ZARC MONITOR USERS' GUIDE

Written by Merlin Skinner-Oakes, 8/5/2022

Last updated .

1 DESCRIPTION

The Z80 Anachronistic Retro Computer (ZARC) monitor programme is an initial operating system for the ZARC computer. Features include:

- Memory examination and modification.
- Debugger.
- Serial port I/O drivers.
- Memory card sector read / write.
- Real time clock read and write.
- Utility routines.
- CP/M support.

In this document, the following style conventions are used:

```
Machine output
User commands
Comments
```

2 OPERATING PRINCIPLES

It is important to note that the monitor presents an environment in which the entire Z80 64 KB address space is addressable by the user. This allows debugging of the entire CP/M environment including BIOS and transient programmes, for example. The monitor is paged out of the memory map while the application programme is active.

Memory mapping should normally be enabled. If not, the application code will only have access to page 0x3f (the highest page). Write protection and supervisor are enabled. This page is used by the monitor, so great care is required to preserve the system. The command to enable memory mapping is:

```
set mmap 1
```

3 COMMAND INTERFACE

Commands may be entered at the monitor command prompt "> ". The following control keys may be used:

Key	***Function***
DEL	Delete the character to the left of the cursor.
^P	Recall previous command in history.
^N	Recall next command in history.

The ^P and ^N history functions are similar to those implemented by most Linux / Unix shells. Commands may be abbreviated with '.', so the command:

disassemble

may be entered as any of the following, for example:

```
disassemble
disass.
dis.
di.
```

3.1 Commands

3.1.1 Help

Displays a summary of the available commands.

Syntax: help

3.1.2 Dump

Display an area of memory in hex and ASCII.

Syntax: dump <start addr> [<length>]

<length> defaults to 0x0040 if not specified.

The format is similar to Linux's “hexdump -C”. Duplicate lines are omitted, and the omission is indicated by an ellipsis.

Example:

```
> dump 1000
1000  DA ED 09 23 18 F4 AF BE  |...#....|
1008  C2 E4 09 CD FD 32 CD 5D  |.....2.]|
1010  2A 2A 2A 2A 2A 2A 2A 2A  |********|
                ...
1020  19 7E D3 07 E1 E5 7C E6  |.~....|.|
```

```
1028  3F C6 C0 67 CD 5D 18 77  |?..g.].w|
1030  E1 23 18 E0 CD 8C 1B DA  |.#......|
1038  E4 09 78 A7 C2 E4 09 D5  |..x.....|
```

3.1.3 Disassemble

Disassemble an area of memory.

Syntax: disassemble [<start addr>] [<length>]

<start addr> defaults to the current PC if not specified.

<length> defaults to 0x0010 if not specified.

Example:

```
> dis. ed30
ED30  31 2D F9     ld sp, 0xF92D
ED33  CD B8 F1     call 0xF1B8
ED36  AF           xor a
ED37  32 03 00     ld (0x0003), a
ED3A  32 04 00     ld (0x0004), a
ED3D  3E 2C        ld a, 0x2C
ED3F  32 5C F4     ld (0xF45C), a
```

3.2 Read

Read a single memory location.

Syntax: read <addr>

Example:

```
> read ed30
31
```

3.3 Write

Write data to memory.

Syntax: write <addr> <data1> [<data2> <data3> …]

Example:

```
> write 100 12 34 56 78
0004 bytes written
> dump 100 8
0100  12 34 56 78 F6 00 00 00  |.4Vx....|
```

3.4 Fill

Fill a block of memory with a byte value.

Syntax: fill <addr> <length> <data>

Example:

```
> fill 100 8 2a
> dump 100 10
0100  2A 2A 2A 2A 2A 2A 2A 2A  |********|
0108  00 00 00 00 00 00 00 00  |........|
```

3.5 In

Read an I/O location.

Syntax: in <addr>

Example:

```
> in 1
08
```

3.6 Out

Write data to I/O.

Syntax: out <addr> <data1> [<data2> <data3> …]

Example:

```
> out 11 2a
*0001 bytes written
```

Note: port 0x11 (IOA_SER1_DATA) is the data port for serial port 1. Writing “*” (ASCII 0x2a) to this address caused the “*” at the start of the line after the command.

3.7 Map

Display or change the user context memory mapping.

Syntax: map [<bank> <page> [""prot""] [""super""]]

With no operands, the command displays the current state. To change the mapping, use: map <register> <page> [<prot>] [<super>]

Example:

```
> map 0 3f prot
> map 1 1
> map 2 2
> map 3 3
> map
0 (0000 - 3FFF): 3F [MONP]   read only
1 (4000 - 7FFF): 01 [CPM1]   writeable
2 (8000 - BFFF): 02 [CPM2]   writeable
3 (C000 - FFFF): 03 [CPM3]   writeable
```

Bank 0 is mapped to the monitor's programme area, and is set to write-protected. The remaining banks are allocated to the CP/M system. Note that only the monitor's pages are fixed. Use the "pages" command to find other allocations.

3.8 Pages

Display page allocations.

Syntax: pages

Example:

```
> pages
Page Allocation Status:
0 - CPM0
1 - CPM1
2 - CPM2
3 - CPM3
62 - MOND
63 - MONP
58 pages free.
```

3.9 Info

Display system information.

Syntax: info

Example:

```
> info
FPGA revision: 00.08
Total RAM: 1024 KB
CPU clock: 8004 KHz
Monitor starts: 6
Console port: ser1
Data transfer port: ser1
```

3.10 MemTest

Test all of memory apart from the two pages used for the monitor. This is a very destructive test, and takes several minutes to execute. Use with caution!

Syntax: memtest

Example:

```
> memtest
Continue (y/n): y
Page 00
Pattern 00
Pattern FF
Pattern 01
Pattern 02
```

```
Pattern 04
Pattern 08
Pattern 10
Pattern 20
Pattern 40
Pattern 80
Pattern FE
Pattern FD
Pattern FB
Pattern F7
Pattern EF
Pattern DF
Pattern BF
Pattern 7F
Addressing test
Passed
Page 01
Pattern 00

…

Paging test
Passed
```

The output for only one page is shown for brevity.

3.11 MonDownload

Runs code very similar to the stage 1 serial bootloader (the one normally entered from the front panel). This allows the monitor to be downloaded again.

Syntax: mondownload

This is a simple binary download, with all received characters being written to RAM starting at address zero. There is no form of checksum or other verification. Termination is by reset from the front panel. The only functional difference between this and the stage 1 bootloader is that this allows the use of either serial port. (see TPort command).

Example:

```
> mon.
Send binary data on port ser1. Press RESET to terminate.
```

Be careful not to send extraneous characters from the terminal before the intended data.

3.12 BinDownload

Binary download from a serial port. This is useful for downloading raw binary files to the address specified. This is similar to the monitor download, but it allows an arbitrary address as destination in user address space. It is up to the user to make sure that the destination is writeable.

Syntax: bindownload <addr>

There is no form of checksum or other verification. Termination is by reset from the front panel.

Example:

```
> bin. 100
Send binary data on port ser1. Press RESET to terminate.
```

3.13 TPADownload

Much as the bindownload command above, but downloads to the CP/M Transient Programme Area (TPA).

Syntax: tpadownload

Example:

```
> tpa.
Send binary data on port ser1. Press RESET to terminate.
```

3.14 Time

Read from or write to the real time clock.

Syntax: time [<h>:<m>[:s>] [<d>/<m>/<y>]]

Example:

```
> time 11:38:00
11:38:0 Thursday 12/5/2022 BST
> time
11:38:5 Thursday 12/5/2022 BST
```

3.15 Go

Call user code, using the current PC or a specified address.

Syntax: go [<addr>]

Example:

```
> go 100
```

3.16 Trace / t

Call user code for one instruction (single step). If an address is specified, the trace is repeated until that address is reached. As this command is so often used during debugging, the command may be abbreviated to a single “t” with no full stop.

Syntax: trace | t [<addr>]

Example:

```
> set pc 100
> t

*** NMI (trace [single step]) ***
A: 07, F: (.Z.H...C), BC: 3E00, DE: 0000, HL: 1234
A': 3E, F': (S.....N.), BC': 2E20, DE': 0000, HL': C100
PC: 0103, SP: 41C1, IX: 4165, IY: 4408, I: FF, R: 22
IEn: 0, MMap: 00 [CPM0] , 01 [CPM1] , 02 [CPM2] , 03 [CPM3]
Stack: 0103 0302 0201 0103 0302 0201 0103 0302 0201 0103
 00FF  00           nop
 0100  21 34 12     ld hl, 0x1234
*0103  76           halt
 0104  03           inc bc
 0105  01 02 03     ld bc, 0x0302
```

3.17 TraceOver / o

As trace, but if the current instruction is a CALL or RST, tracing continues until the call is complete. As this command is so often used during debugging, the command may be abbreviated to a single “o” with no full stop.

Syntax: traceover | o

Example:

```
> set pc 10d
> reg.
A: 00, F: (.Z.H.V..), BC: 015A, DE: E0F8, HL: 0175
A': 3E, F': (S.....N.), BC': 2E20, DE': 0000, HL': C100
PC: 010D, SP: 05F8, IX: 4165, IY: 4408, I: FF, R: 64
IEn: 0, MMap: 00 [CPM0] , 01 [CPM1] , 02 [CPM2] , 03 [CPM3]
Stack: 1A1A 1A1A 1A1A 1A1A 0201 0103 0302 0201 0103 0302
 0107  21 75 01     ld hl, 0x0175
 010A  CD B3 02     call 0x02B3
*010D  CD 5D 02     call 0x025D
 0110  30 09        jr nc, 0x011B
 0112  21 BF 01     ld hl, 0x01BF
> o

*** NMI (trace [single step]) ***
A: 43, F: (...H....), BC: 015A, DE: E0F8, HL: 0175
A': 3E, F': (S.....N.), BC': 2E20, DE': 0000, HL': C100
PC: 0110, SP: 05F8, IX: 4165, IY: 4408, I: FF, R: 4A
IEn: 0, MMap: 00 [CPM0] , 01 [CPM1] , 02 [CPM2] , 03 [CPM3]
Stack: 1A1A 1A1A 1A1A 1A1A 0201 0103 0302 0201 0103 0302
 010A  CD B3 02     call 0x02B3
 010D  CD 5D 02     call 0x025D
*0110  30 09        jr nc, 0x011B
 0112  21 BF 01     ld hl, 0x01BF
 0115  CD B3 02     call 0x02B3
```

In this example, the call at 0x10d is executed without without tracing each instruction with a separate “trace” command.

3.18 Set

Set register to supplied value.

Syntax: set <reg> <value>

"ien" and "mmap" aren't actually registers, but are used here to control the interrupt enable and memory mapping respectively.

Example:

```
> set bc 5a5a
> reg.
A: 43, F: (...H....), BC: 5A5A, DE: E0F8, HL: 0175
A': 3E, F': (S.....N.), BC': 2E20, DE': 0000, HL': C100
PC: 0110, SP: 05F8, IX: 4165, IY: 4408, I: FF, R: 4A
IEn: 0, MMap: 00 [CPM0] , 01 [CPM1] , 02 [CPM2] , 03 [CPM3]
Stack: 1A1A 1A1A 1A1A 1A1A 0201 0103 0302 0201 0103 0302
 010A  CD B3 02     call 0x02B3
 010D  CD 5D 02     call 0x025D
*0110  30 09        jr nc, 0x011B
 0112  21 BF 01     ld hl, 0x01BF
 0115  CD B3 02     call 0x02B3
```

3.19 Registers

Display register state. The instructions surrounding the PC are disassembled, which is very useful when tracing code.

Syntax: registers

Example:

```
> reg.
A: 43, F: (...H....), BC: 5A5A, DE: E0F8, HL: 0175
A': 3E, F': (S.....N.), BC': 2E20, DE': 0000, HL': C100
PC: 0110, SP: 05F8, IX: 4165, IY: 4408, I: FF, R: 4A
IEn: 0, MMap: 00 [CPM0] , 01 [CPM1] , 02 [CPM2] , 03 [CPM3]
Stack: 1A1A 1A1A 1A1A 1A1A 0201 0103 0302 0201 0103 0302
 010A  CD B3 02     call 0x02B3
 010D  CD 5D 02     call 0x025D
*0110  30 09        jr nc, 0x011B
 0112  21 BF 01     ld hl, 0x01BF
 0115  CD B3 02     call 0x02B3
```

3.20 ReadSec

Read sectors from memory card.

Syntax: readsec <start sec> <no. of 512-byte sectors> <address>

<start sec> and <no. of 512-byte sectors> are decimal values.

Example:

```
> readsec 0 1 1000
Reading sector: 0
> dump 1180 80
1180  00 00 00 00 00 00 00 00  |........|
                ...
11B8  06 97 85 77 00 00 00 04  |...w....|
11C0  01 04 7F 47 02 48 00 08  |...G.H..|
11C8  00 00 00 88 00 00 00 00  |........|
```

```
11D0  00 00 00 00 00 00 00 00  |........|
                 ...
11F8  00 00 00 00 00 00 55 AA  |......U.|
```

In this example, the boot section is read. The dump shows the boot signature and partition entry no. 1.

3.21 WriteSec

Write sectors to memory card.

Syntax: writesec <start sec> <no. of 512-byte sectors> <address>

<start sec> and <no. of 512-byte sectors> are decimal values.

Example:

```
> writesec 1 1 1000
Writing sector: 1
```

3.22 Cache

Display memory card cache state.

Syntax: cache

Example:

```
MC Cache Status:
0 - 0 clean
1 - 1 dirty
2 - 4126 clean
3 - 4127 clean
4 - 4096 clean
5 - 4248 clean
6 - 4249 clean
7 - 4250 clean
8 - 4116 clean
9 - 4117 clean
10 - 4118 clean
11 - 4119 clean
12 - 4120 clean
13 - 4121 clean
14 - 4122 clean
15 - 4123 clean
In use: 16, dirty: 1, next read buffer: 2
```

3.23 CPM

Load and start CP/M from the memory card.

Syntax: cpm [“debug"] ["noints"]

debug - enables BIOS debugging messages.

noints - disables interrupts (use basic I/O mode).

Example:

```
> cpm
ZARC CP/M V2.2 BIOS V1.1

A>
```

3.24 CPort

Set console port. This defaults to ser1.

Syntax: cport <port>

<port> ::= "ser1" | "ser2"

Example:

```
> cport ser1
> info
FPGA revision: 00.08
Total RAM: 1024 KB
CPU clock: 8004 KHz
Monitor starts: 21
Console port: ser1
Data transfer port: ser1
```

3.25 TPort

Set data transfer port. This defaults to ser1.

Syntax: tport <port>

<port> ::= "ser1" | "ser2"

Example:

```
> tport ser2
> info
FPGA revision: 00.08
Total RAM: 1024 KB
CPU clock: 8004 KHz
Monitor starts: 21
Console port: ser1
Data transfer port: ser2
```

3.26 Cold

Perform a cold start. This reinitialises certain variables, such as the memory page allocations.

Syntax: cold

Example:

```
> cold

*** Cold start ***
```

```
____  __  ___   ___
   / /  \ |  \ /
  /  |__| |__/ |
 /   |  | |  \ |
/___ |  | |  | \___

ZARC Monitor V1.0
Initialising memory card
Product name: 000000
Serial number: B2000004
Manufactured: 12/2006
Total capacity: 31 MB
Block size: 512 bytes
> info
FPGA revision: 00.08
Total RAM: 1024 KB
CPU clock: 8004 KHz
Monitor starts: 1
Console port: ser1
Data transfer port: ser1
> pages
Page Allocation Status:
62 - MOND
63 - MONP
62 pages free.
```

4 OPERATING ENVIRONMENT

This section describes some of the internal operation of the monitor and the environment it provides.

4.1 Memory Map

The following page map is used by the monitor when it is active.

Bank Number	***CPU Address Range***	***Mode***	***Function***
0	0x0000 to 0x3fff	Read only, supervisor	Monitor programme
1	0x4000 to 0x7fff	Writeable	Monitor variables, stacks, buffers etc.
2	0x8000 to 0xbfff	As required	Mapped as required
3	0xc000 to 0xffff	As required	Mapped as required

Other pages are mapped into banks 2 and 3 as required. Page 0x3f is selected in hardware when memory mapping is not active, so on reset the monitor (page 0x3f at address 0) is entered. One of the monitor's first tasks is to map banks 0 and 1 to establish its normal operating state.

4.2 Externally Callable Functions

Many monitor function are made available to other applications. These have fixed entry points using a jump table in the same manner that CP/M uses. This allows changes to the monitor without having to recompile other modules. The jump vector table begins at location 0x0069 (immediately after the NMI vector). The table is defined in monitor.i. See CP/M BIOS for an example of use.

Use these routines with care. Consideration of memory paging, interrupts and stack location is required. When passing data by reference to its address, make sure the data is paged in and accessible to the routine.

A summary of the available functions follows:

Function	*Source File*	*Function*
MON_ENTRY	monitor.z80	Normal monitor entry
PANIC	monitor.z80	System error monitor entry
CONWCH	io.z80	Write character to the console port
CONRCH	io.z80	Read character from the console port
CONSTA	io.z80	Fetch console port status
S1WCH	io.z80	Write character to serial port 1
S1RCH	io.z80	Read character from serial port 1
S1STA	io.z80	Fetch serial port 1 status
S2WCH	io.z80	Write character to serial port 2
S2RCH	io.z80	Read character from serial port 2
S2STA	io.z80	Fetch serial port 2 status
CWNWLN	utility.z80	Write new line characters to console
CONWMS	utility.z80	Print null-terminated string to the console
CONWMN	utility.z80	Print message <n> in a list
SKPSPC	utility.z80	Step HL past spaces
TOUPCA	utility.z80	Convert character in A to upper case
TOLOCA	utility.z80	Convert character in A to lower case
CWVICH	utility.z80	Print only visible 7-bit characters
CWPSPC	utility.z80	Print a space
MTWDLI	utility.z80	Match word in a string against list of options
CWPHNB	utility.z80	Print least-significant nibble in A in hexadecimal
CWPHBY	utility.z80	Print A in hexadecimal
CWPHWD	utility.z80	Print HL in hexadecimal
CWPDBY	utility.z80	Print A in decimal. Leading zeros are suppressed.
CWPDWD	utility.z80	Print HL in decimal. Leading zeros are suppressed.
CWPDLO	utility.z80	Print DEHL in decimal. Leading zeros are suppressed
RDHXWD	utility.z80	Read a 16-bit hexadecimal number from a string
RDDUWD	utility.z80	Read a 16-bit unsigned decimal number from a string
RDDULO	utility.z80	Read a 32-bit unsigned decimal number from a string
MADLBU	maths.z80	32-bit / 8-bit unsigned divide

Function	*Source File*	*Function*
MADWBU	maths.z80	16-bit / 8-bit unsigned divide
MADBBU	maths.z80	8-bit / 8-bit unsigned divide
MAMWWU	maths.z80	16-bit * 16-bit unsigned multiply
MAMLBU	maths.z80	32-bit * 8-bit unsigned multiply
CRC16X	maths.z80	CRC-16 (XMODEM) – note that this is now replaced by hardware.
TIRD	time.z80	Read current time and date
TIWR	time.z80	Set time and date
TIITOS	time.z80	Convert integer time to structure
TISTOI	time.z80	Convert structure time to integer
CWPTM	time.z80	Print time structure to console
MCPRER	mmc.z80	Print memory card error text
CACRS	cache.z80	CP/M read sector
CACWS	cache.z80	CP/M write sector
CAFLUS	cache.z80	Flush cache (write dirty buffers)
MPALL	memory.z80	Allocate new page
MPFREE	memory.z80	Free allocated page
MPSTAT	memory.z80	Allocation status

See the relevant source files for full information. The terse function names are because the linker (ld80) only considers the first six characters of labels significant.

5 INITIAL BOOT

In normal use, the monitor is maintained by the battery backup mechanism and is started automatically when power is applied or on reset. If the monitor is not present or has become corrupt, it must be re-loaded. Hardware write protection and multiple sources of backup power make this a rare event.

The following short programme or equivalent must be entered from the front panel and executed. See ZARC HARDWARE USERS' GUIDE for front panel operating instructions.

```
3FE0  3E 7F      LD   a,0x7f          ; Page 0x3f, SUPER=1 and PROT=0
3FE2  D3 04      OUT   0x04,a         ; Page and prot. 0x0000-0x3fff
3FE4  3E 02      LD   a,0x02
3FE6  D3 02      OUT   0x02,a         ; Enable memory mapping
3FE8  0E 11      LD   c,0x11          ; Select console input port
3FEA  21 00 00   LD   hl,0x0000       ; Start address
```

```
3FED  LOOP:
3FED  ED A2      INI
3FEF  18 FC      JR   loop            ; Load data until reset
```

Initial Programme (Bootstrap)

This loads raw binary data from the serial port into memory starting at location zero. Press reset to stop the loader and run the newly downloaded code. Note that this transfer is pure binary data with no checksum. It is essential that the serial link is reliable to avoid data loss or corruption.

The monitor does not overwrite this boot loader, so if the monitor requires re-loading it should not be necessary to re-enter it.

6 EXAMPLE SESSION

This section demonstrates the entry and execution of a short programme. Comments are in blue.

```
Setup memory mapping.
> map 0 0                    Map page 0 to bank 0
> set mmap 1                 Ensure memory mapping is enabled
> map                        Display mapping
0 (0000 - 3FFF): 00 [unallocated]   writeable
1 (4000 - 7FFF): 01 [unallocated]   writeable
2 (8000 - BFFF): 02 [unallocated]   writeable
3 (C000 - FFFF): 03 [unallocated]   writeable
Write programme to memory.
> write 0 0b 78 b1 20 fb 76
0006 bytes written
Disassemble it.
> dis. 0
0000  0B          dec bc
0001  78          ld a, b
0002  B1          or c
0003  20 FB       jr nz, 0x0000
0005  76          halt
0006  06 DE       ld b, 0xDE
0008  03          inc bc
0009  01 02 03    ld bc, 0x0302
000C  01 02 03    ld bc, 0x0302
000F  01 02 03    ld bc, 0x0302
Set initial registers.
> set bc 0
> set pc 0
> set ien 0                  Disable interrupts
> reg.                       Display registers
A: 00, F: (........), BC: 0000, DE: 0000, HL: 0000
A': 00, F': (........), BC': 0000, DE': 0000, HL': 0000
PC: 0000, SP: 0000, IX: 0000, IY: 0000, I: 00, R: 00
IEn:  0,  MMap:  00  [unallocated]  ,  01  [unallocated]  ,  02
[unallocated] , 03 [unallocated]
Stack: 780B 20B1 76FB DE06 0103 0302 0201 0103 0302 0201
*0000  0B          dec bc
 0001  78          ld a, b
 0002  B1          or c
Trace a few instructions, one at a time.
> t
```

```
*** NMI (trace [single step]) ***
A: 00, F: (........), BC: FFFF, DE: 0000, HL: 0000
A': 00, F': (........), BC': 0000, DE': 0000, HL': 0000
PC: 0001, SP: 0000, IX: 0000, IY: 0000, I: 00, R: 4E
IEn:  0,  MMap:  00  [unallocated]  ,  01  [unallocated]  ,  02
[unallocated] , 03 [unallocated]
Stack: 780B 20B1 76FB DE06 0103 0302 0201 0103 0302 0201
*0001  78          ld a, b
 0002  B1          or c
 0003  20 FB       jr nz, 0x0000

ZARC Monitor V1.0
> t

*** NMI (trace [single step]) ***
A: FF, F: (........), BC: FFFF, DE: 0000, HL: 0000
A': 00, F': (........), BC': 0000, DE': 0000, HL': 0000
PC: 0002, SP: 0000, IX: 0000, IY: 0000, I: 00, R: 1C
IEn:  0,  MMap:  00  [unallocated]  ,  01  [unallocated]  ,  02
[unallocated] , 03 [unallocated]
Stack: 780B 20B1 76FB DE06 0103 0302 0201 0103 0302 0201
*0002  B1          or c
 0003  20 FB       jr nz, 0x0000
 0005  76          halt

ZARC Monitor V1.0
> t

*** NMI (trace [single step]) ***
A: FF, F: (S....V..), BC: FFFF, DE: 0000, HL: 0000
A': 00, F': (........), BC': 0000, DE': 0000, HL': 0000
PC: 0003, SP: 0000, IX: 0000, IY: 0000, I: 00, R: 6A
IEn:  0,  MMap:  00  [unallocated]  ,  01  [unallocated]  ,  02
[unallocated] , 03 [unallocated]
Stack: 780B 20B1 76FB DE06 0103 0302 0201 0103 0302 0201
*0003  20 FB       jr nz, 0x0000
 0005  76          halt
 0006  06 DE       ld b, 0xDE

ZARC Monitor V1.0
Trace to address 5.
> t 5
This will take a very long time to complete as execution is slow
when tracing. Pressing the NMI on the front panel results in a
return to the monitor.

*** NMI (front panel) (trace [single step]) ***
A: 6D, F: (........), BC: 6824, DE: 0000, HL: 0000
A': 00, F': (........), BC': 0000, DE': 0000, HL': 0000
PC: 0001, SP: 0000, IX: 0000, IY: 0000, I: 00, R: 36
IEn:  0,  MMap:  00  [unallocated]  ,  01  [unallocated]  ,  02
[unallocated] , 03 [unallocated]
Stack: 780B 20B1 76FB DE06 0103 0302 0201 0103 0302 0201
*0001  78          ld a, b
 0002  B1          or c
 0003  20 FB       jr nz, 0x0000

ZARC Monitor V1.0
> set bc 100                Set a shorter delay
> t 5                       Trace again
After a short delay, the machine halts (halt LED comes on). I
pressed NMI to return to the monitor.
```

```
*** NMI (trace [single step]) ***
A: 00, F: (.Z...V..), BC: 0000, DE: 0000, HL: 0000
A': 00, F': (........), BC': 0000, DE': 0000, HL': 0000
PC: 0005, SP: 0000, IX: 0000, IY: 0000, I: 00, R: 20
IEn:  0,  MMap:  00  [unallocated]  ,  01  [unallocated]  ,  02
[unallocated] , 03 [unallocated]
Stack: 780B 20B1 76FB DE06 0103 0302 0201 0103 0302 0201
*0005  76          halt
 0006  06 DE       ld b, 0xDE
 0008  03          inc bc
```

Executing the programme directly (without tracing).

```
ZARC Monitor V1.0
> set pc 0
> go
```

After a short delay, the machine halts (halt LED comes on). I pressed RESET to return to the monitor.

```
____  __  ___   ___
   / /  \ |  \ /
  /  |__| |__/ |
 /   |  | |  \ |
/___ |  | |  | \___

ZARC Monitor V1.0
Initialising memory card
Product name: 000000
Serial number: B2000004
Manufactured: 12/2006
Total capacity: 31 MB
Block size: 512 bytes
>
```

7 CP/M DEBUGGING

This section describes use of the monitor's debugger for use with CP/M. Much of this is also applicable to other operating systems and applications.

Start CP/M.

```
> cpm

ZARC CP/M V2.2 BIOS V1.5
A>
```

Start the application.

```
A> d:bitcoin
```

Press "NMI" on the front panel to enter the monitor.

```
*** NMI (front panel) ***
<register dump>
```

We can examine the code, make modifications and examine variables as required. Note that for programmes consisting of relocatable modules, it will be necessary to add the base address of the module from the .sym file to the address in the listing.

```
> dis. 100
```

```
0100  ED 73 4F 0F  ld (0x0f4f), sp
0104  31 EC 0F     ld sp, 0x0FEC
0107  CD 18 0B     call 0x0B18
...
```

The trace (t) command is very useful when debugging. However, as this traces all instructions including interrupts, it is usually preferable to disable these to avoid unexpected jumps to interrupt service routines.

```
> set ien 0
```

Note that the BIOS will re-enable interrupts when they are required due to serial port access, so it may be necessary to disable them again.

```
> set pc 100
```

Now we can trace through the code.

```
> t
```

We might want to trace to a certain address, for example to just after the sign on message is printed. From the listing file:

```
0018' 219501  on_z80          ld hl, signon_msg
001B' CD0000                  call conwms        ; Print string
              ;
              ; Are we running on a ZARC system?
001E' CD0000                  call zrcdet
```

This is part of bitcoin.z80, which is the first module loaded. From the .sym file we have:

```
Addr  Length Typ Name       Module   File
=========================================
0100   0390   P  -          BITCOI   bitcoin.rel
```

Hence the required address is 0x0100 + 0x001e = 0x011e.

```
> t 11e
*** Bitcoin Mining V1.0 ***

*** NMI (trace [single step]) ***
<register dump>
```

Control is returned to the monitor for further debugging commands. When the session is complete, the following command returns control to CP/M:

```
> go 0
```

This is a warm start, so any I/O redirection is preserved.