University of British Columbia
Electrical and Computer Engineering
Digital Design and Microcomputers
ELEC291/CPEN312

Debugging Assembly Programs with CMON51 in the CV-8052 Soft-Processor

Introduction

This document describes how to use the 8051 monitor CMON51 in the CV-8052 soft processor to debug assembly code. CMON51 uses the USB port of the Altera DE0-CV to communicate with the program Quartus_stp.exe provided by Altera with Quartus II.

Starting CMON51 via USB blaster

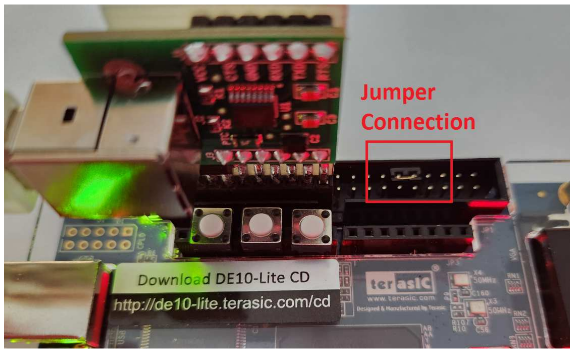
To run CMON51 in the CV-8052 soft-processor, press and hold button 'KEY3' while pressing and releasing button 'FPGA_RESET'. The message 'dEbUGG' is displayed in the 7-segment displays. To access CMON51 from the host computer a TCL script compatible with Quaturs_stp.exe is provided: cmon51.tcl. To start the debug session type in a command prompt:

```
"C:\intelFPGA_lite\23.1std\quartus\bin64\quartus_stp.exe" -t "cmon51.tcl"
```

Please note that the complete path of 'quatus_stp.exe' must be provided and match the Quartus II installation in the current computer. For your convenience, a Windows batch file, 'cmon51.bat' is also provided, but it must be edited so the path of 'quartus_stp.exe' is valid. A debug session can be quickly started by double-clicking 'cmon51.bat' from Windows Explorer, as described in the paragraphs below.

Starting a CMON51 Debug Session using the Serial Port

If a serial port is attached to the CV-8052, it is possible to use it as the console for the debugging session. Configure your serial terminal for 115200 baud, 8 bits, no parity. For the debug session to start using the port, pin P3.3 must be zero. This can be achieved with a jumper between pins 30 and 32 of connector GPIO in the DE10-Lite board as shown in the picture below. For other boards (DE0-CV and DE1-SoC) check the corresponding documentation for the location of pin P3.3.

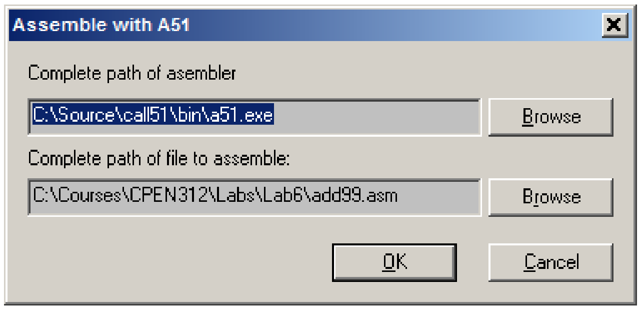


Debugging Session Example

To demonstrate the use of CMON51, let us start with the small 8051 assembly program shown below. The program is supposed to add 99H to the accumulator and save the result into register B. The target processor is the CV-8052 soft processor. We will be using CrossIDE to edit, compile, and flash our code into the processor.

```
$MODDE0CV

org 0000H
    ljmp myprogram

myprogram:
    mov SP, #7FH

    mov a, #10H
    add a, 99H
    mov b, a

forever:
    sjmp forever
END
```

The code above is assembled normally using A51 as shown in the figure below:


Assemble with A51

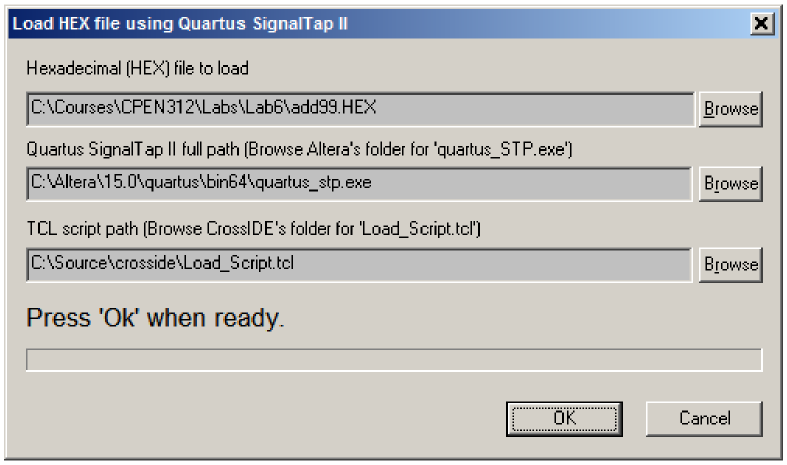
Complete path of asembler

C:\Source\call51\bin\a51.exe

Complete path of file to assemble:

C:\Courses\CPEN312\Labs\Lab6\add99.asm

After assembling the code, the corresponding "add99.HEX" file is generated. Now we 'flash' that file to the CV-8052 soft-processor:



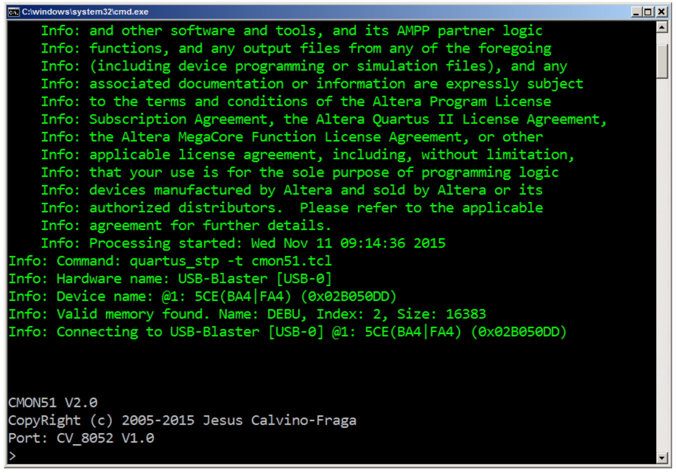
Hexadecimal (HEX) file to load
C:\Courses\CPEN312\Labs\Lab6\add99.HEX

Quartus SignalTap II full path (Browse Altera's folder for 'quartus_STP.exe')
C:\Altera\15.0\quartus\bin64\quartus_stp.exe

TCL script path (Browse CrossIDE's folder for 'Load_Script.tcl')
C:\Source\crosside\Load_Script.tcl

Press 'Ok' when ready.

To start the debug session, double click “cmon51.bat”. A TCL console window pops open. Pressing and holding button KEY3 while pressing and releasing the FPAG_RESET button in the DE0-CV board starts CMON51, which prints the information shown below.



```
    Info: and other software and tools, and its AMPP partner logic
    Info: functions, and any output files from any of the foregoing
    Info: (including device programming or simulation files), and any
    Info: associated documentation or information are expressly subject
    Info: to the terms and conditions of the Altera Program License
    Info: Subscription Agreement, the Altera Quartus II License Agreement,
    Info: the Altera MegaCore Function License Agreement, or other
    Info: applicable license agreement, including, without limitation,
    Info: that your use is for the sole purpose of programming logic
    Info: devices manufactured by Altera and sold by Altera or its
    Info: authorized distributors.  Please refer to the applicable
    Info: agreement for further details.
    Info: Processing started: Wed Nov 11 09:14:36 2015
Info: Command: quartus_stp -t cmon51.tcl
Info: Hardware name: USB-Blaster [USB-0]
Info: Device name: @1: 5CE(BA4|FA4) (0x02B050DD)
Info: Valid memory found. Name: DEBU, Index: 2, Size: 16383
Info: Connecting to USB-Blaster [USB-0] @1: 5CE(BA4|FA4) (0x02B050DD)



CMON51 V2.0
CopyRight (c) 2005-2015 Jesus Calvino-Fraga
Port: CV_8052 V1.0
>
```

At this point we can start debugging our code. For example, let us run the “Registers”, “Dis-assemble”, and “Single Step” commands and see what happens when each line of the program is executed:

```
> r
A =00   B =00   SP=7F   IE=00   DPH=00   DPL=00   PSW=00   PC=0003
R0=00   R1=00   R2=00   R3=00   R4=00   R5=00   R6=00   R7=00   BANK 0
> u 3
0003: 75 81 7F   mov     SP,#7F
> s
A =00   B =00   SP=7F   IE=00   DPH=00   DPL=00   PSW=00   PC=0006
R0=00   R1=00   R2=00   R3=00   R4=00   R5=00   R6=00   R7=00   BANK 0
0006: 74 10      mov     a,#10
> s
A =10   B =00   SP=7F   IE=00   DPH=00   DPL=00   PSW=01   PC=0008
R0=00   R1=00   R2=00   R3=00   R4=00   R5=00   R6=00   R7=00   BANK 0
0008: 25 99      add     a,SBUF
> s
```

```
A =10  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=01  PC=000A
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000A: F5 F0     mov     B,a
> s
A =10  B =10  SP=7F  IE=00  DPH=00  DPL=00  PSW=01  PC=000C
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000C: 80 FE     sjmp    000C
> s
A =10  B =10  SP=7F  IE=00  DPH=00  DPL=00  PSW=01  PC=000C
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000C: 80 FE     sjmp    000C
>
```

Notice from the step by step execution above that the "**add a, SBUF**" instruction "was not" in our code. Actually I made a mistake (intentionally!) when writing the code by missing the 'direct' operator '#" before the add instruction[1]. Therefore the assembler assumed that we were using SFR SBUF (whose address is 99H) instead of number 99H. The corrected source code is shown below:

```
$MODDE0CV

org 0000H
    ljmp myprogram

myprogram:
    mov SP, #7FH

    mov a, #10H
    add a, #99H
    mov b, a

forever:
    sjmp forever
END
```

Before loading the code into the CV-8052 soft-processor, we need to tell Quartus_stp.exe to release the USB port so CrossIDE can use to program the flash memory with the new program:

```
> hang
Info: Disconnecting from USB-Blaster [USB-0] @1: 5CE(BA4|FA4) (0x02B050DD)
Press Enter to reconnect...
```

When the fixed code is run step by step using the debugger (following the steps described above) the program behaves as expected:

```
> r
A =00  B =00  SP=07  IE=00  DPH=00  DPL=00  PSW=00  PC=0003
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
> u 3
0003: 75 81 7F  mov     SP,#7F
> s
A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=0006
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
0006: 74 10     mov     a,#10
```

[1] Recent versions of the assembler will warn about this kind of mistakes.

```
> s
A =10  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=01  PC=0008
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
0008: 24 99     add     a,#99
> s
A =A9  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=000A
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000A: F5 F0     mov     B,a
> s
A =A9  B =A9  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=000C
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000C: 80 FE     sjmp    000C
> s
A =A9  B =A9  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=000C
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000C: 80 FE     sjmp    000C
>
```

As expected, the accumulator ends up with the right result 10H+99H=0A9H.

CMON51 Command Summary

Single Step

Syntax:

s [n]

Description:

Execute [n] assembly instructions starting from the current program counter. After executing the current assembly instruction the registers and the next instruction are displayed.

Examples:

```
> s

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01A6
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01A6: E4        clr     a
> s

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01A7
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01A7: F5 E8     mov     LEDRA,a
>
```

Step until Address

Syntax:

t [address]

Description:

This command is similar to the step ('S') command, but single steps will be executed until the program counter (PC) reaches [address]. Every instruction will be displayed in the terminal as it is executed. When the PC reaches [address] the registers and the next instruction to be executed are displayed in the screen.

Example:

```
> t 33bc

33AE: 74 F0     mov     a,#F0
33B0: 55 89     anl     a,TMOD
33B2: 44 01     orl     a,#01
33B4: F5 89     mov     TMOD,a
33B6: 75 8A EA  mov     TL0,#EA
33B9: 75 8C 03  mov     TH0,#03

A =21  B =00  SP=5C  IE=80  DPH=00  DPL=00  PSW=00  PC=33BC
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
33BC: E4        clr     a
>
```

Step Until Address with Registers

Syntax:

tr [address]

Description:

This command is similar to the step until address (‘T’) command, but every instruction will be displayed in the terminal as it is executed along with the registers. When the PC reaches [address] it stops.

Example:

```
> tr 1b5
01A6: E4        clr     a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01A7
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01A7: F5 E8     mov     LEDRA,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01A9
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01A9: F5 95     mov     LEDRB,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01AB
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01AB: F5 9E     mov     LEDRC,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01AD
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01AD: F5 F8     mov     LEDG,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01AF
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01AF: F5 30     mov     30,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01B1
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01B1: F5 31     mov     31,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01B3
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01B3: F5 32     mov     32,a

A =00  B =00  SP=7F  IE=00  DPH=00  DPL=00  PSW=00  PC=01B5
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
01B5: F5 33     mov     33,a
>
```

Go

Syntax:

g

Description:

Execute the user code from the current program counter.

Example:

```
> g
```

Set Breakpoint

Syntax:

bron [address]

Description:

This command sets a breakpoint at the given address. Upon execution of the ‘go’ command and after the running program reaches the given address, the user program stops and Cmon51 is reactivated showing the registers.

Example:

```
> u 0 10
0000: 02 E0 00  ljmp    E000
0003: 7A 1E     mov     r2,#1E
0005: 79 FA     mov     r1,#FA
0007: 78 FA     mov     r0,#FA
0009: D8 FE     djnz    r0,0009
000B: D9 FA     djnz    r1,0007
000D: DA F6     djnz    r2,0005
000F: 22        ret
0010: 75 81 7F  mov     SP,#7F
0013: 75 E8 00  mov     LEDRA,#00
0016: 75 95 00  mov     LEDRB,#00
0019: 75 9E 00  mov     LEDRC,#00
001C: 75 F8 00  mov     LEDG,#00
001F: B2 E8     cpl     LEDRA.0
0021: 12 00 03  lcall   0003
0024: 80 F9     sjmp    001F
> bron 000F
> g

A =00  B =00  SP=81  IE=00  DPH=00  DPL=00  PSW=00  PC=000F
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
000F: 22        ret
>
```

Clear Breakpoint

Syntax:

broff [address]

Description:

This command clears a breakpoint at the given address.

Example:

```
> broff 000F
```

List Breakpoints

Syntax:

brl

Description:

This command lists all the activated breakpoints.

Example:

```
> brl
000F
001F
>000F
```

Clear all Breakpoints

Syntax:

brc

Description:

This command clears all breakpoints.

Example:

```
> brl
000F
001F
> brc
> brl
>
```

Display Registers

Syntax:

r

Description:

Display the microcontroller's main registers.

Example:

```
> r

A =00  B =00  SP=81  IE=00  DPH=00  DPL=03  PSW=00  PC=00F9
R0=00  R1=00  R2=00  R3=00  R4=00  R5=00  R6=00  R7=00  BANK 0
```

Dis-assemble

Syntax:

u [address] [number]

Description:

Disassemble [number] opcodes starting at [address].

Example:

```
> u e000 10
E000: 02 E0 06  ljmp    E006
E003: 02 ED 11  ljmp    ED11
E006: 75 81 60  mov     SP,#60
E009: 12 E0 60  lcall   E060
E00C: E5 82     mov     a,DPL
E00E: 60 03     jz      E013
E010: 02 E0 57  ljmp    E057
E013: 75 82 00  mov     DPL,#00
E016: 75 83 00  mov     DPH,#00
E019: 75 84 00  mov     84,#00
E01C: 75 85 00  mov     85,#00
E01F: 78 00     mov     r0,#00
E021: 79 00     mov     r1,#00
E023: E8        mov     a,r0
E024: 49        orl     a,r1
E025: 60 12     jz      E039
>
```

Display Data Memory

Syntax:

d

Description:

Display data memory.

Example:

```
> d
```

```
D:00:  00 00 00 00 00 00 00 00-A3 01 00 00 00 00 00 00    ................
D:10:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:20:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:30:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:40:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:50:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:60:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:70:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Display Idata Memory

Syntax:

i

Description:

Display idata memory.

Example:

```
> i
I:80:  BF 01 F9 00 00 00 03 00-03 00 00 00 00 00 00 00    ................
I:90:  00 00 57 00 8C 03 69 01-A4 00 46 AB 00 00 00 00    ..W...i...F.....
I:A0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:B0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:C0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:D0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:E0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:F0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Display Expanded Memory

Syntax:

x [address] [number]

Description:

Display [number] of external data bytes starting at [address].

Example:

```
> x 0 40
X:0000:  E3 F1 5B EC 5C 6B 87 8B-BA C0 BF 3F 59 E8 D7 FF  ..[.\k.....?Y...
X:0010:  DF 9E 66 31 9F 2C F5 8D-66 FF FE 55 7E DF 64 91  ..f1.,..f..U~.d.
X:0020:  FE A2 97 8E 7A 25 F4 22-E0 AD F5 F8 B7 B3 D6 2E  ....z%."........
X:0030:  DE 6F ED 66 B7 9E 2F FA-7F 4B CB BE A9 F9 FF 7F  .o.f../..K......
```

Display Code Memory

Syntax:

c [address] [number]

Description:

Display [number] of code data bytes starting at [address].

Example:

```
> c 100 30
C:0100:  80 37 30 F1 04 7F 01 80-30 30 F2 04 7F 02 80 29  .70.....00.....)
C:0110:  30 F3 04 7F 03 80 22 30-F4 04 7F 04 80 1B 30 F5  0....."0......0.
C:0120:  04 7F 05 80 14 30 F6 04-7F 06 80 0D 7F 07 80 09  .....0..........
```

Fill Data Memory

Syntax:

fd [address] [number] [value]

Description:

Fill [number] of internal data bytes starting at [address] with [value].

Example:

```
> fd 0 10 55
> d
D:00:  55 55 55 55 55 55 55 55-55 55 55 55 55 55 55 55    UUUUUUUUUUUUUUUU
D:10:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:20:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:30:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:40:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:50:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:60:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:70:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Fill Idata Memory

Syntax:

fi [address] [number] [value]

Description:

Fill [number] of indirect internal data bytes starting at [address] with [value].

Example:

```
> fi 80 20 aa
> i
I:80:  AA AA AA AA AA AA AA AA-AA AA AA AA AA AA AA AA    ................
I:90:  AA AA AA AA AA AA AA AA-AA AA AA AA AA AA AA AA    ................
I:A0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:B0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:C0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:D0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:E0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:F0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Fill Expanded Memory

Syntax:

fx [address] [number] [value]

Description:

Fill [number] of expanded data bytes starting at [address] with [value].

Example:

```
> x 0 20
X:0000:  E3 F1 5B EC 5C 6B 87 8B-BA C0 BF 3F 59 E8 D7 FF  ..[.\k.....?Y...
X:0010:  DF 9E 66 31 9F 2C F5 8D-66 FF FE 55 7E DF 64 91  ..f1.,..f..U~.d.
> fx 0 20 0
> x 0 20
X:0000:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
X:0010:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Display/Change General Purpose Registers

Syntax:

r0[=value]
r1[=value]
r2[=value]
r3[=value]
r4[=value]
r5[=value]
r6[=value]
r7[=value]

Description:

Display the selected register or changes it to [value].

Example:

```
> r0=01
> r0
01
```

Notes:

The r0 to r7 commands display/change the register from the register bank selected with bits RS0 and RS1 of the PSW register.

Hang

Syntax:

hang

Description:

Releases the USB blaster port so it can be used by another application

Example:

```
> hang
Info: Disconnecting from USB-Blaster [USB-0] @1: 5CE(BA4|FA4) (0x02B050DD)
Press Enter to reconnect...
```

Modify Data Memory

Syntax:

md [address]

Description:

Modify data memory starting at [address].

Example:

```
> md 30
D:30=  55
D:31=  66
```

```
D:32=  77
D:33=  'Hello'
D:38=  q
> d
D:00:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:10:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:20:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:30:  55 66 77 48 65 6C 6C 6F-00 00 00 00 00 00 00 00    UfwHello........
D:40:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:50:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:60:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
D:70:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Modify Idata Memory

Syntax:

mi [address]

Description:

Modify idata memory starting at [address].

Example:

```
> mi a0

> mi a0
I:A0=  55
I:A1=  55
I:A2=  20
I:A3=  'ABC'
I:A6=  q

> i
I:80:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:90:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:A0:  55 55 20 41 42 43 00 00-00 00 00 00 00 00 00 00    UU.ABC..........
I:B0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:C0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:D0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:E0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
I:F0:  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

Modify Expanded Memory

Syntax:

mx [address]

Description:

Modify expanded memory starting at [address].

Example:

```
> mx 200
X:0200=  00
X:0201=  11
X:0202=  22
X:0203=  33
X:0204=  q

> x 200 10
X:0200:  00 11 22 33 00 00 00 00-00 00 00 00 00 00 00 00    .."3............
```

Modify Code Memory

Syntax:

mc [address]

Description:

Modify code memory starting at [address]. Note: These changes are not stored in flash memory. After a reset, the code memory is restored to its original values.

Example:

```
> mc 10
C:0010=  'Hello!'
C:0016=  00
C:0017=  q

> c 10 10
C:0010:  48 65 6C 6C 6F 21 00 00-00 00 00 00 00 00 00 00    Hello!..........
```

Display/Change Special Function Registers

Syntax:

ACC[=value]
A[=value]
PC[=value]
PSW[=value]
B[=value]
IE[=value]
DPL[=value]
DPH[=value]
DPTR[=value]
SP[=value]
Any_SFR_name[=value]

Description:

Display the selected register or changes it to [value]. Any SFR name for the current processor can be displayed or changed this way. For most SFR, bits values can also be changed individually by using the SFR name followed by period ('.') and the bit number. This works even for non-bit-addressable SFR, if the SFR can be read by the Cmon51.

Examples:

```
> ACC
00
> ACC.1=1
> ACC
02
```

Display/Change Special Function Bits

Syntax:

CY[=value]
AC[=value]
FO[=value]
RS1[=value]
RS0[=value]
OV[=value]
UD[=value]

P[=value]
[Any SFR bit name][=value]

Description:

Display the selected bit or changes it to [value]. Any read/write SFR bit for the current microcontroller can be displayed or changed this way.

Examples:

```
> cy=1
> cy
1
> cy=0
> cy
0
> LEDRA.0=1
```

Display/Change Special Function Bits

Syntax:

exit

Description:

Terminates the debug session.

Example:

```
> exit
Info (23030): Evaluation of Tcl script cmon51.tcl was successful
Info: Quartus II 64-Bit SignalTap II was successful. 0 errors, 0 warnings
    Info: Peak virtual memory: 370 megabytes
    Info: Processing ended: Wed Nov 11 10:39:37 2015
    Info: Elapsed time: 00:33:07
    Info: Total CPU time (on all processors): 00:00:34
```