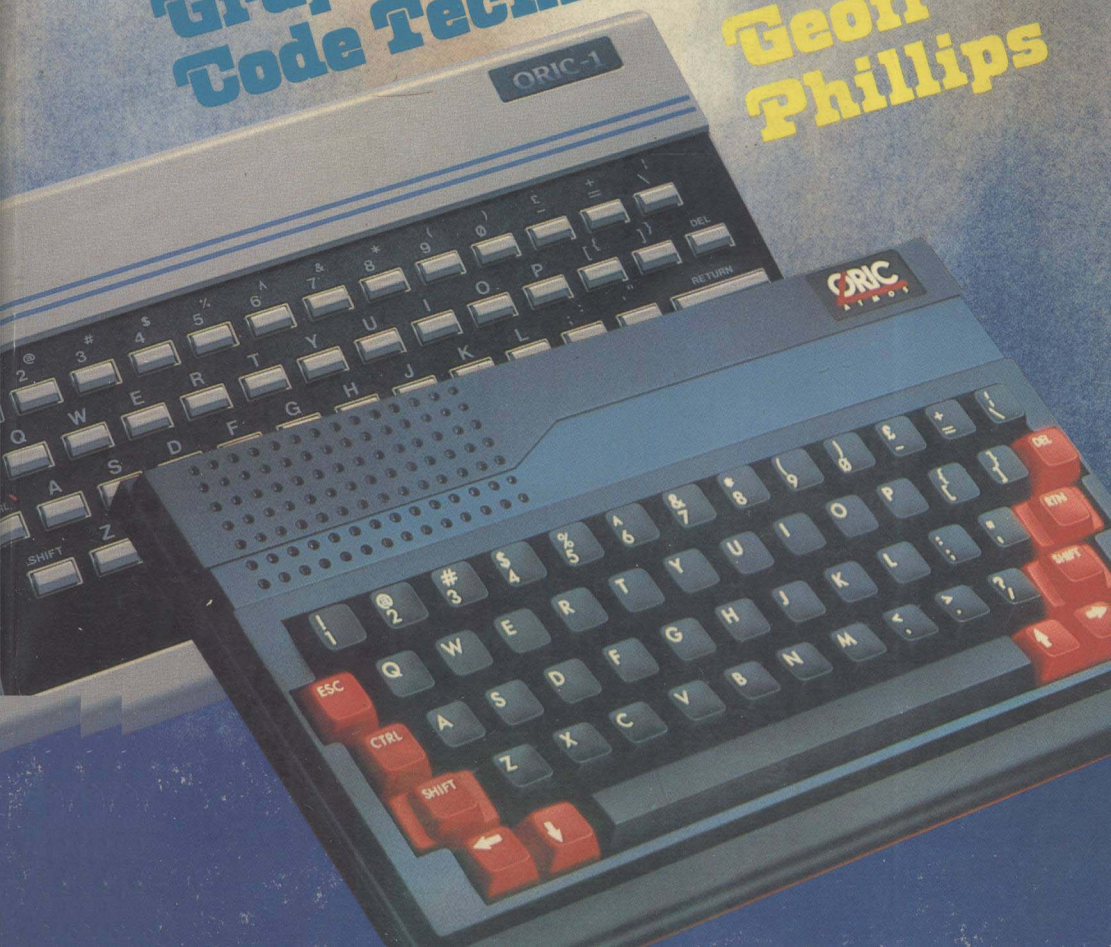
ORIC ATMOS and ORIC 1

Graphics and Machine Code Techniques

Geoff Phillips



Oric Atmos and Oric 1 Graphics and Machine Code Techniques

Geoff Phillips

McGRAW-HILL Book Company (UK) Limited

London · New York · St Louis · San Francisco · Auckland · Bogotá
Guatemala · Hamburg · Johannesburg · Lisbon · Madrid
Mexico · Montreal · New Delhi · Panama · Paris · San Juan
São Paulo · Singapore · Sydney · Tokyo · Toronto

Published by
McGRAW-HILL Book Company (UK) Limited
MAIDENHEAD · BERKSHIRE · ENGLAND

British Library Cataloguing in Publication Data

Phillips, Geoff
Oric Atmos and Oric 1 graphics and machine code techniques.
1. Oric computers—Programming
2. Machine codes (Electronic computers)
I. Title
001.64'24 QA76.8.07

ISBN 0-07-084743-6

Library of Congress Cataloging in Publication Data

Phillips, Geoff
Oric Atmos and Oric 1 graphics and machine code techniques.
1. Oric Atmos (Computer)—Programming
2. Oric 1 (Computer)—Programming
3. Computer graphics 4. Basic (Computer program language)
I. Title
QA76.8.068P48 1984 001.64'43 84-17116

ISBN 0-07-084743-6

12345 CUP8654

Typeset by TC Photo-Typesetters, Maidenhead, England

Printed in Great Britain at the University Press, Cambridge

ORIC ATMOS AND ORIC 1
GRAPHICS AND MACHINE CODE TECHNIQUES

PREFACE

This book is for Atmos and Oric 1 users who want detailed information about their computer. For machine code programmers, an account of the various ROM calls is given with a full description of the methods of handling the different parts of the machine.

This book was not written to teach machine code, but to provide enough background information for existing 6502 programmers to use an Oric/Atmos.

If you are not an experienced machine code programmer, you will still find a great number of hints and tips in the book. Even if you do not understand machine code at all you will still be able to use the numerous utilities – such as Renumber, Merge and Auto.

Chapter one summarizes the hardware that makes up an Oric or Atmos computer.

Chapter two explains how BASIC works, from the way that programs and variables are stored, to creating different windows of scrolling text. A list of Oric 1 and Atmos bugs concludes the chapter.

Chapter three is about how machine code programs are entered, methods of calling your program, and how a machine code program can use the software timers. Some machine code pitfalls and tips are given at the end of the chapter, along with a real-time clock program.

Chapter four describes two important sections of Oric 1 or Atmos – the keyboard and the cassette system.

This chapter describes how individual keypresses are detected – very useful for games where several keys are used at the same time. A complete account of the cassette system is given, and after reading this chapter you will be able to write machine code programs that save and load blocks of memory, or individual bytes. A verify program is listed for Oric 1 owners.

Chapter five gives an account of how BASIC uses RAM and ROM. All important ROM and RAM addresses are printed, plus details of how the stack area is used.

Chapter six explores three important subjects – maths, HIRES and music. On the maths side, a machine code programmer will now be able to use the ROM's floating point routines. On the HIRES side, you will find out how the high-resolution graphics can be used with

different mixtures of text, and a complete account of the ROM routines for CURSET, DRAW etc is given.

On the music side, this chapter describes how the ROM routines for MUSIC, PLAY and SOUND are used, as well as giving details of how the sound chip is accessed.

Chapter seven presents a number of fast high-resolution graphics routines. A single-point plotter is given which runs about 70 times faster than BASIC's CURSET command. A PAINT routine is listed that will fill in any shape on the high-resolution screen.

Chapter eight gives six utility programs to help BASIC programmers. These are: Renumber, Delete, Merge, Auto-Data, Trace, and ON-ERROR. Other utilities can be found throughout the book.

Chapter nine completes the book with some ambitious ideas, including a primitive form of speech synthesis, a multiprocessor and a program that allows single key entry of BASIC keywords.

Geoff Phillips

CONTENTS

1 LOOKING INSIDE THE ORIC

1.1 Introduction

In this chapter we shall look at the various components of the Oric. Some of the features discussed will be further explored later in this book – the workings of the cassette system, for example.

1.2 The ROM

The Read Only Memory device contained in each Oric is responsible for supplying the BASIC interpreter program. It contains some 16K of instructions located between #C000 and #FFFF (on all machines). Since a program cannot overwrite the ROM area, the area #C000 to #FFFF is not affected by any write operations.

1.3 Use of RAM

Any BASIC program that you write is stored in the Random Access Memory located between 0 and #BFFF (or up to #3FFF for 16K users).

However, since the ROM needs a certain amount of working space, and because of other considerations, any BASIC program that you write will start at #501. The top of usable memory for your BASIC program is also going to be reduced, to at least as low as #B3FF, or #33FF for 16K machines.

If you are using high-resolution mode and have not issued a GRAB command, then the top of BASIC memory becomes #97FF (#17FF for 16K machines). This means that you have lost more than 11K! The actual layout and use of the RAM is described in more detail in Chapter 5.

1.4 Differences between machines

From the point of view of hardware, there are very few differences between machines.

There are two major categories:

1. Your Oric is either a 16K or a 48K machine.

2. Your Oric is either version 1.0 (i.e., the ORIC-1) or 1.1 (i.e., the ORIC ATMOS).

When you first power up your Oric, you will be advised of which version you are running. Chapter 2 lists the differences between the ROMs, but there is no apparent difference when looking at the hardware. Take note of your version number, so that you know which addresses apply to your particular machine.

The terms '16K machine' and '48K machine' relate to the total memory capacity. On a 16K machine, there would seem to be a gap between the end of the RAM (#3FFF) and the start of ROM (#C000). In practice, this is not the case, as the 16K of RAM is mirrored through each 16K block of addresses; e.g., location 0 is the same as locations #4000 and #8000. This is the reason why a program can still write to the screen at #BB80 on a 16K machine. Do not worry that this feature is 'accidental' and might not be true for all ORICs – the start-up routines use the mirroring to detect which machine is which.

Some very early machines have slightly different insides, but the only important difference is that the sound on these machines is much louder and can cause the break-up of a TV picture.

In this book, version 1.0 addresses are given first, followed by the version 1.1 address in brackets.

1.5 The microprocessor – 6502

The 6502 is the heart of the computer, obeying instructions held in ROM or RAM. When writing BASIC programs, the function of the 6502 is entirely invisible, but if you are going to write machine code programs, you will need to know quite a lot about this device. It is certainly worth while buying a book devoted to the subject. The programs in this book will help you to understand some aspects of machine code programming, and part of Chapter 3 gives a few guidelines on the use of some 6502 instructions.

1.6 The 6522 – VIA

The Versatile Interface Adaptor (VIA) is a microchip that belongs to the same family as the 6502 processor (hence the similar number). It is a complicated, but invaluable, device which links the Oric's 6502 to its peripherals, as well as providing two timers.

A book devoted to the 6502 will often have a chapter on the usage of the 6522; here we are only concerned with its use in connection with the Oric.

TALKING TO THE VIA

The 6522 chip is linked to page 3 of your memory map, so that whenever you read or write to an address between #300 and #30F you are enabling the VIA. These 16 addresses are normally mirrored throughout page 3 – so #380 is the same as #300 – but there is no reason to use any location between #310 and #3FF.

A quick summary of these locations follows; for more information you will need to use a book on the 6502 family of chips.

Address	Description
#300	Port B in and out
#301	Port A in and out
#302	Define port B output or input (output if bits set)
#303	Define port A output or input
#304,5	Timer-1 counter
#306,7	Timer-1 latch
#308,9	Timer-2 counter/latch
#30A	Shift register (not used by ORIC)
#30B	Auxiliary control register
#30C	Peripheral control register
#30D	Interrupt flag register
#30E	Interrupt enable register (indicates what sort of event will cause an interrupt).
#30F	Read/write to port A without handshake.

CONTROL LINES ON THE 6522

The two ports can each contain one byte of information, but each bit can be separately set as input or output. In addition to the ports, there are four control lines, called CA1, CA2, CB1, and CB2. Here is a summary of how the Oric uses all of the I/O lines:

Port A – connects to the printer's 8-bit bus. It is also wired into the 8912 sound chip.

Port B – this port is easier to look at bit by bit. Starting from the right, the lowest three bits are used to supply the row when looking at the keyboard (see Sec. 1.10).
Bit 3 of port B is set to 1 when a key is pressed – more on this later.
Bit 4 is connected to the strobe line on the printer socket – when 0 the printer will expect data to be present on port A.
Bit 6 controls the relay circuit on the cassette socket.
Bit 7 connects to the cassette output circuitry.

CA1 – this line is input from the acknowledge signal on the printer port.

CA2 – this line connects to the 8912 sound chip (see Sec. 1.7).

CB1 – this line is connected to the cassette input circuitry.
CB2 – when 1 the 8912 reads from port A of the 6522.

THE 6522 TIMERS

It is not often realized that the Oric has two versatile timers at its disposal. Later in this book it will be shown how easy it is to use these timers to provide a real-time clock facility – in BASIC or machine code.

The most important timer is designated timer-1 and is used mainly to count time between each interrupt. Without any supervision from the 6502, timer-1 counts down from a given 16-bit value (at location #306,7) to zero – this counter can be read from addresses #304,5.

When zero is reached, timer-1 starts counting again, using the 16-bit value stored at #306,7, and bit 6 in the interrupt flag register is set. When this happens, the 6522 will cause an interrupt signal to be sent to the 6502 processor. If the 6502 has interrupts enabled, then the appropriate interrupt handling subroutine will be called. It is very important to realize that the timer will operate regardless of the state of the 6502 – disabling interrupts does not stop the clock.

During cassette saving and loading, the VIA is set up differently, and the timers operate in a different fashion:

Timer-2, which is idle at other times, is used when receiving bits from the cassette input port in order to wait an exact amount of time.

The function of timer-1 is altered (by setting bits 6 and 7 of the auxiliary control register) so that instead of causing an interrupt bit 7 of port B is toggled and the timer is automatically set running again.

When a cassette operation is complete, the registers in the 6522 are set back to their initial values in order for the keyboard and printer to work normally.

1.7 The 8912 sound chip

All sound effects produced on the Oric are performed by the 8912 sound chip. In addition to being able to generate music, this device has one input/output port – port A – which is used to output the column number when polling the keyboard.

The 8912 is controlled by 15 eight-bit registers stored inside the chip. These are set up whenever the Oric's sound commands are executed. Here is a summary of how each register is used:

Register	Use
0,1	The lowest 12 bits give the pitch of channel A
2,3	The lowest 12 bits give the pitch of channel B
4,5	The lowest 12 bits give the pitch of channel C
6	The lowest 5 bits give the pitch of the noise channel
7	Enables: each bit has a different meaning: Bit 6: set port A as output or input Bits 3,4,5: mix noise with channels A, B, and C Bits 0,1,2: enable channels A, B, and C
8	Channel A amplitude. If bit 4 is set then the music 'envelope' is used; otherwise bits 0 to 3 give the fixed volume
9	Channel B as above
10	Channel C as above
11,12	Length of the envelope
13	The lowest four bits give the shape of the envelope. This is different from the value you would use in the PLAY command, according to the following table:
14	Register 14 is the I/O port A.

PLAY value	Actual register value
1	0,1,2,3, or 9
2	4,5,6,7, or 15
3	8
4	10 or 14
5	11
6	12
7	13

The 8912 registers cannot be accessed directly by the 6502 – but instead via port A of the 6522 and a couple of control lines.

CB2 or the 6522 is set in order to select the 8912, and then immediately cleared (the 8912 chip will accept data as fast as you send it).

CA2 of the 6522 is either set when a register number is being passed in port A or cleared if it is data for the register. So in order to write #F7 to register 1, you would:

1. Store 1 in #30F – port A without any handshake signals.
2. Set CA2 and CB2.
3. Immediately clear CB2.
4. Put #F7 in #30F.
5. Clear CA2 and set CB2.
6. Immediately clear CB2.

There is a subroutine in the ROM to handle the above procedure –

see Chapter 6 – but as this is unbelievably inefficient, you will find a faster version used in the speech synthesis program of Chapter 9 (see 9.2).

1.8 Text screen

The text screen is organized as 28 rows by 40 columns of character cells. Each character cell occupies one byte of memory between #BB80 and #BFDF, but creates a display of a character 6 pixels wide by 8 pixels down.

The information for each character is retrieved by the graphics chip depending on the ASCII value of that character. Eight consecutive bytes are used for each ASCII character – one for each line of pixels. The formula for the start address of the definition of a particular character is: character value * 8 + start of character set.

The start of the character set (in TEXT mode) is #B400 for the standard character set (A to Z, 0 to 9, etc.) and #B800 for the alternate character set. Since these are in an area of RAM, it is quite a simple matter to redefine any character.

ATTRIBUTES

If the screen memory contains a control value, i.e., an ASCII value between 0 and 31, then this value is taken as an attribute, and the character set is not referenced. This means that the first 256 bytes of both the standard and alternate character sets is wasted. Also, you may have noticed that the alternate character set overlaps with the screen!

An attribute changes the way that a particular line is interpreted by the VDU chip. Appendix C of the Oric manual (or Appendix 2 of the Atmos manual) gives the 32 possible attribute values. Some attributes – those between 8 and 15 – affect three different features of a line – double height, flashing, and character set.

At the beginning of each line, five attributes are always assumed:

1. No flashing.
2. Standard character set.
3. Paper of 0 (black).
4. Ink of 7 (white).
5. Single height.

If the character being displayed has a value between 128 and 255, then the character will be used as though 128 had been subtracted – except that whatever colours would have been displayed become inverted.

For instance, if you POKE #BB80 with 65 – i.e., the letter A – you will get a white A on a black background. If you POKE 65+128 instead, then the colours change – white (7) becomes 0 (7−7) and black (0) becomes white (7−0). This rule also works when you are setting a paper attribute: POKE #BB80,17 leaves a red square at the top left of the screen, whereas POKE #BB80,17+128 – although correctly setting the paper colour to red – creates a square which is cyan (7−1).

USING ESCAPE

One source of confusion lies when looking at how PRINT uses ESC (CHR$(27)) in order to set attributes. It is a good idea to totally ignore what the manual tells you about using ESCAPE when writing in machine code.

The important fact is that ESCAPE only works because BASIC is creating all the attributes for you – POKE 27 onto the text screen and nothing will happen. (Try poking it onto the high-resolution screen!)

The use of ESCAPE when using PRINT is unavoidable because this command traps any ASCII value less than 32 and treats them like control characters (changing parameters like keyclick, etc.).

The PLOT command, like POKE, does not understand escape sequences, so direct attributes must be used.

1.9 High-resolution mode

High-resolution mode moves away from using a character set, and instead causes the screen to directly reflect the contents of the video memory.

Each byte in the area #A000 to #BF3F affects 6 horizontal pixels in a matrix 240 across by 200 down. Part of the text screen remains at the bottom, at addresses #BF68 to #BFDF, although in high-resolution mode, these three lines use the character sets at #9800 to #9FFF.

This is necessary as the high-resolution screen overwrites the normal character set area. The exact details of how BASIC enters into high-resolution mode can be found in Chapter 6.

From a hardware point of view, the graphics chip switches modes when an attribute of 30 or 31 is interpreted. When an attribute of 26 or 27 is encountered, the mode is switched back to text. All the copying of character sets, etc., is carried out by software.

1.10 Keyboard

The keyboard on the Oric is a matrix of 8 columns connecting to 8 rows. By writing down one column and along one row, it is possible to examine the state of an individual key.

After every three interrupts, the system scans all the columns and rows in an attempt to find any depressed keys. The ROM subroutine only looks for one key down at a time, except that it does one extra search for the SHIFT and CONTROL keys. There is no reason, however, why a program cannot look at every key individually – this is very useful for games.

Two ports are used to poll the keyboard – port B of the 6522 and port A of the 8912.

The column is output on port A of the 8912 in the form of one bit cleared in a byte containing #FF. The row is output as a number (0–7) on port B of the 6522 and bit 3 of port B read back to determine whether that key is pressed.

Chapter 4 gives further details about reading from the keyboard.

1.11 Printer interface

When a byte is sent to the printer, the following occurs:

1. The byte is sent along port A of the 6522.
2. Bit 4 of port B is cleared and then set (this is the printer's strobe line).
3. The Oric waits until CA1 is pulsed by the acknowledge line on the printer. CA1 is not read directly, but causes bit 1 of the interrupt flag register to be set inside the 6522.

1.12 Cassette system

Chapter 4 explains how your programs can use the cassette system to save and load data, so this section is only concerned with some of the hardware aspects.

CASSETTE OUTPUT

The cassette output circuitry can only handle one bit at a time, creating either a high tone or a low tone depending on bit 7 of the 6522's port B.

This bit is set or reset automatically by the 6522 after timer-1 has finished counting down; the length of time to be counted depends on both the tape speed and whether the bit is 0 or 1. In order to save a whole byte, the cassette routines use a series of eight shift instruc-