

Penguin Books

Creative Assembler
How To Write Arcade Games

As one of Acornsoft's games programmers, Jonathan Griffiths was responsible for such widely popular programs as Snapper and JCB Digger for the BBC Microcomputer Model B and Acorn Electron. He has prepared a cassette program, which can be used in conjunction with this book, available from Acornsoft Limited, c/o Vector Marketing Ltd, Denington Industrial Estate, Wellingborough, Northants NN8 2RL.

Acknowledgements

Thanks are due to David Johnson-Davies for providing an earlier version of the first few chapters, Jim Dobson and Jeremy Bennett for helping in the writing of this book, and to Philippa Bush and Sharron Fellows for editing it. Also, thanks are due to Orlando M. Pilchard (Q.C.), Chris Jordan, Paul Hudson, Peter Cockerell, Dominic Verity, Jeremy San, Robert Macmillan, Paul Fellows, John Collins, Simon Hughes and Mark Holmes for proof reading.

This book was written and prepared on a BBC Microcomputer Model B using the VIEW word processor.

This book is dedicated to all the staff at Acornsoft.

CREATIVE ASSEMBLER

How To Write Arcade Games

Jonathan Griffiths



Penguin Books

The Penguin Acorn Computer Library is a joint venture, produced by Acornsoft Limited (in association with Pilot Productions Limited), and published by Penguin Books Limited.

Penguin Books Ltd, Harmondsworth, Middlesex, England

Penguin Books, 40 West 23rd Street, New York, New York, 10010, U.S.A.

Penguin Books Australia Ltd, Ringwood, Victoria, Australia

Penguin Books Canada Ltd, 2801 John Street, Markham, Ontario, Canada L3R 1T

Penguin Books (N.Z.) Ltd, 182-190 Wairau Road, Auckland 10, New Zealand

First published 1984

Copyright © Acornsoft Limited, 1984

All rights reserved

Set in Palatino by Repro Graphics, 61 Cromwell Road, Southampton

Colour origination by RCS Graphics Ltd, 39-40 Springfield Mills, Farsley, Pudsey, Leeds, and
MRM Graphics, 61 Station Road, Winslow, Bucks

Made and printed in Spain by Printer industria grafica s.a., Sant Vicenc dels Horts, Barcelona
D.L.B. 1517O-1984

Line illustrations by Rob Shone

Original photography by Nick Wright

Except in the United States of America, this book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out, or otherwise circulated without the publisher's consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser

CONTENTS

INTRODUCTION	9
--------------	---

SECTION 1

1 DATA STORAGE	15
----------------	----

- 1.1 Hexadecimal notation
- 1.2 Binary notation and bits
- 1.3 Memory locations and bytes
- 1.4 More about memory locations
- 1.5 Negative numbers - two's complement
- 1.6 Storing text

2 CARRYING OUT INSTRUCTIONS	25
-----------------------------	----

- 2.1 The CPU
- 2.2 Machine code v assembler
- 2.3 The accumulator and the carry flag
- 2.4 Writing an assembler program
- 2.5 Executing a machine-code program
- 2.6 Adding two-byte numbers
- 2.7 Subtraction
- 2.8 Comments
- 2.9 Printing a character
- 2.10 Immediate addressing
- 2.11 Using addresses

3 JUMPS, BRANCHES AND LOOPS	39
-----------------------------	----

- 3.1 Jumps
- 3.2 The zero and negative flags
- 3.3 Conditional branches
- 3.4 Two pass assembly
- 3.5 X and Y registers
- 3.6 Iterative loops
- 3.7 Comparing values
- 3.8 Using the control variable
- 3.9 Conditional assembly

4	LOGICAL OPERATIONS, SHIFTS AND ROTATES	53
4.1	Logical operations	
4.2	The BIT instruction	
4.3	Rotates and shifts	
5	ADDRESSING MODES	59
5.1	Indexed addressing	
5.2	String types	
5.3	Summary of addressing modes	
6	THE STACK	69
6.1	Using the stack	
6.2	How the CPU stores addresses	
6.3	Recursion	
SECTION 2		
7	MACROS	79
7.1	Generating and calling a macro	
7.2	Macro parameters	
7.3	Conditional assembly in macros	
7.4	Labels in macros	
8	BASIC I, BASIC II and ELECTRON BASIC	85
8.1	Distinguishing BASIC I from BASIC II	
8.2	The main differences between BASIC I and BASIC II	
8.3	BASIC I versions of EQUW, EQUW, EQUW and EQUW	
8.4	BASIC I version of OSCLI	
9	OPERATING SYSTEM ROUTINES AND SPECIAL EFFECTS	89
9.1	OSBYTE and OSWORD	
9.2	Revectoring operating system routines	
9.3	Screen scrolling	
9.4	Palette handling	
9.5	Interrupts, events and BREAKS	

10	LARGE ASSEMBLER PROGRAMS	105
10.1	Source files and the master compiler program	
10.2	Saving source files	
10.3	Macro source files	
10.4	Initialisation file	
11	PROGRAM STRUCTURE	113
11.1	Where to start	
11.2	Self-documenting code	
11.3	Parameters	
11.4	Size of routines	
11.5	Conditional assembly in an aid to debugging	
11.6	Lower case variable names	
11.7	Constants	
11.8	Lookup tables	
11.9	Use of absolute addresses	
SECTION 3		
12	UTILITIES FOR ASSEMBLER PROGRAMS	123
12.1	Input/output	
12.2	Analogue to digital routines	
12.3	Numerical routines	
12.4	Miscellaneous	
12.5	General purpose macros	
12.6	BASIC routines for use with assembler	
13	GRAPHICS	145
13.1	Shape designer - DESIGN	
13.2	Plotting a shape on the screen	
14	EXAMPLE GAME	159
APPENDIX A		171
APPENDIX B		177



INTRODUCTION

The BASIC assembler which is available on the BBC Microcomputer and Acorn Electron is a very powerful tool for programmers. It provides a comprehensible interface between the programmer and the machine code language which the 6502 processor itself uses. Hence the programmer is able to control the machine more directly using assembler.

The main reason why people write programs in assembler rather than BASIC is probably because of the speed difference between the two. Assembler instructions can be executed extremely quickly, a program written in BASIC will take between 10 and 100 times as long. Hence assembler is particularly useful for games' programmers since it enables them to move missiles and creatures across the screen quickly and smoothly. If BASIC was used to calculate the new coordinates of each object and draw them at those positions then movement would tend to occur in jerky leaps.

However, speed is not the only factor to be taken into consideration. Assembler programming gives you more power to solve a problem than BASIC does. All high-level languages require programs to have a certain structure and this puts constraints on programs written in that language.



Sceptics may advise against using assembler on the grounds that it is too complex. It is true that operations such as multiplication and division which are easy to perform in BASIC are not as straightforward in assembler. For what might be considered a trivial task, for example multiplying a number by three, several assembler instructions are required instead of just a single BASIC one. A further problem is that there are no FOR ... NEXT or REPEAT ... UNTIL loops in assembler; if you require a loop you must set one up yourself. The same applies to floating point arithmetic -- assembler only supports integer calculations.

'Rocket Raid' — an excellent example of the use of the assembler.

My advice is that you ignore these sceptics. It isn't difficult to learn to program in assembler. The programs look much less like English than BASIC I

ones do but nevertheless to someone who knows the language they are easy to understand. Like learning to do anything else, all that is required is a certain amount of knowledge and a lot of practice. This book has been written to provide the knowledge -- the rest is up to you.

The book is divided into three sections, each of which has a different task to perform. The first part aims to introduce the more useful assembler instructions available for the 6502 processor, giving simple examples of how they can be used. The second part introduces some of the more complex programming techniques which are aimed in particular at people writing large assembler programs. The third part is aimed mainly at the games' programmer. It provides many useful routines and finally shows how these can all be linked together to produce a complete game.