Steven Blake

LEARNING TO USE THE ORIC 1 COMPUTER

Steven Blake

LEARNING TO USE THE ORIC 1 COMPUTER

This beginners' guide really does begin at the beginning. It assumes that you want to learn to *use* the Oric 1 computer in your work or leisure, not become a theorist in computing. *Learning to Use the Oric 1 Computer* provides a simple, down to earth, jargon-free introduction to the machine and its software. Follow the text and illustrations and you will end up operating the Oric 1 and understanding its many capabilities.

Many applications of the Oric 1 are described, including business, educational and hobby uses. Additionally, a simple and direct introduction to programming the Oric 1 is given in a way which will help motivate the user to further investigation of the Oric 1's capabilities. The Oric 1's ability to produce and draw pictures and diagrams is explored and explained, and programs for a large number of graphics applications are presented.

This book will appeal to new Oric 1 owners, students in schools and colleges where Oric 1s are used, businessmen who wish to learn about how to use the Oric 1 and program it. It will help those who are already learning to use the Oric 1, but find their current manuals difficult to follow. It also provides the would-be purchaser of microcomputers with information on how the Oric 1 operates and performs, which will help him to assess whether the machine will suit his need.

About the series

This series of books has been designed to provide potential users, established users, teachers, students and businessmen with standardised introductions to the use of popular microcomputers. Extensive use has been made of photographs, diagrams and drawings to illustrate the text and make it easy to read and understand.

As the layout and content of the books in the series are similar, each book may be used in conjunction with others for purposes of comparison of performance and capabilities. The *Learning to Use* series is an inexpensive way of checking that the would-be purchasers' provisional choice of machine is the correct one.

The series is open-ended and will cover new models of microcomputers as they appear on the market.

£4.95

Gower

ISBN 0 566 03495 6
Gower Publishing Company Limited, Gower House, Croft Road, Aldershot, Hampshire GU11 3HR, England

Learning to Use the Oric 1 Computer

Learning to Use the Oric 1 Computer

by Steven Blake

Gower

Published by
Gower Publishing Company Limited,
Gower House, Croft Road, Aldershot,
Hampshire GU11 3HR, England

British Library Cataloguing in Publication Data

Blake, Steven,
Learning to Use the Oric 1 Computer.—(Learning to use)
1. Oric 1 (Computer)
I. Title
001.64'04 QA76.8.0/

ISBN 0 566 03495 6

Typeset and printed in Great Britain
at FD Graphics, Fleet, Hants.

Contents

List of Figures

Foreword

This series of books is designed to fill the gap left by nearly every other book ever published on microcomputing. For some reason there has always been a surfeit of books which assume a thorough basic knowledge; but few books to help you find that knowledge in the first place. Even the manuals supplied with most systems seem to assume you understand the main principles from the beginning: in fact, most are only useful if you don't need them! This open-ended series of books from the Gower Publishing Company provides the new computer user with a jargon-free introduction to his machine. He doesn't need to have any previous computing experience, for the book itself provides all the relevant information necessary to get started: and if he does come across any strange new jargon, there is a useful glossary at the back. It is not, and is not intended to be, a text book on BASIC programming; but after reading this book even the complete beginner will be able to refer to a text book with complete confidence.

Most young readers, and the books are primarily (but not solely!) designed for young readers, will be using their machines at school or at home. It was probably bought by parents to help the family get used to the new technology; or perhaps it is the school's new system to help in teaching computer science. If it is used at home, it is almost certain that parents will join in the learning process – and in many cases, they may well start to monopolise it! Fifty years ago, these same parents would have bought their children a train set, and would then play with it for hours on end themselves. We can expect the same to happen with the new micros, for programming can become a very addictive pastime! This series could be as useful for the adult novice as it is for the younger reader.

Computing and computer skills are a modern concept, and we will all need to understand at least the basic principles in a world that is becoming increasingly computer orientated. Since our future is to be based on the computer as a useful tool to help run our factories, offices, schools and homes, we need to know how to get the most out of them. At best, the computer can be a useful, profitable, enjoyable, and beneficial aid; and at worst it can be an expensive, useless conglomeration of electronic junk. The *Learning to Use* series of books is designed to help new users get the best from their computers as quickly and as easily as possible.

The books describe a number of applications for each computer, including business, education and hobbyist. Furthermore, a simple and direct introduction to programming is included in such a way as to motivate further investigation of the computer and its capabilities. Each computer's ability to draw pictures and diagrams, in black and white or colour, is explored and explained, and programs for a large number of graphics applications are presented. Wherever available, details of each system's sound reproducing capabilities, including example programs and a version of the national anthem, are also included. For the programs, the series is indebted to William Turner, a lecturer in statistics at the Oxford Polytechnic – and no mean programmer! – who has had the unenviable task of converting somebody else's programs for each new book. The Tower of Hanoi game at the end of each book is entirely his own program and illustrates the teaching philosophy he employs at his college: elegant simplicity.

Two further commendations are necessary: to Michael Fluskey of Newtech Publishing Ltd who first conceived the idea for this series and has throughout been the driving force behind it; and to Garry Marshall who wrote the first book in the series: *Learning to Use the PET Computer*. It was the joint work of these two that developed the basic structure that is now used throughout the series.

This said, I can only wish the reader as much enjoyment from the book and his computer, as I have from mine.

Kevin Townsend
Editor, Micro Software and Systems Magazine

Chapter 1
Introduction to the Oric

What is the Oric?

The Oric 1 (which, for the rest of this book, we shall abbreviate to 'the Oric') is a computer. It is usually called a microcomputer (and sometimes a personal computer or home computer) because it is extremely small compared to early computers – and also because its electronic 'heart' is a microprocessor. As you can see from Figure 1.1, the Oric appears like a case with a keyboard, rather like a conventional typewriter. In use, it requires a screen of some sort for its display. In order to keep down costs, it has been designed to work primarily with an ordinary television set (either black and white, or colour).

Inside the Oric there is a number of integrated circuits, or chips, as shown in Figure 1.2. One contains the microprocessor. Others provide the computer's memory, and can store information. Initially, there is no need to worry about the inside of the computer. The electronic circuitry

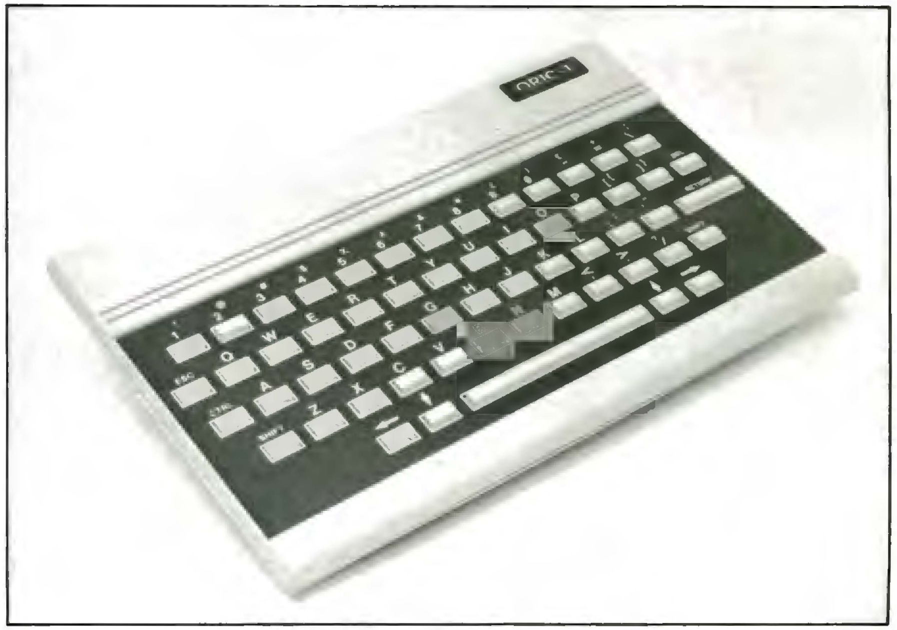


Figure 1.1 The Oric.

and the devices that make the Oric work are fascinating, but a detailed understanding of them is certainly not necessary in order to use the computer – and this introductory book is about learning to use the Oric computer. Indeed, since early versions of the Oric are supplied with the dire warning '**WARRANTY INVALID** if this case is opened' it is advisable that you learn not to worry too much about what the inside looks like.

The main feature, the keyboard, is for communicating to the computer. Commands that are to be obeyed, and information that is to be stored, can simply be typed in. Because the keyboard is set out in the same way as a typewriter, a good typist can type almost as quickly as on an ordinary typewriter. (We shall look at some of the 'new' keys shortly.) Notice, however, that if you try to type too fast, and do not perhaps press the keys firmly enough, it is quite possible for the Oric to miss some of the keys you think you have pressed. Nevertheless, it is a good idea from the start to try to use the professional 'five-finger' typing techniques rather than one-finger tapping: in the long run, this will save a considerable amount of time. Once the correct connections are made to the television set, anything you type on the keyboard will automatically appear on the television screen.

The Oric computer possesses a number of what are called 'screen editing facilities'. These make it fairly simple and easy for you to 'edit' your typing; that is, to correct errors, to make changes, and to arrange for the revised typing to appear on the screen. The Oric's screen editing facilities have been very carefully thought out. With a little practice, you will find them easy to use.

Besides letters and numbers, the Oric also lets you produce simple pictures on the screen. This facility is known as 'graphics', and is an impressive bonus to the use of any computer. The imaginative use of pictures, diagrams and graphs enlivens the presentation of information,

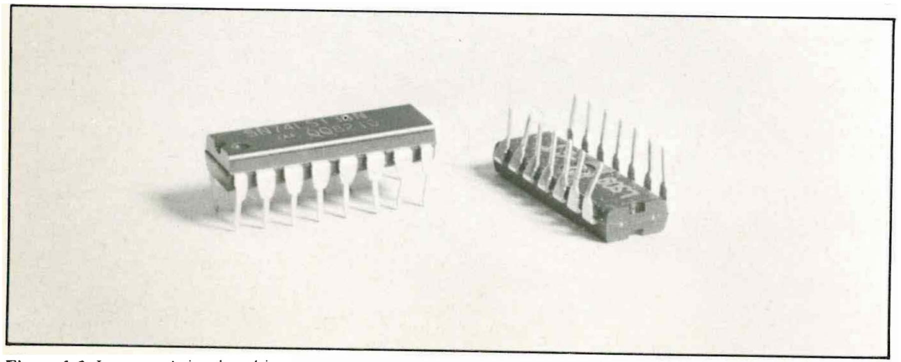


Figure 1.2 Integrated circuits: chips.

and can be used in computer games, in business applications, and in educational programs.

The Oric is light and compact, weighing less than 2lbs and measuring only 11-inches by 7-inches, and is small enough to be carried from room to room, or from house to house, or even from schoolroom to schoolroom. It can be set up and used in its new location quickly and easily, for it needs only to be connected via its own power transformer to the mains and connected to an ordinary domestic television set. Furthermore, as soon as it is switched on it is ready to accept commands typed in at the keyboard, provided only that they are typed in a language that the computer understands. The language is BASIC, and it enables you to issue commands that are promptly and automatically obeyed by the Oric computer.

There are many different versions of BASIC, produced by different software companies. The Oric has its own extended BASIC, which is held inside the computer. Throughout this book we shall concentrate on the inbuilt Oric BASIC that is supplied with the computer. Unless we specifically state otherwise, whenever we refer to 'BASIC', we are referring to this Oric BASIC.

How was the Oric developed?

The two main events that have hastened the advance of microelectronics, and microprocessors in particular, are the space race of the 1960s and the general requirements of the world's various defence organisations (particularly the US Department of Defense, and the UK Ministry of Defence) ever since the end of the Second World War. In the space race, because the US rockets were less powerful than those of the Russians, the Americans needed to reduce the size and weight of everything that had to be carried by their rockets – including the electronics. At the same time, defence leaders began to demand computers that were small enough, light enough, and tough enough to be carried around in, and survive, battle conditions (for example, to provide computerised range finding within tanks). In particular, this stimulated the American electronics industry to investigate and develop means of miniaturising electronic circuitry – and the result is the microprocessor, often referred to as the chip.

Not only is the microprocessor extremely small (smaller than the size of the average finger-nail) and at the same time relatively powerful (as powerful as the early computers that would fill an average sized room), it is a multi-purpose device that can perform any electronic function for which it can be programmed. This versatility has led to the use of microprocessors in a wide and ever-growing range of applications (the most common, and best known, is, of course, as the 'heart' of a general purpose microcomputer like the Oric). The consequent mass production of microprocessors has caused the cost per unit (of the microprocessor – not the microcomputer!) to drop to just a few pence.

The particular history of the Oric microcomputer can be traced to the great boom in small, inexpensive home computers caused by Clive Sinclair's innovative ZX80 microcomputer. Other manufacturers had tried to break into the potentially massive home/hobby computer market (notably the Personal Electronic Transactor – PET), but had, at least on this side of the Atlantic, conspicuously failed.

Sinclair succeeded where his predecessors failed because he understood the price limitations for the UK and European markets. Where the American family might spend up to $1,000 for a home computer, the European family would not spend more than £100. Sinclair introduced a microcomputer for less than £100. Since that time, the success of the ZX80 and his subsequent machines has educated and opened the market to a new range of slightly more expensive and considerably more powerful home microcomputers. The Oric is one of these.

The Oric was designed by Tangerine Computer Systems, a UK company with an established history in the manufacture of hobby equipment. The principal designer, Dr Paul Johnson, together with the current Oric Products International managing director, Barry Muncaster, had been jointly responsible for the development of Tangerine's Microtan self assembly computer and, perhaps more relevantly, the Tantel Prestel adaptor. It is consequently not surprising to note that the Oric has several Prestel compatible features, including a similar graphics character set and the seven standard Viewdata colours among its total of sixteen colours. The same two gentlemen were, incidentally, also responsible for the development of the electronic meter used in today's taxi cabs.

Oric Products International was subsequently established to market the new Oric computer. The name chosen for the new machine is variously supposed to be a 'beheaded anagram of micro'; or, perhaps more credibly, by phonetic association, a variant and therefore relative of Avon's (from the television series 'Blake's Seven') ORAC. ORAC's great strength lay in its ability to tap the communications channels of all other computers until it became, in a sense, the sum total of all existing computers. With the history of its designers' association with Prestel equipment, we may well expect communications to figure strongly in the Oric's future development.

The Oric is based on the MCS 6502 microprocessor, a processor manufactured by a subsidiary of Commodore, and identical to the one

used in many of the amusement arcade games and a number of other successful microcomputers – notably, the PET itself, the VIC-20, and the Apple II. Ironically, given the military origins of microelectronics, this microprocessor is a more advanced example of its technology than those used in the guidance systems of the inter-continental ballistic missiles, and even the ultra-modern Exocet-type missiles! This processor is the heart (and brain!) of the Oric. Although it does all the hard work, all the computing and calculations, you can tap its potential and make it work for you without having any detailed knowledge of how it functions.

While many manufacturers release several versions of any new computer (usually based on the size of its RAM, or internal memory), Oric Products International has so far released just the single machine: the Oric 48K system. The '48' refers to the size of its memory, which is 48K RAM, or 48 x 1024 bytes of random access memory. One byte is actually made up of 8 bits of information, but the important thing to remember is that it takes approximately one byte to store a single character. Thus, the Oric can store up to a little over 48,000 characters in its internal memory.

(However, it is worth noting that the company originally intended to release a cheaper 16K version as well as the larger system, and, indeed, got as far as sending out pre-release copies to magazine reviewers. Continuing problems with the chips bought by the company eventually delayed the launch beyond the publication date for this book, but we can nevertheless expect to see a 16K version shortly after you are able to read this passage. In the meantime, it has been reported that the company has ordered $4m worth of 64K chips from Texas Instruments, so speculation is growing over the possibility of a powerful 64K machine that would be technically as powerful as many of the current business microcomputers.)

When using the computer, the internal memory (comprising both RAM (random access memory that can be used to store programs, data, computations, etcetera) and ROM (read only memory that cannot be utilised by the user, and usually contains the language interpreter and other systems software) has to store both the program that is operating, the language interpreter that converts the program instructions into machine instructions, and any information you have to type in. Thus, smaller computers like the old 8K PETs, and the 1K ZX80s, are all capable of operating simple games programs, but are probably not large enough to operate the more sophisticated fantasy games like Dungeons and Dragons; and certainly not large enough for the majority of business uses. But since the first Oric produced has a fairly substantial memory size, certainly in terms of home computers, it is more than likely that

Oric Products International has further plans for the machines, even into basic business applications.

What can the Oric do?

Fundamentally, the Oric microcomputer can do anything that you can tell it to do. That is, it will obey any instruction or set of instructions that is correctly given. A set of instructions to a computer is usually called a computer program, and is written in a special language called a programming language. Like any other computer, the Oric executes programs and does what you tell it to do. Thus, one way to make use of the computer is to learn to program it in its own language, which is BASIC. Now although BASIC is the natural language of the Oric computer, it is not the natural language of the 6502 microprocessor. The BASIC program must therefore be translated into the language, or code (known as 'machine code'), that is understood by the microprocessor. This second level of translation happens automatically when you RUN a program, and you will be unaware of it when you are using the computer. Although it is possible to write programs directly in the Oric's machine code, this book will not go into the methods. Machine code programs are considerably more difficult to write than BASIC programs, even though they operate at a much faster speed. One reason for this faster operation is that there is no time lost in the translation between BASIC and machine code. Another reason is that no translation is ever as good as the original. Just as a word for word translation of Shakespeare into French can never be as good as either Shakespeare's English, or Molière's French, so a translation from BASIC to machine code can never be as efficient as a program written directly in machine code.

However, it is not essential to be an expert programmer to use the Oric computer since a growing number of ready-made programs can already be purchased. These programs either come on a cassette tape, from which they are transferred into the memory via an ordinary cassette recorder/player unit; or, at some time in the future, on small floppy disks that can be used with the forthcoming microdrives. Since many programs are already available, you may like to have a look at the hobby computer magazines for their advertisements. A much wider range will soon be available. Many commercial firms will be supplying programs specifically for the Oric, and some are already advertised in the press. Incidentally, since the Oric uses the same microprocessor as several other microcomputers, it is quite possible that some programs already written for these other systems may be easily rewritten for the Oric.

Programs that are purchased from a separate source usually arrive on an ordinary cassette. Figure 1.3, shows just such a cassette. To transfer a

Figure 1.3 A cassette tape.

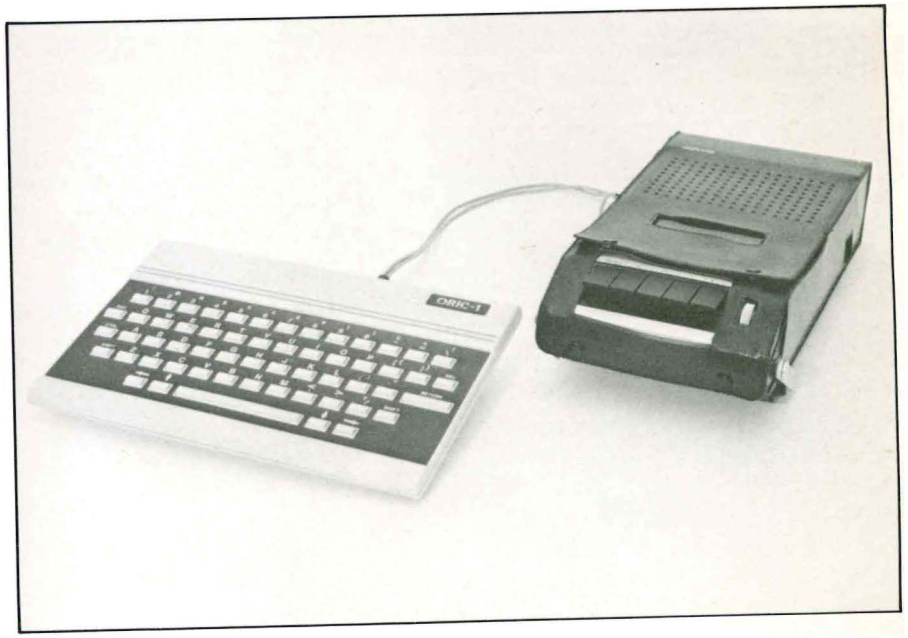


Figure 1.4 A cassette recorder attached to the Oric.

program from this to the Oric requires a cassette tape system (player/recorder). Almost any cassette recorder will do, but surprisingly, the cheap portable systems are better than the more expensive units. Figure 1.4 shows a cassette tape recorder attached to the Oric. The programs themselves are often referred to as 'software', in contrast to the computer itself, which is known as the 'hardware'.

The Oric computer can do many things. As with most microcomputers, you can often make it do these things without having any personal knowledge of programming. Nevertheless, it is often useful to be able to program, if only to amend or modify an existing program. Besides, programming is fun! It is easy to do, and it provides a means of expressing and communicating your own ideas to the computer so that it can test them for you. You will learn how to write simple programs later on in this book.

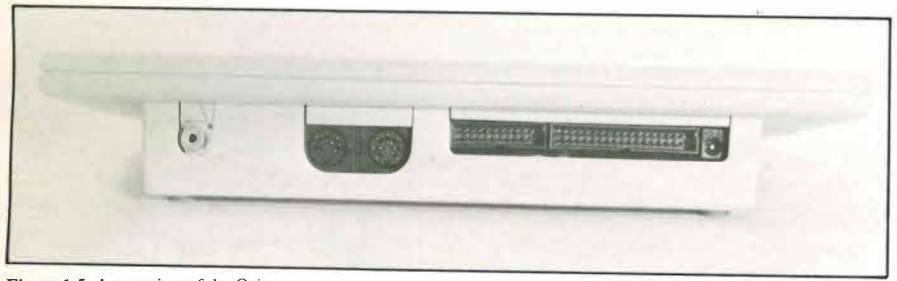


Figure 1.5 A rear view of the Oric.

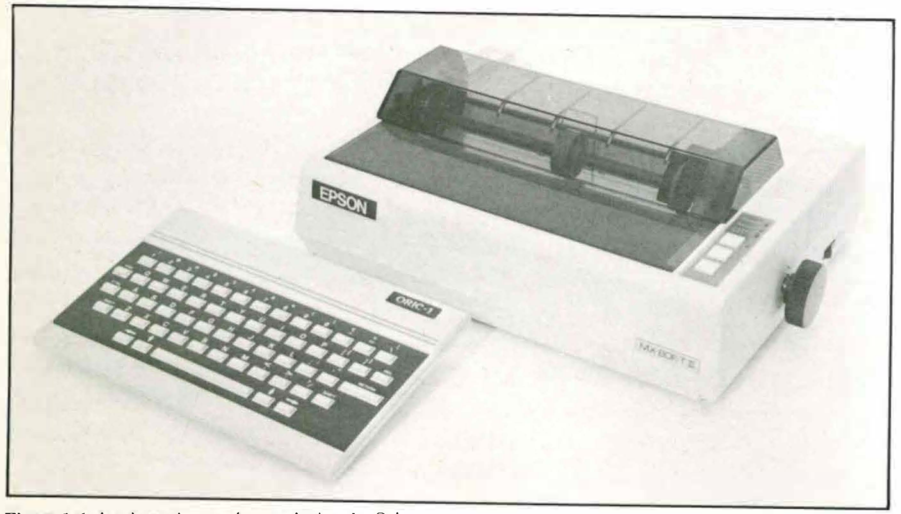


Figure 1.6 A printer that can be attached to the Oric.

How can the Oric be extended?

Besides performing computations and storing information the Oric can, again like most other computers, be used in conjunction with other devices. Units that can be connected to it, and controlled by it, are called 'peripherals'. You met one of them in the previous section: the cassette recorder. This is a peripheral device used for the permanent storage of information or programs by means of a magnetic pattern on the tape. Although the early Orics were not supplied with a cable to connect it and the recorder, we understand that in future Oric Products International will be supplying the cable. Figure 1.5 shows a rear view of the computer with the various peripheral connection sockets. One end of the cable connects to the right-hand (viewed from the rear) of the two DIN plug sockets, while the other end connects to the 'earphone' and 'microphone' sockets of the recorder.

For many computer applications it is useful to have the results of the computer's work in written form to provide a permanent record of the results of computations. It is also a useful aid for when you start to write your own programs, and particularly when you start to amend programs, to have a written copy. This printed output is called either a 'printout', or a 'listing'. A printout usually refers to the results of computations, while a listing refers specifically to a printed copy of a program. Obviously, a computer printer is an essential peripheral for obtaining printouts and listings. A printer such as that shown in Figure 1.6 can be attached to the Oric. If you wish to buy a printer, ask your supplier, or Oric Products International, for advice. The important technical details to remember are that the printer should be 'Centronics compatible', and that the connection is made via a parallel interface. Consult the manual supplied with the particular printer that you buy for details on its operation.

All peripherals are attached to the Oric using the connections on the rear of the computer. A 'port' is another name for this connection point between a computer and the outside world because it is similar to the way in which an airport or shipping port connects a country or region to its own outside world. From left to right (Figure 1.5), these ports are for connection to: a television set; an industry standard monitor (which is a form of visual display unit – VDU – designed specifically for use with computers, and therefore usually providing a better picture); a tape cassette player/recorder; the printer; and finally the bus expansion port (a bus is simply an interconnection channel via which any one of many devices may be connected to any one of many devices). This last port provides the means to connect, among others, extra memory, joysticks, program cartridges, and a modem. A modem is a device that enables the computer to attach to the public telephone network. It converts the digital signals of the computer into analogue signals for the telephone network; that is, it **MO**dulates and **DEM**odulates the signals. With this device you will be able to access the Prestel information service, and more particularly the new Micronet 800 service in order to receive programs automatically. You could also use it to send and receive electronic mail via the Prestel Mailbox system.

Next to the bus expansion port is a further socket which is not, perhaps, a port in the true sense; that is, the power input socket.

What are some typical applications of the Oric microcomputer?

The Oric computer was designed with many serious applications in mind. The areas in which it can be used can be broadly classified as: in the home for personal and recreational use; in educational institutions; and, to a certain extent, in business.

For personal use, there are games programs of many kinds already available, and more becoming available all the time. There is even one, called the Tower of Hanoi (a classic logic problem) at the end of this book, that you can type into the Oric and play. Using a computer for playing games is sometimes criticised as a frivolous use of an advanced electronic device, and there is no doubt that many games are lighthearted. Nevertheless, they do serve a useful purpose for relaxation and entertainment. Furthermore, there are many imaginative and stimulating games that, like the Tower of Hanoi, have a definite educational value. Other games can teach or help to develop attributes ranging from simple physical manipulation and co-ordination skills, to the mental disciplines required to find solutions to both puzzles and situations provided by the computer. There are, for example, many chess-playing programs of a formidable standard, and although they are (at the time of writing!) only available on other microcomputers, they will undoubtedly soon be available for the Oric. (In a short while you might even be writing your own games programs.)

The presence of an Oric in the home means that educational activities need not be restricted to schools and colleges. Computer assisted learning packages have been available for some time on other computers and are already being converted for use on the Oric. These can be used just as effectively at home as at school. Computer assisted learning is not, however, intended to replace teachers, but to assist them by providing another tool. In a post-industrial society (it is often claimed that, just as the country went through an industrial revolution during the last century, so it is now going through a technological and information

revolution), it is important to expose everyone, as early as possible, to the current technology. Only in this way will young people today be made aware of the possibilities presented by modern electronics, and be in a position to take advantage of the potential of computers. The presence of an Oric computer as an everyday item in the home or school can help achieve this objective.

In schools, microcomputers have many valid uses, ranging over computer assisted learning, instructional programs, and the use of quiz programs. In higher education, the Oric is ideal for activities such as Sixth Form and undergraduate programming projects.

Use of the Oric as a business aid is, at least for the time being, somewhat limited. Most, but by no means all, business applications require the speed capabilities of a floppy disk storage unit – and the Oric (at the time of writing this book) has no such peripheral. We understand, however, that disk drives will be available shortly. When all of these additions are available, the Oric will be capable of most business applications.

For any activity where large amounts of information have to be stored, and certain items have to be retrieved (for example, examining the stock-levels of individual items), it is almost essential to use a disk rather than a cassette unit for storage. This is not only because a disk has a greater storage capacity, but also because it permits an item of information to be got out of storage or 'accessed' much more rapidly. Any item stored on a disk can be accessed almost immediately regardless of its position. This is because the 'read head' on a disk unit can move over the surface of the disk to the required point, and for this reason, a disk unit is sometimes called a 'random access' or 'direct access' device. By contrast, using a cassette tape, it is necessary to wind the tape sequentially until the required point is reached. This, of course, can mean long and frustrating delays, and is most irritating when repeated access to information is required.

However, it is nevertheless possible to use the Oric, even without a disk unit, in a business environment in certain cases. These are primarily where the whole application is loaded into the computer once only every time it is used, and where no further access to the tape is required. A typical example of this type of application is the dynamic spreadsheet analysis program of the type that was first made famous by VisiCalc. Here, mathematical formulae and relationships can be defined directly on the screen and the computer will work out the results. The program is called dynamic because you can change parts immediately. At the time of writing this book, there is no such program on the market for the Oric, but there is little doubt that some enterprising programmer will very soon produce one!

Apart from this last example, all of these activities can be performed using existing programs. However, if you learn to program the Oric microcomputer, you can write your own programs. (It is worth remembering that a number of very successful commercial programs were first written because ordinary people at home, at school or in the office, were dissatisfied with the programs they could buy – so they wrote their own). Not only can you express your own ideas, but you can adapt programs that you buy to suit your own requirements more exactly. Since the Oric computer is much faster at numerical calculations than mere people, it would seem sensible to get the computer to do all your complicated calculations. Besides being used as a source of entertainment, and of educational and business convenience, the Oric computer can also be used, in this new Information Age, to extend and amplify the human brain.

Summary

The Oric computer is a small microcomputer which currently comes as a single 48K version. (16K and perhaps 64K versions may also be available by the time this book is printed.) The word 'micro' is used to describe the physical size of the machine, and particularly the processor inside, and NOT its ability to compute. However, the smallness of a microcomputer is what makes its computing power available for use as a personal tool at home, at school or in the office. This smallness is a direct result of recent technological developments stimulated mainly by the rivalry between the world's major nations.

The Oric can be used in many ways, but its main areas of application seem to be for personal entertainment and education, with a certain and lesser amount of business applications likely to develop. Computer programs can already be bought to perform many tasks in these areas. This allows the Oric to be usefully employed as soon as it is acquired without any expertise in programming being necessary. Clearly, as time progresses, more and more programs will become available.

The capabilities of the Oric can be extended in a variety of ways by acquiring further units or peripherals, such as a printer, which can then be attached to the computer and used in conjunction with it.

Chapter 2
Using the Oric

Switching on

The Oric must never be attached directly to the mains power supply. To turn it on, it should first be connected to the supplied power transformer unit, and this in turn plugged into the mains in the usual manner. There is no ON/OFF switch on the Oric.

However, before you can do anything meaningful with the computer, it must also be attached to a display screen so that you can see what you are doing. This may be either a domestic television set or an industry monitor as described in Chapter 1. Most people will use their own home television set. Although it is preferable to use a colour set, in order to gain the full benefit of the colour capabilities of the Oric, you can also use one of the more modern black and white sets. A portable black and white set has the added advantage of being easily moveable to any location in the building without being excessively expensive (about £60, for example). And although you lose the use of colour with a monochrome set, many people think that the definition of the subsequent black and white image is better than that on a colour screen. A connection cable is provided with the Oric. It fits into the socket on the left-hand side of the rear panel of the computer, and the ordinary aerial socket on the back of the television set.

Once the connections have been correctly made, signals produced by the Oric can be seen as a picture on the television screen. To begin with you may get a weak picture of the channel last viewed on the set; that is, BBC 1 or ITV. You must now tune the set into the frequency used by the Oric. A picture will form when you tune around frequency 36. It will show a white background with black text. Both of these colours can be changed by using the INK (for the text colour) and PAPER (for the background colour) commands. More on these later.

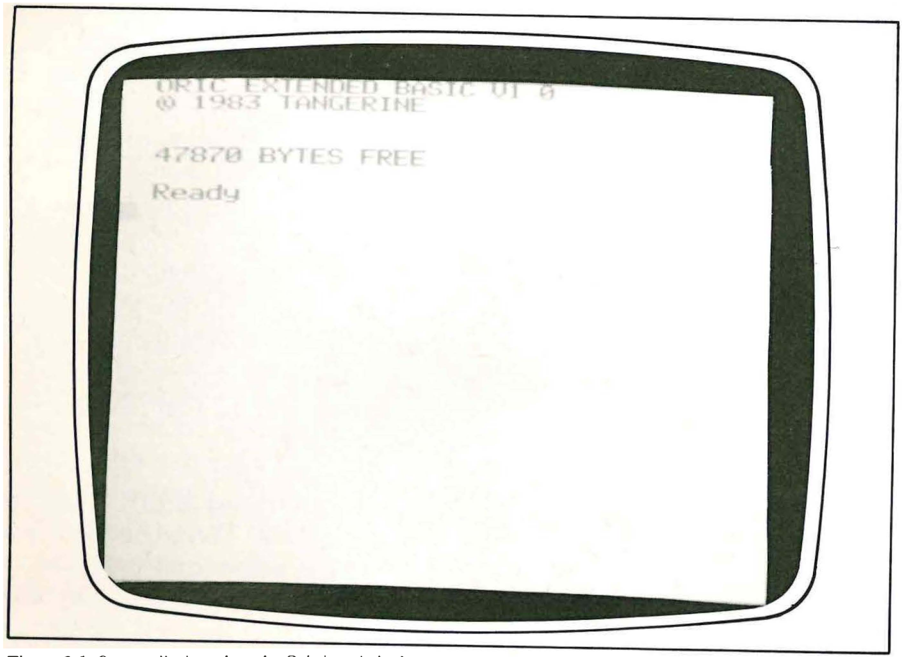
The initial display from the Oric is shown in Figure 2.1. The top line names the version of BASIC; that is, ORIC Extended BASIC, version 1.0. We will learn more about BASIC in Chapter 3. The second line is a copyright notice from Tangerine. Below this is a line informing you of how much computer memory is free, or available, for use – in this instance, 47,870 bytes. The amount will vary depending on the size of the machine you are using, although at the time of writing only the 48K version has been released. Finally, the system's 'Ready' prompt is given. This prompt is there to show you that the computer is ready for another command. It will appear at the left-hand margin every time that the Oric has finished a task.

Below this is a flashing block, known as the cursor. While the prompt will always show against the left-hand margin, the cursor can appear anywhere on the screen. It is there to show you precisely where the next letter you type will appear on the screen. Test this by pressing the <SPACE BAR> a few times, and then the letter 'A'. You will see that the cursor moves away from the margin as you enter the spaces, and that an 'A' will appear at the point of the cursor when you press the 'A' key.

The screen

The letters, symbols and numbers that you can type from the keyboard are all called 'characters'. The central square of the display is wide enough to take 38 characters on a single line. Once the line is full up with 38 characters, the cursor automatically moves to the beginning of the next line.

The display area can hold 26 lines, each of 38 characters. A character can therefore be placed in any of 26x38 (=988) positions on the screen.



```
ORIC EXTENDED BASIC V1.0
© 1983 TANGERINE

47870 BYTES FREE

Ready
```

Figure 2.1 Screen display when the Oric is switched on.

(Although the screen actually has 40 columns, usually only 38 of these can be used.) When the bottom line of the screen is full, the screen contents automatically shift up by one line giving a new blank bottom line. This is called 'scrolling upwards'. The previous top line disappears off the top of the screen.

The keyboard

The keyboard is very much like any ordinary typewriter keyboard. It is known as a QWERTY keyboard because of the organisation of the top row of letter keys, reading from left to right.

The keys consist of letters, numbers and symbols. There is also a space bar which produces the spaces between words. Using the keyboard is just like using an ordinary typewriter. There are, however, a number of special features which this section will explain.

Notice to begin with that there is no <SHIFT LOCK> key. All the characters that you type in at the keyboard will be displayed on the screen in capitals. This is because the BASIC programming language only understands capital letters. By using the <CTRL> key, you can, however, make the Oric behave like an ordinary typewriter; that is, with lower case characters as the norm, and uppercase characters when shifted.

The <CTRL> key is located at the left of the middle row of letters. This is a special computer key. It is used in conjunction with other keys to produce a special effect other than a normal character. This is done by holding down the control key with one hand and typing a character with the other hand. A table of control characters, as they are called, is shown in Figure 2.2.

<CTRL/T>	Caps lock
<CTRL/P>	Printer
<CTRL/F>	Keyclick
<CTRL/D>	Auto double height
<CTRL/Q>	Cursor
<CTRL/S>	VDU
<CTRL/]>	Protected column (far left)
<CTRL/C>	Break from program
<CTRL/J>	Line feed
<CTRL/L>	Clear return
<CTRL/M>	Carriage return
<CTRL/N>	Clear row

Figure 2.2 Table of Oric control characters.

To use the Oric keyboard like a typewriter keyboard, press <CTRL/T> (this is the standard method of indicating that the CONTROL key and the named character key must be pressed simultaneously, and will be used throughout this book). After you have pressed <CTRL/T> the keyboard will produce lowercase characters normally, and uppercase keys when shifted. The 'T' stands for Typewriter mode. You will learn more about modes later. <CTRL/T> is a toggle switch. Press it once and you enter typewriter mode; press it again and you leave typewriter mode. Notice that when you leave typewriter mode a little message saying 'CAPS' appears in the top right-hand corner of the screen. Apart from when you first turn the computer on and are automatically in CAPS mode, this message will always tell you when you have moved from typewriter to CAPS mode.

Other useful toggle switches accessed with the CONTROL key include <CTRL/F>, which turns OFF or ON the little click you hear when you press a key; and <CTRL/Q>, which turns the flashing cursor ON or OFF. <CTRL/D> is supposed to produce double height characters, but on the Oric used for this book, it merely produces two lines of the normal height!

<CTRL/C> is a very important command. You can use it if you want to stop the computer while it is working. It makes the computer break out from the current program that is in operation. When you do this, you will find that the program stops operating immediately, and the screen displays something like the message:

```
 BREAK IN 8Ø
Ready
```

The number is the line of the program that was being executed at the moment you pressed <CTRL/C>. The Ready message tells you that the Oric is ready to accept another command.

Stopping a program may sound a strange thing to want to do, but in practice you will find that you need to use it quite frequently. It is particularly useful when you start writing and testing your own programs. Using <CTRL/C> on its own will not harm the program that is in the computer's memory; indeed you can restart the program from the precise point at which you broke out by typing:

```
CONT <RETURN>
```

If you type:

```
RUN <RETURN>
```

then the program will restart from the beginning.

Finally, note <CTRL/L>. This command clears the screen and places the cursor in the 'home' position. The home position is a term often used to describe the top left hand corner of the screen.

There are several other keys that also deserve attention. These are the <RETURN> key (situated to the right of the middle row of letters); the <DEL> key (situated to the right of the top row of letters); and the four <ARROW> keys (situated on either side of the <SPACE BAR>). We shall look at the <DEL> and <ARROW> keys more closely in the section on editing later on in this chapter.

The <RETURN> key is in many ways similar to the carriage-return key on a typewriter since it makes the cursor (the flashing block that tells you where you are) move down one line and to the left-hand margin. On the Oric, however, you also use it to tell the computer that you are satisfied with what you have typed and that you want the computer to do something with it. In other words, pressing the <RETURN> key enters what you have typed into the computer and requires the Oric to take some action as a result. For example, if you have just typed an instruction, pressing the <RETURN> key instructs the computer to obey that instruction.

Finally, there are some other features that are best associated with this section on the keyboard. First of all, note that the keyboard is designed with an automatic repeat function. This means that if you hold down any key on the keyboard for longer than approximately one second, then that key will repeat automatically.

Note also that on the underside of the Oric, approximately underneath the 'l' key is a RESET button. Unless you have an exceptionally tiny finger you will need a pen or pencil or other similar shaped implement in order to reach it. This is done on purpose. The RESET button is an emergency button to get you out of never ending loops. You will understand this better after you have read Chapter 3 on programming. However, a never ending loop may be caused if you incorrectly instruct the computer to do the same job over and over again, automatically and without stopping. The problem is that you can't give the computer any new instructions, like 'stop', until it finishes its current job: but you've already told it never to finish! The RESET button does not switch off the power, but simply stops the execution of the program.

However, we recommend that (wherever possible) you always use the alternative method that involves holding down the key marked 'CTRL' and pressing the 'C' key simultaneously, as described earlier. This is because on the one hand it seems a waste of time and effort turning the computer over to get at the RESET button when <CTRL/C> has a similar effect; and because on the other hand, and perhaps more importantly, the CONT command will **probably** not work after a RESET. The likely failure will be caused by the Oric attempting to continue from the line in which it stopped, and thereby failing to take into account the logic of earlier commands. In other words, trying to CONTinue after a RESET will probably cause an error in program logic.

Finally, approximately underneath the semi-colon key, is another tiny hole in which a very small brass screw is visible. This is used to fine tune the colour signals sent to the television set.

Loading a program

Probably the most enjoyable and painless way to become familiar with the Oric and its keyboard is to use them to play a game that requires responses from the keyboard. A growing number of games are already available for this.

To load any program – irrespective of its name, from a cassette unit to the Oric, connect the player/recorder to the computer and insert the cassette. The biggest criticism levelled against the early Oric computers was that the company seemed to make it difficult for new users to correctly attach a cassette player/recorder to the computer. To begin with, Oric Products International did not supply any connecting lead. Then, after much adverse criticism, including:

> It's disappointing that Oric don't supply a cassette lead with the computer. There's nothing more annoying than finding you can't just 'plug in and go', but that you've got to visit the hi-fi shop first to buy extra leads if you want to Save or Load your programs.
> *Which Micro & Software Review*, February 1983

Oric began to provide a lead. But for some reason the lead is a DIN to DIN lead, rather than the standard DIN (at the computer end) to jack plugs (at the recorder end). Let us assume, however, that you have either purchased, or Oric has started to provide, a standard connecting lead with three jack plugs at the recorder end. Insert the computer input lead into the earphone socket (usually labelled 'ear'), and the computer output lead into the microphone socket (usually labelled 'mic'). The third and smaller pin will fit into the socket labelled 'remote', but note that some cassette player/recorders do not have this socket. If your cassette unit does have this socket, it can be used by the Oric to control the unit 'remotely'. If you do not have a remote socket, it will not affect the loading and saving of programs other than that you will have to operate the cassette recorder manually.

Let us pretend that you have a cassette with a recording of all the programs that are used in this book. One of the programs is a game known as the Tower of Hanoi, which is called simply 'Tower' on the cassette. To load this program into the computer, first make sure all the connecting leads are correctly attached. Rewind the cassette to the beginning (you will have to remove the remote pin to do this), and then press the PLAY switch. At the keyboard, on the line immediately below 'Ready', type:

```
CLOAD "TOWER" <RETURN>
```

As soon as you press <RETURN>, two things will happen. The Oric will switch on the cassette unit automatically, and the message 'Searching' will appear at the top-left of the screen (see Figure 2.3). This means that the Oric is Searching along the tape for the program called TOWER. When the Oric finds the program named in the inverted commas, the message at the top of the screen changes to:

```
Loading . . Tower
```

As soon as the loading is complete, the loading message disappears from the top of the screen, and the Ready prompt appears by the cursor position. If your recorder has a remote function and the relevant connection is made to the Oric, the cassette motor should now stop automatically. If not, you must now press the STOP switch.

If you do not know the name of a particular program that is stored on the cassette, but simply wish to load the next program that is found, enter:

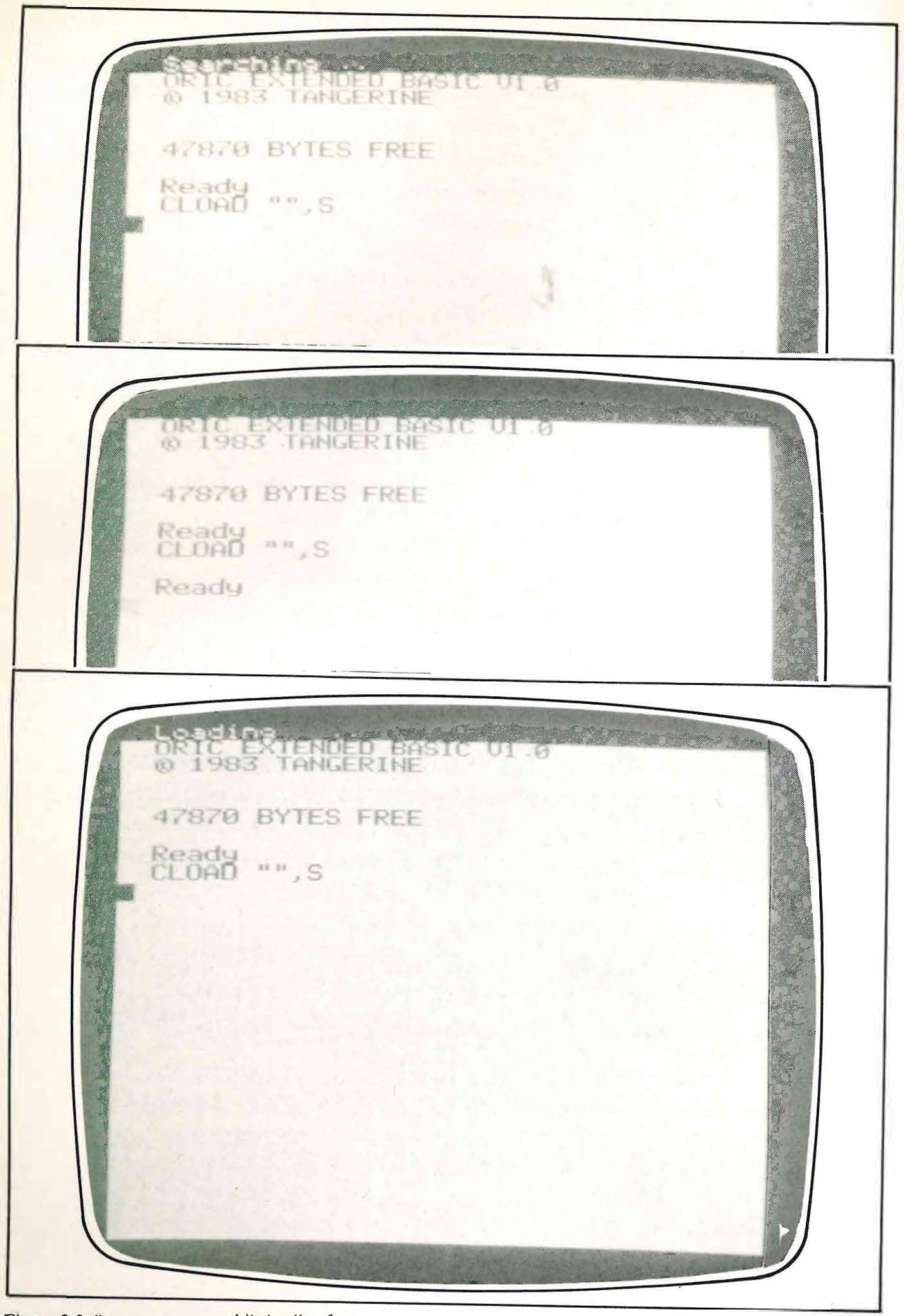
```
CLOAD "" <RETURN>
```

The Oric is able to send information to, and read information from, the cassette at two different speeds. The slower speed is available for greater security during transmission, and is less likely to produce errors. For this reason we recommend that you use the slow speed. The process is identical, but with the addition of ',S', thus:

```
CLOAD "FILENAME",S <RETURN>
```

Despite the security of the slower transmission speed, the Oric can be very sensitive to the volume setting on your cassette player/recorder. If the volume is set too high, or too low, you will be unable to load your program from the cassette. If the volume is very wrong, you will probably get a message like the following:

```
FILE ERROR / LOAD ABORTED
```



```
Searching ...
ORIC EXTENDED BASIC V1.0
© 1983 TANGERINE

47870 BYTES FREE

Ready
CLOAD "",S
```

```
ORIC EXTENDED BASIC V1.0
© 1983 TANGERINE

47870 BYTES FREE

Ready
CLOAD "",S

Ready
```

```
Loading ...
ORIC EXTENDED BASIC V1.0
© 1983 TANGERINE

47870 BYTES FREE

Ready
CLOAD "",S
```

Figure 2.3 Screen messages while loading from tape.

(Note, however, that the volume setting can be so very incorrect that the Oric will fail to register anything. If this happens, it will simply continue 'searching' without any hope of ever finding anything. If it happens to you, the only thing you can do is press the RESET button underneath the Oric and start again. In this instance, <CTRL/C> has no effect.)

If it is only a little way out, it may appear at first as if the loading has been successful. However, in these circumstances it is quite likely that one or two spurious characters or alterations to the text may have been introduced to the program during the loading procedure, but not enough to be registered by the Oric as errors during the loading. The first that you will know about these errors is when an appropriate Oric error message appears on the screen when you try to run the program, such as:

```
?SYNTAX ERROR IN 8Ø
Ready
```

This message tells you where the Oric has first detected an error in the logic of the program. There may be others that it hasn't yet found. You may then correct all such errors using the editing features that we shall look at in the next section, or you can try to load the whole program again, but correctly. The problem with these small errors is that they are enough to stop the program from running, but not enough to make it clear that the errors were introduced during the loading process rather than that they are inherent to the original program on the cassette tape. When this happens, particularly when you first start to use the Oric, it is easy to believe that your new program simply doesn't work; and this in itself can be very annoying if you've just spent something between £5 and £50 at the local computer store! We suggest, then, that the first program you try to load from cassette into your new Oric is one that you are certain is completely accurate on cassette.

Then adopt the following procedure:

1. Ensure that the cassette player/recorder and the television set are both correctly attached to the computer.
2. Ensure that the cassette player/recorder, the computer and the television set are all turned ON.
3. Wind the cassette to the beginning.
4. Set the volume control on the recorder to approximately mid-way.
5. Enter: CLOAD "" <RETURN>
6. Press the PLAY switch on the recorder.
7. If the loading is correct, mark the volume level on the recorder and try not to move it – ever!

8. If the loading failed, rewind the cassette, re-enter the load command, set the volume level on the recorder to a slightly lower level and try again.
9. Repeat this last stage, with the volume slightly lower each time, until the loading is successful. Then repeat stage 7.

Once the program has been loaded successfully, you can then run it merely by typing RUN, and entering the command by pressing the <RETURN> key. If you have loaded a games program, the program itself should now give you instructions on how to play the game. These instructions, or any set of messages between the computer and user, are called a 'dialogue'. Examples of this dialogue from the program called Tower are shown in Figures 2.4, and 2.5.

Using the program name in the loading instruction saves time because the computer can ignore other programs which appear before it on the tape. But notice that if you ask for a program that does not exist, or if you have typed its name wrongly, you may have to wait a long time while the computer searches fruitlessly. To avoid these long waits you may consider using the shorter C12 cassettes and having only one or two programs on each cassette. Alternatively you can keep a written record of the general location of the beginning of each program (if your cassette unit has a counter, you can keep an exact record). Finally, you can also detach all the connections from the recorder and use the cassette as a simple voice player/recorder unit to speak the name of each program just before it starts on the tape. This way you can use the fast forward and back facilities of the cassette unit to listen for the name of the program you want. When you have found the program you are looking for, reconnect the unit to the Oric, enter:

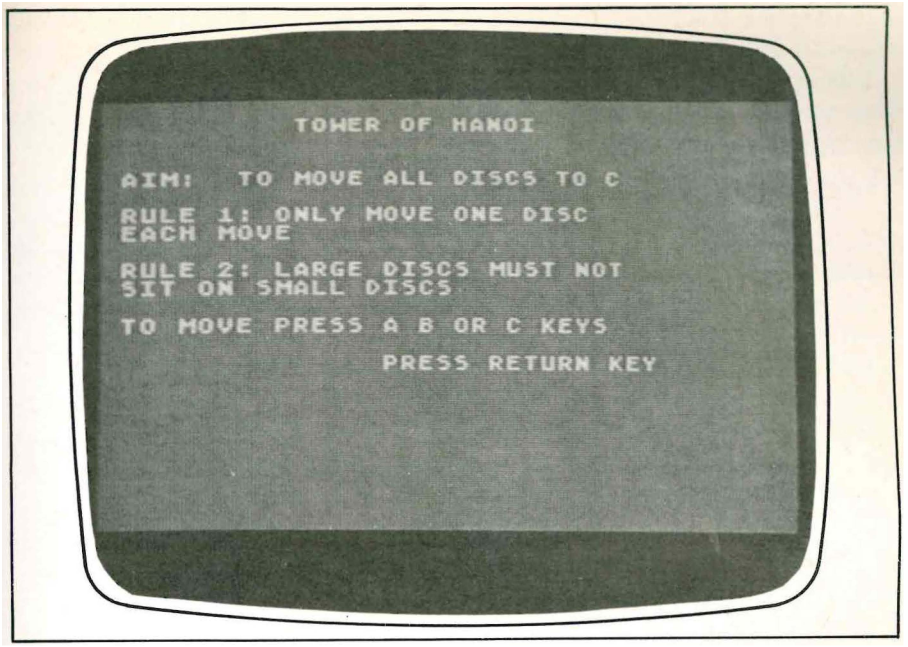
```
CLOAD "" <RETURN>
```

at the keyboard, and press the play switch on the cassette unit. The Oric will load the very next program it finds on the cassette.

As a general rule, it is useful to develop individual programs, particularly if they start to get quite long and elaborate, on individual tapes. When you have completely finished the program, you may then store it with several others on a single cassette.

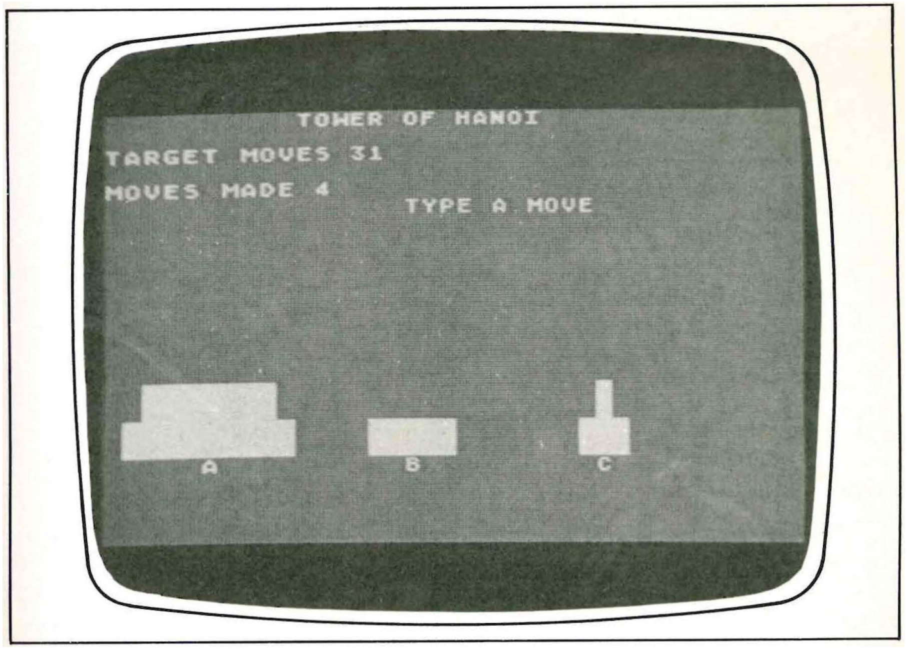
Editing

'Editing' is the name given to correcting or altering a piece of typing. The Oric has excellent facilities. If you realise that you have made a typing mistake immediately you have made it, the simplest way to put it right is by using the <DEL> key. Pressing it once makes the cursor go



```
            TOWER OF HANOI

AIM:   TO MOVE ALL DISCS TO C

RULE 1: ONLY MOVE ONE DISC
EACH MOVE

RULE 2: LARGE DISCS MUST NOT
SIT ON SMALL DISCS

TO MOVE PRESS A B OR C KEYS

                  PRESS RETURN KEY
```

Figure 2.4 Dialogue from the Tower of Hanoi program.



```
              TOWER OF HANOI
TARGET MOVES 31
MOVES MADE 4
                      TYPE A MOVE
```

Figure 2.5 Dialogue from the Tower of Hanoi program.

back a space, while at the same time deleting the character it moves over. You can then type the correct letter.

If you have already entered the line of typing with the mistake in it, you would normally have to re-enter the whole line. Let us say that you are typing a program with the following line:

```
11Ø PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Once you have typed <RETURN> at the end of the line, you will be unable to use the <DEL> key to correct the word 'MATTHER'. Nevertheless, you still need to remove the incorrect 'H'. You may decide to simply retype the whole line. To do this, re-enter the line correctly, making sure that you start the line with the correct line number. Use this method with care. Every time you start a line of text with a number, the BASIC interpreter inside the Oric will assume that it is a program line; and that if another with the same number already exists, the new line is to replace it completely. Thus if you enter:

```
1ØØ PRINT 'WHAT APPEARS TO BE THE MATTER?'
    <RETURN>
```

you will not have changed the incorrect line at 110, but will have changed whatever used to be line 100. Line 110 will remain exactly as it is.

Similarly, if you enter a line number without any text you will delete any existing text on that line (what you will be doing is instructing the Oric to replace that line with nothing – which is exactly what it will do!). Thus, if you enter '100', then realise that you really want to change line 110, and immediately type '<RETURN>' so that you can start typing line 110, what you are doing is deleting line 100.

It is a better idea to get used to using the Oric's Editor. An Editor is a small program or routine that will specifically enable you to edit text on the screen. (It is the very simplest and very earliest form of word processor.) To use the Editor, simply type 'EDIT' and the line number, thus:

```
EDIT 11Ø <RETURN>
```

The screen will print out the whole of the line as follows:

```
EDIT 11Ø
11Ø PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Before we go any further, it is a good idea to grasp the concepts behind the Oric's editor. Some people find it complicated, but if you can understand the basic principle behind it, the Oric editor becomes very simple and very logical. But first we must understand a new word: buffer. Buffers are used extensively by computers and are no more and no less than small and temporary storage areas for small amounts of data. The Oric editor has its own 'copy buffer' capable of holding up to 78 characters. (Incidentally, this is the maximum length of any line allowable by the Oric. If you try to type in more than this, the Oric will give a 'ping' warning on the 76th, 77th, and 78th characters, and will finally, on the 79th character, produce a back-slash and cancel the line. Try it. Since you cannot have a line of more than 78 characters, there is no need to have an editing copy buffer of more than 78 characters.)

When you use the editor, all you need to remember is that <CTRL/A> will copy the character currently under the cursor into the copy buffer. At the same time, <CTRL/A> will move the cursor forwards across the screen by one character. (<CTRL/A does not affect the characters it passes over in any way. They are still there to be copied again as often as you like until the copy buffer is full.) You may then copy the next character into the buffer, **or** you may ty e directly into the buffer. If you choose to type, then you will overtype the existing characters on the screen, and will not be able to copy them subsequently with <CTRL/A> because they won't be there to copy. The second point to remember is that the four <ARROW> keys will move the cursor to any position on the screen **without having any effect on the characters showing on the screen**. It thus follows that there is nothing to stop you using this feature to move the cursor to a completely different line, as long as it is visible on the screen, and copying text from an earlier line into the current line to save you typing it all out again. As you get used to this, you will find that it is particularly beneficial when you are typing in your own long and complicated programs. Finally, you tell the Oric that you do not wish to enter any more characters into the copy buffer by pressing <RETURN>.

Let us now try to use some of these features. Enter the incorrect line as shown above. Press <CTRL/L> to clear the screen and remove any confusing elements. Now type:

```
EDIT 11Ø <RETURN>
```

The screen should display the whole of the line as follows:

```
EDIT 11Ø
11Ø PRINT "WHAT APPEARS TO BE THE MATTHER?"
```

Now use <CTRL/A> to move the cursor to the required position over the incorrect 'H'. Do not press <CTRL/A>. Remember that every time that you do, and with every character you have passed over so far, you have taken a copy into the copy buffer. But now you do not want to copy the 'H'. Instead, use the <RIGHT ARROW> key to skip over the 'H'. After this you need only continue to press <CTRL/A> until you reach the end of the line, followed by <RETURN> to tell the Oric that you have finished copying into the copy buffer. The cursor will now move to the next line on the screen. Enter:

```
LIST 11Ø <RETURN>
```

and you will see the corrected line, thus:

```
11Ø PRINT "WHAT APPEARS TO BE THE MATTER?"
```

In other words, if you want to delete an unwanted letter or letters from a line that you have already typed, simply copy into the copy buffer all the correct characters with <CTRL/A>, and use the <ARROW> keys to skip over and leave out the errors. Let us say, however, that you now want to change the line:

```
11Ø PRINT "WHAT APPEARS TO BE THE MATTER?"
```

to

```
11Ø PRINT "WHATEVER APPEARS TO BE THE MATTER?"
```

This time we need to insert characters rather than merely delete the 'H' in 'MATTHER'. Proceed as above until the screen shows:

```
EDIT 11Ø
11Ø PRINT "WHAT APPEARS TO BE THE MATTER?"
```

Now press <CTRL/A> until the cursor is over the space between WHAT and APPEARS. If you press <CTRL/A> again, you will copy the space into the buffer when what you really want to do is enter the letters EVER. But if you simply type in EVER, you will remove the space and the letters APP from the screen; and once gone, those letters will no longer be available to copy with <CTRL/A>! You actually have a choice of two options, although each one actually does exactly the same as far as the Oric is concerned. Remember that the <ARROW> keys do not affect the text at all. The first option, then, is to move the cursor down, or up, but specifically to a blank area of the screen. At this point type in the letters EVER. Finally, move the cursor back up to line 110 and continue pressing <CTRL/A> from where you left off. When you have finished copying and entering into the buffer, clear the screen (it's not necessary, but it does prevent any confusion) and type:

```
LIST 11Ø <RETURN>
```

You should now see the full and correct line, thus:

```
11Ø PRINT "WHATEVER APPEARS TO BE THE MATTER?"
```

The second option has the same effect, but may appear more confusing until you are sure of what you are doing. This time, simply move the cursor backwards with the <LEFT ARROW> key for as many characters as you wish to insert. When you have done this, type in the new letters EVER. You will in fact remove the word WHAT from the screen, but since you have already copied this into the buffer, it doesn't matter. Continue to copy the rest of the line with <CTRL/A>.

Note that the EDIT command does not work correctly if the cursor is on the bottom line of the screen. You must either use <CTRL/L> to clear the screen and take the cursor to the top left-hand corner; or you must type in:

```
CLS <RETURN>
```

This has a similar effect except that since it is a BASIC command, the Ready prompt appears at the top of the screen to tell you that the Oric has obeyed your instruction and is ready for the next one. Alternatively, you could simply use the <ARROW> keys to move up the line on the screen that you wish to edit.

The Oric also has two other features that can be of great benefit during editing sessions. These are the LIST and program trace commands. We have already come across the LIST command. It allows you to look at all or parts of the program currently stored in memory. If you enter:

```
LIST <RETURN>
```

all of the current program will scroll up across the screen. However, if the program is more than 24 lines long, you will lose the early lines off the top of the screen before you have time to examine them. LIST also allows you to look at parts of the program by using the format:

```
LIST fromline-toline <RETURN>
```

Thus, the command:

```
LIST 1Ø–1ØØ <RETURN>
```

will display on the screen all and only the lines in and including the range of 10 to 100. This can be particularly useful not only when you are examining your programs, but also if you wish to create a new line from existing old lines by copying the relevant parts into the copy buffer.

The trace feature provides the ability to trace the logic of a program while it is being executed. The two commands are TRON and TROFF, used within the program to turn the trace ON and OFF. Let us say that you have a program that simply doesn't do what it is supposed to. By turning on the trace feature, you can make the Oric display on the screen the precise order in which it obeys the instructions of the program as it executes them. In this way you can see exactly where the error occurs and then make the necessary amendments. Enter the following program:

```
1Ø I = I + 1 <RETURN>
2Ø PRINT I <RETURN>
3Ø WAIT 2ØØ <RETURN>
4Ø GOTO 1Ø <RETURN>
```

Now type in:

```
RUN <RETURN>
```

and you will see the Oric start counting (see Figure 2.6). You will need to use <CTRL/C> to stop the program when you have seen enough. Now add the line:

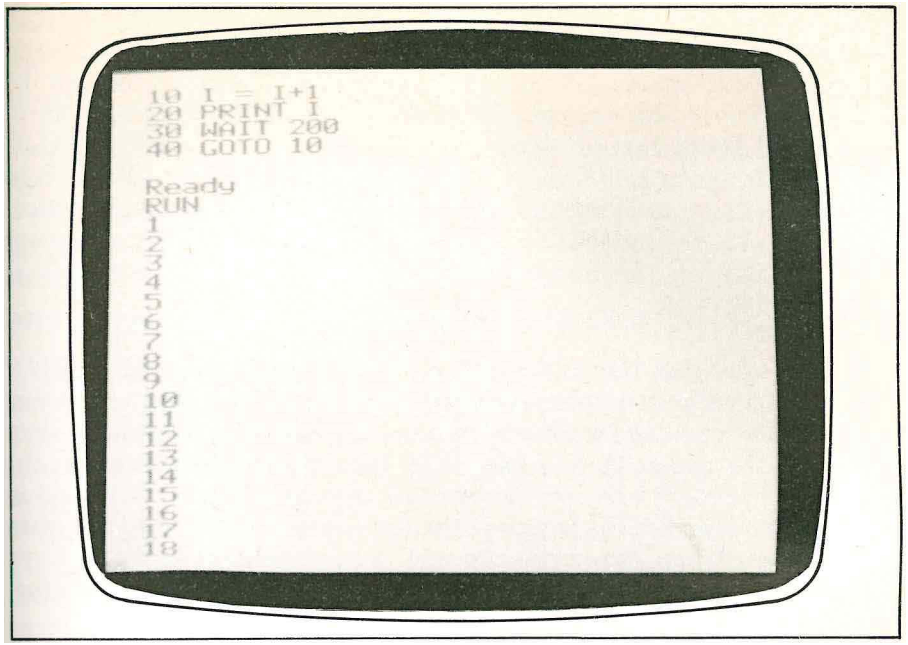
```
5 TRON <RETURN>
```

and run the program again. This time you will see the Oric showing you the sequence of line numbers it obeys (see Figure 2.7) as it executes them.

Giving simple instructions to the Oric

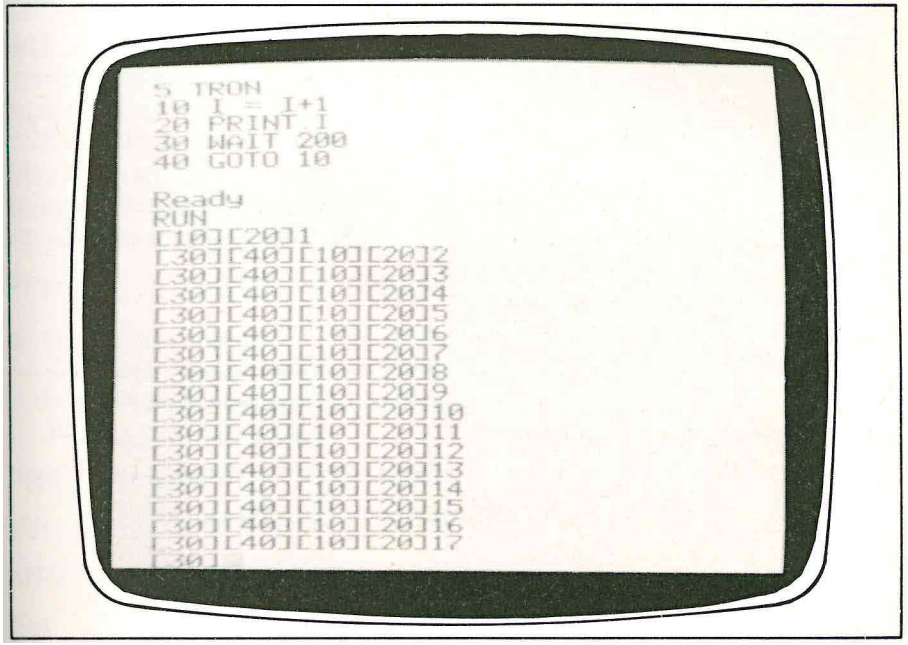
As we have already seen, a 'mode' is an operational condition under which a certain set of rules are obeyed. Most computers can operate in several different modes. Apart from the typewriter mode that we have talked about earlier, there are two other modes that you must understand. These are IMMEDIATE mode and DEFERRED mode. If you enter text into the Oric in a line starting with a number, the computer will store that line without acting on it. That is, if you start a line with a number, you are automatically using DEFERRED mode. In this case, the computer assumes that you have typed a line of a program that you will want to RUN later on in conjunction with many other lines.

If you do not use a number at the beginning of the line, the computer will automatically enter IMMEDIATE mode. In this case, the Oric will act on the instructions it receives immediately you press the <RETURN> key. In the rest of this chapter we will look at using the computer in IMMEDIATE mode. In Chapter 3, we will begin to look at using DEFERRED mode: that is, at programming.



```
10 I = I+1
20 PRINT I
30 WAIT 200
40 GOTO 10

Ready
RUN
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

Figure 2.6 The Counting program – without 'trace on' (TRON).



```
5 TRON
10 I = I+1
20 PRINT I
30 WAIT 200
40 GOTO 10

Ready
RUN
[10][20]1
[30][40][10][20]2
[30][40][10][20]3
[30][40][10][20]4
[30][40][10][20]5
[30][40][10][20]6
[30][40][10][20]7
[30][40][10][20]8
[30][40][10][20]9
[30][40][10][20]10
[30][40][10][20]11
[30][40][10][20]12
[30][40][10][20]13
[30][40][10][20]14
[30][40][10][20]15
[30][40][10][20]16
[30][40][10][20]17
[30]
```

Figure 2.7 The Counting program – with 'trace on' (TRON).

Generally speaking, what a computer does in most situations is to accept and store information of some kind, manipulate or process it, and then give out the results in some appropriate form. This can be illustrated by instructing the computer to do something with a set of characters. Such a set of characters is called a 'string'. In the BASIC language the dollar sign $ is used to tell the computer that you want it to treat something as a string. For example, the string of characters 'NICHOLAS AND JONATHAN' can be stored in the Oric's memory by typing and entering the following:

```
A$ = "NICHOLAS AND JONATHAN"
```

You will, of course, remember that you have to press the <RETURN> key to instruct the computer to obey the commands you give it. Pressing <RETURN> causes the Oric to execute this instruction, which it does by giving the name A$ to a part of its memory in which it stores the string of characters between the inverted commas. This is illustrated in Figure 2.8. This BASIC language instruction is equivalent to the plain English instruction: 'store the string of characters between the inverted commas in a part of the memory, and give it the name A$'. Note that spaces count as characters just as letters do.

When you press <RETURN> and the command is executed, there is no visible outward sign that anything has happened, other than the appearance of the Ready prompt at the beginning of the next line. This means that the Oric is ready to accept your next command. However, there is now an area of the computer memory that is storing the phrase 'NICHOLAS AND JONATHAN'. To demonstrate that the instruction has been correctly obeyed, you now need to know how to get at the information that has just been stored. You can print out the information stored in the part of the memory given the name A$ by typing and entering:

```
PRINT A$ <RETURN>
```

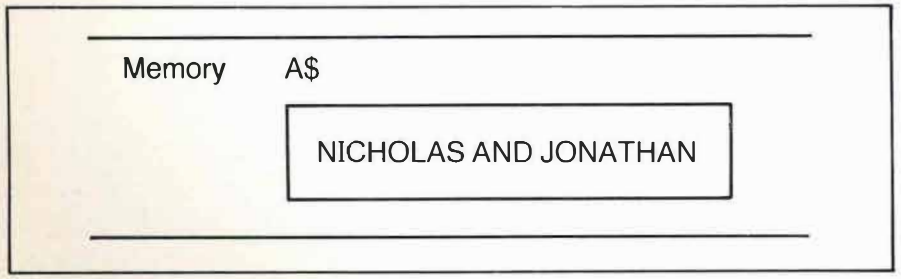


Figure 2.8 A string stored in memory.

In response, the computer will then print on the screen:

```
NICHOLAS AND JONATHAN
Ready
```

This BASIC instruction can be understood as 'print out what is stored in A$'.

You can instruct the Oric to find the number of characters stored in the area of memory called A$ with the LEN instruction, where LEN stands for 'length'. For example, suppose you now type:

```
PRINT LEN(A$)
```

When you press <RETURN>, the number '21' will appear on the screen. In this case the 21 characters consist of 19 letters and two spaces. This instruction can be interpreted as 'print out the length of the string stored in A$', or as 'print out the number of characters stored in the string A$'.

BASIC also has some features that enable you to manipulate strings. The LEFT$ command allows you to pick off a number of characters from the left of the string. RIGHT$ enables you to pick off a number of characters from the right of the string. For example, typing:

```
PRINT LEFT$(A$,8) <RETURN>
```

Will cause the computer to print the following eight characters from the left of the string:

```
NICHOLAS
```

That is, starting from the left edge of the string known as A$, the computer will display the first 8 characters.

Similarly, typing:

```
PRINT RIGHT$(A$,8) <RETURN>
```

gives the eight characters 'JONATHAN' from the right of the string.

This last instruction can be interpreted as 'print the eight characters at the right of the string of characters stored in A$'.

Finally, the command:

```
PRINT MID$(A$,10,3) <RETURN>
```

will produce the word:

```
AND
```

That is, it will start from the tenth character of the string known as A$ and pick off the next three characters.

You can see, then, that BASIC programming instructions are a sort of shorthand for instructions expressed in English. But because the Oric cannot think for itself, all of its instructions must be expressed in precisely the correct way. If there is even a slight error in the way that an instruction is written, no computer yet available will be able to recognise it. The Oric will, however, let you know that it doesn't understand what you are trying to do by printing an error message on the screen. Here are some examples, with their meanings:

?SYNTAX ERROR	(this is a syntax error, the most frequent of all errors. Usually caused by a typing or spelling error, or simply not knowing the correct format of the BASIC command)
?CAN'T CONTINUE ERROR	(this occurs if you try to continue a program that has ended, or does not exist)
?DIVISION BY ZERO ERROR	(this occurs if you attempt to divide a number by zero; which is, of course, impossible)

A full list of the error codes can be found in the BASIC programming manual supplied with the Oric.

We have seen how BASIC makes it easy to dissect strings. In the same way, it is also easy to build them up. To illustrate this you will need to store at least two character strings. Do this first by typing:

```
S$ = "SKY" <RETURN>
T$ = "TRAIN" <RETURN>
```

Now see what you build up from these strings. Try typing:

```
PRINT S$ + T$ <RETURN>
```

This instruction means 'print the string stored in S$ followed by the string stored in T$'. It produces:

```
SKYTRAIN
```

The following is a more complicated problem, but try to work out for yourself exactly what it tells the computer to do:

```
PRINT LEFT$(S$,2) + RIGHT$(T$,2) <RETURN>
```

The result is 'SKIN'.

Using the same principle, it is possible to get the Oric to produce a number of other words from the two strings stored as S$ and T$. Try

and make the computer produce the following: 'INKY', 'TRAY', 'STRAIN', 'STINKY'.

The Oric as a calculator

The Oric can be used as a calculator. If it seems like a rather expensive calculator, remember that this is only one, and perhaps the least useful, of ways in which it can be used.

As you will by now have realised, the number keys are situated on the top row of the keyboard. There are also five arithmetic keys for you to remember. The four main ones are:

+	means 'add to'
-	means 'subtract' or 'take away'
★	means 'multiplied by'
/	means 'divided by'

The fifth instruction is the more advanced arithmetic function of exponentiation. The symbol to use is '↑'. This function is used to raise a number to the power of a second number. Thus, 'squaring' 6 is the same as raising it to the power of 2, which is of course 6★6, or 36. The way to enter this is:

```
6 ↑ 2
```

If you wish to 'cube' the figure; that is, raise it to the power of 3 (6★6★6), use:

```
6 ↑ 3
```

For the time being, however, we shall concentrate on the four major functions. The symbol ★ is used for multiplication to avoid confusion with the letter X. The symbol / is used because there is no ÷ on the keyboard, and the alternative fraction display must be expressed on a single line.

Arithmetic calculations can be performed by instructions such as the following:

```
PRINT 2+3+4 <RETURN>
```

The answer 9 is immediately displayed on the next line. A more complicated calculation could be:

```
PRINT (2★3+4)/5 <RETURN>
```

The answer is 2. How you express the calculation instructions is very important, for the computer will always obey a certain sequence when it performs the arithmetic. This sequence is the generally accepted sequence taught in schools and should not, therefore, cause any problems.

The sum is performed basically from left to right; but multiplications and divisions are performed first. Thus:

```
1Ø−2★3
```

will first do 2★3, and then subtract the result from 10. The answer is 4 (and not, as you might otherwise think, 24). You can change this sequence if you wish to by using brackets. Brackets are always calculated first. Thus, if you type:

```
PRINT (1Ø−2) ★ 3 <RETURN>
```

the answer will now be 24.

Numbers can also be stored by using lines such as the following:

```
A = 3 <RETURN>
B = 4 <RETURN>
```

Notice that with numbers, you do not need to use the $ symbol for the location. What you have done, of course, is to store the value of 3 at a location called A, and the value of 4 at a location called B. The result of this operation on the contents of memory is represented in Figure 2.9. You can now perform calculations on the contents of these locations even if you do not know what the actual values are. For example:

```
PRINT A <RETURN>
```

gives:

```
3
```

Similarly:

```
PRINT A ★ B + 8 <RETURN>
```

gives:

```
2Ø
```

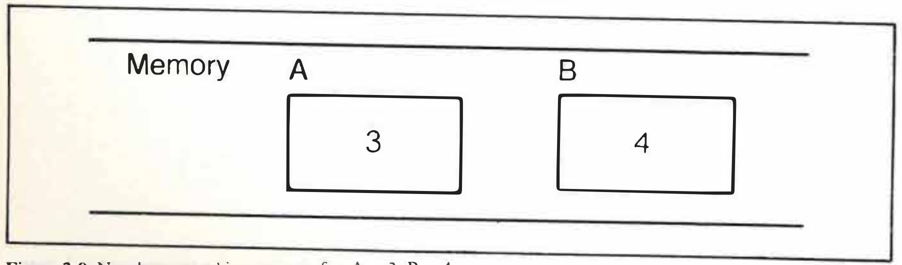


Figure 2.9 Numbers stored in memory after A = 3: B = 4.

Summary

A good way to become familiar with the Oric's keyboard is to run a games program that requires responses to be made from the keyboard. A program can be loaded into the computer's memory from a cassette by typing a CLOAD command. Once a program is loaded, it can be RUN. So, when starting to use the Oric computer, it is a good idea to practice with programs already recorded on cassette.

Most of the keys on the keyboard cause the corresponding symbol to appear on the screen when they are pressed, as one would expect. However, the <CTRL> key pressed in conjunction with a few other keys will produce different results.

Instructions can be given directly to the Oric in its own BASIC language. With the aid of a small repertoire of instructions it is able to make the computer perform such diverse activities as storing and manipulating words, and storing and performing calculations on numbers.

Self-test questions

1. What is the instruction which starts the procedure for loading a program into the Oric computer from a cassette?
2. How do you start the Editor so that you can edit a program line? Using as few instructions as possible, change the line:

   ```
   1ØØ THE ASCENT OF MAN
   ```
   to
   ```
   1ØØ THE ANCIENT OMEN
   ```
3. Make the computer store the words 'LEAD' and 'POSE' in its memory. Using these two stored words only, write the instructions that will cause the Oric to display:
   ```
   PLEAD
   POSE
   PLEASE
   LOSE
   ADDLE
   ```
4. Make the Oric store the numbers 4 and 5 in its memory. Using these stored numbers only, enter the instructions necessary to make the computer produce the results 16, 24 and 36.

Chapter 3

Introduction to programming

Writing and running a simple program

Towards the end of Chapter 2 we looked at using the Oric in IMMEDIATE mode; that is, we gave it single line instructions that were to be obeyed immediately. In this chapter we shall examine the alternative DEFERRED mode, which is another way of saying: 'programming'. In DEFERRED mode, all instructions must be on a line starting with a number. The computer will then store these commands until it is later told to RUN them. At this point it will obey each stored line of instructions in the order of the numbers at the beginning of the line.

A program, then, is a sequence of commands for the Oric to obey. The language in which the commands must be written in order that the Oric can respond to them is BASIC, and a few examples of individual BASIC commands have already been introduced in the previous chapter. BASIC is a simple programming language that was devised at Dartmouth College in the USA, and which first came into use in the early 1960s. It was intended to be easy to learn and easy to teach: and, indeed, its overwhelming popularity as a language for microcomputers stems from the fact that it is very easy to learn.

The Oric first deals with a program by storing it, so that it can then execute it when instructed to do so. When a program is stored in the Oric's memory it can be executed as many times as desired, or it can be modified prior to running it again. When a BASIC command is preceded by a number, the Oric treats that line as an instruction belonging to a BASIC program and stores both the number and the command. When the line is first entered, the command is not immediately executed, but is stored so that it can be executed later. An Oric program consists of a set of numbered commands. The numbers give the order in which the commands are to be executed when the program is run. The command or commands on the line with the lowest number are the ones to be executed first, and so on in ascending order. In fact, the instructions that make up a program can be entered in any order because the Oric uses the line numbers to put the instructions in the correct order.

To summarise this: a program line consists of a number followed by a command or commands; the Oric stores these instructions, and puts them in the correct order by using their line numbers; a program consists of a set of instructions. When a program is executed, each instruction is dealt with in sequence by executing its command part.

Now let us write a short program to store the three words 'THE', 'DOG', and 'SHOW', and to use these stored words to write out the phrases 'THE DOG SHOW', 'SHOW THE DOG' and 'DOG THE SHOW'. Before starting it is a good idea to type:

```
NEW <RETURN>
```

because this clears any program previously stored in the Oric. Then type in the following program exactly as shown, pressing the <RETURN> key at the end of each line to cause it to be stored in the computer's memory (note that in all of the example programs included in this book, the symbol '(SPC)' indicates that you must here type a single blank space – if more than one 'space' is required, the symbol is '(nSPC)'; for example, '(3SPC)'):

```
1Ø A$ = "THE(SPC)" <RETURN>
2Ø B$ = "DOG(SPC)" <RETURN>
3Ø C$ = "SHOW(SPC)" <RETURN>
4Ø CLS <RETURN>
5Ø PRINT A$ + B$ + C$ <RETURN>
6Ø PRINT C$ + A$ + B$ <RETURN>
7Ø PRINT B$ + A$ + C$ <RETURN>
```

In this program, the three words are stored at lines 10 to 30. Note that a space is included at the end of each word to act as a word separator when the phrases are printed. Line 40 causes the screen to be cleared when executed. CLS is the BASIC command to erase everything currently on the screen. It is a good general rule to get into the habit of using this command at the beginning of all of your programs.

Lines 50 to 70 cause the required phrases to be printed. Figure 3.1 shows the consequence of executing each instruction of the program. The result of executing the program is the cumulative effect produced by executing all of its instructions.

To demonstrate that the program has been stored by the Oric, type:

```
LIST <RETURN>
```

in response to which the Oric will always produce a 'listing' (a printout) of the program it is currently storing. Check the listing given on the Oric screen against the listing above to see that they agree exactly and, if they

do, execute the program by typing

```
RUN <RETURN>
```

You will then see the results of the program, looking like this:

```
THE DOG SHOW
SHOW THE DOG
DOG THE SHOW
Ready
```

Remember that because the program is stored in the Oric it can be executed or listed as often as you like. If you wish to list only a part, or just an individual line of the program, remember that you can type LIST and the line number, or 'LIST from-to'. If a line of the program has been entered incorrectly, it can be corrected by first listing it on the screen or using the EDIT command, and then using the normal editing procedures as described in Chapter 2. For example, if line 20 has been incorrectly entered, it can be listed by typing:

```
LIST 2Ø <RETURN>
```

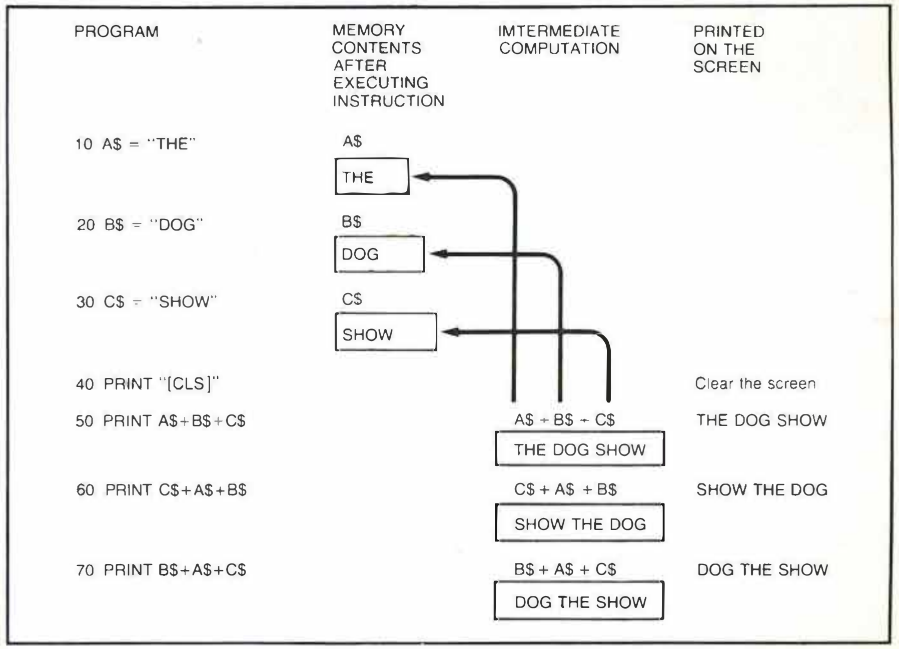


Figure 3.1 The result of running a program.

which could give, say,

```
2Ø B$ = "DIG"
```

Because this is a short line, it could be corrected most quickly by simply retyping the whole line. If it were a longer line, you should use the Oric's Editor as described in the previous chapter. The corrected program can then be RUN as described above.

To delete a complete line, all that is necessary is to type the number of the line to be removed followed by <RETURN>.

Try typing:

```
6Ø <RETURN>
```

and then list the program to see the effect it has had. When you have finished experimenting with the program, type:

```
NEW <RETURN>
```

to delete it. After typing this, the LIST command evokes no response. Try it.

Some more BASIC instructions

In this section we meet a few more BASIC instructions. They are incorporated in short programs to illustrate their usefulness. The fact that the <RETURN> key has to be pressed after a command or at the end of an instruction to cause the Oric to take the appropriate action will not be mentioned explicitly any more. So, as a last reminder, if you have typed something out and nothing appears to be happening as you sit and wait, it may well be that you have not pressed <RETURN>!

Input

Suppose that we should like to modify the program given in the first section of this chapter so that it accepts any three words we might care to give it when we run the program, and then prints out the first followed by the second and then the third; then the third followed by the first and the second; and finally the second followed by the first and the third. The new instruction that we need in order to make the program accept an input is INPUT. When an INPUT instruction is executed it causes a question mark to be printed on the screen to indicate that a response is required, and then stops the program execution and makes the Oric wait until the user types a response which it can accept. Thus, the instruction:

```
1Ø INPUT A$
```

produces the question mark on the screen. If you then type:

```
THE <RETURN>
```

the word THE is accepted and stored in A$. So, in this case, the effect is the same as that of the instruction

```
1Ø A$ = "THE"
```

The difference is that the latter will assign 'THE' to the variable A$, whereas the former can assign whatever you type, or 'input' to the computer, after the question mark.

An example program can be based on the previous program by replacing the lines that store the three words with three INPUT instructions – one for each word. When the words have been entered in this way, lines 50 to 70 will print out the phrases as before. Note, however, that when a word is entered with an INPUT command, although you can include spaces at the beginning and end of the word, it will be very difficult to see them. For this reason it would be better to include the spaces to separate the words in the PRINT commands. In this way, the following program for accepting any three words and printing three phrases involving them is obtained:

```
1Ø CLS
2Ø INPUT A$
3Ø INPUT B$
4Ø INPUT C$
5Ø PRINT A$ + "(SPC)" + B$ + "(SPC)" + C$
6Ø PRINT C$ + "(SPC)" + A$ + "(SPC)" + B$
7Ø PRINT B$ + "(SPC)" + A$ + "(SPC)" + C$
```

When this program is executed it could result in a dialogue like this:

```
? THE
? KIT
? BAG
THE KIT BAG
BAG THE KIT
KIT THE BAG
Ready
```

Decisions

The Oric can be programmed to make decisions. This ability can be used to produce some very interesting and powerful programs. The command which permits decision-making uses the BASIC words IF and THEN.

It has the form:

```
IF condition THEN command
```

In the condition part of this instruction, both variables and/or values can be compared, typically to see if they are the same or if they differ. The command part must be another BASIC command; for example, an assignment or a PRINT command. When the IF/THEN command is executed, the condition part is first tested. Only if it proves positive will the command part be executed. If the condition part does not hold, then the instruction part is ignored. An example of this type of command is:

```
IF N$ = "PASSWORD" THEN PRINT "ACCEPTED"
```

When this is executed, the Oric tests to see if the most recent assignment to N$ is PASSWORD: if it is, then ACCEPTED is printed out. If it is not, nothing is done. A second example is:

```
IF N$ <> "PASSWORD" THEN PRINT "REJECTED"
```

In this example the pair of symbols <> means 'not equal to'. (The < symbol on its own means 'less than', while the > symbol on its own means 'greater than'. If something is either less than or greater than the subject, then the only thing it is not, is 'equal to' the subject; that is, it is 'not equal to'.) Thus, when this command is executed, REJECTED is printed only if the most recent assignment to N$ is not PASSWORD.

Now consider a short program to create a sum, display it, accept an answer to it and decide if the answer is correct or not before printing an appropriate message. This requirement is also shown in Figure 3.2, which is an example of a flow chart. The program starts by storing two numbers in A and B, and then line 30 uses these numbers to print out a question about their sum. The question that is printed is WHAT IS 2+3? Using a semicolon to separate the items in a PRINT command gives a spacing different to that produced when a comma is used. Having set a problem, the program accepts an answer at line 40, storing it in C. Then in line 50 the offered answer is tested to see if it is equal to the right answer, and if it is, then an encouraging message is printed. The final line detects when a wrong answer is given and causes the correct answer to be printed out on these occasions. The program is:

```
 5 CLS
1Ø A = 2
2Ø B = 3
3Ø PRINT "WHAT IS(SPC)";A;"+";B;"?"
4Ø INPUT C
5Ø IF C = A + B THEN PRINT "GOOD. THAT IS CORRECT."
6Ø IF C <> A + B THEN PRINT "NO. THE ANSWER IS";A + B
```

This program can be adapted to give you more than one attempt to find the answer to the way illustrated in Figure 3.3 by using the GOTO command. The command:

```
GOTO 3Ø
```

instructs the program to go to line 30 and execute that line next. A program that expects the user to keep attempting to answer until the correct answer is given is obtained by altering line 50 so that it causes the last line of the program (line 80) to be executed next if the correct answer is given. The last line supplies the reinforcing message of encouragement. If the correct answer is not provided, then line 60 is executed. This indicates that the answer is wrong before line 70 causes a jump back to line 30, so that the question is posed again and a further opportunity to answer is given. This more sophisticated program is listed below:

```
 5 CLS
1Ø A = 2
```

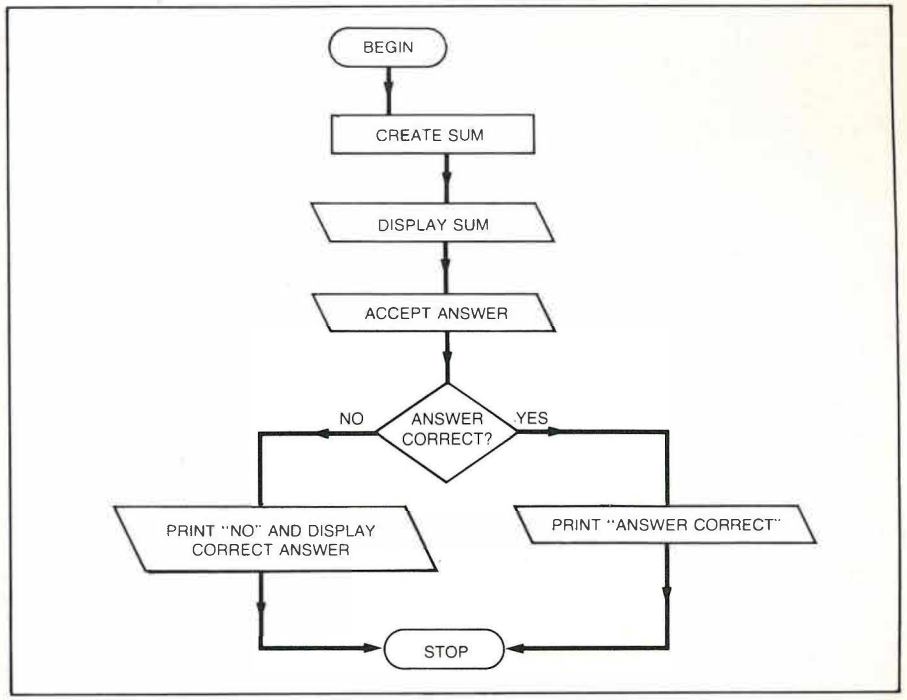


Figure 3.2 Flow chart for simple maths drill program.

```
2ϕ B = 3
3ϕ PRINT "WHAT IS(SPC)";A;"+(SPC)";B;"?"
4ϕ INPUT C
5ϕ IF C = A + B THEN GOTO 8ϕ
6ϕ PRINT "SORRY. WRONG ANSWER. TRY AGAIN."
7ϕ GOTO 3ϕ
8ϕ PRINT "GOOD, THAT IS CORRECT."
```

A typical dialogue produced by this program could be:

```
WHAT IS 2 + 3?
? 6
SORRY. WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3?
? 4
SORRY. WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3?
? 5
GOOD, THAT IS CORRECT.
Ready
```

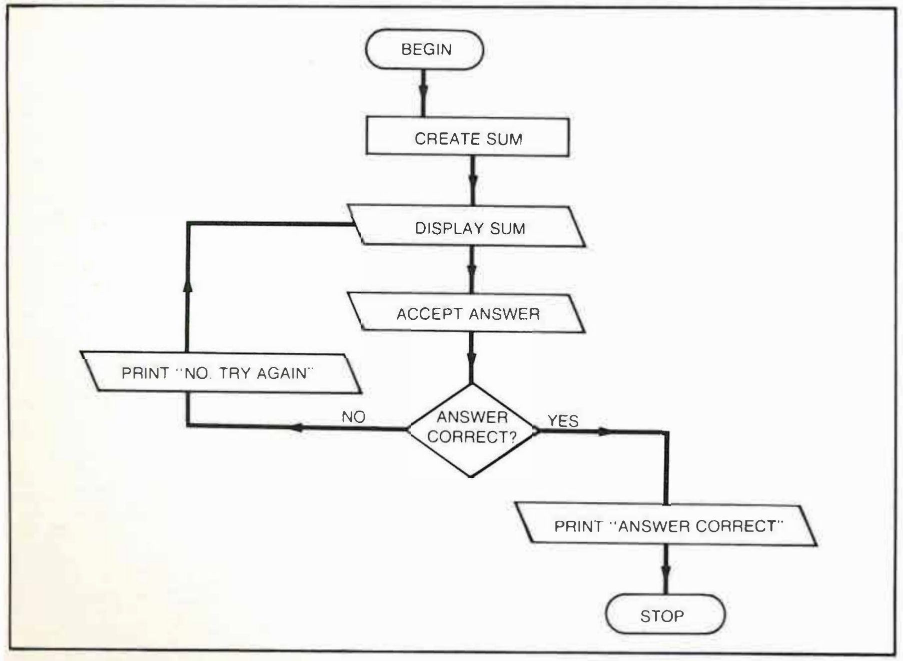


Figure 3.3 Flow chart for improved maths drill program.

Now try changing line 30 to:

```
3ϕ PRINT "WHAT IS(SPC)";A;"+(SPC)";B;
```

This time, you have not included a question mark to be printed by the instruction. Notice, however, that the semicolon at the end of the line 'pulls up' the question mark produced by the INPUT command. In this way you can eliminate the occurrence of double question marks and make the question and answer sequence more visually acceptable. A typical dialogue might now be:

```
WHAT IS 2 + 3 ? 6
SORRY, WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3 ? 4
SORRY, WRONG ANSWER. TRY AGAIN.
WHAT IS 2 + 3 ? 5
GOOD, THAT IS CORRECT.
Ready
```

Repetition

The previous program has shown that the Oric can be programmed to do things repeatedly by using the GOTO command. The mathematical problem is posed repeatedly until the correct answer is given. BASIC, however, has a more direct way to achieve this effect: the use of the BASIC FOR . . . NEXT command. To illustrate its use, enter the following program:

```
 5 CLS
1ϕ FOR I = 1 TO 16
2ϕ PRINT "JOANNE"
3ϕ NEXT I
```

This program will cause 'JOANNE' to be printed sixteen times, because all the instructions between FOR and NEXT are repeated as many times as directed by the FOR instruction. In this case, line 10 becomes a counter that goes up by one each time the command 'PRINT "JOANNE"' is executed. Notice that the program is going round in circles. This is called a 'loop'. Line 30 instructs the computer to loop back to line 10 until the counter matches the number specified in the 'TO' statement; that is, 16. At this point, the program will come out of the loop and go on to execute the next command, if there is one. The next program illustrates that you can put as many instructions as you want between the FOR and NEXT statements:

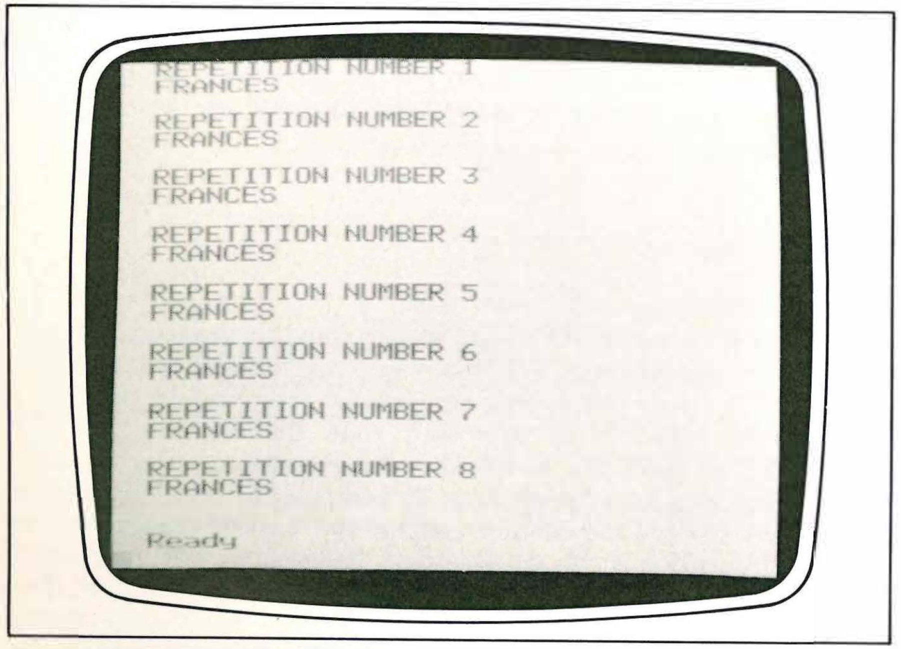
```
 5 CLS
1Ø FOR K = 1 TO 8
2Ø PRINT "REPETITION NUMBER(SPC)";K
3Ø PRINT "FRANCES"
4Ø PRINT
5Ø NEXT K
```

This causes eight repetitions and produces the output:

```
REPETITION NUMBER 1
FRANCES

REPETITION NUMBER 2
FRANCES
```

and continues up to the eighth repetition (see Figure 3.4).

Now try a program that accepts a word and spells out its letters one at a time. The program must accept the word, find its length and then repeatedly pick out and print the first letter, second letter, and so on, up to the last letter. We have already seen how to separate letters from the left or right of a word, using LEFT$, RIGHT$ and MID$, so you know all the commands necessary to write the program. It will start with



```
REPETITION NUMBER 1
FRANCES

REPETITION NUMBER 2
FRANCES

REPETITION NUMBER 3
FRANCES

REPETITION NUMBER 4
FRANCES

REPETITION NUMBER 5
FRANCES

REPETITION NUMBER 6
FRANCES

REPETITION NUMBER 7
FRANCES

REPETITION NUMBER 8
FRANCES

Ready
```

Figure 3.4 Display produced by Repetition program.

a polite request to enter a word, which will be followed by an INPUT command to accept and store the word in W$. At line 30 the number of letters in the entered word is found, and stored in L. Note that line 50 contains MID$(W$,I,1), and that I has already been assigned the variable loop counter of a FOR . . . NEXT command that is as long as the length of the word: in other words, I is at first letter one of the word, then letter two, etcetera. Now MID$(W$,3,1) will find the string of letters in the word stored in W$ which starts with the third letter and is one letter long, which means that it will find the third letter of the stored word. The effect of lines 40 to 60 is therefore to find repeatedly the successive letters of the word and to print out, on the first loop, that 'letter number I' is whatever the first letter is, and so on. The entire program for spelling out the letters that make up a word is:

```
 5 CLS
1Ø PRINT "ENTER A WORD, PLEASE."
2Ø INPUT W$
3Ø L = LEN(W$)
4Ø FOR I = 1 TO L
5Ø PRINT "LETTER NUMBER(SPC)";I;
   "(SPC)IS(SPC)"; MID$(W$,I,1)
6Ø NEXT I
```

More programs

Let us now write a program to accept any name written in the form:

```
JAMES JOYCE
```

and to produce the output:

```
YOUR FIRST NAME IS JAMES
YOUR SECOND NAME IS JOYCE
```

This may seem at first sight very easy, since after:

```
INPUT N$
```

the first name could be found by LEFT$(N$,5) and the surname by RIGHT$(N$,5). Unfortunately this will produce nonsense if the name entered is WILLIAM SHAKESPEARE (or any name that is not a combination of two five letter names). The trick is, of course, to locate the position of the space separating the two parts of the name. Then, assuming that the name has been entered correctly, everything to the left of the space is the first name and everything to the right is the surname. If the name is not entered in the way we expect strange results can still be printed, so it is sensible to ask that the name be entered in a standard fashion and then to check it, rejecting it if it does not conform. This reasoning leads us to the following program:

```
 1Ø CLS
 2Ø PRINT "ENTER YOUR NAME, PLEASE. TYPE"
 3Ø PRINT "YOUR FIRST NAME, THEN ONE"
 4Ø PRINT "SPACE THEN YOUR SECOND NAME."
 5Ø INPUT N$
 6Ø L = LEN(N$):C = Ø
 7Ø FOR I = 1 TO L
 8Ø IF MID$(N$,I,1) = "(SPC)" THEN C = C + 1
 9Ø NEXT 1
1ØØ IF C <> 1 THEN
    PRINT "PLEASE ENTER YOUR NAME AS REQUESTED"
11Ø IF C <> 1 THEN GOTO 2Ø
12Ø FOR J = 1 TO L
13Ø IF MID$(N$,J,1) = "(SPC)" THEN B = J
14Ø NEXT J
15Ø PRINT "YOUR FIRST NAME IS(SPC)"; LEFT$(N$,B−1)
16Ø PRINT "YOUR SURNAME IS(SPC)";RIGHT$(N$,L−B)
```

In this program the screen is first cleared, before lines 20 to 40 display on the screen the instructions for using the program. Line 50 accepts a name and stores it in N$. Line 60 contains two commands which are separated by a colon. The first command stores the number of characters in the name in L. The second sets a location C to zero, which will be used to count the number of spaces in the name. Notice that you can enter two separate commands on the same line if you separate them by a colon. Lines 70 to 90 scan each character of the name, counting the number of spaces. At the end of the repetitions the number of spaces in the name is held in C. If there is not exactly one space in the name, then lines 100 and 110 indicate that the entry is not satisfactory and cause a return to line 20 to permit the name to be entered again. Lines 120 to 140 locate the position of the space, storing it in B, so that line 150 can print all the characters to the left of the space as the first name and line 160 can print all those to the right as the surname.

Our next program produces a rather fascinating mobile display of a worm-like object that moves backwards and forwards across the screen! Although there are other ways in which this effect could be achieved, this method concentrates on the commands that we have already examined. It does, however, introduce one new command:

```
PLOT X,Y,"STRING"
```

To understand how this command works imagine the screen as a grid made up of 38 columns numbered 1 to 38, and 27 rows numbered 0 to 26 (see Figure 4.1). The total number of points on this grid is therefore 1026, each one of which is known to the Oric by a unique X,Y co-ordinate (the X refers to the column number, and the Y refers to the row number). The top left point is therefore 0,0, and the bottom right point is 38,26.

```
PLOT 1,1Ø,"STRING"
```

will make the computer print the characters in the string beginning in column 1 of the tenth row of the screen. (Avoid using column 0 for displays since it is reserved for the foreground colour.)

Notice in our worm program that we also introduce an extension to the FOR . . . NEXT command. At line 50, we have FOR . . . NEXT . . . STEP. If we do not include a STEP value, the computer will assume that we wish to use STEP +1. In other words, the FOR . . . NEXT counter will go up by one in each repetition loop. Here, however, the STEP is −1, and the counter will consequently decrease by 1 for each loop. The mobile display program is:

```
1Ø CLS
2Ø PRINT FOR I = 1 TO 3Ø
3Ø PLOT I,5,"(SPC)★★★★(SPC)"
4Ø NEXT I
5Ø FOR I = 3Ø TO 1 STEP −1
6Ø PLOT I,5,"(SPC)★★★★(SPC)"
7Ø NEXT I
8Ø GOTO 2Ø
```

Because the last line of the program always causes line 20 to be executed again, starting another pass across the screen and back by the worm, the program runs indefinitely when executed. To stop it, it is necessary to press the <CTRL/C> key.

We hope you will experiment with all the programs throughout this book, and make changes here and there to see what effect they have. On this last program, for example, try a new command: RND(1). This command will make the Oric produce a random number between 1 and the number you include in the brackets. By multiplying this random number with an upper limit value, and then extracting the integer part, you can create a random number up to a maximum value that you can specify yourself. If you need to make the range inclusive, you need only add 1 to the upper limit value. Thus, the formula to produce a random number is:

```
RND(1)★N+1
```

Change the program to the following and see what happens. Try to work out for yourself exactly what the program is now doing.

```
1Ø CLS
15 N = RND(1)★25
2Ø PRINT FOR I = 1 TO 3Ø
3Ø PLOT I,5,"(SPC)★★★★(SPC)"
4Ø NEXT I
5Ø FOR I = 3Ø TO 1 STEP −1
6Ø PLOT I,5,"(SPC)★★★★(SPC)"
7Ø NEXT I
75 CLS
8Ø GOTO 15
```

Run this program and see what happens. Now delete line 75 and see what happens. Try to work out the difference it makes.

We shall now develop a program to translate French words into their English equivalents. To do this, we shall need to store French words and the corresponding English words in such a way that they can be related to one another. BASIC provides the 'array' which is useful for doing this. The single BASIC command:

```
DIM A$(2Ø)
```

tells the Oric that 20 variables are to be established as an array, and that their names are to be A$(1) to A$(2Ø). Once established, these array variables can be used in the same way as ordinary variable names. For example, we can make the assignment

```
A$(6) = "MAN"
```

The DIM command also reserves storage space for all the variables in the array. As the following program shows, arrays can be used to great advantage in FOR . . . NEXT repetitions. In this program we shall use two arrays, as illustrated in Figure 3.5, with one (F$) holding French

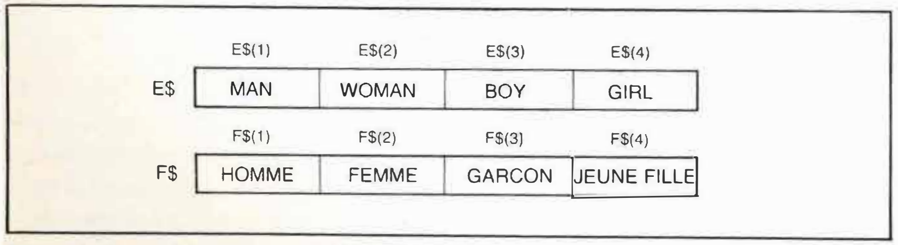


Figure 3.5 Two parallel arrays for translation program.

words and the other (E$) holding the equivalent English words in the same order.

The program translates by seeking to match the French word to be translated with one of the French words stored in the array F$. If a match is found then the word is associated with the English word in the corresponding position in the array E$. In the program line 10 reserves space for the two arrays which will hold the French and the English words. The words themselves are stored by lines 20 to 50. When the lines up to 50 have been executed, the state of the memory is as shown in Figure 3.5. Line 60 requests the entry of a French word and line 70 accepts and stores it. Line 80 then sets the variable T to zero: it will stay zero unless a match is found for the entered French word so that it can be translated. Lines 90 to 120 search the entire stored French vocabulary for a match to the entered word and, if a match is found, the corresponding English word is printed by line 100, and T is then changed to one by line 110. After these repetitions line 130 tests T: if T is still zero the word cannot be translated and an appropriate message is produced. Line 140 causes line 60 to be executed next, so that another French word can be entered, and the subsequent part of the program repeated. This is the program for translating French words into English:

```
 1Ø DIM E$(4), F$(4)
 2Ø E$(1) = "MAN" : F$(1) = "HOMME"
 3Ø E$(2) = "WOMAN" : F$(2) = "FEMME"
 4Ø E$(3) = "BOY" : F$(3) = "GARCON"
 5Ø E$(4) = "GIRL" : F$(4) = "JEUNE FILLE"
 6Ø PRINT "ENTER FRENCH WORD."
 7Ø INPUT W$
 8Ø T = Ø
 9Ø FOR I = 1 TO 4
1ØØ IF W$ = F$(I) THEN PRINT E$(I)
11Ø IF W$ = F$(I) THEN T = 1
12Ø NEXT I
13Ø IF T = Ø THEN
       PRINT W$;"(SPC)IS NOT IN MY VOCABULARY"
14Ø GOTO 6Ø
```

Clearly, as presented here, this program has a very limited vocabulary. It can, however, be extended in a very simple manner (by adding more lines of the type shown at lines 20 to 50 and making other straightforward adjustments). Also it is not difficult to adapt the program so that it translates from English to French. As we have already mentioned, all of the programs presented in this section are intended to be used as vehicles for experimenting with programming in BASIC. They can be amended, extended and improved in many ways.

Saving programs

When the Oric is switched off, the program stored in its internal memory is lost. To avoid having to type in the same program every time you want to use it, it is necessary to copy it on to some form of permanent storage. A program that is stored in the Oric can be permanently saved using either a cassette or a disk unit (if you have one) so that it can be loaded again later. We shall not discuss the concepts of saving on to disk in this book for beginners, and shall concentrate on saving a program on to a cassette tape.

To save the program stored in the Oric's internal memory on to a cassette tape, first ensure that the cassette unit is correctly attached to the Oric and then put a tape cassette (preferably blank) into it. Completely rewind the tape and then wind it forward a little to avoid trying to record on the tape leader. When the tape is positioned properly, decide on a name for your program, say 'Lexicon' (if you are not sure what lexicon means, look it up in a dictionary and remember the last program we typed into the Oric), and then type:

```
CSAVE "LEXICON"
```

Press the PLAY and RECORD switches on the cassette unit until they both lock on. Now type <RETURN> and a copy of the program will be copied on to and stored on the tape under the name of 'LEXICON'. The original program is still contained in the Oric's internal memory.

Note that if your cassette unit has the 'remote' connection, you can press the PLAY and RECORD buttons before typing the command:

```
CSAVE "LEXICON"<RECORD>
```

This time, the Oric will start the cassette motor automatically when you press the <RETURN> key, and will stop it when the program has been saved.

We have already seen how to load a program from tape into the computer's memory. On nearly all microcomputers, the level at which the volume is set on the player/recorder is critical in both saving and loading. If the level is incorrectly set (and to begin with, it probably will be!), the screen may display:

```
FILE ERROR / LOAD ABORTED
```

Refer back to the section dealing with loading programs for details on the best method of setting the volume level correctly.

If you wish the program to run automatically as soon as it is reloaded into the computer, save it on to tape with the command:

```
CSAVE "FILENAME",AUTO
```

A subsequent:

```
CLOAD "FILENAME"
```

will cause the program to run automatically as soon as it is loaded.

Finally, we recommend that you always use the ',S' speed setting for both saving and loading programs – it provides just that extra bit of security. Thus, you should get into the habit of using the two commands:

```
CSAVE "FILENAME",S
```

and

```
CLOAD "FILENAME",S
```

Using the printer

The addition of a printer to the Oric can enhance its usefulness considerably. Initially, the main uses for a printer are to provide program listings and to print results in a permanent and portable form. A program listed on paper is not only a convenient record but can also be taken away from the Oric and studied at leisure. If a program produces a lot of results, it is more convenient to print them out than to copy them from the screen: it is also much more reliable.

After the initial precautions of ensuring that the printer is attached to the Oric, switched on and loaded with paper, the program stored in the Oric can be listed on the printer rather than the screen by giving the simple command:

```
LLIST
```

This command works in precisely the same way as the LIST command; that is, you specify a portion of the program only by including the relevant lines:

```
LLIST 1Ø-1ØØ
```

If you want to send output from the program to the printer rather than the screen, use the command LPRINT instead of the usual PRINT.

To illustrate how the printer can be used from a program in this manner, let us modify the simple program from the beginning of this chapter. As well as giving the same output on the screen as it did before, it also produces identical output on the printer. Lines 10 to 70 comprise the first program of this chapter. Lines 80 to 100 cause the same phrases as are printed on the screen to be typed out on the printer. The program is:

```
 1Ø A$ = "THE(SPC)" <RETURN>
 2Ø B$ = "DOG(SPC)" <RETURN>
 3Ø C$ = "SHOW(SPC)" <RETURN>
 4Ø CLS <RETURN>
 5Ø PRINT A$ + B$ + C$ <RETURN>
 6Ø PRINT C$ + A$ + B$ <RETURN>
 7Ø PRINT B$ + A$ + C$ <RETURN>
 8Ø LPRINT A$ + B$ + C$ <RETURN>
 9Ø LPRINT C$ + A$ + B$ <RETURN>
1ØØ LPRINT B$ + A$ + C$ <RETURN>
```

Summary

The Oric can store a BASIC program which can then be executed as often as required, or which can be modified before it is run again. The Oric's BASIC language is a simple English-like language that provides, among other things, facilities for storing and manipulating information, making decisions and for repeating an action as often as necessary. In this chapter these facilities are introduced and incorporated in simple programs to illustrate ways in which they can be used. When a program has been written it can be saved on cassette or disk, and the way in which this is done is also described.

Self-test questions

1. What is the command to start the procedure for saving the program stored in the Oric on a cassette?
2. What are the BASIC words used for:
   (a) repetition
   (b) making a test and acting on the result, and
   (c) giving data to a program while it is running?
3. Write short programs for the following:
   (a) to print your name 10 times
   (b) to enter a word and decide if it has more than 7 characters. If it has more than 7 characters, indicate that a long word was entered, otherwise print that it was a short one;
   (c) to accept different words entered at the keyboard and then print them out without either their first or last letter.

4. Explain in the way illustrated in Figure 3.1 the computations performed when the following programs are executed:
   (a)
```
1Ø A$ = "ALGORITHMIC"
2Ø L = LEN (A$)
3Ø FOR I = 1 TO L
4Ø PRINT LEFT$(A$,I)
5Ø NEXT I
```
   (b)
```
1Ø A = 1: B = 1
2Ø PRINT A: PRINT B
3Ø FOR I = 1 TO 12
4Ø C = A + B
5Ø PRINT C
6Ø A = B: B = C
7Ø NEXT I
```
5. Write a program to accept a word, store it in A$ and then create in B$ the reverse of the word. This can be done by starting with a string of zero characters in B$, and then adding one character at a time from the right of A$. The program should print the reversed word and then decide if the original word is a palindrome; that is, if it reads the same forwards and backwards.
   A typical dialogue from the program might be:
```
ENTER A WORD, PLEASE.
?MADAM
THE REVERSE OF MADAM IS MADAM.
MADAM IS PALINDROMIC.
```

Chapter 4
Graphics

Introduction

In computer terminology, graphics are pictures. Many of the programs written for the Oric that will be of lasting interest and value will make good use of graphics. This will particularly apply to educational programs and computer games, where the interest and compulsion of the programs often lies in the attractiveness of the graphics. Business programs can also be made more effective if they present information and results in pictorial as well as numerical form.

Obviously, some numerical computation is necessary in any reasonably complex program whatever its application, but the results from it can be presented in one of three ways: by numbers, words or pictures. While it is necessary in some applications to have accurate numerical results, in many others the presentation of a screen full of numbers inevitably becomes rather dull. To present information using words is better than just numbers, but a television screen is designed to show pictures rather than words. Anyway, as everybody knows, a picture is worth a thousand words, and pictorial presentations are much more natural and informative than their alternatives. In this chapter we look at the Oric's graphics capabilities.

The Oric microcomputer has the ability to produce both graphics and colours on the screen. Furthermore, the graphics may be of either 'low resolution' or 'high resolution'. In this context, resolution refers to the size of the individual graphics elements that appear on the screen. Low resolution refers to graphics where the individual elements are fairly large and the consequent clarity and definition of pictures is relatively poor. This is usually achieved by using predefined characters that can be either alphabetic, or patterned blocks. High resolution graphics produce much smaller graphics elements, and the consequent clarity and definition of the pictures produced is that much higher. High resolution graphics is achieved by turning very small individual dots ON or OFF on the screen. It is the pattern produced by these individual picture elements, or pixels, that provides the picture.

For this reason most programs that include graphics as an effect rather than the main reason for the program tend to use mainly low resolution commands, and we would certainly recommend that you do the same at least until you become more accomplished at programming the Oric. (The Oric training guide includes some fascinating routines to show some of the effects of high resolution pattern production – and also some things that just don't work!)

On the face of it, it might seem that only a limited range of pictures could be generated within the low resolution format, but, as many existing programs have shown, and a number of the programs in this chapter will demonstrate, displays of surprising complexity can still be produced. Some patience and ingenuity may be required to produce them, but a little knowledge and some effort are really all that is needed to start. Many people find that investigating and using the graphics facilities of the Oric are among its most interesting aspects. The inclusion of good graphic effects has certainly been a major reason for the

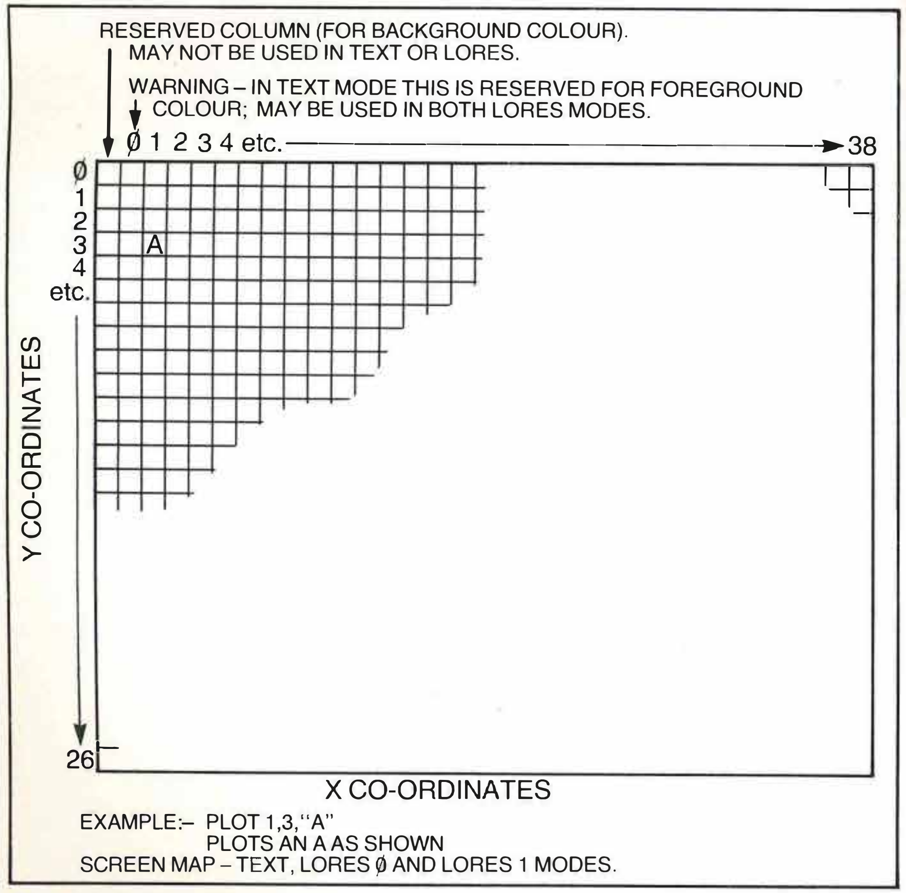

RESERVED COLUMN (FOR BACKGROUND COLOUR). MAY NOT BE USED IN TEXT OR LORES.
WARNING – IN TEXT MODE THIS IS RESERVED FOR FOREGROUND COLOUR; MAY BE USED IN BOTH LORES MODES.
Ø 1 2 3 4 etc.
38
Ø 1 2 3 4 etc.
A
26
Y CO-ORDINATES
X CO-ORDINATES
EXAMPLE:– PLOT 1,3,"A" PLOTS AN A AS SHOWN
SCREEN MAP – TEXT, LORES Ø AND LORES 1 MODES.

Figure 4.1 The low resolution screen grid.

success of many of the better programs written for microcomputers, and will furthermore help to ensure that new programs become a source of lasting pleasure and usefulness.

The screen and memory

A number of microcomputers use the facility of a POKE command to push special symbols into particular positions on the screen in order to build up pictures. This is NOT the method used by the Oric, and we mention it only because nearly everybody has heard of 'memory-mapped' screens and the POKE command.

In fact, the principle of graphics on the Oric is very similar. Imagine the screen as a grid of small squares made up of 39 columns by 27 rows for the low resolution screen (see Figure 4.1); and 240 columns by 200 rows for the high resolution screen (see Figure 4.2). Individual squares can be accessed by treating them as X,Y co-ordinates; that is, square 0,0 is the top left hand square in both resolutions, while square 38,26 is the bottom right square for low resolution, and square 239,199 is the bottom right for the high resolution screen. In all cases, the X co-ordinate corresponds to the column number, and the Y co-ordinate corresponds to the row number.

The total number of squares on the screen thus actually depends on the graphics resolution you use. If you are using low resolution, it is 1053, while it is as many as 48,000 in high resolution graphics.

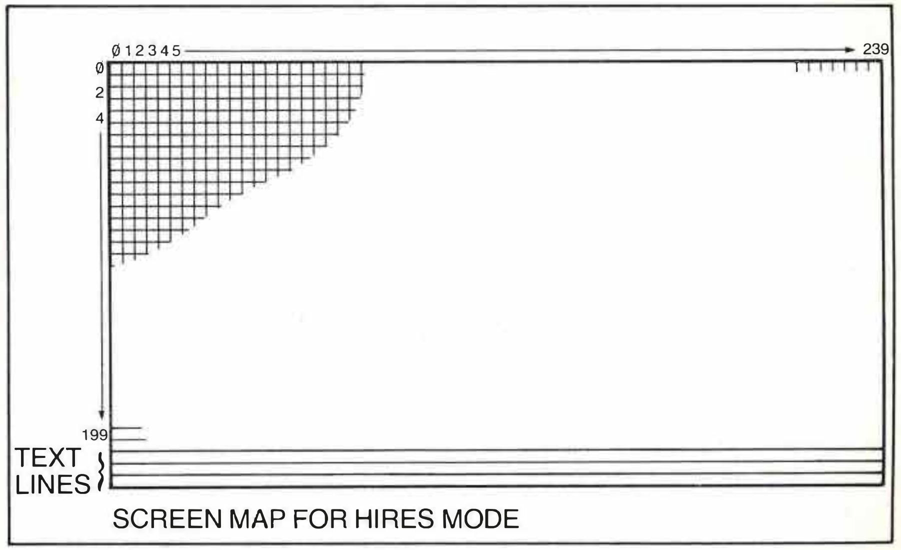


Figure 4.2 The high resolution screen grid.

Low resolution

There are three modes that can be used to produce low resolution graphics on the Oric screen:

```
TEXT
LORES Ø
LORES 1
```

TEXT is the default mode when you first witch on the Oric. The characters are those that you are now accustomed to produce from the keyboard. In this, as in the other modes, characters can be placed in specific positions on the screen with the command:

```
PLOT X,Y,"STRING"
```

The expression CHR$(n) can be used to tell the Oric which character you wish it to display. In TEXT mode, 'n' is a value corresponding to the ASCII code for that particular character. Try the following routine to show all the characters on the screen (we start at 32 – which is a space – because earlier character values produce commands that change the characteristics of the screen):

```
 5 CLS
1Ø FOR I = 32 TO 128
2Ø PRINT "CHR$ CODE(SPC)";I;"(SPC)=(SPC)";CHR$(I)
3Ø WAIT 5Ø
4Ø NEXT I
```

Now use the following routine to demonstrate PLOTting:

```
1Ø CLS
2Ø FOR I = 33 TO 47
3Ø FOR X = 15 TO 2Ø
4Ø FOR Y = 8 TO 13
5Ø PLOT X,Y,CHR$(I)
6Ø NEXT Y: NEXT X
7Ø WAIT 1ØØ
8Ø NEXT I
```

Now make the following changes to this last routine in order to look at LORES Ø and LORES 1 modes. First enter a new line:

```
 5 PRINT CHR$(17)
```

This line 'switches off' the flashing cursor so that it cannot interfere with or spoil the screen display. It has the same effect as typing <CTRL/Q><RETURN> at the keyboard. If you use it, you must remember either to enter PRINT CHR$(17) or <CTRL/Q><RETURN> in immediate mode when the program has finished in order to find where the cursor is currently located. Now change line 10 to read:

```
LORES Ø
```

The only difference that you will see is a change in the background colour of the screen. This mode is for entering ordinary characters as graphics. Now change line 10 to read:

```
LORES 1
```

and run the program again. This time you will see a completely different set of characters. These are the Oric's predefined low resolution graphics characters. Enter CLS <RETURN> to clear the screen at the end. The following routine displays all the alternate graphics characters on the screen at once (see Figure 4.3):

```
1Ø LORES 1
2Ø FOR I = 32 TO 128
3Ø PRINT CHR$(I);"(2SPC)";
4Ø NEXT I
```

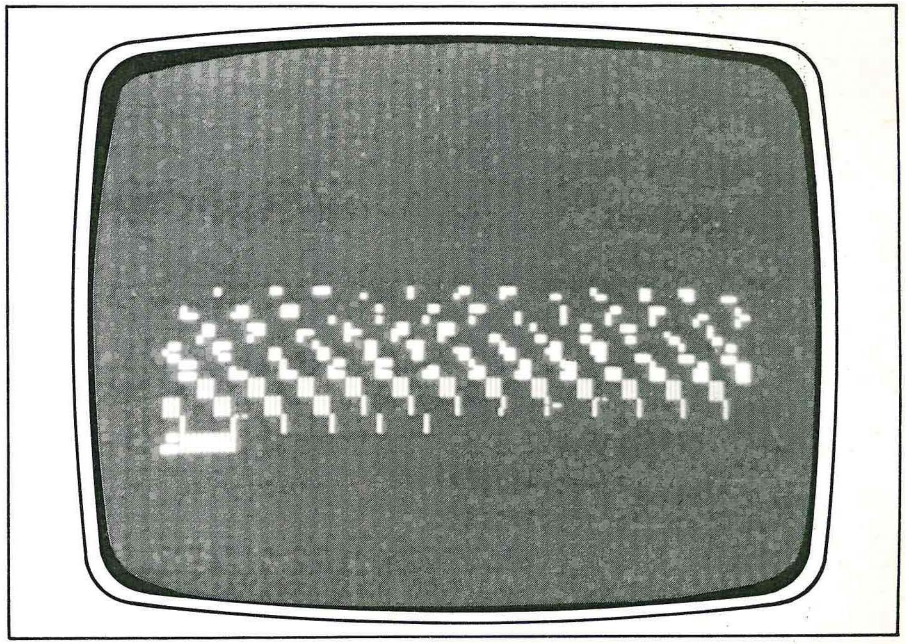


Figure 4.3 The alternate (graphics) character set.

If we now go back to our last program, we can see the effect of PLOTting all the graphics characters into our central square by changing line 20 to read:

```
20 FOR I = 33 TO 128
```

Also add:

```
90 CLS
```

to return the screen to the normal TEXT mode automatically at the end. The full program should now look like this:

```
 5 PRINT CHR$(17)
10 LORES 1
20 FOR I = 33 TO 128
30 FOR X = 15 TO 20
40 FOR Y = 8 TO 13
50 PLOT X,Y,CHR$(I)
60 NEXT Y: NEXT X
70 WAIT 100
80 NEXT I
90 CLS
```

Figure 4.4 shows a montage of some of the graphics squares produced by this program.

Finally, in this introduction to graphics, we can look at the colours available to the Oric. There are two commands to consider:

```
INK n
PAPER n
```

where 'n' is a value between zero and seven that corresponds to one of the colours in the table below:

0	BLACK	4	BLUE
1	RED	5	MAGENTA
2	GREEN	6	CYAN
3	YELLOW	7	WHITE

INK refers to the colour used to display the characters on the screen: PAPER refers to the background colour of the screen itself. The default setting when you first switch on the Oric is:

```
PAPER 7
INK 0
```

It is probably a good idea to include these settings as commands at the end of all your programs that change the colours on the screen, particularly where the changes are made in a random manner. In this way the screen will automatically revert to the standard settings at the end of the routine. The following short program shows all the colours available for the INK and PAPER commands. Note that when the two colours are identical, no characters can be perceived even though they are there:

```
 5 CLS
10 FOR I = 0 TO 7
```

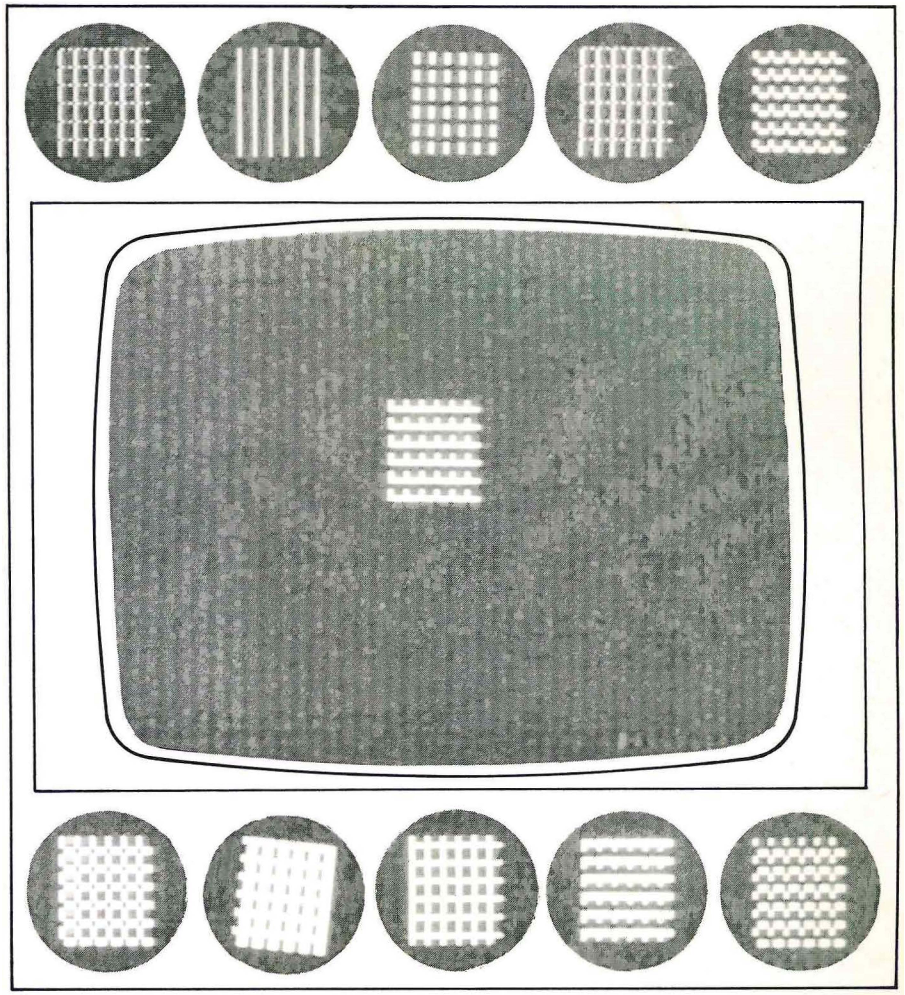


Figure 4.4 A montage of some of graphics 'squares.

```
20 FOR P = 0 TO 7
30 PAPER P: INK I
40 PLOT 1,12,"CHARACTER COLOURS ARE SHOWN"
50 PLOT 1,13,"IN FOREGROUND BY THE INK COMMAND"
60 WAIT 50
70 NEXT P: NEXT I
80 PAPER 7: INK 0
```

We now know enough to start producing patterns in low resolution graphics.

Screen patterns

As we have already seen, the screen can be filled with a particular symbol by a program such as:

```
  5 PRINT CHR$(17)
 10 LORES 0
 20 FOR X = 1 TO 38
 30 FOR Y = 0 TO 26
 40 PLOT X,Y,"*"
 50 NEXT Y,X
 60 GOTO 60
```

However, when the character at each screen position is generated by a systematic method, patterns that can be both informative and aesthetically pleasing can be produced. A general scheme that can be used to give a wide variety of interesting patterns involves the three stages of computation, classification, and representation. A value is first computed for each position on the screen using its row and column number. That value is then classified by assigning it to one of a number of predefined classes. Each class is represented by a particular character. In this way a character can be obtained for, and plotted in, each screen position. The process is, essentially, that used to make a coloured contour map where the height of each point is measured (computed), classified into the appropriate height interval, and then represented on the map by the colour assigned to that height interval. A general program scheme for generating screen patterns of this nature is given in Figure 4.5, and this can be refined to give a program such as the following:

```
 10 CLS
 15 PRINT CHR$(17)
 20 FOR R = 1 TO 26
 30 FOR C = 0 TO 38
 40 H = R*R + C*C
 50 IF H < 100 THEN I = 35
 60 IF H > 100 THEN I = 43
 70 IF H > 300 THEN I = 36
 80 IF H > 900 THEN I = 58
 90 PLOT C,R,CHR$(I)
100 NEXT C:NEXT R
110 GOTO 110
```

If you follow the program closely, you will see that it keeps very close to the program scheme in Figure 4.5. Line 20 is for each row, while line 30 is for each column. Line 40 computes a value, while lines 50 to 80 classify the values. Lines 90 then plots the relevant class code on the individual points of the screen.

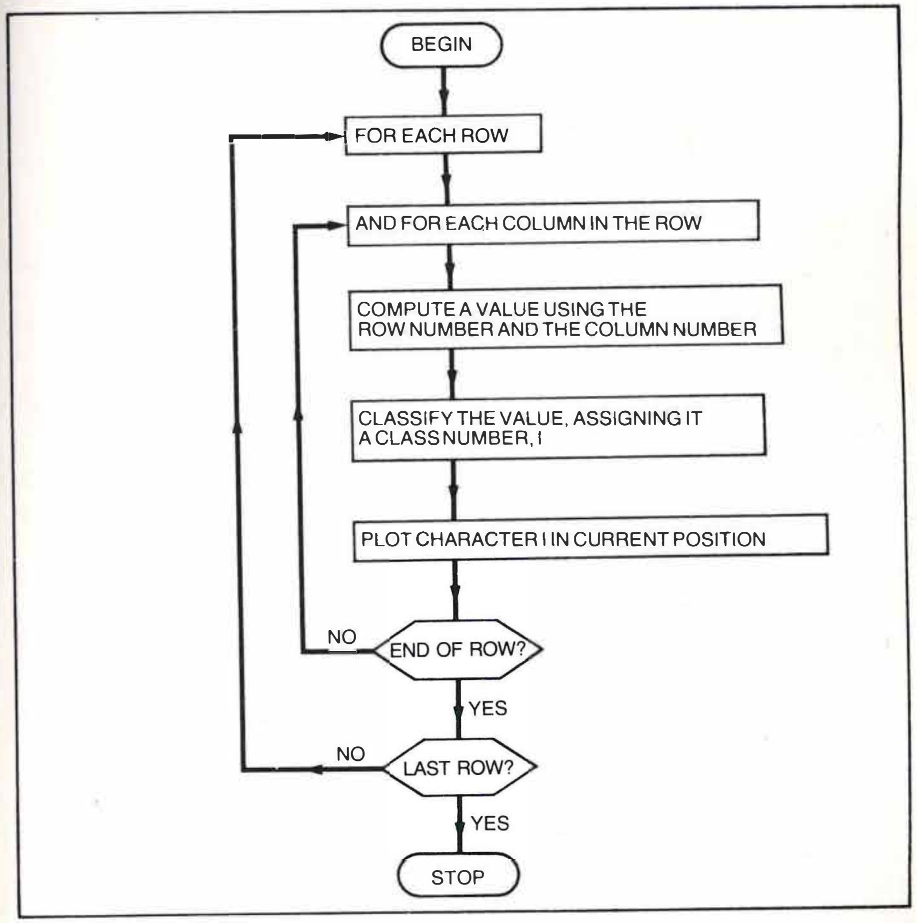

BEGIN
FOR EACH ROW
AND FOR EACH COLUMN IN THE ROW
COMPUTE A VALUE USING THE ROW NUMBER AND THE COLUMN NUMBER
CLASSIFY THE VALUE, ASSIGNING IT A CLASS NUMBER, I
PLOT CHARACTER I IN CURRENT POSITION
END OF ROW?
NO
YES
LAST ROW?
NO
YES
STOP

Figure 4.5 Program scheme for screen patterns.

Change line 10 to:

```
1Ø LORES 1
```

and run it again. Figures 4.6 and 4.7 show the screen results of the two versions. Remember to type <CTRL/Q> to switch the cursor back on when you have finished with the program. Line 110 simply prevents the program from terminating and spoiling the display with the Ready sign. Type <CTRL/C> when you have finished looking at the program.

A second program following the same pr gram scheme is:

```
 1Ø CLS
 2Ø PRINT CHR$(17)
 3Ø FOR I = 1 TO 1Ø
 4Ø READ A(I)
 5Ø NEXT I
 6Ø DATA 35,36,37,38,42,43,61,63,92,94
 7Ø FOR R = 1 TO 26
 8Ø FOR C = Ø TO 38
 9Ø H = (R★C) ↑ (1/3)
1ØØ H = INT(H)+1
11Ø PLOT C,R,CHR$(A(H))
12Ø NEXT C:NEXT R
13Ø GOTO 13Ø
```

Here there are ten intervals and plotting symbols. Line 90 computes the value (R ★ C raised to the power of 1/3; that is, the cube root of R ★ C), while line 100 classifies the value by finding its whole part (the INT command). To change the appearance of the pattern, try changing the values of the symbols being used. You could, for example, literally change all the values of the items in the DATA list, or you could simply change line 10 to:

```
1Ø LORES 1
```

Figures 4.8 and 4.9 show the program with line 10 as CLS and LORES 1 respectively.

A wide range of patterns can be produced by using this method. In general, a distinct pattern results from each choice of computation, classification and set of plotting symbols chosen to represent the classes. Classification can be achieved in many ways other than by dividing a range of values into intervals; for example the number in the first place after the decimal point in the computed value can be used to give the class number. The selection of plotting characters is vital to the presentation of effective patterns. The characters chosen for the last two

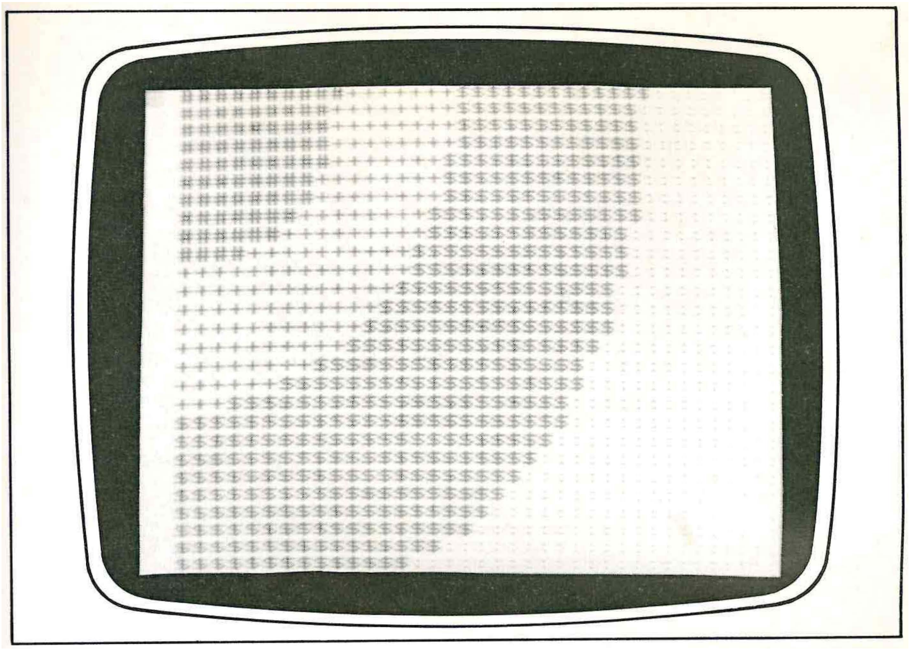


Figure 4.6 Screen pattern 1 (CLS).

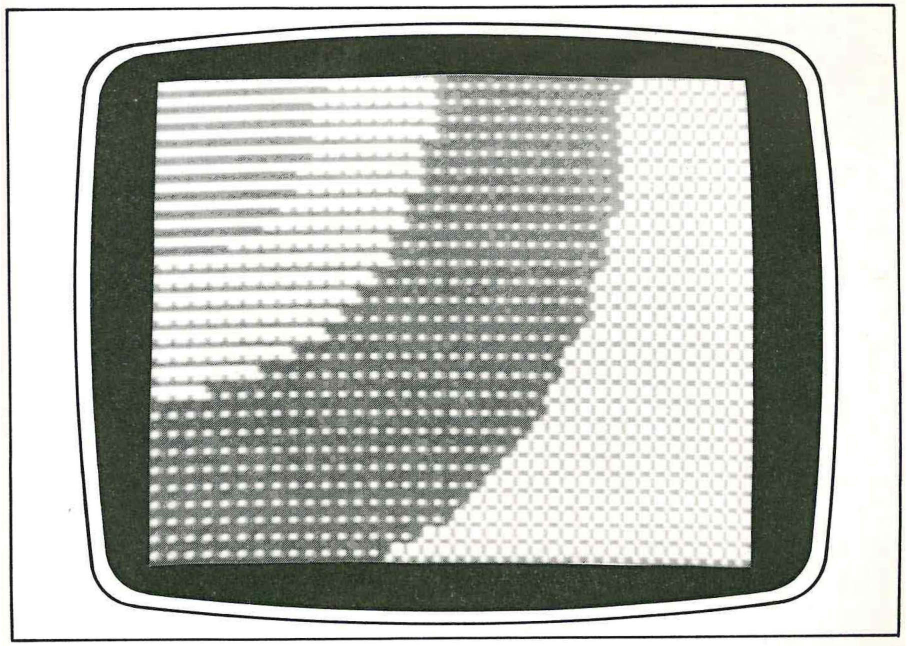


Figure 4.7 Screen pattern 1 (LORES1).

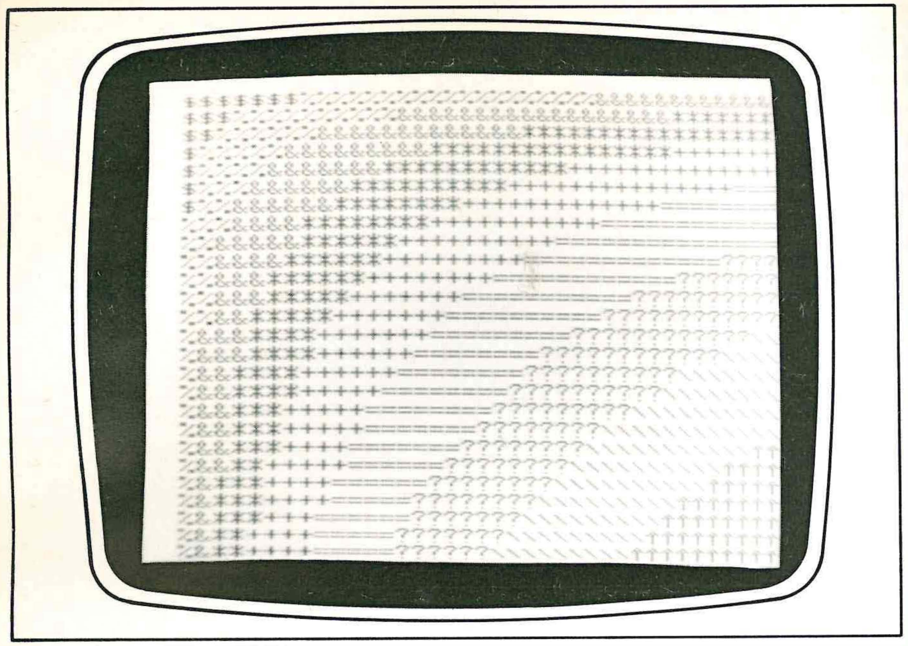


Figure 4.8 Screen pattern 2 (CLS).

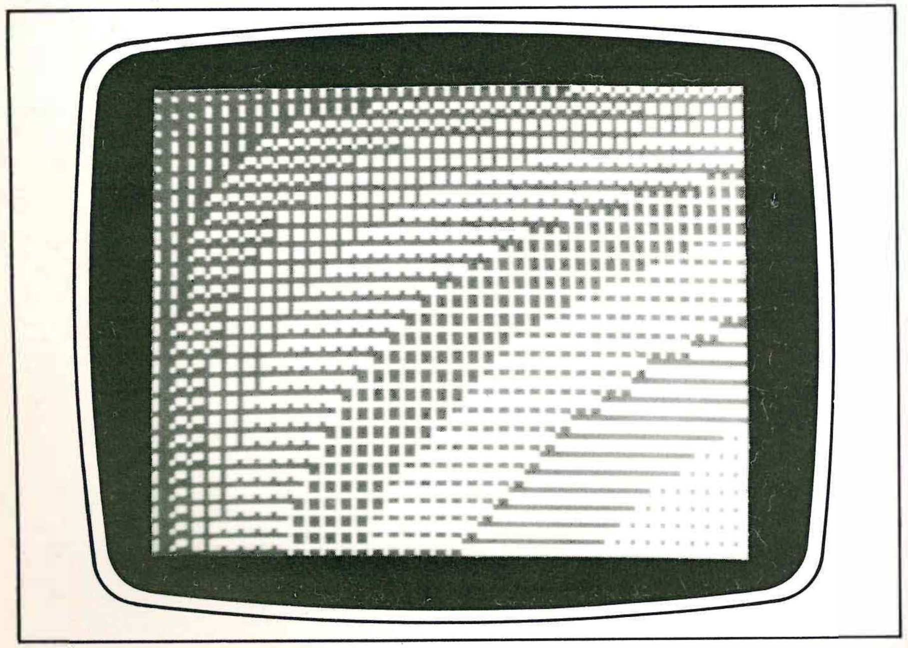


Figure 4.9 Screen pattern 2 (LORES1).

programs are intended to accentuate the transition from one class to another, but other characters may well prove more effective or attractive.

High resolution

By choosing and grouping graphics characters with care, not only patterns but also pictures can be drawn on the screen. However, in many ways, it is better to use high resolution graphics to produce pictures because the clarity is that much greater. With just two commands, you will be able to draw very complicated and detailed graphics pictures and patterns in the Oric's high resolution mode, HIRES. The two commands are:

```
CURSET X,Y,n
DRAW X,Y,n
```

In both cases, the 'n' is a value in the range of 0 to 3 corresponding to the table below:

0	background colour
1	foreground colour
2	invert colours
3	do nothing

Before we examine the two commands we must look at the concept of high resolution graphics. This is achieved by 'turning on' individual and very small points on the screen. These points are called 'pixels' because they are the smallest possible picture elements. In the Oric's high resolution mode, which is entered via the command HIRES (and exited via the command TEXT), there are 200 rows (or rasters) of 240 pixels each (see Figure 4.2). CURSET X,Y,n is the command that turns on individual pixels, where X is the column co-ordinate, and Y is the row co-ordinate. Pictures may thus be drawn by 'turning on' a particular pattern of pixels.

CURSET also places an invisible cursor at the X,Y co-ordinates. The second command will automatically draw a line from the CURSET position to that specified by the DRAW command. However, you must take note that the DRAW position is not an absolute X,Y position, but one that is relative to the original CURSET position. Run the following program to demonstrate HIRES, CURSET, and DRAW. When you have worked out what each line does, you will be well on the way to mastering high resolution graphics.

```
1Ø HIRES
2Ø FOR I = Ø TO 239 STEP 3
3Ø CURSET Ø,Ø,1
4Ø DRAW I,199,1
```

```
 5Ø NEXT I
 6Ø FOR I = Ø TO 239 STEP 3
 7Ø CURSET I,199,1
 8Ø DRAW 239−I,−199,1
 9Ø NEXT I
1ØØ GOTO 1ØØ
```

A second program that uses the same commands but in a random manner is as follows:

```
 1Ø HIRES
 2Ø S = 1
 3Ø X = 12Ø: Y = 1ØØ
 4Ø CURSET X,Y,3
 45 FOR N = 1 TO 75
 5Ø I = INT(RND(1)★2ØØ)★S
 6Ø J = INT(RND(1)★2ØØ)★S
 7Ø IF X + I < 2Ø OR X + I > 22Ø THEN 5Ø
 8Ø IF Y + J < 2Ø OR Y + J > 18Ø THEN 6Ø
 9Ø DRAW I,J,1
1ØØ X = X + I : Y = Y + J
11Ø S = −S
12Ø NEXT N
13Ø GOTO 13Ø
```

In this program, an initial cursor position is set by lines 30 and 40. In line 45 a loop counter is begun to prevent the screen from overfilling with too many lines. Then, in lines 50 and 60, random values for the DRAW command are generated. Lines 70 and 80 check to make sure that no DRAW commands will take the cursor outside of the screen area, and, incidentally, also explain the otherwise puzzling pauses when the program is run. Figure 4.10 shows a pattern produced by this program.

Producing a drawing

To demonstrate the production of a picture, try a short program that draws the infamous space invader. We will use a basic programming technique that you should try to understand and use as soon as possible. This is the use of the READ and DATA commands. The information required to produce the space invader is stored as data in lines 30 50. That data is then read into the program proper by the READ commands in lines 20 and 40. The DIM command at line 10 ensures that the program sets aside sufficient internal storage for the two DATA lines, and enables the program to treat them as arrays. The structure of the

program is therefore to store all the column co-ordinates in line 30, and all the row co-ordinates in line 50. These are then translated to the screen by line 70. Refer to any BASIC programming manual for a more detailed explanation of READ and DATA.

```
 5 HIRES
1Ø DIM X (4Ø),Y(4Ø)
2Ø FOR I = 1 TO 31 : READ X(I) : NEXT I
3Ø  DATA  3,4,5,2,3,4,5,6,1,2,4,6,7,1,2,3,4,5,6,7,2,3,5,6,3,4,5,2,6,1,7
4Ø FOR I = 1 TO 31 : READ Y(I) : NEXT I
5Ø DATA 1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,4,4,5,5,
   5,5,6,6,6,7,7,8,8
6Ø X = 1Ø : Y = 1Ø
7Ø FOR I=1 TO 31 : CURSET X+X(I),Y+Y(I),1 : NEXT I
```

A 'space invader' is an artificial image in the sense that it takes its form from the available graphics characters rather than an inherent shape of its own that is modelled, or approximated, using the graphics characters. In the following section a procedure for sketching a shape on the screen is described. For demonstration purposes we shall revert to low resolution graphics. The principle, however, is the same whether you use high or low resolution. It demonstrates that recognisable sketches

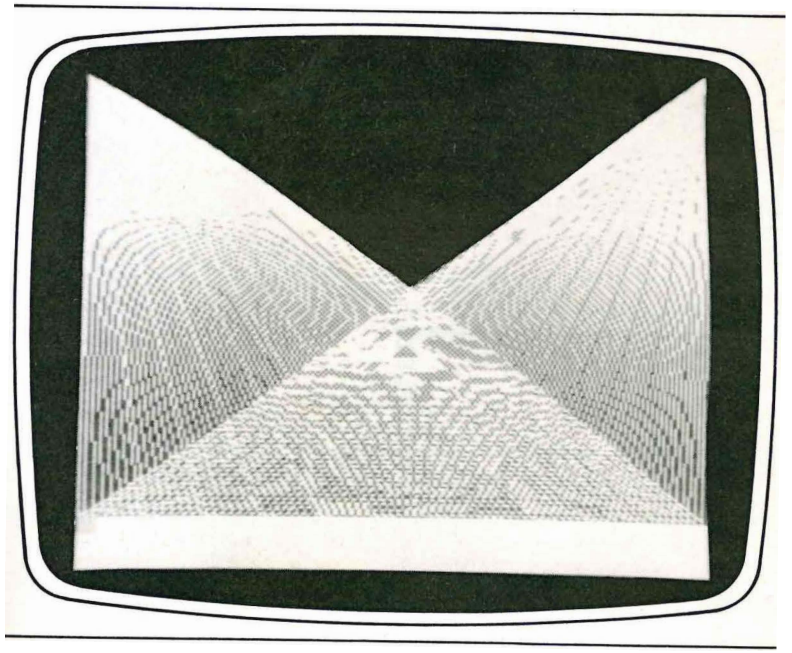


Figure 4.10 Display produced by Lines program.

can be produced, while at the same time showing that the limited definition of low resolution display can cause problems regarding the accuracy of the sketch.

Suppose we want to draw the butterfly shown in Figure 4.11(a) on the screen. To do this, draw a square grid over it as shown in Figure 4.11(b), and then, for each square of the grid in turn find the graphics character most closely approximating to it. The result of this will be something like that shown in Figure 4.11(c) while the outline of the butterfly as it will appear on the screen is shown in Figure 4.11(d). Finally, find the codes for the graphics characters and write a program to print them in the right place on the screen. The best method is once again to use the READ and DATA features of BASIC. The names are fairly self-explanatory; a READ command reads data from a DATA list. The first READ command executed in a program will read the first data item from the DATA list, the second takes the second item, and so on. By putting the READ command into a FOR . . . NEXT loop, you can read through the entire DATA lists quite simply.

In the following program to draw the butterfly in an 8 x 14 block, notice that we create an 8 x 14 matrix at line 10. The data is contained in lines 100 to 180. Lines 30 to 80 READ the DATA and PLOT the picture. Line 90 simply prevents the program from terminating. This prevents

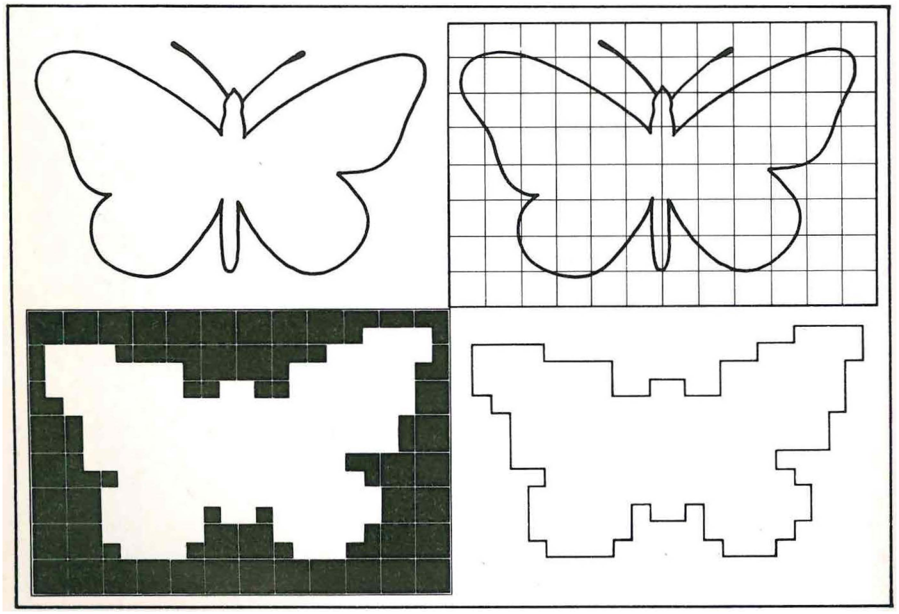


Figure 4.11 (a) Butterfly. (b) Butterfly with grid. (c) Butterfly composed of graphics characters. (d) Outline of image plotted on screen.

the appearance of the Ready message that would otherwise spoil the display on the screen.

```
  5 PRINT CHR$(17) : INK 2 : PAPER 4
 1∅ DIM A(8,14)
 2∅ CLS
 3∅ FOR R = 1 TO 8
 4∅ FOR C = 1 TO 14
 5∅ READ A(R,C)
 6∅ PLOT C,R,CHR$(A(R,C))
 7∅ NEXT C
 8∅ NEXT R
 9∅ GOTO 9∅
1∅∅ DATA 32,32,32,32,32,32,32,32,32,32,32,32,32,32
11∅ DATA 32,32,128,128,32,32,128,32,32,32,128,128,32,32
12∅ DATA 128,128,128,128,128,128,128,128,128,128,128,128,32,32
13∅ DATA 128,128,128,128,128,128,128,128,128,128,128,128,128,
    32
14∅ DATA 32,128,128,128,128,128,128,128,128,128,128,128,128,32
15∅ DATA 128,128,128,128,128,32,128,32,128,128,128,128,32,32
16∅ DATA 32,128,128,128,128,32,32,32,128,128,128,128,128,32
17∅ DATA 32,32,128,128,32,32,32,32,32,128,128,32,32,32
```

A display produced by such a program is shown in Figure 4.12. The problems of resolution can be tackled in a number of ways. The simplest is to stand further away from the screen, letting your eye and brain integrate and resolve the image as its finer detail becomes less clear. A more active measure would be to switch to high resolution graphics that would enable the use of a much larger number of grid squares. The positioning of the grid is also important, since the details that are vital for recognition should be captured as accurately as possible. Finally, a little artistic licence in the design of the displayed image may also help considerably.

Movement

Once static displays can be produced, it seems natural to progress to the generation of moving displays. The programs presented in this section make it possible for the user to control the movement of a shape on the screen. Besides being fascinating in itself such programs illustrate the techniques used in many games programs.

It is worth mentioning that the Oric has the facility to use joysticks to control movement on the screen, but since this is a more advanced form of programming we will not discuss it in this book.

Using the instructions for PLOTting a space invader we can write a program to move the invader backwards and forwards across the screen. The program should scan the keyboard to see if any 'movement' keys have been pressed, and if so, it should move the invader appropriately. A refinement is needed to prevent the moving shape from leaving a trail behind itself. Movement is simulated by redrawing the entire shape by one position to the left or right. If it moves to the right, however, it will leave its leftmost characters where they were on the screen. One method of avoiding this is for the shape to have a surround of spaces so that the part left behind is always blank. In the following program, which you will see is an expansion of our earlier space invader drawing program, movement is effected by the <LEFT ARROW> and <RIGHT ARROW> keys.

```
  5 HIRES
 1Ø DIM X(6Ø),Y(6Ø)
 2Ø FOR I = 1 TO 31 : READ X(I) : NEXT I
 3Ø DATA 3,4,5,2,3,4,5,6,1,2,4,6,7,1,2,3,4,5,6,7,2,3,5,6,3,4,5,2,6,1,7
 4Ø FOR I = 1 TO 31 : READ Y(I) : NEXT I
 5Ø DATA 1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,4,4,5,5,5,5,6,6,6,7,7,8,8
 6Ø FOR I = 32 TO 54 : READ X(I) : NEXT I
 7Ø DATA 2,6,1,7,Ø,3,5,8,Ø,8,1,4,7,2,6,1,3,5,7,Ø,2,6,8
 8Ø FOR I = 32 TO 54 : READ Y(I) : NEXT I
 9Ø DATA 1,1,2,2,3,3,3,3,3,4,4,5,5,5,6,6,7,7,7,7,8,8,8,8
1ØØ X = 1Ø : Y = 1Ø
11Ø FOR I = 1 TO 31 : CURSET X+X(I),Y+Y(I),1 : NEXT I
12Ø GET A$
13Ø P = -1 : IF ASC(A$) = 9 THEN P = 1
14Ø X = X + P
15Ø IF X < 1 OR X > 36 THEN GOTO 12Ø
16Ø FOR I = 32 TO 54 : CURSET X+X(I),Y+Y(I),Ø : NEXT I
17Ø GOTO 11Ø
```

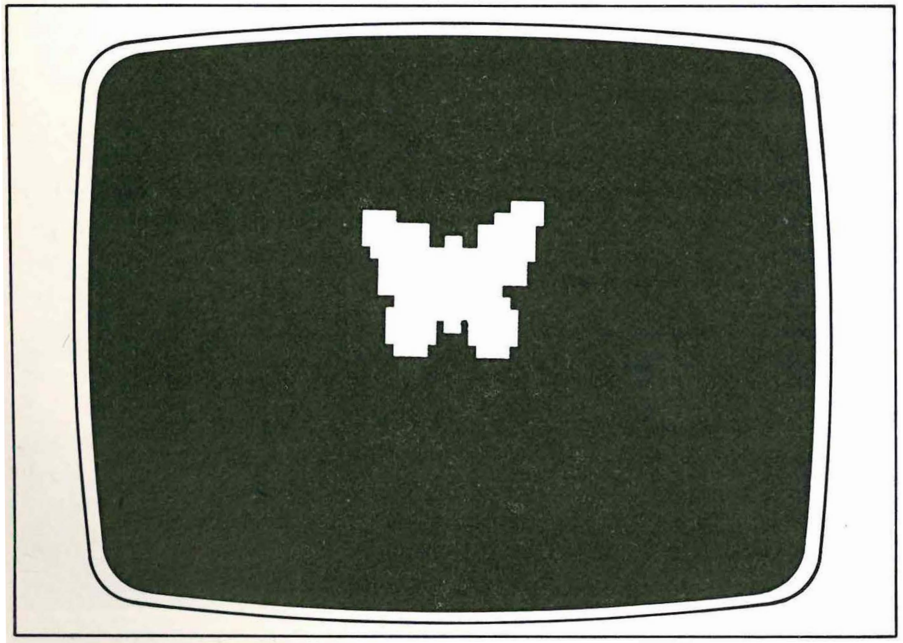


Figure 4.12 Butterfly as displayed on the screen.

Notice line 120 and the GET$ command. This command scans the keyboard and looks for a key that is depressed. The next three lines check that only the required movement keys will have any effect, jumping back to line 120 if spurious keys are pressed.

Animation

Displaying still pictures at a sufficiently high rate produces the illusion of continuous movement. All moving-picture systems including films and television rely on this effect, which depends on several human

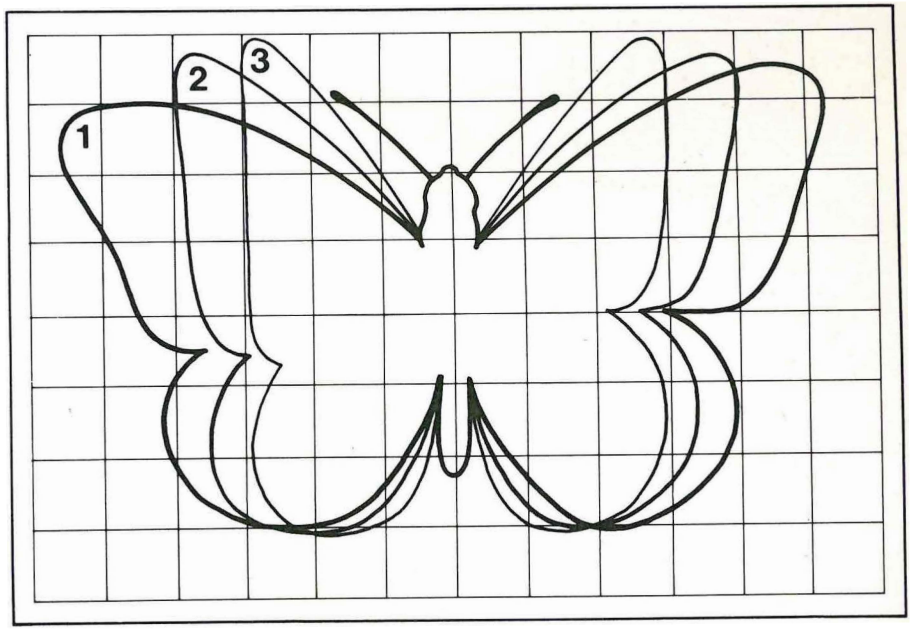


Figure 4.13 Frames 1, 2 and 3 for flying butterfly.

characteristics including the persistence of vision. The program presented in this section attempts to produce a mobile display by plotting successive static images in precisely the same way as a moving picture is produced by showing successive static images quickly enough.

The following program produces a mobile display of a flying butterfly: the successive frames catch different positions of the butterfly's wing when in flight. The sequence from which the frames are derived is shown in Figure 4.13. The first frame, with the wings fully extended, is the image produced earlier. The other frames are obtained in the same way as the first. In the program, the codes for the three frames are read in first, and then the frames are plotted repeatedly in the sequence 1,2,3,2. The flow chart of the program is given in Figure 4.14.

The program is:

```
  5 PRINT CHR$(17) : INK 2 : PAPER 4
 1Ø DIM  A(9,14) ,B(9,14),C (9,14)
 2Ø CLS
 3Ø FOR R = 1 TO 9
 4Ø FOR C = 1 TO 14
 5Ø READ A(R,C)
 6Ø NEXT C
 7Ø NEXT R
1ØØ DATA 32,32,32,32,32,32,32,32,32,32,32,32,32,32,
```

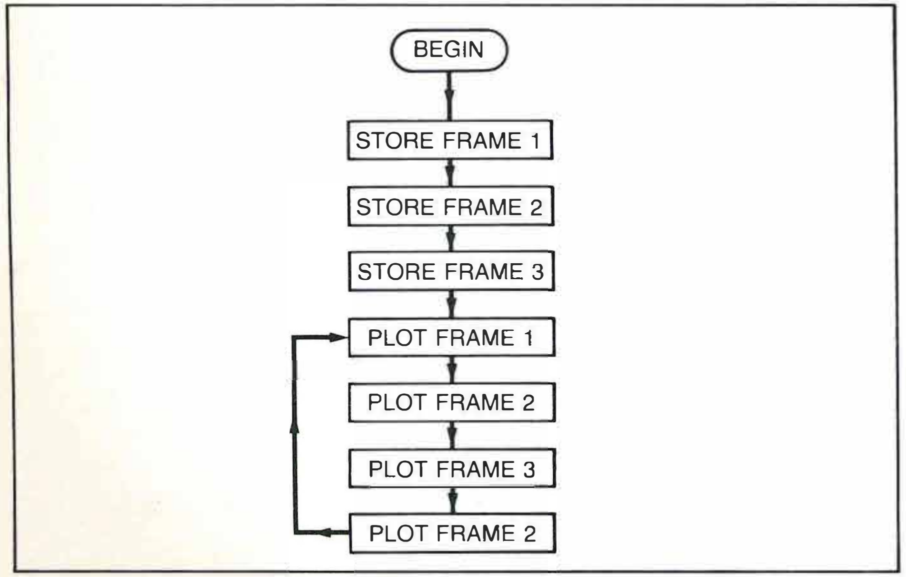


Figure 4.14 Flow chart for flying butterfly program.

```
11Ø DATA 32,32,128,128,32,32,128,32,32,32,128,128,32,32
12Ø DATA 128,128,128,128,128,128,128,128,128,128,128,
     128,32,32
13Ø DATA 128,128,128,128,128,128,128,128,128,128,
     128,128,128,32
14Ø DATA 32,128,128,128,128,128,128,128,128,128,128,
     128,128,32
15Ø DATA 128,128,128,128,128,32,128,32,128,128,128,
     128,32,32
16Ø DATA 32,128,128,128,128,32,32,32,128,128,128,128,
     128,32
17Ø DATA 32,32,128,128,32,32,32,32,32,128,128,32,32,32
18Ø DATA 32,32,32,32,32,32,32,32,32,32,32,32,32,32
2ØØ FOR R = 1 TO 9
21Ø FOR C = 1 TO 14
22Ø READ B(R,C)
23Ø NEXT C
24Ø NEXT R
3ØØ DATA 32,32,32,32,32,32,128,32,32,32,32,32,32,32
31Ø DATA 32,32,32,128,32,32,128,32,32,128,128,32,32,32
32Ø DATA 32,128,128,128,128,128,128,128,128,128,128,32,
     32,32
33Ø DATA 32,128,128,128,128,128,128,128,128,128,128,32,
     32,32
34Ø DATA 32,128,128,128,128,128,128,128,128,128,128,
     128,32,32
35Ø DATA 32,32,32,128,128,128,128,128,128,128,128,128,
     32,32,32
36Ø DATA 32,128,128,128,128,128,128,128,128,128,128,
     128,32,32
37Ø DATA 32,32,128,128,128,32,128,32,128,128,128,32,
     32,32
38Ø DATA 32,32,32,128,128,32,32,32,32,128,32,32,32,32
4ØØ FOR R = 1 TO 8
41Ø FOR C = 1 TO 14
42Ø READ C(R,C)
43Ø NEXT C
44Ø NEXT R
5ØØ DATA 32,32,32,128,32,32,128,32,32,128,32,32,32,32
51Ø DATA 32,32,128,128,128,32,128,32,128,128,128,32,32,32
52Ø DATA 32,32,128,128,128,128,128,128,128,128,128,32,32,32
53Ø DATA 32,32,128,128,128,128,128,128,128,128,128,32,32,32
```

```
540 DATA 32,32,128,128,128,128,128,128,128,128,128,32,32,32
550 DATA 32,32,32,128,128,128,128,128,128,128,32,32,32,32
560 DATA 32,32,128,128,128,128,128,128,128,128,128,32,32,32
570 DATA 32,32,128,128,128,32,128,32,128,128,128,32,32,32
580 DATA 32,32,32,128,32,32,32,32,32,128,32,32,32,32
600 FOR R = 1 TO 9 : FOR C = 1 TO 14
610 PLOT C,R,CHR$(A(R,C))
620 NEXT C,R
630 FOR R = 1 TO 9 : FOR C = 1 TO 14
640 PLOT C,R,CHR$(B(R,C))
650 NEXT C,R
660 FOR R = 1 TO 9 : FOR C = 1 TO 14
670 PLOT C,R,CHR$(C(R,C))
680 NEXT C,R
690 FOR R = 1 TO 9 : FOR C = 1 TO 14
700 PLOT C,R,CHR$(B(R,C))
710 NEXT C,R
720 GOTO 600
```

This program shows how the simulation of animation can be achieved. In this particular instance it does not work quickly enough to be very effective. Obviously, the larger and more complicated the image, the longer it takes the computer to display the different frames, and the less effective the animation. One method that can be used to increase the frame sequence in animation is to plot only the changes necessary to convert one frame to the next rather than to plot entire frames all the time. You might also like to produce an animation of a much smaller display to see how much faster and more effective it becomes. Clearly, animation in high resolution graphics is much more effective.

Dynamic simulation

This section provides a dynamic simulation of a system that experiences random growth and decay. It displays a community that grows initially from a single cell. When it reaches a certain size it decays to a lower level and then fluctuates between those two levels. The display can be taken as a simulation of the growth of a town or of a community of insects, although budding town planners will already know that real towns do not grow randomly! The random element of the program is provided by a command RND that generates a pseudo random number.

```
   5 DIM A(12,16) : INK 2 : PAPER 4
  10 CLS : G = 1 : C = 1
  20 T = 1 : A(5,6) = 1 : GOSUB 1000
  30 FOR I = 1 TO 12
  40 FOR J = 1 TO 16
  50 IF RND(1) > 0.9 THEN A(I,J) = T
  60 NEXT J : NEXT I
  70 C = 0
  80 FOR I = 1 TO 12
  90 FOR J = 1 TO 16
 100 IF A(I,J) = 1 THEN C = C + 1
 110 NEXT J : NEXT I
 120 G = G + 1 : GOSUB 1000
 130 IF C > 99 THEN T = 0
 140 IF C < 30 THEN T = 1
 150 GOTO 30
1000 FOR I = 1 TO 12
1010 FOR J = 1 TO 16
1020 IF A(I,J) = 1 THEN PLOT J+5,I+5,"★"
1030 IF A(I,J) = 0 THEN PLOT J+5,I+5,"(SPC)"
1040 NEXT J : NEXT I
1050 PLOT 5,20,"CONGRATULATIONS" + STR$(G) +
     "(2SPC)POPULATION" + STR$(C) + "(2SPC)"
1060 RETURN
```

Special commands

Finally, there is a number of commands in the Oric's graphics particularly worthy of attention. These include FILL, CIRCLE, and PATTERN. FILL is demonstrated in the first of the following three programs, CIRCLE in the second, and PATTERN (which can be used to alter the make-up of lines – particularly in the DRAW command – and includes useful pattern values of 15, 170 and 30) is demonstrated in the third. This last one can be called Random Fungus. See Figures 4.15, 4.16 and 4.17.

```
 5 HIRES
10 I = INT(RND(1)★200)
20 CURSET 0,I,0
30 X = RND(1)★8+16
40 FILL 1,1,X
50 W = INT(RND(1)★10)
60 Q = INT(RND(1)★4)+3
70 N = INT(RND(1)★12)+1
80 MUSIC 1,Q,N,5
```

```
 9Ø WAIT W
1ØØ GOTO 1Ø

 1Ø HIRES
 2Ø I = INT(RND(1)★17Ø)+3Ø
 3Ø J = INT(RND(1)★12Ø)+4Ø
 4Ø CURSET I,J,3
 5Ø K = INT(RND(1)★1Ø)+5
 6Ø CIRCLE K,1
 7Ø GOTO 2Ø

 1Ø PAPER Ø
 2Ø HIRES
 3Ø PATTERN 17Ø
 4Ø FOR N = 1 TO 25
 5Ø I = INT(RND(1)★17Ø)+3Ø
 6Ø J = INT(RND(1)★12Ø)+4Ø
 7Ø CURSET I,J,3
 8Ø FOR K = 1 TO INT(RND(1)★1Ø)+5
 9Ø CIRCLE K,1
1ØØ NEXT K
```

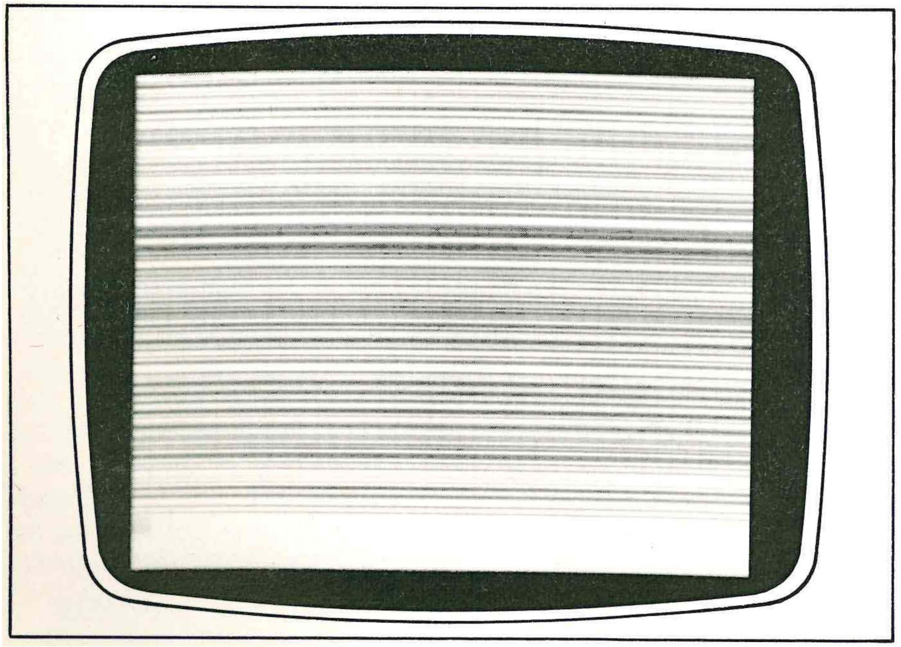


Figure 4.15 Display produced with the FILL command.

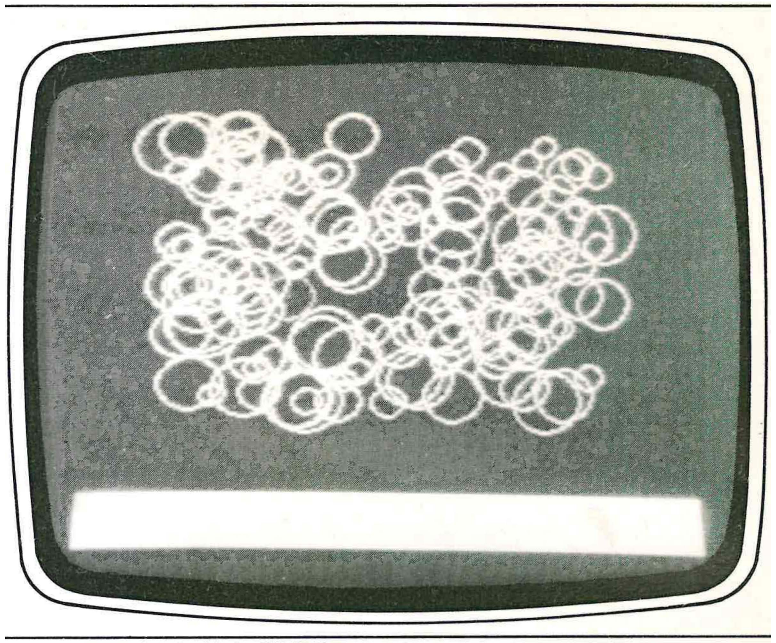


Figure 4.16 Display produced with the CIRCLE command.

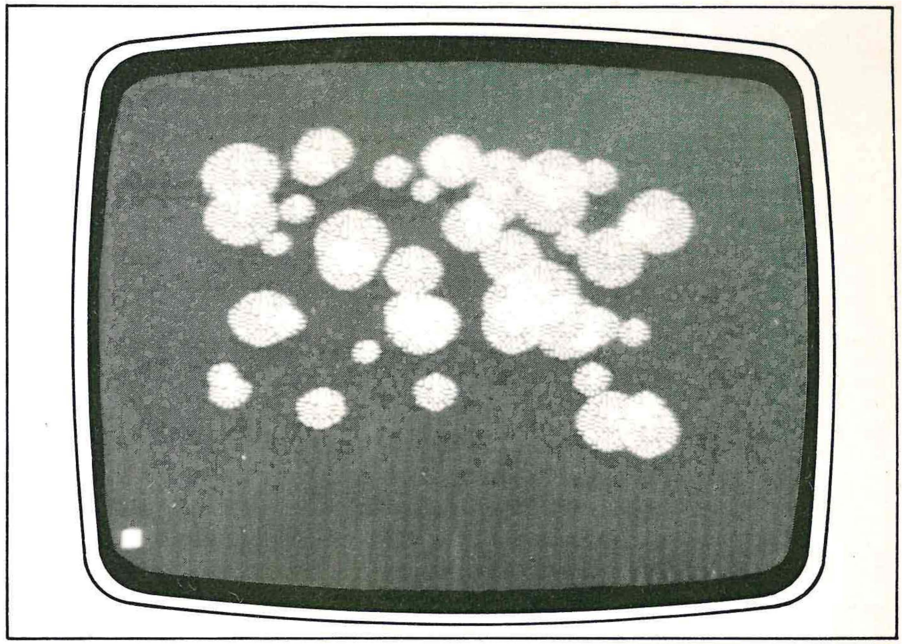


Figure 4.17 Fungus display produced with the PATTERN command.

Chapter 5
Special features of the Oric

Specification of the Oric

In this chapter, information about the Oric and some of its special features either not mentioned or mentioned only briefly in earlier chapters, is now gathered together to provide a basic reference chapter. It is not intended to be an exhaustive collection of data about the Oric, but it does include features that are of interest to the new user of the computer.

Manufacturer:	Oric Products International
Microprocessor:	MCS 6502
Screen display:	40 (38 usable) characters by 27 lines.
Keyboard:	56 keys and one space-bar. The letters are laid out in typewriter style: QWERTY
Memory size:	48K
Language:	Oric Extended BASIC
Graphics repertoire:	low resolution – a number of graphics characters accessed in LORES 1; with a grid size of 39 x 27 squares high resolution – one grid sizes: 240 x 200 pixels
Peripherals:	The Oric will connect to a normal colour television (or black and white), or to a visual display monitor. It will also connect to a standard domestic cassette recorder. At the time of writing, Oric Products International does not produce its own joysticks, although the computer has a port to which joysticks may be connected. Oric Products International does not produce its own printers. The computer, however, does have a Centronics compatible parallel printer port, and will therefore accept a wide range of printers. At the time of writing, we know of no disk drives available for the Oric. These will probably be available in the future.

Software: Machine language code can be put into memory by a BASIC program using the POKE command. Routines thus entered can then be accessed by using the DEF USR command.

If you buy disk drives to go with the Oric, the internal ROM contained BASIC interpreter is automatically masked out. This leaves a massive 64K of user addressable RAM, and makes the use of disk-based high level languages like Forth, Prolog, Pascal, Lisp and Logo possible.

Inside the Oric microcomputer

You must provide your own screen for the display, which could be a domestic television set (black and white or colour), or a separate industry monitor. The keyboard is similar to a typewriter differing only a few extra keys. Since these external features are familiar, most of this section is devoted to the inside of the computer.

Inside the Oric are the electronics for producing the specialised screen displays, the sound producing circuits, the memory, and of course all the logic for running a sophisticated computer. All of this is mounted on one medium sized printed circuit board.

A printed circuit board is the most convenient way to mount and interconnect the large number of components of the computer. It has copper tracks laid down on it to connect the mounts into which the various chips are inserted. The layout of the printed circuit board reveals the essential structure of the microcomputer.

Unlike some other microcomputers, the Oric does not have its power supply unit within the main casing. The Oric's power unit is a separate box that connects the computer to the domestic mains supply. It converts the 240 volts alternating supply from the mains to those required by the computer components.

The memory available to the user, particularly for storing BASIC programs, is provided by 'random access' chips, or RAMs. The information stored in this type of memory can be accessed, and can be replaced by the program as required. When the computer is switched off, all the information stored in RAM is lost.

There are, of course, certain features of the Oric that are always required and which must not be replaced or lost when the computer is turned off. To give just two examples: BASIC must always be available, and the characters to be displayed on the screen should be able to be generated at any time. Such functions are provided by chips with

information permanently stored in them. These chips are known as 'read only memories', or ROMs.

The sockets for connecting the Oric computer to other devices are at the edge of the printed circuit board and at the two sides and the back of the case.

Sound on the Oric computer

The Oric computer enjoys a growing reputation for the quality and range of its sound capabilities. Sound is, furthermore, very simple to produce.

The Oric computer includes an internal loudspeaker. Unlike some of its competitors, this loudspeaker can produce adequate volume levels. The following program is an example of its capabilities:

```
 1Ø DIM O(45),N(45),W(45)
 2Ø FOR I = 1 TO 45 : O(I) = 3 : NEXT I
 3Ø O(4) = 2 : O(15) = 2
 4Ø FOR I = 1 TO 45 : READ N(I) : NEXT I
 5Ø DATA 1,1,3,12,1,3,5,5,6,5,3,1,3,1,12,1
 6Ø DATA 1,3,5,6,8,8,8,8,6,5,6,6,6,6,5,3
 7Ø DATA 5,6,5,3,1,5,6,8,1Ø,6,5,3,1
 8Ø FOR I = 1 TO 45 : READ W(I) : NEXT I
 9Ø DATA 2,2,2,3,1,2,2,2,2,3,1,2
1ØØ DATA 2,2,2,2,1,1,1,1,2,2,2,3,1,2
11Ø DATA 2,2,2,3,1,2,2,1,1,1,1,3,1,2,1,1,2,2,3
12Ø FOR I = 1 TO 45
13Ø MUSIC 1,O(I),N(I),5
14Ø WAIT 3Ø ★ W(I)
15Ø MUSIC 1,1,1,Ø
16Ø NEXT I
```

The use of the sound commands can be used with great effect in games with graphics – Arcade Space Invaders will probably be familiar to everyone. To demonstrate the concept of mixing sound and graphics, try the following program, which you may recognise as an adaptation of an earlier program:

```
 5 PAPER
1Ø HIRES
17 FOR N = 1 TO 5Ø
2Ø I = INT(RND(1)★17Ø)+3Ø
3Ø J = INT(RND(1)★12Ø)+4Ø
4Ø CURSET I,J,3
45 EXPLODE
```

```
50 FOR K = 1 TO INT(RND(1)*10)+5
60 CIRCLE K,1
70 NEXT K
75 WAIT 100
80 NEXT N
```

There are also three other ready-made arcade sounds that you may use. Try them in immediate mode by typing:

```
PING <RETURN>
SHOOT <RETURN>
ZAP <RETURN>
```

Using the Oric as a timer

If you read other books on programming in BASIC, you may come across a command called TI or TI$. This command instructs the processor to display the current value of an internal counter that automatically starts whenever you turn on the computer. The timing of this counter is extremely accurate, and it is consequently often referred to as a clock. Computers that include this clock/counter can be readily used as an elaborate and very accurate form of timer. Unfortunately, at the time of writing, there is no such clock/counter in the Oric. Test this by typing in the command:

```
PRINT TI <RETURN>
```

The response, you will see, is to display:

```
0
Ready
```

The command, which is a valid BASIC instruction, has been accepted by the BASIC interpreter, but there is no value for the processor to return.

This section, however, will show that with a little imagination, the Oric can still be made to simulate a timer with a fair degree of accuracy. The routine itself could be incorporated quite effectively into a recipe program. Let us say that the recipe has given both the ingredients and the instructions, and that the mixture has to be cooked for a certain length of time. It could call the following as a subroutine (using GOSUB and RETURN) and act as a timer for the recipe. Enter the following program and try to work out what is happening. There are no new commands, but notice that we use a 'question-mark' instead of the PRINT command. '?' is much quicker to enter and takes up less space on the line: it is simply a form of BASIC shorthand for 'PRINT'.

```
 10 CLS
 20 ? "(5SPC)HOW LONG DO YOU WANT"
 30 ? "(5SPC)TO SET THE TIMER FOR?"
 40 ?
 50 ? "ENTER THE NUMBER OF MINUTES";
 60 INPUT A
 70 CLS:? "COUNTING UP TO(SPC)";A;"(SPC)MINUTE/S"
 80 FOR I=1 TO A : FOR J=1 TO 25885 : NEXT J
 90 CLS : ? "COUNTING UP TO(SPC)";A;"(SPC)MINUTE/S"
100 ?
110 ? I;"(SPC)MINUTE/S OF(SPC)";A;"(SPC)COUNTED"
120 PING
130 NEXT I
140 FOR I = 1 TO 4
150 PING : WAIT 10 : NEXT I
160 CLS
170 ? "TIMES UP!!!"
180 ?
190 ? "DO YOU WANT TO RESET TIMER? Y/N";
200 INPUT B$
210 IF B$ = "Y" THEN GOTO 10 ELSE END
```

This program illustrates a number of the more simple features of BASIC that you have already come across: PRINT, FOR . . . NEXT, GOTO and PING. But it also demonstrates an interesting use of the FOR . . . NEXT loop that is particularly useful when programming the Oric: that is, as a 'delay loop'. Notice the format of the loop:

```
FOR I = 1 TO n:NEXT I
```

There is no separate command between the FOR and the NEXT part of the instruction. In other words, this construction instructs the computer to do nothing but go round in circles for a specified number of times. By varying the number of loops you can specify the length of time the computer takes to complete the command, and hence the duration of the delay introduced into the program.

Notice that our delay loop (which is the 'timer') at line 80 is for 1 to 25885 in steps of +1. Experimentation has shown that it takes the Oric almost exactly one minute to complete this number of loops. If you find that it is in fact only 59.9 seconds, try adding a few more loops: conversely, reduce the number to, say, 25800 if you find the Oric is taking too long.

Line 80 also shows a feature known as 'nested loops'. The first loop is

FOR I = 1 TO A. Now, A, as you will see at line 60, is a variable input by the user to specify the duration of the timer in minutes. The FOR J loop is thus repeated as many times as the FOR I loop specifies.

Lines 90-120 provide a simple counter and buzzer to show the passage of time. Try changing line 110 to read:

```
11Ø ? A–I;"MINUTES TO GO"
```

Lines 140 and 150 provide the alarm that is activated as soon as the nested FOR . . . NEXT timer is complete, while the rest of the program provides the opportunity to reset the timer or exit from the program.

The Oric has another command that can be used for introducing delays:

```
WAIT (n)
```

Here the value 'n' is equal to n times 10 milliseconds. Thus:

```
WAIT 1ØØ
```

introduces a pause of 1 second, while:

```
WAIT 6ØØØ
```

produces a pause of 1 minute. As a self-test project, you may like to write your own timer using the WAIT command rather than the delay loop.

Conclusions

This book has aimed to provide an easy introduction to using the Oric microcomputer, and it has described many of the applications in which the Oric can be used to good effect simply by loading and running a program. There are a large number of applications of this kind, including some business applications, which require no knowledge of how the Oric microcomputer works and need only a minimal knowledge of the instructions required to operate it. In these circumstances the program is all important. The Oric microcomputer is merely a vehicle for running the program. A special purpose system of this kind can demonstrate its worth by paying for itself in quite a short time. However, the Oric microcomputer is extremely versatile, being capable of as many activities as it can be programmed for. This versatility can be harnessed by running different programs for each of a range of applications.

Purchased programs do not always do exactly what you may want, so it is useful to be able to program the Oric microcomputer in order to modify such programs. Whether for this reason, or as a result of curiosity on how to tap the full potential of the Oric microcomputer, it is useful to be able to write programs. An introduction to programming the Oric

microcomputer is provided by this book, but it is only an introduction and Appendix 1 indicates several sources of information which can be used for further study.

The importance of the Oric microcomputer used as an educational tool has been stressed more than once in these pages. Its importance as an example of modern technology should not be overlooked. It has a merit merely existing as an available product of the technology that will be used increasingly in the future, by providing an appreciation of how the technology is applied in everyday situations.

When viewed from different perspectives, the Oric microcomputer is seen as a tool which can be used in a number of ways. This book has attempted to introduce many of these uses and to indicate the sources of information which will help in developing these avenues further.

Appendix 1

Further reading

This appendix lists some books and magazines that are suitable for further reading to follow up particular topics that are mentioned, introduced or developed within the book.

New books are appearing daily, but the quality and usefulness varies enormously. Basically they fall into three categories: the general purpose book (dealing with microcomputing in general); the hardware specific book (dealing in detail with one particular computer); and the book of programs that is simply a list of programs with a varying degree of excellence, usefulness and even accuracy. This book aims to be a compromise between all three types, selecting and compiling only the best and most useful elements of each.

Like book titles, new computer magazines appear almost daily: but unlike book titles, others seem to disappear equally fast. It may be that some of the magazines listed below may have been disbanded by the time this book is on the market: equally, other new magazines not here mentioned will by then have been published. There are four major publishers of microcomputer magazines (ECC, EMAP, IPC and VNU), although there is a number of independent publishers with just one or two magazine titles.

Books on the Oric

Learning to Use . . . is, as far as we know, the very first book to be published specifically on the Oric microcomputer. It won't be the last!

General Books

Illustrating BASIC, by Donald Alcock (Cambridge University Press, 1978).
Somewhat dated now, but still the best general introduction to BASIC programming available. Like Dennis Jarrett's book (see below), an easy style can (wrongly, we think) be interpreted as a patronising approach from the author.

Illustrating Computers, by Colin Day and Donald Alcock (Pan, 1982).
Companion book to Illustrating BASIC; and equally recommended for the newcomer to computers.

Software Secrets, by Graham Beech (Sigma Technical Press, 1981).
This book is actually written for a Sharp MZ-80K microcomputer, but since it is really a book about programming ideas and techniques, it makes useful and interesting reading.

The Good Computing Book for Beginners, by Dennis Jarrett (ECC Publications Ltd, 1980).
Claimed to be 'all you need to know about computers (and nothing you don't)'; its main use is in an extensive glossary (over 220 pages!). Jarrett writes in an easy and colloquial style ('ECMA – It sounds like a skin complaint but it stands for the European Computer Manufacturers' Association'). If you object to the style, don't buy the book!

The First Book of Microcomputers, by Robert Moody, (Hayden).
Claimed to be the home computer owner's 'best friend'.

Books about programming

Inside BASIC Games, by R Mateosian (Sybex).
Teaches interactive BASIC programming through games.

BASIC Computer Programs for the Home, by Charles D Sternberg, (Hayden, 1980).
A comprehensive book of practical home application programs that will be helpful to both the novice and experienced owner by increasing the usefulness of any home computer.

The BASIC Workbook, Kenneth Schoman, Jr (Hayden).
A hands-on approach to learning BASIC and the fundamentals of problem-solving using a computer. The book is subtitled: 'Creative techniques for Beginning Programmers'.

BASIC and the Personal Computer, Dwyer and Critchfield (Addison Wesley).

BASIC from the Ground Up, by David E Simon (Hayden Book Co).

Computer Games for Businesses, Schools and Homes, by Gary Orwig and William S Hodges (Little, Brown and Co).

Magazines

The Oric User
At the moment, this magazine does not exist. But we are absolutely certain that one or other (or all!) of the major magazine publishers (IPC, EMAP, VNU etcetera) will sooner or later produce a periodical with a

similar title and designed specifically for Oric users. While we obviously have no idea of how good this (or these) magazines will eventually be, nevertheless we are fairly confident that at least one will exist by the time or soon after you read this book.

Computing Today
This magazine is considered by many to be the best of the popular computing magazines. It covers the whole field of microcomputing and often provides listings of useful programs. Converting these to run on the Oric could be both entertaining and educational.

Personal Computer World
Often abbreviated to *PCW*, this is in many ways required reading for microcomputer users.

Micro Software and Systems Magazine
This is a new magazine that is devoted mainly to business software. Each issue focuses on a different computing application: graphics, operating systems, word processing, etc. It may be of interest to those who intend to use their Oric to develop serious software, by providing an outline of the existing software in specific fields.

Which Computer?
A magazine devoted to evaluating mini and microcomputer systems for business. One of the best in its category.

Which Word Processor? and Office Systems
Originally simply *WWP?* this magazine was a bimonthly supplement to *Which Computer?* It did and does serve a similar purpose within its own specialised market, and has since expanded its scope to include office systems in general (largely because the modern word processor has similarly expanded its scope). Expect to see the magazine become monthly in the near future.

Which Micro and Software Review
This is one of the better of the new crop of computer magazines. It features articles on a wide range of equipment from the lower end of business machines to the new small home computers. Articles on the Oric do appear now and again, but it is particularly useful for gaining an overall view of the general trends in microcomputing.

What Micro?
Produced by a rival publisher to the three 'Whiches' and clearly aimed at the same reader, this magazine seems to follow the general approach not of the consumer magazine 'Which?', but that of the consumer magazine

'What Car?'. After all, why re-invent the wheel when the one you've got works perfectly well?

MicroDecision
From the same publishers as What Micro? there is inevitably some overlap of general coverage. Nevertheless, and despite the fact that it has come in for considerable criticism from 'professionals', it is a well produced and very good value magazine.

Appendix 2

Glossary

Access
To obtain data from, or place data into storage; which may be either main memory or backing storage.

Acoustic coupler
A portable modem that can be used to transmit signals from one computer to another via the public telephone network. It takes its name from its main visible feature; that is, a coupling device that receives the telephone handset during transmission.

Address
The storage location of information, either in the computer's memory, or on cassette tape or floppy disk.

Alphanumeric
A term used to describe a string composed of either letters and/or numbers. In American literature, following the general American tendency to condense words wherever possible, it is sometimes found as 'alphameric'.

Archive
As a noun, it describes a long term storage medium, such as a cassette or floppy disk. As a verb, it describes the action of placing a program or other information on to a cassette or floppy disk for the purpose of long term storage.

Argument
Commonly used to describe the value associated with a command.

Array
A linear arrangement of individual items of data that can each be identified by an index that allows single items to be examined. Thus, the command DIM A$(2Ø) will provide an array of 20 memory locations that can be examined sequentially by their names A$(1), A$(2) and so on.

ASCII characters
The *A*merican *S*tandard *C*ode for *I*nformation *I*nterchange (pronounced 'as-key'): a code used by most computers to represent 128 different text

and computer control characters. It uses 7 bits for each character. For example, the ASCII code for the character A is 1000001.

Assembly language
A language similar in structure to machine language, but made up of mnemonics and symbols. Programs written in assembly language are slightly less difficult to write and understand than programs in machine language.

BASIC
The computer language immediately available when many microcomputers are turned on (including the Dragon and the BBC Microcomputer), and in which commands to it are expressed. BASIC actually stands for Beginner's All-purpose Symbolic Instruction Code.

Baud
A measure of the speed by which signals are sent over a communications line. In practice it corresponds more or less with 'bits per second'.

Binary
A number system with two digits, '0' and '1', with each digit in a binary number representing a power of two. Most digital computers are essentially binary in nature.

Bit
Short for '*b*inary dig*it*'. A bit (0 or 1) is the smallest unit of digital information.

Board
A printed circuit board, or PCB, is sometimes called a printed circuit card. It is usually plastic and has its required circuits (in a conducting medium like copper) printed on its surface. There is also a number of small holes where individual electronic components can be plugged or soldered into the board to make contact with those circuits. At the other end of the circuit is the edge of the board, which is equipped with connectors. The connectors engage with further circuitry in the backplane, which is the part of the computer that interconnects the various boards. The processing functions of a microcomputer and its main memory will be held on a small number of PCBs.

Boot
Short for 'bootstrap': the process of loading an operating system from disk or tape into computer memory.

Branch
A departure from the sequential performance of program statements. An

unconditional branch causes the computer to jump to a specified program line every time the branching statement is encountered. A conditional branch transfers program control in accordance with the result of a conditional test.

Buffer
An area of computer memory for temporary storage of either input or output data.

Bug
An error in computer programming. Because the error may only be noticed when it affects a different part of the program, many bugs are very difficult to find.

Byte
A unit of computer storage that comprises 8 bits. It is almost the same as the storage needed for a single character. Thus, a 32K Byte computer has approximately 32000 storage locations each able to store a character.

Central processing unit (cpu)
The 'brains' of the computer, containing the electronic circuits that interpret and execute instructions.

Character
A letter, number, punctuation symbol, or special graphics symbol.

Chip
Literally, the chip of silicon from which an integrated circuit is fabricated, but used popularly to refer to the integrated circuit itself.

Constant
A specific numeric or string value. A numeric constant is any real number, such as 1.2 or −4321. A string constant is any combination of characters, up to the limit set by each different version of BASIC, enclosed in quotation marks, such as "HELLO" or "221b Baker Street". See also 'Variable'.

CP/M
An operating system originally devised by Gary Kildall and now owned by the company, Digital Research, of which he is president. It was designed to run on the Intel 8080, 8085 and Zilog Z80 processors, and has become the industry standard operating system for small business microcomputers, with a software library of, by now, well over 5000 independently produced software packages. A number of the new 'home' computers are capable of running CP/M either directly or via add-on cartridges. The Commodore 64 and TI 99/4A are examples of the

latter, while, at the time of going to press there are plans to implement it on the Lynx, and rumours that it will be implemented on the Spectrum.

Cursor
The flashing bar or square on a computer's visual display screen which indicates the position at which the next item will be displayed.

Daisywheel
A type of print element that loosely resembles a daisy, and the type of printer that uses that element. On the print el ment, characters are held on the end of spokes (corresponding to the petals of the daisy) radiating from a central hub. The daisywheel printer is the current standard printer for all applications, such as business word processing, where good quality printing is necessary.

Database
An organised collection of data from which either data or the properties of items of data can easily be retrieved.

Debug
To find and correct errors (bugs) in a program.

Default
This is a value that is automatically assigned by the program being used whenever the user of that program does not specify a particular value for a given variable.

Disk
A disk on which programs or data can be stored as magnetic patterns on the surface of the disk, and from which recorded information can be rapidly retrieved. Also known as a floppy disk.

Disk Operating System (DOS)
An operating system specifically for a disk drive. A program to facilitate the storage of information on disk and its retrieval from the disk. The DOS selects unused portions of the disk surface for data storage, and then remembers where everything is for data retrieval. See also, operating system.

Double-density
It is possible for disk drive manufacturers to double the number of bits stored per inch on a disk. You pay extra for the drives and for the disks themselves for this increased storage capacity – but the price increase is less than that for buying a second disk drive. Since double-density drives came on to the market, the original density disks are now often referred to as 'single-density'.

Double-sided
It is now possible for disk drive manufacturers to produce disks and disk drives with data storage facilities on both sides of the disk. The extra technology involved in writing to and reading from both sides of a disk means that these disks are far more expensive than is usually acceptable for home computers. These drives are therefore more frequently found on business computers. Since double-sided drives came on to the market, the original disks are now often referred to as 'single-sided'.

Execute
The act of obeying the instructions contained in a computer program. Synonymous with running a program.

File
A collection of related data records stored on a device, such as a cassette tape or floppy disk.

Floppy disk drive
A peripheral device used to store programs and data on disks made of a thin flexible plastic coated with a magnetic recording surface (called a floppy disk or diskette). Floppy disks are more reliable and much faster in operation than simple cassette tapes.

Flow chart
A diagram indicating in stylised form the steps of a computation. It is used as an aid to program development.

Graphics
Pictures produced by a computer.

Hardware
More properly called 'computer hardware', it is the collection of physical devices that make up a computer system.

High level language
A language that is more intelligible to human beings than it is to machines; for example, BASIC, Pascal, FORTRAN. See also: Low level language.

Increment
A value that consistently modifies a variable. The FOR . . . NEXT . . . STEP instruction consistently modifies (increments) the FOR variable by the STEP value.

Integer
A whole number, either positive, negative, or zero.

Integrated circuit
An electronic circuit fabricated in extreme miniature form on a silicon chip typically a few millimetres square.

Interface
An electronic and/or physical connection between different devices. A serial interface transmits or accepts information one bit at a time, whereas a parallel interface transmits or accepts information several bits at a time.

Interpreter
Software which translates a program in a high-level language into machine code, which comprises the binary instructions which correspond directly to computer operations and is the 'language' that the microprocessor understands. The program is executed at the same time as it is interpreted. This is distinct from a 'compiler', which performs a similar operation but produces a compiled program from the user's source program. This compiled program is the program that is ultimately executed. With an interpreter, each statement in the high-level language program is translated and executed immediately. This means you can add or delete instructions and see the effect immediately, so it speeds the process of program development. Interpreters might take up some memory, since they have to be waiting to translate; and interpreted programs are certainly slower when it comes to run-time (because a program already in machine code is inevitably much more efficient). But because interpreter languages do not require the compile process they are generally preferred for home computers. Apart from BASIC, you will find the APL and PASCAL languages frequently in interpreter form.

K
1K stands for 1 kilobyte of memory, and gives the size of memory consisting of multiples of 1024 storage locations.

Listing
A printout (which can be either as a display on the screen, or as a physical printed list) of the lines of instruction that makes up a program.

Loop
A group of consecutive program lines that are repeatedly performed, usually a specified number of times.

Low level language
A language that is more intelligible to machines than it is to human beings; for example, assembler. See also: High level language.

Machine code
The code in which instructions must be conveyed to a microprocessor in order that it may respond to them directly.

Mainframe
A term, whose derivation is now somewhat obscure, loosely used to describe any computer larger than a home, micro, or mini computer.

Memory
Also called main memory, core memory, or main storage. The integrated circuits of a computer in which information is stored that is directly accessible to the cpu, as opposed to peripheral, or backing storage which is accessible only via interfaces.

Microcomputer
A computer whose central processor is on a microprocessor.

Microprocessor
Physically, a very complex integrated circuit. Functionally, an electronic device that can be programmed and can, in consequence, perform a variety of tasks.

Microsecond
A period of time equal to one millionth of a second.

Millisecond
A period of time equal to one thousandth of a second.

Mode
A condition or a set of conditions under which a particular set of rules applies.

Modem
A device that allows signals to be sent from one computer to another via the telephone network. There must be a modem between the sending computer and the beginning of the telephone link, and between the end of the telephone link and the receiving computer. A modem converts the digital signals from the computer into analogue signals for the telephone system, and back again; that is, it MOdulates and DEModulates the signals.

Operating system
Systems software that controls the computer and its peripheral devices.

Output
Information sent by the cpu to any peripheral device.

Peripheral
Equipment that can be attached to a computer, and can be used and controlled by the computer. Examples are cassette units, television screens, and printers.

Pixel
A graphics 'picture element' – the smallest programmable element on the screen.

Port
A socket on the computer into which you can plug a terminal or some other input/output device.

Printed circuit board
(see Board)

Prestel
The name for British Telecom's pioneering viewdata service. It is becoming increasingly important for home computer users with the launch of Micronet, a viewdata system specifically for the home computer user.

Program
An ordered sequence of commands given to a computer, so that when it obeys them it automatically performs a specified and complete task.

Prompt
A symbol (different for the different versions of BASIC), which marks the beginning of each program line during input from the keyboard; a symbol or phrase that requests input from the user.

RAM
Random-access memory. Memory whose contents are lost when the power supply is turned off. The amount of RAM determines how much memory is available for the user to store programs and data.

Record
A collection of related data elements, such as an individual's payroll information or a student's exam scores. A group of similar records, such as a company's payroll information, or a school's exam results, is called a file.

ROM
Read-only memory. This is permanent memory, typically used to store information that is always required, such as that which provides BASIC. This memory is not available to store the user's programs: it provides facilities required by the user.

Scroll
To move all the text on the screen of a video monitor (usually upwards) in order to make room for more (usually at the bottom).

Sprite
A high resolution programmable object used in sophisticated graphics. It is created by coding individual pixels in a matrix, and can then be treated as a whole. It can be made into almost any shape and can be moved freely around the screen.

Software
Computer programs; the list of instructions that tell a computer to perform a given task or tasks – as opposed to hardware (the computer itself). There are basically two types of software: systems and applications. Examples of systems software includes operating systems and language interpreters. Applications software includes programs that instruct the computer to perform specific applications, such as word processing, playing computer games etcetera.

String
A sequence of letters, numbers or symbols, usually arranged in some specific order, and treated as a unit.

Subroutine
A program segment which can be used more than once during the execution of a program, such as a complex set of calculations. In most forms of BASIC, a subroutine is best defined with the GOSUB and RETURN statements.

Telecommunications
A wide ranging term used to describe the transmission and reception of information by electronic means. It therefore includes the communication of one computer with another, whether it be by radio, cable, satellite or a combination of all three.

Trace
Listing the order in which the computer performs program statements. Tracing the line numbers can help you find errors in a program flow.

User
Any person or persons who use a computer.

User port
One of the connections at the rear of a number of microcomputers, which can be used to send or receive signals under the control of the user's program.

Users group
A group of people who have computers from a particular supplier, or who have some kind of computing interest. It is worth stressing the value that this kind of organisation can offer, even when, as sometimes happens, it is really a manufacturer-inspired mouthpiece designed primarily for marketing purposes. Users can discuss problems, swap solutions and programs, band together to get discounts on bulk-buying consummables like paper and disks, and if necessary present a coherent front to get some action from the supplier.

Variable
A name given to a value that may vary during the execution of a program. Think of a variable as a memory location or pigeon-hole where values can be replaced by new values during program execution.

Viewdata
The generic term for the database system pioneered by British Telecom and its public Prestel service. Typically, it involves either an adapted television set, or a computer, that accesses a large centralised database of information. When used with a home computer, it can provide information in both directions: to the user and from the user. The potential applications for this technology are limited only by our imagination, and range from the automatic dissemination of software (telesoftware), to armchair mail-order (with inspection, ordering and payment all done electronically), and to improving the democratic process of government by allowing snap referenda at almost any time on any subject.

Word processor
A system for processing textual material electronically and then printing it or, perhaps, transmitting it to a similar system. In this context, the processing is mainly editing.

Appendix 3
The Tower of Hanoi a game

This game is one of the classic logic problems. You are given a pile of discs of different sizes, and you have to move the discs one at a time from location A to location C in as few moves as possible. Location B is available as a temporary position to help you sort the discs. The problem, however, is that you must never put a disc on top of one that is smaller in size than itself.

To use the program, enter the following listing accurately. When entered, type RUN and follow the instructions that appear on the screen. You will soon be presented with a picture showing a Tower of the size you chose on the left hand side. Locations B and C, empty to begin with, are shown to the right. To move the discs, simply type the location of the disc you wish to move (say, 'A') followed by the location you wish to move it to (say, 'C'). Thus, 'A,C' may well be your first move. If it is, then 'A,B' must be your second move – you cannot do 'A,C' again (big disc on to little disc!), and there is little point in moving the small disc again before making any other move. Good luck!

```
 1Ø INK 2 : PAPER 4
 2Ø PRINT CHR$(17)
 3Ø CLS
 4Ø PLOT 1Ø,3,"TOWER OF HANOI"
 5Ø PLOT 1,7,"AIM: TO MOVE ALL DISCS TO C"
 6Ø PLOT 1,9,"RULE 1: ONLY MOVE ONE DISC"
 7Ø PLOT 9,1Ø,"AT EACH MOVE"
 8Ø PLOT 1,12,"RULE 2: LARGE DISCS MUST NOT"
 9Ø PLOT 9,13,"SIT ON SMALL DISCS"
1ØØ PLOT 1,16,"TO MOVE PRESS A B OR C KEYS"
11Ø PLOT 15,22,"PRESS ANY KEY"
12Ø GET A$
14Ø CLS : PLOT 1Ø,3,"TOWER OF HANOI"
15Ø PLOT 1,9,"HOW MANY DISCS DO YOU WANT?"
155 PLOT 1,12,"CHOOSE A NUMBER BETWEEN 2 AND 6
16Ø GET A$ : C = VAL(A$)
```

```
180 IF C > 1 AND C < 7 THEN 200
190 PLOT 1,12,"YOU HAVE TYPED AN INVALID NUMBER"
195 PLOT 1,15,"TRY AGAIN"
196 WAIT 200 : GOTO 140
200 FOR I = 1 TO C : A(I,1) = I : A(I,2) = 0 :
    A(I,3) = 0 : NEXT I
220 H(1) = C : H(2) = 0 : H(3) = 0 : M = 0
230 CLS : PLOT 12,2,"TOWER OF HANOI"
240 Y$ = "TARGET MOVES" : W$ = "MOVES MADE"
250 PLOT 1,4,Y$ + STR$(INT(2 ↑ C−1))
260 PLOT 1,6,W$ + STR$(M)
310 FOR I = 0 TO 37 : FOR K = 20 TO 26
320 PLOT, I,K,CHR$(149)
330 PLOT 38,K,CHR$(147)
340 NEXT K,I
350 PLOT 6,21,"A"
360 PLOT 19,21,"B"
370 PLOT 32,21,"C"
400 P$ = CHR$(128) : R$(6) = "(SPC)"
410 Q$(2) = P$
420 FOR I = 2 TO 6 : Q$(I+1) = Q$(I)+P$+P$ : NEXT I
430 FOR I = 5 TO 1 STEP − 1 : R$(I) = R$(I+1)+"(SPC)" :
    NEXT I
440 Q$(1) = R$(2) + R$(2) + "(SPC)"
450 FOR I = 2 TO 6 : Q$(I) = R$(I) + Q$(I) + R$(I) :
    NEXT I
470 GOSUB 5000
520 GOSUB 6000
522 GOTO 530
525 GOSUB 6075
530 IF H(I1) < 1 THEN PLOT 20,7,"IMPOSSIBLE(2SPC)" :
    GOTO 525
540 IF H(I2) = 0 THEN 560
550 IF A(C + 1 − H(I1),I1) > A(C + 1 − H(I2),I2)
    THEN PLOT 20,7,"INVALID MOVE" : GOTO 525
560 H(I2) = H(I2) + 1
570 A(C + 1 − H(I2),I2) = A(C + 1 − H(I1),I1)
580 A (C + 1 − H(I1),I1) = 0
590 H(I1) = H(I1) −1
600 M = M + 1
610 PLOT 1,6,W$ + STR$(M)
620 GOSUB 5000
```

```
630 IF H(3) < > C THEN 520
640 PLOT 1,10,"CONGRATULATIONS"
650 PLOT 1,12,"PRESS ANY KEY"
660 PLOT 1,14,"FOR A NEW GAME"
690 GOTO 120
5000 FOR I = 1 TO C
5010 T$ = Q$(A(I,1) + 1)
5011 R$ = Q$(A(I,2) + 1)
5012 S$ = Q$(A(I,3) + 1)
5015 P$ = T$ + "(2SPC)" + R$ + "(2SPC)" + S$
5030 PLOT 1,19 −(C − I)★2,P$
5035 PLOT 1,19 −(C − I)★2 − 1,P$
5040 NEXT I
5050 RETURN
6000 PLOT 20,7,"TYPE A MOVE(2SPC)"
6010 GET Z$
6020 PLOT 20,7,"(13SPC)"
6025 I1 = ASC(Z$) − 64
6030 GET Z$
6035 I2 = ASC(Z$) − 64
6050 IF I1 = I2 THEN 6070
6060 IF I1 > 0 AND I1 < 4 AND I2 > 0 AND I2 < 4
     THEN RETURN
6070 PLOT 20,7,"INVALID MOVE"
6075 WAIT 200
6080 GOTO 6000
```

Index