micron 1 computer user manual

MICRON COMPUTERS LTD

23 Holly Tree Lane, Cambridge, England

Manual version 1.0
Firmware version 1.3
Typeset by Giles Booth Associates

Based on a hardware design by Benjamin Eater

Printed in England

welcome

Congratulations on making an investment in the Micron 1 training computer. It's an investment in your future, as we feel sure that microprocessors and microcomputers will be hugely important in the coming decade.

In the 1980s learning to program computers will also be a valuable skill, increasingly in demand in industry.

We also believe that learning to create your own simple computer programs is a transferable skill. The MOS Technology 6502 processor at the heart of the Micron 1 has a limited instruction set, forcing the programmer to break down any problem into the simplest constituent parts before she can start typing in any instructions.

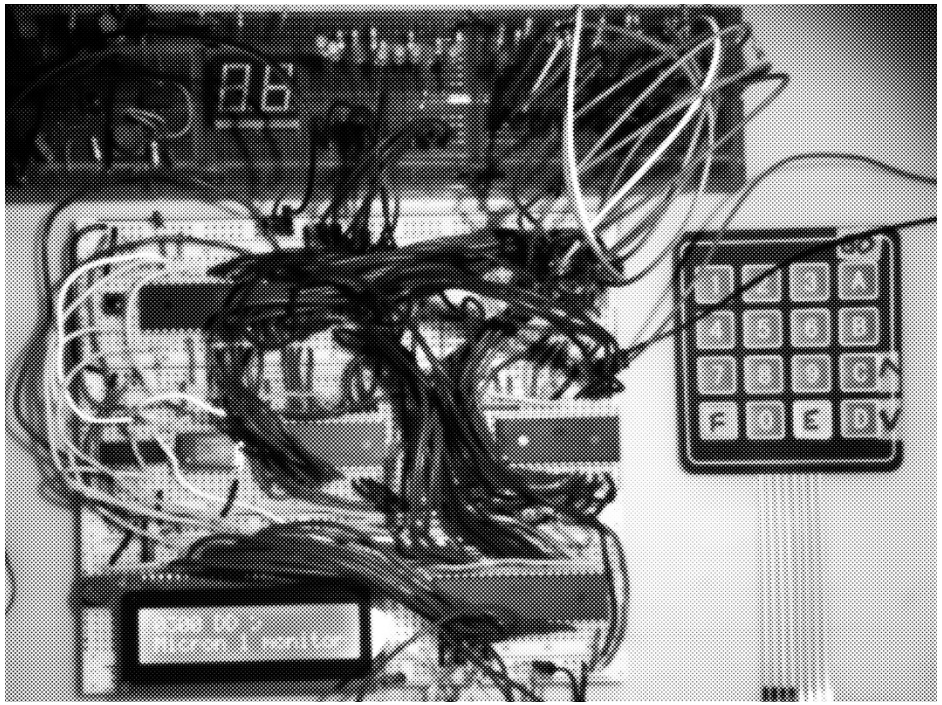
Any user who learns the fundamentals of machine code and assembly language programming on the Micron 1 will also be better prepared and write better code when writing software in higher level languages like FORTRAN and BASIC.

Welcome on board the Micron 1 learning express!

specifications

Each Micron 1 computer is lovingly hand made to the following specification:

- 6502 processor running at 1 MHz
- Optional single-step and adjustable slow speed clock for debugging
- 6522 versatile interface adaptor (VIA)
- 32K RAM
- 32K EEPROM containing the monitor program
- 16-key hexadecimal keypad for input
- 16 x 2 alphanumeric LCD display for program output
- ‘Blinkenlights’ LEDs across data and address busses and keypad encoder for debugging



getting to know you

Take a moment to familiarise yourself with the key user features of your new computer.

1. Hexadecimal keypad for instruction and data entry. The GO, up and down keys are also on here, accessed by also pressing the shift key.

2. Shift key

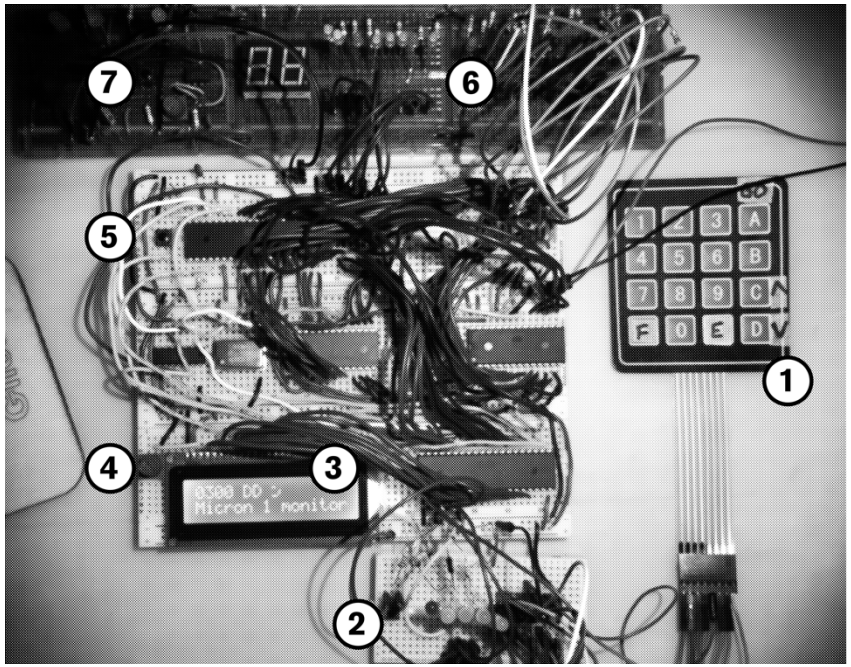
3. Alphanumeric 16 x 2 LCD display

4. Display contrast control

5. Reset button

6. Data and address bus lights

7. Single-step and slow clock which can be wired to replace the 1 MHz crystal oscillator for program debugging.



get started

When you power on your Micron 1 for the first time, you should press the reset button (5) situated to the left of the 6502 CPU. You should do this even if it appears to be working normally, to ensure all registers and parameters are correctly initialised.

You should see the monitor program displayed on the LCD display (3). If the characters aren't clear, adjust the contrast potentiometer (4) at the top left corner with a small screwdriver until you can read it easily.

The display shows the current memory address being examined, the single byte contents of that location as a two-digit hexadecimal number, and an ASCII representation of the location's contents.

All numbers for memory locations and contents are shown in the hexadecimal (or base 16) number system:

decimal number	hexadecimal number
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10

Memory locations contain 1 byte of data, which is represented as two hexadecimal digits. For example, if a location contains decimal value 255 this will be shown as FF.

using the monitor

A monitor is a simple program that allows you to enter, view, edit and run programs on a computer.

This manual does not attempt to teach 6502 machine code, but you will learn some basic concepts using the example programs we provide. We recommend studying a text book such as *Programming the 6502* by Rodnay Zaks.

Normally programs for microprocessors are written in assembly language. Each instruction has a short text label, often called a mnemonic, which must be converted into a hexadecimal before entering into the computer.

For example, you may write a program to load the hexadecimal number 40 into the accumulator. The accumulator (A) is one of the computer's registers and is used for most calculations. You would write the program in assembly language thus:

LDA #$40

LDA is the mnemonic for the instruction to load a number into the accumulator. The hash sign # shows the addressing mode, in this case the immediate mode. The number that follows is the data to be loaded (rather than an address pointing to the data). The $ sign is used to show that the following number is in hexadecimal format. You would omit it for decimal numbers or use a % sign for binary numbers.

To enter this program on the computer, we must translate the assembly language program to machine code – raw numbers:

A9 40

A9 is the machine code representation of the instruction LDA # - load the next digit into the accumulator (A).

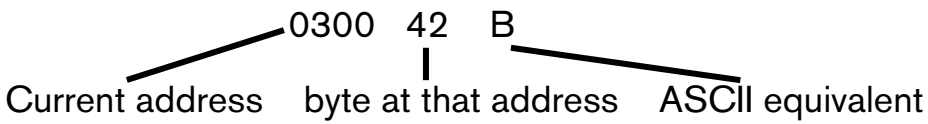
On your Micron 1, user-entered programs always start at memory location $0300. They can be up to 256 bytes in length. Programs are written in 6502 machine code and must be entered as hexadecimal numbers.

You may wish to use a cross-assembler to write your programs. A cross-assembler is a program that runs on another computer which translates assembly language instructions like 'LDA #$40' into machine code, numbers that the 6502 processor can understand, such as 'A9 40'.

If you don't have access to a cross-assembler, you can translate the instructions yourself. A table of instruction mnemonics and their numerical values is provided on a laminated card.

your first program

The display is in this format:



When you turn on your Micron 1, the RAM memory can contain anything, so you may see different symbols and numbers displayed.

Let's enter and run our first program.

Make sure the first part of the display is showing 0300. If not, press the reset button.

Here's the program we're going to enter. It will add two hexadecimal numbers $40 and $2 and store the answer at memory location $0309:

Addr.	**assembler**	**machine code**	**notes**
$0300	CLC	18	clear carry
$0301	LDA #$40	A9 40	load $40 into A
$0303	ADC #$02	69 02	add $2
$0305	STA $0309	8D 09 03	store result at $0309
$0308	RTI	40	jump back to monitor

To enter the program, just use the numbers in the machine code column. On the keypad press '18'. You should see the number in the middle of the display change to 18, so the display will show:

0300 18

Well done, you've entered your first instruction to your computer! The CLC (clear carry) instruction is often found at the start of 6502 programs, to make sure the carry flag is zero before doing addition.

To enter the next instruction, we need to move to the next memory location by holding down the shift button and pressing the D↓ key on the keypad.

The display will now show memory location $0301. Enter A9 and press shift-D ↓ again to move to the next location. Keep going until you get to location $0308 and enter 40.

It's a good idea to check your program before you run it, so press shift-C ↑ repeatedly until you get back to location $0300. Step forwards again, checking each memory location contains the numbers listed in the machine code column. If you find any mistakes, just type over the number shown. The monitor program loads anything you type into the memory location being examined.

ready? go!

If you're happy you've entered the correct numbers for your program, you're ready to run it. Hold down the shift button and press key A GO on the keypad.

The monitor program will jump back to $0300 and it will look like nothing has happened. But if you step forwards to memory location $0309 using shift-D you should see that location now contains the number $42:

0309 42 B

You added $40 and $2, got the result $42 and stored it in your computer's memory. Why not bask in the glory of this achievement by mixing yourself a well-earned Pan Galactic Gargle-blaster, or perhaps just have a nice cup of tea.

Let's examine some of the instructions in more detail. We've already discussed the CLC (clear carry) instruction.

LDA #$40 loads the next number ($40) into the accumulator (A). The accumulator is one of the 6502 processor's registers, and the main one used for mathematical operations like addition and subtraction. There are also X and Y registers, often used as indices. We'll look at some uses of these later.

ADC #$02 (add with carry) adds the number immediately following it to whatever is currently in the accumulator. In this case it adds $2.

STA $0309 stores the new value of the accumulator, $42, in memory location $0309. You may notice that in the machine code version you typed in, the address of the location for storing the result is back-to-front. This is not a mistake!

The 6502 processor has a 16-bit address bus. This means it can read to and write from 65,535 different locations, from $0000 to $FFFF in hexadecimal. So, when referring to absolute memory addresses, we need to use two bytes.

The processor refers to absolute addresses using the low byte first, and then the high byte. So our instruction to store the contents of the accumulator in location $0309 becomes '8D 09 03' in machine code.

Finally, the RTI (return from interrupt) instruction returns execution of the program you entered to the monitor program so you can examine memory locations or enter and run a new program.

adding larger numbers

Our previous program can only add two small, one-byte numbers together. Indeed, if the result is larger than 255 ($FF), the result will be incorrect.

Here's a program to add two 2-byte numbers together. As long as the result is $FFFF or less, the result will be correct.

The sample numbers being added here are $14FF and $2317. The result should be $3816. The first number is stored at locations $0320 and $0321. The second number is stored at $0322 and $0323. When you run the program, the result of the addition will be stored at locations $0324 and $0325.

We're using a new instruction here as well: CLD (clear decimal mode). As well as being able to work with hexadecimal numbers, as is normal with microprocessors, the 6502 can also work with binary-coded decimal numbers. It's common practice in 6502 programs to clear the decimal mode at the start of a program, just as you should clear the carry flag before doing any addition.

This program also makes use of the carry flag by adding the low (smaller value) halves of the numbers first. If there is an overflow, this will set the carry flag and this will be taken into account when adding the high (bigger value) halves of the two numbers.

```
addr      m/c code       notes
-----------------------------------------------------
program:
$0300   18             CLC clear carry
$0301   D8             CLD clear decimal mode
$0302   AD 21 03       LDA low half of num1
$0305   6D 23 03       ADC add low halves num1+2
$0308   8D 25 03       STA save low half of result
$030B   AD 20 03       LDA high half of num1
$030E   6D 22 03       ADC add high halves num1+2
$0311   8D 24 03       STA save high half of result
$0314   40             RTI jump back to monitor

data:
$0320  14              Num1 high
$0321  FF              Num1 low
$0322  23              Num2 high
$0323  17              Num2 low
```

displaying text

As part of your program, you may want to display text on the Micron 1's LCD display. The monitor program has a routine that will allow you to do just that, one character at a time. Load the ASCII value of the character you want to display into the accumulator (A) and use the JSR (jump subroutine) instruction to call the print_char routine in the system ROM at location $80D6.

You can also send control codes to the LCD, for example to clear the display, using the lcd_instruction routine at location $80CO.

addr	assembler	machine code	notes
$0300	LDA #$01	A9 01	clear screen
$0302	JSR lcd_instruction	20 C0 80	
$0305	LDA #"h"	A9 68	letter h
$0307	JSR print_char	20 D6 80	
$030A	JMP [here]	4C 0A 03	infinite loop

This program clears the display, prints the letter 'h' and then sits in an infinite loop so you can see the letter on the screen. The JMP (jump) instruction tells the processor where to look for its next instruction. To return to the monitor program, simply press the reset button next to the CPU.

This would be a tedious way to enter a long amount of text, so instead you can create a string of text in memory and create a loop that iterates over it until it reaches the value zero:

addr	assembler	machine code	notes
$0300	LDA #$01	A9 01	clear screen
$0302	JSR lcd_ instruction	20 C0 80	
$0305	LDX #$00	A2 00	set x index to 0
print $0307	LDA text,x	BD 17 03	load next character
$030A	BEQ end_print	F0 14 03	if zero, jump to the end

$030D	JSR print_char	20 D6 80	otherwise print the character
$0310	INX	E8	increment character index
$0311	JMP print	4C 07 03	fetch next character
end_print $0314	JMP end_print	4C 14 03	infinite loop
text $0317		68	h
$0318		65	e
$0319		6C	l
$031A		6C	l
$031B		6F	o
$031C		00	zero terminates string

There is much to examine here as we have introduced some new instructions, some new addressing modes and some new programming concepts.

In short, the program clears the screen and then loads a string one character at a time into the accumulator and calls the ROM subroutine to print it on the display. The string is terminated with the number zero, which is how it knows when to stop. It then goes into an infinite loop until you press the reset key so you can read the word.

A new concept here is using an area of memory to store a string of data and use the X register as an index to iterate over it. You'll also notice that some of the memory locations have text labels like 'print' and 'end_print.' These are commonly used when using an assembler program, which keeps track of these text labels and turns them into numerical addresses when the program is compiled into machine code. If we're assembling our programs by hand, however, we need to keep track of these addresses ourselves and remember they may change if we add instructions before the end of the program.

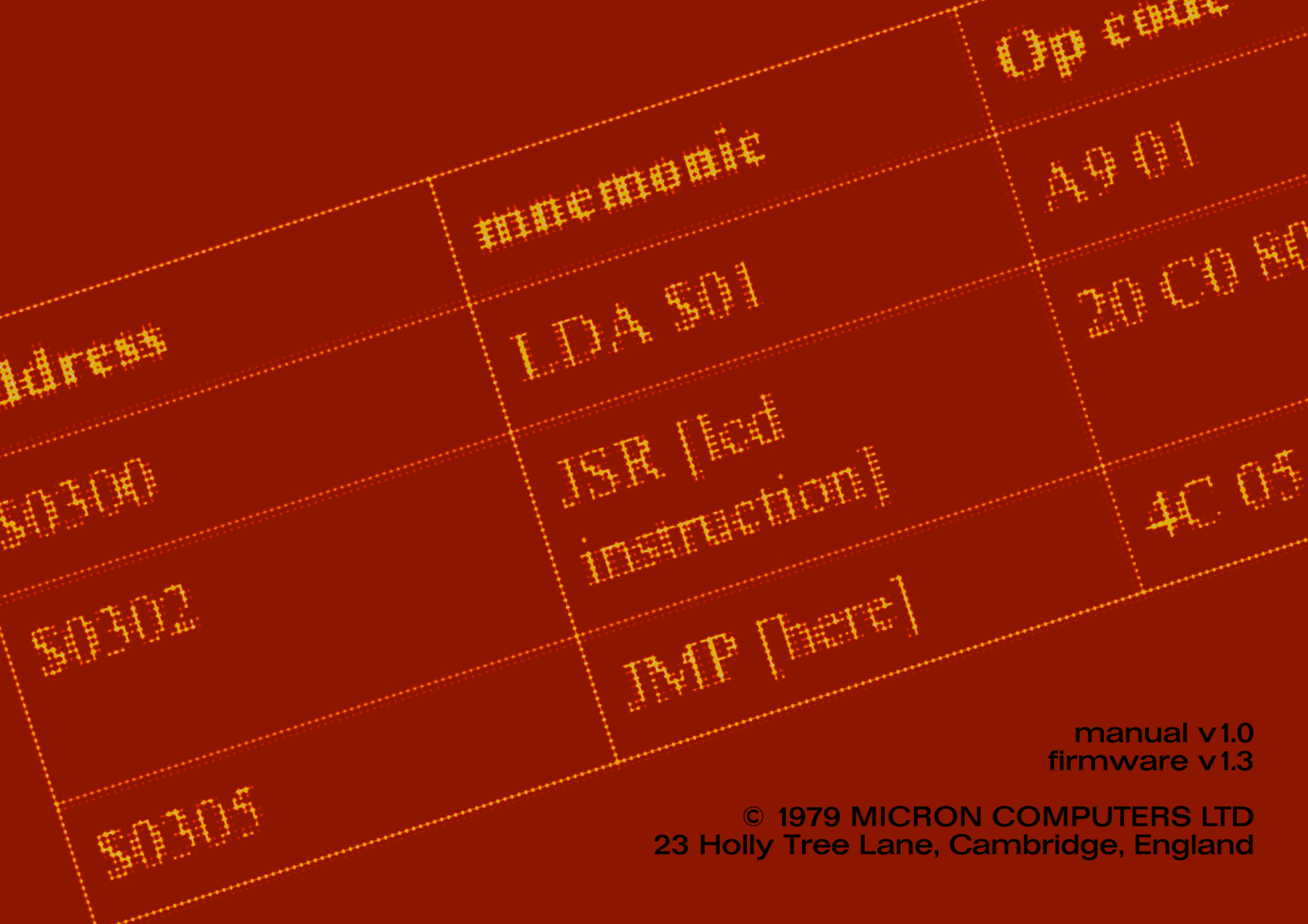
The program sets X to 0 (LDX #$00) and then uses a new address mode to load a value into the accumulator. LDA text,x means load the value found at the text address offset by the current value of X. Note that the machine code for the LDA instruction here is BD, not A9 as at the start of the program. This is because instructions with different address modes have different machine code values. A9 is the code for LDA in immediate mode, where the value to be loaded follows in the next byte. BD is the code for LDA in absolute,x mode where the instruction is followed by a 2 byte, 16 bit address which will be offset by the current

value of the X register. The contents of that memory location will be loaded into the accumulator (A).

The next crucial new concept is conditional branching. The ability to execute different instructions depending on the values of register contents is what distinguishes a general-purpose computer from a mere adding machine.

Having loaded a character from memory, the BEQ (branch on equal) instruction is used to determine if the character should be printed to the display, or if the end of the string has been reached. We terminated the string with the numerical value zero, so if this has been reached, the program branches to the infinite loop at the end. If the character's value is not zero, it calls the subroutine to display it, increments the index (X) by 1 and jumps back to the point in the program where it fetches characters from the string.

Hopefully you can see how a very limited set of instructions can be quite powerful and produce interesting results.



manual v1.0
firmware v1.3

23 Holly Tree Lane, Cambridge, England