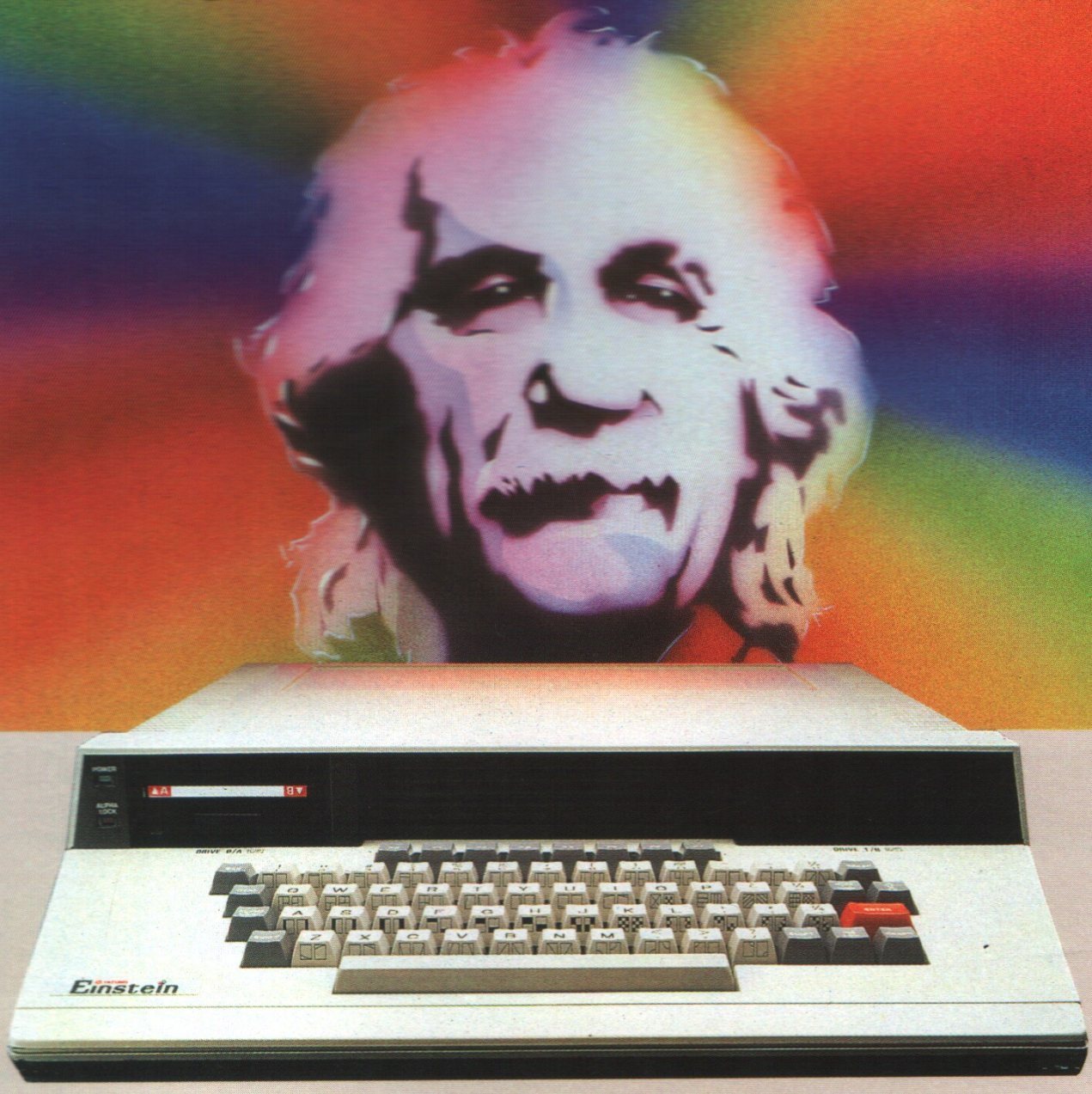
An Introduction to



TATUNG

Einstein

COLOUR MICRO COMPUTER

Acknowledgements

Written By: Alan Stancliffe
Edited by: R.M. Clarke

Our grateful thanks to Crystal Research Limited of Torquay, for their kind permission to reproduce extracts from their manuals, and for their help in checking this publication.

ISBN:1-85086-000-9. 1st Edition 1984
ISBN:
ISBN:
ISBN:

Tatung (UK) Ltd.,
Computer Division,
BRIDGNORTH,
Shropshire. WV15 6BQ

First Printed in 1984 by:
Spellman Walker Ltd.,
Bradford, West Yorkshire.
ENGLAND.

CONTENTS

APPENDICES

INTRODUCTION

This manual describes the features and operation of the **EINSTEIN.** It is divided into three distinct sections, the first of which deals with setting up the computer, general information, loading BASIC and familiarisation of the keyboard.

The second section provides an introduction to BASIC and includes several exercises for the beginner.

The third section describes some of the more advanced facilities of EINSTEIN including details of Graphics, Colour and Sound.

In producing this manual we have assumed no previous knowledge of the material and, therefore, offer lengthy explanations to accommodate the inexperienced reader (THE USER). At the end of each topic a summary has been included which contains the main elements of information and it would be sufficient for the experienced user to consult these rather than the full text.

A chart is provided which guides the reader through this manual according to individual experience. We would stress the importance of reading carefully through the information relating to setting up EINSTEIN, regardless of previous knowledge.

The inexperienced user should adhere to the sequence of the chart when working through the manual as information presented in one section often expands on knowledge gained in previous sections.

For the absolute novice we have included a very simple figurative explanation of the elements of a computer.

SECTION A

1

SETTING UP

UNPACKING

Carefully unpack your EINSTEIN and check that you have the following items.

1. Computer unit with one integral disc drive.
2. UHF lead (for connecting to T.V. aerial socket).
3. A SYSTEM MASTER disc in a plastic case.
4. An EINSTEIN "INTRODUCTORY MANUAL"
5. An EINSTEIN "BASIC REFERENCE MANUAL"
6. Quick reference card for BASIC
7. An EINSTEIN DOS/MOS MANUAL

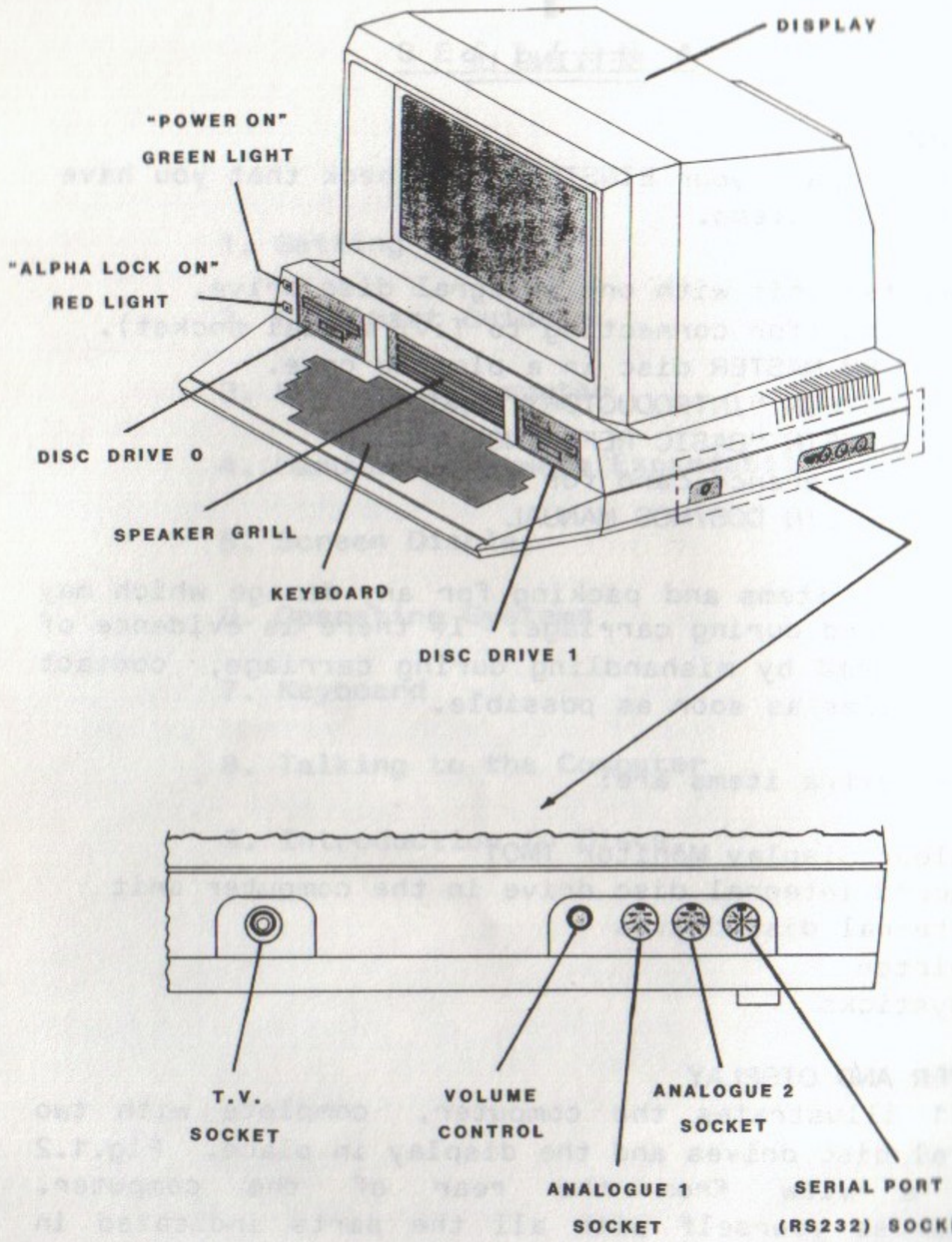
Inspect all items and packing for any damage which may have occurred during carriage. If there is evidence of damage caused by mishandling during carriage, contact your supplier as soon as possible.

Optional extra items are:

1. Colour Display Monitor TMO1
2. Second integral disc drive in the computer unit
3. External disc drives
4. Printer
5. Joysticks

COMPUTER AND DISPLAY

Fig.1.1 illustrates the computer, complete with two integral disc drives and the display in place. Fig.1.2 shows a view from the rear of the computer. Familiarise yourself with all the parts indicated in the diagrams.



COMPUTER WITH DISPLAY IN PLACE

Fig.1.1

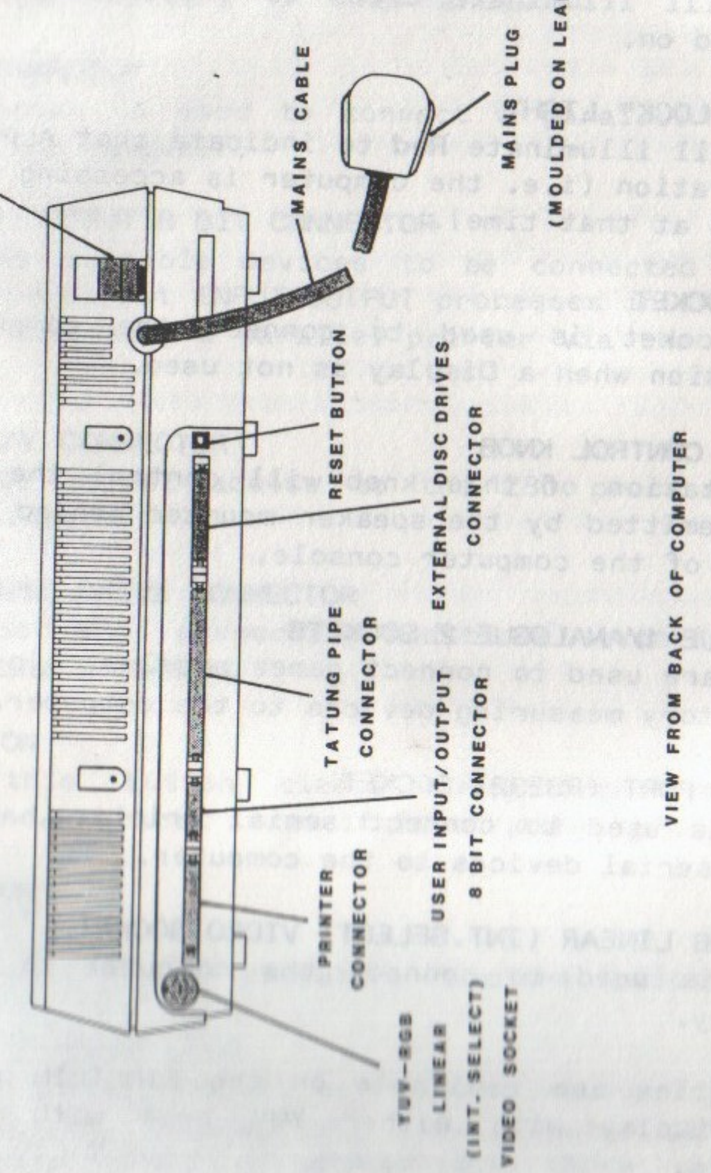


Fig.1.2

"POWER ON" LIGHT
This will illuminate **Green** to indicate the power is switched on.

"ALPHA-LOCK" LIGHT
This will illuminate **Red** to indicate that ALPHA-LOCK is in operation (i.e. the computer is accessing upper case letters at that time).

T.V. SOCKET
This socket is used to connect the computer to a television when a Display is not used.

VOLUME CONTROL KNOB
The rotation of this knob will control the volume of sound emitted by the speaker mounted behind the centre grille of the computer console.

ANALOGUE 1/ANALOGUE 2 SOCKETS
These are used to connect games paddles, joysticks and laboratory measuring devices to the computer.

SERIAL PORT (RS232) SOCKET
This is used to connect serial printers, modems and other serial devices to the computer.

YUV-RGB LINEAR (INT.SELECT) VIDEO SOCKET
This is used to connect the computer to a suitable display.

Facilities are available on the EINSTEIN for feeding the Display with either YUV or, with RGB output signals.

Normally as supplied, the computer is set to provide YUV via this socket, located at the rear of the computer housing, to accomodate the TATUNG Display.

If RGB format is required, refer to Appendix I (display Monitor Signal Options) of this manual.

PRINTER CONNECTOR
This connector is used to connect a parallel printer device to the computer.

USER INPUT/OUTPUT 8 BIT CONNECTOR
This allows suitable devices to be connected to the computer for direct INPUT/OUTPUT processes in a similar manner to accessing a parallel printer via the printer socket.

TATUNG PIPE CONNECTOR
This allows direct access to the Z80 processor for future expansion.

EXTERNAL DISC DRIVE CONNECTOR
This is used when connecting additional external disc drives to the computer.

RESET BUTTON
Pressing this button clears everything from the computer and restarts it as if from "power up".

INSTALLATION

Before setting up your computer, read the following:

1. **COMPUTER MAINS LEAD**
   If the socket outlets to be used are not suitable for the plug supplied with the computer it should be cut off and an appropriate three pin plug fitted.

   NOTE: The plug severed from the mains lead must be destroyed, as a plug with bared flexible cord is hazardous if engaged in a live socket outlet.

The mains lead contains three wires coloured in accordance with the following code:-

BLUE - Neutral
BROWN - Live
GREEN WITH YELLOW STRIPE - Earth

The colours of these wires may not correspond with the coloured markings identifying the terminals of your plug, therefore **CONNECT AS FOLLOWS:-**

a) The wire coloured BROWN must be connected to the terminal marked L or coloured RED.

b) The wire coloured BLUE must be connected to the terminal marked N or coloured BLACK.

c) The wire coloured GREEN AND YELLOW must be connected to the terminal marked E or coloured GREEN, or coloured GREEN AND YELLOW, or marked with the earth symbol ⏚.

Use a 3 amp fuse approved by ASTA to BS1362 ie. carries the following mark.

Always replace the fuse cover, never use the plug with the fuse cover omitted.

WARNING - THIS APPLIANCE MUST BE EARTHED.

2. **DISPLAY MAINS LEAD**
Refer to the operating instructions supplied with the display unit for installation procedure.

3. **VENTILATION**
It is important that all ventilation slots in both the COMPUTER and DISPLAY cabinets are clear of any obstruction so that a free flow of air is available. Do not cover with mats or material of any kind, and avoid placing near heating devices.

Always stand the computer on a FIRM, FLAT surface to allow a free passage of air to the ventilation slots in the underside of the cabinet. Do **NOT** stand it on upholstered surfaces or soft cushions, etc., as these might obstruct the required air flow necessary for efficient ventilation.

4. **WARNING**
To prevent shock or fire hazard, do not expose the equipment to rain or moisture. If such exposure occurs, remove the plug from the mains power point and have the exposed unit checked by a competent technician.

5. **PROTECTION AND SAFETY**
Should any unit fail to operate after turning on the power or during use, switch off and remove the plug from the mains power point.

Do not operate the units in a faulty condition as this may cause further damage to the system and could be hazardous.

Ask your dealer or Service Centre for assistance.

6. **CLEANING**
The computer and display cabinets may be cleaned periodically with a soft clean damp cloth. To avoid damage to painted surfaces proprietary cleaners and polishes should **not** be used.

SETTING UP

If using a DISPLAY:-

Connect the computer to the display as illustrated in Fig.1.3, using the lead supplied with the display unit.

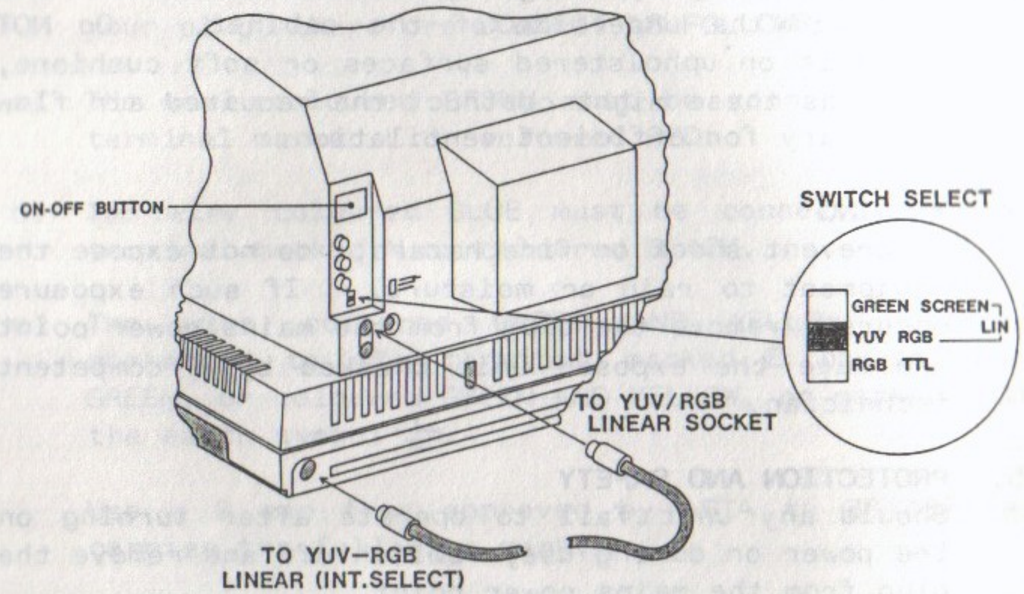


Fig.1.3

The YUV-RGB socket at the rear of the computer is connected to the YUV/RGB socket at the rear of the display using the lead supplied with the unit.

The LINEAR YUV/RGB socket is the upper of the two sockets situated at the rear of the display, the lower one being a TTL/RGB socket for use with other computer systems.

The switch located above the two sockets should be in the CENTRE position when using the LINEAR YUV/RGB socket (i.e. with EINSTEIN) and the LOWER position when using the RGB.TTL socket.

If using a TELEVISION:-

* First remove the aerial lead from the rear of the T.V.

* Connect the computer T.V. socket to the aerial socket of the T.V. as illustrated in Fig.1.4, using the U.H.F. lead supplied with the computer.

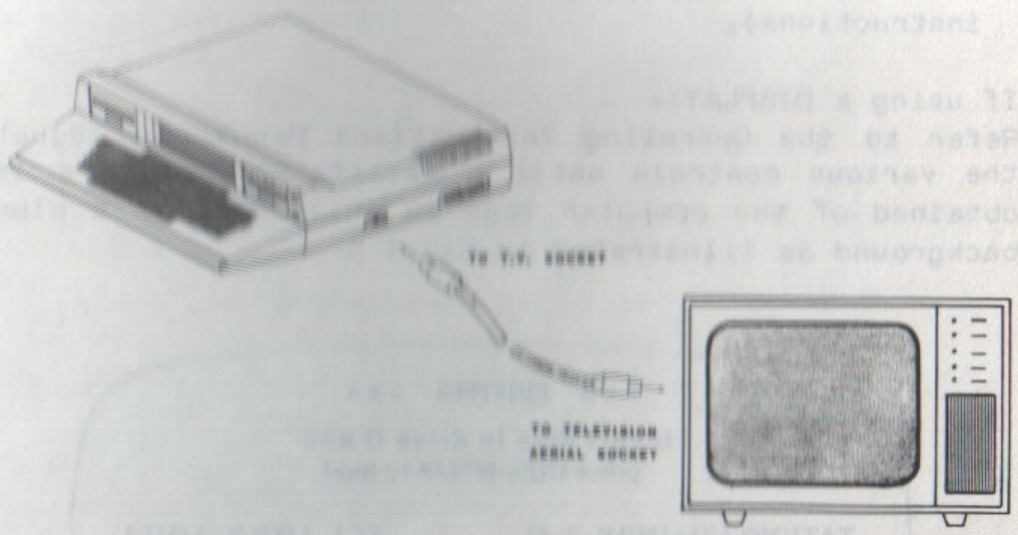


Fig.1.4

POWER UP

Having set up your computer and display/T.V. you are now ready to switch on (i.e. POWER UP).

* Plug the mains lead from the rear of the computer into a mains power point (household 13 Amp socket) and likewise for the display/T.V. (see appropriate operating instructions).

* Turn the computer on by operating the main power switch at the rear of the unit. The ON position is indicated by a red bar on the switch rocker. The "Power On" light on the front fascia will illuminate when ON.

* Turn on the display/T.V. (see appropriate operating instructions).

If using a DISPLAY:-
Refer to the Operating Instructions Manual to adjust the various controls until a satisfactory picture is obtained of the computer logo in white text on a blue background as illustrated in Fig.1.5.

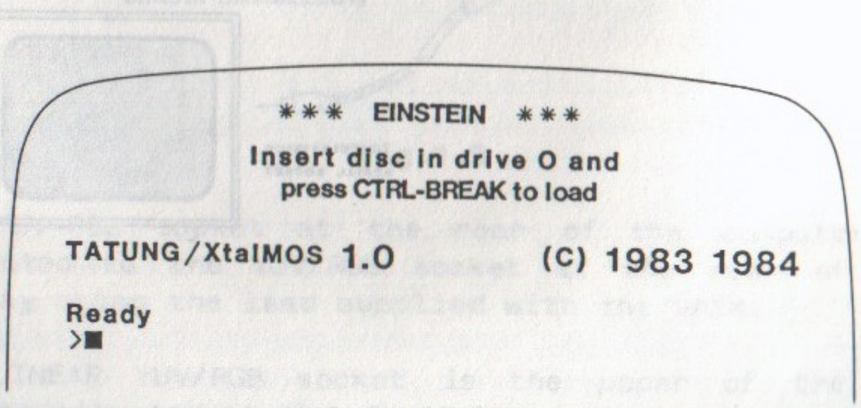


Fig.1.5

If using a TELEVISION:-

The television may need to be "tuned in" to the computer, using one of the vacant channel selectors, until a picture appears as above, bearing the computer logo in white text on a blue background.

Refer to your television Operating Instructions Manual for information relating to the tuning procedure. Having decided on a particular channel selector, it should be "tuned in" to channel 36.

Some televisions may operate more successfully by making use of the A.V. facility (when provided). Refer to appropriate operating instructions for this.

2

FLOWCHART GUIDE

The flowchart guide on the following page is designed to guide the user through this manual.

The following points should be noted as an aid to using the guide correctly.

1. Rectangular Boxes contain instructions for you to follow.

2. Diamond Boxes contain questions which require a YES/NO answer.

3. Read the instructions in the boxes carefully and follow the route from one box to the next as indicated by the arrows.

4. Diamond Boxes give an option of two routes. The route you take will be selected according to your answer to the question, and is indicated on the chart by YES and NO directions.

5. Plan your route through the guide before proceeding any further.

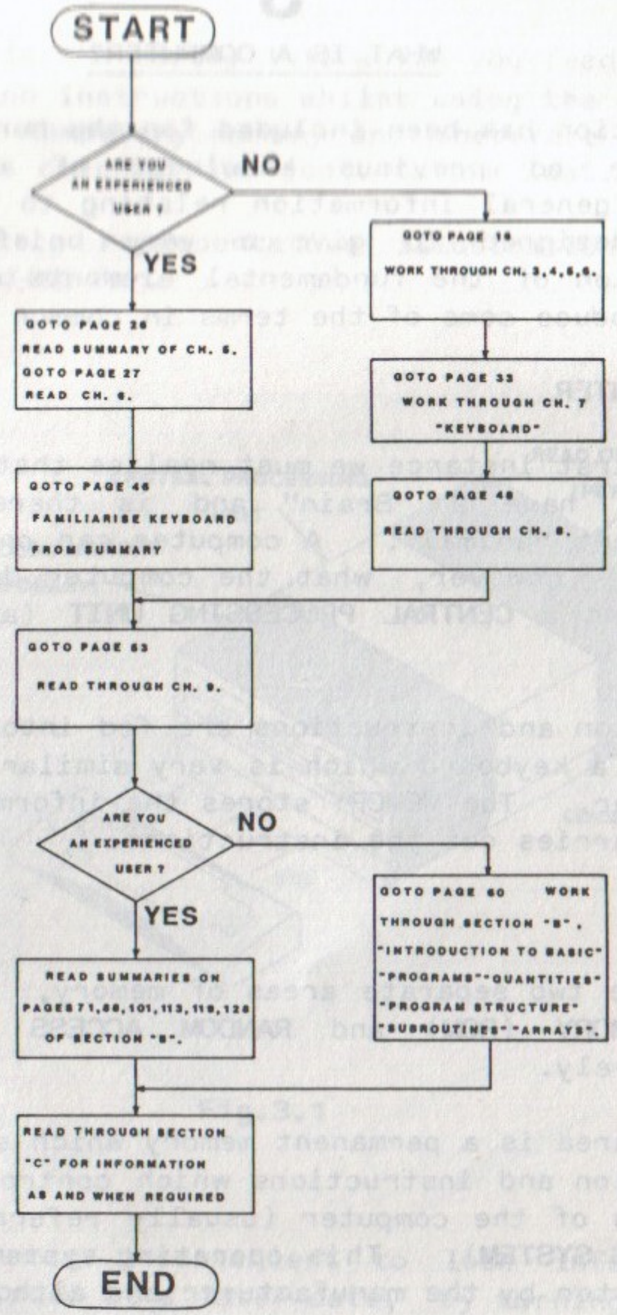


Fig.2.1

3

WHAT IS A COMPUTER?

This section has been included for the benefit of those who have no previous knowledge of computers and provides general information relating to the EINSTEIN. It is designed to give a very brief and simple explanation of the fundamental elements of a computer and introduce some of the terms in common use.

THE COMPUTER

In the first instance we must realise that the computer does not have a "Brain" and is therefore **not** an intelligent "animal". A computer can only do what it is **told.** However, what the computer does have is a **MEMORY** and a **CENTRAL PROCESSING UNIT** (abbreviated to C.P.U.).

Information and instructions are fed into the computer by using a keyboard which is very similar to that of a typewriter. The MEMORY stores the information and the C.P.U. carries out the instructions.

MEMORY

There are two separate areas of memory, known as **READ ONLY MEMORY (ROM)** and **RANDOM ACCESS MEMORY (RAM)** respectively.

The ROM area is a permanent memory which stores all the information and instructions which control the general operation of the computer (usually referred to as the **OPERATING SYSTEM**). This operating system is put into the computer by the manufacturer and although there are variations from one machine to another it cannot normally be affected by you. Therefore no matter what you do with your computer the ROM will remain permanently unchanged.

The RAM area is the memory into which you feed your own information and instructions whilst using the computer. It is only a **temporary** memory and therefore when you switch off the computer the contents are lost.

Fig.3.1 gives a representative illustration of the points covered so far.

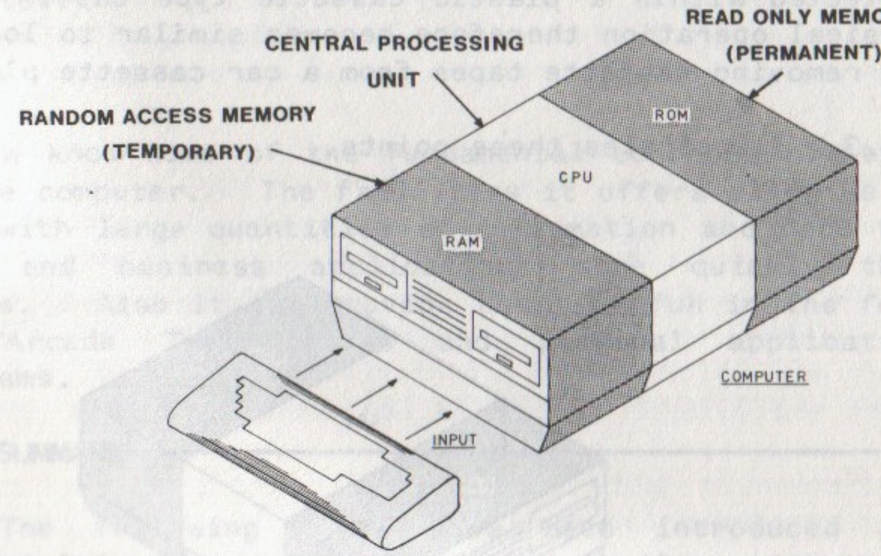


Fig.3.1

BACKING STORE

It would be most inconvenient to lose information, which you may need at a later date, by turning off the computer. Therefore we incorporate a system which is known as a "BACKING STORE" This is simply a method of storing the information which was in the RAM so that it can be used again on another occasion.

Many home computers use cassette tape as a backing store. EINSTEIN however uses DISC storage. The information can be stored on them and retrieved as with cassette tapes.

Just as tapes need a tape recorder for operation, then DISCS need DISC DRIVE units. EINSTEIN uses disc storage only.

EINSTEIN uses a 3 inch compact floppy disc, which is protected within a plastic cassette type case. The physical operation therefore becomes similar to loading and removing cassette tapes from a car cassette player.

Fig.3.2 illustrates these points.

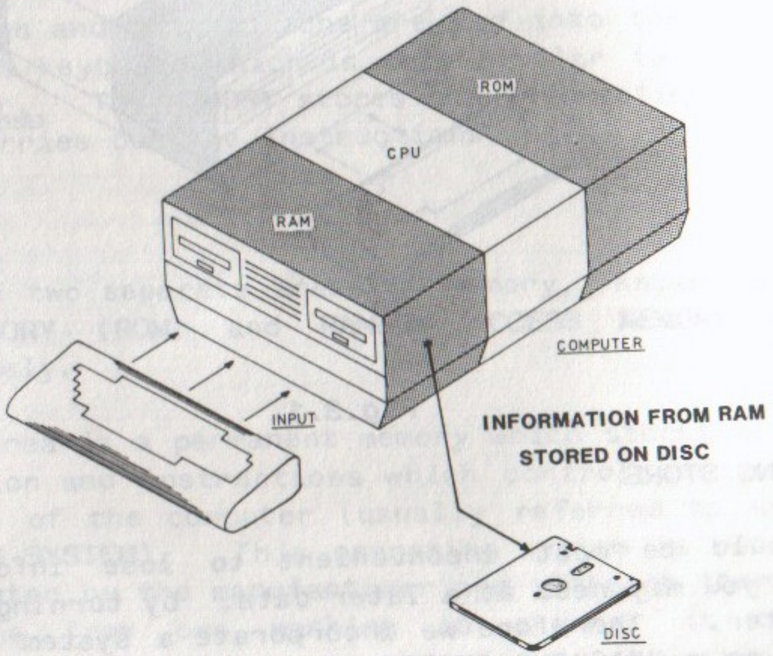


Fig.3.2

The advantage of Disc Storage is the facility of "random access" to information which is a much quicker process than the "sequential access" associated with tapes.

CENTRAL PROCESSING UNIT

The CPU is that part of the machine which is responsible for all the calculating and processing of information, and might be considered the "heart" of the computer.

USE

We now know some of the fundamental concepts relating to the computer. The facilities it offers allow us to deal with large quantities of information and data for home and business applications more quickly than before. Also it can provide hours of fun in the form of "Arcade Type" games and personal application programs.

SUMMARY

The following terms have been introduced and explained. You should now be familiar with their use. If not, please read through the appropriate section again.

a) KEYBOARD
b) ROM (Read Only Memory)
c) RAM (Random Access Memory)
d) CPU (Central Processing Unit)
e) OPERATING SYSTEM
f) BACKING STORE
g) DISCS and DISC DRIVES

4

HARDWARE/SOFTWARE EXPANDABILITY

HARDWARE/SOFTWARE

All the solid physical objects incorporated within the system are known as the **HARDWARE**, e.g. Computer, Printer, Display, Keyboard etc.

The **SOFTWARE** is a sequence of instructions and information which enables the computer to perform a particular task. Software is usually contained on disc, but may, in some cases, be fed into the computer via one of the ports (at the rear of the computer).

The essential items of HARDWARE are the **COMPUTER**, the **KEYBOARD** and some form of **DISPLAY** (VDU or TV). This can then be expanded to incorporate the facilities illustrated in Fig.4.1, which include a printer, up to three extra disc drives, two joy sticks for games and a variety of applications involving the 8-bit user port and TATUNG 'PIPE'. (Further details of the User Port and TATUNG 'PIPE' are included in chapter 16)

A printer is quite useful for producing printed copies of any particular results the computer might give.

INPUT

You are the person using the computer and putting information into it via the KEYBOARD. In other words you are a **USER** and the keyboard is an **INPUT** device.

OUTPUT

The **DISPLAY** and **PRINTER** are facilities for displaying what is coming out of the computer either on screen or paper. They are therefore known as **OUTPUT** devices.

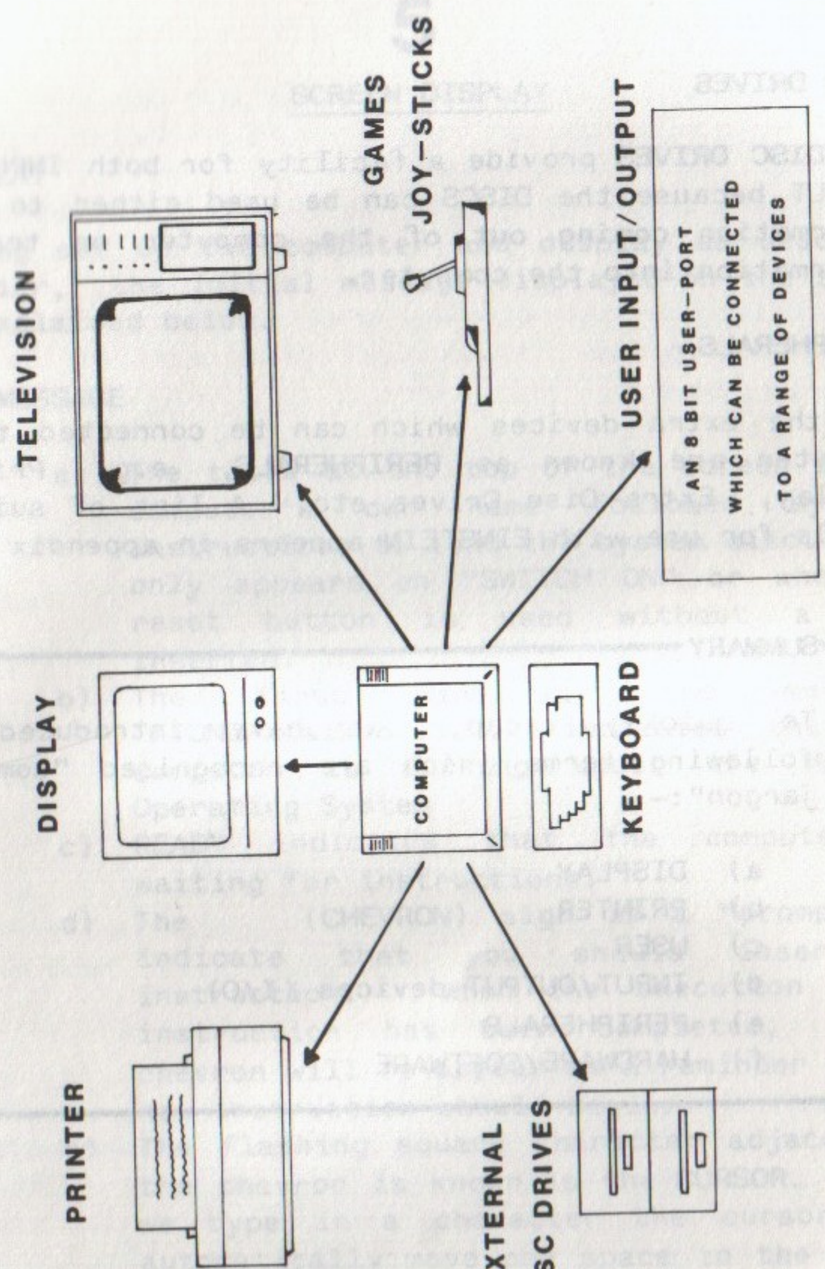


Fig.4.1

DISC DRIVES

The **DISC DRIVES** provide a facility for both INPUT and OUTPUT because the DISCS can be used either to store information **coming out** of the computer or transfer information **into** the computer.

PERIPHERALS

All the extra devices which can be connected to the computer are known as **PERIPHERALS**, e.g. Printer, Display, Extra Disc Drives etc. A list of suitable models for use with EINSTEIN appears in appendix G.

SUMMARY

To quickly recap, we have introduced the following terms which are recognised "computer jargon":-

a) DISPLAY
b) PRINTER
c) USER
d) INPUT/OUTPUT devices (I/O)
e) PERIPHERALS
f) HARDWARE/SOFTWARE

5

SCREEN DISPLAY

DISPLAY

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. This only appears on "SWITCH ON" or when the reset button is used without a disc inserted.
b) The first line of the message, TATUNG/XtalMOS 1.0, indicates that the computer is working under the **Machine Operating System**
c) READY indicates that the computer is waiting for instructions.
d) The > (CHEVRON) sign is a "prompt" to indicate that you should insert an instruction. When the execution of an instruction has been completed, the chevron will re-appear as a reminder that a new instruction should begin.
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 certain keys on the keyboard.

SCREEN GRID

It will help to understand the positioning of characters on the screen if we know a little about the "Screen Grid".

We can imagine the display area of the screen to be mapped out by an "invisible" grid similar to that shown in Fig.5.1.

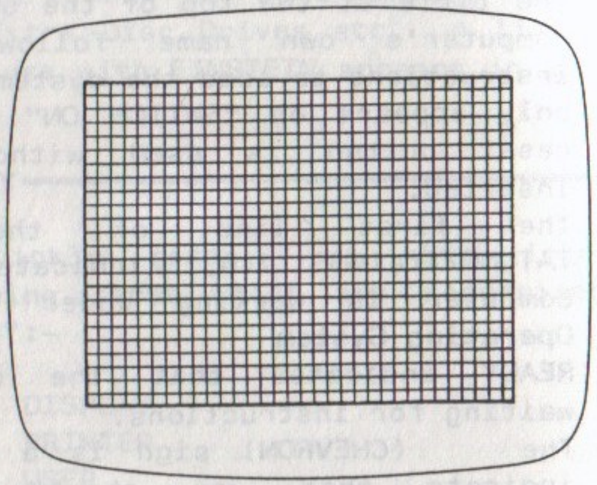


Fig.5.1

The grid does not cover the full area of the screen and a border is therefore left around the perimeter. This grid represents all the possible **CHARACTER** positions which can be displayed. Each small "square" of the grid corresponds to one character **CELL.** These character cells are then sub-divided again into small squares known as **PIXELS.** The PIXEL is the smallest unit of display possible on the screen and all character shapes are formed by displaying the necessary combination of pixels within a cell to produce the required figure.

There are two "GRID MODES" available being referred to as 32 COLUMN DISPLAY and 40 COLUMN DISPLAY.

In 32 COLUMN display the grid is 32 cells across by 24 cells down, each cell containing 64 pixels (8 x 8).

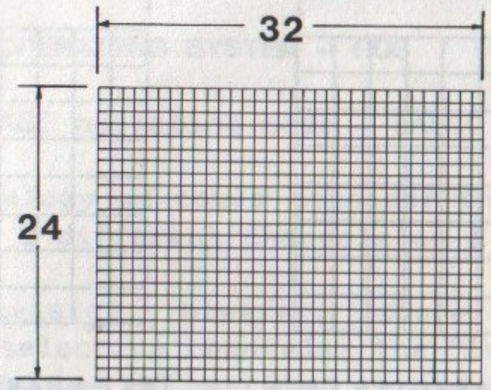


Fig.5.2

In 40 column display the grid is 40 cells across by 24 cells down, each cell containing 48 pixels (6 x 8).

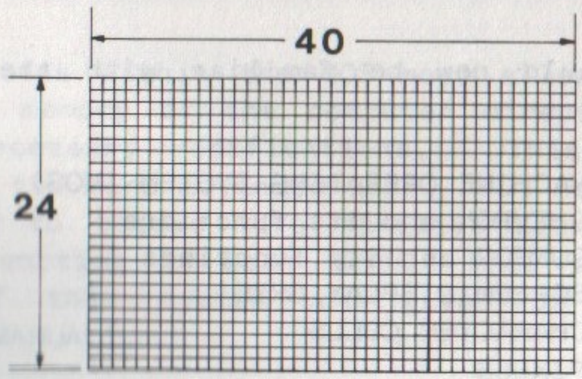
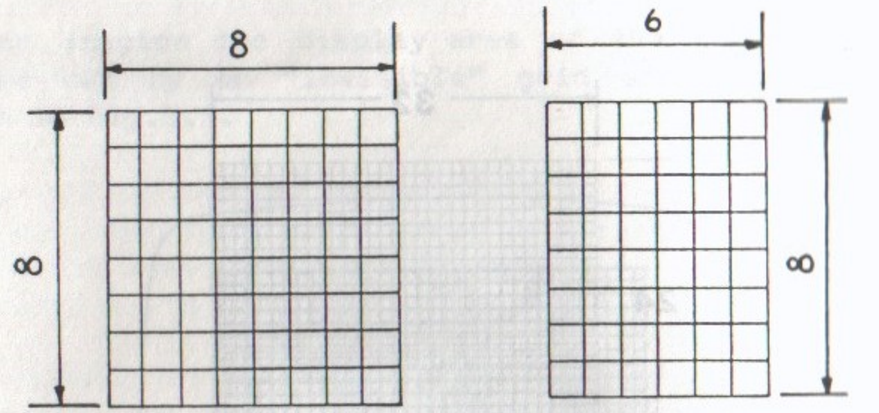


Fig.5.3

We see therefore that the 40 column Character cells are smaller in width than the 32 column Character cells as illustrated in Fig.5.4.



GRAPHICS CHARACTER CELL

(8X8 PIXELS)

TEXT CHARACTER CELL

(6X8 PIXELS)

Fig.5.4

SUMMARY

You should now be familiar with the following terms:-

a) MACHINE OPERATING SYSTEM (MOS)
b) CHEVRON and its functions
c) CURSOR and its functions
d) SCREEN DISPLAY GRID
e) CHARACTER CELLS
f) PIXELS

6

THE OPERATING SYSTEMS

There are **three** different levels of operation available with the EINSTEIN, these being:-

1. The **MACHINE CODE DISPLAY** section of the "Machine Operating System" - **MOS**

2. The **DISC OPERATING SYSTEM - DOS**

3. The **LANGUAGE** operating mode - BASIC is supplied.

To make an analogy we could consider these "levels" to be similar to 3 different channels on a television.

Unlike a television, however, there are no physical switches for selecting channels; the "levels" (systems) are selected by use of certain commands which are typed in from the keyboard (these commands in fact act like switches to "turn on" a particular system).

Various different facilities are available within these "levels" (systems) which are self-contained, but there is some interaction between them.

MOS:

The facilities within the MOS mode allow the user to access the memory of the computer directly and carry out any necessary modifications. This is a useful aspect for the more advanced user, but new users would be advised to become more familiar with the machine before attempting to work within this area. (More details of this system are given in the DOS/MOS REFERENCE MANUAL).

The prompt message displays "Ready" with a chevron below when in MOS.

DOS:

The DOS concerns itself with accessing and processing of Discs. There are limited facilities available within this mode but more often it is used in conjunction with the particular LANGUAGE in operation within the machine.

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 and the language mode are complete. (More details of DOS are given in the DOS/MOS REFERENCE MANUAL).

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

LANGUAGE:

The LANGUAGE "mode" is perhaps the most used system for general purposes. BASIC is supplied, but other languages can be used with EINSTEIN. Newcomers should familiarise themselves with the BASIC language before attempting to work in either MOS or DOS exclusively (some interaction will be found between BASIC, MOS and DOS).

The prompt message displays "Ready" with cursor below when in language mode.

LOADING THE LANGUAGE (BASIC)

As mentioned earlier, when the machine is initially turned on it will "power up" in the operating system (ie. MOS).

The following message should be present on the screen:-

Insert disc in drive A and
press CTRL - BREAK to load.

Remove the SYSTEM MASTER disc from the protective case and place it in the disc-drive as shown in Fig.6.1, with the label "SYSTEM MASTER" uppermost.

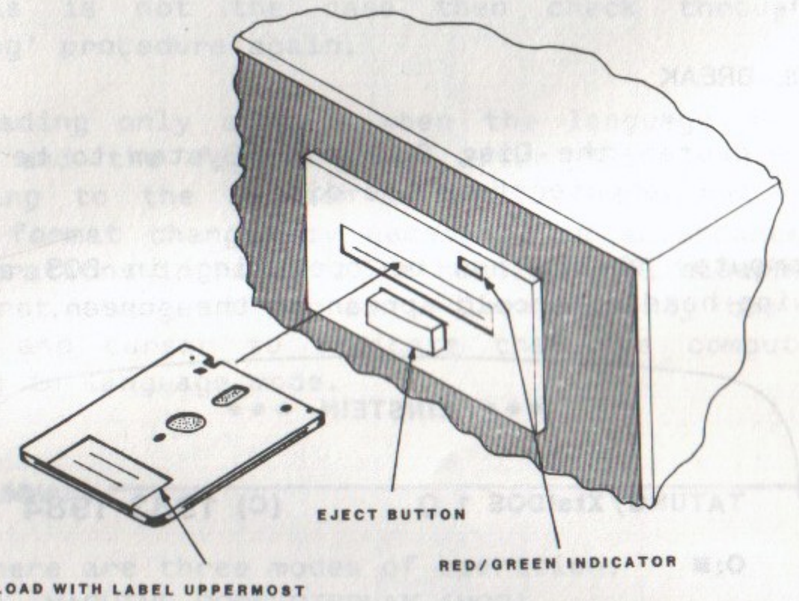


Fig.6.1

i) 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 GREEN for side A and RED for side B. If loaded with the "SYSTEM MASTER" side uppermost the light should be GREEN.

iii) To remove the disc press the eject button. This will cause the cassette to release and jump out slightly so that it can be removed by hand.

Having correctly inserted the disc press the key labelled CTRL and the key labelled BREAK simultaneously, as indicated by the message on the screen.

CTRL-BREAK

This 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 ***

TATUNG/XtalDOS 1.0      (C) 1983 1984

O:■
```

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

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

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

```
TATUNG/Xtal BASIC 4.0      (C) 1983 1984

SIZE: 00000
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 language mode.

SUMMARY

There are three modes of operation:
a) MACHINE CODE DISPLAY (MOS)
b) DISC OPERATING SYSTEM (DOS)
c) LANGUAGE SYSTEM - LOADING THE LANGUAGE

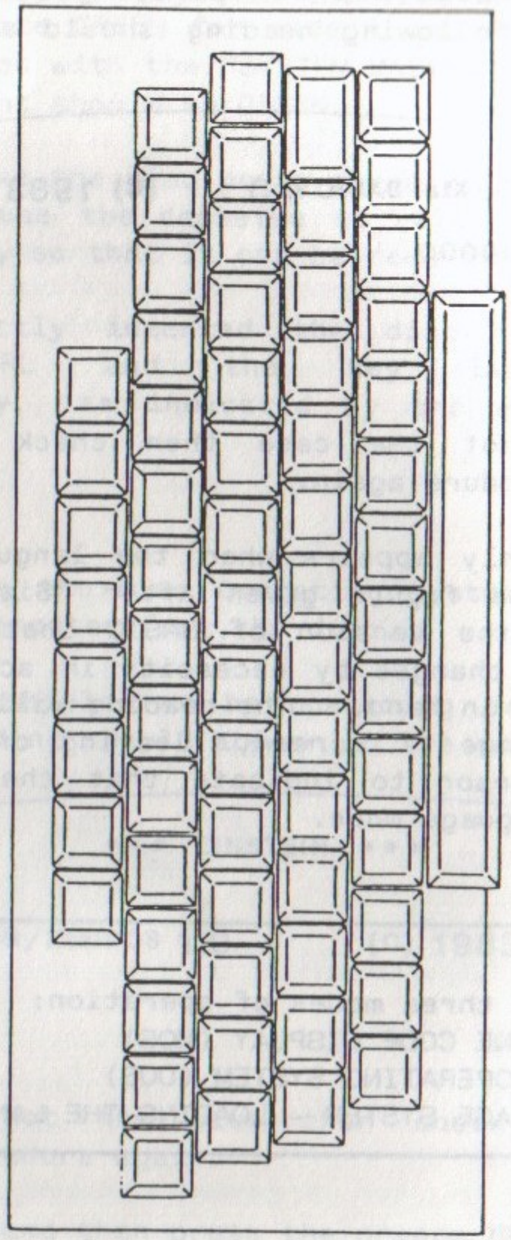


Fig.7.1

7

KEYBOARD

LAYOUT

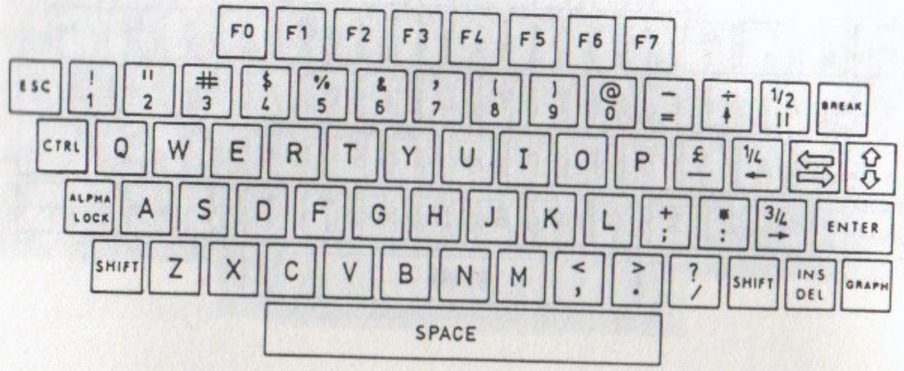


Fig.7.2

Fig.7.2 above represents the layout of the keyboard. The labelling is very similar to a typewriter, with the addition of a few extra keys. The LIGHT GREY keys are all **CHARACTER** keys, the DARK GREY keys are **FUNCTION** keys, and there is one ORANGE key labelled **ENTER.**

The keys fall into three distinct groups:-

a) "User Defined" Function Keys - DARK GREY
b) Character Keys - LIGHT GREY
c) Ancillary Function Keys - DARK GREY/ ORANGE

"USER DEFINED" FUNCTION KEYS

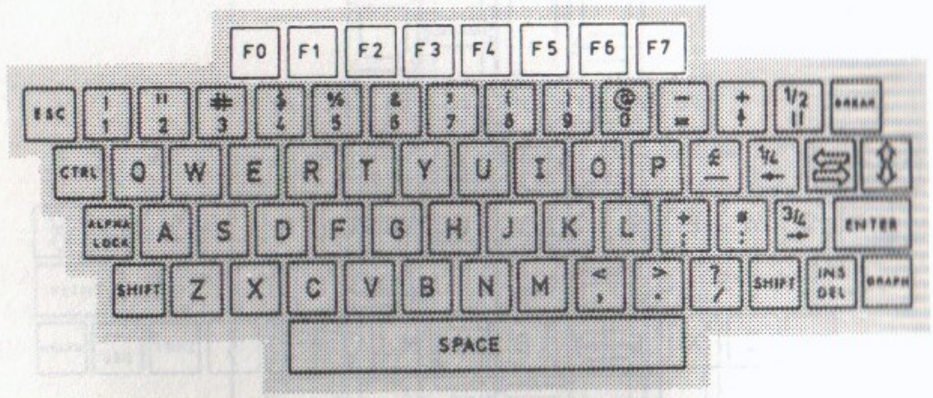


Fig.7.3

The top row of eight grey keys labelled F0 to F7 are "user defined" function keys. Each key can be programmed by the user to have a very clearly defined function. The details of programming these functions are described in the BASIC manual and should be left by the inexperienced user until the following sections have been covered:-

a) Familiarisation of the remaining keys
b) Introduction to BASIC

CHARACTER KEYS

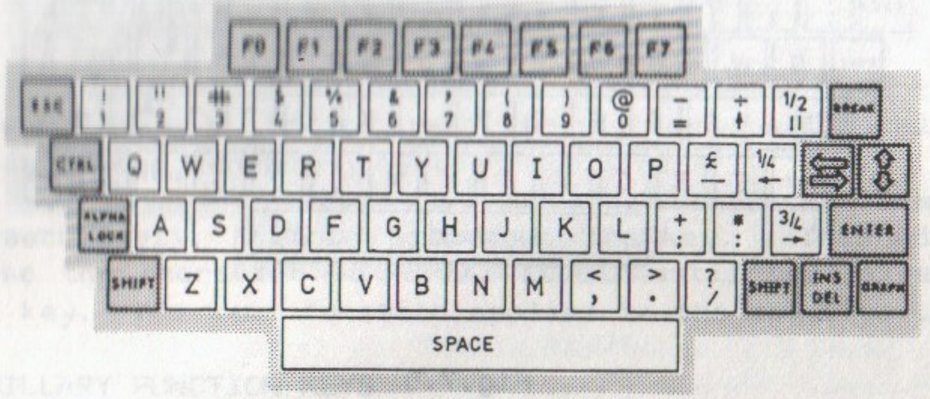


Fig.7.4

This group includes the numbers (**NUMERIC**), letters (**ALPHABETIC**), symbols and fraction keys used to "type in" information and instructions to the computer. Each key can access two characters as indicated by the labelling on the top surface, these being either upper or lower case letters for the alphabetic keys (upper case only are labelled).

The computer is set to access upper case letters when first switched on (Powered Up), and the characters illustrated on the lower half of the key-top surface. To access the "second" character we incorporate the use of the SHIFT key (one of the ancillary functions).

It can be seen from Fig.7.5 that there are two shift keys. This is for convenience only; both keys perform the same function.

Fig.7.5

The shift key must be held down whilst the appropriate key is pressed to access the "second" (SHIFTED) character required.

KEYBOARD TRAINER

A KEYBOARD TRAINER program is supplied on the MASTER disc and this can be loaded into the computer by typing in the following exactly as shown.

```
RUN"KEYBOARD"
```

Then press the orange ENTER key and the program will load and RUN. Follow the instructions presented on the screen by the program.

DEMONSTRATION PROGRAM

When you have completed the keyboard trainer exercises, type in RUN"DEMO" and then press the orange ENTER key. This will load a demonstration program which illustrates some of the facilities of EINSTEIN. Follow the instructions presented on the screen.

When you have completed the DEMO program read through the following sections so as to reinforce and extend the knowledge you have gained.

If you complete a full line of characters across the screen you will notice that the cursor automatically moves to the beginning of the next line.
If you wish to type a character several times consecutively, simply hold down the key. This will cause the character to REPEAT itself until you release the key. The same function applies to the space bar.

ANCILLARY FUNCTION KEYS

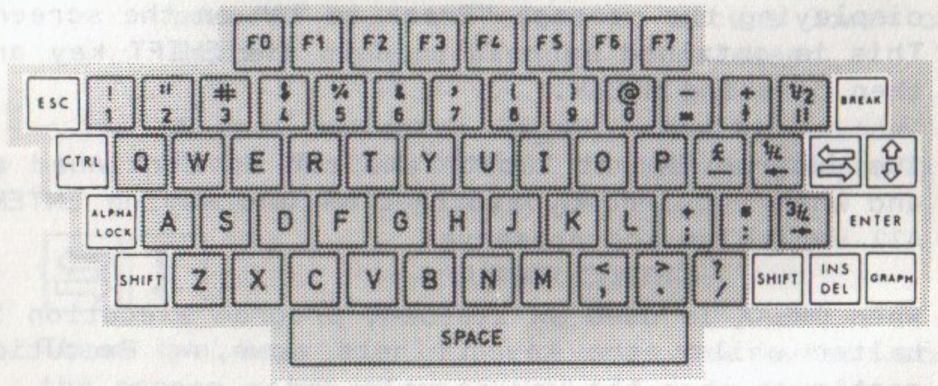


Fig.7.6

These keys provide us with various useful facilities. The SHIFT keys have already been discussed, others are as follows.

a)  - **ENTER**

Under normal circumstances the ENTER key is used at the end of a valid instruction in order to enter that instruction for processing. It invokes a "carriage return line feed" (CRLF) ie. transfers the cursor to the beginning of a new line. If an ERROR message is displayed, there is a mistake in the instruction format. Information on how to deal with error messages is given in later sections.

b)  - **BREAK**

The normal use of this function is to cause a break during the execution of a program, simultaneously displaying the message "Break in **" on the screen. This is obtained by holding down the SHIFT key and then pressing BREAK.

The program is not "lost" and can be continued as and when required by typing CONT and keying ENTER. All variables are preserved.

When BREAK is used on its own, program execution is halted whilst the key is held down. Execution continues when the key is released.

When BREAK is used in conjunction with the CTRL key (ie. CTRL-BREAK) the machine returns to the Disc Operating System (DOS).

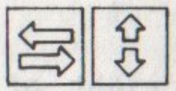
c)  = **ALPHA LOCK**

The ALPHA LOCK key provides a facility which maintains the alphabetic characters in "Shifted mode". This is particularly useful if a large amount of text is to be used in upper case letters.

The facility applies to the alpha characters only and all other keys remain unaffected by alpha lock.

i) The alpha lock is activated by pressing the key once and released when the key is operated a second time. Note that the system is designed such that alpha lock is initially activated each time the machine is powered up, therefore to access lower case letters alpha lock must be released.

ii) A red indicator light, positioned on the front fascia of the computer, illuminates when alpha is locked **ON.**

d)  - **CURSOR CONTROL**

These two keys enable us to move the cursor about the screen quite independently.

i) The first key provides horizontal movement either left or right according to whether in shifted mode or not, as indicated by the arrow heads.

ii) The second key provides vertical movement either up or down according to whether in shifted mode or not, as indicated by the arrow heads.

This provides a facility for positioning the cursor over any individual character, or character space, on the screen.

e) INS DEL **– DELETE/INSERT**

As the title suggests, this key provides a facility for insertion and deletion of characters from a piece of text relative to the cursor position (i.e., the cursor is used to indicate the particular character/position in question).

i) Quite often when "typing in" characters, errors are made and the insert/delete key provides the means of correcting the errors.

ii) The DELETE function is activated in "unshifted mode".

iii) When used in conjunction with the SHIFT key, the INSERT function is activated.

To **DELETE** a character:-

1. Position the cursor to the right of the character (by use of the cursor control keys).

2. Press the DELETE key.

3. The character is deleted and the line of text automatically "closes up" one space **from** the right to fill the gap.

4. Remove the cursor.

EXAMPLE

Original: **AJKLMNP**

We wish to delete 'L'
Cursor positioned:- **AJKL■NP**
Delete activated:- **AJK■NP**
Text 'closes up' from the right
Cursor removed:- **AJKMNP**

When the cursor is positioned to the right of a line of text, " DELETE" acts as a "rub-out" facility, deleting characters and spaces to the left of the cursor. Experiment with a line of text so as to familiarise yourself with this function.

To **INSERT** a character:-

1. Position the cursor where a character is to be inserted.

2. Press the SHIFT and INSERT key simultaneously.

3. The line of text will move one space to the right from the cursor position, leaving a gap ready for insertion of the extra character.

4. Press the required character key.

5. Remove the cursor.

EXAMPLE

Original: **ZKJLMOP**

We wish to insert a 'Y' to the right of 'J' in the text.

Cursor positioned:- **ZKJ▮MOP**
Insert activated:- **ZKJ▮LMOP**
Text moves one space to the right
Key in character:- **ZKJY▮MOP**
When the character is keyed in the cursor moves to the right as in normal operation
Remove cursor:- **ZKJYLMOP**

Experiment with a few examples making use of the INSERT/DELETE facilities.

To **CHANGE** a character:-

To CHANGE an existing character rather than insert or delete, simply position the cursor over the particular character and "type in" the new character.

This OVER-WRITES the existing character, leaving the new character in its place without any alteration to the position of the remaining text, and **without** the operation of any other keys.

NOTE:

These three operations (INSERT, DELETE, CHANGE) fall under the umbrella of a function known as **EDITING.** There are other editing facilities and these will be described later.

f) GRAPH = **GRAPHICS**

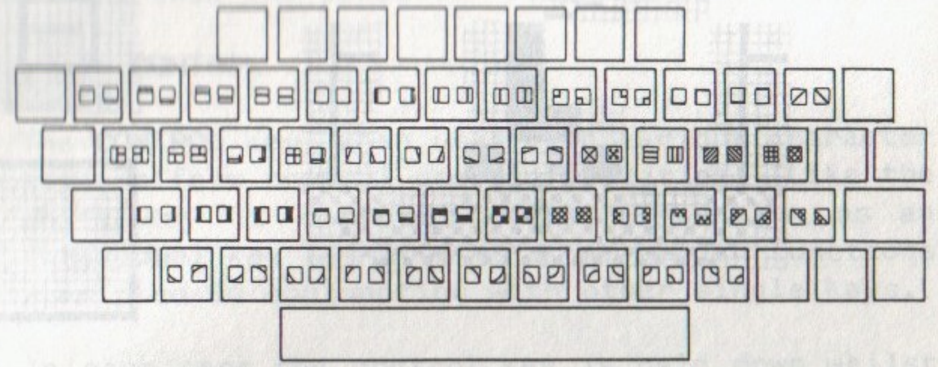


Fig.7.7

The various character keys (numbers, letters, symbols) provide a secondary function incorporating graphic characters. Each key provides two different graphic symbols which are shown on the front face of the key. For the purposes of illustration the symbols are indicated as being on the top surface of the keys as shown in Fig.7.7. 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, but 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.

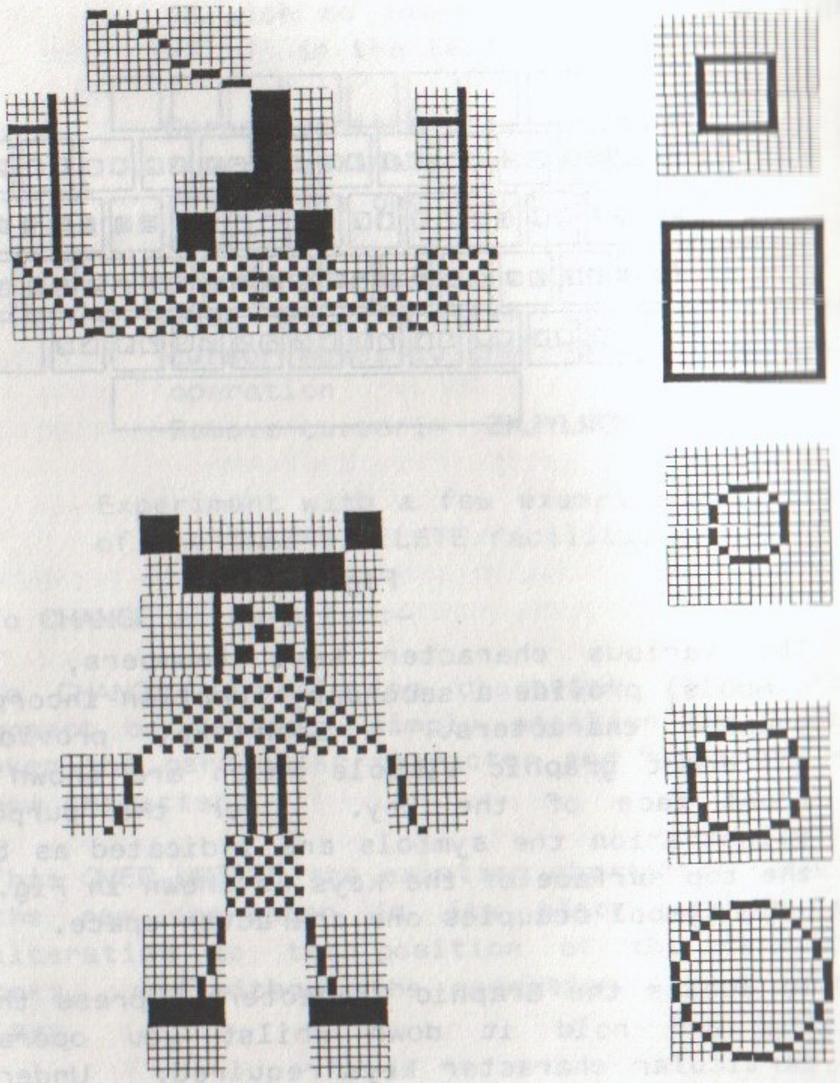


Fig.7.8

Try a few examples of graphic symbols. It is quite enjoyable building up various patterns and shapes using the symbols. The only limitation is your own imagination. A few examples are illustrated in Fig.7.8 to show what is possible. (The grid patterns show the pixels of each character cell.)

ii) CTRL – **CONTROL**

The **CONTROL** key, when used with various character keys, offers several useful facilities. Like the shift key it does not offer a function as an individual key but creates a series of functions when used in conjunction with other single keys.

In each case the control key is held down whilst a second key is then depressed. Some of the functions will REPEAT if the second key is **also** held down.
All the control functions are described in detail in the "REFERENCE MANUAL". It is common practice to abbreviate CONTROL to CTRL for documentation purposes.

f) ESC – **ESCAPE**

The ESCAPE key provides a specialised function which causes listings and tabulations to be aborted.

SCROLL

When we reach the end of the screen whilst "typing in", the display will automatically move up one line at a time to accommodate each new line of characters. This facility is known as SCROLLING and produces a similar effect to that which we see at the end of cinema films.

SUMMARY

a) KEYBOARD
   i) LIGHT GREY keys - CHARACTERS
   ii) DARK GREY keys - FUNCTIONS
   iii) ORANGE key - ENTER

b) LIGHT GREY keys can access TWO characters in TEXT mode (illustrated on upper surface of each key) and TWO characters in GRAPHICS mode (illustrated on front surface of each key).

c) [SHIFT] - when held down, provides access to the "second" character of each key

d) ALPHA characters are LOWER CASE in "unshifted" mode and UPPER CASE in "shifted" mode.

e) NUMERIC characters are "unshifted", alternative SYMBOLS "shifted".

f) [ALPHA LOCK] - maintains alpha characters in UPPER CASE without affecting other keys.

g) SPACE BAR - provides a spacing facility

h) [ENTER] - returns "prompt" and "cursor" to beginning of next line.

i) [⇦⇨] [⇧⇩] move the cursor either horizontally or vertically. Use of the shift key will determine the direction.

j) [INS DEL] - provides a DELETE facility in "unshifted" mode and an INSERT facility in "shifted" mode.

cont.

cont.

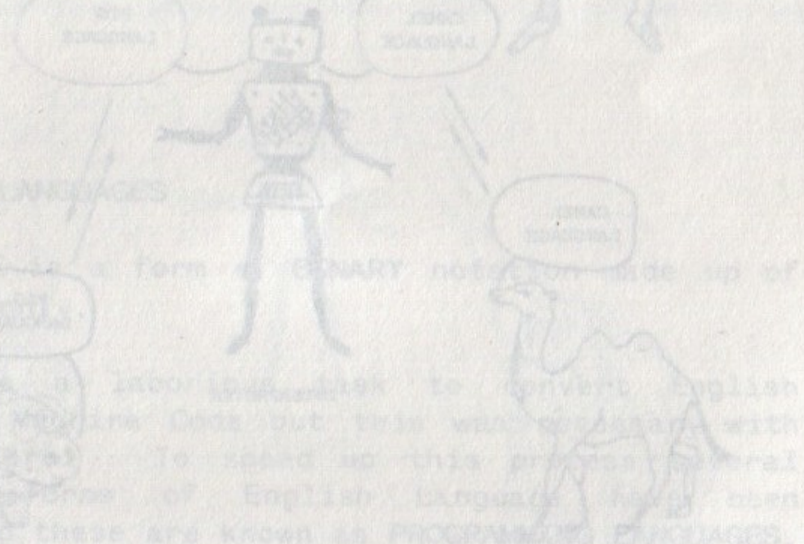
k) [GRAPH] = invokes the GRAPHICS mode
LEFT HAND character-"unshifted"
RIGHT HAND character-"shifted"

l) [BREAK] = causes a break in program execution and displays the message 'BREAK IN LINE ..' Can be used with CTRL and SHIFT keys.

m) [CTRL] - performs special functions in conjunction with other keys (Details in EINSTEIN BASIC REFERENCE MANUAL)

n) CHARACTER keys, CURSOR CONTROL keys, and SPACE BAR exhibit a **REPEAT** facility when held down.

o) The screen is provided with a **SCROLL** facility.



8

TALKING TO THE COMPUTER

COMMUNICATION

Until now we have been making use of the screen display functions available with EINSTEIN. This has provided a few useful exercises to introduce you to the world of the computer. We have **not** asked the computer to carry out any kind of processing for us. To do this we must learn how to communicate with the computer.

Let us imagine that we have two different animals who wish to communicate with each other. They do not have a common language so therefore they must use an interpreter to translate for them as shown in Fig.8.1.

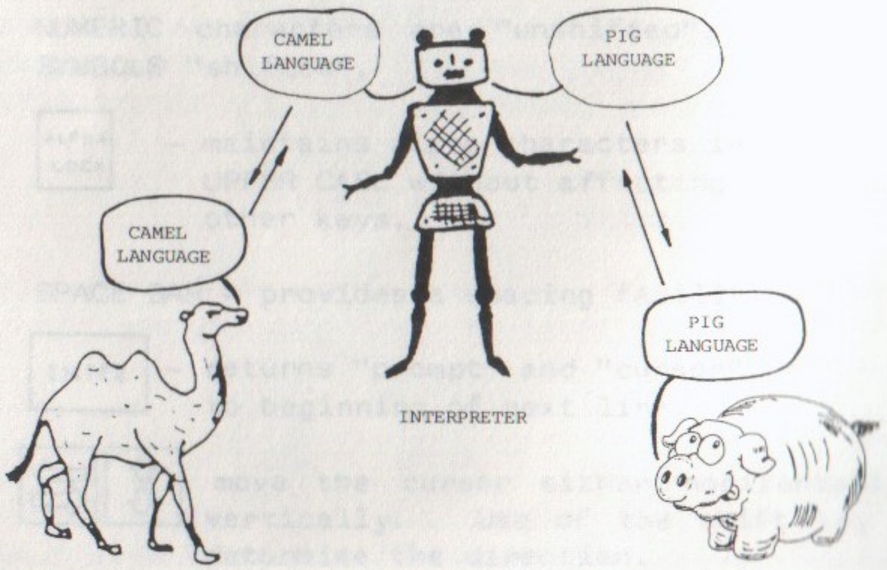


Fig.8.1

The same situation arises when we wish to communicate with the computer. Computers use a language known as **MACHINE CODE**, therefore we need to translate English Language into Machine Code for the computer, and Machine Code into English Language for our benefit. This is illustrated in Fig.8.2.

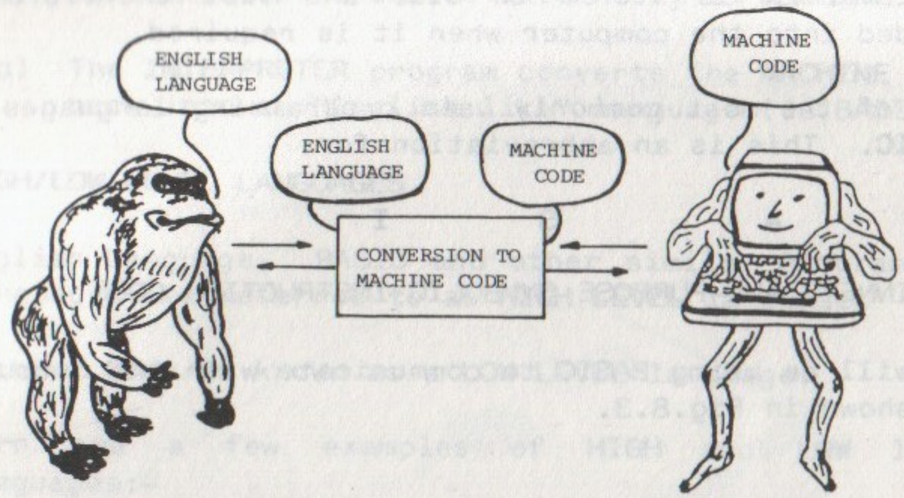


Fig.8.2

PROGRAMMING LANGUAGES

MACHINE CODE is a form of **BINARY** notation made up of 0's and 1's only.

It is quite a laborious task to convert English Language to Machine Code but this was necessary with early computers. To speed up this process several abbreviated forms of English Language have been developed and these are known as **PROGRAMMING LANGUAGES.**

Programming is simply one method of giving the computer instructions and is dealt with in more detail in later sections.

Programming Languages usually take the form of codes or special words to represent lengthy processes and therefore are more easily converted to Machine Code. Today this conversion is normally carried out by a master program which is loaded into the computer. As with our previous examples, this master program acts as an **INTERPRETER.** In the case of EINSTEIN the INTERPRETER is stored on disc and must therefore be loaded into the computer when it is required.

One of the most commonly used programming languages is **BASIC.** This is an abbreviation for:

B A S I C

BEGINNERS **A**LL-PURPOSE **S**YMBOLIC **I**NSTRUCTION **C**ODE

We will be using BASIC to communicate with the computer as shown in Fig.8.3.

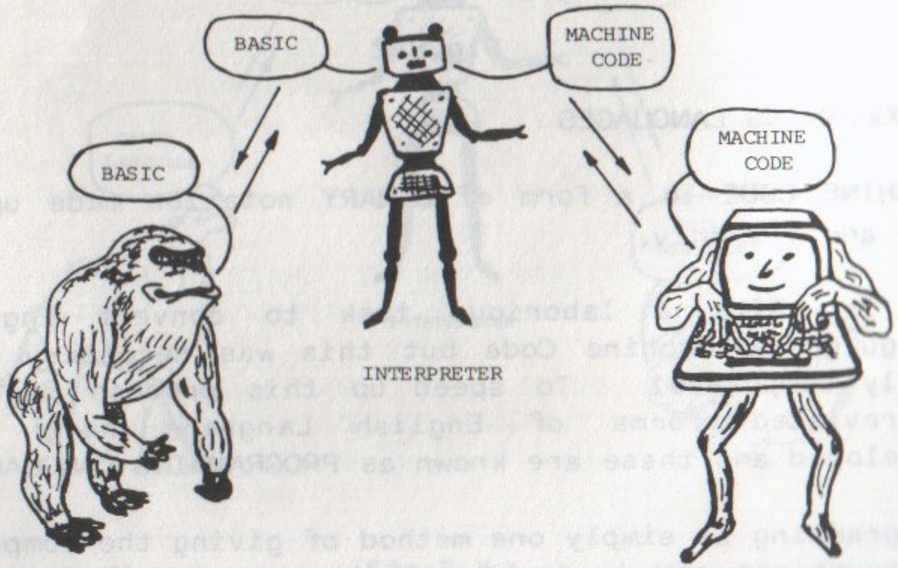


Fig.8.3

Therefore:-

a) We **INPUT** in **BASIC**

b) The **INTERPRETER** program converts **BASIC** to **MACHINE CODE.**

c) The computer functions internally in **MACHINE CODE**

d) The **INTERPRETER** program converts the **MACHINE CODE** back into an "English like" language (ie.BASIC).

HIGH/LOW LEVEL LANGUAGES

English Language, BASIC and other similar Programming Languages are referred to as **HIGH LEVEL** languages

Machine Code is known as a **LOW LEVEL** language.

Here are a few examples of **HIGH** and **LOW** level languages:-

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

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

We are no exception, and whilst adhering to a common core of standard (DARTMOUTH) BASIC, TATUNG/Xtal BASIC 4 includes additions which provide extra facilities for you as a USER of the EINSTEIN.

SUMMARY

a) The **INTERPRETER** program is loaded from disc to facilitate the use of BASIC

b) TATUNG/Xtal BASIC 4 adheres to a common core of standard (DARTMOUTH) BASIC but with additional facilities provided within its own dialect

9

INTRODUCTION TO DISCS

THE DISC CASSETTE

Fig.9.1 illustrates the main characteristics of a disc cassette which the user needs to be familiar with.

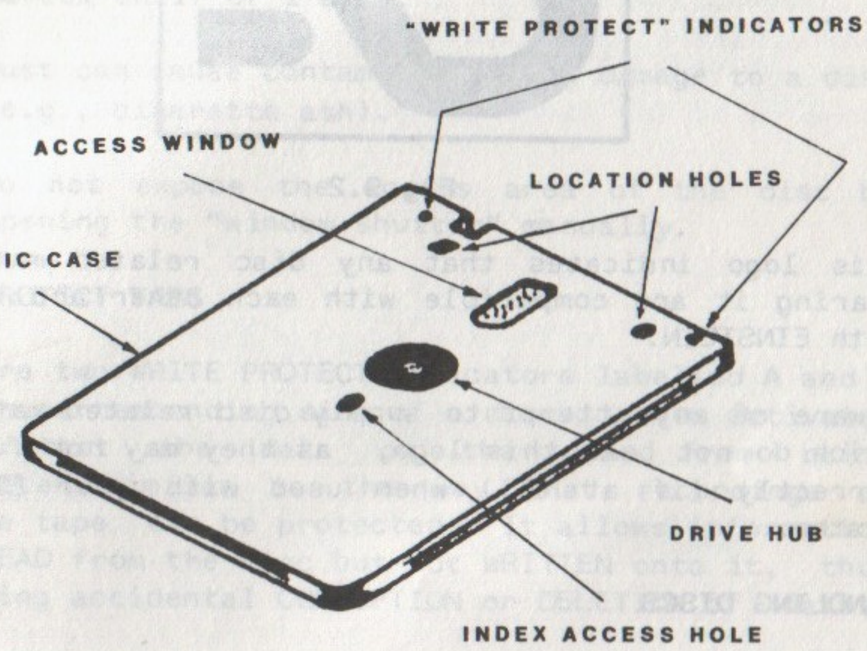


Fig.9.1

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.

NOTE:
When purchasing disc cassettes (sometimes known as compact floppy discs) or extra disc drives, make sure they are marked with the logo illustrated in Fig.9.2.

Fig.9.2

This logo indicates that any disc related materials bearing it are compatible with each other and thereby with EINSTEIN.

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 System.

HANDLING DISCS

Care should be taken when using a disc to avoid any damage which might be caused by unnecessary rough handling. The following precautions should be observed at all times in respect of disc cassettes.

i) Discs not intended for immediate use should be stored in their box for protection.

ii) Keep discs away from magnetic fields and materials which might become magnetised. Strong magnetic fields can distort the data recorded on a disc.

iii) Use identification labels in the correct position and never overlay them.

iv) Never use erasers (rubbers).

v) The disc could be damaged if exposed to HEAT or DIRECT SUNLIGHT, therefore avoid placing near windows or heating appliances (particularly rear parcel shelf of a car).

vi) Dust can cause contamination and damage to a disc (e.g., cigarette ash).

vii) Do **not** expose the access area of the disc by opening the "window shutter" manually.

WRITE PROTECT TABS

There are two WRITE PROTECT indicators labelled **A** and **B** for each corresponding 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.

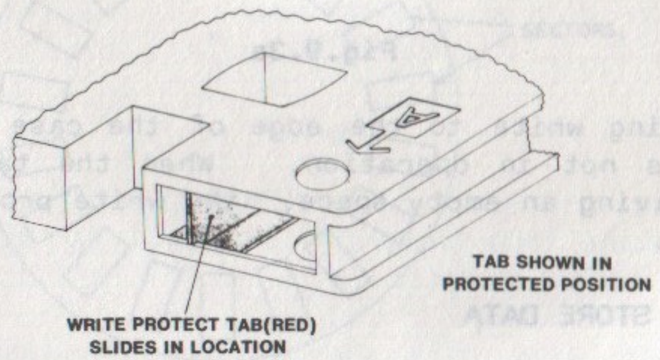


Fig.9.3

The WRITE PROTECT TABS slide in a small housing at each corner of the cassette case as illustrated in Fig.9.3. If the tab obscures the hole, giving a RED indicator in the hole, the write protect is **NOT** in operation. To acivate 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.

An alternative form of write protect tab used by some manufacturers is illustrated in Fig.9.3a. In this instance the tabs are WHITE, and slide backwards and forwards in the direction shown in the illustration.

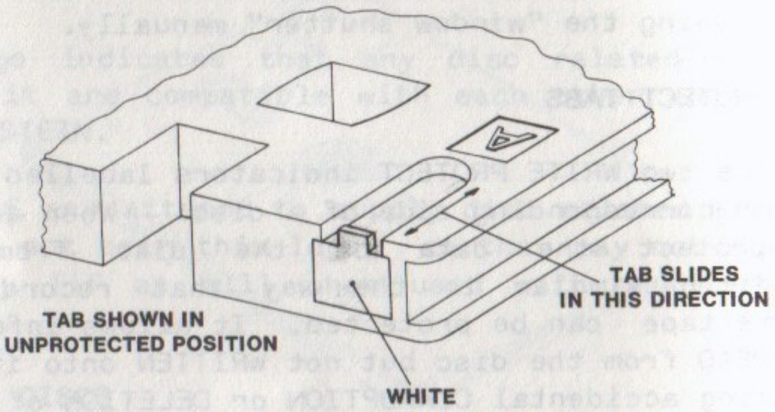


Fig.9.3a

When showing white to the edge of the case the write protect is not in operation. When the tab is slid back, leaving an empty space, the write protect is in operation.

HOW DISCS STORE DATA

Data is stored on a number of concentric tracks on the disc surface similar to music tracks on a record.

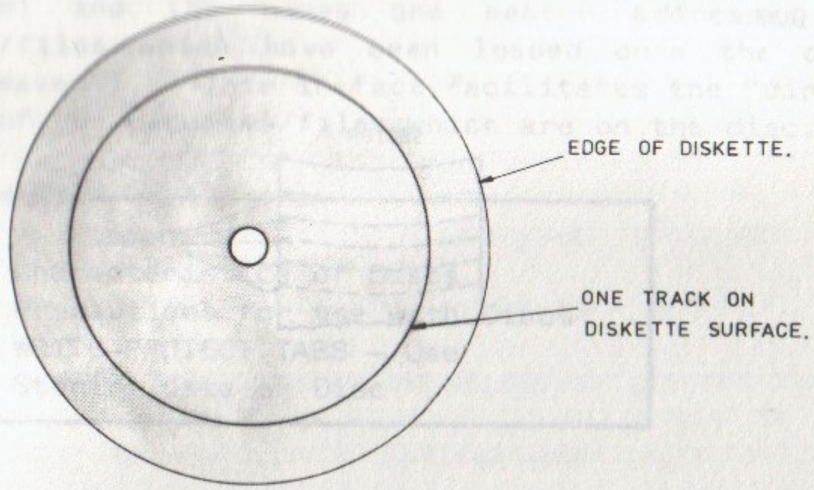


Fig.9.4

The disc surface is divided into **SECTORS,** each SECTOR embracing all the tracks on a particular side as shown in Figs.9.5 and 9.6.

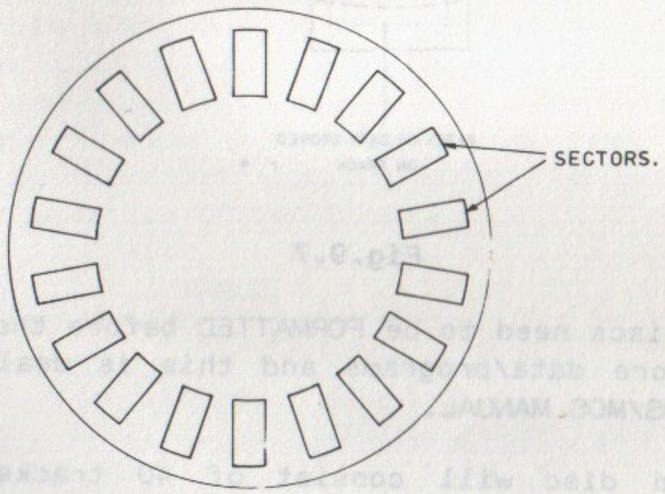


Fig.9.5

Each SECTOR holds exactly the same amount of data and this is measured in **BYTES** per track as shown in Fig.9.7.

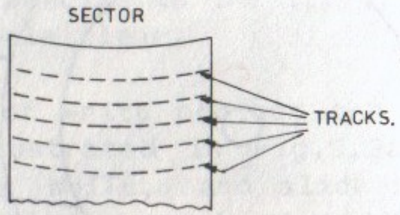


Fig.9.6

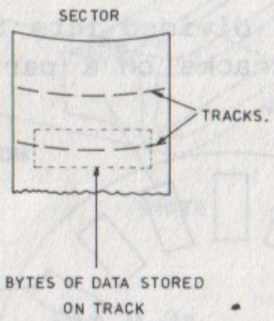


Fig.9.7

All blank discs need to be FORMATTED before they can be used to store data/programs and this is dealt within EINSTEIN DOS/MOS MANUAL.

A formatted disc will consist of 40 tracks and 10 sectors with a storage capacity of 512 bytes per track for each sector. (total storage = 40 x 10 x 512 bytes).

Normally, when **FORMATTED**, each disc has a **DIRECTORY** track. This contains the name assigned to the disc (by the user) and the names and sector addresses of programs/files which have been loaded onto the disc (i.e. "saved"). This in fact facilitates the "direct access" of the programs/files which are on the disc.

SUMMARY

a) Characteristics of Discs
b) Precautions for use with Discs
c) WRITE-PROTECT TABS - Use
d) Storing data on Disc

S E C T I O N B

10

INTRODUCTION TO BASIC

LOADING BASIC

To make the EINSTEIN more flexible, the BASIC Interpreter has **not** been installed within the system. A method has been adopted whereby the user can load the language to be used from disc as described earlier on Page 28. Each time the computer is switched on, the master disc must be inserted in the disc drive and the BASIC INTERPRETER LOADED INTO RAM
(Other languages can be loaded in the same manner).

If the computer has been switched off or re-set since loading the language then it will be necessary to re-load the language (ie. BASIC INTERPRETER) 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** words, or **RESERVED** words. (The EINSTEIN BASIC REFERENCE MANUAL deals with each of the RESERVED words in detail and a list also appears in the appendices.)

The first principle to appreciate is that BASIC operates in TWO quite separate **MODES**, these being known as **DIRECT** mode and **DEFERRED** mode.

BASIC

DIRECT MODE	DEFERRED MODE
Instructions in the form of COMMANDS	Instructions in the form of COMMANDS and STATEMENTS

DIRECT MODE

Direct mode (sometimes known as **COMMAND** 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 the instruction ready for execution. The computer then acts accordingly.

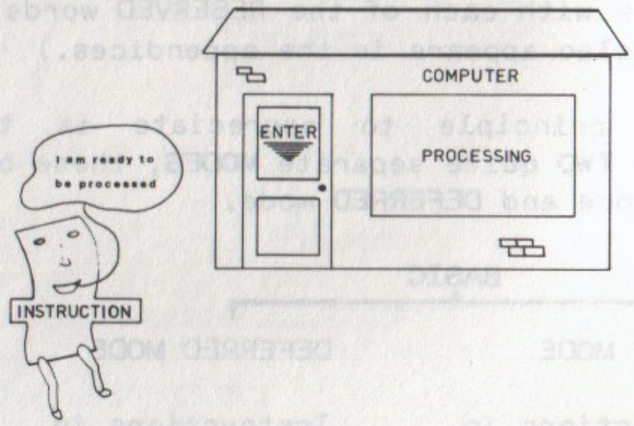


Fig.10.1

IMPORTANT

Care must be taken to ensure all the example exercises are typed in EXACTLY 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 out).

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

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** commanda 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 know as a program.

EXAMPLE -

LINE NUMBERS

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

Type in the above example exactly as shown, using the ENTER key at the end of each line. 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 key to execute the program.

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

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

To summarise:-

a) DIRECT MODE is executed immediatly 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.

More details on PROGRAMS will be given in a later section. For the moment work through the following examples of DIRECT MODE and DEFERRED MODE processes. Some of the results should prove to be quite interesting.

DIRECT MODE EXAMPLES

A. CALCULATIONS - Normal calculation operations can be conducted using the following mathematical and relational operators:

Descending order of precedence

MATHEMATICAL	RELATIONAL
() Parenthesis	
↑ Raise to power	
* 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 the EINSTEIN BASIC REFERENCE MANUAL.

The following examples should be typed **EXACTLY** as printed and the ENTER key used at the end of each one.

Ex. 1 **PRINT 8*7**
Ex. 2 **PRINT 9/3**
Ex. 3 **PRINT 7+4-2**
Ex. 4 **PRINT 33-4*6**

The order of precedence in Ex.4 will be:

i) 4*6 = 24 (multiplication first)
ii) 33-24 (subtraction next)
iii) Answer = **9** (result)

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

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

B. GRAPHICS - Here we illustrate some of the graphics capabilities **other** than the key based graphics characters. Type each line followed by the ENTER key.

Ex. 1 **DRAW 50,50 TO 150,50,**

Ex. 2 **DRAW 150,50 TO 175,100,0**

Ex. 3 **DRAW 175,100 TO 50,50,0**

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 E

Ex. 4 **ELLIPSE 175,100,65**

C. COLOUR - There are 16 colour options available as shown in the table below, each one being allocated a code number from 0 to 15.

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

0 = Transparent	8 - Medium Red
1 = Black	9 - Light Red
2 = Medium Green	10 - Dark Yellow
3 = Light Greeen	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.

Ex. 1 **BCOL 3**
Ex. 2 **BCOL 6**
Ex. 3 **BCOL 10**
Ex. 4 **BCOL 13**
Ex. 5 **BCOL 4**

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

Now Key CLS E to clear the screen again.

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.

Ex.1 TCOL 1,15 displays BLACK character on a white
EINSTEIN cell background

Ex.2 TCOL 11,6 EINSTEIN — displays YELLOW characters on a RED cell background

Ex.3 TCOL 3,13 EINSTEIN — displays GREEN characters on a MAGENTA cell background

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

Key CLS E to clear the screen.

The **GCOL** command offers a similar function to TCOL but in this instance is related to graphics. Type in the following examples keying ENTER after each line. Clear the screen after each example using CLS E.

Ex.1
```
GCOL 1,15
DRAW 25,25 TO 100,100
```
displays a BLACK line on a WHITE background

Ex.2
```
GCOL 11,6
DRAW 25,25 TO 100,100
```
displays a YELLOW line on a RED background

Ex.3 GCOL 15,0 — returns to normal display setting (WHITE on TRANSPARENT background)

D. SOUND - Various options are available for sound and music; details will be found in the EINSTEIN BASIC REFERENCE MANUAL. For the moment type in the example given below followed by the ENTER key and observe what happens.

Ex.
```
BEEP
```

Now key CLS E to clear screen.

DEFERRED MODE EXAMPLES

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

Ex. 1
```
10 PRINT 7-3
20 PRINT 9-6/2
30 PRINT (10+2)-4*2
40 END
RUN
```

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 (in the following text this procedure will be abbreviated to **key NEW E**).

Ex. 2
```
10 CLS
20 GCOL 6
30 DRAW 70,100 TO 170,100,3
40 DRAW 120,35 TO 120,165,3
50 ELLIPSE 120,100,50
60 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 E.

Ex.3
```
10 CLS
20 BCOL 3
30 BEEP
40 DRAW 70,100 TO 170,100,0
50 VOICE 1,0,12,0,1,20
60 A$= "G2AGFE5CG3AGFE5C"
70 MUSIC "V1"+A$
80 BCOL 6
90 BEEP
```

```
100  DRAW 120,50 TO 120,150,0
110  MUSIC "V1"+A$
120  BCOL 10
130  BEEP
140  ELLIPSE 120,100,50
150  MUSIC "V1"+A$
160  BCOL 4
170  END
RUN
```

On completion key NEW E

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. Key CLS E to clear the screen.

2. Key NEW E to clear the previous program from RAM.

3. Type each example program as before.

```
10 REM RANDOM CIRCLES
20 BCOL1:CLS: ORIGIN 0,0
30 FOR I=0 to 30
40 X=RND(256):Y=RND(192):R=RND(60)
50 GCOL RND(13)+2
60 ELLIPSE X,Y,R
70 NEXT
80 BCOL4
90 CLS
```


This program should display circles of random size, colour, and position on a black background. When the program has completed execution key NEW E to clear memory.

```
10 REM RANDOM POLYGONS
20 BCOL1:CLS: ORIGIN 0,0
30 FOR I=0 TO 30
40 X=RND(200)+20:Y=RND(170)+20:R=RND(50)+10
50 N=RND(10)+3:GCOL RND(14)+2
60 POLY N,X,Y,R
70 NEXT
80 BCOL4
90 CLS
```

This program should display polygons of random size, colour and position on a black background. At the end of execution key NEW E.

SUMMARY

a) BASIC - Consists of the following:
   i) COMMANDS
   ii) STATEMENTS
   iii) KEYWORDS/RESERVED WORDS

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

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

11
PROGRAMS

You should now be familiar with some of the capabilities of the EINSTEIN. 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 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.

The analogy of buying a new car and disposing of the old one to make space in the garage can be made.

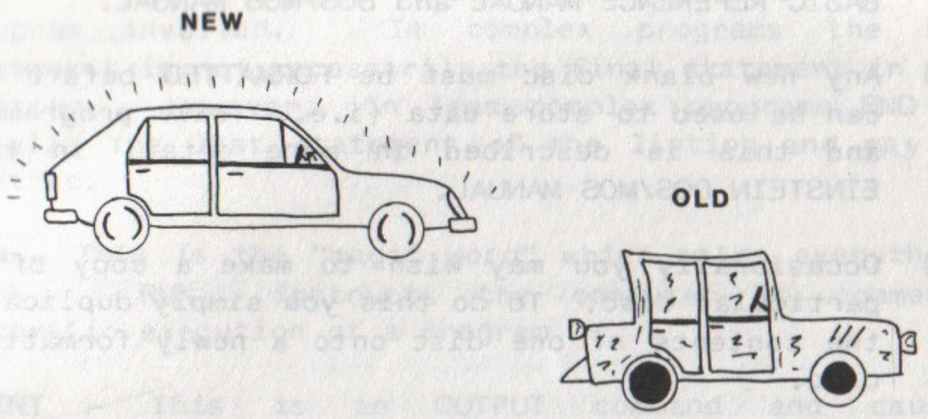
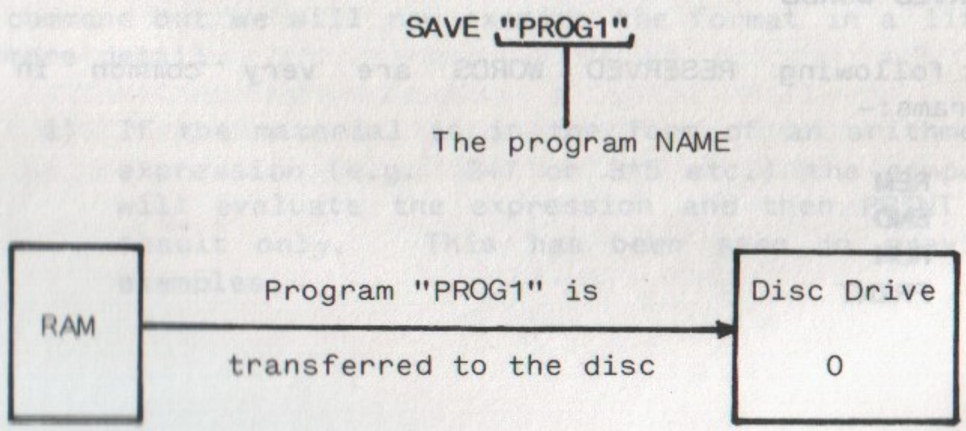


Fig.11.1

SAVING YOUR PROGRAM

To store a program on a more permanent basis it must be transferred from RAM into backing store (i.e., onto disc). This is done by using the **SAVE** command which has the following format:



NOTE:

i) The program name can be up to 8 characters in length, but must **not** be a reserved word, and is devised by the user.

ii) Further details of other facilities available with the SAVE command are given in the EINSTEIN BASIC REFERENCE MANUAL and DOS/MOS MANUAL.

iii) Any new blank disc must be **FORMATTED** before it can be used to store data (i.e., save programs) and this is described in more detail in the EINSTEIN DOS/MOS MANUAL.

iv) Occasionally you may wish to make a copy of a particular disc. To do this you simply duplicate the contents of one disc onto a newly formatted disc.

This facility is quite useful for creating security discs as a safeguard against corruption of "master discs" as was mentioned in an earlier section.

The use of this "backup" facility is described in more detail in the DOS/MOS MANUAL.

RESERVED WORDS

The following RESERVED WORDS are very common in programs:-

i) **REM**
ii) **END**
iii) **RUN**
iv) **PRINT**

REM - This is an abbreviation for REMARK. 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 indicates the "finishing line" of the program involved. In complex programs the END statement is not necessarily the final statement in the listing. However, in less complex programs END is usually the last statement of the listing and 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, but further details involving other output devices are given in the EINSTEIN BASIC REFERENCE MANUAL.

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 is enclosed by double quote marks. Any displayable character within the quotes will be output to the particular device in use at the time, after displaying on the screen.

EXAMPLE: **PRINT "COMPUTERS ARE QUICK"**

This will cause COMPUTERS ARE QUICK to appear on the output device. (screen)

Look at the following program:-

```
10  REM A PROGRAM TO PRINT A NAME AND ADDRESS
20  PRINT " A.N. ASTRONAUT"
30  PRINT "  2001 MOONCAKE WALK"
40  PRINT "   SPACE MOUNTAIN"
50  PRINT "    SOLAR SYSTEM"
60  END
RUN
```

Use NEW E to **clear previous programs** and CLS E to **clear the screen** then modify the above program to accommodate your own personal details and try it on the computer.

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**

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 points are simply pre-set positions on each line of the screen grid and are 10 columns apart as shown in Fig.11.2. (They can be altered by the user - see EINSTEIN BASIC REFERENCE MANUAL)

EXAMPLE:

PRINT "THIS", "JARGON"

Character positions

```
                    01234567890123456
This will appear as THIS      JARGON
                    ↑         ↑
                    Beginning Tab
                    of line   point
```

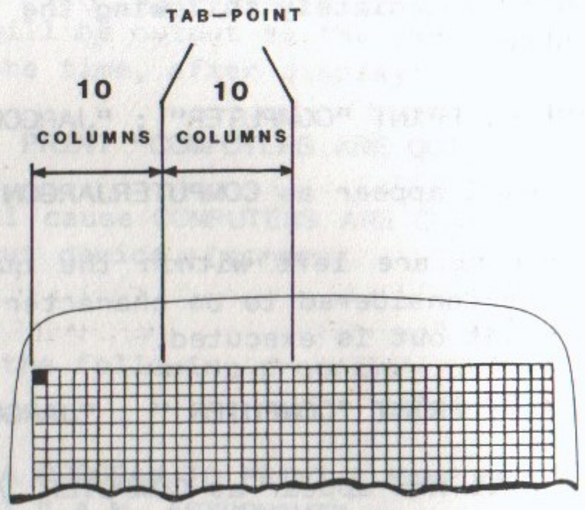


Fig.11.2

EXAMPLE:

```
PRINT "COMPUTER";"JARGON";"IT","IS"
```

This will appear as:-

```
COMPUTERJARGONIT    IS
```

Beginning of line — 1st tab point — 2nd tab point

iv) It can be seen from the examples that by careful manipulation of the two separators (; and ,), combined with spaces as character positions, we have a facility for controlling the layout of a printed expression.

No separator acts as a "carriage return/line feed" (abbreviated to CRLF)

PRINT on its own acts as a CRLF

PRINT can be abbreviated to ?

Look at the following example:

```
10  REM BOOK TITLE PROGRAM
20  PRINT "TITLE", "SHORT STORIES"
30  PRINT "AUTHOR", "A.S. MICHAELS"
40  ? "PUBLISHER", "METCALFE"
50  ? "DATE", 1978
60  END
RUN
```

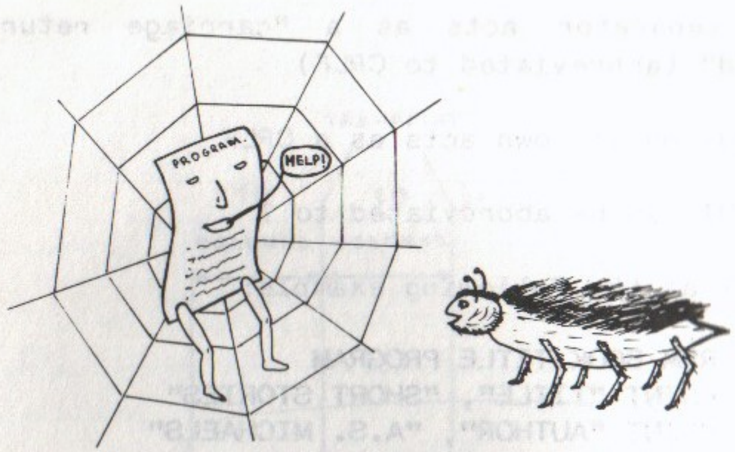
Note down on paper what you anticipate will be displayed, then type in the program and RUN it.

Experiment by altering the PRINT formats in the above program using (,), spaces and (;). RUN the program and observe the results on the screen.

ERRORS/BUGS

Occasionally 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 **DEBUGGING** to indicate tracing and correcting errors .



BUGS PREVENT PROGRAMS FROM RUNNING CORRECTLY

Fig.11.3

ERRORS/SYNTAX

The most common category of errors in program listings are **SYNTAX ERRORS.** 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 manual. 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 the EINSTEIN BASIC REFERENCE MANUAL.

BREAK - To exit a 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 the program in order to examine the listing and discover the cause of the unwanted loop. The listing is then rectified so as to produce the correct processing.

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

EDITING

In order to correct errors in a program listing we have various facilities available to delete and insert individual characters, words or whole lines. These functions are controlled by the BASIC EDITOR.

The BASIC EDITOR contained within the system is quite powerful. It has been specifically designed to make program entry and debugging a pleasure rather than a difficult task. To this end it provides more editing facilities than most other BASIC interpreters.

The use of the DEL/INS key has already been explained in relation to EDITING. More details of the specific functions available can be found in the EINSTEIN BASIC REFERENCE MANUAL.

SYSTEM COMMANDS

System Commands are so called because they affect modification and **overall control** of programs and the system (operating system).

The following are **some** of the system commands which relate to program listings:

a) LIST
b) DEL
c) RENUM
d) AUTO
e) SAVE
f) NEW

LIST - Instructs the computer to display the listing of that particular program in memory at that time.

DEL L1,L2 - Deletes all lines from L1 to L2 inclusive.

RENUM - Can be used to renumber the lines of a program in any increment but defaults to 10 if no increment is specified.

EXAMPLE:

Original listing	Listing after insertion	Listing after renumbering
10.....	10.....	10.....
20.....	15.....	20.....
30.....	20.....	30.....
40.....	25.....	40.....
	30.....	50.....
	40.....	60.....

AUTO - This command can be used to create automatic numbering of lines as they are typed in. This removes the need for the user to type in line numbers.

SAVE - Can be used as a command to place a program in backing store for future reference (.e. place it on disc as explained earlier).

NEW - Clears all existing program listings currently in the memory and is used prior to commencing a new program (as explained earlier).

For in depth details of these functions, refer to the EINSTEIN BASIC REFERENCE MANUAL. However, we suggest that you work through the following examples to give yourself a fundamental understanding of these commands.

EXAMPLES

Work through the following examples, taking care to type in the data as instructed.

Ex. 1. Key **NEW** E to clear any previous programs.

Ex. 2. Type in the following listing:

```
10  REM EXAMPLE PROGRAM
20  PRINT "RESULTS"
30  PRINT "FIRST PLACE--ARCHIBOLD"
40  PRINT "SECOND PLACE--SMITHSON"
50  PRINT"THIRD PLACE--WELLINGTON"
60  PRINT "FOURTH PLACE--WOLSEY"
70  END
```

i) RUN the program
ii) Type in CLS and key E (clears the screen)
iii) Type LIST and key E - program listed again
iv) Continue with next example

Ex. 3. To insert extra lines using the listing on the screen

i) Type the following:

```
15  CLS
25  PRINT
35  PRINT
45  PRINT
55  PRINT
```

ii) Type LIST

The new program listing will have incorporated the new lines (15, 25, 35, 45, 55) and should appear as below:

```
10  REM EXAMPLE PROGRAM
15  CLS
20  PRINT "RESULTS"
25  PRINT
30  PRINT "FIRST PLACE--ARCHIBOLD"
35  PRINT
40  PRINT "SECOND PLACE--SMITHSON"
45  PRINT
50  PRINT "THIRD PLACE--WELLINGTON"
55  PRINT
60  PRINT "FOURTH PLACE--WOLSEY"
70  END
```

iii) RUN this program and examine the layout of the screen display. A line space should appear between each printed result.

This is caused by the "**empty**" PRINT statements in lines 25, 35, 45, 55. In other words the computer has been told to print NOTHING on those lines.

Ex. 4. Clear the screen by keying CLS E

i) Type in RENUM and key E (renumbers the program)

ii) Type in LIST and key E

The new program listing appears with all line numbers in increments of 10, as a result of the RENUM command, as shown below:

```
10  REM EXAMPLE PROGRAM
20  CLS
30  PRINT "RESULTS"
40  PRINT
50  PRINT "FIRST PLACE--ARCHIBOLD"
60  PRINT
70  PRINT "SECOND PLACE--SMITHSON"
80  PRINT
90  PRINT "THIRD PLACE--WELLINGTON"
100 PRINT
110 PRINT "FOURTH PLACE--WOLSEY"
120 END
```

Ex. 5. Key NEW E to clear previous program.

i) Type AUTO and key E

ii) Type in the program listing from above without the line numbers

The line numbers will appear automatically at the beginning of each line as a result of the AUTO command.

iii) Before running the program use the SHIFT-BREAK keys to return to direct mode (i.e. escape from AUTO).

Ex. 6 Key CLS E to clear screen

i) Type LIST and key E
ii) Type DEL 30,60 and key E
iii) RUN the program

You should observe that the display now has **"RESULTS"** and **"FIRST PLACE--ARCHIBOLD"** omitted.

iv) LIST the program

Note that lines 30,40,50,60 are now missing as a result of the DEL command. This command, (in iii above), told the computer to delete all lines from 30 to 60 inclusive from the program listing.

Key NEW E and then clear the screen.

Ex.7. A program must have a name (title) if it is to be saved and recalled later (as shown on Page 73).

i) Type in the following program listing:

```
  5 REM - PROG-1
 10 CLS
 20 PRINT "1 2"
 30 PRINT " 3 4"
 40 PRINT "  5 6"
 50 PRINT "   7 8"
 60 PRINT "    9 10"
 70 PRINT "   7 8"
 80 PRINT "  5 6"
 90 PRINT " 3 4"
100 PRINT "1 2"
110 PRINT
120 PRINT
130 PRINT "THAT WAS CLEVER"
140 END
```


ii) RUN the program

iii) Key CLS E to clear the screen

iv) Remove the "SYSTEM MASTER" disc

v) Check that the write-protect is not operative on the second side of the disc (ie side B), which is blank but has been formatted.

vi) Replace the disc in the drive unit with the blank side uppermost. (ie. side B).

vii) Type SAVE "PROG1" and key E

viii) Key NEW E

ix) Type LIST and key E

There will be no listing because the memory has been cleared by the NEW command.

x) Type RUN and key E - Nothing will happen because the program has been "cleared out".

xi) Type LOAD "PROG1" and key E

This will load PROG-1 back into the memory from the disc.

xii) Type LIST and key E

The program listing now appears on the screen.

xiii) Type RUN and key E

The program will operate again and produce the display on the screen as before.

SUMMARY

a) A PROGRAM is:-
   i) A list of instructions
   ii) Each line is numbered
   iii) 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

g) Introduction to SYSTEM COMMANDS:
   i) LIST
   ii) DEL
   iii) RENUM
   iv) AUTO
   v) SAVE
   vi) NEW

h) Examples using system commands

12
QUANTITIES

Quantities in BASIC fall into two main categories:

a) NUMBERS
b) STRINGS

STRINGS are combinations of characters (letters, numbers, symbols) used for storing names, titles, labels and text.

A string can be any combination of up to 255 characters usually contained within quotes " ", and we can imagine them being held together like beads on a string.

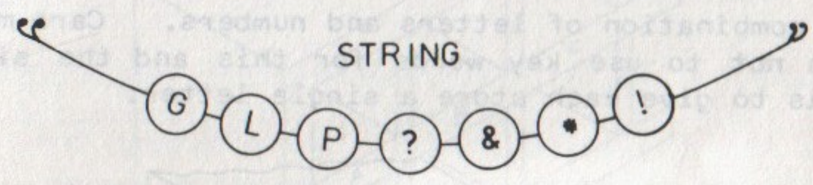


Fig.12.1

NUMBERS

These are either numeric **literals** or numeric **variables.**

i) Numeric LITERALS have been used in the previous examples and exercises. They are simply numbers as given

PRINT 9*6

9 and 6 are numeric literals here

ii) Numeric VARIABLES are represented by combinations of letters and numbers, the first character always being a letter.

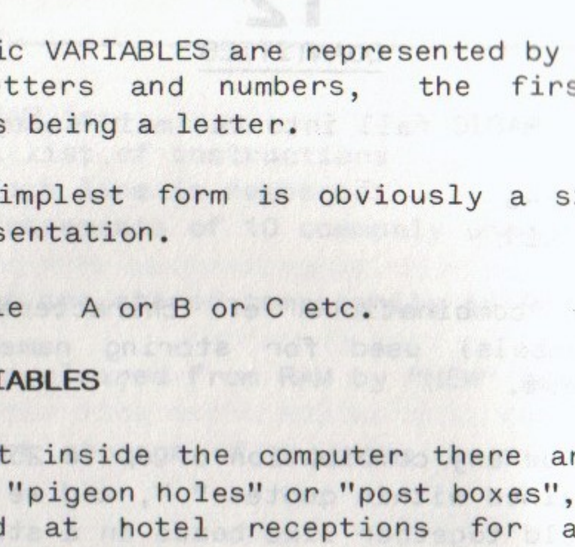
The simplest form is obviously a single letter representation.

Example A or B or C etc.

STORING VARIABLES

Imagine that inside the computer there are stores in the form of "pigeon holes" or "post boxes", similar to those found at hotel receptions for accommodating guests' post and messages.

These stores are labelled either with a single letter or a combination of letters and numbers. Care must be taken **not** to use key words for this and the simplest way is to give each store a single letter.



STORE FOR VARIABLES

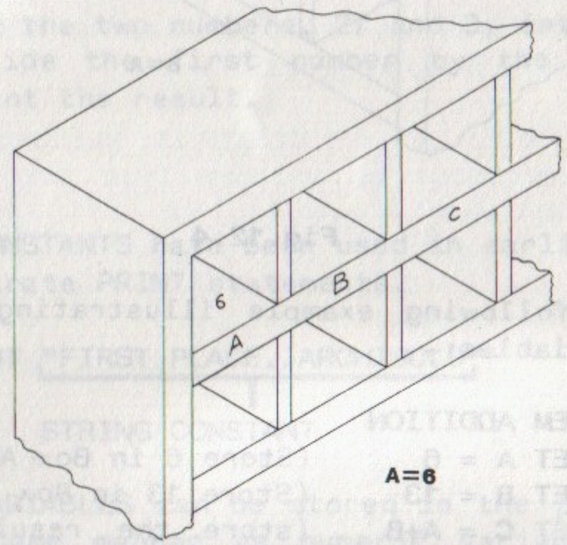
Fig.12.2

ASSIGNING VALUES

Values are **assigned** to variables by the use of the LET statement and **stored** using the = (equals) sign.

Example **LET A = 6**

This instructs the computer to store the value 6 in box A.



A=6

Fig.12.3

Example **LET B = A**

This instructs the computer to copy the contents of box A and store it also in box B (i.e. the same value is stored in both boxes).

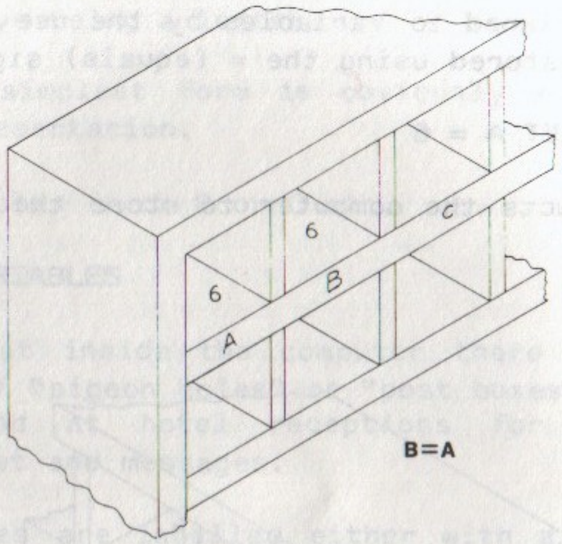


Fig.12.4

Study the following example illustrating the use of numeric variables:-

```
10  REM ADDITION
20  LET A = 6       (Store 6 in Box A)
30  LET B = 13      (Store 13 in Box B)
40  LET C = A+B     (store the result of adding
                     the contents of Box A to
                     contents of Box B in Box C)
50  PRINT A,B
60  PRINT "RESULT IS ";C (output result onto
                          screen)
70  END
```

NOTE: Other values of A and B can be used and this would only necessitate changes in lines 20 and 30, therefore the program becomes quite versatile.

Try the following examples:

Ex.1. Using the format of the previous program, write a similar program to carry out the following operations and RUN it.

a) Put four numbers (5,7,12,20) into four different stores.
b) Add them together, then print the result

Ex.2. Write another similar program to carry out the following and RUN it.

a) Put the two numbers, 27 and 3, into stores.
b) Divide the first number by the second and print the result.

STRINGS

i) STRING CONSTANTS have been used in earlier examples to illustrate PRINT statements.

Ex. **PRINT "FIRST PLACE..ARCHIBOLD"**

STRING CONSTANT

ii) STRING VARIABLES can be stored in the pigeon holes in the same manner as numeric variables. A $ (dollar) sign is used as a suffix to signify that the store contains a **string** of characters.

Ex. **A$ = "J. SOAP"**

A$ — STRING VARIABLE; "J. SOAP" — STRING

A$ is a STRING VARIABLE containing the string "J. SOAP". (A$ reads as 'A' STRING.)

Study the following example illustrating the use of string variables.

```
10  REM NAMES
20  LET A$ = "J. SOAP"      (Store "J. SOAP" in Box A,
                             i.e. A$)
30  LET B$ = "J.  RIDDLE"   (Store "J. RIDDLE" in Box
                             B, i.e. B$)
40  LET C$ = "A.  NIP"      (Store "A. NIP" in Box C,
                             i.e. C$)
50  PRINT A$, B$, C$,       (Print the contents of A$,
                             B$, C$)
```

Clear the previous examples by keying NEW E and then type in and RUN this program.

NOTE: The inclusion of "LET" is optional and under normal circumstances is omitted from the statement in order to save space in memory.

Ex.
```
A = 6
B$ = "J. RIDDLE"
etc.
```

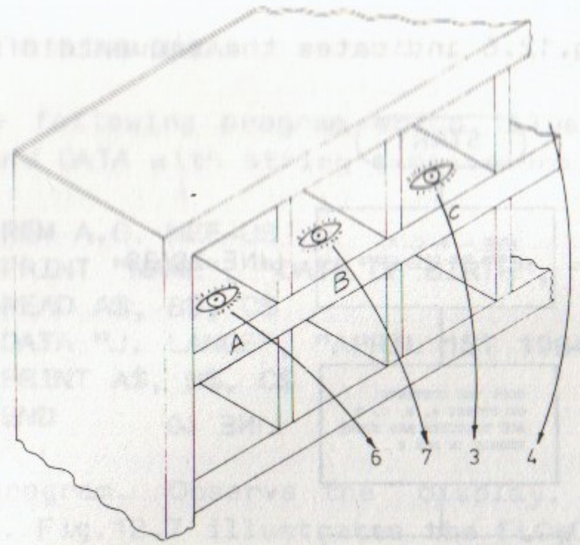
READ/DATA

The READ and DATA statements provide one method of inputting data, either numeric or string, to store.

Effectively the READ statement instructs a series of stores to pair up with a list of elements presented in a DATA statement.

EXAMPLE:

```
READ   A    B    C    D
       ↓    ↓    ↓    ↓
DATA   6    7    3    4
```

READ THE DATA FOR STORE

Fig.12.5

It can be seen from this example that the READ/DATA statement replaces several LET statements, therefore providing a greater efficiency within a program.

READ WITH NUMERIC DATA

The following program illustrates the use of READ and DATA in respect of numeric quantities.

```
10  REM FRED SMITH
20  READ A,B,C,D
30  DATA 6,7,3,4
40  LET E = A+B+C+D
50  PRINT A,B,C,D
60  PRINT "THE SUM IS  "; E
70  END
```

The chart in Fig.12.6 indicates the sequence of events.

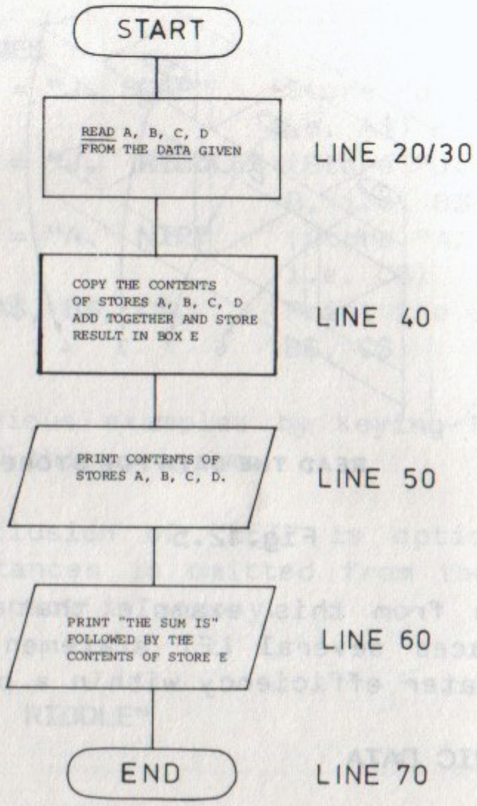


Fig.12.6

Diagrams of this kind illustrating the sequence of a program are known as **FLOW CHARTS.** Rectangles are INSTRUCTION boxes, parallelograms are INPUT/OUTPUT boxes.

NOTE: A READ statement must **always** be paired with a DATA statement. Data statements appear at some point after the corresponding Read statement in a program.

RUN the above program, observing what appears on the screen display. On completion key NEW E.

READ WITH STRING DATA

Study the following program which illustrates the use of READ and DATA with string expressions.

```
10  REM A.G. NEE-US
20  PRINT "NAME", "DATE OF BIRTH", "COMPANY"
30  READ A$, B$, C$
40  DATA "J. LANGE", "APRIL 1ST 1984", "O.W.NERS"
50  PRINT A$, B$, C$
60  END
```

RUN the program. Observe the display. On completion key NEW E. Fig.12.7 illustrates the FLOWCHART sequence.

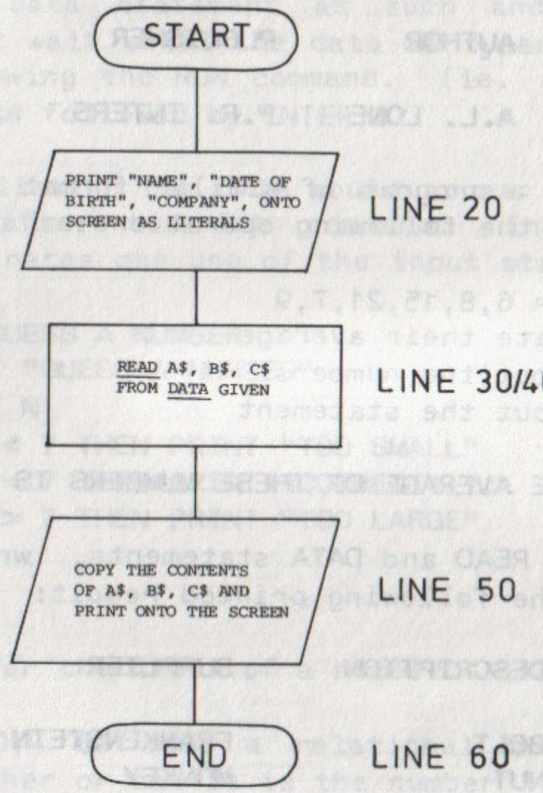


Fig.12.7

EXAMPLES: Work through the following examples incorporating READ/DATA statements.

Ex.1. Using the READ and DATA statements, write a program of similar format to the "FRED SMITH" program to carry out the following sequence of operations:-

i) Read in TWO numbers, for example 7 and 28
ii) Divide the second number by the first
iii) Output the result

Ex.2. Using the READ and DATA statement, write a program of similar format to "A.G. NEE-US" to produce a printed result as shown below:

BOOK	AUTHOR	PUBLISHER
COMPANIONS	A.L. LONE	P.R. INTERS

Ex.3. Write a program of similar format to FRED SMITH to carry out the following operations:

i) READ in 6,8,15,21,7,9
ii) Calculate their average
iii) Print out the numbers
iv) Print out the statement

"THE AVERAGE OF THESE NUMBERS IS "...

Ex.4. Using READ and DATA statements, write a program to produce the following printed result:

QTY	DESCRIPTION	SUPPLIER
50	BOLT	FRANKENSTEIN
29	NUT	MONKEY
30	WASHER	CLEANALL
40	PIN	NEEDLER

INPUT

The INPUT statement provides a facility for the input of data from the keyboard whilst a program is running.

With this facility the user can vary the data input each time the program runs, or input several elements of data as a program progresses.

The format is: **INPUT A,B,C,D,....**

The action is similar to the read statement whereby the A,B,C,D.... will pair up with elements of data.

There is no data statement as such and the input statement must wait until the data is typed in via the keyboard following the RUN command. (ie. INPUT waits for a key press followed by ENTER).

In a program listing the DATA would appear as a series of elements **after** the RUN command. The following program illustrates one use of the input statement.

```
10  REM GUESS A NUMBER
20  PRINT "GUESS A NUMBER"
30  INPUT N
40  IF N < 7 THEN PRINT "TOO SMALL"
50  IF N = 7 THEN PRINT "CORRECT"
60  IF N > 7 THEN PRINT "TOO LARGE"
70  END
RUN
```

Line 30 asks for the input of a number.

Lines 40, 50, 60 make a relational comparison to determine whether or not it is the number 7. They also cause an appropriate message to be displayed. Run the program again to have another try.

The FLOW CHART below illustrates the sequence of operations.

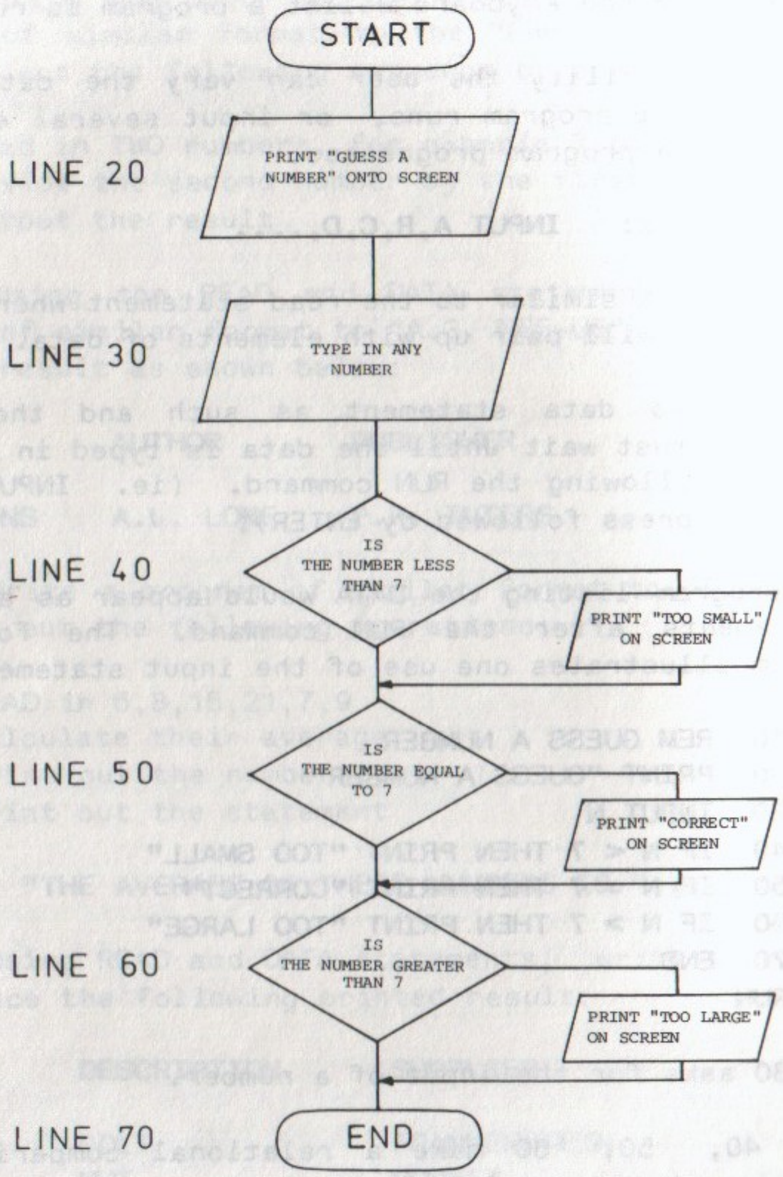


Fig.12.8

SUMMARY

a) Introduction to:
   i) NUMBERS)
   ii) STRINGS) as quantities

b) Principles of LITERALS and VARIABLES in numbers and strings

c) How the information is stored

d) Introduction to the LET statement

e) Introduction to:
   i) READ/DATA
   ii) INPUT

13

PROGRAM STRUCTURE

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

1. SEQUENCE
2. BRANCH
3. LOOP

SEQUENCE

This is the ability to execute instructions one after another in a continuous sequence.

All the example programs used previously exhibit this facility.

BRANCH

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

Branching is controlled by three statements in BASIC:

i) **IF-THEN** statement
ii) **GOTO** statement
iii) **GOSUB** statement

i) The IF-THEN statement is known as a CONDITIONAL branch and has the following format:

IF "a condition exists" **THEN** "execute a given set of instructions".

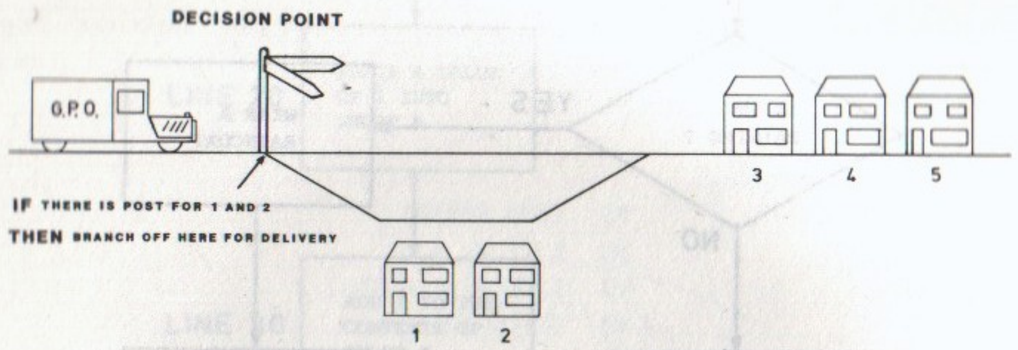


Fig.13.1

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.13.2.

The "diamond" box is the standard method of indicating a decision point in a flowchart, thereby creating a branch (i.e. a question box).

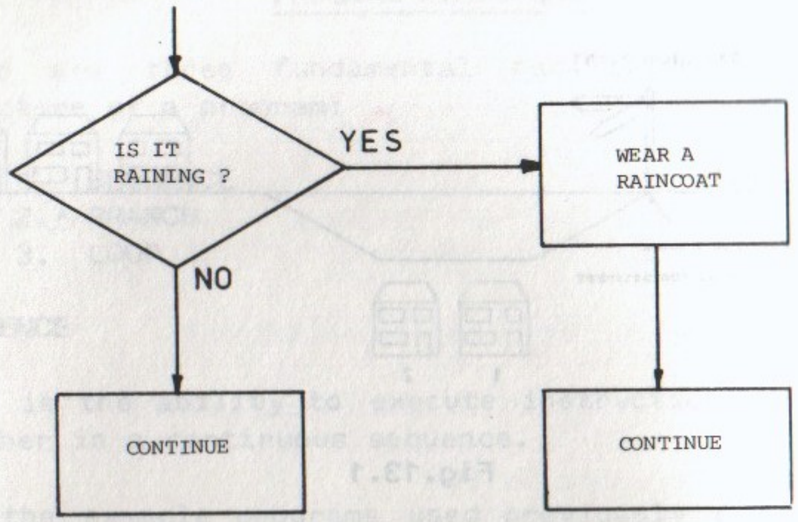


Fig.13.2

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.13.3.)

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.

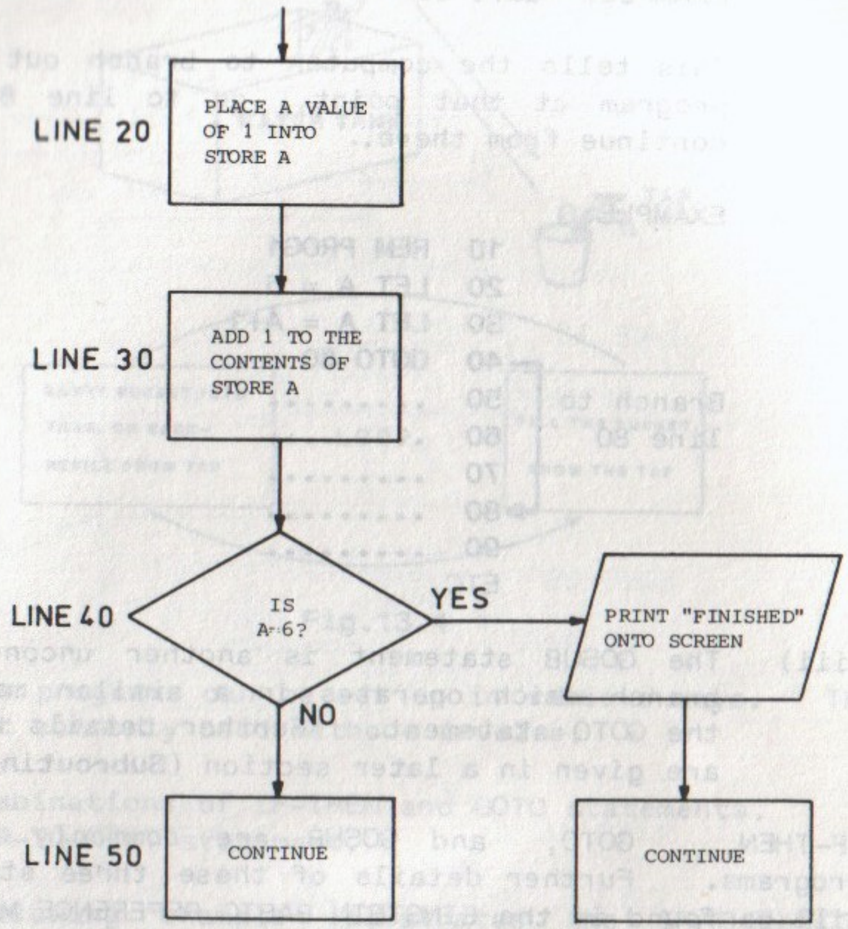


Fig.13.3

ii) The GOTO statement is known as an unconditional branch and has the following format:

GOTO line number

This is a direct command to execute a particular statement without considering any conditions.

EXAMPLE: **GOTO 80**

This tells the computer to branch out of the program at that point, go to line 80, and continue from there.

EXAMPLE:

```
          10  REM PROG1
          20  LET A = 6
          30  LET A = A+1
          40  GOTO 80
Branch to 50  .........
line 80   60  .........
          70  .........
          80  .........
          90  .........
          ETC.
```

iii) The GOSUB statement is another unconditional branch which operates in a similar manner to the GOTO statement. Further details of GOSUB are given in a later section (**Subroutines**).

IF-THEN, GOTO, and GOSUB are commonly used in programs. Further details of these three statements will be found in the EINSTEIN BASIC REFERENCE MANUAL.

LOOP

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

Fig.13.4 illustrates a practical application of the loop principle

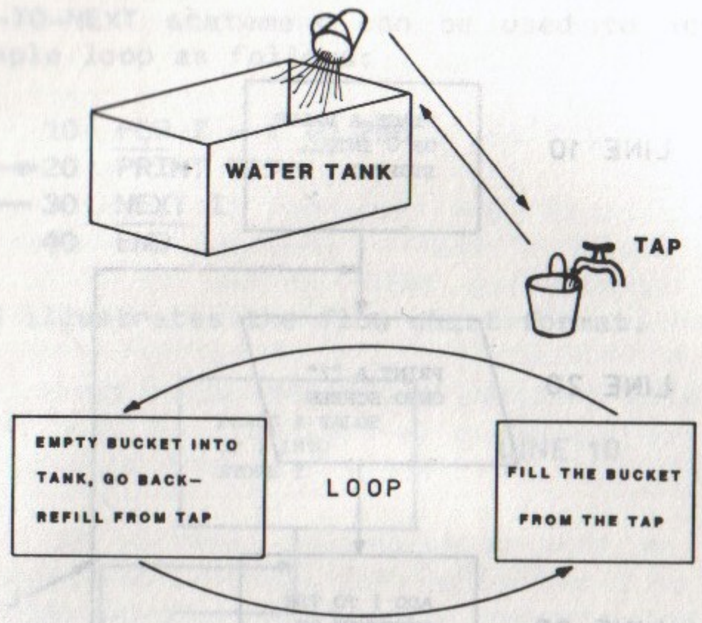


Fig.13.4

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.

The following example 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.

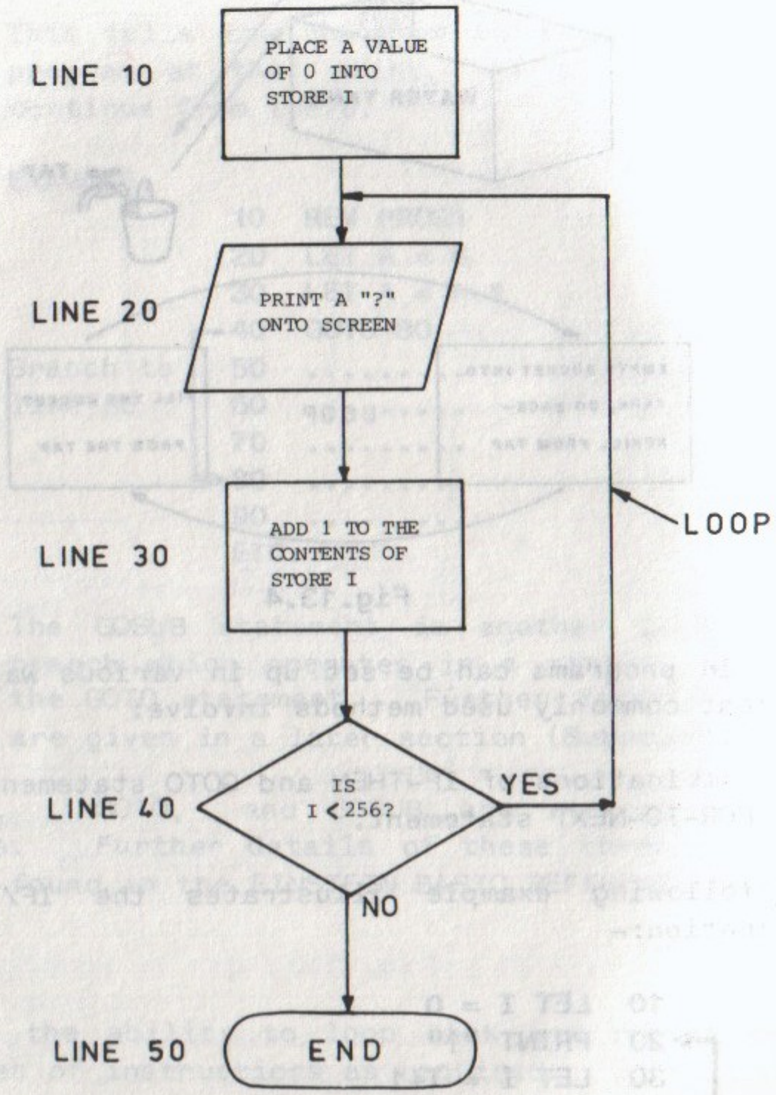


Fig.13.5

The computer will continually loop back to line 20 until the **I** store is NOT less than 256, at which point it will END the program. The display will have shown a total of 256 "?" on the screen.

The FOR-TO-NEXT statement can be used to actuate the same simple loop as follows:

```
10  FOR I = 1 TO 256
20  PRINT "?"
30  NEXT I
40  END
```

Fig.13.6 illustrates the flow chart format.

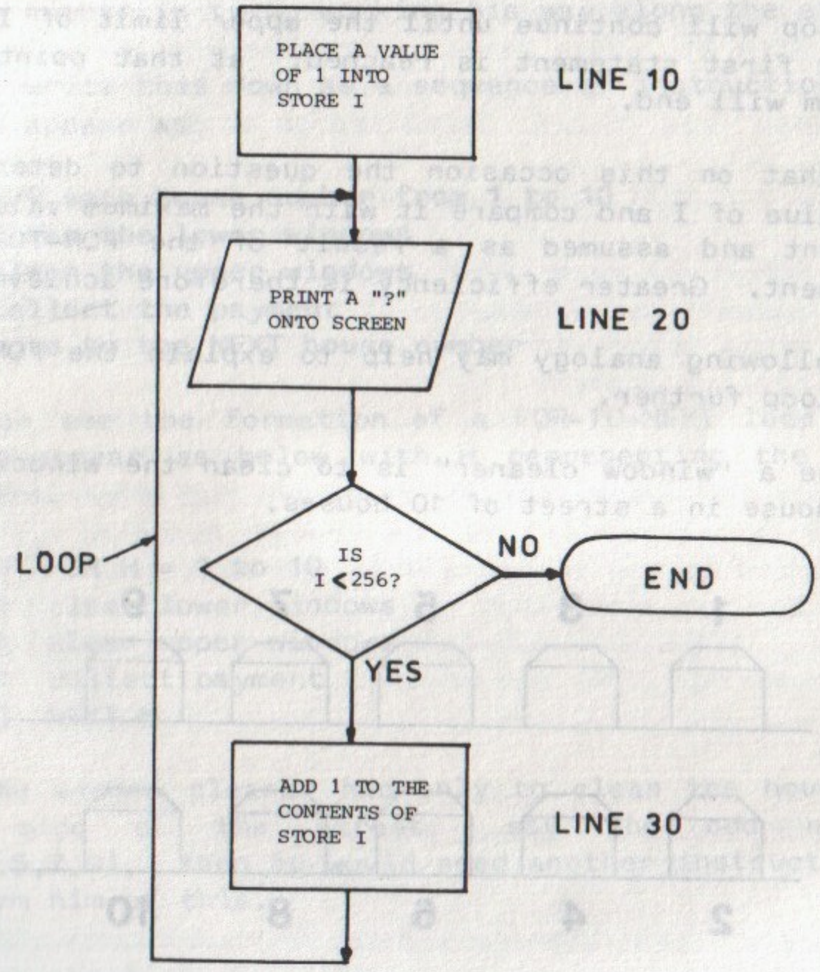


Fig.13.6

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 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.

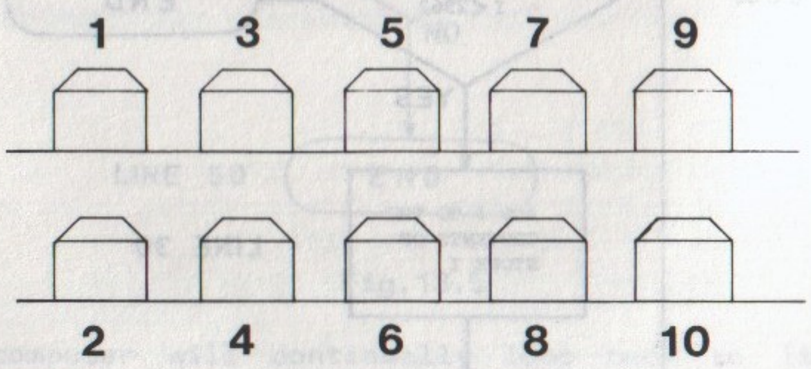


Fig.13.7

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 would appear 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 which would appear as 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 might 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.

Example.

```
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
```

More information and details relating to the FOR-TO-NEXT statement will be found in the EINSTEIN BASIC REFERENCE MANUAL.

EXAMPLES

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

2.
```
10  for I = 1 TO 20
20  PRINT "!";
30  NEXT I
40  END
```

3.
```
10  FOR I = 1 TO 20
20  PRINT "X"
30  NEXT I
40  END
```

SUMMARY

a) SEQUENCE

b) BRANCH - Introduction to:
   i) IF-THEN (conditional)
   ii) GOTO (unconditional)

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

14

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.

GOSUB

The instruction "GOSUB" followed by a line number is used in a program to inform the computer that it should access a subroutine at that point.

GOSUB 90 accesses a subroutine which starts at line 90.

The final line of any subroutine contains the instruction **RETURN.** 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 instruction, the computer returns to line 70.

```
60  GOSUB 120
70  .........
80  .........
```

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.14.1 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 would appear as shown in Fig.14.2.

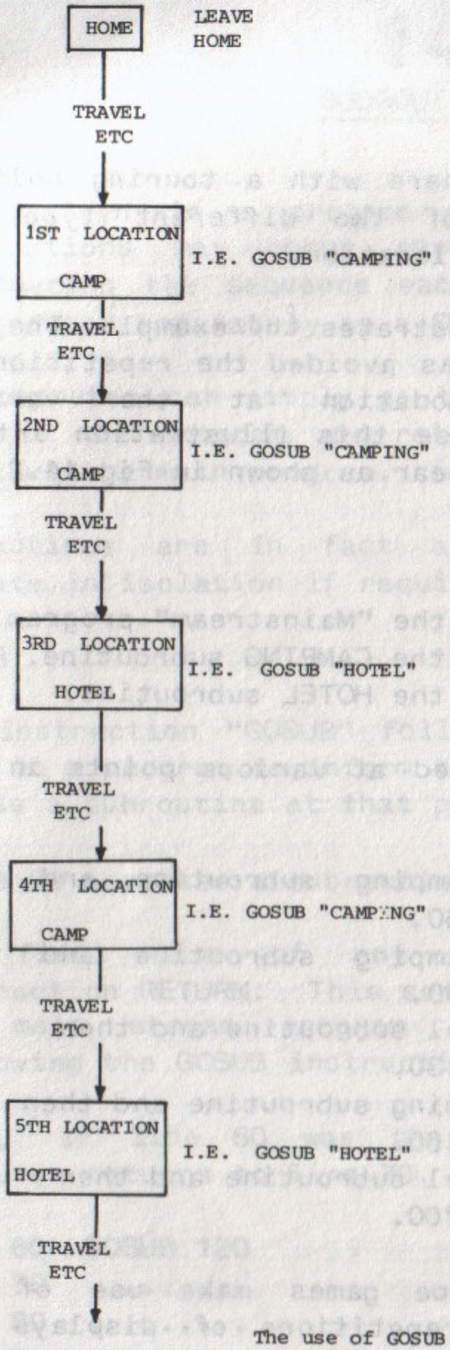
In this example:-

LINES 10 to 210 represent the "Mainstream" 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.

Many of the "arcade" type games make use of the subroutine facility for repetitions of displays and functions.



SUBROUTINE FOR CAMPING

i. Erect the tent.
ii. Cook own meals.
iii. Use sleeping bags
iv. Collapse tent in morning.
v. Continue journey.

SUBROUTINE FOR HOTEL

i. Check into hotel.
ii. Meals in restaurant
iii. Sleep in beds.
iv. Check out of hotel in morning.
v. Continue journey.

1. Travelling - equates with the mainstream program.
2. When camping the appropriate subroutine is used.
3. When in Hotel the appropriate subroutine is used.
4. Continue journey - equates with RETURN to program.

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 would appear as follows:-

Fig.14.1


```
10    Leave Home
20
30    Travel to 1st location.
40
50    GOSUB 220 (Camping)
60
70    Travel to 2nd location.
80    GOSUB 220 (Camping)
90
100   Travel to 3rd location
110
120   GOSUB 280 (Hotel)
130
140   Travel to 4th Location.
150   GOSUB 220 (Camping)
160
170   Travel to 5th location
180
190   GOSUB 280 (Hotel)
200   Travel to Home
210   END (Arrive Home)          END OF MAIN PROGRAM.
-------------------------------------------------------
220   REM  Camp Subroutine
230   Erect tent
240   Cook Meals                 Subroutine for camping
250   Use Sleeping Bags
260   Collapse tent in morning
270   RETURN (continue journey)  INSTRUCTS COMPUTER
280   REM  HOTEL SUBROUTINE      TO RETURN TO MAIN
290   Check into Hotel           PROGRAM.
300   Meals in Restaurant.       Subroutine for Hotel.
310   Sleep in Beds.
320   Check out in morning
330   RETURN (Continue journey).
```

Instructs computer to go to appropriate subroutine.

Fig.14.2

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.

SUMMARY

a) Introduction to:
   i) GOSUB
   ii) RETURN

b) EXAMPLE of use

15
ARRAYS

LISTS

When information needs to be stored as a list, we refer to it as an **ARRAY.**

One advantage of an ARRAY is that it can be referred to by the use of a single variable, thus avoiding the need to allocate individual variables to each piece of data in the list.

EXAMPLE:

A record of the average daily temperature for one week would probably be of a similar format to the following:

	1	2	3	4	5	6	7
DAY –	SUN	MON	TUE	WED	THUR	FRI	SAT
TEMP –	73	77	78	68	72	75	67

ARRAY – 'T'

i) The data is the temperature on each day.
ii) The ARRAY is the whole list of temperatures and is allocated the single variable 'T'.
iii) The **individual** temperature values in the ARRAY are known as ELEMENTS.

If the days were referenced by number (1 to 7) rather than name, then the individual elements of the ARRAY can be identified as follows:

T_1 T_2 T_3 T_4 T_5 T_6 T_7

Thus =

T_1 is the temperature for Sunday, i.e. day 1

T_2 is the temperature for Monday, i.e. day 2

T_3 is the temperature for Tuesday, i.e. day 3

T_4 is the temperature for Wednesday, i.e. day 4

T_5 is the temperature for Thursday, i.e. day 5

T_6 is the temperature for Friday, i.e. day 6

T_7 is the temperature for Saturday, i.e. day 7

It can be seen that each element of an array can be identified by a single VARIABLE followed by a NUMBER. This number is known as a **SUBSCRIPT** and refers to the position of each element within the ordered progression of any array. Within TATUNG/Xtal BASIC 4 subscripts begin from 0. Hence an array A containing 3 elements would be subscripted A0,A1,A2.

Normally the SUBSCRIPT is enclosed in brackets immediately following the VARIABLE.

EXAMPLE:

T(2) refers to the third element of the array **'T'.**

If the elements are STRING expressions then the ARRAY would be allocated a string variable.

EXAMPLE:

Let **Y$** be the variable representing the array whose elements are the months of the year.

Thus:

Y$(0) is "January"
Y$(1) is "February"
Y$(2) is "March"
Y$(3) is "April"
etc.

DIMENSION

When an ARRAY is used the computer needs to know how many items are in the list so as to allocate space in memory accordingly.

This information is provided by use of the DIMENSION statement as illustrated below.

DIM A(15)

This informs the computer that the array 'A' has 16 elements (0 to 15).

The dimension (DIM) statement is always included in the program listing prior to the use of an ARRAY.

EXAMPLE:

A(0)	2.2
A(1)	6.8
A(2)	7.1
A(3)	9.3
A(4)	4.2
A(5)	3.1
A(6)	7.6
A(7)	12.3
A(8)	11.2
A(9)	8.5
A(10)	5.2
A(11)	4.7

This ARRAY would be given a DIMENSION of 11, i.e. DIM A(11)

READING A LIST

To read this list and print results within a program structure, the following format would be used:

```
10  REM TO PRINT RESULTS FROM LIST
20  DIM A(11)
30  FOR I = 0 TO 11
40  READ A(I)
50  PRINT A(I)
60  NEXT I
70  DATA 2.2, 6.8, 7.1, 9.3, 4.2, 3.1, 7.6,
    12.3, 11.2, 8.5, 5.2, 4.7
```

Note the use of the FOR-TO-NEXT statement to set up a loop to read each element of the array in turn.

LINE 20 dimensions the array.

LINE 40 reads each element of the array in turn.

LINE 50 prints the value of each element of the array in turn onto the screen.

LINE 70 lists the elements of the array.

Try this example in the computer - first key NEW E to clear previous programs.

TWO-DIMENSIONS

The lists examined so far have only required a single dimension and are therefore known as ONE-DIMENSIONAL ARRAYS.

However, if we have a TABLE of results, as opposed to a single list, then a second dimension is used and it becomes known as a TWO-DIMENSIONAL ARRAY.

The FIRST dimension indicates the number of horizontal rows, and the second the number of vertical columns making up the table.

EXAMPLE:

	0	1	2	3	4	5	6
0	7				9		
1		5					
2				3		6	
3		4					
4					8		2
5			9				

This ARRAY would be given a dimension of (5,6) i.e. DIM A(5,6).

A single variable is still used to refer to the complete array, but it will be followed by TWO subscripts which together indicate an individual element of the array.

A(0,0) Refers to the element in the 1st row of the first column, i.e. 7

A(2,5) Refers to the element in the 3rd row of the 6th column, i.e. 6

A(5,2) Refers to the element in the 6th row of the 3rd column, i.e. 9

It can be seen that the two subscripts are used in a similar manner to co-ordinates for graphs and maps.

Fig.15.1 illustrates how each box of a table is referred to by use of the subscripts.

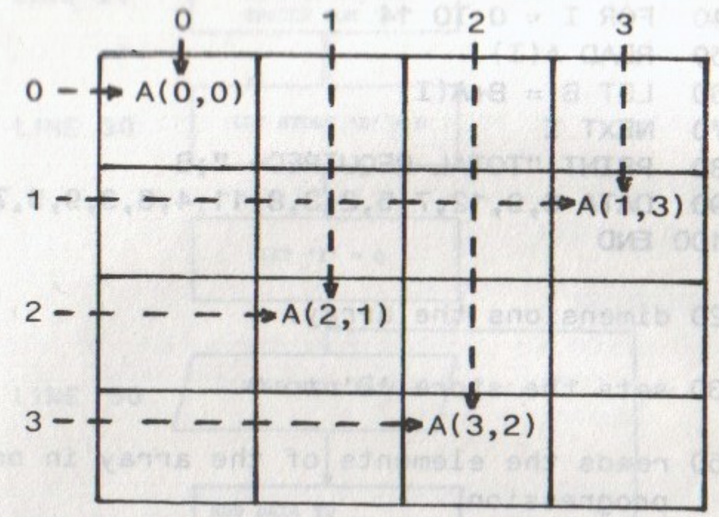


Fig.15.1

Further information concerning ARRAYS will be found in the EINSTEIN BASIC REFERENCE MANUAL.

EXAMPLE

The program listed below illustrates a method of obtaining a total value from an array. The elements of the array could represent quantities of materials, size of objects, physical measurements, scores or test results, price of articles, etc.

The program could be adapted to suit a variety of criteria according to the requirements of the user.

Run the program in the computer and observe the result on the screen.

```
10  REM TO READ AN ARRAY AND PRINT TOTAL
20  DIM A(14)
30  LET B=0
40  FOR I = 0 TO 14
50  READ A(I)
60  LET B = B+A(I)
70  NEXT I
80  PRINT "TOTAL REQUIRED= ";B
90  DATA 6,9,12,7,5,2,3,8,11,4,6,3,9,5,7
100 END
```

LINE 20 dimensions the array.

LINE 30 sets the store 'B' to 0.

LINE 50 reads the elements of the array in ordered progression.

LINE 60 adds each element in turn to the contents of store 'B'.

LINE 80 prints out "TOTAL REQUIRED =" followed by the sum of the elements (i.e. final contents of store 'B').

LINE 90 lists the elements of the array.

Adapt the program to a particular use of your own, making the necessary changes in data, dimension statement, and loop control (FOR-TO-NEXT statement). The flow chart is illustrated in Fig.15.2 to assist you with this operation.

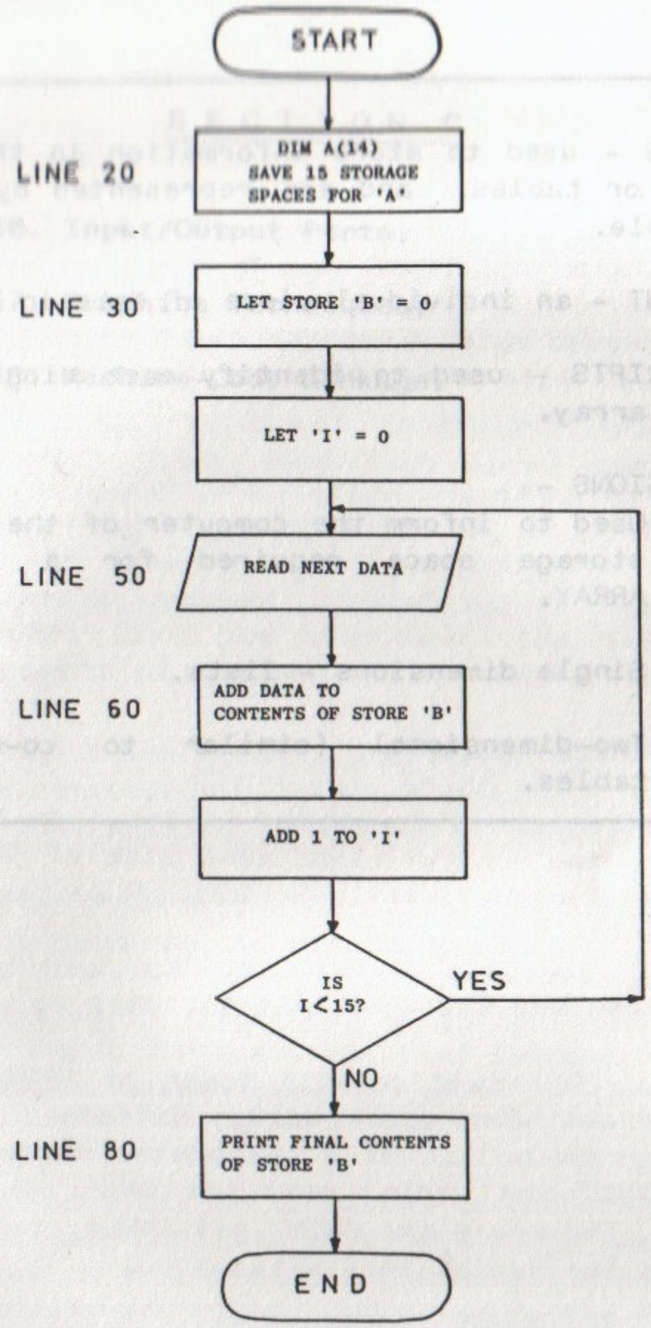


Fig.15.2

SUMMARY

a) ARRAYS - used to store information in the form of lists or tables, and are represented by a single variable.

b) ELEMENT - an individual piece of data in an array.

c) SUBSCRIPTS - used to identify each single element of an array.

d) DIMENSIONS -

  i) Used to inform the computer of the necessary storage space required for a particular ARRAY.

  ii) Single dimensions - lists.

  iii) Two-dimensional (similar to co-ordinates) tables.

S E C T I O N C

16. Input/Output Ports.

17. Graphics Techniques.

18. Machine-Code Linkage.

16

INPUT OUTPUT PORTS

INTRODUCTION

This section outlines the INPUT/OUTPUT facilities available with the EINSTEIN.

It is intended mainly for the experienced user who may be familiar with the application of these facilities using computers.

The newcomer may well read through this section to become aware of the kinds of facilities available but would probably only be concerned with the application of Joysticks at the moment. However, after a period of familiarisation, newcomers may well return to this section to further their knowledge of these facilities.

SERIAL I/O PORT(RS232)

A serial I/O port is provided in the form of an 8 pin socket located to the right hand side of the computer (refer to illustrations in SETTING UP section).

This port provides a facility for use with serial printers, modems and other similar devices.

To access this port special forms of PRINT and INPUT statement are used which assign different I/O devices to the system by use of a designated number. Within Tatung BASIC the serial port has been designated the number **2**. Therefore the **PRINT** and **INPUT** statements used to access the serial port will be:-

```
10 PRINT # 2
20 INPUT # 2
```


These statements will direct all output to the device numbered 2 and receive all input from the device numbered 2, i.e. the serial port(RS232).

Further explanation of these statements will be found in the EINSTEIN BASIC REFERENCE MANUAL.

THE TATUNG PIPE

The **PIPE** provides a direct connection and software interface which gives very high speed communication with the processor. It takes the form of a ribbon connector at the rear of the computer (refer to illustrations in SETTING UP section) which provides a facility for considerable expansion in the future.

ANALOGUE INPUT

There are two 6 pin sockets located to the right hand side of the computer and labelled ANALOGUE 1 ANALOGUE 2 (refer to illustrations in SETTING UP section).

These sockets provide the facility for analogue input which can be used for two main purposes:-

i) To plug in joysticks for games
ii) An input for measurement of voltages from scientific devices.

a) **Joysticks**
Two joysticks can be connected to the computer, one to each socket. Each joystick can be used to move an object on the screen, for example, either up and down or left and right. This means the object can be moved to any position within the screen area.

b) **Voltages**
Each input can be used for measuring voltages. Since it is possible to use a "transducer" to provide a voltage proportional to temperature, light intensity, water pressure, gas concentration, smoke density etc., the computer can be used to monitor these parameters using appropriate hardware devices.

8 BIT USER INPUT/OUTPUT PORT (Digital input/output)

The computer contains an "8 bit user port" in the form of a ribbon connector at the rear of the computer. This can be used for a wide range of devices and other general interfacing boxes which are then accessed as follows.

a) **OUTPUT**
For output to the port the following command is used:-

OUT J1,J2
where J1 is the "address" of the port and J2 is a value from 0-255 decimal (00 to FF hexadecimal)

The "address" of the port is &32 (i.e. HEX 32) so if a value of 5 is to be output to the port the command would appear as follows:-

OUT &32,5

b) **INPUT**
For input from the port the following command is used:-

INP(J)
where J is the "address" of the port

The "address" of the port is &32 (as above) therefore the command would appear as follows:-

A%=INP(&32)

This would return the value currently at the port as a number in the range 0-255 decimal (00 to FF hexadecimal).

EXPLANATION OF USER PORT

As suggested by the title, the user port has 8 pins available for use. Each pin can be connected directly to a hardware device using a ribbon of 8 parallel wires as illustrated in Fig.16.1.

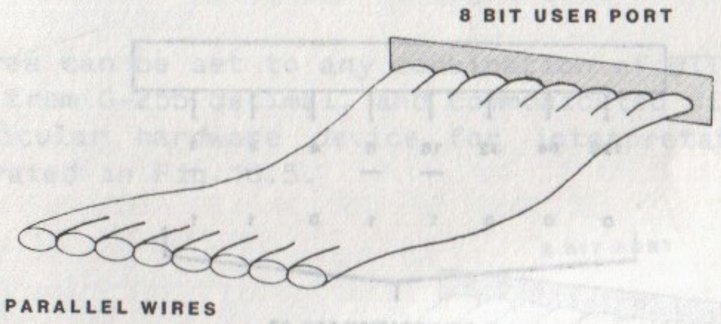


Fig.16.1

The pins are allocated values as shown in Fig.16.2.

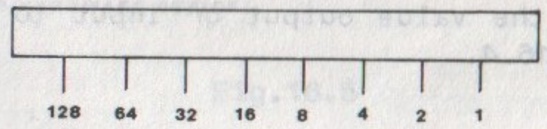


Fig.16.2

When a value is output to the port it is converted to an 8 digit (BIT) binary number using the values on the pins.

e.g. a value of 27 is made up of:-
one 16
one 8
one 2
one 1

When related to the pins the configuration shown in Fig.16.3 would appear.

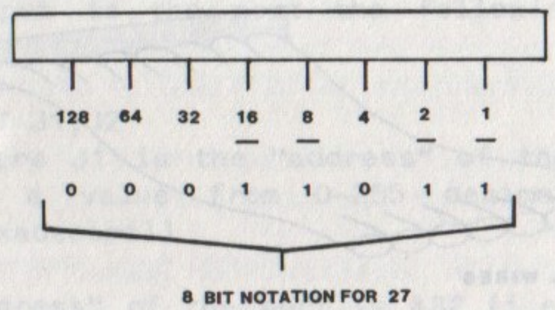


Fig.16.3

Therefore each of the 8 wires is set either to 1 or 0 according to the value output or input to the port as shown in Fig.16.4.

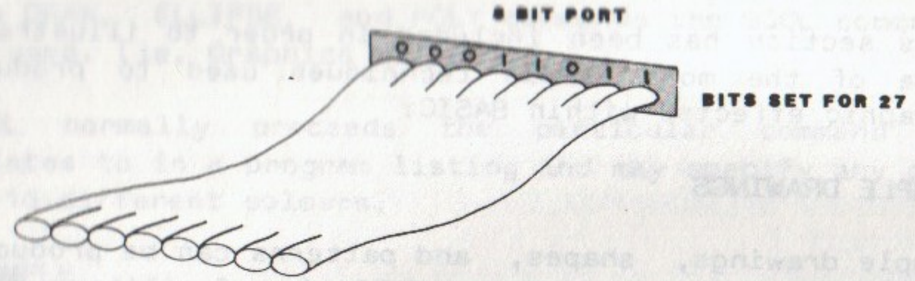


Fig.16.4

Each of the 8 binary digits (BITS) acts like a switch setting each wire either on or off. If we imagined the wires connected to 8 lights, then "1" would switch the light on and "0" would turn a light off.

The wires can be set to any combination of BITS for all values from 0-255 decimal, and communicated directly to a particular hardware device for interpretation, as illustrated in Fig.16.5.

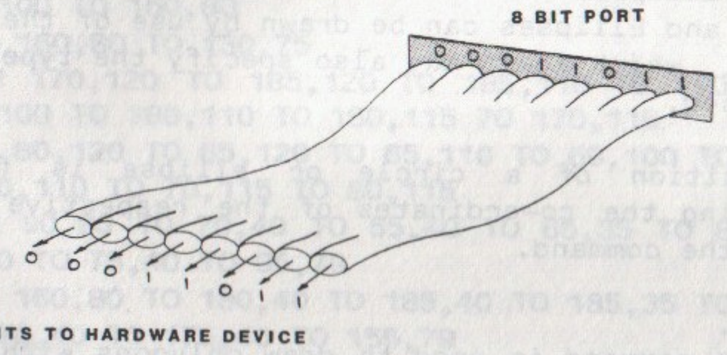


Fig.16.5

The technical specifications for all the I/O ports will be found in appendix E and F.

17

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 is used to 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.

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 example 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 then observe 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
```

Programming drawings such as the one produced in the above example is made simple if a sheet of "grid paper" is used to map out the lines and points etc. The paper represents the imaginary "Pixel grid" of the screen, which is numbered 0 to 255 horizontally from left to right, and 0 to 191 vertically from bottom to top, as shown in Fig. 17.1.

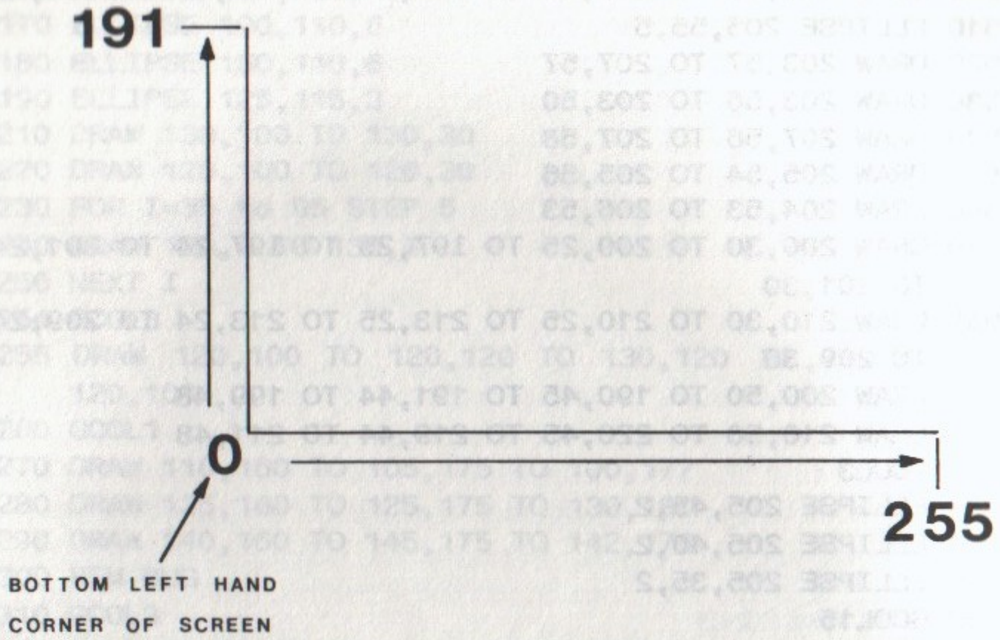


Fig.17.1

A sheet of A4 size graph paper with 10mm squares sub-divided into 1mm squares would be suitable for this purpose and is normally available from a stationer.

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 then it is necessary to draw it in several positions between the points, "rubbing out" the last drawn character immediately prior 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

Imagine that 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.

The following example executes this sequence of operations.

```
 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 key NEW E first 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.

To give the movement a continuous effect simply create a loop by sending control back to the line of listing which represents the beginning of the sequence. This is done by replacing line 240 with the following new line.

```
240 GOTO 30
```

Add this new line to the listing you have already typed into the computer and then once again observe the result when you RUN the program.

It will be necessary to use 'SHIFT BREAK' to break out of the loop once it is set in motion. Key CLS E 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 observe the result when the program is RUN.

```
 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 key CLS E 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. Remember to key NEW E before typing in the listing.

```
 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.

Having observed the result key CLS E to clear the screen followed by NEW E to clear the 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 positioned 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".

Type in the above example and observe the result when you press ENTER. Clear the screen using CLS E .

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
```

Type in the above listing and observe the result when the program is RUN.

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. (Key NEW E before typing this listing).

```
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 ascertain the sequence cycle 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. (Key NEW E before typing the listing).

```
10 CLS
20 FOR I=10 TO 80
30 PRINT@I,12,"*"
40 PRINT@I,12," "
50 NEXT I
60 GOTO20
```

Type in the listing and observe the result when the program is run. Use 'SHIFT BREAK' to break out of the loop when required then key NEW E to clear the memory.

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.17.2 illustrates the formation of the letter E in the character set.

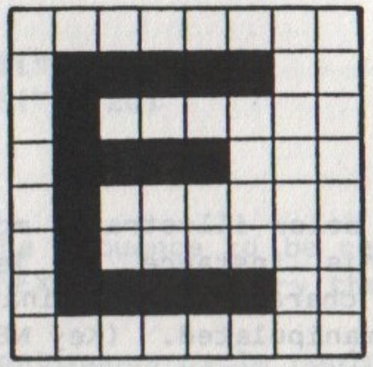


Fig. 17.2

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.

EINSTEIN uses the following codes for the inbuilt character set.

ASCII CODES 0 to 31
These are control codes and therefore not displayable as characters.

ASCII CODES 33 to 64, 91 to 96, 123 to 127
These are used to identify the numbers, symbols, and fractions in the character set.

ASCII CODES 65 to 90 - These are "upper case" letters of the character set.
ASCII CODES 97 to 122 - These are "lower case" letters of the character set.

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 the BASIC 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 in which to construct the desired character. Look at the example given in Fig.17.3. which illustrates a "little man".

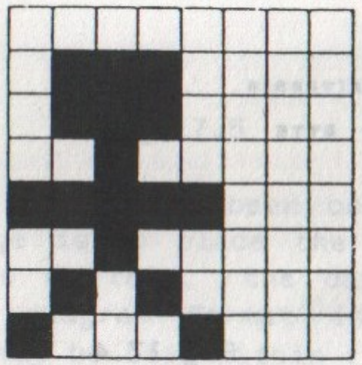


Fig.17.3

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.17.4 illustrates this.

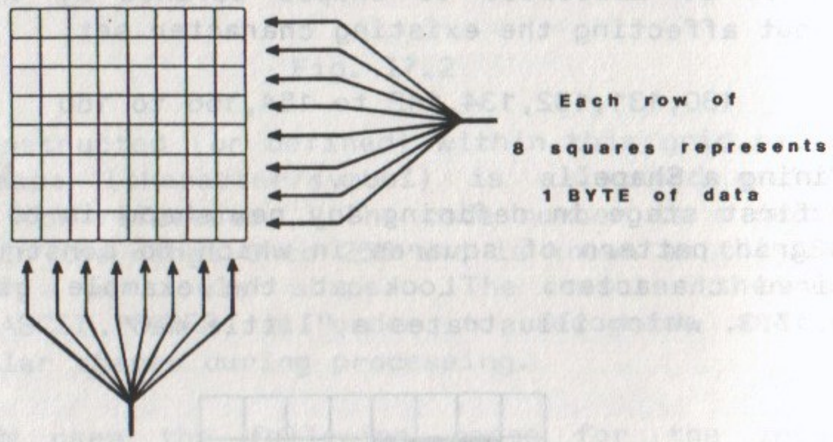


Fig.17.4

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 as shown in Fig.17.5.

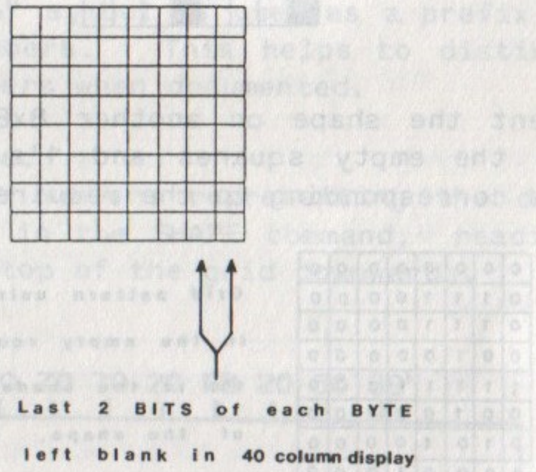
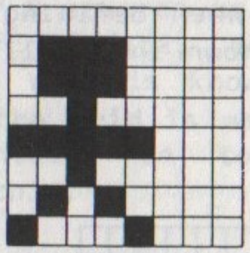


Fig.17.5

Once a particular shape has been constructed on the grid the next stage is to place the information into the computer. To do this, the data must first be converted from a diagram format into "Hexadecimal" format so that it may be used within the SHAPE command.

The following example illustrates the conversion process.

1. Contruct the shape on a grid pattern of 8x8 squares.



2. Represent the shape on another 8x8 grid by using 0's in the empty squares and 1's in the shaded squares corresponding to the required shape.

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.

Thus each BYTE (row) is now represented as an eight digit BINARY NUMBER.

3. Convert each BINARY number (row of 0's and 1's) into a HEXADECIMAL number by referring to the table of equivalent values in appendix H.

The first 4 bits represent the 1st hexdigit, the last 4 bits represent the 2nd hexdigit.

BINARY FORMAT	HEXADECIMAL FORMAT
0 0 0 0 0 0 0 0	&00
0 1 1 1 0 0 0 0	&70
0 1 1 1 0 0 0 0	&70
0 0 1 0 0 0 0 0	&20
1 1 1 1 1 0 0 0	&F8
0 0 1 0 0 0 0 0	&20
0 1 0 1 0 0 0 0	&50
1 0 0 0 1 0 0 0	&88

Note that an "&" symbol is used as a prefix to signify HEXADECIMAL numbers. This helps to distinguish them from other numbers when documented.

Entering a Shape.

The HEXADECIMAL numbers representing the data of the shape are used in the SHAPE command, reading them in order from the top of the grid downwards.

```
SHAPE 150,"00 70 70 20 F8 20 50 88"
```

Particular ASCII CODE selected by the user and allocated to this shape.

'HEX' data representing the particular shape required.

This command places the required data in the computer.

When selecting ASCII codes the following points should be considered.

i) ASCII codes 32 to 127 and 161 to 255 are used for the keyboard character set. Unless there is a need to redefine any of these characters avoid these codes.

ii) ASCII codes 0 to 31 are used for CONTROL-CODES and cannot be displayed.

iii) The range of codes left vacant and therefore available for use in the range 128 to 160 (see Page 151).

Should any of the alphanumeric characters be destroyed by the use of SHAPE, they can be restored using a cold start, but all programs and variables will be lost (see BASIC REFERENCE MANUAL).

Use of the Shape.
Having defined a shape and entered it in the computer, the next process is to make use of it.

Unlike the inbuilt character set, the new shape cannot be accessed from the keyboard (unless an existing keyboard character has been re-defined using the same ASCII code) therefore we have to make use of a different method for accessing characters. To do this the CHR$ function is applied.

By specifying the particular ASCII code in the CHR$ function then it will access **that** particular shape, character, or symbol.

Thus the "little man" shape previously defined and entered in the computer could be accessed as follows:-

```
PRINT CHR$(150)
```

By using this method of accessing the shape it can be manipulated and displayed in the same manner as any normal character shape on the screen.

```
PRINT CHR$(150)
```

Now try the following:-

```
10 CLS 32
20 SHAPE 150, "00 70 70 20 F8 20 50 88"
30 PRINT@ 15,10,CHR$(150)
40 END
```

Having observed the result key CLS E to clear the screen and key NEW E to clear the memory.

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 observe 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, key CLS E to clear the screen. Key NEW E 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 know 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,"081422410103070F1F3F7FFFFFFFFE7
    F8FCFEFFFFFFFE71028448280C0E0F0"
 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. The grid pattern is illustrated in Fig.17.6 to show the construction of the shape.

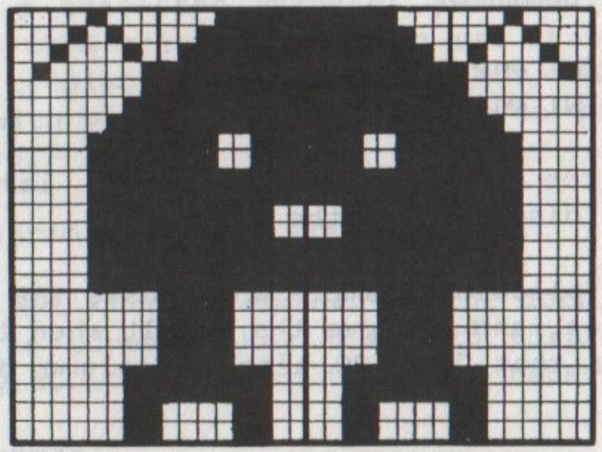


Fig.17.6

Now key CLS40 E to clear the screen and return to normal 40 column display.

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 principles described previously.

```
 10 REM ALIEN MOVE
 20 CLS32
 30 SHAPE 130,"081422410102070F1F3F7FFFFFFFFE7
    F8FCFEFFFFFFFFE71028448280C0E0F0"
 40 SHAPE 134,"0F0F0F0F0F0F0F0FE7FFFFFCFCFFFFFF
    E7FFFF3F3FFFFFFFF0F0F0F0F0F0F0F0"
 50 SHAPE 138,"0000000003030303F0F0F0F0FCFC0C0C0F
    0F0F0F3F3F303000000000000C0C0C0C0"
 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, then key CLS40 E 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 E

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 are so thin that the naked eye perceives them as one, then we have a comparison with sprite planes on the screen. Fig.17.7 illustrates this.

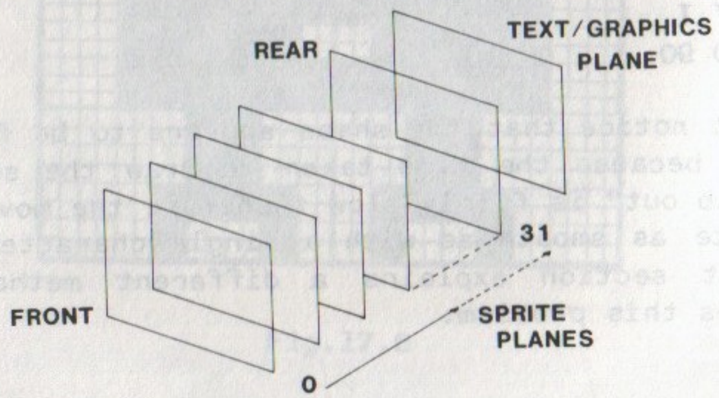


Fig.17.7

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).

The SPRITE command is as follows:-

SPRITE M,x,y,C,N

M - is the "sprite number" allocated by the user from the range 0 to 31.

N - is the number of a previously defined shape (ie. from the SHAPE command)

C - is the specified foreground colour (a number in the range 0 to 15) the background always being transparent).

x and y - are the co-ordinates on the screen pixel grid of the TOP LEFT HAND CORNER of the SPRITE. (positioning a sprite on the screen).

MAGNIFICATION

There are four modes of magnification numbered **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.17.8. In mode 0 the shape remains as a single 8x8 pixel character.

Fig.17.8

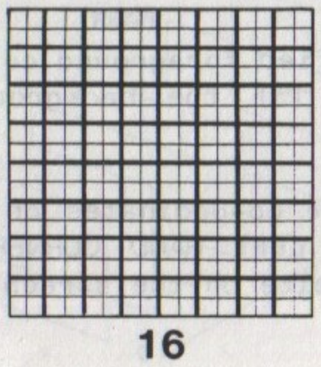


Fig.17.9

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.17.9.

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 70 20 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 E to return to 40 column display. Now type SPRITE OFF and key E to clear the screen. Key NEW E 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.17.10.

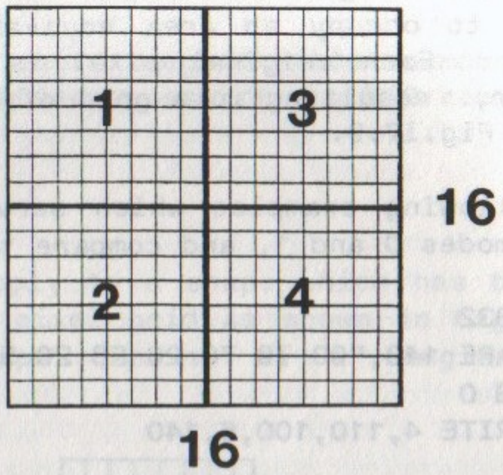


Fig.17.10

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 mulitple of 4 (eg. 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,"006040404141141E17F3F1F0000000000
    00303030B0FCB4FCFFFEFC0000000000"
30 MAG 2
40 SPRITE 6,70,100,6,152
```

2.
```
10 CLS32
20 SHAPE 152,"006040404141141E17F3F1F0000000000
    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 (ie. MAG 3). Type CLS40 and key E to clear the screen then type SPRITE OFF and key E 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
```

Having observed the result of this program use the ESC key to breakout of the loop then key CLS40 E 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 E to return the backdrop colour to normal.

FILLING AREAS OF COLOUR

Finally just a brief word in respect of 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 example 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
```

Further details of the FILL command are given in the EINSTEIN BASIC REFERENCE MANAUAL.

18

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 (see EINSTEIN BASIC REFERENCE MANUAL) 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 HEXNUMBER which is known as the address for the data it contains.

e.g. &0013
&012E
&215D
— Memory Locations

Each memory location can hold 1 BYTE of data represented by a two digit HEX NUMBER.

e.g. &34
&3F
&A2
— Two digit HEX NUMBERS

(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. Fig. 18.1 illustrates this.

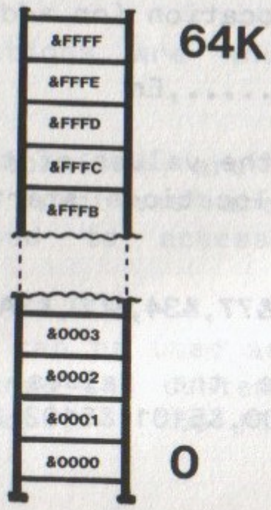


Fig.18.1

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. If this happens the only way out is to use the re-set button, thereby destroying any current user program.

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 such publications, for users to type into their own computers. The listings for MACHINE-CODE programs might appear in one of three different formats as follows:-

1. 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
0B13	C9

2. The presentation might consist of the ASSEMBLY SOURCE CODE LISTING which uses MACHINE-CODE MNEMONICS as shown below.

LINE NUMBERS		SOURCE CODE LISTING	
1563	VRMWR	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.

When loaded into the computer the assembler takes on the task of converting the "source-code" into "machine-code" automatically as it is typed in.

3. The presentation could consist of both the MACHINE-CODE LISTING with data, and the SOURCE-CODE LISTING using MNEMONICS combined into a single program list as follows:-

```
LOCATION  DATA    LINE    SOURCE-
                  NUMBERS CODE LISTING

0B07 F5         1563 VRMWR  PUSH    AF
0B08 79         1564        LD      A,C
0B09 D309       1565        OUT     (H'09'),A
0B0B 78         1566        LD      A,B
0B0C CBF7       1567        SET     6,A
0B0E D309       1568        OUT     (H'09'),A
0B10 F1         1569        POP     AF
0B11 D308       1570        OUT     (H'08),A
0B13 C9         1571        RET
```

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. (Details of the MODIFY command under MOS are given in the EINSTEIN DOS/MOS MANUAL).

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 Example '2' above, then an assembler becomes necessary. Without the assembler the SOURCE-CODE cannot be typed in. Assemblers available for EINSTEIN are indicated in the literature enclosed with the machine.

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)

1. 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.

The procedure for re-loading a program which has been saved in this manner is as follows:-

i) Go into **DOS** (CTRL/BREAK if not already there, or key DOS from BASIC).

ii) ENTER the program NAME only.

e.g. MAK

when ENTERED will load and run the program MAK.

2. To save a program which occupies locations between &5000 and &E000:-

a) Go into **BASIC**

b) Use the SAVE command in the following format.

SAVE "Name.OBJ",Start Address,End Address

where Name is the particular program title selected by the user, Start and End Addresses being the first and last hexadecimal memory locations of the listings.

e.g. SAVE"MACH.OBJ",&6020,&60FA

this will save the program MACH, which starts at &6020 and ends at &60FA in memory, as an .OBJ file on disc.

The procedure for re-loading a program which has been saved in this manner is as follows:-

i) Go into **BASIC**

ii) Use the CLEAR command to fix the memory location for loading of the program.

e.g. CLEAR &6018

this will cause the program to be loaded in beginning at location &6018

iii) Use the LOAD command as follows:-

LOAD "Name.OBJ"

where Name is the program title.

iv) Use the CALL command to invoke execution of the program.

e.g. CALL &6018

this will execute the machine-code program which starts at address &6018

FURTHER READING

For more information relating to Machine-Code and Machine-Code Programs the user should refer to publications written specifically on the subject.

GLOSSARY

Access:
The means of acquiring or adding information from a computer register, memory or peripheral unit.

Access Time:
The time which is taken to reference a particular item in storage, to read from or write to a memory location or a stored record on disc.

Accumulator:
A place in the computer (where operations take place on data)

Acoustic coupler:
A device which enables audible tones, say from a telephone, to be turned into the digital form which the computer can understand (and vice-versa).

Address:
In the computer, information in the form of numbers is moved around from place to place. Each place has an 'address' which is the number of the memory location where particular data is stored (like the number of a house in a street).

ALGOL:
A high level language - often used by mathematicians.

Algorithm:
The procedures leading to the accomplishment of a particular event?

Alphanumeric:
A mixture of characters which can be a combination of numbers, letters and symbols. The typewriter or computer keyboard is an 'alphanumeric' keyboard.

Analogue:
A quantity (e.g. temperature, time, voltage) which can vary and can be measured, giving them a numerical value.

A/D converter - Analogue to digital converter:
(See under CONVERTER.)

Applications program (or applications software):
A computer program designed for a particular outside purpose - it might be for a business application or a game or be an eductional program. As opposed to systems software (see Software.)

Arithmetic & Logic unit (ALU):
An area in the central processor of the computer where arithmetical and logical processes (such as comparing two numbers) take place.

Array:
An orderly list or a table of numbers or words (data) where every position is labelled and can be handled separately or in a sequence by the computer.

Artificial Intelligence:
The ability of a computer (or computer-controlled machine) to perform a task which, if a human being were to perform the same task, would be said to require 'intelligence'. Often the computer will learn from experience and improve its performance of a particular task.

ASCII:
'American Standard Code for Information Interchange'. The internationally accepted code which represents numbers, letters and symbols with unique binary code values which the computer can then deal with.

Assembly language ('assembler'):
A low level language which uses mnemonics rather than ordinary words to give instructions to the computer. These mnemonics translate directly into the binary instructions which the computer understands in a much more economical way than does a high level language in that they take up less memory space.

Backing storage:
Any outside storage medium (usually magnetic tape or disc) which supports and can be linked to the main memory in the computer. When the power is off, information in the backing storage is not lost. The capacity of a backing storage memory is much greater than the computer's internal working memory.

Bar code:
A pattern of printed lines on an object identifying it and containing information about it which can be read into the computer by scanning it with a light pen. Common now in supermarkets and libraries.

BASIC:
A very popular 'high level language' for microcomputers. Stands for 'Beginners All-purpose Symbolic Instruction Code'.

Baud rate:
This is a measure of the number of bits per second travelling from one part of a computer system to another, or between computers and peripherals. It is the rate at which data is transmitted, or received in serial data system.

Binary:
A way of counting using only two alternative values - 0 or 1, 'on' or 'off', 'black' or 'white'. It is the foundation of all computers.

Bit:
A binary digit - a '0' or a '1' (see binary).
A Number to base 2.

Boot:
To start up a computer. Abbreviation for executing bootstrap program.

Bootstrap Program:
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'.

Central processing unit: (CPU)
The control 'brain' of the computer where all parts of the computer system are linked together and where the calculations and manipulation of data take place.

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. When packaged up, looks like a centipede because of its many metal legs.

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.

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 digitial (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 the reverse.

CP/M: (Control Program for Microprocessors)
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.

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 with 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. TATUNG/CRYSTAL BASIC, BBC BASIC, RML BASIC - all are different dialects of BASIC with some things in common, others not.

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 peform the task specified.

Exponent:
A number indicating the power to which a number or 'expression' is to be raised.

Expression:
Representation of a mathemetical 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 '0s' and '1s' 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 '1s' and '0s'. 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 sumbol 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 centra 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 control.

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 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.

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 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.

APPENDIX A

ERROR MESSAGES

Error Messages	CODE 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	Auxiliary 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	Tape/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
DISC ERRORS			
No File	19	25	File not found
File Exists	1A	26	File already present in REName)
File Locked	1B	27	File has been locked ('Read Only')
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

APPENDIX B

CODE TABLE/GRAPHICS SET

b_4 b_3 b_2 b_1 ROW / COL	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	p
0 0 0 1	1	SCREEN DUMP	CURSOR ON	!	1	A	Q	a
0 0 1 0	2	STX	PRINTER ON	"	2	B	R	b
0 0 1 1	3	ETX	PRINTER OFF	#	3	C	S	c
0 1 0 0	4	CURSOR RIGHT	CURSOR OFF	$	4	D	T	d
0 1 0 1	5	ENQ	DELETE TO END OF LINE	%	5	E	U	e
0 1 1 0	6	DELETE CHAR FROM RIGHT	ERASE TO END OF SCREEN	&	6	F	V	f
0 1 1 1	7	BEEP	ETB	'	7	G	W	g
1 0 0 0	8	FE,(BS)	DELETE WHOLE LINE	(	8	H	X	h
1 0 0 1	9	FE,(HT)	DEL	)	9	I	Y	i
1 0 1 0	A	FE,(LF)	INS	*	:	J	Z	j
1 0 1 1	B	FE,(VT)	ESC	+	;	K	←	k
1 1 0 0	C	FE,(FF)	FS	,	<	L	½	l
1 1 0 1	D	FE,(CR)	GS	-	=	M	→	m
1 1 1 0	E	SO	HOME	.	>	N	↑	n
1 1 1 1	F	SI	US	/	?	O	↓	o

Key to Control Character Codes:-

NUL	-	Null
DLE	-	Data Link Escape
STX	-	Start of Text
ETX	-	End of Text
ENQ	-	Enquiry
FE,(BS)	-	Back Space
FE,(HT)	-	Horizontal Tabulation
FE,(LF)	-	Line Feed
FE,(VT)	-	Vertical Tabulation
FE,(FF)	-	Form Feed
FE,(CR)	-	Carriage Return
SO	-	Shift - out
SI	-	Shift - in
ETB	-	End of Transmission Block
DEL	-	Delete
INS	-	Insert
ESC	-	Escape
FS	-	File Separator
GS	-	Group Separator
HOME	-	Cursor Home
US	-	Unit Separator

APPENDIX C

COLOUR TABLE

0	Transparant
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 D

ASCII CODES

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
32	SP	64	@	96	£	128		160		192		224	
33	!	65	A	97	a	129		161		193		225	
34	"	66	B	98	b	130		162		194		226	
35	#	67	C	99	c	131		163		195		227	
36	$	68	D	100	d	132		164		196		228	
37	%	69	E	101	e	133		165		197		229	
38	&	70	F	102	f	134		166		198		230	
39	'	71	G	103	g	135		167		199		231	
40	(	72	H	104	h	136		168		200		232	
41	)	73	I	104	i	137		169		201		233	
42	*	74	J	106	j	138		170		202		234	
43	+	75	K	107	k	139		171		203		235	
44	,	76	L	108	l	140		172		204		236	
45	-	77	M	109	m	141		173		205		237	
46	.	78	N	110	n	142		174		206		238	
47	/	79	O	111	o	143		175		207		239	
48	0	80	P	112	p	144		176		208		240	
49	1	81	Q	113	q	145		177		209		241	
50	2	82	R	114	r	146		178		210		242	
51	3	83	S	115	s	147		179		211		243	
52	4	84	T	116	t	148		180		212		244	
53	5	85	U	117	u	149		181		213		245	
54	6	86	V	118	v	150		182		214		246	
55	7	87	W	119	w	151		183		215		247	
56	8	88	X	120	x	152		184		216		248	
57	9	89	Y	121	y	153		185		217		249	
58	:	90	Z	122	z	154		186		218		250	
59	;	91	←	123	¼	155		187		219		251	
60	<	92	½	124	‖	156		188		220		252	
61	=	93	→	125	¾	157		189		221		253	
62	>	94	↑	126	÷	158		190		222		254	
63	?	95	_	127	■	159		191		223		255	DEL

APPENDIX E

HARDWARE TECHNICAL SPECIFICATIONS

CPU:

Z80A-clock frequency 4MHz

MEMORY:

64k RAM
8k ROM with expansion for up to 32k internally.
16k separate video RAM

DISPLAY:

Hi-res graphics 256 x 192 pixels, plus 32 planes of "sprites". In addition to "sprites" there is a text/graphics and backdrop plane. The display is capable of 16 colours:

Transparent	Dark Blue	Mid Red	Dark Green
Black	Pale Blue	Pale Red	Magenta
Mid Green	Light Yellow	Dark Red	Grey
Pale Green	Cyan	Dark Yellow	White

DISPLAY FORMAT:

Raster scanned 625 lines 50Hz field non-interlaced.

Resolution: 256 x 192 pixels.

Display Modes:

5 - Graphics I 32x24 cells of 8x8 pixels 2 colours/cell (one foreground, one background colour).

- Graphics II 32x24 cells of 8x8 pixels 16 colours/cell (8 foreground, 8 background colours).

- Multicolour 64x48 cells of 4x4 pixels 1 colour/cell (foreground colour only)

- 32 Sprites either 8x8, or 16x16 with an optional magnification factor of two. 1 colour/sprite. Sprites are arranged in 32 planes, plane 0 having the highest priority for display.

- Text 40 x 24 or 32 x 24 characters on a 5x7 matix 256 characters/patterns may be stored in memory, 960 can be displayed on the screen.

Backdrop One of 16 colours. The backdrop illuminates the whole screen behind the text, graphics & sprites.

DISPLAY PRIORITY:

Sprite 0 (highest), Sprite 31, Text/Graphics, and Backdrop (lowest).

CHARACTER GENERATOR:

96 alphanumeric characters and 160 graphics symbols. All characters and symbols are software re-programmable.

RF OUTPUT:

UHF: 591.25MHz 1.5mV peak syncs, negative modulation, PAL encoded.

DISPLAY OUTPUT: (user selectable)

YUV: Y-luminance signal with negative going syncs. 1V p-p into 75 ohms.
U & V-chrominance signals. 0.68V p-p into 75 ohms.

RGB and Syncs.: 1V p-p into 75 ohms.

RS232-C PORT:

Full duplex capability to RS232-C/V24 standards. Transmission speeds are software programmable between 75 & 9600 bauds. Signals available - Tx data, Rx data, RTS, CTS, Ground.

USER PORT:

8 bit bi-directional with strobe and ready signals TTL levels.

ANALOGUE TO DIGITAL CONVERTER:

4 Channels, 8 bits resolution. Conversion speed less than 40us (for 8 bits). Input range 2V.

PRINTER OUTPUT:

Centronics interface standard.

TATUNG 'PIPE':

Z80A buffered bus to TTL levels, with clock and control signals.

EXTERNAL DISC INTERFACE:

Provides signals for up to two self-powered external disc drives. Either 3 inch compact floppy, 3½ inch micro floppy disc, or 5¼ inch mini floppy.

PROGRAMMABLE SOUND GENERATOR:

A means of providing a variety of sounds, including chromatic music, with envelope shaping is provided. The sound generator has three 'voices'.

SOUND OUTPUT:

250mW into internal 3½ x 2¼ inch elliptical loudspeaker.

SWITCH MODE POWER SUPPLY UNIT:

Type - Self Oscillating Flyback Converter
20 - 40KHz

Mains - 220V to 240V
50 to 60 Hz

Outputs (Fused @ 1AMP)
External 5V output load -
Tatung pipe 1 amp per pin max.
Analogue ports 1 amp per port max.
User port 1 amp max.

All outputs are overvoltage and short circuit protected.

KEYBOARD: Full travel typewriter style QWERTY keyboard

1) 48 alphanumeric/graphics keys, 8 function keys 11 'control' keys.

2) All keys to have n lockout with 1 key roll-over.

3) Automatic repeat.

4) Programmable repeat delay and repeat speed.

DISC DRIVES:

Teac FD30A 3 inch disc drive is to be incorporated

Specification:

Single sided
100 tpi
40 tracks
MFM coding
Disc fully enclosed in a cassette.

Access time:

12ms track-track
171ms average

Transfer rate:

250k bit/sec.

Sectors/track:

10

Bytes/Sector:

512

Media Specification:

500k byte unformatted capacity (250k/side)
400k byte formatted 200k/side)
80 tracks (40/side)
Double density (MFM) recording
100 tpi
Double sided 'flip-over' cassette

FEATURES AND SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT NOTICE.

APPENDIX F

EXTERNAL SOCKETS AND CONNECTORS - Pin Specifications

1) Analogue 1/Ánalogue 2 7-Pin Din connectors M014 & M015

2) RS232-C 5-Pin Din connector M013

3) User Input/output IDC Connector M002

4) Printer Output IDC Connector M001

5) Printer Connector (Epson MX80)

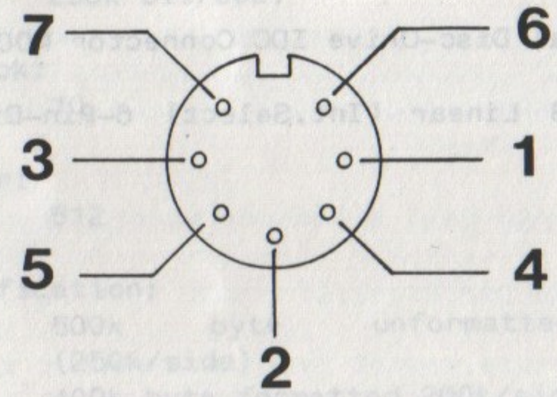
6) Tatung Pipe 60-way IDC Connector M003

7) External Disc-Drive IDC Connector M004

8) YUV-RGB Linear (Int.Select) 6-Pin-Din connector M016

1) Analogue 1/Analogue 2 7-Pin Din Connector M014 & M015

PIN NO.	ANALOGUE - 1 CONNECTOR M014	ANALOGUE- 2 CONNECTOR M015
1	Channel 0	Channel 2
2	Signal Ground	Signal Ground
3	Channel 1	Channel 3
4	Fire - 1	Fire - 2
5	Vref	Vref
6	Ov	Ov
7	+5v	+5v

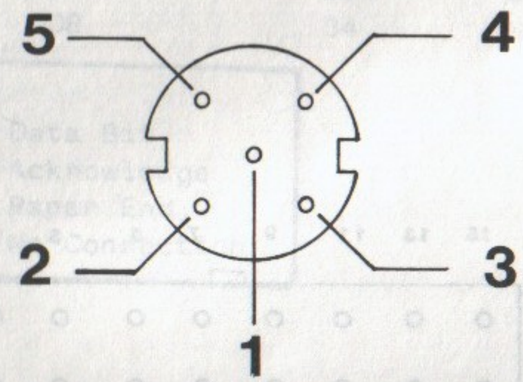


VIEW LOOKING INTO SOCKET

2) RS232-C 5-Pin Din Connector M013

PIN NO.	SIGNAL	DIRECTION
1	Ov	- -
2	CTS	To Computer
3	TxD	To DCE
4	RTS	To DCE
5	RxD	To Computer

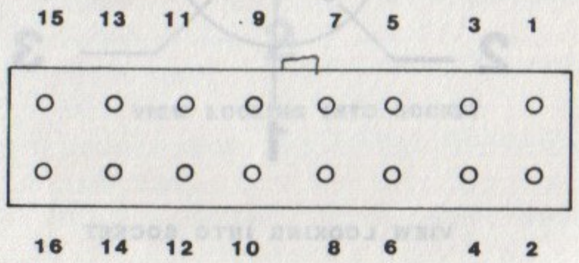
Key:-
DCE - Data Communication equipment (eg. MODEM)
TxD - Transmit Data
RxD - Receive Data
CTS - Clear to Send
RTS - Request to Send.



VIEW LOOKING INTO SOCKET

3) User Input/output IDC Connector M002

PIN NO.	SIGNAL
1	5v
2	DO
3	Ov
4	D1
5	RDY
6	D2
7	Ov
8	D3
9	Ov
10	D4
11	STB
12	D5
13	Ov
14	D6
15	5v
16	D7

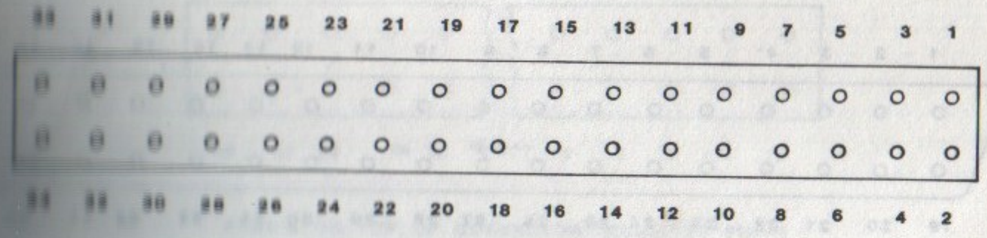


VIEW LOOKING INTO CONNECTOR

4) Printer Output IDC Connector M001

PIN NO.	SIGNAL	PIN NO.	SIGNAL
1	STROBE	18	Ov
2	Ov	19	ACK
3	D1	20	Ov
4	Ov	21	BUSY
5	D2	22	Ov
6	Ov	23	PE
7	D3	24	Ov
8	Ov	25	N/C
9	D4	26	N/C
10	Ov	27	N/C
11	D5	28	ERROR
12	Ov	29	N/C
13	D6	30	N/C
14	Ov	31	Ov
15	D7	32	N/C
16	Ov	33	Ov
17	D8	34	N/C

KEY:-
D - Data Bit
ACK - Acknowledge
PE - Paper End
N/C - No Connection

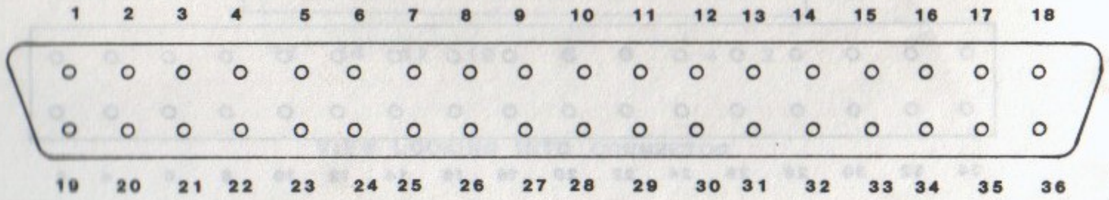


VIEW LOOKING INTO CONNECTOR

5) Printer Connector (Epson MX80)

PIN NO.	SIGNAL	RETURN
1	STROBE	Pin 19
2	D1	20
3	D2	21
4	D3	22
5	D4	23
6	D5	24
7	D6	25
8	D7	26
9	D8	27
10	ACK	28
11	BUSY	29
12	PE	30
13	N/U	
14	N/U	
15	N/U	
16	OV	
17	GND	
18	N/C	
31	N/U	
32	ERROR	
33	N/U	
34	N/C	
35	N/U	
36	N/C	

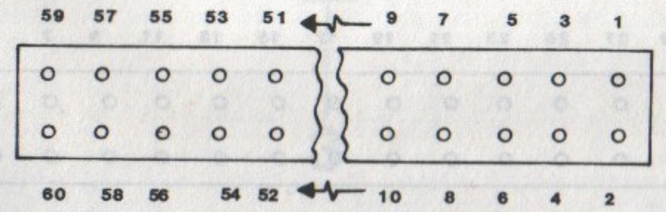
KEY:-
D - Data Bit
N/U - Not used by the TATUNG COMPUTER
N/C - No Connection
ACK - Acknowledge
PE - Paper End



VIEW LOOKING INTO CONNECTOR

6) Tatung Pipe 60-Way IDC Connector M003

PIN NO.	SIGNAL	PIN NO.	SIGNAL	PIN NO.	SIGNAL
1	+5v	21	A12	41	OV
2	D7	22	A11	42	$\overline{WR}$
3	+5V	23	A10	43	OV
4	D6	24	A9	44	$\overline{RD}$
5	OV	25	A8	45	OV
6	D5	26	A7	46	$\overline{IORQ}$
7	OV	27	A6	47	OV
8	D4	28	A5	48	MREQ
9	Ov	29	A4	49	OV
10	D3	30	A3	50	HALT
11	OV	31	A2	51	OV
12	D2	32	A1	52	NMI
13	Ov	33	A0	53	OV
14	D1	34	$\overline{RST}$	54	INT
15	OV	35	OV	55	OV
16	D0	36	$\overline{RFSH}$	56	WAIT
17	Ov	37	OV	57	OV
18	A15	38	$\overline{M1}$	58	BUSREQ
19	A14	39	OV	59	OV
20	A13	40	$\overline{BUSACK}$	60	SYS CLK (4MHz)

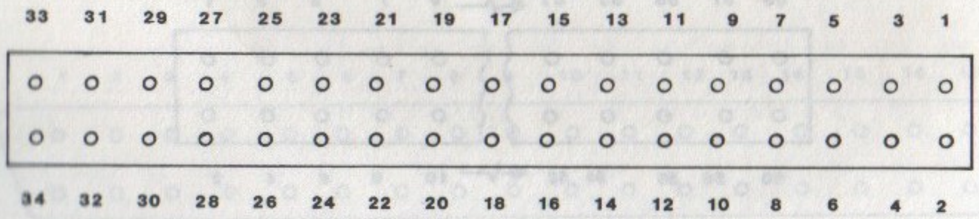


ARROW ON TOP OF CONNECTOR INDICATES PIN-1

7) External Disc-Drive IDC Connector M004

PIN NO.	SIGNAL	PIN NO.	SIGNAL	DIRECTION
1	Ov	2	N/C	---
3	Ov	4	N/C	---
5	Ov	6	D/S-3	TO DRIVE
7	Ov	8	INDEX	TO COMPUTER
9	Ov	10	D/S-0	TO DRIVE
11	Ov	12	D/S-1	TO DRIVE
13	Ov	14	D/S-2	TO DRIVE
15	Ov	16	MOTOR ON	TO DRIVE
17	Ov	18	DIR/S	TO DRIVE
19	Ov	20	STEP	TO DRIVE
21	Ov	22	WRITE DATA	TO DRIVE
23	Ov	24	WRITE GATE	TO DRIVE
25	Ov	26	TRACK 0	TO COMPUTER
27	Ov	28	WRITE PROTECT	TO COMPUTER
29	Ov	30	READ DATA	TO COMPUTER
31	Ov	32	SIDE-SELECT	TO DRIVE
33	Ov	34	N/C	

KEY:-
N/C - No connection
D/S - Drive-Select
DIR/S - Direction Select



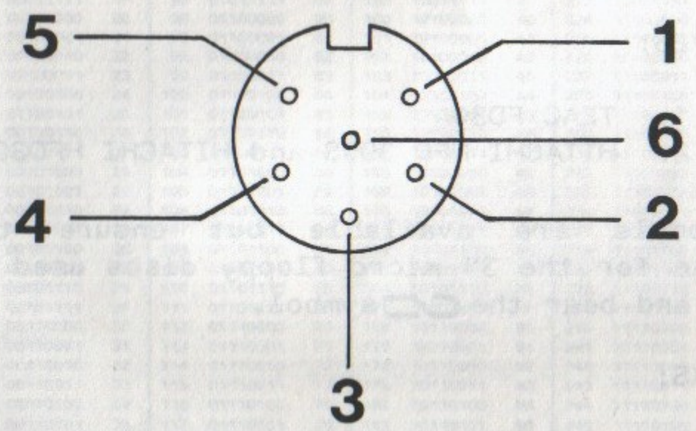
VIEW LOOKING INTO CONNECTOR

8) YUV-RGB Linear (Int.Select), 6-Pin-Din Socket for Video

PIN	MONO	RGB SIGNAL*	YUV OUTPUT*
1		R	V
2	CORE	G	Y+SYNCS
3		B	U
4		SYNCS	N/C
5	SCREEN	OV	OV
6		N/C	N/C

N/C - No Connection

*The outputs are internally user selectable via link switches M100 and M101. (See Appendix I)



VIEW LOOKING INTO SOCKET

APPENDIX G

PERIPHERALS

The following models are suitable for use as peripherals with the EINSTEIN.

PRINTERS:

1. TATUNG TP80
2. EPSON FX80 - The TATUNG character set can be programmed into this model

The following printers can also be used with EINSTEIN 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 RX80
2. FACIT 4510
3. SHINWA CT1CP80

DISC DRIVES:

1. TEAC FD30A
2. HITACHI HFD 305S and HITACHI HFD305SX

Other models are available but ensure they are compatable for the 3" micro floppy discs used with the computer and bear the CE symbol.

JOY STICKS:

Model	Type	Manufacturer
J1 (TATUNG)	Analogue	Flight Link Control
'Sure Shot'	Digital	R.P. Products

APPENDIX H

DECIMAL/BINARY/HEXADECIMAL EQUIVALENTS

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 I

DISPLAY MONITOR SIGNAL OPTIONS

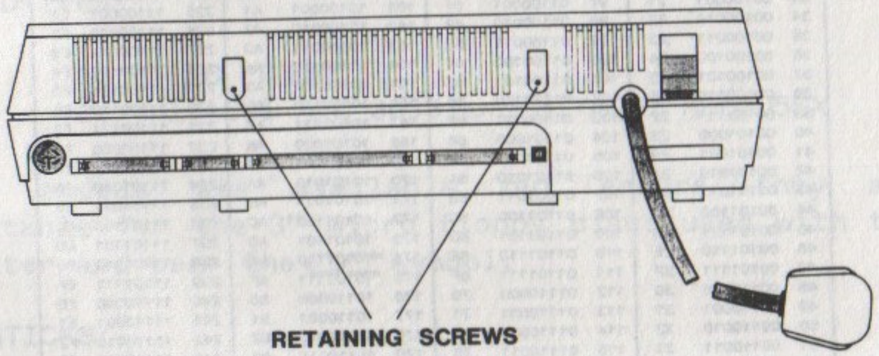
Selection of the desired standard (YUV or RGB), is determined by the positioning of 4 'Mini-shunt' links located on the main printed circuit board assembly.

To gain access - refer to the following notes:-

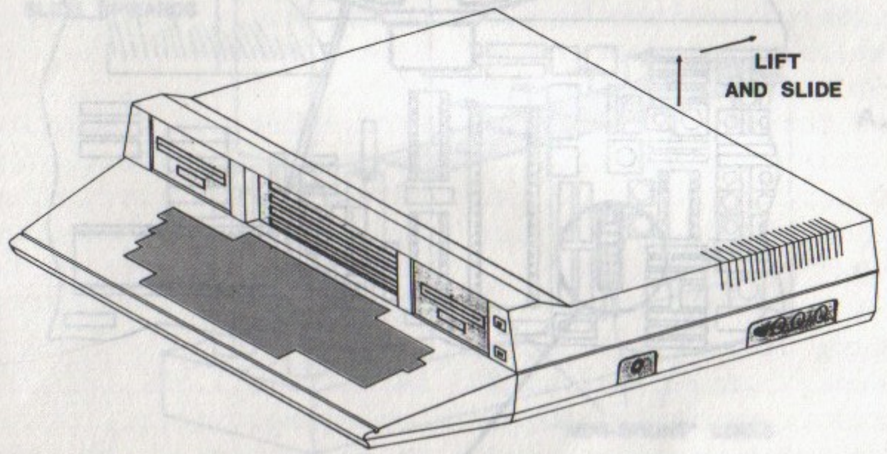
Note: Before commencing - first switch-off the power supply, unplug the mains lead from the supply and disconnect all other inter-connecting cables.

Removal of Cover:
The moulded top cover of the microcomputer may be removed as follows:-

1. Unscrew and remove the 2 retaining screws located in the rear face of the cover.



2. Lift the back of the cover gently - and in the same movement - slide the cover towards the rear of the unit, to disengage and clear from their respective locations in the base unit assembly, the 4 retaining lugs moulded into the cover's front top and sides.



3. Lift the cover away to provide access to the microcomputer interior.

Selection of 'Mini-Shunt' Locations:
With the cover removed, refer to Fig.A and Fig.B. These diagrams show the location of the 'Mini-Shunt' plug link connectors (M100 & M101) on the main PCB, and the respective positions of the links themselves.

Link positions for YUV format are shown in chain dotted outline. Positions for the change to RGB format are shown in solid outline.

Fig.A

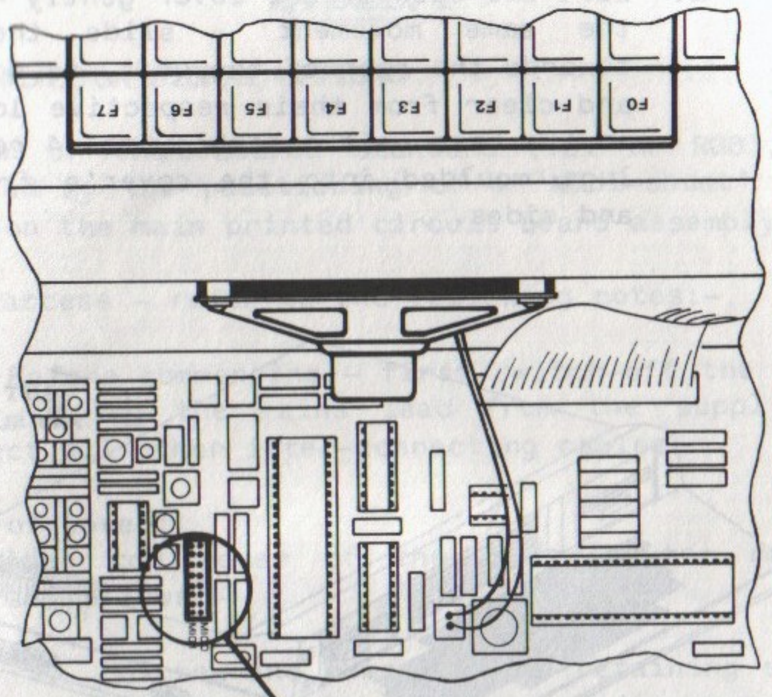
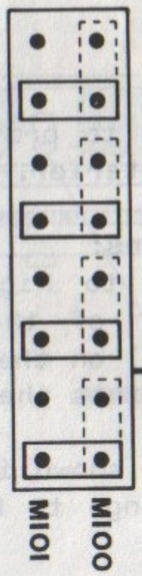


Fig.B

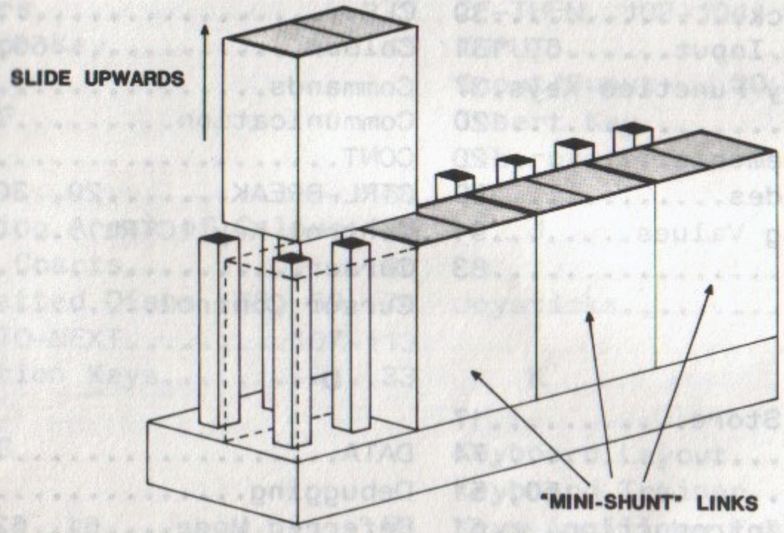


Location of "Mini-Shunt" link connectors on P.C.B.

Link positions

YUV -Dotted outline
RGB -Solid outline

To change from one format to the other, carefully slide all 4 links upwards off the vertical location Pins (as illustrated below) and replace them in the required alternative positions as indicated in Fig.B.



Finally, replace the top cover as decribed below.

Replacing the Cover:

1. Ensure the 4 retaining lugs are correctly located into their respective slots and into the retaining lip on the moulded base unit assembly.

2. Finally, check that the cover and base are correctly aligned along their edges, then press the cover down evenly onto the base moulding and refit the two retaining screws.

INDEX