The Monitor Program for Nios V

Part 1 - Introduction

This document introduces the *Monitor Program* application software, which allows you to develop and debug Nios V code that is being executed on a *real* DE1-SoC board. You can't use *CPUlator* to control a real hardware board, because the *CPUlator* only *simulates* the features of the DE1- SoC Computer and does not use the actual hardware.

To use the Monitor Program software, your computer must be connected by a USB cable to a DE1-SoC board (the cable has to be plugged into the *USB Blaster* port on the board). In a similar manner to using the *CPUlator*, you will develop software code that runs on the DE1-SoC Computer, which includes Nios V, memory, and various I/O devices. However, in this case you are not using a *simulation* of the DE1-SoC Computer. Instead, the *DE1-SoC Computer* is implemented as a circuit which is downloaded by the Monitor Program application into the FPGA device on the DE1-SoC board. You use the Monitor Program to control the Nios V processor on the board.

The remaining parts of this exercise require the use of a DE1-SoC board. The Monitor Program cannot be used at all without a hardware board attached to your computer.

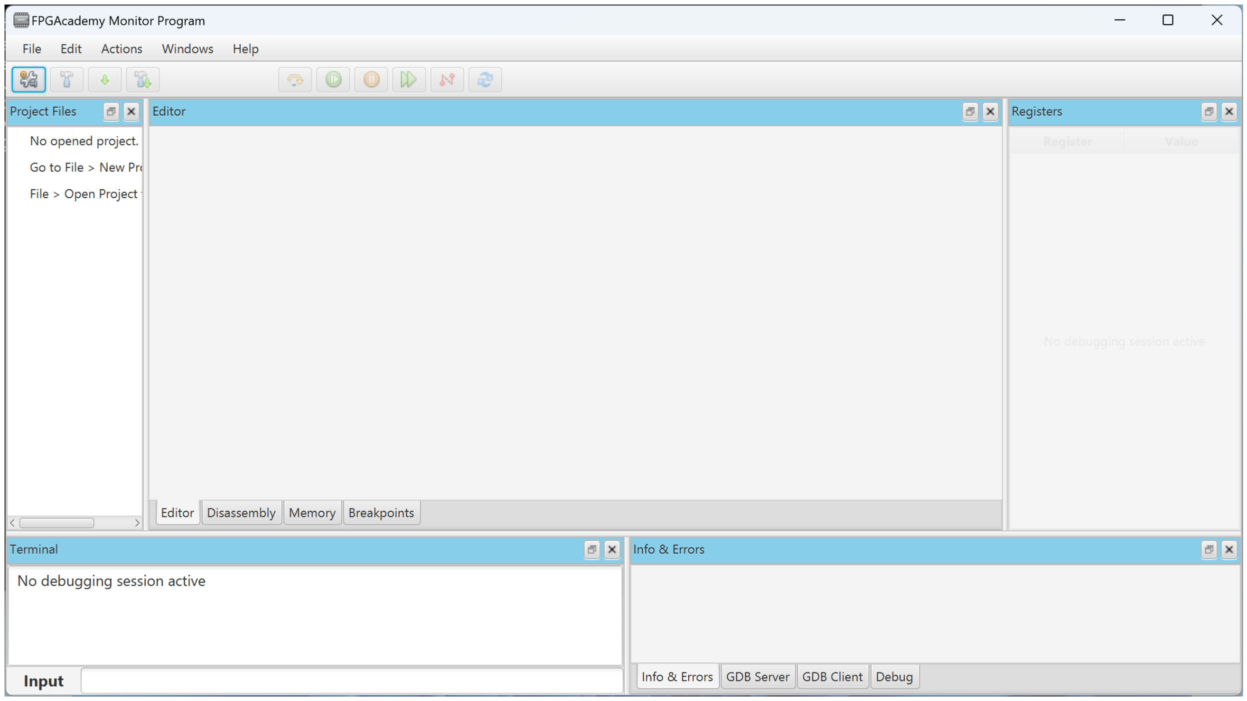
The first step in using the Monitor Program is to set up a Nios V software development project. Perform the following:

1. Make sure that the power is turned on for the DE1-SoC board.

2. Open the *Monitor Program* software, which leads to the window in Figure 13.

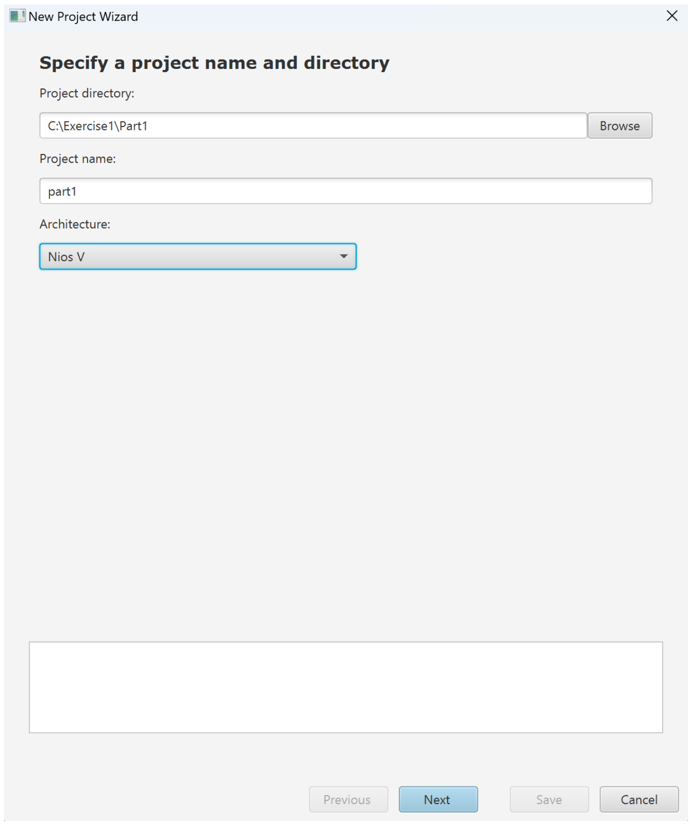
   To develop Nios V software code using the Monitor Program it is necessary to create a new project. Select File > New Project to reach the window in Figure 14. Give the project a name and indicate the folder for the project; we have chosen the project name *part1* in the folder *Exercise1\Part1*, as indicated in the figure. Use the drop-down menu shown in Figure 14 to set the target architecture to Nios V. Click Next, to get the window in Figure 15.

3. Now, you can select your own custom computer system (if you have one) or a pre-designed (by Altera) system. As shown in Figure 15 select the *DE1-SoC Computer*. Once you have selected the computer system the display in the window will now show where files that implement the chosen system are located. Click Next.



No debugging session active

Figure 13: The *Monitor Program* window.



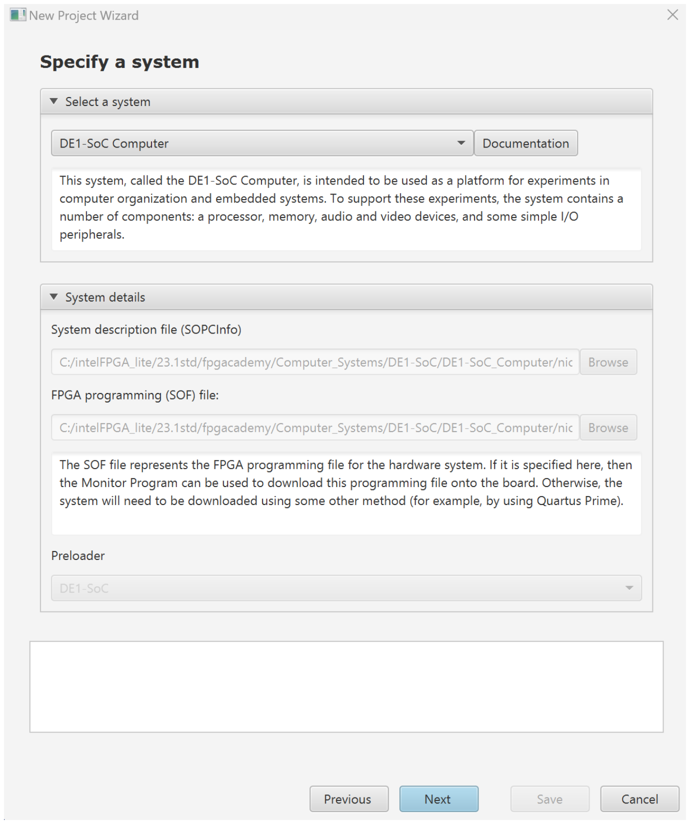
Specify a project name and directory

Project directory: C:\Exercise1\Part1

Project name: part1

Architecture: Nios V

Figure 14: Specify the folder and the name of the project.



This system, called the DE1-SoC Computer, is intended to be used as a platform for experiments in computer organization and embedded systems. To support these experiments, the system contains a number of components: a processor, memory, audio and video devices, and some simple I/O peripherals.

System description file (SOPCInfo)

C:/intelFPGA_lite/23.1std/fpgacademy/Computer_Systems/DE1-SoC/DE1-SoC_Computer/nic

FPGA programming (SOF) file:

C:/intelFPGA_lite/23.1std/fpgacademy/Computer_Systems/DE1-SoC/DE1-SoC_Computer/nic

The SOF file represents the FPGA programming file for the hardware system. If it is specified here, then the Monitor Program can be used to download this programming file onto the board. Otherwise, the system will need to be downloaded using some other method (for example, by using Quartus Prime).

Preloader

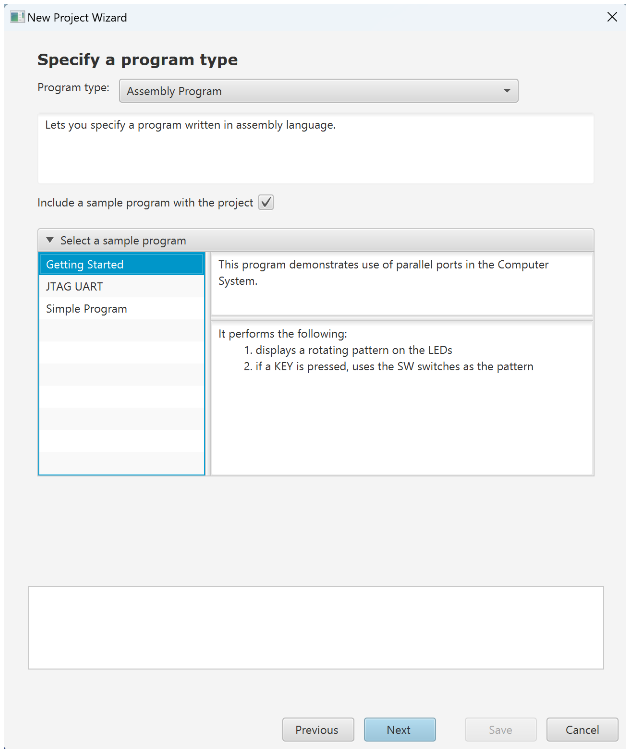
DE1-SoC

Figure 15: Specification of the system.

4. In the window in Figure 16 you can specify the type of application programs that you wish to run. They can be written in either assembly language or the C programming language. Specify that an assembly language program will be used. The *Monitor Program* package contains several sample programs. Select the box Include a sample program with the project. Then, choose the Getting Started program, as indicated in the figure, and click Next.

5. The window in Figure 17 is used to specify the source file(s) that contain the application program(s). Since we have selected the *Getting Started* program, the window indicates the source code file for this program. This window also allows the user to specify the starting point in the selected application program. The default symbol is *_start*, which is used in the selected sample program. Click Next.

6. The window in Figure 18 indicates some system parameters. Note that the figure indicates that the *DE-SoC [USB-1]* cable is selected to provide the connection between the DE-series board and the host computer. This is the name assigned to the Altera USB Blaster connection between the computer and the board. If no `Host Connection` is shown, click on the `Refresh` button.



Specify a program type

Program type: Assembly Program

Lets you specify a program written in assembly language.

Include a sample program with the project ✓

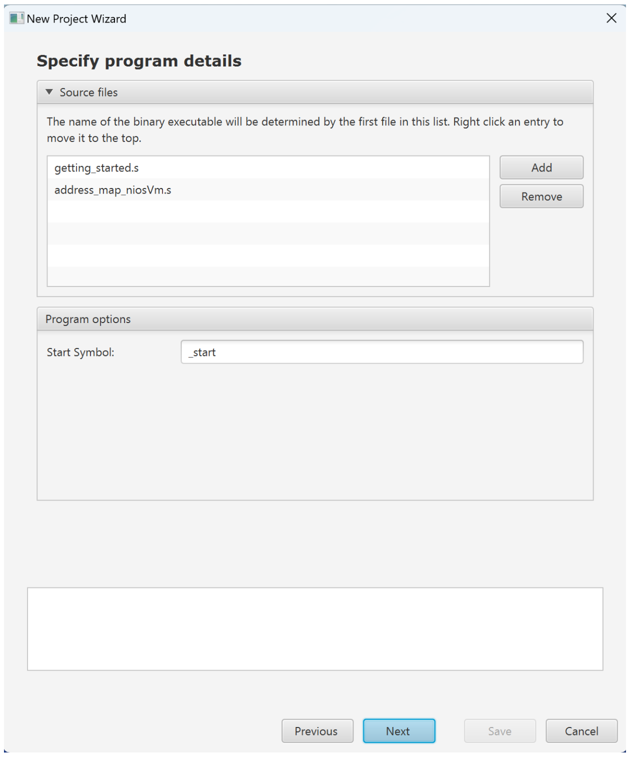
Select a sample program

- Getting Started
- JTAG UART
- Simple Program

This program demonstrates use of parallel ports in the Computer System.

It performs the following:
1. displays a rotating pattern on the LEDs
2. if a KEY is pressed, uses the SW switches as the pattern

Figure 16: Selection of an application program.



Specify program details

Source files

The name of the binary executable will be determined by the first file in this list. Right click an entry to move it to the top.

- getting_started.s
- address_map_niosVm.s

Program options

Start Symbol: _start

Figure 17: Source files used by the application program.

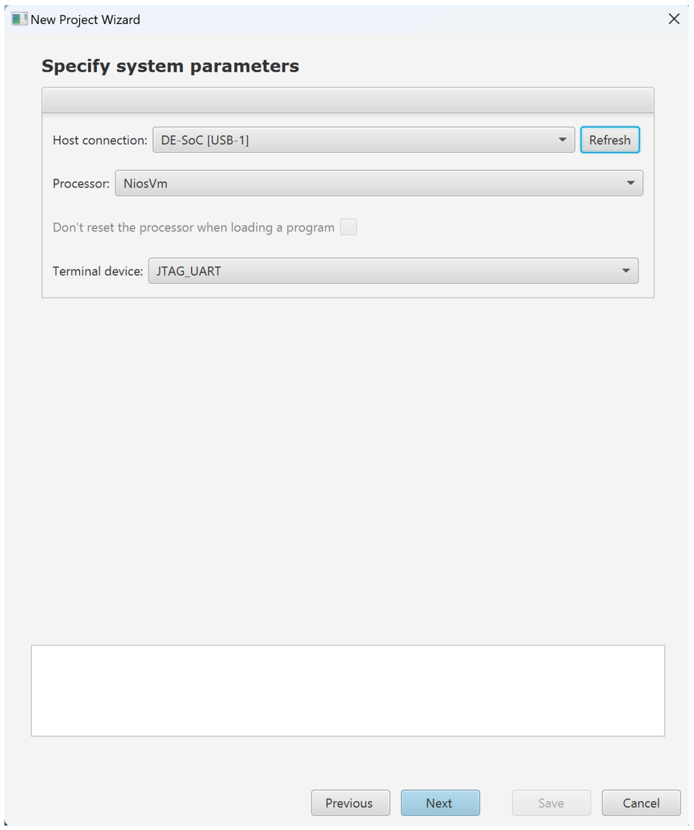
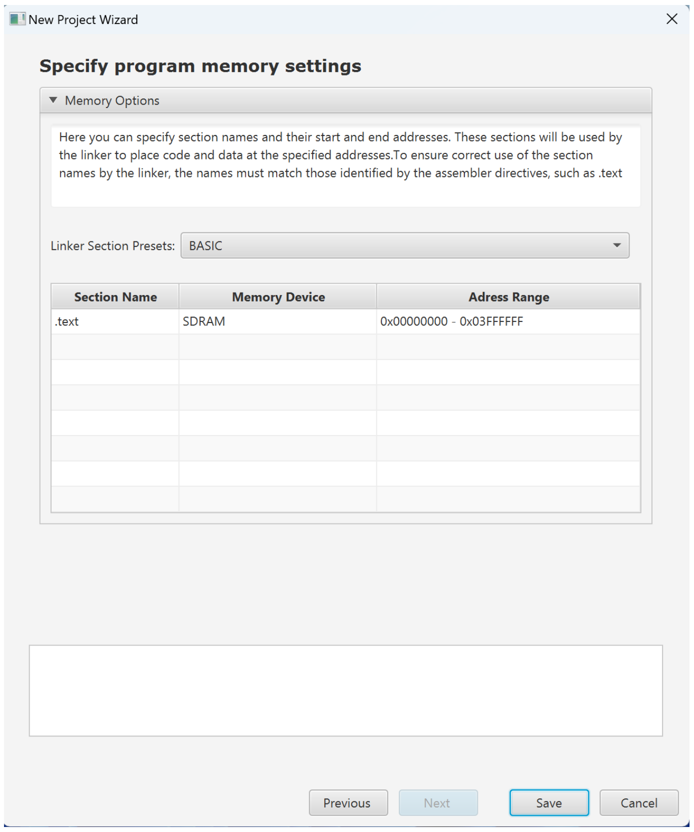

New Project Wizard
Specify system parameters
Host connection: DE-SoC [USB-1]
Refresh
Processor: NiosVm
Don't reset the processor when loading a program
Terminal device: JTAG_UART
Previous
Next
Save
Cancel

Figure 18: Specify the system parameters.

7. The window in Figure 19 displays the names of Assembly sections that will be used for the program, and allows the user to select a target memory location for each section. In this case only the *.text* section, which corresponds to the program code (and data), will be used. As shown in the figure, the *.text* section is targeted to the memory in the DE-series board that is mapped to start at address 0. Click Save to complete the specification of the new project.

8. After your project is saved, you can download the DE1-SoC Computer to your board. Click on `Actions > Download system`. Observe the messages in the `Info & Errors` area in the Monitor Program screen to see when the *Quartus Programmer*, which performs the download, is finished.



Specify program memory settings

Memory Options

Here you can specify section names and their start and end addresses. These sections will be used by the linker to place code and data at the specified addresses.To ensure correct use of the section names by the linker, the names must match those identified by the assembler directives, such as .text

Linker Section Presets: BASIC

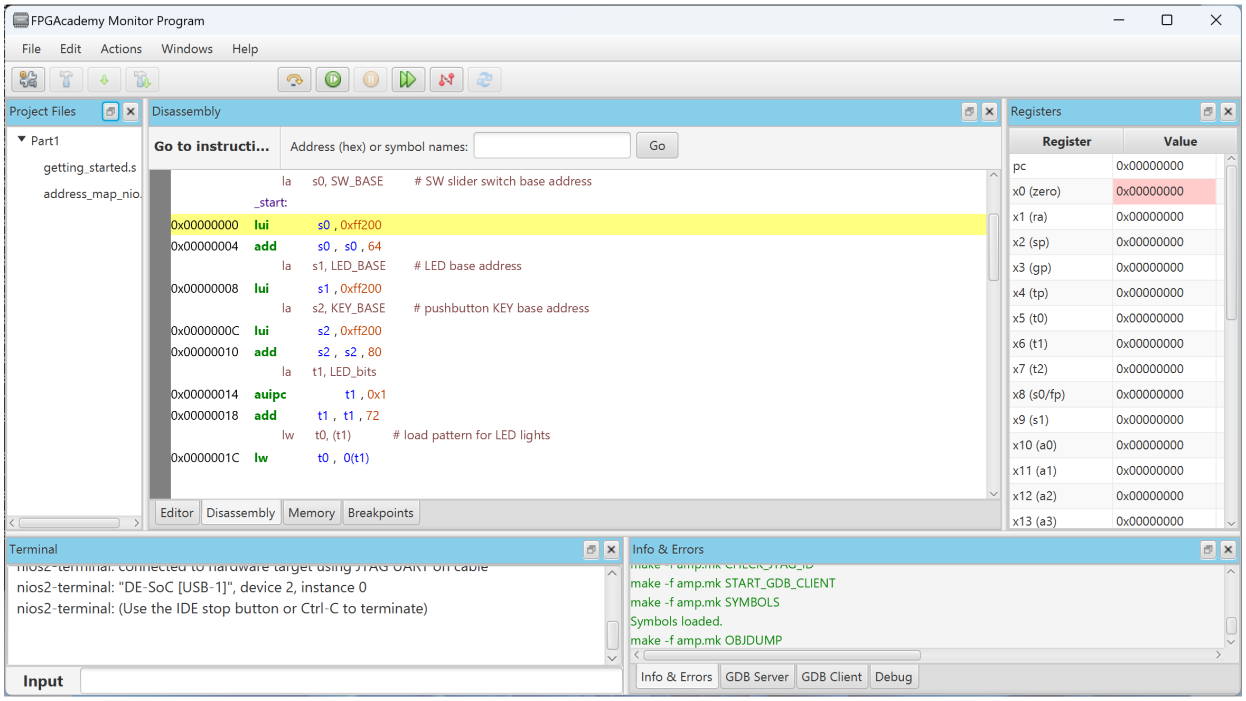
Section Name	Memory Device	Adress Range
.text	SDRAM	0x00000000 - 0x03FFFFFF

Figure 19: Specify the program memory settings.

9. Having downloaded the computer system into the FPGA on your DE1-SoC board, we can now load and run the sample program. In the main Monitor Program window, shown in Figure 20, select Actions > Compile & Load to assemble the program and load it into the memory on the board. Figure 20 shows the Monitor Program window after the sample program has been loaded.

10. Run the program by selecting Actions > Continue or by clicking on the toolbar icon, and observe the patterns displayed on the LEDs.

11. Pause the execution of the sample program by clicking on the icon, and disconnect from this session by clicking on the icon,



```
                la     s0, SW_BASE        # SW slider switch base address
           _start:
0x00000000 lui       s0 , 0xff200
0x00000004 add       s0 ,  s0 , 64
                la     s1, LED_BASE       # LED base address
0x00000008 lui       s1 , 0xff200
                la     s2, KEY_BASE       # pushbutton KEY base address
0x0000000C lui       s2 , 0xff200
0x00000010 add       s2 ,  s2 , 80
                la     t1, LED_bits
0x00000014 auipc          t1 , 0x1
0x00000018 add       t1 ,  t1 , 72
                lw     t0, (t1)         # load pattern for LED lights
0x0000001C lw        t0 ,  0(t1)
```

Register	Value
pc	0x00000000
x0 (zero)	0x00000000
x1 (ra)	0x00000000
x2 (sp)	0x00000000
x3 (gp)	0x00000000
x4 (tp)	0x00000000
x5 (t0)	0x00000000
x6 (t1)	0x00000000
x7 (t2)	0x00000000
x8 (s0/fp)	0x00000000
x9 (s1)	0x00000000
x10 (a0)	0x00000000
x11 (a1)	0x00000000
x12 (a2)	0x00000000
x13 (a3)	0x00000000

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE-SoC [USB-1]", device 2, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)
```

```
make -f amp.mk CHECK_JTAG_ID
make -f amp.mk START_GDB_CLIENT
make -f amp.mk SYMBOLS
Symbols loaded.
make -f amp.mk OBJDUMP
```

Figure 20: The monitor window showing the loaded sample program.

Part 2 - More Features

Now, we will explore some features of the Monitor Program by using a Nios V assembly-language program which finds the largest number in a list of 32-bit integers that is stored in the memory. The code is shown in Figure 21. It is also in files available with this lab in the file part2.s.

```
# Program that finds the largest number in a list of integers
            .text
            .global _start
_start:     la      t0, result      # t0 = pointer to the result
            lw      t1, 4(t0)       # t1 = counter, initialized with N
            addi    t2, t0, 8       # t2 = pointer to the first number
            lw      t3, (t2)        # t3 = largest found so far
loop:       addi    t1, t1, -1      # decrement counter
            beqz    t1, done        # done when counter is 0
            addi    t2, t2, 4       # point to the next number
            lw      t4, (t2)        # get the next number
            bge     t3, t4, loop    # compare to largest found
            mv      t3, t4          # remember new largest
            j       loop
done:       sw      t3, (t0)        # store result

stop:       j       stop            # wait here

            .data
result:     .word   0               # result will be stored here
N:          .word   7               # number of entries in the list
numbers:    .word   4, 5, 3, 6      # numbers in the list
            .word   1, 8, 2         # ...
```

Figure 21: Assembly-language program that finds the largest number.

Perform the following:

1. Create a new folder for this part of the exercise, with a name such as *Part2*. Create a file named *part2.s* and enter the code from Figure 21 into this file. Use the Monitor Program to create a new project in this folder; we have chosen the project name *part2*. When you reach the window in Figure 16 choose Assembly Program but do not select a sample program. Click Next.

2. Upon reaching the window in Figure 17, you have to specify the source code file for your program. Click Add and in the pop-up box that appears indicate the desired file name, *part2.s*. Click Next to get to the window in Figure 18. Again click Next to get to the window in Figure 19. Notice that the SDRAM is selected as the memory device. Your program will be loaded starting at address 0 in this memory. Click Save. Note that you do not need to download the DE1-SoC Computer system to your board, as you already programmed the FPGA chip when performing Part VI.

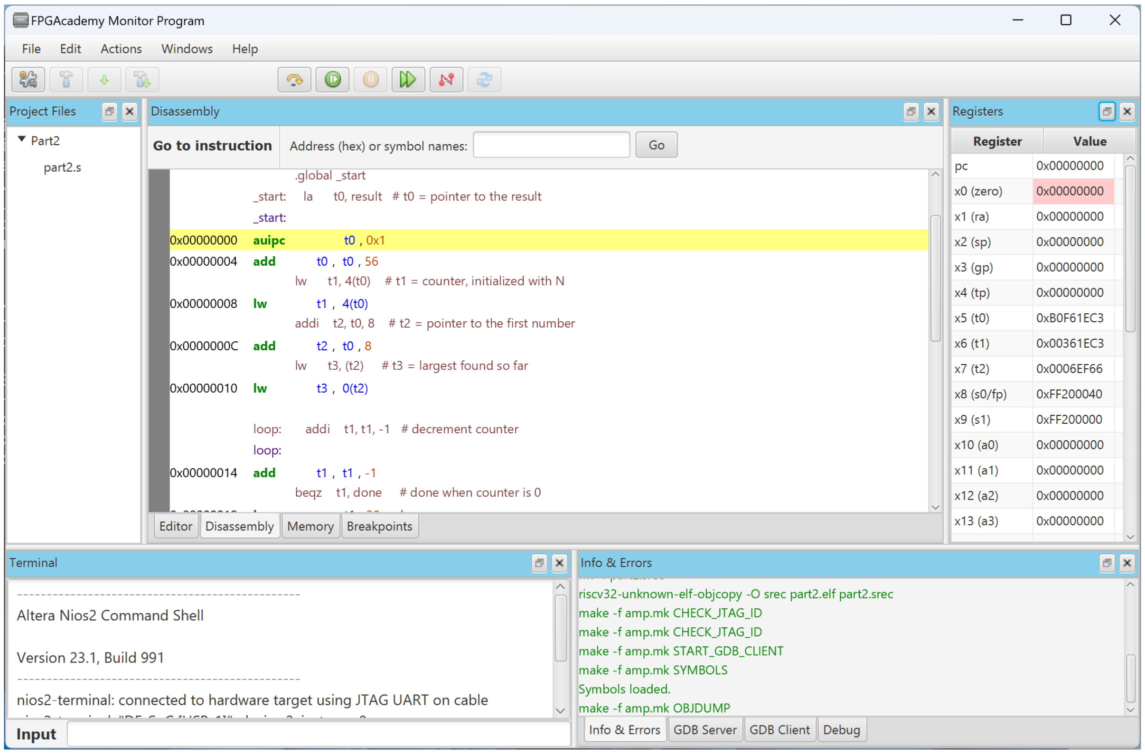
3. Compile and load the program. The Monitor Program will display a disassembled view of the machine code loaded in the memory, as indicated in Figure 22.

4. Execute the program. When the code is running, you will not be able to see any changes (such as the contents of registers or memory locations) in the display for the Monitor Program, because it does not communicate with the Nios V processor while code is being executed. But, if you pause the program then the Monitor Program window will be updated. Pause the program and observe that the processor stops within the endless loop **stop: j stop**. Note that the largest number found in the sample list is 8 as indicated by the contents of register *t3*. This result is also stored in memory at the label *result*. The address of the label *result* for this program in the Monitor Program is 0x00000038. Use the Monitor Program's `Memory` tab, as illustrated in Figure 23, to verify that the resulting value 8 is stored in the correct location.

5. You can return control of the program to the start by clicking on the icon , or by selecting Actions > Restart. Do this and then single-step through the program by clicking on the icon . Watch how the instructions change the data in the processor's registers.

6. Double-click on the pc register in the Monitor Program and then set the program counter to 0. This action has the same effect as clicking on the restart icon ⏩.

7. Now set a breakpoint at address 0x0000002C by clicking on the gray bar to the left of this address, as illustrated in Figure 24. Restart the program and run it again. Observe the contents of register *t3* each time the instruction at the breakpoint, which is `j loop`, is reached.



```
                      .global _start
               _start:   la     t0, result   # t0 = pointer to the result
               _start:
0x00000000    auipc         t0 , 0x1
0x00000004    add        t0 ,  t0 , 56
                      lw      t1, 4(t0)    # t1 = counter, initialized with N
0x00000008    lw         t1 ,  4(t0)
                      addi    t2, t0, 8    # t2 = pointer to the first number
0x0000000C    add        t2 ,  t0 , 8
                      lw      t3, (t2)     # t3 = largest found so far
0x00000010    lw         t3 ,  0(t2)

               loop:     addi    t1, t1, -1    # decrement counter
               loop:
0x00000014    add        t1 ,  t1 , -1
                      beqz    t1, done      # done when counter is 0
```

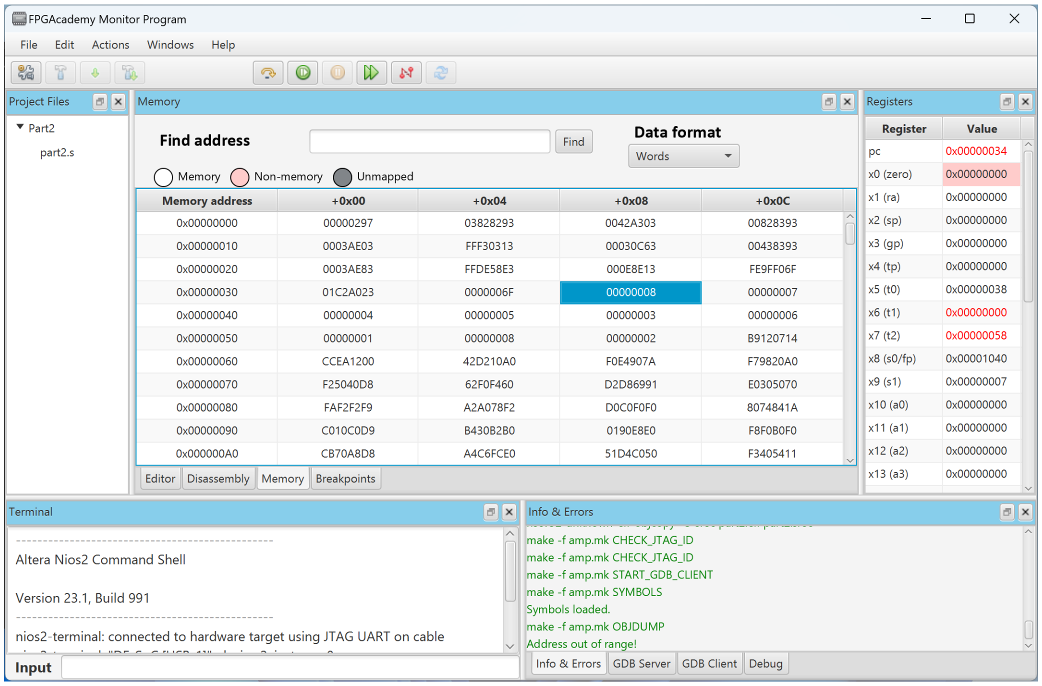
Register	Value
pc	0x00000000
x0 (zero)	0x00000000
x1 (ra)	0x00000000
x2 (sp)	0x00000000
x3 (gp)	0x00000000
x4 (tp)	0x00000000
x5 (t0)	0xB0F61EC3
x6 (t1)	0x00361EC3
x7 (t2)	0x0006EF66
x8 (s0/fp)	0xFF200040
x9 (s1)	0xFF200000
x10 (a0)	0x00000000
x11 (a1)	0x00000000
x12 (a2)	0x00000000
x13 (a3)	0x00000000

```
------------------------------------------------
Altera Nios2 Command Shell

Version 23.1, Build 991
------------------------------------------------
nios2-terminal: connected to hardware target using JTAG UART on cable
```

```
riscv32-unknown-elf-objcopy -O srec part2.elf part2.srec
make -f amp.mk CHECK_JTAG_ID
make -f amp.mk CHECK_JTAG_ID
make -f amp.mk START_GDB_CLIENT
make -f amp.mk SYMBOLS
Symbols loaded.
make -f amp.mk OBJDUMP
```

Figure 22: The disassembled view of the program in Figure 1.


FPGAcademy Monitor Program
File Edit Actions Windows Help
Project Files
Part2
part2.s
Memory
Find address
Find
Data format
Words
Memory Non-memory Unmapped
Editor Disassembly Memory Breakpoints
Registers
Terminal
Input
Info & Errors GDB Server GDB Client Debug

Memory address	+0x00	+0x04	+0x08	+0x0C
0x00000000	00000297	03828293	0042A303	00828393
0x00000010	0003AE03	FFF30313	00030C63	00438393
0x00000020	0003AE83	FFDE58E3	000E8E13	FE9FF06F
0x00000030	01C2A023	0000006F	00000008	00000007
0x00000040	00000004	00000005	00000003	00000006
0x00000050	00000001	00000008	00000002	B9120714
0x00000060	CCEA1200	42D210A0	F0E4907A	F79820A0
0x00000070	F25040D8	62F0F460	D2D86991	E0305070
0x00000080	FAF2F2F9	A2A078F2	D0C0F0F0	8074841A
0x00000090	C010C0D9	B430B2B0	0190E8E0	F8F0B0F0
0x000000A0	CB70A8D8	A4C6FCE0	51D4C050	F3405411

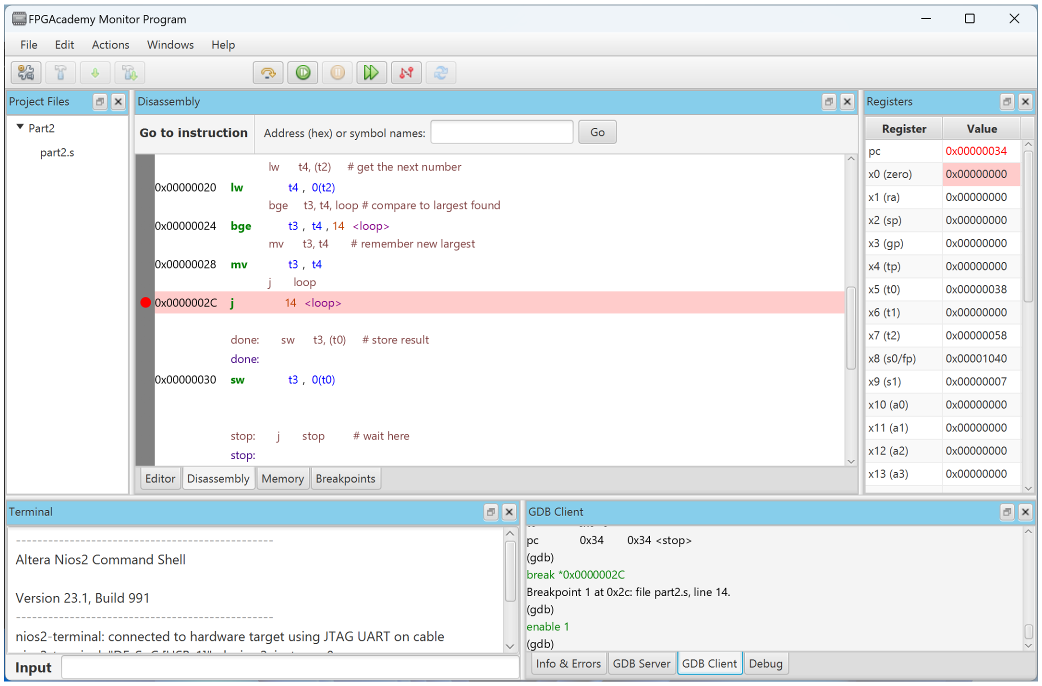
Register	Value
pc	0x00000034
x0 (zero)	0x00000000
x1 (ra)	0x00000000
x2 (sp)	0x00000000
x3 (gp)	0x00000000
x4 (tp)	0x00000000
x5 (t0)	0x00000038
x6 (t1)	0x00000000
x7 (t2)	0x00000058
x8 (s0/fp)	0x00001040
x9 (s1)	0x00000007
x10 (a0)	0x00000000
x11 (a1)	0x00000000
x12 (a2)	0x00000000
x13 (a3)	0x00000000

```
------------------------------------------------
Altera Nios2 Command Shell

Version 23.1, Build 991
------------------------------------------------
nios2-terminal: connected to hardware target using JTAG UART on cable
```

```
make -f amp.mk CHECK_JTAG_ID
make -f amp.mk CHECK_JTAG_ID
make -f amp.mk START_GDB_CLIENT
make -f amp.mk SYMBOLS
Symbols loaded.
make -f amp.mk OBJDUMP
Address out of range!
```

Figure 23: Displaying the result in the memory tab.


FPGAcademy Monitor Program
File Edit Actions Windows Help
Project Files
Part2
part2.s
Disassembly
Go to instruction
Address (hex) or symbol names:
Go
Editor Disassembly Memory Breakpoints
Registers
Terminal
Input
Info & Errors GDB Server GDB Client Debug

```
                    lw      t4, (t2)     # get the next number
0x00000020  lw        t4 ,  0(t2)
                    bge     t3, t4, loop # compare to largest found
0x00000024  bge       t3 ,  t4 , 14  <loop>
                    mv      t3, t4       # remember new largest
0x00000028  mv        t3 ,  t4
                    j       loop
0x0000002C  j         14  <loop>

            done:   sw      t3, (t0)     # store result
            done:
0x00000030  sw        t3 ,  0(t0)


            stop:   j       stop         # wait here
            stop:
```

Register	Value
pc	0x00000034
x0 (zero)	0x00000000
x1 (ra)	0x00000000
x2 (sp)	0x00000000
x3 (gp)	0x00000000
x4 (tp)	0x00000000
x5 (t0)	0x00000038
x6 (t1)	0x00000000
x7 (t2)	0x00000058
x8 (s0/fp)	0x00001040
x9 (s1)	0x00000007
x10 (a0)	0x00000000
x11 (a1)	0x00000000
x12 (a2)	0x00000000
x13 (a3)	0x00000000

```
------------------------------------------------
Altera Nios2 Command Shell

Version 23.1, Build 991
------------------------------------------------
nios2-terminal: connected to hardware target using JTAG UART on cable
```

```
pc          0x34      0x34 <stop>
(gdb)
break *0x0000002C
Breakpoint 1 at 0x2c: file part2.s, line 14.
(gdb)
enable 1
(gdb)
```

Figure 24: Setting a breakpoint.