LAUTERBACH
DEVELOPMENT TOOLS

Z80 Monitor

Release 09.2023

MANUAL

Z80 Monitor

Z80 Monitor

Version 10-Oct-2023

```
P:000D75  \\IARZ80\iarz80\sieve+97                    ..........  MIX  AI

E::w.d.l
 addr/line  code              label      mnemonic                 comment
   P:000D70 AF                           xor    a
   P:000D71 ED42                         sbc    hl,bc
   P:000D73 381E                         jr     c,0D93             ; c,?0176
                                      {
        491                                     flags[ k ] = FALSE;
   P:000D75 0E00              ?0178:     ld     c,0                ; c,0
   P:000D77 2111C2                       ld     hl,0C211           ; hl,flags
   P:000D7A DD5EFC                       ld     e,(ix-4)           ; e,(ix-4)

E::w.v.f /l /c
     j = 15
             {
                       sieve();
-000 sieve()
     i = 0
     primz = 3
     k = 15
     anzahl = 0

E::w.r
CY   _  A      0   BC   800F   SP >0000
N    N  F      2   DE 0C40C   -06 0003
P/V  _  B     80   HL      3   -04 000F
Hc   _  C     0F   IX 0C40F   -02 0000
Zr   _  D   0C4    IY   2D8   FP >C41F
Sig  _  E     0C   SP 0C407   +02 0CDC
IFF  _  I     0    PC  0D75   +04 000F
Tsk                AF'    0   +06 0000
```

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Training Basic Debugging"** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"T32Start"** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **"General Commands"** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **"OS Awareness Manuals"** (rtos_<*os*>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Warning

NOTE:	Do not connect or remove probe from target while target power is ON. Power up: Switch on emulator first, then target Power down: Switch off target first, then emulator

Quick Start of the Z80 ROM Monitor

Starting up the ROM Monitor is done as follows:

1. Select the device **B:** for the ROM Monitor.

```
b:
```

2. Power the system down (optional).

```
sys.d
```

This instruction is necessary when the system is restarted. When the system is active while you try to reinitialize it, you get an error message.

3. Set the CPU type in the ROM Monitor program to load the CPU specific

```
sys.cpu z80
```

4. Map the EPROM simulator. The mapping of the EPROM simulator is described in the section "**Mapping the EPROM simulator**".

5. Load the monitor program. Usually the monitor program runs at address 0 in the ROM area.

```
d.load.b romz80.bin /ny
```

6. Configure the Monitor program. The processor type is required for startup. Other parameters are optional.

```
d.s 0x400 0x0                   ; for Z80 processor
```

7. Load the program.

```
d.load.u iarz80.dbg
```

The format of the Data.LOAD command depends on the file format generated by the compiler. The corresponding options for all available compilers are listed in the compiler list. A detailed description of the Data.LOAD command is given in the Emulator Reference Manual.

8. Set the polarity of the Reset and NMI signal according to your target. The NMI signal is optional, it can be used to interrupt the program.

```
x.respol -
x.nmipol -
x.nmibreak on
```

9. Start the ROM Monitor. If the RESET output of the ESI is not connected you must press the RESET button on your target after entering this command.

```
SYStem.Up
```

A typical start sequence is shown below:

```
; the EPROM is in the addressrange 0x0--0x7fff
; the RAM is in the addressrange 0x8000--0x0ffff

b:                              ; select the Debugger device
sys.d                           ; switch the system dow
winclear                        ; clear all windows
map.res                         ; map the EPROM simulator
map.rom 0x0--0x7fff             ; part of EPROM after internal registers
d.load.u iarz80.dbg             ; load the application
d.load.b romz80.bin /ny         ; overload is with the monitor
x.respol -
x.nmipol -                      ; adapt the polarity of RES and NMI
x.nmibreak on                   ; enables the connection of the NMI signal
SYStem.Up                       ; power the system up
```

The start up can be automated using the programming language PRACTICE.

Troubleshooting

No information available.

FAQ

EPROM Simulator Error on Data Modification Ref: 0056	**Why does the ROM monitor crash after modification of EPROM?** Check that there is enough space left on the stack. See also "Restrictions for Stack Requirements".
Step or Breakpoint Fails Ref: 0061	**Why does single step or breakpoint not work?** Check that there is enough space left on the stack before and after the execution of the instruction. See "Restrictions for Stack Requirements". Make sure that the single step and INT3 vector (1 + 3) are valid and point to the correct monitor entry.
Stepping Fails when Executing MOV SP,xxx Ref: 0062	**Why does stepping fail, when executing a MOV SP,xxx instruction?** Check that there is enough space left on the stack before and after the execution of the instruction. See "Restrictions for Stack Requirements". Check that the value for the CP is within limits for the CPU and that the register space is not being overwritten by the stack. See "Restrictions for Stack Requirements".

Basics

Monitor Features

The monitor requires no stack during startup and memory operations. A valid stack is only required for modifications in the EPROM while the monitor is running (Hot Patch) and for single step and go commands. This allows to use the monitor even when the stack is not valid. External RAM memory is not required during startup and for memory operations. This allows to use the monitor also on not fully functional hardware. The NMI pin of the EPROM Simulator can be used to manually stop the target program. The monitor fully supports bank switching (breakpoints, memory access, symbols). The monitor can operate also when the EPROM is not always visible, i.e. is one of the banks.

Monitor Files

The 'romz80' monitor is for Eprom Simulator solutions, while the 'romz80e' monitor is used as foreground monitor for Emulators. By using a foreground monitor the target program can be single stepped without stopping the target processors interrupts or DMA transfers. Both monitors have the same source file 'romz80.asm'. This source file should not be modified, it is only included for reference purposes. There are to possibilities to include the monitor in the application: loading the '.bin' by the Eprom Simulator or linking the '.src' file together with the application. The '.src' files contain only the monitor code, a corresponding configuration table must be included in the target program.

Address Layout

The Rom Monitor is freely relocatable in the whole address space by reassembling the source. The communication area for the Eprom Simulator is located at the fixed address 1000 to 1FFF.

The monitor program consists of three parts:

- Vector Table
- Configuration Table
- Monitor Program Code

The '.bin' and '.asm' files contain all three parts of the monitor. The address layout of the default monitor is as follows:

```
0x0000--0x03FF                  ; Vector Table

0x0400--0x041F                  ; Configuration Table

0x0420--0x0FFF                  ; Monitor Code
```

For the first tests of a software, the '.bin' files can be loaded with vector and configuration table. When the vector table becomes part of the application, it is not loaded with the monitor. Instead the table is setup according to the application (the table may also reside in RAM). Some vectors must be set up to point into the monitor program code. The entry points are located at the beginning of the monitor.

vec	offs	ent	usage
00	000	+20	Reset (optional, can also go to application)
38	038	+30	Breakpoint Trap, (used for breakpoints, can be changed)
66	XXX	+30	Manual Break (optional)
..	...	+40	Any unused vector may be handled by the monitor

The breakpoint trap vector can be configured by the **SYStem.Option.BrkVector** command. The default is vector 38.

Configuration

The configuration table of the monitor must always be located directly before the monitor code. The default location used in the binary files is 400 (hex).

- Processor core type (byte at offset 00H):

  00 = Z80 (default)

  01 = Z180 (NOTE: Requires also SYStem.CPU Z180 and MMU.ON commands)

- Z180 MMU address (byte at offset 01H):

  38 or 78 or 0b8 or 0f8

- Monitor Interrupt Level (byte at offset 04H)

  00 = all interrupts enabled in monitor

  01 = all interrupts locked in monitor

- Set target bank (four jumps at offset 0cH,10H,14H,18H)
- Get target bank (jump at offset 1cH)

SYStem.CPU — CPU type

Format:	**SYStem.CPU** <*mode*>
<*mode*>:	**Z80** | **Z180** | **Z181** | **Z182**

Selects the processor type. The ROM debugger requires also a modification in the debug monitor for different processor types.

SYStem.Mode — Establish the communication with the CPU

Format:	**SYStem.Mode** <*mode*>
<*mode*>:	**Down** **NoDebug** **Go** **Up**

Default: Down. Selects the target operating mode.

Down	The CPU is in reset. Debug mode is not active. Default state and state after fatal errors.
NoDebug	The CPU is running. Debug mode is not active. Debug port is tristate. In this mode the target should behave as if the debugger is not connected.
Go	The CPU is running. Debug mode is active. After this command the CPU can be stopped with the break command or if any break condition occurs.
Up	The CPU is not in reset but halted. Debug mode is active. In this mode the CPU can be started and stopped. This is the most typical way to activate debugging.

If the mode “Go” is selected, this mode will be entered, but the control button in the SYStem window jumps to the mode “UP”.

General Restrictions

Stack Memory	The ROM debugger needs 32 bytes of memory on the current stack. The stack is not required for starting the Monitor and memory read or modify commands. Modification of the EPROM while the monitor is running (Hot Patch) requires 40 bytes on the stack.
CBAR/CBR/BBR Access (Z180)	These registers should NOT be accessed by Data.SET commands.
CBAR value (Z180)	The address between 0000 and 1FFF must be COMMON0 area.

SYStem.Option.BrkVector Breakpoint trap

Format: **SYStem.Option.BrkVector** <*trap*>

Defines the rst number used for breakpoints and single stepping. The default is RST 38.

SYStem.Option.BASE Base address of internal registers

Format: **SYStem.Option.BASE** <*address*>

The **SYStem.Option.BASE** defines the base address of the internal registers.

Using the MMU for Z180

This command and the commands **MMU** support the built-in MMU of the Z180 processors.

The analyzer and all memory systems and breakpoints are based on the physical address. The display in the analyzer can be both physical or logical addresses. A logical address can have two formats: smaller than 64K or larger. Smaller addresses are assumed to be an logical address as seen by the CPU in the current MMU configuration. If an address is larger than 64 K, the address bits A16 to A23 define the bank base address used for the BBR or CBR register. Logical above 64 K addresses should only be used, if the MMU registers were already setup. The following schematic shows these relations for some examples:

```
preset: CBAR=84, BBR=10, CBR=20

logical address:    5    0     4    5    6    7    (Hex)
                    |    |     |    16 bit       |
               CBR/BBR = 50  logical CPU address

                  -->  physical address: 54567


logical address:    0    0     1    5    6    7    (Hex)
                    |    |     |    16 bit       |
                 current-mmu logical CPU address

                  -->  physical address:  1567


logical address:    0    0     4    5    6    7    (Hex)
                    |    |     |    16 bit       |
                 current-mmu logical CPU address

                  -->  MMU Bank Area
                  -->  physical address: 04567
                               +BBR    10---
                                      =14567


logical address:    0    0     c    d    e    f    (Hex)
                    |    |     |    16 bit       |
                 current-mmu logical CPU address

                  -->  COMMON1 Area
                  -->  physical address: 0cdef
                               +CBR   +20---
                                      =2cdef
```

To activate the correct address translation for breakpoints, the **MMU** command must be activated. The following example loads a banked application:

```
mmu.off
map.rom 0x0--0x7ffff
...
mmu.on
d.load.u iarz180.dbg
```

The next example loads a banked application in two logical units:

```
CBAR=84, CBR=0, BBR=10 or 20
mmu.reset
symbol.reset
map.rom 0x--0x7ffff
…
mmu.create 0x104000--0x107fff
mmu.create 0x204000--0x207fff
mmu.on
d.load.b bank1.cod 0x104000 /nosymbol
d.load.b bank2.cod 0x204000 /nosymbol
d.load.b common.cod 0x2000 /nosymbol
d.load.sym bank1.sym /noclear
symbol.reloc p:100000 0x4000--0x4fff
d.load.sym bank2.sym /noclear
symbol.reloc p:200000 0x4000--0x4fff
d.load.sym common.sym /noclear
```

Memory Classes

Memory Class	Description
D	Data
P	Program
C	Memory access by CPU
E	Emulation memory access
A	Absolute (physical) memory access