

Small Computer Monitor

User Guide

**Monitor version 0.2
for the Z80 CPU**

Contents

| | |
|--|----|
| OVERVIEW..... | 3 |
| <i>Serial Port</i> | 3 |
| <i>RC2014 Systems</i> | 3 |
| COMMANDS..... | 4 |
| ? <i> or Help</i> | 4 |
| <i>Assemble instructions</i> | 4 |
| <i>Breakpoint set or clear</i> | 5 |
| <i>Devices</i> | 6 |
| <i>Disassemble instructions</i> | 6 |
| <i>Edit memory</i> | 7 |
| <i>Flags display or modify</i> | 7 |
| <i>Go to program</i> | 8 |
| <i>Input from port</i> | 8 |
| <i>Memory display</i> | 9 |
| <i>Output to port</i> | 9 |
| <i>Registers display or edit</i> | 10 |
| <i>Reset</i> | 11 |
| <i>Step one instruction</i> | 11 |
| HEX FILE LOADER..... | 13 |
| APPLICATION PROGRAMMING INTERFACE (API)..... | 14 |
| <i>Function 1, Input character</i> | 15 |
| <i>Function 2, Output character</i> | 15 |
| <i>Function 3, Input status</i> | 16 |
| <i>Function 4, Input line</i> | 17 |
| <i>Function 5, Input line default</i> | 17 |
| <i>Function 6, Output line</i> | 17 |
| <i>Function 7, Output new line</i> | 18 |
| <i>Function 8, Get version details</i> | 18 |
| <i>Function 9, Claim jump table entry</i> | 19 |
| <i>Function 10, Delay</i> | 19 |
| <i>Function 11, Output system message</i> | 20 |
| SOURCE CODE AND ASSEMBLY..... | 21 |
| <i>Small Computer Workshop</i> | 21 |
| <i>Memory Map</i> | 21 |
| TARGET HARDWARE..... | 23 |
| <i>Hardware type 1: Small Computer Simulator (Z80)</i> | 23 |
| <i>Hardware type 2: Small Computer Development Kit (Z80)</i> | 23 |

| | |
|---|----|
| <i>Hardware type 3: RC2014 (Z80)</i> | 23 |
| BUGS, QUIRKS, LIMITATIONS AND TO DO LIST..... | 24 |
| <i>Bugs</i> | 24 |
| <i>Quirks</i> | 24 |
| <i>Limitations</i> | 24 |
| <i>To do list</i> | 24 |
| HISTORY..... | 24 |
| FUTURE PLANS..... | 24 |
| CONTACT INFORMATION..... | 25 |

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.

The Monitor includes a capable debugging environment with the following features:

- Boot loader to load Intel HEX files from a PC or similar
- Memory display and editing
- Register display and editing
- In-line disassembler
- In-line assembler
- Breakpoint debugging
- Single step debugging (without the need for special hardware)

Primary input/output is via a serial port to a terminal.

Drivers and support for common hardware is included.

The monitor can be modified for different hardware and can be compiled with only the basic features in order to fit in a small ROM.

An executable file version of the Small Computer Monitor can also be compiled, thus allowing it to be run from a storage device rather than from ROM.

The monitor is implemented without using interrupts, thus leaving them free for applications and additional device drivers.

An Application Programming Interface (API) is provided to enable some of these features to be used by other software.

Serial Port

The default configuration is to use a terminal or terminal emulation software to communicate with the target hardware running the Small Computer Monitor program.

The serial port is set for 115200 bits per second, 8 data bits, 1 stop bit, no parity, no flow control. Line termination is Carriage Return (0x0D) plus Line Feed (0x0A). There should not normally be any need to add delays when sending characters to the target.

RC2014 Systems

This document describes the Small Computer Monitor as configured for RC2014 systems.

The RC2014 is an open system and can thus have diverse hardware. The Small Computer Monitor program is designed to work with the official modules, and may not work with third party modules.

The ROM version of the Small Computer Monitor fits in an 8k byte ROM which is mapped into memory from address 0x0000 to 0x1FFFF. It requires RAM from 0xFFE0 to 0xFFFF. The ROM is not paged out of memory during operation.

The Small Computer Monitor for RC2014 does not use Interrupts.

Commands

The Small Computer Monitor is designed to be used with a simple serial terminal or terminal emulation software. Commands are therefore typed in plain text and results are displayed on the terminal.

The Monitor's commands, described below, have the general syntax:

command-name parameter(s)

The command name is often just a single letter.

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.

In the examples below, 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" followed by the Return key, and the monitor displays "Breakpoint set".

```
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 an numeric character. The exception to this rule is operands in the assembler.

? 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: DEVICES, HELP, RESET
```

Assemble instructions

Syntax: A [<memory address>]

The in-line assembler is invoked by this command.

The memory address parameter is optional. If supplied the assembler begins at the specified address. If not, the last referenced address is used instead.

The address is shown in hexadecimal followed by the hexadecimal byte or bytes which make up the machine code instruction at that address. The ASCII characters represented by those bytes is then shown, followed by the instruction mnemonic. Non-printable ASCII characters are shown as a dots.

The user may then enter a new instruction mnemonic which is assembled into machine code and entered into memory at the address shown. The new instruction is then displayed in the format specified above, and the next instruction displayed.

For example:

```
A 5000 {return}
5000: C3 56 10   .V.    JP $1056      >
5003: 00        .      NOP          > ld a,12 {return}
5003: 3E 12     >.    LD A,$12
5005: 00        .      NOP          >
```

Instead of entering a new instruction, the user can press {return} to move to the next instruction, or {escape} to exit the assembler and return to the monitor prompt.

The delimiter between the operation and the first operand must be a space, while the delimiter between operands can be a comma or a space.

The assembler supports all the documented Z80 instructions and uses standard Zilog mnemonics. A good reference document is Zilog's Z80 CPU User Manual, just search for "zilog um0080".

Instruction operands can be hexadecimal or decimal numbers. The default is hexadecimal, but unlike monitor command parameters they may need clarification. For example, the hexadecimal number BC is also the name of the register pair BC.

To clarify that a hexadecimal number is intended it is necessary to ensure the first character is numeric (ie. 0 to 9, not A to F). So prefixing BC with a zero clarifies it is intended to be a hexadecimal number.

Alternatively a hexadecimal number can be prefixed with a "\$" or "0x" to ensure it is interpreted correctly. Thus "\$BC" and "0xBC" are the hexadecimal number BC.

To identify a number as decimal, prefix with a "+" sign.

For example:

```
LD A,B      = Load register A with register B
LD A,0B     = Load register A with hexadecimal value 0B (decimal 11)
LD A,$B     = Load register A with hexadecimal value 0B (decimal 11)
LD A,0xB    = Load register A with hexadecimal value 0B (decimal 11)
LD A,+11    = Load register A with decimal value 11 (hexadecimal 0B)
```

When entering the address for a relative jump, the address can be either the displacement or the absolute address. An address of 0x00 to 0xFF is treated as a displacement, while an address of 0x0100 to 0xFFFF is treated as an absolute address. Absolute addresses must be within range of a relative jump or the wrong address will be shown.

For example:

```
a 5010 {return}
5010: 00        .      NOP          > jr 30
5010: 18 30     .0    JR $30   (to $5042)
5012: 00        .      NOP          > jr 5010
5012: 18 FC     ..    JR $FC   (to $5010)
5014: 00        .      NOP          > jr 9999
5014: 18 FC     ..    JR $83   (to $4F99)
5016: 00        .      NOP          >
```

One little quirk with the assembler syntax is that the Restart instruction parameter must be a two digit hexadecimal number and must not be prefixed with "\$" or "0x". Thus the instruction "RST 08" is valid, while "RST 8" and "RST \$08" are not.

Breakpoint set or clear

Syntax: B [<memory address>]

Breakpoints provide an aid to program debugging. When a program is running and reaches a breakpoint, program execution stops and the current state of the processor registers is displayed. Register values can be altered if required, and execution continued.

This command enables the breakpoint to be set or cleared.

To set the breakpoint, enter the command 'B' followed by the address at which the breakpoint should be set. The breakpoint can only be set in random access memory (RAM), not in read only memory (ROM). Only one breakpoint is provided.

For example:

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

To clear the breakpoint, enter the command 'B' with or without an address. If no address is specified the current breakpoint is cleared. If an address is specified the current breakpoint is cleared and then the new breakpoint is set.

For example:

```
B {return}  
Breakpoint cleared
```

To continue execution after a breakpoint, use the with the "Go" command without specifying an address. Just enter *G {return}*.

Breakpoints (and single stepping) work by replacing the instruction in memory with the instruction RST 0x28. This causes a call to the Monitor to handle the breakpoint (or step). If a RST 0x28 instruction is encountered which is not there as a breakpoint or step instruction, program execution stops and "Trap" is displayed.

Devices

Syntax: Devices

A list is displayed of hardware devices detected by the Small Computer Monitor when it started up.

For example:

```
Devices {return}  
Serial ACIA
```

The Small Computer Monitor can detect the presence of some hardware devices and includes driver software to support them.

The real reason for this feature is to support both types of serial interfaces supplied as official RC2014 modules, namely the classic 6850 ACIA and the newer Z80 SIO/2. By detecting these modules the Small Computer Monitor can configure itself to use whichever module is available and thus avoid the need for two versions of this program.

In addition to using whichever serial module is detected for its own input and output, the Small Computer Monitor passes on this benefit to any software which makes use to the Monitor's API.

Currently supported devices:

- Serial ACIA (6850 family)
- Serial SIO/2 (Z80 peripheral)

When the ACIA device is detected the Small Computer Monitor initialises channel A for 115200 bits per second, 8 data bits, 1 stop bit, no parity, no flow control, no interrupts. It is then configured as the console input and output device.

When the SIO/2 device is detected the Small Computer Monitor initialises channel A for 115200 bits per second, 8 data bits, 1 stop bit, no parity, no flow control, no interrupts. It is then configured as the console input and output device. Channel B is left uninitialised and completely free for the user.

Disassemble instructions

Syntax: D [<memory address>]

The in-line disassembler is invoked by this command.

The memory address parameter is optional. If supplied the disassembler begins at the specified address. If not, the last referenced address is used instead.

The address is shown in hexadecimal followed by the hexadecimal byte or bytes which make up the machine code instruction at that address. The ASCII characters represented by those bytes is then shown, followed by the instruction mnemonic. Non-printable ASCII characters are shown as a dots.

For example:

```
d 1066 {return}
1066: C3 03 FF      ...      JP $FF03
1069: 31 80 FE      1..      LD SP,$FE80
106C: 11 00 00      ...      LD DE,$0000
106F: 21 00 10     !..      LD HL,$1000
1072: 01 69 00      .i.      LD BC,$0069
1075: ED B0        ..       LDIR
```

After the block of instructions is shown, the user can press {return} to display the next block, or {escape} to exit the disassembler and return to the monitor prompt. Alternatively, a new command can be entered without first returning to the monitor prompt.

The disassembler supports all the documented Z80 instructions and uses standard Zilog mnemonics.

When disassembling a relative jump instruction, both the displacement and the absolute address is shown:

```
d 5010 {return}
5010: 18 30          .0       JR $30 (to $5042)
```

Edit memory

Syntax: E [<memory address>]

The memory editor is presented in a similar way to the assembler, but instead of entering instruction mnemonics you enter hexadecimal or ASCII values.

The memory address parameter is optional. If supplied the editor begins at the specified address. If not, the last referenced address is used instead.

The address is shown in hexadecimal followed by the hexadecimal byte or bytes which make up the machine code instruction at that address. The ASCII characters represented by those bytes is then shown, followed by the instruction mnemonic. Non-printable ASCII characters are shown as a dots.

The user may then enter new memory contents, press {return} to move to the next instruction, or {escape} to exit the editor and return to the monitor prompt. Entering “^” followed by the return key causes the editor to go back one location.

Hexadecimal numbers are entered without the need to clarify as hexadecimal. ASCII characters are entered by preceding with a quote character.

For example:

```
e 5040 {return}
5040: 00          .      NOP          > 3e ff
5042: 00          .      NOP          > "Hellp
5047: 00          .      NOP          > ^
5046: 70          p      LD (HL),B    > "o
5047: 00          .      NOP          >
```

Flags display or modify

Syntax: F [<status>]

The processor's flags are displayed or modified.

If the command is entered without any parameters the processor's registers, including the flags register, is displayed:

```
f {return}
PC:0001 AF:00D7 BC:0003 DE:0004 HL:0005 IX:0006 IY:0007 Flags:SZ-H-PNC
```

The flags register is shown in hexadecimal as part of the AF register pair and also as individual flag bits. If the flag letter is shown, the flag is set, otherwise it is clear.

If a valid parameter is entered the appropriate flag bit is set or cleared.

Valid parameters include the flag letter (eg. "Z"), the flag letter prefixed with "N" (eg. "NZ") and the 'condition' in conditional instructions (eg. "PO" in "JP PO, <address>"). Thus there are several ways to reference the flags. Due to a conflict between Positive (P) and Parity (P), the parity flag is set with "Pa", not just "P".

The table below shows valid flag names in square brackets and valid condition names in curly brackets and valid .

| Flag | Description | Set / Clear | Set meaning | Clear meaning | Bit |
|------|------------------------------|-------------|---------------------------|-----------------------------------|-----|
| S | Sign | [S] [NS] | Negative {M} | Positive {P} | 7 |
| Z | Zero | [Z] [NZ] | Zero {Z} | Not Zero {NZ} | 6 |
| H | Half carry or Half borrow | [H] [NH] | Half carry Half borrow | Not Half carry Not Half borrow | 4 |
| P | Parity or Overflow (V) | [Pa] [NP] | Even {PE} | Odd {PO} | 2 |
| N | Add/subtract | [N] [NN] | Subtract | Add | 1 |
| C | Carry | [C] [NC] | Carry {C} | No Carry {NC} | 0 |

For example, to set the zero flag:

```
f z {return}
PC:0001 AF:0042 BC:0003 DE:0004 HL:0005 IX:0006 IY:0007 Flags:-Z---N-
```

Go to program

Syntax: G [<memory address>]

This command allows a machine code program to be executed (run).

Program execution begins at the specified address. If no address is specified execution begins at the address set in the PC variable, as displayed by the Register command. All other processor registers are set to the values stored in the associated variables, again as displayed by the Register command.

If the program being executed is written as a subroutine, whereby it ends with a RET (return) instruction, then at the end of the program control is passed back to the monitor and a monitor prompt is displayed. If the program does not return then control does not pass back to the monitor until the system is reset or a breakpoint is encountered. Of course if the program has a problem and crashes then anything can happen!

In the example below the test program runs until the breakpoint is reached.

```
a 8000 {return}
8000: 00      .      NOP                > Ld a,55 {return}
8000: 3E 55   >.     LD A,$55
8002: 00      .      NOP                > ret {return}
8002: C9      .      RET
```

```
d 8000 {return}
8000: 3E 55   >.     LD A,$55
8002: C9      .      RET
```

```
r {return}
PC:0000 AF:0002 BC:0003 DE:0004 HL:1234 IX:0006 IY:0007 Flags:-----N-
SP:FE7E AF'0012 BC'0013 DE'0014 HL'0015 (S)0016 IR:0017 Flags'---H--N-
```

```
b 8002 {return}
Breakpoint set
```

```
g 8000 {return}
PC:8002 AF:5502 BC:0003 DE:0004 HL:1234 IX:0006 IY:0007 Flags:-----N-
8002: C9      >.     RET
```

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
```

The Z80 has a separate address range for input/output (I/O) devices, together with separated processor instructions to access them. This command addresses devices in the I/O space, not the memory space. Generally the I/O space is limited to 256 address (0x00 to 0xFF).

Users of the official RC2014 digital I/O card can typically read the switch inputs with the command:

```
I 0 {return}
```

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 1080 {return}
1080: 19 10 F9 3E 00 11 AC 10 CD 83 19 3E 01 11 AE 10 ...>.....>....
1090: CD 83 19 CD C1 24 CD 90 17 11 B0 10 CD 50 18 CD .....$......P..
10A0: D8 24 31 80 FE CD 3F 11 C3 22 12 C9 ED 4D ED 45 .$1...?.."...M.E
10B0: 05 05 53 6D 61 6C 6C 20 43 6F 6D 70 75 74 65 72 ..Small Computer
10C0: 20 4D 6F 6E 69 74 6F 72 20 62 79 20 53 74 65 70 Monitor by Step
10D0: 68 65 6E 20 43 20 43 6F 75 73 69 6E 73 05 56 65 hen C Cousins.Ve
10E0: 72 73 69 6F 6E 20 30 2E 31 2E 32 20 66 6F 72 20 rsion 0.1.2 for
10F0: 00 50 43 3A 2C 41 46 3A 2C 42 43 3A 2C 44 45 3A .PC:,AF:,BC:,DE:
```

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.

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}
```

The Z80 has a separate address range for input/output (I/O) devices, together with separated processor instructions to access them. This command addresses devices in the I/O space, not the memory space. Generally the I/O space is limited to 256 address (0x00 to 0xFF).

Users of the official RC2014 digital I/O card can typically write to the LED outputs with the command:

```
I 0 5 {return}
```

In the above example 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.

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-
```

The first line shows the most commonly used register:

- PC Program Counter
- AF Accumulator (A) and flags (F)
- BC Register pair BC
- DE Register pair DE
- HL Register pair HL
- IX Index register IX
- IY Index register IY

Flags register broken down into individual flag bits:

- S Sign flag
- Z Zero flag
- H Half carry flag
- P Parity/overflow flag
- N Add/subtract flag
- C Carry flag

If the flag letter is shown, the flag is set, otherwise is it clear.

The second line shows the rest of the registers:

- SP Stack Pointer
- AF' Alternative accumulator and flags
- BC' Alternative register pair BC
- DE' Alternative register pair DE
- HL' Alternative register pair HL
- (S) Contents of the stack pointer
- IR Interrupt vector register (I) and memory refresh register (R)

Flags' is the alternative flags register broken down into bits (as above)

When the optional parameter is entered, the specified register can be edited.

For example:

```
r hl {return}
HL: 0005 1234 {return}
```

In the above example the HL register pair is specified. The current value of HL is displayed (0005) and the user can either enter a new value for HL (1234), or press {escape} to leave the register unchanged. Entering the command "r" now shows the updated registers:

```
r {return}
PC:0001 AF:0002 BC:0003 DE:0004 HL:1234 IX:0006 IY:0007 Flags:-----N-
SP:0011 AF'0012 BC'0013 DE'0014 HL'0015 (S)0016 IR:0017 Flags'---H--N-
```

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.

All execution stops, including interrupt routines. The monitor then restarts.

Memory is not cleared, but essential variables are initialised.

The Small Computer Monitor outputs a sign-on message to the console output device (usually a terminal). For example:

```
Small Computer Monitor by Stephen C Cousins
Version 0.1.0 for Z80 based RC2014 systems
```

The second line gives details of the code's version number and the target hardware it supports.

Step one instruction

Syntax: S [<memory address>]

This command allows single stepping of machine code programs.

Program execution begins at the specified address. If no address is specified execution begins at the address set in the PC variable, as displayed by the Register command. All other processor registers are set to the values stored in the associated variables, again as displayed by the Register command.

A single instruction is executed, and then the processor registers and flags are displayed. A press of the Return key causes the next instruction to be executed.

Pressing {return} again will step another instruction and again display the processor state.

Pressing {escape} will exit single stepping mode and return to the monitor prompt.

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

Below is an example of a simple program being stepped.

```
s 8000 {return}
PC:8000 AF:0040 BC:125C DE:FF86 HL:8000 IX:2034 IY:0007 Flags:-Z-----
8000: 3E 02      >.  LD A,$02
PC:8002 AF:0240 BC:125C DE:FF86 HL:8000 IX:2034 IY:0007 Flags:-Z-----
8002: CD 10 80   ...  CALL $8010
PC:8010 AF:0240 BC:125C DE:FF86 HL:8000 IX:2034 IY:0007 Flags:-Z-----
8010: 3C        <    INC A
PC:8011 AF:0300 BC:125C DE:FF86 HL:8000 IX:2034 IY:0007 Flags:-----
8011: C9        .    RET
PC:8005 AF:0300 BC:125C DE:FF86 HL:8000 IX:2034 IY:0007 Flags:-----
8005: C9        .    RET
```

No special hardware is required for single stepping as it is all handled in software. However, like breakpoints, single stepping can only occur when the code is in RAM. If a call is made to a routine in read only memory, the call is stepped over. Single stepping then continues after the call is complete.

Similarly, calls into the monitor code are stepped over. This means calling the Monitor's API appears as just one step and not hundreds.

When a call into read only memory or monitor code occurs, the message: "Stepping over code in ROM or in monitor" is displayed.

Single stepping (and breakpoints) work by replacing the instruction in memory with the instruction RST 0x28. This causes a call to the monitor to handle the step (or breakpoint). If a RST 0x28 instruction is encountered which is not there as a breakpoint or step instruction, program execution stops and "Trap" is displayed. Single stepping can not pass a "Trap" instruction.

Hex File Loader

The Small Computer Monitor includes a means of receiving an Intel Hex File from the the serial port.

This allows an assembly language program to be created, edited and assembled on a PC (or similar), and then 'sent' to the target computer from a terminal program.

Terminal programs usually have a feature called "Send text file" (or similar). This opens a file on the PC and 'sends' it to the target as if it were typed in on the terminal.

Assemblers usually have a means of creating an Intel Hex File, thus it is quick and easy to author a program on a PC and send it to target hardware running the Small Computer Monitor.

Below is an example of an Intel Hex File.

```
:1040000021354006090E80EDB3061A3E41F5DB80EE  
:10401000CB5728FAF1D3813CCD2A4010F0DB80CB7E  
:104020004728FADB81D38118F4C9F5C50E0911645C  
:104030000F7C1F1C91814C403C105681100FFFDE  
:00000001FF
```

There is no need to set the terminal to add any delays as it sends the characters, as the monitor can handle hex file loading at a continuous 115200 bits per second when the processor is running at 7.3 MHz.

Application Programming Interface (API)

The monitor provides an Application Programming Interface (API) to enable other software to use some of its features.

The following functions are available:

0. System reset
1. Input character
2. Output character
3. Input status
4. Input line
5. Input line default
6. Output line
7. Output new line
8. Get version details
9. Claim jump table entry
10. Delay in milliseconds
11. Output system message

To access these functions from a program, a CALL is made to address \$0030ⁿ with the function number in the C register. Any parameters and results are passed in other registers.

Unless otherwise stated, calling any of these functions may result in registers AF, BC, DE and HL being modified and therefore they may not contain the same value after the function call as they did before the call. Registers IX IY I AF' BC' DE' HL' are not modified (unless otherwise stated).

As a simple example, function zero is used to reset the system:

```
8000: 0E 00    LD C,$00    ; Function 0 = System reset
8002: CD 30 00  CALL $0030  ; Call API
```

The Z80 processor has a special instruction to CALL a few specific addresses in memory. This is called a Restart instruction and has the mnemonic RST.

RST 30 performs the same function as CALL \$0030, but it does it with a single byte instruction. It is therefore smaller and faster.

The above code can therefore be improved:

```
8000: 0E 00    LD C,$00    ; Function 0 = System reset
8002: F7      RST 30     ; Call API
```

ⁿ Address 0x0030 and RST 30 only apply if the Small Computer Monitor located at the bottom of memory or the bottom of memory is RAM allowing the Monitor to modify these locations.

If the Small Computer Monitor is not located at the bottom of memory, then the API call address is the start of the Monitor + 0x0030. Thus if the monitor starts at address 0xE000, the API call address is 0xE030.

However, if the monitor can write to lower memory, due to RAM being located there and not ROM, then it is able to write the relevant jump instruction to location 0x0030. In which case the CALL 0x0030 or RST 30 instructions can still be used.

Function 0, System reset

Parameters: none

Returns: none

This function 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.

All execution stops, including the user program. The monitor then restarts.

Memory is not cleared, but essential variables are initialised.

Example:

```
8000: 0E 00    LD C,$00    ; Function 0 = System reset
8002: F7      RST 30     ; Call API
```

This function is equivalent to the Monitor's RESET command.

The Small Computer Monitor outputs a sign-on message to the console output device (usually a serial terminal). For example:

```
Small Computer Monitor by Stephen C Cousins
Version 0.1.0 for Z80 based RC2014 systems
```

The second line gives details of the code's version number and the target hardware it supports.

Function 1, Input character

Parameters: none

Returns: A = Character input from console

This function waits for a character from the current console input device, usually a serial terminal.

When a character arrives, it is returned in the A register.

The function does not return until a character arrives.

Example:

```
8000: 0E 01    LD C,$01    ; Function 1 = Input character
8002: F7      RST 30     ; Call API

8003: C9      RET      ; Return to monitor
```

Function 2, Output character

Parameters: A = Character to output to console

Returns: A = Character output to console

This function outputs the specified character to the current console output device, usually a serial terminal.

The ASCII value of the character to be output is passed in the A register.

The function does not return until the character has been output.

Example:

```
8000: 3E 21    LD A,'!'    ; ASCII value of character '!'  
8002: 0E 02    LD C,$02    ; Function 2 = Output character  
8004: F7      RST 30      ; Call API  
  
8005: C9      RET        ; Return to monitor
```

The above example causes a pling character ('!') to be output to the console.

By combining this example with the previous example, we can input a character from the console and then output it to the console, thus enabling us to see what we have typed. The example below repeats this process until the Return key is pressed.

```
8000: 0E 01    LD C,$01    ; Function 1 = Input character  
8002: F7      RST 30      ; Call API  
; Register A now contains the input character  
; which is then output to the console  
8003: 0E 02    LD C,$02    ; Function 2 = Output character  
8005: F7      RST 30      ; Call API  
; Now repeat until the Return key is pressed  
8006: FE 0D    CP $0D      ; Is the character a Return ?  
8008: 20 F6    JR NZ,$8000 ; No, so repeat  
800A: C9      RET        ; Yes, so exit program
```

Function 3, Input status

Parameters: none

Returns: NZ flagged in input character available from console

This function checks the status of the current console input device, usually a serial terminal.

When a character is available the function returns with NZ flagged, otherwise Z is flagged.

The function does not wait until a character arrives.

Example:

```
8000: 0E 03    LD C,$03    ; Function 3 = Input status  
8002: F7      RST 30      ; Call API  
  
8003: C9      RET        ; Return to monitor
```

Function 4, Input line

Parameters: DE = Start of line in memory
A = Size of line in memory
Returns: DE = Start of line in memory
A = Number of characters in line

This function inputs a line of characters from the console input device.

The line contains any characters entered from the console input device until a Carriage Return character (ASCII \$0D) is entered. The line ends when the Carriage Return is entered but does not include the Carriage Return character.

The line is input to memory starting at DE and is returned as a null (zero) terminated list of characters. Register A sets the maximum size of the line in bytes, which must include one byte for the null terminator.

On return the null (zero) terminated line of characters is stored in memory at the requested location. This location is returned in DE. Register A contains the number of characters in the line, excluding the null terminator.

For example:

```
8000: 11 00 90    LD DE,$9000    ; Start of line = $9000
8003: 3E 51      LD A,$51        ; Size of line = $50 + 1 for terminator
8005: 0E 04      LD C,$04        ; Function 4 = Input line
8007: F7        RST 30          ; Call API

8008: C9        RET            ; Return to monitor
```

After the line "hi" is entered, DE is \$9000, A is \$02 and the memory contains:

```
9000: 68 69 00
```

Function 5, Input line default

Parameters: none
Returns: DE = Start of line
A = Number of characters in line

Input a line of characters from the console input device to the default line buffer.

The function is similar to function 4, above, except the memory used is the memory reserved for the monitor's own line input. As a result the line will be overwritten when characters are entered at the monitor prompt.

The benefit of using this function is that it does not require its own memory allocation and the call is simpler:

```
8000: 0E 05      LD C,$05        ; Function 5 = Input line default
8002: F7        RST 30          ; Call API

8003: C9        RET            ; Return to monitor
```

Function 6, Output line

Parameters: DE = Start of line in memory

Returns: none

This function outputs the specified line of characters to the output device.

The line of characters must be null (zero) terminated. The line is therefore in the same format as the input line obtained with functions 4 and 5.

```
8000: 11 00 90   LD DE,$9000   ; Start of line = $9000
8003: 0E 06     LD C,$06      ; Function 6 = Output line
8005: F7       RST 30        ; Call API

8006: C9       RET          ; Return to monitor

9000: 68 69 00   DB "hi",0     ; Line = "hi"
```

To embed a new line code into the string it is best to avoid specifically using Carriage Return and/or Line Feed as the sequence varies with different hardware. The monitor provides a special character to signify a new line. This is the constant `kNewLine` which has the value 5. The above example can be modified to include a cursor move to the start of the next lines:

```
9000: 68 69 05   DB "hi",5,0   ; Line = "hi" + new line
9003: 00
```

Function 7, Output new line

Parameters: none

Returns: none

This function outputs a new line character sequence to the console output device.

This will cause the console's cursor to move to the start of the next line. The character or characters output will depend on the console device. Typically this will be a Carriage Return and Line Feed (ASCII \$0D, \$0A).

```
8000: 0E 07     LD C,$07      ; Function 7 = Output new line
8002: F7       RST 30        ; Call API

8003: C9       RET          ; Return to monitor
```

Function 8, Get version details

Parameters: none

Returns: D, E and B are the monitor code version
D = Major, E = Minor, B = Revision
H, L are the target hardware ID
H = Primary ID, L = Options
A and C are reserved for future use, currently zero

The function returns details of the monitor code version and the target hardware identifier.

The monitor code version is detailed in registers D, E and B. Register D contains the major version number, E contains the minor version number and B the revision number. For example, if D=1, E=2 and B=3, the version is 1.2.3

The target hardware identifier is detailed in registers H and L. Register H is the primary identifier, with register L describing options within the target hardware.

Target hardware identifiers currently assigned:

H = 1 Small Computer Simulator
L = 0
H = 2 Small Computer Development Kit
L = 0
H = 3 RC2014-Z80
L, bit 0 set if serial 6850 ACIA detected
L, bit 1 set if serial SIO/2 detected
L, bit 2 to 7 = Not currently used, all cleared to zero

Function 9, Claim jump table entry

Parameters: A = Entry number (0 to n)
DE = Address of function code
Returns: none

Some system features, such as console in and out, are redirected through a jump table in RAM. This function enables these jump table entries to be set to jump to any code required.

The following jump table entries are available:

0. Initialisation (reserved, not currently used)
1. Interrupt handler
2. Non-maskable interrupt handler
3. Console input routine
4. Console output routine

By changing the console input and output routines, any custom hardware can be integrated into the monitor and used instead of the standard hardware.

If the monitor program is in ROM and that ROM is mapped to the very bottom of memory, it is not possible to directly change the interrupt code for interrupt mode 1 and non-markable interrupts. This is because code for these interrupts must begin at fixed locations in memory which happen to be very near the bottom of memory, just where the ROM is! By redirecting these interrupts through the jump table it is possible to configure your own interrupt handler.

In the example below, the console output routine is replaced by a new routine at location \$9000.

```
8000: 3E 04    LD A,$04    ; Console output jump table entry
8002: 11 00 80  LD DE,$9000 ; Console output routine at $9000
8005: 0E 09    LD C,$09    ; Function 9 = Claim jump
8007: F7      RST 30     ; Call API

8008: C9      RET        ; Return to monitor
```

Function 10, Delay

Parameters: DE = Number of milliseconds delay

Returns: none

This function simply waits for the specified number of milliseconds to pass before returning.

The delay is created by a simple software loop and during this time the processor is not looking for key presses or any other activity. Therefore, unless the system is using interrupts to capture such actions, they will be missed.

The delay time is dependent on the clock speed of the processor. If a non-standard clock speed is used the delay time will be some multiple or fraction of DE milliseconds.

For the RC2014-Z80 system the standard clock speed is 7.3728 MHz.

Function 11, Output system message

Parameters: A = Message number (0 to n)

Returns: none

This function outputs messages embedded in the Small Computer Monitor to the console output device.

The following messages are available:

0. Message = None (null)
1. Message = "Feature not included"
2. Message = "Bad command"
3. Message = "Bad parameter"
4. Message = "Syntax error"
5. Message = "Breakpoint set"
6. Message = "Breakpoint cleared"
7. Message = "Unable to set breakpoint here"
8. Message = Help text

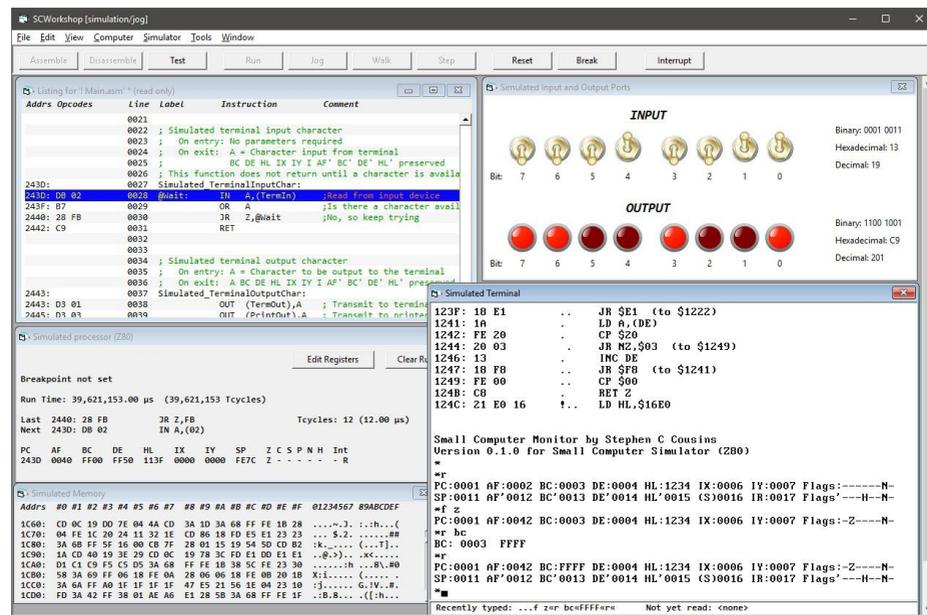
All the message end with a new line character sequence, except for message zero. Message zero outputs nothing.

Source Code and Assembly

The source code for the Small Computer Monitor is freely available. Assembling it however may not be so easy.

Small Computer Workshop

The source code has been written specifically within the Small Computer Workshop integrated development environment (IDE), which includes its own editor, assembler and simulator, as illustrated below.



When writing the development environment a study was made of currently available assemblers. Unfortunately what the study revealed was they there is not one common standard, meaning source code written for one assembler would not usually assemble using a different assembler.

Initially it was hoped that an assembler could be designed which could cope with all variations of syntax and conventions. This however was deemed unrealistic due to the extra code complexity it created and the problem caused by some mutually exclusive features.

The final decision was to write the system in the way that provided the most desirable features without creating unnecessary complications. The resulting assembler is, ironically, not totally compatible with any other and thus exasperates the very compatibility problem originally identified. However, it is also not hampered by any restrictions such compatibility would have imposed.

The assembler has a 'C' style pre-processor which includes some special statements relating to the development environment, such as #TARGET. It also supports a range of IF.. ELSE.. END and INCLUDE statements to allow conditional assembly, which is how it supports different builds of functionality options and target hardware options, from the same source code.

Assembler directives are prefixed with a period ('.'), such as ".ORG". Assembler directives include support for memory sections using ".CODE" and ".DATA".

Local labels are supported. These begin with the '@' character. These labels are local to the most recent global label, allowing local labels to be reused in other functions. There are quite a few local versions of the label "@Loop" in the source code!

Hexadecimal values are prefixed with "0x".

Memory Map

The Small Computer Monitor's code can be assembled to almost anywhere in memory and so can its workspace (in RAM). However, there are some locations in the bottom of memory which have fixed uses and can not be moved.

The standard build places the monitor code starting at 0x0000 and currently extends to about 5 kbytes. Workspace and stack is at the very top of memory from about 0xFE00 to 0xFFFF.

The fixed locations at the bottom of memory are determined by the Z80's special features, namely hardware reset, restart instructions and interrupt service. These addresses can fall within the monitor ROM or be in RAM.

These locations are:

| | |
|--------|---|
| 0x0000 | Reset monitor or CP/M 2 warm boot |
| 0x0003 | Reserved for CP/M 2, IOBYTE |
| 0x0004 | Reserved for CP/M 2, drive and user numbers |
| 0x0005 | Reserved for CP/M 2, FDOS entry point |
| 0x0008 | Restart 08, not currently used |
| 0x0010 | Restart 10, not currently used |
| 0x0018 | Restart 18, not currently used |
| 0x0020 | Restart 20, not currently used |
| 0x0028 | Restart 28, used for debugging breakpoint |
| 0x0030 | Restart 30, API entry point |
| 0x0038 | Restart 38, interrupt mode 1 handler |
| 0x005C | Reserved for CP/M 2, default FCB (to 0x007F) |
| 0x0066 | NMI interrupt handler (or CP/M 2 default FCB) |
| 0x0080 | Reserved for CP/M 2, DMA buffer (to 0x00FF) |

If the monitor code is assembled to the bottom of memory, these special locations are included in the monitor code and are thus set up as required. If the monitor is assembled elsewhere in memory these locations must be in RAM and are written to during initialisation to establish the required contents.

Target Hardware

The Small Computer Monitor can be built for a range of Z80 based computer hardware and simulators.

The Monitor code can be run from ROM or RAM, and can be assembled to any desired location. However, to fully utilise features such as its Application Programming Interface (API) and Breakpoints it must either be running at the bottom of memory (from location 0x0000 upwards) or the bottom of memory must be in RAM to allow the monitor to write Jump instructions to key locations in the range 0x0000 to 0x0068. Very simple Z80 hardware is likely to meet this requirement by having ROM at location 0x0000. More sophisticated hardware will likely have ROM at location 0x0000 but allow it to be paged out and replaced by RAM once the system is up and running.

The only other hardware requirement is an interface to a terminal or a computer running terminal software. This interface will normally be a simple serial port. The terminal only needs to be a simple Teletype style device.

The hardware does not need to support interrupts as the Monitor does not currently use them. Thus interrupts are free for user programs and additional device drivers.

Currently the Monitor's full feature set occupies just over 5 kbytes of code space and requires about 400 bytes of RAM (including stack space). The code size can be significantly reduced by excluding features like the in-line assembler.

Hardware type 1: Small Computer Simulator (Z80)

This 'hardware' is a software simulation only. It forms part of the Small Computer Workshop's integrated development environment (IDE). This is used to write and test hardware independent software.

Hardware type 2: Small Computer Development Kit (Z80)

The kit is in-house development hardware only.

Hardware type 3: RC2014 (Z80)

Type 3 hardware covers RC2014 systems available from "Semachthemonkey" via Tindie: <https://www.tindie.com/stores/Semachthemonkey/>

Z80 based RC2014 systems are supported provided they meet the ROM/RAM requirements above and use one of the two standard RC2014 serial I/O modules as the primary connection to a terminal or terminal emulation software.

As all the currently available complete RC2014 kits (Mini, Classic, Plus and Pro) meet these requirements the Small Computer Monitor is compatible with all of them. All these kits provide a means of running the Small Computer Monitor from ROM, whilst some also provide configurations suitable to run the Small Computer Monitor in RAM.

The standard RC2014 build is designed to run from ROM starting at address 0x0000. It is an 8 kbyte ROM image which supports both standard serial I/O modules.

Serial modules supported:

- Serial I/O Module (68B50 or 63B50 ACIA)
- Dual Serial Module SIO/2

The Small Computer Monitor auto detects these modules so one version of the Monitor code supports both modules, although you can only have one of these in the system at a time.

For information about RC2014 systems visit RC2014.co.uk

Bugs, Quirks, Limitations and To do list

This section details all those embarrassing little issues that make software development such fun and mean software it is never really finished.

Bugs

The outstanding bugs are currently:

B0001 First bug in released version yet to be found. Won't be long!!

Quirks

Q0001 The assembler's Restart instruction parameter must be a two digit hexadecimal number and must not be prefixed with "\$" or "0x". Thus the instruction "RST 08" is valid, while "RST 8" and "RST \$08" are not.

Q0002 The assembler and disassembler's handling of instructions JP (HL), JP (IX) and JP (IY) does not follow standard Z80 mnemonics. These are actually input and output as JP HL, JP IX and JP IY.

Limitations

L0001 The Intel hex file loader does not currently check the integrity of the data by testing the file's checksum values.

L0002 The assembler does not trap all cases where parameter values are too large and those it does trap are only indicated by the message "Syntax error".

L1003 The assembler and disassembler only handles the official Zilog documented Z80 instructions.

To do list

T0001 Fix the bugs!

T0002 Consider addressing the Quirks and Limitations

T0003 Expose more functionality via the API

History

This section documents the release history of the Small Computer Monitor.

| <i>Date</i> | <i>Version</i> | <i>Description</i> |
|-------------|----------------|---------------------------------|
| 2017-11-18 | v0.2.1 | Preview release, needs testing. |

Future Plans

It is my intention to continue development of the Small Computer Monitor; to fix bugs, add additional hardware support and increase functionality.

Possible future additions:

- Conditional breakpoints
- Logging of events just prior to breakpoint
- Simple filing system
- Command to launch operating systems such as CP/M
- Command to launch other ROM based software such as BASIC
- Scripting language to write small programs

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 group.

Stephen C Cousins, Chelmsford, Essex, United Kingdom.