the beginning of the next line down).

When = 0, this outputs a carriage return only (i.e. Print Head returns to the beginning of the same line).

The setting of this BIT has NO affect on device 0 (the display screen), thus normal PRINT statements to the screen are unchanged.

It is useful when set OFF for reducing the size of a file on output, since the Line Feed code is ignored in input of a file using PRINT#. However, some files may need the Line Feed code, in which case this BIT should be set ON.

Bit 7 (Expand TABs):

When = 1, the TAB character is expanded to the required number of spaces (eg. the comma (,) is expanded to produce a 10 space zone in PRINT statements).

When = 0, the actual TAB character itself is output as an ASCII code, and is then transmitted to the output device in use.

Example: IOM 7,0

This TAB character is output as an ASCII code and transmitted to the output device for interpretation (eg. printers which have no special TAB settings). Thus the format of the output on a device can be controlled from the software of the computer.

Bit 8 (Printer Port definition)

When = 0 the printer port is defined as the RS232-C PORT. This is the default setting.

Bit 9 (Combined output selection)

When = 1, this causes output to the selected output device only.

When = 0 this causes output to device 0 (VDU) as well as the selected device.

Bit 10 (Printer Echo)

When = 1, output is to the current device only.

When = 0 this causes the PRINTER to "echo" the current output device.

This command is mutually exclusive with IOM bit 11.

Bit 11

When = 1 this inhibits the modification of "ESC" and "FF" to the VDU when the VDU "echos" the current output device.

When = 0, there is no inhibition of the functions.

This command is mutually exclusive with IOM bit 10.

As A Function:

IOM may be used as a function to return the current setting of any of the BITS.

Example: PRINT IOM(3)

This will display the current setting of BIT 3 (i.e. either 0 or 1) on the screen.

Related Keywords:

KBD (Keyboard)

Syntax: KBD

Purpose: This is a Function similar to INCH but only scans the keyboard for input. It does not wait for an input character.

This function returns a value of 0 if no character is input, or the ASCII value if one has been input.

A useful application of this function (and also KBD$) can be seen in "space invader" type games where horizontal and vertical movement of an object is controlled by allocating certain keys for left/right and up/down movement. The object appears to wait for a response from the particular keys but without halting program execution (i.e. other objects on the screen continue to move).

Example:

```
x = KBD
```

This will return the ASCII value of any character, input from the keyboard, in x.

Related Keywords: INCH INCH$ INPUT KBD$

KBD$ (Keyboard String)

Syntax: KBD$

Purpose: This is a String Function which performs in similar manner to INCH$ except that it does not wait for an input character, but will return a null string if no character is available.

This function is only operative with the computer keyboard, irrespective of what alternative input device is in use at the time, whereas INCH$ responds to all input devices.

This function can be used in the same format as INCH$ bearing in mind the points of difference indicated above.

Related Keywords: INCH INCH$ INPUT KBD

KEY

Syntax: KEY N, <string expression>

Where N is the function key number and the data is the particular function to be allocated to that key

Purpose: The KEY command allows the user to program the 8 special "function keys" labelled F0 to F7 on the keyboard, according to individual requirements.

The key numbers are 0 to 7 in "unshifted" mode (as labelled) and 8 to 15 in shifted mode (i.e. each key can be used for two separate functions)

BASIC reserved words are stored as tokens for efficiency. ASCII characters can also be stored.

The total storage capacity for all 8 keys is 128 bytes. This can be allocated to one key or shared between the 16 functions of all 8 keys.

Example:

`KEY 0,"LISTCR-`	
`KEY 1,"PRINT CHR$(A)CR-`	This programs function key 1 to print CHR$(A) when pressed.
`KEY 3,"BCOL5:TCOL15 CR -`	This programs function key 3 to set backdrop and text colours when pressed.
`KEY 5,"B=58:PRINT CHR$(B)CR-`	This programs function key 5 to set variable B to 58 and then print it.

To display the contents of the function keys use the KEY LIST command.

`KEY LIST`	This will display the contents of all the function keys on the screen, as shown in the example display given below.

```
F0: LISTCR
F1: PRINT CHR$(A)CR
F2:
F3: BCOL5:TCOL15 CR
F4:
F5: B = 58:PRINT CHR$(B)CR
F6:
F7:
```

Shifted functions are shown in listings as follows:

```
sF0:
sF1:
sF2:
sF3:
sF4:
sF5:
sF6:
sF7:
```

NOTES

Each time a programmed function key is used, the BASIC statement stored in the string expression is displayed on the screen. In order to execute the statement, press ENTER in the usual way.

Should you wish to execute the function key immediately when the key is pressed, the ENTER command must be embedded within the print statement. This is done by simultaneously pressing the GRAPH and ENTER keys. The command is shown in the listing as ␍(carriage return).

In a similar way the GRAPH key can be used to embed control codes within function keys.

The function keys are pre-programmed when the master disc is loaded, or when any user disc has been configured to define the function keys. See DOS utilities, Chapter 5.

Related Keywords:

LEFT$ (Left String)

Syntax: LEFT$(string expression **,J)**

Purpose: This is a String Function which returns the left most J characters from the string expression.

Examples:

a) PRINT LEFT$("ABCDEF",3)

The 3 left most characters of the string will appear on the screen (i.e. "ABC")

b) A$ = LEFT$("XYZLBJ",4)

This will return the 4 left most characters in A$. Thus A$ will be "XYZL"

Related Keywords: LEN$ MID$ MUL$ RIGHT$ SCRN$

LEN

LEN (Length)

Syntax: LEN (<string expression>)

Purpose: This is a String Function which returns the length of the string expression as the number of characters, including punctuation marks, control characters, and spaces.

Examples:

a) PRINT LEN("HELLO,AND WELCOME")

This will give a value of 17 which then appears on the screen.

b) PRINT LEN("AB   ,CDE   ,X,Y,Z")

This will return a value of 18.

Related Keywords: LEFT$ MID$ RIGHT$

LET

LET

Syntax: LET V=E

Purpose: This is used to assign the value of E (an expression) to variable V. V and E must both either be string or numeric. The word LET is optional and can be omitted.

INTEGER variables can be assigned to ordinary real variable names. They are shown with a percent % symbol suffix.

FLOATING POINT expressions can be assigned to INTEGER variable names but the following points must be taken into consideration,

a) the result must be in the range 0 to +65535.

b) any decimal part will be lost, just as if an INT function has been performed before assigning the result.

c) all variables not specified in a program return the value 0 or the string "", (null).

Examples:

LET A=9	Assigns a value of 9 to the variable A
LET AA=1+2*3/4	Assigns a value of 2.5 to the variable AA (i.e. the result of the mathematical expression).
LET TEMP%=12	Assigns a value of 12 to the integer variable TEMP%
LET B$="YES"	Assigns the string YES to the string variable B$
LET NAME$="JOHN"	Assigns the string "JOHN" to the string variable NAME$
C(4)=6	Assigns the value 6 to variable C(4) which represents the fifth element in the array C.
D$="NO" -	Assigns the string "NO" to the string variable D$.
NAME$="FRED"	Assigns the string "FRED" to the string variable NAME$.
A%=PI	This would give the same result as A=INT(PI)

Therefore the value assigned to A% would be 3. (PI=3.14159)

Related Keywords:

LIST

LIST

Syntax: LIST L1,L2,L3

L1 is a line number from which the listing will commence and defaults to 0 if omitted.

L2 is the number of lines to be listed at one time and defaults to 15 until changed by a value in the command.

L3 is the last line number to be listed.

L1,L2,L3, can also be in the form of expressions.

Purpose: This is a System Command which lists the current program to the current output device.

After L2 lines have been listed there is a pause. The listing will continue when the user presses the space-bar or any key, and another L2 lines will be listed. BASIC always remembers the last value used for L2 and will continue to use that value until LIST is used with a different L2 value.

Any or all of the expressions L1,L2,L3, may be omitted but if L2 or L3 are specified, either individually or together, then the appropriate "commas" must be inserted.

Listings may be abandoned at any time, whether paused or not, by use of the ESC key.

When a listing has paused, the cursor movement keys may be used for editing purposes and this simultaneously abandons the listing without the use of ESC. (This is only applicable in screen edit mode).

LIST may be used as a normal statement within a program. This is useful for displaying segments of a program listing during execution, and a delay could be incorporated so as to hold the display on the screen for a short period of time. REM statements containing titles, and statements containing data, for example, could be programmed to appear for a short time on the screen at predetermined stages of execution.

Examples:

LIST 300,5,999	- This will list 5 lines at a time starting at line 300 and ending at line 999.
LIST	- Lists the whole program.
LIST,5	- Lists the whole program, 5 lines at a time.
LIST 100,7	- Lists 7 lines at a time, starting from line 100
LIST 200	- Lists from line 200 using the last used value for L2 for the number of lines to be listed at one time
LIST 100,,199	- Lists from 100 to 199 using the last used value for L2 for the number of lines to be listed at one time.
LIST,4,299	- Lists 4 lines at a time from the start of the program to line 299.
LIST,,199	- Lists from the start of the program to line 199 using the last used value for L2 for the number of lines to be listed at one time.
X=100:Y=50:LISTX,Y,X+99	- This will list from 100 to 199, 50 lines at a time.

Related Keywords: LISTP

LISTP (List to Printer)

Syntax: LISTP L1,L2,L3

Purpose: This is a System Command which will list the current program to the printer from line number L1 to line number L3.

If L1, L2, and L3 are omitted the whole program will be listed.

It is the same as doing PRINT#1;LIST.

During the listing, L2, the 'number of lines at a time' feature incorporated in the LIST command, is ignored and the list is printed continuously up to line L3.

After listing, the output reverts back to device 0 (the monitor)

If the command is called from within a program, device 1 (the printer) will remain selected.

Related Keywords: LIST PRINT# PRINT

LN (Logarithm - Natural)

Syntax: LN(N)

Purpose: This is a Function which returns the natural logarithm of the numeric expression N (Natural logs are to the base e).

If the value given by N is less than, or equal to, zero a QTY ERROR occurs.

Example:

```
PRINT LN(2)
```

The value 0.693147 will appear on the screen.

Related Keywords: EXP LOG

LOAD

Syntax: LOAD "<ufn>"

Purpose: This is a System Command which loads files/programs from the disc into the computor's memory. If a file is not present on the target disc then a NO FILE ERROR will be given.

If the drive name is omitted from within the file the current default drive will be assumed. The default drive is initially set up as 0 but can be changed using the DRIVE command.

If the file type is not declared within file then XBS will be assumed.

Loading Basic Program Files

When loading BASIC program files any existing BASIC program in memory is deleted but variables are not destroyed.

Example: LOAD "1:PROG.XBS"

This will load the BASIC program file whose name is PROG from the disc in drive 1.

Example: LOAD "PROG"

This will load the BASIC program file PROG.XBS from the default drive 0 into the computer's memory.

Loading ASCII Program Files

When loading ASCII program files, any ex.sting program is not deleted and variables are not destroyed. Thus extra routines may be added to existing programs and the extra lines will appear at their correct positions in relation to the existing ones (care being taken not to duplicate line numbers otherwise the original will be over written by the new line).

If a "new" program is to be loaded as an ASCII file then the NEW command must be executed prior to loading. In this mode the user may observe the program file loading line by line on the screen.

Example: LOAD "1:TEST.ASC"

This will load the ASCII program file TEST.ASC from drive 1 , into the computer's memory, the lines of the program appearing on the screen as it is loading.

Loading Object Program Files (Machine-Code Files)

When loading OBJECT files, an area in memory has to be reserved prior to loading using the CLEAR command.

The start address will be assumed to be the first location above the cleared area. (i.e. if CLEAR &9FFF has been used then loading will commence from &A000).

Machine Code Subroutines can be loaded during execution of a BASIC program using this facility and then accessed by use of the CALL command as required. The subroutine should terminate with a &C9 code in order to return execution to the BASIC program at the line immediately following that containing the CALL command.

If the size of a file is larger than the area of memory available then a MEM FULL ERROR will be given.

Example: LOAD "0:ROUTINES.OBJ"

This will load the machine-code routines, or data from the file ROUTINES.OBJ, from the disc currently in drive 0, into the area of memory previously reserved by a CLEAR command, i.e. &A000 in the example above.

GENERAL NOTE: If a file is not present on a particular disc then a NO FILE error will be given.

Related Keywords: CLEAR DRIVE SAVE

LOCK

Syntax: LOCK "<ufn>"

Purpose: This is a Disc Command which locks the specified file. Files PROTECTED in this manner may not be re-written over, erased, or renamed (i.e. cannot be corrupted).

A 'Locked' file is marked in the directory with a star preceding the filename, e.g:-

```
*SOUND.XBS
```

If the file does not exist a NO FILE error occurs.

Related Keyword: UNLOCK

LOG (Logarithm)

Syntax: LOG(N)

Purpose: This is a Function which returns the logarithm to the base 10, of the value given by N.

If N is less than or equal to zero a QTY ERROR occurs.

Example:

```
PRINT LOG(2)
```

The value .30103 will appear on the screen.

Related Keywords: LN EXP

MAG

MAG (Magnify)

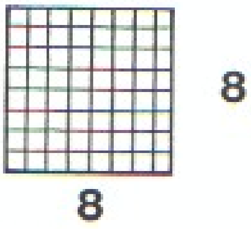
Syntax: MAG J

Where J can have a value from 0 to 3, each number representing a particular sprite magnification. All 32 sprites are affected equally.

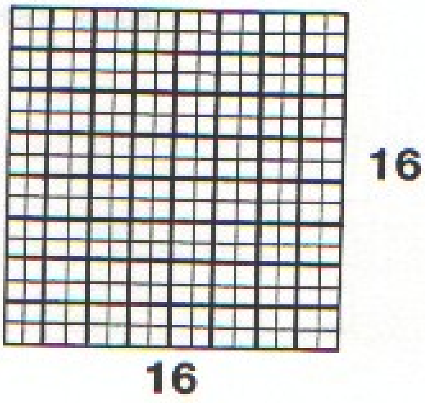
Purpose: This is the Sprite Magnification command.

J	MAGNIFICATION
0	- defines an 8x8 pixels sprite.
1	- defines a double sized 8x8 sprite
2	- defines a 16x16 pixels sprite
3	- defines a double sized 16x16 sprite

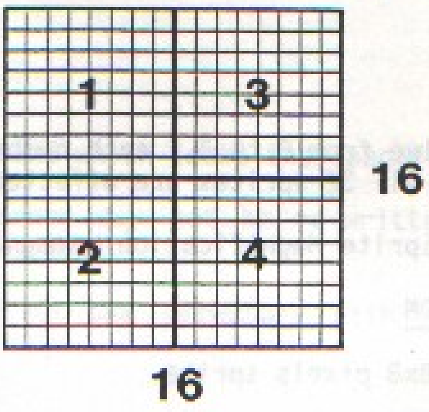
MAG 0 and 1 apply to a shape which has been defined on a single 8x8 pixel grid. In MAG 0 the shape remains as a single 8x8 pixel character as shown below.



In MAG 1 the single 8x8 shape is doubled in size (magnified) to occupy an area equivalent to a 16x16 pixel grid. Each original pixel is in fact made 4 times larger, resulting in a grid of 8x8 larger pixels as shown below.



MAG 2 and 3 apply to a shape which is built up from four 8x8 pixel grid shapes to form a single shape on a 16x16 grid. In MAG 2 the four shapes are printed as a single shape as a 16x16 pixel grid as shown below.



When "defining" the complete shape the data for each 8x8 grid should be used with the SHAPE command in the order shown above. In MAG 3 the shape on this 16x16 grid is doubled in size to occupy an area equivalent to 32x32 pixels.

Example:

MAG 1

Following this command all 32 sprites would be double sized 8x8 until another MAG command was introduced to change the selections.

Related Keywords: SHAPE SPRITE

MGE

MGE (Merge)

Syntax: MGE

Purpose: This is a command used to "merge" sections of a program which have been "held" or, add a second program to one already in memory.

MGE takes no account of two or more lines having the same number and therefore both lines would appear in the final result.

Example:

```
 10 REM LINES KEPT IN VIEW
 20 PRINT
 30 PRINT "*"
 40 PRINT
 50 PRINT "#"
 60 REM LINES HIDDEN
 70 PRINT "A"
 80 PRINT
 90 PRINT "B"
 100 PRINT
```

A HOLD 10,50 will keep lines 10 to 50 of the program while lines 60 to 100 are apparently lost. Thus a LIST after this use of HOLD would give the following.

```
 10 REM LINES KEPT IN VIEW
 20 PRINT
 30 PRINT "*"
 40 PRINT
 50 PRINT "#"
```

The execution of MGE brings back lines 60 to 100 and if LIST is used after MGE the original listing will reappear (i.e. lines 10 to 100).

See also the example mailing list in File Handling which illustrates the use of MGE.

Related Keywords: CHAIN HOLD RENUM

MID$(Middle of String)

Syntax: MID$ (<string expression>, J1,J2)

Purpose: This is a String Function which returns a number of characters, given by the value of J2, starting from the character position given by J1, in respect of the string specified in the function.

J2 may be omitted, in which case the remainder of the string, starting from the character position given by J1, is returned.

Examples:

```
PRINT MID$("HELLO",3,2)
```

The result will appear on the screen as "LL".

```
PRINT MID$("HELLO",3)
```

The result will appear on the screen as "LLO".

Related Keywords: LEFT$ RIGHT$

MOD (Modulus)

Syntax: I1 MOD I2

Purpose: This is an ARTHMETIC OPERATOR which forms an expression equal to the integer value of the remainder resulting when I1 is divided by I2.

Example:

5 MOD 3 returns a value of 2

Related Keyword: INT

MODE (serial port mode)

Syntax: MODE L,P,S

Purpose: This is a command to set up the mode of the serial interface.

L = Word length in bits. L can have values in the range 5 to 8.
P = Parity. 0 = no parity; 1 = odd and 2 = even parity.
S = Number of stop bits. 0 = 1 stop bit; 1 = 1½ stop bits and 2 = 2 stop bits.

The default values are: L = 8; P = 0; S = 2

Example: MODE 7,2,1

This gives 7 data bits, even parity and 1½ stop bits

Related Keyword: BAUD

MOS (Machine Operating System)

Syntax: MOS

Purpose: This is a System Command and is used to transfer control to the "Machine Code Monitor" section of the "Machine Operating System".

The X and Y commands in MOS can be used for cold and warm starts when re-entering BASIC.

Related Keyword: DOS

MUL$ (Multiple String)

Syntax: MUL$(<string expression> ,J)

Purpose: This is a String Function which will cause a string to be repeated the number of times, given by the value of J (i.e. String Multiplication).

The resultant string must not exceed 255 characters.

Example: PRINT MUL$("AB",10)

This will give the following display.

```
ABABABABABABABABABAB
```

It is a useful function for displaying repeated patterns.

Examples:

i) PRINT MUL$("*",15)

This will give the following.

```
***************
```

ii) PRINT MUL$("+=",6)

This will give the following.

```
+=+=+=+=+=+=
```

Related Keyword: STR$

MUSIC

Syntax: MUSIC S1;S2;S3

Purpose: This command can be used to create various sounds and play tunes composed by the user.

String expressions S1,S2 and S3 each represent musical expressions. These expressions are each allocated to a separate channel, the 3 channels are then played simultaneously.

It is not necessary to use all the three channels every time and any expressions which are left "empty" will cause a previous string expression specified for that channel to be replayed.

It is therefore good practice to specify a rest (R) for channels not required to sound. The MUSIC statement does not end until all three channels have completed their appropriate measures. This makes it important to ensure that all three channels have the same execution time within a given measure.

The contents of the expressions S1,S2, and S3 indicate the actual structure of the music to be played. Individual notes are indicated by the letters CDEFGAB, these representing the chromatic scale.

The following can be included in a music string:-

1. R represents a 'rest'.
2. X represents a 'beat', and turns on the noise generator of the PSG rather than a tone.
3. Vn, where n is numeric data in the range 0 to 6 which selects a voice defined using the VOICE commands.

The duration of a note (or rest) is specified by a number, in the range 0 to 9, immediately following the note (or rest), giving the note values as listed below.

0 = 32nd note (demi-semi-quaver)
1 = 16th note (semi-quaver)
2 = "dotted" 16th note
3 = 8th note (quaver)
4 = "dotted" 8th note
5 = Quarter note (crotchet)
6 = "dotted" quarter note
7 = Half note (minim)
8 = "dotted" half note
9 = Whole note (semi-breve)

If the duration specification is missing from a note, the duration defaults to the last specified duration on that channel. On power up the duration is 5.

When written into the form of the MUSIC command the result would be:

```
MUSIC "CR","ER","GR"
```

All three notes would be played simultaneously thereby producing a chord. It is of course possible to play one single channel by omitting the other two from the command as shown below.

```
MUSIC "CEGR","R","R"
```

This would play the three single notes in turn, on channel A, one after the other, followed by a rest in order to silence the last note.

The simple format, shown above, can be extended such that the contents of each channel may be represented in any of the following formats.

a) a single String of notes.
b) a single String Variable (previously allocated to a set of notes).
c) a combination of several String Variables or Strings.

Example:

```
10 A$ = "AFGCDE"
20 B$ = "EFCDG"
30 C$ = "CDEGBR"
40 MUSIC "GEBCD",A$,B$+C$
```

To increase the octave range each note can be preceded by a symbol to produce the effect given in the list which follows:-

+ - raises the note one octave above middle C.
' - raises the note two octaves above middle C.
- - lowers the note one octave below middle C.
= - lowers the note two octaves below middle C.
. - lowers the note three octaves below middle C.

"Sharps" and "flats" are indicated as follows.

- precedes the note and indicates a "sharp".
b - precedes the note and indicates a "flat".

Example:

```
10 MUSIC "O +#A7 =bE5 R","R","R"
```

Music is played on channel A as follows:-

O - VOICE O is selected (VOICE O having been defined in a voice statement).

+ - Raises the note one octave.

- Sharpens the note.

A - The note played

7 - Duration setting of a half note (minim)

NOTE: Spaces included in the string are ignored and can be used to increase legibility.

The second note played is E flat played two octaves down for a duration of a quarter note (crochet).

Related Keywords: BEEP PSG TEMPO VOICE

NEW

Syntax: NEW

Purpose: This command deletes all program lines and variables currently in the memory.

It is used to delete any program currently in memory in preparation for the input of another program and different variables.

Used in "Direct Mode".

Related Keywords:

NEXT

Syntax: NEXT V1,V2,...,Vn

V1,V2, etc. are the control variables used in the associated FOR statements, and must be numeric.

Purpose: This statement is used to indicate the end of a FOR statement loop.

NEXT is always used in conjunction with the FOR statement to produce what is commonly referred to as a FOR-NEXT LOOP.

For a more detailed explanation of see FOR command.

Example:

```
  10 CLS
  20 FOR I=1 TO  10
  30 PRINT I
  40 NEXT I
  50 END
```

Related Keywords: FOR STEP

NOT

Syntax: <statement> NOT <statement>

Purpose: This is a LOGICAL OPERATOR used in the evaluation/comparision of statements.

Example:

```
IF NOT X THEN END
```

This shows the use of NOT in an "IF" statement. In other words IF the condition is NOT true THEN act accordingly.

Related Keywords: AND OR XOR

NULL

Syntax: NULL J

The value given by J is the number of nulls to be printed and can be from 0 to 255. The default value is 0.

Purpose: This is a special command relating to output. It sets the number of nulls to be printed after every carriage return (i.e. acts as a delay).

This command is designed to be used when operating slow serial devices, where the carriage return code may take a little over the time allowed for one character to print.

The correct setting for a particular device will be found by experimenting with various values but normally 1 would be sufficient for most devices.

NULL can be used as a function to return the current setting.

Example:

```
NULL 5

PRINT NULL
```

This will display the value of the current setting for NULL on the screen (i.e. 5 in this instance).

Related Keyword: SPEED

OFF

Syntax: OFF <command>

Purpose: This command is used to disable a particular action or operation setting. Often used with ERR and EOF commands. For further information refer to Chapter 11, Error Handling Within BASIC.

Example:

```
  OFF ERR - Disables the ON ERROR trap
  OFF EOF - Disables the ON EOF (end-of-file) trap
```

Related Keywords: EOF ERR

ON

Syntax: ON J GOTO L1,L2,..,Ln

J is an expression and L1,L2 etc are line numbers. If J is negative then a QTY ERROR occurs.

Purpose: This alters the order in which BASIC executes a program by jumping to one of a selection of lines depending on the value of expression J.

The expression J is evaluated to give a number. If the number is 1 then execution transfers to the 1st line number after the GOTO. If the number is 2 then execution transfers to the 2nd line number after GOTO. and so on.

Example:

```
  10 INPUT "TYPE IN A NUMBER FROM (1-5)";A
  20 PRINT "YOU HAVE SELECTED";
  30 ON A GOTO 40, 60, 80, 100, 120
  40 PRINT "GLASSES";
  50 GOTO 130
  60 PRINT "CUPS;
  70 GOTO 130
  80 PRINT "TANKARDS";
  90 GOTO 130
  100 PRINT "DISHES";
  110 GOTO 130
  120 PRINT "MUGS";
  130 END
```

When this program is RUN the following appears on the screen to begin with.

```
TYPE IN A NUMBER FROM (1-5)
```

When a number is typed in then the appropriate line number is selected from the ON-GOTO statement.

```
NUMBER 1 selects line number 40
NUMBER 2 selects line number 60
NUMBER 3 selects line numner 80
NUMBER 4 selects line number 100
NUMBER 5 selects line number 120
```

Execution is then transferred to the selected line and the appropriate action is taken; in this case a message is output to the screen. Thus transfer of execution has been controlled by the variable A.

Syntax: ON J GOSUB L1,L2,..Ln

Purpose: The same principle can be applied to accessing subroutines using the GOSUB format.

Thus several subroutines can be accessed from a single line of program depending on the evaluation of a variable.

Example:

```
 10 REM PROCESSING TO EVALUATE A VARIABLE B
 20 - - - - - - - - - -
 30 - - - - - - - - - -
 40 - - - - - - - - - -
 50 - - - - - - - - - -
 60 ON B GOSUB 100,140,180.
 70 - - - - - - - - - -
 80 - - - - - - - - - -
 90 END
100 SUBROUTINE 1
110 - - - - - - - - - -
120 - - - - - - - - - -
130 RETURN
140 SUBROUTINE 2
150 - - - - - - - - - -
160 - - - - - - - - - -
170 RETURN
180 SUBROUTINE 3
190 - - - - - - - - - -
200 - - - - - - - - - -
210 RETURN
```

Let us assume that there are three subroutines in the program which will be accessed by the ON-GOTO statement in line 60

The variable B is evaluated by the processing involved in lines 20 to 50 and will produce a number from (1-3).

The Subroutines are then accessed via the ON-GOSUB statement, selection depending on the value of B.

If B = 1 then line 100 (subroutine 1) would be accessed.
If B = 2 then line 140 (subroutine 2) would be accessed.
If B = 3 then line 180 (subroutine 3) would be accessed.

The subroutines would RETURN to line 70 as normal to continue program execution. (See page 205).

Syntax: ON ERR GOTO
ON ERR GOSUB
ON EOF

Purpose: See ERR and EOF commands and also Chapter 11, Error Handling Within Basic.

Related Keywords: EOF ERR GOSUB GOTO OFF

OPEN

Syntax: OPEN <ufn> ,SV,R

Purpose: This is a File-Handling Command which opens an existing file, assigns internal file information and buffer space to the file descriptor, and indicates the record size to be used.

SV is a string variable name (but not a string array element) and is the file descriptor.

R is the random record size (length) and is given as a value in the range 0 to 65535, indicating the number of characters involved. R is only specified for "random access"; if "sequential access" is to be applied then R is omitted (in fact a random record length of 0 indicates that sequential access is to be performed).

If a file is not present on the specified drive then a NO FILE error will be given.

Example:

```
OPEN "0:SILLY.DAT",FD$,15
```

This will perform the following:-

a) opens the file SILLY.DAT on the disc currently in drive 0.
b) assigns FD$ as the file descriptor.
c) sets up for random access using 15-character length records.

Related Keywords: APPEND CLOSE CREATE

OR

Syntax: <statement> OR <statement>

Purpose: This is a logical operator used in the evaluation/comparison of statements (i.e. defines an alternative).

Example:

```
10 IF (X AND Y)=3 OR Y=0 THEN 100
```

This will cause execution to transfer to line 100 if the result of either of the statements (X and Y)=3, or Y=0 is TRUE

Related Keywords: AND NOT XOR

ORIGIN

Syntax: ORIGIN x,y

Both x and y can have values in the range -32768 to +32767 and are the co-ordinates of a point on the screen.

Purpose: This statement defines the origin of the imaginary screen grid in respect of PLOT, UNPLOT, DRAW, ELLIPSE, and POLY commands, x and y being the co-ordinates of the new origin. On entry to BASIC the default co-ordinates are 0,0 and this represents the bottom left hand corner of the screen.

Examples:

a) ORIGIN 20, 24
This would establish the new position of the origin on the screen grid and all subsequent PLOT, UNPLOT, DRAW, ELLIPSE, and POLY commands would be executed relative to the new origin.

b) PLOT 128,96
This plots a point at the centre of the screen, assuming that the origin has not been redefined since entry to BASIC.

ORIGIN 128,96:PLOT 128,96
Redefines the origin and plots a point at the top right hand corner of the screen.

Related Keywords: DRAW ELLIPSE PLOT POINT POLY SCREEN UNPLOT

OUT

Syntax: OUT J1,J2

J1 is the exchange "address" of the port, and J2 is the value which is to be output.

Purpose: This command provides direct output to the ports of the computer.

Example:

The joystick port is at I/O address &32 so this will be the value of J1 each time this command is used to access the port.

```
OUT &32,5
```

This sends the value 5 to the 8 bit user port (&32).

For information relating to the ports, refer to appendix L.

Related Keywords: INP WAIT

PCOL (Palette Colour)

Syntax: PCOL N,R,G,B

Purpose: This command re-defines the colour produced by the BCOL, GCOL and TCOL commands. N represents the colour value within the palette, and can be in the range 0-15. R,G,and B represent the amount of Red, Green and Blue which make up a particular colour, each can have values in the range 0-7, where 7 represents the maximum amount of colour, and 0 represents no colour. Any unspecified colour parameters default to zero.

When BASIC is first loaded, after a X command from MOS or after an RST command in BASIC, the colour palette is initialised to its default colours. The default colour palette is:

0 Transparent
1 Black
2 Medium Green
3 Light Green
4 Dark Blue
5 Light Blue
6 Dark Red
7 Cyan
8 Medium Red
9 Light Red
10 Dark Yellow
11 Light Yellow
12 Dark Green
13 Magenta
14 Grey
15 White

Up to 512 colours colour can be generated using this command.

Example: PCOL14,7,5,5 will define palette colour 14 to be pink.

A Note About Colour.

The best way to determine the values for a particular colour is to experiment. However, here are a few guidelines:

Einstein 256 works on the colour addition principle. This is a little different from the colour subraction principle which artists use to mix colours.

The primary colours are:

Red (R)
Green (G)
Blue (B)

The complementary colours are:

Cyan (Bluish/Green)	Blue + Green
Yellow	Red + Green
Magenta (Purple)	Red + Blue

White is made up of equal amounts of Red, Green and Blue.

Red + Green + Blue.

Pastel colours are made by adding white to saturated colours, thus Pink is made by adding White to Red. e.g. Saturated Red would be given by:

PCOL 6,7,0,0, (i.e. 7 units Red, no Green, No Blue)

To make a Pink, this could be:-

PCOL 6,7,5,5 (i.e. 5 units of White and 2 units of Red)

Grey would be :- PCOL 14,3,3,3

A lighter grey could be: PCOL 14,5,5,5

White is PCOL 15,7,7,7

Related Keywords: BCOL GCOL TCOL

PEEK

Syntax: PEEK(I)

I must evaluate to a number in the range 0 to +65535 and can be given in decimal or hexadecimal.

Purpose: This is a Machine Code related command which returns an integer in the range 0 to 255 which represents the contents of the memory location given by I.

Example:

```
PRINT PEEK(&4100)
```

This will display the contents of memory location & 4100 as an integer.

Related Keywords: CALL DEEK DOKE POKE VDEEK VDOKE VPEEK VPOKE

PI

Syntax: PI

Purpose: This is a Function which returns the value of pi as 3.14159 for use in expressions.
It is much faster than using a variable to hold the value of pi.

Example:

```
PRINT PI
```

The value 3.14159 will appear on the screen.

Related Keywords:

PLOT

PLOT

Syntax: PLOT x,y

x and y are the co-ordinates of any point on the screen. x being horizontal and ranging from -32768 to +32767, y being vertical and ranging from -32768 to +32767.

Purpose: This is a Command used to illuminate (turn on) a single pixel point on the display screen.

The screen can be thought of as being divided into a number of horizontal pixels, and a number of vertical pixels. The larger the number of pixels in either direction, the better is the resolution of the system.

Einstein 256 has a number of graphics modes. BASIC supports two of these. In graphics mode 2, the screen has a resolution of 256 pixels horizontally, and 192 pixels vertically. In graphics mode 6, the screen has a resolution of 512 pixels horizontally, and 192 pixels vertically. The two display modes are selected using the SCREEN command.

Plotting cannot take place in screen mode 6, which is text only.

If the origin is defined as 0,0 (the default value) then only pixels in the range x = 0 to 255 (or 511) and y = 0 to 191 can be illuminated on the screen. However, all other values in the range specified, above, are allowed, but will be off the screen.

Mode	V9938 Mode	Text Resolution	Graphics Resolution (pixels)
SCREEN 0	graphics 2	32 x 24	256 x 192
SCREEN 1	graphics 2	40 x 24	256 x 192
SCREEN 2	text 2	80 x 24	Text Only
SCREEN 3	graphics 6	32 x 24	512 x 192
SCREEN 4	graphics 6	40 x 24	512 x 192
SCREEN 5	graphics 6	60 x 24	512 x 192
SCREEN 6	graphics 6	80 x 24	512 x 192

Example:

```
PLOT 120,90
```

This will illuminate the pixel at 120,90 in the current foreground graphics colour (as set by a previous GCOL command), provided the ORIGIN is defined as 0,0

Related Keywords: DRAW ELLIPSE ORIGIN POINT POLY UNPLOT

POINT

Syntax: POINT(x,y)

x and y are the co-ordinates of a graphics point on the imaginary screen grid. Values for x and y can be in the range -32768 to +32767.

Purpose: In screen modes 0 and 1 the function returns a value corresponding to the state of the pixel at x,y. 0 = pixel off, 1 = pixel illuminated, and 255 = off screen, in either x or y or both.

In screen modes 3 to 6, inclusive, the function returns the value of the colour palette (0-15) at the point, or 255 if either x, or y are off screen.

Example:

```
1) SCREEN 1
X = POINT(70,65)
PRINT X
```

This will display a value of 0 or 1, according to the condition of the point whose co-ordinates are 70,65. (i.e. whether lit or not).

```
2) SCREEN 4
X = POINT(70,65)
```

This will display a value of 0 to 15, according to the value of the colour palette at the point 70,65

Related Keywords: DRAW ELLIPSE ORIGIN PLOT POLY UNPLOT

POKE

Syntax: POKE I,J1,J2,..,Jn

Purpose: This is a Machine Code related command which places the values of the expressions J1,J2 etc., into memory, starting at the location given by I.

Examples:

```
POKE 16384,132
```

This will place 132 (i.e.&84) into location 16384 (i.e.&4000)

```
POKE &5100,&77,&34,&61
```

This will put &77 into location &5100, &34 into location &5101, and &61 into location &5102.

Related Keywords: DEEK DOKE PEEK VDEEK VDOKE VPEEK VPOKE

POLY

POLY (Polygon)

Syntax: POLY N,x,y,R,T,z,a,b

Purpose: This graphics command will draw a polygon according to the values given in the parameters.

N is the number of sides of the polygon. x,y are the co-ordinates of the centre of the polygon and can have values in the range -32768 to +32767. R is the distance from point (x,y) to the vertices of the polygon i.e. R is the horizontal radius of an ellipse which would contain the polygon and T is the ellipse qualifier given by the following:-

$$T = \frac{\text{VERTICAL AXIS (of ellipse)}}{\text{HORIZONTAL AXIS (of ellipse)}}$$

If T is omitted it will default to 4/3 thereby having the same effect as in the ELLIPSE command, resulting in a REGULAR POLYGON (owing to the aspect ratio of the screen being 4:3).

NOTE: The aspect ratio of 4/3 is true only in screen modes 0 and 1, and in 625 lines. The default value is retained for compatibility with Einstein programs.

For other screen modes, and line standards, the value for T, needed to display a regular polygon is shown in the table below.

Screen mode	T(525 lines)	T(625 lines)
0	1.167	1.333
1	1.167	1.333
2*	not applicable	not applicable
3	0.58	0.67
4	0.58	0.67
5	0.58	0.67
6	0.58	0.67

* POLY has no meaning in screen mode 2 (80 column text only).

The value of z is a number in the range 0 to 5 indicating the type of line to be drawn (if omitted z will default to 0).

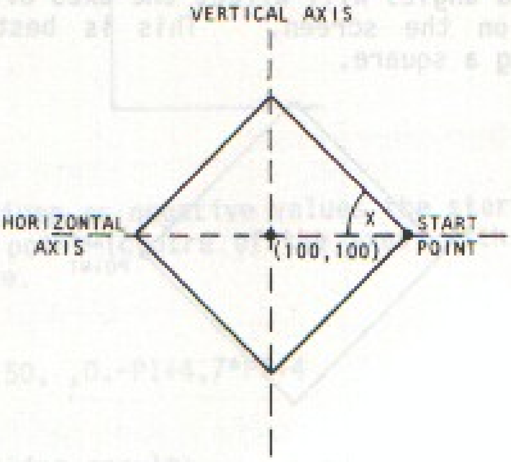
0 - Continuous Line
1 - Continuous Unplot
2 - Dotted line 2 dots on, 2 dot off.
3 - Dashed line 4 dots on, 2 dots off.
4 - Dotted-Dashed line 10 dots on, 2 dots off, 2 dots on, 2 dots off.
5 - Dashed-dotted line 10 dots off, 2 dots on, 2 dots off, 2 dots on, 10 dots off.

'a' and 'b' are start and end angles which indicate where the drawing of the polygon should begin and end. These are optional. The start and end angles are specified in radians and are numbered in an anticlockwise direction from 0 at the right hand horizontal axis up to 2 PI radians for one complete revolution. The values may be specified as numbers or an expression in terms of PI.

The orientation of polygons on the screen is determined by the internal angle between the horizontal axis and the first side of the polygon drawn from the start point. Look at the following example:-

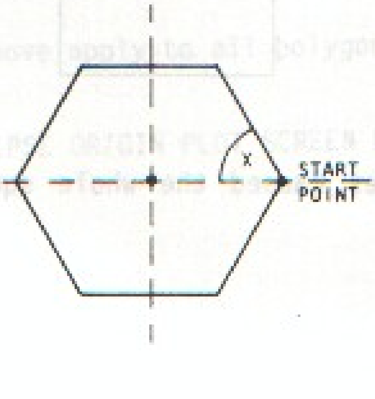
```
    POLY 4,100,100,50, ,0
```

The 'T' parameter and start and end angles ('a' and 'b') are omitted thus producing a square on the screen as shown below (solid outline only). Notice the orientation which depends on the angle at x.



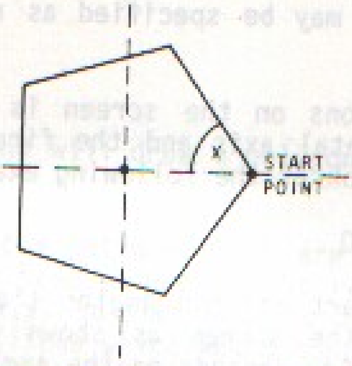
The first side of the polygon commences from the start point on the horizontal and is drawn at the angle given by 'x'. This angle varies according to individual polygons and is equal to half the interior angle at that point. In the case of a square 'x' is 45°, for a regular hexagon 'x' will be 60° giving the orientation shown below.

```
    POLY 6,100,100,50, ,0
```

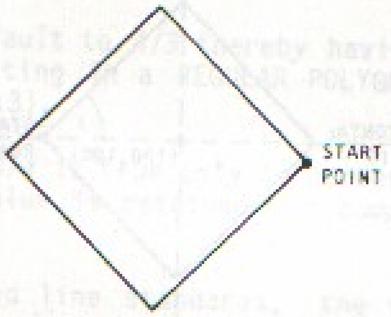


In the case of a regular Pentagon x will be 54° giving the orientation shown below

```
POLY 5,100,100,50, ,0
```

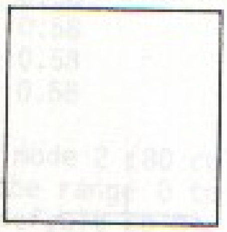


Specifying start and end angles will affect the axes of a polygon and give a different orientation on the screen. This is best illustrated by the following examples using a square.



```
POLY 4,100,100,50, ,0,PI÷4,PI÷4
```

Here the 'T' parameter is omitted but the start and end angles are specified as both being PI÷4 (i.e.0.785). This now produces a complete square with the orientation as shown below.

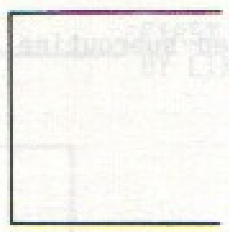


The start and end angles caused the whole square (polygon) to be turned through 45° (PI/4).

If 'a' and 'b' are specified as different values then an incomplete square (polygon) would be drawn (as for an ellipse). The orientation will be affected just as before. Look at the following example.

```
POLY 4,100,100,50, ,0,PI÷4,7*PI÷4
```

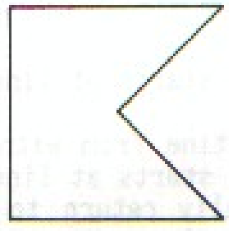
This will produce the following result on the screen.



If 'a' and 'b' are given as negative values the start and end points will be joined to the 'x,y' point (centre of the axes) with lines as illustrated in the following example.

```
POLY 4,100,100,50, ,0,-PI÷4,7*PI÷4
```

This gives the following result:



The principles described above apply to all polygons.

Related Keywords: DRAW ELLIPSE ORIGIN PLOT SCREEN UNPLOT

POP

POP

Syntax: POP

Purpose: This statement is used in association with subroutines.

It allows a nested subroutine to return to the statement immediately following the GOSUB statement preceding the GOSUB relating to the particular routine which is being executed.

POP is only used from within a nested subroutine and this is best explained by the example below:-

```
  10 GOSUB 50
  20 - - - -
  30 - - - -
  40 END
  50 REM.SUBROUTINE
  60 - - - -
  70 - - - -                  OUTER SUBROUTINE
  80 GOSUB 120                CALLED BY LINE 10
  90 - - - -
 100 - - - -
 110 RETURN

 120 REM-NESTED SUBROUTINE
 130 - - - -                  INNER SUBROUTINE
 140 - - - -                     (NESTED)
 150 - - TEST CONDITION       CALLED BY LINE 80
 160 IF "TEST CONDITION" THEN POP
 170 RETURN
```

LINE 10 calls up a subroutine which starts at line 50.

LINE 80 calls up a second subroutine from within the existing subroutine (i.e. the nested subroutine), which starts at line 120
This second subroutine would normally return to line 90 and continue with the remainder of the original subroutine. However, as a result of some kind of "test condition" we might wish it to "return" to line 20 (i.e. to the line following the original GOSUB call). Thus the processing contained within lines 90 and 100 of the original subroutine would be omitted. It is this facility which POP provides.

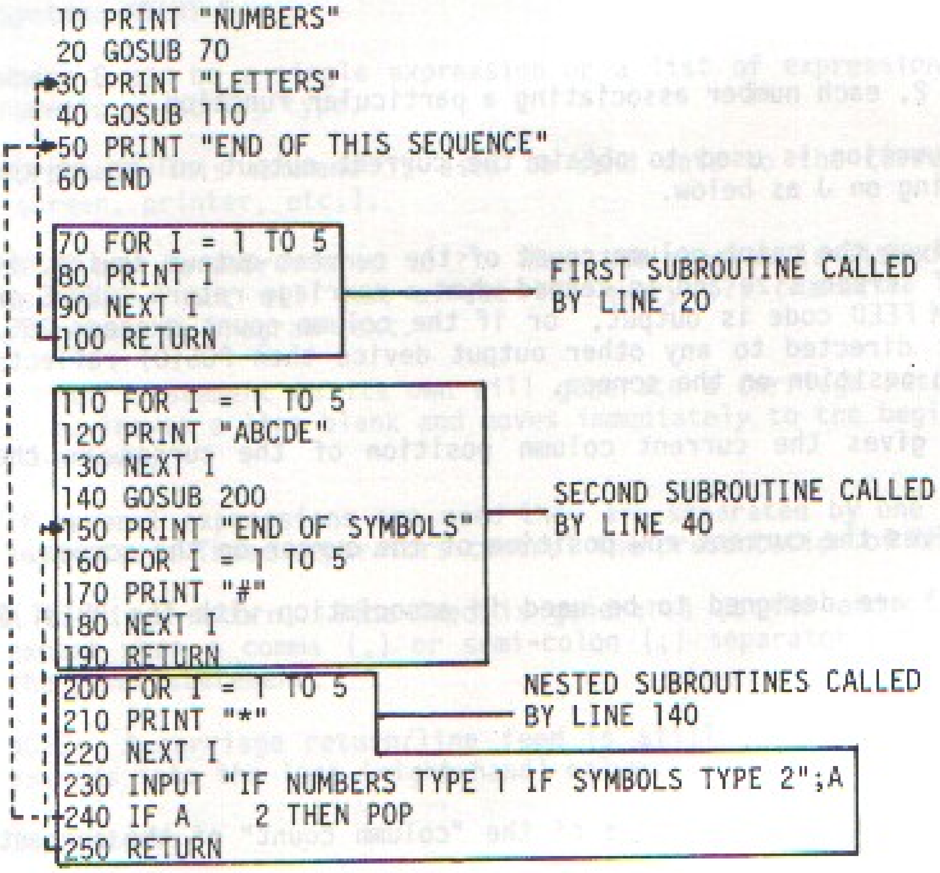
Example:

In the example given below, the result from execution of lines 230 and 240 will direct subsequent execution to either invoke POP or transfer to line 250.

The resulting action from line 250 is indicated by the arrowed lines and will determine whether or not the # symbol will be printed (as contained in the processing of lines 160, 170, and 180).

If the POP statement is executed, i.e. A is not 2, the RETURN statement will direct execution to line 50 rather than returning to line 150.

Example:



```
10 PRINT "NUMBERS"
20 GOSUB 70
30 PRINT "LETTERS"
40 GOSUB 110
50 PRINT "END OF THIS SEQUENCE"
60 END

70 FOR I = 1 TO 5
80 PRINT I
90 NEXT I
100 RETURN

110 FOR I = 1 TO 5
120 PRINT "ABCDE"
130 NEXT I
140 GOSUB 200
150 PRINT "END OF SYMBOLS"
160 FOR I = 1 TO 5
170 PRINT "#"
180 NEXT I
190 RETURN
200 FOR I = 1 TO 5
210 PRINT "*"
220 NEXT I
230 INPUT "IF NUMBERS TYPE 1 IF SYMBOLS TYPE 2";A
240 IF A    2 THEN POP
250 RETURN
```

FIRST SUBROUTINE CALLED BY LINE 20

SECOND SUBROUTINE CALLED BY LINE 40

NESTED SUBROUTINES CALLED BY LINE 140

Related Keyword: GOSUB

POS

POS (Position)

Syntax: POS(J)

J can be 0,1, or 2, each number associating a particular function.

Purpose: This Function is used to obtain the current output column or row position, depending on J as below.

POS (0) - This gives the print column count of the current output device. It is independant of screen size and is zeroed when a carriage return, HOME or clear screen/FORM FEED code is output, or if the column count exceeds 255. If output is not directed to any other output device then POS(0) reflects the cursor column position on the screen.

POS (1) - This gives the current column position of the cursor on the screen.

POS (2) - This gives the current row position of the cursor on the screen.

POS(1) and POS(2) are designed to be used in association with the PRINT @ facility.

Example:

```
PRINT POS(0)
```

This will display the current value of the "column count" of the current output device.

Related Keywords: PRINT PRINT@

PRINT

PRINT

Syntax: PRINT E

Where E can be a single expression or a list of expressions, which may be numeric or string type.

Purpose: This command is used to send data to the current output device (screen, printer, etc.).

PRINT may be abbreviated to ? When the program is listed this will appear as PRINT, but will not affect the query (?) character where it appears elsewhere in program text.

A PRINT statement on its own will generate a carriage return and line feed (i.e. leaves a line blank and moves immediately to the beginning of the next line down).

If several expressions are used they are separated by one of a selection of separators. These separators control the presentation of the final output.

A carriage-return, line-feed is generated at the end of PRINT statements except when a comma (,) or semi-colon (;) separator appears at the end of the print statement.

NOTE: A carriage return/line feed is still generated if the print output extends into the last (right-hand) column.

Separators:

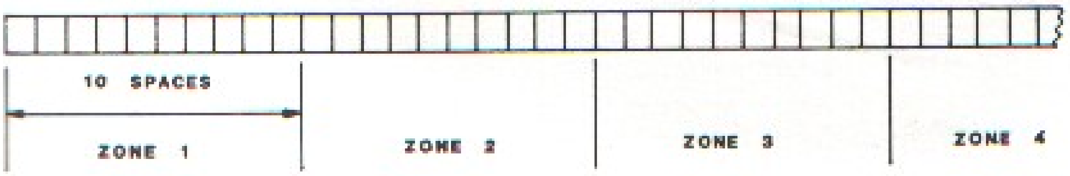
Semi-Colon (;) This leaves the cursor where it is so that the next expression will print from the end of the previous one.

Example: `PRINT "JABBER";"WOKKY"`

This will be output as follows:-

```
JABBERWOKKY
```

Comma (,) This moves the cursor to the start of the next print zone. Print zones are simply specified columns which are situated at intervals of 10 spaces (character positions).



Example: `PRINT "JABBER","WOKKY","WOKKY"`

This will be output as follows:-

```
JABBER    WOKKY     WOKKY
```

Each line on the screen display contains 40 or 80 character positions (according to screen mode) therefore there are 4 print zones. (Or 8 in 80 column).

The zone limit indicates the character position at which the final ZONE starts in a line of text. When the existing printing has gone beyond this point the next expression will be printed at the beginning of the next line (i.e. a carriage-return, line-feed is generated).

The zone width and zone limit may be modified by use of the ZONE command.

The @ Symbol. This allows printing of expressions to commence from a specified point on the screen by use of co-ordinates. (See PRINT@).

Printing Numbers:

All numbers are printed with a leading and trailing space.

Example: PRINT "TO YOU";987,71;321

This will give the following output:-

```
TO YOU987        71321
```

The leading space is reserved for the sign of the number (+ or -) which is only shown when the number is negative. Both spaces may be removed, if desired, by use of the IOM command.

Numeric printout may be specially formatted by use of the FMT command.

Related Keywords: FMT IOM PRINT@ PRINT# SCREEN SPC TAB WIDTH ZONE

PRINT@

PRINT@

Syntax: PRINT@ x,y;E

Where x and y are the co-ordinates of the first character position to be used and E is the expression to be output. x and y can have values in the range 0 to 255.

Purpose: This command allows printing of expressions to commence from a specified point on the screen by use of the co-ordinates x and y. The co-ordinates must each be separated by a comma and there must be either a comma or semi-colon between the expression E and the co-ordinates (in this instance the commas and semi-colons are not executed as print separators).

The screen is divided into rows and columns with the origin, 0,0 at the top left. The number of rows and columns varies with the screen mode.

Screen Mode	Text Display		Display Mode
	Rows	Columns	
0	24	32	Graphics 2
	1	24	40 Graphics 2
2	24	80	Text 2
3	24	32	Graphics 6
4	24	40	Graphics 6
5	24	64	Graphics 6
6	24	80	Graphics 6

For example, in screen mode 2, the screen is 'divided' into a grid of 80 columns across by 24 rows down.

NOTE:

If either of the co-ordinates are greater than the maximum number of columns or rows then a "wrap around" will occur. Thus in screen mode 1 PRINT@ 50,38 will cause the cursor will move to 10,14.

Related Keywords: FMT IOM PRINT PRINT# SCREEN SPC TAB WIDTH ZONE

PRINT#

PRINT#

Syntax: PRINT# J

J is a "device number " previously assigned to a device and in the range 0 to 254, although only devices 0, 1 and 2 are defined in EBASIC.

Purpose: This statement assigns a new output device (eg. Printer etc.) indicated by the value of J.

All output from statements such as PRINT and LIST will be directed to the new device selected by N until another PRINT# statement is encountered to change the device selection, the program ends or the program is aborted either by an error, or from the keyboard.

In direct mode each line acts like a small program therefore if PRINT# is used, the corresponding output statement must appear in the same line. After execution in direct mode the keyboard and display (input device 0 and output device 0) are automatically selected. The following three devices are currently assigned within EBASIC.

DEVICE	DEVICE NUMBER
VDU (SCREEN) -	0
PRINTER -	1
SERIAL PORT - (RS232)	2

When a program ends or aborts, either through an error or as directed from the keyboard, the output device reverts back to the display unit screen (i.e.device 0).

Example: PRINT# 1

All output following this statement in a program will be directed to the Printer.

PRINT # can be used in the same manner as a normal PRINT statement but the "device number" must be followed by a semi-colon (;) so as to distinguish the remainder of the statement.

Example: PRINT#1; "ANSWER THE FOLLOWING QUESTIONS"

Example:

```
10 PRINT#1
20 LIST
```

This will list a whole program to the printer device. When the listing of the program is complete "ready" is printed on the display.

Example: PRINT#2

All output following this statement in a program will be directed to the serial port (RS232).

Example: PRINT#1:LIST

When used in direct mode also lists to the printer. When listing is complete an internal PRINT#0 is performed, switching the output back to the screen.

Use With Files:

Syntax: PRINT# SV,R;E

Purpose: This command is used to output the expression list given by E, to the file given by the file-descriptor SV, from the start of the record number given by R in the file.

The location relative to the start of the file is calculated as R multiplied by the record length given when the file was opened. (This only applies to random access and is not allowed in sequential access). For "sequential access", omit the (,R) but keep the (;) giving the following format.

PRINT# SV;E

Output will then start from the current place in that file since the BASIC keeps account of its place in a particular file even when several files at once may be open for output. (In fact the only purpose of specifying the record number R is to define the point within the file at which input or output is to begin, therefore it will be assumed that it "starts from where it left off" if no record number is given).

With disc files opened for sequential access, the internal file pointer can be set to the start of the file by specifying a record number (any number will do, since it will be multiplied by the ZERO record length).

The expression list given by E is as for a normal PRINT statement and the data output will be EXACTLY as for that. Hence a carriage-return/line-feed is output at the end of the statement unless terminated by the semi colon.

All subsequent statements supplying output, following ths command, will now go to a file until another PRINT# or CLOSE statement is encountered.

PRINT#SV,E (no semi-colon) and PRINT#SV can be used to set up the specified file for output. All subsequent normal output statements will then direct data to the file (e.g. PRINT,LIST etc).

If PRINT statements are then terminated with a semi-colon (;) there will be no carriage return/line feeds, and a stream of data may be output to a file.

The automatic tab expansion (where CHR$(9) is expanded to spaces) may need suspending by use of the IOM7,0 command.

This now facilitates the output of strings which contain machine-code.

Related Keywords: LIST LISTP PRINT

PSG

PSG (Programmable Sound Generator)

Syntax: PSG R,J

Purpose: This is a Sound Generator command which allows direct access to the sound generator "registers".

R is the register number and has a value from 0 to 15.

J is the register value in the range 0 to 255.

This command can be used to create particular "sound effects" as described in a later section which is devoted to the details of the Programmable Sound Generator.

Example:

```
PSG 12,120
```

This will store 120 in register 12 of the sound generator.

PSG can be used as a function to obtain the current value of a specified register.

Example:

```
X = PSG(8)
```

This will return the value of the channel A amplitude register in X.

Related Keywords: MUSIC TEMPO VOICE

PSW

PSW (Password)

Syntax: PSW password

Where password is an 8 character name enclosed in quotes, selected by the user and may contain any characters other than control characters.

Purpose: This command sets up the password protection facility which can be used for security purposes to limit the access to any given file to authorised personnel only.

Once a password has been invoked any files saved can then only be loaded back under the same password. Any files which exist on the disc either without a password, or under a different password, cannot be loaded whilst the current password is in operation.

To change the password use PSW again with a different password. To turn off the password (or make sure that no password is in force!) use PSW by itself (i.e. PSW and no password).

NOTES:

1. An unprotected file (i.e. a file saved with no password invoked) must be read back without a password being in force.

2. The password itself is not stored anywhere on the disc, therefore the user must know it or record it elsewhere.

3. There is no indication given in the directory that a file has been protected. The file can apparently be read, but appears to be complete rubbish.

4. The directory itself is unaffected by the password so that it is perfectly acceptable to mix unprotected files and files saved under various passwords stored on the same drive (as long as you know which are which!).

Example:

```
PSW"IXZ247Y5"
SAVE"MAIL.XBS"
```

Having saved the file MAIL.XBS under the password IXZ347Y5 it can only be read or loaded back if that password is in force. Likewise other files not saved under this password cannot be read or loaded while it is in force.

Related Keywords:

PTR

PTR (Pointer)

Syntax: PTR J,I

J is a number in the range 0 to 24.

Purpose: This allows the user to set selected scratch pad locations without using PEEK or POKE, but using the number J to select the locations, and I to be the new value. J is in the range 0 to 24.

Location numbers, J, are selected from the list given below.

0	HTEXT	Default or 'hard' pointer to start of BASIC program
1	TEXT	Pointer to start of BASIC program (modified by HOLD)
2	SCMD	Pointer to standard reserved word table.
3	AUXCMD	Pointer to auxiliary (user) reserved word table.
4	ERRTAB	Pointer to normal error message table.
5	AUXERR	Pointer to auxiliary error message table.
6	SADR	Pointer to standard address table.
7	SFNADR	Pointer to standard function table.
8	AUXADR	Pointer to auxiliary address table.
9	USRLOC	Pointer to user machine-code routine (CALL as function).
10	DEVPTR	Pointer to list of available I/O devices.
11	DEFLST	No of lines to 'LIST' at a time.
12	BUFPTR	Pointer to start of input buffer.
13	BUFLEN	Length of input buffer.
14	TXTTOP	Pointer to end of BASIC program.
15	VARTOP	Pointer to end of simple variable space.
16	ARRTOP	Pointer to end of array space.
17	STRBOT	Pointer to bottom of string space.
18	STKBOT	Pointer to bottom of stack area.
19	IVDU	Pointer to bottom of 'internal VDU' area.
20	LIMIT	Pointer to top of RAM used by Tatung/Xtal BASIC.
21	TOPRAM	Pointer to top byte of RAM available to user.
22	LNNO	Current line number being executed.
23	DATLN	Line number of current DATA statement (undefined before a READ statement has been done).
24	DATPTR	Pointer to current position in DATA statement (If using READ statements). Can be moved to specified line by RESTORE L statement

Any value for J outside the range listed above will give a RANGE ERROR.

The current value of any PTR location may be accessed by using PTR as a function with the argument representing the required location as follows.

Examples:

```
A= PTR(12)
```

This puts the start address of the current input buffer area into the variable A.

```
PRINT HEX$(PTR(12))
```

This will display the current address value for the input buffer area (in hexadecimal format).

NOTE: Great caution should be adopted when using this command as there are no facilites for checking that alterations are not affecting other locations which might already be in those areas of memory. For example, the CLEAR command should be used to set up the LIMIT and STKBOT locations, not PTR 20,E and PTR18,E.

Related Keywords: DEEK DOKE PEEK POKE

RAD (Radians)

Syntax: RAD(N)

Purpose: This is a Function which converts the angle given by N (in degrees) to radians.

Examples:

```
X = RAD(30)     - Returns a value of .523599 radians
                  in X.
PRINT RAD(30) - This will display the value .523599
                  on the screen.

X = SIN(RAD(30)) - Returns the sine of 30° in X (i.e. 0.5)
```

Related Keywords: ATN COS DEG SIN TAN

READ

READ

Syntax: READ V1,V2,..,Vn

V1,V2,etc. are variables which are linked up to corresponding values in the same order as listed in a DATA statement.

Purpose: This statement is used to access data, stored in DATA statements, from within a program as opposed to input from the keyboard. BASIC positions a "pointer" at the last item of data read so that subsequent READ statements will continue from that point.

If there is insufficient data available for the READ statement a DATA ERROR will occur.

Example:

```
10 READ A,B,C
20 - - - - -┐
30 - - - - -├──Processing lines.
40 - - - - -┘
50 DATA 9,20,30
```

This will READ a value of 9 for A, 20 for B, and 30 for C.

Example:

```
10 READ A,B,C
20 - - - - -
30 - - - - -
40 READ D,E,F
50 - - - - -
60 - - - - -
70 DATA 30,6,19,20,64,71
```

The first READ statement in line 0 will read a value of 30 for A, 6 for B, and 19 for C. The pointer is then positioned at 19 in the DATA statement so that the second READ statement, in line 40, will start from that point.

```
30,6,19,20,64,71
      ↑
      Pointer
```

Thus the second READ statement will read a value of 20, for D, 64 for E, and 71 for F.

Related Keywords: DATA RESTORE

REM (Remark)

Syntax: REM

Purpose: This causes the remainder of the line to be ignored by the Interpreter (i.e. it is not processed).

It is used for entering notes anywhere in a program to clarify the purpose of particular sections and their functions.

Example:

```
10 REM PROGRAM TEST FOR COLOURS
```

This line is not processed and is merely a comment line as an aid to the programmer.

NOTE: No further BASIC statements can be entered on the same line after a REM statement.

```
eg. 100 REM PROGRAM ENDING:STOP
```

Here the stop will not be executed.

Related Keywords:

REN (Rename)

Syntax: REN "old <ufn>" TO "new <ufn>"

Purpose: This is a Disc Command which renames the file given by old file to the name given by new file for the disc in the current default drive.

If new <ufn> is already in use on the disc a FILE EXISTS error will occur.

If old <ufn> does not exist a NO FILE error will occur.

If old <ufn> is a locked file a FILE LOCKED error will occur.

Example:

```
REN "ROUTINES" TO "PROCESS"
```

This will change the name ROUTINES to PROCESS for that particular BASIC file.

```
REN "PARTY.ASC" TO "GROUP.ASC"
```

This will rename the .ASC file PARTY to become GROUP.ASC.

Related Keywords: DRIVE DIR

RENUM (Renumber)

Syntax: REN L1,L2

L1 is the new starting line and L2 is the increment to be used. If omitted, both L1 and L2 will default to 10.

Purpose: This is a System Command used to renumber a whole program or a "held" part of a program.

All line number references following GOTO, GOSUB, RUN, THEN, ELSE, AND RESTORE commands are modified accordingly during the renumbering process.

Examples:

RENUM 1000,5 Will renumber, making the first line 1000 and then increment by 5 (1000,1005,1010,1015 etc).

RENUM Will renumber making the first line 10 and then increment by 10 (10,20,30,40 etc).

RENUM 500 Will make the first line 500, and then increment by 10 (500,510,520,530 etc).

RENUM, 20 Will make the first line 10 and then increment by 20 (10,30,50,70, etc).

Related Keywords: LIST HOLD MGE

RESTORE

Syntax: RESTORE L

Where L is given as a line number.

Purpose: This statement positions (restores) the internal pointer, used by BASIC in DATA statements, to the beginning of the first DATA statement following the line number L, regardless of where the pointer had been left by previous READ statements.

This facility allows DATA statements to be re-read several times within the same program thus avoiding having to store the "data items" in variables throughout the whole execution of a program.

L (line number) is optional and, if omitted, the pointer is restored to the beginning of the very first DATA statement in the program.

Example:

```
 10 READ A,B,C
 20 - - - - -
 30 - - - - -
 40 DATA 10,20,40,70,90,110,15,17,150
 50 READ X,Y,Z
 60 - - - - -
 70 - - - - -
 80 RESTORE 30
 90 READ D,E,F
100 - - - - -
110 - - - - -
```

The sequence of operations would be as follows:-

i) The READ statement in line 10 will give A a value of 10, B a value of 20, and C a value of 40.

ii) The internal pointer is then positioned at 40 in the DATA statement.

```
10,20,40,70,90,110,15,17,150
      ↑
      Pointer
```

iii) The READ statement in line 50 will continue from the pointer and give X a value of 70, Y a value of 90, and Z a value of 110.

iv) The internal pointer is then repositoned to 110.

```
10,20,40,70,90 110,15,17,150
               ↑
               Pointer
```

v) The RESTORE statement in line 60 causes the pointer to move to the beginning of the DATA statement in line 30.

```
10,20,40,70,90,110,15,17,150
↑
Pointer
```

vi) The read statement in line 90 therefore gives D a value of 10, E a value of 20, F a value of 40.

vii) The pointer is then positioned once again at 40.

```
10,20,40,70,90,110,15,17,150.
      ↑
      Pointer
```

Related Keywords: READ DATA

RETURN

Syntax: RETURN

The last line of any subroutine should always be RETURN.

Purpose: This terminates a subroutine accessed by a GOSUB statement.

Execution is transferred back to the line immediately following the original GOSUB statement.

If a RETURN is encountered without having been preceded by a GOSUB then a RETURN ERROR will occur.

Related Keywords: GOSUB

RIGHT$

Syntax: RIGHT$(<string expression>,J)

Purpose: This is a String Function which will return the rightmost number of characters, given by the value of J, of the string specified in the function.

Example:

```
 PRINT RIGHT$("HELLO",2)
```

This will display the following result on the screen:-

```
        LO
```

Related Keywords: LEN MID$ LEFT$

RND (Random)

Syntax: RND(I)

Purpose: This is a Function which returns a random number.

When I=1 the function returns a random number in the range 0 to 1, as a floating point number.

When I is in the range 2 to 65535 the function returns an integer random number ranging from 0 to (I-1).

When I=0 the functions returns the last random number produced, whether integer or real.

Example:

```
RND(9) - returns a number in the range 0 to 8
RND(307) - returns a number in the range 0 to 306.
```

Example:

```
PRINT RND(11)
```

The random number selected from the range 0 to 10 will be displayed on the screen.

Related Keywords:

RST (Restart)

Syntax: RST

Purpose: This command is used to reinitialise some default settings within the machine.

a) Clears the screen and sets screen mode 1.
b) Resets the character generator, to the default character set.
c) Removes any sprites which may be present.
d) Turns off the disc drive motors if they are running.
e) Resets the sound generator.
f) Resets the default auxiliary table pointers.
g) Resets the default WIDTH, SEP and ZONE values.
h) Sets the text and graphics foreground colour to white, background colour to transparent.

The command does not destroy the current BASIC program or variables. It can, therefore, be used within a program where it would otherwise be tedious to use a string of commands to ensure that the correct modes were all set up.

Example:

```
  10 RST
  20 REM START TO PROGRAM
  30 - - - - -
  40 - - - - - -
  etc.
```

Related Keywords: CLS SCREEN

RUN

Syntax: RUN L

Where L is a line number

Purpose: RUN is used to begin the execution of a program currently in memory, starting at the line number given by L, and clearing all variables.

If L is omitted execution will begin from the lowest line number of the program.

Example:

```
 RUN 45
```

This will cause a program to begin execution at line 45.

Syntax: RUN file

Purpose: In this alternative form RUN will load the program file declared in file and then commence its execution from the lowest line number.

Example:

```
 RUN "TESTPROG"
```

This will load the program whose name is TESTPROG.XBS from disc and then execute it.

Related Keywords: CHAIN LOAD

SAVE

SAVE

Syntax: SAVE "<ufn>",N1,N2

Purpose: This Command saves files/programs, currently in the computer's memory, onto disc. When saving BASIC programs, or complete ASCII (.ASC) files, N1 and N2 may be omitted, For object code (.OBJ) files,N1 and N2 must be declared.

N1 and N2 represent the start and end line of an ASCII file, or the start and finish address of object code files repectively.

When saving files as ASCII files, the file extension .ASC must be used. When saving files as object code files, the file extension .OBJ must be used.

If the drive is omitted from within the file the current "default drive" will be assumed. The default drive is initially set up as 0 but can be changed using the DRIVE command.

If the file type is not declared within <ufn> then XBS will be assumed.

Example: SAVE "1:PROG.XBS"

This will save the BASIC Program file currently in memory onto the disc in drive 1, giving it the name PROG. (assuming drive 1 is fitted to EINSTEIN 256).

Example: SAVE"PROG"

This will save the program onto the disc in the current default drive, as a XBS type file. It will over-write any file of that name on the disc.

Example: SAVE"0:PROG.ASC"

This will save the whole of the program currently in memory onto the disc in drive 0, as an ASCII file PROG.ASC.

Example: SAVE"1:PROG.ASC",90,150

This will save the section of the program currently in memory from line 90 to line 150 onto the disc in drive 1, as an ASCII file PROG.ASC.

Example: SAVE"1:TEST.OBJ",&8A3D,&9000

This will save the area of memory from location &8A3D to location &9000 onto the disc in drive 1 as the object file TEST.OBJ.

NOTE: Remember to reserve an error of memory, for your object file, using the CLEAR command.

Related Keywords: DRIVE LOAD

SCREEN

SCREEN

Syntax: SCREEN N

Purpose: This command selects graphics mode 2 or 6, or text mode 2, in screen widths of 32,40,64 or 80 columns in text, or 256 and 512 pixels in graphics. N can have values 0 to 6, according to the table below:

SCREEN	MODE	TEXT	RESOLUTION (PIXELS)
0	Graphics 2	32 x 24	256 x 192
1	Graphics 2	40 x 24	256 x 192
2	Text 2	80 x 24	no graphics in this mode
3	Graphics 6	32 x 24	412 x 192
4	Graphics 6	40 x 24	512 x 192
5	Graphics 6	64 x 24	512 x 192
6	Graphics 6	80 x 24	512 x 192

In graphics mode 6, the background parameters in the GCOL commands have no meaning, and any pixels can be any colour, i.e. screen attributes in colour are bit mapped.

In graphics mode 2, colour is selected on a cell basis, there being 32 cells, each 8 pixels wide, Within each cell, there can only be one foreground and one background colour. This mode is Einstein compatible.

In text mode 2 only one foreground colour is allowed for the whole screen.

The SCREEN command automatically clears the screen and clears sprites when invoked. A form feed character (OC$_H$) is sent to the selected output device.

When changing between screen modes, sprites are also cleared.

Related keywords: GCOL, TCOL, CLS32, CLS40

SCRN$ (Screen String)

Syntax: SCRN$(J)

J must be a value in the range 0 to 23 (i.e. number of rows available on the screen).

Purpose: This is a String Function which will return the full string of characters from a row on the screen, indicated by the value of J.

Examples:

```
X$ = SCRN$(11)
```

This will return the string of characters contained in row 11 of the display screen in X$.

```
PRINT SCRN$(11)
```

This will output the contents of screen row 11.

Related Keywords: LEFT$ MID$ RIGHT$

SEP (Separator)

Syntax: SEP J

Where J is given as an ASCII value.

Purpose: This command is used to re-define the separator character used in DATA and INPUT statements, J being the ASCII value of the required character.

Under normal operation the separator is a comma (,) but this can be changed by use of the SEP command.

One common application is to use SEP 0. This is used when only one item is required which is to include commas as part of the input data. It allows the user to put any string of characters (including the comma) into an INPUT or DATA statement as a single item.

On conclusion of the processing involving the redefined separator character, normal operation can be restored by use of the SEP 44 command (44 being the ASCII value for the comma).

SEP can be used as a function to return the value of the current separator.

NOTE:

1) Some characters will not work well as separators with numeric data, the full stop (.) for example. Obvious confusion could occur here with decimal points.

2) Remember that SEP will affect DATA statements as well as INPUT statments.

Examples:

```
SEP 43
```

This would change the separator to a + symbol as given by the ASCII value 43.

```
PRINT SEP
```

This will output the ASCII value of the current separator on the screen.

```
A = SEP
```

Thus A will contain the ASCII value of the current separator character.

The following example illustrates another use of SEP.

```
10 SEP 47:REM '/' IS SEPARATOR
20 INPUT "TYPE IN THE DATE AS DD/MM/YY:";.DAY,MNTH,YEAR
30 PRINT "DAY IS  ",DAY,"MONTH IS  ";MNTH;"YEAR IS  ";
   YEAR
40 END
```

This program will display the following.

```
TYPE IN THE DATE AS DD/MM/YY:
```

The date is then typed as - 14/12/84

As a result of the SEP command the slash symbol (/) is accepted in place of the comma separator given in the corresponding part of the INPUT statement. Program execution then continues and displays the following.

```
DAY IS  14 MONTH IS  12 YEAR IS  84
```

Related Keywords: INPUT DATA

SGN (Sign)

Syntax: SGN(N)

Where N can be given either as a number or a numeric expression

Purpose: This is a Function which returns the sign of N (i.e. indicates whether N is a positive or negative number).

The following values are returned according to the given conditions.

N less than 0 a value of -1 is returned.
N equal to 0 a value of 0 is returned
N greater than 0 a value of +1 is returned.

Examples:

```
PRINT SGN(-5.721) - a value of -1 appears on screen
                    (i.e. a negative number).

PRINT SGN(7.6219) - a value of 1 appears on screen
                    (i.e. a positive number).

PRINT SGN(0)      - a value of 0 appears on screen
                    (i.e. a zero value).
```

Related Keywords: ABS INT

SHAPE

Syntax: SHAPE N,<string expression>

N is the ASCII code nominated for a character.

<String expression> is the required data and must consist of 2 digit hexadecimal numbers contained within quote marks ("").

Purpose: This command allows the user to define a character shape or re-define an existing character.

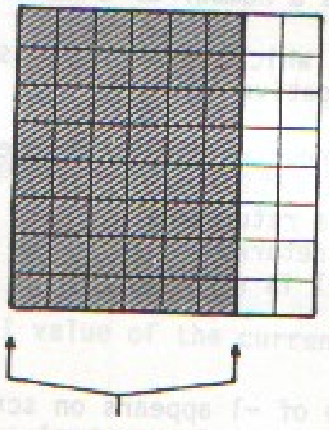
ASCII codes 32 to 127 and 161 to 255 are used for the keyboard text and graphic characters. Unless there is a need to redefine any of the characters avoid using the codes in these ranges.

The following character codes do not have any shape programmed at power-up.

130 to 134, 142 to 154, 156 to 160

A shape consists of 8 bytes, each byte representing one row of the character cell. The most significant bit of each byte is the leftmost pixel of each row.

If the shape programmed is to be displayed in 40 column display, only the 6 most significant bits will be displayed on the screen. When in 32 column display or when defining a sprite shape all 8 bits are displayed.

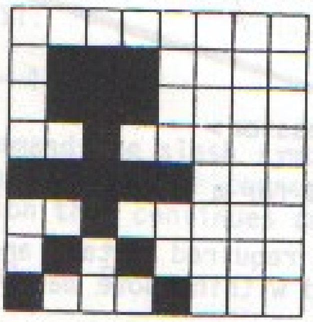


6 Most Significant BITS of each BYTE

displayed in 40 column display

Two or more shapes can be defined from within a single command by adding 8 bytes for each new shape. Each block of eight bytes will be assigned to the next ASCII character code in sequence.

Example: The following "little man" shape is defined within one character cell.



The HEX values for each byte are placed into the shape command (in order from the top downwards), and an ASCII code is selected (eg. 130). The command would then appear as follows:-

```
SHAPE 130, "00 70 70 20 F8 20 50 88"
```

The shape can be called onto the screen as follows:-

```
   PRINT CHR$(130)
```

The shape can also be used with the SPRITE command.

If ASCII code 160 is used then the shape can be accessed from the keyboard by pressing the GRAPH key and the SPACE bar simultaneously.

Example:

```
  SHAPE 160,"00 70 70 20 F8 20 50 88 00 88 50 20 F8 20 70 70"
```

The first 8 bytes define character code 160
The second 8 bytes define character code 161

Related Keywords: SPRITE MAG

SIN (Sine)

Syntax: SIN(N)

Where N is an angle given in radians.

Purpose: This is a Function which returns the SINE of N.

Examples:

```
  A = SIN(0.523599)   -   For an angle of 30°
```

This returns a value of 0.5 in A

The values are returned for use in expressions but can be output by using the PRINT command.

```
  PRINT SIN(1.0472)   -   For an angle of 60°
```

The value 0.866027 will appear on the screen.

Related Keywords: ATN COS DEG TAN EXP

SIZE

Syntax: SIZE

Purpose: This is a Function which returns the size of memory available for the program, variables, pointers, and strings, as a positive value in the range 0 to maximum size of the system.

NOTE: Also used to clear space prior to a HOLD and MERGE.

Example: X = SIZE

This will return a value in X indicating the size of memory available.

Related Keywords:

SPC (Space)

Syntax: SPC(J)

Purpose: This is a function which prints J spaces. It is only valid within a PRINT statement. It is different to TAB, in that TAB is absolute (i.e. works on column numbers), whereas SPC is relative.

Example:

```
10 PRINT "ONE";SPC(10);"TWO"
```

which will produce:-

```
ONE◄————►TWO
```

with 10 spaces between the 'E' and the 'T'

Related Keywords: PRINT TAB

SPEED

SPEED

Syntax: SPEED J

J can be any integer value from 0 to 255.

Purpose: This is a special command relating to output. It sets a delay on character output, to the current output device, according to the value of J.

0 gives the longest delay i.e. slowest speed. 255 is the fastest speed. (BASIC defaults to 255)

SPEED can also be used as a function to return the current set speed and the value given can be stored as a variable.

Examples: SPEED 100

All output following this statement in a program will be slower than normal (normal being 255).

```
PRINT SPEED
```

This will display the current value (as a number from 0-255) of speed set, onto the screen.

```
SPEED = 200
A = SPEED
```

Thus A will contain 200.

Related Keyword: NULL

SPRITE

Syntax: SPRITE S,x,y,C,N

Purpose: This is a Graphics Command which sets up a particular sprite.

S is the "sprite number" and can be in the range 0 to 31. This allocates each sprite a priority (0 being the highest priority), this determines which shape is masked when two sprites have the same screen position.

x and y are the co-ordinates which position the top left hand corner of the sprite on the screen. The range of values for x and y is -32768 to +32767. C is a value in the range 0 to 15 indicating the foreground colour as defined by the palette command (C is optional and if omitted it will default to the last previously specified value of foreground colour). The background colour is always zero, i.e. transparent.

N is the ASCII code number chosen to represent the sprite. ASCII codes 0 to 127 should be avoided, where possible, as these codes represent alphanumeric characters, control codes and punctuation marks.

In screen modes 0 and 1, there are 4 active sprites per horizontal row, whereas in screen modes 3 to 6, there are 8.

Screen mode 2 is a text only mode, and sprites are not displayed.

Related Keywords: SHAPE MAG SPRITE OFF

SPRITE OFF

Syntax: SPRITE OFF S

Where S is given as a sprite number in the range 0 to 31.

Purpose: This is a graphics Command which will "turn off" the sprite given by S (sprite number)

If S is omitted then all sprites will be "turned off".

Related Keywords: RST SPRITE SHAPE MAG

SQR (Square Root)

Syntax: SQR(N)

Purpose: This is a Function which returns the square root value of N for use in expressions.

If N is given as being less than 0 a QTY ERROR will occur.

Examples:

```
X = SQR(22)
This returns a value of 4.69042 in X

PRINT SQR(25)
The result 5 will appear on the screen.
```

Related Keywords: EXP LOG LN

STEP

Syntax: FOR V = N1 TO N2 STEP N3

Purpose: This command is used in FOR-NEXT loops to specify a particular increment within the loop.

Example:

```
FOR I = 1 TO 10 STEP 2
```

Related Keywords: FOR TO NEXT

STOP

Syntax: STOP

Purpose: This is similar to END but is used to terminate programs at various points from which they may be restarted again.

The message BREAK IN L is displayed, where L is the line number at which execution has stopped.

Program execution can be restarted using the CONT command, provided that no alterations have been made to the program during the break (variables may, however, have their values altered).

This command is useful when debugging BASIC programs as it allows sections of the program to be executed and intermediate results inspected.

Related Keywords: CONT END

STR$ (Str̲ing s̲tring)

Syntax: STR$(N)

Where N can be given as a numeric variable or numeric expression.

Purpose: This is a String Function which returns a string representation of the value given by N.

The format in which the number is given by this function can be manipulated using the FMT command and also the IOM 5 command.

Leading spaces are maintained by this function but NOT trailing spaces, in respect of the numerical format output as a string.

Example:

```
A$ = STR$ (1.234)
```

This will give the string "1.234" in A$

```
PRINT STR$ (1.234)
```

The string " 1.234" will be displayed on the screen.

```
FMT 2,3:A$ = STR$ (37.7325)
```

This places the string " 37.733" into A$ as a result of the combination of FMT and the STR$ function,
thus A$ = "37.733".

Related Keywords: ASC FMT LEFT$ LEN MID$ MUL$ RIGHT$ SCRN$ VAL

STOP

Syntax: STOP

Purpose: This is similar to END but is used to terminate programs at various points from which they may be restarted again.

The message BREAK IN L is displayed, where L is the line number at which execution has stopped.

Program execution can be restarted using the CONT command, provided that no alterations have been made to the program during the break (variables may, however, have their values altered).

This command is useful when debugging BASIC programs as it allows sections of the program to be executed and intermediate results inspected.

Related Keywords: CONT END

STR$ (<u>Str</u>ing <u>s</u>tring)

Syntax: STR$(N)

Where N can be given as a numeric variable or numeric expression.

Purpose: This is a String Function which returns a string representation of the value given by N.

The format in which the number is given by this function can be manipulated using the FMT command and also the IOM 5 command.

Leading spaces are maintained by this function but NOT trailing spaces, in respect of the numerical format output as a string.

Example:

```
A$ = STR$ (1.234)
```

This will give the string "1.234" in A$

```
PRINT STR$ (1.234)
```

The string " 1.234" will be displayed on the screen.

```
FMT 2,3:A$ = STR$ (37.7325)
```

This places the string " 37.733" into A$ as a result of the combination of FMT and the STR$ function,
thus A$ = "37.733".

Related Keywords: ASC FMT LEFT$ LEN MID$ MUL$ RIGHT$ SCRN$ VAL

SWAP

Syntax: SWAP V1,V2

V1 and V2 may be numeric or string variables, or array elements. They must be of a similar type in any one statement otherwise a TYPE ERROR will occur.

Purpose: This statement "swaps" the contents of the variables V1 and V2 with each other.

The command is very useful in "sorting" algorithms.

Example:

```
  SWAP A,C        - the contents of A become the contents of C and
                    vice-versa.

  SWAP D$,E$      - the contents of D$ become the contents of E$ and
                    vice-versa.

  SWAP A(I),B(I) -  the array element of A(I) becomes the array element of
                    B(I) and vice-versa.
```

Related Keywords:

TAB (Tabulate)

Syntax: TAB(J1,J2)

Purpose: Prints characters (the ASCII value of) J2, to an 'imaginary' print head on the output device until the cursor reaches a position J1 columns from the left of the "page", or Column 0. If the print head is past or at the point J1, then no tab will occur.

If J2 is omitted, either a previously defined character will be printed, or, (where not previously defined), a space. Any valid ASCII code maybe used for J2. Uses include date on printed forms.

Example:

```
 10 PRINT "NAME";TAB(10,46); "TATUNG"
    (ASCII 46 is a full stop code,
    space is 32)
 20 PRINT "ADDRESS";TAB(10); "TELFORD";
    TAB(20); "SALOP"
    (J2 defaults to code 46)

    This should give:-
    NAME..............TATUNG
    ADDRESS...........TELFORD...SALOP
```

Related Keywords: PRINT SPC SEP ZONE WIDTH

SWAP

Syntax: SWAP V1,V2

V1 and V2 may be numeric or string variables, or array elements. They must be of a similar type in any one statement otherwise a TYPE ERROR will occur.

Purpose: This statement "swaps" the contents of the variables V1 and V2 with each other.

The command is very useful in "sorting" algorithms.

Example:

```
  SWAP A,C        - the contents of A become the contents of C and
                    vice-versa.

  SWAP D$,E$      - the contents of D$ become the contents of E$ and
                    vice-versa.

  SWAP A(I),B(I) - the array element of A(I) becomes the array element of
                    B(I) and vice-versa.
```

Related Keywords:

TAB (Tabulate)

Syntax: TAB(J1,J2)

Purpose: Prints characters (the ASCII value of) J2, to an 'imaginary' print head on the output device until the cursor reaches a position J1 columns from the left of the "page", or Column 0. If the print head is past or at the point J1, then no tab will occur.

If J2 is omitted, either a previously defined character will be printed, or, (where not previously defined), a space. Any valid ASCII code maybe used for J2. Uses include date on printed forms.

Example:

```
  10 PRINT "NAME";TAB(10,46); "TATUNG"
     (ASCII 46 is a full stop code,
     space is 32)
  20 PRINT "ADDRESS";TAB(10); "TELFORD";
     TAB(20); "SALOP"
     (J2 defaults to code 46)

     This should give:-
     NAME..............TATUNG
     ADDRESS...........TELFORD...SALOP
```

Related Keywords: PRINT SPC SEP ZONE WIDTH

Example:

```
TCOL 9,12
```

Any text produced following this command will appear as Light Red (foreground) characters on a Dark Green background (in respect of individual cells).

NOTE: When a background colour other than 0 (transparent) is specified, a CLS will fill the display area with the new background colour.

In screen mode 2, N2 has no meaning as there is no background colour in screen mode 2. It can be specified, for convenience, but will have no effect on the display. All characters have the same foreground colour. Changing TCOL changes the foreground colour of the entire screen.

Related Keywords: BCOL GCOL PALETTE SCREEN

TEMPO

Syntax: TEMPO N

Where N is given as a number in the range 0 to 7.

Purpose: This command sets the tempo (speed) of the music output according to the value given by N. (If TEMPO is not specified a value of 4, see table, is assumed).

Each value of N represents a tempo as listed below:-

0 - 50 beats per minute
1 - 100 beats per minute
2 - 150 beats per minute
3 - 200 beats per minute
4 - 250 beats per minute
5 - 300 beats per minute
6 - 350 beats per minute
7 - 400 beats per minute

Example: TEMPO 3

This will set a tempo of 200 beats per minute for the music output.

Related Keywords: BEEP MUSIC PSG VOICE

THEN

Syntax: IF <condition> THEN <statement>

Purpose: This is used in the IF statement to direct the resulting sequence of operation.

Example:

```
  IF X = 10 THEN 120
```

Related Keywords: IF ELSE GOTO GOSUB

TI$ (Time String)

Syntax: TI$="HHMMSS"

Purpose: This command is used to set the real-time 24 hour clock contained within the system to a particular value given by HH as hours, MM as minutes, and SS as seconds.

On power up the clock is set to "00000" but once set by the TI$ command it will continue to keep the time until the machine is switched off or reset.

The time may only be set to an even number of seconds, although both odd and even numbered seconds are displayed.

The current time value can be returned using TI$ as a function.

Examples:
```
TI$ = "174032"
```

This will set the clock to a time of 17 hours, 40 minutes, 32 seconds.

```
PRINT TI$
```

This will display the current time setting in the following format:-

```
HHMMSS
```

Related Keywords:

TO

Syntax: FOR V=N1 TO N2

Purpose: This is used within the FOR statement in order to specify the upper limit for a required loop.

Example: FOR x = 3 TO 19

Related Keywords: FOR NEXT STEP

UNLOCK

Syntax: UNLOCK "<ufn>"

Purpose: This is a Disc Command which unlocks a previously locked file (given by <file>) on the disc in the current default drive.

Files unlocked using this command may then be written to, erased, or renamed as required.

If the <file> does not exist a NO FILE error occurs.

Related Keywords: DRIVE DIR LOCK

UNPLOT

Syntax: UNPLOT x,y

Where x and y are the co-ordinates of a point on the screen and can have values in the range of -32768 to +32767(the screen grid is 256 pixels horizontal by 192 pixels vertical).

Purpose: This is a Graphics Command which turns off a pixel which is illuminated i.e. the pixel at the co-ordinates specified is changed from foreground to background. If the pixel is already in background, then the pixel will remain unchanged.

Example:

```
UNPLOT 120,90
```

This will turn off the pixel at co-ordinates 120,90.

NOTE: In screen modes 2 to 6 inclusive, UNPLOT has no meaning. It's use in these modes will not produce an error, but will have no effect on the screen. It is used in screen modes 0 and 1 only.

Related Keywords: DRAW ELLIPSE PLOT POINT POLY

VAL (Value)

Syntax: VAL(<string expression>)

Purpose: This is a String Function which returns the numerical value of the specified string up to the first non-numeric character.

("+","-",".", and "E" are regarded as numeric).

The "&" character is taken to indicate that a "hex number " will follow.

Examples:

```
VAL("1.234ABC")
```

This returns the value 1.234

```
PRINT VAL("1.7993XYZ")
```

The result will appear on the screen as 1.7993.

```
VAL("&" + "ABCD")
```

This gives a value of 43981 (i.e. decimal equivalent of &ABCD)

Related Keywords: ASC EVAL STR$

VDEEK (Video Deek)

Syntax: VDEEK(I,J)

Where I represents a memory location, in 64K pages and J represents the page number, 0 to 2.

Purpose: This is a Machine Code related command which operates in a similar manner to DEEK with the following differences:

1) The memory operations take place on the VIDEO RAM.

2) The video memory location given in I must be in the range 0 to 65535 (0 to &FFFF).

Related Keywords: DEEK DOKE PEEK POKE VDOKE VPEEK VPOKE

VDOKE (Video Doke)

Syntax: VDOKE (I,I1,I2,...,In)

Purpose: This is a Machine Code related command which operates in a similar manner to DOKE with the following differences.

1) The memory operations take place on the VIDEO RAM.

2) The memory location (I) must have values in the range 0 to 65535 (0 to &FFFF).

Related Keywords: DEEK DOKE PEEK POKE VDEEK VPEEK VPOKE

VDP

VDP (Video Display Processor)

Syntax VDPN,d

Purpose: Loads register N of the display processor with the data, d. The command can write to VDP registers 0 to 23, and 32 to 46. Values of N from 24 to 31 will result in a RANGE ERROR. d can have any value in the range 0 to 255. Registers 0 to 7 are Einstein compatible in the range 0 to 255. A summary of the registers is shown below.

The functions of the registers are detailed below. For a more detailed explanation of how to use the display processor, refer to the "V9938 Technical Data Book" published by ASCII corporation/Nippon Gakki Co. Ltd or "Einstein 256's Display Processor Explained" by Syntaxsoft.

Register Functions

Mode Registers (write only)

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Mode Register 0	0	DG	IE2	IE1	M5	M4	M3	0	0
Mode Register 1	0	BL	IE0	M1	M2	0	SI	MAG	1
Mode Register 2	MS	LP	TP	CB	VR	0	SPD	BW	8
Mode Register 3	LN	0	S1	S0	IL	E0	NT	DC	9

R0 DG: Sets the colour bus to input mode, and inputs data into the VRAM.

IE2: Enable interrupt from Lightpen.

IE1: Enables interrupt from Horizontal line - see register 19

M5: Used to select the display mode.

M4: Used to select the display mode. See mode table

M3: Used to select the display mode.

R1 BL: When = 1, screen display enabled. When 0, screen disabled.

IE0: Enables interrupt from Horizontal line - see register 19.

M1: Used to select the display mode.

M2: used to select the display mode. See mode table

SI: When = 1, sets 16 x 16 sprites sixe; when = 0 sets 8 x 8 sprite size.

MA: Sprite expansion. 1 = double size; 0 = normal size.

R8 MS: When = 1, sets the colour bus to input mode and enables mouse.
When = 0, sets the colour bus to output mode and disables mouse.

LP: When = 1, enables light pen. When = 0 disable light pen.

TP: Sets the colour of code - to the colour of the palette.

CB: When = 1, sets the colour bus to input.
When = 0, sets the colour bus to output.

VR: Selects the type of Video RAM. Always set to 1.

SPD: 1 = SPRITE OFF; 0 = SPRITE ON

BW: When = 1, sets black and white (32 grey levels) only affects composite video output
When = 0, sets colour

R9 LN: When = 1, sets vertical resolution to 212 pixels
When = 0, sets vertical resolution to 192 pixels

S1: Selects simultaneous mode.

S0: Selects simultaneous mode.

IL: When = 1, interlace
When = 0, non-interlace

EO: When = 1, displays two graphic screens interchangeably by Even field/Odd field.

NT: When = 1, PAL (313 lines); when = 0, NTSC (262 lines).
For RGB output only).

DC: When = 1, sets DLCLK to input; when = 0, sets DLCLK to output

Table Base Address Registers (Write only)

The table base address registers are a set of registers to declare the addresses of tables in the VRAM.

NOTE: When these registers are accessed, the control codes that the screen may receive depends on the display mode. For this purpose, you must mask the unwanted bits.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Pattern table base register	0	A16	A15	A14	A13	A12	A11	A10	2
Colour table base address register low order bits	A13	A12	A11	A10	A9	A8	A7	A6	3
Colour table base address register high order bits	0	0	0	0	0	A16	A15	A14	10
Pattern generator table base address register	0	0	A16	A15	A14	A13	A12	A11	4

Sprite attribute table base address register low order bits	A14	A13	A12	A11	A10	A9	A8	A7	5
Sprite attribute table base address register high order bits	0	0	0	0	0	0	A16	A15	11
Sprite pattern generator table base address register	0	0	A16	A15	A14	A13	A12	A11	6

Colour Register (Write only)

The colour register is used to control Einstein 256's text and background screen colours, and also blinking.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Text Colour/ Backdrop colour register	TC3	TC2	TC1	TC0	BD3	BD2	BD1	BD0	7

TC3 to TC0: Specifies the text colour according the TEXT 1 and TEXT 2 modes.
BD3 to BD0: Specifies the back drop colour in all display modes.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Text Colour/ Backdrop Colour register	T23	T22	T21	T20	BC3	BC2	BC1	BC0	12

In Text mode 2, if the attributes for blinking are set, the colour set in this register and set in R7 are displayed alternately.

T23 to T20: Text foreground colour text modes only
BC3 to BC0: Text background colour

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Blinking period register	ON3	ON2	ON1	ON0	OF3	OF2	OF1	OF0	13

In the bit map modes (Graphics 4 to Graphics 7), the two pages are displayed alternately (blinked). Place data in this register to set the display page to an odd page to begin blinking. This register is also used in the Text mode 2.

ON3 to ON0: even page display duration.
OF3 to OF0: odd page display duration

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Colour burst register 1	0	0	0	0	0	0	0	0	R20
Colour burst register 2	0	0	1	1	1	0	1	1	R21
Colour burst register 3	0	0	0	0	0	1	0	1	R22

The above values are preset when the power is applied. If all values in the above three registers are set to 0, the colour burst amplitude of the composite video output will be zero.

Display Registers (Write only)

The display registers are used to control the display position on the CRT.

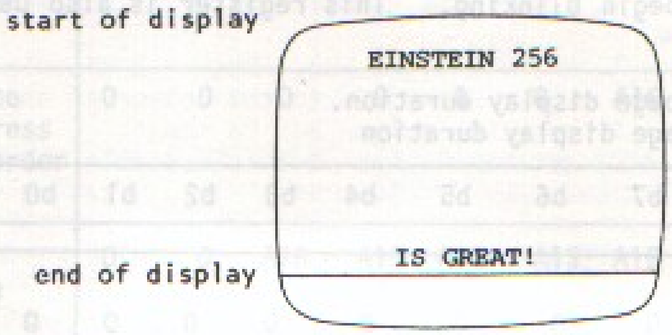
	b7	b6	b5	b4	b3	b2	b1	b0	VDP register number
Display adjust register	V3	V2	V1	V0	H3	H2	H1	H0	R18

The above register is used to adjust the display position on the CRT.

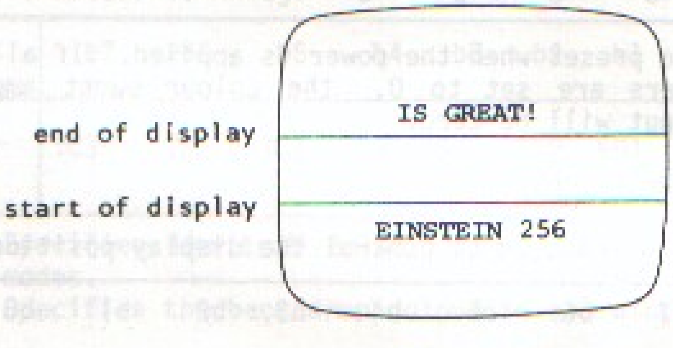
H = horizontal position in 2's complement; 0 = centre, 7 left, 15 right
V = vertical position in 2's complement; 0 = centre, 7 down, 15 right

	b7	b6	b5	b4	b3	b2	b1	b0	VDP register number
Display offset register	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0	R23

The above register determines the line number on which the display starts.



R23 = 0



R23 = 100

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Line Interrupt Register	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0	R19

Specifies which active scanning line generates an interrupt

Access Registers (Write only)

The access registers are used to access the register of the video display processor VDP, or the VRAM.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register number
VRAM Access base address register	0	0	0	0	0	A16	A15	A14	R14

When accessing the VDP and the Video RAM (VRAM), set the high-order three bits of the address in the VRAM access base address register.

When data is set in this register, and the VRAM is accessed, if there is a carry from A13, the data in the register is automatically incremented. In GRAPHIC 1, GRAPHIC 2, MULTICOLOR, and TEXT 1 modes, the data in the register is not automatically incremented.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register number
Status register pointer	0	0	0	0	S3	S2	S1	S0	R15

When reading the VDP status registers (S0 to S9), set the contents of the Status register pointer.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register number
Colour palette address register	0	0	0	0	C3	C2	C1	C0	R16
Control register pointer	A11	0	RS5	RS4	RS3	RS2	RS1	RS0	R17

The control register pointer may be used to access another register, or to automatically increment the data.

A11 = 1 Auto increment disable; 0 = disable

Command Registers (Write only)

The following command registers are used when issuing commands to the VDP.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register number
Source X low register	SX7	SX6	SX5	SX4	SX3	SX2	SX1	SX0	R32
Source X high register	0	0	0	0	0	0	0	SX8	R33
Source Y low register	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0	R34
Source Y high register	0	0	0	0	0	0	SY9	SY8	R35
Destination X low register	DX7	DX6	DX5	DX4	DX3	DX2	DX1	DX0	R36
Destination X high register	0	0	0	0	0	0	0	DX8	R37
Destination Y low register	DY7	DY6	DY5	DY4	DY3	DY2	DY1	DY0	R38
Destination Y high register	0	0	0	0	0	0	DY9	DY8	R39

Number of pixels, X low register	NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0	R40
Number of pixels, X high register	0	0	0	0	0	0	0	NX8	R41
Number of pixels, Y low register	NY7	NY6	NY5	NY4	NY3	NY2	NY1	NY0	R42
Number of pixels, Y high register	0	0	0	0	0	0	NY9	NY8	R43
Colour register	CH3	CH2	CH1	CH0	CL3	CL2	CL1	CL0	R44
Argument register	0	MXC	MXD	MXS	DIY	DIX	EQ	MAJ	R45
Command register	CM3	CM2	CM1	CM0	LO3	LO2	LO1	LO0	R46

Related keywords: VSTAT

VSTAT (Video Status)

Syntax: VSTATN

Purpose: Reads the video display processor's status registers 0 to 9.

STATUS REGISTER 0 to 9 (Read Only)

The following status registers are read-only registers for reporting the status

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Status registers 0	F	5S	C	Fifth sprite number					S0

F: Vertical scanning interrupt flag
When S0 is read,this flag is reset.

5S: Flag for the fifth sprite
Five sprites are aligned on the first horizontal line (In the G3 to G7 modes, 9 sprites are allowed).

C: Collision flag
Two sprites have collided.

Fifth sprite number:
The number of the fifth (or ninth) sprite.

	b7	b6	b5	b4	b3	b2	b1	b7	VDP Register Number
Status register 1	FL	LPS	Identification Number					FH	S1

FL: Lightpen flag (Lightpen flag set)
If the lightpen is to detect light, this bit as well as the IE2 bit must be both set in order for an interrupt to be enabled, When S1 is read, FL is reset.

Mouse switch 2 (Mouse flag set)
The second switch on the mouse was pressed.
In this case, when S1 is read, FL is not reset.

LPS: Lightpen switch (Lightpen flag set)
The lightpen switch was pressed.
In this case, when S1 is read, LPS is not reset.

Mouse switch 1 (Mouse flag set)
The first switch on the mouse was pressed.
In this case, when S1 is read, LPS is not reset.

Identification number:

The identification number of the VDP chip.

FH: Horizontal scanning interrupt flag
Horizontal scanning interrupt (which is specified in R19) flag. If IE1 is set, an interrupt is enabled. When S1 is read, FH is reset.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Status register	TR	VR	HR	BD	1	1	EO	CE	S2

TR: Transfer ready flag
When the CPU sends commands to the VRAM and other devices, the CPU checks this flag while transferring data. When this flag is set to 1, transfer may be done.

VR: Vertical scan line timing flag
During vertical scanning, this flag is set to 1.

HR: Horizontal scan line timing flag
During horizontal scanning, this flag is set to 1.

BD: Boundary colour detect flag
When the search command is executed, this flag detects whether the boundary colour was detected or not.

EO: Display field flag
When 0, indicates the first field.
When 1, indicates the second field.

CE: Command execution flag
Indicates that a command is being executed.

	b7	b6	b5	b4	b3	b2	b1	b0	VDP Register Number
Column register low	X7	X6	X5	X4	X3	X2	X1	X0	S3
Column register high	1	1	1	1	1	1	1	X8	S4
Row register low	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	S5
Row register high	1	1	1	1	1	1	E0	Y8	

The above registers are set to indicate the collision location of sprites, the location of lightpen detection, and the relative movement of the mouse.

Colour register	C7	C6	C5	C4	C3	C2	C1	C0	S7

The above colour register is used when the POINT and VRAM to CPU commands are executed. The VRAM data is set in this register.

Border X register low	BX7	BX6	BX5	BX4	BX3	BX2	BX1	BX0	S8
Border X register high	1	1	1	1	1	1	1	BX8	S9

When the search command is executed and the border colour has been detected, the X co-ordinate is set in the above registers.

Related Keyword: VDP

VERIFY

Syntax: VERIFY file

Purpose: This is a System Command which checks a file on disc.

It is commonly used following execution of a SAVE command in order to check that a program has been transferred to disc correctly.

Any error is reported as BAD DATA ERROR and a NO FILE ERROR indicates that a program file is non-existent.

Example:

```
VERIFY "1:HICK.XBS"
```

This will check the disc in drive 1 for the BASIC program HICK and then verify the the file.

Related Keywords: DRIVE SAVE LOAD

VOICE

Syntax: VOICE N,N1,N2,N3,N4,N5

Purpose: This is a Sound Command which sets up a voice for use by the MUSIC statement.

N is the "voice number" and can have values in the range 0 to 7.

Each voice then has given parameters as follows:

N1 is the "noise period", in the range 0 to 31. 0 represents the highest frequency noise.

N2 is the "maximum amplitude" of notes, in the range 0 to 15, where 0 is the quietest and 15 the loudest sound.

N3,N4,N5 allow construction of the sound envelope where N3 represents the "attack", N4 represents the "sustain", and N5 the "decay" times of each note played in a given channel. These timings are independent of the "note length", so that it is possible to start a new note before the last one has completed, or to complete a note before its time has expired, thus allowing "legato" and "staccato" playing. The values for all three timings fall within the range 0 to 255 where 0 represents the shortest time.

Up to eight voices may be defined in this way, which can be invoked by means of the letter V within a MUSIC statement. e.g. V0 selects voice 0.

Related Keywords: BEEP MUSIC PSG TEMPO

VPEEK (Video Peek)

Syntax: VPEEK(I)

Purpose: This is a Machine Code related command which operates in a similar manner to PEEK with the following differences.

1) The memory operations take place on the Video RAM.

2) The memory locations specified by I must be in the range 0 to 65535.

Related Keywords: DEEK DOKE PEEK POKE VDEEK VDOKE VPOKE

VPOKE (Video Poke)

Syntax: VPOKE I,J1,J2,...,Jn

Purpose: This is a Machine Code related command which operates in a similar manner to POKE with the following differences.

1) The memory operations take place on the video RAM

2) The memory locations specified by I must be in the range 0 to 16383 (0 to &3FFF)

Related Keywords: DEEK DOKE PEEK POKE VDEEK VDOKE VPEEK

WAIT

Syntax: WAIT J1,J2,J3

J1 is a port number and J2 and J3 are data items treated as 8 bit binary numbers.

Purpose: This command monitors directly the input and output from a particular I/O port.

The command suspends execution of a program whilst it monitors the port given by the value of J1. Execution will continue when a required condition, controlled by the selection of the data for J2 and J3, exists at the port.

The data which appears at the port is combined with the two items of data given in J2 and J3 in the following sequence.

1) The port data and J3 are combined using an EXCLUSIVE OR operation which is performed bit-by-bit on the two numbers.

J3 is optional and may be omitted. If J3 is not used it is assumed to be 0. Any XOR comparison with 0 simply produces a result identical to the port data.

2) The result of the first operation is combined with J2 using an AND operation, again on a bit-by-bit basis. The result of this becomes the final result.

3) The two steps above are repeated until the final result is non-zero. At this point program execution will then continue.

The comparisons are made using the two tables given below.

"XOR" (Exclusive OR) TABLE

BIT SETTING COMBINATIONS		RESULT
A	B	R
0	0	0
0	1	1
1	0	1
1	1	0

"AND" TABLE

BIT SETTING COMBINATIONS		RESULT
A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

a) These tables give all the possible combinations of settings for two bits.

b) Column A represents the setting (i.e. 0 or 1) of one bit, and column B the setting of the second bit.

c) For each combination of setting a result is given in column R.

Thus if two bits are set to 0 and 1 respectively, and we are doing an "AND" comparison, that particular combination is located in the "AND" table and the value of the result given in column R is read.

Example: WAIT &32,&FF,&0F

i) Execution of the program is suspended whilst the 8 bit user port (&32) is monitored and the bit setting first combined with the &0F data. Let us assume that the bit settings at the port are as follows:-

00000111

ii) This is now compared with the &0F data (J3) in accordance with the XOR table.

&0F data	-	00001111
Port data	-	00001111
Results from XOR table	-	00000000

iii) The result from the XOR comparison is now combined with the &FF data in accordance with the AND table.

&FF data	-	11111111
Result from XOR comparison	-	00000000
Final RESULT of AND comparison	-	00000000

Thus a final result of 0 is obtained.

iv) Execution of the program remains suspended and the sequence is repeated until the bit settings at the user port produced a "non-zero" final result, at which point execution then continues.

In this example, for a "non-zero" final result to be obtained, one of the following conditions must exist.

EITHER - any of the 4 most significant bits of the user port must be "set" (i.e. 1)

OR - any of the 4 least significant bits of the user port must be "reset" (i.e. 0)

Thus execution would be suspended until one of these conditions exist. The following example illustrates this.

Example: WAIT &32,&FF,&OF

i) 8 bit port set to 01101111 (i.e. 2 of the 4 most significant bits are "set").

ii) XOR comparison

```
&OF       - 00001111
Port data - 01101111
Result    - 01100000
```

iii) AND comparison

```
&FF            - 11111111
Result of XOR  - 01100000
Final result   - 01100000
```

Thus a "non-zero final result" is obtained and program execution will then continue.

Example:

WAIT &32,&40

i) Let us assume a 00001010 setting at the 8 bit user port.

ii) 'AND' comparison

```
&40       - 01000000
Port data - 00001010
Result    - 00000000
```

iii) A zero result is given, therefore the sequence repeats until a non-zero result is obtained.

Example: WAIT &32,&40

i) 8 bit port setting - 00001010

ii) 'AND' comparison

```
&40        - 01000000
Port data  - 01001100
Result     - 01000000
```

iii) A "non-zero" result is given and therefore program execution will continue.

It can be seen that in this example execution is suspended until BIT 6 of the port is set (1) (i.e. the condition required to produce a non-zero result)

Related Keywords: INP OUT

WIDTH

Syntax: WIDTH J

Purpose: This is a special command relating to output. It sets the width of the current output device, so that an automatic carriage return/line feed is generated as soon as the column count reaches the value given in J.

This is useful in certain printers such as Teletypes and Teleprinters where overprinting might occur when the print head reaches the end of a line.

Under normal operation the WIDTH is set to 0, when no automatic carriage return/line feed is produced.

WIDTH can also be used as a function to return the current width setting.

Example:

```
WIDTH 20
```

This would invoke a carriage return/line feed at columnn 20 of the output.

Example:

```
PRINT WIDTH
```

This would display the value of the current width setting on the screen.

Related Keyword: ZONE

XOR (Exclusive OR)

Syntax: I XOR I

Purpose: This is a logical operator used in the evaluation/comparison of statements.

Example:

```
PRINT CHR$(&61 XOR &20)
```

This example shows on application of XOR which results in the lower case 'a' character (ASCII &61) being printed as upper case 'A'. The logical process is illustrated below:-

0 1 1 0 0 0 0 1 (61_H = "a")
XOR 0 0 1 0 0 0 0 0 (20_H)
RESULT 0 1 0 0 0 0 0 1 (42_H - "A")

Related Keywords: OR AND NOT

ZONE

Syntax: ZONE J1,J2

J1 gives the value of the largest column number and is known as the zone limit. J2 gives the value of the zone width in number of columns.

Purpose: This is a special command relating to the format of output. It sets the zone width, and the largest column number for which printing to the next zone will stay on the same line.

The command is used to change the settings of the TAB functions contained in PRINT statements, according to particular requirements of output.

If omitted, J1 will default to 28 and J2 to 10.

ZONE can be used as a function to return the current values of J1 and J2.

Examples:

```
ZONE 32,16
```

This sets a zone width of 16 columns with column 32 indicating the zone limit.

PRINT ZONE(0) - this displays the current zone limit(J1)

PRINT ZONE(1) - this displays the current zone width (J2)

Related Keyword: WIDTH

CHAPTER 11

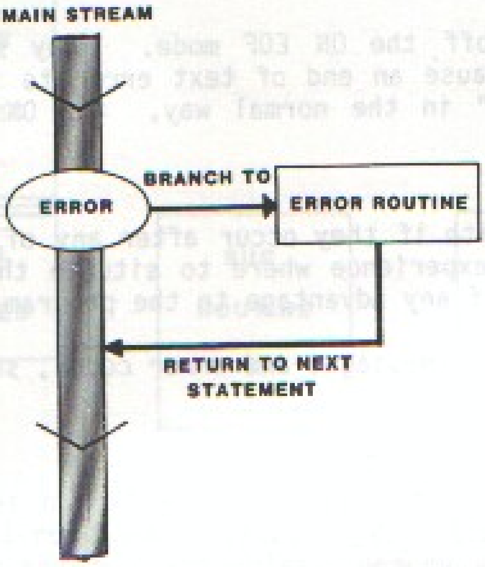
ERROR HANDLING WITHIN BASIC

The facility allows the handling of errors from within a BASIC program rather than abandoning execution. The errors are simply allowed to occur and are then dealt with by subroutines.

The following commands are used:

ON ERR GOTO "Line No."
ON ERR GOSUB "Line No."

Either of these two commands may be contained within a program listing. If an error occurs after one of these commands then a GOTO or GOSUB is made to the particular line number denoting the start of the error handling routine(s) as defined by the user.



If ON ERROR GOSUB "Line No." is used, the last statement of an error handling routine should be a RETURN. Program execution starts at the statement immediately following the one which caused the error.

When either of the two commands are used, an internal flag is set in order to activate the above procedure.

This flag reverts to "normal" after the first error and will therefore need setting again by another "ON ERR" statement positioned either at the end of the error routine, or soon after re-entering the main program.

OFF ERR
This command is used to restore the ON ERR flag to "normal" from within a program. However when a program "ends" normally, the flag reverts automatically. When OFF ERR has been used, any subsequent errors will be displayed as normal.

ERR,ERL,ERR$
These three statements are useful when included in the error handling routines.

ERR - returns the code number of the last error thereby indicating the nature of the error.

ERL - returns the line number at which the last error occurred.

ERR$ - returns the error STRING, without the word "ERROR", corresponding to the last error that occurred. This is useful when one particular error is expected and avoids having to flag every possible kind of error.

ON EOF GOTO "Line No."
ON EOF GOSUB "Line No."

This is a similar operation to ON ERR but relates specifically to routines which deal with encountering an "end-of-file" when "reading".

The difference is that the ON EOF flag is not reset by the execution of an ON EOF routine and therefore it remains in force.

OFF EOF
This is used to turn off the ON EOF mode. Any subsequent end-of-file encountered will then cause an end of text error to be displayed. Again when the program "ends" in the normal way, the ON EOF is automatically turned off.

REMEMBER:-
ERRORS are only dealt with if they occur after any of the statements. The programmer learns from experience where to situate the statements within a program in order to be of any advantage to the program execution.

NOTE: For details on error messages and error codes, see Appendix A.

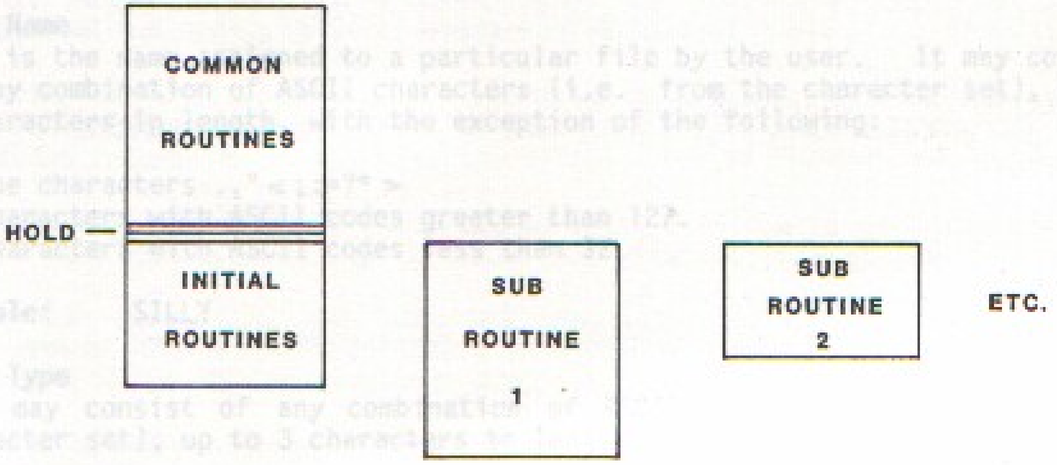
CHAPTER 12

CHAINING AND SEMI-CHAINING PROGRAMS

In addition to the normal use of the RUN and CHAIN commands in direct or deferred mode, EBASIC provides the facility to semi-chain programs.

This allows several program to access a "common pool of routines" without having to keep the same set of routines within each sub-program.

To do this the HOLD command is used immediately prior to executing a RUN or CHAIN. Both RUN and CHAIN will restore a "held" program by re-setting the text pointer as soon as the program is loaded. However, execution of the resulting program will commence from the start of the added section not from the beginning of the original progam.



The diagram illustates an original program consisting of a common routines section and an initial routines section. The initial routines would only be needed once to set up arrays, variables, memory space as in DIM and CLEAR statements etc.

The HOLD/CHAIN combination separates the two sections such that the other sub-programs illustrated can be called up in place of the initial routines section each time. (The flow of the original program would be structured such that the initial routines are executed before the line containing the HOLD/CHAIN commands is encountered).

Thus:-

```
HOLD line number :CHAIN"SUB1" calls up the 1st sub-program.
HOLD line number :CHAIN"SUB2" calls up the 2nd sub-program.
HOLD line number :CHAIN"SUB3" calls up the 3rd sub-program.
```

Each time execution would continue from the beginning of the added sub-program which can then access the COMMON ROUTINES section of the original program.

This method saves file space and greately improves the efficiency of a CHAIN, by speeding up the loading of each program.

NOTE: The line numbers of the sub-program must be selected so as to be greater than those of the common routines section.

The example Mailing List program given in Appendix M illustrates the use of this method of semi-chaining programs.

The original program consists of the common routines section in lines 10 to 890 and the inital routines section in lines 1000 to 1110.

The HOLD/CHAIN combination is found in line 900 and the flow of the original program is structured such that the initial routines (lines 1000 to 1110) are executed before line 900 is encountered.

There are three sub-programs involved with titles "MSUB1", and "MSUB2", and "MSUB3". These sub-programs are accessed according to the input in line 870 which places a value in N$ (given by the user response to the question "which?")

CHAPTER 13

FILE HANDLING IN BASIC

FILE NAMING CONVENTIONS

The file naming conventions used in EBASIC require the following 3 items to be specified when naming a file:

a) an optional, one character drive name.
b) a file name of up to 8 characters in length.
c) a file type of up to 3 characters in length, known as the file extension.

Drive Name
The drive name, when used, is specified as a single number from 0 to 1. If not given then the default drive is assumed. The default drive may be changed at any time by use of the DRIVE command.

File Name
This is the name assigned to a particular file by the user. It may consist of any combination of ASCII characters (i.e. from the character set), up to 8 characters in length, with the exception of the following:

a) the characters .," <;:=?* >
b) characters with ASCII codes greater than 127.
c) characters with ASCII codes less than 32

Example: SILLY

File Type
This may consist of any combination of ASCII characters (i.e. from the character set), up to 3 characters in length, with the following exceptions:

a) the characters .," <;:?*= >
b) characters with ASCII codes greater than 127.
c) characters with ASCII codes less than 32

There are 4 file types recognised by EBASIC. These are as follows:

1) XBS - This is a BASIC source file. If the "file type" is not specified then XBS is assumed.

2) ASC - This is an ASCII program file. These files are uncompressed source files. (XBS files contain "tokens" for reserved words whereas ASC files contain the words in full as they appear in a listing).

3) OBJ - This is an OBJECT file, or machine-code subroutine/data. A special area can be set up within the memory for the storage of machine-code routines by use of the CLEAR command. Anything stored in this area can be SAVEd as a .OBJ file, and LOADed into the area.

4) DATA FILES - Any other combination of characters specified for "file type" will be treated as a data file. Data files consist of a series of ASCII characters divided into one or more records, which can be serially accessed by a BASIC program (in its broadest sense this category also encompasses the three special file types .XBS, .OBJ, and .ASC).

The user may select combinations of characters for data file extensions according to individual requirements. The following are given as suggestions.

.BAK for backup copies
.DAT for data storage
.DOC for document files

Examples:

0:CATA.ASC -	refers to an ASCII file named CATA. for a disc in drive 0.
1:SILK.XBS -	refers to a BASIC source file named SILK, for a disc in drive 1
0:MEMO.OBJ -	refers to an OBJECT file named MEMO, for a disc in drive 0.
1:SILLYDAT -	refers to a DATA file named SILLYDAT, for a disc in drive 1.

NOTE: When working in BASIC, file names are contained within quote marks (") as shown in the example below:

"1:CATEL.XBS"

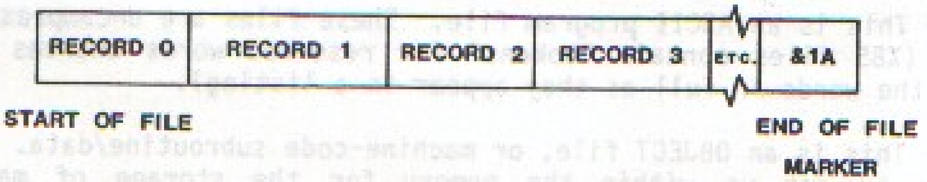
ACCESSING FILES

There are two commonly used methods for accessing files and these are known as:-

a) Sequential Access.
b) Random Access.

a) Sequential Access

Sequential Access is most often used for the manipulation of text or index files, where records may be of variable length, and need to be scanned (examined) sequentially i.e. one after the other, starting from the beginning of the file until the desired record location is found.

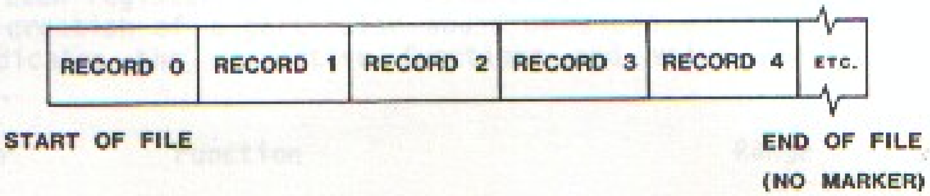


Each record should normally have a terminator, such as a carriage return code. There is a special code to mark the end-of-file (EOF).

EBASIC generates an EOF code when closing a sequential file, if the last operation was a WRITE, and will normally detect the end-of-file marker on a READ.

An end-of-file condition will occur if an attempt is made to read beyond the last allocated sector of the file.

b) Random Access
Records stored in random-access files are normally of a fixed length, OR of a variable length but contained within fixed-length blocks.



When a file is opened the "record length" is specified. Whenever a file input or output is required a "record number" is specified. This means that there can be free movement about the file in a completely random fashion, accessing only those records required.

Having accessed a particular record it is also possible to then read or write sequentially to the file from that point onwards (even though a random record length has been specified).

It is also possible, if required, to write a file with one record length, and then read or write to the same file with a different record length.

An EOF marker is not supplied when closing a random-access file. However, the end-of-file condition will occur when an attempt is made to read a sector of the disc that does not exist.

Unfortunately, this will not always be the case with a non-existent record, since the disc space may have already been created for it as a side effect of writing another record which uses the same physical disc sector. In that case it will be read as an empty record.

FILE-HANDLING COMMANDS
The commands which provide the facilities within file-handling are listed below:-

```
DRIVE    OPEN     CREATE
CLOSE    APPEND   PRINT#
INPUT#   INCH#    INCH#(N)
```

Refer to Chapter 10 for full details.

CHAPTER 14

PROGRAMMABLE SOUND GENERATOR

All the functions of the Programmable Sound Generator (usually abbreviated to PSG) are controlled by the Z80 processor by means of a series of register loads. Each register of the PSG relates to a specific function involved with the creation of a particular sound or sound effect. The following table indicates the respective functions and value ranges of the PSG registers.

Register	Function	Range
0	Channel A - lower 8 bits Pitch	0 to 255
1	Channel A - upper 4 bits	0 to 15
2	Channel B - lower 8 bits Pitch	0 to 255
3	Channel B - upper 4 bits	0 to 15
4	Channel C - lower 8 bits Ptich	0 to 255
5	Channel C - upper 4 bits	0 to 15
6	Noise period	0 to 31
7	Enable	0 to 255
8	Channel A - Amplitude	0 to 31
9	Channel B - Amplitude	0 to 31
10	Channel C - Amplitude	0 to 31
11	Envelope period lower 8 bits	0 to 255
12	Envelope period upper 8 bits	0 to 255
13	Envelope shape/cycle	0 to 15
14	Port A	0 to 255
15	Port B	0 to 255

REGISTERS 0 to 5

The values stored in these registers determine the frequency, or pitch, of the outputs of the respective channels A, B and C. Registers 0 and 1 control the pitch of Channel A, register 2 and 3 the pitch of Channel B, and registers 4 and 5 the pitch of Channel C.

To Determine the Pitch
The pitch generated by each of the 3 channels is determined by two registers for each channel. The values stored in the two registers represent a 12-bit number (0 to 4095 in decimal).

The equivalent 12-bit number, in decimal, can be found from:-

$TP = 256 \times RU + RL$

Where:-

TP is the decimal equivalent of the 12-bit tone period number.
RL is the lower 8 bits (Register 0 for Channel A)
RU is the upper 4 bits (Register 1 for Channel A)

RL may be thought of as a "fine tune" register and can have any value in the range 0 to 255.
RU may be thought of as a "coarse tune" register and can have any value in the range 0 to 15.

To find the pitch:-

$$\text{Pitch} = \frac{2 \times 10^6}{16 \times TP} \quad \text{Hertz}$$

Thus the higher the register values, the lower the pitch.

To find the Register Values
Given the desired pitch, it is possible to find the values for the two registers for each channel.

First find the Tone Period (TP) from:-

$$TP = \frac{2 \times 10^6}{16 \times \text{pitch (in Hertz)}}$$

Having determined the tone period, the register values can be obtained as follows:-

$$RU + \frac{RL}{256} = \frac{TP}{256}$$

Thus RU is given by the integer of (TP ÷ 256) and RL is given by multiplying the remainder from $\frac{TP}{256}$ by 256

Example:
Required frequency is 100Hz - what are the register values for Channel A?

For channel A RL is register 0 and RU is register 1

Calculating the Tone Period (TP):-

$$TP = \frac{2 \times 10^6}{16 \times 100}$$

$$= \underline{1250}$$

To find the value in register 1 (R1)

$$R1 = \frac{1250}{256} \text{ (ie integer of } 1250 \div 256$$

$$= \underline{4} \quad (1250 \div 256 = 4.88281)$$

Remainder from division is 0.88281

$\therefore$ register 0 (R0) = 0.88281 x 256

$$= \underline{226}$$

Although the range of frequencies which can be produced by the sound generator is from 30.5Hz to 125kHz, this is, in practice, limited by the capabilities of the audio amplifier/speaker combination, which sets an upper frequency limit of about 15kHz. (In practice, the human ear also sets a limit of about 10 to 17kHz, depending upon the age of the listener).

REGISTER 6

This register controls the frequency of the noise generated by the PSG. It is similar to the pitch registers, previously described, except that there is only one register, and the range of noise frequencies produced is from 4kHz to 125kHz.

To determine the Noise Frequency

$$\text{Noise frequency} = \frac{2 \times 10^6}{16 \times R6} \text{ Hertz}$$

R6 represents the contents of register 6 (in decimal), and can have values from 0 to 31.

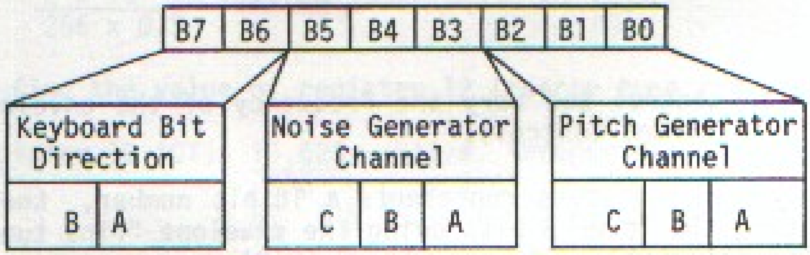
Similarly, given the noise frequency:-

$$\text{Then } R6 = \frac{2 \times 10^6}{16 \times \text{Noise Frequency (Hertz)}}$$

REGISTER 7

This is an 8-bit control register which is used to enable noise and pitch on channels A, B and C, and to control the direction of the keyboard scan ports, A and B.

REGISTER 7



Bits 0 to 2 - Enable pitch generator on channels A, B and C
Bits 3 to 5 - Enable noise generator on channels A, B and C

A logic '0' in bits 0 to 5 will enable the respective channel (ie 0=on, 1=off)

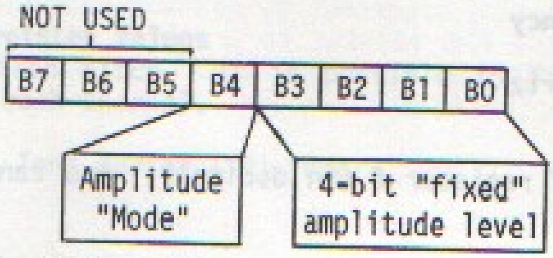
Bits 6 and 7 - A '0' in bits 6 and 7 will configure the keyboard ports as inputs, whilst a '1' will configure them as outputs. The machine software normally configures port A as output and port B as input (i.e. B6 = '1', B7 = '0')

NOTES: Care should be excercised when setting up this register, as writing to bits 6 and 7 could result in the keyboard being disabled.

The PSG command in BASIC expects entry in decimal or hexadecimal. It would be wise to select values for this register which do not change the setting of bits 6 and 7 (as configured by the machine software).

REGISTERS 8 to 10

These registers control the amplitude of the signals generated by each of the channels A, B and C, and also selects the "amplitude mode". Register 8 controls channel A, register 9 controls channel B, and register 10 controls channel C.



Bits 0 to 3 - Define the "fixed" level amplitude of a channel according to the value loaded for each register (values from 0 to 15)

Bit 4 - A '0' in bit 4 selects "fixed level amplitude" mode, whilst a '1' will select "variable level amplitude" and hence enable the Envelope Generator. It follows therefore, that bits 0 to 3, defining the value of a "fixed" level amplitude, are only active when bit 4 = '0' (i.e. they are ignored when bit 4 = 1 and amplitude control passes to the Envelope Generator)

NOTE: To turn a channel off the all zeros code is used in bits 0 to 3 (i.e. 0000)

REGISTERS 11 and 12

These registers are used to control and vary the frequency of the envelope generated (i.e. the Envelope Period Control).

The values stored in these two registers represents a 16 bit number, known as the Envelope Period (EP), the lower 8 bits being the envelope "fine tune" (R11) and the upper 8 bits the envelope "coarse tune" (R12).

To Determine the Envelope Period (EP)

$EP = 256 \times CT + FT$

Where:-

EP is the decimal equivalent of the 16 bit Envelope period number.
FT is the decimal equivalent of the "fine tune" register bits (i.e. lower 8 bit number).
CT is the decimal equivalent of the "coarse tune" register bits (i.e. upper 8 bit number).

To Determine the Envelope Frequency

$$\text{Envelope Frequency} = \frac{2 \times 10^6}{256 \times EP}$$

To Find The Register Values

Given the envelope frequencies it is possible to find the values of the two registers.

First find the Envelope Period (EP) from:-

$$EP = \frac{2 \times 10^6}{256 \times \text{Frequency}}$$

Having determined the envelope period, the register values can be obtained as follows:-

$$CT + \frac{FT}{256} = \frac{EP}{256}$$

Thus CT is given by the integer of (EP÷256) and FT is given by multiplying the remainder from $\frac{EP}{256}$ by 256

Example:

Required envelope frequency is 0.5Hz - what are the register values?

Calculating the Envelope Period (EP):-

$$EP = \frac{2 \times 10^6}{256 \times 0.5} = \underline{\underline{15,625}}$$

To find the value of register 12 (coarse tune register CT)

$$\text{Register 12 (CT)} = \frac{15,625}{256} \quad \text{(i.e. integer of 15,625} \div 256\text{)}$$

$$= \underline{\underline{61}} \quad (15,256 \div 256 = 61.035156)$$

Remainder from division is 0.035156

$$.'.\ \text{register 11 (FT)} = 0.035156 \times 256 = \underline{\underline{9}}$$

REGISTER 13

The lower 4 bits of register 13 control the envelope shape and cycle. The upper 4 bits of the register are not used.

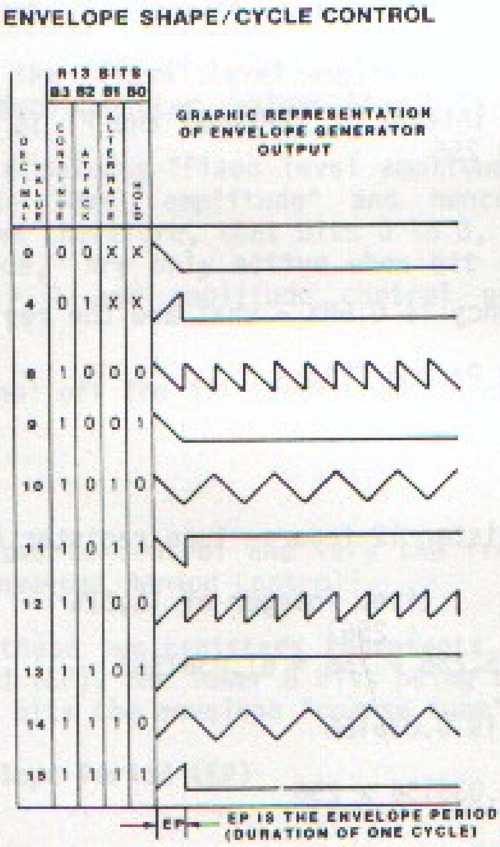
Each of the lower 4 bits controls a function in the envelope generator as follows:-

Bit 0	-	HOLD
Bit 1	-	ALTERNATE
Bit 2	-	ATTACK
Bit 3	-	CONTINUE

HOLD - When set to logic '1' limts the envelope to one cycle, holding the last count of the envelope counter (0000 or 1111, depending whether the envelope counter was in a count-down or count-up mode, respectively).

ALTERNATE - When set to logic '1', the envelope counter reverses count direction (up-down) after each cycle.
ATTACK - When set to logic '1', then envelope counter will count up (attack) from 0000 to 1111; when set to logic '0', the envelope counter will count down (decay) from 1111 to 0000.
CONTINUE - When set to logic '1', the cycle pattern will be as defined by the HOLD bit; when set to logic '0', the envelope generator will reset to 0000 after one cycle and hold at that count.

NOTE: When both the HOLD bit and ALTERNATE bit are set to '1', the envelope counter is reset to its initial count before holding.


ENVELOPE SHAPE/CYCLE CONTROL
R13 BITS
GRAPHIC REPRESENTATION OF ENVELOPE GENERATOR OUTPUT
EP
EP IS THE ENVELOPE PERIOD (DURATION OF ONE CYCLE)

	B3	B2	B1	B0
DECIMAL VALUE	CONTINUE	ATTACK	ALTERNATE	HOLD
0	0	0	X	X
4	0	1	X	X
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Fig 14.1 Envelope Shape/Cycle Control

REGISTER 14 and 15

These registers function as intermediate data storage registers between the PSG/CPU data bus and the two I/O ports available on the PSG. Using these registers for the transfer of I/O data has no effect at all on sound generation.

To output data from CPU to a peripheral on I/O Port A

1. Latch address R7 (select Enable register)
2. Write data to PSG (setting bit 6 of R7 to '1')
3. Latch address R14 (select IOA register)
4. Write data to PSG (data to be output on I/O Port A)

To input data from I/O Port A to CPU

1) Latch address R7 (select Enable register)
2) Write data to PSG (setting bit 6 of R7 to '0')
3) Latch address R14 (select IOA register)
4) Read data from PSG (data from I/O Port A)

NOTES: Once loaded with data in the output mode, the data will remain on the I/O port(s) until changed either by loading different data, by applying a reset or by switching to the input mode.
When in the input mode, the contents of registers 14 and/or 15 will follow the signals applied to the I/O port(s). However, transfer of this data to the CPU bus requires a "read" operation as described above.

SOUND PROGRAM

On the master disc, there is a sound program included to allow the user to experiment with the various register values of the Programmable Sound Generator and observe the results.

To use the program

1. Ensure that BASIC is loaded.

2. Run the sound program by typing RUN "SOUND".

3. Use the cursor keys to position the cyan coloured cursor to the desired register, and bit, and then key 0 or 1 to set the desired bit value. The effects of any changes are heard immediately.

NOTE: the two most significant bits of register 7 are masked (indicated by X's). These two bits enable the keyboard, and do not affect sound generation.

The program does not allow access to register 14 and 15. These registers are concerned with scanning the keyboard, and do not affect sound generation.

Try experimenting with the program. When you have found the sound you want to create, make a note of the register value(s) for use in your own programs.

SOUND VARIATION

Relative Channel Volume

The independently programmable amplitude control for each channel allows up to 16 levels if using the processor controlled amplitude mode (bit 4 of registers 8, 9, or 10 = 0). In the case of a decaying or steady note, when a note is played or "fired", a frequency may be set up in the coarse and fine tune registers and then an amplitude value placed in the respective register 10, 11, or 12. The value which is placed to play the tune can be an independent variable, allowing channels to play their respective melody lines with varying force.

Decay
One difference between sounds is the speed with which the note gains and loses volume. This is known as attack and decay. If all of the notes can be decayed at a uniform rate, the automatic envelope generator can be set to produce a decaying waveform. Each of the three channels can have the same decay constant but differing playing times to simulate the same instrument with differing note strike-times.

Other Effects
The addition of variable noise to any or all of the channels can produce effects such as "breathing" with a wind instrument. Or noise can be used alone to produce a drum rhythm. The fact that the noise dominant frequencies are variable allows "synthesizer" type effects with simple processor interaction.

Other pleasing effects include vibrato and tremolo, the cyclical variation of the frequency and volume. Because an intelligent microprocessor is controlling the effect, they can be all keyed to the tune itself or to other external stimuli.

SPECIAL SOUND EFFECTS
One of the main uses of the PSG is to produce non-musical sound effects to accompany visual action or as a feature in itself. The following sections outline techniques and provide actual examples of some popular effects.

To prevent you being deafened or worse, a short delay routine (FOR I = 1 to 100:NEXT) followed by RST may be put before the END statement. Please note that this may affect any program variables held, as well as graphics. It should be used with care.

Tone Only Effects
Many effects are possible using only the tone generation capability of the PSG without adding noise and without using the PSG's envelope generation capability. Examples of this type of effect would include telephone tone frequencies (two distinct frequencies produced simultaneously) or the European Siren effect listed below (two distinct frequencies sequentially produced).

EUROPEAN SIREN SOUND EFFECT

The following BASIC listing will produce the European Siren effect.

```
 10 REM SIREN EFFECT
 20 FOR I=0 TO 9
 30 PSG0,254:PSG1,0
 40 PSG7,126
 50 PSG8,15
 60 FOR J=1 TO 300:NEXT
 70 PSG0,86:PSG1,1
 80 FOR J=1 TO 300:NEXT
 90 NEXT I
100 PSG8,0
110 END
```

Noise Only Effects
Some of the more commonly required sounds require only the use of noise and the envelope generator (or processor control of channel envelope if other channels are using the envelope generator).

Examples of this, listed below, are gunshot and explosion. In both cases pure noise is used with a decaying envelope.

In the examples shown the only changes are in the length of the envelope as modified by the coarse tune register and in the noise period.

GUNSHOT SOUND EFFECT

The following BASIC listing will produce the effect.

```
 10 REM GUNSHOT
 20 FOR J=1 TO 4
 30 PSG6,15:PSG7,71
 40 PSG8,16:PSG9,16:PSG10,16
 50 PSG12,16
 60 PSG13,0
 70 T=RND(2000):FOR I=1 TO T:NEXTI
 80 NEXT J
 90 END
```

EXPLOSION SOUND EFFECT

The following BASIC listing will produce the Explosion effect

```
 10 REM EXPLOSION
 20 PSG6,31:PSG7,71
 30 PSG 8,16:PSG9,16:PSG10,16
 40 PSG12,100
 50 PSG13,0
 60 END
```

Frequency Sweep Effect
The Laser, Whistling Bomb, Wolf Whistle, and Race Car sounds, listed below, all utilize frequency sweeping effects. In all cases they involve the increasing or decreasing of the values in the tone period registers with variable start, end, and time between frequency changes. For example, the sweep speed of the Laser is much more rapid than the high gear accelerate in the race car, yet both use the same computer routine with differing parameters.

Other easily achievable results include "doppler" and noise sweep effects. The sweeping of the noise clocking register (R6) produces a "doppler" effect which seems well suited for "space war" type games.

LASER SOUND EFFECT

The following BASIC listing will produce the Laser Sound effect.

```
 10 REM LASER
 20 PSG7,126
 30 PSG8,15
 40 FOR I=18 TO 255 STEP 40
 50 PSG0,I
 60 NEXT
 70 FOR I=255 TO 18 STEP-10
 80 PSG0,I
 90 NEXT
100 PSG,7 7F
110 REM
120 END
```

WHISTLING BOMB EFFECT

The following BASIC listing will produce the Whistling Bomb effect.

```
 10 REM WHISTLING BOMB
 20 PSG7,126
 30 PSG8,15
 40 FOR I=1 TO 255
 50 PSG0,I
 60 NEXT
 70 PSG6,31:PSG7,71
 80 PSG8,16:PSG9,16:PSG10,16
 90 PSG12,100
100 PSG13,0
110 END
```

Multi-Channel Effect

Because of the independent architecture of the PSG, many rather complex effects are possible without burdening the processor. For example, the Wolf Whistle effect below shows two channels in use to add constant breath hissing noise to the three concentrated frequency sweeps of the whistle. Once the noise is put on the channel, the processor only need be concerned with the frequency sweep operation.

WOLF WHISTLE SOUND EFFECT

The following BASIC listing will produce the Wolf Whistle effect.

```
10 REM WOLF WHISTLE
20 PSG6,1:PSG7,110:PSG9,9
30 PSG1,0:PSG8,15
40 FOR I=64 TO 32 STEP-.35:PSG0,I:NEXT
50 FOR I=0 TO 150:NEXT
60 FOR I=64 TO 48 STEP-.17:PSG0,I:NEXT
70 FOR I=48 TO 104 STEP.5:PSG0,I:NEXT
80 PSG8,0:PSG9,0
90 END
```

RACE CAR SOUND EFFECT

The following BASIC listing will produce the Race Car effect, including gear changes.

```
 10 REM RACING CAR
 20 PSG3,15
 30 PSG7,124
 40 PSG8,15:PSG9,10
 50 S=11:F=4:GOSUB110
 60 S=9:F=3:GOSUB110
 70 S=6:F=1:GOSUB110
 80 PSG8,0:PSG9,0
 90 END
100 REM SWEEP ROUTINE
110 FOR I=S TO F STEP-1:PSG1,I
120 PSG0,255
130 FOR J=255 TO 0 STEP-1:PSG0,J
140 NEXT J,I
150 RETURN
```

CHAPTER 15

GRAPHICS TECHNIQUES

This section has been included in order to illustrate some of the more simple techniques used to produce "graphic effects" within BASIC.

SIMPLE DRAWINGS

Simple drawings, shapes, and patterns can be produced on the screen by use of the following facilites which can be manipulated in either 40 column display or 32 column display.

Lines.
The DRAW command will produce straight lines from one specified point to another specified point on the screen (co-ordinates being used to position each point).

The command also contains a qualifier which specifies the type of line to be drawn (full,dotted,dashed etc.).

Circles/Ellipse.
Circles and Ellipses can be drawn by use of the ELLIPSE command, which again may also specify the type of line to be used, (dots, dashes, etc.).

The position of a circle or ellipse is fixed by specifying the co-ordinates of the respective centres within the command.

Polygon.
The POLY command is used to draw polygons with varying numbers of sides and again the type of line to be used may be specified within the command.

The position of a polygon is fixed by specifying the co-ordinates of the centre within the POLY command.

Colour.
To select a particular colour for the lines produced by the DRAW, ELLIPSE, and POLY commands the GCOL command is used. (ie. Graphics colour).

GCOL normally preceeds the particular command it relates to in a program listing and may specify any one of 16 different colours.

Example

The following program illustrates the method of producing simple coloured drawings on the screen using the DRAW and ELLIPSE commands. Carefully type in the program listing and note the result when it is RUN.

```
 10 REM SPACESHIP
 20 REM HULL
 30 CLS BCOL13
 40 GCOL 9
 50 DRAW  120,75  TO  90,80  TO 80,100 TO 80,130  TO  90,150  TO  110,160  TO
    140,160 TO 160,150 TO 170,130 TO 170,100 TO 160,80
 60 DRAW 160,80 TO 130,75
 70 DRAW 170,120 TO 185,120 TO 185,110 TO 190,100 TO 175,100 TO 180,110
    TO 180,115 TO 170,115
 80 DRAW  80,120  TO  65,120  TO  65,110  TO  60,100  TO  75,100  TO  70,110  TO
    70,115 TO 80,115
 90 DRAW  90,80 TO 70,40 TO 65,40 TO 65,35 TO 80,35 TO 80,40 TO 75,40 TO
    95,79
100 DRAW  160,80  TO  180,40  TO  185,40  TO  185,35  TO  170,35  TO  170,40  TO
    175,40 TO 155,79
110 GCOL11
120 ELLIPSE 92,130,4
130 ELLIPSE 102,130,4
140 ELLIPSE 125,130,5
150 ELLIPSE 150,130,4
160 ELLIPSE 160,130,4
170 ELLIPSE 100,110,6
180 ELLIPSE 150,110,6
190 ELLIPSE 125,115,3
210 DRAW 130,100 TO 130,30
220 DRAW 120,100 TO 120,30
230 FOR I=35 to 95 STEP 5
240 DRAW 120,I TO 130,I
250 NEXT I
254 GCOL3
255 DRAW 120,100 TO 120,120 TO 130,120 TO 130,100 TO 120,100
260 GCOL1
270 DRAW 110,160 TO 105,175 TO 100,177
280 DRAW 125,160 TO 125,175 TO 130,180
290 DRAW 140,160 TO 145,175 TO 142,178
300 REM BUG
310 GCOL3
320 ELLIPSE 40,40,10
330 ELLIPSE 27,40,3
340 ELLIPSE 53,40,3
350 ELLIPSE 36,43,2
360 ELLIPSE 44,43,2
370 DRAW 36,35 TO 36,37 TO 44,37 TO 44,35 TO 36,35
380 DRAW 45,31 TO 45,25 TO 47,25 TO 47,24 TO 44,24 TO 44,31
390 DRAW 35,31 TO 35,25 TO 33,25 TO 33,24 TO 36,24 TO 36,31
400 GCOL1
410 DRAW 34,48 TO 30,55
420 DRAW 28,53 TO 32,57
430 DRAW 46,48 TO 50,55
440 DRAW 52,53 TO 48,57
450 REM MARTIAN
460 GCOL9
470 DRAW 200,60 TO 205,65 TO 210,60 TO 200,60
480 ELLIPSE 205,67,2
490 GCOL11
500 DRAW 200,50 TO 210,50 TO 215,30 TO 195,30 TO 200,50
510 ELLIPSE 205,55,5
520 DRAW 203,57 TO 207,57
```

```
530 DRAW 203,56 TO 203,50
540 DRAW 207,56 TO 207,58
550 DRAW 205,54 TO 205,56
560 DRAW 204,53 TO 206,53
570 DRAW 200,30 TO 200,25 TO 197,25 TO 197,24 TO 201,24 TO 201,30
580 DRAW 210,30 TO 210,25 TO 213,25 TO 213,24 TO 209,24 TO 209,30
590 DRAW 200,50 TO 190,45 TO 191,44 TO 199,48
600 DRAW 210,50 TO 220,45 TO 219,44 TO 211,48
610 GCOL3
620 ELLIPSE 205,45,2
630 ELLIPSE 205,40,2
640 ELLIPSE 205,35,2
650 GCOL15
660 PRINT @2,21;MUL$("*",38)
670 END
```

Doing programs like these can often be made easier, (despite the "I draw at the keyboard" problem), using 1mm square, A4 size linear graph paper. There are available in some stores special printed "screen sheets" for planning graphics. (Don't forget that in graphics modes, 0,0 is at the bottom left of your screen.) and that the first digit is along the base line (left to right) and the second digit is up the screen, (bottom to top). See ORIGIN command.

MOVEMENT

Simple.
There are several methods which can be utilised to create an impression of simple movement in graphics.

Perhaps the most fundamental way is to use a loop within a program listing which will print a shape on the screen, then apparently "rub it out" and print it again in a new position. This is similar to the process involved in making cine films produce movement on a screen.

To create the impression of a character or shape moving between two specified points it is necessary to draw it in several positions between the points, and erase the last drawn character immediately before to printing each new one.

The smaller the distance between each position, the smoother will be the movement. If the positions are too far apart then the movement created will be more like a jumping action from one place to the next.

Example
Suppose we wish to make a horizontal line appear to move down the screen. The sequence of operations required would be as follows:-

1. Draw the line in its initial position in a contrasting colour to the background.

2. Draw the line again in exactly the same position but this time using the same colour as the background (this gives the effect of "rubbing out" the line).

3. Re-draw the line (in a contrasting colour) in a new position which is a relatively small distance from the original position.

4. Draw over this line using the same colour as the background (ie. repeat as in 2 above).

5. Repeat as in 3 above - and so on.

Thus the sequence continues, selecting a number of positions until the final location has been reached.

Try the following example:

```
 10 CLS
 20 GCOL6
 30 DRAW 70,120 TO 180,120
 40 GCOL4
 50 DRAW 70,120 TO 180,120
 60 GCOL6
 70 DRAW 70,100 TO 180,100
 80 GCOL4
 90 DRAW 70,100 TO 180,100
100 GCOL6
110 DRAW 70,80 TO 180,80
120 GCOL4
130 DRAW 70,80 TO 180,80
140 GCOL6
150 DRAW 70,60 TO 180,60
160 GCOL4
170 DRAW 70,60 TO 180,60
180 GCOL6
190 DRAW 70,40 TO 180,40
200 GCOL4
210 DRAW 70,40 TO 180,40
220 GCOL6
230 DRAW 70,10 TO 180,20
240 END
```

Before reading any further type in the above listing (remember to use the NEW command in order to clear any existing programs) and observe the result when you RUN the program.

The speed of operation makes the line appear to move from it's initial position to the final location.
We can give the movement a continuous effect simply create a loop by sending control back to the line which represents the beginning of the sequence. This is done by replacing line 240 with the following:-

```
240 GOTO 30
```

Add this new line and then sit and watch.

Remember to use 'SHIFT BREAK' to break out of the loop once it is set in motion. Use CLS ENTER to clear the screen.

The following example uses this method to illustrate "rubbing out" and "movement" in respect of smoke coming from a ships' funnel. Type in the following listing and note the result.

```
 10 CLS
 20 GCOL15
 30 DRAW 0,60 TO 55,60
 40 DRAW 185,60 TO 240,60
 50 REM HULL
 60 GCOL6
 70 DRAW 60,50 TO 180,50 TO 190,70 TO 160,70 TO 157,65 TO 90,65 TO 80,70 TO
    65,70 TO 60,75 TO 50,75 TO 50,70 TO 60,50
 80 REM FLAGS
 90 GCOL12
100 DRAW 65, 70 TO 65,105 TO 75,100 TO 65,95
110 GCOL10
120 DRAW 180,70 TO 180,100 TO 185,100 TO 185,95 TO 180,95
130 REM CABIN
140 GCOL14
150 DRAW 100,65 TO 100,90 TO 110,90 TO 110,80 TO 150,80 TO 150,65
160 REM FUNNEL
170 GCOL1
180 DRAW 120,80 TO 120,100 TO 130,100 TO 130,80
190 REM CIRCLES
200 GCOL6
210 ELLIPSE 115,75,3
220 ELLIPSE 125,75,3
230 ELLIPSE 135,75,3
240 REM SMOKE
250 GCOL1
260 DRAW 126,103 TO 134,103
270 GCOL4
280 DRAW 126,103 TO 134,103
290 GCOL1
300 DRAW 133,105 TO 141,105
310 GCOL4
320 DRAW 133,105 TO 141,105
330 GCOL1
340 DRAW 138,107 TO 142,107
350 GCOL4
360 DRAW 138,107 TO 142,107
370 GCOL1
380 DRAW 140,110 TO 149,110
390 GCOL4
400 DRAW 140,110 TO 149,110
410 GCOL1
420 DRAW 145,112 TO 151,112
430 GCOL4
440 DRAW 145,112 TO 151,112
450 GCOL1
```

```
460 DRAW 149,114 TO 154,114
470 GCOL4
480 DRAW 149,114 TO 154,114
490 GCOL1
500 DRAW 152,116 TO 158,116
510 GCOL4
520 DRAW 152,116 TO 158,116
530 GOTO 250
```

Again use 'SHIFT BREAK' to break out of the loop and CLS to clear the screen.Another example is given below to illustrate the "rubout" and "movement" effect but in this case the FOR-NEXT statement has been used to create the required loop for the sequence.

```
 10 REM FANLINE
 20 CLS
 30 FOR J=1 TO 10
 40 BCOL1
 50 FOR I=50 TO 200 STEP 10
 60 GCOL RND(13)+2
 70 DRAW 50,50 to 200,I
 80 NEXT I
 90 FOR I=200 TO 50 STEP -10
100 GCOL1
110 DRAW 50,50 TO 200,I
120 NEXT I
130 NEXT J
140 BCOL4
150 END
```

There are in fact two loops in this example. An outer loop using J (lines 30 and 130) and an inner loop using I (lines 50 and 120).

The inner loop on I creates the necessary movement cycle and the outer loop on J causes the whole cycle to repeat 10 times.

Don't forget to clear both screen and memory.

USE OF SYMBOLS

Imaginative use of keyboard characters and symbols can often produce "graphic effects".

For example the asterisk (*) and hash (#) symbols are commonly used to make up borders by printing lines of characters in succession (horizontal and vertical).

Lines of Symbols.
The MUL$ command can be used to produce a number of characters in a single line as follows:-

```
  PRINT MUL$("*",30)
```

This will print a line of 30 asterisks across the screen.

```
  PRINT MUL$("#",25)
```

This will print a line of 25 hash symbols across the screen.

Type in the two statements given above (separately) and observe the result when you press ENTER. Clear the screen using CLS.

A line of symbols/characters can be placed anywhere on the screen by specifying the co-ordinates of the first character of the line using the PRINT @ command.

```
  PRINT@ 8,9,MUL$("*",15)
```

This will print a line of 15 asterisks across the screen starting from the character position 8,9 on the imaginary "character grid" of the screen.

The "character grid" is different from the "pixel grid". The character grid is numbered 0 to 39 horizontally from left to right and 0 to 23 vertically from TOP to BOTTOM in "40 column display", and 0 to 31 horizontally (left to right) 0 to 23 vertically (top to bottom) in "32 column display".

Movement of Symbols.
Lines of characters/symbols can be made to move down the screen in a similar process to making drawn lines move.

In this case a line of "space" needs to be printed over the top of the line of characters to produce the "rub out" effect. A new line of characters is then printed in a lower position and the process repeated as before until the characters reach the final location.

Changing the vertical component of the character grid co-ordinates in the PRINT@ command will alter the position of the line of characters on the screen. A FOR-NEXT loop can be set up to carry out this process thereby producing movement of the line down the screen (or up) as illustrated by the example given below:-

```
10 CLS
20 FOR I = 1 TO 20
30 PRINT@10,I,MUL$("*",20)
40 PRINT@10,I,MUL$(" ",20)
50 NEXT I
```

A second FOR-NEXT loop could be introduced in order to cause the sequence to repeat a given number of times and this is illustrated by the following listing.

```
10 CLS
20 FOR J=1 TO 5
30 FOR I=1 TO 20
40 PRINT@10,I,MUL$("*",20)
50 PRINT@10,I,MUL$(" ",20)
60 NEXT I
70 NEXT J
```

This will cause the sequence to be repeated 5 times as given by the FOR-NEXT loop J. Try the above example.

Alternatively, a continuous cycle could be set up by the use of the GOTO command as shown below in line 60. "NEW" then add program, first.

```
10 CLS
20 FOR I=1 TO 20
30 PRINT@10,I,MUL$("*",20)
40 PRINT@10,I,MUL$(" ",20)
50 NEXT I
60 GOTO 20
```

In this case 'SHIFT BREAK' would then have to be used to break out of the loop.

The following is another similar example but presented slightly differently. Try to work out the sequence before running the program in the computer.

```
 10 REM ADVANCE
 20 CLS
 30 LET A$="# # #"
 40 LET B$=" "
 50 FOR J=1 TO 5
 60 FOR I=1 TO 20
 70 BCOL6
 80 PRINT@10,I;MUL$(A$,5)
 90 PRINT@10,I;MUL$(B$,25)
100 NEXT I
110 NEXT J
120 BCOL4
130 CLS
140 END
```

The example given below illustrates movement across the screen. In this instance it is the horizontal component of the character co-ordinate in the PRINT@ command which is manipulated.

```
10 CLS
20 FOR I=10 TO 80
30 PRINT@I,12;"*"
40 PRINT@I,12;" "
50 NEXT I
60 GOTO20
```

Use 'SHIFT BREAK' to break out of the loop when required then clear the memory, (NEW)..

SHAPES

The characters and symbols which can be accessed from the keyboard are simply regarded as shapes by the computer which make up the character set.

Each shape (character/symbol) is defined and constructed within a grid pattern of 8x8 pixels. The example in Fig.15.1 illustrates the formation of the letter E in the character set.

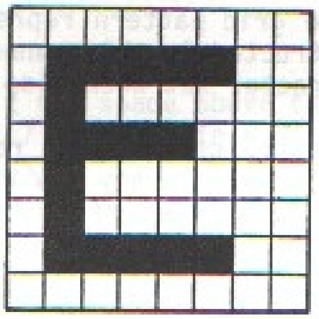


Fig. 15.1

Once constructed (or defined) within this grid pattern each shape (character/symbol) is allocated a "code number" for reference. The code number is a decimal number in the range 0 to 255 and is known as the ASCII code for a particular shape. The computer then uses these ASCII codes in order to recognise and use particular shapes during processing.

A table is given in appendix D which illustrates the ASCII codes and respective characters. It will be seen that codes 128 to 160 are apparently left free.

Facilities are provided within EBASIC which allow users to define (construct) their own shapes, symbols, or redefine the character set if so desired but we would not recommend the latter if there is a lack of experience. Some of the ASCII codes from 128 to 160 contain control characters but the following codes are free to be allocated to shapes defined by the user without affecting the existing character set:

130,131,132,134,142 to 154,156 to 160

Defining a Shape.
The first stage in defining any new shape is to use an 8x8 grid pattern of squares on which to construct the desired character. Look at the example given in Fig.15.2. which illustrates a "little man".

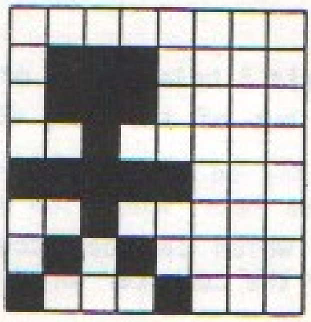


Fig.15.2

Remember, each square of the grid pattern represents a single pixel on the screen, thus each shape constructed in this manner will be of a similar size to the existing keyboard character set when displayed on the screen.

Each horizontal row of 8 squares on the grid represents what is known as 1 byte of data, and each pixel square in any row represents what is known as 1 bit of that byte of data. Thus the grid is made up of 8 bytes of data stacked horizontally on top of each other. Fig.15.3 illustrates this.

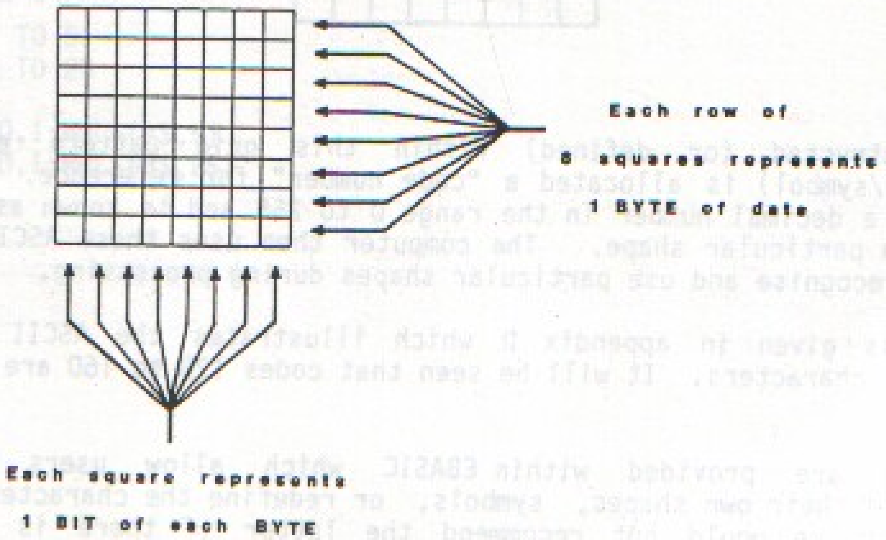


Fig.15.3

Another point to consider when constructing shapes on the grid is that in 32 column display all the squares of the grid may be used whereas for normal 40 column display the two most right hand columns of squares must be left blank (since they are not displayed in this case). In other words the last two BITS of each BYTE must be left blank when defining shapes for use in 40 column display.

BASIC expects the data which defines a shape to be entered in hexadecimal format. Fig. 15.4 shows a suggested 'grid' for the calculation of the Hex numbers for the shape definition. Each 8 x 8 pattern is divided into 2 nibbles per row (a nibble is 4 bits). This makes conversion into hexadecimal convenient, as per "little man" example following.

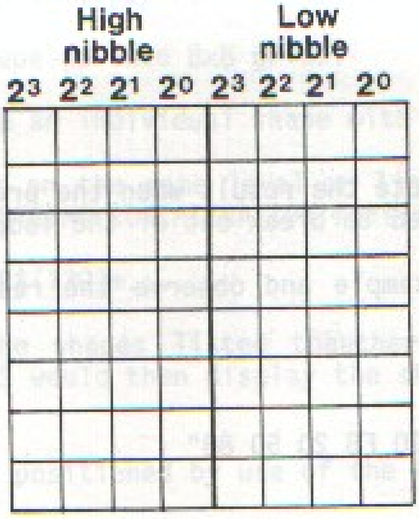


Fig.15.4

```
SHAPE 159,"00 .....   (That's the space above the head),
          "00 70      (4+2+1=7  Low nibble = 0)
          "00 70 70   (and again)
          "00 70 70 20
                      (High nibble 2, Low nibble 0)
          "00 70 70 20 F8
                      (High nibble 8+4+2+1=15, loop
                       up Decimal 15 in the table:
                       Its F. Low nibble = 8)
          "00 70 70 20 F8 20
                        (High nibble 2, Low nibble 0)
          "00 70 70 20 F8 20 50
                        (High nibble 4+1=5, Low nibble=0)
          "00 70 70 20 F8 20 50 88
                        (High and low both = 8)
```

So the shape line is:-

SHAPE 159, "00 70 70 20 F8 20 50 88"
which will put the shape into memory. The best way to check it is:

PRINT CHR$(159),

If you've succeeded, the shape will be printed on the screen.

Movement of a Shape.
The principles of simple movement as described earlier can also be applied. The following example enters the shape "little man" and then makes it appear to move.

```
10 CLS 32
20 SHAPE 150, "00 70 70 20 F8 20 50 88"
30 FOR I=10 TO 30
40 PRINT@I,12,CHR$(150)
50 PRINT@I,12," "
60 NEXT I
70 GOTO 30
```

Type in the listing and note the result when the program is RUN. The 'SHIFT BREAK' will have to be used to break out of the loop.

Type in the following example and observe the result when the program is RUN.

```
 10 REM ARMY ADVANCE
 20 CLS 32
 30 SHAPE 150, "00 70 70 20 F8 20 50 88"
 40 LET A$=CHR$(150)
 50 LET B$=" "
 60 LET C$=A$+B$
 70 BCOL6
 80 FOR I=1 to 20
 90 PRINT@10,I,MUL$(C$,12)
100 PRINT@10,I,MUL$(B$,24)
110 NEXT I
120 GOTO 80
```

It will be necessary to use the 'SHIFT BREAK' to break out of the loop once it is set up.

Now define a character shape of your own, using the method described previously, then use it in the above example by replacing the data in the SHAPE command (line 30) with your new data relating to the shape you have defined.

RUN the program with your new shape in.

Use 'SHIFT BREAK' to break out of the loop, clear the screen. Key NEW ENTER to clear the memory followed by BCOL4 to return to normal backdrop colour.

Building Shapes.

It is possible to build up larger shapes by printing several individual character shapes adjacent to each other. This is known as contiguous graphics. To produce extended shapes of this kind the following procedure should be adopted.

1) Draw the whole shape on grid paper.

2) Divide the complete shape up into 8x8 grids.

3) Define each 8x8 grid as an individual shape with its own ASCII CODE.

4) Individual grid shapes on the same level or line can be printed next to each other by use of the STRING CONCATENATION facility as follows:-

```
A$=CHR$(140)+CHR$(120)+CHR$(130)+.....
```

Thus A$ would contain the shapes listed together according to the ASCII codes specified. PRINT A$ would then display the shapes side by side on the screen.

5) Lines of shapes can be positioned by use of the PRINT@ command.

```
eg. PRINT @12,15,A$
    PRINT @12,14,B$ etc.
```

Using the method outlined above, larger shapes can be displayed on the screen.

Type in the following example which uses the principles outlined above, and observe the result when the program is RUN (note that line 20 sets 32 column display for this example)

```
 10 REM ALIEN
 20 CLS32
 30 SHAPE 130,"081422410103070F1F3F7FFFFFFFFFE7
    F8FCFEFFFFFFFFE71028448280C0E0F0"
 40 SHAPE 134,"0F0F0F0F0F0F0F0FE7FFFFFCFCFFFFFF
    E7FFFF3F3FFFFFFFF0F0F0F0F0F0F0F0"
 50 SHAPE 138,"0000000003030303F0F0F0F0FCFC0C0C
    0F0F0F0F3F3F303000000000C0C0C0C0"
 60 A$=CHR$(130)+CHR$(131)+CHR$(132)+CHR$(133)
 70 B$=CHR$(134)+CHR$(135)+CHR$(136)+CHR$(137)
 80 C$=CHR$(138)+CHR$(139)+CHR$(140)+CHR$(141)
 90 PRINT@ 12,10;A$
100 PRINT@ 12,11;B$
110 PRINT@ 12,12;C$
120 END
```

Note how each SHAPE command (lines 30,40,50) gives the data for four shapes thereby allocating consecutive ASCII codes from the first one specified. Thus the complete shape which appears on the screen consists of three lines of shapes, each line containing four adjacent shapes.

Now clear the screen and return to normal 40 column display, (CLS40).

Try the following example which makes use of the built up shape again. Observe the result when the program is RUN and examine the listing so as to understand how the application of simple movement has been used by adopting the pri.iciples described previously.

```
 10 REM ALIEN MOVE
 20 CLS32
 30 SHAPE 130,"081422410102070F1F3F7FFFFFFFFFFE7
    F8FCFEFFFFFFFFFE71028448280C0E0F0"
 40 SHAPE 134,"0F0F0F0F0F0F0F0FE7FFFFFCFCFFFFFF
    E7FFFF3F3FFFFFFFFF0F0F0F0F0F0F0F0"
 50 SHAPE 138,"000000000030303F0F0F0F0FCFC0C0C0F
    0F0F0F3F3F30300000000000000C0C0C0C0"
 60 A$=CHR$(130)+CHR$(131)+CHR$(132)+CHR$(133)
 70 B$=CHR$(134)+CHR$(135)+CHR$(136)+CHR$(137)
 80 C$=CHR$(138)+CHR$(139)+CHR$(140)+CHR$(141)
 90 FOR I=1 TO20
100 PRINT@ 12,I,A$
110 PRINT@ 12,I+1,B$
120 PRINT@ 12,I+2,C$
130 PRINT@ 12,I,MUL$("",4)
140 PRINT@ 12,I+1,MUL$("",4)
150 PRINT@ 12,I+2,MUL$("",4)
160 NEXT I
170 GOTO 90
```

You will notice that the shape appears to be flashing. This is because the time taken to draw the shape and then "rub out" is fairly slow therefore the movement is not quite as smooth as with a single character shape. The next section explains a different method which overcomes this problem.

Use 'SHIFT BREAK' to break out of the loop, and CLS40 to return to normal 40 column display. Position the disc with side B uppermost in the drive and then save the program as follows:-

i) Type in SAVE "ALIEN"
ii) Key ENTER

SPRITES

Sprites provide an alternative method of manipulating shapes on the screen. The shapes are defined using grids of 8x8 or 16x16 pixel squares but the SPRITE command in conjunction with the MAG command allows magnification of a particular shape, and more versatile movement.

Sprites 'exist' on what are known as "sprite planes". If we imagine a number of glass sheets stacked up one behind the other, but each so thin that the naked eye perceives them as one, then we have a comparison with sprite planes on the screen. Fig 15.5 illustrates this principle.

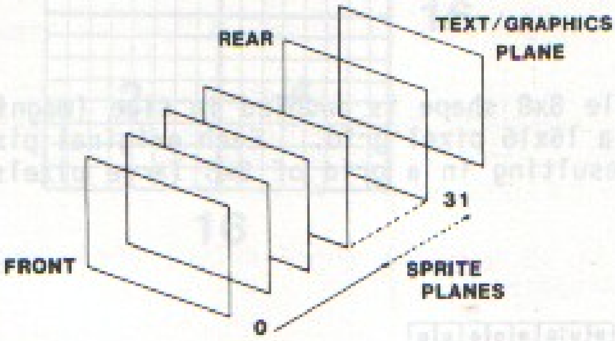


Fig.15.5

There are 32 sprite planes in all and they are numbered from 0 to 31. Shapes drawn on different sprite planes can overlap, therefore, creating a 3 dimensional effect. By careful planning of the sprite plane numbers the effect of one shape moving in front of, or behind, another can be created.

For example a train could appear to move behind a building but in front of some trees by using different sprite planes for each shape accordingly.

The lower the sprite number, the nearer to the eye it becomes. Thus a sprite numbered 4 would appear to be in front of a sprite numbered 6, on the screen. (ie. lower numbered sprites have display priority).

There are four modes of magnification number 0, 1, 2, 3. They are specified by use of the MAG command.

eg. MAG 2

Modes 0 and 1.
Modes 0 and 1 apply to a shape which has been defined on a single 8x8 pixel grid as shown in Fig.15.6. In mode 0 the shape remains as a single 8x8 pixel character.

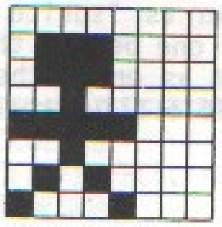


Fig.15.6

In mode 1 the single 8x8 shape is doubled in size (magnified) to occupy an area equivalent to a 16x16 pixel grid. Each original pixel is in fact made 4 times larger, resulting in a grid of 8x8 large pixels as shown in Fig. 15.7.

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	0	1	0	0	0	0	0
1	1	1	1	1	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0

Grid pattern using 0's in the empty squares and 1's in the shaded squares of the shape.

Fig.15.7

Try the following examples which serve to illustrate the use of modes 0 and 1, and compare the results.

```
i) 10 CLS32
   20 SHAPE 140,"00 70 7020 F8 20 50 88"
   30 MAG 0
   40 SPRITE 4,110,100,6,140

ii)10 CLS32
   20 SHAPE 140, "00 70 70 20 F8 20 50 88"
   30 MAG 1
   40 SPRITE 5,150,100,8,140
```

Notice that the first sprite is not cleared and is also affected by the second MAG command (ie. MAG 1).

Sprites cannot be cleared by the usual CLS but must be "turned off" by use of the SPRITE OFF command. Key CLS40 Enter to return to 40 column display. Now type SPRITE OFF and key Enter to clear the screen. Key NEW ENTER to clear the memory.

Modes 2 and 3.
Modes 2 and 3 apply to a shape which is built up from four 8x8 pixel grid shapes to form a single shape on a 16x16 grid (in the same manner as described in building shapes).

In mode 2 the four shapes are printed as a single shape on a 16x16 pixel grid as shown in Fig.15.8

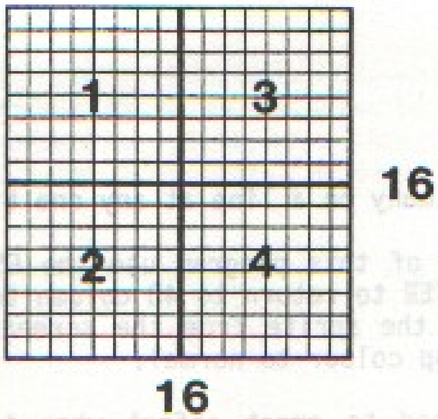


Fig 15.8

When "defining" the complete shape the data for each 8x8 grid should be used with the SHAPE command in the order shown above. Note also that the selected ASCII code number must be 0 or a multiple of 4 (e.g. 156, 144 etc). In mode 3 the shape on this 16x16 grid is doubled in size to occupy an area equivalent to 32x32 pixels.

Work through examples which illustrate the use of modes 2 and 3, and compare the results.

1.
```
10 CLS32
20 SHAPE 152, "00604040414141E17F3F1F0000000000
00303030B0FCB4FCFFFEFC0000000000"
30 MAG 2
40 SPRITE 6,70,100,6,152
```

2.
```
10 CLS32
20 SHAPE 152, "00604040414141E17F3F1F0000000000
00303030B0FCB4FCFFFEFC0000000000"
30 MAG 3
40 SPRITE 7,150,100,6,152
```

Notice again that the first sprite is not cleared and is also affected by the second MAG command (i.e. MAG 3). Type CLS40 and key E to clear the screen then type SPRITE OFF and key ENTER to remove the sprites. The sprite definition, however, remains in memory until a reset or cold start is used.

MOVEMENT OF SPRITES

Sprites can be made to move by setting up a loop which continuously changes the values of the co-ordinates in the SPRITE command.
For vertical movement, change the y co-ordinates.
For horizontal movement, change the x co-ordinates

The example given below illustrates how a FOR-NEXT loop can be used to make a sprite move horizontally across the screen by continuously changing the x co-ordinates in the SPRITE command.

```
10 CLS32
15 BCOL2
20 SHAPE 0,"0060404141F17F3F0000000000000000"
30 SHAPE 2,"003030B0FC54FFFE0000000000000000"
40 MAG 2
50 FOR I=230 TO 1 STEP -1
60 SPRITE 0,I,132,4,0
70 NEXT I
80 GOTO 50
```

Be careful not to have too many on a line at any one time.

Having observed the result of this program use the ESC key to breakout of the loop then key CLS40 ENTER to return to 40 column text mode. Type SPRITE OFF,0 and key E to clear the sprite from the screen. Type BCOL4 and key ENTER to return the backdrop colour to normal.

The KEY command can be used to great effect when typing in experimental lines or altering lines:-

KEY 0, "CLS40:SPRITE OFF: LIST
(Don't forget that the   code is GRAPH & ENTER together).

When function key 0 is pressed, the screen will clear to 40 column mode, the sprites erased and the list put up on the screen. You can, of course, code all the keys for a variety of little jobs when experimenting in BASIC.

Filling Areas Of Colour

Finally just a brief word about the FILL command. This provides a facility for filling enclosed areas of line drawings with colour.

By specifying a single point within the boundries of an enclosed area, then that area can be filled with a selected colour.

The format of the FILL command is as follows:-

FILL x,y,n

x and y indicate the specified point within an area, and n selects the colour (a number in the range of 0 to 15).

Try the following, which illustrates the use of this command.

```
 10 CLS
 20 GCOL10
 30 DRAW 100,40 TO 140,40 TO 140,70 TO 100,70 TO 100,40
 40 DRAW 140,55 TO 170,55
 50 DRAW 100,40 TO 70,55 TO 100,70
 60 ELLIPSE 180,55,10
 70 FILL 120,55,6
 80 FILL 90,55,1
 90 FILL 180,55,12
100 END
```

CHAPTER 16

MACHINE CODE LINKAGE

MACHINE-CODE SUB ROUTINES

Quite often programs (particularly games programs) use a combination of BASIC and machine code.

Machine code is a method of writing programs which "talk" directly to the computer without the necessity of an interpreter (ie. a direct linkage to the machine operating system). Thus execution is faster because the delay introduced when converting from one language to another is no longer present.

BASIC can be used to create the main core of a program and then subroutines written in machine code can be linked into it. This is quite useful when a large amount of graphics is involed, the faster machine code sections producing spectacular effects on the screen.

Machine code subroutines are linked into a BASIC program by use of the CALL command which has the following format.

```
  CALL I
```

I refers to the "start address" of a particular routine required (ie. the location in the computers memory store where the particular machine code subroutine starts). It can be "called up" into use at any point in the main program. The address given in I can be in either decimal or Hexadecimal format.

Example: CALL &0F00

This will access a machine-code subroutine stored in memory starting at the location &0F00

Obviously machine-code routines need to be placed into memory before they can be accessed by the CALL command.

Some routines are already in store as an integral part of the machine "operating system" and the BASIC. For example the routine which handles error messages is found at location &06CF.

Generally machine-code subroutines accessed by the CALL command are those which have been created by the user for a particular program and placed into memory.

Program listings found in magazines and other publications often illustrate the use of machine-code routines but the writing of machine-code programs is beyond the scope of this book. The user must therefore research other publications for further information on the subject.

Areas in memory can be reserved for machine-code subroutines by the facilities offered with the CLEAR command, thus avoiding any clash with BASIC programs when being stored.

MEMORY LOCATIONS AND CONTENTS

Memory locations are listed in Hexadecimal in ascending order starting from 0 for identification purposes. Each location is referred to by a four digit hexadecimal number which is known as the address for the data it contains.

```
    e.g.  &0013
          &012E        Memory Locations
          &215D
```

Each memory location can hold 1 BYTE of data represented by a two digit hexadecimal number.

```
    e.g.  &34
          &3F          Two digit HEX numbers
          &A2
```

(Remember that the '&' symbol is used as a prefix to denote Hexadecimal numbers in documentation).

We can imagine these memory locations to be stacked vertically on top of one another. For example the spaces between the rungs of a ladder could be envisaged as memory locations into which data can be placed.

To access the memory locations directly from BASIC the two commands PEEK and POKE can be used.

The PEEK command allows the user to examine the contents of a particular memory location (or address) by using it as follows:-

```
   PRINT PEEK(I)
```

Where I indicates the particular memory location

```
   e.g.  PRINT PEEK(&41FD)
```

This will return an integer in the range of 0 to 255, according to the contents of the location specified (in this instace &41FD). The integer will be displayed on the screen.

The POKE command allows the user to insert data to a particular memory location (or address) as follows.

```
   POKE I,E1,E2,E3,.....,En
```

This will place the values of the expressions E1, to En into memory locations starting at the location given by I.

```
   e.g. POKE &5100,&77,&34,&CD,&3A
```

This will place the values &77,&34,&CD,&3A into locations &5100,&5101,&5102,&5103 respectively. Thus:-

```
&5100 contains &77 (ie. 119 decimal)
&5101 contains &34 (ie.  52 decimal)
&5102 contains &CD (ie. 205 decimal)
&5103 contains &3A (ie.  58 decimal)
```

Try entering these values and then use PEEK to check the contents of the locations.

Care should be taken if using this facility to avoid corruption of any essential utilities programs which would "crash" the computer.

Some selected memory locations contain data for various functions relating to the operation of the computer. It is the corruption of these by careless use of the POKE command which should be avoided. e.g. location &011E - contains the data which determines the number of characters which can be used in a single line of input. This is referred to as the length of input buffer (BUFLEN) and is 126 characters (ie. &7E).

These selected locations are known as SCRATCH-PAD LOCATIONS. Some scratch-pad locations are labelled with a "pointer" (PTR). The "pointers" are numbered from 1 to 24 and can be used to access those particular locations.

Thus the PTR command can be used as an alternative to POKE in order to change the contents of any of those particular 24 locations. The PTR command has the following format.

```
PTR N,E
```

"N" indicates the pointer number for a particular location, and "E" is the two digit HEXNUMBER representing the new data to be inserted at that location.

```
e.g. PTR13,&3C
```

"PTR 13" refers to the location which contains the data relating to the length of input buffer (ie. number of characters allowed in one single line of input). Normally the length is 126 characters (maximum allowed) but the data inserted above, changes the value to 60 characters.

Again care should be taken to avoid corruption of data necessary for the operation of the computer.

MACHINE-CODE PROGRAMS

Quite often machine-code programs appear in magazines and other publications, for users to type into their own computers. The presentation might consist of the code listing in hexadecimal giving the memory locations and their corresponding bytes of data to be typed in. A section of one such listing is shown below.

LOCATION	DATA
0B07	F5
0B08	79
0B09	D309
0B0B	78
0B0C	CBF7
0B0E	D309
0B10	F1
0B11	D308

The presentation might consist of the assembly source code listing which uses machine-code mnemonics similar to shown below.

```
LINE NUMBERS   SOURCE CODE  LISTING

    1563       PUSH      AF
    1564        LD       A,C
    1565       OUT    (H'09'),A
    1566        LD       A,B
    1567       SET       6,A
    1568       OUT    (H'09'),A
    1569       POP       AF
    1570       OUT    (H'08'),A
    1571       RET
```

This type of listing is the equivalent of the program language before conversion to "machine-code".

It is necessary to use an "assembler" to convert the source coding into machine-code data. Assemblers are special programs written specifically for this purpose and are similar in function to an interpreter when using BASIC.

The following is one method which can be used to copy a machine code program from a listing into the computer.

It does not involve the use of an "assembler". Memory locations and bytes of data are typed in from a code listing similar to that in example '1' above. The procedure is as follows:-

a) Set up the computer to work in MOS (Machine Operating System)

b) Type M immediately followed by the first memory location of the listing and then ENTER. This invokes the "modify" command under MOS which then allows data to be entered into consecutive memory locations.

c) The first memory location appears on the screen ready for the data to be typed in as shown in the listing. As the bytes of data are entered the next consecutive memory location automatically appears on the screen ready for the corresponding data to be typed in.

d) At the end of the listing type a full stop (.) and key ENTER so as to exit the sequence.

e) To run the program type G followed by the first memory location of the listing and then key ENTER. This instructs the computer to go to the program starting at the address given, and execute.

If the source-code listing only is given, as in the second example above, then an assembler becomes necessary. Without the assembler the SOURCE-CODE cannot be typed in. Assemblers available for EINSTEIN 256 are indicated in the literature enclosed with the machine.

```
ADDRESS/HEX         DATA         SOURCE CODE

         CODE       VALUE/S

  8000  060A                     LD   B,0AH
  8002  C5          AGAIN:       PUSH BC
  8003  CF                       RST  8
  8004  BE                       DB   0BEH
  8005  060C                     LD   B,0CH
  8007  C5           LOOP        :    PUSH  BC
  8008  CF                       RST  8
  8009  A6                       DB   0A6H
  800A  C1                       POP  BC
  800B  10FA                     DHNZ LOOP
  800D  CF                       RST  8
  800E  CF                       DB   0CFH
  800F  2A2A2A2A                 DB   "****** This could
  8023  2A2A2054
  8017  68697320
  801B  636F756C
  801F  64
  8020  20626520                 DB   " be your message"
  8024  796F7572
  8028  206D6573
  802C  73616765
  8030  202A2A2A                 DB   " ******"
  8034  2A2A2A
  8037  80                       DB   80H
  8038  C1                       POP  BC
  8039  10C7                     DJNZ AGAIN
  803B  C30001                   JP   0100H
                                 END
```

NOTE: Start at 8000 in MOS and type in the Hex code: the address will automatically increment, so don't worry about going out of sequence.

TO SAVE MACHINE-CODE PROGRAMS

Machine Code programs should normally start at location &0100 in which case they can be saved using DOS commands without having to load BASIC (saved as .COM files in this instance).

If a machine code program starts at an address different from &0100 then it should be a location above &5000 and end below location &E000. This is to avoid any overwriting of BASIC and DOS which might cause corruption. Programs within this catageory can be saved using commands in BASIC. (saved as .OBJ files)

To save a program which begins at &0100:-

a) Determine the number of blocks of memory which the program occupies using the following calculation with the computer working in BASIC.

(&XXXX - &0100)/256 = NUMBER OF BLOCKS

where xxxx is the end address of the program.

If necessary the answer given (No. of blocks) should be rounded up to the next whole number.

b) Go into DOS (CTRL/BREAK or type DOS from BASIC)

c) Use the SAVE command in the following format.

SAVE N Name.COM

where N is the number of blocks, Name is the program title as selected by the user, .COM is the file type for machine-code programs saved in this manner.

e.g. SAVE 4 MAK.COM

This will save the program MAK, which occupies 4 BLOCKS, as a .COM file on disc.

BIBLIOGRAPHY

1. "Einstein User" magazine.
   Publisher: Tatung (UK) Ltd., Stafford Park 10,
   Telford, Shropshire, TF3 3AB, England

2. "The Brain"
   Publisher: Syntaxsoft, The Northbridge Centre,
   Elm Street, Burnley, Lancashire, England.

3. "Einstein News"
   Publisher: Screens Microcomputers, Main Avenue, Moor Park,
   Northwood, Middlesex, England.

4. "Einstein Primer"
   Publisher: Screensoft, Main Avenue, Moor Park, Northwood,
   Middlesex, England.

5. "Einstein Assembly Language Course"
   Publisher: Glentop Publishers Ltd., Standfast House,
   Bath Place, High Street, Barnet,
   Hertfordshire EN5 5XE, England.

6. "Relatively BASIC"
   Publisher: Solo Software, 95B Blackpole Trading Estate West,
   Worcester.

7. UK Einstein Users Group (UKEUG)
   c/o Keith Stokes, "Hillcroft", Codmore Hill,
   Pulborough, West Sussex, RH2 1BQ, England.

8. AVON Einstein User Group,
   74, Great Hayles Road, Crowndale Estate, Witchurch,
   BRISTOL, Avon, BS14 0SJ (Membership Free)

9. Software Interest Group
   c/o P.D. Sig, 138 Holtye Road, East Grinstead, Sussex, RH19 3EA.

GLOSSARY OF TERMS

Boot:
To start up a computer. Abbreviation for executing bootstrap program.

Bootstrap Program:
The part of DOS in ROM is a good example. It contains just enough information to load the rest of the DOS into RAM. A program kept in memory which uses certain preliminary instructions to load or read other programs or data.

Branch:
A part of a computer program where a choice is made between alternative routes - the decision maker in computing.

Breakpoint:
A point in a computer program where normal execution is interrupted to enable visual checking, allowing debugging, or to obtain print-outs.

Buffer:
An area of computer memory used for temporary storage of input or output data until a particular device is ready for it.

Bug:
A defect or mistake in a computer program.

Bus:
A set of electrical pathways or connectors inside a computer.

Byte:
'By eight'. Usually means a group of eight bits.

CAD:
Stands for 'computer aided design'.

CAL:
Stands for 'computer aided learning'.

CAM:
Stands for 'computer aided manufacture'.

Central processing unit: (CPU)
The 'brain' of the computer where all parts of the computer system are linked together and where the calculations and manipulation of data take place. It usually contains the Arithmetic Logic Unit (ALU) and several other stages once needing several chips.

Characters:
The expression used for numerals, letters & symbols which a computer can print, or display on a screen.

Chip:
A single device containing many transistors and other components formed on the surface of a piece of silicon.

COBOL: (Common Business Oriented Language)
A high level language usually used for business applications.

Command:
A direct instruction to the computer.

Compatible:
Two computers are said to be compatible if a program written on one will run on the other without modification.

Compiler:
A program which converts like an applications program written in a high level language into the machine code version which the computer needs to be able to run it without any interpretation.

Computer:
The computer is a device which can process information according to instructions given to it, and in this way perform useful or entertaining tasks.

Concatenation:
The linking of two or more "strings" to make a single longer "string".

Constant:
An item of data used, but not altered by a program. The data can be either a numeric constant or a string constant.

Converter analogue to digital (or vice-versa):
A device for converting anologue information in the form of a continuously varying electrical voltage from some kind of electrical sensor into the digital form which the computer can cope with - or vice-versa.

CP/M: (Control Program for Microcomputers)
This is an operating system most commonly used in microcomputers. All micros which use a CP/M system can use the same software.

Crash:
A computer is said to 'crash' when a program which is running cannot be completed and cannot be restarted.

Cursor:
Some way of marking the screen with the position at which the next character typed in at the keyboard will appear. It takes several forms, from a chevron, , to a flashing square block.

Cycle Time:
The time required by a computer to read from or write into the system memory. Cycle time is often used as a measure of computer performance, since this is a measure of the time required to fetch an instruction.

Daisy wheel printer:
A printer which makes use of a plastic disc around the edge of which is a set of print characters. The wheel rotates at speed until the required character is brought before a hammer which strikes it against a ribbon. One wheel can easily be replaced with another having a different typeface.

Data:
Loosely, means 'information' which a computer program can deal with. Data can be in the form of numbers or characters.

Database:
An organised collection of files of information to which the computer has access. If many people have access to it through different terminals it might then be called a data bank.

De-bugging:
The business of testing a program and then changing it to get rid of 'bugs' or faults.

Default:
A standard characteristic or value which the computer assumes if certain specifications are omitted within a statement or program.

Device:
A particular unit or processing equipment in a computer system external to the Central Processing Unit and usually in the form of a peripheral.

Dialect:
A version of particular computer language e.g. EBASIC, TATUNG/CRYSTAL BASIC, BBC BASIC, RML BASIC - all are different dialects of BASIC with some things in common.

Digital:
To do with numbers, c.f. 'analogue'.

Digitizer:
A device which converts "continuous information" into the numbers a computer can understand. (e.g. a graphics tablet).

Directory:
An area of storage on a disc indicating the contents and their locations on the disc (similar to contents listing in a book).

Disc or 'floppy disc':
A flat magnetic disc on which programs and data may be stored and retrieved quickly - much faster than cassette tape.

Documentation:

a) The collation of documents or the information recorded in documents.
b) A collection of documents or information on a particular subject.

Dot matrix printer:
A printer using a series of electrically 'hammered' moving pins to create characters composed of a pattern of dots.

EPROM: (Erasable, Programmable Read-Only Memory)
A chip which can be fed with a program or data and which will hold it until it is erased (usually by exposing the surface of the chip to ultravoilet light). After that it can be re-programmed. (See PROM. ROM).

Execute:
To run a program or perform the task specified.

Exponent:
A number indicating the power to which a number or 'expression' is to be raised.

Expression:
Representation of a mathematical or logical statement by symbols.

File:
An organised collection of information - e.g. computer programs.

Firmware:
A program permanently held in a 'read only memory' chip in a computer. The term usually refers to the programs which manage the internal operations of the computer rather than applications programs.

Flow chart:
A diagram on paper showing the sequence of events and choices which need to be made in the solution of a problem - usually (though not exclusively) relating to a computer program.

FORTRAN: (FORmula TRANslation)
A high level language mainly for scientific and mathematical use.

Garbage:
Meaningless or unwanted data coming from the computer.

Graphics:
The overall term meaning the appearance of pictures or diagrams on the screen as opposed to letters and numbers.

Handshaking:
A 'dialogue' between two computers or a computer and a 'peripheral device' - like a printer - which establishes that a message is passed between them to their mutual satisfaction.

Hardware:
The physical bits and pieces of the computer - as opposed to the 'software' or the programs.

Hard copy:
Tangible and permanent output from a computer, on paper.

Hexadecimal or 'HEX':
A method of counting in a base of 16. Used for programming in low level languages. One 'byte' can be represented by two hexadecimal symbols.

High level language:
A programming language where the programmer uses instructions which are close to his ordinary familiar language rather than machine code. In effect, the higher the 'level' of the language the nearer it is to ordinary language and the easier it is for the uninitiated to understand.

Housekeeping:
Refers to instructions, usually at the beginning of a program which help to organise the tasks involved but do not contribute directly to the solution of a problem (e.g. clearing fields to zero).

Integer:
A number which does not contain a fractional part.

Integrated circuit (IC):
The circuits combined together on the surface of a silicon chip.

Interactive:
A way of operating where the user is in direct and continual two way communication with the computer, maybe answering its questions and receiving its reactions to the answer.

Input:
The route whereby information gets into the computer or the putting in of information by the operator (e.g. from a keyboard).

Instructions:
A computer program consists of a series of instructions, often used interchangably with 'commands'.

Interface:
The boundary between two parts of a computer system. Often the boundary consists of a piece of electronic circuitry. Also means to make one part of a computer system run smoothly with another.

Interpreter
A program which translates the keywords in a high level language program, line by line, into machine code which the processor can cope with.

Iteration:
Refers to the technique of repeating a group of program statements and is one repetition of such a group (ie. one pass of a loop).

Keyboard:
One form of input device for a computer. Keyboards are usually 'Alphanumeric' (q.v.) but also contain special keys which perform particular functions on the computer.

Keywords:
Words in the vocabulary of a high level language which have a special meaning to the computer.

Kilobyte:
Approximately a thousand bytes (actually it is 2 to the power of 10, which is 1024) e.g. 64k memory is 64 thousand bytes of memory.

Language:
A computer 'language' is an organised way of communicating with a computer using precisely defined instructions in the form of 'English like' statements.

LED: (Light Emitting Diode)
An electronic component which emits light when excited by an electric current.

Listing:
A list of items (program instructions, data, etc.) printed on a peripheral device by the computer. Usually refers to the 'listing' of a program.

Load:
To enter programs or data into storage or working registers.

Location:
A place in the computer's memory where information is to be stored (see address).

Loop:
A group of consecutive program lines which are repeatedly performed, usually a specified number of times.

Low level language:
(See machine code)

Machine code:
The pattern of 'Os' and 'ls' which the computer actually understands. It is the lowest level of language for a programmer to work in and all high level programs are converted into machine code instructions automatically when they run. Programs written directly in a low level language run faster than those in high level language.

Memory:
A computer's memory is a device or series of devices capable of storing information temporarily or permanently in the form of patterns of binary 'ls' and 'Os'. The computer then 'reads' information from the memory or in some cases also 'writes' information into it when it operates.
1. Internal Memory - ROM and RAM.
2. External Memory - Magnetic tape or disc on which binary information is stored and 'retreived' by the computer as required. The information is not lost when the computer is switched off.

Menu-driven programs:

Programs which present the operator with a list of choices at any particular time and these are displayed on the screen for him to choose from. Each choice leads down a different branch of the program.

Micro:
Has two meanings - (i) 'small' - as in microcomputer' - and (ii) a millionth of something - e.g. microsecond, a millionth of a second.

Microcomputer:
A small computer system built round a microprocessor but having all the necessary bits and pieces (peripherals and memory) to link with the outside world and store information.

Microelectronics:
The use of electrical devices in which many different components are formed together (integrated) into microscopically small circuits on the surface of single 'chips' (usually of silicon).

Microprocessor:
A microprocessor is the central chip containing the control unit for the computer.

Minicomputer:
A medium sized computer of the kind which might be used by a medium sized company to keep its records, work out its payroll, stock control, etc., Midway between a 'micro' and a 'mainframe' computer.

MOS: (Machine Operating System.)
That system which controls the internal function of the computer.

MP/M: (Multi-user CP/M)
An operating system like a CP/M except that many people can use it simultaneously.

Multi-user:
A computer system where a group of terminals are connected to a single microcomputer so several people can use the micro simultaneously.

Network:
A system where a number of computers, terminals and other components (like printers and disc drives) can be linked together electronically - sometimes over some distance.

Numeric:
To do with numbers.

Operand:
The data upon which a machine-code instruction operates.

Operating System:
The software program residing permanently inside the computer which supervises the running of applications programs and controls the operations of the various input and output devices like the video display unit, keyboard etc.

Operator:
This is a symbol used within program instructions to indicate numeric or relational comparisons. There are sets of numeric and relational operators.

Output:
Information which a computer sends out to a screen or a printer or to a backing memory store.

Paddle:
Another name for a joystick control - e.g. for a T.V. game.

Parallel:
When electrical patterns of all 8 bits in a byte travel simultaneously along separate wires they are said to be in 'parallel'.

Pascal:
A high level language preferred by many to BASIC for general programming work.

PCB (printed circuit board):
The plastic board into which the computer's various electronic components are soldered. These are linked by thin inter-connecting wires printed on its surface.

Peripherals:
Bits and pieces of a computer system which connect in different ways with the central processor and memory and which form its input and output devices. Peripherals include printers, disc drives, joy sticks, graphics tables, light pens etc.

Pixels:
The smallest dot on a screen which the computer can display.

Port:
A place where electrical connection can be made with the central processor in the computer.

Portability:
Programs are portable if they run on different computer systems.

Processor:
(See central processing unit)

Program:
A series of instructions which the computer carries out in sequence.

PROM (Programmable Read Only Memory):
A chip which can be programmed by the user. Once programmed, its contents are 'non-volatile'. (See also ROM, EPROM).

RAM (Random Access Memory):
Memory into which information can be put (written) and from which it can instantly be copied (read) no matter where it is in the memory. RAM is the 'working memory' of the computer into which applications programs can be loaded from outside and then run. Sometimes called a read/write memory.

Real time:
A computer system is operating in 'real time' if the processing of information fed in takes place instantly.

Reserved words:
A special word with a predefined meaning, used in a programming language. Reserved words must be spelled correctly, appear in the proper order in a statement or command, and cannot be used as a 'variable' name.

ROM (Read Only Memory):
A memory circuit in which the information stored is 'built into' the chip when it is made and which cannot subsequently be changed by the user. Information can be copied from ROM but it cannot be written there, hence the name read only memory. Another name for read only memory is 'firmware' since this implies software which is permanent or firmly in place, on the chip.

Robot:
A computer-controlled device which is fitted with sensors and activating mechanisms. The sensors receive information about the surrounding environment, send it to a computer which then decides on the basis of its program how the mechanical parts should respond - e.g. to pick something up or to move about. Some robots can be programmed to improve their performance as a result of their experience, (see artificial intelligence).

Scanning:
This word usually refers to the very rapid examination of every item in a computer's 'list' of data to see if some condition is met.

Scrolling:
The automatic upward movement of the information on a screen to allow new information to be displayed at the bottom of the screen. Sideways scrolling is often used in graphics to scan a particular scene.

Serial:
When electrical patterns of bits travel one after the other down a wire in a computer they are said to be a 'serial' bit stream - as opposed to a 'parallel' bit stream (q.v.)

Silicon:
The chemical element which is used as the basis for the increasingly more complex integrated electronic circuits which have been responsible for the 'microelectronics revolution'. Silicon is present in sand (which is silicon dioxide). It has odd electrical properties, sometimes conducting electricity and sometimes not, depending, for example, on what other substances are mixed with it in minute quantities.

Software:
The general term which refers to all computer programs which can be run on computer hardware. A distinction can be made between the programs responsible for the running of the computer - its internal 'housekeeping' and operating systems and so on - and 'applications programs'. Ultimately, all software consists of patterns of binary information which give the computer instructions.

Statement:
A numbered line of a computer program.

Storage:
Another word for memory - a place where information can be kept in a form which is accessible to the computer.

String:
A set of characters one after the other which the computer can deal with, usually enclosed by quote marks and a distinction is usually made between strings and numbers. Spaces contained within a string count as characters.

Subroutine:
A self-contained part of a program that can be called up and run by other parts of the program. It is usually written to perform a task that is needed frequently by the main program.

Syntax:
The structure or 'grammar' of program statements and commands.

Systems analyst:
A person trained in the analysis of complex physical or organisational problems and able to offer solutions calling on a range of skills, one of which may involve the use of the computer and computer programming.

Tape:
Magnetic tape or punched paper tape can both be used to store computer programs or data. Neither is as fast as disc systems when it comes to finding the information stored.

Telesoftware:
Computer programs sent by telephone line or by television as part of the teletext signal. With a suitable decoder the computer program can be entered directly into the memory of a computer and then 'run'.

Terminal:
A peripheral device usually consisting of a keyboard and a screen which can link into a computer network sometimes using a telephone as the link.

Time-sharing:
A way of sharing out powerful computer facilites between a number of users who want those facilities apparently at the same time on a number of separate terminals. Each user gets the impression that he has sole use of the computer.

Unary Operation:
The processing operation carried out on one operand (eg. negation)

User:
The individual person using a machine.

Variable:
An electronic 'box' or pigeon hole into which data can be put and subsequently be changed. A variable has a name and a value. The name does not change but the value can. Variables can also be 'numeric' or 'string' variables.

VDU (Visual Display Unit):
A television-like screen on which the output of the computer can be displayed. The VDU is the most usual 'output peripheral device' of the computer.

Voice recognition:
The ability of a computer to match the pattern of signals coming into it from a microphone with stored 'templates' held in its electronic memory and thus recognise words.

Voice synthesis:
The ability of the computer to use stored patterns of sounds within its memory to assemble words which can be played through a loudspeaker.

Volatile memory:
Memory in which information is lost when the power is switched off.

Wand:
A pen-like device able to read optically coded labels (see bar codes).

Winchester disc:
A form of back-up storage for a computer. It consists of a rigid magnetic disc in a sealed container scanned by a head which does not quite touch the disc, therefore not wearing it out.

Word:
When a computer operates it deals with groups of bits at a time. The minimum number of bits which the central processor handles at any one moment is called a 'word'.

Word processing:
An office procedure for electronically storing, editing and manipulating text using an electronic keyboard, computer and printer. The text is recorded on a magnetic medium rather than on paper, except for the final 'print-out'.

Write protect:
To physically prevent write cycles from accessing a particular device or memory location as a safeguard against accidental overwriting of data.

APPENDICES

APPENDIX A

ERROR MESSAGES IN EBASIC

	CODE		
Error Messages	HEX	DECIMAL	
Break	00	0	Interruption from Keyboard!
Next	01	1	NEXT statement found without corresponding FOR
Syntax	02	2	Typing error in line
Return	03	3	RETURN or POP found without corresponding GOSUB
Data	04	4	No more DATA statements for READ
Qty	05	5	Number specified outside allowable range
Ovfl	06	6	Number too large
Mem Full	07	7	No more memory left
Branch	08	8	Attempt to refer to non-existent line
Range	09	9	Outside dimensions specified for array
Dimension	0A	10	DIM encountered for already dimensioned array
Division	0B	11	Divide by Zero!
Stack Full	0C	12	No more stack for FOR, GOSUB or expressions
Type	0D	13	String given when number expected or vice versa
Cmd	0E	14	Reserved Word not defined in system
Str Ovfl	0F	15	String expression too long
Str Complex	10	16	String expression too complex - Split it up!
Cont	11	17	Cannot continue after error or program mod.
Fn Defn	12	18	FN user function not defined by a previous DEF
Operand	13	19	Operand expected in expression
Bad data	14	20	Disc checksum error
End of Text	15	21	End of File encountered
File	16	22	FDESC not defined (or used by another file)
Drive Select	17	23	Drive selected not available in system
File Type	18	24	File of incorrect type
No File	19	25	File not found
File Exists	1A	26	File already present
File Locked	1B	27	File has been locked
Disc Locked	1C	28	Disc is in 'Read Only' mode
Disc Seek	1D	29	Attempt to seek beyond end of disc
Disc Full	1E	30	No space for file contents
Dir Full	1F	31	Too many files in Directory

When an error occurs, either in direct mode or from within a program, execution halts and a message will be output. Unless error trap statements are used (as described in the previous section) the message will appear in the following format:-

```
For direct mode   - "Description of Error" Error
For deferred mode - "Description of Error" Error in
                    "Line No."
```

The "Description of Error" is a statement which indicates the type of error which has been made. The "Line No." in deferred mode is the program line number in which the error occurred.

```
Examples:
  Direct Mode     - Branch Error
  Deferred Mode   - Branch Error in 75
```

Details of the various error messages which might be output are given below.

Bad Data

A checksum error has been detected while loading or verifying a program/data file from disc. i.e. disc has been corrupted or memory contents do not match file (in the case of verify).

ACTION:- Retry with backup disc!

Branch

Reference has been made to a non-existent line number i.e. an attempt to use a line which is not included in a program.

ACTION:- Check program listing and make the necessary corrections in relation to the particular line/line number.

Cmd (Command)

An attempt has been made to reference a reserved word which does not exist. This often happens when programs are adapted from other systems.

ACTION:- Trace the offending word and convert or re-structure to comply with current system.

Cont (Continue)

An attempt has been made to continue a program, (using the CONT command after a specified interrupt) when either:-

a) an error occurred or

b) alterations have been made within the program.

ACTION:- Check for errors which may have been introducted in any modifications and rectify as necessary.

Data

A READ statement has been used but insufficient data has been presented in the corresponding DATA statement.

ACTION:- Check the corresponding DATA statement and rectify as necessary.

Dimension

An attempt has been made to redimension an array. Arrays may only be dimensioned once within a program, including those arrays of under 10 elements which have not been formally dimensioned.

ACTION:- Check the appropriate statements, processes, and corresponding logic sequences. Rectify as necessary.

Division

An attempt has been made to divide a number by zero.

ACTION:- Check the appropriate expression and rectify as necessary.

Drive Select

A disc drive has been selected which is not available on the system.

ACTION:-
i) Check the appropriate select statement and rectify as necessary.
ii) Check that the particular drive is included in the system.

End of Text

Either:-
a) An end-of-file marker has been encountered in a data file or
b) The last block of a file has been read.

This error may be controlled within the system by use of the ON EOF command.

File

Either:-
a) An attempt has been made to open a file which is already open or
b) An attempt has been made to read from, or write to, a file which is not open.

ACTION:- Check the logical sequence of the operations involved and rectify as necesaary.

File Type

A particular file type has been specified when in fact another type was expected.

ACTION:- Check the appropriate file types and rectify as necessary for them to correspond as required.

Fn Defn (Function Definition)

Either:-
a) A user-defined function has been used without first defining it or
b) CALL has been used as a function without first setting up the USRLOC.

ACTION:-
i) Check definition of appropriate user defined functions or
ii) Check that USRLOC has been correctly set up. Rectify as necessary.

Mem Full

An attempt has been made to use a command which would need more memory than is available.

ACTION:- Either re-structure or eliminate the command so as to comply with current memory space available.

Next

A NEXT has been used which does not correspond to a FOR statement.

ACTION:- Check the loop structure and either:-

a) eliminate the NEXT or
b) insert the required FOR statement

Operand

Operand is missing after an operator.

Example: PRINT 6.2*7+

ACTION: Check the expression and rectify as necessary.

Ovfl (Overflow)

Numeric overflow from a calculation, i.e. number is outside the normal range for numeric variables.

ACTION:- Check the appropriate expressions and rectify as necessary.

Qty (Quantity)

A particular parameter in an array, command, or function, falls outside the declared range.

ACTION:-
i) Check parameter values with the individual commands and functions concerned to determine the maximum and minimum values allowed.
ii) Rectify values or re-structure as necessary.

Range

An attempt has been made to access an element of an array which does not fall within the limits of the delcared dimensions.

ACTION:- Check the dimensions of the array and rectify to accommodate the particular element as required.

Return

A RETURN has been used without a corresponding GOSUB.

ACTION:- Check the logical sequence of the processes involved and either:-
i) re-structure the sequence or
ii) insert a GOSUB.
depending on the particular requirements.

Stack Full

This will make reference to one or more of the following situations:-

a) FOR loops
b) GOSUBs
c) Parentheses in Expressions
d) FILL

The error message will be displayed if any of the above conditions have been NESTED too deeply, causing a "stack overflow".

ACTION:- Re-structure so as to reduce the nesting to an acceptable level and thereby rectify the situation. In the case of FILL check that the area in question is fully enclosed.

Str Ovfl (String Overflow)

A string has been included which exceeds the maximum number of characters allowed (255).

ACTION:- Check the offending string and either:-
i) re-structure the string, reducing the number of characters or
ii) transpose the single string into two or more separate strings with less then 255 characters per string.

Str Complex (String Complex)

A string expression has been used which is too long or complex.

ACTION:- Break the expression into smaller sections.

Syntax

This indicates that either a typing error has been made or a particular statement has been constructed incorrectly.

ACTION:- Check the appropriate sections of data and carry out the following:-
i) correct typing errors.
ii) correct statement construction errors.

Type

An incorrect data type has been used. i.e. a "numeric" quantity has been used when a "string" type was expected, or vice versa.

Dir Full (Directory Full)

This indicates the directory section of a disc is full.

ACTION:- If further information is to be placed in the directory then it must be at the expense of some of the existing data (either by deletion or overwriting).

Disc Full

This indicates that there is no more space available on a particular disc.

ACTION:- Further information can only be placed on the disc at the expense of existing data (either by deletion or overwriting).

Disc Locked

An attempt has been made to write to a disc which does not match the map of the disc held in the computer memory.

Example: This error usually occurs when a disc is changed and an attempt is made to save a file on this disc.

ACTION:- Execute a DRIVE n before using a SAVE command, where n is the drive number containing the new disc.

Disc Seek

An attempt has been made to access a particular sector which is not on the disc. This quite often happens with "random-access" files when the record required is off the disc.

ACTION:- Check the appropriate data and rectify as necessary.

File Exists

An attempt has been made to use a name for a file which is already in existance (usually with the REN statement).

ACTION:- Check file names and rectify as necessary.

File Locked

An attempt has been made tu erase, or write to, a file which has been "locked".

ACTION:- Check that the correct file has been referenced and rectify as necessary.

No File

A particular file cannot be found in a directory.

ACTION:-
i) Check that the file name has been constructed correctly and rectify as necessary.
ii) Check documentation to determine whether or not the file has been deleted previously.

Shape Defn (Shape Definition)

The number of (hexadecimal) characters entered in the shape definition string are not a multiple of 2. Two characters must be entered for each row of the shape being formed.

ACTION:- Edit SHAPE string by adding zero's or removing any extra characters to give 2 characters for each row of the shape.

Key Defn (Programmable Function Key Definition)

A function key string has been declared beyond its legal length.

ACTION:- Redefine the key concerned with a shorter string.

Write Protect

An attempt has been made to write to a disc which is "write protected" (on a "read only disc").

ACTION:- Check that the correct disc has been used and rectify as necessary, or de-activate the write protect tabs on the disc cassette.

APPENDIX B

CODE TABLE - EINSTEIN (ISO 646) CHARACTER SET

					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b4	b3	b2	b1	COL ROW	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NUL	SET 80 COL CLEAR SCREEN	SP	0	@	P	£	p		8o	■	□	□	▥	▧	▤
0	0	0	1	1	SCREEN DUMP	CURSOR ON	!	1	A	Q	a	q	SD	CO	□	□	□	□	□	□
0	0	1	0	2	STX	PRINTER ON	"	2	B	R	b	r		PO	□	□	□	□	□	⊞
0	0	1	1	3	ETX	PRINTER OFF	#	3	C	S	c	s		PF	□	□	□	□	□	□
0	1	0	0	4	CURSOR RIGHT	CURSOR OFF	$	4	D	T	d	t	⇒	CF	□	□	□	□	□	□
0	1	0	1	5	ENQ	DELETE TO END OF LINE	%	5	E	U	e	u		DL	□	□	□	□	□	□
0	1	1	0	6	DELETE CHAR FROM RIGHT	ERASE TO END OF SCREEN	&	6	F	V	f	v	→|	DS	□	□	□	□	□	□
0	1	1	1	7	BEEP	INV. VIDEO	'	7	G	W	g	w	bL		□	□	□	□	□	□
1	0	0	0	8	CURSOR LEFT	DELETE WHOLE LINE	(	8	H	X	h	x	⇐	DR	□	□	□	□	□	□
1	0	0	1	9	HORIZONTAL TAB	DEL	)	9	I	Y	l	y	hT	|←	□	□	□	□	□	□
1	0	1	0	A	LINE FEED	INS	*	:	J	Z	j	z	⇓	∴	□	□	▚	□	▚	□
1	0	1	1	B	VERTICAL TAB	ESC	+	;	K	←	k	¼	⇑	EC	□	□	▩	▦	▩	▩
1	1	0	0	C	CLEAR SCREEN & CURSOR HOME (FORM FEED)	FS	,	<	L	½	l	||	FF		□	□	□	□	□	⧅
1	1	0	1	D	CARRIAGE RETURN	DIRECT CURSOR ADDRESS	—	=	M	→	m	¾	CR		□	□	□	□	□	□
1	1	1	0	E	SET 40 COL CLEAR SCREEN	CURSOR HOME	.	>	N	↑	n	÷	4o	↖	□	□	□	□	□	□
1	1	1	1	F	SET 32 COL CLEAR SCREEN	US	/	?	O	_	o	■	32		□	□	☒	▨	☒	■

CODE TABLE - ASCII CHARACTER SET

					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	COL ROW	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NUL	SET 80 COL CLEAR SCREEN	SP	0	@	P	£	p		$^{S}_{O}$	■	□	□	▥	▧	☰
0	0	0	1	1	SCREEN DUMP	CURSOR ON	!	1	A	Q	a	q	$^{S}_{D}$	$^{C}_{O}$	□	□	◨	□	◧	□
0	0	1	0	2	STX	PRINTER ON	”	2	B	R	b	r		$^{P}_{O}$	□	□	□	□	□	⊞
0	0	1	1	3	ETX	PRINTER OFF	#	3	C	S	c	s		$^{P}_{F}$	⊟	⊟	□	◨	□	◧
0	1	0	0	4	CURSOR RIGHT	CURSOR OFF	$	4	D	T	d	t	⇒	$^{C}_{F}$	⊟	⊟	◨	□	◧	□
0	1	0	1	5	ENQ	DELETE TO END OF LINE	%	5	E	U	e	u		$^{D}_{L}$	□	□	□	□	□	□
0	1	1	0	6	DELETE CHAR FROM RIGHT	ERASE TO END OF SCREEN	&	6	F	V	f	v	⇥	$^{D}_{S}$	□	□	□	□	□	□
0	1	1	1	7	BEEP	INV. VIDEO	’	7	G	W	g	w	$^{b}_{L}$		□	□	▄	□	▀	□
1	0	0	0	8	CURSOR LEFT	DELETE WHOLE LINE	(	8	H	X	h	x	⇐	$^{D}_{R}$	□	□	▄	□	▀	□
1	0	0	1	9	HORIZONTAL TAB	DEL	)	9	I	Y	i	y	$^{h}_{T}$	⇤	□	□	□	□	□	□
1	0	1	0	A	LINE FEED	INS	*	:	J	Z	j	z	⇓	∴	◩	◩	▚	□	▚	□
1	0	1	1	B	VERTICAL TAB	ESC	+	;	K	[	k	{	⇑	$^{E}_{C}$	□	□	▦	⊞	▦	▦
1	1	0	0	C	CLEAR SCREEN & CURSOR HOME (FORM FEED)	FS	,	<	L	\	l	¦	$^{F}_{F}$		□	□	□	◪	□	◹
1	1	0	1	D	CARRIAGE RETURN	DIRECT CURSOR ADDRESS	–	=	M	]	m	}	$^{C}_{R}$		□	□	□	□	□	□
1	1	1	0	E	SET 40 COL CLEAR SCREEN	CURSOR HOME	.	>	N	↑	n	~	$^{4}_{0}$	↘	□	□	□	□	□	□
1	1	1	1	F	SET 32 COL CLEAR SCREEN	US	/	?	O	_	o	■	$^{3}_{2}$		□	□	⊠	▨	⊠	■

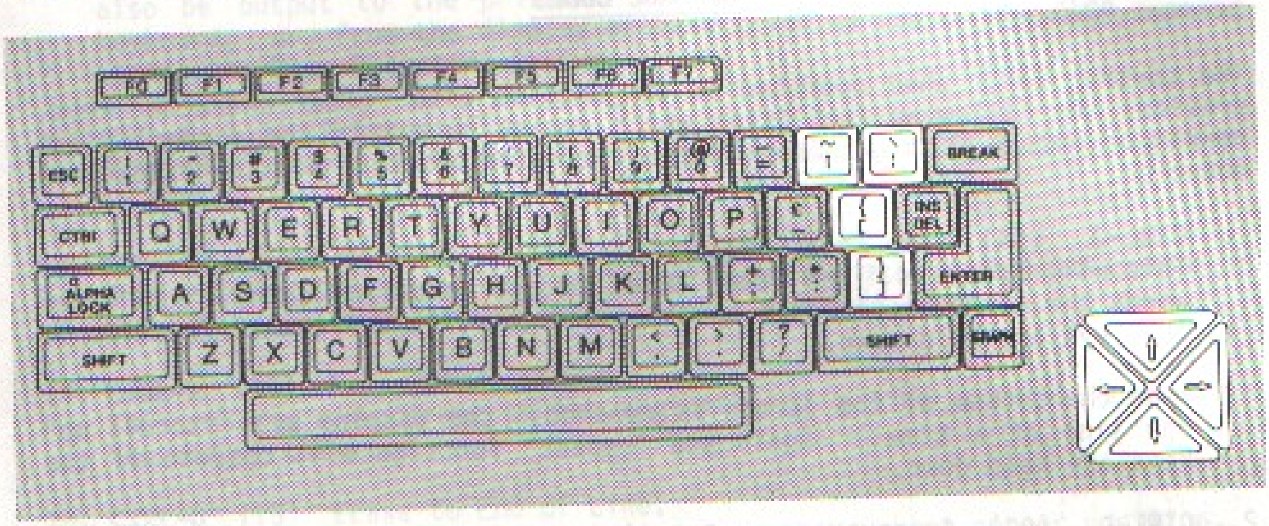
CODE TABLE - GERMAN CHARACTER SET

					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	COL ROW	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NUL	SET 80 COL CLEAR SCREEN	SP	0	§	P	´	p		$^{8}{}_{o}$	■	◳	◲	▥	▧	▤
0	0	0	1	1	SCREEN DUMP	CURSOR ON	!	1	A	Q	a	q	$^{S}{}_{D}$	$^{C}{}_{O}$	□	□	◨	⊞	◧	⊞
0	0	1	0	2	STX	PRINTER ON	"	2	B	R	b	r		$^{P}{}_{O}$	□	□	◟	□	◸	⊞
0	0	1	1	3	ETX	PRINTER OFF	#	3	C	S	c	s		$^{P}{}_{F}$	⊟	⊟	◿	◨	◺	◧
0	1	0	0	4	CURSOR RIGHT	CURSOR OFF	$	4	D	T	d	t	⇒	$^{C}{}_{F}$	⊟	⊟	◨	◺	◧	◸
0	1	0	1	5	ENQ	DELETE TO END OF LINE	%	5	E	U	e	u		$^{D}{}_{L}$	□	□	□	□	□	□
0	1	1	0	6	DELETE CHAR FROM RIGHT	ERASE TO END OF SCREEN	&	6	F	V	f	v	→|	$^{D}{}_{S}$	□	□	□	◝	□	◸
0	1	1	1	7	BEEP	INV. VIDEO	'	7	G	W	g	w	$^{b}{}_{L}$		□	□	▄	⊞	▀	⊞
1	0	0	0	8	CURSOR LEFT	DELETE WHOLE LINE	(	8	H	X	h	x	⇐	$^{D}{}_{R}$	□	□	▄	◝	▀	◿
1	0	0	1	9	HORIZONTAL TAB	DEL	)	9	I	Y	i	y	$^{h}{}_{T}$	|←	□	□	□	◿	□	□
1	0	1	0	A	LINE FEED	INS	*	:	J	Z	j	z	⇓	∴	◿	◸	▚	◸	▚	◟
1	0	1	1	B	VERTICAL TAB	ESC	+	;	K	Ä	k	ä	⇑	$^{E}{}_{C}$	□	□	▩	▦	▩	▩
1	1	0	0	C	CLEAR SCREEN & CURSOR HOME (FORM FEED)	FS	,	<	L	Ö	l	ö	$^{F}{}_{F}$		◸	◹	▚	◿	▚	◹
1	1	0	1	D	CARRIAGE RETURN	DIRECT CURSOR ADDRESS	—	=	M	Ü	m	ü	$^{C}{}_{R}$		□	□	◞	◹	◝	◹
1	1	1	0	E	SET 40 COL CLEAR SCREEN	CURSOR HOME	.	>	N	^	n	ß	$^{4}{}_{o}$	↖	◸	◺	◿	□	◝	□
1	1	1	1	F	SET 32 COL CLEAR SCREEN	US	/	?	O	_	o	■	$^{3}{}_{2}$		◝	◿	☒	▨	☒	■

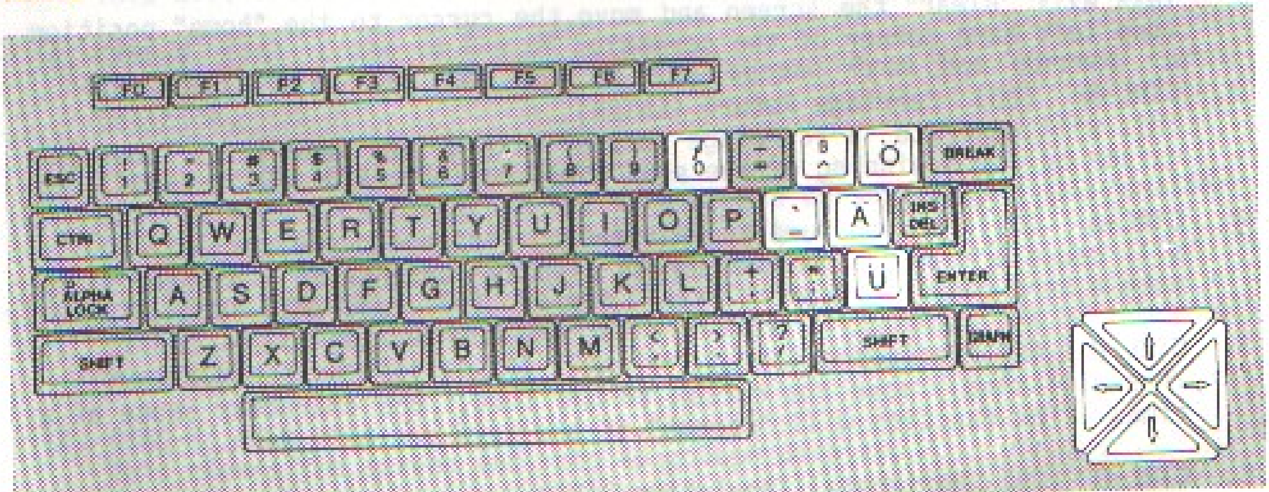
CODE TABLE - SPANISH CHARACTER SET

					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b₄	b₃	b₂	b₁	COL ROW	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NUL	SET 80 COL CLEAR SCREEN	SP	0	@	P	´	p		$^8{}_0$	■	◳	◲	▥	▧	▤
0	0	0	1	1	SCREEN DUMP	CURSOR ON	!	1	A	Q	a	q	$^S{}_D$	$^C{}_O$	□	□	◨	◫	◧	◫
0	0	1	0	2	STX	PRINTER ON	"	2	B	R	b	r		$^P{}_O$	□	□	◺	◨	◸	⊞
0	0	1	1	3	ETX	PRINTER OFF	#	3	C	S	c	s		$^P{}_F$	⊟	⊟	◿	◨	◹	◧
0	1	0	0	4	CURSOR RIGHT	CURSOR OFF	$	4	D	T	d	t	⇒	$^C{}_F$	⊟	⊟	◨	◸	◧	◸
0	1	0	1	5	ENQ	DELETE TO END OF LINE	%	5	E	U	e	u		$^D{}_L$	□	□	□	□	□	□
0	1	1	0	6	DELETE CHAR FROM RIGHT	ERASE TO END OF SCREEN	&	6	F	V	f	v	→|	$^D{}_S$	◫	◫	□	◹	□	◸
0	1	1	1	7	BEEP	INV. VIDEO	`	7	G	W	g	w	$^b{}_L$		◫	◫	⬓	⊞	⬒	⊞
1	0	0	0	8	CURSOR LEFT	DELETE WHOLE LINE	(	8	H	X	h	x	⇐	$^D{}_R$	◫	◫	⬓	◝	⬒	◞
1	0	0	1	9	HORIZONTAL TAB	DEL	)	9	I	Y	i	y	$^b{}_T$	|←	□	□	□	◿	□	□
1	0	1	0	A	LINE FEED	INS	*	:	J	Z	j	z	⇓	∴	◪	◩	▚	◸	▚	◟
1	0	1	1	B	VERTICAL TAB	ESC	+	;	K	¡	k	{	⇑	$^E{}_C$	▩	▩	▩	▦	▩	▩
1	1	0	0	C	CLEAR SCREEN & CURSOR HOME (FORM FEED)	FS	,	<	L	Ñ	l	ñ	$^F{}_F$		◜	◝	▩	⧄	▩	⧅
1	1	0	1	D	CARRIAGE RETURN	DIRECT CURSOR ADDRESS	—	=	M	¿	m	}	$^C{}_R$		□	□	◞	◪	◝	◪
1	1	1	0	E	SET 40 COL CLEAR SCREEN	CURSOR HOME	.	>	N	↑	n	~	$^4{}_0$	↖	◸	◟	◿	□	◝	□
1	1	1	1	F	SET 32 COL CLEAR SCREEN	US	/	?	O	_	o	■	$^3{}_2$		◹	◞	▩	▨	⊠	■

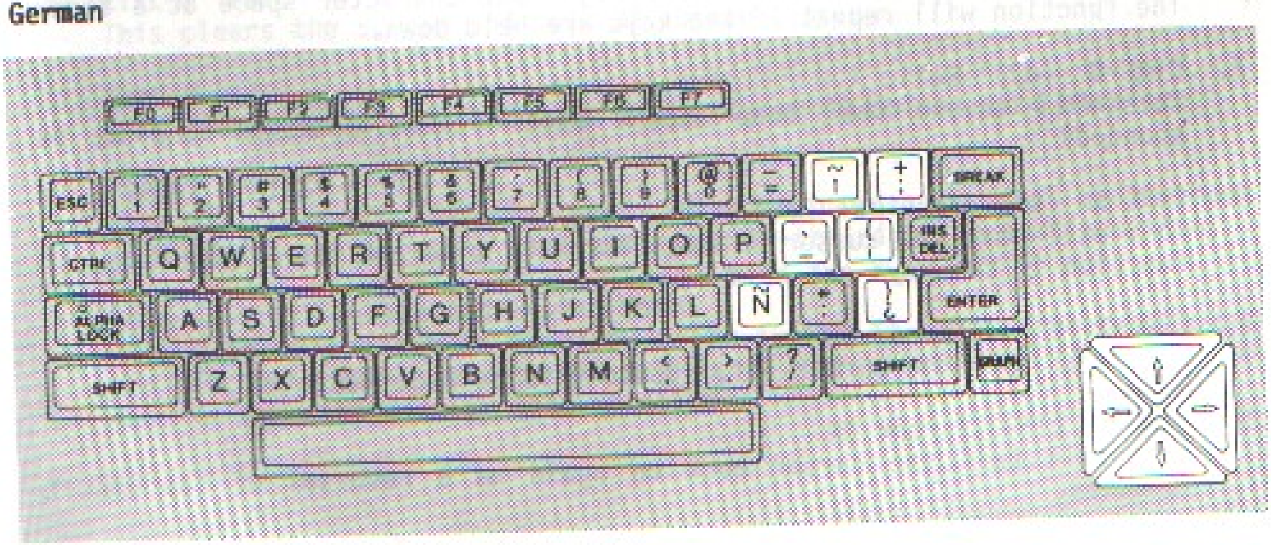
KEYTOP CHANGES WITH ALTERNATIVE CHARACTER SETS


F0 F1 F2 F3 F4 F5 F6 F7
ESC 1 2 3 4 5 6 7 8 9 0 BREAK
CTRL Q W E R T Y U I O P INS DEL
ALPHA LOCK A S D F G H J K L ENTER
SHIFT Z X C V B N M SHIFT GRAPH

ASCII


F0 F1 F2 F3 F4 F5 F6 F7
ESC 1 2 3 4 5 6 7 8 9 0 Ö BREAK
CTRL Q W E R T Y U I O P Ä INS DEL
ALPHA LOCK A S D F G H J K L Ü ENTER
SHIFT Z X C V B N M SHIFT GRAPH

German


F0 F1 F2 F3 F4 F5 F6 F7
ESC 1 2 3 4 5 6 7 8 9 0 BREAK
CTRL Q W E R T Y U I O P INS DEL
ALPHA LOCK A S D F G H J K L Ñ ENTER
SHIFT Z X C V B N M SHIFT GRAPH

Spanish

APPENDIX C
CONTROL CODES

The control key, when operated in conjunction with other character keys, provides the following facilities which assist output to the screen whilst working in MOS. All the numbers quoted below in brackets are in hexadecimal notation.

1. **CTRL-J** (0A) Line Feed (LF);
   This will create a "line feed" (move to the next line down). In other words the cursor moves down one line and there is a "repeat" function if the keys are held down.

   When the bottom of the screen is reached, the display will "scroll up" one line at a time. Again the "repeat" function will operate if the keys are held down.

2. **CTRL-L** (0C) Cursor Home and Clear Screen.
   This will "clear" the screen and move the cursor to the "home" position (top left hand corner of the screen).

3. **CTRL-M** (0D) Carriage Return (ENTER)
   This will generate a "carriage return" with a line feed (i.e. move the cursor to the beginning of the next line).

4. **CTRL-A** (01) Screen Dump to Printer.
   Will cause a transfer of the screen display contents to a printer (i.e. make a hard copy on paper)

5. **CTRL-H** (08) Backspace (BS).
   This will simply move the cursor to the left, one character space at a time. The function will "repeat" if the keys are held down.

6. **CTRL-D** (09) Horizontal Tabulation. (HT).
   This will move the cursor to the right one character space at a time. The function will repeat if the keys are held down.

7. **CTRL-G** (07) Bell.
   This will invoke the "Beep" sound (i.e. cause a 880Hz tone to be sounded)

8. **CTRL-K** (0B) Vertical Tabulation. (VT).
   This will move the cursor UP one line at a time and there is a "repeat" function if the keys are held down.

9. **CTRL-T** (14) Cursor OFF.
   This will turn the cursor off, should this be required.

10. **CTRL-Q** (11) Cursor ON.
    This will turn the cursor back on again as a reversal of the CTRL-T function.

11. **CTRL-R** (12) Printer ON.
This enables the printer such that any data output to the screen will also be output to the printer. (The data normally coming from the keyboard, or, from the display of text files by use of the DISP command within the Disc Operating System).

12. **CTRL-S** (13) Printer OFF.
This simply turns the printer off as a direct reversal of the operation in CTRL-R.

13. **CTRL-↑** (1E) Cursor Home.
This returns the cursor to the "home" position (top left hand corner of the screen) without affecting the screen contents.(escape).

14. **CTRL-X** (18) Erase Whole Line.
This returns the cursor to the beginning of a line and then erases to the end of the line.

15. **CTRL-U** (15) Erase to End of Line.
This will erase to the end of a line from the current cursor position.

16. **CTRL-V** (16) Erase to End of Screen.
This will erase to the end of the screen from the current cursor position.

17. **CTRL-Y** or **DEL** (19) Delete Character.
This deletes a character to the left of the cursor, moving the remainder of the line one character space to the left.

18. **CTRL-F** or **CTRL-DEL** (06) Delete Character at Cursor.
Either of these combinations will delete a character at the cursor, moving the remainder of the line one character space to the left.

19. **CTRL-N** (0E)
This clears the screen and sets 40 column display.

20. **CTRL-O** (0F)
This clears the screen and sets 32 column display.

21. **CTRL-P** (10)
This clears the screen and sets the optional 80 column card display.

22. **CTRL-Z** or **INS** (1A)
This will insert a character space at the cursor position, at the same time moving the existing text one space to the right of that position.

23. **CTRL →** (1D) Cursor Addressing Mode.
This sets up the cursor address. The syntax is &1D,X,Y where X and Y are the location on the screen. Y is in the value 0-23. the X value depends upon the display mode selected.

APPENDIX D

DEFAULT COLOUR PALETTE

0	Transparent
1	Black
2	Medium Green
3	Light Green
4	Dark Blue
5	Light Blue
6	Dark Red
7	Cyan
8	Medium Red
9	Light Red
10	Dark Yellow
11	Light Yellow
12	Dark Green
13	Magenta
14	Grey
15	White

Appendix E

PERIPHERALS

The following models are suitable for use as peripherals with the MICRO-EINSTEIN.

PRINTERS:

1. TATUNG TP100
2. EPSON FX80 - The TATUNG character set can be programmed into this model

The following printers can also be used with EINSTEIN 256 but do not necessarily conform to the full TATUNG character set. Some alpha-numeric characters/symbols will be different and the graphics characters will not exist on these models.

1. EPSON LX80
2. FACIT 4510
3. SHINWA CT1CP80

Einstein 256 can support serial printers. The default setting is for parallel printers. This can be changed by altering the settings of the DIP switches inside Einstein 256. See Appendix P.

DISC DRIVES:

National Panasonic EME 150C

JOY STICKS:

Model	Type	Manufacturer
"Sure shot"	Digital	Cookridge Computer Supplies

APPENDIX F

DECIMAL/BINARY/HEXADECIMAL CONVERSION

DEC	BINARY	HEX	DEC	BINARY	HEX	DEC	BINARY	HEX	DEC	BINARY	HEX
0	00000000	0	64	01000000	40	128	10000000	80	192	11000000	C0
1	00000001	1	65	01000001	41	129	10000001	81	193	11000001	C1
2	00000010	2	66	01000010	42	130	10000010	82	194	11000010	C2
3	00000011	3	67	01000011	43	131	10000011	83	195	11000011	C3
4	00000100	4	68	01000100	44	132	10000100	84	196	11000100	C4
5	00000101	5	69	01000101	45	133	10000101	85	197	11000101	C5
6	00000110	6	70	01000110	46	134	10000110	86	198	11000110	C6
7	00000111	7	71	01000111	47	135	10000111	87	199	11000111	C7
8	00001000	8	72	01001000	48	136	10001000	88	200	11001000	C8
9	00001001	9	73	01001001	49	137	10001001	89	201	11001001	C9
10	00001010	A	74	01001010	4A	138	10001010	8A	202	11001010	CA
11	00001011	B	75	01001011	4B	139	10001011	8B	203	11001011	CB
12	00001100	C	76	01001100	4C	140	10001100	8C	204	11001100	CC
13	00001101	D	77	01001101	4D	141	10001101	8D	205	11001101	CD
14	00001110	E	78	01001110	4E	142	10001110	8E	206	11001110	CE
15	00001111	F	79	01001111	4F	143	10001111	8F	207	11001111	CF
16	00010000	10	80	01010000	50	144	10010000	90	208	11010000	D0
17	00010001	11	81	01010001	51	145	10010001	91	209	11010001	D1
18	00010010	12	82	01010010	52	146	10010010	92	210	11010010	D2
19	00010011	13	83	01010011	53	147	10010011	93	211	11010011	D3
20	00010100	14	84	01010100	54	148	10010100	94	212	11010100	D4
21	00010101	15	85	01010101	55	149	10010101	95	213	11010101	D5
22	00010110	16	86	01010110	56	150	10010110	96	214	11010110	D6
23	00010111	17	87	01010111	57	151	10010111	97	215	11010111	D7
24	00011000	18	88	01011000	58	152	10011000	98	216	11011000	D8
25	00011001	19	89	01011001	59	153	10011001	99	217	11011001	D9
26	00011010	1A	90	01011010	5A	154	10011010	9A	218	11011010	DA
27	00011011	1B	91	01011011	5B	155	10011011	9B	219	11011011	DB
28	00011100	1C	92	01011100	5C	156	10011100	9C	220	11011100	DC
29	00011101	1D	93	01011101	5D	157	10011101	9D	221	11011101	DD
30	00011110	1E	94	01011110	5E	158	10011110	9E	222	11011110	DE
31	00011111	1F	95	01011111	5F	159	10011111	9F	223	11011111	DF
32	00100000	20	96	01100000	60	160	10100000	A0	224	11100000	E0
33	00100001	21	97	01100001	61	161	10100001	A1	225	11100001	E1
34	00100010	22	98	01100010	62	162	10100010	A2	226	11100010	E2
35	00100011	23	99	01100011	63	163	10100011	A3	227	11100011	E3
36	00100100	24	100	01100100	64	164	10100100	A4	228	11100100	E4
37	00100101	25	101	01100101	65	165	10100101	A5	229	11100101	E5
38	00100110	26	102	01100110	66	166	10100110	A6	230	11100110	E6
39	00100111	27	103	01100111	67	167	10100111	A7	231	11100111	E7
40	00101000	28	104	01101000	68	168	10101000	A8	232	11101000	E8
41	00101001	29	105	01101001	69	169	10101001	A9	233	11101001	E9
42	00101010	2A	106	01101010	6A	170	10101010	AA	234	11101010	EA
43	00101011	2B	107	01101011	6B	171	10101011	AB	235	11101011	EB
44	00101100	2C	108	01101100	6C	172	10101100	AC	236	11101100	EC
45	00101101	2D	109	01101101	6D	173	10101101	AD	237	11101101	ED
46	00101110	2E	110	01101110	6E	174	10101110	AE	238	11101110	EE
47	00101111	2F	111	01101111	6F	175	10101111	AF	239	11101111	EF
48	00110000	30	112	01110000	70	176	10110000	B0	240	11110000	F0
49	00110001	31	113	01110001	71	177	10110001	B1	241	11110001	F1
50	00110010	32	114	01110010	72	178	10110010	B2	242	11110010	F2
51	00110011	33	115	01110011	73	179	10110011	B3	243	11110011	F3
52	00110100	34	116	01110100	74	180	10110100	B4	244	11110100	F4
53	00110101	35	117	01110101	75	181	10110101	B5	245	11110101	F5
54	00110110	36	118	01110110	76	182	10110110	B6	246	11110110	F6
55	00110111	37	119	01110111	77	183	10110111	B7	247	11110111	F7
56	00111000	38	120	01111000	78	184	10111000	B8	248	11111000	F8
57	00111001	39	121	01111001	79	185	10111001	B9	249	11111001	F9
58	00111010	3A	122	01111010	7A	186	10111010	BA	250	11111010	FA
59	00111011	3B	123	01111011	7B	187	10111011	BB	251	11111011	FB
60	00111100	3C	124	01111100	7C	188	10111100	BC	252	11111100	FC
61	00111101	3D	125	01111101	7D	189	10111101	BD	253	11111101	FD
62	00111110	3E	126	01111110	7E	190	10111110	BE	254	11111110	FE
63	00111111	3F	127	01111111	7F	191	10111111	BF	255	11111111	FF

APPENDIX G

ROM ROUTINES (MCAL's)

Einstein 256's machine operating system (MOS) resides in 16k bytes of ROM. In addition to the usual bootstrap and housekeeping functions, it contains a lot of very useful routines. It goes without saying that to fully utilise all the features of the MOS, some knowledge of Z80 programming is essential, and is not a task for the faint-hearted, or the unwary.

Each of Einstein 256's routines is accessed by a series of vectors, - machine calls, or MCAL's. Whilst it is possible to access these routines directly, it is not recommended, since the actual locations of the routines may change with various version of the firmware. Adhering to MCAL's will ensure compatibility with any future versions of the firmware. Accordingly, the absolute addresses of the routines are not given in this handbook.

Using MCAL's

Machine calls (MCAL's) are executed on a restart 8 (RST08) instruction, immediately followed by the MCAL function number.

When using MCAL's, it is necessary to ensure that all parameters are met when executing them, particularly when using graphics, sound and disc routines.

EINSTEIN 256 MOS INFORMATION

Function Number	Source Label	Description
80	ARITH	Performs as "A" (ARITHMETIC) from MOS **Values Passed** xxxx in HL pair yyyy in DE pair
81	BAUD	Performs as "B" (BAUD) from MOS **Values Passed** x - Receive rate - upper nibble of L register y - Transmit rate - lower nibble of L register ww - Mode Byte - D register zz - Command Byte - E register (See chaper 5) If DE is zero, the mode and command bytes will remain unchanged. For the correct receive and transmit rates, the baud rate factor x 16 must be used.
82	COPY	Performs as "C" (COPY) from MOS **Values Passed** xxxx - Start - HL pair yyyy - finish - in DE pair zzzz - destination - in BC pair.
83	DECIML	Performs as "D" (DECIMAL) from MOS **Values Passed** xxxx in HL pair
84	DECML	Performs as "D" (DECIMAL) from MOS **Values Passed** xxxx - Start - in HL pair yyyy - finish - in DE pair zzzz - destination - in BC pair.

85	MFILL	Performs as "F" (FILL) from MOS **Values Passed** xxxx - Start - HL pair yyyy - finish - DE pair zz - Value - C register
86	GOTO	Performs as "G" (GOTO) from MOS **Values Passed** xxxx - Execution address - HL pair yyyy - break point -DE pair (See chapter 5 handbook). If DE is zero, no break point is set.
87	HEX	Performs as "H" (HEXADECIMAL) from MOS **Values Passed** Pointer to text (in RAM) - DE pair The decimal number is held at (DE) and terminated with 00. The hexadecimal number is returned in the HL pair in addition to being displayed.
8B	CHARS	Selects the character set, and performs a system reset (MCAL ZSYSRS BE). Performs as "L" command in MOS - See chapter 5. **Values Passed** x in HL 0 = Einstein (ISO646) 1 = ASCII 2 = German 3 = Spanish **Values Returned** C flag = 1 on return for error **Modifies** All
8C	MODIFY	PERFORMS AS "M" (MODIFY) from MOS **Values Passed** xxxx - Address to modify from - HL pair
91	RDBLOK	Performs as "R" (READ) from MOS **Values Passed** pointer to text (in RAM) - DE pair The text takes the format as shown in chapter 5 (See ZRBLK function no. A4)

9A	ZINIT	Re-entry point to MOS.
9B	ZRSCAN	- Repeat key scan This will return the value of any key pressed, in the "A" register 00 is returned if no key is pressed 00 can also be returned if the keyboard poll rate (polled with this MCAL) is greater than the key repeat speed. This repeat speed can be altered in scratch pad location FB43H. **Note** This works as the KBD command in EBASIC On return, the Z80 "Z" flag is set to zero for a valid key.
9C	ZKEYIN	- Input key This will return the value of any key pressed, in the "A" register. Unlike MCAL 9B, this will wait for a key to be pressed. **Note** This is similar to the INCH command in Tatung/Xtal Basic. On return, the Z80 "Z" flag is set to zero for a valid key.
9D	ZGETLN	- Get text from keyboard This will enter a line of text from the keyboard into RAM from the address held in the DE pair. The test is displayed on the screen on pressing the keys and the line is terminated with an ENTER.
9E	ZOUTC	- Character Output Outputs a character to screen held in the "A" register.

9F ZPOUT Outputs a character to the parallel printer held in the "A" register.

A0 ZSLOUT - Serial Output

Outputs a character to the serial port from the "A" register.

A1 ZSRLIN - Serial Input

Read a byte from the serial port and returns the value in the "A" register.

A2 ZRSECT - Sector Read

Reads a sector from the disk into the sector buffer (A sector is 200H Bytes).

The following are set up in the scratch pad.

Location	**Label**
FB50H	HSTDSC - Drive number (0-3)
FB51H	HSTTRK - Track (0-27H)
FB52H	HSTSEC - Sector (0-9)
FB53H	HSTDMA- Sector buffer address (normally FE00H)

Drive error reports in the "A" register on return:-

01 bad date (CRC error)
02 Write protect (MCALS A3 & A5)
03 No sector
04 No disk
05 No drive
00 or 08 operation successful.

A3 ZWSECT - Sector Write

Writes the sector to the disk (see MCAL A2)

Drive error reports in the "A" register (see MCAL A2).

A4 ZRBLK - Read block

Read a block of data from the disk

Values Passed

Drive no. (0-3) in A register
Start address in HL pair
Finish address in DE pair
Sector (0-9) in B register
Track (0-27H) in C register

Memory is filled to the next complete sector (200H bytes) i.e. if the start and finish address is specified as 6000H and 6001H respectively, 6000H to 6200H will be read from the disk.

Drive error reports in the "A" register on return (see MCAL A2).

A5 ZWBLK - Write Block

Writes a block of data to the disk. The values passed are the same as for MCAL A4 and again is written to the next complete sector (200H bytes)

Drive error reports in "A" register (see MCAL A2).

A6 ZCRLF Outputs a CR (0DH) and an LF (0AH).

A7 ZCRLFZ Outputs a CR and LF if the cursor is not at column zero.

A8 ZSPACE Outputs 1 space.

A9 ZPR4HX Outputs 4 hex digits held in the HL pair. e.g. if HL= 1234H, 1234 is output

---

AA ZP2HXZ Outputs two hex digits held in the "A" register followed by a space.

AB ZPR2HX Outputs two hex digits held in the "A" register (as MCAL AA with no space output).

AC	ZFC4HX	Get a hex number (up to 4 digits) from text into the HL pair. DE points to the text (in RAM). The number is terminated on a non-hex character.
AD	ZFCZHX	Get a hex number (up to 2 digits) from text into the "A" register. DE points to the text. The number is terminated with a non-hex character.
AE	ZDCMD	Outputs a command to the floppy disk controller. The command type is passed in the "A" register. On return, "A" is set to 00 unless the FDC is not executing a command; then "A" is set to FFH.
AF	ZHMDSC	Takes the drive head to track 00 The drive is selected by outputting 1,2,4 or 8 to port 23H (DSCPRT) on return Z = 1 if disk present Z = 0 if disk not present
B0	ZIGBLK	Returns a value in the "A" register from RAM pointed to by the DE pair if DE 7FFFH, or ROM if DE 8000H. NOTE: Commas and spaces are ignored (i.e. the first non comma/non-zero character is displayed).
B1	ZRDMEM	Returns a value in the "A" register from RAM pointed to by the HL pair.
B2	ZRCPYU	Performs an LDIR instruction (switches ROM out first).
B3	ZRPCYD	Performs an LDDR instruction (switches ROM out first).
B4	ZMOUT	Outputs a value held in the "B" register to the PSG port no. held in the "C" register.

B5	ZKSCAN	Returns the value in the "A" register on any key pressed. 00 is returned if no key is pressed. This is similar to MCAL 9B (ZRSCAN) except that it is unaffected by the key repeat speed. That is for each MCAL execution, a value is returned.
BC	ZZTIME	Set up CTC channels 2 and 3 to generate 1 second interrupts for the clock.
BD	ZFDRST	Resets the FDC after an error. Also resets the PSG (using MCAL C0 (XPINIT)
BE	ZSYSRS	This performs the following:- 1. Clears the screen to 40 columns 2. Resets all characters 3. Removes sprites 4. Resets the FDC and PSG 5. Masks the keyboard, fire and ADC interrupts.
BF	ZLOGO	Outputs ***Einstein 256*** logo
C0	ZPINIT	Sets PSG resistor 7 to 7FH and all other registers to 00.
C1	ZSREG	Sends an address held in the BC pair to the EVDP. Data can then be output to VRAM from port 8. Subsequent data bytes sent will be loaded into subsequency VRAM locations. NOTE: A delay of 8μs is necessary between any VRAM reads or writes (e.g. PUSH/POP). NOTE: When ROM is switched on, RST 20H will execute this MCAL.
C2	ZVRIN	Returns a value in the "A" register from the VRAM address pointed to by the BC pair.

C3	ZVROUT	Writes data held in the "A" register to the VRAM address held in the BC pair.
C4	ZPLOT	PLOTS or UNPLOTs pixel. A = 1 for PLOT A = 0 for UNPLOT IX holds the x coordinate IY holds the y coordinate NOTE: UNPLOT has no meaning in graphics mode 6.
C5	ZPLTXY	Plots a point according to the line type (see below). IX holds the x coordinate IY holds the y coordinate **Line Type** Four scratch pad values contain information as to the line type to be drawn, e.g. **SCRATCH LOCATION** FBA8H DOTON - Length to end of first line FBA9H DOTOFF- Length to end of first space FBAAH DOTON2- Length to end of second line FBABH DOTOF2- Length to end of second space line type DOTON DOTOFF DOTON2 DOTOF2 Normally these 4 values are in ascending order. For a continuous line, DOTON is set to FFH and the other 3 zero. NOTE: If all line type bytes are zero, the system will hand up. In graphics mode 6, unplotted has no meaning.

C6 ZPOINT Returns the status of any pixel in the "A" register.

In graphics mode 2:

1 = foreground (or Z=0)
0 = background (or Z=1)
255 = off screen

In graphics mode 6:

The colour value (of the current pallette) at the point specified in the BC & DE registers is returned.

In both cases: BC contains the x coordinate.
DE contains the y coordinate

The VRAM address of the pixel is returned in the BC register pair.

C7 ZPNTXY Returns the status of any pixel in the A register.

This is identical to MCAL C6 except IX contains the x coordinate and IY contains the Y coordinate.

The VRAM address of the pixel is returned in the BC pair.

C8 ZDRWTO This will draw a line from the coordinates held in the IX and IY registers to values in scratch pad locations FB96H (X1) and FB98H (Y1). Each is a two-byte number.
The type of line drawn is determined by the line type values (see MCAL C5).

C9 ZPOLYG This draws a polygon (or ellipse)

The polygon centre coordinates are held in scratch pad locations FB9EH (CX) and FBA0H (CY). Each is a two byte number.

The horizontal and vertical radii are held in locations FBA2H (RADX-2BYTE) and FBA4H (RADY-2BYTE).
The number of sides on the polygon is determined by a two-byte number in scratch pad location FBA6H (CINC).

		A value of 4 will give a circle (256 sided polygon) A value of 80H will give an octagon A value of 100H will give a rectangle etc The start angle is passed in the DE pair, the finish angle is passed in the BC pair, each is in the range 0 to 1024. The lines drawn to the polygon centre are selected by setting the carry flag. The type of line drawn is determined by the line type values (see MCAL C5)
CA	ZORGCO	Adds the x coordinate Origin value held in scratch location FB9AH (ORGX-2Byte) to the contents of BC and returns the result in the BC pair and adds the y coordinate Origin value held in scratch location FB9CH (ORGY-2Byte) to the contents of DE and returns the result in the DE pair. This MCAL is of little use and is called from within other graphics MCALS. The values ORGX and ORGY are normally xero but when altered will cause all graphics output to be offset by that value. This is similar to the ORIGIN command in BASIC. - Not used in graphics mode.
CB	ZCALAD	Returns the VRAM address in the BC pair for coordinates x and y passed in IX and IY registers respectively. The pixel position within the 8 pixel row is returned in the E register counting from the left hand pixel (0-7). Not used in graphics mode 6.
CC	ZSETCL	Writes data held in the "A" register to the pattern-generator table address passed in the BC pair (0 to 17FFH) and sets the corresponding byte in the pattern colour table (2000H to 37FFFH) to the contents of scratch locations FB39H (GCOLR). Not used in graphics mode 6.

CD	ZFILL	Fills an area on screen surrounding coordinates passed in the IX and IY registers (x and y coordinates respectively). If the fill is in foreground (i.e. the point at x,y is not set) then scratch location FBADH (FILLMOD) must be set to zero. MCAL ZPNTXY (C7) can be used to find the fill type needed. In graphics mode 6 FILL fills over the point specified, up to a border defined by a colour change.
CE	ZIMULT	Multiplies the contents of the DE pair and the contents of the BC pair and returns the value in DEHL (The DE pair is the most significant).
CF	ZPRM	Outputs a message to the screen. The data follows the CF data byte and must be a character in the range 0 to 7PH. The message is terminated by adding 80H to the last character in the message. NOTE: This MCAL will not work from ROM. From ROM a "RST 18H" will print out a message in the same manner.
D0	ZVOUT	Outputs a character from the "A" register to the current cursor position without incrementing the cursor position. This can be useful to prevent scrolling, linefeeds etc.
D1	ZSCURS	Returns VRAM addresses relating to the current cursor position. The ASCII text map address is returned in the BC pair (will be in the range 3C00H to 3FBFH). The table pattern generator address (first byte) is returned in the DE pair (will be in the range 3C00H to 3FBFH). The table pattern generator address (first byte) is returned in the DE pair (will be in the range 0000 to 17FFH). The start of the sprite pattern/text pattern table is returned in the HL pair (normally 1800H).

D5	ZWTVDP	Write data B to EVDP register C Values passed: BC Values returned: None modifies: AF;BC
D6	ZWTRGS	Copy RAM from (HL) to the EVDP register range B to C. Values passed: HL,BC Values returned: None Modifies: AF;BC;HL
D7	ZRDVDP	Read from EVDP status register A into A Values passed: A Values returned: A Modifies: AF;BC
D8	ZSETBL	Set VRAM address A14 to A17 for read/write (A17=upper 64K VRAM). Values passed: A Values returned: None Modifies: AF
D9	ZVCMD	Issues a command to EVDP according to the command table data in the scratch pad. Values passed: None Values returned: None Modifies: AF;BC;HL
DA	ZSET16	Sets address lines A14 and A15 (16K block) for VRAM page 0. Values passed: 2MSB's of B register Values returned: None Modifies: AF
DB	ZBCOL	Sets EVDP register 7 (colour reg) to the text colour from (TCOLR) - $0FB38_H$ (upper nibble), and the backdrop colour from (BCOLR) - $0FB36_H$ Values passed: None Values returned: None Modifies: AF,BC
DC	ZUTIL	Loads COPY, or BACKUP utility from ROM to RAM at 100_H Values passed: A 0 = Load copy 1 = Load BACKUP Values returned : None Modifies: AF;BC;DE;HL

APPENDIX H

DISC FORMAT

Track Preamble	Bytes	Value	
	32	4E	
	12	00	
	3	F5	
	1	FE	Ident Address Mark
	1	"	Track No.
	1	00	Side No.
	1	"	Sector No.
	1	02	
	1	F7	CRC (Converted to 2 bytes by controller)
	22	4E	
	12	00	
	3	F5	
	1	FB	Data Address Mark
	512	xx	Data
Padding	44	4E	To end to track

NOTES:

On the disk, tracks 0 & 1 are "System tracks".

APPENDIX J

INTERFACING TRANSIENT PROGRAMS

This information is included for the experienced user who has some knowledge of Z80 Machine Code programming. Inexperienced users are advised to refer to other publications relating to Machine Code Programming before a full understanding of this section can be grasped.

To run transient programs will normally require the use of DOS facilites such as input/output, file creation, opening, closing, reading and writing. All of these and other facilities are handled by a set of DOS "functions", accessed by placing the "function number" into the Z80 C register, and performing a CALL instruction. DOS functions may also be called by means of a CALL to location 0005H in memory.

In addition to supplying a function number, it is usually necessary to pass parameters into the DOS function, and to obtain results. The following conditions then usually apply.

E register - 8 bit value passed to the function.
DE register - 16 bit value passed to the function.
A register - 8 bit value returned by the function.
HL register - 16 bit value returned by the function.

DOS Functions

Function 0 Warm Boot (System Reset)
Parameters: C: 00H
Returned Value: NIL

Function 1 Console Input
Parameters: C: 01H
Returned Value: A: ASCII character input

Function 2 Console Output
Parameters: C: 02H E: ASCII character to output
Returned Value: NIL

Function 3 Auxiliary Input
Parameters: C: 03H A: ASCII character input
Returned Value: A: ASCII character input

Function 4 Auxiliary Output
Parameters: C: 04H E: ASCII character to output
Returned Value: NIL

Function 5 Printer Output
Parameters: C: 05H E: ASCII character to output
Returned Value: NIL

```
Function 6     Direct I/O
   Parameters:     C: 06H  E: OFEH if status required,
                              OFFH if input required,
                              else ASCII character to
                              output
   Returned Value: A: ASCII character or Status value

Function 7     Get I/O vector
   Parameters:     C: 07H
   Returned Value: A: I/O vector setting

Function 8     Set I/O vector
   Parameters:     C: 08H  E: I/O vector setting
   Returned Value: NIL

Function 9     Display message
   Parameters:     C: 09H  DE: Address of Start of
                               Message
   Returned Value: NIL

Function 10    Console Line Input
   Parameters:     C: OAH  DE: Input Buffer Address
   Returned Value: Input characters in buffer

Function 11    Console Status
   Parameters:     C: OBH
   Returned Value:  A:  Status value (FFH if character ready.  Otherwise
   NIL).

Function 12    Return Version No.
   Parameters:     C: OCH
   Returned Value: HL: Version No.

Function 13    Reset Disc System
   Parameters:     C: ODH
   Returned Value: NIL

Function 14    Select Drive
   Parameters:     C: OEH  E: Drive No.
   Returned Value: NIL

Function 15    Open File
   Parameters:     C: OFH  DE: FDESC Address
   Returned Value: A: Directory Code

Function 16    Close File
   Parameters:     C: 10H  DE: FDESC Address
   Returned Value: A: Directory Code

Function 17    Get 1st Directory Entry
   Parameters:     C: 11H  DE: FDESC Address
   Returned Value: A: Directory Code
```

```
Function 18    Get next Directory Entry
   Parameters:      C: 12H
   Returned Value: A: Directory Code

Function 19    Erase File
   Parameters:      C: 13H  DE: FDESC Address
   Returned Value: A: Directory Code

Function 20    Read Sequential
   Parameters:     C: 14H  DE: FDESC Address
   Returned Value: A: Directory Code

Function 21    Write Sequential
   Parameters:     C: 15H  DE: FDESC Address
   Returned Value: A: Directory Code

Function 22    Create File
   Parameters:     C: 16H  DE: FDESC Address
   Returned Value: A: Directory Code

Function 23    Rename File
   Parameters:     C: 17H  DE: FDESC Address
   Returned Value: A: Directory Code

Function 24    Get Drive Log Vector
   Parameters:     C: 18H
   Returned Value: HL: Log Vector

Function 25    Get Current Drive
   Parameters:      C: 19H
   Returned Value: A: Current Drive No.

Function 26    Set Buffer Address
   Parameters:     C: 1AH  DE: Buffer Address
   Returned Value: NIL

Function 27    Get Allocation Vector
   Parameters:     C: 1BH
   Returned Value: HL: Allocation Vector Address

Function 28    Lock Drive
   Parameters:     C: 1CH
   Returned Value: NIL

Function 29    Get Drive Lock Vector
   Parameters:     C: 1DH
   Returned Value: HL: Drive Lock Vector

Function 30    Set File Attributes
   Parameters:     C: 1EH  DE: FDESC Address
   Returned Value: A: Directory Code

Function 31    Get Drive Parameter Address
   Parameters:     C: 1FH
   Returned Value: HL: Drive Parameter Block Address
```

Function 32 Set/Get User Code
Parameters: C: 20H E: OFFH if getting USER code
CODE if setting USER code
Returned Value: A: Current User code

Function 33 Read Random
Parameters: C: 21H DE: FDESC Address
Returned Value: A: Error Code

Function 34 Write Random
Parameters: C: 22H DE: FDESC Address
Returned Value: A: Error Code

Function 35 Compute File Size
Parameters: C: 23H DE: FDESC Address
Returned Value: A: Random RECORD set

Function 36 Set Random Record No.
Parameters: C: 24H DE: FDESC Address
Returned Value: A: Random RECORD set

Function 37 Reset Drive
Parameters: C: 25H DE: Drive Vector
Returned Value: A: 0

Function 38 Return to MOS
Parameters: C: 26H
Returned Value: NIL

Function 39 Set Password
Parameters C: 27H DE: Address of start of
password
Returned Value: A: Error Code

Function 40 Write Random (zero fill)
Parameters: C: 28H DE: FDESC Address
Returned Value: A: Error Code

Error Codes:

01 reading unwritten data
02 (not returned in random mode)
03 cannot close current extent
04 seek to unwritten extent
05 (not returned in read mode)
06 seek past end of disc

APPENDIX K

I/O PORTS

Address (HEX)	Direction	Function

Programmable Sound Generator

Address (HEX)	Direction	Function
00	R/W	Software reset for PSG and FDC
01		
02	R	Read from PSG
03	W	latch address

Video Display Processor (V9938)

Address (HEX)	Direction	Function
08	R/W	VRAM data
09	W only	register data
0A	W	V9938 port for writing to palette
0B	W	V9938 port for indirect access to registers

Programmable Communications Interface (8251A)

Address (HEX)	Direction	Function
10	R/W	data register
11	R/W	control/status register

Floppy Disc Controller (WD1770)

Address (HEX)	Direction	Function
18	R/W	status/command register
19	R/W	track register
1A	R/W	sector register
1B	R/W	data register

Auxiliary Command/Status "Register"

Address (HEX)	Direction	Function
20	R	b0 "Fire" Button 1 b1 "Fire" Button 2 b2 Printer "Busy" b3 Printer "paper empty" b4 Printer "Error" b5 GRAPH/ALPHA key b6 Control key b7 shift key b0 keyboard interrupt mask = "1" to mask b1-b7 not used

Address (HEX)	Direction	Function
Psuedo ADC Mask		
21	W	b0 = 1 to mask, b1-7 not used.
22	R/W	Alpha Lock LED. Toggles with each port access. A reset light the LED.
Disc Drive Select Port		
23	W only	b0 Drive 1 b1 2 b2 3 = 1 to select b3 4 b4 Side Select
24	R/W	Rom select port. Toggles within each access (ROM selected on Reset).
25	W only	"Fire" Button interrupt mask. b0 = 1 to mask; b1-7 not used.
Various System Inputs and Parameters		
26	RD	b0 1 = Alpha Lock key pressed b1 1 = ROM enabled b2 Dipswitch 1 b3 Dipswitch 2 b4 Dipswitch 3 b5 Dipswitch 4 b6 0 = mouse connected b7 cassette input.
Counter-Timer Circuit (Z80ACTC)		
28	R/W	Channel 0 control/data register
29	R/W	Channel 1 control/data register
2A	R/W	Channel 2 control/data register
2B	R/W	Channel 3 control/data register

Address (HEX)	Direction	Function

Input/Output Port A (parallel printer and joystick)

Address (HEX)	Direction		PIN	INPUT	OUTPUT
30	R/W	b0	1	Forward	Data 0
		b1	2	Backward	Data 1
		b2	3	Left	Data 2
		b3	4	Right	Data 3
		b4	6	Fire 1	Strobe
		b5	7	Fire 2	
				Paper empty	General
		b6	8	Error	General
		b7		No connection	Read as 0

Port A interrupt

Address (HEX)	Direction	Function
31	W	80_H = Enable Interrupt 00_H = Disable Interrupt Default value = 00 (Interrupts not used for printing!)

Input/Output Port B (parallel printer and joystick)

Address (HEX)	Direction		PIN	INPUT	OUTPUT
32	R/W	b0	1	Forward	Data 4
		b1	2	Backward	Data 5
		b2	3	Left	Data 6
		b3	4	Right	Data 7
		b4	6	Fire 1	General
		b5	7	Fire2/Busy	General
		b6	8	Acknowledge	General
		b7	no connection read as "0"		
		Default = all outputs = all high			

Pseudo ADC

Address (HEX)	Direction	Function
38	W	Control Register 4 = Left/Right, joystick 2 5 = Forward/Backward, joystick 2 6 = Left/Right, joystick 1 7 = Forward/Backward, joystick 1
	R	Data Register 127= Centre, 0 down/left 255 = up/right. No other values returned.

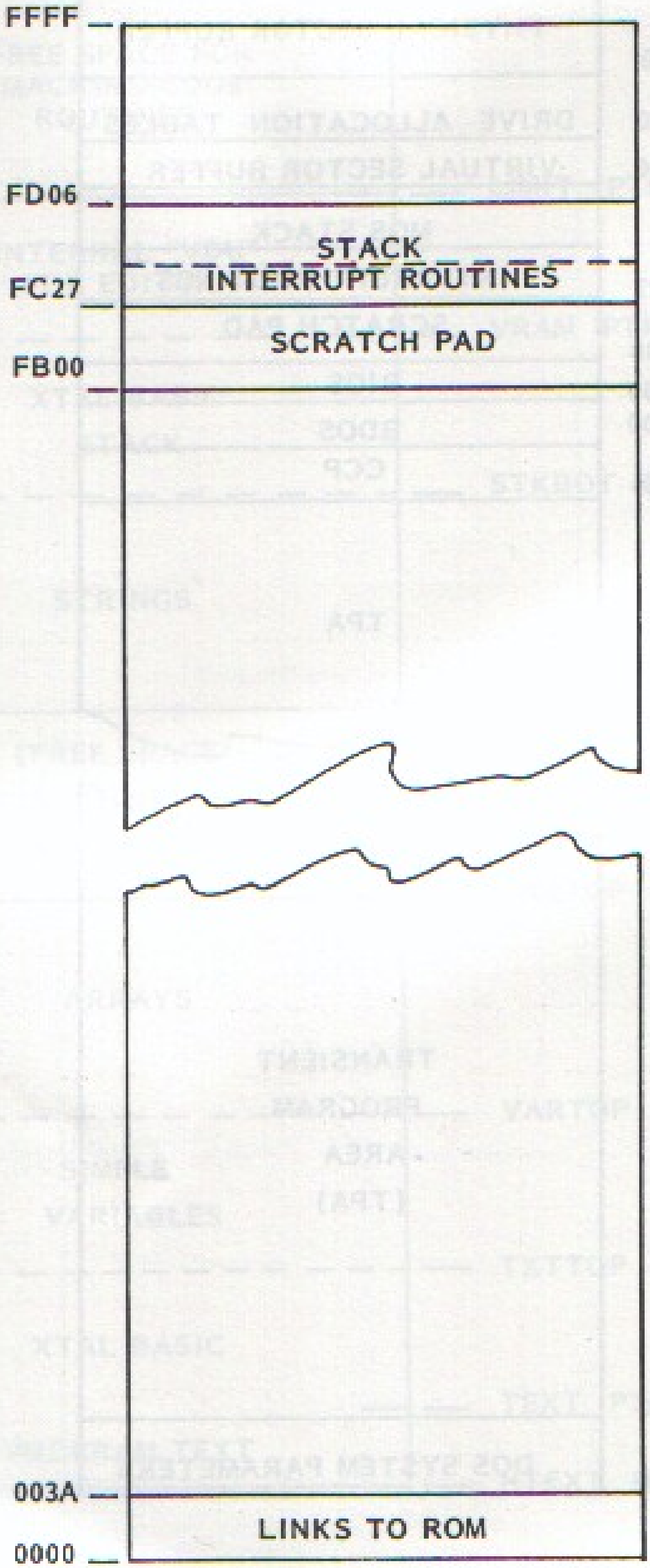
Printer Strobe, and EVDP interrupt mask

Address (HEX)	Direction	Function
80	W	Bit 0: 0 = VDP mask on Bit 1: 0 = Strobe mask on

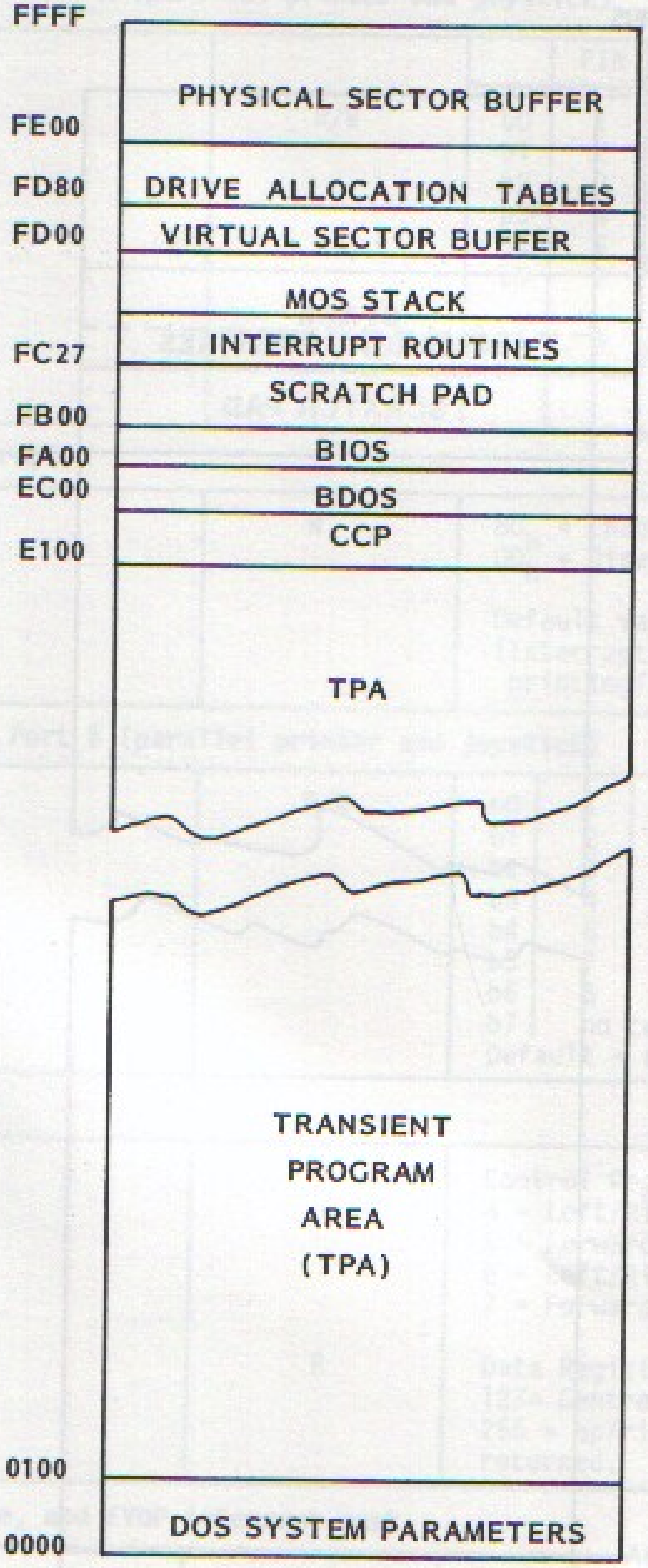
APPENDIX L

MEMORY MAPS

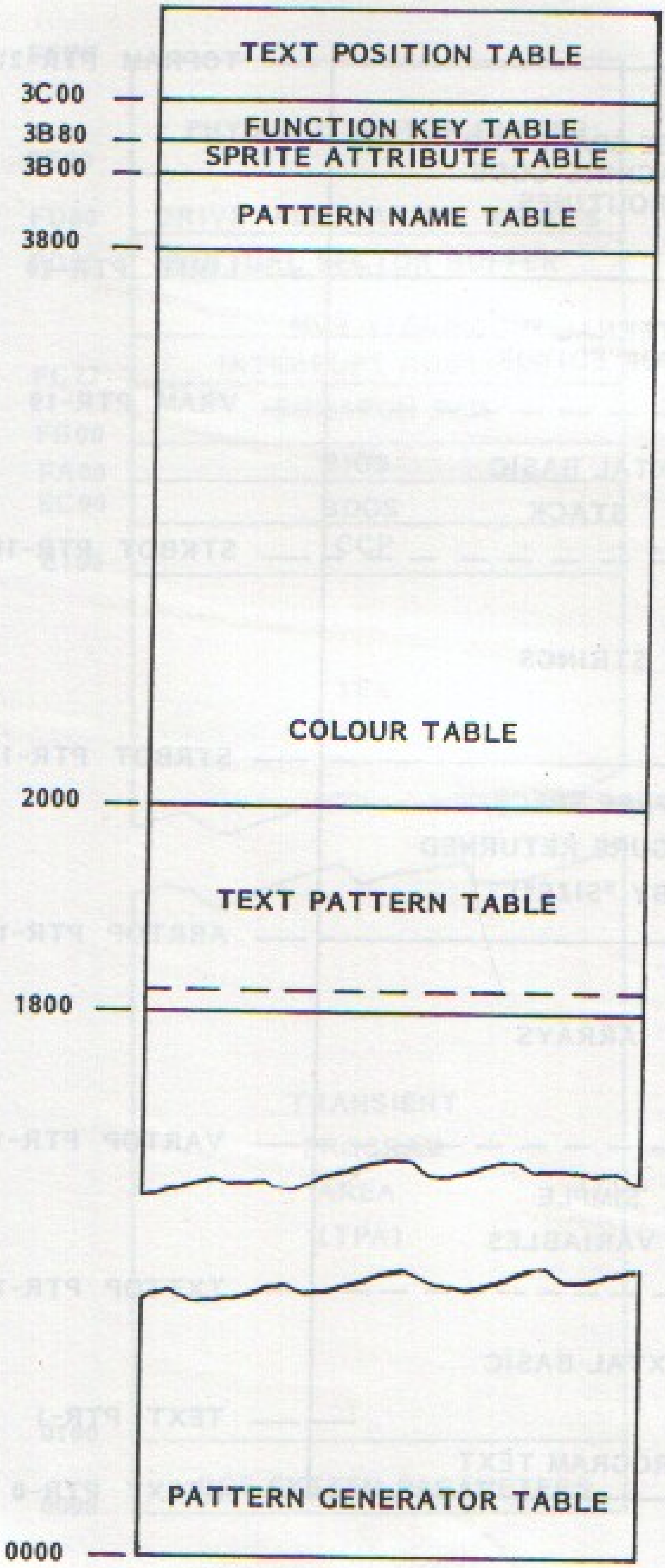
CPU MEMORY MAP - MOS



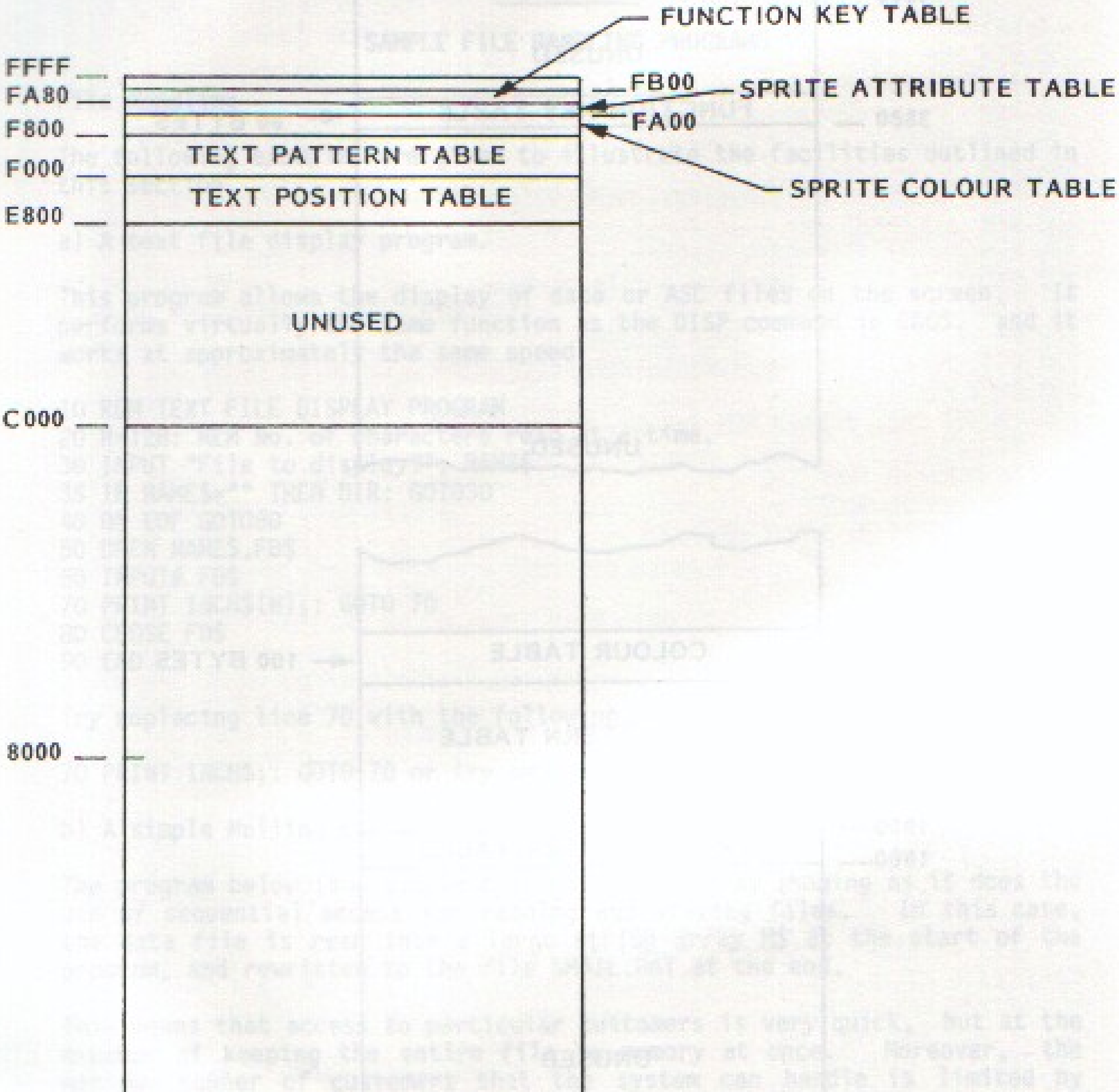
CPU MEMORY MAP - DOS



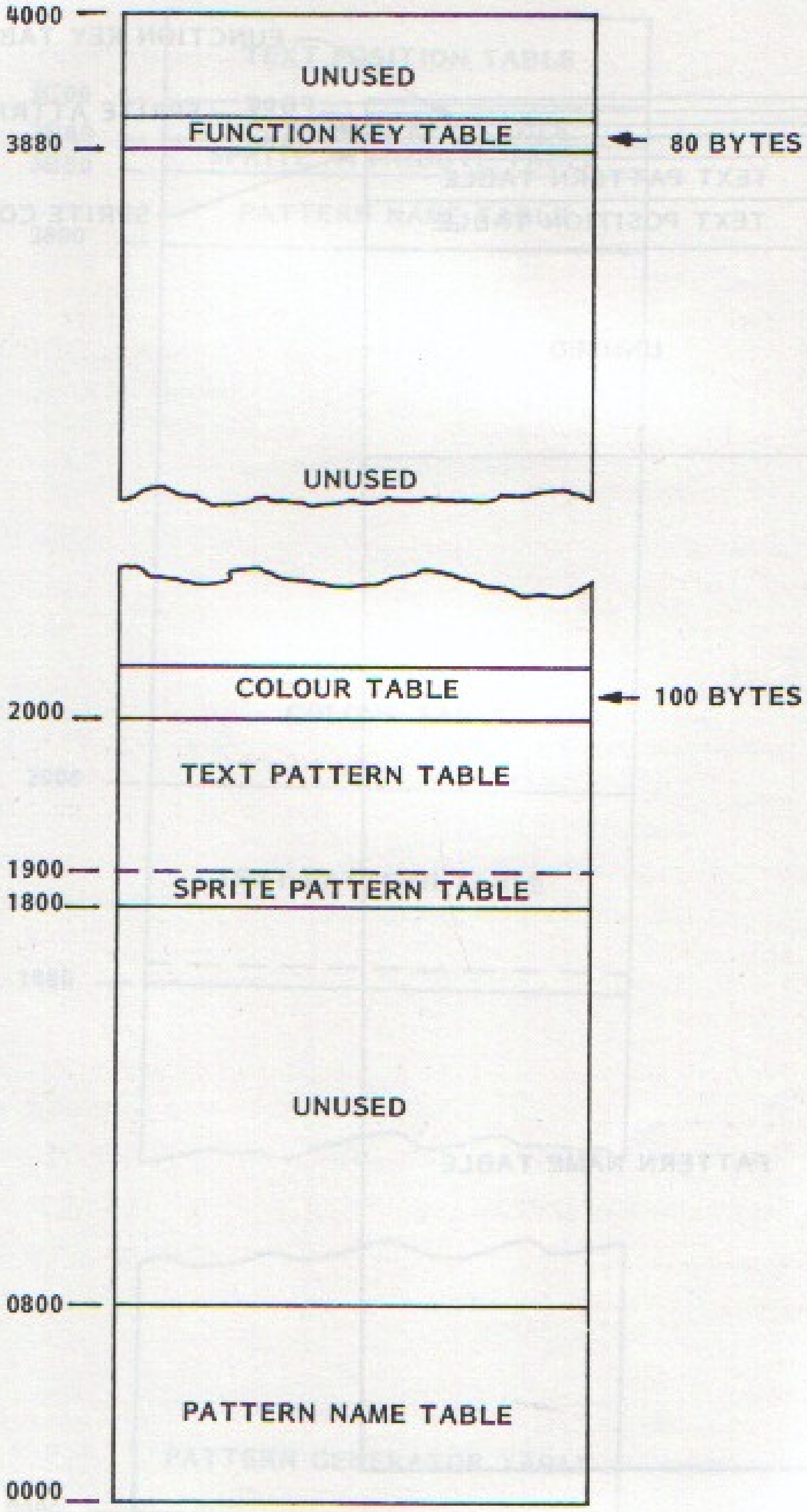
VRAM VIDEO MEMORY MAP - GRAPHICS MODE 2



VRAM VIDEO MEMORY MAP - GRAPHICS MODE 6



VRAM VIDEO MEMORY MAP - TEXT MODE 2



APPENDIX M

SAMPLE FILE HANDLING PROGRAMS

File Handling

The following examples are given to illustrate the facilities outlined in this section.

a) A text file display program.

This program allows the display of data or ASC files on the screen. It performs virtually the same function as the DISP command in EDOS, and it works at approximately the same speed.

```
10 REM TEXT FILE DISPLAY PROGRAM
20 N=128: REM No. of characters read at a time.
30 INPUT "File to display?"; NAME$
35 IF NAME$="" THEN DIR: GOTO30
40 ON EOF GOTO80
50 OPEN NAME$,FD$
60 INPUT# FD$
70 PRINT INCH$(N);: GOTO 70
80 CLOSE FD$
90 END
```

Try replacing line 70 with the following, noting how much slower it is:

70 PRINT INCH$;: GOTO 70 or try smaller values of N in line 20.

b) A simple Mailing List (Sequential Access).

The program below is a simple mailing list program showing as it does the use of sequential access for reading and writing files. In this case, the data file is read into a large string array M$ at the start of the program, and rewritten to the file SMAIL.DAT at the end.

This means that access to particular customers is very quick, but at the expense of keeping the entire file in memory at once. Moreover, the maximum number of customers that the system can handle is limited by memory size, and the size of M$ as dimensioned in line 9000.

The information under each customer consists of his/her name, telephone no. and address, the address being stored in to lines, or fields. The array CUST$ holds these items temporarily when being accessed by one of the program options.

The options supported by the program are to add a customer to the list, to access a customer from the list for modification, and to list all customers to the screen or printer.

```
  5 DRIVE0
 10 REM *** SIMPLE MAILING LIST PROGRAM (SEQUENTIAL
    ACCESS) ***
 20 REM
 30 GOTO 9000
 98 REM
 99 REM ***COMMON ROUTINES***
198 REM
199 REM     ***OPEN DATA FILE***
200 PRINT:PRINT "Do you have a file to load
    (Y/N)?";:Y$=INCH$
210 PRINT Y$: IF Y$="N" THEN RETURN
220 CLS: PRINT@4,10;"Reading in data file..."
230 OPEN FILE$,FD$
240 INPUT# FD$; NCUST: REM Get No. of customers on
    file.
250 IF NCUST=0 THEN 290
260 FOR I=0 TO NCUST-1
270 FOR J=0 TO 3: INPUT M$(I,J)
280 NEXT J,I
290 CLOSE
295 RETURN
298 REM
299 REM ***HEADING DISPLAY***
300 CLS: PRINT@8,0;HEAD$
310 PRINT@3,2;"Number of customers on file: ";NCUST:
    PRINT
320 RETURN
398 REM
399 REM ***WRITE NEW DATA FILE***
400 PRINT: PRINT "Do you wish to save the file
    (Y/N)?";: Y$=INCH$
410 PRINT Y$: IF Y$="N" THEN RETURN
420 CLS: PRINT@4,10;"Writing New Data file..."
430 CREATE FILE$,FD$
440 PRINT# FD$; NCUST
450 IF NCUST=0 THEN 490
460 FOR I=0 TO NCUST-1
470 FOR J=0 TO 3: PRINT M$(I,J)
480 NEXT J,I
490 CLOSE
499 RETURN
798 REM
799 REM ***MENU DISPLAY***
800 CLS: PRINT@8,0;"SIMPLE MAIL LIST PROGRAM"
810 PRINT@4,3;"Options:"
820 PRINT@4,5;"0. Exit Program"
830 PRINT@4,7;"1. Enter Customers"
840 PRINT@4,9;"2. Modify Customers"
850 PRINT @4,11;"3.List Customers"
870 PRINT@4,13;"Which? ";: N$=INCH$(1): PRINT
880 N=VAL(N$): IF N 0 OR N 3 THEN PRINT BEL$: GOTO 870
890 IF N=0 THEN GOSUB 400: CLS:PRINT@8,0;"GOODBYE!";
    BEL$: END
 898 REM
 899 REM ***SELECT OPTIONS***
 900 ON N GOTO 1000,2000,3000
 910 STOP: REM SHOULD NEVER GET HERE!
 998 REM
 999 REM ***END OF COMMON ROUTINES***
1000 REM ***MSUB1 -- Enter Customers***
1010 HEAD$="ENTER CUSTOMERS"
1020 GOSUB 300
1030 PRINT"Any more customers to add (Y/N)?";:Y$=INCH$:
     PRINT Y$: PRINT
1040 IF Y$  "Y" THEN 800
1050 FOR I=0 TO 3
1060 PRINT PRMPT$(I);: INPUT CUST$(I)
1070 NEXT
1080 FOR I=0 TO 3: M$(NCUST,I)=CUST$(I):NEXT:NCUST=
     NCUST+1
1090 GOTO 1020
1999 REM
2000 REM ***MSUB2 -- Modify Customers***
2010 HEAD$="MODIFY CUSTOMERS"
2030 GOSUB 300
2040 INPUT "Customer No.?(If mod's finished press F)
     .. ";CN$: IF CN$="F" THEN 800
2050 CN=VAL(CN$): IF CN=0 OR CN NCUST THEN 2040
2060 CN=CN-1
2070 FOR I=0 TO 3: CUST$(I)=M$(CN,I): NEXT
2080 PRINT@3,8;"Customer No. :",CN+1
2090 FOR I=0 TO 3
2100 PRINT I+1;PRMPT$(I),CUST$(I)
2110 NEXT: PRINT
2120 PRINT "Any changes for this item
     (Y/N)?";:Y$=INCH$:PRINT Y$
2130 IF Y$  "Y" THEN 2180
2140 PRINT "Which line (2-4)?";:Y$=INCH$: PRINT Y$:
     PRINT
2150 I=VAL(Y$)-1
2160 IF I=0 OR I 4 THEN 2030 ELSE PRINT PRMPT$(I);:
     INPUT CUST$(I): CLS
2170 GOTO 2080
2180 FOR I=0 TO 3: M$(CN,I)=CUST$(I): NEXT
2190 GOTO 2030
2999 REM
3000 REM ***MSUB3 -- List Customers***
3010 HEAD$="LIST CUSTOMERS"
3020 GOSUB 300: IF NCUST=0 THEN 800
3030 PRINT "To Screen or Printer (S/P)?";: PF$=INCH$:
     PRINT PF$:PRINT
3040 IF PF$="P" THEN PRINT#1
3050 FOR CN=0 TO NCUST-1
3060 PRINT "Customer No. :",CN+1
3070 FOR I=0 TO 3: PRINT PRMPT$(I),M$(CN,I): NEXT:
     PRINT
3080 IF PF$  "P" THEN INPUT "PRESS ENTER to go on:";Y$:
     PRINT
3090 NEXT CN
3100 PRINT# 0: GOTO 800
```

```
8998 REM
8999 REM ** INITIALISATION **
9000 SEP 44: REM Use separator for DATA below
9010 BEL$=CHR$(7):REM Beep
9020 CMAX=100: REM Max. No. of customers allowed
9030 DIM M$(CMAX-1,3),PRMPT$(3),CUST$(3)
9040 FOR I=0 TO 3: READ PRMPT$(I): NEXT
9050 FILE$="SMAIL.DAT": REM file name
9060 SEP 0: REM Allow commas in input text
9070 ZONE 28,20: REM Set up zone width
9080 GOSUB 200: REM Read in data file
9090 GOTO 800: REM Go and do your stuff!
9098 REM
9099 REM *** DATA FOR FIELD PROMPTS***
9100 DATA "Customer Name:","Telephone No.:"
9110 DATA "Addr. Line 1","Addr. Line 2 :"
```

c) A simple Mailing List (Random Access)

The program suite below is given to illustrate both the use of random-access files and the 'semi-chain' facility. It does the same job as the single program at example b), but with much less memory, and shows how the random-access method improves the file-handling capability. The limit on the number of customers is now dictated only by the free disc space available, and the array M$ of example b., is dispensed with.
The suite consists of four programs, the common and setting-up routines, and the three sub-programs which deal with the three options currently supported (see example b. above).

A record length of 75 characters is used, this limits the amount of information that may be held on each customer, checks being needed to ensure that the total lengths of the fields entered (NB, including CR and LF codes!) do not exceed this length. Such checking may be found at lines 1090-1100 in MSUB1, and 1170-1180 in MSUB2 below. This kind of check is not necessary with a sequential file.

The first record contains the total number of records on file (NCUST), and provides a useful way of preventing access above the limit available.

Finally, note the use of the ON ERR routine at 100, which makes special checks for CHAINing to a non-existent sub-program, and allows the user to create a new data file if one is not present.

```
10 REM **SIMPLE MAILING LIST PROGRAM (RANDOM ACCESS)***
20 REM *** COMMON ROUTINES ***
30 ON ERR GOTO 100
40 GOTO 1000
98 REM
99 REM *** ERROR ROUTINE ***
100 IF ERL=900 THEN PRINT "CANNOT INVOKE DESIRED
    OPTION";BEL$: GOTO 800
110 IF ERR  25 THEN PRINT ERR$;" Error in line ";ERL:
    END
120 PRINT "No data file -- Create (Y/N)?";: Y$=INCH$
130 IF Y$="Y" THEN CREATE FILE$,FD$: PRINT# FD$;"0":
    CLOSE
140 GOTO 800
198 REM
199 REM *** OPEN DATA FILE ***
200 OPEN FILE$,FD$,RL
210 INPUT# FD$,0;NCUST: INPUT# 0: REM Get No. of
    customers on file
220 RETURN
298 REM
299 REM *** HEADING DISPLAY ***
300 CLS: PRINT@8,0;HEAD$
310 PRINT@3,2;"Number of customers on file: ";NCUST:
    PRINT
320 RETURN
798 REM
799 REM *** MENU DISPLAY ***
800 CLOSE: CLS: PRINT@8,0;"SIMPLE MAIL LIST PROGRAM"
810 PRINT@4,3;"Options:"
820 PRINT@4,5;"0. Exit Program"
830 PRINT@4,7;"1. Enter Customers"
840 PRINT@4,9;"2. Modify Customers"
850 PRINT@4,11;"3. List Customers"
870 PRINT@4,13;"Which? ";: N$=INCH$: PRINT N$
880 N=VAL(N$): IF N 0 OR N 3 THEN PRINT BEL$: GOTO 870
890 IF N=0 THEN CLS: PRINT@8,0;"GOODBYE!";BEL$: END
898 REM
899 REM *** CHAIN TO OTHER SUB-PROGRAMS ***
900 HOLD 1000: CHAIN "MSUB"+N$
910 STOP: REM SHOULD NEVER GET HERE!
998 REM
999 REM *** END OF COMMON ROUTINES ***
1000 REM ** INITIALISATION **
1010 SEP 44: REM Use separator for DATA below
1020 BEL$=CHR$(7):REM Beep
1030 DIM CUST$(3),PRMPT$(3)
1040 FOR I=0 TO 3: READ PRMPT$(I): NEXT
1050 FILE$="RMAIL.DAT": RL=75: REM File name & record
    size
1060 SEP 0: REM Allow commas in input text
1070 ZONE 28,20: REM Set up zone width
1080 GOTO 800:
1098 REM
1099 REM *** DATA FOR FIELD PROMPTS ***
1100 DATA "Customer Name:","Telephone No.:"
1110 DATA "Addr. Line 1 :","Addr. Line 2 :"
```

```
1000 REM *** MSUB1 -- Enter Customers ***
1010 HEAD$="ENTER CUSTOMERS"
1020 GOSUB 200
1030 GOSUB 300
1040 PRINT"Any more customers to add (Y/N)?";:Y$=INCH$: PRINT Y$: PRINT
1050 IF Y$  "Y" THEN 800
1060 FOR I=0 TO 3
1070 PRINT PRMPT$(I);: INPUT CUST$(I)
1080 NEXT
1090 L=0: FOR I=0 TO 3: L=L+LEN(CUST$(I))+2: NEXT
1100 IF L RL THEN PRINT "RECORD TOO LONG";BEL$: GOTO 1030
1110 PRINT# FD$,NCUST+1
1120 FOR I=0 TO 3: PRINT CUST$(I): NEXT: NCUST=NCUST+1
1130 PRINT# FD$,0; NCUST: PRINT# 0: REM Update No. of customers
1140 GOTO 1030
```

```
1000 REM *** MSUB2 -- Modify Customers ***
1010 HEAD$="MODIFY CUSTOMERS"
1020 GOSUB 200
1030 GOSUB 300
1040 INPUT "Customer No.?(If mod's finished press F)
     .. ";CN$: IF CN$="F" THEN 800
1050 CN=VAL(CN$): IF CN=0 OR CN NCUST THEN 1040
1060 INPUT# FD$,CN
1070 FOR I=0 TO 3: INPUT CUST$(I): NEXT: INPUT# 0
1080 PRINT@3,8;"Customer No. :",CN
1090 FOR I=0 TO 3
1100 PRINT I+1;PRMPT$(I),CUST$(I)
1110 NEXT: PRINT
1120 PRINT "Any changes for this item (Y/N)?";:Y$=
     INCH$: PRINT Y$
1130 IF Y$  "Y" THEN 1170
1140 PRINT "Which Line (2-4)?";: Y$=INCH$: PRINT Y$: PRINT
1150 I=VAL(Y$)-1: PRINT PRMPT$(I);: INPUT CUST$(I): CLS
1160 GOTO 1080
1170 L=0: FOR I=0 TO 3: L=L+LEN(CUST$(I))+2:NEXT
1180 IF L RL THEN PRINT "RECORD TOO LONG";BEL$:GOTO
     1080
1190 PRINT# FD$,CN: FOR I=0 TO 3: PRINT CUST$(I): NEXT: PRINT# 0
1200 GOTO 1030
```

```
1000 REM *** MSUB3 -- List Customers ***
1010 HEAD$="LIST CUSTOMERS"
1020 GOSUB 200
1040 GOSUB 300
1050 PRINT "To Screen or Printer (S/P)?";: PF$=INCH$: PRINT PF$: PRINT
1060 IF PF$="P" THEN PRINT#1
1070 FOR CN=1 TO NCUST
1080 INPUT# FD$,CN: REM Read Customer record from file
1090 FOR I=0 TO 3: INPUT CUST$(I): NEXT: INPUT# 0
1100 PRINT "Customer No. :",CN
1110 FOR I=0 TO 3: PRINT PRMPT$(I),CUST$(I): NEXT: PRINT
1120 IF PF$  "P" THEN INPUT "PRESS ENTER to go on:";Y$: PRINT
1130 NEXT CN
1140 GOTO 800
```

APPENDIX N

BASIC RESERVED WORDS

ABS
ADC
AND
APPEND
ASC
ATN
AUTO
BAUD
BCOL
BEEP
BIN$
BTN
CALL
CHAIN
CHAR
CHR$
CLEAR
CLOSE
CLS
CONT
COS
CREATE
DATA
DEEK
DEF FN
DEG
DEL
DIM
DIR
DOKE
DOS
DRAW
DRIVE
ELLIPSE
ELSE
END
EOF
ERA
ERL
ERR
ERR$
EVAL
EXP
FILL
FMT
FN
FOR
GCOL
GOSUB
GOTO
HEX$

HOLD
IF
INCH
INCH$
INP
INPUT
INPUT#
INT
IOM
JOY
KBD
KBD$
KEY
LEFT$
LEN
LET
LIST
LISTP
LN
LOAD
LOCK
LOG
MAG
MGE
MID$
MOD
MODE
MOS
MUL$
MUSIC
NEW
NEXT
NOT
NULL
OFF
ON
OPEN
OR
ORIGIN
OUT
PCOL
PEEK
PI
PLOT
POINT
POKE
POLY
POP
POS
PRINT
PRINT@
PRINT#

PSG
PSW
PTR
RAD
READ
REM
REN
RENUM
RESTORE
RETURN
RIGHTS$
RND
RST
RUN
SAVE
SCREEN
SCRN$
SEP
SGN
SHAPE
SIN
SIZE
SPC
SPEED
SPRITE
SPRITE OFF
SQR
STEP
STOP
STR$
SWAP
TAB
TAN
TCOL
TEMPO
THEN
TI$
TO
UNLOCK
UNPLOT
VAL
VDEEK
VDOKE
VDP
VSTAT
VERIFY
VOICE
VPEEK
VPOKE
WAIT
WIDTH
XOR
ZONE

APPENDIX P

HARDWARE

This appendix forms the Hardware Manual for Einstein 256. For service information on the TM11 monitor, and TA11 television adaptor, consult the respective service manuals.

The Einstein Home Computer - model TCS256, features modular construction for easy service access.

As standard the computer is supplied with on integrally mounted 3-inch disc drive, provision is made for the installation of a second, external disc drive, should this be required.

Please Note: In accordance with their policy of continued improvement, Tatung reserve the right to change, alter or modify parts, components or specifications as deemed necessary in order to maintain or improve product performance, quality assurance and/or specifications.

As a consequence, this manual may be subject to change without notice, and no responsibility can be accepted for errors and omissions.

1. HARDWARE INFORMATION

General Precautions

To prevent damage, and ensure correct functioning of the microcomputer, it is recommended that no form of repair, maintenance, or service, be attempted by any person other than a competent engineer.

Safety - Power Supply

The d.c. power for Einstein 256 is provided by the TM11 monitor, or the TA11 television adaptor. **On no account must the Einstein 256 be connected directly to the mains supply.**

Handling Precautions

Einstein 256 contains semiconductor devices which may be damaged by static electrical charges during handling. These devices are indicated by the symbol on circuit diagrams. When replacing, or handling these devices, care should be taken. Soldering irons must be earthed and personnel should use conductive wrist bands earthed via a 1Mohm resistor. If the latter is not practicable, they should discharge themselves by touching an earthed point prior to handling any devices.

Component Replacement

Before removing, or replacing, any parts or components, always disconnect the computer from its power supply (TM11 or TA11).

Construction And Service Access

Einstein 256 features modular construction for easy service access.

NOTE: Before removing the cover, be sure to switch off the TM11 monitor, or TA11X television adaptor, and disconnect the monitor lead.

Cover

The moulded cover of the Einstein 256 may be removed as follows:-

1. Unscrew and remove the 6 retaining screws located under the base of the computer.

2. Lift the cover clear, taking care not to place any strain on the keyboard ribbon cable. Access to the DIP switch can be gained without disconnecting the keyboard ribbon cable. To remove the cover completely carefully disconnect the two keyboard ribbon cables.

Replacing the cover is the reverse of removal.

Sub-Assemblies

Einstein 256 comprises the following sub-assemblies:

Printed Circuit Board Assembly

Keyboard Assembly

Disc Drive Unit

2. TECHNICAL SPECIFICATION: TCS256 COMPUTER

C.P.U.	Z80A - Clock frequency 4MHz
Memory:	CPU 64k ROM 16k Video 192k Total RAM 256k.
Display Generator:	V9938 MSX2 Enhanced Video Display Processor (EVDP). 7 graphic modes 2 Text modes - 40 column and 80 column text 32 multicoloured sprites 8 active sprites per line 512 colours colour palette
Resolution:	512 x 424 pixels (max)
Backdrop:	16 colours

Sprites:

Sprite Mode 1 - 8 x 8 or 16 x 16 pixels with an optional magnification factor of 2. Up to 4 sprites per horizontal row can be displayed. Each sprite can have one of 16 foreground colours, the background colour is always transparent.

Sprite Mode 2 - 8 x 8 or 16 x 16 pixels with an optional magnification factor of 2. Up to 8 sprites per horizontal row can be displayed, Each horizontal line of the sprites can have one of sixteen foreground colours, the background colour is always transparent.

Sprite Mode 1 is available in graphic 1, 2 and multicolour modes, and is software compatible with Einstein sprites

Sprite Mode 2 is available in graphic modes 3,4,5,6 and 7

Sprites are not available in text modes.

Sprite Colour Selections:

For graphics 1 to graphics 6 modes inclusive, the sprite display colour is determined by the colour palette. In graphics mode 7, the 16 colours available are fixed. The sprite colours for graphics mode 7 are:-

Code	R	G Values	B	Colour
0	0	0	0	Transparent
1	0	0	2	Dark Blue
2	3	0	0	Dark Red
3	3	0	2	Dark Magenta
4	0	3	0	Dark Green
5	0	3	2	Dark Cyan
6	3	3	0	Dark Yellow
7	3	3	2	Grey
8	7	4	2	Orange
9	0	0	7	Blue
10	7	0	0	Red
11	7	0	7	Magenta
12	0	7	0	Mid Green
13	0	7	7	Cyan
14	7	7	0	Yellow
15	7	7	7	White

Character Sets:

Four character sets are provided in ROM.

ISO646 (UK English) (default)
ASCII
SPANISH
GERMAN

The default "language" setting is selectable using DIP switches 3 and 4 (See Fig.P.5) according to the table below. Any language may be selected from software regardless of the default setting.

Language	S3	S4
ISO646	OFF	OFF
ASCII	ON	OFF
GERMAN	OFF	ON
SPANISH	ON	ON

All character sets are software programmable. In addition to the 96 alphanumeric characters of each "language", there are 160 graphics symbols common to all four "languages".

Display Output:

Linear RGB + Syncs, composite video, and audio.

Colour Encoding Standards:

NTSC only. The composite video output is encoded to NTSC standards for both 525 and 625 lines. The subcarrier frequency is 3.58 MHz.

Line Standards:

Raster scanned 625 lines, 50Hz or 525 line 60Hz, interlace or non-interlaced. Line standard and interlace mode are software selectable, the default line standard is selectable by means of DIP switch S1. (See fig. P.5). OFF selects 525 lines 60 Hz. ON selects 625 lines, 50 Hz.

Printer Interface

Selectable using DIP switch S2 (see Fig P5). OFF selects parallel, ON selects serial.

Display Clock:

21.47727MHz for both 625/525 lines and 50/60Hz field.

Display Device:

An optional 14 inch TV grade colour display monitor, TM11, is available. The monitor will provide the necessary power for the TCS256 (Einstein 256) and incorporates a loudpeaker and volume control.

For use with a standard domestic TV receiver, a television adaptor, TA11X, is available. The TA11X supplies power to the TCS256 and provides an RF output, modulated with sound and vision.

Input/Output Facilities

Serial Port

8 Pin DIN connector (M003), see Fig P.1.
Full duplex capability to RS232-C/V24 standards. Transmit and receive speeds are software programmable between 45.5 and 9600 bauds.

1. DSR (Data Set Ready)
2. +5V (15mA max)
3. DTR (Data Terminal
   Ready)
4. RTS (Request to send)
5. TxD (Transmit data)
6. RxD (Receive data)
7. CTS (Clear to send)
8. OV

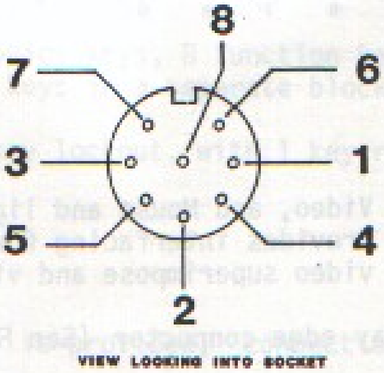


Fig P.1.

Joystick Ports:

Two 9 pin "D" connectors (see Fig. P.2) to accept "Atari" and MSX style digital joysticks. The two ports also double as a centronics printer or general purpose port.

	Pin	Joystick	Centronics
Port A			
	1	Up	DB0
	2	Down	DB1
	3	Left	DB2
	4	Right	DB3
	5	+5V (15mA max)	-
	6	FIRE 1	Strobe
	7	FIRE 2	PE
	8	Output 1	ERR
	9	OV	OV
Port B			
	1	Up	DB4
	2	Down	DB5
	3	Left	DB6
	4	Right	DB7
	5	+5 (15mA max)	-
	6	FIRE 1	-
	7	FIRE 2	BUSY
	8	Output 2	ACKNOWLEDGE
	9	OV	OV

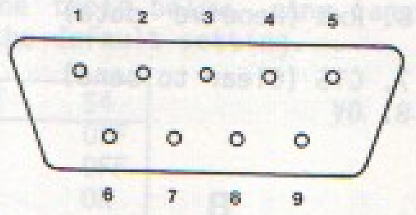


Fig P.2

"VAMP" Interface:

Video, and Mouse and light pen interface. Provides interfacing for Mouse, Light pen, video superimpose and video to VRAM transfer.

Connection is via a 2 x 17 way edge connector (See Fig. P3) on the main CPU pcb.

1*	+5V		18	C4	
2	B	Blue analogue o/p	19	0V	
3	LPD	Light pen detect	20	C5	Colour
4	R	red analogue o/p	21	0V	
5	GND	analogue ground	22	C6	bus
6	G	green analogue o/p	23	0V	
7	LPS	light pen select	24	C7	0V digital ground
8	COMVID	composite video o/p	25	0V	
9	GND	analogue ground	26	CBDR	colour bus direction
10	C0		27	0V	digital ground
11	0V		28	YS	ext. video switch
12	C1	colour	29	0V	digital ground
13	0V		30	DHLCK	high res dot clock
14	C2	bus	31	0V	digital ground
15	0V		32	DHLCK	high res dot clock
16	C3		33	CSYNC	composite/field sync
17	0V		34	HSYNC	horizontal sync

* Power may be taken from this pin providing that the total external power lead, from all connectors, does not exceed 45mA.

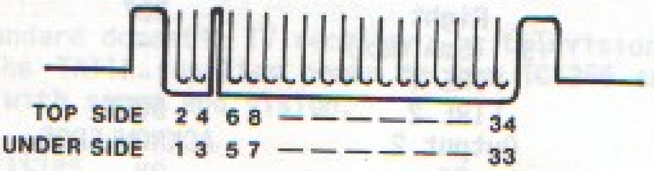

TOP SIDE 2 4 6 8 — — — — — — 34
UNDER SIDE 1 3 5 7 — — — — — — 33

Fig P.3

Sound Output:

Monitor audio: monophonic, 3V p-p (for all 3 channels at max amplitude (MOO5 pin 4).

Stereo Jack: 200mV/Channel into 4 ohms.

The stereo jack is capable of driving low impedance headphones.

Keyboard:

Typewriter style QWERTY keyboard.

1. 48 alphanumeric/graphics keys, 8 function keys and 13 "control" keys, 4 of which are cursor keys in a separate block.

2. All keys to have n key lockout, with 1 key rollover.

3. Automatic repeat.

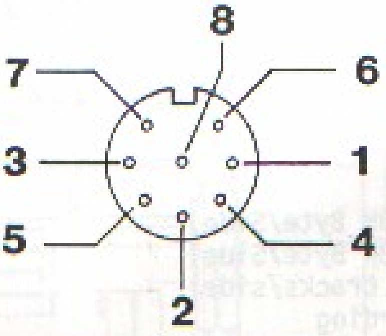
Cassette Interface:

A cassette read facility is provided, connection is via a standard 3.5mm jack, M011.

Monitor Connector:

8 pin DIN (M005) (Fig. P4)

1	Red 0.7v p-p (black = 1.5V)	5	Blue 0.7V p-p (Black 1.5V)
2	0V	6	+12V
3	Green 0.7V p-p (black = 1.5V)	7	+5V
4	Audio	8	Composite Video 0.65V p-p



VIEW LOOKING INTO SOCKET

Fig. P.4

Sound:

Sound output is via the TM11, or via the T.V. when the TA11 is used.
A means of providing a variety of sounds, including chromatic music, with envelope shaping is provided.
The sound generator has three "music" channels and one noise channel.
A stereophonic output is available from MO12, a 3.5mm jack, and can drive low impedance headphones. Minimum impedance is 4 ohm/channel.

Sound channel A corresponds to left.

Sound channel B corresponds to the centre.

Sound channel C corresponds to right.

Disc Drive:

Panasonic EME-150 low profile inch disc drive is to be incorporated.

Specification:	Single Sided
	100 t.p.i.
	40 tracks
	MFM coding, double density
	Fully enclosed cassette

Access time 171 ms average; 12ms track to track
Transfer rate 250 kbit/sec.
Sectors/track 10
Bytes/sector 512

Drive Expansion:

A further, self powered, drive may be added externally.

Media Specification:

500k Byte unformatted	(250k Byte/Side)
400k Byte formatted	(200k Byte/Side)
80 tracks total	(40 tracks/side)

Double Density (MFM) recording
100 t.p.i
Double sided "flippy" cassette.

Software/Firmware:

Operating system	EDOS 1.4
Language	EBASIC 4.5
Utilities	COPY ; BACKUP; FORMAT; DOSCOPY; FKEY

Compatibility:

With the exception of the Joystick inputs, "Einstein 256",is downwards compatible with Einstein (TC01).
However, compatibility may not be preserved where Einstein (TC01) software writes directly to hardware.

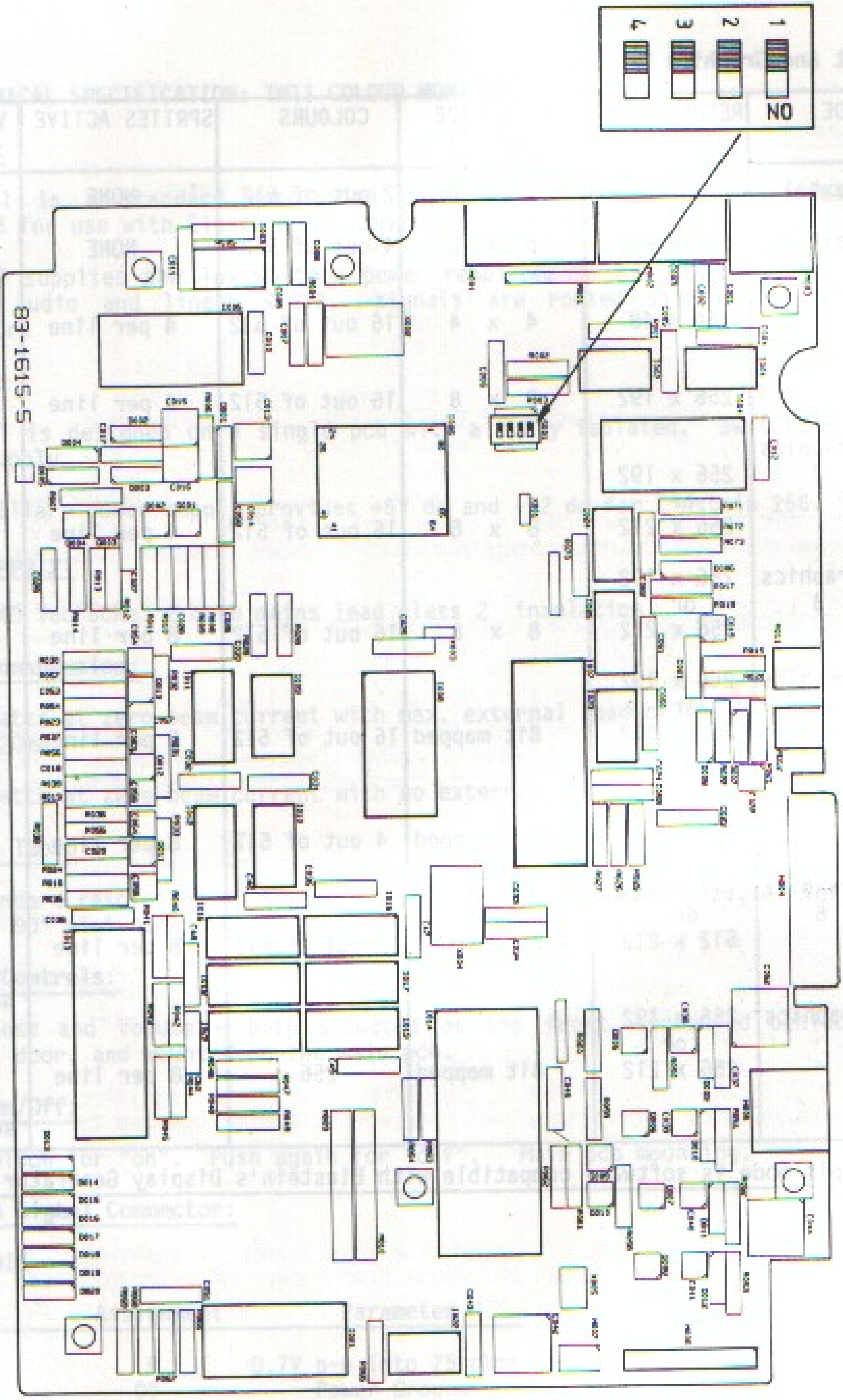


Fig. P5. DIP Switches

Text And Graphics Modes

MODE	RESOLUTION (PIXELS)	CELL SIZE (PIXELS)	COLOURS	SPRITES ACTIVE	VRAM USED
* Text 1	256 x 192	6 x 8	2 out of 512	NONE	4k
Text 2	512 x 192	6 x 8	4 out of 512	NONE	8k
* Multi Colour	64 x 48	4 x 4	16 out of 512	4 per line	4k
* Graphics 1	256 x 192	8 x 8	16 out of 512	4 per line	4k
* Graphics 2	256 x 192 or 256 x 212	8 x 8	16 out of 512	4 per line	16k
Graphics 3	256 x 192 or 256 x 212	8 x 8	16 out of 512	8 per line	16k
Graphics 4	256 x 192 or 256 x 212	Bit mapped	16 out of 512	8 per line	32k
Graphics 5	512 x 192 or 512 x 212	Bit mapped	4 out of 512	8 per line	32k
Graphics 6	512 x 192 or 512 x 212	Bit mapped	16 out of 512	8 per line	128k. (two screens)
Graphics	256 x 192 or 256 x 212	Bit mapped	256	8 per line	128k (two screens)

* This mode is software compatible with Einstein's Display Generator

3. TECHNICAL SPECIFICATION: TM11 COLOUR MONITOR

General:

The TM11 is a standard resolution colour display monitor specifically designed for use with Einstein 256 computer.

The TM11 supplies the low voltage power requirements of the Einstein 256. Power, audio and linear R.G.B. signals are routed via an 8-pin DIN connector.

Chassis:

The TM11 is designed on a single pcb with a fully isolated, switched mode power supply.

An auxilliary power supply provides +5V dc and +12 dc for Einstein 256.

Mains Supply:

220 - 240 Vac 50Hz. 2 core mains lead class 2 insulation.

Power Consumption:

a) 50 watts at zero beam current with max. external load of +5V @ 1.7A and +12V @ 220mA.

b) 38 watts at zero beam current with no external load.

Picture Tube:

14" standard resolution (0.63mm stripe pitch) preconverged high voltage focus. 90° mini-neck C.R.T.

Rotary Controls:

Brightness and Volume - both situated at the front, concealed behind an opening door, and mounted on the main pcb.

Mains On/Off:

Push switch for "on". Push again for "off". Main pcb mounting.

Power & Signal Connector:

8 Pin DIN

Pin No.	Assignment	Parameter
1	R	0.7V p-p into 75 ohms
2	OV	Power Ground
3	G	0.7V p-p into 75 ohms
4	Audio	3V p-p
5	B	0.7V p-p into 75 ohms
6	+12V dc	0.5A dc max
7	+5V dc	2.5 dc max
8	Comp Sync	0.65V p-p into 75 ohms

Sound Output:

1 watt rms (nom) 10% T.H.D.

Speaker:

3" Round 16 ohm

Safety:

Conforms to the requirements of BS415: 1979   Class 2.

Servicing:

There are no user serviceable parts inside the TM11. Servicing should be carried out by a competent engineer. If in doubt, contact your dealer.

4. TECHNICAL SPECIFICATION: TA11 TELEVISION ADAPTOR

General:

The TA11 is a low voltage power supply and modulator unit which is designed to be used in conjunction with Einstein 256 as an alternative to a TM11 monitor and allows Einstein 256 to be used with a domestic TV or composite video input monitor.

Chassis:

The TA11 is designed on two pcb's. One carries the colour modulator section and the other carries the power supply section.

The modulator section produces a composite video output and RF output with intercarrier sound. The power supply section produces +12V and +5V to supply Einstein 256.

The +5V supply incorporates a soft start switching regulator for improved efficiency with overvoltage and overcurrent protection and the +12V supply is a conventional series regulator.

Mains Supply:

TA11 - 220/240V ac 50Hz. 2 core mains lead, class 2 insulation.
TA11X - 90/120V ac 60Hz. Standard black two core mains lead class 2 insulation.

Power Consumption:

25 watts typical (with Einstein 256 connected).

Power & Signal Connector:

8-Pin DIN

Pin No.	Assignment	Parameter
1	R	700mV p-p
2	OV	Power Ground
3	G	700mV p-p
4	Audio	3V p-p
5	B	0.7V p-p
6	+12V dc	0.5A dc max
7	+5B dc	2.5A dc max
8	Comp Video	0.65V p-p into 75 ohms

Video Phono Socket:

Composite Video Output 1V p-p, +10% into 75 ohms.

RF Output (Phono Socket):

	TA11	TA11X
Channel	36 (UHF)	13(VHF)
Encoding standard	PAL	NTSC
Output Level	1.5mV typical	1.5mV typical

Safety:

Conforms to the requirements of BS451: 1979 Class 2.

Servicing:

There are no user serviceable parts inside the TA11. Servicing should be carried out by a competent engineer. If in doubt contact your dealer.

5. CIRCUIT DESCRIPTION

General

Einstein 256 computer uses the Z80A microprocessor operating at its maximum 4MHz clock-rate. All peripheral functions are performed by large scale integration (LSI) devices and the hardware design has been specifically directed at minimizing the component count. A large part of the port decoding and interrupt logic has been included in the semi-custom (gate array) device.

Each functional circuit block is described in detail, but no attempt will be made to describe the internal architecture of any device except where this pertains to the external operation of the circuit. Further information on specific devices is available in the relevant manufacturers data. A circuit diagram is included at the end of this section.

Central Processor Unit (CPU)

The CPU (I007) operates at a clock rate of 4MHz. This signal is generated from the 8MHz master crystal X002 by the gate array I006. The gate array provides buffered clock signals at 2MHz, 4MHz, and 8MHz as appropriate for each peripheral device, except the video display processor (I014) which has its own crystal oscillator.

The address, data, and control signals from the CPU are not buffered as the Z80A provides sufficient drive capability directly from its output pins. The RESET signal for the Z80A is provided by the gate-array. A RESET signal may be generated on two seperate conditions, namely: a keyboard reset (by pressing the CRTL, GRAPH, and ALPHA-LOCK keys together) or a power-on reset. If a keyboard reset is generated, the RESET and RESET signals remains active for the period that the keys are held. The BUSRQ, WAIT, and NMI pins on the Z80A have discrete pull-up resistors fitted, but the INT signal is "pulled up" by the internal circuitry of the gate-array which uses this function.

CPU Memory

The CPU memory consists of three devices I008, I011, I012 which provide 16 kilo bytes of Read Only Memory (ROM) and 64 kilo bytes of Random Access Memory (RAM). All the necessary control signals for memory access are generated by the gate array which allow the ROM to be "bank switched" with the lower 32k of RAM under software control and generates the 8-bit refresh signals for the dynamic RAM. The RAM used requires a multiplexed address bus which is generated by I009 and I010. The gate array controls the state of these multiplexors via the MPX signal and ensures that both devices are in the correct state for the Z80A refresh cycle to be used. The delay introduced by resistor-capacitor combination R005, C011, and two gates from I003 allows the multiplexors I009, I010 to change state before the column address is latched into the RAM.

Port Address Decoding

All the Input/Output (I/O) ports for the Z80A are decoded within the gate array with the exception of the Strobe Port at address 80H. In most cases the decoded port address alone is generated by the gate array, but in some cases signals specifically related to a particular device have been provided.

Programmable Sound Generator (PSG)

The Programmable Sound Generator PSG) generates audio on three different channels, these are then buffered and combined into two seperate outputs. The PSG also provides two 8-bit parallel ports which are used to scan the keyboard matrix under software control. The PSG is provided with a reset signal from gate array which allows a reset to be generated from software, as well as during power-on or keyboard reset. This allows the registers of the PSG to be set to a known state very quickly when required.

The three audio outputs from the PSG are buffered by transistors Q011, Q012, Q013 and their associated components. The resulting collector outputs are combined in a resistor summing network to give a mono audio signal which is fed via M005 (the monitor socket) to the monitor or television adaptor. The output from the emitters of Q011, Q012, Q013 provide stereo output for the Audio output socket.

Keyboard Interface

The keyboard is scanned under software control via the parallel ports of the PSG, the only exceptions to this are the GRAPH, CTRL, SHIFT, and ALPHA LOCK keys which are connected directly to the gate array. A diode matrix, consisting of diodes D013 - D020, is used to give a "wired AND" logic function which is also used by the gate array to generate a keyboard interrupt when this function is enabled. Combinational logic internal to the gate array allows a "keyboard reset" to be generated when the GRAPH, CTRL, and ALPHA-LOCK keys are pressed simultaneously.

The keyboard unit also has two Light Emitting Diodes (LED) fitted to it, these give a "power-on" (green LED) and "Alpha Lock on" (red LED) indication when illuminated. The Alpha Lock LED is fitted in the Alpha Lock key top, and is driven from the gate array via an open collector inverter I022. The Power-On LED is driven directly from the +5V supply rails. Both LEDs rely on the track resistance of the keyboard unit to provide current limiting.

Video Display Processor (VDP)

The display processor used is the Yamaha Enhanced Video Display Processor (EDVP). This device is capable of displaying text and graphics in a number of different modes, offering a variety of colour/resolution combinations. The maximum addressable memory (192 kilo-bytes - I015, I016, I017, I018, I019, I020) is fitted as standard to allow all possible modes to be exploited. The EDVP generates all the signals necessary to support this memory which is accessed via the EVDP's internal registers.

The internal timing necessary for all operations within the EVDP is provided by a single crystal oscillator which uses a 21.47727 MHz crystal (X004) as a frequency reference. The video outputs from the EVDP give R, G, B, and composite synchronising pulses directly. A composite video signal is also provided.

The line standard may be set to 625 or 525 lines under software control. Additionally, the composite video output is encoded to NTSC standards. The RGB and Sync signals from the EVDP are buffered and clamped by transistors Q005, Q006, Q007, Q008 and their associated components before being fed to the monitor and VAMP connectors. The VAMP connector also carries all the expansion signals from the EVDP (Colour Bus, Dot clocks, and Mouse & Light Pen interface signals).

Floppy Disc Controller (FDC)

The FDC (IO21) handles all data transfer between the disc drive(s) and the CPU. A maximum of two disc drives may be connected to the main circuit board. A replacement signal cable is required for fitting the second disc drive which also requires an external supply.

A disc drive is selected from software by setting the appropriate bit in the drive select port, which controls the DS0, and DS1 pins from the gate array. All other control signals are generated and interpreted by the FDC. The reset signal for the FDC (MR) is generated by the gate array. This allows a reset to both on a power-on or keyboard reset, and also under software control in the event that the FDC cannot complete a valid command (usually when a disc is not present in the drive).

Counter Timer Circuit (CTC)

The CTC is a device from the Z80A family, and is used to provide the timing for the real-time clock and for baud rate generation for the serial RS232 port. The CTC has four timing channels and these are dedicated one each to receive and transmit clocks for the serial port, and the remaining two channels are cascaded to give the slow clock rate required for the real-time clock.

Programmable Communications Interface (PCI)

The PCI handles all serial input and output via the RS232 connector (M003). All the necessary signals are controlled directly by the PCI and are subsequently buffered by the level shifting devices I001, I002. The PCI transmit and receive clocks are provided by the CTC. Thus the baud rate may be set to a convenient rate under software control. Split transmit/receive rates can be accommodated.

Joystick Ports

The joystick ports are connected directly to the gate array and have a variety of means of access (from software). The basic architecture of these ports is that of open collector outputs with input lines connected directly to these outputs. This arrangement allows simple, switch type, joysticks or games paddles to be readily interfaced when using the ports as inputs. To maintain compatibility with Einstein (TC01) computer circuitry has been included to mimic the Analogue to Digital Convertor (ADC) fitted in that machine. The parallel output ports are also used to drive a parallel (Centronics type) printer if necessary and a strobe signal is provided to complete this interface. A small section of circuitry (I023, I024, I025) has been included external to the gate array to give control over this strobe signal and also the EVDP interrupt line in order to mask these signals when required.

Interrupt Handling

The Z80A family parts contain circuitry to allow vectored interrupts to be used when the Z80 is in Interrupt Mode 2. Additional circuitry has been incorporated in the gate arrays to enable the non-Z80 peripheral devices to use this interrupt structure also. Interrupts generated by the printer port, the joystick "fire" buttons, the keyboard, and the EVDP are all available to the programmer. Each interrupt signal has mask logic associated with it which gives control over the times at which the interrupt is recognised by the CPU, and "Daisy Chain" priority encoding to ensure that high priority interrupts are serviced first.

-12V Supply

The power supply unit (either a TM11 Monitor or TA11 Television Adaptor) provides +5V and +12V supplies. The RS232 output buffer device (I002) requires a -12V supply. This -12V is generated by an astable multivibrator Q002, Q003 and driver circuit Q001, Q004 and associated components which drive a voltage doubler consisting of C013, C014, C015, C016, D002, D003, D004, D005. This output from the doubler is regulated by a zener diode D001 and fed to the RS232 output buffer I002.

PARTS LIST

The majority of components used in Einstein 256 (TC11) are generally available from stockists. If any are replaced, they should be of the same type and rating as the original.

The following components can only be obtained from a Tatung authorised stockist. When ordering quote the description and code number given in the list.

Cct Ref.	Description	Code No.
X002	Crystal - 8 MHz	16-1903-9
X004	Crystal - 21.47727 MHz	16-1910-1
X005	Suppression Filter	15-7620-8
I006	Gate Array	19-8243-5
I008	Integrated Circuit - 16K ROM	19-8198-6/PG01
I013	Integrated Circuit - P.S.G.	19-8100-5
I014	Integrated Circuit - E.V.D.P.	19-8244-3
I021	Integrated Circuit - F.D.C.	19-8110-2
RS001	Resistor Pack - 6K8x12	11-5209-4
S001	DIP Switch Assembly (4-way)	20-4068-9
M001	Connector - Joystick Port	22-8130-9
M002	Connector - Joystick Port	22-8130-9
M003	Connector (DIN 8-way)	22-8129-5
M005	Connector (DIN 8-way)	22-8129-5
M006	Connector - Disc Drive	22-8131-7
M007	Connector - Drive "A" Power	22-8132-5
M009	Connector - Keyboard	22-8133-3
M010	Connector - Keyboard	22-8133-3
--	Keyboard	83-1587-6
--	Disc Drive	17-0062-6
--	Mounting Bracket - Disc Drive	83-1586-8
--	Power Head - Disc Drive	22-8136-8
--	Signal Head - Disc Drive	22-8137-6
--	Lens - Function Key	83-1614-7
--	Moulded Cover - Top	83-1583-3
--	Moulded Cover - Base	83-1584-1/10110
--	Disc Copyright Assembly (Master Disc)	05-2666-5

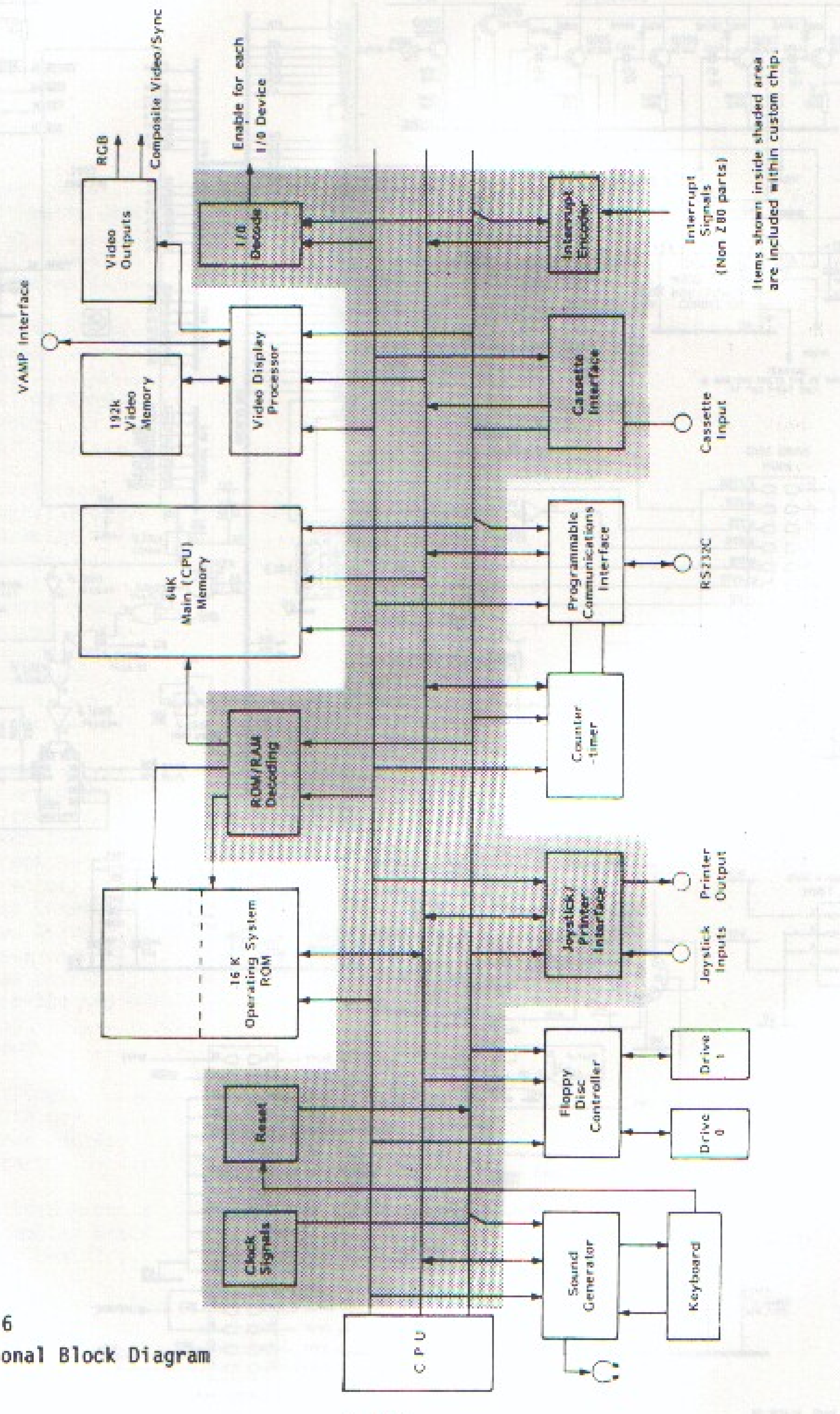
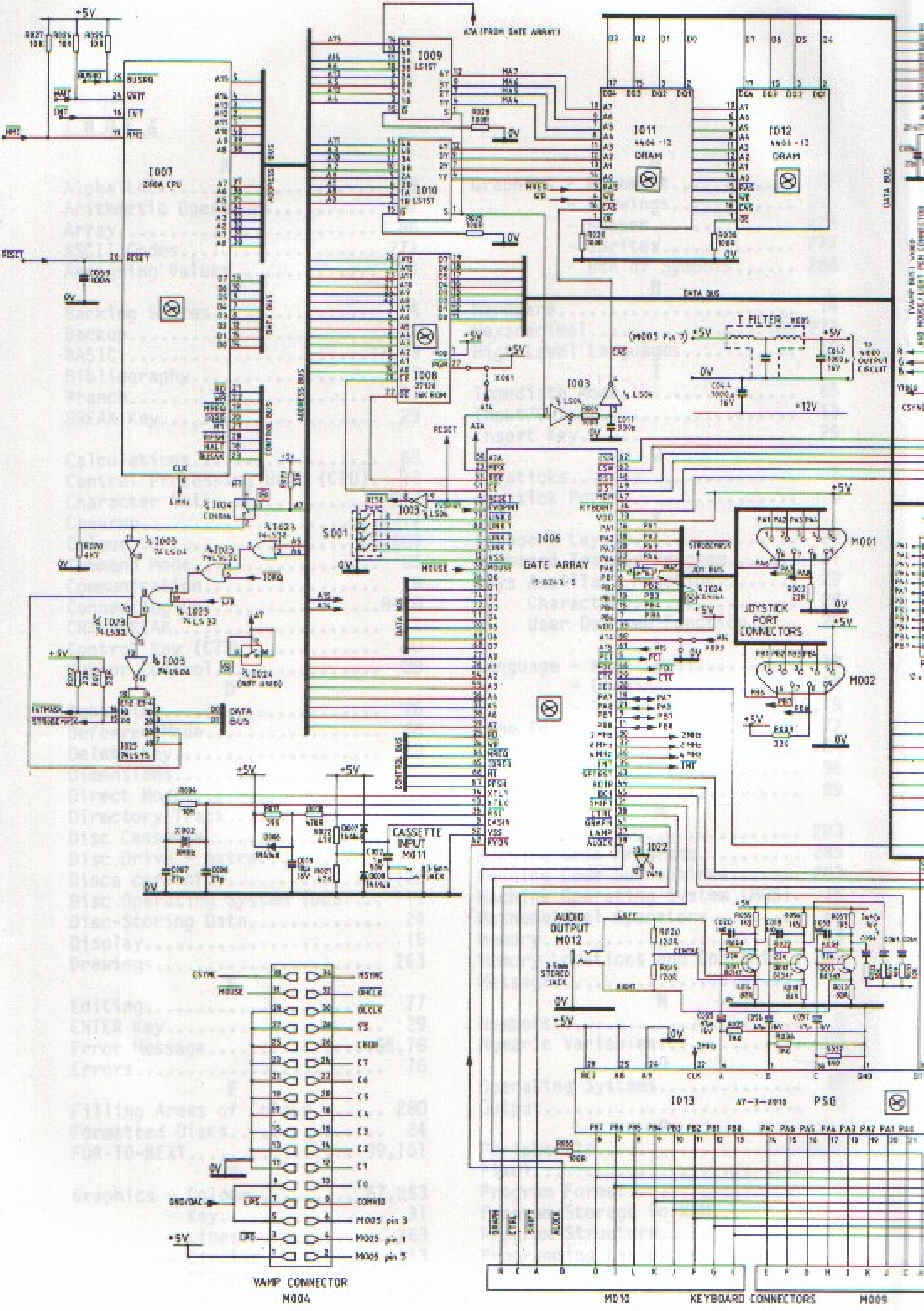


Fig P.6
Functional Block Diagram


+5V
I007
Z80A CPU
BUSRQ
WAIT
INT
NMI
RESET
ADDRESS BUS
DATA BUS
CONTROL BUS
A15 (FROM GATE ARRAY)
I009
LS157
I010
LS157
I008
27128
16K ROM
I011
4464-12
DRAM
I012
4464-12
DRAM
DATA BUS
I003
LS04
I006
GATE ARRAY
40007
T FILTER
TO VIDEO OUTPUT CIRCUIT
+12V
0V
I004
I005
74LS04
I025
74LS32
I023
74LS32
I029
74LS32
I024
I026
74LS175
S001
JOYSTICK PORT CONNECTORS
M001
M002
CASSETTE INPUT
M011
3.5mm JACK
AUDIO OUTPUT
M012
3.5mm STEREO JACK
LEFT
RIGHT
I022
I013
AY-3-8910
PSG
DATA BUS
VAMP CONNECTOR
M004
CSYNC
MOUSE
HSYNC
CRTC
C7
C6
C5
C4
C3
C2
C1
C0
M005 pin 3
M005 pin 1
M005 pin 5
M010
KEYBOARD CONNECTORS
M009

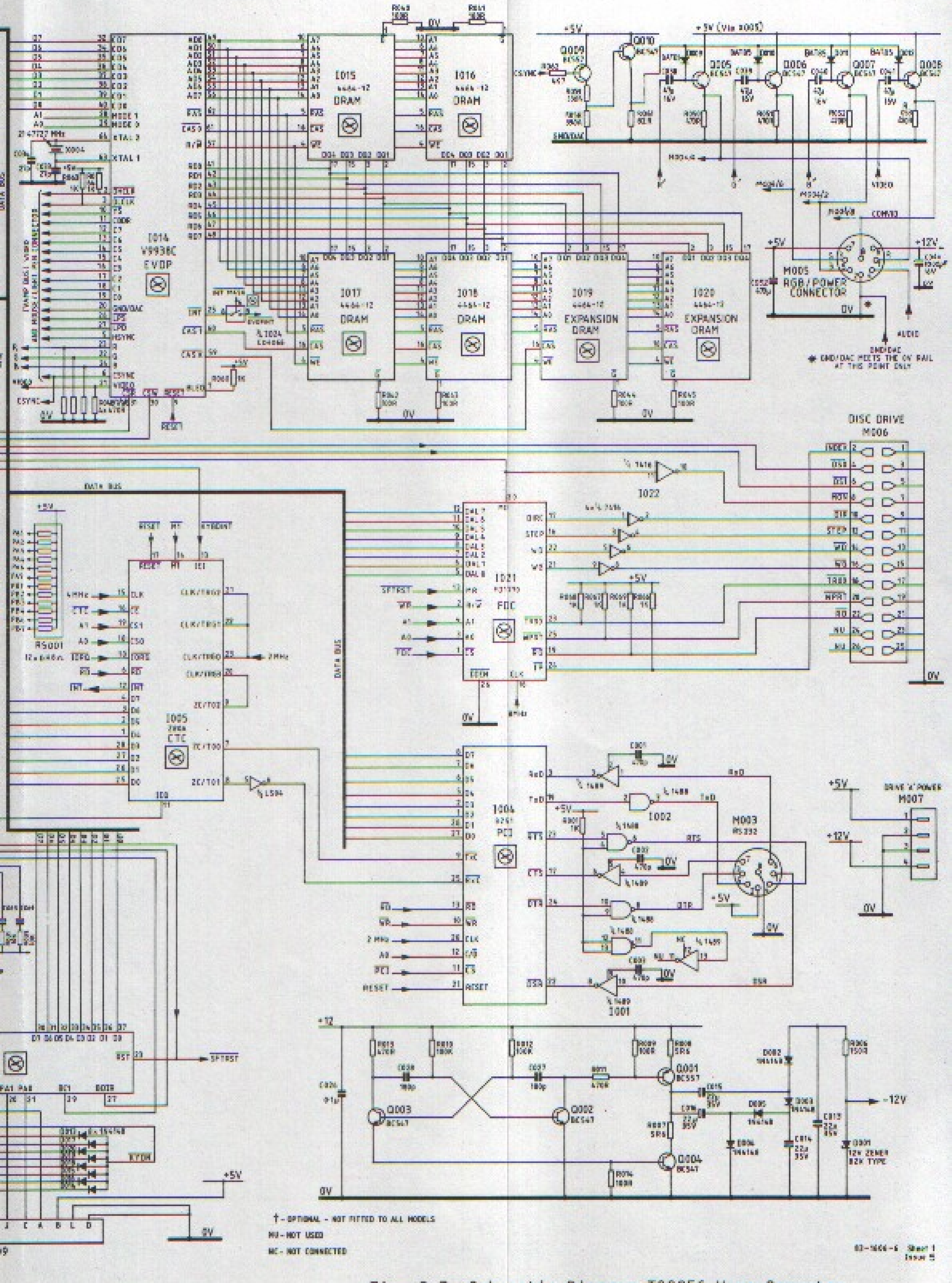


Fig. P.7. Schematic Diagram TCS256 Home Computer

INDEX