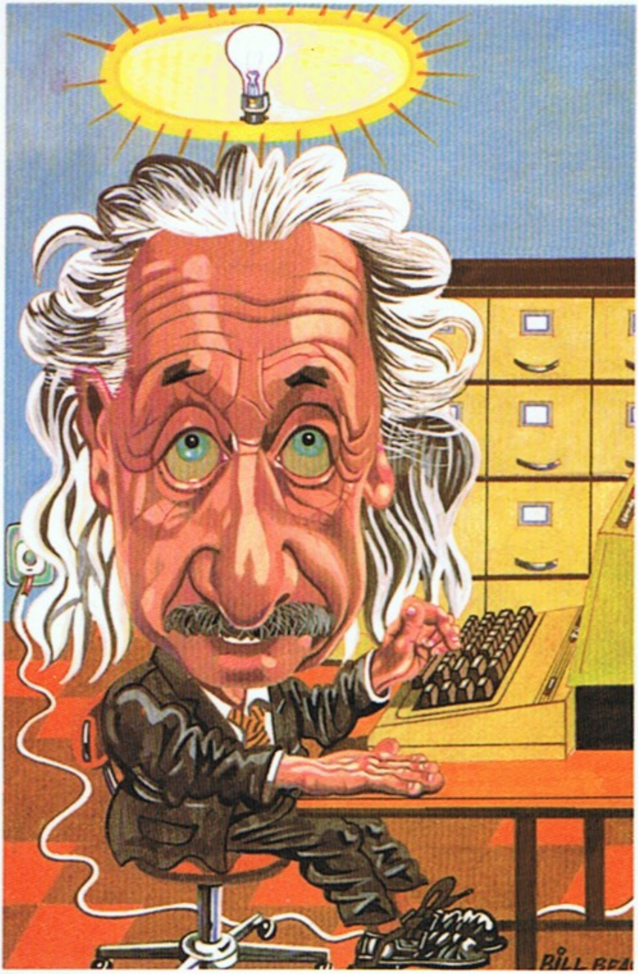
IMPOSSIBLE ROUTINES



FOR THE COMMODORE 64

KEVIN BERGIN

Impossible Routines for the Commodore 64

Kevin Bergin

Duckworth

First published in 1984 by
Gerald Duckworth & Co. Ltd.
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1806 7

British Library Cataloguing in Publication Data
Bergin, Kevin
Impossible routines for the Commodore 64.
1. Commodore 64 (Computer)
I. Title
001.64'04 QA76.8.C64

ISBN 0-7156-1806-7

Typeset by The Electronic Village, Richmond
from text stored on a Commodore 64
Printed in Great Britain by
Redwood Burn Ltd., Trowbridge
and bound by Pegasus Bookbinding, Melksham

Contents

Appendices:

Preface

This book was written using the Commodore 64, a 1541 disk drive and an Epson FX-80 (oh! and a TV with me pushing the keys occasionally). The programs were formatted using the Epson's facilities. The book was largely inspired by my sleepless nights huddled over my Vic and later my 64 trying to get commercial software to work.

So my thanks go to the incompetent software houses for their blunders and to Jim Butterfield for his excellent Supermon 64. Thanks are also due to many friends and colleagues including Nick Hampshire for the job, and especially to my publishers for supporting and indulging me.

The programs and information in this book expose areas of the 64 not often talked about and offer different ways of tackling the problems this presents. All the routines are fully explained, including parts of the 64's operating system.

I have included some utility routines which should also prove useful. The end result for the reader should be a more understandable 64.

K.B.

I dedicate this book and the leaves within to all those who are 'locked in' and don't know, also to Mary Smythe who resides with the dust now, but knew what it was like to be here.

Introduction

My career as a programmer started at Middlesex Polytechnic, where I was studying to become a primary school teacher. One day I spotted a notice labelled 'Terminal room', and having determined that it wasn't a place in which overwrought students were disposed of, I asked about this curious room full of odd objects and tense humans.

It turned out to be the college's computer system, a Dec 10, which was spread over five colleges and had 120 people using the system at any one time. I was not scheduled to use the system for some two years, so I nagged for permission to 'challenge' the machine until I was eventually given a number. The following three months saw me in the computer room for two hours every day with a ragged and inadequate manual, trying to discover if I was telling the machine what to do or the reverse.

Having managed to come to grips with Basic by spending too much of my study time on it, I invested in a Vic-20 and spent all of my time and money on that, often to the exclusion of sleep.

At this point I decided that I had better make micros my career. I began work on *Commodore Computing International*, where I carried on with my obsession. This book is probably more a result of the past four years than any other writing I have done, and I hope that you enjoy it.

The title of the book refers to those moments when everything you want to do or try to do with the 64 seems impossible. It also refers to those particular routines and techniques which are at certain levels impossible. In general, *Impossible Routines* is intended to be a very usable guide to those tricky and lesser-known areas of the 64. I hope that you will be able to use it in this way and enjoy the process.

Symbol Chart

Included here is a chart of the abbreviations used in the listings to indicate the 64's control characters. This should avoid any confusion.

[CD]	=	CURSOR DOWN
[CR]	=	CURSOR RIGHT
[CL]	=	CURSOR LEFT
[CU]	=	CURSOR UP
[CLR]	=	SHIFT AND CLR/HOME
[HME]	=	CLR/HOME
[F1]–[F8]	=	FUNCTION KEYS
[BLK]	=	CTRL & 1
[WHT]	=	CTRL & 2
[RED]	=	CTRL & 3
[CYN]	=	CTRL & 4
[PUR]	=	CTRL & 5
[GRN]	=	CTRL & 6
[BLU]	=	CTRL & 7
[YEL]	=	CTRL & 8
[SH]	=	SHIFT (with character following)
[LO]	=	LOGO (with character following)

The shift and logo keys are used for the graphics and any numbers inside the square brackets indicate the number of characters.

1. Supermon 64

I have included a copy of Jim Butterfield's excellent Supermon, which you will definitely need. This kind of utility is usually placed at the back of books. It was decided in this case to place Supermon at the front as it will be in constant use.

Unfortunately Supermon is rather a large program. It occupies the addresses from 2048 decimal $0800 hex to 4591 decimal $11EF hex, some 2543 bytes.

In order to make entering Supermon as smooth as possible and avoid previous confusions, it is presented here as a Basic program with a checksum; a memory dump is also included as you may well want to see it. A disassembly would have been too long and untidy.

Entering Supermon

In order to enter Supermon we must first leave enough room for it by moving the beginning of Basic, so before you start tapping away enter the following in direct mode:

```
POKE 8192,0:POKE 44,32<press return>
```

This leaves enough room for us to enter Supermon. Now start the laborious task of entering the program with all the data statements exactly as shown. It is best to keep track of your position by marking it with a pencil when you have (eventually) finished. Then save the program onto tape or disk in the normal way.

Testing Supermon

Now that you have a copy of the Basic program, RUN the program. There will be a pause and a message will tell you when Supermon has been entered. If you got it right first time, congratulations. If you didn't it's back to the drawing board to discover the error. If you make any corrections don't forget to re-save the program before trying again.

```
10 POKE53280,2:POKE53281,0:PRINT"PLEASE WAIT.....
....."
20 MEM = 2048:COUNT = 0
30 READ NUM:IF NUM =-1 THEN60
40 POKE MEM,NUM:MEM = MEM +1:COUNT = COUNT + NUM
50 GOTO30
60 IF CO <> 283598 OR ME <> 4591 THEN PRINT"DATA
ERROR SHOULD BE 283598 ";" NOT";CO:END
70 PRINT" DATA ENTERED OK NOW ENTER FINAL POKES"
80 END
90 DATA0,26,8,100,0,153,34,147,18,29,29,29,29,83,
85,80
100 DATA69,82,32,54,52,45,77,79,78,0,49,8,110,0,1
53,34
110 DATA17,32,32,32,32,32,32,32,32,32,32,32,32,32
,32,32
120 DATA0,75,8,120,0,153,34,17,32,46,46,74,73,77,
32,66
130 DATA85,84,84,69,82,70,73,69,76,68,0,102,8,130
,0,158
140 DATA40,194,40,52,51,41,170,50,53,54,172,194,4
0,52,52,41
150 DATA170,49,50,55,41,0,0,0,170,170,170,170,170
,170,170,170
160 DATA170,170,170,170,170,170,170,170,170,170,1
70,170,170,170,170,170
170 DATA165,45,133,34,165,46,133,35,165,55,133,36
,165,56,133,37
180 DATA160,0,165,34,208,2,198,35,198,34,177,34,2
08,60,165,34
190 DATA208,2,198,35,198,34,177,34,240,33,133,38,
165,34,208,2
200 DATA198,35,198,34,177,34,24,101,36,170,165,38
,101,37,72,165
210 DATA55,208,2,198,56,198,55,104,145,55,138,72,
165,55,208,2
220 DATA198,56,198,55,104,145,55,24,144,182,201,7
9,208,237,165,55
230 DATA133,51,165,56,133,52,108,55,0,79,79,79,79
,173,230,255
240 DATA0,141,22,3,173,231,255,0,141,23,3,169,128
,32,144,255
250 DATA0,0,216,104,141,62,2,104,141,61,2,104,141
,60,2,104
260 DATA141,59,2,104,170,104,168,56,138,233,2,141
,58,2,152,233
270 DATA0,0,141,57,2,186,142,63,2,32,87,253,0,162
,66,169
280 DATA42,32,87,250,0,169,82,208,52,230,193,208,
6,230,194,208
290 DATA2,230,38,96,32,207,255,201,13,208,248,104
```

```
,104,169,158,32
300 DATA210,255,169,0,0,133,38,162,13,169,46,32,8
7,250,0,169
310 DATA158,32,210,255,32,62,248,0,201,46,240,249
,201,32,240,245
320 DATA162,14,221,183,255,0,208,12,138,10,170,18
9,199,255,0,72
330 DATA189,198,255,0,72,96,202,16,236,76,237,250
,0,165,193,141
340 DATA58,2,165,194,141,57,2,96,169,8,133,29,160
,0,0,32
350 DATA84,253,0,177,193,32,72,250,0,32,51,248,0,
198,29,208
360 DATA241,96,32,136,250,0,144,11,162,0,0,129,19
3,193,193,240
370 DATA3,76,237,250,0,32,51,248,0,198,29,96,169,
59,133,193
380 DATA169,2,133,194,169,5,96,152,72,32,87,253,0
,104,162,46
390 DATA76,87,250,0,169,158,32,210,255,162,0,0,18
9,234,255,0
400 DATA32,210,255,232,224,22,208,245,160,59,32,1
94,248,0,173,57
410 DATA2,32,72,250,0,173,58,2,32,72,250,0,32,183
,248,0
420 DATA32,141,248,0,240,92,32,62,248,0,32,121,25
0,0,144,51
430 DATA32,105,250,0,32,62,248,0,32,121,250,0,144
,40,32,105
440 DATA250,0,169,158,32,210,255,32,225,255,240,6
0,166,38,208,56
450 DATA165,195,197,193,165,196,229,194,144,46,16
0,58,32,194,248,0
460 DATA32,65,250,0,32,139,248,0,240,224,76,237,2
50,0,32,121
470 DATA250,0,144,3,32,128,248,0,32,183,248,0,208
,7,32,121
480 DATA250,0,144,235,169,8,133,29,32,62,248,0,32
,161,248,0
490 DATA208,248,76,71,248,0,32,207,255,201,13,240
,12,201,32,208
500 DATA209,32,121,250,0,144,3,32,128,248,0,169,1
58,32,210,255
510 DATA174,63,2,154,120,173,57,2,72,173,58,2,72,
173,59,2
520 DATA72,173,60,2,174,61,2,172,62,2,64,169,158,
32,210,255
530 DATA174,63,2,154,108,2,160,160,1,132,186,132,
185,136,132,183
540 DATA132,144,132,147,169,64,133,187,169,2,133,
188,32,207,255,201
```

```
550 DATA32,240,249,201,13,240,56,201,34,208,20,32
,207,255,201,34
560 DATA240,16,201,13,240,41,145,187,230,183,200,
192,16,208,236,76
570 DATA237,250,0,32,207,255,201,13,240,22,201,44
,208,220,32,136
580 DATA250,0,41,15,240,233,201,3,240,229,133,186
,32,207,255,201
590 DATA13,96,108,48,3,108,50,3,32,150,249,0,208,
212,169,158
600 DATA32,210,255,169,0,0,32,239,249,0,165,144,4
1,16,208,196
610 DATA76,71,248,0,32,150,249,0,201,44,208,186,3
2,121,250,0
620 DATA32,105,250,0,32,207,255,201,44,208,173,32
,121,250,0,165
630 DATA193,133,174,165,194,133,175,32,105,250,0,
32,207,255,201,13
640 DATA208,152,169,158,32,210,255,32,242,249,0,7
6,71,248,0,165
650 DATA194,32,72,250,0,165,193,72,74,74,74,74,32
,96,250,0
660 DATA170,104,41,15,32,96,250,0,72,138,32,210,2
55,104,76,210
670 DATA255,9,48,201,58,144,2,105,6,96,162,2,181,
192,72,181
680 DATA194,149,192,104,149,194,202,208,243,96,32
,136,250,0,144,2
690 DATA133,194,32,136,250,0,144,2,133,193,96,169
,0,0,133,42
700 DATA32,62,248,0,201,32,208,9,32,62,248,0,201,
32,208,14
710 DATA24,96,32,175,250,0,10,10,10,10,133,42,32,
62,248,0
720 DATA32,175,250,0,5,42,56,96,201,58,144,2,105,
8,41,15
730 DATA96,162,2,44,162,0,0,180,193,208,8,180,194
,208,2,230
740 DATA38,214,194,214,193,96,32,62,248,0,201,32,
240,249,96,169
750 DATA0,0,141,0,0,1,32,204,250,0,32,143,250,0,3
2,124
760 DATA250,0,144,9,96,32,62,248,0,32,121,250,0,1
76,222,174
770 DATA63,2,154,169,158,32,210,255,169,63,32,210
,255,76,71,248
780 DATA0,32,84,253,0,202,208,250,96,230,195,208,
2,230,196,96
790 DATA162,2,181,192,72,181,39,149,192,104,149,3
9,202,208,243,96
800 DATA165,195,164,196,56,233,2,176,14,136,144,1
```

```
1,165,40,164,41
810 DATA76,51,251,0,165,195,164,196,56,229,193,13
3,30,152,229,194
820 DATA168,5,30,96,32,212,250,0,32,105,250,0,32,
229,250,0
830 DATA32,12,251,0,32,229,250,0,32,47,251,0,32,1
05,250,0
840 DATA144,21,166,38,208,100,32,40,251,0,144,95,
161,193,129,195
850 DATA32,5,251,0,32,51,248,0,208,235,32,40,251,
0,24,165
860 DATA30,101,195,133,195,152,101,196,133,196,32
,12,251,0,166,38
870 DATA208,61,161,193,129,195,32,40,251,0,176,52
,32,184,250,0
880 DATA32,187,250,0,76,125,251,0,32,212,250,0,32
,105,250,0
890 DATA32,229,250,0,32,105,250,0,32,62,248,0,32,
136,250,0
900 DATA144,20,133,29,166,38,208,17,32,47,251,0,1
44,12,165,29
910 DATA129,193,32,51,248,0,208,238,76,237,250,0,
76,71,248,0
920 DATA32,212,250,0,32,105,250,0,32,229,250,0,32
,105,250,0
930 DATA32,62,248,0,162,0,0,32,62,248,0,201,39,20
8,20,32
940 DATA62,248,0,157,16,2,232,32,207,255,201,13,2
40,34,224,32
950 DATA208,241,240,28,142,0,0,1,32,143,250,0,144
,198,157,16
960 DATA2,232,32,207,255,201,13,240,9,32,136,250,
0,144,182,224
970 DATA32,208,236,134,28,169,158,32,210,255,32,8
7,253,0,162,0
980 DATA0,160,0,0,177,193,221,16,2,208,12,200,232
,228,28,208
990 DATA243,32,65,250,0,32,84,253,0,32,51,248,0,1
66,38,208
1000 DATA141,32,47,251,0,176,221,76,71,248,0,32,2
12,250,0,133
1010 DATA32,165,194,133,33,162,0,0,134,40,169,147
,32,210,255,169
1020 DATA152,32,210,255,169,22,133,29,32,106,252,
0,32,202,252,0
1030 DATA133,193,132,194,198,29,208,242,169,145,3
2,210,255,76,71,248
1040 DATA0,160,44,32,194,248,0,32,84,253,0,32,65,
250,0,32
1050 DATA84,253,0,162,0,0,161,193,32,217,252,0,72
,32,31,253
```

```
1060 DATAØ,104,32,53,253,Ø,162,6,224,3,208,18,164
,31,24Ø,14
1070 DATA165,42,201,232,177,193,176,28,32,194,252
,Ø,136,208,242,6
1080 DATA42,144,14,189,42,255,Ø,32,165,253,Ø,189,
48,255,Ø,24Ø
1090 DATA3,32,165,253,Ø,202,208,213,96,32,205,252
,Ø,17Ø,232,208
1100 DATA1,20Ø,152,32,194,252,Ø,138,134,28,32,72,
25Ø,Ø,166,28
1110 DATA96,165,31,56,164,194,17Ø,16,1,136,1Ø1,19
3,144,1,2ØØ,96
1120 DATA168,74,144,11,74,176,23,2Ø1,34,24Ø,19,41
,7,9,128,74
1130 DATA17Ø,189,217,254,Ø,176,4,74,74,74,74,41,1
5,208,4,16Ø
1140 DATA128,169,Ø,Ø,17Ø,189,29,255,Ø,133,42,41,3
,133,31,152
1150 DATA41,143,17Ø,152,16Ø,3,224,138,24Ø,11,74,1
44,8,74,74,9
1160 DATA32,136,208,25Ø,2ØØ,136,208,242,96,177,19
3,32,194,252,Ø,162
1170 DATA1,32,254,25Ø,Ø,196,31,2ØØ,144,241,162,3,
192,4,144,242
1180 DATA96,168,185,55,255,Ø,133,4Ø,185,119,255,Ø
,133,41,169,Ø
1190 DATAØ,16Ø,5,6,41,38,4Ø,42,136,208,248,1Ø5,63
,32,21Ø,255
1200 DATA2Ø2,208,236,169,32,44,169,13,76,21Ø,255,
32,212,25Ø,Ø,32
1210 DATA1Ø5,25Ø,Ø,32,229,25Ø,Ø,32,1Ø5,25Ø,Ø,162,
Ø,Ø,134,4Ø
1220 DATA169,158,32,21Ø,255,32,87,253,Ø,32,114,25
2,Ø,32,2Ø2,252
1230 DATAØ,133,193,132,194,32,225,255,24Ø,5,32,47
,251,Ø,176,233
1240 DATA76,71,248,Ø,32,212,25Ø,Ø,169,3,133,29,32
,62,248,Ø
1250 DATA32,161,248,Ø,208,248,165,32,133,193,165,
33,133,194,76,7Ø
1260 DATA252,Ø,197,4Ø,24Ø,3,32,21Ø,255,96,32,212,
25Ø,Ø,32,1Ø5
1270 DATA25Ø,Ø,142,17,2,162,3,32,2Ø4,25Ø,Ø,72,2Ø2
,208,249,162
1280 DATA3,1Ø4,56,233,63,16Ø,5,74,11Ø,17,2,11Ø,16
,2,136,208
1290 DATA246,2Ø2,208,237,162,2,32,2Ø7,255,2Ø1,13,
24Ø,3Ø,2Ø1,32,24Ø
1300 DATA245,32,208,254,Ø,176,15,32,156,25Ø,Ø,164
,193,132,194,133
1310 DATA193,169,48,157,16,2,232,157,16,2,232,208
```

```
,219,134,40,162
1320 DATA0,0,134,38,240,4,230,38,240,117,162,0,0,
134,29,165
1330 DATA38,32,217,252,0,166,42,134,41,170,188,55
,255,0,189,119
1340 DATA255,0,32,185,254,0,208,227,162,6,224,3,2
08,25,164,31
1350 DATA240,21,165,42,201,232,169,48,176,33,32,1
91,254,0,208,204
1360 DATA32,193,254,0,208,199,136,208,235,6,42,14
4,11,188,48,255
1370 DATA0,189,42,255,0,32,185,254,0,208,181,202,
208,209,240,10
1380 DATA32,184,254,0,208,171,32,184,254,0,208,16
6,165,40,197,29
1390 DATA208,160,32,105,250,0,164,31,240,40,165,4
1,201,157,208,26
1400 DATA32,28,251,0,144,10,152,208,4,165,30,16,1
0,76,237,250
1410 DATA0,200,208,250,165,30,16,246,164,31,208,3
,185,194,0,0
1420 DATA145,193,136,208,248,165,38,145,193,32,20
2,252,0,133,193,132
1430 DATA194,169,158,32,210,255,160,65,32,194,248
,0,32,84,253,0
1440 DATA32,65,250,0,32,84,253,0,169,158,32,210,2
55,76,176,253
1450 DATA0,168,32,191,254,0,208,17,152,240,14,134
,28,166,29,221
1460 DATA16,2,8,232,134,29,166,28,40,96,201,48,14
4,3,201,71
1470 DATA96,56,96,64,2,69,3,208,8,64,9,48,34,69,5
1,208
1480 DATA8,64,9,64,2,69,51,208,8,64,9,64,2,69,179
,208
1490 DATA8,64,9,0,0,34,68,51,208,140,68,0,0,17,34
,68
1500 DATA51,208,140,68,154,16,34,68,51,208,8,64,9
,16,34,68
1510 DATA51,208,8,64,9,98,19,120,169,0,0,33,129,1
30,0,0
1520 DATA0,0,89,77,145,146,134,74,133,157,44,41,4
4,35,40,36
1530 DATA89,0,0,88,36,36,0,0,28,138,28,35,93,139,
27,161
1540 DATA157,138,29,35,157,139,29,161,0,0,41,25,1
74,105,168,25
1550 DATA35,36,83,27,35,36,83,25,161,0,0,26,91,91
,165,105
1560 DATA36,36,174,174,168,173,41,0,0,124,0,0,21,
156,109,156
```

```
1570 DATA165,105,41,83,132,19,52,17,165,105,35,16
0,216,98,90,72
1580 DATA38,98,148,136,84,68,200,84,104,68,232,14
8,0,0,180,8
1590 DATA132,116,180,40,110,116,244,204,74,114,24
2,164,138,0,0,170
1600 DATA162,162,116,116,116,114,68,104,178,50,17
8,0,0,34,0,0
1610 DATA26,26,38,38,114,114,136,200,196,202,38,7
2,68,68,162,200
1620 DATA58,59,82,77,71,88,76,83,84,70,72,68,80,4
4,65,66
1630 DATA249,0,53,249,0,204,248,0,247,248,0,86,24
9,0,137,249
1640 DATA0,244,249,0,12,250,0,62,251,0,146,251,0,
192,251,0
1650 DATA56,252,0,91,253,0,138,253,0,172,253,0,70
,248,0,255
1660 DATA247,0,237,247,0,13,32,32,32,80,67,32,32,
83,82,32
1670 DATA65,67,32,88,82,32,89,82,32,83,80,0,0,0,0
1680 DATA-1

READY.
```

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
.:0800 00 1A 08 64 00 99 22 93
.:0808 12 1D 1D 1D 1D 53 55 50
.:0810 45 52 20 36 34 2D 4D 4F
.:0818 4E 00 31 08 6E 00 99 22
.:0820 11 20 20 20 20 20 20 20
.:0828 20 20 20 20 20 20 20 20
.:0830 00 4B 08 78 00 99 22 11
.:0838 20 2E 2E 4A 49 4D 20 42
.:0840 55 54 54 45 52 46 49 45
.:0848 4C 44 00 66 08 82 00 9E
.:0850 28 C2 28 34 33 29 AA 32
.:0858 35 36 AC C2 28 34 34 29
.:0860 AA 31 32 37 29 00 00 00
.:0868 AA AA AA AA AA AA AA AA
.:0870 AA AA AA AA AA AA AA AA
.:0878 AA AA AA AA AA AA AA AA
```

```
.:0880 A5 2D 85 22 A5 2E 85 23
.:0888 A5 37 85 24 A5 38 85 25
.:0890 A0 00 A5 22 D0 02 C6 23
.:0898 C6 22 B1 22 D0 3C A5 22
.:08A0 D0 02 C6 23 C6 22 B1 22
.:08A8 F0 21 85 26 A5 22 D0 02
.:08B0 C6 23 C6 22 B1 22 18 65
.:08B8 24 AA A5 26 65 25 48 A5
.:08C0 37 D0 02 C6 38 C6 37 68
.:08C8 91 37 8A 48 A5 37 D0 02
.:08D0 C6 38 C6 37 68 91 37 18
.:08D8 90 B6 C9 4F D0 ED A5 37
.:08E0 85 33 A5 38 85 34 6C 37
.:08E8 00 4F 4F 4F 4F AD E6 FF
.:08F0 00 8D 16 03 AD E7 FF 00
.:08F8 8D 17 03 A9 80 20 90 FF
.:0900 00 00 D8 68 8D 3E 02 68
.:0908 8D 3D 02 68 8D 3C 02 68
.:0910 8D 3B 02 68 AA 68 A8 38
.:0918 8A E9 02 8D 3A 02 98 E9
.:0920 00 00 8D 39 02 BA 8E 3F
.:0928 02 20 57 FD 00 A2 42 A9
.:0930 2A 20 57 FA 00 A9 52 D0
.:0938 34 E6 C1 D0 06 E6 C2 D0
.:0940 02 E6 26 60 20 CF FF C9
.:0948 0D D0 F8 68 68 A9 9E 20
.:0950 D2 FF A9 00 00 85 26 A2
.:0958 0D A9 2E 20 57 FA 00 A9
.:0960 9E 20 D2 FF 20 3E F8 00
.:0968 C9 2E F0 F9 C9 20 F0 F5
.:0970 A2 0E DD B7 FF 00 D0 0C
.:0978 8A 0A AA BD C7 FF 00 48
.:0980 BD C6 FF 00 48 60 CA 10
.:0988 EC 4C ED FA 00 A5 C1 8D
.:0990 3A 02 A5 C2 8D 39 02 60
.:0998 A9 08 85 1D A0 00 00 20
.:09A0 54 FD 00 B1 C1 20 48 FA
.:09A8 00 20 33 F8 00 C6 1D D0
.:09B0 F1 60 20 88 FA 00 90 0B
.:09B8 A2 00 00 81 C1 C1 C1 F0
.:09C0 03 4C ED FA 00 20 33 F8
.:09C8 00 C6 1D 60 A9 3B 85 C1
.:09D0 A9 02 85 C2 A9 05 60 98
.:09D8 48 20 57 FD 00 68 A2 2E
.:09E0 4C 57 FA 00 A9 9E 20 D2
.:09E8 FF A2 00 00 BD EA FF 00
.:09F0 20 D2 FF E8 E0 16 D0 F5
.:09F8 A0 3B 20 C2 F8 00 AD 39
.:0A00 02 20 48 FA 00 AD 3A 02
.:0A08 20 48 FA 00 20 B7 F8 00
.:0A10 20 8D F8 00 F0 5C 20 3E
```

```
.:0A18 F8 00 20 79 FA 00 90 33
.:0A20 20 69 FA 00 20 3E F8 00
.:0A28 20 79 FA 00 90 28 20 69
.:0A30 FA 00 A9 9E 20 D2 FF 20
.:0A38 E1 FF F0 3C A6 26 D0 38
.:0A40 A5 C3 C5 C1 A5 C4 E5 C2
.:0A48 90 2E A0 3A 20 C2 F8 00
.:0A50 20 41 FA 00 20 8B F8 00
.:0A58 F0 E0 4C ED FA 00 20 79
.:0A60 FA 00 90 03 20 80 F8 00
.:0A68 20 B7 F8 00 D0 07 20 79
.:0A70 FA 00 90 EB A9 08 85 1D
.:0A78 20 3E F8 00 20 A1 F8 00
.:0A80 D0 F8 4C 47 F8 00 20 CF
.:0A88 FF C9 0D F0 0C C9 20 D0
.:0A90 D1 20 79 FA 00 90 03 20
.:0A98 80 F8 00 A9 9E 20 D2 FF
.:0AA0 AE 3F 02 9A 78 AD 39 02
.:0AA8 48 AD 3A 02 48 AD 3B 02
.:0AB0 48 AD 3C 02 AE 3D 02 AC
.:0AB8 3E 02 40 A9 9E 20 D2 FF
.:0AC0 AE 3F 02 9A 6C 02 A0 A0
.:0AC8 01 84 BA 84 B9 88 84 B7
.:0AD0 84 90 84 93 A9 40 85 BB
.:0AD8 A9 02 85 BC 20 CF FF C9
.:0AE0 20 F0 F9 C9 0D F0 38 C9
.:0AE8 22 D0 14 20 CF FF C9 22
.:0AF0 F0 10 C9 0D F0 29 91 BB
.:0AF8 E6 B7 C8 C0 10 D0 EC 4C
.:0B00 ED FA 00 20 CF FF C9 0D
.:0B08 F0 16 C9 2C D0 DC 20 88
.:0B10 FA 00 29 0F F0 E9 C9 03
.:0B18 F0 E5 85 BA 20 CF FF C9
.:0B20 0D 60 6C 30 03 6C 32 03
.:0B28 20 96 F9 00 D0 D4 A9 9E
.:0B30 20 D2 FF A9 00 00 20 EF
.:0B38 F9 00 A5 90 29 10 D0 C4
.:0B40 4C 47 F8 00 20 96 F9 00
.:0B48 C9 2C D0 BA 20 79 FA 00
.:0B50 20 69 FA 00 20 CF FF C9
.:0B58 2C D0 AD 20 79 FA 00 A5
.:0B60 C1 85 AE A5 C2 85 AF 20
.:0B68 69 FA 00 20 CF FF C9 0D
.:0B70 D0 98 A9 9E 20 D2 FF 20
.:0B78 F2 F9 00 4C 47 F8 00 A5
.:0B80 C2 20 48 FA 00 A5 C1 48
.:0B88 4A 4A 4A 4A 20 60 FA 00
.:0B90 AA 68 29 0F 20 60 FA 00
.:0B98 48 8A 20 D2 FF 68 4C D2
.:0BA0 FF 09 30 C9 3A 90 02 69
.:0BA8 06 60 A2 02 B5 C0 48 B5
```

```
.:0BB0 C2 95 C0 68 95 C2 CA D0
.:0BB8 F3 60 20 88 FA 00 90 02
.:0BC0 85 C2 20 88 FA 00 90 02
.:0BC8 85 C1 60 A9 00 00 85 2A
.:0BD0 20 3E F8 00 C9 20 D0 09
.:0BD8 20 3E F8 00 C9 20 D0 0E
.:0BE0 18 60 20 AF FA 00 0A 0A
.:0BE8 0A 0A 85 2A 20 3E F8 00
.:0BF0 20 AF FA 00 05 2A 38 60
.:0BF8 C9 3A 90 02 69 08 29 0F
.:0C00 60 A2 02 2C A2 00 00 B4
.:0C08 C1 D0 08 B4 C2 D0 02 E6
.:0C10 26 D6 C2 D6 C1 60 20 3E
.:0C18 F8 00 C9 20 F0 F9 60 A9
.:0C20 00 00 8D 00 00 01 20 CC
.:0C28 FA 00 20 8F FA 00 20 7C
.:0C30 FA 00 90 09 60 20 3E F8
.:0C38 00 20 79 FA 00 B0 DE AE
.:0C40 3F 02 9A A9 9E 20 D2 FF
.:0C48 A9 3F 20 D2 FF 4C 47 F8
.:0C50 00 20 54 FD 00 CA D0 FA
.:0C58 60 E6 C3 D0 02 E6 C4 60
.:0C60 A2 02 B5 C0 48 B5 27 95
.:0C68 C0 68 95 27 CA D0 F3 60
.:0C70 A5 C3 A4 C4 38 E9 02 B0
.:0C78 0E 88 90 0B A5 28 A4 29
.:0C80 4C 33 FB 00 A5 C3 A4 C4
.:0C88 38 E5 C1 85 1E 98 E5 C2
.:0C90 A8 05 1E 60 20 D4 FA 00
.:0C98 20 69 FA 00 20 E5 FA 00
.:0CA0 20 0C FB 00 20 E5 FA 00
.:0CA8 20 2F FB 00 20 69 FA 00
.:0CB0 90 15 A6 26 D0 64 20 28
.:0CB8 FB 00 90 5F A1 C1 81 C3
.:0CC0 20 05 FB 00 20 33 F8 00
.:0CC8 D0 EB 20 28 FB 00 18 A5
.:0CD0 1E 65 C3 85 C3 98 65 C4
.:0CD8 85 C4 20 0C FB 00 A6 26
.:0CE0 D0 3D A1 C1 81 C3 20 28
.:0CE8 FB 00 B0 34 20 B8 FA 00
.:0CF0 20 BB FA 00 4C 7D FB 00
.:0CF8 20 D4 FA 00 20 69 FA 00
.:0D00 20 E5 FA 00 20 69 FA 00
.:0D08 20 3E F8 00 20 88 FA 00
.:0D10 90 14 85 1D A6 26 D0 11
.:0D18 20 2F FB 00 90 0C A5 1D
.:0D20 81 C1 20 33 F8 00 D0 EE
.:0D28 4C ED FA 00 4C 47 F8 00
.:0D30 20 D4 FA 00 20 69 FA 00
.:0D38 20 E5 FA 00 20 69 FA 00
.:0D40 20 3E F8 00 A2 00 00 20
```

```
.:0D48 3E F8 00 C9 27 D0 14 20
.:0D50 3E F8 00 9D 10 02 E8 20
.:0D58 CF FF C9 0D F0 22 E0 20
.:0D60 D0 F1 F0 1C 8E 00 00 01
.:0D68 20 8F FA 00 90 C6 9D 10
.:0D70 02 E8 20 CF FF C9 0D F0
.:0D78 09 20 88 FA 00 90 B6 E0
.:0D80 20 D0 EC 86 1C A9 9E 20
.:0D88 D2 FF 20 57 FD 00 A2 00
.:0D90 00 A0 00 00 B1 C1 DD 10
.:0D98 02 D0 0C C8 E8 E4 1C D0
.:0DA0 F3 20 41 FA 00 20 54 FD
.:0DA8 00 20 33 F8 00 A6 26 D0
.:0DB0 8D 20 2F FB 00 B0 DD 4C
.:0DB8 47 F8 00 20 D4 FA 00 85
.:0DC0 20 A5 C2 85 21 A2 00 00
.:0DC8 86 28 A9 93 20 D2 FF A9
.:0DD0 98 20 D2 FF A9 16 85 1D
.:0DD8 20 6A FC 00 20 CA FC 00
.:0DE0 85 C1 84 C2 C6 1D D0 F2
.:0DE8 A9 91 20 D2 FF 4C 47 F8
.:0DF0 00 A0 2C 20 C2 F8 00 20
.:0DF8 54 FD 00 20 41 FA 00 20
.:0E00 54 FD 00 A2 00 00 A1 C1
.:0E08 20 D9 FC 00 48 20 1F FD
.:0E10 00 68 20 35 FD 00 A2 06
.:0E18 E0 03 D0 12 A4 1F F0 0E
.:0E20 A5 2A C9 E8 B1 C1 B0 1C
.:0E28 20 C2 FC 00 88 D0 F2 06
.:0E30 2A 90 0E BD 2A FF 00 20
.:0E38 A5 FD 00 BD 30 FF 00 F0
.:0E40 03 20 A5 FD 00 CA D0 D5
.:0E48 60 20 CD FC 00 AA E8 D0
.:0E50 01 C8 98 20 C2 FC 00 8A
.:0E58 86 1C 20 48 FA 00 A6 1C
.:0E60 60 A5 1F 38 A4 C2 AA 10
.:0E68 01 88 65 C1 90 01 C8 60
.:0E70 A8 4A 90 0B 4A B0 17 C9
.:0E78 22 F0 13 29 07 09 80 4A
.:0E80 AA BD D9 FE 00 B0 04 4A
.:0E88 4A 4A 4A 29 0F D0 04 A0
.:0E90 80 A9 00 00 AA BD 1D FF
.:0E98 00 85 2A 29 03 85 1F 98
.:0EA0 29 8F AA 98 A0 03 E0 8A
.:0EA8 F0 0B 4A 90 08 4A 4A 09
.:0EB0 20 88 D0 FA C8 88 D0 F2
.:0EB8 60 B1 C1 20 C2 FC 00 A2
.:0EC0 01 20 FE FA 00 C4 1F C8
.:0EC8 90 F1 A2 03 C0 04 90 F2
.:0ED0 60 A8 B9 37 FF 00 85 28
.:0ED8 B9 77 FF 00 85 29 A9 00
```

```
.:0EE0 00 A0 05 06 29 26 28 2A
.:0EE8 88 D0 F8 69 3F 20 D2 FF
.:0EF0 CA D0 EC A9 20 2C A9 0D
.:0EF8 4C D2 FF 20 D4 FA 00 20
.:0F00 69 FA 00 20 E5 FA 00 20
.:0F08 69 FA 00 A2 00 00 86 28
.:0F10 A9 9E 20 D2 FF 20 57 FD
.:0F18 00 20 72 FC 00 20 CA FC
.:0F20 00 85 C1 84 C2 20 E1 FF
.:0F28 F0 05 20 2F FB 00 B0 E9
.:0F30 4C 47 F8 00 20 D4 FA 00
.:0F38 A9 03 85 1D 20 3E F8 00
.:0F40 20 A1 F8 00 D0 F8 A5 20
.:0F48 85 C1 A5 21 85 C2 4C 46
.:0F50 FC 00 C5 28 F0 03 20 D2
.:0F58 FF 60 20 D4 FA 00 20 69
.:0F60 FA 00 8E 11 02 A2 03 20
.:0F68 CC FA 00 48 CA D0 F9 A2
.:0F70 03 68 38 E9 3F A0 05 4A
.:0F78 6E 11 02 6E 10 02 88 D0
.:0F80 F6 CA D0 ED A2 02 20 CF
.:0F88 FF C9 0D F0 1E C9 20 F0
.:0F90 F5 20 D0 FE 00 B0 0F 20
.:0F98 9C FA 00 A4 C1 84 C2 85
.:0FA0 C1 A9 30 9D 10 02 E8 9D
.:0FA8 10 02 E8 D0 DB 86 28 A2
.:0FB0 00 00 86 26 F0 04 E6 26
.:0FB8 F0 75 A2 00 00 86 1D A5
.:0FC0 26 20 D9 FC 00 A6 2A 86
.:0FC8 29 AA BC 37 FF 00 BD 77
.:0FD0 FF 00 20 B9 FE 00 D0 E3
.:0FD8 A2 06 E0 03 D0 19 A4 1F
.:0FE0 F0 15 A5 2A C9 E8 A9 30
.:0FE8 B0 21 20 BF FE 00 D0 CC
.:0FF0 20 C1 FE 00 D0 C7 88 D0
.:0FF8 EB 06 2A 90 0B BC 30 FF
.:1000 00 BD 2A FF 00 20 B9 FE
.:1008 00 D0 B5 CA D0 D1 F0 0A
.:1010 20 B8 FE 00 D0 AB 20 B8
.:1018 FE 00 D0 A6 A5 28 C5 1D
.:1020 D0 A0 20 69 FA 00 A4 1F
.:1028 F0 28 A5 29 C9 9D D0 1A
.:1030 20 1C FB 00 90 0A 98 D0
.:1038 04 A5 1E 10 0A 4C ED FA
.:1040 00 C8 D0 FA A5 1E 10 F6
.:1048 A4 1F D0 03 B9 C2 00 00
.:1050 91 C1 88 D0 F8 A5 26 91
.:1058 C1 20 CA FC 00 85 C1 84
.:1060 C2 A9 9E 20 D2 FF A0 41
.:1068 20 C2 F8 00 20 54 FD 00
.:1070 20 41 FA 00 20 54 FD 00
```

```
.:1078 A9 9E 20 D2 FF 4C B0 FD
.:1080 00 A8 20 BF FE 00 D0 11
.:1088 98 F0 0E 86 1C A6 1D DD
.:1090 10 02 08 E8 86 1D A6 1C
.:1098 28 60 C9 30 90 03 C9 47
.:10A0 60 38 60 40 02 45 03 D0
.:10A8 08 40 09 30 22 45 33 D0
.:10B0 08 40 09 40 02 45 33 D0
.:10B8 08 40 09 40 02 45 B3 D0
.:10C0 08 40 09 00 00 22 44 33
.:10C8 D0 8C 44 00 00 11 22 44
.:10D0 33 D0 8C 44 9A 10 22 44
.:10D8 33 D0 08 40 09 10 22 44
.:10E0 33 D0 08 40 09 62 13 78
.:10E8 A9 00 00 21 81 82 00 00
.:10F0 00 00 59 4D 91 92 86 4A
.:10F8 85 9D 2C 29 2C 23 28 24
.:1100 59 00 00 58 24 24 00 00
.:1108 1C 8A 1C 23 5D 8B 1B A1
.:1110 9D 8A 1D 23 9D 8B 1D A1
.:1118 00 00 29 19 AE 69 A8 19
.:1120 23 24 53 1B 23 24 53 19
.:1128 A1 00 00 1A 5B 5B A5 69
.:1130 24 24 AE AE A8 AD 29 00
.:1138 00 7C 00 00 15 9C 6D 9C
.:1140 A5 69 29 53 84 13 34 11
.:1148 A5 69 23 A0 D8 62 5A 48
.:1150 26 62 94 88 54 44 C8 54
.:1158 68 44 E8 94 00 00 B4 08
.:1160 84 74 B4 28 6E 74 F4 CC
.:1168 4A 72 F2 A4 8A 00 00 AA
.:1170 A2 A2 74 74 74 72 44 68
.:1178 B2 32 B2 00 00 22 00 00
.:1180 1A 1A 26 26 72 72 88 C8
.:1188 C4 CA 26 48 44 44 A2 C8
.:1190 3A 3B 52 4D 47 58 4C 53
.:1198 54 46 48 44 50 2C 41 42
.:11A0 F9 00 35 F9 00 CC F8 00
.:11A8 F7 F8 00 56 F9 00 89 F9
.:11B0 00 F4 F9 00 0C FA 00 3E
.:11B8 FB 00 92 FB 00 C0 FB 00
.:11C0 38 FC 00 5B FD 00 8A FD
.:11C8 00 AC FD 00 46 F8 00 FF
.:11D0 F7 00 ED F7 00 0D 20 20
.:11D8 20 50 43 20 20 53 52 20
.:11E0 41 43 20 58 52 20 59 52
.:11E8 20 53 50 00 00 00 00 00
.
.
```

Saving Supermon

When your Basic program has successfully entered the data, you will need to save the machine code version of Supermon. To do this, enter the following in direct mode:

POKE 44,8:POKE 45,235:POKE 46,17:CLR <press return>

You now have a working copy of Supermon in memory, and a normal save to tape or disk will save it for you.

Using Supermon

To load Supermon use a normal load and run. This will load and initialise Supermon. It is advisable to exit the monitor at this point (see exit command) and new the Basic area. To re-enter Supermon enter SYS 8 <press return> in direct mode: this command will always take you back to the monitor unless run/stop and restore has been pressed.

Supermon colours

Anyone familiar with Supermon 64 will be aware that the colours are none too good on an ordinary television (I must buy a real monitor), so using the advice Jim gave in an article for those of us who don't like the colour combination I changed them! You may not like my choice, so I will explain how to change the colours.

Load Supermon and run it. This will put you into the monitor with the colours I set. To change the colours temporarily enter the following command:

```
.H 97ED 9FFF A9 98 20 D2 FF
```

This should give you one or possibly two locations. Change the 98 to the ASCII code for the colour you require.

Now you will need to change the other colours. Enter the following command:

```
.H 97ED 9FFF A9 9E 20 D2 FF
```

This will give you about thirteen locations. Change the 9E (ASCII code for yellow) to the colour you require. This will only give temporary changes; to make permanent changes you will need to make the hunt from the beginning of Basic:

```
.H 0800 11EF...
```

Instructions

Below is a full list of Supermon 64 instructions. The left-hand column gives the command, the middle column contains the syntax, and the right-hand column the action.

Command	Syntax	Action
Simple Assembler	`.A C000 LDX #$00`	starts assembly at $C000 hex.
Disassembler	`.D C000`	disassembles from $C000 hex onwards
Printing Disassembler	`P C000 C100`	disassembles to printer once engaged with OPEN4,4: CMD4:SYS8.
Fill memory	`F C000 C100 AA`	fills memory from C000 to C100 with the hex byte AA.
Go run	`G C000`	jumps to $C000 hex and executes program there.
Hunt memory	`H C000 C100 STAR`	hunts through memory $C000 to $C100 hex for the ASCII string STAR.
Load	`.L"filename",08`	loads a program from disk into memory.

Memory display	.M C000 C020	displays memory from C000 to C020 hex.
Register status	.R	displays current register values.
Save	.S"nn",08,C000,C100	saves memory from C000 to C100 hex onto disk and calls it nn.
Transfer memory	.T C000 C100 C200	transfers contents of memory in the range C000 to C100 hex to new start address of C200 hex
Exit to Basic	.X	return to Basic and perform a CLR before doing anything else

2. Protection

A trick of the trade?

The word 'protection', when applied to computer programs, often conjures up the idea of an impenetrable defence. However, protection is merely a trick of the trade, in other words some fancy routines that a programmer has added to his program in order to make it harder to unravel and examine or copy it.

There is still a lot of talk and speculation about pirating, but not much action. The software houses would look pretty silly if they tried to sue one of their customers who made a one-off copy for a friend. Although they should be protecting themselves against large-scale copying and selling, perhaps profits don't warrant it.

Having decided that there is no such thing as a piece of totally protected software and that any program is only as well protected as the programmer wishes to make it, we can look at various aspects of protection relating to the 64.

I always think of protection as being two distinct areas and label them internal and external protection. The terms are very easily explained. Internal protection refers to all methods of protection within the main program. That is, routines that stop the user from examining, saving or abusing the main program in any way after the program has been loaded and executed.External protection is any routine used as a loader for the main program, and is normally only used for this purpose and then discarded, e.g. an auto-run would only be used to load and execute a program and then would be of no further use. External protection may well be used in more than one way, but is not used once all the programs have been loaded and executed.

Internal protection

Any protected program will have several layers of protection. If the program is tape based then some of the layers will be inside the program. Probably the best internal protection I have seen has come from Terminal and Legend software. There are several points to remember when writing internal protection:

1. The run/stop and restore keys should be disabled or reassigned.

2. Unwanted I/O facilities should be disabled or reassigned.

3. The program should be hidden and perhaps scrambled.

4. On the 64 the ROM could be switched out as could the Kernal (tricky though); Valhalla achieves this nicely.

5. The program will be hard to copy or examine if it is split up.

6. The screen, character set and the start of RAM can be moved.

7. More than one of these routines should be used - possibly two or three.

8. A final caution is to reset the 64 or preferably crash it if all of the above fail, thus protecting the program by brute force.

It may well be a good move to purchase one of the advanced books on the 64, giving memory maps and descriptions of the chips, if you have not already invested in one. This information does not come within the scope of my book. However, you will be able to use the information in this book on its own.

To put the above suggestions into action you will need to have a good understanding of how they work, so read on.

Disabling run/stop & restore

Much has been written about disabling these keys on the 64, but it will not hurt to recap on the information. The 'key' to the run/stop and restore keys on the 64 is in locations 808 and 809 decimal, $0328 and $0329 hex. This is the Kernal stop routine vector.

The contents of these locations need to be altered to disable one or both the these keys. Try entering the following in direct mode:

```
PRINTPEEK(808),PEEK(809)<return>
```

The result should be 237 and 246, unless you have already been meddling with these locations. The meaning of these numbers is simply a jump to a Kernal routine that checks for the stop key being pressed. The routine when intialised sits at location 63213 dec. $F6ED hex.

The stop key and the stop and restore keys can be disabled from Basic with a simple poke, but this produces complications, as we will see. First, let's experiment a little by loading a program that is written in Basic and then entering the following in direct mode:

POKE 808,251<return>

Now list your Basic program and try to stop the listing by pressing run/stop. It worked? Good, now list the program again and press both run/stop and restore keys. Not so good, the listing stopped and the run/stop key is no longer disabled.

One more experiment: enter in direct mode:

POKE 808,237<return>

to reset the stop key. Then enter, again in direct mode:

POKE 808,225<return>

Try pressing the run/stop and restore keys together, and hey presto! it worked. However, if you list your Basic program you will get a weird display on the screen. Don't worry, the program is still there but the listing is corrupted. In fact the program will still run: try it. Perhaps that is worth remembering.

What our experiments have shown is that these two methods are not very clean and a little less than perfect. What we actually need to do is to change the vector to point at a routine of our own, so that we can disable the run/stop and restore keys or set them up to do our own bidding.

Below are two assembly listings which will do this. They can be located in any available memory and may be overwritten if they are no longer required. This would stop anyone working out how you changed the vector. Of course you may also use the knowledge to improve upon it or write your own routines, which is the whole idea. The routines are given as disassembly listings, and there is also a memory dump which is easier to enter using your by now working copy of Supermon!

Disable 1

This first routine will reassign the run/stop and restore keys so as to disable them. An explanation of the routine is hardly needed, but briefly, the first instruction sets the interrupts; the second instruction loads the new high byte for the stop vector; the third instruction stores the new high byte in the high byte of the stop vector; the fourth instruction loads the low byte of the new stop vector; and the fifth instruction stores it in the low byte of the stop vector.

Once this part of the routine has been called, any time the run/stop key is pressed Disable 1 points to a new routine which starts at location 4109 decimal $100D hex. This part of the routine simply places the value to disable the run/stop into the accumulator and returns.

The run/stop and restore keys are now disabled. The first part of the Disable 1 routine is used to point to the routine at $100D hex, but the second part can be set to do just about anything!

```
B*
   PC   SR AC XR YR SP
.;0008 30 00 00 00 F6
.
1000 78          SEI
1001 A9 0D       LDA #$0D
1003 8D 28 03    STA $0328
1006 A9 10       LDA #$10
1008 8D 29 03    STA $0329
100B 58          CLI
100C 60          RTS
100D A5 91       LDA $91
100F 60          RTS
.
.
```

```
.:1000 78 A9 0D 8D 28 03 A9 10
.:1008 8D 29 03 58 60 A5 91 60
.
.
```

Disable 2

Only the second part of this routine need be explained, from location 4109 decimal $100D hex, as the first part is identical to Disable 1. With this routine any press of the run/stop key will point to our new routine starting at $100D. A jump is then made to location $FCE2 which is the entry point for the reset routine and will reset the 64.

Although this looks quite good and seems to be effective, there are drawbacks. After the 64 has been reset with a call to $FCE2, any Basic program in RAM has the first two pointers removed, but the rest of the program is still there and can be recovered. Secondly, if the program in memory is in machine code then the whole program is still there intact and can be got at with a good machine language monitor and enough knowledge.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
1000 78          SEI
1001 A9 0D       LDA #$0D
1003 8D 28 03    STA $0328
1006 A9 10       LDA #$10
1008 8D 29 03    STA $0329
100B 58          CLI
100C 60          RTS
100D 20 E2 FC    JSR $FCE2
.
.
```

```
.:1000 78 A9 0D 8D 28 03 A9 10
.:1008 8D 29 03 58 60 20 E2 FC
.
.
```

Disable 3

This routine has the edge on the above two for internal protection. It does not actually stop the program at any point or lock the run/stop and restore keys. In essence it re-runs the program if the run/stop key is pressed. For our purposes it has been set up to re-run a Basic program, but could easily be altered to point to the beginning of a machine code program or indeed to point to some other position of any program.

Again the first part of the routine is the same as the first two routines. The second part at location $100D executes a JSR to $A65E, which is the entry postion for the CLR instruction. The next instruction is a JSR to $A68E, which is the entry point for the back-up text pointer routine, and the last instruction executes a JMP to location $A7AE, which causes the program in memory to re-run. This should keep a great number of people busy for a long time.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
1000 78          SEI
1001 A9 0D       LDA #$0D
1003 8D 28 03    STA $0328
1006 A9 10       LDA #$10
1008 8D 29 03    STA $0329
100B 58          CLI
100C 60          RTS
100D 20 5E A6    JSR $A65E
1010 20 8E A6    JSR $A68E
1013 4C AE A7    JMP $A7AE
.
.
```

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
.
.
.:1000 78 A9 0D 8D 28 03 A9 10
.:1008 8D 29 03 58 60 20 5E A6
.:1010 20 8E A6 4C AE A7 00 00
.
.
```

Other vectors

At this point it is only fair to mention that there are many other things that it may be necessary to take into account when protecting a program internally. I will attempt to cover as many as possible, but you must remember that the only sure method of protection is to blow up the 64; anything that falls short of this is likely to be tampered with eventually.

If you happen to have an adequate copy of the 64's memory map then you may have noticed a number of vectors from location 768 to 819 decimal; most of these can be altered so that programs can't be listed, saved or loaded. The error messages can be altered or disabled as can the warm start vector, the open and close vector – in fact all the vectors can be disabled or altered. How they are altered and in what way depends very much upon your individual needs.

The routines given above outline one way of disabling or resetting these functions, and I advise that a similar method is used. However, most of these vectors and links can be altered from Basic and in order to give you a taste of what is possible I have included here a list of the pokes and what they do to include in your programs and experiment with.

To disable the 'list' command is very easy. Simply enter POKE 775,200, and this will prevent any prying eyes from looking at your listing. To return to normal enter POKE 775,167.

To disrupt the load and save commands is fairly easy; the two pokes given simply swap commands: POKE 816,237:POKE 817,245:POKE 818,165:POKE 819,244<return>. Any load or save command will now produce the opposite. To disable the error messages enter POKE 768,226:POKE 769,252<return>. This will cause any error encountered to simply reset the 64. You may wish to alter it to point to some other routine in ROM or a routine of your own. Although this is fairly simple, it may be important to change the load and save commands. If this is so it would be better practice to set the 64 to crash on any I/O operations. Resetting location one on an I/O operation would achieve this nicely. Don't forget the open and close command vectors and the I/O links.

Moving Basic

Although this is not in itself a protective measure, it can be added to aid you in your attempts to fool, confuse and generally beat potential pirates (don't get paranoid, though!).

When the 64 goes through its power up routines the start of Basic RAM is normally at 2048 decimal $0800 hex. This can easily be altered by changing locations 43 and 44 decimal (start of Basic pointer). By changing the contents of location 44 decimal the page that Basic starts at can be altered and by changing the contents of location 43 the number of bytes from the top of the page can be altered.

At power up location 44 contains 8 and location 43 contains 1. This points to location 2049 decimal, although in reality Basic starts at location 2048. The first position of the start of Basic must contain a zero or very strange things happen. Every 1 added to location 44 moves Basic by one page and every 1 added to location 43 adds 1 byte to the start of Basic. To set the start of Basic to 2304 decimal (up one page) enter the following:

```
POKE 2304,0:POKE 44,9<return>
```

This not only moves the start of Basic but also leaves some room for machine code routines from 2049 to 2303.

To re-cap, so far we have covered some of the possibilities for internal protection. They include disabling and resetting the run/stop and restore keys; locating and changing other vectors and links; and moving Basic. Once you have mastered the above you will begin to see how powerful protection can be. There is much more, however, so read on.

Scrambling programs

It is possible to save programs in a scrambled form and use the same routine to unscramble them. This is very useful because it is difficult to unscramble the program unless you know how it was scrambled in the first place.

Scrambling programs is almost but not quite as simple as scrambling eggs. The idea is very simple, but effective. The key is the exclusive or (EOR) instruction. Using this command different bytes of memory may be scrambled. The magic comes when you use the EOR instruction on the same part of memory a second time: it restores it to its original state.

The power of this is fairly obvious: you will be able to make your programs meaningless until they go through the scrambling routine a second time. The way I use the routine is to scramble the programs and save them, then have the same routine unscramble them when they are loaded and before any attempt to run them is made, since they will not work until they are put through the routine a second time.

Warnings

There a few things to remember before using this method. The program you wish to scramble should be EOR'd with a stable part of memory (in the example, $A000 hex on) like the ROM. It is no good if the part of memory used is unstable.

Secondly, the first routine given below is a simple version of the two pass scrambler and only deals with one page of memory (256 bytes). It is fairly easy to make it do more, but first you should check the size of the program you are attempting to scramble carefully. Then see the second routine below.

The first routine scrambles a page of memory from 2112 decimal $0840 hex. The routine starts at 49152 decimal $C000 hex and is called by SYS49152. The best way to get to grips with it is to experiment. Load a program that uses normal memory and execute the routine. Try listing the program, which should be garbage now, but don't panic! Execute the scramble routine again and list the now restored program!

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
C000 A2 00          LDX #$00
C002 BD 40 08       LDA $0840,X
C005 5D 00 A0       EOR $A000,X
C008 9D 40 08       STA $0840,X
C00B E8             INX
C00C D0 F4          BNE $C002
C00E 60             RTS
.
.
```

```
.:C000 A2 00 BD 40 08 5D 00 A0
.:C008 9D 40 08 E8 D0 F4 60 00
.
.
```

To extend the number of pages that are scrambled you will need to add some more instructions. For instance, to scramble four pages starting from 2048 decimal $0800 hex the routine would look like this:

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
C000 A9 A0        LDA #$A0
C002 85 FC        STA $FC
C004 A9 00        LDA #$00
C006 85 FB        STA $FB
C008 A9 08        LDA #$08
C00A 85 FE        STA $FE
C00C A9 00        LDA #$00
C00E 85 FE        STA $FE
C010 A0 00        LDY #$00
C012 B1 FD        LDA ($FD),Y
C014 51 FB        EOR ($FB),Y
C016 91 FD        STA ($FD),Y
C018 C8           INY
C019 D0 F7        BNE $C012
C01B E6 FC        INC $FC
C01D E6 FE        INC $FE
C01F A5 FC        LDA $FC
C021 C9 A4        CMP #$A4
C023 D0 EB        BNE $C010
C025 60           RTS
.
.
```

```
.:C000 A9 A0 85 FC A9 00 85 FB
.:C008 A9 08 85 FE A9 00 85 FE
.:C010 A0 00 B1 FD 51 FB 91 FD
.:C018 C8 D0 F7 E6 FC E6 FE A5
.:C020 FC C9 A4 D0 EB 60 00 00
.
.
```

This is slightly different from the first scramble routine in that it actually uses four zero page locations to store the current ROM and RAM bytes, the Y register is used as an offset and the program continues until the check $C021 is true and four pages of RAM have been scrambled. This program must be used again to restore the program.

Finally, here is a third scramble routine. In essence it is the same as the above except that it uses $FF hex to 'exclusive or' the program and it is not necessary to use the ROM. It is set to go through the normal RAM from $0800 hex to $9FFF hex. You may alter this if you wish, by altering the value in the CMP instruction.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
C000 A9 08         LDA #$08
C002 85 FC         STA $FC
C004 A9 00         LDA #$00
C006 85 FB         STA $FB
C008 A0 00         LDY #$00
C00A B1 FB         LDA ($FB),Y
C00C 49 FF         EOR #$FF
C00E 91 FB         STA ($FB),Y
C010 C8            INY
C011 D0 F7         BNE $C00A
C013 E6 FC         INC $FC
C015 E6 FE         INC $FE
C017 A5 FC         LDA $FC
C019 C9 A0         CMP #$A0
C01B D0 EB         BNE $C008
C01D 60            RTS
.
.
```

```
.:C000 A9 08 85 FC A9 00 85 FB
.:C008 A0 00 B1 FB 49 FF 91 FB
.:C010 C8 D0 F7 E6 FC E6 FE A5
.:C018 FC C9 A0 D0 EB 60 00 00
.
.
```

In some cases it may be necessary to replace the first three bytes of RAM by hand (locations 2048 ,2049 and 2050), so check these before you scramble the program.

These routines are called with a SYS 49152.

Screen and character set

It is also possible to have more than one screen and character set on the 64. This is included here because it can aid protection, though it is not any protection on its own, as is the case with many of these routines. The keys to moving the screen are the screen memory pointer at location 648 decimal $0288 hex and location 56576 decimal $DD00 hex which switches banks.

At power up the content of location 648 is 4, which points to 1024 (4 x 256). By altering the contents of this location we can move the screen. Try poking a higher value into 648 – it's messy – so we need to alter some other things in order to set up our new screen.

In order to move the screen it is necessary to make sure that the VIC chip can access all the information it needs. This means changing the screen pointer, switching banks, switching character sets and ensuring that the VIC chip is looking at the right part of memory.

To show how to do this, there is a Basic program which places the screen at 50176 decimal, the character set at 53248 decimal, and selects bank 3, which looks at memory from 49152 decimal $C000 hex to 65535 decimal $FFFF hex.

```
10 POKE 56333,127:REM *** SET INTERRUPTS ***
20 POKE 1,51:REM *** SWITCH IN CHARACTER GENERATOR
 ROM ***
30 FOR I=0 TO 4095:REM *** LOOP TO ***
40 POKE 53248+I,PEEK(53248+I):REM *** MOVE COMPLET
E CHARACTER SET ***
50 NEXT I:REM *** END OF LOOP ***
60 POKE1,55:REM *** SWITCH OUT CHARACTER GENERATOR
 ROM ***
70 POKE 56333,129:REM *** RESET INTERRUPTS ***
80 POKE 648,196:REM *** SET POINTER FOR SCREEN ***
90 POKE 56576,4:REM *** SELECT NEW BANK ***
100 POKE53272,21:REM: ENSURE VIC CHIP KNOWS WERE T
O LOOK
```

```
110 PRINT"          NEW SCREEN READY":REM *** CLEA
R SCREEN AND DISPLAY MESSAGE ***
120 END

READY.
```

The program is documented with REM statements, but briefly: the interrupts are set; the character generator ROM is switched in; the character set is read in; the character generator ROM switched out; the interrupts are reset. So far this is fairly common stuff, but line 80 resets the screen pointer, and the next poke switches banks. Finally the VIC chip is set to ensure that it can see the screen and the character set.

To place the screen elsewhere in memory you will need to change the screen pointer at location 648 decimal, the bank selection at location 56576 decimal and the pointer to the character set 53272 decimal. You may also need to place the character set elsewhere in memory. Good luck with your calculating! It is worth mentioning that the colour memory on the 64 is not movable, so that's one less headache.

A faster version!

If you have just entered and tried the above Basic program and are at this moment cursing me and kicking your 64 because it doesn't seem to be doing anything, my apologies. The routine does work, but takes an awful long time to transfer the complete character set. Here's the same thing in machine code with a memory dump for those of you in a hurry. It is incredibly quick and does the same thing as the Basic program.

```
B*
   PC  SR AC XR YR SP
.;0008 B1 27 01 C4 F6
.
1000 78          SEI
1001 A9 33       LDA #$33
1003 85 01       STA $01
1005 A9 D0       LDA #$D0
1007 85 FC       STA $FC
1009 A9 00       LDA #$00
100B 85 FB       STA $FB
100D A0 00       LDY #$00
100F B1 FB       LDA ($FB),Y
1011 91 FB       STA ($FB),Y
1013 C8          INY
```

```
1014 D0 F9          BNE $100F
1016 E6 FC          INC $FC
1018 A5 FC          LDA $FC
101A C9 E0          CMP #$E0
101C D0 EF          BNE $100D
101E A9 37          LDA #$37
1020 85 01          STA $01
1022 58             CLI
1023 A9 C4          LDA #$C4
1025 8D 88 02       STA $0288
1028 A9 04          LDA #$04
102A 8D 00 DD       STA $DD00
102D A9 15          LDA #$15
102F 8D 18 D0       STA $D018
1032 20 44 E5       JSR $E544
1035 60             RTS
.
.
```

```
B*
   PC   SR AC XR YR SP
.;0008 B1 27 01 C4 F6
.
.:1000 78 A9 33 85 01 A9 D0 85
.:1008 FC A9 00 85 FB A0 00 B1
.:1010 FB 91 FB C8 D0 F9 E6 FC
.:1018 A5 FC C9 E0 D0 EF A9 37
.:1020 85 01 58 A9 C4 8D 88 02
.:1028 A9 04 8D 00 DD A9 15 8D
.:1030 18 D0 20 44 E5 60 00 00
.
.
```

This routine uses the same technique as the scramble routine. The position it occupies can of course be easily changed to please you. The assembly listing is provided and will explain what it is doing. A final point to mention is that the character set has not been re-defined, but just swapped; anyone wishing to add user-defined characters must add their own.

Other forms of protection

All other forms of protection are tricks that are either external protection or a mixture of internal and external. The first point to make is that any program loaded from tape uses the cassette buffer and places the name of the program and the start and end addresses of the program in the buffer. Address 828 decimal $033C hex holds the secondary address for the load. In other words, whether it is a LOAD"" or LOAD"",1,1 or LOAD"",1,3. The first is an ordinary load; the second loads back into the memory it was saved from and the third signifies an auto-run!

The start and end addresses of programs are held in addresses 829 to 832 decimal. 829 holds the low byte and 830 the high byte of the start of the program, 831 holds the low byte and 832 the high byte of the end of the program. So it is sometimes possible to calculate where in memory a program is using these locations. The filename is also stored in the locations after 832 decimal. Experiment with this and see what you get?

Protected software

Most commercial software has some protection. For some reason a number of games writers still use loaders for the 64. Usually these loaders set up some parts of the 64 and load the main program. The loaders often look like the sample below (for tape):

```
10 C=C+1:IF C=2 THEN SYS 49152

20 LOAD"",1,1
```

This is a simple example, but we can learn something from it. In line 10 the variable 'C' is not initialised; on the 64 it is assumed to be zero if there is no other reference to it. So on line 10 'C' is set to one and a check is made for C having the value of two. On the first pass it has a value of one and line 20 then loads the next program on the tape into the memory it came from.

This of course applies only to machine code programs. If the variable 'C' was not used and we had a loader like:

```
10 LOAD"",1,1

20 SYS 49152
```

it would never be called as line 10 would be repeated eternally! This is because after a load from a program a run is performed and the circle begins. This kind of loader can also apply to programs from disk, but is normally only used to load more than one program from disk. An example might look like this:

```
10 C=C+1:IF C=3 THEN SYS49152

20 IF C=2 THEN LOAD"file 2",8

30 LOAD"file 1",8
```

This will load two programs into the 64 from disk and call one of them. It is always worth paying a lot of attention to loaders. They are often used to protect things from prying eyes as well as set up sprites and character sets.

It is interesting to note that any 'awkward' program could usefully be explored by the command OPEN1 for tape, which will find the header and stop. If there is a routine in the header it will then have been loaded and will probably be in memory somewhere above or below the filename (tape buffer). Have a look at the tape buffer to discover if anything has loaded or have a look around location $0351 hex or $02A5 onwards.

Auto-run

A complete auto-run does not come within the scope of this book, but we can still discuss it in detail. The idea is to have a program loaded into the 64 and on completion of the load execute the program. The secret is in how it is saved. One very effective way of doing this is to save the loader and the 'run' in the header. To do this you will need to write a short program that actually saves a piece of code in the header and then saves the main program. The save and load may not be done in the usual way and will take some practice before you get it perfect. It is a good idea to have a look at an already existing auto-run, so I will include a limited auto-run and some other techniques to get you started. First, it is possible to manipulate a program from tape or disk using the keyboard buffer which is located from 631 decimal $0277 hex to 640 decimal $0280 hex. This gives you ten locations, and although it is possible to extend the keyboard buffer it is not always advisable.

Any characters stored in the keyboard buffer remain there until the program halts unless they are overwritten or the number exceeds the limit of the buffer. Therefore commands can be placed in the keyboard buffer and left there to execute when the program stops. You could also force a program to stop and execute commands in the buffer and then jump back to the program.

It is equally easy to use the buffer from Basic or machine code. You should always remember that location 198 decimal $C6 hex, which is the pointer for the number of characters in the buffer, should be written to. Let's have a look at a small example:

```
10 POKE 631,131:POKE 198,1

20 END
```

This will put 1 into the buffer counter and the token for SHIFT/RUN into the first location of the keyboard buffer and will load and run the next program on tape. It is also possible to put most other commands into the buffer (though not all at once) and have them execute when the program stops.

The ASCII codes for the commands can be calculated from the tables in your manual. If we wanted to hide the fact that anything was happening, the colours the commands were printed in could be set to the background colour, although messages would still have to be visible.

Each Basic command has a token, and this may be placed in the buffer rather than the full command (e.g. L SHIFT O instead of LOAD). In fact each command has a single number as a token. This is harder to calculate but may be gleaned by looking at the keyword table $A09E to A19D hex. This will allow you to place more commands into the buffer.

A limited auto-run

Now on to the auto-run. I have placed it from location $C000 to $C0E2 hex. This may not suit your needs. Remember to change the jumps and storage if you relocate the routine.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
C000 A5 2B        LDA $2B
C002 8D D9 C0     STA $C0D9
C005 A5 2C        LDA $2C
C007 8D DA C0     STA $C0DA
```

```
C00A A9 A5        LDA #$A5
C00C 85 2B        STA $2B
C00E 8D 02 03     STA $0302
C011 A9 02        LDA #$02
C013 85 2C        STA $2C
C015 8D 03 03     STA $0303
C018 A5 2D        LDA $2D
C01A 8D DB C0     STA $C0DB
C01D A5 2E        LDA $2E
C01F 8D DC C0     STA $C0DC
C022 A9 03        LDA #$03
C024 85 2E        STA $2E
C026 A9 04        LDA #$04
C028 85 2D        STA $2D
C02A A2 55        LDX #$55
C02C BD 83 C0     LDA $C083,X
C02F 9D A5 02     STA $02A5,X
C032 CA           DEX
C033 10 F7        BPL $C02C
C035 20 D4 E1     JSR $E1D4
C038 A9 03        LDA #$03
C03A 85 B9        STA $B9
C03C 20 59 E1     JSR $E159
C03F AD D9 C0     LDA $C0D9
C042 85 2B        STA $2B
C044 AD DA C0     LDA $C0DA
C047 85 2C        STA $2C
C049 AD DB C0     LDA $C0DB
C04C 85 2D        STA $2D
C04E AD DC C0     LDA $C0DC
C051 85 2E        STA $2E
C053 A9 ED        LDA #$ED
C055 8D 32 03     STA $0332
C058 A9 F5        LDA #$F5
C05A 8D 33 03     STA $0333
C05D A9 83        LDA #$83
C05F 8D 02 03     STA $0302
C062 A9 A4        LDA #$A4
C064 8D 03 03     STA $0303
C067 A9 00        LDA #$00
C069 85 9D        STA $9D
C06B A9 01        LDA #$01
C06D A2 01        LDX #$01
C06F A0 01        LDY #$01
C071 20 BA FF     JSR $FFBA
C074 A9 00        LDA #$00
C076 20 BD FF     JSR $FFBD
C079 A6 2D        LDX $2D
C07B A4 2E        LDY $2E
C07D A9 2B        LDA #$2B
C07F 20 D8 FF     JSR $FFD8
```

```
C082 60          RTS
C083 A9 83       LDA #$83
C085 8D 02 03    STA $0302
C088 A9 A4       LDA #$A4
C08A 8D 03 03    STA $0303
C08D A9 00       LDA #$00
C08F 85 9D       STA $9D
C091 20 D5 FF    JSR $FFD5
C094 A9 01       LDA #$01
C096 AA          TAX
C097 A8          TAY
C098 20 BA FF    JSR $FFBA
C09B A9 00       LDA #$00
C09D A2 00       LDX #$00
C09F A0 00       LDY #$00
C0A1 20 BD FF    JSR $FFBD
C0A4 A9 FB       LDA #$FB
C0A6 8D 28 03    STA $0328
C0A9 A9 F6       LDA #$F6
C0AB 8D 29 03    STA $0329
C0AE A9 02       LDA #$02
C0B0 8D 20 D0    STA $D020
C0B3 A9 00       LDA #$00
C0B5 20 D5 FF    JSR $FFD5
C0B8 86 2D       STX $2D
C0BA 86 2F       STX $2F
C0BC 86 31       STX $31
C0BE 84 2E       STY $2E
C0C0 84 30       STY $30
C0C2 84 32       STY $32
C0C4 A9 F6       LDA #$F6
C0C6 8D 29 03    STA $0329
C0C9 A9 ED       LDA #$ED
C0CB 8D 28 03    STA $0328
C0CE A9 00       LDA #$00
C0D0 20 5E A6    JSR $A65E
C0D3 20 8E A6    JSR $A68E
C0D6 4C AE A7    JMP $A7AE
C0D9 00          BRK
.
.
```

```
.:C000 A5 2B 8D D9 C0 A5 2C 8D
.:C008 DA C0 A9 A5 85 2B 8D 02
.:C010 03 A9 02 85 2C 8D 03 03
.:C018 A5 2D 8D DB C0 A5 2E 8D
.:C020 DC C0 A9 03 85 2E A9 04
.:C028 85 2D A2 55 BD 83 C0 9D
```

```
.:C030 A5 02 CA 10 F7 20 D4 E1
.:C038 A9 03 85 B9 20 59 E1 AD
.:C040 D9 C0 85 2B AD DA C0 85
.:C048 2C AD DB C0 85 2D AD DC
.:C050 C0 85 2E A9 ED 8D 32 03
.:C058 A9 F5 8D 33 03 A9 83 8D
.:C060 02 03 A9 A4 8D 03 03 A9
.:C068 00 85 9D A9 01 A2 01 A0
.:C070 01 20 BA FF A9 00 20 BD
.:C078 FF A6 2D A4 2E A9 2B 20
.:C080 D8 FF 60 A9 83 8D 02 03
.:C088 A9 A4 8D 03 03 A9 00 85
.:C090 9D 20 D5 FF A9 01 AA A8
.:C098 20 BA FF A9 00 A2 00 A0
.:C0A0 00 20 BD FF A9 FB 8D 28
.:C0A8 03 A9 F6 8D 29 03 A9 02
.:C0B0 8D 20 D0 A9 00 20 D5 FF
.:C0B8 86 2D 86 2F 86 31 84 2E
.:C0C0 84 30 84 32 A9 F6 8D 29
.:C0C8 03 A9 ED 8D 28 03 A9 00
.:C0D0 20 5E A6 20 8E A6 4C AE
.:C0D8 A7 00 00 00 00 00 00 00
.
.
```

The first part of the routine stores the values for the start of Basic and the start of Basic variables. It then resets the start of Basic to 677 decimal $02A5 hex (a good place for machine code). The loop from $C02C to $C034 takes the code from $C08A onwards and stores it at $02A5 onwards.

A save is then performed with a name given by the user. This saves off the code at $02A5 and the start of Basic and the other pointers are restored. The main program is then saved off immediatedly after this and the program ends.

The way to use this routine is by entering the following in direct mode:

```
SYS 49152"filename"
```

This will do the trick for a tape auto-run; the filename is optional. When you load the program back the routine from $02A5 is executed. It loads the rest of the program and disables the run/stop key. At the end of the load a run is executed and the program starts. This particular method is just one way of achieving an auto-run and may not suit your needs. Try experimenting with the program.

One warning when saving a program with this routine: do not try to stop it with the run/stop key. Other features could be built into the routine like scrambling the program or a routine to reset the machine if the run/stop is pressed. A routine to wipe out the auto – run after it has done its work may be a good idea.

3. Printer, Disk, Tape and Other Utilities

Some of these utilities originated from Germany (author unknown). I have updated and revised them, but my thanks for the ideas.

Hard copy

This is a hard copy routine. Although it is not a hi-res dump it can be quite useful. The routine is placed in zero page around the area labelled as the tape input error log ($0100 to $0200 hex). Some reading and experimentation will show that this area is also used for other things, and any routines should be placed here cautiously. However, one advantage of placing routines here is that you don't get an 'out of memory' message and have to new the Basic area, as you do with routines loaded at $C000 hex, for example. The routine can be loaded and executed in program or direct mode and is called with SYS 300.

It first places the current device number (in this case 4 for printer) into $00BA hex and the logical file number into $00B8 hex. The secondary address of 4 is placed into location $00B9 hex and the routine then branches to the Kernal routine at $FFC0 hex, which opens a file to the printer.

The next instruction opens the channel for output with a branch to the Kernal routine at $FFC9. The screen is dumped to the printer by using locations $0071 and $0072 hex as a pointer to the start of the screen and loading them into the accumulator where they are formatted for output.

A branch to the Kernal routine at $FFD2 hex (output character to channel) prints out the contents of the screen one line at a time. This continues until the end of the screen is reached and the two Kernal routines are used. The first, $FFCC hex, closes all input and output channels and the second, $FFC3 hex, closes the logical file. If nothing else this routine is a good example of using the Kernal routines on the 64, of which more later.

```
B*
   PC  SR AC XR YR SP
.;0008 72 00 01 21 F6
.
012C A9 04        LDA #$04
012E 85 BA        STA $BA
0130 A9 7E        LDA #$7E
0132 85 B8        STA $B8
0134 A9 00        LDA #$00
0136 A0 04        LDY #$04
0138 85 71        STA $71
013A 84 72        STY $72
013C 85 B7        STA $B7
013E 85 B9        STA $B9
0140 20 C0 FF     JSR $FFC0
0143 A6 B8        LDX $B8
0145 20 C9 FF     JSR $FFC9
0148 A2 19        LDX #$19
014A A9 0D        LDA #$0D
014C 20 D2 FF     JSR $FFD2
014F 20 E1 FF     JSR $FFE1
0152 F0 2E        BEQ $0182
0154 A0 00        LDY #$00
0156 B1 71        LDA ($71),Y
0158 85 67        STA $67
015A 29 3F        AND #$3F
015C 06 67        ASL $67
015E 24 67        BIT $67
0160 10 02        BPL $0164
0162 09 80        ORA #$80
0164 70 02        BVS $0168
0166 09 40        ORA #$40
0168 20 D2 FF     JSR $FFD2
016B C8           INY
016C C0 28        CPY #$28
016E D0 E6        BNE $0156
0170 98           TYA
0171 18           CLC
0172 65 71        ADC $71
0174 85 71        STA $71
0176 90 02        BCC $017A
0178 E6 72        INC $72
017A CA           DEX
017B D0 CD        BNE $014A
017D A9 0D        LDA #$0D
017F 20 D2 FF     JSR $FFD2
0182 20 CC FF     JSR $FFCC
0185 A2 7E        LDX #$7E
0187 4C C3 FF     JMP $FFC3
.
.
```

```
.:012C A9 04 85 BA A9 7E 85 B8
.:0134 A9 00 A0 04 85 71 84 72
.:013C 85 B7 85 B9 20 C0 FF A6
.:0144 B8 20 C9 FF A2 19 A9 0D
.:014C 20 D2 FF 20 E1 FF F0 2E
.:0154 A0 00 B1 71 85 67 29 3F
.:015C 06 67 24 67 10 02 09 80
.:0164 70 02 09 40 20 D2 FF C8
.:016C C0 28 D0 E6 98 18 65 71
.:0174 85 71 90 02 E6 72 CA D0
.:017C CD A9 0D 20 D2 FF 20 CC
.:0184 FF A2 7E 4C C3 FF 00 00
.
.
```

Old for new

Although there a few routines around that restore a program that has been NEWed (excluding Simon's Basic, unless you'll trust it), this one is in here as it is written for use at any time. It does not have to be in memory before the accidental NEW is entered. It can be loaded and called after you have lost the program and will quickly soothe the nerves and cure the cursing.

The routine is again located at $012C hex 300 decimal and is called with SYS 300. It does not really require a detailed description. It simply restores the pointers to the beginning of the program and restores your Basic program. Of course the best advice is always to save programs under development, making this sort of program obsolete.

```
B*
   PC  SR AC XR YR SP
.;0008 72 00 01 21 F6
.
012C A5 2B       LDA $2B
012E A4 2C       LDY $2C
0130 85 22       STA $22
0132 84 23       STY $23
0134 A0 03       LDY #$03
0136 C8          INY
0137 B1 22       LDA ($22),Y
0139 D0 FB       BNE $0136
013B C8          INY
013C 98          TYA
013D 18          CLC
013E 65 22       ADC $22
0140 A0 00       LDY #$00
```

```
0142 91 2B        STA ($2B),Y
0144 A5 23        LDA $23
0146 69 00        ADC #$00
0148 C8           INY
0149 91 2B        STA ($2B),Y
014B 88           DEY
014C A2 03        LDX #$03
014E E6 22        INC $22
0150 D0 02        BNE $0154
0152 E6 23        INC $23
0154 B1 22        LDA ($22),Y
0156 D0 F4        BNE $014C
0158 CA           DEX
0159 D0 F3        BNE $014E
015B A5 22        LDA $22
015D 69 02        ADC #$02
015F 85 2D        STA $2D
0161 85 23        STA $23
0163 69 00        ADC #$00
0165 85 2E        STA $2E
0167 4C 63 A6     JMP $A663
.
.
```

```
.:012C A5 2B A4 2C 85 22 84 23
.:0134 A0 03 C8 B1 22 D0 FB C8
.:013C 98 18 65 22 A0 00 91 2B
.:0144 A5 23 69 00 C8 91 2B 88
.:014C A2 03 E6 22 D0 02 E6 23
.:0154 B1 22 D0 F4 CA D0 F3 A5
.:015C 22 69 02 85 2D 85 23 69
.:0164 00 85 2E 4C 63 A6 00 00
.
.
```

Some disk routines

Disk error display

Any disk errors generated on the 64 have to be collected as they are not given, only indicated by an infuriating flashing light. The format for doing this usually looks like this:

```
10 OPEN1,8,15:REM OPEN CHANNEL

20 INPUT#1,A,B$,C,D:REM GET ERROR

30 PRINT A;B$;C;D:REM DISPLAY ERROR

40 CLOSE1:REM CLOSE FILE

50 END:REM END OR STOP ROUTINE
```

The above will give the error, but it can be awfully annoying to have to type this in, usually at the beginning of a program you are working on. Below is a routine in machine code that can be loaded and called at any time and will display the error.

The routine sits at $012C hex 300 decimal and will not disturb anything else. First the current device number is stored at $B8 hex (current device number). The secondary address is stored in location $B9 hex and the secondary address is sent after 'send talk' has been called with the Kernal routine at $FF96 hex.

The next two Kernal routines input a byte from the serial port $FFA5 hex and output character to channel $FFD2 hex. The routine branches to collect the next byte until a carriage return is found and a command is sent to the serial bus to 'untalk' with $FFAB hex, when the routine stops.

```
B*
   PC   SR AC XR YR SP
.;0008 72 00 01 21 F6
.
012C A9 08          LDA #$08
012E 85 BA          STA $BA
0130 20 B4 FF       JSR $FFB4
0133 A9 6F          LDA #$6F
0135 85 B9          STA $B9
0137 20 96 FF       JSR $FF96
013A 20 A5 FF       JSR $FFA5
013D 20 D2 FF       JSR $FFD2
0140 C9 0D          CMP #$0D
0142 D0 F6          BNE $013A
0144 20 AB FF       JSR $FFAB
0147 60             RTS
.
.
```

```
.:012C A9 08 85 BA 20 B4 FF A9
.:0134 6F 85 B9 20 96 FF 20 A5
.:013C FF 20 D2 FF C9 0D D0 F6
.:0144 20 AB FF 60 00 00 00 00
.
.
```

This may well be the best place for a list of the 1541 disk commands and error messages, and a comprehensive explanation of their meaning. An interesting thing about error messages is that they are often used to protect programs on disk. A particular error can be written on disk with a check in the program to collect and make sure it is the correct one. I believe writing a 29 error is currently popular.

Another way of protecting disks (probably the most effective) is to damage the disk physically and have the program check that a particular track and sector are damaged, and if it is not abort the program or wipe the disk clean.

The other method is to have a security key, as the word processor I am using (Paperclip) does. This can be very effective. With Paperclip you can remove the key while the word processor is in use and the 64 will freeze until the machine is turned off or the key re-inserted. This should stop any nosey or clumsy colleagues from copying or destroying your work.

Disk commands

OPEN	OPEN15,8,15<return> Opens a file (15). The device number is set to 8 (disk) and a command is sent (15)
BACK UP	PRINT#15,"D(x) = (y)"<return> Two drives are required for this operation and all files from drive y are copied to drive x.
DIRECTORY	LOAD"$",8<return> This will load the directory of any disk into memory, but replaces any program in memory.
VALIDATE	PRINT#15,"V"<return> This will validate a disk, but can take some time. It will often sort out any corrupt disks!

CLOSE	CLOSE<device number> <return> Closes a file
HEADER	PRINT#15,"N:<diskname>,<id>"<return> This will place a name of up to 16 characters with an ID of two characters. It will erase all information on the disk and takes approx. 80 seconds.
RENAME	PRINT#15,"R:newname=oldname"<return> The new name will replace the old name on the disk
SCRATCH	PRINT#15,"S:filename"<return> The action here takes the filename out of the directory, but does not wipe out the program, as is often assumed. Therefore the program can be restored!
INITIALISE	PRINT#15,"I" You should not really need this command as the drive should do the work for you. But you may need it one day.

That is about it for disk commands. The list is, as I am sure you will agree, quite insignificant compared to the Basic 4.0 commands. This is why utilities are constantly written for the 64. It has virtually no peripheral support - could it be a plot?

Disk error messages

Should you get it wrong or make a 'beep' up then you may get a flashing light on your drive. Having collected it with a puzzled brow, your brow could well be more puzzled by the error message. We had better have a look and try to decipher them.

0, OK, 00, 00	no errors were encountered
01, files scratched, 0<n>, 00	Returns the number of files scratched in <n>.
block header not found, 20, T, S	A block header was not found. Either the header has been destroyed, an illegal sector number was encountered or your 64

has flipped. In any event this spells trouble.

no sync character, 21, T, S	Either there is no disk present (silly!) or the read-write head is misaligned. At the very worst it could indicate a hardware failure.
data block not present, 22, T, S	This message indicates an illegal track and/or sector.
checksum error in data block, 23, T, S	This could be a general error on the checksum of the data or a problem with grounding (who wants to be grounded?).
byte decoding error, 24, T,S	This may also indicate grounding problems or an invalid bit pattern in the data byte. Don't ask me what to do, just keep struggling.
write verify error, 25, T, S	Only generated when the written data and the data in the DOS memory does not match.
write protect on, 26, T, S	This indicates that a write operation has been tried while the write protect switch is down. In other words you have probably got a write protect tab on your diskette which you must remove, or use another disk, or write down the program on paper? N.B. this message is not always generated by a write protect, which is confusing.

checksum error in header, 27, T, S	Again there maybe grounding problems, certainly if all three of these errors are occuring. It may just be a data error in the header.
long data block, 28, T, S	Caused by a bad diskette format or a hardware failure!
disk id mismatch, 29, T, S	The diskette either needs initialising or the header on the diskette is bad. Try causing this error.
general syntax, 30, T, S	The 1541 cannot make sense of the command just sent. Either there is an illegal number of filenames or the patterns are illegally used.
invalid command, 31, T, S	The command you sent was completely unrecognisable.
long line, 32, T, S	The command you sent was too long!
invalid file name, 33, T, S	Pattern matching is invalid in the save or open commands
no file given, 34, T, S	The filename was omitted or a " mark or : was omitted.
invalid dos command, 39, T, S	An unrecognised command was sent
record not present, 50, T, S	An INPUT# or GET# statement selected a record beyond the current end of file. This is only an

	error if you are attempting to read a record, not if you are positioning to the end of the file to write new records to an old file.
overflow in record, 51, T, S	A PRINT# statement was used to write more than the allowed number of characters to a relative file.
file too large, 52, T, S	The current record position will result in a disk overflow on the next write operation to disk.
write file open, 60, 00, 00	The file being used for a write is already open after being used for a read.
file not open, 61, 00, 00	This message is usually generated when the file being accessed has not been opened. This may not generate a message.
file not found, 62, 00, 00	The file being accessed does not exist.
file exists, 63, 00, 00	There is already a file on the diskette with filename being used in the command.
file type mismatch, 64, 00, 00	The file being used does not match the directory entry for this filename.
no block, 65, T, S	This message is generated when the B – A command finds the block to be allocated has already been allocated. The numbers give the

	next available track and sector. If zero then all blocks are in use.
illegal track and sector, 66, T, S	This message is generated when an attempt has been made to access a sector that does not exist. The track or sector is out of range.
illegal sys/track & sector, 67, T, S	An attempt has been made to access a reserved sector.
no channel, 70, 00, 00	The channel is not available or too many files are open.
dir, 71, 00, 00	The diskette needs initialising as the BAM does not match the internal count. You may lose some files with this one.
disk full, 72, 00, 00	Either the disk is full (all blocks used) or the number of entries is at its limit (152). When the disk is nearly full it may be difficult to write @ a file. In this case scratching the file and re-saving should work.
dos mis-match, 73, 00, 00	This error is generated when an attempt is made to write to a disk initialised on a different DOS. However, you may read disks initialised by different versions of the same DOS.

drive not ready, 74, 00, 00	The drive will not accept commands, commonly because the drive door is open or there is no diskette in the drive.

Disk directory

The usual way of viewing the directory is to load it into memory and list it. This can be a pain as it means that any program currently in RAM will be overwritten, and you will have to save it off first and then load the directory. After this you will have to load your program back into the 64 in order to continue. This can be very time consuming.

Therefore a routine that allows you to display the directory of the disk without loading it into memory is a must. This is exactly what the following routine does.

It is located at our favourite position $012C hex 300 decimal and is therefore loadable and usable at any time. The routine is called with SYS 300. The routine first sets the parameters and calls the Kernal routine to send 'SA'. It then calls the Kernal routine to command the serial bus to 'talk' $FFB4.

The next call is to the Kernal routine to send the secondary address $FF96 and the call to $FFA5 calls the routine to input a byte from the serial port. The routine to print a line number is used to display the directory $BDCD and the routine to output the character to channel $FFD2 completes the program until the directory has been displayed and the call to $F642 sends an 'untalk' before quitting the program.

```
B*
   PC  SR AC XR YR SP
.;0008 72 00 01 21 F6
.
012C A9 24        LDA #$24
012E 85 FB        STA $FB
0130 A9 FB        LDA #$FB
0132 85 BB        STA $BB
0134 A9 00        LDA #$00
0136 85 BC        STA $BC
0138 A9 01        LDA #$01
013A 85 B7        STA $B7
013C A9 08        LDA #$08
013E 85 BA        STA $BA
```

```
0140 A9 60       LDA #$60
0142 85 B9       STA $B9
0144 20 D5 F3    JSR $F3D5
0147 A5 BA       LDA $BA
0149 20 B4 FF    JSR $FFB4
014C A5 B9       LDA $B9
014E 20 96 FF    JSR $FF96
0151 A9 00       LDA #$00
0153 85 90       STA $90
0155 A0 03       LDY #$03
0157 84 FB       STY $FB
0159 20 A5 FF    JSR $FFA5
015C 85 FC       STA $FC
015E A4 90       LDY $90
0160 D0 2F       BNE $0191
0162 20 A5 FF    JSR $FFA5
0165 A4 90       LDY $90
0167 D0 28       BNE $0191
0169 A4 FB       LDY $FB
016B 88          DEY
016C D0 E9       BNE $0157
016E A6 FC       LDX $FC
0170 20 CD BD    JSR $BDCD
0173 A9 20       LDA #$20
0175 20 D2 FF    JSR $FFD2
0178 20 A5 FF    JSR $FFA5
017B A6 90       LDX $90
017D D0 12       BNE $0191
017F AA          TAX
0180 F0 06       BEQ $0188
0182 20 D2 FF    JSR $FFD2
0185 4C 78 01    JMP $0178
0188 A9 0D       LDA #$0D
018A 20 D2 FF    JSR $FFD2
018D A0 02       LDY #$02
018F D0 C6       BNE $0157
0191 20 42 F6    JSR $F642
0194 60          RTS
.
.
```

```
.:012C A9 24 85 FB A9 FB 85 BB
.:0134 A9 00 85 BC A9 01 85 B7
.:013C A9 08 85 BA A9 60 85 B9
.:0144 20 D5 F3 A5 BA 20 B4 FF
.:014C A5 B9 20 96 FF A9 00 85
.:0154 90 A0 03 84 FB 20 A5 FF
.:015C 85 FC A4 90 D0 2F 20 A5
```

```
.:0164 FF A4 90 D0 28 A4 FB 88
.:016C D0 E9 A6 FC 20 CD BD A9
.:0174 20 20 D2 FF 20 A5 FF A6
.:017C 90 D0 12 AA F0 06 20 D2
.:0184 FF 4C 78 01 A9 0D 20 D2
.:018C FF A0 02 D0 C6 20 42 F6
.:0194 60 00 00 00 00 00 00 00
.
.
```

Disk directory and auto-load

This program will display the directory in a different format and only a page at a time. Each file is given a letter and may be loaded by pressing that letter as long as it is a program and not a sequential or relative file.

The program also mixes Basic with machine code and will be useful to explain how to achieve this. This routine was converted from a similiar one written for the PET, which has been substantially modified.

The Basic program should be entered exactly as shown, as the machine code is placed directly after the Basic program and will not function if any additions or deletions are made. The 64's control characters have been replaced by more readable and understandable characters: refer to the symbol chart at the beginning of the book for details.

After entering the Basic program, save it and verify it to make sure that you have a correct copy. At this point it is advisable to turn your 64 on and off again. You will now need to load a monitor to enter the machine code.

If you are using Supermon then load and run it. Use the 'X' command to quit the monitor and 'new' the program. Re-enter the monitor with SYS8<return> and enter: M 0D44 0DB4. This is where the code is to be entered. Using the memory dump below enter the code carefully and save it with the following command: SAVE"CODE",08,0D44,0DBC, or SAVE"CODE",01,0D44,0DBC for tape.

At this point you have a copy of the Basic program and the code, hopefully both correct. They now need to be merged and saved together. To do this, reload the Basic program and afterwards reload the code. This can be done by entering Supermon and loading the code. Both the programs need to be saved to tape or disk. To do this save the programs with the following: SAVE"<filename>",08,0801,0DBC or SAVE"<filename>,01,0801,0DBC for tape. The program can now be loaded from Basic in the normal way and will load both Basic and machine code programs into the right place.

This is the time to try the program. Enter run and see what happens. You should get a nicely formatted display of the directory with the letter for each file on the far right of the screen. If this is so, well done! Try loading a program, if this works then you succeeded first time and can go to the top of the class.

If the program does not work and crashes the 64 or does something equally odd, you have an error in the code or the Basic program is the wrong length. You will have to check both carefully, make the changes, and then save the whole thing as described above.

If you get a Basic error message then the problem may only be an error in the Basic program. After correcting it, however, you will need to save it in the way described above. Good luck with this one, and let me assure you that the results are worth the effort.

```
1 REM *** AN EASY WAY TO LOAD THE DIRECTORY AND L
OAD A PROGRAM FROM DISK
2 GOTO32
4 FI$="
      ":FI$=LEFT$(FI$,EE)
5 X$=MID$(STR$(PEEK(252)*256+PEEK(251)),2):RETURN
6 SYS(C):GOSUB4:Z$(Ø)=FI$:C=3399
7 GOSUB16
8 FORB=1TOBB:SYS(C):GOSUB4:Z$(B)=FI$:IFSTTHENB=BB
+1:GOTO14
9 Z$(B)=X$+S2$+Z$(B)
1Ø PRINT"  "Z$(B);:PRINTTAB(33)"...[ON] "CHR$(VR)"
 [OFF]":VR=VR+1
11 IFPEEK(DO)<22THEN14
12 GOSUB17:VR=65:G=G+1
13 GOSUB16
14 NEXT:PRINTVV$:GOSUB18
15 GOTO43
16 PRINTRR$W$"DISK NAME    [ON]"Z$(Ø)DF$SS$DF$:RET
URN
17 PRINTY$
18 GOSUB3Ø:IFFI$="  "ANDB<BBTHENRETURN
19 IFFI$="  "THEN43
2Ø IFASC(FI$)<65ORASC(FI$)>(VR-1)THEN18
21 NN=ASC(FI$):NN=(NN-64)+16*(G-1):RE$=Z$(NN):RE$
=MID$(RE$,8,16)
22 IFMID$(Z$(NN),25,3)="PRG"THEN27
```

```
23 PRINTOT$"[ON]ERROR[OFF].. [CL]NOT PROGRAM..SPA
CE TO CONTINUE"
24 GOSUB30:IFFI$<>" "THEN24
25 IFB<BBTHEN17
26 PRINTVV$:GOTO18
27 P=LEN(RE$):IFRIGHT$(RE$,1)=" "THENRE$=LEFT$(RE
$,P-1):GOTO27
28 PRINT"[CLR]LOAD"CHR$(34)RE$CHR$(34)",08":PRINT
"[4 CD]RUN":CLOSE1:CLOSE15
29 POKE631,19:POKE632,13:POKE633,13:POKE198,3:END
30 GETFI$:IFFI$=""THEN30
31 RETURN
32 PRINT"[CLR]":POKE53272,23:C=3396:G=1:VR=65:DO=
214:BB=245:EE=24:OPEN15,8,15
33 DIMZ$(BB):OPEN1,8,0,"$0":GOSUB41
34 DF$=CHR$(13):OT$="[HME][23 CD]":W$=DF$+DF$
35 RR$="[CLR]  [ON]  [SH DISK] AUTO LOAD[ON]    F
OR THE [SH CBM] 64  [OFF]"
36 S1$="      ":S2$="    "
37 VV$=OT$+"LOAD TYPE <LETTER>...TERMINATE <SPACE
>"
38 Y$=OT$+"CONTINUE <SPACE>...LOAD <TYPE LETTER>"
39 SS$=DF$+"[ON][SH BLOCKS][OFF]  [ON][SH PROGRAM
 TITLE]"
40 SS$=SS$+"[OFF]    [ON][SH TYPE[ON]       [ON][S
H LOAD[OFF]:GOTO6
41 INPUT#15,EN$,EM$:IFEN$="00"THENRETURN
42 PRINT"[CLR]  [ON][SH D]ISK [SH ERROR][OFF]"DF$
EM$
43 CLOSE1:CLOSE15:END
```

Using the disassembly given below we can dissect the machine code part of the program. The first thing the routine does when called is to perform two jumps to other routines in the program. I suppose you could say that there are three separate routines here.

The first jump to $0D9E sets the input device and branches to the input routine. The rest of the routine collects the directory from the disk and stores it at $085B onwards, this is line 4 of the Basic program, into FI$. This is yet another reason to ensure that the Basic program is entered exactly as shown, including the REM statement and the following comments.

Now for a look at the Basic program: the first active instruction at line 2 jumps to line 32. The screen is cleared and the variable 'C' set, the 64 is then put into lower case with POKE 53272,23. Other variables are initialised and a file opened to disk. Line 33 is obvious except for the OPEN1,8,0"$0", try it, it's interesting!

At line 40 control jumps to line 6 of our Basic program and our machine code routine is called for the first time using the value of (C). Then a branch to line 4 collects the value of FI$ and takes the block count for the file from temporary storage in $FB and $FC hex. The variable 'C' is reset 3 higher than its original value and the program branches to the routine to display the directory at line 16.

The file name and block count are collected for each disk entry and displayed until it ends or PEEK (DO) is equal to 22. The value of DO is 214. This is the current line the cursor is on. So the display stops when the cursor has moved 22 lines down the screen.

A message is printed giving a choice of continuing with the directory or loading one of the programs displayed. Lines 28 and 29 in the Basic program load and run the program. This is done by clearing the screen and printing 'load' and the first quotation mark CHR$(34). The file-name is displayed RE$ and the closing quotes printed. The cursor is positioned four lines down and 'run' is printed.

At this point the files to disk are closed. Line 29 places three characters in the keyboard buffer: they place the cursor at the home position and two carriage returns, one over the load and the second over the run. The program is of course loaded and run as long as it is a program entry on the disk.

```
B*
   PC  SR AC XR YR SP
.;0008 72 00 01 21 F6
.
0D44 4C 9E 0D     JMP $0D9E
0D47 4C 4A 0D     JMP $0D4A
0D4A 20 A4 0D     JSR $0DA4
0D4D A5 FB        LDA $FB
0D4F C9 64        CMP #$64
0D51 B0 12        BCS $0D65
0D53 A9 20        LDA #$20
0D55 E8           INX
0D56 9D 5B 08     STA $085B,X
0D59 A5 FB        LDA $FB
0D5B C9 0A        CMP #$0A
0D5D B0 06        BCS $0D65
0D5F A9 20        LDA #$20
0D61 E8           INX
0D62 9D 5B 08     STA $085B,X
0D65 A0 04        LDY #$04
0D67 C8           INY
0D68 A5 90        LDA $90
```

```
0D6A D0 2D        BNE $0D99
0D6C 20 57 F1     JSR $F157
0D6F C9 22        CMP #$22
0D71 F0 02        BEQ $0D75
0D73 D0 F2        BNE $0D67
0D75 C8           INY
0D76 E8           INX
0D77 20 57 F1     JSR $F157
0D7A C9 22        CMP #$22
0D7C F0 06        BEQ $0D84
0D7E 9D 5B 08     STA $085B,X
0D81 38           SEC
0D82 B0 F1        BCS $0D75
0D84 18           CLC
0D85 20 57 F1     JSR $F157
0D88 C9 29        CMP #$29
0D8A B0 02        BCS $0D8E
0D8C A9 20        LDA #$20
0D8E 9D 5B 08     STA $085B,X
0D91 E8           INX
0D92 C8           INY
0D93 C0 20        CPY #$20
0D95 F0 02        BEQ $0D99
0D97 D0 EB        BNE $0D84
0D99 A9 00        LDA #$00
0D9B 85 99        STA $99
0D9D 60           RTS
0D9E 20 A4 0D     JSR $0DA4
0DA1 4C 65 0D     JMP $0D65
0DA4 A2 01        LDX #$01
0DA6 20 0E F2     JSR $F20E
0DA9 A2 00        LDX #$00
0DAB 20 57 F1     JSR $F157
0DAE 20 57 F1     JSR $F157
0DB1 20 57 F1     JSR $F157
0DB4 85 FB        STA $FB
0DB6 20 57 F1     JSR $F157
0DB9 85 FC        STA $FC
0DBB 60           RTS
.
.
```

```
.:0D44 4C 9E 0D 4C 4A 0D 20 A4
.:0D4C 0D A5 FB C9 64 B0 12 A9
.:0D54 20 E8 9D 5B 08 A5 FB C9
.:0D5C 0A B0 06 A9 20 E8 9D 5B
```

```
.:0D64 08 A0 04 C8 A5 90 D0 2D
.:0D6C 20 57 F1 C9 22 F0 02 D0
.:0D74 F2 C8 E8 20 57 F1 C9 22
.:0D7C F0 06 9D 5B 08 38 B0 F1
.:0D84 18 20 57 F1 C9 29 B0 02
.:0D8C A9 20 9D 5B 08 E8 C8 C0
.:0D94 20 F0 02 D0 EB A9 00 85
.:0D9C 99 60 20 A4 0D 4C 65 0D
.:0DA4 A2 01 20 0E F2 A2 00 20
.:0DAC 57 F1 20 57 F1 20 57 F1
.:0DB4 85 FB 20 57 F1 85 FC 60
.
.
```

Tape control

First, here is a Basic program that gives you control over the tape motor.

```
10 A = PEEK(1) OR 32:B = PEEK (1) AND 16

20 POKE 192,A:POKE 1,A

30 PRINT "[CLR] TAPE MOTOR STOPPED"

40 IF B <> 0 THEN 60

50 PRINT "[CD] PRESS STOP ON TAPE"

60 IF PEEK (1) AND 16 = 0 THEN 60

70 PRINT "[CD] ALL SWITCHES OFF"

80 END
```

This small program might be better written in machine code as it uses the all too sensitive location 1. The variable 'A' is set up in line 10 and used in line 20 to stop the tape motor by placing 'A' in location 1 and location 192 (tape motor interlock). Line 30 merely confirms that the tape motor has stopped, and line 40 checks to see if the play key is pressed. Lastly line 60 waits until all the keys on the tape are off.

If you intend to write programs that directly control the tape motor then it is advisable to become familiar with location 1 and location 192.

Tape Search

This is another Basic program that uses tape control. It simply allows you to load your programs quickly by saving them at set points on the tape. Tape Search can be useful for saving programs onto tape and locating them quickly in order to load them.

The program actually eliminates a lot of the drudgery from using a cassette deck and waiting sleepily while the program is found and then loads. As listed here the programs have been given dummy names (PROG 1 to PROG 9), and your program names should be inserted in these places, giving you a menu of your tape.

Tape Search should ideally be placed at the beginning of each cassette used and the program names added to the menu. You could have many more than the nine places made available in this program.

There are several ways of using the program. You could place the programs on the tape and then alter the timing within the program to stop at the the right place. Alternatively Tape Search could be recorded on the cassette and each program added and recorded as you go along. This would be a simpler and faster method, as the timing could be adjusted simply and quickly to stop the tape at the correct position.

Tape Search is set up to present a menu of nine programs giving the user fast access to the position on the tape of any one of those programs. When the program is RUN the menu is displayed on two screens. To move between these screens use the F7 (forward) and F5 (back) keys.

To load a program choose the relevant number (1 – 9). The program then checks for the PLAY button on the recorder and if it is depressed displays a message and waits for it to be released. Once this has been accomplished the program asks for the fast forward button to be depressed and searches for the program using the TI function and the user input.

Once the program has reached the required time it halts, waits for the fast forward button to be released, NEW(s) itself, LOAD(s) and RUN(s) the program at that particular position on the tape. It is probably best to leave about ten cassette digits between programs.

Explanation

A detailed look at the program is probably the best way to explain how to use it.

Line 200 resets two Basic pointers, assuming that other programs may have been in memory. You may have to add other statements to reset the 64 if you have been using programs which alter important pointers.

Line 300 sets screen and border colours and prints the title.

Lines 400 – 1100 are two screens of instructions for using the program.

Lines 1200 – 1300 format the screen headings.

Lines 1400 – 1500 and lines 1900 – 2000 are the spaces for the titles of user programs to be inserted.

Line 1600 branches to the routine that waits for F7 to be pressed.

Line 2100 waits for the F7 key (select) or the F5 key (return to start of menu) to be pressed.

Line 2300 returns the program to the start of the memory if the F5 key is pressed.

Line 2500 is the input for the number of the program the user wishes to access. The number selected is put in the variable J.

Line 2600 checks that the input was within range, in this case between 1 and 9.

Line 2700 jumps to the routine to load the first program if 1 is selected.

Lines 2800 – 3500 set the variable Q according to the value held in J. The number placed in Q is used to determine the positioning of the tape.

Lines 3700 – 3800 both look at memory location 1, which is the 6510 I/O port, and bit four, which is the switch cassette sense. Line 3700 checks to see if any keys on the cassette are depressed. If they are it prints a message. Line 3800 waits until the key is released.

Line 3900 prints a message.

Line 4000 checks location 1 (bit 4) and waits until a key is pressed on the tape. This is checking for any button on the cassette so make sure you depress the fast forward.

Line 4100 prints a message and sets A equal to TI.

Line 4200 halts program execution until the statement is true and then continues. Therefore by adjusting the values of Q the user may lengthen or shorten this delay while the tape continues.

Line 4300 stops the cassette motor with a poke to location 1 (bit 5) and a poke to location 192. Both of these locations have to be altered to start or stop the tape motor.

Line 4500 waits for the fast forward key to be released.

Line 4800 puts five into the count for the keyboard buffer and pokes the values for NEW, LOAD and a carriage return into the keyboard buffer. These statements come into effect as soon as the program finishes.

Lines 4900 – 5100 are the routine to wait for the F7 key to be pressed.

If you attempt to alter this program, be sparing with any random experiments affecting location 1, as any mistakes will probably cause your 64 to nod off, and to wake it you will have to power down!

```
100 REM *** RESET BASIC POINTERS
200 POKE56,160:POKE52,160:CLR
300 POKE53280,1:POKE53281,2:PRINT"[CLR]"SPC(14)"[
5 CD][YEL]TAPE SEARCH"
400 PRINTSPC(14)"[3 CD]FOR QUICK"
500 PRINTSPC(14)"[3 CD]AND EASY ACCESS":PRINTSPC(
14)"[3 CD]TO YOUR PROGRAMS"
600 PRINTSPC(14)"[3 CD][ON]PRESS F7 TO CONTINUE"
700 GETA$:IFA$<>"[F7]"THEN700
800 PRINT"[CLR][4 CD]"SPC(10)"SIMPLY PLACE  THE":
PRINTSPC(10)"[3 CD]NAMES OF YOUR"
900 PRINTSPC(10)"[3 CD]PROGRAMS IN THE BLANK SPAC
ES"
1000 PRINTSPC(10)"[6 CD][ON]PRESS F7 TO CONTINUE"
1100 GETA$:IFA$<>"[F7]"THEN1100
1200 PRINT"[CLR]SPC(9)"[CYN]MENU":PRINT"[CD][YEL]
 #"SPC(4)"PROGRAM"
1300 PRINT"[WHT][12 SH E]"SPC(3)"[12 SH E]"
```

```
1400 PRINT"[2 CD] 1"SPC(14)"PROG 1":PRINT"[2 CD]
2"SPC(14)"PROG 2"
1500 PRINT"[2 CD] 3"SPC(14)"PROG 3":PRINT"[2 CD]
4"SPC(14)"PROG 4":PRINT"2 CD] 5"SPC(14)"PROG 5"
1600 GOSUB4900
1700 PRINT"[CLR]"SPC(9)"[CYN]MENU":PRINT"[CD][YEL
] #"SPC(4)"PROGRAM"
1800 PRINT"[WHT][12 SH E]"SPC(3)"[12 SH E]"
1900 PRINT"[CD] 6"SPC(14)"PROG 6":PRINT"[CD] 7"SP
C(14)"PROG 7"
2000 PRINT"[2 CD] 8"SPC(14)"PROG 8":PRINT"[2 CD]
9"SPC(14)"PROG 9"
2100 PRINT"[2 CD]        [ON]PRESS F7 TO SELECT":PR
INT"[CD]        OR F5 TO RETURN TO MENU"
2200 GETA$:IFA$<>"[F7]"ANDA$<>"[F5]"THEN2200
2300 IFA$="[F5]"THEN1200
2400 REM *** SET Q FOR TIMING
2500 INPUT"[2 CD]        SELECT # :";J:PRINT
2600 IFJ<1ORJ>9THEN1200
2700 IFJ=1THEN4600
2800 IFJ=2THENQ=1.5
2900 IFJ=3THENQ=2.8
3000 IFJ=4THENQ=3.7
3100 IFJ=5THENQ=4.5
3200 IFJ=6THENQ=6.7
3300 IFJ=7THENQ=7.6
3400 IFJ=8THENQ=8.65
3500 IFJ=9THENQ=12.9
3600 REM *** SET UP CASSETTE AND GO FORWARD
3700 IF(PEEK(1)AND16)=0THENPRINT"[CLR][12 CD][9 C
R]PRESS STOP ON CASSETTE"
3800 IF(PEEK(1)AND16)=0THEN3800
3900 PRINT"[CLR][11 CD][10 CR]PRESS FAST FORWARD"
:PRINT
4000 IF(PEEK(1)AND16)=16THEN4000
4100 PRINT"[CLR]":PRINTSPC(20)"[11 CD]OK":PRINT:A
=TI
4200 IFABS(TI-A)<(Q*360)THEN4200
4300 Z=PEEK(1):POKE192,ZOR32:POKE1,ZOR32
4400 PRINT"[CLR][11 CD][10 CR]RELEASE FAST FORWAR
D"
4500 IF(PEEK(1)AND16)=0THEN4500
4600 PRINT"[CLR]"
4700 REM *** NEW PROG AND LOAD PROG
4800 POKE198,5:POKE631,78:POKE632,69:POKE633,87:P
OKE634,13:POKE635,131:END
4900 PRINT"[3 CD]        [ON]PRESS F7 TO CONTINUE
"
5000 GETA$:IFA$<>"[F7]"THEN5000
5100 RETURN

READY.
```

Word processor

A short routine that could be built into a word processor. It will work as it stands on any Commodore machine.

The program does not produce a prompt, but waits for any input (maximum of 88 characters). It will carry on inputting and displaying characters until a carriage return is executed. This is a simple way to start inputting and displaying formatted text on the screen.

```
10 OPEN4,0:REM OPEN KEYBOARD AS A DEVICE
20 PRINTCHR$(147);: REM CLEARS SCREEN
30 DIM A$ (100):REM SET UP ARRAY FOR TEXT STORAGE
40 INPUT#4,A$(I)
50 FOR I = 0 TO 100:REM INPUT LOOP FOR TEXT
60 PRINT:REM SKIP TO START OF NEXT LINE
70 IF A$(I) = "" THEN I=100:REM TEST FOR END OF PR
INT LOOP
80 NEXT:REM END OF INPUT LOOP
90 FOR I = 0 TO 100:REM PRINTING OF TEXT LOOP
100 IF A$ (I) = "" THEN 170:REM TEST FOR END OF PR
INT LOOP
110 FOR J = 1 TO LEN(A$(I)):REM LOOP FOR LENGTH OF
 STRING
120 B$ = MID$ (A$(I),J,1):REM B$ = JTH CHARACTER F
ROM STRING
130 IF B$ = "!" THEN PRINT:GOTO200
140 REM DO CARRIAGE RETURN IF EXCLAMATION MARK
150 PRINT B$;:REM PRINT CHARACTER OF TEXT
160 NEXT J,I:REM CLOSE LOOPS
170 CLOSE 4:REM CLOSE KEYBOARD CHANNEL
180 END
```

Sell that 1540

If you happen still to have a 1540 drive then there is a way to load some programs from the 1540 into the 64. This will work with most, but not all programs.

The problem with loading programs from the 1540 into the 64 is the screen refresh. The 64 will keep the screen on while it tries to load programs from the 1540. This will cause the 1540 to whirr madly and not much else.

However, if the screen is turned off before loading, saving or verifying from the 1540, you will have more success. The 64's screen is turned off with POKE 53265,11 and on again with POKE 53265,27. This proves to be tricky as one has to type blind, so here is a little tip for setting up the screen to load, save or verify.

The screen should look like this:

POKE 53265,11:REM top line of screen

(leave blank)

(leave blank)

LOAD"<prog name>",8:REM load prog

(leave blank)

(leave blank)

(leave blank)

(leave blank)

POKE 53265,27:REM bring screen back

To do this the first statement should be on the top line of the screen. DO NOT press return until all the lines have been typed in, instead press SHIFT RETURN. Having typed in the last line, press HOME (unshifted) and press RETURN, which will blank the screen. The next return should load the program and the last bring the screen back.

Dumping the screen

This is a Basic program that will dump the screen to printer. It is set up for Commodore printers, but with small alterations should work on most. The routine is formatted specifically for the 64's screen.

```
10 OPEN6,4,6:PRINT#6,CHR$(18):CLOSE18
20 OPEN4,4:CMD4
30 FORI=0TO24
40 FORJ=0TO39
50 A=PEEK(1024+I*40+J)
60 GOSUB200
```

```
70 PRINTA1$;A2$;A3$;
80 NEXT
90 PRINT
100 NEXT
110 PRINT#4:CLOSE4
120 END
200 A1$="":A2$="":A3$=""
210 IFA1>127THENA1$=CHR$(18):A3=CHR$(146):A=A-128:
RETURN
220 IFA<32THENA2$=CHR$(A+64):RETURN
230 IFA>31ANDA<64THENA2$=CHR$(A):RETURN
240 IFA>63ANDA<96THENA2$=CHR$(A+128):RETURN
250 A2$=CHR$(A+64):RETURN
```

More memory

If you should feel cheated by the amount of RAM that is actually available when you switch the 64 on (see power up screen), study the following short routine:

```
7000 LDA $01

7002 AND #$FE

7004 STA $01

7006 RTS
```

This will give you another 8K of usable memory (from $A000 to BFFF hex) but be warned that this can only be done in machine code. Attempts to use a Basic program to do this will only crash the 64.

To return the 64 to the normal configuration use the following routine:

```
70A0 LDA $01

70A2 ORA #$01

70A4 STA $01

70A6 RTS
```

Merging and appending programs

Merge

This is a tricky little routine that will merge two Basic programs from tape. The technique was first outlined by Jim Butterfield.

First, save the lines to be merged onto tape with:

OPEN1,1,1,"FILENAME":CMD1:LIST<return>

When this operation has finished, enter:

PRINT#1:CLOSE1<return>

To merge the program you have just saved with the program in memory rewind the tape (of course). Now enter:

POKE19,1:OPEN1<return>

When the ready message appears, clear the screen (shift and clr/home). Press the cursor down key three times and enter the following:

PRINTCHR$(19):POKE198,1:POKE631,13:POKE153,1<return>

The tape will finally stop and return an error message. For once you can ignore this, as all is well. Have a look and you will find that your two programs are now merged!

Append

Now for a routine to join one program to another. This, unlike the merge program, does not renumber the lines. It merely joins one Basic program to the end of another, and the one being joined should have higher line numebers if the routine is to make any sense at all!

To do it from Basic is fairly simple, but will give us a good insight into the general technique. Enter the following:

```
100 PRINT"THIS IS THE SECOND PART"
```

```
110 PRINT"OF OUR APPEND PROGRAM"

120 PRINT"WE ARE WRITING IT FIRST"

130 PRINT"SO THAT IT CAN BE SAVED"

140 PRINT"BEFORE WE ENTER THE FIRST"

150 PRINT"PART AND APPEND THIS PART"

160 END
```

Now save the program to disk or tape and enter the following:

```
10 PRINT"[CLR]THIS IS THE FIRST PART"

20 PRINT"OFF OUR PROGRAM AND THIS"

30 PRINT"WILL REMAIN IN MEMORY"

40 PRINT"WHILE WE TAG THE SECOND"

50 PRINT"PART ON TO THE END"
```

Now clear the screen and enter the following in direct mode:

```
PRINT PEEK(43),PEEK(44)<return>
```

You will get 1 and 8, or at least you should. Scratch your head a lot if you don't. These numbers represent the start of Basic. You may need to remember them.

Now some more entering in direct mode:

```
POKE 43,PEEK(45)-2:POKE 44,PEEK(46)<return>

LOAD"PART TWO"<return>
```

When the ready message comes back enter:

```
POKE 43,1:POKE 44,8<return>
```

The two programs are now merged and can be saved to tape or disk. The key here is to take the start of Basic pointers (43 and 44) and alter them to point at the end of the current program, using the end of program pointers (45 and 46). You can then load in the program to be appended and save the program off only after the start of Basic pointers have been reset (43 and 44).

Now that you understand the method in general we need a program that will do this for us. The program below starts at location $012C hex 300 decimal.

The idea is the same as outlined above with our two Basic programs. Line numbers must not be duplicated, but should be consecutive. The routine first sets up a load by placing a zero in location $0A hex and then branches to the routine to set the parameters for the load ($E1D4 hex).

The pointer for the start of Basic variables is set to the actual end of the program. This means that the zeros indicating the end of the Basic program are subtracted from the pointers. The program to be loaded is then called from tape or disk ($FFD5 hex) and the routine to re-chain the Basic lines is called ($A533 hex).

The rest of the routine loops through the program until the Basic program is appended and the pointers are reset. This routine is called in the following way:

SYS300"(filename)",dn<return>

where filename is the name of the program to be appended and 'dn' is the device number.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
012C A9 00        LDA #$00
012E 85 0A        STA $0A
0130 20 D4 E1     JSR $E1D4
0133 A5 2D        LDA $2D
0135 38           SEC
0136 E9 02        SBC #$02
0138 AA           TAX
0139 A5 2E        LDA $2E
013B E9 00        SBC #$00
013D A8           TAY
013E A5 0A        LDA $0A
0140 20 D5 FF     JSR $FFD5
0143 20 33 A5     JSR $A533
```

```
0146 A5 2D        LDA $2D
0148 A4 2E        LDY $2E
014A 38           SEC
014B E9 02        SBC #$02
014D 85 57        STA $57
014F 98           TYA
0150 E9 00        SBC #$00
0152 85 58        STA $58
0154 A0 00        LDY #$00
0156 B1 57        LDA ($57),Y
0158 D0 1B        BNE $0175
015A C8           INY
015B B1 57        LDA ($57),Y
015D D0 16        BNE $0175
015F A5 57        LDA $57
0161 18           CLC
0162 69 02        ADC #$02
0164 85 2D        STA $2D
0166 85 2F        STA $2F
0168 85 31        STA $31
016A A5 58        LDA $58
016C 69 00        ADC #$00
016E 85 2E        STA $2E
0170 85 30        STA $30
0172 85 32        STA $32
0174 60           RTS
0175 A0 00        LDY #$00
0177 B1 57        LDA ($57),Y
0179 85 59        STA $59
017B C8           INY
017C B1 57        LDA ($57),Y
017E 85 58        STA $58
0180 A5 59        LDA $59
0182 85 57        STA $57
0184 4C 54 01     JMP $0154
.
.:012C A9 00 85 0A 20 D4 E1 A5
.:0134 2D 38 E9 02 AA A5 2E E9
.:013C 00 A8 A5 0A 20 D5 FF 20
.:0144 33 A5 A5 2D A4 2E 38 E9
.:014C 02 85 57 98 E9 00 85 58
.:0154 A0 00 B1 57 D0 1B C8 B1
.:015C 57 D0 16 A5 57 18 69 02
.:0164 85 2D 85 2F 85 31 A5 58
.:016C 69 00 85 2E 85 30 85 32
.:0174 60 A0 00 B1 57 85 59 C8
.:017C B1 57 85 58 A5 59 85 57
.:0184 4C 54 01 00 00 00 00 00
.
.
```

4. New Commands and Interrupts

Having just spent most of the night trying to write a 'pop' command for the 64, I thought this must be the place to talk about adding commands to Basic. I don't think my 64 would agree with me, as it cowers in the corner from the night's abuse.

Interrupts

So perhaps we will look at interrupts first. The interrupt is a routine in the 64 that does all the housework, checks the keyboard and updates timing and the screen. It does all this approximately 60 times a second.

If one is very careful, the interrupts can be momentarily diverted from their housekeeping to a routine that we have written. There are some elementary rules to remember. The routine that the interrupt is diverted to will add time to the interrupt and the last instruction in our routine should send the interrupts back to their housekeeping.

We are aiming to have the interrupt check our routine and speed it up. Below are two interrupt-driven routines that will demonstrate the technique and enable you to understand it better. They both slow the 64 down considerably, but will serve as demos. Both of these routines are serious only in as much as they are meant to explain how to start using the interrupts for your own routines.

It is vital when developing interrupt-driven routines to make sure that the instruction SEI (set interrupts) is issued before changing the interrupt vector and that the instruction CLI (clear interrupts) is issued before leaving the routine.

Interrupt 1

In the first example the interrupt vector is changed to point at $100D hex, by replacing the the interrupt vector at $0314 and $0315 hex with $100D, in low byte high byte format. The routine places characters on the screen and changes them and their colours constantly.

The routine that the interrupts are directed to must always end with a jump back to the normal interrupt routine or else you are in trouble. The last instruction should be JMP $EA31, which jumps back to the normal interrupt routine. To call this routine enter SYS 4096 <return>.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
1000 78          SEI
1001 A9 0D       LDA #$0D
1003 8D 14 03    STA $0314
1006 A9 10       LDA #$10
1008 8D 15 03    STA $0315
100B 58          CLI
100C 60          RTS
100D A9 00       LDA #$00
100F 85 FB       STA $FB
1011 A9 04       LDA #$04
1013 85 FC       STA $FC
1015 A9 00       LDA #$00
1017 85 FD       STA $FD
1019 A9 D8       LDA #$D8
101B 85 FE       STA $FE
101D A0 00       LDY #$00
101F B1 FB       LDA ($FB),Y
1021 69 01       ADC #$01
1023 91 FB       STA ($FB),Y
1025 B1 FD       LDA ($FD),Y
1027 69 01       ADC #$01
1029 91 FD       STA ($FD),Y
102B C8          INY
102C D0 F1       BNE $101F
102E 4C 31 EA    JMP $EA31
.
.
```

```
.:1000 78 A9 0D 8D 14 03 A9 10
.:1008 8D 15 03 58 60 A9 00 85
.:1010 FB A9 04 85 FC A9 00 85
.:1018 FD A9 D8 85 FE A0 00 B1
.:1020 FB 69 01 91 FB B1 FD 69
.:1028 01 91 FD C8 D0 F1 4C 31
.:1030 EA 00 00 00 00 00 00 00
.
.
```

Interrupt 2

The second routine is slightly longer, but in fact the interrupt-driven part of the routine ($1040 hex onwards) does less. The routine puts the 64 into lower case, clears the screen, sets the screen and border colours. The next part of the routine puts two messages on the screen.

Finally the interrupt-driven routine is called at $1040 hex and the routine exits. What is happening? Well, the screen is being scrolled continuously except for the top row. Try clearing the screen and typing something on the top line. It's easy, but slow.

Now move the cursor down a few lines and try typing something sensible. Not so easy, is it? The message is set to start at $1071 hex, but I have chosen to leave you to put the hex equivalent of the message in. The number of characters as the routine stands is 23: have fun!

```
B*
   PC   SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
1000 A9 17        LDA #$17
1002 8D 18 D0     STA $D018
1005 A9 93        LDA #$93
1007 20 D2 FF     JSR $FFD2
100A A9 00        LDA #$00
100C 8D 20 D0     STA $D020
100F A9 01        LDA #$01
1011 8D 21 D0     STA $D021
1014 A9 90        LDA #$90
1016 20 D2 FF     JSR $FFD2
1019 A2 00        LDX #$00
101B BD 71 10     LDA $1071,X
101E 9D 00 04     STA $0400,X
1021 E8           INX
1022 E0 17        CPX #$17
1024 D0 F5        BNE $101B
```

```
1026 A2 00        LDX #$00
1028 BD 88 10     LDA $1088,X
102B 9D E0 05     STA $05E0,X
102E E8           INX
102F E0 1C        CPX #$1C
1031 D0 F5        BNE $1028
1033 78           SEI
1034 A9 40        LDA #$40
1036 8D 14 03     STA $0314
1039 A9 10        LDA #$10
103B 8D 15 03     STA $0315
103E 58           CLI
103F 60           RTS
1040 A9 28        LDA #$28
1042 A2 18        LDX #$18
1044 85 57        STA $57
1046 A9 04        LDA #$04
1048 85 58        STA $58
104A A0 00        LDY #$00
104C B1 57        LDA ($57),Y
104E 85 59        STA $59
1050 C8           INY
1051 B1 57        LDA ($57),Y
1053 88           DEY
1054 91 57        STA ($57),Y
1056 C8           INY
1057 98           TYA
1058 C9 27        CMP #$27
105A D0 F4        BNE $1050
105C A5 59        LDA $59
105E 91 57        STA ($57),Y
1060 A5 57        LDA $57
1062 18           CLC
1063 69 28        ADC #$28
1065 85 57        STA $57
1067 90 02        BCC $106B
1069 E6 58        INC $58
106B CA           DEX
106C D0 DC        BNE $104A
106E 4C 31 EA     JMP $EA31
.
.
```

```
.:1000 A9 17 8D 18 D0 A9 93 20
.:1008 D2 FF A9 00 8D 20 D0 A9
.:1010 01 8D 21 D0 A9 90 20 D2
.:1018 FF A2 00 BD 71 10 9D 00
```

```
.:1020 04 E8 E0 17 D0 F5 A2 00
.:1028 BD 88 10 9D E0 05 E8 E0
.:1030 1C D0 F5 78 A9 40 8D 14
.:1038 03 A9 10 8D 15 03 58 60
.:1040 A9 28 A2 18 85 57 A9 04
.:1048 85 58 A0 00 B1 57 85 59
.:1050 C8 B1 57 88 91 57 C8 98
.:1058 C9 27 D0 F4 A5 59 91 57
.:1060 A5 57 18 69 28 85 57 90
.:1068 02 E6 58 CA D0 DC 4C 31
.:1070 EA 00 00 00 00 00 00 00
.
.
```

Using charget to add commands

Charget or character get is a short program in zero page from $73 hex 115 decimal to $8A hex 138 decimal. Its function is to provide the link between Basic and the interpreter. When you type 'run', each line of the program is put into the Basic input buffer. The charget routine then scans through it until it finds a recognisable byte. This is then put into the accumulator where the interpreter deals with it.

This is the routine that we need to modify in order to add new commands to Basic. The program is set up to add a command called 'DI' which will display the directory of the disk, but could easily be changed to add other commands.

The routine starts at $C000 hex and the first thing that it does is to transfer the instruction JMP $C00F to $73 hex (start of charget). This means that the charget routine will scan a specified area for a new word. If it is found then control jumps to $C043 hex to execute the statement.

By replacing the directory command from $C043 hex onwards you can add other statements. You will also need to change the ASCII characters that the charget routine is searching for. Call this routine with SYS 49152 and use DI to display the directory.

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
.
C000 A2 02        LDX #$02
C002 BD 0B C0     LDA $C00B,X
```

```
C005 95 73       STA $73,X
C007 CA          DEX
C008 10 F8       BPL $C002
C00A 60          RTS
C00B 4C 0F C0    JMP $C00F
C00E 00          BRK
C00F E6 7A       INC $7A
C011 D0 02       BNE $C015
C013 E6 7B       INC $7B
C015 8E 0E C0    STX $C00E
C018 BA          TSX
C019 38          SEC
C01A BD 01 01    LDA $0101,X
C01D E9 8C       SBC #$8C
C01F 7D 02 01    ADC $0102,X
C022 E9 A4       SBC #$A4
C024 D0 07       BNE $C02D
C026 20 79 00    JSR $0079
C029 C9 44       CMP #$44
C02B F0 06       BEQ $C033
C02D AE 0E C0    LDX $C00E
C030 4C 79 00    JMP $0079
C033 20 73 00    JSR $0073
C036 C9 49       CMP #$49
C038 F0 09       BEQ $C043
C03A C9 52       CMP #$52
C03C F0 05       BEQ $C043
C03E A2 0B       LDX #$0B
C040 6C 00 03    JMP ($0300)
C043 A9 24       LDA #$24
C045 85 FB       STA $FB
C047 A9 FB       LDA #$FB
C049 85 BB       STA $BB
C04B A9 00       LDA #$00
C04D 85 BC       STA $BC
C04F A9 01       LDA #$01
C051 85 B7       STA $B7
C053 A9 08       LDA #$08
C055 85 BA       STA $BA
C057 A9 60       LDA #$60
C059 85 B9       STA $B9
C05B 20 D5 F3    JSR $F3D5
C05E A5 BA       LDA $BA
C060 20 B4 FF    JSR $FFB4
C063 A5 B9       LDA $B9
C065 20 96 FF    JSR $FF96
C068 A9 00       LDA #$00
C06A 85 90       STA $90
C06C A0 03       LDY #$03
C06E 84 FB       STY $FB
C070 20 A5 FF    JSR $FFA5
```

```
C073 85 FC        STA $FC
C075 A4 90        LDY $90
C077 D0 2F        BNE $C0A8
C079 20 A5 FF     JSR $FFA5
C07C A4 90        LDY $90
C07E D0 28        BNE $C0A8
C080 A4 FB        LDY $FB
C082 88           DEY
C083 D0 E9        BNE $C06E
C085 A6 FC        LDX $FC
C087 20 CD BD     JSR $BDCD
C08A A9 20        LDA #$20
C08C 20 D2 FF     JSR $FFD2
C08F 20 A5 FF     JSR $FFA5
C092 A6 90        LDX $90
C094 D0 12        BNE $C0A8
C096 AA           TAX
C097 F0 06        BEQ $C09F
C099 20 D2 FF     JSR $FFD2
C09C 4C 78 01     JMP $0178
C09F A9 0D        LDA #$0D
C0A1 20 D2 FF     JSR $FFD2
C0A4 A0 02        LDY #$02
C0A6 D0 C6        BNE $C06E
C0A8 20 42 F6     JSR $F642
C0AB 6C 00 03     JMP ($0300)
.
.
```

```
.:C000 A2 02 BD 0B C0 95 73 CA
.:C008 10 F8 60 4C 0F C0 00 E6
.:C010 7A D0 02 E6 7B 8E 0E C0
.:C018 BA 38 BD 01 01 E9 8C 7D
.:C020 02 01 E9 A4 D0 07 20 79
.:C028 00 C9 44 F0 06 AE 0E C0
.:C030 4C 79 00 20 73 00 C9 49
.:C038 F0 09 C9 52 F0 05 A2 0B
.:C040 6C 00 03 A9 24 85 FB A9
.:C048 FB 85 BB A9 00 85 BC A9
.:C050 01 85 B7 A9 08 85 BA A9
.:C058 60 85 B9 20 D5 F3 A5 BA
.:C060 20 B4 FF A5 B9 20 96 FF
.:C068 A9 00 85 90 A0 03 84 FB
.:C070 20 A5 FF 85 FC A4 90 D0
.:C078 2F 20 A5 FF A4 90 D0 28
.:C080 A4 FB 88 D0 E9 A6 FC 20
```

```
.:C088 CD BD A9 20 20 D2 FF 20
.:C090 A5 FF A6 90 D0 12 AA F0
.:C098 06 20 D2 FF 4C 78 01 A9
.:C0A0 0D 20 D2 FF A0 02 D0 C6
.:C0A8 20 42 F6 6C 00 03 00 00
.
.
```

5. Kernal Routines

There are many Kernal routines that can be very useful. A few examples are given here.

Kernal 1

This routine demonstrates the CHRIN and the CHROUT routines. The CHRIN routine lives at $FFCF hex and the CHROUT routine lives at $FFD2 hex.

The CHRIN routine can be used by any device, as long as it has been set up to receive the information with the OPEN and CHKIN routines. In this case we will use the keyboard and no preparatory routines are needed. When this routine is called it will accept up to 88 characters from the keyboard terminated by a carriage return.

This is exactly what the demo does: it waits for input from the keyboard and stores the data and then uses CHKIN to place it on the screen again. This is a fairly simple use of the routine, but quite a demonstrative one. The routine used to demonstrate CHRIN and CHKIN starts at $C000 hex and is called from Basic with SYS49152. Don't forget the return.

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
C000 A9 17        LDA #$17
.
.
C000 A9 17        LDA #$17
C002 8D 18 D0     STA $D018
C005 A9 00        LDA #$00
C007 8D 20 D0     STA $D020
C00A A9 01        LDA #$01
```

```
C00C 8D 21 D0     STA $D021
C00F A2 00        LDX #$00
C011 20 CF FF     JSR $FFCF
C014 9D 2F C0     STA $C02F,X
C017 E8           INX
C018 C9 0D        CMP #$0D
C01A D0 F5        BNE $C011
C01C A9 93        LDA #$93
C01E 20 D2 FF     JSR $FFD2
C021 A2 00        LDX #$00
C023 BD 2F C0     LDA $C02F,X
C026 20 D2 FF     JSR $FFD2
C029 E8           INX
C02A C9 0D        CMP #$0D
C02C D0 F5        BNE $C023
C02E 60           RTS
.
.
```

```
.:C000 A9 17 8D 18 D0 A9 00 8D
.:C008 20 D0 A9 01 8D 21 D0 A2
.:C010 00 20 CF FF 9D 2F C0 E8
.:C018 C9 0D D0 F5 A9 93 20 D2
.:C020 FF A2 00 BD 2F C0 20 D2
.:C028 FF E8 C9 0D D0 F5 60 00
.
.
```

Kernal 2

Our second routine demonstrates the use of the GETIN routine, which is at $FFE4 hex. It takes a character from the keyboard buffer and places it into the accumulator. If there are no characters then a zero is returned.

Our routine below waits for a key press and then outputs to the screen using the CHKIN routine. To call it enter SYS49152.

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
C000 20 E4 FF     JSR $FFE4
```

```
C003 C9 00        CMP #$00
C005 F0 F9        BEQ $C000
C007 20 D2 FF     JSR $FFD2
C00A 60           RTS
.
.
```

```
.:C000 20 E4 FF C9 00 F0 F9 20
.:C008 D2 FF 60 00 00 00 00 00
.
.
```

Kernal 3

The third routine uses the PLOT routine at $FFF0 hex. This routine can be used to read the current cursor position or to position the cursor.

The 'X' and 'Y' registers must contain the row and column destination of the cursor. The routine below clears the screen, uses PLOT to position the cursor and places a 'C' in the position stated. Use SYS49152 to call this routine.

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
C000 A9 93        LDA #$93
C002 20 D2 FF     JSR $FFD2
C005 A9 00        LDA #$00
C007 A2 10        LDX #$10
C009 A0 10        LDY #$10
C00B 18           CLC
C00C 20 F0 FF     JSR $FFF0
C00F A9 43        LDA #$43
C011 20 D2 FF     JSR $FFD2
C014 60           RTS
.
.
.:C000 A9 93 20 D2 FF A9 00 A2
.:C008 10 A0 10 18 20 F0 FF A9
.:C010 43 20 D2 FF 60 00 00 00
.
.
```

Kernal 4

The fourth routine is one of many ways of saving programs. The routine collects the parameters for the save and then branches to perform the save ($E159 hex). Use SYS49152 from Basic to save a Basic program to tape.

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
C000 20 D4 E1     JSR $E1D4
C003 20 59 E1     JSR $E159
C006 60           RTS
.
.
```

```
.:C000 20 D4 E1 20 59 E1 60 00
.
.
```

Kernal 5

This will LOAD a program from tape. It uses three routines. The first routine sets the length of the file SETLFS at $FFBA. The second routine, SETNAM at $FFBD, sets the name of the file. The accumulator should contain the length of the filename. The 'X' and 'Y' registers should contain the low and high address of the filename. If no filename is used then load the accumulator with zero. The third routine is LOAD at $FFD5. This routine can also be used to verify a program. To load a program the accumulator must contain a zero. To verify it must contain a 1. The routine is then called. Use SYS49152 to load a Basic program.

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
C000 A9 01        LDA #$01
C002 A2 01        LDX #$01
C004 A0 01        LDY #$01
C006 20 BA FF     JSR $FFBA
C009 A9 00        LDA #$00
C00B 20 BD FF     JSR $FFBD
C00E A9 00        LDA #$00
C010 20 D5 FF     JSR $FFD5
C013 60           RTS
.
.
```

```
.:C000 A9 01 A2 01 A0 01 20 BA
.:C008 FF A9 00 20 BD FF A9 00
.:C010 20 D5 FF 60 00 00 00 00
.
.
```

Kernal and ROM routines

Given below is as complete a list as possible of the Kernal (operating system) and Basic ROM routines and how to use them.

The Kernal routines use what is commonly termed the Jumbo jump table from $FF81 hex to FFFF hex. The last section, $FFF6 to $FFFF, are hardware vectors. The function of the jumbo jump table is to give control to the operating system routines. I have therefore decided to include the jump address where the table is used.

The following format is used in describing the routines:

Name: name of routine

Purpose: purpose of routine

Jump address: call address of routine in hex

Address: start address of routine in hex

Communication registers: the registers accessed in order to pass data to and from the subroutine

Preparatory routines:routines that need to be called to set up data before the Kernal routine can be used. This often depends upon the particular use of the Kernal routine.

Errors: any errors returned from the routines will have their code placed in the accumulator

Stack use: number of stack bytes used by the routine

Registers affected: a list of all registers affected by the subroutine

Function: a brief description of the routine

1.	Name:	ACPTR
	Purpose:	Get data from the serial bus
	Jump address:	FFA5
	Address:	EE13
	Communication registers:	A. Data is returned in accumulator
	Preparatory routines:	TALK and TKSA
	Errors:	see READST
	Stack use:	13
	Registers affected:	X and A
	Function:	This routine gets one byte of data at a time from the serial bus, and places it in the accumulator.
2.	Name:	CHKIN
	Purpose:	Open a channel for input
	Jump address:	FFC6

Address:	031E (vector)
Communication registers:	X
Preparatory routines:	OPEN
Errors:	3, 5 and 6
Stack use:	0
Registers affected:	A and X
Function:	The open routine must used before this routine the default is keyboard. The X register must be loaded with the logical file number.

3.

Name:	CHKOUT
Purpose:	Open a channel for output
Jump address:	FFC9
Address:	0320 (vector)
Communication registers:	X
Preparatory routines:	OPEN
Errors:	0, 3, 5 and 7
Stack use:	4
Registers affected:	A and X
Function:	Use this routine to output data to device. The OPEN routine must be used first unless screen output is desired. The X register should contain the logical file number.

4. Name: CHRIN

Purpose:	Get a character from input channel
Jump address:	FFCF
Address:	0324 (vector)
Communication registers:	A
Preparatory routines:	OPEN and CHKIN
Errors:	see READST
Stack use:	7
Registers affected:	A and X
Function:	Assumes keyboard unless the OPEN and CHKIN routines have been used. The routine gets one byte of data from the input channel and places it in the accumulator.

5. Name: CHROUT

Purpose:	Output a character
Jump address:	FFD2
Address:	0326 (vector)
Communication registers:	A
Preparatory routines:	OPEN and CHKOUT
Errors:	see READST
Stack use:	8
Registers affected:	A
Function:	Assumes keyboard unless the OPEN and CHKOUT routines have been

	used. The routine outputs data, which has been placed in the accumulator before the routine is called.

6. Name:	CIOUT
Purpose:	Transmit a byte over the serial bus
Jump address:	FFA8
Address:	EDDD (send serial deferred)
Communication registers:	A
Preparatory routines:	LISTEN and SECOND
Errors:	see READST
Stack use:	5
Registers affected:	A
Function:	Used to send information to devices using the serial bus. Will need the LISTEN routine and SECOND if a secondary address is needed. Load accumulator with byte to be sent.

7. Name:	CLALL
Purpose:	Close all files
Jump address:	FFE7
Address:	032C (vector)
Communication registers:	None
Preparatory routines:	None
Errors:	None

Stack use:	11
Registers affected:	A and X
Function:	Closes all files and resets the I/O.

8.	Name:	CLOSE
	Purpose:	Close a logical file
	Jump address:	FFC3
	Address:	031C (vector)
	Communication registers:	A
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	A and X
	Function:	Closes a logical file using the number set by the OPEN routine.
9.	Name:	CLRCHIN
	Purpose:	Clear I/O channels
	Jump address:	FFCC
	Address:	0322 (vector)
	Communication registers:	None
	Preparatory routines:	None
	Errors:	None
	Stack use:	9

Registers affected:	A and X
Function:	Clears all open channels and resets I/O to default values.

10.	Name:	GETIN
	Purpose:	Get character from keyboard buffer queue
	Jump address:	FFE4
	Address:	032A (vector)
	Communication registers:	A
	Preparatory routines:	None
	Errors:	None
	Stack use:	7
	Registers affected:	A and (X, Y)
	Function:	Takes one character at a time from the keyboard buffer and returns it in the accumulator.

11.	Name:	IOBASE
	Purpose:	Define I/O memory page
	Jump address:	FFF3
	Address:	E500 (get I/O address)
	Communication registers:	X and Y
	Preparatory routines:	None
	Errors:	None
	Stack use:	2

Registers affected:	X and Y
Function:	The X and Y registers return the low and high address respectively of memory mapped I/O devices. Exists to aid compatibility with past and future machines.

12.	Name:	IOINIT
	Purpose:	Initialise I/O devices
	Jump address:	FF84
	Address:	FDA3 (initialise I/O)
	Communication registers:	None
	Preparatory routines:	None
	Errors:	None
	Stack use:	None
	Registers affected:	A, X and Y
	Function:	Initialises all I/O devices. Used by cartridges.

13.	Name:	LISTEN
	Purpose:	Command a device on the serial bus to listen
	Jump address:	FFB1
	Address:	ED0C
	Communication registers:	A
	Preparatory routines:	None
	Errors:	see READST

Stack use:	None
Registers affected:	A
Function:	Will command device to listen. The accumulator must be loaded with the device number.

14.	Name:	LOAD
	Purpose:	Load RAM from device or verify
	Jump address:	FFD5
	Address:	F49E (load program)
	Communication registers:	A, X and Y
	Preparatory routines:	SETLFS and SETNAM
	Errors:	0, 4, 5, 8 and 9
	Stack use:	None
	Registers affected:	A, X and Y
	Function:	Use this routine to load RAM from device or verify. The accumulator must be loaded with 0 for load or 1 for verify. LOAD can be set to ignore the header by giving a secondary address of 0 (in the OPEN routine). In this case the start and end addresses must be given and the program may be located where desired.
15.	Name:	MEMBOT
	Purpose:	Set or read the bottom address of RAM
	Jump address:	FF9C

	Address:	FE34 (read or set bottom of memory)
	Communication registers:	X and Y
	Preparatory routines:	None
	Errors:	None
	Stack use:	None
	Registers affected:	X and Y
	Function:	This routine will read or set the bottom of RAM. If the carry flag equals 1 then read, if 0 then set. Normal value $0800
16.	Name:	MEMTOP
	Purpose:	Set or read the top address of RAM
	Jump address:	FF99
	Address:	FE34 (read or set top of memory)
	Communication registers:	X and Y
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	X and Y
	Function:	This routine will read or set the top of RAM. If the carry flag equals 1 then read if 0, then set.
17.	Name:	OPEN
	Purpose:	Open a logical file

Jump address:	FFC0
Address:	031A (vector)
Communication registers:	None
Preparatory routines:	SETLFS and STENAM
Errors:	1,2,4,5,6
Stack use:	None
Registers affected:	A, X and Y
Function:	This routine requires SETLFS (length of name) and SETNAM (filename). It opens a logical file for use in any I/O operation.

18. Name:	PLOT
Purpose:	Set or read current cursor location
Jump address:	FFF0
Address:	E50A (put/get row/column)
Communication registers:	A, X and Y
Preparatory routines:	None
Errors:	None
Stack use:	2
Registers affected:	A, X and Y
Function:	This routine reads or sets the cursor position. If the carry flag is set, the cursor is set. If it is clear a read cursor is performed.

19.	Name:	RAMTAS
	Purpose:	Perform RAM test
	Jump address:	FF87
	Address:	FD50 (initialise system constants)
	Communication registers:	A, X and Y
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	A, X and Y
	Function:	Tests and sets RAM. Also sets the screen and is used by cartridges.

20.	Name:	RDTIM
	Purpose:	Read system clock
	Jump address:	FFDE
	Address:	F6DD (get time)
	Communication registers:	A, X and Y
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	A, X and Y
	Function:	Reads system clock (3 bytes) and returns most significant byte in accumulator, next significant byte in X register and least significant byte in Y register.

21. Name: READST

Purpose: Read status word

Jump address: FFB7

Address: FE07 (get status)

Communication registers: A

Preparatory routines: None

Errors: None

Stack use: 2

Registers affected: A

Function: Returns current status of I/O devices in accumulator. Information returned includes device status and error codes. Bits returned in accumulator contain the following information:

BIT	VALUE	CASSETTE READ	SERIAL R/W	TAPE VERIFY + LOAD
0	1		time out write	
1	2		time out read	
2	4	short block		short block
3	8	long block		long block
4	16	unrecoverable		any mismatch
5	32	checksum error		checksum error
6	64	end of file	EO1	
7	128	end of tape	device not present	end of tape

22. Name: RESTOR

Purpose: Restore default system and interrupt vectors

Jump address: FF8A

Address: FD15 (Kernal reset)

Communication registers: None

Preparatory routines: None

Errors: None

Stack use: 2

Registers affected: A, X and Y

Function: Restores all interrupt, I/O Kernal and Basic to default values.

23. Name: SAVE

Purpose: Save memory to device

Jump address: FFD8

Address: F5DD (save program)

Communication registers: A, X and Y

Preparatory routines: SETLFS and SETNAM

Errors: 5, 8, 9

Stack use: 2

Registers affected: A, X and Y

Function: Saves memory to device; needs SETLFS and SETNAM. Accumulator must contain zero page offset to start of save and X and Y registers should be loaded with low and high bytes of end of save.

24. Name: SCNKEY

Purpose: Scans keyboard

Jump address: FF9F

Address: EA87 (read keyboard)

Communication registers: None

Preparatory routines: IOINIT

Errors: None

Stack use: 5

Registers affected: A, X and Y

Function: Any key pressed is placed by this routine into the keyboard buffer. This is usually done by the normal interrupts, but can be called independently if required, usually when interrupts are bypassed.

25. Name: SCREEN

Purpose: Return number of screen rows and columns

Jump address: FFED

Address: E505 (get screen size)

Communication registers: X and Y

Preparatory routines: None

Errors: None

Stack use:	2
Registers affected:	X and Y
Function:	Returns screen format. X register contains number of columns and Y register contains number of rows.

26.	Name:	SECOND
	Purpose:	Send secondary address for LISTEN
	Jump address:	FF93
	Address:	EDB9 (send listen secondary address)
	Communication registers:	A
	Preparatory routines:	LISTEN
	Errors:	see READST
	Stack use:	8
	Registers affected:	A
	Function:	Used to send to device after LISTEN has been called. The address is loaded into the accumulator before the routine is called.

27.	Name:	SETLFS
	Purpose:	Set up a logical file
	Jump address:	FFBA
	Address:	FE00 (save file details)
	Communication registers:	A, X and Y
	Preparatory routines:	None

Errors:	None
Stack use:	2
Registers affected:	A, X and Y
Function:	Sets logical file number, secondary address and device address. Load accumulator with logical file number, X register with device number and Y register with secondary address (command).

28.	Name:	SETMSG
	Purpose:	Control system message output
	Jump address:	FF90
	Address:	FE18 (flag staus)
	Communication registers:	A
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	A
	Function:	Sets control or error messages for the operating system. The user can set these messages. The accumulator must contain the value before calling this routine.

29.	Name:	SETNAM
	Purpose:	Set up file name
	Jump address:	FFBD

Address:	FDF9 (save filename data)
Communication registers:	A, X and Y
Preparatory routines:	None
Errors:	None
Stack use:	None
Registers affected:	A, X and Y
Function:	Used to set filename for OPEN, SAVE or LOAD routines. The accumulator should be loaded with the length of the filename and the X and Y registers with the low and high bytes of the address in memory where the filename is stored.

30. Name: SETTIM

Purpose:	Set the system clock
Jump address:	FFDB
Address:	F6E4 (set time)
Communication registers:	A, X and Y
Preparatory routines:	None
Errors:	None
Stack use:	2
Registers affected:	A, X and Y
Function:	Resets the system clock. The accumulator must be loaded with the most significant byte, the X register with the next most significant byte and the Y register with the least significant byte before calling this routine.

31.	Name:	SETTMO
	Purpose:	Set the IEEE bus card timeout flag
	Jump address:	FFA2
	Address:	FE21 (set timeout)
	Communication registers:	A
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	A
	Function:	Sets a timeout condition until data is received or an error condition is set up. The accumulator is loaded with 0 and timeout is set; a 1 in Bit 7 will disable timeout.
32.	Name:	STOP
	Purpose:	Check if stop key is pressed
	Jump address:	FFE1
	Address:	0328 (vector)
	Communication registers:	A
	Preparatory routines:	None
	Errors:	None
	Stack use:	2
	Registers affected:	A and X
	Function:	Any interruption with the stop key sets the Z flag and all channels are reset to default

33. Name: TALK

Purpose:	Command a device on the serial bus to talk
Jump address:	FFB4
Address:	ED09 (send talk)
Communication registers:	A
Preparatory routines:	None
Errors:	see READST
Stack use:	None
Registers affected:	A
Function:	Device number should be placed into the accumulator before this routine is called.

34. Name: TKSA

Purpose:	Send secondary address to device commanded to TALK
Jump address:	FF96
Address:	EDC7 (send talk SA)
Communication registers:	A
Preparatory routines:	None
Errors:	see READST
Stack use:	8
Registers affected:	A
Function:	Any secondary address should be placed into the accumulator before this routine is called.

35. Name: UDTIM

Purpose:	Updates system clock
Jump address:	FFEA
Address:	F69B (bump clock)
Communication registers:	None
Preparatory routines:	None
Errors:	None
Stack use:	2
Registers affected:	A and X
Function:	Updates clock normally called by interrupts. If user interrupts are installed then this routine must be called.

36. Name: UNLSN

Purpose:	Command all devices on serial bus to stop receiving data
Jump address:	FFAE
Address:	EDFE (send unlisten)
Communication registers:	None
Preparatory routines:	None
Errors:	see READST
Stack use:	8
Registers affected:	A
Function:	When called this routine will stop all devices on serial bus from listening to the 64.

37. Name: UNTLK

Purpose: Send an UNTALK command to all devices on serial bus

Jump address: FFAB

Address: EDFE (send untalk)

Communication registers: None

Preparatory routines: None

Errors: see READST

Stack use: 8

Registers affected: A

Function: When called this routine will stop all devices set with TALK from sending data.

38. Name: VECTOR

Purpose: Set or read system RAM vectors

Jump address: FF8D

Address: FD1A (Kernal move)

Communication registers: X and Y

Preparatory routines: None

Errors: None

Stack use: 2

Registers affected: X and Y

Function: If the carry flag is set when this routine is called, the RAM vectors are read into a list pointed by the X and

Y registers. If the carry flag is clear the content of the list pointed to by the X and Y registers is read into RAM vectors.

Error codes

Value	Meaning
0	Routine terminated by STOP key
1	Too many files open
2	File already open
3	File not open
4	File not found
5	Device not present
6	File is not an input file
7	File is not an output file
8	File name is missing
9	Illegal device number

This concludes the main list of Kernal routines.

Other ROM and Kernal routines

These routines have been listed separately because they are not so easy to locate and use. Their documentation is almost non-existent, with the exception of the *Complete Commodore 64 ROM Disassembly*, by Pete Gerrard and myself, available from Duckworth.

1. New line

   A49C – A530

Function: Each new line entered in a Basic program is handled by this routine.

2. Crunch tokens

A579 - A612

Function: Commands are reduced to tokens by this routine.

3. Perform RUN

A871 - A882

Function: Routine to perform RUN on Basic program.

4. Garbage collection

B526 - B5BC

Function: Checks string storage and clears memory of unwanted strings.

5. Perform Save

E156 - E164

Function: Performs a save. This is a short routine and can be accessed and used in many ways. It can be called in two or three places other than E156.

6. Perform Load

E165 - E1BD

Function: Performs a load. Like the save it can be used in many ways. Definitely worth close study. The routine is not very long.

7. Warm restart

E37B - E393

Function: Clears all channels and resets pointers to default value. Will not disturb any program in memory.

8. Power up message

E45F - E4FF

Function: The wonderful power up message comes from here.

9. Clear screen

E544 - E565

Function: This routine can be used to clear the screen, but there are other ways to achieve the same thing.

10. Set up screen print

E691 - E6B5

Function: Arranges the screen for printing.

11. Advance cursor

E6B6 - E6EC

Function: Advances cursor one position.

12. Retreat cursor

E6ED - E700

Function: Cursor back one position.

13. Back on to previous line

E701 - E715

Function: Cursor up one line.

14. Output to screen

E716 - E879

Function: This routine can be used to set up and place characters on the screen. Needs careful study.

15. Input

F157 - F198

Function: Use this routine directly to get data from devices.

16. Find any tape header

F72C - F769

Function: Will get tape header and can be very useful. Why not experiment a little!

17. Write tape header

F76A - F7CF

Function: Using this routine, a header can be written (rather than using the save routine to write the header).

18. Kill tape motor

FCCA - FCD0

Function: Stops tape motor.

19. Power reset entry point

FCE2 - FD01

Function: Resets all pointers and restores the 64 to default values. This is not the same as switching the 64 off and back on. Any program in RAM will have pointers removed by this routine, but the code will still be there!

Vectors

Below is a list of the main vectors with their labels and addresses. Also given are the default addresses they point to.

1. IERROR

   Print Basic error message link

   $0300

2. IMAIN

   Basic warm start

   $0302

3. ICRNCH

   Crunch Basic token link

   $0304

4. IQPLOP

   Print token link

   $0306

5. IGONE

   Start new Basic code link

   $0308

6. IEVAL

   Get arithmetic link

   $030A

7. USR function jump

   $0310 default value (B248)

8. CINV

   Hardware IRQ interrupt

   $0314 default value (EA31)

9. CBINV

   BRK instruction interrupt

   $0316 default value (FE66)

10. NMINV

   Non-maskable interrupt

   $0318 default value (FE47)

11. IOPEN

   Kernal OPEN routine

   $031A default value (F34A)

12. ICLOSE

   Kernal CLOSE routine

   $031C default value (F291)

13. ICHKIN

Kernal CHKIN routine

$031E default value (F20E)

14. ICKOUT

Kernal CHKOUT routine

$0320 default value (F2500)

15. ICLRCH

Kernal CLRCHN routine

$0322 default value (F333)

16. IBASIN

Kernal CHRIN routine

$0324 default value (F157)

17. IBASOUT

Kernal CHROUT routine

$0326 default value (F1CA)

18. ISTOP

Kernal STOP routine

$0328 default value (F6ED)

19. IGETIN

Kernal GETIN routine

$032A default value (F13E)

20. ICLALL

Kernal CLALL routine

$032C default value (F32F)

21. USRCMD

User defined vector

$032E default value (FE66)

22. ILOAD

Kernal LOAD routine

$0330 default value (F4A5)

23. ISAVE

Kernal SAVE routine

$0332 default value (F5ED)

6. 64 to FX-80

This chapter deals with the Epson FX-80 in some detail. The justification for this is the popularity of the Epson range and the program techniques needed to use them. The only advantage of Commodore's own printers is their ability to display control characters.

Perhaps this chapter will encourage Commodore to make a fast and more flexible printer that competes in quality and price with the FX-80. It is reproduced here by kind permission of *Commodore User*, where it originally appeared, and Chris Durham; my thanks for their co-operation. The article and the program have simply been reproduced as they were printed. The program demonstrates clearly the flexibility and quality of the FX-80.

Downloading the character set

The main advantage of a Commodore printer is its ability to reproduce the graphics and the control characters in listings and printouts. Most non-Commodore printers either print nothing or something that looks like Greek letters. Neither of these are really desirable or acceptable, and they are of course impossible to read.

By using the FX-80's ability to download a user-defined character set, we can make the FX-80 produce the complete Commodore character set.

What is needed is a fancy bit of programming to pass the 64's ROM-based characters to the FX-80. This is not quite as simple as it may sound, as the 64 builds up the characters row by row. The FX-80, on the other hand, builds them up column by column. If you were to try passing the data for the character set to the FX-80 as it is held normally, you would end up with all the characters lying flat on their backs!

The program given here illustrates how to convert the characters so that they appear the right way around and how to download them to the FX-80. The program contains some screen messages and prompts. The messages double as progress reports since the program takes a couple of minutes to complete. Of course you will only need to run the program once before using the FX-80, and the Commodore character set will remain in the FX-80 until it is switched off.

Instructions are included on how to select either the standard Epson character set or the Commodore character set. This can be done from program or direct mode.

There are a number of points to note before using this program. First, the maximum number of adjacent horizontal dots in a printer character is six. Some Commodore characters, like the heart or the spade, use seven horizontal dots on the TV screen. These will be truncated when printed, and the only way to avoid this is to design your own characters for these symbols.

Secondly, there seems to be no way of replacing the printer control codes in character positions 18-20 inclusive. This means that the character codes for HOME, REV ON, and INSERT cannot be printed, since they occupy these character positions in the Commodore ASCII set. Thirdly, because both upper and lower case characters are held in the printer, there is not enough room for reversed characters as well. Finally, the 'zero font' switch on the printer must be set to the off position.

Within these constraints this program should at least provide readable listings.

```
10 REM **********************************************
20 REM PROGRAM TO DOWNLOAD COMMODORE CHARACTER SET
30 REM TO AN EPSON FX-80 PRINTER - BY CHRIS DURHAM
40 REM **********************************************
45 POKE52,152:POKE56,152:CLR:REM RESERVE SPACE IN
MEMORY FOR CHARACTER SET
47 POKE53280,14:POKE53281,6
50 PRINT"  SWOP CHAR SET INTO MAIN MEMORY"
60 CS=53248:CL=CS+512:ML=38913
70 PRINTCHR$(142):REM SWITCH TO UPPER CASE
80 POKE 56333,127:REM TURN OFF KEYSCAN INTERRUPT T
IMER
90 POKE 1,51:REM SWITCH IN CHARACTER SET
95 FOR A = 0 TO 511:POKE ML+A,PEEK (CL+A):NEXT A:R
EM TRANSFER CHARACTERS
100 ML=ML+512:FOR CH=1 TO 27
105 READ X:FOR A=0 TO 7
110 IF CH<25 THEN POKE ML+A,255-PEEK (CS+(X*8)+A):
```

```
REM TURN INTO RESERVED CHARS
115 IF CH>=25 THEN POKE ML+A,PEEK (CS+(X*8)+A): RE
M CHARS NOT IN EPSON CHAR SET
120 NEXT A:ML=ML+8:NEXT CH
125 POKE 1,55:REM SWITCH IN I/O
130 POKE 56333,129:REM TURN ON KEYSCAN INTERRUPT T
IMER
135 PRINT"  CONVERT CHARS TO PRINTER FORMAT"
137 DIM B1(8):FOR A=0 TO 7:B1(A+1)=2^A:NEXT A
140 PL=39700:MP=38913
145 FOR Y=PL TO PL+546:POKEY,0:NEXT Y
150 FOR Y=PL TO PL+540 STEP 6
160 FOR A=7 TO 2 STEP -1
170 FOR B=0 TO 7
180 IF (PEEK (MP+B) AND B1(A)) THEN POKE (Y+7-A),P
EEK (Y+7-A) OR B1(8-B)
190 NEXT B,A:MP=MP+8:NEXT Y
200 OPEN4,4
210 REM TRANSFER EXISTING EPSON CHARACTER SET TO U
SER AREA
215 PRINT#4,CHR$(27);"R";CHR$(0);:REM SELECT USA S
ET
220 PRINT#4,CHR$(27);":";CHR$(0);CHR$(0);CHR$(0);
225 PRINT" NOW TRANSFER COMMODORE CHARACTERS"
227 FOR L=1 TO 2:READFC,LC
230 PRINT#4,CHR$(27);"&";CHR$(0);CHR$(FC);CHR$(LC)
;
235 FOR CH=0 TO 31:PRINT#4,CHR$(139);
240 FOR A=0 TO 4
250 PRINT#4,CHR$(PEEK(PL+(CH*6)+A));:PRINT#4,CHR$(
0);
255 NEXT A:PRINT#4,CHR$(PEEK(PL+(CH*6)+5));
260 NEXT CH:PL=PL+(32*6):NEXT L
262 REM ALLOW ALL ASCII CODES (0 - 255) TO BE PRIN
TABLE
264 PRINT#4,CHR$(27);"I";CHR$(1);CHR$(27);"6";
266 PRINT"  NOW TRANSFER CONTROL / COLOUR CODES"
268 REM ALSO INCLUDES CHARACTERS NOT IN STANDARD E
PSON SET
270 PL=40084:FOR CH=0 TO 26
280 READ CP
290 PRINT#4,CHR$(27);"&";CHR$(0);CHR$(CP);CHR$(CP)
;
300 PRINT#4,CHR$(139);
310 FOR A=0 TO 4
320 PRINT#4,CHR$(PEEK(PL+(CH*6)+A));:PRINT#4,CHR$(
0);
325 NEXT A:PRINT#4,CHR$(PEEK(PL+(CH*6)+5));
330 NEXT CH
335 REM SWITCH TO USER DEFINED CHARACTER SET
340 PRINT#4,CHR$(27);"%";CHR$(1);CHR$(0);
```

```
350 PRINT#4,CHR$(27);"E";:REM SET EMPHASISED MODE
360 PRINT#4:CLOSE4
370 PRINT" CHARACTER SET COMPLETE":PRINT
375 PRINT"  COMMODORE CHARACTER SET SELECTION"
377 PRINT"  *********************************"
380 PRINT"  TO SELECT EPSON CHAR SET, TYPE:"
390 PRINT"  PRINT#4,CHR$(27);"CHR$(34)"%"CHR$(34
)";CHR$(0);CHR$(0);"
400 PRINT"  TO RE-SELECT COMMODORE CHAR SET TYPE:"
410 PRINT"  PRINT#4,CHR$(27);"CHR$(34)"%"CHR$(34
)";CHR$(1);CHR$(0);"
420 PRINT"  ENSURING STREAM 4 IS OPEN FOR PRINT
  OUTPUT."
430 POKE56,160:POKE52,160:CLR:END
1000 DATA 80,5,28,95,92,30,31,94,65,85,86,87
1010 DATA 88,89,90,91,18,70,83,19,81,17,66,29,28,3
1,94
1015 DATA 192,223,160,191
1020 DATA 144,5,28,159,156,30,31,158,129,149,150,1
51
1030 DATA 152,153,154,155,18,146,147,19,145,17,157
,29,92,95,255

READY.
```

7. General Utilities, Hints and Tips

Reserved words

For those of you unfamiliar with the term 'reserved words', it simply refers to the 64's Basic commands and instructions. This includes all I/O commands, such as 'OPEN' or 'LOAD'.

The point about reserved words is that they cannot be used in programs except in their legal sense. This means that they may not be used as variables, etc. For example, the statement 'FOR ST = 1 TO 10' is illegal since ST is a reserved word for the current I/O status of the 64.

However, it is possible to reconfigure the 64 completely in terms of its reserved words. This is done by copying the Basic ROM (from $A000 to BFFF hex) into the underlying RAM. When you have done this the reserved words can be replaced with user-defined words.

Once you have set up your own reserved words the normal ones will no longer be recognised and can be used as variables, etc. The replacement words become reserved words and must be used for the purpose for which they were defined.

Why change them?

Well, it is fun to a have a customised version of Basic, but it may have more serious implications in aiding program protection, for example. Once a customised version of Basic has been set the commands and their tokens will be accepted.

The problem is remembering what your replacement words are. I suggest that you write them down on paper or create a file to disk or tape containing the replacement words you choose.

Customising Basic

This is quite simple. As described above, the Basic ROM must first be copied and switched out. To do this I have written a small machine code routine and mixed it with the Basic program. This means that the program must be entered exactly as shown or the machine code will not work.

The easiest way to enter and save this program is to load and RUN Supermon, and then intialise and new it. The Basic program can then be entered but not tested. The next step is to re-enter Supermon with SYS8<return> and enter the code from $0E00 to $0E1F hex. Staying in Supermon, the program should then be saved with S"<return>",08,0801,0E20

This will give you a copy of the program which can be tested and used. Remember to save any version that has been corrected or altered.

Although I said earlier that this program was fairly simple, it does deserve a bit of explanation. The first line of the Basic program sets the screen and border colours before printing a message, which you should not get much time to read if your version is working. If it stays on the screen for more than a few seconds, then there is an error and your 64 may well have gone to sleep. To awaken it you may need to switch off and on again. You did remember to save it, didn't you?

The machine code is accessed at line 120, and the time taken to evaluate the SYS takes longer than the routine does to complete. The machine code routine stores the start of the Basic ROM in FB and FC hex and then, using the Y register as an offset, stores the ROM in the underlying RAM.

The high byte of the ROM address is incremented after completion of each page until it reaches $C000 hex, the end of the Basic ROM and the start of the alternate RAM. The Basic ROM is then switched out by placing $36 hex 54 decimal into location 1. Finally control is handed back to the Basic program.

The Basic program continues execution and asks for the reserved word you wish to change. Once the word has been selected an end of word marker is added and the program scans through the reserved words for the one chosen. If it is not found an error message is displayed.

Once the word to be changed has been found a replacement is requested. It must be the same length as the word it is replacing and must not duplicate any other reserved word currently in use. The program lastly requests another word or finishes the program.

Once you have left the program you may use your customised version of Basic as you would the normal Basic. This includes saving and loading programs. You will need to re-initialise or change your version of Basic after resetting the 64. You may switch between the normal and customised versions with 'POKE 1,n' where n equals 55 for normal and 54 for the customised version.

Try thinking of some uses for the program. A rude version of Basic would save you swearing at the 64, as it could do it for you.

```
100 POKE53280,6:POKE53281,7:PRINT"  READI
NG ROM INTO RAM PLEASE WAIT...."
110 REM *** LOOP TO COPY ROM INTO RAM
120 SYS3584
130 REM *** TAKE OUT BASIC ROM
160 POKE1,54
170 REM *** PUT RESERVED WORDS INTO R$
180 INPUT"     RESERVED WORD    ";RO$
185 PRINT"      SEARCHING FOR WORD......
"
190 REM *** SET TERMINATOR MARKER ON LAST BYTE OF
STRING
200 RO$=LEFT$(RO$,LEN(RO$)-1)+CHR$(ASC(RIGHT$(RO$,
1))+128)
210 REM *** ROUTINE TO SEARCH FOR RESERVED WORD
220 GOSUB390
230 IFFL=0THENPRINT"              NOT FOUND  ":F
ORDE=1TO4000:NEXT:GOTO330
240 INPUT"      YOUR WORD (SAME LENGTH)
  ";US$
250 REM *** CHECK LENGTH OF WORDS ARE THE SAME
260 IFLEN(US$)<>LEN(RO$)THEN240
270 REM *** ADD TERMINATOR
280 US$=LEFT$(US$,LEN(US$)-1)+CHR$(ASC(RIGHT$(US$,
1))+128)
290 REM *** LOOP TO POKE IN NEW WORD
300 FORJ=1TOLEN(US$)
310 POKEHO+J-1,ASC(MID$(US$,J,1))
320 NEXT
330 PRINT"      ANOTHER WORD (Y/N)"
340 GETA$:IFA$<>"Y"ANDA$<>"N"THEN340
350 IFA$="N"THENEND
360 REM *** ANOTHER WORD
370 GOTO180
```

```
380 REM *** START ADDRESS OF ROM
390 HO=40960
400 REM *** GET FIRST CHARACTER
410 C=ASC(MID$(RO$,1,1))
420 REM *** IF FIRST CHARACTER MATCHES CHECK OTHER
S
430 IFPEEK(HO)=CTHEN510
440 REM *** LOOK AT NEXT ROM POSITION
450 HO=HO+1
460 REM *** CHECK FOR END OF WORD TABLE IN ROM
470 IFHO=>42000THENFL=0:RETURN
480 REM *** STARTS NEXT CHECK
490 GOTO410
500 REM *** SET POINTER TO POSITION OF SECOND CHAR
ACTER
510 HO=HO+1
520 REM *** THIS LOOP CHECKS THAT THE REST OF THE
CHAR.'S MATCH
530 FORJ=2TOLEN(RO$)
540 REM *** CHECK EACH CHARACTER
550 IFPEEK(HO+J-2)<>ASC(MID$(RO$,J,1))THEN410
560 NEXT
570 REM *** SET POINTER TO START OF WORD AND SET F
OUND FLAG
580 HO=HO-1:FL=-1
590 RETURN
```

```
B*
   PC  SR AC XR YR SP
.;0008 F0 C7 00 40 F6
.
0E00 A9 00        LDA #$00
0E02 85 FB        STA $FB
0E04 A9 A0        LDA #$A0
0E06 85 FC        STA $FC
0E08 A0 00        LDY #$00
0E0A B1 FB        LDA ($FB),Y
0E0C 91 FB        STA ($FB),Y
0E0E C8           INY
0E0F D0 F9        BNE $0E0A
0E11 E6 FC        INC $FC
0E13 A5 FC        LDA $FC
0E15 C9 C0        CMP #$C0
0E17 D0 EF        BNE $0E08
0E19 D0 ED        BNE $0E08
0E1B A9 36        LDA #$36
0E1D 85 01        STA $01
0E1F 60           RTS
.
.
```

```
.:0E00 A9 00 85 FB A9 A0 85 FC
.:0E08 A0 00 B1 FB 91 FB C8 D0
.:0E10 F9 E6 FC A5 FC C9 C0 D0
.:0E18 EF D0 ED A9 36 85 01 60
.
.
```

Both sides!

Users of the 1541 will be familiar with the trip to get more disks. Wouldn't it be pleasant to use both sides of your disk?

You will be pleased to know that not only is it possible to use both sides of a single disk, but each side can be formatted on a different drive. To do this you will need to cut a duplicate notch in your diskette carefully or the drive will not write to it.

Having done this you can use both sides of the disk: just format the reverse side in the usual way. This procedure is definitely not recommended, but it seems to work (most of the time). I am not quite sure why it works, so if any of you know or have an idea on the subject I would be pleased to hear from you.

Joysticks

Two short routines to aid joystick control are given here. Nothing fancy, but they should give you the general idea.

First, a routine that will detect and print the direction in which the stick was moved.

```
10 POKE 56322,224

20 J=PEEK (56320)

30 IF (JAND1)=0 THEN PRINT"GOING UP"

40 IF (JAND2)=0 THEN PRINT"DOWN WE GO"

50 IF (JAND4)=0 THEN PRINT"TO THE LEFT"

60 IF (JAND8)=0 THEN PRINT"NOW THE RIGHT"

70 IF (JAND16)=0 THEN PRINT"UNDER FIRE"
```

```
80 GOTO20
```

This simply returns the current direction of the joystick.

The second routine might be more suitable for inclusion in user programs, but also serves as a good demo.

```
10 PRINTCHR$(147)
20 J=PEEK(56320)
30 PRINT"[SH V]";CHR$(157);
40 IF(JAND1)=0THENPRINTCHR$(20);" ";CHR$(145);
50 IF(JAND2)=0THENPRINTCHR$(20);" ";CHR$(17);
60 IF(JAND4)=0THENPRINTCHR$(20);" ";CHR$(157);
70 IF(JAND8)=0THENPRINTCHR$(20);" ";CHR$(29);
80 IF(JAND16)=0THENPRINT"Q";CHR$(157);
90 GOTO20
```

The program places the character shift V on the screen and moves it in the direction of the stick. If the fire button is pressed the character shift Q is superimposed over shift V. As it stands it is only good for a demo, but it could easily be converted for use in your programs.

Input routine

This short routine will clear the screen, place an asterisk on it and wait for an input. Pressing a key will place the appropriate character on the screen with the asterisk on the rightmost of the last character input. When a carriage return is found the program exits.

The program works fine as it stands, but needs to be adapted for your particular needs if you intend to use it from your programs. It can easily be adapted for use as a protected input for adventure games. The routine uses the following Kernal routines:

CHROUT ($FFD2) output character to channel

GETIN ($FFE4) wait for keypress, using the SCNKY routine.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
C000 A9 93       LDA #$93
C002 20 D2 FF    JSR $FFD2
```

```
C005 A9 2A       LDA #$2A
C007 20 D2 FF    JSR $FFD2
C00A 20 E4 FF    JSR $FFE4
C00D C9 00       CMP #$00
C00F F0 F9       BEQ $C00A
C011 C9 0D       CMP #$0D
C013 F0 11       BEQ $C026
C015 8D 27 C0    STA $C027
C018 A9 14       LDA #$14
C01A 20 D2 FF    JSR $FFD2
C01D AD 27 C0    LDA $C027
C020 20 D2 FF    JSR $FFD2
C023 4C 05 C0    JMP $C005
C026 60          RTS
.
.
```

```
.:C000 A9 93 20 D2 FF A9 2A 20
.:C008 D2 FF 20 E4 FF C9 00 F0
.:C010 F9 C9 0D F0 11 8D 27 C0
.:C018 A9 14 20 D2 FF AD 27 C0
.:C020 20 D2 FF 4C 05 C0 60 00
.
.
```

Cursor control

By now most people will be familiar with the techniques of positioning the cursor from Basic, but it is also possible to write a very short routine in machine code to give you excellent control of the cursor from your Basic programs.

This routine sits at $C000 hex, but could easily be relocated to any free part of memory. So, instead of the following:

```
10 A$="[25 CD][40 CR]"

20 B$=LEFT$ (A$,N)
```

etc.

we can simply enter the screen co-ordinates for the next cursor position followed by the characters to be placed there.

The machine code part of the routine first checks for a comma ($AEFD) and then gets a byte ($B79E). The byte is placed on the stack and the next byte is input. This gives the row and column co-ordinates for the position of the cursor. The cursor is then positioned using the Kernal routine PLOT ($FFF0), and the routine checks for a second comma. A jump to the PRINT routine displays your message at the selected co-ordinates.

```
C000 JSR $AEFD

C003 JSR $B79E

C006 TXA

C007 PHA

C008 JSR $AEFD

C00B JSR $B79E

C00E PLA

C00F TAY

C010 CLC

C011 JSR $FFF0

C014 JSR $AEFD

C017 JMP $AAA4
```

The Basic subroutine to call the above is fairly simple and can easily be placed within your programs. The first line sets the SYS address, the next three lines select co-ordinates for the cursor and messages to be displayed. The fifth line is a delay loop and the last two lines display a message and wait for a key press to exit the routine.

```
63000 CP=49152

63010 SYSCP,10,10,"DEMONSTRATES"

63020 SYSCP,10,20,"CURSOR CONTROL"

63030 SYSCP,10,0,"THIS PROGRAM"
```

```
63040 FOR PAUSE = 1 TO 2000:NEXT PAUSE

63050 SYSCP,10,5,"[ON HIT]ANY KEY[OFF]"

63060 GETA$:IF A$ = "" THEN 63040
```

String memory

Setting up strings from a program in the following way:

```
DIM EX$(900):FOR S = 1 TO 900:EX$(S) = CHR$(82):NEXT
```

will store the strings in the string storage area were they remain until memory runs out because of 'dead' strings or the 64 does a forced garbage collection (PRINT FRE(0)). It is of course almost impossible to calculate when your 64 will need to perform a garbage collection and it can take a considerable time to do.

It is therefore advisable to avoid strings that use CHR$ or STR$, and avoid as much as possible the storage of strings that will be discarded but not recovered. The two Basic commands INPUT and GET will also use the string storage area and should have their string variables cleared after use in your programs. So the following:

```
1 GET A$:IF A$="YES" THEN etc
```

could be cleared after use by setting A$ to a null string (A$ = "").

Using strings that are read in from data statements or assigning string variables such as EX$(S) = "n" (where n is the character required), uses the strings directly from the program and does not use the string storage area. This is a much more economical use of memory.

In general, strings should not be used repeatedly without being set to a null string in between uses. If there is a routine which builds up a block of strings and then discards them, take the first possible chance to perform a FRE(0). This should save memory and time.

Hex to Dec

Instead of reproducing the usual hexadecimal to decimal conversion table I have decided to include a Basic program that will convert hex to decimal or decimal to hex and display it on the screen in an easily readable form. The program was originally written for the Pet by Pete Gabor; I have updated and converted it for the 64. My thanks to Pete.

The program uses data statements to set up two boxes in the middle of the screen and one at the bottom of the screen. The two smaller boxes are used to display the number to be converted and its equal in hex or decimal. The box at the bottom of the screen is the command area. It is used to display the mode you are in (hex to decimal or decimal to hex).

To quit the program use the HOME key (unshifted). I am afraid that you will have to bear with me when you get to entering the data statements, as they are all with the shift or logo key. That is about it for this program. The result is a very readable and easy to use converter.

```
10 MD=0:M$(0)="DEC -> HEX":M$(1)="HEX -> DEC"
20 P0$="[HME][7 CD]"
30 P1$=P0$+"[7 CD][15 CR]"
40 P2$=P1$+"[5 CD]":P0$=P0$+"[6 CR]"
50 PRINT"[CLR]";
60 FORK=1TO23
70 READA$:PRINTA$
80 NEXT
90 P$="":N=0
100 PRINTP1$;
110 PRINT"[CL][ON]        [OFF][7 CL]";
120 GETC$:IFC$=""THEN120
130 C=ASC(C$)
140 IFC=13THEN240
150 IFC=20THEN90
160 IFC=19THENPRINT"[CLR]":END
170 IFC=64THENMD=1-MD:PRINTP2$M$(MD):GOTO90
180 IFC<48ORC>70OR(MD=0ANDC>57)THEN120
190 IFLEN(P$)>6THEN120
200 N=16*N+C-48+(C>57)*7
210 PRINTC$;
220 P$=P$+C$
230 GOTO120
240 P$=RIGHT$("       "+P$,7)
250 IFMDTHEN330
260 D$=P$:D=VAL(P$):H$="":A=D
270 FORK=1TO6:A=A/16:IFA<1THEN290
280 NEXT
290 FORJ=KTO1STEP-1
300 H%=D/16^(J-1):D=D-16^(J-1)*H%
310 H$=H$+CHR$(H%+48-(H%>9)*7):NEXT
320 H$=RIGHT$("       "+H$,7):GOTO350
330 D$=RIGHT$("       "+STR$(N),7)
340 H$=P$:IFN>999999999 THEND$="*******"
```

```
350 PRINTP0$H$;"[13 CR]";D$
360 GOTO90
370 DATA"              [21 LO @]"
380 DATA"              [ON]* HEX-DEX CONVERTER *[OFF]
"
390 DATA" "
400 DATA" "
410 DATA" "
420 DATA"         H  E  X                DECIMAL"
430 DATA"         [SH O][5 LO Y][SH P]
[SH O][5 LO Y][SH P]"
440 DATA"         [LO H]      [LO N]                [LO
 H]      [LO N]"
450 DATA"         [SH L][5 LO P][SH @]
[SH L][5 LO P][SH@]"
460 DATA" "
470 DATA" "
480 DATA" "
490 DATA"                     INPUT"
500 DATA"                   [SH O][7 LO Y][SH P]"
510 DATA"                   [LO H]        [LO N]"
520 DATA"                   [SH L][5 LO P][SH @]"
530 DATA" "
540 DATA" "
550 DATA"                 [SH O][11 LO Y][SH P]"
560 DATA" INPUT MODE: [LO H]           [LO N]"
570 DATA"                 [SH L][11 LO P][SH @]"
580 DATA"        [LO P]                  [3 LO P]"
590 DATA"PRESS [ON]@[OFF] TO CHANGE MODE; [ON]HME
[OFF] TO QUIT PGM"
600 DATA" "
610 DATA" "
```

Code to Basic

A short routine is included here to convert machine code routines to Basic data statements. In essence this is quite simple: each address of the code is looked at and the value is placed in a data statement.

The difficult part of the process is to create the new lines for the data statements and the data statements themselves from within a Basic program. The program makes extensive use of the keyboard buffer to create the next Basic line number, the data statement and the data.

The information is set up on each pass through the program and displayed on the screen. The program then places two carriage returns into the keyboard buffer and exits. The new line is then created and the program re-entered at line 200.

A closer look at the program may enlighten you somewhat. The program first requires the first new line number. I advise a number above 380 or you will write over the program. The increment between lines is requested and then the start and end addresses of the machine code program. Don't forget to have the machine code in memory at this point!

When you have given the program this information, it is converted into variables, and the low and high byte of the start and end addresses are stored. The line number and the data statement are then printed. The section of code is peeked, converted into data and printed. The statement GOTO 200 is then displayed on the screen. The current line number is incremented using the step you gave and the variables are set to point at the next section of code.

At this point you have on the screen a line number, the data statement and a line of data, but to transform this into a Basic line number a carriage return must be performed on it. In order to achieve this the program places 2 into location 198 (the counter for the number of characters in the buffer) and two carriage returns into the keyboard buffer, and stops the program.

The first carriage return is executed over the new line and the second one over the GOTO 200 statement which re-enters the program at line 200! The program then checks for the end of code addresses and stops if they have been reached, but continues to create new lines, data statements and data if they have not been reached.

This program is good enough. It is fairly quick and will produce data from huge machine code programs!

```
100 PRINT"[CLR][ON]CREATE DEC. DATA STATEMENTS FR
OM M/C."
110 PRINT"[CD]                FOR THE [ON]CBM64[OFF]"
120 INPUT"[HME][5 CD]            START LINE NO.#    [3 CL
]";S$:IFS$=" "THEN120
130 INPUT"[HME][7 CD]            STEP    [3 CL]";T$:IFT$=
" "THEN130
140 INPUT"[HME][9 CD]            START ADD. DEC.    [3 CL
]";B$:IFB$=" "THEN140
150 INPUT"[HME][11 CD]            END ADD. DEC.    [3 CL]
";E$:IFE$=" "THEN150
160 S=VAL(S$):T=VAL(T$):B=VAL(B$):E=VAL(E$):F=B:L
=F+6:PRINT"[4 CD]"
170 POKE831,INT(E/256)
180 POKE832,E-INT(E/256)*256
```

```
190 POKE828,T:GOTO270
200 T=PEEK(828)
210 S=PEEK(826)*256+PEEK(827)
220 L=PEEK(829)*256+PEEK(830)
230 E=PEEK(831)*256+PEEK(832)
240 IFL>=ETHENEND
250 F=L+1:L=L+7
260 PRINT"[CU]                          "
270 PRINTS;
280 PRINT"DATA";
290 FORP=FTOL:PRINTMID$(STR$(PEEK(P)),2);",";:NEX
TP
300 PRINT"[CL] "
310 PRINT"GOTO200[3 CU]";
320 POKE198,2:POKE631,13:POKE632,13
330 S=S+T
340 POKE826,INT(S/256)
350 POKE827,S-INT(S/256)*256
360 POKE829,INT(L/256)
370 POKE830,L-INT(L/256)*256:END
380 END
```

Hi-res

Again my thanks go to our German friends for parts of the following routine. Essentially it sets up a hi-res screen and allows the user to PLOT, UNPLOT, COLOUR, DUMP and GOFF on the hi-res screen. I make no claim that this is a complete hi-res package, but it is well on the way. Its features include setting up a hi-res screen, clearing the screen, changing screen colours, inverting the screen, plotting and unplotting. The routine will also save users' screens on to tape or disk.

The disassembly of the program looks a bit odd in places as it includes tables. You will need a monitor to enter this routine: Supermon will do nicely, or any other monitor that does not occupy the top part of the alternate RAM ($C000 to $C255), as this is where the program sits in memory. It could well be relocated, but this would take some time as there a quite a few jumps that would have to be changed by hand.

Below is a list of the entry points into the routine and the entry points for the various subroutines:

SYS 49152 ($C000 hex enters package, sets up and clears graphics screen)

(SYS 49155 $C003 hex clears graphics screen)

SYS 49158 ($C006 hex sets the colour for the screen, e.g. SYS 49158,7 sets the screen to yellow)

SYS 49161 ($C009 hex inverts the graphics screen)

SYS 49164 ($C00C hex plots a point, e.g. SYS 49164,n1,n2 where n1 is in the range 0 to 199 and n2 is in the range 0 to 255)

SYS 49167 ($C00F hex unplots a point: same format as SYS 49164)

SYS 49170 ($C012 hex loads a previously saved screen from tape or disk, e.g. SYS 49170,"filename",dn where dn is 1 for tape and 8 for disk)

SYS 49173 ($C015 hex saves the graphics screen to tape or disk, e.g. SYS 49173,"filename",dn)

SYS 49176 ($C018 hex turns the graphics screen off and returns to normal screen)

SYS 49179 ($C01B hex will dump the contents of the hi-res screen to printer. Make sure you have one hooked up before calling this routine)

Some subroutines could be added to this routine. First, a routine to set up hi-res sprites; secondly a fill routine; and lastly a DRAW routine would of course be very useful.

The code sits in RAM from $C000 to $C055. The first part of the listing, $C000 to $C01B, is a jump table for the entry points of the routines described above.

Before using any of the wonderful features of this routine, the hi-res screen must first be initialised. This is carried out by the code from $C01E to $C03C hex.

Before initialising the hi-res screen, the contents of $D011 hex 53265 decimal and $D018 hex 53272 decimal are stored in order to reset the 64 to its normal screen after use. The contents are then changed to set up a hi-res screen. In Basic their equivalent would be POKE 53265,59:POKE 53272,24. This sets up the hi-res screen, but it still needs to be cleared.

The usual shift HOME will not do for the hi-res screen. So a routine to clear the hi-res screen is included from $C03D to $C053 hex. It sets the start of the hi-res screen from $2000 hex 8192 decimal and places zeros in every location from $2000 hex to $3FFF, the end of the hi-res screen. In Basic this routine would look something like this:

```
FOR SCREEN = 8192 TO 16383:POKE SCREEN,0:NEXT
```

Next the colour has to be placed on the screen. The screen is cleared initially by the code from $C05A to $C070 hex, but after that is set by the user with the entry to the routine at $C054 hex. The equivalent in Basic would be:

```
FOR COLOUR = 1024 TO 2023:POKE COLOUR,16:NEXT
```

We now have the same set up in Basic as the first three routines in code. They are position the screen, clear the screen and colour the screen. Indeed we now have an elementary Basic version which looks like this:

```
10 POKE 53265,59:POKE 53272,24

20 FOR SCREEN = 8192 TO 16383

30 POKE SCREEN,0

40 NEXT SCREEN

50 FOR COLOUR = 1024 TO 2023

60 POKE COLOUR,16

70 NEXT COLOUR
```

The next routine from $C071 to $C08A hex inverts the screen. This is done with the EOR instruction. Every location on the hi-res screen is EOR'd with $FF hex.

The next routine at $C08B to $C107 hex will plot a point on the hi-res screen. To unplot a point we use the routine at $C08E to $C107.

The routine from $C152 to $C161 hex loads a previously saved screen. This routine uses the LOAD Kernal routine at $FFD5 hex, and the routine from $C162 to $C171 saves a screen using the perform save routine $E544. The routine that sets the parameters for the load and save is at $C13A to $C151 hex.

The hi-res dump (for the FX-80 and with small adjustments other Epson printers, although there are plenty of hi-res dumps available for Commodore printers) is at $C180 to $C055. It scans the hi-res screen and dumps the contents to the FX-80. Although it is set up to dump the area from $2000 to $3FFF hex, it can easily be changed to look at another area. In fact the whole routine could be used to place a hi-res screen in any available memory.

```
B*
   PC  SR AC XR YR SP
.;0008 30 00 00 00 F6
.
C000 4C 1E C0     JMP $C01E
C003 4C 3D C0     JMP $C03D
C006 4C 54 C0     JMP $C054
C009 4C 71 C0     JMP $C071
C00C 4C 8B C0     JMP $C08B
C00F 4C 8E C0     JMP $C08E
C012 4C 52 C1     JMP $C152
C015 4C 3A C1     JMP $C13A
C018 4C 62 C1     JMP $C162
C01B 4C 80 C1     JMP $C180
C01E AD 11 D0     LDA $D011
C021 8D 72 C1     STA $C172
C024 AD 18 D0     LDA $D018
C027 8D 73 C1     STA $C173
C02A A9 3B        LDA #$3B
C02C 8D 11 D0     STA $D011
C02F A9 18        LDA #$18
C031 8D 18 D0     STA $D018
C034 20 3D C0     JSR $C03D
C037 A2 10        LDX #$10
C039 20 5A C0     JSR $C05A
C03C 60           RTS
C03D A0 00        LDY #$00
C03F A9 20        LDA #$20
C041 84 FD        STY $FD
C043 85 FE        STA $FE
C045 98           TYA
C046 91 FD        STA ($FD),Y
C048 C8           INY
C049 D0 FB        BNE $C046
C04B E6 FE        INC $FE
C04D A5 FE        LDA $FE
C04F C9 40        CMP #$40
C051 D0 F2        BNE $C045
C053 60           RTS
```

```
C054 20 FD AE     JSR $AEFD
C057 20 9E B7     JSR $B79E
C05A A0 00        LDY #$00
C05C A9 04        LDA #$04
C05E 84 FD        STY $FD
C060 85 FE        STA $FE
C062 8A           TXA
C063 91 FD        STA ($FD),Y
C065 C8           INY
C066 D0 FB        BNE $C063
C068 E6 FE        INC $FE
C06A A5 FE        LDA $FE
C06C C9 08        CMP #$08
C06E D0 F2        BNE $C062
C070 60           RTS
C071 A0 00        LDY #$00
C073 A9 20        LDA #$20
C075 84 FD        STY $FD
C077 85 FE        STA $FE
C079 B1 FD        LDA ($FD),Y
C07B 49 FF        EOR #$FF
C07D 91 FD        STA ($FD),Y
C07F C8           INY
C080 D0 F7        BNE $C079
C082 E6 FE        INC $FE
C084 A5 FE        LDA $FE
C086 C9 40        CMP #$40
C088 D0 EF        BNE $C079
C08A 60           RTS
C08B A9 00        LDA #$00
C08D 2C A9 80     BIT $80A9
C090 85 97        STA $97
C092 20 FD AE     JSR $AEFD
C095 20 EB B7     JSR $B7EB
C098 E0 C8        CPX #$C8
C09A B0 EE        BCS $C08A
C09C A5 15        LDA $15
C09E C9 01        CMP #$01
C0A0 90 08        BCC $C0AA
C0A2 D0 E6        BNE $C08A
C0A4 A5 14        LDA $14
C0A6 C9 40        CMP #$40
C0A8 B0 E0        BCS $C08A
C0AA 8A           TXA
C0AB 4A           LSR
C0AC 4A           LSR
C0AD 4A           LSR
C0AE A8           TAY
C0AF B9 21 C1     LDA $C121,Y
C0B2 8D 75 C1     STA $C175
C0B5 B9 08 C1     LDA $C108,Y
C0B8 8D 76 C1     STA $C176
```

```
C0BB 8A          TXA
C0BC 29 07       AND #$07
C0BE 18          CLC
C0BF 6D 75 C1    ADC $C175
C0C2 8D 75 C1    STA $C175
C0C5 A5 14       LDA $14
C0C7 29 F8       AND #$F8
C0C9 8D 74 C1    STA $C174
C0CC 18          CLC
C0CD A9 00       LDA #$00
C0CF 6D 75 C1    ADC $C175
C0D2 85 FD       STA $FD
C0D4 A9 20       LDA #$20
C0D6 6D 76 C1    ADC $C176
C0D9 85 FE       STA $FE
C0DB 18          CLC
C0DC A5 FD       LDA $FD
C0DE 6D 74 C1    ADC $C174
C0E1 85 FD       STA $FD
C0E3 A5 FE       LDA $FE
C0E5 65 15       ADC $15
C0E7 85 FE       STA $FE
C0E9 A5 14       LDA $14
C0EB 29 07       AND #$07
C0ED 49 07       EOR #$07
C0EF AA          TAX
C0F0 A9 01       LDA #$01
C0F2 CA          DEX
C0F3 30 03       BMI $C0F8
C0F5 0A          ASL
C0F6 D0 FA       BNE $C0F2
C0F8 A0 00       LDY #$00
C0FA 24 97       BIT $97
C0FC 10 05       BPL $C103
C0FE 49 FF       EOR #$FF
C100 31 FD       AND ($FD),Y
C102 2C 11 FD    BIT $FD11
C105 91 FD       STA ($FD),Y
C107 60          RTS
C108 00          BRK
C109 00          BRK
C10A 01 02       ORA ($02,X)
C10C 03          ???
C10D 05 06       ORA $06
C10F 07          ???
C110 08          PHP
C111 0A          ASL
C112 0B          ???
C113 0C          ???
C114 0D 0F 10    ORA $100F
C117 11 12       ORA ($12),Y
C119 14          ???
```

```
C11A 15 16       ORA $16,X
C11C 17          ???
C11D 19 1B 1C    ORA $1C1B,Y
C120 1D 00 40    ORA $4000,X
C123 80          ???
C124 C0 00       CPY #$00
C126 40          RTI
C127 80          ???
C128 C0 00       CPY #$00
C12A 40          RTI
C12B 80          ???
C12C C0 00       CPY #$00
C12E 40          RTI
C12F 80          ???
C130 C0 00       CPY #$00
C132 40          RTI
C133 80          ???
C134 C0 00       CPY #$00
C136 40          RTI
C137 80          ???
C138 C0 00       CPY #$00
C13A 20 FD AE    JSR $AEFD
C13D 20 D4 E1    JSR $E1D4
C140 A2 00       LDX #$00
C142 A0 40       LDY #$40
C144 A9 00       LDA #$00
C146 85 FD       STA $FD
C148 A9 20       LDA #$20
C14A 85 FE       STA $FE
C14C A9 FD       LDA #$FD
C14E 20 D8 FF    JSR $FFD8
C151 60          RTS
C152 20 FD AE    JSR $AEFD
C155 20 D4 E1    JSR $E1D4
C158 A9 61       LDA #$61
C15A 85 B9       STA $B9
C15C A9 00       LDA #$00
C15E 20 D5 FF    JSR $FFD5
C161 60          RTS
C162 AD 72 C1    LDA $C172
C165 8D 11 D0    STA $D011
C168 AD 73 C1    LDA $C173
C16B 8D 18 D0    STA $D018
C16E 20 44 E5    JSR $E544
C171 60          RTS
C172 00          BRK
C173 00          BRK
C174 00          BRK
C175 00          BRK
C176 00          BRK
C177 00          BRK
C178 00          BRK
```

```
C179 00         BRK
C17A 00         BRK
C17B 00         BRK
C17C 00         BRK
C17D 00         BRK
C17E 00         BRK
C17F 00         BRK
C180 48         PHA
C181 4A         LSR
C182 48         PHA
C183 98         TYA
C184 48         PHA
C185 08         PHP
C186 A5 01      LDA $01
C188 29 FE      AND #$FE
C18A 85 01      STA $01
C18C A9 00      LDA #$00
C18E AA         TAX
C18F A8         TAY
C190 20 BD FF   JSR $FFBD
C193 A9 04      LDA #$04
C195 AA         TAX
C196 A0 FF      LDY #$FF
C198 20 BA FF   JSR $FFBA
C19B 20 C0 FF   JSR $FFC0
C19E A2 04      LDX #$04
C1A0 20 C9 FF   JSR $FFC9
C1A3 A9 1B      LDA #$1B
C1A5 20 D2 FF   JSR $FFD2
C1A8 A9 41      LDA #$41
C1AA 20 D2 FF   JSR $FFD2
C1AD A9 08      LDA #$08
C1AF 20 D2 FF   JSR $FFD2
C1B2 A9 19      LDA #$19
C1B4 85 FB      STA $FB
C1B6 A9 00      LDA #$00
C1B8 85 F7      STA $F7
C1BA A9 20      LDA #$20
C1BC 85 F8      STA $F8
C1BE A9 1B      LDA #$1B
C1C0 20 D2 FF   JSR $FFD2
C1C3 A9 4B      LDA #$4B
C1C5 20 D2 FF   JSR $FFD2
C1C8 A9 40      LDA #$40
C1CA 20 D2 FF   JSR $FFD2
C1CD A9 01      LDA #$01
C1CF 20 D2 FF   JSR $FFD2
C1D2 A9 28      LDA #$28
C1D4 8D 34 03   STA $0334
C1D7 A5 F7      LDA $F7
C1D9 85 F9      STA $F9
C1DB A5 F8      LDA $F8
```

```
C1DD 85 FA       STA $FA
C1DF A9 08       LDA #$08
C1E1 85 FC       STA $FC
C1E3 A2 00       LDX #$00
C1E5 A0 01       LDY #$01
C1E7 A1 F9       LDA ($F9,X)
C1E9 99 75 C2    STA $C275,Y
C1EC C6 FC       DEC $FC
C1EE F0 0F       BEQ $C1FF
C1F0 C8          INY
C1F1 18          CLC
C1F2 A9 01       LDA #$01
C1F4 65 F9       ADC $F9
C1F6 85 F9       STA $F9
C1F8 90 02       BCC $C1FC
C1FA E6 FA       INC $FA
C1FC 4C E7 C1    JMP $C1E7
C1FF A9 08       LDA #$08
C201 85 FC       STA $FC
C203 A2 08       LDX #$08
C205 1E 75 C2    ASL $C275,X
C208 6E 75 C2    ROR $C275
C20B CA          DEX
C20C D0 F7       BNE $C205
C20E AD 75 C2    LDA $C275
C211 20 D2 FF    JSR $FFD2
C214 C6 FC       DEC $FC
C216 D0 EB       BNE $C203
C218 18          CLC
C219 A9 08       LDA #$08
C21B 65 F7       ADC $F7
C21D 85 F7       STA $F7
C21F 90 02       BCC $C223
C221 E6 F8       INC $F8
C223 CE 34 03    DEC $0334
C226 F0 03       BEQ $C22B
C228 4C D7 C1    JMP $C1D7
C22B A9 0D       LDA #$0D
C22D 20 D2 FF    JSR $FFD2
C230 C6 FB       DEC $FB
C232 F0 03       BEQ $C237
C234 4C BE C1    JMP $C1BE
C237 A9 07       LDA #$07
C239 20 D2 FF    JSR $FFD2
C23C A9 1B       LDA #$1B
C23E 20 D2 FF    JSR $FFD2
C241 A9 40       LDA #$40
C243 20 D2 FF    JSR $FFD2
C246 20 CC FF    JSR $FFCC
C249 A5 01       LDA $01
C24B 09 01       ORA #$01
C24D 85 01       STA $01
```

```
C24F 28         PLP
C250 68         PLA
C251 A8         TAY
C252 68         PLA
C253 A8         TAY
C254 68         PLA
C255 60         RTS
.
.
```

```
.:C000 4C 1E C0 4C 3D C0 4C 54
.:C008 C0 4C 71 C0 4C 8B C0 4C
.:C010 8E C0 4C 52 C1 4C 3A C1
.:C018 4C 62 C1 4C 80 C1 AD 11
.:C020 D0 8D 72 C1 AD 18 D0 8D
.:C028 73 C1 A9 3B 8D 11 D0 A9
.:C030 18 8D 18 D0 20 3D C0 A2
.:C038 10 20 5A C0 60 A0 00 A9
.:C040 20 84 FD 85 FE 98 91 FD
.:C048 C8 D0 FB E6 FE A5 FE C9
.:C050 40 D0 F2 60 20 FD AE 20
.:C058 9E B7 A0 00 A9 04 84 FD
.:C060 85 FE 8A 91 FD C8 D0 FB
.:C068 E6 FE A5 FE C9 08 D0 F2
.:C070 60 A0 00 A9 20 84 FD 85
.:C078 FE B1 FD 49 FF 91 FD C8
.:C080 D0 F7 E6 FE A5 FE C9 40
.:C088 D0 EF 60 A9 00 2C A9 80
.:C090 85 97 20 FD AE 20 EB B7
.:C098 E0 C8 B0 EE A5 15 C9 01
.:C0A0 90 08 D0 E6 A5 14 C9 40
.:C0A8 B0 E0 8A 4A 4A 4A A8 B9
.:C0B0 21 C1 8D 75 C1 B9 08 C1
.:C0B8 8D 76 C1 8A 29 07 18 6D
.:C0C0 75 C1 8D 75 C1 A5 14 29
.:C0C8 F8 8D 74 C1 18 A9 00 6D
.:C0D0 75 C1 85 FD A9 20 6D 76
.:C0D8 C1 85 FE 18 A5 FD 6D 74
.:C0E0 C1 85 FD A5 FE 65 15 85
.:C0E8 FE A5 14 29 07 49 07 AA
.:C0F0 A9 01 CA 30 03 0A D0 FA
.:C0F8 A0 00 24 97 10 05 49 FF
.:C100 31 FD 2C 11 FD 91 FD 60
.:C108 00 00 01 02 03 05 06 07
.:C110 08 0A 0B 0C 0D 0F 10 11
```

```
.:C118 12 14 15 16 17 19 1B 1C
.:C120 1D 00 40 80 C0 00 40 80
.:C128 C0 00 40 80 C0 00 40 80
.:C130 C0 00 40 80 C0 00 40 80
.:C138 C0 00 20 FD AE 20 D4 E1
.:C140 A2 00 A0 40 A9 00 85 FD
.:C148 A9 20 85 FE A9 FD 20 D8
.:C150 FF 60 20 FD AE 20 D4 E1
.:C158 A9 61 85 B9 A9 00 20 D5
.:C160 FF 60 AD 72 C1 8D 11 D0
.:C168 AD 73 C1 8D 18 D0 20 44
.:C170 E5 60 00 00 00 00 00 00
.:C178 00 00 00 00 00 00 00 00
.:C180 48 4A 48 98 48 08 A5 01
.:C188 29 FE 85 01 A9 00 AA A8
.:C190 20 BD FF A9 04 AA A0 FF
.:C198 20 BA FF 20 C0 FF A2 04
.:C1A0 20 C9 FF A9 1B 20 D2 FF
.:C1A8 A9 41 20 D2 FF A9 08 20
.:C1B0 D2 FF A9 19 85 FB A9 00
.:C1B8 85 F7 A9 20 85 F8 A9 1B
.:C1C0 20 D2 FF A9 4B 20 D2 FF
.:C1C8 A9 40 20 D2 FF A9 01 20
.:C1D0 D2 FF A9 28 8D 34 03 A5
.:C1D8 F7 85 F9 A5 F8 85 FA A9
.:C1E0 08 85 FC A2 00 A0 01 A1
.:C1E8 F9 99 75 C2 C6 FC F0 0F
.:C1F0 C8 18 A9 01 65 F9 85 F9
.:C1F8 90 02 E6 FA 4C E7 C1 A9
.:C200 08 85 FC A2 08 1E 75 C2
.:C208 6E 75 C2 CA D0 F7 AD 75
.:C210 C2 20 D2 FF C6 FC D0 EB
.:C218 18 A9 08 65 F7 85 F7 90
.:C220 02 E6 F8 CE 34 03 F0 03
.:C228 4C D7 C1 A9 0D 20 D2 FF
.:C230 C6 FB F0 03 4C BE C1 A9
.:C238 07 20 D2 FF A9 1B 20 D2
.:C240 FF A9 40 20 D2 FF 20 CC
.:C248 FF A5 01 09 01 85 01 28
.:C250 68 A8 68 A8 68 60 00 00
.
.
```

Borders

Drawing a border in Basic is fairly simple but very slow. You could go and make your lunch while you wait! Just to prove the point, there is a Basic subroutine included here that will give you a border from Basic.

Basic border

The line numbers start from 63000, but can be changed to suit your needs. The variable 'C' is set to the difference between the normal screen and the colour memory 54272. The variable 'SC' is set to the start of the screen.

The next line places the first part of the border along the top of the screen using the reversed space character (160) and makes the border yellow (7). The next loop places a yellow border along the bottom of the screen, and the last two loops place the two sides of the border by looping through the screen locations with a step of 40. This is of course fairly simple in Basic. The same thing in machine code is a little tricky although well worth the effort, if only for the speed.

```
63000 C=54272:SC=1024
63010 FORA=SCTOSC+39:POKEA,160:POKEA+C,7:NEXT
63020 FORI=SC+40TO1983STEP40:POKEI,160:POKEI+C,7:NEXT
63030 FORI=SC+39TO1983STEP40:POKEI,160:POKEI+C,7:NEXT
63040 FORA=1984TO2023:POKEA,160:POKEA+C,7:NEXT
```

Code border

The program included here will do exactly the same as the Basic program, but so quickly that it is not really possible to see it happening.

The code sits from $C000 to $C09E hex, but could of course be relocated to a place of your choice. First the border colour is set to blue and the screen colour to red. The screen is then cleared using the CHROUT Kernal routine ($FFD2 hex).

The 'X' register is then loaded with the screen length and the accumulator with the reversed space ($A0 hex). Using the 'X' register as an offset a reversed space is stored in the top right of the screen and the vaulue for yellow is loaded into the accumulator and stored in the top right of colour memory.

This gives us one yellow reversed space in the top right of the screen. The contents of the 'X' register are then decremented and the process continues until the 'X' register contains zero and our top line is complete.

The same process is then carried out for the bottom line and the carry flag is cleared. In order to draw the border at the sides of the screen the start addresses of the screen and colour memory are first stored and then manipulated using the 'Y' register to step through the screen and colour memory and place the reversed space and the colour in the correct positions.

The code is a little long-winded and there are better ways of writing it, but it is presented in this form as it is fairly easy to follow the program flow.

```
C000 A9 06        LDA #$06
C002 8D 20 D0     STA $D020
C005 A9 02        LDA #$02
C007 8D 21 D0     STA $D021
C00A A9 93        LDA #$93
C00C 20 D2 FF     JSR $FFD2
C00F A2 28        LDX #$28
C011 A9 A0        LDA #$A0
C013 9D FF 03     STA $03FF,X
C016 A9 07        LDA #$07
C018 9D FF D7     STA $D7FF,X
C01B CA           DEX
C01C D0 F3        BNE $C011
C01E A2 28        LDX #$28
C020 A9 A0        LDA #$A0
C022 9D BF 07     STA $07BF,X
C025 A9 07        LDA #$07
C027 9D BF DB     STA $DBBF,X
C02A CA           DEX
C02B D0 F3        BNE $C020
C02D 18           CLC
C02E A9 27        LDA #$27
C030 85 FB        STA $FB
C032 A9 04        LDA #$04
C034 85 FC        STA $FC
C036 A9 27        LDA #$27
C038 85 FD        STA $FD
C03A A9 D8        LDA #$D8
C03C 85 FE        STA $FE
C03E A0 00        LDY #$00
C040 A9 A0        LDA #$A0
C042 91 FB        STA ($FB),Y
C044 A9 27        LDA #$27
C046 65 FB        ADC $FB
C048 85 FB        STA $FB
C04A A9 00        LDA #$00
C04C 65 FC        ADC $FC
C04E 85 FC        STA $FC
```

```
C050 A9 07        LDA #$07
C052 91 FD        STA ($FD),Y
C054 A9 27        LDA #$27
C056 65 FD        ADC $FD
C058 85 FD        STA $FD
C05A A9 00        LDA #$00
C05C 65 FE        ADC $FE
C05E 85 FE        STA $FE
C060 C8           INY
C061 C0 18        CPY #$18
C063 D0 DB        BNE $C040
C065 18           CLC
C066 A9 28        LDA #$28
C068 85 FB        STA $FB
C06A A9 04        LDA #$04
C06C 85 FC        STA $FC
C06E A9 28        LDA #$28
C070 85 FD        STA $FD
C072 A9 D8        LDA #$D8
C074 85 FE        STA $FE
C076 A0 00        LDY #$00
C078 A9 A0        LDA #$A0
C07A 91 FB        STA ($FB),Y
C07C A9 27        LDA #$27
C07E 65 FB        ADC $FB
C080 85 FB        STA $FB
C082 A9 00        LDA #$00
C084 65 FC        ADC $FC
C086 85 FC        STA $FC
C088 A9 07        LDA #$07
C08A 91 FD        STA ($FD),Y
C08C A9 27        LDA #$27
C08E 65 FD        ADC $FD
C090 85 FD        STA $FD
C092 A9 00        LDA #$00
C094 65 FE        ADC $FE
C096 85 FE        STA $FE
C098 C8           INY
C099 C0 18        CPY #$18
C09B D0 DB        BNE $C078
C09D 18           CLC
C09E 60           RTS
.
.
```

```
.:C000 A9 06 8D 20 D0 A9 02 8D
.:C008 21 D0 A9 93 20 D2 FF A2
.:C010 28 A9 A0 9D FF 03 A9 07
```

```
.:C018 9D FF D7 CA D0 F3 A2 28
.:C020 A9 A0 9D BF 07 A9 07 9D
.:C028 BF DB CA D0 F3 18 A9 27
.:C030 85 FB A9 04 85 FC A9 27
.:C038 85 FD A9 D8 85 FE A0 00
.:C040 A9 A0 91 FB A9 27 65 FB
.:C048 85 FB A9 00 65 FC 85 FC
.:C050 A9 07 91 FD A9 27 65 FD
.:C058 85 FD A9 00 65 FE 85 FE
.:C060 C8 C0 18 D0 DB 18 A9 28
.:C068 85 FB A9 04 85 FC A9 28
.:C070 85 FD A9 D8 85 FE A0 00
.:C078 A9 A0 91 FB A9 27 65 FB
.:C080 85 FB A9 00 65 FC 85 FC
.:C088 A9 07 91 FD A9 27 65 FD
.:C090 85 FD A9 00 65 FE 85 FE
.:C098 C8 C0 18 D0 DB 18 60 00
.
.
```

Colour border

This routine is the same, but tagged on to the end of it are a few lines of code that change the interrupts to point at a routine from $C0AF to $C0D1 hex. This routine loops through the top line of the screen and changes the colour character. It loops through all the available colours so quickly that they are a blur. Note that the operating speed of the 64 is not noticeably affected.

```
B*
   PC  SR AC XR YR SP
.;0008 B0 C0 00 18 F6
.
C000 A9 06       LDA #$06
C002 8D 20 D0    STA $D020
C005 A9 02       LDA #$02
C007 8D 21 D0    STA $D021
C00A A9 93       LDA #$93
C00C 20 D2 FF    JSR $FFD2
C00F A2 28       LDX #$28
C011 A9 A0       LDA #$A0
C013 9D FF 03    STA $03FF,X
C016 A9 07       LDA #$07
C018 9D FF D7    STA $D7FF,X
C01B CA          DEX
C01C D0 F3       BNE $C011
C01E A2 28       LDX #$28
C020 A9 A0       LDA #$A0
```

```
C022 9D BF 07     STA $07BF,X
C025 A9 07        LDA #$07
C027 9D BF DB     STA $DBBF,X
C02A CA           DEX
C02B D0 F3        BNE $C020
C02D 18           CLC
C02E A9 27        LDA #$27
C030 85 FB        STA $FB
C032 A9 04        LDA #$04
C034 85 FC        STA $FC
C036 A9 27        LDA #$27
C038 85 FD        STA $FD
C03A A9 D8        LDA #$D8
C03C 85 FE        STA $FE
C03E A0 00        LDY #$00
C040 A9 A0        LDA #$A0
C042 91 FB        STA ($FB),Y
C044 A9 27        LDA #$27
C046 65 FB        ADC $FB
C048 85 FB        STA $FB
C04A A9 00        LDA #$00
C04C 65 FC        ADC $FC
C04E 85 FC        STA $FC
C050 A9 07        LDA #$07
C052 91 FD        STA ($FD),Y
C054 A9 27        LDA #$27
C056 65 FD        ADC $FD
C058 85 FD        STA $FD
C05A A9 00        LDA #$00
C05C 65 FE        ADC $FE
C05E 85 FE        STA $FE
C060 C8           INY
C061 C0 18        CPY #$18
C063 D0 DB        BNE $C040
C065 18           CLC
C066 A9 28        LDA #$28
C068 85 FB        STA $FB
C06A A9 04        LDA #$04
C06C 85 FC        STA $FC
C06E A9 28        LDA #$28
C070 85 FD        STA $FD
C072 A9 D8        LDA #$D8
C074 85 FE        STA $FE
C076 A0 00        LDY #$00
C078 A9 A0        LDA #$A0
C07A 91 FB        STA ($FB),Y
C07C A9 27        LDA #$27
C07E 65 FB        ADC $FB
C080 85 FB        STA $FB
C082 A9 00        LDA #$00
C084 65 FC        ADC $FC
C086 85 FC        STA $FC
```

```
C088 A9 07        LDA #$07
C08A 91 FD        STA ($FD),Y
C08C A9 27        LDA #$27
C08E 65 FD        ADC $FD
C090 85 FD        STA $FD
C092 A9 00        LDA #$00
C094 65 FE        ADC $FE
C096 85 FE        STA $FE
C098 C8           INY
C099 C0 18        CPY #$18
C09B D0 DB        BNE $C078
C09D 18           CLC
C09E A9 00        LDA #$00
C0A0 85 FD        STA $FD
C0A2 78           SEI
C0A3 A9 AF        LDA #$AF
C0A5 8D 14 03     STA $0314
C0A8 A9 C0        LDA #$C0
C0AA 8D 15 03     STA $0315
C0AD 58           CLI
C0AE 60           RTS
C0AF A9 00        LDA #$00
C0B1 85 FB        STA $FB
C0B3 A9 D8        LDA #$D8
C0B5 85 FC        STA $FC
C0B7 A0 00        LDY #$00
C0B9 A5 FD        LDA $FD
C0BB 91 FB        STA ($FB),Y
C0BD C8           INY
C0BE C0 28        CPY #$28
C0C0 D0 F9        BNE $C0BB
C0C2 E6 FD        INC $FD
C0C4 A5 FD        LDA $FD
C0C6 C9 07        CMP #$07
C0C8 F0 03        BEQ $C0CD
C0CA 4C 31 EA     JMP $EA31
C0CD A9 00        LDA #$00
C0CF 85 FD        STA $FD
C0D1 4C 31 EA     JMP $EA31
.
.
```

```
.:C000 A9 06 8D 20 D0 A9 02 8D
.:C008 21 D0 A9 93 20 D2 FF A2
.:C010 28 A9 A0 9D FF 03 A9 07
.:C018 9D FF D7 CA D0 F3 A2 28
.:C020 A9 A0 9D BF 07 A9 07 9D
.:C028 BF DB CA D0 F3 18 A9 27
```

```
.:C030 85 FB A9 04 85 FC A9 27
.:C038 85 FD A9 D8 85 FE A0 00
.:C040 A9 A0 91 FB A9 27 65 FB
.:C048 85 FB A9 00 65 FC 85 FC
.:C050 A9 07 91 FD A9 27 65 FD
.:C058 85 FD A9 00 65 FE 85 FE
.:C060 C8 C0 18 D0 DB 18 A9 28
.:C068 85 FB A9 04 85 FC A9 28
.:C070 85 FD A9 D8 85 FE A0 00
.:C078 A9 A0 91 FB A9 27 65 FB
.:C080 85 FB A9 00 65 FC 85 FC
.:C088 A9 07 91 FD A9 27 65 FD
.:C090 85 FD A9 00 65 FE 85 FE
.:C098 C8 C0 18 D0 DB 18 A9 00
.:C0A0 85 FD 78 A9 AF 8D 14 03
.:C0A8 A9 C0 8D 15 03 58 60 A9
.:C0B0 00 85 FB A9 D8 85 FC A0
.:C0B8 00 A5 FD 91 FB C8 C0 28
.:C0C0 D0 F9 E6 FD A5 FD C9 07
.:C0C8 F0 03 4C 31 EA A9 00 85
.:C0D0 FD 4C 31 EA 00 00 00 00
.
.
```

Basic graph

To finish the book here is a small Basic program that will plot a sine wave in Basic using the 64's on board graphics. This should also serve you well for calculations for the hi-res routine.

The program draws the axis for the sine wave and uses the variables 'SC' and 'C' for screen and colour memory. The rest of the routine is fairly straightforward as the program calculates and outputs the sine wave.

The routine at lines 230 to 260 checks for the position of the sine wave and changes the character used to draw the wave. Line 270 does the actual display and colouring.

```
100 DEFFNP(X)=SIN(X/6.28)
110 PRINT"[CLR]":SC=1024:C=54272
120 FORA=SCTO1984STEP40:POKEA,101:POKEA+C,7:NEXT
130 PRINT"[YEL][HME][12 CD][SH L][39 LO P]"
140 FORX=1TO79
150 Y1=FNP(X)
160 Y=24+24*Y1
170 X2=INT(X/2):Y2=INT(Y/2)
```

```
180 IFX2>39ORY2>25THEN280
190 X1=X/2-X2:Y1=Y/2-Y2
200 A=1984-Y2*40+X2:CO=A+54272
210 IFX1<.5THENX1=0
220 IFY1<.5THENY1=0
230 IFX1=0ANDY1=0THENC=123:GOTO270
240 IFX1<>0ANDY1<>0THENC=124:GOTO270
250 IFX1<>0ANDY1<>0THENC=108:GOTO270
260 IFX1<>0ANDY1<>0THENC=126
270 POKEA,C:POKECO,7
280 NEXTX
```

Farewell

A close friend of mine thought that it would be rather nice to end the main part of the book with a few sentences from me rather than a program. So here they are.

I hope that you have enjoyed the book and that it has given you many wonderful ideas. I have particularly enjoyed writing this book, despite problems with my printer and 'interpod' and despite the deadline... I must now rush off to my publishers and will look forward to writing for you again.

Appendix A

64 memory map revisited

By now many memory maps for the 64 have been published. However, every programmer likes to feel that he has published a copy with more information than any other. Anyone reading this map can be sure that I have included everything I possibly can in it. To my mind this justifies its inclusion.

All locations are given in hex and decimal, and some locations may include extensive comments. The hex numbers are in the left-hand column.

0000	0	Chip directional register
0001 - 0002	1 - 2	Chip I/O & tape control (Bit 0; 0 = switch out Basic ROM) (Bit 1; 0 = switch out Kernal) (Bit 2; 0 = switch in Character generator) (Bit 3; 1 = cassette write line output) (Bit 4; 0 = cassette switch sense input) (Bit 5; 0 = cassette motor on; 1 = off)
0003 - 0004	3 - 4	Floating point & fixed point vector
0005 - 0006	5 - 6	Fixed point & floating point vector
0007	7	Search character for end of line
0008	8	Scan-quotes flag
0009	9	Column position of cursor on line
000A	10	Flag; 0 = load; 1 = verify

000B	11	BASIC input buffer pointer/ subscript no.
000C	12	Default DIM flag
000D	13	Variable flag; FF = string; 00 = numeric
000E	14	Numeric flag: 80 = integer; 00 = floating point
000F	15	Flag; DATA scan; LIST quote; memory
0010	16	Flag; Subscript - FNx
0011	17	Flag; 0 = INPUT; 152 = READ; 64 = GET
0012	18	Flag; ATN sign - comparision evaluation
0013	19	Current I/O prompt flag (1 = prompt off)
0014 - 0015	20 - 21	BASIC stores integer values here
0016	22	Pointer; temporary string stack
0017 - 0018	23 - 24	Last temporary string vector
0019 - 0021	25 - 33	Stack for temporary string descriptors
0022 - 0025	34 - 37	Utility pointer area
0026 - 002A	38 - 42	Product area for multiplication
002B - 002C	43 - 44	Pointer; start of BASIC program (normally 1 & 8, but start of BASIC can be changed by altering values)
002D - 002E	45 - 46	Pointer; start of BASIC variables - end of current BASIC program

002F - 0030	47 - 48	Pointer; start of arrays - end of variables
0031 - 0032	49 - 50	Pointer; end of arrays
0033 - 0034	51 - 52	Pointer; start of string storage (moves down from from top of available memory to arrays and OUT OF MEMORY)
0035 - 0036	53 - 54	Pointer; end of string storage
0037 - 0038	55 - 56	Pointer; to top of current RAM available to BASIC (alter these values to reset top of RAM)
0039 - 003A	57 - 58	Current BASIC line number
003B - 003C	59 - 60	Previous BASIC line number
003D - 003E	61 - 62	Pointer; BASIC statement for CONT
003F - 0040	63 - 64	Current DATA line number
0041 - 0042	65 - 66	Pointer; current DATA item
0043 - 0044	67 - 68	Vector; jump for INPUT statement
0045 - 0046	69 - 70	Current variable name
0047 - 0048	71 - 72	Current variable address
0049 - 004A	73 - 74	Variable pointer for FOR - NEXT statement
004B - 004C	75 - 76	Y-save; operator-save; BASIC pointer-save
004D	77	Comparison symbol
004E - 004F	78 - 79	Work area; function definition pointer
0050 - 0051	80 - 81	Work area; string descriptor pointer

0052	82	Length of string
0053	83	Garbage collect use
0054 - 0056	84 - 86	Jump vector for functions
0057 - 0060	87 - 96	Numeric work area
0061	97	Accumulator #1; exponent
0062 - 0065	98 - 101	Accumulator #1; mantissa
0066	102	Accumulator #1; sign
0067	103	Series evaluation constant pointer
0068	104	Accumulator #1; hi-order (overflow)
0069 - 006E	105 - 110	Accumulator #2; floating point
006F	111	Sign comparision; Accumulator 1 - Accumulator 2
0070	112	Accumulator #2; lo-order (rounding)
0071 - 0072	113 - 114	Cassette buffer length - series pointer
0073 - 008A	115 - 138	CHRGET BASIC subroutine; get next character (change routine to add new commands)
007A - 007B	122 - 123	BASIC pointer within routine
008B - 008F	139 - 143	RND storage and work area
0090	144	Status byte - ST
0091	145	Flag; STOP and RVS; Keyswitch PIA
0092	146	Timing constant for tape

0093	147	Flag; 0 = load; 1 = verify
0094	148	Serial output; deferred character flag
0095	149	Serial deferred character
0096	150	Tape EOT received
0097	151	Register save
0098	152	Number of open files
0099	153	Current input device; normally 0
009A	154	Current output device; normally 3
009B	155	Tape character parity
009C	156	Flag; Byte received
009D	157	Output control flag; $80 (128) = direct 0 = RUN
009E	158	Tape pass 1 error log; character buffer
009F	159	Tape pass 2 error log corrected
00A0 - 00A2	160 - 162	Jiffy clock - used by TI and TI$
00A3	163	Serial bit count; EOI flag
00A4	164	Cycle count
00A5	165	Countdown, tape write - bit count
00A6	166	Pointer; tape buffer
00A7	167	Tape write count; input bit storage
00A8	168	Tape write new byte; read error; input bit count
00A9	169	Write start bit; read bit error

00AA	170	Tape scan; count
00AB	171	Write read length; read checksum; parity
00AC - 00AD	172 - 173	Pointer; tape buffer - scrolling
00AE - 00AF	174 - 175	Tape end addresses; end of program
00B0 - 00B1	176 - 177	Tape timing constants
00B2 - 00B3	178 - 179	Pointer; start of tape buffer
00B4	180	Tape timer; bit count
00B5	181	Tape EOT - RS232 next bit to send
00B6	182	Read character error; output to buffer
00B7	183	Number of characters in current file name (needs to be set even if Kernal routines not used)
00B8	184	Current logical file number
00B9	185	Current secondary address
00BA	186	Current device number (tape, disk, etc)
00BB - 00BC	187 - 188	Pointer; to current file name
00BD	189	Write shift - read input character
00BE	190	Number of blocks remaining to write; read
00BF	191	Serial word buffer
00C0	192	Tape motor interlock; (along with loc. 1 controls the tape motor).
00C1 - 00C2	193 - 194	Tape I/O start address

00C3 - 00C4	195 - 196	Pointer; Kernal set up
00C5	197	Current key pressed (see key values)
00C6	198	No. of characters in keyboard buffer (can be used from direct or progam mode)
00C7	199	Flag; screen reverse; 1 = on; 0 = off
00C8	200	Pointer; end of line for input
00C9 - 00CA	201 - 202	Cursor log; row, column
00CB	203	Current key pressed
00CC	204	Flag; cursor blink; 0 = on
00CD	205	Cursor timing countdown
00CE	206	Character under cursor
00CF	207	Flag; cursor on or off
00D0	208	Input from screen or keyboard
00D1 - 00D2	209 - 210	Pointer; to screen line on which cursor appears
00D3	211	Position of cursor on line
00D4	212	0 = direct, else programmed
00D5	213	Current screen line length
00D6	214	Row were cursor lives (to change position 201, 210, 211, and 214 must be changed)
00D7	215	Ascii value of last character printed
00D8	216	Number of inserts outstanding
00D9 - 00F0	217 - 240	Screen line link table

00F1	241	Dummy screen line link
00F2	242	Screen row marker
00F3 - 00F4	243 - 244	Pointer; current loc. in colour memory
00F5 - 00F6	245 - 246	Keyboard pointer
00F7 - 00F8	247 - 248	Pointer; RS-232 receiver
00F9 - 00FA	249 - 250	Pointer; RS-232 transmitter
00FB - 00FE	251 - 254	Free zero page locations
00FF	255	BASIC storage
0100 - 010A	256 - 257	Floating to Ascii work area
0100 - 013E	256 - 318	Tape error log (can use part of this area 'carefully')
0100 - 01FF	256 - 511	Processor stack area
0200 - 0258	512 - 600	BASIC input buffer
0259 - 0262	601 - 610	Logical file table for OPEN files
0263 - 026C	611 - 620	Device number for OPEN files
026D - 0276	621 - 630	Secondary addresses table
0277 - 0280	631 - 640	Keyboard buffer (see $C6)
0281 - 0282	641 - 642	Pointer; start of memory for op. system
0283 - 0284	643 - 644	Pointer; end of memory for op. system
0285	645	Serial bus timeout flag
0286	646	Current colour code for character
0287	647	Colour under cursor

0288	648	Pointer; screen memory page (normally 4) (change value when switching screen)
0289	649	Maximum size of keyboard buffer (can be lengthened, but tricky)
028A	650	Key repeat; 0 normal; 255 repeat all
028B	651	Repeat speed counter
028C	652	Repeat delay counter
028D	653	Flag; keyboard SHIFT key CTRL key and C=keys; SHIFT = set bit 0; CTRL = set bit 1; C = set bit 2
028E	654	Last SHIFT pattern
028F - 0290	655 - 656	Pointer; keyboard table set up
0291	657	Keyboard shift mode; 0 = enabled; 128 = disabled
0292	658	Auto scroll 0 = enabled
0293	659	RS-232 control register
0294	660	RS-232 command register
0295 - 0296	661 - 662	Bit timing
0297	663	RS-232 status register
0298	664	Number of bits to send
0299 - 029A	665 - 666	RS-232 speed code
029B	667	RS-232 receive pointer
029C	668	RS-232 input pointer
029D	669	RS-232 transmit pointer
029E	670	RS-232 output pointer

029F - 02A0	671 - 672	IRQ save during tape I/O
02A1	673	CIA 2 (NMI) interrupt control
02A2	674	CIA 1 timer A control log
02A3	675	CIA 1 interrupt log
02A4	676	CIA 1 timer A enable flag
02A5	677	Screen row marker
02A6	678	PAL-NISC flag; 0 = NTSC, 1 = PAL
02A7 - 02BF	679 - 703	Unused (useful for m/c programs in header)
02C0 - 02FE	704 - 766	(Sprite 11)
0300 - 0301	768 - 769	Error message link
0302 - 0303	770 - 771	Basic warm start link
0304 - 0305	772 - 773	Crunch Basic tokens link
0306 - 0307	774 - 775	Print tokens link
0308 - 0309	776 - 777	Start new Basic code link
030A - 030B	778 - 779	Get arithmetic element link
030C	780	Temp A save during SYS
030D	781	Temp X save during SYS
030E	782	Temp Y save during SYS
030F	783	Temp P save during SYS
0310 - 0312	784 - 786	USR function jump
0314 - 1315	788 - 789	Hardware interrupt vector (norm = EA31)

0316 - 0317	790 - 791	Break interrupt vector (BRK) (norm = FE66)
0318 - 0319	792 - 793	NMI interrupt vector (norm = FE47)
031A - 031B	794 - 795	OPEN vector (norm = F34A)
031C - 031D	796 - 797	CLOSE vector (norm = F291)
031E - 031F	798 - 799	Set input device vector (norm = F20E)
0320 - 0321	800 - 801	Set output device vector (norm = F250)
0322 - 0323	802 - 803	Restore I/O vector (norm = F333)
0324 - 0325	804 - 805	Input vector (norm = F157)
0326 - 0327	806 - 807	Output vector (norm = F1CA)
0328 - 0329	808 - 809	Test-STOP key vector (norm = F6ED)
032A - 032B	810 - 811	GET vector (norm = F13E)
032C - 032D	812 - 813	Abort I/O vector (norm = F32F)
032E - 032F	814 - 815	Warm start vector (norm = FE66)
0330 - 0331	816 - 817	Load from device vector (norm = F4A5)
0332 - 0333	818 - 819	Save to device vector (norm = F5ED)
0334 - 033B	820 - 827	Unused
033C - 03FB	828 - 1019	Cassette buffer (useful for m/c programs when no tape I/O's are performed
0340 - 037E	832 - 895	Sprite 13
0380 - 03BE	896 - 958	Sprite 14

03C0 - 03FE	960 - 1022	Sprite 15
0400 - 07FF	1024 - 2039	Screen memory
078F - 07FF	2040 - 2047	Sprite pointers
0800 - 9FFF	2048 - 40959	BASIC RAM memory and variables
8000 - 9FFF	32768 - 40959	Alternate; ROM plug in area (if cartridge in at power up memory is re-structured and this area is not) available for user programs
A000 - BFFF	40960 - 49151	ROM; BASIC (underlying RAM can be switched in)
C000 - CFFF	49152 - 53247	Alternate; RAM (available for user programs and is also used as a buffer during I/O operations)
D000 - D02E	53248 - 53294	6566 video chip
D000 - DFFF	53248 - 57343	Character set (D000 - D1FF = Upper case) (D200 - D3FF = Graphics) (D400 - D5FF = Reversed upper case) (D600 - D7FF = Reversed graphics) (D800 - D9FF = Lower case) (DA00 - DBFF = Upper case & graphics) (DC00 - DDFF = Reversed lower case) (DE00 - DFFF = Reversed upper case & graphics)
D400 - D41C	54272 - 54300	Sound chip (SID) 6581
D800 - DBFF	55296 - 56319	Colour memory
DC00 - DC0F	56320 - 56335	Interface chip 1, IRQ (6526 CIA)
DC10 - DD0F	56576 - 56591	Interface chip 2, NMI (6526 CIA)
E000 - FFFF	57334 - 65535	ROM; operating system (underlying

		RAM can be switched in)
FF81 - FFF5	65409 - 65525	Jump table. Includes the following:
FF84	65412	Initialise I/O
FF87	65415	Initialise system constants
FF8A	65418	Kernal reset
FF8D	65421	Kernal move
FF90	65424	Flag status
FF93	65427	Send listen (secondary address)
FF96	65430	Send talk (secondary address)
FF99	65433	Read-Set top of memory
FF9C	65436	Read-Set bottom of memory
FF9F	65439	Read keyboard
FFA2	65442	Set timeout
FFA5	65445	Receive from serial bus
FFA8	65448	Send serial deferred
FFAB	65451	Send untalk
FFB7	65463	Get status
FFBA	65466	Save file details
FFBD	65469	Save filename data
FFC0	65472	Do open file (via OPEN vector 031A)
FFC3	65475	Close file (via CLOSE vector 031C)
FFC6	65478	Set input device (via Set input vector 031E)

FFC9	65481	Set output device (via Set output vector 0320)
FFCC	65484	Restore default I/O (via Restore I/O vector 0322)
FFCF	65487	INPUT (via Input vector 0324)
FFD2	65490	Output (via Output vector 0326)
FFD5	65493	Load program
FFD8	65496	Save program
FFDB	65499	Set time
FFDE	65502	Get time
FFE1	65505	Check STOP key (via test STOP vector 0328)
FFE4	65508	Get (via Get vector 032A)
FFE7	65511	Abort all files (via Abort I/O vector)
FFEA	65514	Bump clock
FFED	65517	Get screen size
FFF0	65520	Put/get row/column
FFF3	65523	Get I/O address

Commodore 64 – ROM Memory Map

A000; ROM control vectors
A00C; Keyword action vectors
A052; Function vectors
A080; Operator vectors
A09E; Keywords
A19E; Error messages
A328; Error message vectors
A365; Misc messages
A38A; Scan stack for FOR/GOSUB
A3B8; Move memory
A3FB; Check stack depth
A408; Check memory space
A435; 'out of memory'
A437; Error routine
A469; BREAK entry
A474; 'ready.'
A480; Ready for Basic
A49C; Handle new line
A533; Re–chain lines
A560; Receive input line
A579; Crunch tokens
A613; Find Basic line
A642; Perform [NEW]
A65E; Perform [CLR]
A68E; Back up text pointer
A69C; Perform [LIST]
A742; Perform [FOR]
A7ED; Execute statement
A81D; Perform [RESTORE]
A82C; Break
A82F; Perform [STOP]
A831; Perform [END]
A857; Perform [CONT]
A871; Perform [RUN]
A883; Perform [GOSUB]
A8A0; Perform [GOTO]
A8D2; Perform [RETURN]
A8F8; Perform [DATA]
A906; Scan for next statement
A928; Perform [IF]
A93B; Perform [REM]
A94B; Perform [ON]
A96B; Get fixed point number
A9A5; Perform [LET]
AA80; Perform [PRINT*]
AA86; Perform [CMD]
AAA0; Perform [PRINT]
AB1E; Print string from (y.a)
AB3B; Print format character
AB4D; Bad input routine
AB7B; Perform [GET]
ABA5; Perform [INPUT*]
ABBF; Perform [INPUT]
ABF9; Prompt & input
AC06; Perform [READ]
ACFC; Input error messages
AD1E; Perform [NEXT]
AD78; Type match check
AD9E; Evaluate expression
AEA8; Constant – pi
AEF1; Evaluate within brackets
AEF7; ')'
AEFF; comma..
AF08; Syntax error
AF14; Check range
AF28; Search for variable
AFA7; Setup FN reference
AFE6; Perform [OR]
AFE9; Perform [AND]
B016; Compare
B081; Perform [DIM]
B08B; Locate variable
B113; Check alphabetic
B11D; Create variable
B194; Array pointer subrtine
B1A5; Value 32768
B1B2; Float–fixed
B1D1; Set up array
B245; 'bad subscript'
B248; 'illegal quantity'
B34C; Compute array size
B37D; Perform [FRE]
B391; Fix–float
B39E; Perform [POS]
B3A6; Check direct
B3B3; Perform [DEF]
B3E1; Check fn syntax
B3F4; Perform [FN]
B465; Perform [STR$]
B475; Calculate string vector
B487; Set up string
B4F4; Make room for string
B526; Garbage collection
B5BD; Check salvageability
B606; Collect string
B63D; Concatenate
B67A; Build string to memory
B6A3; Discard unwanted string
B6DB; Clean descriptor stack
B6EC; Perform [CHR$]
B700; Perform [LEFT$]
B72C; Perform [RIGHT$]
B737; Perform [MID$]
B761; Pull string parameters
B77C; Perform [LEN]
B782; Exit string–mode
B78B; Perform [ASC]
B79B; Input byte paramter
B7AD; Perform [VAL]
B7EB; Parameters for POKE/WAIT
B7F7; Float–fixed
B80D; Perform [PEEK]
B824; Perform [POKE]
B82D; Perform [WAIT]

B849;	Add 0.5
B850;	Subtract-from
B853;	Perform [subtract]
B86A;	Perform [add]
B947;	Complement FAC*1
B97E;	'overflow'
B983;	Multiply by zero byte
B9EA;	Perform [LOG]
BA2B;	Perform [multiply]
BA59;	Multiply-a-bit
BA8C;	Memory to FAC*2
BAB7;	Adjust FAC*1/*2
BAD4;	Underflow/overflow
BAE2;	Multiply by 10
BAF9;	+10 in floating pt
BAFE;	Divide by 10
BB12;	Perform [divide]
BBA2;	Memory to FAC*1
BBC7;	FAC*1 to memory
BBFC;	FAC*2 to FAC*1
BC0C;	FAC*1 to FAC*2
BC1B;	Round FAC*1
BC2B;	Get sign
BC39;	Perform [SGN]
BC58;	Perform [ABS]
BC5B;	Compare FAC*1 to mem
BC9B;	Float-fixed
BCCC;	Perform [int]
BCF3;	String to FAC
BD7E;	Get ascii digit
BDC2;	Print 'IN..'
BDCD;	Print line number
BDDD;	Float to ascii
BF16;	Decimal constants
BF3A;	TI constants
BF71;	Perform [SQR]
BF7B;	Perform [power]
BFB4;	Perform [negative]
BFED;	Perform [EXP]
E043;	Series eval 1
E059;	Series eval 2
E097;	Perform [RND]
E0f9;	?? breakpoints ??
E12A;	Perform [SYS]
E156;	Perform [SAVE]
E165;	Perform [VERIFY]
E168;	Perform [LOAD]
E1BE;	Perform [OPEN]
E1C7;	Perform [CLOSE]
E1D4;	Parameters for LOAD/SAVE
E206;	Check default parameters
E20E;	Check for comma
E219;	Parameters for open/close
E264;	Perform [COS]
E26B;	Perform [SIN]
E2B4;	Perform [TAN]
E30E;	Perform [ATN]
E37B;	Warm restart
E394;	Initialize
E3A2;	CHRGET for zero page
E3BF;	Initialize Basic
E447;	Vectors for $300
E453;	Initialize vectors
E45F;	Power-up message
E500;	Get I/O address
E505;	Get screen size
E50A;	Put/get row/column
E518;	InitializeI/O
E544;	Clear screen
E566;	Home cursor
E56C;	Set screen pointers
E5A0;	Set I/O defaults
E5B4;	Input from keyboard
E632;	Input from screen
E684;	Quote test
E691;	Setup screen print
E6B6;	Advance cursor
E6ED;	Retreat cursor
E701;	Back into previous line
E716;	Output to screen
E87C;	Go to next line
E891;	Perform <return>
E8A1;	Check line decrement
E8B3;	Check line increment
E8CB;	Set color code
E8DA;	Color code table
E8EA;	Scroll screen
E965;	Open space on screen
E9C8;	Move a screen line
E9E0;	Synchronize color transfer
E9F0;	Set start-of-line
E9FF;	Clear screen line
EA13;	Print to screen
EA24;	Synchronize color pointer
EA31;	Interrupt - clock etc
EA87;	Read keyboard
EB79;	Keyboard select vectors
EB81;	Keyboard 1 - unshifted
EBC2;	Keyboard 2 - shifted
EC03;	Keyboard 3 - 'comm'
EC44;	Graphics/text contrl
EC4F;	Set graphics/text mode
EC78;	Keyboard 4
ECB9;	Video chip setup
ECE7;	Shift/run equivalent
ECF0;	Screen In address low
ED09;	Send 'talk'
ED0C;	Send 'listen'
ED40;	Send to serial bus
EDB2;	Serial timeout
EDB9;	Send listen SA
EDBE;	Clear ATN
EDC7;	Send talk SA
EDCC;	Wait for clock
EDDD;	Send serial deferred
EDEF;	Send 'untalk'

EDFE;	Send 'unlisten'
EE13;	Receive from serial bus
EE85;	Serial clock on
EE8E;	Serial clock off
EE97;	Serial output '1'
EEA0;	Serial output '0'
EEA9;	Get serial in & clock
EEB3;	Delay 1 ms
EEBB;	RS-232 send
EF06;	Send new RS-232 byte
EF2E;	No-DSR error
EF31;	No-CTS error
EF3B;	Disable timer
EF4A;	Compute bit count
EF59;	RS232 receive
EF7E;	Setup to receive
EFC5;	Receive parity error
EFCA;	Receive overflow
EFCD;	Receive break
EFD0;	Framing error
EFE1;	Submit to RS232
F00D;	No-DSR error
F017;	Send to RS232 buffer
F04D;	Input from RS232
F086;	Get from RS232
F0A4;	Check serial bus idle
F0BD;	Messages
F12B;	Print if direct
F13E;	Get..
F14E;	..from RS232
F157;	Input
F199;	Get.. tape/serial/rs232
F1CA;	Output..
F1DD;	..to tape
F20E;	Set input device
F250;	Set output device
F291;	Close file
F30F;	Find file
F31F;	Set file values
F32F;	Abort all files
F333;	Restore default I/O
F34A;	Do file open
F3D5;	Send SA
F409;	Open RS232
F49E;	Load program
F5AF;	'searching'
F5C1;	Print filename
F5D2;	'loading/verifying'
F5DD;	Save program
F68F;	Print 'saving'
F69B;	Bump clock
F6BC;	Log PIA key reading
F6DD;	Get time
F6E4;	Set time
F6ED;	Check stop key
F6FB;	Output error messages
F72D;	Find any tape headr
F76A;	Write tape header
F7D0;	Get buffer address
F7D7;	Set buffer start/end pointers
F7EA;	Find specific header
F80D;	Bump tape pointer
F817;	'press play..'
F82E;	Check tape status
F838;	'press record..'
F841;	Initiate tape read
F864;	Initiate tape write
F875;	Common tape code
F8D0;	Check tape stop
F8E2;	Set read timing
F92C;	Read tape bits
FA60;	Store tape chars
FB8E;	Reset pointer
FB97;	New character setup
FBA6;	Send transition to tape
FBC8;	Write data to tape
FBCD;	IRQ entry point
FC57;	Write tape leader
FC93;	Restore normal IRQ
FCB8;	Set IRQ vector
FCCA;	Kill tape motor
FCD1;	Check r/w pointer
FCDB;	Bump r/w pointer
FCE2;	Power reset entry
FD02;	Check 8-rom
FD10;	8-rom mask
FD15;	Kernal reset
FD1A;	Kernal move
FD30;	Vectors
FD50;	Initialize system constnts
FD9B;	IRQ vectors
FDA3;	Initialize I/O
FDDD;	Enable timer
FDF9;	Save filename data
FE00;	Save file details
FE07;	Get status
FE18;	Flag status
FE1C;	Set status
FE21;	Set timeout
FE25;	Read/set top of memory
FE27;	Read top of memory
FE2D;	Set top of memory
FE34;	Read/set bottom of memory
FE43;	NMI entry
FE66;	Warm start
FEB6;	Reset IRQ & exit
FEBC;	Interrupt exit
FEC2;	RS-232 timing table
FED6;	NMI RS-232 in
FF07;	NMI RS-232 out
FF43;	Fake IRQ
FF48;	IRQ entry
FF81;	Jumbo jump table
FFFA;	Hardware vectors

Processor I/O Port (6510)

$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR 0
$0001			Tape Motor	Tape Sense	Tape Write	D-ROM Switch	EF RAM Switch	AB RAM Switch	PR 1

SID (6581)

Voice 1	Voice 2	Voice 3	Register		Voice 1	Voice 2	Voice 3
$D400	$D407	$D40E	Frequency	L	54272	54279	54286
$D401	$D408	$D40F	Frequency	H	54273	54280	54287
$D402	$D409	$D410	Pulse Width	L	54274	54281	54288
$D403	$D40A	$D411	0 0 0 0 / Pulse Width	H	54275	54282	54289
$D404	$D40B	$D412	Voice Type: NSE / PUL / SAW / TRI	Key	54276	54283	54290
$D405	$D40C	$D413	Attack Time 2ms – 8ms / Decay Time 6ms – 24 sec		54277	54284	54291
$D406	$D40D	$D414	Sustain Level / Release Time 6ms 24 sec		54278	54285	54292

Voices (write only)

$D415	0 0 0 0 0 / Filter Frequency	L	54293
$D416	Filter Frequency	H	54294
$D417	Resonance / Filter Voices: Ext / V3 / V2 / V1		54295
$D418	V3 off / Passband: HI / BP / LO / Master Volume		54296

Filter & Volume (write only)

$D419	Paddle X (A/D #1)	54297
$D41A	Paddle Y (A/D #2)	54298
$D41B	Noise 3 (random)	54299
$D41C	Envelope 3	54300

Sense (read only)

Note: Special Voice Features
(TEST, RING MOD, SYNC)
are omitted from the above diagram.

CIA 1 (IRQ) (6526)

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register	Decimal
$DC00	Paddle Sel A	Paddle Sel B		Joystick 0 Fire	Right	Left	Down	Up	PRA	56320
	Keyboard Row Select (inverted)									
$DC01				Joystick 1 Fire	Right	Left	Down	Up	PRB	56321
	Keyboard Column Read									
$DC02	$FF – All Output								DDRA	56322
$DC03	$00 – All Input								DDRB	56323
$DC04	Timer A								TAL	56324
$DC05									TAH	56325
$DC06	Timer B								TBL	56326
$DC07									TBH	56327
~										
$DC0D				Tape Input			Timer Interrupt B	A	ICR	56333
$DC0E					One Shot	Out Mode	Time PB6 Out	Timer A Start	CRA	56334
$DC0F					One Shot	Out Mode	Time PB7 Out	Timer B Start	CRB	56335

CIA 2 (NMI) (6526)

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register	Decimal
$DD00	Serial IN	Clock IN	Serial OUT	Clock OUT	ATN OUT	RS-232 OUT	VIC II addr 15	VIC II addr 14	PRA	56576
$DD01	DSR IN	CTS IN		DCD* IN	RI* IN	DTR OUT	RTS OUT	RS-232 IN	PRB	56577
$DD02	$3F – Serial								DDRA	56578
$DD03	$00 – P.U.P. All Input or $06 – RS-232								DDRB	56579
$DD04	Timer A								TAL	56580
$DD05									TAH	56581
$DD06	Timer B								TBL	56582
$DD07									TBH	56583
~										
$DD0D				RS-232 IN			Timer Interrupt B	A	ICR	56589
$DD0E								Timer A Start	CRA	56590
$DD0F								Timer B Start	CRB	56591

* Connected but not used by O.S.

Appendix B

Key values

Location 197 decimal $C6 hex contains the value for the current key depression. This is a table of the codes stored and their Ascii equivalents.

Key	**Value**	**Key**	**Value**
left arrow	57	INST DEL	0
1	56	up arrow	54
2	59	*	49
3	8	@	46
4	11	P	41
5	16	O	38
6	19	I	33
7	24	U	30
8	27	Y	25
9	32	T	22
0	35	R	17
+	40	E	14
−	43	W	9
pound sign	48	Q	62
CLR HOME	51	RUN/STOP	63

Key	Value	Key	Value
A	10	Cursor up/down	7
S	13	/	55
D	18	.	44
F	21	,	47
G	26	M	36
H	29	N	39
J	34	B	28
K	37	V	31
L	42	C	20
:	45	X	23
;	50	Z	12
=	53	Space	60
Return	1	F	14
Cursor right/left	2	F	35
		F	56
		F	73

Appendix C

Basic tokens

There are tokens for most of the commands and statements. They allow easier entry and longer lines. Overleaf is a complete list of the tokens. They will take a while to memorise, but the effort is generally worth the result.

Abbreviations for Keywords

Command	Abbreviation	Looks like this on screen
ABS	A SHIFT B	A ▕
AND	A SHIFT	A ╱
ASC	A SHIFT S	A ♥
ATN	A SHIFT T	A ▏
CHR$	C SHIFT H	C ▕
CLOSE	CL SHIFT O	CL
CLR	C SHIFT L	C
CMD	C SHIFT M	C ╲
CONT	C SHIFT O	C
DATA	D SHIFT A	D ♠
DEF	D SHIFT E	D ▔
DIM	D SHIFT I	D ╮
END	E SHIFT N	E ╱
EXP	E SHIFT X	E ♣
FOR	F SHIFT O	F
FRE	F SHIFT R	F ▁
GET	G SHIFT E	G ▔
GOSUB	GO SHIFT S	GO ♥
GOTO	SHIFT O	G
INPUT#	I SHIFT N	I ╱
LET	L SHIFT E	L ▔
LEFT$	LE SHIFT F	LE ▁
LIST	L SHIFT I	L ╮
LOAD	L SHIFT O	L
MID$	M SHIFT I	M ╮
NEXT	N SHIFT E	N ▔
NOT	N SHIFT O	N
OPEN	O SHIFT P	O
PEEK	P SHIFT E	P ▔
POKE	P SHIFT O	P
PRINT	?	?
PRINT#	P SHIFT R	P ▁
READ	R SHIFT E	R ▔
RESTORE	RE SHIFT S	RE ♥
RETURN	RE SHIFT T	RE ▏
RIGHT$	R SHIFT I	R ╮
RND	R SHIFT N	R ╱
RUN	R SHIFT	R ╭
SAVE	S SHIFT A	S ♠
SGN	S SHIFT G	S ▏
SIN	S SHIFT I	S ╮
SPC(	S SHIFT P	S
SQR	S SHIFT Q	S ●
STEP	ST SHIFT E	ST ▔
STOP	S SHIFT T	S ▏
STR$	ST SHIFT R	ST ▁
SYS	S SHIFT Y	S ▕
TAB	T SHIFT A	T ♠
THEN	T SHIFT H	T ▕
USR	U SHIFT S	U ♥
VAL	V SHIFT A	V ♠
VERIFY	V SHIFT E	V
WAIT	W SHIFT A	W ♠

Appendix D

Machine code instruction set

The following notation applies to this summary:

A	Accumulator
X, Y	Index registers
M	Memory
P	Processor status register
S	Stack Pointer
√	Change
–	No change
+	Add
∧	Logical AND
−	Subtract
V	Logical Exclusive-OR
→, ←	Transfer to
∨	Logical (inclusive) OR
PC	Program counter
PCH	Program counter high
PCL	Program counter low
#dd	8-bit immediate data value (2 hexadecimal digits)
aa	8-bit zero page address (2 hexadecimal digits)
aaaa	16-bit absolute address (4 hexadecimal digits)
↑	Transfer from stack (Pull)
↓	Transfer onto stack (Push)

ADC

Add to Accumulator with Carry

Operation: $A + M + C \rightarrow A, C$

N	Z	C	I	D	V
√	√	√	—	—	√

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	ADC #dd	69	2	2
Zero Page	ADC aa	65	2	3
Zero Page, X	ADC aa,X	75	2	4
Absolute	ADC aaaa	6D	3	4
Absolute, X	ADC aaaa,X	7D	3	4*
Absolute, Y	ADC aaaa,Y	79	3	4*
(Indirect, X)	ADC (aa,X)	61	2	6
(Indirect), Y	ADC (aa),Y	71	2	5*

*Add 1 if page boundary is crossed.

AND

AND Memory with Accumulator

Logical AND to the accumulator

Operation: $A \wedge M \rightarrow A$

N	Z	C	I	D	V
√	√	—	—	—	—

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	AND #dd	29	2	2
Zero Page	AND aa	25	2	3
Zero Page, X	AND aa,X	35	2	4
Absolute	AND aaaa	2D	3	4
Absolute, X	AND aaaa,X	3D	3	4*
Absolute, Y	AND aaaa,Y	39	3	4*
(Indirect, X)	AND (aa,X)	21	2	6
(Indirect), Y	AND (aa),Y	31	2	5*

*Add 1 if page boundary is crossed.

ASL

Accumulator Shift Left

Operation: C ← [7|6|5|4|3|2|1|0] ← 0

N Z C I D V
✓ ✓ ✓ — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ASL A	0A	1	2
Zero Page	ASL aa	06	2	5
Zero Page, X	ASL aa,X	16	2	6
Absolute	ASL aaaa	0E	3	6
Absolute, X	ASL aaaa,X	1E	3	7

BCC

Branch on Carry Clear

Operation: Branch on C = 0

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCC aa	90	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to different page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BCC aaaa), and convert it to a relative address.

BCS

Branch on Carry Set

Operation: Branch on C = 1

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCS aa	B0	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to next page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BCS aaaa), and convert it to a relative address.

BEQ

Branch on Result Equal to Zero

Operation: Branch on $Z = 1$

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BEQ aa	F0	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to next page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BEQ aaaa), and convert it to a relative address.

BIT

Test Bits in Memory with Accumulator

Operation: A M, $M_7 \rightarrow N$, $M_6 \rightarrow V$
Bit 6 and 7 are transferred to the Status Register. If the result of A M is zero then $Z = 1$, otherwise $Z = 0$

N Z C I D V
M_7 ✓ — — — M_6

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	BIT aa	24	2	3
Absolute	BIT aaaa	2C	3	4

BMI

Branch on Result Minus

Operation: Branch on $N = 1$

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BMI aa	30	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to different page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BMI aaaa), and convert it to a relative address.

BNE

Branch on Result Not Equal to Zero

Operation: Branch on Z = 0

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BNE aa	D0	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to different page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BNE aaaa), and convert it to a relative address.

BPL

Branch on Result Plus

Operation: Branch on N = 0

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BPL aa	10	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to different page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BPL aaaa), and convert it to a relative address.

BRK

Force Break

Operation: Forced Interrupt PC + 2 ↓ P ↓

B N Z C I D V
1 — — — 1 — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	BRK	00	1	7

BVC

Branch on Overflow Clear

Operation: Branch on $V = 0$

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVC aa	50	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to different page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BVC aaaa), and convert it to a relative address.

BVS

Branch on Overflow Set

Operation: Branch on $V = 1$

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVS aa	70	2	2*

*Add 1 if branch occurs to same page.
Add 2 if branch occurs to different page.
Note: AIM 65 will accept an absolute address as the operand (instruction format BVS aaaa), and convert it to a relative address.

CLC

Clear Carry Flag

Operation: $0 \rightarrow C$

N Z C I D V
— — 0 — — —

Addressing Mode	Assembly Language Form	OP CODE	No Bytes	No. Cycles
Implied	CLC	18	1	2

CLD

Clear Decimal Mode

Operation: $0 \rightarrow D$

N Z C I D V
– – – – 0 –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLD	D8	1	2

CLI

Clear Interrupt Disable Bit

Operation: $0 \rightarrow I$

N Z C I D V
– – – 0 – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLI	58	1	2

CLV

Clear Overflow Flag

Operation: $0 \rightarrow V$

N Z C I D V
– – – – – 0

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

CMP

Compare Memory and Accumulator

Operation: A — M

N Z C I D V
√ √ √ — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CMP #dd	C9	2	2
Zero Page	CMP aa	C5	2	3
Zero Page, X	CMP aa,X	D5	2	4
Absolute	CMP aaaa	CD	3	4
Absolute, X	CMP aaaa,X	DD	3	4*
Absolute, Y	CMP aaaa,Y	D9	3	4*
(Indirect, X)	CMP (aa,X)	C1	2	6
(Indirect), Y	CMP (aa),Y	D1	2	5*

*Add 1 if page boundary is crossed.

CPX

Compare Memory and Index X

Operation: X — M

N Z C I D V
√ √ √ — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPX #dd	E0	2	2
Zero Page	CPX aa	E4	2	3
Absolute	CPX aaaa	EC	3	4

CPY

Compare Memory and Index Y

Operation: Y — M

N Z C I D V
√ √ √ — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	CPY #dd	C0	2	2
Zero Page	CPY aa	C4	2	3
Absolute	CPY aaaa	CC	3	4

DEC

Decrement Memory by One

Operation: M − 1 → M

N Z C I D V
√ √ — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	DEC aa	C6	2	5
Zero Page, X	DEC aa,X	D6	2	6
Absolute	DEC aaaa	CE	3	6
Absolute, X	DEC aaaa,X	DE	3	7

DEX

Decrement Index X by One

Operation: X − 1 → X

N Z C I D V
√ √ — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

DEY

Decrement Index Y by One

Operation: Y − 1 → Y

N Z C I D V
√ √ — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEY	88	1	2

EOR

Exclusive-OR Memory with Accumulator

Operation: A V M → A

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	EOR #dd	49	2	2
Zero Page	EOR aa	45	2	3
Zero Page, X	EOR aa,X	55	2	4
Absolute	EOR aaaa	4D	3	4
Absolute, X	EOR aaaa,X	5D	3	4*
Absolute, Y	EOR aaaa,Y	59	3	4*
(Indirect, X)	EOR (aa,X)	41	2	6
(Indirect), Y	EOR (aa),Y	51	2	5*

*Add 1 if page boundary is crossed.

INC

Increment Memory by One

Operation: M + 1 → M

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	INC aa	E6	2	5
Zero Page, X	INC aa,X	F6	2	6
Absolute	INC aaaa	EE	3	6
Absolute, X	INC aaaa,X	FE	3	7

INX

Increment Index X by One

Operation: X + 1 → X

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INX	E8	1	2

INY

Increment Index Y by One

Operation: $Y + 1 \rightarrow Y$

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form	OP Code	No. Bytes	No. Cycles
Implied	INY	C8	1	2

JMP

Jump

Operation: $(PC + 1) \rightarrow PCL$
$(PC + 2) \rightarrow PCH$

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JMP aaaa	4C	3	3
Indirect	JMP (aaaa)	6C	3	5

JSR

Jump to Subroutine

Operation: $PC + 2 \downarrow, (PC + 1) \rightarrow PCL$
$(PC + 2) \rightarrow PCH$

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JSR aaaa	20	3	6

LDA

Load Accumulator with Memory

Operation: M → A

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDA #dd	A9	2	2
Zero Page	LDA aa	A5	2	3
Zero Page, X	LDA aa,X	B5	2	4
Absolute	LDA aaaa	AD	3	4
Absolute, X	LDA aaaa,X	BD	3	4*
Absolute, Y	LDA aaaa,Y	B9	3	4*
(Indirect, X)	LDA (aa,X)	A1	2	6
(Indirect), Y	LDA (aa),Y	B1	2	5*

*Add 1 if page boundary is crossed.

LDX

Load Index X with Memory

Operation: M → X

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDX #dd	A2	2	2
Zero Page	LDX aa	A6	2	3
Zero Page, Y	LDX aa,Y	B6	2	4
Absolute	LDX aaaa	AE	3	4
Absolute, Y	LDX aaaa,Y	BE	3	4*

*Add 1 when page boundary is crossed.

LDY

Load Index Y with Memory

Operation: M → Y

N	Z	C	I	D	V
✓	✓	–	–	–	–

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Immediate	LDY #dd	A0	2	2
Zero Page	LDY aa	A4	2	3
Zero Page, X	LDY aea,X	B4	2	4
Absolute	LDY aaaa	AC	3	4
Absolute, X	LDY aaaa,X	BC	3	4*

*Add 1 when page boundary is crossed.

LSR

Local Shift Right

Operation: 0 → [7 | 6 | 5 | 4 | 3 | 2 | 1 | 0] → C

N	Z	C	I	D	V
0	✓	✓	–	–	–

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	LSR A	4A	1	2
Zero Page	LSR aa	46	2	5
Zero Page, X	LSR aa,X	56	2	6
Absolute	LSR aaaa	4E	3	6
Absolute, X	LSR aaaa,X	5E	3	7

NOP

No Operation

Operation: No Operation (2 cycles)

N	Z	C	I	D	V
–	–	–	–	–	–

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

ORA

OR Memory with Accumulator

Operation: A V M → A

N Z C I D V
√ √ – – – –

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	ORA	#dd	09	2	2
Zero Page	ORA	aa	05	2	3
Zero Page, X	ORA	aa,X	15	2	4
Absolute	ORA	aaaa	0D	3	4
Absolute, X	ORA	aaaa,X	1D	3	4*
Absolute, Y	ORA	aaaa,Y	19	3	4*
(Indirect, X)	ORA	(aa,X)	01	2	6
(Indirect), Y	ORA	(aa),Y	11	2	5*

*Add 1 on page crossing.

PHA

Push Accumulator on Stack

Operation: A ↓

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHA	48	1	3

PHP

Push Processor Status on Stack

Operation: P↓

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHP	08	1	3

PLA

Pull Accumulator from Stack

Operation: A ↑

N Z C I D V
√ √ — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLA	68	1	4

PLP

Pull Processor Status from Stack

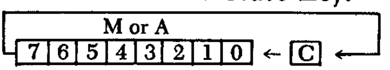
Operation: P ↑

N Z C I D V
From Stack

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLP	28	1	4

ROL

Rotate Left

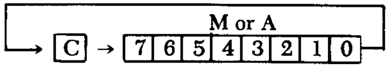
Operation: 

N Z C I D V
√ √ √ — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ROL A	2A	1	2
Zero Page	ROL aa	26	2	5
Zero Page, X	ROL aa,X	36	2	6
Absolute	ROL aaaa	2E	3	6
Absolute, X	ROL aaaa,X	3E	3	7

ROR

Rotate Right

Operation: 

N Z C I D V
✓ ✓ ✓ – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Accumulator	ROR A	6A	1	2
Zero Page	ROR aa	66	2	5
Zero Page, X	ROR aa,X	76	2	6
Absolute	ROR aaaa	6E	3	6
Absolute, X	ROR aaaa,X	7E	3	7

RTI

Return from Interrupt

Operation: P↑ PC↑

N Z C I D V
From Stack

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTI	40	1	6

RTS

Return from Subroutine

Operation: PC↑, PC + 1 → PC

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTS	60	1	6

SBC

Subtract from Accumulator with Carry

Operation: $A - M - \overline{C} \rightarrow A$
Note: $\overline{C}$ = Borrow

N	Z	C	I	D	V
√	√	√	—	—	√

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	SBC	#dd	E9	2	2
Zero Page	SBC	aa	E5	2	3
Zero Page, X	SBC	aa,X	F5	2	4
Absolute	SBC	aaaa	ED	3	4
Absolute, X	SBC	aaaa,X	FD	3	4*
Absolute, Y	SBC	aaaa,Y	F9	3	4*
(Indirect, X)	SBC	(aa,X)	E1	2	6
(Indirect), Y	SBC	(aa),Y	F1	2	5*

*Add 1 when page boundary is crossed.

SEC

Set Carry Flag

Operation: $1 \rightarrow C$

N	Z	C	I	D	V
—	—	1	—	—	—

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEC	38	1	2

SED

Set Decimal Mode

Operation: $1 \rightarrow D$

N	Z	C	I	D	V
—	—	—	—	1	—

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SED	F8	1	2

SEI

Set Interrupt Disable Status

Operation: $1 \rightarrow I$

N Z C I D V
– – – 1 – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEI	78	1	2

STA

Store Accumulator in Memory

Operation: $A \rightarrow M$

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STA aa	85	2	3
Zero Page, X	STA aa,X	95	2	4
Absolute	STA aaaa	8D	3	4
Absolute, X	STA aaaa,X	9D	3	5
Absolute, Y	STA aaaa,Y	99	3	5
(Indirect, X)	STA (aa,X)	81	2	6
(Indirect), Y	STA (aa),Y	91	2	6

STX

Store Index X in Memory

Operation: $X \rightarrow M$

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STX aa	86	2	3
Zero Page, Y	STX aa,Y	96	2	4
Absolute	STX aaaa	8E	3	4

STY

Store Index Y in Memory

Operation: Y → M

N Z C I D V
— — — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Zero Page	STY aa	84	2	3
Zero Page, X	STY aa,X	94	2	4
Absolute	STY aaaa	8C	3	4

TAX

Transfer Accumulator to Index X

Operation: A → X

N Z C I D V
√ √ — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAX	AA	1	2

TAY

Transfer Accumulator to Index Y

Operation: A → Y

N Z C I D V
√ √ — — — —

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAY	A8	1	2

TSX

Transfer Stack Pointer to Index X

Operation: S → X

N Z C I D V
✓ ✓ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TSX	BA	1	2

TXA

Transfer Index X to Accumulator

Operation: X → A

N Z C I D V
✓ ✓ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXA	8A	1	2

TXS

Transfer Index X to Stack Pointer

Operation: X → S

N Z C I D V
– – – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXS	9A	1	2

TYA

Transfer Index Y to Accumulator

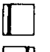
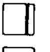
Operation: Y → A

N Z C I D V
✓ ✓ – – – –

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TYA	98	1	2

Appendix E

Screen Display Codes

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	[		27	6		54
A	a	1	£		28	7		55
B	b	2	]		29	8		56
C	c	3	↑		30	9		57
D	d	4	←		31	:		58
E	e	5	SPACE		32	;		59
F	f	6	!		33	<		60
G	g	7	"		34	=		61
H	h	8	#		35	>		62
I	i	9	$		36	?		63
J	j	10	%		37			64
K	k	11	&		38		A	65
L	l	12	'		39		B	66
M	m	13	(		40		C	67
N	n	14	)		41		D	68
O	o	15	*		42		E	69
P	p	16	+		43		F	70
Q	q	17	,		44		G	71
R	r	18	–		45		H	72
S	s	19	.		46		I	73
T	t	20	/		47		J	74
U	u	21	0		48		K	75
V	v	22	1		49		L	76
W	w	23	2		50		M	77
X	x	24	3		51		N	78
Y	y	25	4		52		O	79
Z	z	26	5		53		P	80

SET 1	SET 2	POKE
	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125
		126
		127

Codes from 128-255 are reversed images of codes 0-127.

Appendix F

Ascii values

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0	CRSR ↓	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
DISABLES SHIFT C=	8		25	*	42	;	59
ENABLES SHIFT C=	9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	−	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
SWITCH TO LOWER CASE	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
D	68	♠	97		126		155
E	69		98		127	PUR	156
F	70		99		128	CRSR	157
G	71		100		129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84	●	113	SWITCH TO UPPER CASE	142		171
U	85		114		143		172
V	86	♥	115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90	○	119	INST DEL	148		177
[	91	♣	120		149		178
£	92		121		150		179
]	93	♦	122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	184		186		188		190
	185		187		189		191

CODES	**192-223**	**SAME AS**	**96-127**
CODES	**224-254**	**SAME AS**	**160-190**
CODE	**255**	**SAME AS**	**126**

Appendix G

Basic error messages

BAD DATA String data was received from an open file, but the program was expecting numeric data.

BAD SUBSCRIPT The program was trying to reference an element of an array whose number is outside of the range specified in the DIM statement.

CAN'T CONTINUE The CONT command will not work, either because the program was never RUN, there has been an error, or a line has been edited.

DEVICE NOT PRESENT The required I/O device was not available for an OPEN, CLOSE, CMD, PRINT#, INPUT#, or GET#.

DIVISION BY ZERO Division by zero is a mathematical oddity and not allowed.

EXTRA IGNORED Too many items of data were typed in response to an INPUT statement. Only the first few items were accepted.

FILE NOT FOUND If you were looking for a file on tape, and END-OF-TAPE marker was found. If you were looking on disk, no file with that name exists.

FILE NOT OPEN The file specified in a CLOSE, CMD, PRINT#, INPUT#, or GET#, must first be OPENed.

FILE OPEN An attempt was made to open a file using the number of an already open file.

FORMULA TOO COMPLEX The string expression being evaluated should be split into at least two parts for the system to work with.

ILLEGAL DIRECT The INPUT statement can only be used within a program, and not in direct mode.

ILLEGAL QUANTITY A number used as the argument of a function or statement is out of the allowable range.

LOAD There is a problem with the program on tape.

NEXT WITHOUT FOR This is caused by either incorrectly nesting loops or having a variable name in a NEXT statement that doesn't correspond with one in a FOR statement.

NOT INPUT FILE An attempt was made to INPUT or GET data from a file which was specified to be for output only.

NOT OUTPUT FILE An attempt was made to PRINT data to a file which was specified as input only.

OUT OF DATA A READ statement was executed but there is no data left unREAD in a DATA statement.

OUT OF MEMORY There is no more RAM available for program or variables. This may also occur when too many FOR loops have been nested, or when there are too many GOSUBs in effect.

OVERFLOW The result of a computation is larger than the largest number allowed, which is 1.70141884E+38.

REDIM'D ARRAY An array may only be DIMensioned once. If an array variable is used before that array is DIM'd, an automatic DIM operation is performed on that array setting the number of elements to ten, and any subsequent DIMs will cause this error.

REDO FROM START Character data was typed in during an INPUT statement when numeric data was expected. Just re-type the entry so that it is correct, and the program will continue by itself.

RETURN WITHOUT GOSUB A RETURN statement was encountered, and no GOSUB command has been issued.

STRING TOO LONG A string can contain up to 255 characters.

?SYNTAX ERROR A statement is unrecognizable by the Commodore 64. A missing or extra parenthesis, misspelled keywords, etc.

TYPE MISMATCH This error occurs when a number is used in place of a string, or vice-versa.

UNDEF'D FUNCTION A user defined function was referenced, but it has never been defined using the DEF FN statement.

UNDEF'D STATEMENT An attempt was made to GOTO or GOSUB or RUN a line number that doesn't exist.

VERIFY The program on tape or disk does not match the program currently in memory.

Further Reading

Apart from advising you read anything written by me, I include here a list of books and magazines that will certainly be enjoyable and informative.

Books

Using the 64 by Peter Gerrard, published by Duckworth (£9.95). Good for novice or expert; contains all the necessary information for the 64.

Illustrating Basic by Donald Alcock, published by C.U.P. (£1.99). Good for beginners to Basic.

Advanced 6502 Programming by Rodnay Zaks, published by Sybex (£10.25). Invaluable, as are all of Zaks' books.

6502 Machine Code for Humans by Alan Toothill and David Barrow, published by Granada (£7.95). If you are just starting on machine code, this is as good a place as any to start.

6502 Assembly Language Programming by L.A. Leventhal, published by McGraw Hill, Berkeley, California. A must for everyone using machine code.

Commodore 64 Programmer's Reference Guide, published by Commodore (£14.95).

The Complete Commodore 64 ROM Disassembly by Peter Gerrard and Kevin Bergin, published by Duckworth (£5.95).

Programming the Pet/CBM by Raeto West, published by Level (£14.90).

Commodore 64 Exposed by Bruce Bayley, published by Melbourne House (£6.95).

Advanced 6502 Programming by Rodnay Zaks (£10.25).

Magazines

Commodore Horizons. Very informative about hardware and software, also contains many fair user programs.

Commodore User. The best specialist magazine for Commodore owners. The contributors include some of the best Commodore programmers.

Personal Computer News. The best weekly by far, this contains a great deal of information about hardware and software, and is particulary good for games.

Compute. The best magazine for Commodore owners, although it is not solely a Commodore magazine. This is an American publication, but some shops do stock it.

Personal Computer World. A journal for those wishing to keep up with the whole range of hardware and software, this also has good columns for beginners and on machine code.

Micro Adventurer. This magazine is dedicated to adventure and strategy games. It includes reviews, readers' programs and an excellent help column.

Index