EEE 3530
Spring 2020

Project #2: Single-Cycle RISC-V Implementation

Objectives:

To learn the basic structure and operation of the single-cycle RISC-V processor

Due Dates:

Softcopy: June 1st, 6 PM (To YSCEC)

** Late submission within 24 hours automatically deducts 20 percents of the point.*

Things to Submit:

1. Cover page with your name and Student ID
2. Schematic diagrams printed from Quartus II (your design)
3. Simulation results and analysis (the wave form)
4. Discussion

**** In report paper, working out must be shown in English.***

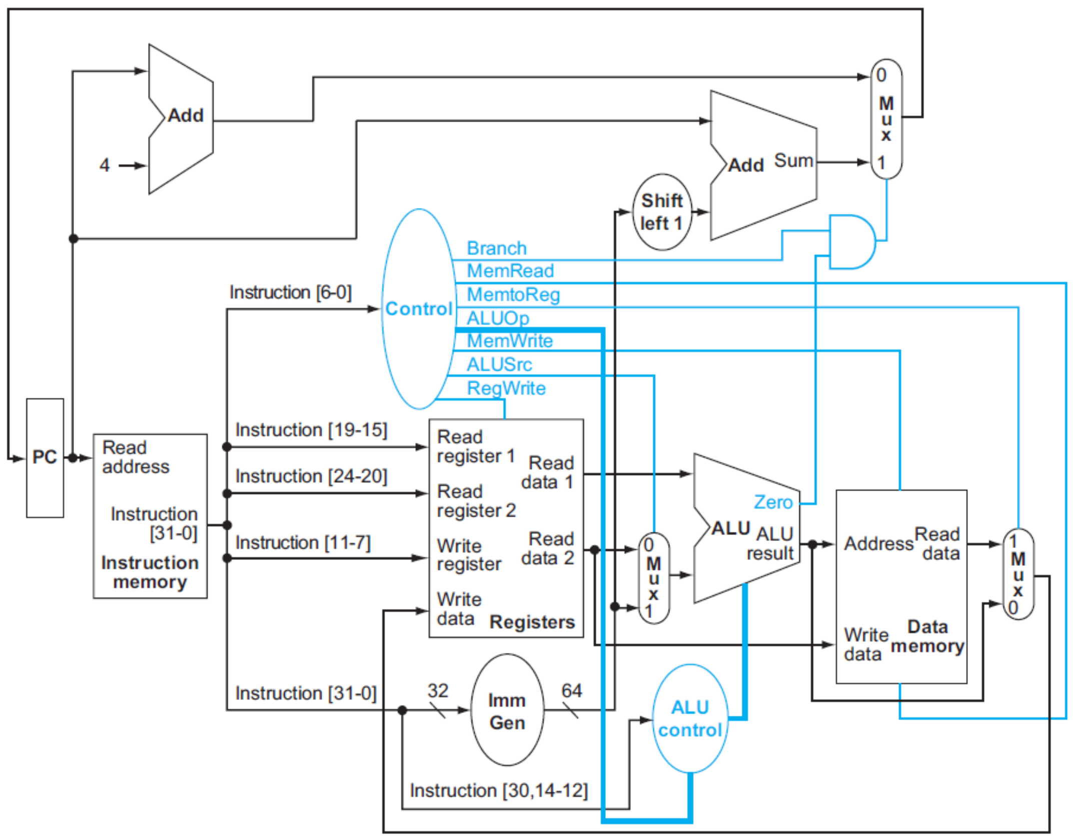
Professor: W. W. Ro

T. A.: Jonghyun Lee, Jeonghun Choi

Feel free to contact us whenever you meet some difficulties or errors in the manual
(but, spend some time and efforts to figure them out for yourself first)

64-Bit Single-Cycle RISC-V Microprocessor Design

The RISC-V processor to be implemented for this lab is identical to the single-cycle RISC-V processor in Chapter 4. The schematic for the single-cycle implementation is also included in the following diagram.

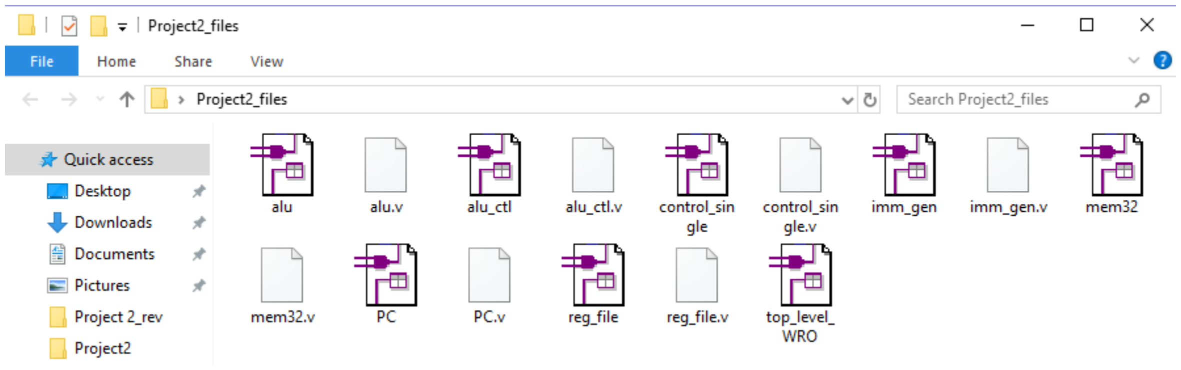


The requirement for the project is to implement and to complete the datapath and control logic for single-cycle implementation; major tasks include completion of the design and simulation of the seven instructions: `add, sub, and, or, sd, ld, beq`. Operation of each instruction follows the descriptions given in the textbook.

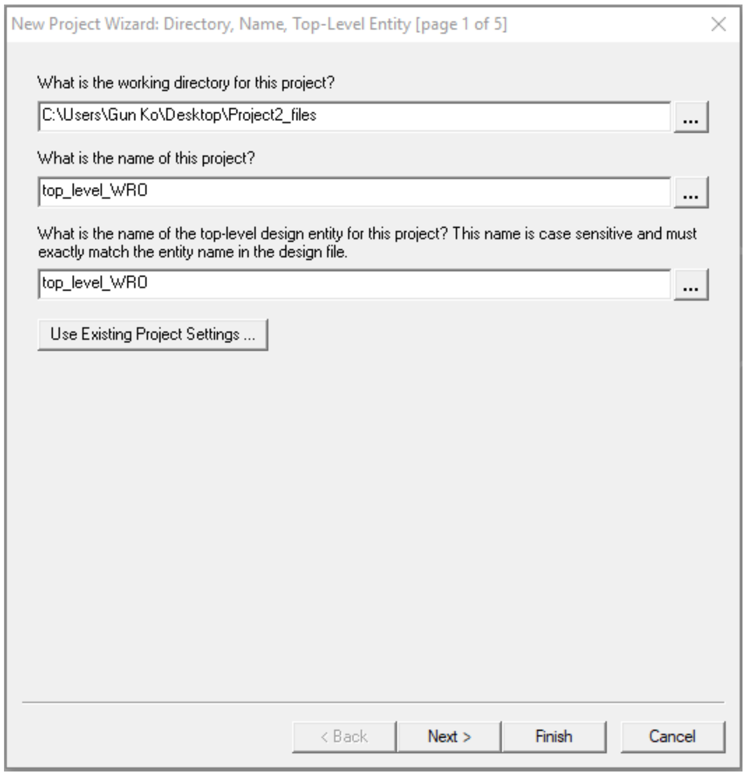
The design is implemented hierarchically with the several block structures. The operation and internal structure of each block are identical to the descriptions in the textbook.

File download and setup procedure

1. Copy the following 15 files (You can download the files from the class web site) to your working directory for the project.



2. Change the file "top_level_WRO.bdf" to "top_level_*your intial*.bdf".
3. Start Quartus II and go to "File" → "New Project Wizard".
4. In the following window, find your working directory that you saved the 15 files in. Then, give the project name and the top-level design name as you named in step 2 (actually, <u>**you can find the file and click that**</u>).


New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

C:\Users\Gun Ko\Desktop\Project2_files

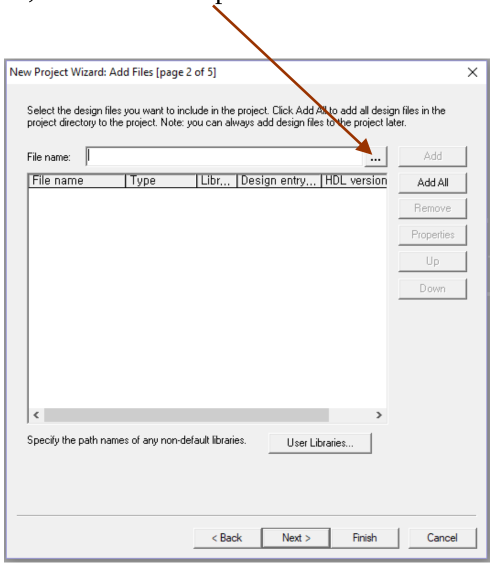
What is the name of this project?

top_level_WRO

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

top_level_WRO

5. Then, in the following window, click here this point. You need to locate your working directory.

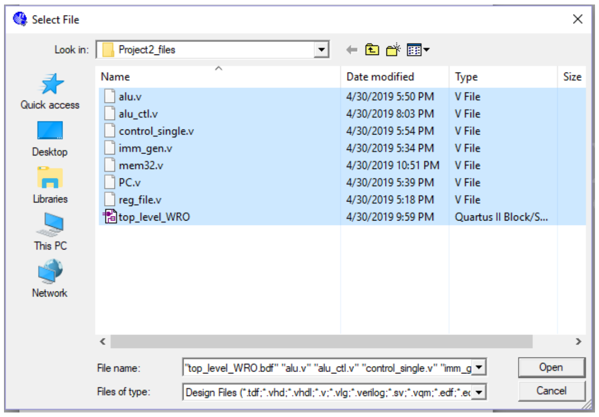


Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project. Note: you can always add design files to the project later.

File name	Type	Libr...	Design entry...	HDL version

Specify the path names of any non-default libraries.

6. In the following window, select the nine files.

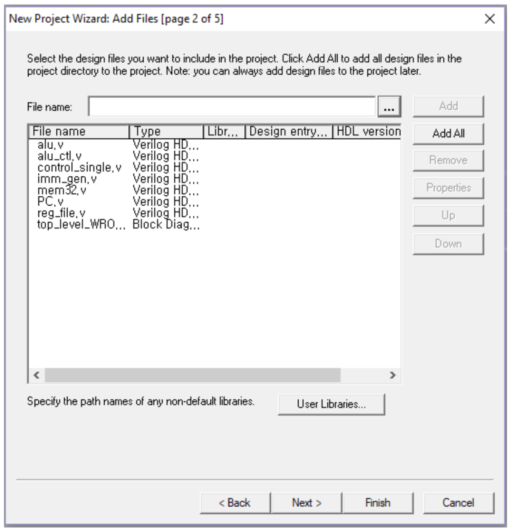


Name	Date modified	Type	Size
alu.v	4/30/2019 5:50 PM	V File	
alu_ctl.v	4/30/2019 8:03 PM	V File	
control_single.v	4/30/2019 5:54 PM	V File	
imm_gen.v	4/30/2019 5:34 PM	V File	
mem32.v	4/30/2019 10:51 PM	V File	
PC.v	4/30/2019 5:39 PM	V File	
reg_file.v	4/30/2019 5:18 PM	V File	
top_level_WRO	4/30/2019 9:59 PM	Quartus II Block/S...	

File name: "top_level_WRO.bdf" "alu.v" "alu_ctl.v" "control_single.v" "imm_g

Files of type: Design Files (*.tdf;*.vhd;*.vhdl;*.v;*.vlg;*.verilog;*.sv;*.vqm;*.edf;*.ec

7. Then, you will see the following box pop up. Just click “Next”.

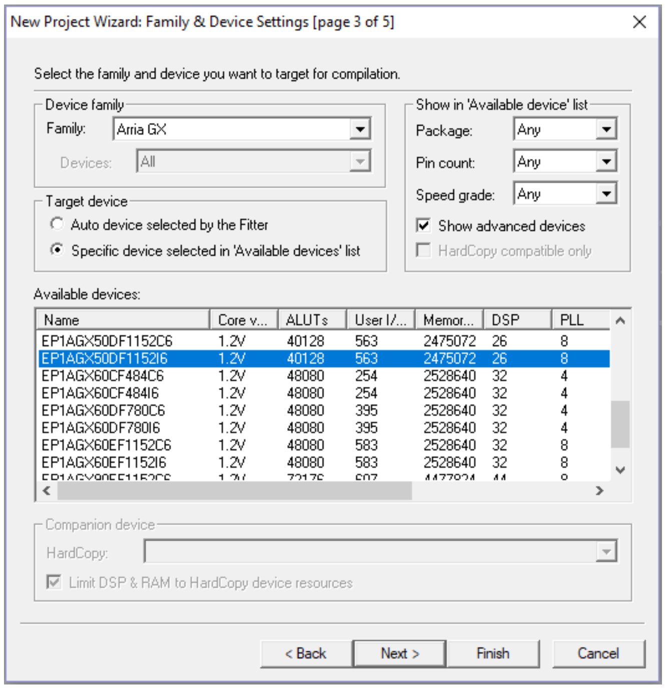


Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project. Note: you can always add design files to the project later.

File name	Type	Libr...	Design entry...	HDL version
alu.v	Verilog HD...			
alu_ctl.v	Verilog HD...			
control_single.v	Verilog HD...			
imm_gen.v	Verilog HD...			
mem32.v	Verilog HD...			
PC.v	Verilog HD...			
reg_file.v	Verilog HD...			
top_level_WRO...	Block Diag...			

Specify the path names of any non-default libraries.

8. In the next pop-up window, select “Arria GX” and “EP1AGX50DF1152I6” for the device.



New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family
Family: Arria GX
Devices: All

Target device
( ) Auto device selected by the Fitter
(•) Specific device selected in 'Available devices' list

Show in 'Available device' list
Package: Any
Pin count: Any
Speed grade: Any
[✓] Show advanced devices
[ ] HardCopy compatible only

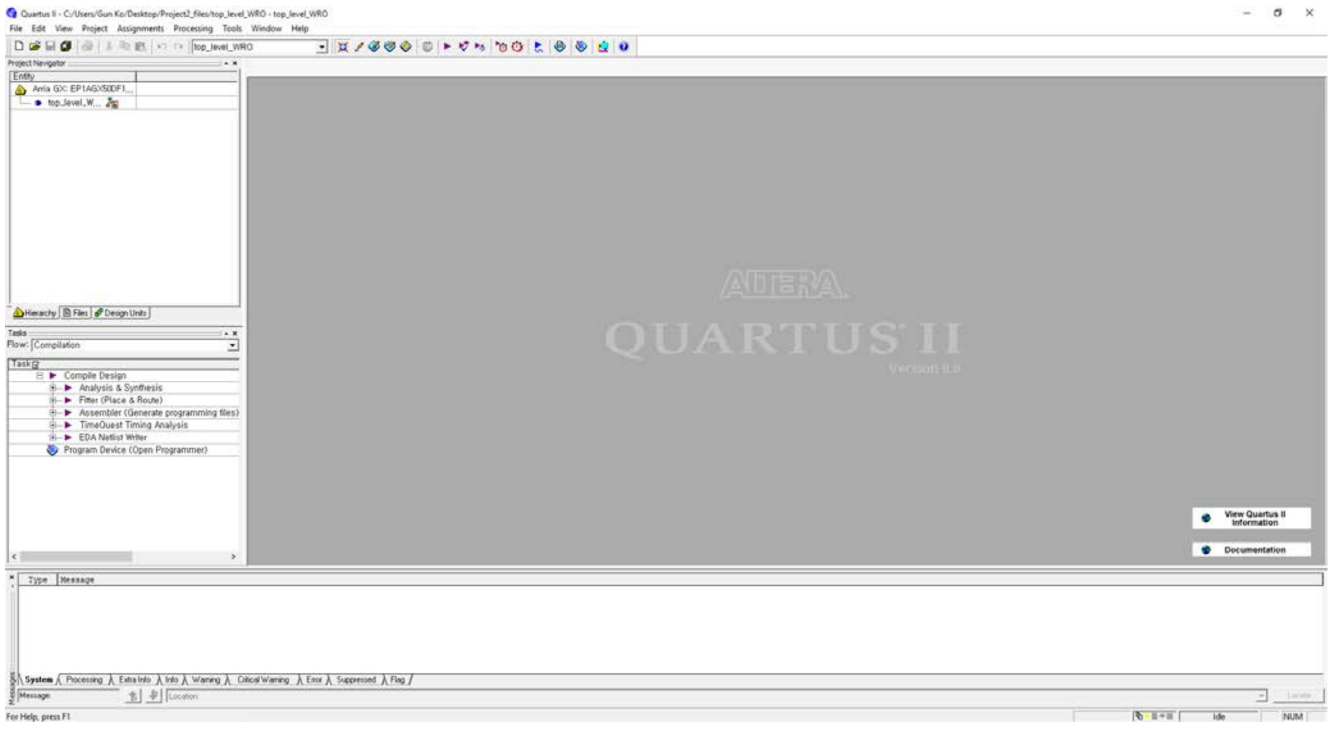
Available devices:

Name	Core v...	ALUTs	User I/...	Memor...	DSP	PLL
EP1AGX50DF1152C6	1.2V	40128	563	2475072	26	8
EP1AGX50DF1152I6	1.2V	40128	563	2475072	26	8
EP1AGX60CF484C6	1.2V	48080	254	2528640	32	4
EP1AGX60CF484I6	1.2V	48080	254	2528640	32	4
EP1AGX60DF780C6	1.2V	48080	395	2528640	32	4
EP1AGX60DF780I6	1.2V	48080	395	2528640	32	4
EP1AGX60EF1152C6	1.2V	48080	583	2528640	32	8
EP1AGX60EF1152I6	1.2V	48080	583	2528640	32	8

Companion device
HardCopy:
[✓] Limit DSP & RAM to HardCopy device resources

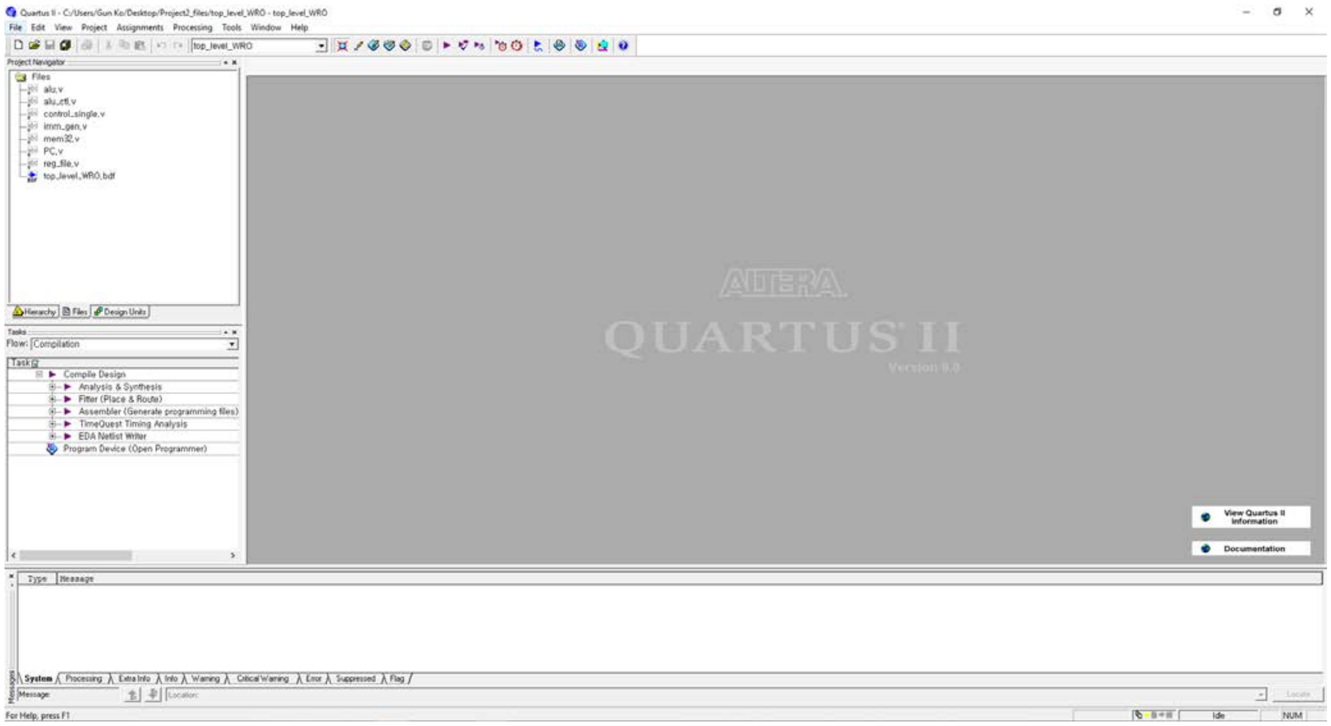
< Back | Next > | Finish | Cancel

9. Again, just select the “Next” buttons until you get to the following window.



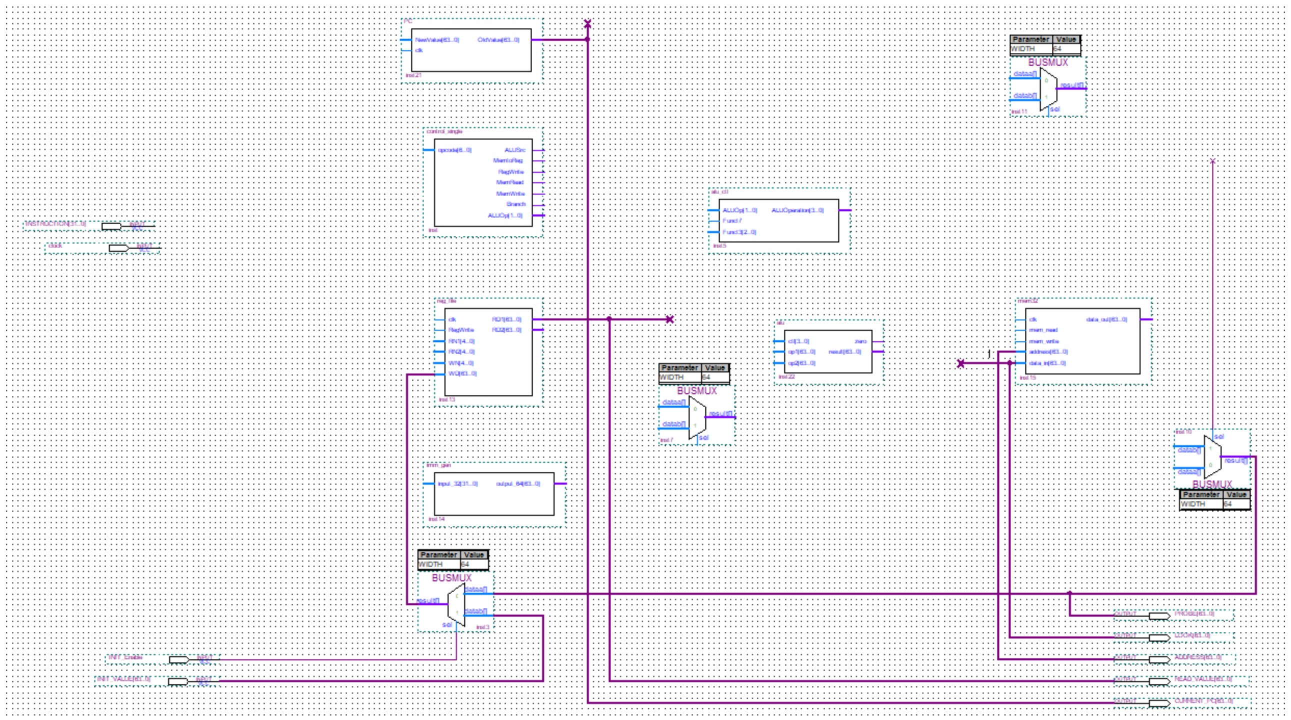
Quartus II - C:/Users/Gun Ko/Desktop/Project2_files/top_level_WRO - top_level_WRO

10. Now, click the "Files" and double click the "*top_level_your initial.bdf*" file.



11. Now the following diagram is an incomplete version of the single-cycle RISC-V design.

<u>(You should connect the disconnected parts)</u>



* The top-level file contains two special parts for the simulation purpose. Those two parts are not included in the original design of the textbook.

- The first part is in the bottom of the diagram with `“INIT_Enable”` and `“INIT_VALUE”` (also the BUSMUX at the bottom with those two signals.). This is designed to load initial values for the register file. When `“INIT_Enable”` is “1”, the `“INIT_VALUE”` provides the data to be written in the register file; also, the instruction should provide the register name for that initialization. You don’t have to implement this initialization circuit since it is already completed. You will see how to initialize the register file in the simulation part.

- The second part is 5 output buses: `“PROBE”`, `“LOOK”, “ADDRESS”, “CURRENT_PC”` and `“READ_VALUE”`. Those five signals are inserted in the original design to probe each point of the circuit. Our simulation can show run-time signals for those five points.

12. Double click the blocks and fill in the blanks in the source code.

    (1) `control_single.v`

    (2) `alu_ctl.v`

    (3) `alu.v`

    (4) `reg_file.v`

    (5) `sign_extend.v`

13. You need to include some other functional blocks to complete the design.

- The given top module design doesn’t have any blocks to implement branch instruction. Therefore, you need to add some functional blocks to complete the design. You can see “**Create Symbol Files for Current File**” by right-clicking on the file name in your project file list window after you complete and save the source code of the new module. When you save the source code, **be sure that the file name is the same as the module name**.

- A register to hold the PC value is given. You don’t need to use the PC value to fetch an instruction, but you still have to count the PC value and use it for branch instruction.

Simulation:

We provide a sequence of instructions to simulate. Also, we provide the list of initial values of the register file so that the given instruction sequence can be executed properly. You need to initialize the register file before you execute the given instructions. The initialization can be done by asserting INIT_Enable signal and INIT_VALUE. The address of the destination register can be given by the INSTRUCTION input. The initial values are given in the table below. Note that you don't need to initialize the PC.

Initial Values

Register No.	Initial Value
X1	10
X2	7
X3	3
PC	A0000000

The instruction sequence is given in the table below. The instructions are given in the assembly codes and you need to translate the instruction to the corresponding binary code for the simulation. Note that you need to make two of your own instructions. Use any kind of instructions and show those instructions work. You should **refer to the text book for OPCODE and FUNCT code**.

Instruction Sequence

Sequence	Instruction
1	`and  X4, X1, X3`
2	`sub  X6, X1, X3`
3	`add  X5, X3, X2`
4	`beq  X1, X6, 0x440`
5	`sd   X4, 24(X5)`
6	`add  X6, X1, X5`
7	`ld   X6, 24(X5)`
8	`or  X2, X4, X2`
9	`Your own instruction 1`
10	`Your own instruction 2`

Things to turn in:

(1) Submit a soft copy of your report. The report should include the **schematic design of your top module, source codes, handful of discussion and simulation results (including the waveform).**

(2) **Also, upload a compressed file of all the project files (whether modified or not, all of them) on YSCEC.** Please match the format *Name(Korean)_StudentID.zip*.

(3) Analysis of the simulation results must be included in the report. You should completely describe and analyze the results and it should be written **in English**.

(4) All output signals for each instruction should be shown.

- **PROBE** : ALU results or Memory output controlled by MemtoReg
- **LOOK** : data for memory input
- **ADDRESS** : memory address
- **READ VALUE** : RD1 (rs) from register file
- **CURRENT_PC** : output from PC 'module'