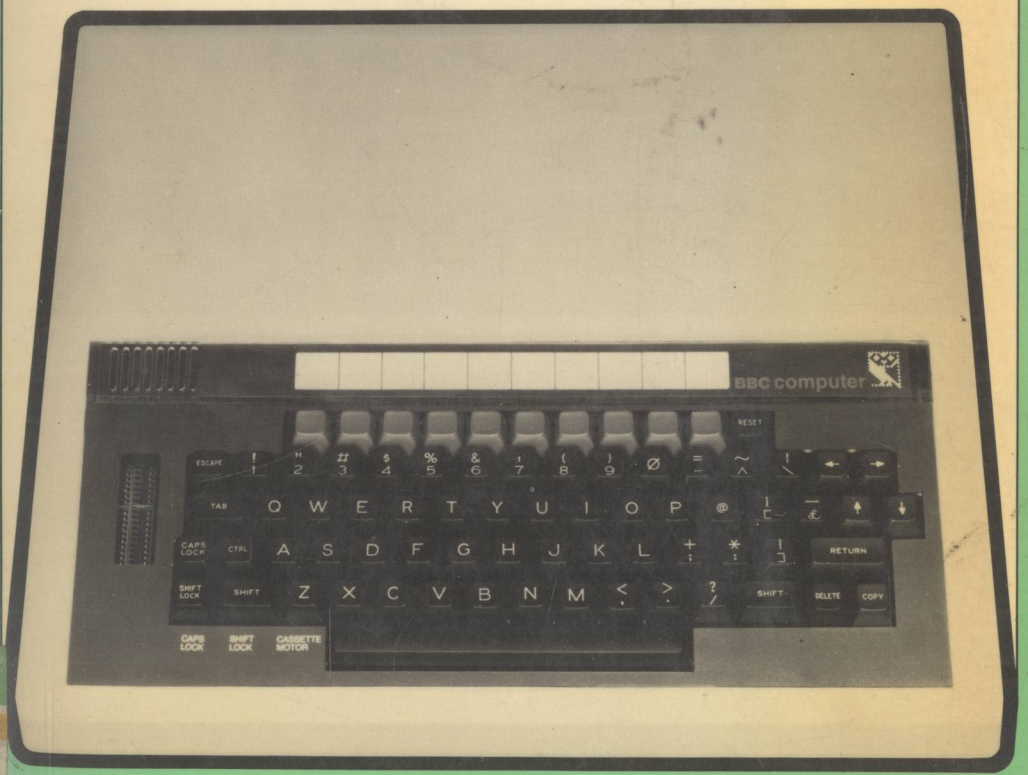
ASSEMBLY LANGUAGE PROGRAMMING for the BBC MICROCOMPUTER



Ian Birnbaum

Assembly Language Programming for the BBC Microcomputer

Ian Birnbaum

First edition 1982
Reprinted (with corrections) 1982, 1983 (twice)

Published by
THE MACMILLAN PRESS LTD.
London and Basingstoke
Companies and representatives
throughout the world

Printed in Great Britain by Unwin Brothers Limited,
The Gresham Press, Old Woking, Surrey.

ISBN 0 333 34585 1

Dedicated to Theresa

Assembly Language Programming for the BBC Microcomputer

Macmillan Computing Books

Advanced Graphics with the Sinclair ZX Spectrum
Ian O. Angell and Brian J. Jones

Advanced Graphics with the BBC Microcomputer
Ian O. Angell and Brian J. Jones

Assembly Language Programming for the BBC Microcomputer
Ian Birnbaum

Advanced Programming for the 16K ZX81 Mike Costello

Microprocessors and Microcomputers – their use and programming
Eric Huggins

The Alien, Numbereater, and Other Programs for Personal Computers – with notes on how they were written John Race

Beginning BASIC Peter Gosling

Continuing BASIC Peter Gosling

Program Your Microcomputer in BASIC Peter Gosling

Practical BASIC Programming Peter Gosling

The Sinclair ZX81 – Programming for Real Applications Randle Hurley

More Real Applications for the Spectrum and ZX81 Randle Hurley

Assembly Language Assembled – for the Sinclair ZX81 Tony Woods

Digital Techniques Noel Morris

Microprocessor and Microcomputer Technology Noel Morris

Understanding Microprocessors B. S. Walker

Codes for Computers and Microprocessors P. Gosling and Q. Laarhoven

Z80 Assembly Language Programming for Students Roger Hutty

Preface

Every BBC Microcomputer, whether Model A or Model B, comes equipped with an immensely powerful and very fast assembler. What is more, assembly language statements and BASIC statements can be freely mixed, hugely increasing the programmer's potential control over the machine.

This book shows you how to establish that control. It assumes that you are proficient in BASIC, for if you are not this is probably not the best time to learn assembly language. But it assumes no knowledge of assembler at all, taking you step by step from the basics to their complex implementation.

Since every user of the BBC Microcomputer assembler will have a working knowledge of BASIC, it is possible to use that knowledge to motivate and illustrate the ideas in assembly language. This book takes that approach, and this should help you to master assembly code, for you will always be acquainted with the fundamental concepts by seeing their connection with BASIC.

I had three types of readers in mind in writing this book. Firstly all current owners of BBC Microcomputers who want to extend their knowledge into machine code. To help them with self-instruction, this book contains a considerable number of exercises and *a full solution is provided for every one.* Secondly, the teacher or student of Computer Science who wants to use this text in a structured course. The book is the result of many years' teaching experience, and it is designed according to a teaching strategy which the author has found to be very successful.

And thirdly, those people, already experienced BASIC programmers, who are wondering whether to buy the BBC Microcomputer, when there seems so much competition from cheaper and seemingly comparable computers. This book should help to convince them that the BBC Micro is worth the extra expense. Quite apart from its superb BASIC, the Micro possesses an assembler which turns it into a potential 6502 development system in its own right! It also possesses an operating system which is designed to mesh with assembly language programming in an extraordinarily simple way. One of the aims of this book is to show you how to exploit these features to the full.

The book contains 73 listings of programs, many of which will be found to be useful utilities in their own right, quite apart from their value in teaching you assembly language. In particular

it contains a full machine code monitor, a suite of machine code sorting programs which you can use on BASIC variables, a high resolution screen copy to the Epson printer and a program compactor. There are two companion tapes available with the book if you do not feel you want to type in the programs yourself. Each tape also contains two extra programs: the first, a universal graph plotter and a 6502 disassembler which displays in standard 6502 mnemonics. The second contains two utilities for programming: a machine code program which will find the locations of any segment of code in a BASIC program (equivalent to the FIND command found in some utility packages); and another machine code program which will replace any segment of code in a BASIC program by any other segment; so, for example, you can change any variable's name in the whole program in an instant.

The book is completely self-contained: full information on the 6502 instruction set is provided throughout and summarised in an Appendix. Other Appendices cover floating point and the user port, and there is a section on combining programs in the BBC Computer using PAGE and *LOAD.

Thanks are due to Mrs Barry who typed up a very untidy manuscript quickly and efficiently, and with very few errors.

Ian Birnbaum
Needingworth
May 1982

Assembly Language Programming for the BBC Microcomputer

Two software cassettes are available to accompany this book. They cost £9.00 each, if bought separately, or £16.00 for both when ordered together.

TAPE 1

Contains all the listings in Chapters 2 to 9
Plus these two extra programs

GRAPHPLOT which draws up to two graphs in the highest resolution available on your computer. (If you have an Epson printer, you can also obtain a hard copy of the graphs)

and

DISASSEMBLER which will translate any section of machine code back into standard 6502 mnemonics. This program is written entirely in BASIC, so it can be loaded into any page.

Cassette 1 ISBN 0 333 34587 8 £9.00 (inc. VAT)

TAPE 2

Contains all the listings in Chapter 10 and in the Answer section
Plus these two extra programs

FINDCODE which will locate any section of code in a program and display all the lines containing that code

and

REPLACE which will locate any section of code and replace it by any other

Cassette 2 ISBN 0 333 35016 2 £9.00 (inc. VAT)

These cassettes are available through all major bookshops ... but in case of difficulty order direct from

Globe Book Services
Canada Road
Byfleet
Surrey KT14 7JL

£9.00 each
or
£16.00 for the two

Contents

Chapter 1 Preliminary ideas

1.1 WHAT IS A COMPUTER?

In its simplest form, a computer can be considered in four sections: *input, output, microprocessor, internal memory.* Figure 1.1 shows the interrelationship:

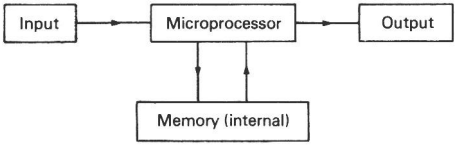


Figure 1.1: Simplified diagram of a computer

Most *input* is through the keyboard, but other input devices and channels include: casette tape system, disc system and sensor devices connected to an input port.

Most *output* is through the TV or monitor (the VDU), but other output devices and channels include: casette tape system, disc system, printer and control devices connected to an output port.

Notice that the cassette and disc systems are both input and output devices. These are sometimes referred to as backing store. Printers and suchlike are sometimes referred to as peripherals, things outside the main system.

All input must pass through the *microprocessor,* at least at some time or another. It may reside temporarily in some internal memory (often referred to in this context as a buffer or latch), but the microprocessor will deal with it when it can. Similarly all output will be directed by the microprocessor and all will pass through it. The microprocessor is the 'brain' of the system, and this book is concerned with how to program it directly. There are also other 'lesser brains' to be found in a computer, but these are not usually under our control and correspond roughly, following the metaphor, to the autonomic nervous system.

Internal memory can be divided into two parts: *ROM* (read only memory) and *RAM* (misleadingly called random access memory - read/write memory is a better name). ROM contains information which is fixed: the microprocessor is unable to modify it. Usually it contains instructions and data which the microprocessor will need in the same form when a specific task is demanded. In particular, in the BBC micro it contains the Operating System (OS) and the BASIC Interpreter. A considerable advantage of ROM is that information in it does not disappear when the computer is switched off.

Information in RAM will, by contrast, disappear if power is removed. It has the advantage, however, of allowing information to be modified. It is in RAM that all the programs and data we input will reside.

Some RAM is given a special function. Some, for example, will be used by the microprocessor as a sort of 'scratch pad' (this is called the stack and is covered in Chapter 9). Other parts are reserved for the OS or the BASIC Interpreter in which to store results and information. And some parts are connected to input/output channels. RAM used in this way is often referred to as *memory mapped*. For example, the BBC micro has a memory-mapped VDU, each character on the screen corresponding to a specific portion of memory which is fixed for each graphics mode chosen (this is not strictly true, as we shall see in Chapter 9).

There are some specialised chips in the computer that act as RAM, though they are not usually referred to in this way. The best example of these are the input/output chips, going under a variety of names (*PIO* - programmable input/output, *PIA* - peripheral interface adapter, *VIA* - versatile interface adapter). PIO is the most descriptive: the chip consists of a series of memory locations in which data passing in or out can be latched; certain locations contain information on whether a particular channel is to be conceived as input or output, and this information can be changed by the microprocessor. Although these chips contain more than just memory (for example some contain a timer, results of which are available at a specific location), they are most conveniently thought of as RAM because they are *addressable*, an idea to which we now turn.

1.2 HOW IS MEMORY ORGANISED IN A COMPUTER?

If we want to refer to a specific location of memory, it will need to have a name. Since we may want to refer to any part of memory at some time, we must make sure that memory is organised in such a way that every location has a name and that this name is unique.

When computers are built they are wired up in such a way that the microprocessor can refer to a specific location by outputting a series of pulses called *an address*. The set of wires through which they pass is called the *address bus*.

Since each wire will either have a pulse or not have a pulse

there are two states which we may label 1 (pulse) and 0 (no pulse). A microprocessor is a *digital device* because it always understands things and communicates with other parts of the computer in this *two-state* way. Thus: a switch is either on or off; an element either has a positive or negative field etc.

The microprocessor in the BBC micro can accept 16 wires on its address bus. Since each wire can either be 1 or 0 this gives a total of 2^{16} = 65536 addresses, that is from 0000000000000000 to 1111111111111111. Hence there can be at most 65536 locations of memory.

Now writing 16 ones and/or zeros like this is very hard to read, and so we will adopt a notation that makes it easier. We will divide our 16 digits into four groups of four. So for example in 0111101101000010 the four groups are:

0111 1011 0100 0010

Now each group of four can have one of sixteen different forms, from 0000 to 1111 (2^4). We can use the numerals 0 to 9 for the first ten; after that we will use A, B, C, D, E, F. Table 1.1 gives the details. In that table we can conceive of the four digits on the left as the display on a rather odd car odometer (the meter measuring the distance covered). Each cog on the dial has just two numerals, 0 and 1. As the cog revolves through one revolution it pushes the next one on half a turn. In this way, the first sixteen numbers (0-15, decimal) are generated in the order shown.

Table 1.1: Relationship between binary and hexadecimal

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Using this notation we can write the number above as 7B42. The number 0010010101000000 will be written 2540, but this is rather misleading because it looks like two thousand five hundred and forty, which it isn't. So to prevent confusion, we precede these hexadecimal numbers, as they are called, by the ampersand sign (&). Thus we write them as &7B42 and &2540 (some computers use the dollar sign, $, but not the BBC micro).

You will notice in Table 1.1 the terms *binary* and *hexadecimal*. Binary means two (there are two possible numerals, 0 and 1) and hexadecimal means sixteen (there are sixteen possible numerals 0 to F). In the same way decimal, our usual system of representing numbers, means ten (numerals 0 to 9).

Many books spend ages explaining how to convert from one system to another, but this is a complete waste of time. Your BBC computer will do it for you.

For example, type into your computer

```
P. &7B42
```

and see what you get. This is the decimal equivalent of &7B42.

Similarly, type in

```
P. ~14321
```

and see what you get. This is the hexadecimal equivalent of 14321.

If you wish you can write a program to convert either way but it is hardly worth it; you might as well operate in direct mode, as above. It is worth experimenting a little with various numbers to see the equivalence operating for yourself.

— || —

It should be clear to you by now why it is convenient for the computer to work in binary and why it is convenient for us to work in hex (the usual abbreviation for hexadecimal). From now on we will think of all the memory locations in terms of hex.

Now in order to make the wiring as simple as possible, a simplifying concept called *pageing* is used. The 65536 addresses of memory are conceived as a series of *pages*. The page number is given by the top two digits of the hexadecimal number; the bottom two digits give the location in that page. The best image is that of a book with 256 pages, each page having 256 lines, each line being a memory location. Rather eccentrically, the book's first page is labelled zero, and is called *zero page*. It is a very important area of memory as we shall see in the next chapter. Figure 1.2 should make the idea clear. Thus address &F1B2 refers to location 178 in page 241; that is, address number 61866. We can think of this in the following way: the high byte &F1 accesses all

the locations in page 241; then the low byte &B2 picks out a particular location in that page, location 178. The top half of the address bus is thus wired up to access pages, and the lower half to access one of 256 locations in any page.

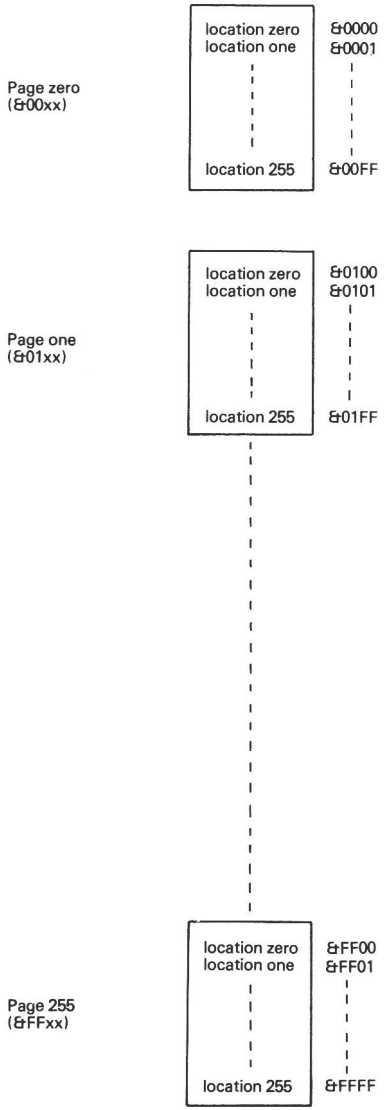


Figure 1.2: Pageing

When referring to memory en masse it is conventional to work in units of 4 pages and refer to this as a *K of memory* locations. Thus the microprocessor in the BBC micro can address 64K of memory locations; your machine will either have 16K of RAM (model A) or 32K of RAM (model B).