Small Computer Monitor

Tutorial

Monitor version 0.2 for the Z80 CPU

Software and Documentation by Stephen C Cousins Tutorial edition 0.2.0

Contents

Overview

The Small Computer Monitor is a classic machine code monitor enabling debugging of programs and general tinkering with hardware and software. It can also act as a boot ROM, so no other software is required on the target computer system.

This tutorial is designed for people with little, or no experience of machine code monitors and Z80 programming. If you have already had experience of these things the Small Computer Monitor User Guide should be all the documentation you need.

This tutorial assumes you are using an RC2014-Z80 system and that the system is up and running with the supplied BASIC ROM. You should therefore have the RC2014 connected to a terminal or a PC (or similar) running terminal emulation software.

The first thing to do is fit a ROM containing the Small Computer Monitor program, and configure the hardware to boot up from this ROM.

After that you can enter and run machine code programs, test and debug programs, and generally experiment with the hardware.

This tutorial concentrates on the use of the Small Computer Monitor, rather aiming to be a definitive guide to Z80 programming. However, to use the monitor requires some understanding of Z80 programming. Therefore simple programming is explained in this document. This should be enough to get you started.

Once the Small Computer Monitor and the basics of Z80 programming are understood there are plenty of suitable resources available on the internet which can help with further progress.

Prerequisites

Before diving in to using the monitor program there are a few things you need to be familiar with. These are:

- Binary numbers
- Hexadecimal numbers
- ASCII characters

The next few pages give a brief introduction to these topics, but if you are in any doubt please seek further information before continuing.

Decimal numbers

The decimal numbers we use in day to day life are expressed in base ten (or denary). This means we use ten distinct symbols (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9).

To express numbers greater than 9 we use more than one of these digits. Using two digits allows us to represent 100 different numbers (0, 1, 2, … , 97, 98, 99).

The least significant digit is on the right, with the digit to its left representing ten times (base ten remember) the value of the digit to its right. And so on.

Thus decimal number 12 means (1 x 10) + (2 x 1), and the decimal number 123 means (1 x 1000) + (2 x 10) + (3 x 1).

Binary numbers

Binary numbers follow the same pattern as decimal numbers but are in base two. This means we use two distinct symbols (0 and 1).

To express numbers greater than decimal 1 we use more than one of these digits. Using two digits allows us to represent 4 different numbers (0, 1, 2 and 3)

The least significant digit is on the right, with the digit to its left representing two times (base two remember) the value of the digit to its right. And so on.

Thus binary number 10 means decimal (1 x 2) + (0 x 1) = 2, and the binary number 101 means decimal (1 x 4) + (0 x 2) + (1 x 1) = 5.

Hexadecimal numbers

Hexadecimal numbers (or Hex for short) follow the same pattern as decimal numbers but are in base sixteen. This means we use sixteen distinct symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F).

The letters A to F represent decimal numbers 10 to 15 respectively. So A=10, B=11, C=12, D=13, E=14 and F=15.

To express numbers greater than decimal 15 we use more than one of these digits. Using two digits allows us to represent 256 different numbers (0, 1, 2, … , 253, 254, 255).

The least significant digit is on the right, with the digit to its left representing sixteen times (base sixteen remember) the value of the digit to its right. And so on.

Thus hexadecimal number 1F means decimal (1 x 16) + (15 x 1) = 31, and the decimal number 1F2 means (1 x 256) + (15 x 16) + (2 x 1) = 498.

Why not just use decimal numbers?

The reason all this matters is that our normal base ten decimal numbering system does not fit well with digital computers.

Computers are made up of electronics which is digital. Being digital means that each of the smallest components that make up the computer can only be in one of two states, on or off (or one or zero).

Binary notation is thus the most basic way to describe the state of any part, or group of parts, in a computer.

Eight bit microprocessors, such as the Z80, process eight bits at once. To describe any combination of these eight bits requires eight binary digits. This gives the range of binary numbers from 00000000 to 11111111 (or decimal 0 to 255).

Now us mere humans have a bit of trouble working with numbers like 11010111, so we need a more convenient notation. This is where hexadecimal comes in.

A hexadecimal digit has sixteen possible values, which happens to be the same number as four binary bits. Thus the following are equivalent:

- Binary 0000 = Hexadecimal 0
- Binary 0001 = Hexadecimal 1

- Binary 0010 = Hexadecimal 2
- Binary 0011 = Hexadecimal 3
- Binary 0100 = Hexadecimal 4
- Binary 0101 = Hexadecimal 5
- Binary 0110 = Hexadecimal 6
- Binary 0111 = Hexadecimal 7
- Binary 1000 = Hexadecimal 8
- Binary 1001 = Hexadecimal 9
- Binary 1010 = Hexadecimal A
- Binary 1011 = Hexadecimal B
- Binary 1100 = Hexadecimal C
- Binary 1101 = Hexadecimal D
- Binary 1110 = Hexadecimal E
- Binary 1111 = Hexadecimal F

By using two hexadecimal digits we can describe 8 binary digits. This allows us to express the above example (binary 11010111) as hexadecimal D7. It is much easier for us to handle “D7” then “11010111”, and thus when programming at the ‘binary’ level we tend to use hexadecimal notation.

So why not decimal?
Because a single hexadecimal digit can exactly represent four binary digits, and two hexadecimal digits can exactly represent eight binary digits, there is a simple relationship between the binary and hexadecimal. This makes hexadecimal numbers a natural choice when working at the lowest level where we are concerned about what individual digital bits are doing. The same relationship does not apply to decimal notation, where a four binary digits requires either one or two decimal digits depending on the value of the binary digits. This just tends to be messy and inconvenient.

Bits, Bytes and Nibbles

With all these different quantities being used regularly, they had to have names. A Bit is a single digital quantity, which can be described in binary by one digit (0 or 1). A Nibble is a group of four binary digits, which can be represented by one hexadecimal digit (0 to F). A byte is a group of eight binary digits, which can be represented by two hexadecimal digits.

Avoiding confusion

You may have noticed above that most times a number was mentioned it was necessary to prefix is with the word decimal, binary or hexadecimal. Without doing

that how do you know if the number “10” is decimal 10, binary 10 (decimal 2) or hexadecimal 10 (decimal 16). In short, you don’t. You can often guess based on context but you can’t be sure.

To avoid this confusion and to avoid continually having to write “decimal”, “binary” or “hexadecimal” before every number, various notation systems have been used in the computer industry. Unfortunately not everyone uses the same system!

The various systems use either a prefix or postfix for identification of the number base. For example, putting “0x” before a number (eg. 0x10) indicates the number is expressed in hexadecimal.

Some commonly used identifiers are:

Identifier	Base	Example
0x	Hexadecimal	0x1FC
$	Hexadecimal	$1FC
H	Hexadecimal	1FCH
0b	Binary	0b101
%	Binary	%101
B	Binary	101B
+	Decimal	+123
D	Decimal	123D

As the Small Computer Monitor is mainly used to directly manipulate memory, registers and ports, which are digital in nature, the default for all numbers is hexadecimal. Thus hexadecimal numbers don’t usually require special identifiers, but there are some exceptions.

The disassembler shows constants as hexadecimal numbers prefixed by a “$”. The prefix is necessary here for clarity because the processor’s registers B, C, D and E are also valid hexadecimal numbers.

The assembler accepts hexadecimal numbers without any identifiers. To specify a decimal number it must be prefixed with “+”. Hexadecimal numbers prefixed with “$” or “0x” are also accepted.

ASCII characters

Within digital computers all the characters of the alphabet and all other symbols are stored as numbers. The character allocation standard used by the Small Computer Monitor is the American Standard Code for Information Interchange (ASCII).

The tables below show the subset of ASCII used by the Small Computer Monitor.

ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.
0	00	NUL	16	10		32	20	(spc)	48	30	0
1	01		17	11		33	21	!	49	31	1
2	02		18	12		34	22	“	50	32	2
3	03		19	13		35	23	#	51	33	3
4	04		20	14		36	24	$	52	34	4
5	05		21	15		37	25	%	53	35	5
6	06		22	16		38	26	&	54	36	6
7	07		23	17		39	27	‘	55	37	7
8	08	BS	24	18		40	28	(	56	38	8
9	09		25	19		41	29	)	57	39	9
10	0A	LF	26	1A		42	2A	*	58	3A	:
11	0B		27	1B	ESC	43	2B	ES+C	59	3B	;
12	0C		28	1C		44	2C	,	60	3C	<
13	0D	CR	29	1D		45	2D	-	61	3D	=
14	0E		30	1E		46	2E	.	62	3E	>
15	0F		31	1F		47	2F	/	63	3F	?

ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

So to refer to the letter ‘A’ when working in the monitor, it is usually necessary to use the decimal value 65 or hexadecimal value 41.

Getting Connected

If you already have your RC2014-Z80 system up and running with the supplied BASIC ROM then you can skip this section.

The RC2014 needs to be connected to a terminal or computer running terminal emulation software. The terminal provides the keyboard input and the displayed output.

The physical connection is a serial cable from the RC2014 to the terminal.

Serial modules

The RC2014 serial port can either be the official Serial I/O module, the official Dual serial module (SIO/2) or the RC2014 Mini's serial port. There are other modules available which might also work.

The Serial I/O module and the RC2014 Mini use a MC68B50 or compatible Asynchronous Communications Interface Adapter (ACIA) chip. The Serial I/O module has the option of either a 5 volt FTDI style serial interface or a 9-pin D-type RS232 serial interface, while the Mini only has the 5 volt FTDI style interface.

The Dual serial module uses a Zilog Z80 SIO/2 chip and provides two serial ports. The Small Computer Monitor only uses port A. This module only provides the 5 volt FTDI style serial interface, not the traditional RS232 serial interface.

Serial cables

To use the RS232 style interface you need a null modem lead to connect the 9-pin D-type connector on the RC2014 board to that on the terminal (or computer).

To use the 5 volt FTDI style interface you need an FTDI (or compatible) USB to serial cable. There are a variety of designs available. There have also been problems experienced with some cables with some values of resistors on the RC2014 boards.

See the section "Parts and Suppliers" for details of known compatible 'cables'.

Terminal programs

Unless you happen to be using a genuine computer terminal you will need to install a terminal emulation program on a PC or similar.

Several freely available terminal emulation programs are detailed below, but in general these are the required settings:

The port number will be determined by the hardware on your computer. Serial ports are normally called “COM#”, where “#” is a number like 3 (eg. COM3). When you plug in a suitable USB to Serial cable a COM port will become available on the computer. If your computer has a legacy RS232 port it will also have a COM port number.

Serial port settings for the RC2014 are normally:

- Baud rate = 115200
- Date bits = 8
- Parity = None
- Stop bits = 1
- Flow control = None
- New line character(s) = Carriage Return
- Local echo = Off
- Backspace key sends = Control+H
- Terminal type = VT100 (not critical as only a basic terminal is required)
- Transmission delay(s) = 0

Terminal software usually has options to add delays to characters sent from the terminal. These can usually be set to zero, but if you experience any loss of characters you can add a small delay.

Tera Term

Unless you already have a preference, I’d suggest using Tera Term.

For Tera Term v4.96, use the Setup menu, Terminal item and Serial ports item to configure as described above. Leave Answerback blank and Auto switch off.

Putty

Putty is a very capable program but has lots of complex options.

For Putty v0.70, use the Configuration dialog box that opens when the program loads.

On the Session page select the Connection Type = Serial.

Select the Category = Serial. Enter the Serial line to connect to = COM# (where # is the number of your serial port), speed = 115200, data bits = 8, stop bits = 1, parity = none, flow control = none.

Use the Terminal page to ensure “Impicit CR in every LF” is cleared, along with “Implicit LF with every CR”. Also local echo and local line editing are Off.

Use the Keyboard page to set Backspace to Control+H.

Return to the Session page, Select Default Settings and click Save. Then click Open.

Once the terminal window opens you can return to the configuration options by clicking on the terminal icon at the top left of the window and selecting Change Settings.

HyperTerminal

For users of older Windows installations, such as XP, you probably already have HyperTerminal installed. For HyperTerminal v5.1, use the File menu, Properties item to configure the settings as detailed above.

Installing the Small Computer Monitor

It is possible to run the Small Computer Monitor by downloading it into RAM, but here it is assumed you are running it from a Read Only Memory (ROM) chip.

If you need to program your own chip you will find the program supplied as an Intel Hex File suitable from opening with most chip programming products.

The programming method is dependent on the programmer you are using and is beyond the scope of this tutorial. See the section “Parts and Suppliers” for details of known compatible programmers.

The Small Computer Monitor can be either in a ROM chip with 8k bytes by 8 bits of memory (often described as 64k bit memory) or in part of a larger chip. Which every chip is used the Monitor program must be located in the Z80’s memory map starting at address 0x0000.

A suitable chip is the AT28C64B-15PU. This is a very convenient chip if you want to reprogram it again later as it is an Electrically Erasable Programmable Read Only Memory (EEPROM). It can be erased and reprogrammed in just a few seconds with a modern low cost programmer like the MiniPro TL866.

The exact fitting instructions depend on the circuit board it is plugged in to. The most common RC2014 boards are detailed below.

RC2014 original 8k ROM board

This ROM board only accepts 8k byte ROM chips and has no links to configure. So plug in the programmed chip and off you go.

RC2014 switchable ROM board

This ROM board can accept a range of chip capacities and has links to select how the ROM is addressed. If you are using an 8k byte chip such as the AT28C64B-15PU the links should be set as follows:

Page selection: A13, A14 and A15 link = Vcc (5 volts)

For larger chips, like the 27C512, links A13 to A15 need to be set to match where in the chip the monitor code resides.

RC2014 pageable ROM board

This is quite a flexible board and has a number of links which need to be set. If you are using an 8k byte chip such as the AT28C64B-15PU the links should be set as follows:

Page size: 8k (links as indicated by the PCB legend)

Page selection: A10, A11 and A12 = no link, A13, A14 and A15 link = 1

For larger chips, like the 27C512:

Page size: 8k (links as indicated by the PCB legend)

Page selection: A10, A11 and A12 = no link, A13, A14 and A15 need to be set to match where in the chip the monitor code resides.

RC2014 Mini board

This ROM board can accept a range of chip capacities and has links to select how the ROM is addressed. If you are using an 8k byte chip such as the AT28C64B-15PU the links should be set as follows:

Page selection: A13, A14 and A15 link = Vcc (5 volts)

For larger chips, like the 27C512:

Page selection: A13, A14 and A15 link = need to be set to match where in the chip the monitor code resides.

Switch on

If all is well, when you switch on your RC2014 system, you should be greeted with something like this on your terminal screen:

```
Small Computer Monitor by Stephen C Cousins
For Z80 based RC2014 systems
*
```

If you do not get a sign on message similar to the above, then consult the “Fault Finding” section which can be found towards the end of this document.

The “*” character displayed on the terminal is the Monitor Prompt. This indicates the monitor is ready to accepts a new command. Try typing the question mark character and then pressing the return key.

In the following text, user input is in a Bold Italic font, while the results are shown in a Regular font. Special key presses, such as Escape, are shown enclosed in curly brackets. Thus the example below means the user types “b 5000” on the terminal’s keyboard followed by a press of the Return key, and the monitor displays “Breakpoint set” on the terminal’s screen.

```
b 5000 {return}
Breakpoint set
```

Unless otherwise stated, parameters are hexadecimal numbers, such as FF12. There is no need to prefix them with a hexadecimal identifier or a numeric character. The exception to this rule is operands in the assembler.

Parameters are shown, in this document, enclosed by “<” and “>”, and further enclosed by “[“ and “]” if the parameter is optional.

Command names and any parameters are delimited by a space character.

Monitor commands are not case sensitive, so can be typed in either upper or lower case, or any combination of upper and lower case.

First Commands

Below are some of the simple monitor commands to try.

? or Help

Syntax: HELP
Or syntax: ?

This displays a list of the monitor commands together with their syntax.

For example:

```
help {return}
Monitor commands:
A [<address>]  = Assemble instructions
B [<address>]  = Breakpoint set or clear
D [<address>]  = Disassemble instructions
E [<address>]  = Edit memory
F <name>       = Flags display or modify
G [<address>]  = Go to program
I <port>       = Input from port
M [<address>]  = Memory display
O <port> <data>= Output to port
R [<name>]     = Registers display or edit
S [<address>]  = Step one instruction
Also: HELP, RESET
```

Reset

Syntax: Reset

This command performs a software reset, similar to pressing the reset button.

It can not perform a physical hardware reset on the electronics, but it does run the same software as a hardware reset.

Memory display

Syntax: M [<memory address>]

A block of memory is displayed, with each line showing the memory address in hexadecimal, the contents of sixteen memory locations in hexadecimal, and the contents of those sixteen memory locations in ASCII. Non-printable ASCII characters are shown as dots.

The memory address parameter is optional. If supplied the memory will be displayed starting at the specified address. If not, the memory display starts from the last address referenced.

For example:

```
m 1600 {return}
1600:  7C FF CB CE 21 15 16 CD  93 15 20 08 21 7C FF CB  |...!..... .!|..
1610:  C6 21 1B 16 C9 DD 15 E6  15 F1 15 A9 15 B2 15 BD  .!..............
1620:  15 11 27 16 C3 34 02 5A  38 30 20 62 61 73 65 64  ..'..4.Z80 based
1630:  20 52 43 32 30 31 34 20  73 79 73 74 65 6D 73 05   RC2014 systems.
1640:  00 21 7C FF 11 55 16 CB  46 C4 34 02 11 62 16 CB  .!|..U..F.4..b..
1650:  4E C4 34 02 C9 53 65 72  69 61 6C 20 41 43 49 41  N.4..Serial ACIA
1660:  05 00 53 65 72 69 61 6C  20 53 49 4F 2F 32 05 00  ..Serial SIO/2..
1670:  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ..............…
```

Pressing {return} again will display the next block of memory.

Pressing {escape} will exit the memory display mode and return to the monitor prompt.

Alternatively a new command can be entered without first returning to the monitor prompt.

Registers display or edit

Syntax: R [<name of register>]

This command can either display the current processor registers or edit the value of a processor register.

When no parameter is entered the current register values are displayed.

For example:

```
r {return}
PC:0001 AF:0002 BC:0003 DE:0004 HL:0005 IX:0006 IY:0007 Flags:------N-
SP:0011 AF'0012 BC'0013 DE'0014 HL'0015 (S)0016 IR:0017 Flags'---H--N-
```

More on registers later.

Input from port

Syntax: I <port address>

The specified input port address is read and the result displayed in hexadecimal.

For example:

```
I F0 {return}
00
```

If you have the RC2014 digital I/O module you can typically read the switch inputs with the command:

```
I 0 {return}
```

Press one of the buttons on the I/O module and keep it held whilst pressing the Return key at the end of the input command. If you you are holding button '1' then you should see this:

```
I 0 {return}
01
```

Output to port

Syntax: O <port address> <data byte>

The specified data byte is written to the specified output port address.

For example:

```
O F0 55 {return}
```

If you have the RC2014 digital I/O module you can typically write to the LED outputs with the command:

```
O 0 <output byte> {return}
```

For example, this command will turn on the LEDs labelled 0 and 2.

```
O 0 5 {return}
```

The value 5 is written to the LED output latch. In binary the number 5 is 00000101, thus bits 0 and 2 are ON. This results in LED 0 and LED 2 lighting up.

Writing Programs

Programs are made up of instructions. At the lowest level, the Z80 microprocessor only understands instructions, or machine code, which are a series of binary bits. These bits are divided into bytes, with a single instruction being between one and four bytes long. Each byte can be described as a decimal number from 0 to 255, or a hexadecimal number from 00 to FF. So a program is just a list of numbers. Not too friendly really.

So what numbers make up a program?

To answer this question you first need to decide what the program is going to do. This should be something you could write down and read out to anyone so that they can understand what it does.

The next step is to break down the task into simple steps, such as a numbered list of things to do or a flow chart. Again this should be in everyday language you can easily get anyone else to understand.

Each step of the task can then be converted into instructions the computer understands.

High Level Language

You can use high level computer programming languages, where the instructions are quite like normal written languages. This makes it relatively easy to convert your design into instructions the computer understands.

On classic 8-bit computers, high level language usually meant Beginner's All-purpose Symbolic Instruction Code (BASIC). Given the limitations of those early computers BASIC was a good solution. Many people knock classic BASIC but with only a few thousand bytes of memory and perhaps only cassette recorder for storage, it was not easy to provide a more desirable programming language.

Machine Code

This document is a tutorial for the Small Computer Monitor, so we are talking about Machine Code, not high level programming languages. Machine Code is fast and efficient, but not as easy to write as high level language code.

You can use the Small Computer Monitor to write binary Machine Code instructions directly to the computer's memory. This is proper hard-core Machine Code. But there are slightly easier ways to do it.

To avoid working directly in binary we have already established that hexadecimal numbers are easier for people to use, but it is still only for crazy people. Instead the instructions are usual written in Assembly Language.

Assembly Language

Assembly Language is a way of writing Machine Code instructions using a series of Mnemonics and Operands which are much easier to read, write and remember than hexadecimal numbers. These Mnemonics and Operands are converted to Machine Code by an assembler.

The Small Computer Monitor can convert Assembly Language instructions into Machine Code, but only one instruction as a time. To write large programs it is desirable to use a program called an Assembler to compile the Machine Code.

Your First Program

It is a tradition in the programming world to make your first program one that prints "Hello World!", but here we start with something simpler.

Lets just start by trying to write the character "!" to the terminal screen.

As explained in the previous section, the first thing to do is define the task. Well we did that in the line above.

Next you should break down the task onto easy steps. In a larger program you'd be lucky to even see a step mentioned which is as small as this whole task, but as this is a tutorial I'll go full Rambo on the design and break it right down.

These are the steps required to write a letter to the terminal screen:

1. Store the character's ASCII value in a variable (a processor register)
2. Call the system function which writes a character to the terminal
3. Finish the program

Step 1

The ASCII value for the character "!" is decimal 33 or hexadecimal 21. The processor has a number of special variables called registers. To prepare a character for display the character's ASCII value must be stored in the A register. The assembly language instruction to do this is:

```
LD   A, 21
```

This instruction can be read as "Load the A register with the value hexadecimal 21". Note that the assembler defaults to the operand (21) being in hexadecimal.

Step 2

The Small Computer Monitor provides a set of functions to help programs perform common tasks. This is called an Application Programming Interface (API). To use the API to display a character you need to store the value 2 in register C and then call the API. The assembly language instructions to do this are:

```
LD   C, 2
CALL 30
```

The API subroutine starts at address hexadecimal 30.

Step 3

To finish a program you use the Return instruction:

```
RET
```

The hole program written in assembly language is:

```
LD   A, 21
LD   C,2
CALL 30
RET
```

The assembler converts these assembly language instructions into binary machine code instructions. Expressed in hexadecimal this is:

```
3E 21 0E 02 CD 30 00 C9
```

The conversion process is called assembling. When assembled into memory starting at address hexadecimal 8000, the assembler output listing will usually look something like this:

```
8000: 3E 21      LD   A, 21
8002: 0E 02      LD   C, 2
8004: CD 30 00   CALL  30
8007: C9         RET
```

To enter this program into memory using the Small Computer Monitor you use the Assemble command (A). To assemble the machine code to address hexadecimal 8000 you type the command:

A 8000 {return}

To monitor then shows the current instruction at address 8000, which looks something like this:

```
8000: 00          .    NOP               >
```

In this case the instruction currently in memory at address 8000 is NOP, but it could be anything at this point. You can then type the instruction you want to assemble there:

```
8000: 00          .    NOP               > LD A,21 {return}
```

The monitor program then displays the instruction you entered in the same list format, followed by the next instruction currently found in memory:

```
8000: 3E 21       >!   LD A,$21
8002: 00          .    NOP               >
```

You can then enter the next instruction: LD C,2

Continue like this until the whole program has been entered.

To exit the assembler press the Escape key.

You can check the whole program is correct by listing the program with the Disassemble command (D):

```
D 8000 {return}
```

The following should then be displayed:

```
8000: 3E 21         >!   LD A,$21
8002: 0E 02         ..   LD C,$02
8004: CD 30 00      .0.  CALL $0030
8007: C9            .    RET
```

Press the Escape key to end the program listing, or the Return key to see the next few instructions.

The characters after the machine code bytes are the ASCII characters for each of those bytes. Thus the first instruction 3E 21, shows “>!”.

Now we have the machine code program in memory we can run it. For this we use the Go command (G):

```
G 8000 {return}
!*
```

The output, illustrated above, is the “!” character the program displays followed by the monitor prompt character “*”.

TODO

Many more examples to introduce the range of capabilities of the monitor.

Later examples to gradually become more complex.

Introduce external Assembler program with Hex file download.

Reminder to now consult the User Guide for reference and the internet for more information on programming the Z80.

Next up:
Hello World!

Fault Finding

If you do not see the monitor sign on message on the terminal when you switch the system on, then here are some things to try:

Press the RC2014 reset button.

If your RC2014 was not previously tested with the supplied BASIC ROM, then if possible check it does work with the BASIC ROM. If that is not possible then you'll need to go through all the usual fault finding process: check the power supply, check all links, check no chips have bent legs and thus not making contact with their socket, carefully inspect all soldering, check all the chips are inserted the right way round, check all the components are in the right place. Check your serial connection looks right and that the terminal is correctly set. Then cry!

If your RC2014 was known to be working with the supplied BASIC ROM, then verify the Small Computer Monitor ROM contains the correct code and check the links related to addressing the ROM (especially if the chip has a different capacity to the one containing BASIC). Other than that you would appear to have an odd problem as the Monitor ROM should, in theory, work if the RC2014 standard BASIC ROM works.

Parts and Suppliers

The following is a list of parts and suppliers used during development of the Small Computer Monitor.

RC2014 official modules

Information at www.rc2014.co.uk
Parts purchased through Tindie:
https://www.tindie.com/stores/Semachthemonkey/?ref=offsite_badges&utm_source=sellers_Semachthemonkey&utm_medium=badges&utm_campaign=badge_medium

Chip programmer

WINGONEER TL866CS Universal USB MiniPro EEPROM FLASH BIOS Programmer AVR GAL PIC SPI
Amazon ASIN: B071H5XGR7
https://www.amazon.co.uk/gp/product/B071H5XGR7/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

FTDI cable

TTL-232R-5V - USB to Serial Converter Cable, 5V, 6Way, 1.8m
Farnell order code: 2419945
http://uk.farnell.com/ftdi/ttl-232r-5v/usb-to-serial-converter-cable/dp/2419945?ost=2419945&iscrfnonsku=false&ddkey=http%3Aen-GB%2FElement14_United_Kingdom%2Fsearch

FTDI 'cable'

HALJIA FT232RL FTDI USB to TTL Serial Converter Adapter Module Mini USB 3.3V 5.5V Board for Arduino
Amazon ASIN: B06XDH2VK9
https://www.amazon.co.uk/gp/product/B06XDH2VK9/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

USB-RS232 cable

UGREEN 20210 USB Serial Cable, USB to RS232 DB9 9 pin Converter Cable
Amazon ASIN: B00QUZY4UG

https://www.amazon.co.uk/gp/product/B00QUZY4UG/ref=oh_aui_search_detailpage?ie=UTF8&psc=1
Note, you still need a null modem lead between this and the RC2014.

EEPROM 8k x 8 bit

Microchip Technology AT28C64B-15PU Parallel EEPROM Memory, 64kbit, 150ns, 4.5 → 5.5 V PDIP 28-Pin
RS part number: 127-6572
http://uk.rs-online.com/web/p/eeprom-memory-chips/1276572/

Contact Information

If you wish to contact me regarding the Small Computer Monitor please use the contact page at www.scc.me.uk (or smallcomputercentral.wordpress.com).

Issues related to the RC2014 can be posted on the RC2014-Z80 google grooup.

Stephen C Cousins, Chelmsford, Essex, United Kingdom.