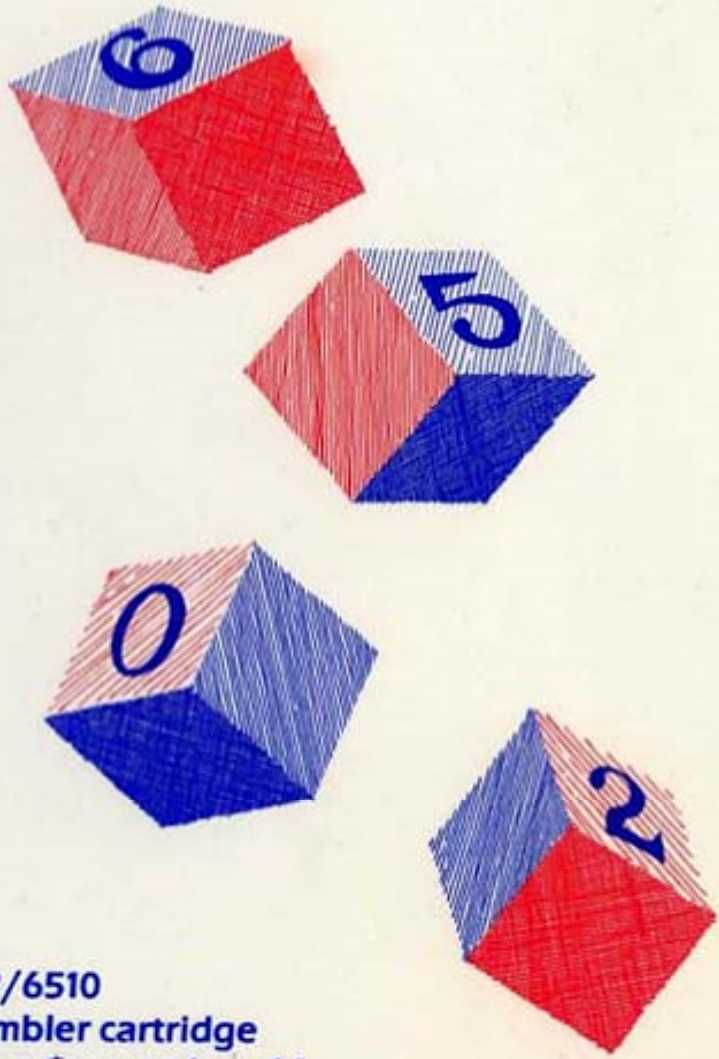
MIKRO ASSEMBLER



6502/6510
assembler cartridge
for the Commodore 64.

MIKRO 64

three-pass assembler
unlimited labels
source files can be linked
machine code monitor
full screen editor
disk or tape operation

Distributed in the United States and Canada by
Skyles Electric Works, 231E S Whisman Road,
Mountain View, CA 94041, U.S.A.

Distributed in Europe and the rest of the world by
SUPERSOFT, Winchester House, Canning Road,
Wealdstone, Harrow HA3 7SJ, England

INTRODUCTION

This manual explains in full the facilities of MIKRO for the 64, but does not attempt to teach assembly language programming, which is something best learnt by experience. If you are familiar with 6502 assembly language these instructions should be sufficient to 'get you going'; if you are new to machine code you should have by you one of the standard reference books on 6502 programming.

INSTALLATION

The MIKRO cartridge fits in the cartridge port at the rear of the 64. Never install or remove a cartridge unless the machine is turned off otherwise damage to one or other could result.

When you power-on your computer you should see the message:

```
**** MIKRO ASSEMBLER (C) AJ TROTT ****

64K RAM SYSTEM  30719 BASIC BYTES FREE

READY
```

Although the cartridge removes 8k of Ram that you can directly address from Basic you can rest assured that it makes full use of the available 64k of Ram during assembly!

JUST LIKE BASIC

When you write assembler source code using MIKRO it is just like writing a BASIC program. Of course the keywords are different, and the syntax is more restrictive, but you'll find it easy to switch between one and the other. When you come to SAVE your source code you can use the same SAVE command that you'd use to save a BASIC program. LOAD and VERIFY work in just the same way too!

Here's a sample MIKRO program that you might like to type in yourself:

```
100 *=$6000
110 SCREEN=$0400
120 LDX #$0
130 LDA #$0
140 LOOP STA SCREEN,X
150 INX
160 BNE LOOP
170 RTS ! BACK TO BASIC
```

Let's just look at each of the elements of the program. Line 100 sets the start address to hex 6000; if you leave this out the start address MIKRO will assume $033C which is the start of the 'cassette buffer', an area of memory that is used only during tape operations. On line 110 the label SCREEN is defined as hex 0400, the first byte of screen memory; if you leave this line out then when you try to assemble the program you will get the message UNDEF'D LABEL ERROR IN 140. Finally, note the exclamation mark that precedes the comment on line 170; this is the equivalent of a REM in BASIC.

The order in which the various elements of MIKRO source code appear on the line is important. MIKRO checks the first word to see whether it is a valid opcode or a pseudo-op; if not it is assumed to be a label. The next element must then be an opcode or pseudo-op, followed in appropriate cases by one or more operands (i.e. INX does not require an operand, but LDX does). The operand often clarifies which of a number of similar 6502 instructions is meant:

```
LDA $10           zero-page absolute
LDA $FFF0         absolute addressing
LDA #$0D          immediate mode
LDA TABLE,X       absolute indexed
LDA (POINTER),Y   indirect indexed
LDA (ADDRESS,X)   indexed indirect
```

Certain pseudo-ops can be followed by more than one operand - see BYT and WOR for example.

After the operand is the comment, if any, separated from the rest of the line by an exclamation mark. Note that whereas BASIC tends to ignore spaces, MIKRO expects at least one space between each field (it helps distinguish bewteen fields). All spaces after the first are however ignored, but then MIKRO has the FORMAT command to organise the various elements into columns.

EDITING COMMANDS

MIKRO has a number of useful commands to help you write and edit your assembler source (some of them can also be used with BASIC programs).

The AUTO function automatically displays a new line number whenever you type RETURN. For example, the sequence of line numbers in the example above can be generated by typing AUTO 100,10 (or just AUTO since a start line of 100 and an increment of 10 are the defaults). To turn off the AUTO function just type RETURN (or shift-RETURN followed by RETURN if you don't want the line to be deleted).

DELETE enables blocks of program lines to be deleted simply, using the same syntax as for the LIST command, e.g. DELETE 100-250 or DELETE -9999 etc.

FORMAT will LIST your source code neatly formatted, making it easier to distinguish between labels, opcodes, operands, and remarks. The syntax of the FORMAT command is exactly the same as for LIST, and you can even FORMAT to the printer with OPEN4,4:CMD4:FORMAT.

FIND is a very useful command: FIND SCREEN would find all occurences of the label SCREEN and list the relevant lines on the screen (or printer if you have done an OPEN4,4:CMD4) You can limit the search to a particular range of lines as follows: FIND SCREEN,100-250 or FIND LDA,5000- and so on. BASIC words (such as LIST or PRINT) and MIKRO commands are 'tokenised', except when they appear within quotes. So you will not find PRINTER if you enter FIND PRIN, though you will find PRINCIPLE.

OTHER COMMANDS

NUMBER will give you the binary, octal, decimal and hexadecimal equivalent of a number in the range 0 to 65535 (decimal). Binary, octal, and hexadecimal numbers are identified by the prefixes %, @, and $. For example, NUMBER$6000 would display the following:

```
      $6000  24576  @060000  %0110000000000000
```

You can also use the NUMBER command to perform simple sums using addition and subtraction only, e.g. NUMBER$033C+1234-%01111111, but if the sum includes subtraction the result must not be negative at any stage of the calculation. NUMBER123-456+579 would give the message ?UNDERFLOW ERROR.

DISASSEMBLE will display a disassembled listing of an area of memory; invalid opcodes are shown as BYT instructions. Examples of the syntax are DISASSEMBLE $9000-$A000 and DISASSEMBLE50000-. A similar disassembly function is available from the machine code monitor (see below). In each case MIKRO will dispay a screenful of information, then wait for a key to be pressed before displaying the next page.

MACHINE CODE MONITOR

The Commodore 64 has many features of the Commodore PET, but one thing it lacks is a machine code monitor like the TIM monitor on the PET. MIKRO makes up for this with a monitor that includes all of the TIM commands with some useful extra functions. To enter the machine code monitor type TIM, which will give you a display something like this:

```
        CALL @ 814B

         ADDR IRQ  SR AC XR YR SP
       .;814B EA31 4D 00 00 00 F9
```

When you enter the monitor with a break (i.e. the 6502 BRK

instruction) the display shows the values of the processor's internal registers at the time. However, when you call the monitor by typing TIM the values are not meaningful.

These registers can be displayed at any other time by typing R then RETURN; to alter the values merely type over them and press RETURN to enter the changes. The register values are loaded into the 6502 registers when you execute a machine code program from the monitor using the G command (see below) so make sure that you are not using spurious values (it's better to enter the monitor with T shift-I which displays a status register value of 0, rather than TIM which gives 4D).

The M command will display the contents (in hex) of any area of memory:

```
.M 033C 0347
.: 033C 60 00 00 1C 00 00 00 00
.: 0344 00 00 00 00 00 FF 00 00
.
```

You can alter the memory contents by overtyping the values and pressing RETURN to effect the changes, which are made at once. Before MIKRO and other assemblers became available for the PET this was how many machine code programs were written, although it is very hard work!

The monitor has load and save commands which work rather differently from a normal LOAD or SAVE:

```
.S "0:TESTPROG",08,2000,2400
```

would save to disk drive 0 the contents of memory from $2000 to $23FF (note that the end address you enter must be one above the actual end address). Use device number 01 for cassette.

```
.L "FILENAME",08
```

would load the program FILENAME from disk; to load a program from tape use device 01 - or omit the device number

altogether. In fact if you are using tape you can type just L and RETURN to load the first program from tape. Loading a program through the monitor is like loading with a secondary address of 1 in BASIC (e.g. LOAD"CODE",8,1) - the program is not relocated on loading. There is another difference - the end-of-program pointers used by BASIC are not altered when you do a monitor load.

To execute a machine language program from the monitor use the G command, e.g.

```
.G
.G 4000
```

G without an address jumps to the program counter address shown in the register display; otherwise it executes the code at the specified address. To return from your machine code routine to the monitor you must terminate it with a BRK opcode, not an RTS.

All the above commands are carried over from the standard Commodore TIM monitor, but there are three new commands included in the MIKRO monitor:

```
.H 2000 3000 20 D2 FF
```

would HUNT through memory from 2000 to 3000 for occurences of the three hex bytes 20 D2 FF. The address of each occurrence is displayed on the screen; you can terminate the search by pressing the RUN/STOP key.

```
.T 4000 5000 6000
```

would TRANSFER the block of memory from 4000-4FFF to 6000-6FFF. The block being copied is not corrupted unless the new block overlaps the old.

```
.D A000 A010
```

would disassemble the code in memory from A000 to A010. Invalid opcodes are displayed as BYT instructions.

To exit from the monitor back to BASIC type X, then RETURN.

THE ASSEMBLER - INTRODUCTION

At last we come to the most important part of MIKRO, the assembler. When you write assembler source code, although it looks much like a BASIC program, it cannot be RUN in the same way. First it must be assembled.

Type ASSEMBLE to start the assembly process, which is in three passes. Most errors will be identified during the first pass. A typical display might look like this:

```
ASSEMBLE

* PASS (1) *

* PASS (2) *

* PASS (3) *


**** ASSEMBLY COMPLETE ****

  START ADDRESS - $033C
  END ADDRESS   - $0360
```

The pass numbers are displayed at the beginning of each pass to let you know what is going on; short programs are assembled instantly, but a really long program may take a minute or so.

After assembly the start and end of the program is displayed, enabling you to save the machine code using the S command in the monitor. If your program includes several *= commands so that the code does not occupy a continuous area of memory then the start and end of each segment is given.

PSEUDO-OPS

Pseudo-ops are instructions to the assembler, unlike opcodes which when assembled end up as instructions to the 6502 processor. The most important pseudo opcode is the *= instruction which sets the address at which assembled code is to be executed:

```
*=$033C
*=*+1024
*=$A000,$6000
```

The first example is straightforward; the code is to be assembled starting at hex 033C. In the second the * symbol appears twice - on the right of the equation it represents the current assembler address (i.e. where the next instruction would go but for the *= command). Statements such as LABEL=* are also permissable.

In the final example the code being assembled is to be executed at $A000 which is a ROM area; the assembler puts the code at the address specified after the comma, but the code will be assembled as if at $A000.

WOR, BYT and TXT are pseudo-ops which place data in memory. WOR allows a series of 16 bit values to be placed in memory in lo-hi form, e.g.

```
100 *=$1000
105 SCREEN=$0400
110 WOR 750,$2345,SCREEN
```

which would store in successive bytes (starting from $1000) the values $E8, $02, $45, $23, $00, and $04. BYT is a similar instruction which handles 8 bit values, for example:

```
150 BYT $03,255,'R,$00
```

Note the use of the ' character which tells MIKRO to use the ASCII value of the character that follows, unless it happens to be a question mark in which case... try it and see! Strings of characters can be entered using BYT or the TXT instruction:

```
160 BYT "MESSAGE",$0D
170 TXT "THIS IS A TEXT STRING"
```

You must terminate a text string with a double quote.

LINKING MIKRO FILES

Because MIKRO manages the 64's available memory very efficiently up to 30k of source code can be entered as a single program. However, even 30k of source will generate only 3 to 4k of finished machine code, well within MIKRO's limit of 12k.

To enable longer machine code programs to be written MIKRO uses the LNK pseudo-op, which tells MIKRO the name of the next file:

```
990 LNK "PART TWO",8
```

Note that the syntax for the link command is very similar to that for LOAD in standard BASIC. If the file is to be loaded from tape the device number can be omitted.

Because MIKRO requires three passes to assemble each file must be loaded in three times, once for each pass. Obviously you must have the first file in memory when you type ASSEMBLE, and so that MIKRO knows that the end of a pass has been reached you must use the END pseudo-op to link the last file to the first, e.g.

```
2500 END "PART ONE",8
```

The END pseudo-op can also be used in a single file to prevent MIKRO looking beyond a specified point in your source code. This is particularly useful if your source code is followed by lines of BASIC.

It is a good idea to avoid an overlap of line numbers between linked files; this avoids confusion, and makes it easier to transfer blocks from one file to another. Of course, ALL of your files must have been saved on disk or tape before assembling - this includes the very first file. In the event that one of a series of linked files cannot be found, or if the device specified is not present, then the assembler is likely to 'crash'.

OUTPUTTING ASSEMBLED CODE

If you have access to a printer, you will find it useful to have a listing of your program which shows both the source code and the assembled machine code. The OUT command allows you to direct the output to a printer connected to the 64's serial port or to a Centronics-compatible parallel printer hooked up (with a suitable cable) to the User Port of the 64. Examples of the OUT command are:

```
10 OUT 4    (printer connected to serial port
             e.g. 1515, 1525)
10 OUT 132  (ditto, but without auto linefeed
             after carriage return)
10 OUT 6    (parallel printer with auto linefeed)
10 OUT 134  (ditto, but without auto linefeed)
```

Device number 6 is reserved for parallel printers; device numbers 4 and 5 are usual for printers connected to the serial IEEE port. Note that 128 should be added to the device number when the printer does not automatically line feed on receiving a carriage return character. BASIC has a similar convention, but uses the file number rather than the device number to select auto linefeed.

The parallel printer routines built into MIKRO can also be accessed from BASIC (with OPEN6,6 or OPEN134,6), and could be used in conjunction with PRINT# commands or CMD followed by LIST, FORMAT or DISASSEMBLE.

OUT without a device number directs output to the screen; in this case only, the code is formatted without line numbers to give a more presentable display on the 40-column screen. If you do want to see the line numbers on the screen use OUT 3 (the screen is device 3).

Output commences with the line following the OUT pseudo-op; it can be terminated with the OFF statement. END also cancels the OUT command. OUT cannot be used to send output to a file on tape or disk.

LABELS AND OPERANDS IN MORE DETAIL

Labels can be used as values, or to identify blocks of code, so that branches and jump instructions do not refer to absolute addresses (note that MIKRO does not allow branch instructions such as BNE *+3 - you must use an address or a label). For example:

```
130 CHAR=0
140 LDA #CHAR
150 LOOP STA SCREEN,X
160 INX
170 BNE LOOP
```

Here the label CHAR is assigned the value 0; the label LOOP is used to identify the target address for the branch on line 170. SCREEN has not been defined and will give UNDEFINED LABEL ERROR IN 150 unless we put the situation right with, say, 190 SCREEN=$0400.

Labels which are not in zero-page need not necessarily have been defined by the time they are used, but there is a limit to the extent that MIKRO looks ahead. For example:

```
110 LDA #FLAG
120 LDA #CHAR
130 CHAR=FLAG
140 FLAG=$22
```

Line 110 will produce the correct code, because MIKRO needs to look ahead only once, but line 120 fails. Labels which relate to zero-page addresses MUST be defined before they are used.

Often you will need to separate the high and low bytes of a 16-bit value; this can be done using < and > which select the low and high bytes, as demonstrated below:

```
200 LDA #<TABLE
210 STA POINTER
220 LDA #>TABLE
230 STA POINTER+1
```

Operands can incorporate simple arithmetic, as in line 230 above - but beware, the results may not always be those that you expect. LDA #>$FRED+$10 where FRED was $1234 would be equivalent to LDA #$22, not LDA #$12.

LABEL TABLE

MIKRO maintains an alphabetical list of labels; this is one reason why it can assemble at such a fast rate. The TABLE command allows you to display (or following OPEN4,4:CMD4 or similar, print) the sorted list together with their values, for example:

```
1234 LABEL
0400 SCREEN
8000 TARGET
```

The label values are shown in hex, and on a printer will be listed in four columns. To find the value of any one label you can, after assembly, use the NUMBER command, e.g. NUMBERLABEL which would print the value of LABEL.

ERROR MESSAGES

If you're just starting to learn assembly language you'll probably make quite a few mistakes. There are a number of error messages, all self-explanatory:

NUMBER TOO LARGE	ILLEGAL OPERAND
OUT OF RANGE	ILLEGAL ADDRESSING
ILLEGAL LABEL	UNDERFLOW
UNDEFINED LABEL	OVERFLOW
BRANCH OUT OF RANGE	LABEL OVERDEFINED
NO OPERAND	NO OPCODE

Errors are often caused by omitting the space between a label and the following opcode. Usually you can spot an error more easily if you FORMAT the source code.

MEMORY USAGE

The MIKRO cartridge occupies memory from $8000 to $9FFF; most of the RAM memory under the cartridge is also used by MIKRO during assembly.

MIKRO reserves the memory from $A000 to $CF00 for its label table. The first 8k of the table from $A000 to $BFFF is in an area normally occupied by the Basic Rom, and thus not normally accessible, but the 4k block starting at $C000 is a useful place to assemble your routines or to locate utilities. MIKRO fills the label table from the bottom, and unless your program is particularly long the table is unlikely to extend into the $C block.

The page from $CF00 to $CFFF is used by MIKRO as workspace and for data storage. You should avoid overwriting this area.

MIKRO stores assembled code in the memory underneath the I/O devices and Kernal at $D000 to $FFFF.

CORRECTIONS AND UPDATES

PROGRAMMERS WANTED !

Most of the programs that we sell come from outside authors, usually writing in their spare time.

We normally remunerate programmers by paying a commission based upon the selling price of the program, and the number of copies sold. In appropriate cases we loan equipment because we don't believe that a good programmer should be restricted by not having the right tools for his task.

Our catalogue includes a wide range of programs, from business programs and word processors through utilities to games. Most are entirely in machine code, but we are more interested in the end product than the method of getting there so we don't automatically turn our nose up at Basic programs. On the other hand, it is difficult to imagine a program making full use of the 64's facilities without some machine code.

How many copies will we sell? That does depend very much on the program: many sell thousands, some will sell ten thousand or more worldwide, but others may struggle to sell a hundred copies. We try not to make promises that we can't keep, so if our estimates of potential sales are lower than other people's put it down to conservatism, not a lack of faith in the product.

If you would like to know more about how you can join the 40 or so programmers whose programs are already in our catalogues write to Peter Calver at SUPERSOFT.

What is machine language?

The 6510 processor at the heart of the Commodore 64 has a language all of its own. Programs written in machine language generally run hundreds of times faster than programs written in BASIC!

How will an assembler help?

Writing in machine language isn't easy. Computers can understand it, but humans find it rather more difficult.

Assembly language is easier for humans to write, but before it can be understood by the processor it must be assembled into machine code - which is where the assembler comes in.

What can MIKRO do?

The MIKRO assembler cartridge enables you to write assembly language quickly and easily (it's almost as easy as writing BASIC). You can then assemble your program into machine language. Most programs can be assembled in a few seconds.

MIKRO has a number of editing commands to help you write your programs - FIND, AUTO, and DELETE are particularly useful - plus a machine code monitor so you can load or save assembled programs, disassemble machine language routines etc.

Because MIKRO is easy to learn and easy to use it's ideal for those learning machine code programming - yet MIKRO is such a powerful tool that it has been chosen by many professional programmers writing for the 64.

MIKRO 64 COMMANDS	MIKRO PSEUDO-OPS	MONITOR COMMANDS
ASSEMBLE	BYT	D (isassemble)
AUTO	END	G (o execute)
DELETE	LNK	H (unt)
DISASSEMBLE	OFF	L (oad)
FIND	OUT	M (emory display)
FORMAT	TXT	R (egister display)
NUMBER	WOR	S (ave)
TABLE	*	T (ransfer)
TIM		X (it to Basic)