Computer Architecture - Lab Assignment 1

RISC-V instruction set architecture and programming of Nios V/m processor

This is an introductory exercise that involves Intel/Altera's Nios V/m processor and its RISC-V assembly language. It uses a simple computer system, called the DE0-Nano Basic Computer, which includes the Nios V/m processor. The system is implemented as a circuit that is downloaded into the Field Programmable Gate Array (FPGA) device on the DE0-Nano board. This exercise illustrates how programs written in the RISC-V assembly language can be executed on the DE0-Nano board.

To prepare for this exercise you have to know the Nios V/m processor architecture and its RISC-V assembly language. This lab consists of four parts. Part I below describes the procedure for compiling, linking RISC-V architecture assembler programs, and running the programs on the DE0-Nano board.

Part I: executing an example program

In this part we will use the **Nios V Command Shell** to download the DE0-Nano Basic Computer circuit into the FPGA device and execute a sample program. This tool is part of the *Intel/Altera Quartus Prime Standard 23.1 Design Suite*.

Perform the following:

1. Turn on the power to the Terasic DE0-Nano board.
2. Open the Nios V Command Shell, which leads to the window in Figure 1.

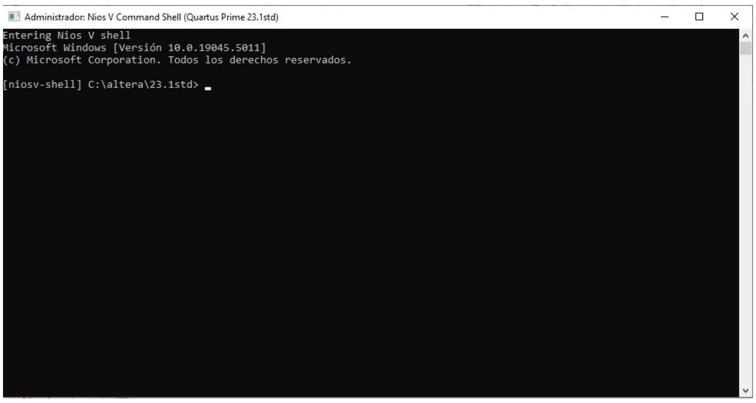


Figure 1: The Nios V Command Shell Window.

   To run an application program it is necessary to create a new BSP project.

3. Create a new BSP directory called for example: `lab1_bsp`.
4. Select a predesigned *System-on-Chip* File: `nios_system_22jul24.sopcinfo`. This file was obtained by using the *Platform Designer* tool. It integrates a Nios V/m soft processor and a 8 KB on-chip SRAM memory, in addition to an I/O controller for the LEDs available in the board.

5. Execute the following command in the Nios V Command Shell Window.

```
$ cd lab1_bsp
$ bash
$ niosv-bsp.exe -c -t=hal -s=nios_system_22jul24.sopcinfo settings.bsp
```

   The output provides two files called: `settings.bsp` and `linker.x`. Figure 2 shows the messages displayed in the Nios V Command Shell window.

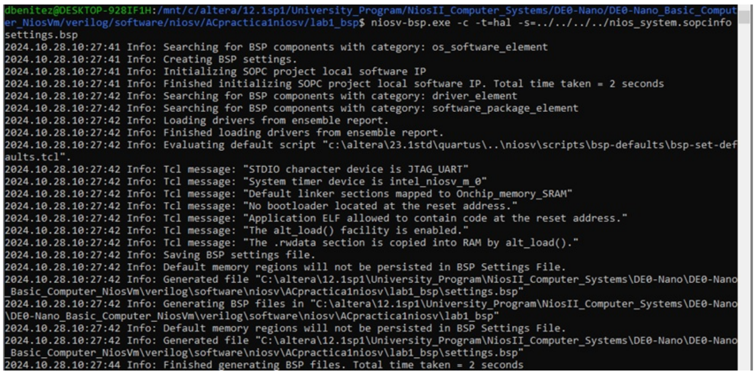


Figure 2: Messages displayed after executing the `niosv-bsp.exe` command.

6. Now, create a new directory for the RISC-V assembler program written in RISC-V assembly language and its building, for example: `lab1_bin`.

7. Copy the sample program `lab1_part1.s` to this directory.

8. The source file `lab1_part1.s` contains the application program. This file specifies the starting point in the selected application program. The default symbol is start, which is used in the selected sample program.

9. Execute the following command in the Nios V Command Shell window.

```
$ cd lab1_bin
$ riscv32-unknown-elf-as.exe lab1_part1.s -o lab1_part1.s.obj
```

   The output is a file called: `lab1_part1.s.obj`.

10. Execute the following command in the Nios V Command Shell window.

```
$ riscv32-unknown-elf-ld.exe -g -T ../lab1_bsp_bsp/linker.x -nostdlib
-e _start -u _start --defsym __alt_stack_pointer=0x08001F00 --defsym
__alt_stack_base=0x08002000 --defsym __alt_heap_limit=0x8002000 --defsym
__alt_heap_start=0x8002000 -o lab1_part1.elf lab1_part1.s.obj
$ niosv-stack-report.exe -p riscv32-unknown-elf- lab1_part1.elf
```

   The output is a file called: `lab1_part1.elf`. Figure 3 shows the messages displayed in the Nios V Command Shell window.

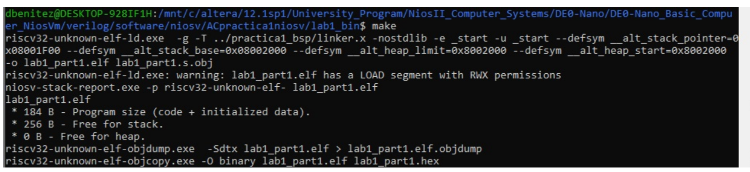
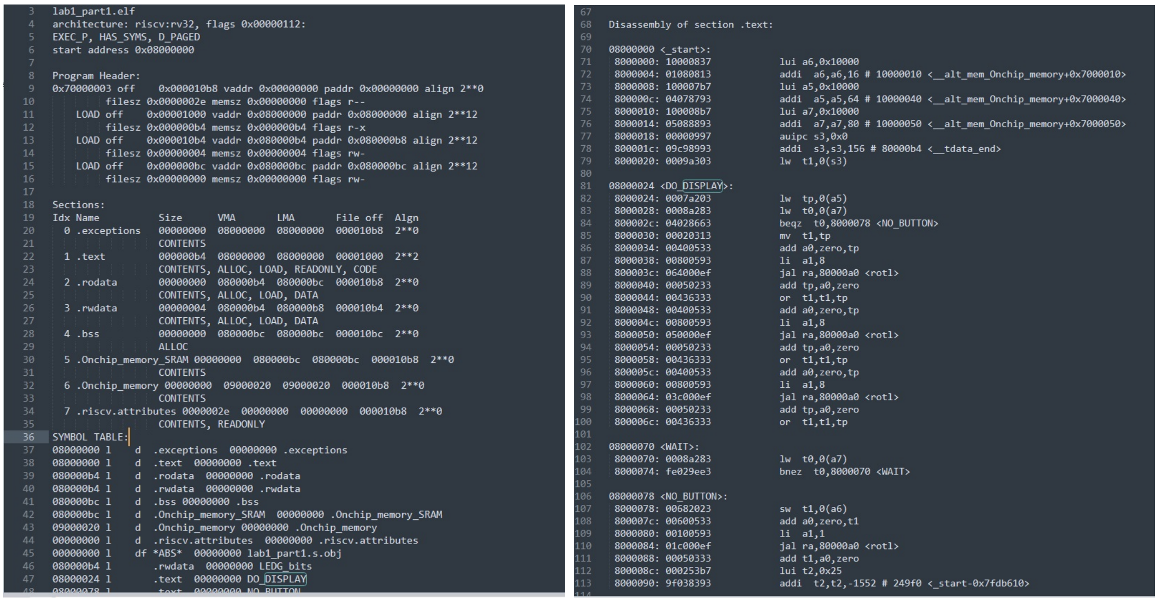


Figure 3: Messages displayed after executing the `riscv32-unknown-elf-ld.exe` and `niosv-stack-report.exe` commands.

11. Execute the following command in the Nios V Command Shell window.

```
$ riscv32-unknown-elf-objdump.exe -Sdtx lab1_part1.elf > lab1_part1.elf.objdump
```

The output is a file called: `lab1_part1.elf.objdump`. Figure 4 shows the RISC-V machine instructions, addresses and assembled instructions that are included in the `lab1_part1.elf.objdump` file.



```
lab1_part1.elf
architecture: riscv:rv32, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x08000000

Program Header:
0x70000003 off    0x000010b8 vaddr 0x00000000 paddr 0x00000000 align 2**0
         filesz 0x0000002e memsz 0x00000000 flags r--
    LOAD off    0x00001000 vaddr 0x08000000 paddr 0x08000000 align 2**12
         filesz 0x000000b4 memsz 0x000000b4 flags r-x
    LOAD off    0x000010b4 vaddr 0x080000b4 paddr 0x080000b8 align 2**12
         filesz 0x00000004 memsz 0x00000004 flags rw-
    LOAD off    0x000000bc vaddr 0x080000bc paddr 0x080000bc align 2**12
         filesz 0x00000000 memsz 0x00000000 flags rw-

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .exceptions   00000000  08000000  08000000  000010b8  2**0
                  CONTENTS
  1 .text         000000b4  08000000  08000000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .rodata       00000000  080000b4  080000bc  000010b8  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  3 .rwdata       00000004  080000b4  080000b8  000010b4  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  4 .bss          00000000  080000bc  080000bc  000010bc  2**0
                  ALLOC
  5 .Onchip_memory_SRAM 00000000  080000bc  080000bc  000010b8  2**0
                  CONTENTS
  6 .Onchip_memory 00000000  09000020  09000020  000010b8  2**0
                  CONTENTS
  7 .riscv.attributes 0000002e  00000000  00000000  000010b8  2**0
                  CONTENTS, READONLY
SYMBOL TABLE:
08000000 l    d  .exceptions  00000000 .exceptions
08000000 l    d  .text  00000000 .text
080000b4 l    d  .rodata  00000000 .rodata
080000b4 l    d  .rwdata  00000000 .rwdata
080000bc l    d  .bss 00000000 .bss
080000bc l    d  .Onchip_memory_SRAM  00000000 .Onchip_memory_SRAM
09000020 l    d  .Onchip_memory 00000000 .Onchip_memory
00000000 l    d  .riscv.attributes  00000000 .riscv.attributes
00000000 l    df *ABS*  00000000 lab1_part1.s.obj
080000b4 l       .rwdata  00000000 LEDG_bits
08000024 l       .text  00000000 DO_DISPLAY
```

```

Disassembly of section .text:

08000000 <_start>:
 8000000: 10000837          lui a6,0x10000
 8000004: 01080813          addi  a6,a6,16 # 10000010 <__alt_mem_Onchip_memory+0x7000010>
 8000008: 100007b7          lui a5,0x10000
 800000c: 04078793          addi  a5,a5,64 # 10000040 <__alt_mem_Onchip_memory+0x7000040>
 8000010: 100008b7          lui a7,0x10000
 8000014: 05088893          addi  a7,a7,80 # 10000050 <__alt_mem_Onchip_memory+0x7000050>
 8000018: 00000997          auipc s3,0x0
 800001c: 09c98993          addi  s3,s3,156 # 80000b4 <__tdata_end>
 8000020: 0009a303          lw  t1,0(s3)

08000024 <DO_DISPLAY>:
 8000024: 0007a203          lw  tp,0(a5)
 8000028: 0008a283          lw  t0,0(a7)
 800002c: 04028663          beqz  t0,8000078 <NO_BUTTON>
 8000030: 00020313          mv  t1,tp
 8000034: 00400533          add a0,zero,tp
 8000038: 00800593          li  a1,8
 800003c: 064000ef          jal ra,80000a0 <rotl>
 8000040: 00050233          add tp,a0,zero
 8000044: 00436333          or  t1,t1,tp
 8000048: 00400533          add a0,zero,tp
 800004c: 00800593          li  a1,8
 8000050: 050000ef          jal ra,80000a0 <rotl>
 8000054: 00050233          add tp,a0,zero
 8000058: 00436333          or  t1,t1,tp
 800005c: 00400533          add a0,zero,tp
 8000060: 00800593          li  a1,8
 8000064: 03c000ef          jal ra,80000a0 <rotl>
 8000068: 00050233          add tp,a0,zero
 800006c: 00436333          or  t1,t1,tp

08000070 <WAIT>:
 8000070: 0008a283          lw  t0,0(a7)
 8000074: fe029ee3          bnez  t0,8000070 <WAIT>

08000078 <NO_BUTTON>:
 8000078: 00682023          sw  t1,0(a6)
 800007c: 00600533          add a0,zero,t1
 8000080: 00100593          li  a1,1
 8000084: 01c000ef          jal ra,80000a0 <rotl>
 8000088: 00050333          add t1,a0,zero
 800008c: 000253b7          lui t2,0x25
 8000090: 9f038393          addi  t2,t2,-1552 # 249f0 <_start-0x7fdb610>
```

Figure 4: Content of the file `lab1_part1.elf.objdump`.

Now, it is needed to download the soft SoC system associated with this program onto the DE0-Nano board. Make sure that the power to the DE0-Nano board is turned on.

12. Execute the following command in the Nios V Command Shell window.

```
$ jtagconfig.exe
```

The following message must appear in the Nios V Command Shell window. In the case the message does not show, repeat again the command.

```
1) USB-Blaster [USB-0]
020F30DD 10CL025(Y|Z)/EP3C25/EP4CE22
```

13. Execute the following command in the Nios V Command Shell window.

```
$ quartus_pgm.exe -c 1 -m JTAG -o "p;DE0_Nano_Basic_Computer_22jul24.sof@1"
```

Watch the change in state of the blue LEDs on the DE0-Nano board that correspond to LOAD and GOOD, which will blink as the circuit is being downloaded. Figure 5 shows the messages displayed on the screen.

14. Having downloaded the `sof` configuration DE0-Nano Basic Computer into the FPGA chip on the DE0-Nano board, we can now load and run programs on this SoC computer. In the Nios V Command Shell window, execute the following command.

```
$ niosv-download.exe -g lab1_part1.elf
```

This command also run the program.

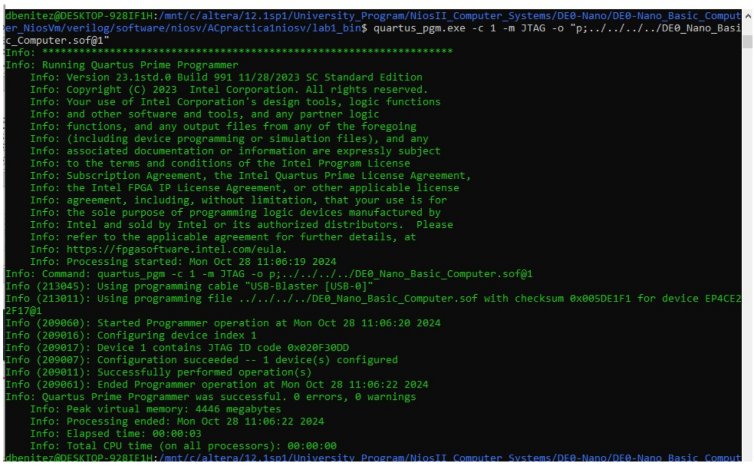


Figure 5: Messages displayed on the screen after configuring the DE0-Nano board.

15. Observe the LEDs located on the board are turning on and off quickly (see Figure 6). This test provides an indication that the DE0-Nano board is functioning properly. Stop the execution of the sample program by typing "CTRL+c".

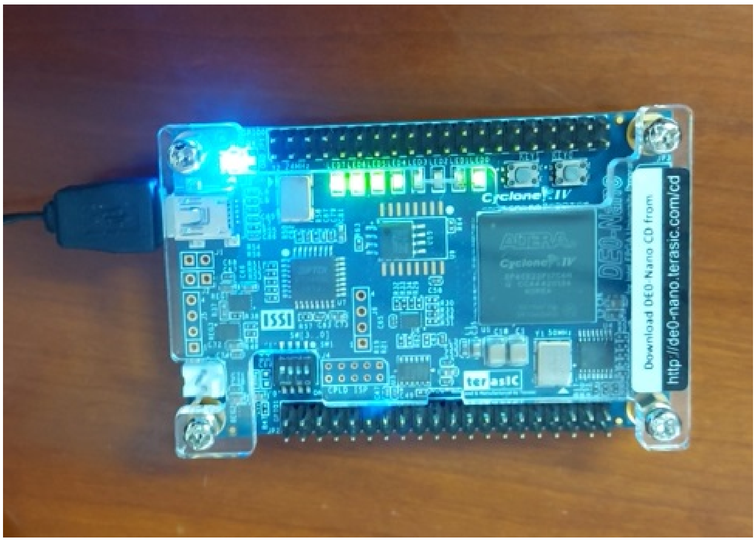


Figure 6: LEDs on the board are turning on and off.