COMMODORE 64 MACHINE LANGUAGE TUTORIAL

Paul Blair, DipCE, MIE Aust.

A self tuition book on Machine Language Programming with the 65XX family of microprocessors. Suitable for use also with other Commodore, & Apple computers.

ISBN 0 9591417 0 7

COMMODORE 64 MACHINE LANGUAGE TUTORIAL

BY PAUL BLAIR

KiM BOOKS

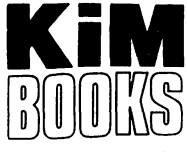
COPYRIGHT NOTICE

WARRANTY

First published 1984

ISBN 0 959 1417 0 7

Printed and assembled in Australia for KIM BOOKS (division of Mervyn Beamish Graphics Pty Ltd), by KANPRINT, Crows Nest, N.S.W.
Phototypeset by Mervyn Beamish Graphics Pty Ltd, Crows Nest, N.S.W.



Suite 102-105, 82 Alexander Street, Crows Nest, N.S.W.

COMMODORE 64
MACHINE LANGUAGE TUTORIAL

PREFACE

PAUL BLAIR has distilled his own starting experiences in learning machine language (ML) into a set of TUTORIALS to ease the transition from BASIC to ML. The layout of the book is designed to provide self tuition and guidance in the simple use of the principal 6500 microprocessor family ML instructions. A computerist will gain confidence to continue into the language.

To assist discussion about ML programming, the book contains a complete program listing for the Commodore 64 (C64). There is a similar program available for the CBM/PET 4000 series of computers. As the Apple computer is also based on the 6500 family, Apple users can also learn from it. The point of the program is illustrative, and study of it will quickly demonstrate many of the points made in the text.

My contact with Paul Blair is as editor of the Commodore Magazine. After our initial introduction by mail, I was soon to find that Paul had a rare gift of being a natural born teacher - a gift he uses very well in this book.

MERVYN BEAMISH
Editor Commodore Magazine

CONTENTS

TOPIC 9: **COMPARES** –
Three OPCODES for Branching on Compare, (CMP, CPX, CPY); Use of compare.

TOPIC 10: **FLAGS** –
FLAG setting; Why flags are used.

TOPIC 11: **EASY ONES** –
Four easily understood OPCODES (PHA, PLA, PHP, PLP); BANKing values.

TOPIC 12: **LOGICAL OPERATIONS** –
BOOLEAN functions; Four more OPCODES, (AND, BIT, EOR, ORA).

TOPIC 13: **ADDITION AND SUBTRACTION** –
Altering values using Addition and Subtraction.

TOPIC 14: **SHIFTS AND THINGS** –
BIT rotation; SHIFTing values; Final four OPCODES (ASL, LSR, ROL, ROR).

TOPIC 15: **OTHER ADDRESSING MODES** –
Storing numbers greater than 255; Non OPCODE addresses.

TOPIC 16: **TO FINISH OFF** –
Comprehensive review.

TOPIC 17: **EDITORS AND ASSEMBLERS** –
Assembler packages and their uses; BINARY files.

CONCLUSION

APPENDIX A: Naming conventions; Notes on using CBM Assembler Suite package; Assembler procedure flowchart.

APPENDIX B: A sample graphics program.

INTRODUCTION

My first toddling steps in machine language were disasters. Those curious listings that kept cropping up in the computer press were unintelligible, the magazines assumed I knew it all, and the books in the shops, mainly of the reference variety, gave little or no help to the novice. The rest is history now, but the further I went, the more conscious I became of the need for a text for the rank amateur. What follows in this book is a distillation of my starting experiences, set down to ease your transition to machine language (ML) programming.

The layout of the book is designed to provide guidance in the simple use of the principal 6500 microprocessor family ML instructions. You should gain sufficient confidence to go on from there.

To assist discussion about ML programmng, the book contains a complete program for the Commodore 64 (C64). There is a similar program available for the CBM/PET 4000 series of computers. As the Apple computer is also based on the 6500 family, Apple users can also learn from it. The point of the program is illustrative, and study of it will quickly demonstrate many of the points made in the text.

Skill in ML programming can be achieved only by working at it - there is no silver bullet solution. Your level of competence will equal the effort you put into it.

PAUL BLAIR

AFTER BASIC — What Next

By now, most of you will have become competent in the use of BASIC – and now wish to widen your programming skill horizons. BASIC is a high level language, and its designers have provided a large number of helpful (and forgiving) features in it. You can edit and run most BASIC programs in a friendly environment – even if your program is full of errors, you will get to run some of it, fixing up errors as you go along.

Not so machine language. By writing programs directly in the computer's own language, not a language interpreted for it, YOU become the definer of program logic – and if that logic is not correct, you may so confuse the computer that you lose control of it. So, be warned, ML programming is demanding, exacting and often exasperating. But it is a great teacher, too. Your logic processes will be strengthened, your temper becomes more elastic and your self satisfaction will grow with each successful excursion.

Why ML? What can you manage with ML that can't be done in BASIC? The answer to the latter question is – nothing! In fact, BASIC provides many routines that you would have to provide yourself in ML. BASIC will multiply 2 numbers for you by saying

'PRINT 2*3'

whereas ML requires you to design a multiply routine.

The advantage of ML is principally speed. By removing the intermediate interpretation step of BASIC, your computer can run hundreds, if not thousands of times faster. For tasks like sorting, searching and some graphics, ML is the fastest (and probably the only satisfactory) way of programming. All tasks that require a large number of repetitions of whatever type – that is where ML will win hands down. You will also find that judicious use of ML permits you to use larger or more complex programs.

You need a few items, so that you can experiment for yourself. The first is easy – get hold of a small hexadecimal to decimal conversion chart, one that can sit just above the keys on the keyboard (or be taped there). The other is an extended monitor program, like 64MONITOR or SUPERMON. This will make learning easier.

There are some things that need to be covered before going too much further, so let's do them now.

TOPIC 1 — Memory Organisation

We are going to concentrate on the '65XX' family of microprosessors. The original Commodore computers (CBM/PET) and the VIC used the 6502 (as did the Apple computer). The Commodore 64 uses the newer 6510 microprocessor. For our purposes, the 6502 and 6510 are the same, and '65XX' as used here will refer to both devices.

Central to an understanding of ML is some idea of computer design, particularly how information is stored.

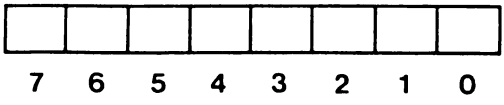
The original storage concept has not varied from the earliest days of data processing, when small magnets were 'core' storage. If the magnetic field went one way, the magnet was 'on', and the other way was 'off'. This idea persists, only the storage medium has changed.

The modern computer relies on the 'on/off' principle. Each 'bit' (BInary digiT) or storage location can be 'on' or 'off'. By convention, 'on' is as if the bit represented one, and 'off' as if the bit represented zero.

For ease of processing, bits are arranged in groups, named 'bytes'. In the 65XX, eight bits makes a byte – so Commodore computers are referred to as '8 bit' machines. Later or larger computers use 16 or 32 bits in a byte.

So let us represent a byte like this. Each small box represents a bit, and the large overall box represents a byte.

Figure 1



For convenience, each bit is given a 'position number', starting with 0 on the right running to 7 on the left (this is akin to decimal units, tens, hundreds and so on). But there is a reason, as we shall see.

How do we use this? As with decimal numbers, each bit (whether on or off) has a 'weight'. In fact, the bit position numbers represent these weights. Because each bit can only represent 2 conditions, on or off (or 1 and 0) the multipliers increase in powers of 2. In this way, the value of a byte is formed.

Using the bit position numbers, the individual bits either represent 2 raised to the power of the bit position, or zero. So Bit 0 can represent 1 or 0, bit 1 can mean 2 (2 to the power of 1) or 0. Bit 2 can mean 4 (2 to the power of 2) or 0, bit 3 can mean 8 or 0, and so on. Bit 7 can mean two to the power of 7 (128) or 0. So we have a range of number combinations in each byte, depending on which bits are on or off. If all possible bit combinations of 'on' and/or 'off' are considered, and the values of individual bits are added together, each byte can mean any number from 0 to 255 (try it yourself and see what you get). To put it more graphically, let's do some examples.

TASK 1

Draw bit patterns for the following numbers-

200 2 89 0 254 32 176

What numbers do these bit patterns represent-

11000101
00011011
10101010
01010101
11110000
00001001
10010010

Decimal numbers are one thing to remember, but who would want to remember bit pattern numbers? There had to be an easier and tidier way.

One byte can represent 0-255, i.e., 256 numbers. 256 is 16 x 16 ..... so every number in the range 0-255 could be represented by not more than two 'digits' if we could think of a way to represent a range of 0-16 with each 'digit'.

TOPIC 2 — Hexadecimal Notation

We have been conditioned to think in numbers of base 10 – i.e., numbers in 'groups' of tens. Any base could be used, but the world has used base ten for normal mathematics. Why base ten and not base 12 or 7 probably stems from the number of fingers we have! But computers don't have fingers, just bits!!

The 16 x 16 product led to thoughts of base 16 – but there weren't enough numerals to go around. To overcome this, the letters A to F were 'added' to the range of numerals. As if counting in 16's wasn't enough!

Instead of counting in tens, we count in 16's. Consider the range of decimal numbers from 0 to 15. Let us draw a table showing equivalent hexadecimal (usually shortened to 'hex') numbers:–

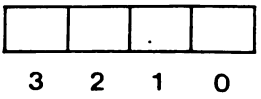
DEC 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
HEX 0 1 2 3 4 5 6 7 8 9 A B C D E F

As with decimal, we add another position ('tens' in decimal) when we go past the limit of the base number. This idea extends to hundreds, thousands, and so on in decimal, so why not hexadecimal? To count on from where we left off–

DEC 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
HEX 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

We can draw a comparison with our description of the bit 'weight' in a byte. Just as each position has a weight or indice, starting at 0 and incrementing by 1 for each step to the left, so hexadecimal numbers have weights for a similar pattern, but this time the 'weight' is in 16's. Figure 2 shows how.

Figure 2



Just as in our example with the bits in a byte, so hexadecimal applies 'weights' in relation to the position number. Think of the position number as representing the lowest value the position can hold – so position 0 starts from 0, position 1 starts from 16, position 2 starts from 256 (16 x 16) and position 3 from 4096 (16 x 16 x 16).

To convert from decimal to hexadecimal, divide the decimal number by 4096, 256 or 16, whichever is the next smallest number to the number to be converted. The whole part of the result converts to its hex equivalent. The process continues with the next smallest divisor, until the final remainder is less than 16. The concept is simple, and with practice you will become familiar with its use. Let us convert a number, - say 5127

5127 / 4096 = 1 and 1031 over = 1 as first hex digit
1031 / 256 = 4 and 7 over = 4 as next digit
7 / 16 = 0 and 7 over = 0 as next digit

So the final conversion of 5127 is 1407 hexadecimal.

The reverse applies – multiply progressively by 16, 256, and (if needed) 4096, adding together the individual results.

There are certain conventions used in hexadecimal notation. To differentiate between decimal and hexadecimal, hex numbers are prefixed with '$'. And hex numbers are usually written in groups of 2 numerals – $XX or $YYYY, not $Z or $ZZZ. This is not an absolute rule, but a tidy one.

TASK 2

Convert some numbers to hex–

10 25 49 8 100 47 19 66

Convert some hex numbers to decimal–

$4E $07 $5B $6A $49 $17 $25

You will come to grips with hexadecimal with practice. No book will do this for you, it's a matter for personal effort. Don't try to remember it all – hence the earlier suggestion for a small hex/decimal table to tape onto your computer keyboard.

TOPIC 3 — The 65XX Microprocessor

You probably have looked into the heart of the main processor chip, either accidently or deliberately. To do this, you would have accessed the monitor, built into 3000/4000/8000 Series Commodore machines, or available as a loadable program for the C64. The monitor is the window on the 65XX, the chip that is the heart of Commodore computers. If you are to succeed in ML programming, you need some basic information to start – and you will learn a lot more as we go along.

When you got to the monitor and peered in, you probably got a picture something like this

```
  PC   IRQ  SR AC XR YR SP
DC4D EA31 CC 30 05 FF F8
```

and (no doubt!) wondered what you had struck. You weren't the first, and won't be the last, let me assure you.

Storage in the 65XX can be considered as a number of memory locations, hooked together by a common line (the data bus) on which all information travels. In any program, the 65XX (by its 'architecture' or the way its brain is arranged) uses the 3 main temporary mailboxes AC, XR and YR to hold information, counting where it's at with the PC, checking its filing system with the SP, watching the instruments with the SR, and keeping overall control with the IRQ. Understand all that? Thought not.

Let's go through all that again, with some explanations.

PC stands for Program Counter. Using this counter, the computer stores enough information (2 bytes, each of 8 bits) to remind itself what to do when it finishes what it is doing now. In the example, $DC4D is the next port of call.

IRQ is short for Interrupt ReQuest. When a program is running, it is interrupted momentarily every so often (like 60 times a second) to check for outside activities, such as whether someone has typed something at the keyboard. In that case, whatever has been typed needs to be stored, then the program will continue. Don't worry too much about it for now, just be aware that it exists, and is very important.

SR stands for Status Register, which is one byte whose bits can be set or unset ('on' or 'off' if you like) to denote certain things – e.g., if bit 7 is 'on' (= 1) it could denote that the number you are working on is negative, while bit 7 'off' (= 0) could indicate a positive number. The bits in the SR are termed 'flags', which we will discuss in Topic 10.

The next 3 locations are also named registers, and are one byte each.

AC is the ACcumulator, probably the most important register of the three. For convenience, the name is abbreviated to '.A' During programs, most operations centre on .A

XR and YR are the X and Y Registers, usually referred to as .X and .Y Like .A, they are each one byte of storage, used for holding variables during run time.

SP stands for Stack Pointer. The stack is a 256 byte block of storage used for holding all sorts of information. It works like a pile of dinner plates – the last plate you put on top will be the first plate you retrieve. Information is 'pushed' onto the stack one byte at a time, and retrieved (or 'pulled') the same way, but in the reverse order to the order in which it was stored. Sometimes called LIFO, or Last In First Out, which sums it up pretty well. Of course, this is not to be confused with GIGO (Garbage in Garbage out) – a well known computer disease.

That completes our introductory sections. Reread these notes from time to time, to keep them fresh in your mind.

TOPIC 4
— Machine Language Programs

Commodore computers were designed around the 6502 (more recently the 6510) chip, and so enjoy the use of what is termed the '6502 instruction set'. The 6510 and the 6502 are identical in this respect. In brief, this means that it is possible to instruct the computer to perform certain functions, such as add, subtract, move data and so on, by using an 'instruction' that is included in the computers vocabulary. The instruction set has been carefully designed to provide for most functions that are needed, but (as ever, I suppose) there are a few not provided for directly, but are available by combining other directions. This is not a serious limitation.

The instruction set contains 56 core instructions, and a number of these have subsets of operating parameters, so that the 56 instructions have a total of 151 different operating techniques. That probably sounds daunting, but there is a pattern in the subsets that makes it much easier to follow. Some texts use the subsets to introduce the core instructions, but we will work the other way, using the instructions to introduce the subsets.

The 56 core instructions are defined by 3 character alphabetic codes, named operating codes – opcodes for short. In the sections that follow, we will examine each major opcode in turn.

THE OPCODE SET WITH TRANSLATIONS

×	ADC	ADd memory to .A with Carry
	AND	Logical AND of memory with .A
	ASL	Arithmetic Shift memory of .A Left one bit
–	BCC	Branch if the C (carry) flag is Clear (0)
–	BCS	Branch if the C flag is Set (1)
–	BEQ	Branch if the Z (zero) flag is EQual to 1
	BIT	Test memory bits
–	BMI	Branch if the N (negative or MInus) flag is 1
–	BNE	Branch if the Z flag is 0 (result Not Equal to 0)
–	BPL	Branch if the N (negative) flag is 0 (result PLus)

BRK	Force a BReaK (stop)
BVC	Branch if the V (oVerflow) flag is Clear (0)
BVS	Branch if the V (oVerflow) flag is Set (1)
CLC	CLear the C (carry) flag to 0
CLD	CLear the D (decimal) flag to 0
CLI	CLear the I (interrupt) flag to 0
CLV	CLear the V (overflow) flag to 0
CMP	CoMPare memory with .A
CPX	ComPare memory with .X
CPY	ComPare memory with .Y
DEC	DECrement memory contents by 1
DEX	DEcrement .X by 1
DEY	DEcrement .Y by 1
EOR	Arithmetic Exclusive OR of .A and memory
INC	INCrement memory contents by 1
INX	INcrement .X by 1
INY	INcrement .Y by 1
JMP	JuMP to new location
JSR	Jump to new location, Saving the Return address
LDA	LoaD .A with a byte
LDX	LoaD .X with a byte
LDY	LoaD .Y with a byte
LSR	Logical Shift memory or .A Right one bit
NOP	No OPeration – do nothing
ORA	Inclusive OR memory with .A
PHA	PusH value of .A onto stack
PHP	PusH value of SR (Processor) onto stack
PLA	PulL top value off stack into .A
PLP	PulL top value off stack into SR (Processor)
ROL	ROtate memory or .A one place to the Left
ROR	ROtate memory or .A one place to the Right
RTI	ReTurn from Interrupt
RTS	ReTurn from Subroutine
SBC	SuBtract memory from .A with Carry
SEC	SEt the C (carry) flag to 1
SED	SEt the D (decimal) flag to 1
SEI	SEt the I (interrupt) flag to 1
STA	STore the value in .A into memory
STX	STore the value in .X into memory
STY	STore the value in .Y into memory
TAX	Transfer .A to .X

TAY	Transfer .A to .Y
TSX	Transfer SP into .X
TXA	Transfer .X into .A
TXS	Transfer .X into SP
TYA	Transfer .Y into .A

Most opcodes have to operate on something – the opcode is not complete in itself, with a few exceptions. Rather than set it all out here, the ideas of labels and operands (as the other parts of a typical instruction are termed) will be set out as we go through the opcode family.

The list looks rather unnerving, so without further ado, let's set about demolishing it.

TOPIC 5 — Load and Store

LOAD

The load and the store operations are very similar, with one important (and obvious) difference.

You can load .A, .X and .Y with either a value, e.g., 20, or you can load the actual value that is already stored in another memory location.

If you want to load a specific value, say decimal 32 ($20) you would write

```
   LDA #$20 (Type 1)
or LDX #$20
or LDY #$20
```

This introduces the '#' (hash) symbol. Put simply, it differentiates between the address of a location ($20) and the value hex $20 (#$20). I prefer to work in hex (then there is no mistake about whether I'm in decimal or hex notation), so I will continue to use the hash convention here.

It also enables an introduction to the term 'operand' – which is simply something to be operated on. The '#$20' is operated on by LDA – and whether it is a value or a memory location, it carries the same title.

So, if the status value (ST) that the C64 uses to detect the end of a file is stored in memory location $90 ($96 in CBM/PET), you can call that value into any of the 3 registers with

```
OPCODE.OPERAND
     LDA $90        (Type 2a)
  or LDX $90
  or LDY $90
```

Of course, if you can load from a memory location with a 2 hex digit address, you can also load from one with a 4 hex digit address – giving

OPCODE.OPERAND
LDA $B000 (Type 2b)
or LDX $B000
or LDY $B000

Each type of operation conforms to a specific type of operation. Not all texts use the same naming convention, so I will use the most common one.

Type 1 statements are uniquely named 'immediate' i.e., the load a value specified in the statement – LoaD Accumulator with the number represented by $20, or 32 decimal. Very straightforward.

Type 2 statements are named 'absolute' – what the statement says it is! Go to that location, get the value found there, and put it into the correct register. Again, no problem.

Remember that each byte can hold only the range of values from 0 to 255.

The next form of load opcode is probably one of the most useful. In this form, an index is used. The index relates to the address specified in the instruction, and is added to it. To be more specific, consider the following –

LDA $B000,Y (Type 3)

Depending on the value in the .Y register, load .A with the value found by adding .Y to $B000. If .Y held 7 (#$07 by our convention), then the instruction becomes –

LDA $B007

Note that by performing the addition, I can remove the comma, turning a Type 3 statement into a Type 2 statement. But if I can somehow manage to arrange a loop in which I increase .Y, then I can sequentially load consecutive bytes into .A

Type 3 instructions are termed 'indexed absolute' and you can use .X and .Y (but NOT .A) as indices. You wouldn't use LDX $B045,X-it wouldn't work.

The final form in this group is really only a change of name. The 256 bytes from $00 to $FF (at the very start of computer memory) are called 'zero page'. Usually the locations in zero page are used by BASIC programs to hold all sorts of useful information – the status word ST is one. If BASIC is not being used, many of the locations are sitting idle, and are an ideal place to store useful values in a ML program. So, using zero page is useful, and leads to Type 4 addressing (which looks rather like Type 3)

LDA $A0,X

This is termed 'indexed zero page' – in reality 'indexed absolute' using zero page. Zero page addressing is useful because it uses 1 less byte in each instruction, and operates more speedily than a reference to any other part of memory.

STORE

As noted earlier, store works like load, with one difference. While you can load any number or from any memory location you wish, (immediate type) you obviously cannot do an identical reverse, such as writing to fixed ROM. But you can store a variable into any random access memory location. Consider the following –

STA $07
STY $0401
STX $E007

The first two will work– in the C64 location $07 is in RAM, and $0401 is the second position in screen RAM. But $E007 is Read Only Memory, and you can't normally write to ROM, so the instruction will fail.

Just as you can perform an indexed load, so you do an indexed store –

```
STA $0401,X
```

is quite legal, as is

```
STX $07,Y
```

SUMMING UP

You have seen lots of LDA's, STX's and so on, and you might (at this time) think that the same rules apply to all instructions. The fact is, they don't. Not every instruction supports the same addressing modes as its brothers, so let's set out a table for future reference. The table will list each command, with 'Y' or 'N' to indicate whether the mode is legal or not.

OPCODE	IMMED	ABSO-LUTE	ZERO PAGE	INDEXED ABSOLUTE X	INDEXED ABSOLUTE Y	INDEXED ZERO PAGE X	INDEXED ZERO PAGE Y
LDA	Y	Y	Y	Y	Y	Y	N
LDX	Y	Y	Y	N	Y	N	Y
LDY	Y	Y	Y	Y	N	Y	N
STA	N	Y	Y	Y	Y	Y	N
STX	N	Y	Y	N	N	N	Y
STY	N	Y	Y	N	N	Y	N

This makes it quite clear which goes with what. Don't try to remember them all – you can look up books or make up a collection from these notes for reference.

TOPIC 6 — Increments and Decrements

The next 4 opcodes are termed 'implied', i.e., they have no operand following the opcode, as the operand is implied in the instruction. They are stand-alone instructions. The 4 instructions are –

INX	which adds one to .X
INY	which adds one to .Y
DEX	which subtracts one from .X
DEY	which subtracts one from .Y

There are 2 other opcodes in this group. Like load and store, they have absolute, indexed and zero page options, and all operate on memory locations in RAM

INC $033C adds one to location $033C
DEC $A8 subtracts one from location $A8

INC and DEC support absolute and zero page, and both of these instructions indexed via the .X register.

TOPIC 7 — Register Transfers

The 65XX can shift values between registers with 6 'implied' instructions.

TAX	transfers .A to .X
TAY	transfers .A to .Y
TXA	transfers .X to .A
TYA	transfers .Y to .A
TSX	transfers SP to .X
TXS	transfers .X to SP

There is no direct transfer available between .X and .Y or vice versa. Note that in each transfer, the contents of the source register are transferred to the target register WITHOUT altering the contents of the source register.

There is no INA (INcrement .A) instruction, so transfer commands are combined to provide for this –

```
TAX ;MOVE .A TO .X
INX ;ADD 1
TXA ;AND RETURN IT TO .A
```

This, of course, loses the value in .X at the start. So, if .X is important at the time, you could –

```
STX $D7 ;STORE HANDY
TAX
INX
TXA
LDX $D7 ;GET IT BACK
```

INA would be a useful command – in this example it would save at least 7 bytes if used in a program. Later on, a substitute method will be shown.

SUMMING UP

It might surprise you to know that of the 153 instructions we started out with, 41 types have been covered in this first round. If you don't believe me, count 'em !

TOPIC 8 — Jumps and Branches

BASIC has some very useful commands that permit movement to other parts of programs – e.g., GOSUB takes you to special or often used routines, and RETURN gets you back home again. GOTO also takes you to other parts, straight off. IF..THEN or ON..GOTO permit you to apply some sort of test or check before going to some special place or other. Luckily for ML programmers, there are similar provisions in the 65XX. Let's explore further.

JUMPS

The most commonly used jump instruction is JSR, which is Jump and Save Return. The ML program must remember the address that it jumped from, so that it can return there later. Sometimes JSR is described as Jump to SubRoutine, but that isn't the best description. However, just as a GOSUB works, so JSR takes you to those places in your program where you want particular things to happen, and brings you home again.

The brother instruction to JSR is RTS, which is ReTurn from Subroutine. RTS can be used to finish a subroutine and go back to the mainstream of a program, or, if all work is finished, to return to BASIC.

The last of the jump-type instructions is JMP, which is a straight GOTO. When you say JMP you mean just that – there is no RTS from a JMP. You can jump anywhere in your program, or jump to a routine in the machine itself, such as a routine to print a string on the screen.

BRANCHES

We will treat branches at face value. In fact, they do more than I'm about to tell you, but we'll come back to that later.

Branches allow you to test some condition, and depending on the result of that test, the program will transfer operations backwards or forwards to another location. The only limitation here is that you can only go forwards or back to the limit of about half the maximum value that can be stored in a byte– so we can go back or forward about 127 bytes (near enough for now). This means that the point to which you want to branch has to be reasonably close by – you can't branch from one end of a long program to another.

NB DO NOT NEED TO CMP AFTER INX OR DEX

The principal mathematical branch opcodes are–

BNE	Branch if the result is Not Equal to zero
BEQ	Branch if the result is EQual to zero
BMI	Branch if the result is MInus (< 0)
BPL	Branch if the result is PLus (=> 0)

Using these, we can test all sorts of things. One major use is to test for the end of a loop where an index, say .Y, has been set to a value. As the loop runs, .Y is decreased (DEY), then BEQ (when .Y = 0) to the next job, or BPL back around the loop if .Y is not zero. In just the same way, you could do one more loop by using BMI instead of BEQ. The variations are endless.

LET'S WRITE SOMETHING

Let's put something on the screen where we can all see it.

```
LDY #$01  ;THIS IS SCREEN ASCII FOR 'A'
STY $0400 ;THE FIRST SCREEN POSITION
RTS       ;BACK TO BASIC
```

That didn't hurt, did it. We put the screen ASCII for 'A' into .Y, then stored it in the top left of the screen. (on paper, at least)

EXAMPLE 2

Surely we can do better, so lets put more than one letter up there.

```
LDY #$01  ;SCREEN 'A'
STY $0400 ;ONTO THE SCREEN
LDY #$02  ;SCREEN 'B'
STY $0401 ;NEXT SCREEN POSITION
LDY #$03  ;SCREEN 'C'
STY $0402 ;NEXT SCREEN POSITION
RTS ;BACK TO BASIC
```

Well, that works too, but it seems to be a long way to go to print, say, the alphabet. Are there better ways?

EXAMPLE 3

There's (nearly) always a better way. Let's use all our cunning and make a loop to do it. Better still, we'll use indexed addressing to make it really swing.

```
      LDY #$01.   ;START A COUNTER
      LDX #$09    ;AND ANOTHER HERE
HERE  TYA         ;USE .Y AS CHAR
      STA $0400,Y ;ONTO SCREEN
      INY         ;NEXT LETTER AND POSITION
      DEX         ;ONE LESS TO DO
      BPL HERE    ;IF +VE GO BACK TO 'HERE'
      RTS         ;IF NOT, ALL DONE
```

The list introduces a new idea – did you notice 'HERE' on the third line? It's called a label, and roughly equates to a line number in BASIC. Labels are used for simplicity during program writing – they do not feature in the final ML. Whether it has a name (HERE) or is some address that the computer can find doesn't matter. It's simply a place in the program that can be identified.

What have we done? We have started the screen print with character 'A', and every time we go through the loop, the value stored in .Y is incremented by 1 (INY). A second counter is initialized with the value 9 in .X Each pass of the loop subtracts one from .X (DEX), and the value in .X is tested (BPL) to test whether to go back through the loop (branch to HERE) or to know the job is done and RTS. On each pass of the loop, the next character of the alphabet is printed to successive screen positions through the indexed counter based on $0400 – 'A' in $0401, 'B' in $0402 and so on.

We now have a picture of the structure of a source file–

```
LABEL           OPC.OPERAND          COMMENT
-------------------------------------------------
                LDY  #$01            ;START A COUNTER
                LDX  #$09            ;AND ANOTHER
HERE            TYA                  ;USE.Y AS CHAR
                STA  $0400,Y         ;ONTO SCREEN
etc
```

We will return to this layout when we discuss program entry techniques later on in Topic 17 and Appendix A

TOPIC 9 — Compares

Just as we can take branches depending on whether results are positive, negative, zero or not zero, so we can generate conditions to facilitate branching on the result of compares. Strictly speaking, BPL, BMI, BEQ and BNE are compares, but they are a special set of the family.

There are 3 compare opcodes – compare memory to .A, compare memory to .X and compare memory to .Y What sets this group aside is the method of operation. But first, the opcodes–

CMP	CoMPare to .A
CPX	ComPare to .X
CPY	ComPare to .Y

The mechanism of compares is not as you might expect. The compare operand is either a value, or the value contained in a memory location. The value is subtracted from .A, .X or .Y, but the value in the register is not changed. Nor is the result stored. The result of the compare can be negative, zero or positive (think about that for a minute), and CMP/CPX/CPY is usually followed by a branch, chosen depending on the desired effect of the compare.

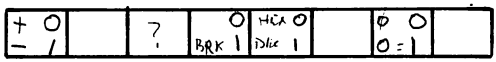
TOPIC 10 — Flags

This topic has been held back until now, whereas in some texts it is introduced earlier. By now, many of the flag-setting instructions have been discussed without reference to flags. In simple programming, the flags are not so critical. But in more complex programming, or programs with more refinement, flag manipulation is very important. Pleasingly, it's really quite easy.

If you cast your mind back, we talked about flags in Topic 3, when we discussed the Status Register. There's nothing different about the Status Register – it's just a byte like any other, except that certain opcodes direct their enquiries there without any further prompting. So what does this byte do?

Just as we numbered bit positions in earlier discussions, the SR is indexed – with names this time. The byte (of the usual 8 bits) looks like this–

Figure 3



Bit position:	7	6	5	4	3	2	1	0
Flag code:	N	V	-	B	D	I	Z	C

All looks mysterious, so some explanations.

N – Bit 7 – The sign flag, negative (N) if the bit is set (=1) or positive if 0.

V – Bit 6 – The oVerflow flag. It is set as a result of some mathematical operations. Little used.

Bit 5 is not assigned

B – Bit 4 – The Break flag, set if a break (BRK) is encountered.

D – Bit 3 – Decimal mode flag. When set, the computer uses BCD (binary coded decimal) mode, rather than hex mode for arithmetic operations.

I – Bit 2 – When set, the normal computer Interrupts (to check the keyboard etc) are defeated.

The next two registers are the most commonly used. Handily, they are at the end of the register where access is easiest.

Z – Bit 1 – The Zero flag. If the result of an operation is zero, Z is set. Otherwise, it is not set.

C – Bit 0 – Carry flag. Set when an operation produces a carry-over from one column of figures to the next more significant one, just as 6 + 5 in normal arithmetic produces a carry digit in the 'tens' column.

A number of opcodes work directly on these flags, to set or reset them. These are–

CLC	CLear the Carry flag (make it 0)
CLD	CLear the Decimal flag
CLI	CLear the Interrupt flag
CLV	CLear the oVerflow flag
SEC	SEt the Carry flag (make it 1)
SED	SEt the Decimal flag
SEI	SEt the Interrupt flag

Flags are important, and worthy of some study on their own. There is not time here to dissect their operation to any depth, but you need to be aware of their existence, and have an eye to their use.

To finish off, there are four Branch operations that we haven't examined, as they needed some appreciation of flags before discussion. They are–

BCC	Branch if Carry (flag) Clear
BCS	Branch if Carry Set
BVC	Branch if oVerflow (flag) Clear
BVS	Branch if oVerflow Set

If the flag is clear (C=0) when BCC is encountered, then the branch is taken. If set, the program merely passes straight through and on. Conversely, if BCS is the instruction and the flag is clear, the branch is not taken.

Similarly with BVC and BVS. The 'branch if clear' – 'branch if set' pattern from above is repeated, this time based on the oVerflow flag.

TOPIC 11 — The Easy Ones

We need a short respite from complex thoughts, so let's look at 4 easily understood instructions.

The Stack (our LIFO pile of plates) can be used for many things, one of the more useful being to 'bank' the values in any or all of .A, .X or .Y before going off to do another task. Many programs in BASIC that have ML sections do this when the BASIC program calls (SYS XXXXX) the ML portion. How do we do this?

The most used instructions are–

PHA PusH Accumulator value onto stack
PLA PulL Accumulator value off stack into .A

If all 3 registers are to be saved, there is a formula to follow that looks like–

```
PHA ;STORE .A
TYA ;PUT .Y IN .A
PHA ;STORE .Y
TXA ;PUT .X IN .A
PHA ;STORE .X
    ;REST OF PROGRAM
    ;
PLA ;GET ONE BACK
TAX ;PUT .A IN .X
PLA ;AND THE NEXT
TAY ;PUT .A IN .Y
PLA ;THAT'S ALL
```

Notice that we put the values onto the stack in order .A-.Y-.X and get them back in order .X-.Y-.A Provided you reverse the order on retrieval, this system is OK. Don't forget that .A must always be first in if you want to keep it.

If you turn back to Topic 7, there was a mention of a better system for temporary storage of a useful parameter. Instead of using STX $D7 to keep the variable, we could use PHA to put the value onto the stack, perform the INX, then PLA to get the value back.

The other instructions are to store the SR contents on the stack–i.e., the flags. The opcodes are–

PHP	PusH Processor onto stack
PLP	PulL Processor into SR

These need little explanation. But it is sometimes a way to preset the flags. If you set up .A, do a PHA then a PLP, the value you set up in .A finishes up in the SR.

TOPIC 12 — Logical Operations

Many of you will not have come across Boolean functions before, so this will all be new to you. But don't worry – it is not difficult.

There are 4 instructions in this group–

AND	...AND!
BIT	This instruction tests memory bits
EOR	Exclusively OR memory with .A
ORA	Inclusively OR memory with .A

That sounds complicated, so a brief exposure to logical operations will help. Such operations do bit-by-bit comparisons of two values, locations, or whatever is specified. Bit 7 in one is compared to Bit 7 in the other, the result is stored, and so on to Bit 0. Bits are either 1 or 0, so a comparison could be comparing 1 with 1, 1 with 0, 0 with 1 or 0 with 0.

AND compares the 8 bits in .A with the specified value or location, bit by bit. If both bits in a comparison are 1 (1 AND 1) then the result is 1. Any other combination gives 0. The result is stored in .A An example–

```
     83 is  #$53 - 01010011
AND 112 is #$70 - 01110000
So (82 AND 112) is 01010000 which is 80 - $50
```

The BIT instruction is an odd one to understand – two bits from memory (Bit 6 and Bit 7) are copied into the V and N flag. The Z flag is set if .A AND memory are not zero. Don't worry too much – this is for more advanced work.

EOR is also a bit-by-bit compare. This time, if either one or the other bit is 1, then the comparison result is 1. If both bits are 1 or both are 0, the result is 0. EOR is very handy for handling the Commodore

screen, particularly for reverse video. In Commodore code, the end bit (Bit 7) is off for normal display, and on for reverse. To reverse a character, Bit 7 has to be turned on–

```
LDA $0400 ;GET CHAR FROM SCREEN
EOR #$80  ;REVERSE IT
STA $0400 ;PUT IT BACK
```

```
'E' is        00000101
Reverse it 10000000
And get..  10000101 which gives reverse 'E'
```

ORA is the last of the four opcodes to consider. During the comparison, if any of the bits being compared is 1, then the result is 1. An example is probably best–

```
Rev 'E' is   10000101
ORA #$80 10000000
And get..    10000101 so it remains unchanged
```

A further example of AND and ORA can be used to blank the C64 screen during program load and save operations – this speeds up the process for disk, and is essential for cassette. Screen blanking is controlled through bit 4 of the VIC Control Register, location $D011 (53265). When bit 4 is 'off', the screen is blanked. The routine looks like this–

```
LDA $D011 ;GET PRESENT VALUE
AND #$EF  ;TURN BIT 4 OFF
STA $D011 ;PUT VALUE BACK
```

To turn the screen on again–

```
LDA $D011
ORA #$10 ;TURN BIT 4 ON
STA $D011
```

The effect of AND #$EF is to leave all bits except bit 4 as they were. Bit 4 is made zero. In reverse, ORA #$10 only affects bit 4. In BASIC, the routine would be–

```
POKE 53265, PEEK(53265) AND 239 = ON
POKE 53265, PEEK(53265) OR 16 = OFF
```

TOPIC 13 — Addition and Subtraction

Arithmetic operations are not well implemented in Commodore computers. Commodore are not unique in this – it is common to all older microprocessors, but is improved in newer devices. But you will be able to get by (most of the time) without doing a vast amount of adding and subtracting. There are only two instructions in this group–

ADC ADd with Carry
SBC SuBtract with Carry

With addition and subtraction, there can often be a carry – if the result exceeds 255 (#$FF) or is less than 0 (#$00), just as 9 + 4 produces a carry from units to tens in decimal addition. When this happens, the C (Carry) flag in the SR is set to 1. Of course, you may have already set C with a previous calculation, so it is usual to perform CLC (who remembers?) before using ADC. With subtraction, it is usual to do it the other way around – set C before an SBC.

The technique can be used to add large numbers, using C to indicate carry within each byte. Carry between bytes is handled by the 65XX for you. But let's stick with a simple addition–

```
CLC       ;JUST TO BE SURE
LDA $70   ;PUT MEM INTO .A
ADC #$16 ;ADD 22
STA $70   ;PUT IT BACK
SEC       ;SET THE FLAG
LDA $77   ;PUT MEM INTO .A
SBC #$04 ;TAKE AWAY 4
STA $77   ;PUT IT BACK
```

There are many and various ways of combining ADC and SBC with other instructions to produce quite powerful operations – and both ADC and SBC set and unset flags with gay abandon. Experiment !!!

TOPIC 14 — Shifts and Things

Let's look at a byte – all 8 bits of it. Any value will do, so lets use decimal 42 – #$2A

00101010

Lets move all the bits one place to the left–

01010100

This comes to #$54, or 84 decimal. What do you notice?

Let's take #$2A again, and shift the bits to the right this time–

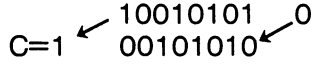
00101010
becomes 00010101

which is #$15, or 21. The plot thickens, if you hadn't noticed. Without further ado, let's introduce the next set of instructions.

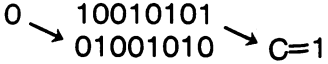
ASL	Arithmetic Shift Left
LSR	Logical Shift Right
ROL	ROtate Left
ROR	ROtate Right

The principal difference between these instructions is what happens to the bit that falls off the end, and what is put into the empty bit at the other 'end'.

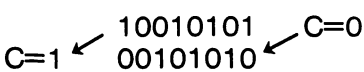
ASL keeps the displaced bit in the Carry flag, and puts a zero into Bit 0.



LSR is near enough to the reverse of ASL. Bit 0 goes into the Carry, and 0 is put into Bit 7



ROL is a nine-bit rotation. All bits are rotated one place left – the value in C goes into Bit 0, and then what was Bit 7 goes into the Carry flag.



ROR is another nine-bit rotation, the reverse of ROL. The Carry goes into Bit 7, and Bit 0 goes into the Carry flag.

THE LAST ONES

There are only three more opcodes. They are:

SEI	which is SEt Interrupt
CLI	which is CLear Interrupt
RTI	which is ReTurn from Interrupt

At this stage (and perhaps for a while yet) you won't need to worry about these. Just note their existence. When you are more experienced, you will learn to work with these useful commands.

TOPIC 15 — Other Addressing Modes

We have covered all the instructions, but there are some addressing modes that are a little more complex. I have left them until now, so that you will be in a better position to understand them.

Before starting in, there is a feature of the 65XX that you may already know. It concerns a peculiarity of 65XX architecture when storing numbers greater than #$FF (255 decimal) To store a number like 1036 ($040C) takes two bytes. One byte stores #$04 and the other byte stores #$0C. But the 65XX likes to work back to front, so it stores #$0C in the lower byte and #$04 in the higher byte. Almost without exception, you will see numbers stored this way in Commodore computers.

The main addressing mode that makes use of this feature is termed 'indirect', and it may be indexed or absolute. Let us consider indirect indexed by an example.

Zero Page
5D
5E 03 LOW
5F 04 HIGH
60

LDA ($5E),Y

translates as follows

LDA (the 2-byte address $5F$5E),Y

so if $5F contains #$04 and $5E contains #$03 we have

LDA ($0403),Y LOAD A WITH CONTENTS OF 0403 +Y

Note that the address is compounded from the two bytes (#$03 and #$04 in reverse order) THEN the value in .Y is added to determine the true location.

The value in .X can also be used, but the operation is different.

LDA ($D7,X)

The value in .X is added to the zero page address, to get a new address. This address, together with the next byte, is the location pointed to. For example, if $D7 is #$5C and .X is #$02, then the instruction goes to the two bytes at $D7 and $D8. If $D7 is #$0A and $D8 is #$80, then the byte at $800A is loaded into .A

eg Zero Page.

D5 + LDA (D5, x)

If x = 2.

D7 0A

D8 80

A is loaded with the contents of Address 800A.

TOPIC 16 — To Finish *Off*

We have now looked at all the opcodes, and introduced the addressing concept. At this point, let's concentrate on reviewing what we know about addressing modes, and tabulate the individual modes.

The first mode was IMMEDIATE, where we loaded a variable into .A, .X or .Y.

```
LDA #$20 puts decimal 32 into .A
LDX #$45 puts decimal 69 into .X
```

The next mode was ABSOLUTE, where we loaded or stored information from or to an address. ABSOLUTE ZERO PAGE uses addresses between $00 and $FF – the first 256 bytes in the computer. ABSOLUTE (on its own) refers to any other address that the computer can access.

```
LDA $B000
STA $A7
```

After that came INDEXED mode – ZERO PAGE INDEXED by .Y or .X (zero page meaning the same as above) and ABSOLUTE INDEXED, for any other address.

```
LDA $B147,Y
STA $0400,X
LDA $D7,Y
```

Branches followed – a mode on its own, termed RELATIVE because the jump 'distance' is related to the bytes being executed.

IMPLICIT was straightforward – INY adds one to .Y, DEX takes 1 from .X, PHA pushes the accumulator onto the stack, and so on. The opcodes stand alone, implying their action without need for any more explanation.

The INDEXED mode was next. In this mode, the computer calculates its action using a 65XX shorthand convention:-

LDA ($42),Y

gets the bytes from $42 and $43, applies the 65XX convention of low byte-high byte and loads.A with the contents of that address- in this case, offset by .Y In this case, the mode is termed INDIRECT INDEXED.

As a shorthand guide, the following table sets out the major modes:-

IMMEDIATE	LDA #$00
ABSOLUTE	LDA $C000
ABSOLUTE ZERO PAGE	LDA $2A
ABSOLUTE INDEXED	LDA $B000,Y
ZERO PAGE INDEXED	LDA $20,Y
RELATIVE	BNE $1024
IMPLIED	INY
INDIRECT INDEXED	LDA ($2A),Y
ACCUMULATOR	ASL A

There is one other mode, not covered in the main text. INDIRECT (sometimes termed ABSOLUTE INDIRECT) means that the program goes straight to an address, finds the contents of the address and the byte following it, and goes to that place to continue.

Some texts expand these to indicate that some modes work only with certain parameters, one example being to indicate whether .X, .Y or both can be used as indices. However, the principal modes are listed above.

These names are used somewhat flexibly by different writers. This can cause some confusion and user difficulty, but if you learn to say the mode to yourself each time you use the various instructions, they will form a distinct pattern in your mind and ease the programming task.

TOPIC 17 — Editors and Assemblers

Programming in ML is nothing like BASIC programming. With BASIC, you are dealing in quasi-English, working with a dynamic screen, and you can insert, delete, move, renumber, rename and otherwise fiddle to your heart's content – all nicely protected by a BASIC interpreter that does all sorts of convenient things for you, even down to telling you where your program ground to a halt. Not so ML – there is no tidy entry routine directly available for use. You have to look for help.

Up to now, we have been dealing in labels, opcodes and operands, which are reasonably easy to read by us, but which are of no direct value to the computer.

At the outset, it was suggested that you get a copy of an extended monitor – SUPERMON or some such. These programs contain a 'tiny assembler', which translates opcodes and operands into the bytes that are understood by the computer.

No, the computer doesn't understand things like LDA #$01 – but it does understand a translation of it. Who or what does the translation?

To understand that, let us go through the process of assembly. For small programs, the tiny assemblers are adequate, but you should have a rough draft of your program down on paper. So, tiny assemblers are useful, but restricted. Fire up your monitor program, and let's practice. Say it's SUPERMON, for the example.

SUPERMON (like most monitors and extensions) prompts you for an entry with a '.' To assemble with SUPERMON, type 'A' after the '.', a space, then the start address (in hex) of the first instruction, then the opcode and operand.

.A 1000 LDA #$01 <Carriage Return>

SUPERMON will promptly translate this into hex bytes, store them and print them thus

.A 1000 A9 01

and prompt you for the next instruction by printing the next address ($1002 in this case).

.A 1000 A9 01
.A 1002

You continue entering the program, and simply press RETURN (<CR>) when entry is finished.

For larger jobs, some form of Assembler package is needed. There are a number of these for disk users, the best known being MAE (disk only) or ASSM/TED (for disk and cassette users) and the CBM Assembler packages for the CBM/PET series and the C64. These will be discussed later. Each package contains programs to edit, assemble and load ML programs. Let's look at each of these functions.

An editor allows the user to enter instructions in simple form, something like entries in SUPERMON as described above. Usually the entry looks something like a BASIC program – line number and so on–

```
10 LDA #$01
20 STA $22    ;COUNTER FOR PAGES
30 LDA #$42
40 STA $23    ;66 LINES PER PAGE
```

Editors usually permit comments ("REMs") as well (the ';' is translated to REM), so that the function of each line can be noted. Files, saved either on disk or cassette, contain sufficient instructions to create the desired program, and are usually termed 'source files', the name indicating that the file contains the original material for the eventual program.

The next task is to assemble (translate, interpret, convert) the source file. The assembler coverts all opcode/operand combinations

to hex, in a form understood by the computer. During this phase, the assembler checks for syntax validity, but not whether the program will work or fail. That's up to you!! Also during assembly, you may create another file, containing the hex bytes only. This file is usually termed the 'object code', 'code' as it is all in hex.

Finally, the object code is loaded, usually using another routine that reads the object code and stores it in memory. At that point, the program is ready for the final test – run time.

Sometimes, the ML bytes that have been loaded into the computer are 'saved' as a file of their own – termed a 'binary file'. This binary file can later be loaded directly back into the computer without need for an intermediate assembler program. At run time, this can be very convenient.

We will use all of these features to provide some knowledge of their use. Most assembler packages are documented (for better or worse – usually ambiguously) so you can get further guidance there. If all else fails, you could always contact a more experienced user.

ASSEMBLER SUITES

As noted above, the principal packages are MAE (Macro Assembler Editor) and the CBM Assembler Packages. The former is a commercial product, with versions for CBM/PET and C64. The package is largely self-contained, very powerful, with some very nice features that make its use a pleasure. In addition, there is a group in the USA (ATUG – ASSM/TED Users Group) who promote/adapt/improve MAE, with membership open to MAE owners. If MAE has a drawback, it is that source files are restricted to about 1K each. As it is possible to 'chain' (connect) source files to form very large program files, this is not a total limitation.

The CBM packages are Public Domain Software, available from Commodore dealers at a nominal price ($60 when last checked) or your local User Group for a copying fee. The package contains 4 discrete program groups for disk operation - Editor, Assembler, Loader and Cross Reference, each of which has to be individually loaded as the need arises. The software lacks the sophistication of MAE, but overall is very useable. Documentation is adequate but not over-helpful.

Because of lower cost, the CBM Assembler Packages (3032/4032/8032, and C64 versions) are attractive, so it may be the starting point for would-be ML programmers. Appendix A sets out an outline technique for using the package.

AND IN CONCLUSION

By now, you should have a general outline of ML programming. The book was not intended to provide an in-depth, exhaustive volume designed to lead straight to a degree in computer science. If I have not confused you too thoroughly, and you still want to do some self-motivated study, then you are on your way to ML proficiency.

A second volume is planned, to bring together some useful ideas and routines that will form a basis for your ML program library.

Paul Blair

APPENDIX A
USING THE COMMODORE 64 PACKAGE

The Commodore package is designed for use with a disk drive. There is no cassette version.

The package consists of 4 main elements –

EDITOR
ASSEMBLER
LOADER
CROSS REFERENCE

EDITOR:-

This is a ML program that loads into memory (LOAD"EDIT.C64",8,1) at $C000 (49152), and is activated by a SYS to that location. The EDITOR provides for preparation of input files, and helps by some additional features. These include –

AUTO – to provide automatic line number entry.
CHANGE/FIND – to find and/or amend any characters.
DELETE – erase any range of lines.
FORMAT – tab source file to the screen.
GET – the EDITOR's 'Load'.
KILL – turn off the EDITOR.
NUMBER – renumber the file.
PUT – the EDITOR's normal 'Save' to disk.
CPUT – the EDITOR's compressed 'Save'.

You will recall from Topic 8 that the format used for file entry is –

LINE NUMBER (LABEL) OPCODE OPERAND (COMMENT)

Titles in parentheses are optional. The operand is not required when using implied opcodes (TAX INY etc.)

So, let's start with a program. Assume that you have drafted out your program on paper, then loaded and SYS'ed the EDITOR into life. What now?

We start typing –

```
100 START  LDX #$00          ;ZERO INTO COUNTERS
110   STX $B2                ;COUNTER 1
120   STX $B3                ;COUNTER 2
130   TXA                    ;MOVE TO .A
140   TAY                    ;SET INDEX TO 0
150 TITLE  LDA (HEADER), Y   ;GET A CHARACTER
160   AND  #$7F              ;CORRECT FOR SCREEN
170   JSR  $FFD2             ;ROM PRINT ROUTINE
180   INY                    ;ADD 1 TO INDEX
190   CPY  #$20              ;DONE 32 YET?
200   BNE  TITLE             ;LOOP BACK
210 OPEN .....
```

The program is entered somewhat like a BASIC program – line number and text, except that the EDITOR program is handling things now, not the BASIC interpreter in the computer. In fact, while in EDITOR mode, it's not a good idea to try any BASIC programs. KILL the EDITOR and go back to BASIC if you want to do that.

The physical layout shown here is deliberate. The fields are separated by a space, essential for the ASSEMBLER program. The exception is where a label is not used, and you should leave TWO spaces after the line number. This is so that the FORMAT command will work. The ASSEMBLER can cope with this variation, so the extra space causes no problems.

When you have typed in the program, the next step is to store it on disk. PUT is the command, so we –

PUT"0:FILENAME"

There are options here. The drive number can be omitted, and the EDITOR will default to the last drive used (not a problem with single disk drives!) and store your text (without line numbers) as a SEQ file. I have used the 'save with replace' ('@0:') without any problems on DOS 1 and 2 – but you should follow your own practice here, as there have been assertions as to the unreliability of this syntax. PUT also permits line number ranges to be selected if desired, as you use LIST in BASIC, to file your text selectively.

A variation of PUT is CPUT. The effect is the same as PUT, except that unnecessary spaces are not written to file. The idea of this is to speed up the Assembler process.

The opposite of PUT is GET – not BASIC GET but EDITOR GET. Syntax is easy

GET "FILENAME"

without drive number will do. The sequential file will be loaded, and assigned line numbers starting from 1000 and going up in tens. If need be, you can now edit your program, then PUT it back on disk.

ASSEMBLER :-

More ML, this time residing at the usual BASIC start. (LOAD"ASM.C64",8) The ASSEMBLER can co-reside with the EDITOR, which is safely at $C000. In fact, to return to the EDITOR after using ASSEMBLER, simply do SYS 49152.

To activate the ASSEMBLER, type RUN and you're in business. You will be asked some questions – note that no quotation marks need be typed for entries in this phase. Do you want an object file, ready to be loaded into the computer? If so, on which drive and with what name? If no file is needed (say during development) press <CR>. The next question concerns hardcopy. If required then <CR> again. If not, type "N" with <CR>. If you said "N" to an object file, you will be asked if you want cross reference files. Respond as your needs dictate. Finally, you need to enter the name of the source file to be assembled. The drive number can be specified, but is not

mandatory. If all has been correctly entered, then ASSEMBLER goes to it.

ASSEMBLER makes two passes, looking for labels, jumps and branches on the first pass, then computing all values on the second pass. At the end (hopefully) you will be told "ERRORS = 0000" or (less hopefully) the number that were found. Remember, these are syntactical errors – the 'nil errors' message does not guarantee the logic of your program!

Assuming that you requested an object file, the next step is to load the information into the computer. Before doing so, it is advisable to completely reset the computer.

At this point, there is one step that you will probably want to take. If you are going to load the object file into your computer, it is likely that you will want to save the ML bytes for later. To do this, you will need a monitor program to provide the necessary save function. Select a monitor unlikely to be affected by the loading process (a monitor in $C000 is usually pretty safe) and get it in place. Now, to continue....

Depending on where you intend to store the program when loaded, you have a choice of LOADERs, LO-LOAD and HI-LOAD. Check this table for usage –

PROGRAM TO BE LOADED TO	USE THE x-LOADER
LOW MEMORY	HIGH
MID MEMORY	LOW or HIGH
HIGH MEMORY	LOW

LOADER	TO RUN
LOW	RUN
HIGH	SYS 51200

The loaders will firstly ask for an offset. If your program is written to reside at $1000 but you want it temporarily stored at $3000, then the offset is $2000. You might not see the point of offsets just yet, but the time will arrive.....

For now, ignore the question by a carriage return (<CR>) and go on to the real task– loading the object file. Enter the filename, <CR>, and all being well you will see the starting address (in hex) of the actual loading place on the screen. This will be followed by a series of dots, indicating the progress of the load. Finally, the end address of the actual storage location is printed on the screen. Remember or write down these addresses for the next step.

The message 'END OF LOAD' signifies the end of the load phase. Type 'NEW' to reset pointers, and then (before doing anything else) SAVE the program. Jump into the monitor at the address you stored it, then

.S'd:filename',08,aaaa,bbbb <CR>

where 'd' is the disk drive number, 'aaaa' is the start of your program and 'bbbb' is ONE BYTE PAST the end of it. If you don't go one byte past, you will lose the last byte of your opus.

That's it. You now have a copy in the computer, and a copy on disk. What you do next is up to you.

If you have any problems with this routine, have a look at this diagram, which is a flowchart of the main points written so far. If this all looks a bit long winded, then consider what life was like before assembler packages were developed and all programs were hand written, assembled and entered!!

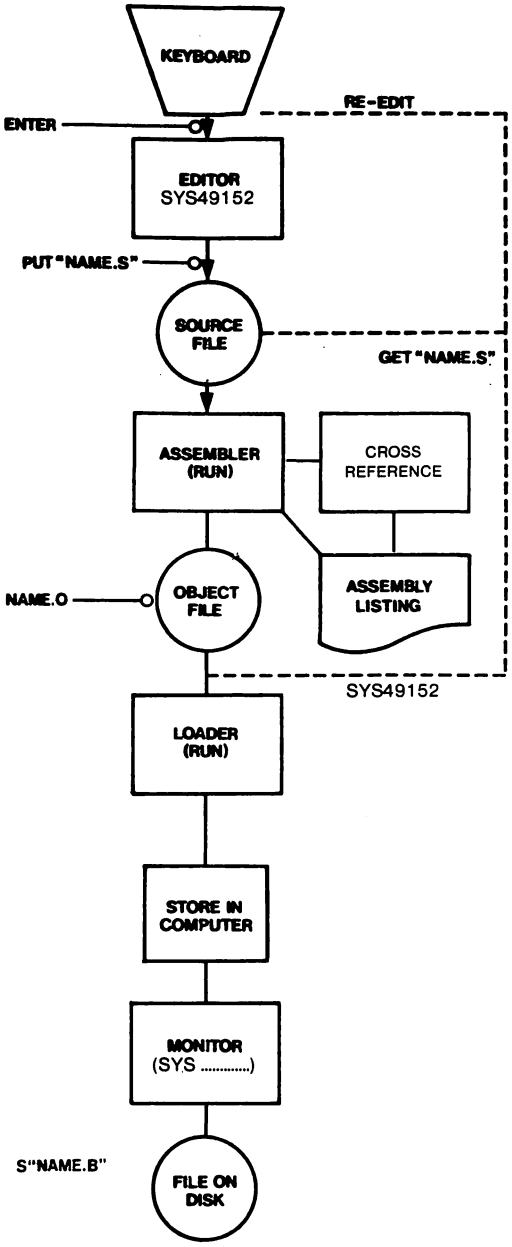
A useful feature now implemented in all Commodore Assembler Packages is the cross reference option. At the start of assembly, you may choose to create an object file, or a series of cross reference files. You can have one or the other, but not both.

If you choose the cross reference option, the Assembler will create a series of files on disk during run time. When the Assembler has finished, load CROSSREF64 and type RUN <CR>. You may elect

to direct output to the screen or printer. The result is a listing that shows all labels used in the source files, and the Assembler output line numbers where the labels occur. This is particularly useful for later reference, trouble shooting or unravelling programs that are more complex.

That completes a run through the principal technique required to use the Commodore Assembler package. The documentation explains a wider range of sub-options that you can use in the development process.

ASSEMBLY PROCEDURE FLOWCHART



NAMING CONVENTIONS

We have noted that the assembly process creates three files (they are usually called 'files' for some obscure reason) the original source file, the object file, and the ML file. To ease the task of subsequent use by you or others, it is convenient (but not mandatory) to use some convention to identify which is which. One convention that has become widespread is to use the following suffixes to file names:-

. SRC or . S for source files
. OBJ or . O for object files
. BIN or . B for ML (binary) files

To some, the SOB initials sum up ML programming very succinctly!

ML is sometimes converted into a BASIC loader program. Each byte of ML is kept in DATA statements, which the BASIC program READs and the values are POKE'd into the appropriate memory location. If you do this, the BASIC program could be given the suffix . BAS- but be careful not to use . B for binary AND Basic files.

APPENDIX B
A SAMPLE GRAPHICS PROGRAM

Appendix B is presented for study in conjunction with the introduction to programming presented in this book.

As the printout indicates, the program has been written for a Commodore 64 computer.

As the program notes, it is long on detail and (somewhat) short on sophistication. This is deliberate, because it will permit a better grasp of the fundamental steps used in the program. To tease you, most of the routines could be accomplished in fewer bytes – but I'll leave the investigation and solution of that to you.If you don't want to type in the program, disk and cassette copies are available from where you purchased this book or through KIM BOOKS.

Read on. The more you read/study other peoples programs, the faster your abilities in ML will grow.

BASIC LOADER

If you would like to load the program without having to go through the assembly process, here is a BASIC loader that will do the job for you. I have given the listing here in short form, without lines numbers and any pretty screen routines. You may add them to taste.

```
BASIC LOADER FOR C64 ML PAUL BLAIR - COMPUCHART
PRINT'LOADING C64 ML PROGRAM'
PRINT'PLEASE WAIT'
S=49152 : F=50263 : FOR I = S TO F
READ A : POKE I,A : NEXT
PRINT'LOADING COMPLETE'
PRINT'AND ROUTINES INITIALIZED'
PRINT
PRINT'(C) KIMBOOKS 1984' : SYS 49152 : NEW

DATA 169,76,133,115,169,26,133,116,169,192,133,117,96,
     169,230,133
DATA 115,169,122,133,116,169,208,133,117,96,142,255,207,
     186,189,1
DATA 1,201,140,208,16,189,2,1,201,164,208,20,230,122,208,2
DATA 230,123,76,76,192,201,230,208,7,189,2,1,201,167
     ,240,236
DATA 174,255,207,230,122,208,2,230,123,76,121,0,160,0,
     177,122
DATA 201,64,240,3,76,121,0,230,122,208,2,230,123,177,
     122,230
DATA 122,208,2,230,123,201,66,208,3,76,223,192,201,67,
     208,3
DATA 76,63,193,201,68,208,3,76,110,193,201,69,208,3,76,175
DATA 193,201,70,208,3,76,240,193,201,73,208,3,76,29,
     194,201
DATA 74,208,3,76,65,194,201,75,208,3,76,112,194,201,76,208
DATA 3,76,129,194,201,77,208,3,76,186,194,201,78,208,3,76
```

```
DATA 225,194,201,82,208,3,76,251,194,201,83,208,3,76,59,
     195
DATA 201,84,208,3,76,116,195,201,85,208,3,76,137,195,201,
     87
DATA 208,3,76,188,195,201,90,240,3,76,8,175,76,13,192,169
DATA 6,133,76,32,155,183,134,77,169,247,229,77,133,75,
     32,241
DATA 183,134,140,32,241,183,142,134,2,165,140,201,8,48,
     26,233
DATA 8,133,140,169,160,145,75,32,45,193,24,165,75,233,
     39,133
DATA 75,165,76,233,0,133,76,208,224,170,189,37,193,145,
     75,32
DATA 45,193,76,115,0,100,111,121,98,248,247,227,160,169,
     220,69
DATA 76,133,76,173,134,2,145,75,169,220,69,76,133,76,96,32
DATA 155,183,169,4,133,76,169,39,134,77,229,77,105,79,
     133,75
DATA 169,32,145,75,24,165,75,105,40,133,75,165,76,105,
     0,133
DATA 76,201,7,208,235,165,75,201,32,48,229,76,115,0,160,40
DATA 132,142,160,0,132,140,160,191,162,7,134,141,134,143,
     177,140
DATA 145,142,32,162,193,177,140,145,142,32,162,193,136,
     192,255,208
DATA 237,202,224,3,208,228,160,39,169,32,145,140,136,16,
     251,76
DATA 121,0,169,220,69,143,133,143,169,220,69,141,133,141,
     96,169
DATA 2,162,8,168,32,186,255,32,13,196,152,162,0,160,207,32
DATA 189,255,32,192,255,162,2,32,198,255,32,207,255,32,
     210,255
DATA 36,144,112,17,201,13,208,242,173,141,2,41,1,208,
     249,165
DATA 197,201,63,208,229,32,204,255,169,2,32,195,255,76,
     115,0
DATA 32,155,183,134,75,32,241,183,142,134,2,160,0,132,
     140,162
DATA 4,165,75,134,141,145,140,32,168,193,173,134,2,145,
     140,32
DATA 168,193,165,75,200,208,238,232,224,8,208,231,96,162,
     4,134
```

```
DATA 76,160,0,132,75,32,155,183,142,134,2,166,76,177,75,73
DATA 128,145,75,32,45,193,200,208,244,232,134,76,224,8,
     208,237
DATA 96,32,13,196,169,40,132,77,229,77,133,77,169,32,
     32,210
DATA 255,198,77,198,77,165,77,201,1,16,241,160,0,185,0,207
DATA 240,6,32,210,255,200,208,245,169,13,32,210,255,76,
     115,0
DATA 169,49,141,20,3,169,234,141,21,3,169,16,133,252,
     76,121
DATA 0,162,4,160,0,132,140,132,142,169,195,133,143,134,
     141,230
DATA 143,230,143,169,0,133,142,177,142,145,140,32,168,
     193,230,143
DATA 177,142,145,140,32,168,193,198,143,200,208,235,232,
     224,8,208
DATA 220,173,254,207,141,33,208,76,121,0,32,155,183,134,
     76,32
DATA 241,183,134,75,160,0,132,140,162,4,165,75,134,141,
     177,140
DATA 197,76,208,4,165,75,145,140,200,208,243,232,224,8,
     208,236
DATA 96,32,158,173,32,247,183,72,169,36,32,210,255,104,
     32,58
DATA 196,152,32,58,196,169,13,32,210,255,96,120,169,14,
     141,20
DATA 3,169,195,141,21,3,169,1,133,78,88,76,122,194,165,197
DATA 197,254,240,9,133,254,169,16,133,253,76,49,234,201,
     255,240
DATA 249,165,253,240,4,198,253,208,241,198,78,208,237,
     169,4,133
DATA 78,169,64,133,197,169,2,133,205,208,223,173,33,208,
     141,254
DATA 207,162,4,160,0,132,140,132,142,169,195,133,143,134,
     141,230
DATA 143,230,143,169,0,133,142,177,140,145,142,32,168,
     193,230,143
DATA 177,140,145,142,32,168,193,198,143,200,208,235,232,
     224,8,208
DATA 220,76,121,0,32,155,183,134,75,32,241,183,134,76,
     166,75
```

```
DATA 164,76,24,32,240,255,76,115,0,162,0,134,142,162,
     40,134
DATA 140,162,4,134,143,134,141,160,0,177,140,145,142,32,
     162,193
DATA 177,140,145,142,32,162,193,200,208,239,232,224,8,
     208,228,169
DATA 32,162,40,157,199,7,202,208,250,76,121,0,173,33,
     208,141
DATA 253,207,162,4,160,0,132,140,132,142,169,195,133,
     143,134,141
DATA 230,143,230,143,177,142,133,75,177,140,145,142,
     165,75,145,140
DATA 32,168,193,230,143,177,142,133,75,177,140,145,142,
     165,75,145
DATA 140,198,143,32,168,193,200,208,219,232,224,8,208,
     208,173,254
DATA 207,141,33,208,173,253,207,141,254,207,76,121,0,32,
     115,0
DATA 201,34,240,3,76,8,175,160,1,177,122,240,247,201,
     34,240
DATA 6,153,255,206,200,208,242,24,169,0,153,255,206,152,
     101,122
DATA 133,122,169,0,101,123,133,123,136,96,72,74,74,74,
     74,201
DATA 10,144,2,105,6,105,48,32,210,255,104,41,15,201,10,144
DATA 2,105,6,105,48,76,210,255
```

C64ML.SRC

```
LINE#  LOCN  M/CODE       LABEL  OPC OPERAND      COMMENT

00001  0000               ;***************************
00002  0000               ;*                         *
00003  0000               ;*    'ML ON THE C64'      *
00004  0000               ;*                         *
00005  0000               ;* ML PROGRAMMING EXAMPLES *
00006  0000               ;*                         *
00007  0000               ;*       COMMODORE 64      *
00008  0000               ;*     40 COLUMN SCREEN    *
00009  0000               ;*                         *
00010  0000               ;*   ADAPTED FOR CBM/PET   *
00011  0000               ;*     -PAUL BLAIR         *
00012  0000               ;*                         *
00013  0000               ;*     ADAPTED FOR C64     *
00014  0000               ;*     -PAUL BLAIR         *
00015  0000               ;*     -PETER FLETCHER     *
00016  0000               ;*                         *
00017  0000               ;***************************


00019  0000               ;THIS PROGRAM PROVIDES SOME USEFUL
00020  0000               ;EXAMPLES OF SHORT ML ROUTINES,
00021  0000               ;MAINLY FOR SCREEN HANDLING.
```

```
00022  0000                ;THE ROUTINES ARE AVAILABLE IN
00023  0000                ;BOTH DIRECT AND PROGRAM MODE.
00024  0000                ;
00025  0000                ;EACH ROUTINE IS COMMENTED TO
00026  0000                ;DESCRIBE ITS METHOD OF OPERATION.
00027  0000                ;
00028  0000                ;THE ROUTINES ARE NOT SOPHISTICATED,
00029  0000                ;THEY HAVE BEEN WRITTEN TO BE
00030  0000                ;ILLUSTRATIVE.  THERE ARE MORE
00031  0000                ;COMPLEX AND INTRICATE WAYS OF
00032  0000                ;DOING THEM - YOU SHOULD BE ABLE
00033  0000                ;TO IDENTIFY WAYS OF MAKING
00034  0000                ;IMPROVEMENTS.


00036  0000                ;THE ROUTINES USE THE '@' KEY TO
00037  0000                ;PREFACE THE COMMANDS - EG @R WILL
00038  0000                ;TURN ON REPEAT FOR ALL KEYS, @K
00039  0000                ;WILL TURN IT OFF. @E "XXXXXX"
00040  0000                ;READS SEQUENTIAL FILE "XXXXXX"
00041  0000                ;DIRECT TO THE SCREEN WITHOUT
00042  0000                ;DISTURBING ANY PROGRAM IN MEMORY.
00043  0000                ;
00044  0000                ;ALL COMMANDS ARE NOTED IN THE PROGRAM
00045  0000                ;LISTING, TOGETHER WITH THE CORRECT
00046  0000                ;SYNTAX FOR USE.  HAVE FUN!!
```

ADDRESSES

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00048  0000              ;***************************
00049  0000              ;*  DEFINE MACHINE LABELS  *
00050  0000              ;***************************
00051  0000              ;
00052  0000              ;---ZERO PAGE---
00053  0000              ;
00054  0000              USR0   =$4B             ;THREE LOCATIONS
00055  0000              USR1   =$4C             ;.. FOR TEMPORARY
00056  0000              USR2   =$4D             ;.. STORAGE
00057  0000              REPDL  =$4E
00058  0000              INDEX1 =$8C             ;FOUR LOCATIONS
00059  0000              INDEX2 =$8D             ;.. TO ACT
00060  0000              INDEX3 =$8E             ;.. AS POINTERS
00061  0000              INDEX4 =$8F             ;.. TO PLACES
00062  0000              CHRGET =$73             ;GET NEXT CHAR FROM BASIC
00063  0000              CHRG7  =CHRGET+7
00064  0000              CHRG8  =CHRGET+8
00065  0000              CHRGOT =$0079           ;REGET CHAR FROM BASIC
00066  0000              SATUS  =$90             ;VARIABLE ST
00067  0000              LSTX   =$C5             ;WHICH KEY HAS BEEN PUSHED?
00068  0000              BLNCT  =$CD             ;BLINK COUNTER FOR CURSOR
00069  0000              TEMP1  =$FC             ;3 STORAGE SPOTS
00070  0000              TEMP2  =$FD
00071  0000              TEMP3  =$FE
00072  0000              ;
00073  0000              ;---OTHER PAGES---
```

```
00074  0000                ;
00075  0000                COLOUR =$0286          ;CHARACTER COLOUR
00076  0000                SFST   =$028D          ;HAS SHIFT KEY BEEN PUSHED?
00077  0000                CINV1  =$0314          ;IRQ INTERRUPT VECTOR LOW
00078  0000                CINV2  =$0315          ;IRQ INTERRUPT VECTOR HIGH
00079  0000                ;
00080  0000                SCREEN =$0400          ;C64 SCREEN RAM
00081  0000                ;
00082  0000                ;---C64 ROM 2.0 ROUTINES---
00083  0000                ;
00084  0000                READY  =$A474          ;BACK TO BASIC
00085  0000                FRMNUM =$AD9E          ;EVALUATE EXPRESSION
00086  0000                SNERR  =$AF08          ;SYNTAX ERROR
00087  0000                GTBYTE =$B79B          ;CHRGET + GET BYTE
00088  0000                COMBYT =$B7F1          ;EVALUATE FORMULA
00089  0000                GETADR =$B7F7          ;FAC TO INTEGER
00090  0000                COLTMP =$CFFD          ;TEMP SCREEN COL
00091  0000                POP    =$CFFF          ;KEY POP
00092  0000                SCCOL  =$D021          ;SCREEN COLOUR
00093  0000                KEY    =$EA31          ;INTERRUPT
00094  0000                CLOSE  =$FFC3          ;CLOSE FILE IN '.A'
00095  0000                SETLFS =$FFBA          ;SET LA,FA,SA
00096  0000                SETNAM =$FFBD          ;SET FILENAME PTR
00097  0000                OPEN   =$FFC0          ;OPEN LOGICAL FILE
00098  0000                CHKIN  =$FFC6          ;JUMP TO SET INPUT DEVICE
00099  0000                CLRCHN =$FFCC          ;CLOSE CHANNELS
00100  0000                INCHR  =$FFCF          ;INPUT A CHAR
00101  0000                OUTCH  =$FFD2          ;OUTPUT A CHAR
00102  0000                PLOT   =$FFF0          ;PLOT CURSOR POS'N
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00104  0000              ;--------------------------
00105  0000                     * =$C000         ;START AT SYS49152
00106  C000              ;--------------------------
00107  C000              ;
00108  C000              ;*****************************
00109  C000              ;* ENABLE ROUTINE : SYS 49152 *
00110  C000              ;*****************************
00111  C000              ;
00112  C000  A9 4C             LDA #$4C         ;POINT TO MAIN ENTRY
00113  C002  85 73             STA CHRGET
00114  C004  A9 1A             LDA #<ENTRY      ;GET LOW ADDRESS
00115  C006  85 74             STA CHRGET+1     ;STORE IT
00116  C008  A9 C0             LDA #>ENTRY      ;GET HIGH ADDRESS
00117  C00A  85 75             STA CHRGET+2     ;STORE IT
00118  C00C  60                RTS              ;BACK TO BASIC
00119  C00D              ;
00120  C00D              ;*******************************
00121  C00D              ;* DISABLE ROUTINE : SYS 49165 *
00122  C00D              ;*******************************
00123  C00D              ;
00124  C00D              ;SYNTAX  @Z
00125  C00D              ;
00126  C00D  A9 E6       Z01    LDA #$E6
00127  C00F  85 73             STA CHRGET
00128  C011  A9 7A             LDA #CHRG7
00129  C013  85 74             STA CHRGET+1
```

```
00130  C015  A9 D0          LDA #$D0
00131  C017  85 75          STA CHRGET+2
00132  C019  60             RTS
00133  C01A                 ;
00134  C01A                 ;**************
00135  C01A                 ;* MAIN ENTRY *
00136  C01A                 ;**************
00137  C01A                 ;
00138  C01A  8E FF CF  ENTRY  STX POP        ;STORE IT
00139  C01D  BA               TSX            ;GET STACK POINTER
00140  C01E  BD 01 01         LDA $0101,X    ;GET CHAR
00141  C021  C9 8C            CMP #$8C       ;DIRECT MODE?
00142  C023  D0 10            BNE MAYBE      ;MAYBE
00143  C025  BD 02 01         LDA $0102,X    ;GET NEXT CHAR
00144  C028  C9 A4            CMP #$A4       ;PRG MODE
00145  C02A  D0 14            BNE NOT        ;NO
00146  C02C  E6 7A       OK   INC CHRG7      ;ADD 1 TO POINTER LOW
00147  C02E  D0 02            BNE NEXT1
00148  C030  E6 7B            INC CHRG8      ;ADD 1 TO POINTER HIGH
00149  C032                 ;
00150  C032  4C 4C C0  NEXT1  JMP KEY01       ;OK TO GO
00151  C035                 ;
00152  C035  C9 E6       MAYBE CMP #$E6
00153  C037  D0 07            BNE NOT
00154  C039  BD 02 01         LDA $0102,X
00155  C03C  C9 A7            CMP #$A7
00156  C03E  F0 EC            BEQ OK
00157  C040  AE FF CF  NOT    LDX POP        ;GET KEY BACK
00158  C043  E6 7A            INC CHRG7      ;ADD 1 TO POINTER LOW
00159  C045  D0 02            BNE LAST1
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE       LABEL  OPC OPERAND      COMMENT

00160  C047  E6 7B               INC CHRG8        ;ADD 1 TO POINTER HIGH
00161  C049                ;
00162  C049  4C 79 00     LAST1  JMP CHRGOT       ;REGET KEY
00163  C04C                ;
00164  C04C                ;****************
00165  C04C                ;* GET KEY CHAR *
00166  C04C                ;****************
00167  C04C                ;
00168  C04C  A0 00         KEY01  LDY #$00         ;SET INDEX TO 0
00169  C04E  B1 7A                LDA (CHRG7),Y    ;GET CHAR
00170  C050  C9 40                CMP #$40         ;START WITH '@'?
00171  C052  F0 03                BEQ KEY02        ;YES, CONTINUE
00172  C054  4C 79 00             JMP CHRGOT       ;NO, REGET CHAR
00173  C057                ;
00174  C057  E6 7A         KEY02  INC CHRG7        ;NEXT PLEASE
00175  C059  D0 02                BNE KEY03        ;CHRG7 < 0
00176  C05B  E6 7B                INC CHRG8        ;CARRY NEEDED
00177  C05D                ;
00178  C05D  B1 7A         KEY03  LDA (CHRG7),Y    ;GET NEXT CHAR
00179  C05F  E6 7A                INC CHRG7        ;COUNT ON
00180  C061  D0 02                BNE KEY04        ;CHRG7 STILL < 0
00181  C063  E6 7B                INC CHRG8        ;SO INCREMENT CHRG8
00182  C065                ;
00183  C065  C9 42         KEY04  CMP #$42         ;'B'?
00184  C067  D0 03                BNE KEY05        ;NO, KEEP SEARCHING
00185  C069  4C DF C0             JMP B01          ;GO TO 'B' ROUTINE
```

```
00186  C06C                    ;
00187  C06C  C9 43      KEY05  CMP #$43        ;'C'?
00188  C06E  D0 03             BNE KEY06       ;NO, KEEP SEARCHING
00189  C070  4C 3F C1          JMP C01         ;GO TO 'C' ROUTINE
00190  C073                    ;
00191  C073  C9 44      KEY06  CMP #$44        ;'D'?
00192  C075  D0 03             BNE KEY07       ;NO, KEEP SEARCHING
00193  C077  4C 6E C1          JMP D01         ;GO TO 'D' ROUTINE
00194  C07A                    ;
00195  C07A  C9 45      KEY07  CMP #$45        ;'E'?
00196  C07C  D0 03             BNE KEY08       ;NO, KEEP SEARCHING
00197  C07E  4C AF C1          JMP E01         ;GO TO 'E' ROUTINE
00198  C081                    ;
00199  C081  C9 46      KEY08  CMP #$46        ;'F'?
00200  C083  D0 03             BNE KEY09       ;NO, KEEP SEARCHING
00201  C085  4C F0 C1          JMP F01         ;GO TO 'F' ROUTINE
00202  C088                    ;
00203  C088  C9 49      KEY09  CMP #$49        ;'I'?
00204  C08A  D0 03             BNE KEY10       ;NO, KEEP SEARCHING
00205  C08C  4C 1D C2          JMP I01         ;GO TO 'I' ROUTINE
00206  C08F                    ;
00207  C08F  C9 4A      KEY10  CMP #$4A        ;'J'?
00208  C091  D0 03             BNE KEY11       ;NO, KEEP SEARCHING
00209  C093  4C 41 C2          JMP J01         ;GO TO 'J' ROUTINE
00210  C096                    ;
00211  C096  C9 4B      KEY11  CMP #$4B        ;'K'?
00212  C098  D0 03             BNE KEY12       ;NO, KEEP SEARCHING
00213  C09A  4C 70 C2          JMP K01         ;GO TO 'K' ROUTINE
00214  C09D                    ;
00215  C09D  C9 4C      KEY12  CMP #$4C        ;'L'?
```

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND     COMMENT

00216  C09F  D0 03              BNE KEY13       ;NO, KEEP SEARCHING
00217  C0A1  4C 81 C2           JMP L01         ;GO TO 'L' ROUTINE
00218  C0A4                ;
00219  C0A4  C9 4D        KEY13 CMP #$4D        ;'M'?
00220  C0A6  D0 03              BNE KEY14       ;NO, KEEP SEARCHING
00221  C0A8  4C BA C2           JMP M01         ;GO TO 'M' ROUTINE
00222  C0AB                ;
00223  C0AB  C9 4E        KEY14 CMP #$4E        ;'N'?
00224  C0AD  D0 03              BNE KEY15       ;NO, KEEP SEARCHING
00225  C0AF  4C E1 C2           JMP N01         ;GO TO 'N' ROUTINE
00226  C0B2                ;
00227  C0B2  C9 52        KEY15 CMP #$52        ;'R'?
00228  C0B4  D0 03              BNE KEY16       ;NO, KEEP SEARCHING
00229  C0B6  4C FB C2           JMP R01         ;GO TO 'R' ROUTINE
00230  C0B9                ;
00231  C0B9  C9 53        KEY16 CMP #$53        ;'S'?
00232  C0BB  D0 03              BNE KEY17       ;NO, KEEP SEARCHING
00233  C0BD  4C 3B C3           JMP S01         ;GO TO 'S' ROUTINE
00234  C0C0                ;
00235  C0C0  C9 54        KEY17 CMP #$54        ;'T'?
00236  C0C2  D0 03              BNE KEY18       ;NO, KEEP SEARCHING
00237  C0C4  4C 74 C3           JMP T01         ;GO TO 'T' ROUTINE
00238  C0C7                ;
00239  C0C7  C9 55        KEY18 CMP #$55        ;'U'?
00240  C0C9  D0 03              BNE KEY19       ;NO, KEEP SEARCHING
00241  C0CB  4C 89 C3           JMP U01         ;GO TO 'U' ROUTINE
```

```
00242  COCE                    ;
00243  COCE  C9 57       KEY19 CMP #$57        ;'W'?
00244  CODO  DO 03             BNE KEY20       ;NO, KEEP SEARCHING
00245  COD2  4C BC C3          JMP WO1         ;GO TO 'W' ROUTINE
00246  COD5                    ;
00247  COD5  C9 5A       KEY20 CMP #$5A        ;'Z'?
00248  COD7  FO 03             BEQ KEY99
00249  COD9  4C 08 AF          JMP SNERR       ;NO, SO GIVE UP
00250  CODC                    ;
00251  CODC  4C OD CO    KEY99 JMP ZO1         ;GO TO 'Z' ROUTINE
00252  CODF                    ;
00253  CODF                    ;*******************
00254  CODF                    ;* B = BAR DRAWING *
00255  CODF                    ;*******************
00256  CODF                    ;
00257  CODF                    ;SYNTAX  @B X,Y,C
00258  CODF                    ; X=O-39 Y=HEIGHT C=COLOUR
00259  CODF                    ;
00260  CODF  A9 06       BO1   LDA #$06        ;LOAD HI OF 19TH LINE
00261  COE1  85 4C             STA USR1        ;STORE IT
00262  COE3  20 9B B7          JSR GTBYTE      ;GET SCREEN POSITION X
00263  COE6  86 4D             STX USR2        ;STORE IT ALSO
00264  COE8  A9 F7             LDA #$F7        ;LOW FOR END OF 19TH LINE
00265  COEA  E5 4D             SBC USR2        ;WORK BACK FROM END
00266  COEC  85 4B             STA USRO        ;END OF LINE
00267  COEE  20 F1 B7          JSR COMBYT      ;EVALUATE BAR HEIGHT
00268  COF1  86 8C             STX INDEX1      ;STORE IT
00269  COF3  20 F1 B7          JSR COMBYT      ;GET COLOUR
00270  COF6  8E 86 02          STX COLOUR      ;AND STORE IT
00271  COF9                    ;
```

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND     COMMENT

00272  C0F9  A5 8C       B02    LDA INDEX1      ;RECALL HEIGHT
00273  C0FB  C9 08              CMP #$08        ;LESS THAN 8?
00274  C0FD  30 1A              BMI B03         ;YES, SO GO THERE
00275  C0FF  E9 08              SBC #$08        ;TAKE 8 AWAY
00276  C101  85 8C              STA INDEX1      ;STORE IT
00277  C103  A9 A0              LDA #$A0        ;LOAD A RVS SPACE
00278  C105  91 4B              STA (USR0),Y    ;STORE ON SCREEN
00279  C107  20 2D C1           JSR FIXCOL      ;STORE COLOUR TOO
00280  C10A  18                 CLC
00281  C10B  A5 4B              LDA USR0        ;GET SCREEN LOW BYTE
00282  C10D  E9 27              SBC #$27        ;BACK ONE SCREEN LINE
00283  C10F  85 4B              STA USR0        ;STORE IT
00284  C111  A5 4C              LDA USR1        ;GET SCREEN HIGH BYTE
00285  C113  E9 00              SBC #$00        ;ADJUST FOR CARRY
00286  C115  85 4C              STA USR1        ;STORE IT
00287  C117  D0 E0              BNE B02         ;BRANCH ALWAYS
00288  C119                 ;
00289  C119  AA           B03   TAX             ;GET FRACTION OF 8
00290  C11A  BD 25 C1           LDA B04,X       ;GET FRACTIONAL CHAR
00291  C11D  91 4B              STA (USR0),Y    ;PUT ONTO SCREEN
00292  C11F  20 2D C1           JSR FIXCOL
00293  C122  4C 73 00           JMP CHRGET      ;FINISHED - GET NEXT
00294  C125                 ;
00295  C125                 ;TABLE OF FRACTIONAL CHARACTERS
00296  C125                 ;TO CONSTRUCT VERTICAL BAR
00297  C125                 ;ACCURATE TO 1/8 OF 1 SCREEN
```

```
00298  C125                ;LETTER POSITION
00299  C125                ;
00300  C125  64            B04     .BYT $64,$6F,$79,$62
00300  C126  6F
00300  C127  79
00300  C128  62
00301  C129  F8                    .BYT $F8,$F7,$E3,$A0
00301  C12A  F7
00301  C12B  E3
00301  C12C  A0
00302  C12D                ;
00303  C12D  A9 DC         FIXCOL  LDA #$DC        ;CONVERT SCREEN
00304  C12F  45 4C                 EOR USR1        ;RAM TO COLOUR
00305  C131  85 4C                 STA USR1
00306  C133  AD 86 02              LDA COLOUR
00307  C136  91 4B                 STA (USR0),Y    ;SET THE COLOUR
00308  C138                ;
00309  C138  A9 DC         SWPCOL  LDA #$DC
00310  C13A  45 4C                 EOR USR1        ;AND BACK TO
00311  C13C  85 4C                 STA USR1        ;SCREEN RAM
00312  C13E  60                    RTS
00313  C13F                ;
00314  C13F                ;*****************************
00315  C13F                ;* C = CLEAR SCREEN SECTION *
00316  C13F                ;*****************************
00317  C13F                ;
00318  C13F                ;SYNTAX  @C X
00319  C13F                ; X=0-39
00320  C13F                ;
00321  C13F  20 9B B7      C01     JSR GTBYTE      ;GET PARAMETER
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE       LABEL  OPC OPERAND      COMMENT

00322  C142  A9 04              LDA #$04         ;SCREEN HIGH BYTE
00323  C144  85 4C              STA USR1         ;STORE IT
00324  C146  A9 27              LDA #$27         ;SCREEN WIDTH (0-39)
00325  C148  86 4D              STX USR2         ;STORE PARAMETER
00326  C14A  E5 4D              SBC USR2
00327  C14C  69 4F              ADC #$4F
00328  C14E  85 4B              STA USR0
00329  C150                ;
00330  C150  A9 20        CO2   LDA #$20         ;LOAD A SPACE
00331  C152  91 4B              STA (USR0),Y
00332  C154  18                 CLC
00333  C155  A5 4B              LDA USR0         ;SCREEN LOW BYTE
00334  C157  69 28              ADC #$28         ;ADD 1 SCREEN LINE
00335  C159  85 4B              STA USR0         ;STORE IT
00336  C15B  A5 4C              LDA USR1         ;FIX HIGH BYTE
00337  C15D  69 00              ADC #$00
00338  C15F  85 4C              STA USR1         ;STORE IT
00339  C161  C9 07              CMP #$07         ;FINISHED SCREEN YET?
00340  C163  D0 EB              BNE CO2          ;NO, SO GO BACK
00341  C165  A5 4B              LDA USR0         ;CHECK LOW BYTE
00342  C167  C9 20              CMP #$20
00343  C169  30 E5              BMI CO2          ;NO, GO BACK
00344  C16B  4C 73 00           JMP CHRGET       ;FINISH OFF GET NEXT
00345  C16E                ;
00346  C16E                ;**************************
00347  C16E                ;* D = SCROLL SCREEN DOWN *
00348  C16E                ;**************************
```

```
00349  C16E                ;
00350  C16E                ;SYNTAX  @D
00351  C16E                ;
00352  C16E                ;SET UP START AND END OF SCREEN TO BE MOVED
00353  C16E                ;SCREEN START IS AT LEFT END OF FIRST SCREEN LINE
00354  C16E                ;SCREEN END IS SECOND LAST SCREEN LINE
00355  C16E                ;ALL LINES MOVE DOWN ONE LINE
00356  C16E                ;TOP LINE IS FILLED WITH SPACES
00357  C16E                ;
00358  C16E  A0 28         DO1     LDY #$28        ;LOAD SCREEN END LOW BYTE
00359  C170  84 8E                 STY INDEX3      ;STORE IT
00360  C172  A0 00                 LDY #$00        ;LOAD LOW BYTE ON NEXT LINE UP
00361  C174  84 8C                 STY INDEX1      ;STORE IT
00362  C176  A0 BF                 LDY #$BF        ;SET INDEX
00363  C178  A2 07                 LDX #$07        ;OF 2ND LAST LINE
00364  C17A                ;
00365  C17A  86 8D         DO2     STX INDEX2      ;STORE IT
00366  C17C  86 8F                 STX INDEX4      ;TWICE
00367  C17E                ;
00368  C17E  B1 8C         DO3     LDA (INDEX1),Y  ;GET CHAR FROM LINE 'L'
00369  C180  91 8E                 STA (INDEX3),Y  ;MOVE IT TO LINE 'L+1'
00370  C182  20 A2 C1              JSR SWAPC2      ;SET COLOUR RAM
00371  C185  B1 8C                 LDA (INDEX1),Y
00372  C187  91 8E                 STA (INDEX3),Y
00373  C189  20 A2 C1              JSR SWAPC2      ;SET SCREEN RAM
00374  C18C  88                    DEY             ;COUNT INDEX DOWN
00375  C18D  C0 FF                 CPY #$FF        ;IS Y < 0?
00376  C18F  D0 ED                 BNE DO3         ;NO
00377  C191  CA                    DEX             ;COUNT MAIN DOWN
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00378  C192  E0 03              CPX #$03        ;RUN OFF SCREEN YET?
00379  C194  D0 E4              BNE D02         ;NO
00380  C196  A0 27              LDY #$27        ;SET COUNTER TO 1 LINE LENGTH
00381  C198  A9 20              LDA #$20        ;LOAD A SPACE
00382  C19A                ;
00383  C19A  91 8C       D04    STA (INDEX1),Y  ;STORE ON TOP LINE
00384  C19C  88                 DEY             ;COUNT DOWN INDEX
00385  C19D  10 FB              BPL D04         ;GO BACK FOR MORE
00386  C19F  4C 79 00           JMP CHRGOT      ;YES, SO FINISHED
00387  C1A2                ;
00388  C1A2  A9 DC       SWAPC2 LDA #$DC
00389  C1A4  45 8F              EOR INDEX4
00390  C1A6  85 8F              STA INDEX4
00391  C1A8                ;
00392  C1A8  A9 DC       SWAPC3 LDA #$DC
00393  C1AA  45 8D              EOR INDEX2
00394  C1AC  85 8D              STA INDEX2
00395  C1AE  60                 RTS
00396  C1AF                ;
00397  C1AF                ;*****************************
00398  C1AF                ;* E = READ SEQUENTIAL FILE *
00399  C1AF                ;*****************************
00400  C1AF                ;
00401  C1AF                ;SYNTAX  @E "FILENAME"
00402  C1AF                ; SHIFT KEY PAUSES SCROLLING
00403  C1AF                ; RUN/STOP EXITS ROUTINE
```

```
00404  C1AF                    ;
00405  C1AF  A9 02      E01    LDA #$02        ;SETUP OPEN
00406  C1B1  A2 08             LDX #$08        ;TO DISK
00407  C1B3  A8                TAY             ;AS 'OPEN 2,8,2'
00408  C1B4  20 BA FF          JSR SETLFS
00409  C1B7  20 0D C4          JSR GETSTR      ;GET FILENAME
00410  C1BA  98                TYA             ;LENGTH OF FN
00411  C1BB  A2 00             LDX #<STRBUF    ;POINTER TO
00412  C1BD  A0 CF             LDY #>STRBUF    ;FILENAME
00413  C1BF  20 BD FF          JSR SETNAM
00414  C1C2  20 C0 FF          JSR OPEN        ;OPEN 2,8,2,FN
00415  C1C5  A2 02             LDX #$02
00416  C1C7  20 C6 FF          JSR CHKIN       ;OPEN CHANNEL
00417  C1CA                    ;
00418  C1CA  20 CF FF    E02   JSR INCHR       ;GET CHR FROM FILE
00419  C1CD  20 D2 FF          JSR OUTCH       ;PRINT IT
00420  C1D0  24 90             BIT SATUS       ;EOF ?
00421  C1D2  70 11             BVS E04         ;YES
00422  C1D4  C9 0D             CMP #$0D        ;END OF LINE?
00423  C1D6  D0 F2             BNE E02
00424  C1D8                    ;
00425  C1D8  AD 8D 02    E03   LDA SFST        ;SHIFT KEY
00426  C1DB  29 01             AND #$01        ;NOT CTRL KEY
00427  C1DD  D0 F9             BNE E03         ;LOOP, AND LOOP...
00428  C1DF  A5 C5             LDA LSTX        ;CHECK RUN/STOP
00429  C1E1  C9 3F             CMP #$3F
00430  C1E3  D0 E5             BNE E02         ;NOT PRESSED - GO ON
00431  C1E5                    ;
00432  C1E5  20 CC FF    E04   JSR CLRCHN      ;BACK TO KEYBOARD
00433  C1E8  A9 02             LDA #$02
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00434  C1EA  20 C3 FF           JSR CLOSE       ;CLOSE 2 AND EXIT
00435  C1ED  4C 73 00           JMP CHRGET
00436  C1F0                ;
00437  C1F0                ;**********************************
00438  C1F0                ;* F = FILL SCREEN WITH CHARACTER *
00439  C1F0                ;**********************************
00440  C1F0                ;
00441  C1F0                ;SYNTAX  @F CH,C
00442  C1F0                ; CH IS SCREEN POKE OF CHARACTER
00443  C1F0                ; C IS SCREEN COLOUR OF CHARACTER
00444  C1F0                ;
00445  C1F0  20 9B B7      F01    JSR GTBYTE       ;GET CHARACTER REQUIRED
00446  C1F3  86 4B                STX USR0         ;STORE IT
00447  C1F5  20 F1 B7             JSR COMBYT
00448  C1F8  8E 86 02             STX COLOUR
00449  C1FB  A0 00                LDY #$00         ;LOAD SCREEN LOW BYTE
00450  C1FD  84 8C                STY INDEX1       ;STORE IT
00451  C1FF  A2 04                LDX #$04         ;LOAD SCREEN HIGH BYTE
00452  C201  A5 4B                LDA USR0         ;GET CHARACTER BACK
00453  C203                ;
00454  C203  86 8D         F02    STX INDEX2       ;STORE SCREEN HIGH BYTE
00455  C205                ;
00456  C205  91 8C         F03    STA (INDEX1),Y   ;STORE CHAR ON SCREEN
00457  C207  20 A8 C1             JSR SWAPC3       ;SET TO COLOUR RAM
00458  C20A  AD 86 02             LDA COLOUR
00459  C20D  91 8C                STA (INDEX1),Y   ;STORE COLOUR
```

```
00460  C20F  20 A8 C1          JSR SWAPC3      ;BACK TO SCREEN RAM
00461  C212  A5 4B             LDA USR0        ;BACK TO FILL CHR
00462  C214  C8                INY             ;INCREMENT LOW BYTE
00463  C215  D0 EE             BNE F03         ;WITHIN RANGE
00464  C217  E8                INX             ;INCREMENT HIGH BYTE
00465  C218  E0 08             CPX #$08        ;END OF SCREEN YET?
00466  C21A  D0 E7             BNE F02         ;NO, GO BACK
00467  C21C  60                RTS             ;YES, SO FINISHED
00468  C21D                  ;
00469  C21D                  ;*********************
00470  C21D                  ;* I = INVERT SCREEN *
00471  C21D                  ;*********************
00472  C21D                  ;
00473  C21D                  ;SYNTAX  @I C
00474  C21D                  ; C IS SCREEN COLOUR
00475  C21D                  ;
00476  C21D  A2 04           I01     LDX #$04        ;LOAD SCREEN HIGH BYTE
00477  C21F  86 4C                   STX USR1
00478  C221  A0 00                   LDY #$00        ;LOAD SCREEN LOW BYTE
00479  C223  84 4B                   STY USR0        ;STORE IT
00480  C225  20 9B B7                JSR GTBYTE      ;GET COLOUR
00481  C228  8E 86 02                STX COLOUR      ;STORE IT
00482  C22B  A6 4C                   LDX USR1        ;RE-GET HI SCREEN
00483  C22D                  ;
00484  C22D  B1 4B           I03     LDA (USR0),Y    ;GET CHAR FROM SCREEN
00485  C22F  49 80                   EOR #$80        ;REVERSE BIT 7
00486  C231  91 4B                   STA (USR0),Y    ;REPLACE IT
00487  C233  20 2D C1                JSR FIXCOL
00488  C236  C8                      INY             ;INCREMENT LOW BYTE
00489  C237  D0 F4                   BNE I03
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE       LABEL  OPC OPERAND      COMMENT

00490  C239  E8                   INX             ;INCREMENT HIGH BYTE
00491  C23A  86 4C                STX USR1
00492  C23C  E0 08                CPX #$08        ;FINISHED SCREEN YET?
00493  C23E  D0 ED                BNE I03         ;NO, GO BACK FOR MORE
00494  C240  60                   RTS             ;YES, SO FINISHED
00495  C241                ;
00496  C241                ;*******************************
00497  C241                ;* J = CENTRE STRING ON SCREEN *
00498  C241                ;*******************************
00499  C241                ;
00500  C241                ;SYNTAX  @J "MESSAGE"
00501  C241                ;
00502  C241  20 0D C4      J01    JSR GETSTR      ;GET MESSAGE
00503  C244  A9 28                LDA #$28        ;LINE LENGTH
00504  C246  84 4D                STY USR2        ;STRING LENGTH
00505  C248  E5 4D                SBC USR2        ;40-LENGTH (CARRY'S SET)
00506  C24A  85 4D                STA USR2        ;STORE IT
00507  C24C                ;
00508  C24C  A9 20        J02     LDA #$20        ;LOAD A SPACE
00509  C24E  20 D2 FF             JSR OUTCH       ;PRINT IT
00510  C251  C6 4D                DEC USR2        ;TAKE 2 AWAY FOR
00511  C253  C6 4D                DEC USR2        ;EVERY SPACE PRINTED
00512  C255  A5 4D                LDA USR2        ;CHECK ITS VALUE
00513  C257  C9 01                CMP #$01        ;IS IT 1 YET?
00514  C259  10 F1                BPL J02         ;NO, SO PRINT MORE SPACES
00515  C25B  A0 00                LDY #$00        ;SET INDEX
```

```
00516  C25D                    ;
00517  C25D  B9 00 CF   JO4     LDA STRBUF,Y     ;GET STRING CHAR
00518  C260  F0 06              BEQ JO5          ;DONE!
00519  C262  20 D2 FF           JSR OUTCH        ;PRINT STRING CHAR
00520  C265  C8                 INY              ;INCREASE POINTER
00521  C266  D0 F5              BNE JO4          ;ALWAYS
00522  C268                    ;
00523  C268  A9 0D      JO5     LDA #$0D
00524  C26A  20 D2 FF           JSR OUTCH
00525  C26D  4C 73 00           JMP CHRGET
00526  C270                    ;
00527  C270                    ;*******************
00528  C270                    ;* K = KILL REPEAT *
00529  C270                    ;*******************
00530  C270                    ;
00531  C270                    ;SYNTAX  @K
00532  C270                    ;
00533  C270  A9 31      KO1     LDA #<KEY        ;IRQ LOW BYTE
00534  C272  8D 14 03           STA CINV1        ;STORE IT
00535  C275  A9 EA              LDA #>KEY        ;IRQ HIGH BYTE
00536  C277  8D 15 03           STA CINV2        ;STORE IT
00537  C27A                    ;
00538  C27A  A9 10      KO2     LDA #$10
00539  C27C  85 FC              STA TEMP1
00540  C27E  4C 79 00           JMP CHRGOT       ;AND OUT AGAIN
00541  C281                    ;
00542  C281                    ;*****************************
00543  C281                    ;* L = LOAD SCREEN FROM RAM *
00544  C281                    ;*****************************
00545  C281                    ;
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00546  C281              ;SYNTAX  @L
00547  C281              ;
00548  C281              ;THIS ROUTINE TAKES ONE SCREENFUL
00549  C281              ;OF CHARACTERS FROM MEMORY
00550  C281              ;AND LOADS THEM ONTO THE SCREEN.
00551  C281              ;THE CHARACTERS MAY HAVE BEEN
00552  C281              ;PREVIOUSLY PLACED IN RAM WITH THE
00553  C281              ;'S' ROUTINE.  THE SCREEN CONTENTS
00554  C281              ;ARE LOST, AND THE SCREENFUL OF
00555  C281              ;CHARACTERS REMAINS IN RAM.
00556  C281              ;
00557  C281              ;FORM PAIRS OF BYTES TO DESCRIBE SCREEN
00558  C281              ;
00559  C281  A2 04       LO1    LDX #$04        ;SCREEN HIGH BYTE
00560  C283  A0 00              LDY #$00        ;SCREEN LOW BYTE
00561  C285  84 8C              STY INDEX1      ;STORE
00562  C287  84 8E              STY INDEX3      ;LOW BYTES
00563  C289  A9 C3              LDA #$C3        ;START ADDRESS-2 BLOCKS
00564  C28B  85 8F              STA INDEX4
00565  C28D              ;
00566  C28D  86 8D       LO2    STX INDEX2      ;STORE HIGH BYTE
00567  C28F  E6 8F              INC INDEX4      ;SKIP LAST
00568  C291  E6 8F              INC INDEX4      ;TWO BLOCKS
00569  C293  A9 00              LDA #$00        ;LOAD SCREEN STORE LOW BYTE
00570  C295  85 8E              STA INDEX3      ;STORE IT
00571  C297              ;
```

```
00572  C297                ;(INDEX3) POINTS TO $C500
00573  C297                ;
00574  C297                ;NOW TO TRANSFER CONTENTS TO SCREEN
00575  C297                ;
00576  C297  B1 8E         LO3    LDA (INDEX3),Y  ;GET A STORED CHAR
00577  C299  91 8C                STA (INDEX1),Y  ;PUT ONTO SCREEN
00578  C29B  20 A8 C1              JSR SWAPC3      ;POINT TO COLOUR RAM
00579  C29E  E6 8F                 INC INDEX4      ;AND NEXT BLOCK
00580  C2A0  B1 8E                 LDA (INDEX3),Y  ;SAVE COLOUR
00581  C2A2  91 8C                 STA (INDEX1),Y  ;RAM AND
00582  C2A4  20 A8 C1              JSR SWAPC3      ;BACK TO SCREEN
00583  C2A7  C6 8F                 DEC INDEX4      ;AND ORIGINAL BLOCK
00584  C2A9  C8                    INY             ;INCREASE COUNT
00585  C2AA  D0 EB                 BNE LO3         ;BLOCK NOT MOVED
00586  C2AC  E8                    INX             ;ADD 1 TO HIGH BYTE
00587  C2AD  E0 08                 CPX #$08        ;FINISHED SCREEN YET?
00588  C2AF  D0 DC                 BNE LO2         ;NO, GO BACK FOR MORE
00589  C2B1  AD FE CF              LDA COLTMP+1
00590  C2B4  8D 21 D0              STA SCCOL
00591  C2B7  4C 79 00              JMP CHRGOT
00592  C2BA                ;
00593  C2BA                ;************************
00594  C2BA                ;* M = REPLACE CHARACTER *
00595  C2BA                ;************************
00596  C2BA                ;
00597  C2BA                ;SYNTAX  @M A,B
00598  C2BA                ; A IS PRESENT CHAR ON SCREEN
00599  C2BA                ; B IS REPLACEMENT CHAR ON SCREEN
00600  C2BA                ;(CHARACTERS ARE SCREEN POKE VALUES)
00601  C2BA                ;
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00602  C2BA  20 9B B7    M01    JSR GTBYTE       ;GET OLD CHAR
00603  C2BD  86 4C              STX USR1         ;STORE IT
00604  C2BF  20 F1 B7           JSR COMBYT       ;GET NEW CHAR
00605  C2C2  86 4B              STX USR0         ;STORE IT
00606  C2C4  A0 00              LDY #$00         ;SCREEN LOW BYTE
00607  C2C6  84 8C              STY INDEX1       ;STORE IT
00608  C2C8  A2 04              LDX #$04         ;SCREEN HIGH BYTE
00609  C2CA  A5 4B              LDA USR0
00610  C2CC                     ;
00611  C2CC  86 8D       M02    STX INDEX2       ;STORE HIGH BYTE
00612  C2CE                     ;
00613  C2CE  B1 8C       M03    LDA (INDEX1),Y   ;GET CHAR FROM SCREEN
00614  C2D0  C5 4C              CMP USR1         ;WANT THIS ONE?
00615  C2D2  D0 04              BNE M04          ;NO, SO BRANCH
00616  C2D4  A5 4B              LDA USR0         ;GET NEW CHAR
00617  C2D6  91 8C              STA (INDEX1),Y   ;STORE IT
00618  C2D8                     ;
00619  C2D8  C8          M04    INY              ;INCREASE COUNTER
00620  C2D9  D0 F3              BNE M03
00621  C2DB  E8                 INX
00622  C2DC  E0 08              CPX #$08         ;FINISHED SCREEN YET?
00623  C2DE  D0 EC              BNE M02          ;NO
00624  C2E0  60                 RTS              ;YES, SO FINISHED
00625  C2E1                     ;
00626  C2E1                     ;*****************************
00627  C2E1                     ;* N = DEC TO HEX CONVERSION *
00628  C2E1                     ;*****************************
```

```
00629  C2E1                    ;
00630  C2E1                    ;SYNTAX  @N X
00631  C2E1                    ; X = 0 -> 65535
00632  C2E1                    ;
00633  C2E1  20 9E AD    NO1     JSR FRMNUM      ;EVALUATE NUMERIC EXPR.
00634  C2E4  20 F7 B7            JSR GETADR      ;CONVERT TO 2-BYTE INTEGER
00635  C2E7  48                  PHA             ;STORE IT
00636  C2E8  A9 24               LDA #$24        ;'$'
00637  C2EA  20 D2 FF            JSR OUTCH       ;PRINT IT
00638  C2ED  68                  PLA             ;GET IT BACK
00639  C2EE  20 3A C4            JSR WROB        ;OUTPUT 1 BYTE
00640  C2F1  98                  TYA             ;GET SECOND BYTE
00641  C2F2  20 3A C4            JSR WROB        ;OUTPUT 1 BYTE
00642  C2F5  A9 0D               LDA #$0D
00643  C2F7  20 D2 FF            JSR OUTCH
00644  C2FA  60                  RTS             ;FINISHED
00645  C2FB                    ;
00646  C2FB                    ;******************
00647  C2FB                    ;* R = REPEAT KEY *
00648  C2FB                    ;******************
00649  C2FB                    ;
00650  C2FB                    ;SYNTAX  @R
00651  C2FB                    ;
00652  C2FB  78          RO1     SEI
00653  C2FC  A9 0E               LDA #<RO2       ;POINT IRQ TO
00654  C2FE  8D 14 03            STA CINV1       ;REPEAT KEY ROUTINE
00655  C301  A9 C3               LDA #>RO2
00656  C303  8D 15 03            STA CINV2
00657  C306  A9 01               LDA #$01
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND     COMMENT

00658  C308  85 4E              STA REPDL
00659  C30A  58                 CLI
00660  C30B  4C 7A C2           JMP KO2         ;TO FINISH OFF
00661  C30E                ;
00662  C30E  A5 C5        RO2    LDA LSTX        ;GET A KEY
00663  C310  C5 FE              CMP TEMP3       ;SAME AS LAST KEY?
00664  C312  F0 09              BEQ RO4         ;YES
00665  C314  85 FE              STA TEMP3       ;STORE IT
00666  C316  A9 10              LDA #$10        ;SET DELAY
00667  C318  85 FD              STA TEMP2       ;STORE IT
00668  C31A                ;
00669  C31A  4C 31 EA     RO3    JMP KEY         ;FINISHED JOB
00670  C31D                ;
00671  C31D  C9 FF        RO4    CMP #$FF
00672  C31F  F0 F9              BEQ RO3
00673  C321  A5 FD              LDA TEMP2       ;RECALL DELAY
00674  C323  F0 04              BEQ RO5         ;REACHED 0
00675  C325  C6 FD              DEC TEMP2       ;DECREASE COUNTER
00676  C327  D0 F1              BNE RO3
00677  C329                ;
00678  C329  C6 4E        RO5    DEC REPDL
00679  C32B  D0 ED              BNE RO3
00680  C32D  A9 04              LDA #$04
00681  C32F  85 4E              STA REPDL
00682  C331  A9 40              LDA #$40        ;NIL KEY
00683  C333  85 C5              STA LSTX        ;STORE IN KEY REGISTER
```

```
00684  C335  A9 02          LDA #$02
00685  C337  85 CD          STA BLNCT       ;STORE IN CURSOR BLINK COUNTER
00686  C339  D0 DF          BNE R03         ;BRANCH EVERY TIME
00687  C33B                ;
00688  C33B                ;*************************
00689  C33B                ;* S = SAVE SCREEN IN RAM *
00690  C33B                ;*************************
00691  C33B                ;
00692  C33B                ;SYNTAX  @S
00693  C33B                ;
00694  C33B                ;THIS ROUTINE TAKES THE SCREEN CONTENTS
00695  C33B                ;AND MOVES THEM TO A LOCATION IN RAM
00696  C33B                ;STARTING AT $C500, FROM WHERE THEY
00697  C33B                ;CAN BE RECALLED WITH '@L' OR SWAPPED
00698  C33B                ;WITH '@W'.
00699  C33B                ;
00700  C33B  AD 21 D0      S01     LDA SCCOL       ;REMEMBER SCREEN
00701  C33E  8D FE CF              STA COLTMP+1    ;COLOUR
00702  C341  A2 04                 LDX #$04        ;SCREEN HIGH BYTE
00703  C343  A0 00                 LDY #$00        ;SCREEN LOW BYTE
00704  C345  84 8C                 STY INDEX1      ;STORE
00705  C347  84 8E                 STY INDEX3      ;LOW BYTES
00706  C349  A9 C3                 LDA #$C3        ;START ADDRESS-2 BLOCKS
00707  C34B  85 8F                 STA INDEX4
00708  C34D                ;
00709  C34D  86 8D         S02     STX INDEX2      ;STORE HIGH BYTE
00710  C34F  E6 8F                 INC INDEX4      ;SKIP LAST
00711  C351  E6 8F                 INC INDEX4      ;TWO BLOCKS
00712  C353  A9 00                 LDA #$00        ;LOAD SCREEN STORE LOW BYTE
00713  C355  85 8E                 STA INDEX3      ;STORE IT
```

C64 DEMONSTRATION

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00714  C357              ;
00715  C357              ;(INDEX3) POINTS TO $C500
00716  C357              ;
00717  C357              ;NOW TO TRANSFER CONTENTS FROM SCREEN
00718  C357              ;
00719  C357  B1 8C       S03    LDA (INDEX1),Y  ;GET A STORED CHAR
00720  C359  91 8E              STA (INDEX3),Y  ;PUT INTO RAM
00721  C35B  20 A8 C1           JSR SWAPC3      ;POINT TO COLOUR RAM
00722  C35E  E6 8F              INC INDEX4      ;AND NEXT BLOCK
00723  C360  B1 8C              LDA (INDEX1),Y  ;SAVE COLOUR
00724  C362  91 8E              STA (INDEX3),Y  ;RAM AND
00725  C364  20 A8 C1           JSR SWAPC3      ;BACK TO SCREEN
00726  C367  C6 8F              DEC INDEX4      ;AND ORIGINAL BLOCK
00727  C369  C8                 INY             ;INCREASE COUNT
00728  C36A  D0 EB              BNE S03         ;BLOCK NOT MOVED
00729  C36C  E8                 INX             ;ADD 1 TO HIGH BYTE
00730  C36D  E0 08              CPX #$08        ;FINISHED SCREEN YET?
00731  C36F  D0 DC              BNE S02         ;NO, GO BACK FOR MORE
00732  C371  4C 79 00           JMP CHRGOT
00733  C374              ;
00734  C374              ;***************************
00735  C374              ;* T = SET CURSOR LOCATION *
00736  C374              ;***************************
00737  C374              ;
00738  C374              ;SYNTAX  @T X,Y
00739  C374              ; X IS HORIZONTAL POSITION (COLUMN)
```

```
00740  C374                    ; Y IS VERTICAL POSITION (LINE)
00741  C374                    ;
00742  C374  20 9B B7    TO1      JSR GTBYTE       ;GET X
00743  C377  86 4B                 STX USRO         ;STORE IT
00744  C379  20 F1 B7              JSR COMBYT       ;GET Y
00745  C37C  86 4C                 STX USR1         ;STORE IT
00746  C37E  A6 4B                 LDX USRO         ;GET X
00747  C380  A4 4C                 LDY USR1         ;GET Y
00748  C382  18                    CLC              ;SETUP FOR KERNAL
00749  C383  20 F0 FF              JSR PLOT         ;SET CURSOR & EXIT
00750  C386  4C 73 00              JMP CHRGET
00751  C389                    ;
00752  C389                    ;*****************************
00753  C389                    ;* U = MOVE SCREEN UP 1 LINE *
00754  C389                    ;*****************************
00755  C389                    ;
00756  C389                    ;SYNTAX  @U
00757  C389                    ;
00758  C389                    ;ROUTINE STARTS WITH FIRST CHAR ON
00759  C389                    ;SECOND LINE OF SCREEN, MOVING ALL
00760  C389                    ;LINES UP ONE.  THE BOTTOM LINE IS THEN
00761  C389                    ;FILLED WITH SPACES.
00762  C389                    ;
00763  C389  A2 00        UO1      LDX #$00         ;LINE 1 SCREEN LOW BYTE
00764  C38B  86 8E                 STX INDEX3       ;STORE IT
00765  C38D  A2 28                 LDX #$28         ;LINE 2 SCREEN LOW BYTE
00766  C38F  86 8C                 STX INDEX1       ;STORE IT
00767  C391  A2 04                 LDX #$04         ;SCREEN HIGH BYTE
00768  C393                    ;
00769  C393  86 8F        UO2      STX INDEX4       ;STORE IT
```

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00770  C395  86 8D              STX INDEX2       ;.. IN BOTH PLACES
00771  C397  A0 00              LDY #$00         ;SET COUNTER
00772  C399              ;
00773  C399  B1 8C       U03    LDA (INDEX1),Y   ;GET CHAR
00774  C39B  91 8E              STA (INDEX3),Y   ;MOVE UP 1 LINE
00775  C39D  20 A2 C1           JSR SWAPC2
00776  C3A0  B1 8C              LDA (INDEX1),Y
00777  C3A2  91 8E              STA (INDEX3),Y
00778  C3A4  20 A2 C1           JSR SWAPC2
00779  C3A7  C8                 INY
00780  C3A8  D0 EF              BNE U03
00781  C3AA  E8                 INX
00782  C3AB  E0 08              CPX #$08         ;FINISHED SCREEN YET
00783  C3AD  D0 E4              BNE U02          ;NO, SO LOOP BACK
00784  C3AF  A9 20              LDA #$20         ;YES, SO LOAD A SPACE
00785  C3B1  A2 28              LDX #$28         ;SET A COUNTER
00786  C3B3              ;
00787  C3B3  9D C7 07    U04    STA $07C7,X      ;ON LAST SCREEN LINE
00788  C3B6  CA                 DEX              ;COUNT DOWN
00789  C3B7  D0 FA              BNE U04          ;NOT DONE YET
00790  C3B9  4C 79 00           JMP CHRGOT
00791  C3BC              ;
00792  C3BC              ;**********************************
00793  C3BC              ;* W = SWAP SCREEN WITH RAM STORE *
00794  C3BC              ;**********************************
00795  C3BC              ;
```

```
00796  C3BC                ;SYNTAX  @W
00797  C3BC                ;
00798  C3BC                ;THIS ROUTINE DOES A BYTE BY BYTE SWAP
00799  C3BC                ;OF SCREEN CONTENTS WITH THE CONTENTS
00800  C3BC                ;OF RAM - MAYBE ANOTHER SCREEN STORED
00801  C3BC                ;THERE.  THE SWAP IS A 3-MOVE S/RTNE
00802  C3BC                ;SCREEN BYTE TO TEMP STORE,
00803  C3BC                ;RAM TO SCREEN, THEN TEMP TO RAM.
00804  C3BC                ;
00805  C3BC  AD 21 D0      WO1     LDA SCCOL
00806  C3BF  8D FD CF              STA COLTMP
00807  C3C2  A2 04                 LDX #$04
00808  C3C4  A0 00                 LDY #$00
00809  C3C6  84 8C                 STY INDEX1
00810  C3C8  84 8E                 STY INDEX3
00811  C3CA  A9 C3                 LDA #$C3
00812  C3CC  85 8F                 STA INDEX4
00813  C3CE                ;
00814  C3CE  86 8D         WO2     STX INDEX2
00815  C3D0  E6 8F                 INC INDEX4
00816  C3D2  E6 8F                 INC INDEX4
00817  C3D4                ;
00818  C3D4  B1 8E         WO3     LDA (INDEX3),Y  ;GET FROM RAM
00819  C3D6  85 4B                 STA USR0        ;STASH IT SAFELY
00820  C3D8  B1 8C                 LDA (INDEX1),Y  ;GET FROM SCREEN
00821  C3DA  91 8E                 STA (INDEX3),Y  ;PUT IN RAM
00822  C3DC  A5 4B                 LDA USR0
00823  C3DE  91 8C                 STA (INDEX1),Y  ;PUT ON SCREEN
00824  C3E0  20 A8 C1              JSR SWAPC3      ;POINT TO COLOUR RAM
00825  C3E3  E6 8F                 INC INDEX4      ;POINT TO COPY
```

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND      COMMENT

00826  C3E5  B1 8E              LDA (INDEX3),Y  ;GET FROM RAM COPY
00827  C3E7  85 4B              STA USRO        ;SAFE FOR SWAP
00828  C3E9  B1 8C              LDA (INDEX1),Y  ;GET THE REAL THING
00829  C3EB  91 8E              STA (INDEX3),Y  ;COPY TO RAM
00830  C3ED  A5 4B              LDA USRO
00831  C3EF  91 8C              STA (INDEX1),Y  ;PUT IN COLOUR RAM
00832  C3F1  C6 8F              DEC INDEX4      ;POINT BACK TO SCREEN COPY
00833  C3F3  20 A8 C1           JSR SWAPC3      ;BACK TO SCREEN
00834  C3F6  C8                 INY
00835  C3F7  D0 DB              BNE W03
00836  C3F9  E8                 INX
00837  C3FA  E0 08              CPX #$08
00838  C3FC  D0 D0              BNE W02
00839  C3FE  AD FE CF           LDA COLTMP+1
00840  C401  8D 21 D0           STA SCCOL
00841  C404  AD FD CF           LDA COLTMP
00842  C407  8D FE CF           STA COLTMP+1
00843  C40A  4C 79 00           JMP CHRGOT
00844  C40D                    ;
00845  C40D                    ;************************
00846  C40D                    ;* GET STRING IN QUOTES *
00847  C40D                    ;************************
00848  C40D                    ;
00849  C40D  20 73 00    GETSTR JSR CHRGET      ;SCAN FOR
00850  C410  C9 22              CMP #$22        ;FIRST QUOTE
00851  C412  F0 03              BEQ STRTOK
```

```
00852  C414  4C 08 AF    JUNK   JMP SNERR        ;UNREADABLE
00853  C417               ;
00854  C417  A0 01       STRTOK LDY #$01
00855  C419               ;
00856  C419  B1 7A       STASH  LDA (CHRG7),Y    ;GET STRING
00857  C41B  F0 F7              BEQ JUNK         ;END OF LINE ?
00858  C41D  C9 22              CMP #$22         ;END QUOTE ?
00859  C41F  F0 06              BEQ ENDSTR       ;YES - FINISH UP
00860  C421  99 FF CE           STA STRBUF-1,Y   ;STORE IN BUFFER
00861  C424  C8                 INY
00862  C425  D0 F2              BNE STASH        ;ALWAYS BACK FOR MORE
00863  C427               ;
00864  C427  18           ENDSTR CLC
00865  C428  A9 00              LDA #$00         ;END-OF-STRING POINTER
00866  C42A  99 FF CE           STA STRBUF-1,Y
00867  C42D  98                 TYA
00868  C42E  65 7A              ADC CHRG7        ;SET CHRGOT
00869  C430  85 7A              STA CHRG7
00870  C432  A9 00              LDA #$00         ;TO END OF
00871  C434  65 7B              ADC CHRG8
00872  C436  85 7B              STA CHRG8        ;LINE
00873  C438  88                 DEY              ;LENGTH IS ONE LESS
00874  C439  60                 RTS
00875  C43A               ;
00876  C43A               ;********************
00877  C43A               ;* OUTPUT HEX BYTE *
00878  C43A               ;********************
00879  C43A                                       ;
00880  C43A  48           WROB   PHA              ;SAVE LOW BITS
00881  C43B  4A                  LSR A            ;GET UPPER BITS
```

```
LINE#  LOCN  M/CODE      LABEL  OPC OPERAND     COMMENT

00882  C43C  4A                  LSR A
00883  C43D  4A                  LSR A
00884  C43E  4A                  LSR A
00885  C43F  C9 0A               CMP #$0A        ;AN ALPHA-HEX CHR ?
00886  C441  90 02               BCC NUM1        ;NO, JUST A NUMBER
00887  C443  69 06               ADC #$06        ;ADD 7, AS CARRY'S SET
00888  C445               ;
00889  C445  69 30        NUM1   ADC #$30        ;BUT NOW IT'S CLEAR
00890  C447  20 D2 FF             JSR OUTCH       ;PRINT IT
00891  C44A  68                  PLA             ;GET LOW NYBBLE
00892  C44B  29 0F               AND #$0F        ;MASK OFF HIGH BITS
00893  C44D  C9 0A               CMP #$0A        ;CHECK FOR ALPHA AGAIN
00894  C44F  90 02               BCC NUM2
00895  C451  69 06               ADC #$06
00896  C453               ;
00897  C453  69 30        NUM2   ADC #$30
00898  C455  4C D2 FF             JMP OUTCH       ;PRINT & EXIT
00899  C458               ;
00900  C458               ;---SET ASIDE A STRING BUFFER--
00901  C458               ;
00902  C458               STRBUF =$CF00
00903  C458               ;
00904  C458                       .END

ERRORS = 00000
```

LABEL INDEX

LABEL	LOCN						
B01	C0DF	B02	C0F9	B03	C119	B04	C125
BLNCT	00CD	C01	C13F	C02	C150	CHKIN	FFC6
CHRG7	007A	CHRG8	007B	CHRGET	0073	CHRGOT	0079
CINV1	0314	CINV2	0315	CLOSE	FFC3	CLRCHN	FFCC
COLOUR	0286	COLTMP	CFFD	COMBYT	B7F1	D01	C16E
D02	C17A	D03	C17E	D04	C19A	E01	C1AF
E02	C1CA	E03	C1D8	E04	C1E5	ENDSTR	C427
ENTRY	C01A	F01	C1F0	F02	C203	F03	C205
FIXCOL	C12D	FRMNUM	AD9E	GETADR	B7F7	GETSTR	C40D
GTBYTE	B79B	I01	C21D	I03	C22D	INCHR	FFCF
INDEX1	008C	INDEX2	008D	INDEX3	008E	INDEX4	008F
J01	C241	J02	C24C	J04	C25D	J05	C268
JUNK	C414	K01	C270	K02	C27A	KEY	EA31
KEY01	C04C	KEY02	C057	KEY03	C05D	KEY04	C065
KEY05	C06C	KEY06	C073	KEY07	C07A	KEY08	C081
KEY09	C088	KEY10	C08F	KEY11	C096	KEY12	C09D
KEY13	C0A4	KEY14	C0AB	KEY15	C0B2	KEY16	C0B9
KEY17	C0C0	KEY18	C0C7	KEY19	C0CE	KEY20	C0D5
KEY99	C0DC	L01	C281	L02	C28D	L03	C297
LAST1	C049	LSTX	00C5	M01	C2BA	M02	C2CC
M03	C2CE	M04	C2D8	MAYBE	C035	N01	C2E1
NEXT1	C032	NOT	C040	NUM1	C445	NUM2	C453
OK	C02C	OPEN	FFC0	OUTCH	FFD2	PLOT	FFF0
POP	CFFF	R01	C2FB	R02	C30E	R03	C31A

LABEL INDEX

LABEL	LOCN						
R04	C31D	R05	C329	READY	A474	REPDL	004E
S01	C33B	S02	C34D	S03	C357	SATUS	0090
SCCOL	D021	SCREEN	0400	SETLFS	FFBA	SETNAM	FFBD
SFST	028D	SNERR	AF08	STASH	C419	STRBUF	CF00
STRTOK	C417	SWAPC2	C1A2	SWAPC3	C1A8	SWPCOL	C138
T01	C374	TEMP1	00FC	TEMP2	00FD	TEMP3	00FE
U01	C389	U02	C393	U03	C399	U04	C3B3
USR0	004B	USR1	004C	USR2	004D	W01	C3BC
W02	C3CE	W03	C3D4	WROB	C43A	Z01	C00D

END OF ASSEMBLY

COMMODORE 64
MACHINE LANGUAGE TUTORIAL

WHAT COMES AFTER BASIC:

BASIC has become the universal computer language for home and personal computers. To the computerist it soon becomes apparent that speed is BASIC's main drawback.

Thats where MACHINE LANGUAGE (ML) comes in. By removing the intermediate interpretation of step BASIC. the computer can run hundreds, if not thousands of times faster. For tasks like sorting. searching and some graphics, ML is the fastest (and probably the only satisfactory) way of programming. All tasks that require a large number of repetitions of whatever type – that is where ML will win hands down. You will also find that judicious use of ML permits larger or more complex programs to reside in the limited memory space of a micro computer.

THE AUTHOR:

Paul Blair (DipCE, MIE Aust.), is a professional engineer in the field of national road policy.

Computing interest started in the sixties when he commenced writing structural design programs in FORTRAN IV. In 1979, he moved into writing custom software. Involvement grew as did international contacts which include Canada, U.S.A., England, New Zealand and the Netherlands. Paul has had items published in most of these countries and is a regular contributor to the Australian Commodore Magazine. He is a strong supporter of computer user groups and is constantly called upon for demonstrations and tutorials in schools and other institutions.

A second or follow up volume to this book is already being formed.

KiM BOOKS

(A division of Mervyn Beamish Graphics Pty Ltd)

ISBN 0 9591417 0 7