Computer Architecture - Lab Assignment 1

Nios II/e instruction set architecture and programming

This is an introductory exercise that involves Altera's Nios II processor and its assembly language. It uses a simple computer system, called the DE0-Nano Basic Computer, which includes the Nios II processor. The system is implemented as a circuit that is downloaded into the FPGA device on the Altera DE0-Nano board. This exercise illustrates how programs written in the Nios II assembly language can be executed on the DE0-Nano board (see Figure 1). We will use the Altera Monitor Program software to compile, load and run the application programs.

Figure 1: DE0-Nano board.

To prepare for this exercise you have to know the Nios II processor architecture and its assembly language; read the tutorial Introduction to the Altera Nios II Soft Processor. You also have to become familiar with the monitor program; read the tutorial Altera Monitor Program. Both tutorials are available on Altera's University Program web site. The monitor tutorial is also included in the Altera Monitor Program package – it can be accessed by selecting Help > Tutorial in the monitor window.

Part I

In this part we will use the Altera Monitor Program to download the DE0-Nano Basic Computer circuit into the FPGA device and execute a sample program.

Perform the following:

1. Turn on the power to the Altera DE0-Nano board.

2. Open the Altera Monitor Program, which leads to the window in Figure 2.

   To run an application program it is necessary to create a new project. Select File > New Project to reach the window in Figure 3. Give the project a name and indicate the directory for the project; we
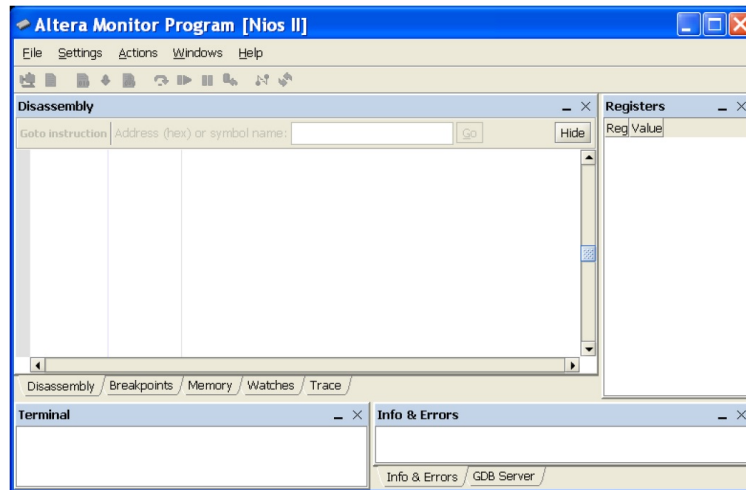
Figure 2: The Altera Monitor Program Window.

have chosen the project name part1 in the directory lab1_part1, as indicated in the figure. Click Next, to get the window in Figure 4.

3. Now, you can select your own custom system (if you have one) or a predesigned (by Altera) system. Choose the DE0-Nano Basic Computer and click Next. The display in the window will now show where files that implement the chosen system are located. This is for information purpose only; if you wanted to use a system that you designed by using Altera's Quartus II software, you would have to provide such files. Click Next.

4. In the window in Figure 5 you can specify the type of application programs that you wish to run. They can be written in either the Nios II assembly language or the C programming language. Specify that an assembly language program will be used. The Altera 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 6 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 files that are used by 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 7 indicates some system parameters. Note that the USB-Blaster cable is selected to provide the connection between the DE0-Nano board and the host computer. Click Next.

7. The window in Figure 8 displays the preselected components of the DE0-Nano Basic Computer. Observe that the SDRAM is selected as the memory device to be used. The start offset is 0x400 (hex 400), which means that the application program will be loaded in the memory locations that begin at address 400. Since this choice was made by the designer of the sample program, you cannot change the selection in Figure 8. Click Finish to complete the specification of the new project.

8. Since you specified a new project, a pop-up box will appear asking you if you want to download the system associated with this project onto the DE0-Nano board. Make sure that the power to the DE0-Nano board is turned on and click Yes. 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. A pop-up box will appear informing you that the circuit has been successfully downloaded - click OK. If the circuit is not successfully downloaded, make sure that the USB connection, through which the USB-Blaster communicates, is established and recognized by the host computer. (If there is a problem, a possible remedy may be to unplug the USB cable and then plug it back in.)

9. Having downloaded the DE0-Nano Basic Computer into the FPGA chip on the DE0-Nano board, we can now load and run programs on this computer. In the main monitor window, shown in Figure 9, select Actions > Compile & Load to load the selected sample program into the FPGA chip. Figure 10 shows the monitor window after the sample program has been loaded.
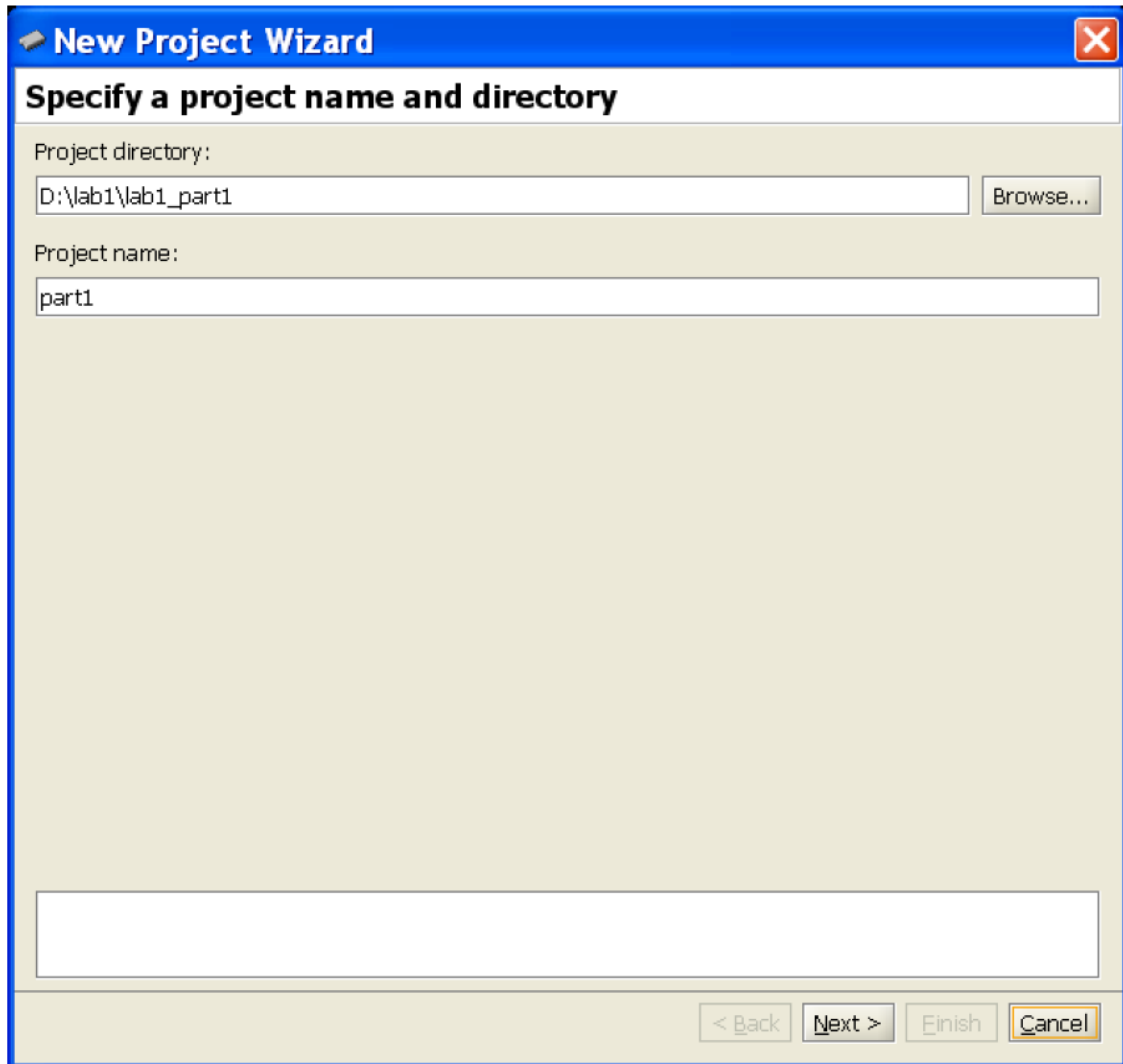
Figure 3: Specify the directory and the name of the project.

10. Run the program by selecting Actions > Continue or by clicking on the toolbar icon ▶, and observe the test displayed on the LEDs and 7-segment displays. This test provides an indication that the DE0-Nano board is functioning properly.

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

Note: Online students can not see the LEDs; click on the toolbar icon ⏸ to stop the execution and observe which registers contents are in red color in the "Registers" window.

Questions: Why some registers contents are in red color and the other registers contents are in black color?, which is the computing function of the registers which contents are in red color?

Part II

Now, we will explore some features of the Altera Monitor Program by using a simple application program written in the Nios II assembly language. Consider the program in Figure 11, which finds the largest number in a list of 32-bit integers that is stored in the memory. This program is available in the file lab1_part2.s.

Note that some sample data is included in this program. The list starts at hex address 500, as specified by the .org assembler directive. The first word (4 bytes) is reserved for storing the result, which will be the
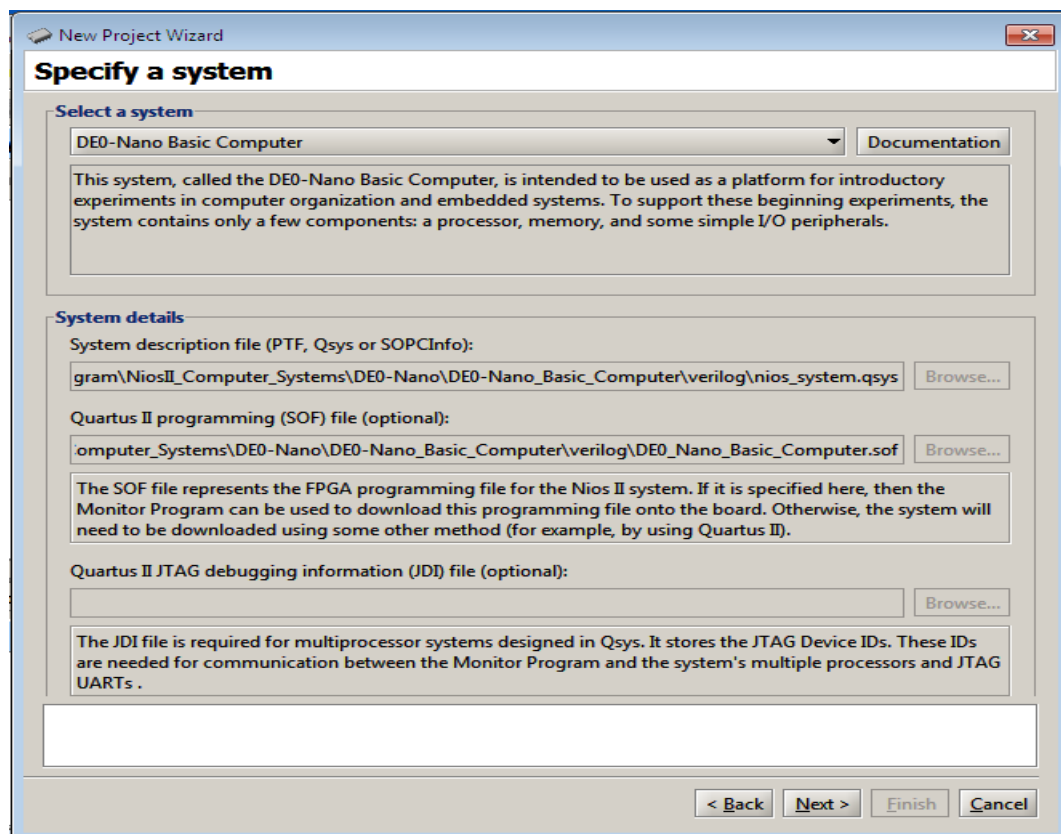
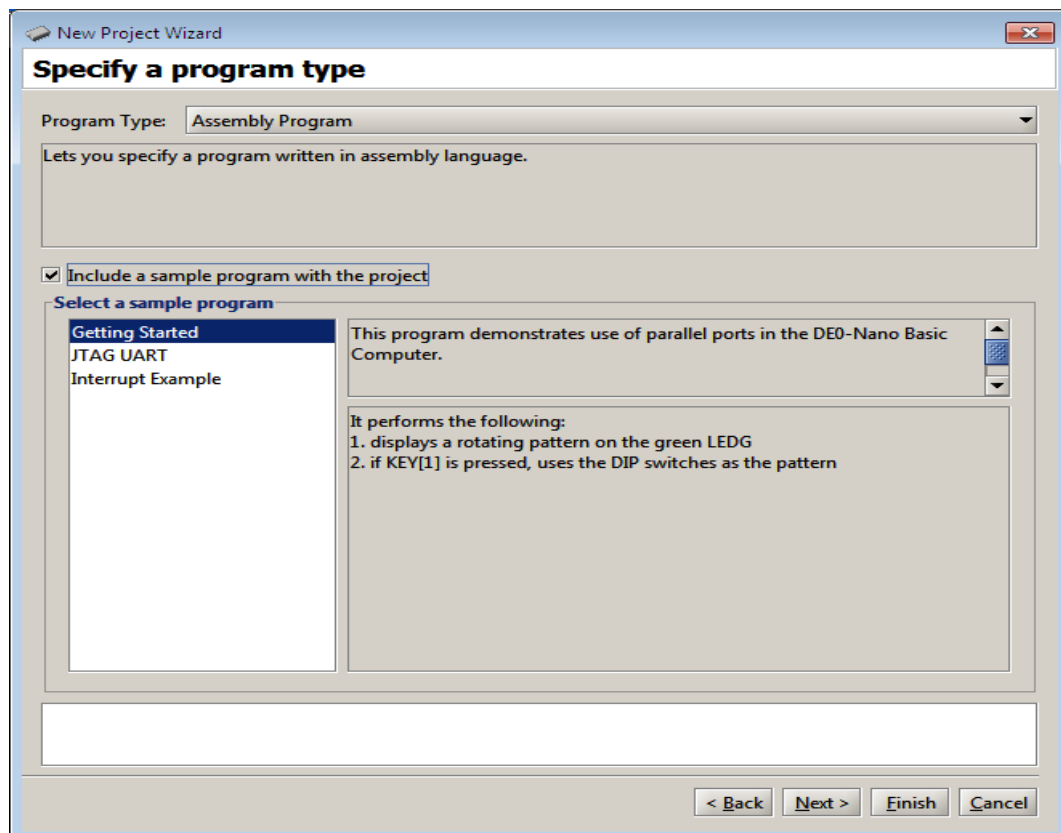Figure 4: Specification of the system.
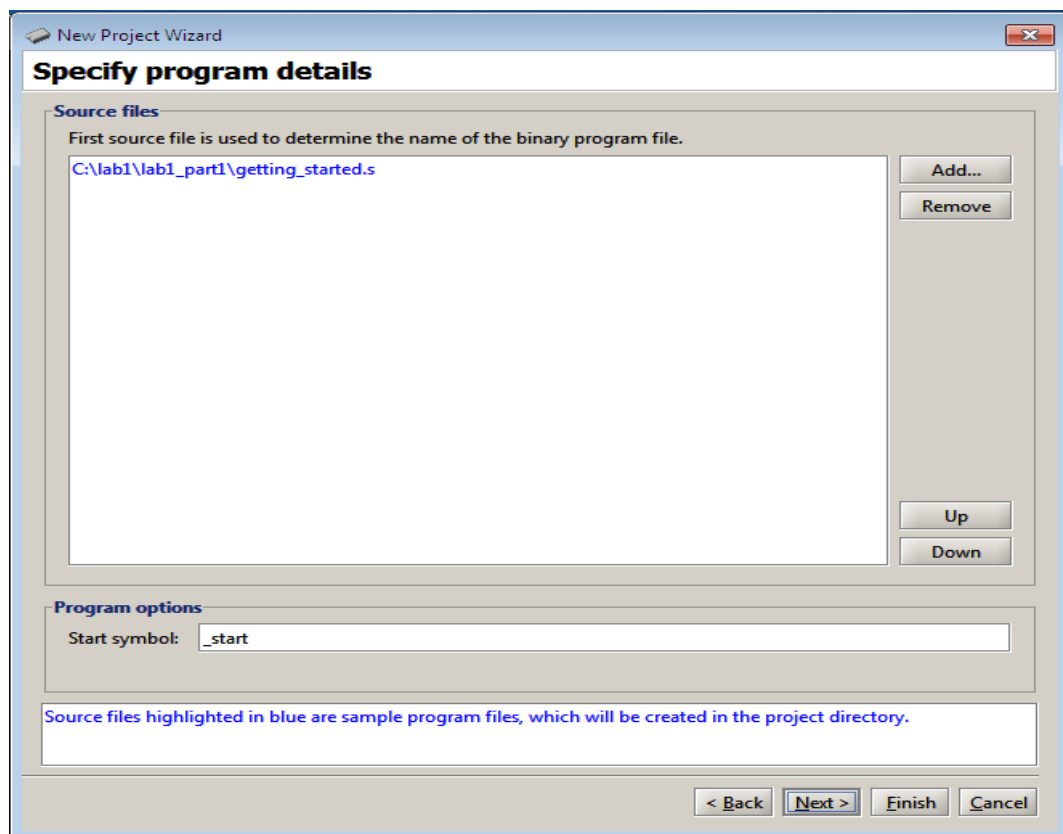

Figure 5: Selection of an application program.

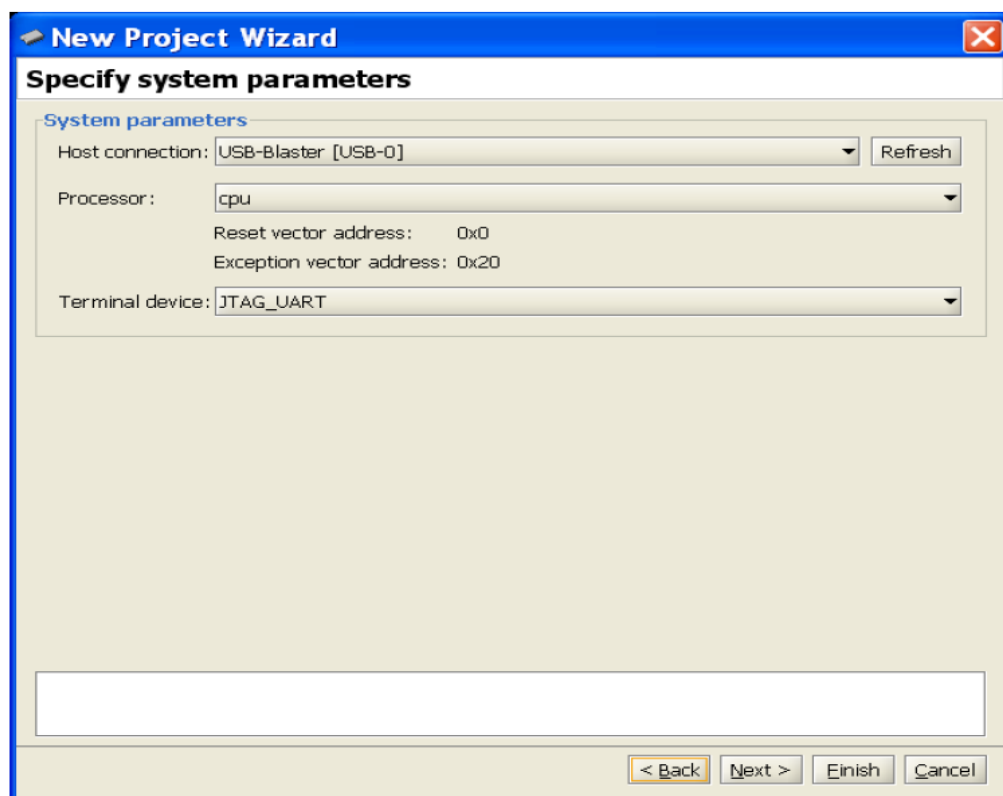Figure 6: Source files used by the application program.

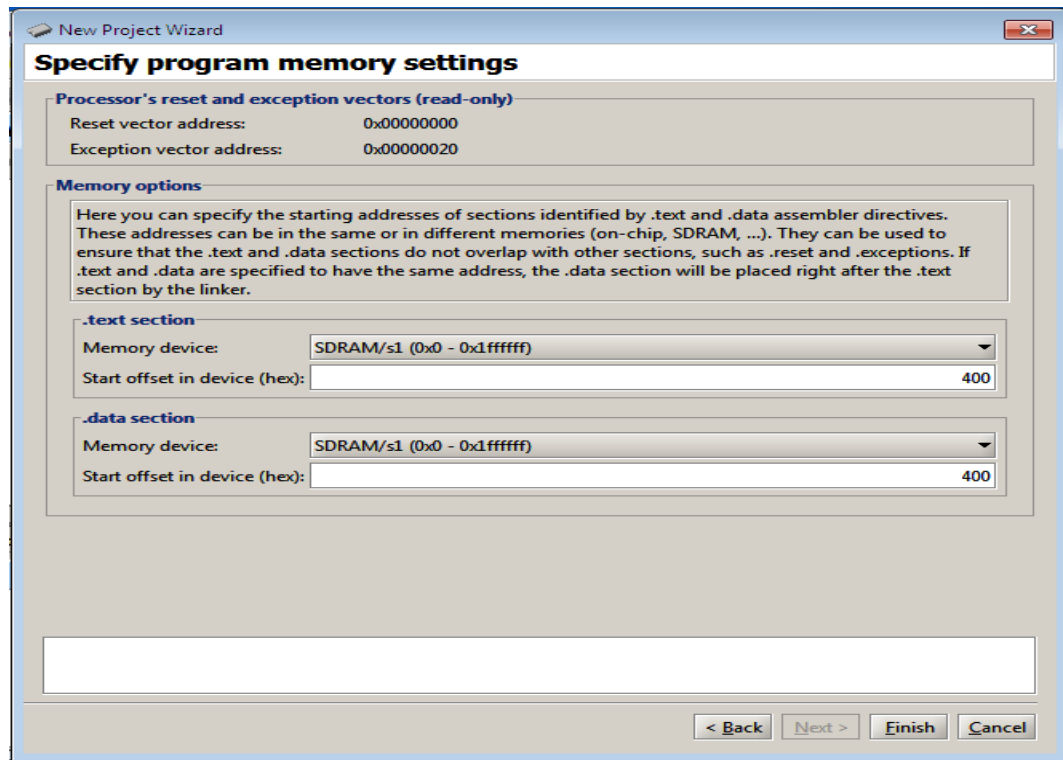Figure 7: Specify the system parameters.

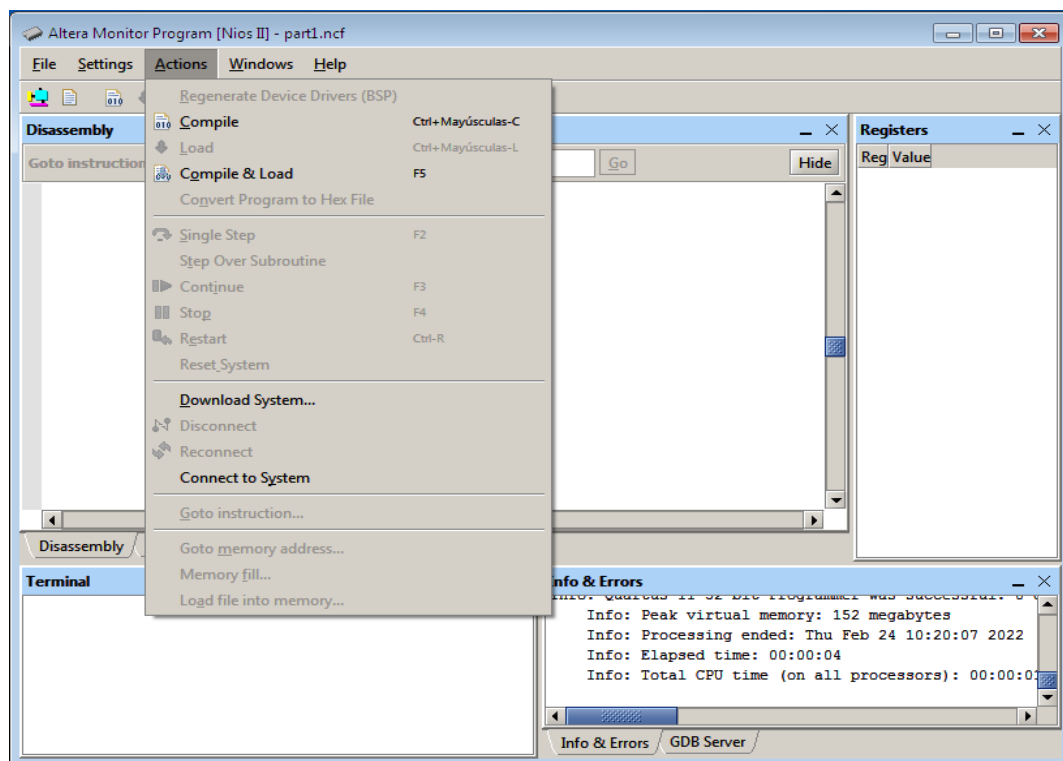Figure 8: Specify the program memory settings..

Figure 9: Specify an action in the monitor window.

largest number found. The next word specifies the number of entries in the list. The words that follow give the the actual numbers in the list.

Make sure that you understand the program in Figure 11 and the meaning of each instruction in it. Note the extensive use of comments in the program. You should always include meaningful comments in programs that you will write!
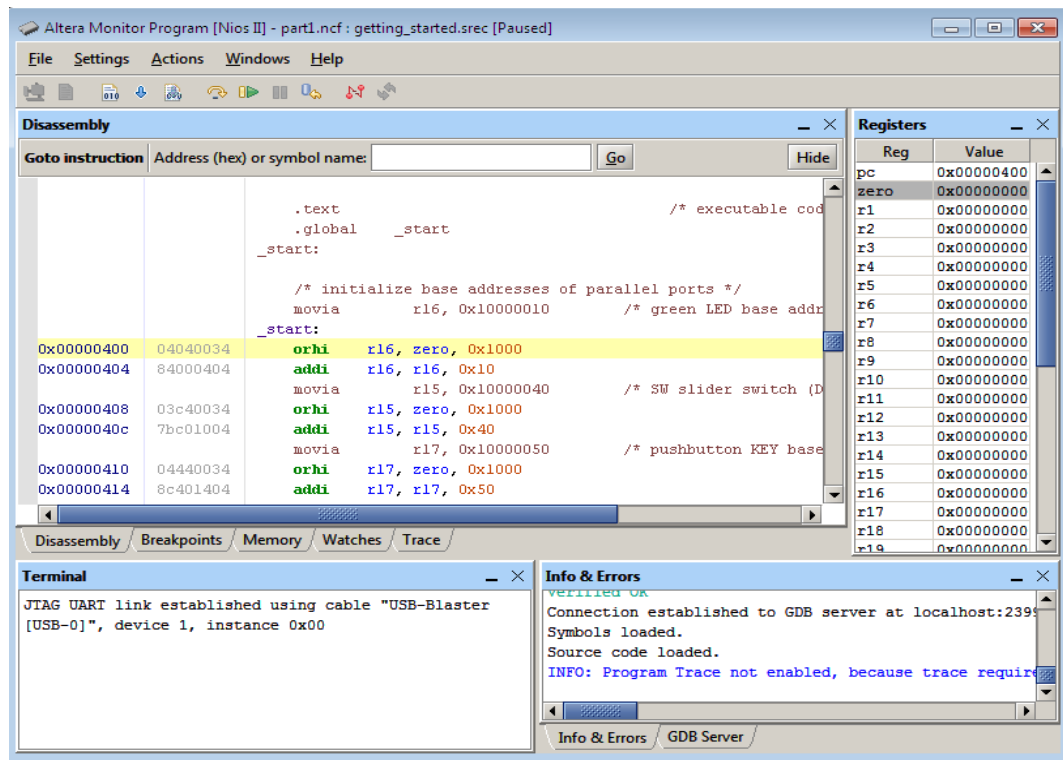
Figure 10: The monitor window showing the loaded sample program. Instruction set architecture registers window is on the right side.

Perform the following:

1. Create a new directory; we have chosen the directory name lab1_part2. Copy the file lab1_part2.s into this directory.

2. Use the Altera Monitor Program to create a new project in this directory; we have chosen the project name part2. When you reach the window in Figure 5 choose Assembly Program but do not select a sample program, as shown in Figure 12. Click Next.

3. Upon reaching the window in Figure 6, you have to specify your program. Click Add and in the pop-up box that appears indicate the desired file name, lab1_part2.s, and its location. This should lead to the image in Figure 13. Click Next to get the window in Figure 7. Again click Next to get to the window in Figure 8. Make sure that the SDRAM is selected as the memory device. Note that the Start offset in device will be 0, because the program in Figure 11 does not indicate that it should loaded at a location that is different from the default location 0. Click Finish.

4. Compile and load the program.

5. The Monitor Program will display the disassembled view of the code loaded in the memory, as indicated in Figure 14. Note that the pseudoinstruction movia in the original program has been replaced with two machine instructions, orhi and addi, which load the 32-bit address LIST into register r4 in two 16-bit parts (because an immediate operand value is restricted to 16 bits). Examine the disassembled code to see the difference in comparison with the original source program. Make sure that you understand the meaning of each instruction. Observe also that your program was loaded into memory locations with the starting address zero. These addresses correspond to the SDRAM memory, which was selected when specifying the system parameters. The DE0-Nano Basic Computer has two more memories which are the on-chip memory (i.e. the memory on the FPGA chip) and the SRAM chip on the DE0-Nano board. See the document Basic Computer System for Altera DE0-Nano Board for full information. This document can be accessed by clicking on the Documentation button in Figure 4.

6. Run the program. When the program is running, you will not be able to see any changes (such as the contents of registers or memory locations) in the monitor windows, because the monitor program cannot communicate with the processor system on the DE0-Nano board. But, if you stop the program

the present state of these components will be displayed. Do so and observe that the program has stopped executing at the last Branch instruction which is loaded in the memory location 0x34. Note that the largest number found in the sample list is 8 as indicated by the contents of register r7. This value is also stored in the memory location 0x500, which can be seen by opening the Memory tab of the monitor window (in Figure 14).

7. Return to the beginning of the program by clicking on the icon ![icon]. Now, single step through the program by clicking on the icon ![icon]. Watch how the instructions change the data in the processor's registers.

8. Set the Program Counter to 0. Note that this action has the same effect as clicking on the restart icon ![icon].

9. This time add a breakpoint at address 0x28 (by clicking on the gray bar to the left of this address), so that the program will automatically stop executing whenever the Branch instruction at this location is about to be executed. Run the program and observe the contents of register r7 each time this breakpoint is reached.

10. Remove the breakpoint (by clicking on it). Then, set the Program Counter to 0x8, which will bypass the first two instructions which load the address LIST into register r4. Also, set the value in register r4 to 0x504. Run the program by clicking on the icon ![icon]. What will be the result of this execution?

Part III

Instructions and data are represented as patterns of 1s and 0s. In this part, we will examine how instructions are formed. We will do this by replacing the instruction bge r7, r8, LOOP in the program in Figure 11 with the instruction blt r7, r8, LOOP. However, instead of replacing this instruction in the source program and then recompiling and loading the altered program, we will load the original program and then make the desired change directly in the program that is already loaded in the memory. To do this it is necessary to derive the machine-code representation of the instruction in question.

Perform the following:

1. Derive the machine code representation of the instruction `blt r7, r8, LOOP`. In the Nios II Processor Reference Handbook, which is available on Altera's website, we can find that the `blt` instruction has the format shown in Figure 15. Use registers `r7` and `r8` as registers A and B, respectively, and determine the branch offset needed to branch to the instruction at location LOOP.

2. Reload your original program (by selecting Actions > Load). Then, execute the program once, stopping at the end.

3. Use the Altera Monitor Program's memory-fill functionality to place the derived `blt` instruction into memory location `0x24`. We should note that you will not see the updated instruction in the disassembly view of the Monitor Program. Set the Program Counter to `0x0` and run the program. What is the result? What are the values in register `r7` and memory location `0x500`?

Part IV

In this part, you are required to write a Nios II assembly language program that generates the first n numbers of the Fibonacci series. In this series, the first two numbers are 0 and 1, and each subsequent number is generated by adding the preceding two numbers. For example, for n = 8, the series is

$$0, 1, 1, 2, 3, 5, 8, 13$$

Your programs should store the numbers in successive memory word locations starting at 0x1000. Place a test value n in location 0xffc.

Perform the following:

1. Create a new directory, lab1 part4.

2. Write an assembly language program that computes the desired Fibonacci series, and place the file in the directory lab1 part4.

3. Then, use the Monitor Program to create a new project, part4, and specify that your program should be run on the DE0-Nano Basic Computer.

4. Run your program.

5. Examine the memory locations starting at 0x1000 to verify that your program is correct.

Preparation

Your preparation should include the following:

1. Read the tutorials Introduction to the Altera Nios II Soft Processor and Altera Monitor Program.

2. Derive the machine code representation for the instruction blt r7, r8, LOOP.

3. Write the assembly language program for Part IV.

```
/* Program that finds the largest number in a list of integers */
.equ  LIST, 0x500              /* Starting address of the list */

.global _start
_start:
    movia   r4, LIST           * r4 points to the start of the list */
    ldw     r5, 4(r4)          * r5 is a counter, initialize it with n */
    addi    r6, r4, 8          /* r6 points to the first number */
    ldw     r7, (r6)           /* r7 holds the largest number found so far */
LOOP:
    subi    r5, r5, 1          /* Decrement the counter */
    beq     r5, r0, DONE       /* Finished if r5 is equal to 0 */
    addi    r6, r6, 4          /* Increment the list pointer */
    ldw     r8, (r6)           /* Get the next number */
    bge     r7, r8, LOOP       /* Check if larger number found */
    add     r7, r8, r0         /* Update the largest number found */
    br      LOOP
DONE:
    stw     r7, (r4)           /* Store the largest number into RESULT */
STOP:
    br      STOP               /* Remain here if done */

.org    0x500
RESULT:
.skip   4                      /* Space for the largest number found */
N:
.word 7                        /* Number of entries in the list */
NUMBERS:
.word 4, 5, 3, 6, 1, 8, 2      /* Numbers in the list */
.end
```

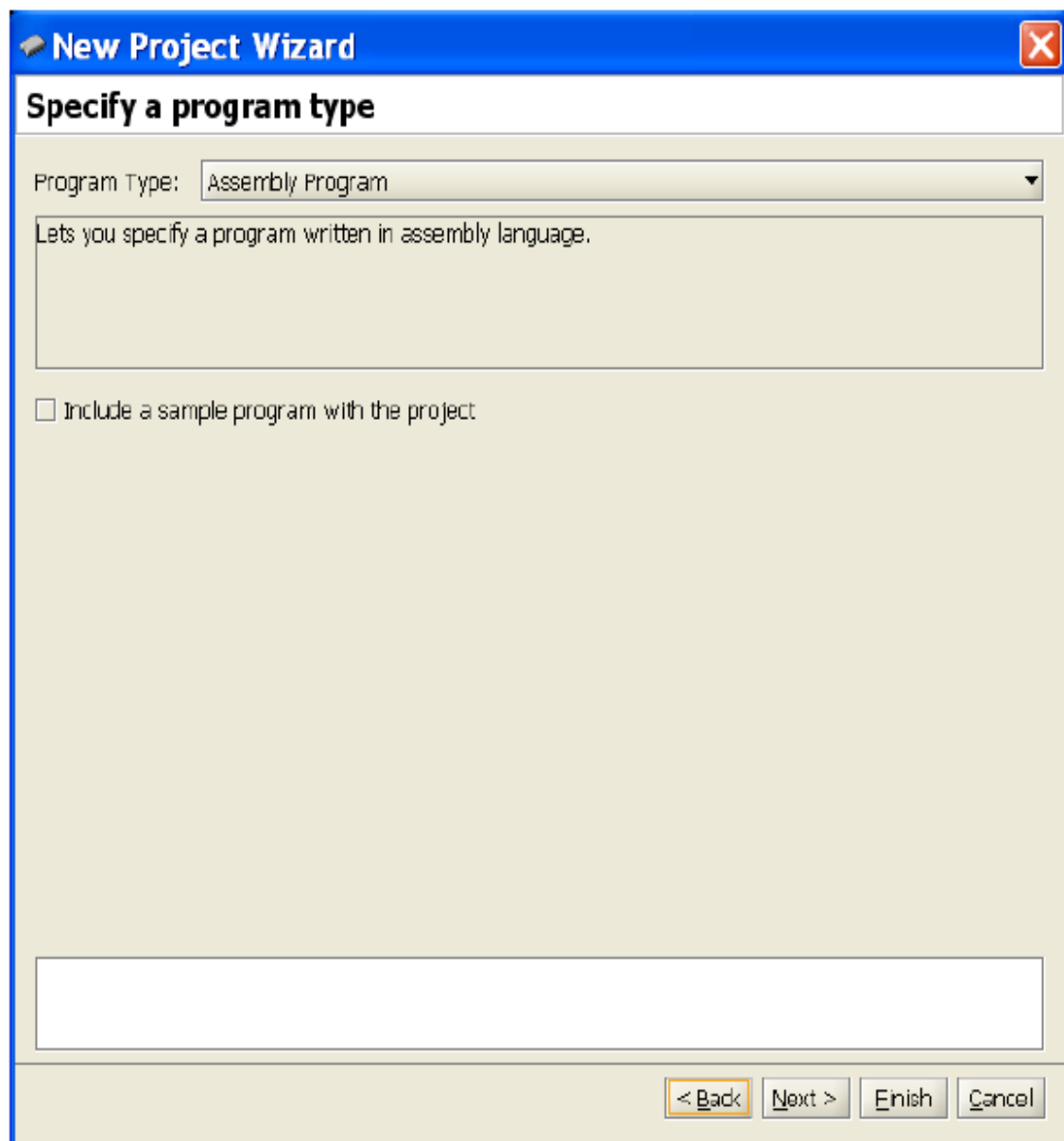Figure 11: Assembly-language program that finds the largest number.

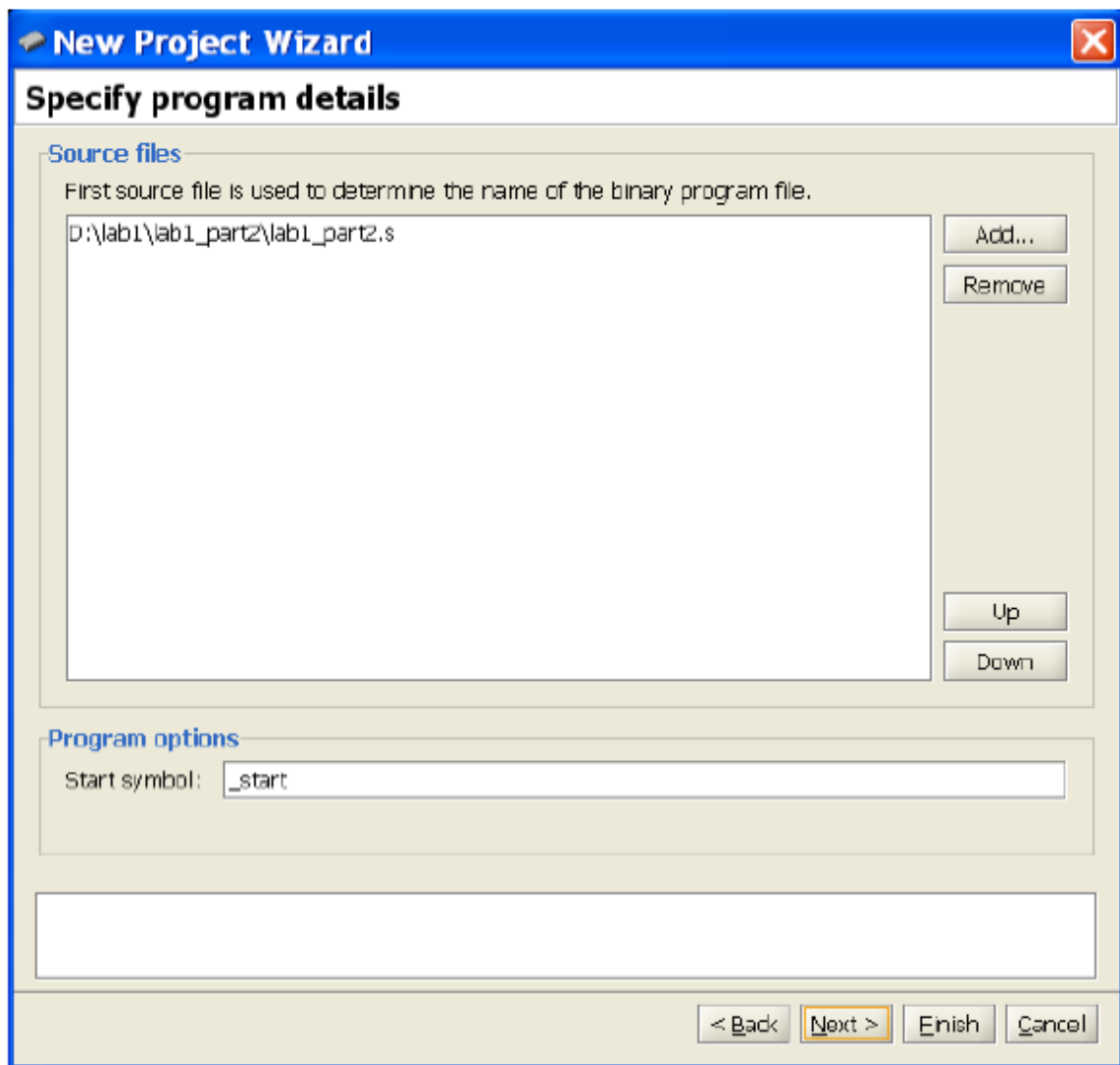Figure 12: Select the assembly language program.
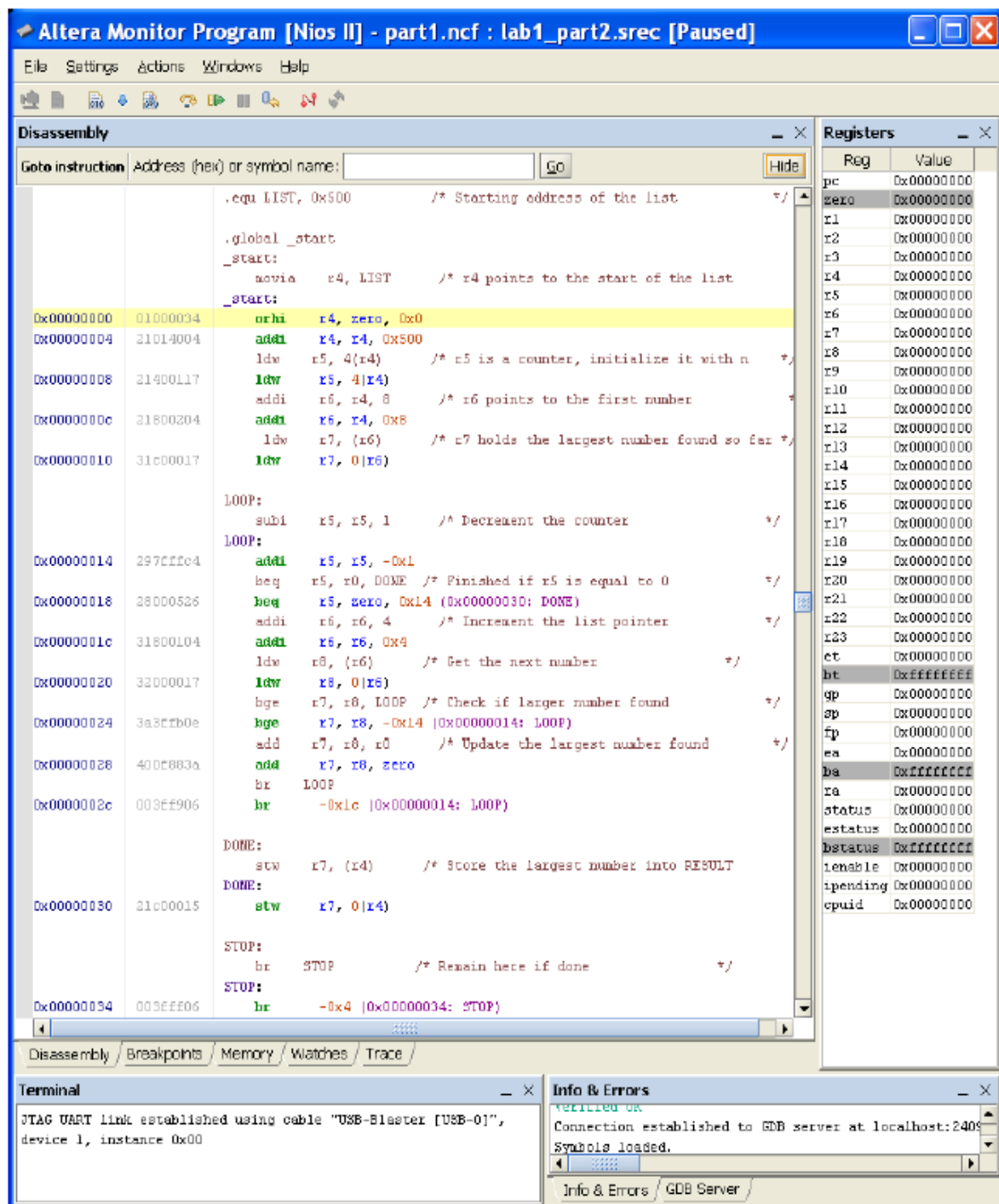
Figure 13: Select your source program.
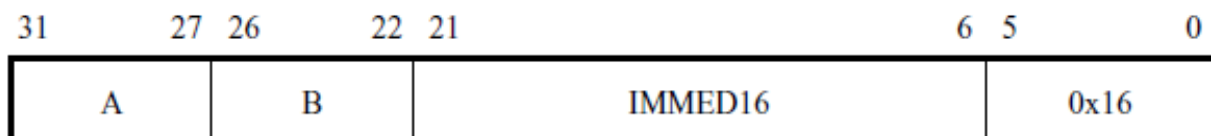
Figure 14: The disassembled view of the program in Figure 10.

Figure 15: Format for the blt instruction.