Submission notes

In this assignment you work on the final project of this course. This document covers the general description of the project and how to submit it to the dropbox. Deadline for the first part is *Monday night 11/28/2016.*

There would be an additional document explaining final project part2 that would explain how to implement your project on FPGA and showcase it in the Week 10 lab sessions. There are two showcase sessions will be hold on Wednesday (11/30/2016) 7-10 pm and Thursday (12/1/2016) 7-10 pm.

Your submission should include the following:

- All Block designs in System Verilog
- Processor capable of running 32-bit instructions
- A comprehensive report of the Design and testbench architecture. In the report, include the information about the programs you have run on the processor, the expected simulation results, synthesis completion and if you have found any bugs in your design. Report all of your assumptions in your report.
- Any additional file required for simulation and test of your processor.
- Optional: there would be another deadline and showcase for the FPGA implementation. You can submit the files related to the implementation in part2 but if you have them inside your zip file for part 1 it wouldn't affect your score.

IMPORTANT: only ONE submission per group is required.
Note1: Compress all your files in "zip" or "tar" format and then submit the compressed file.
Note2: The compressed file should include all the related files. For example, basic block design, sim, synthesis doc. Your report's pdf file be expected to be inside the zip file.
Note3: Remember to include the group ID or group name and the name and student ID of each group member in the report.
Note4: Assuming you have finished the simulation and synthesis of your project, you still will have couple days to work on your FPGA implementation till the showcase.

Final Project Part1: Design, Simulation and Synthesis (75 points)

The goal of this part of the project is to design a simple processor, we call it as 31L processor. The instruction format of this processor is as shown in figure1.Please refer to Lecture-7 slide for more information.

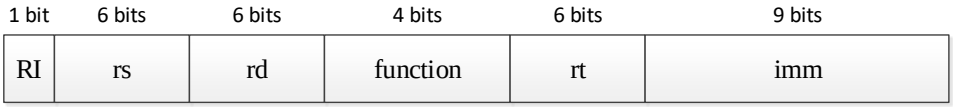


Figure 1. Instruction format

31L processor supports two types of instructions including R-type (RI='0') and I-type instructions (RI='1'). R-type is a register-based instruction and I-type is an immediate-based instruction. In R-type instructions the instruction is composed of three operands, two source registers, and one destination register. In immediate-based type, one of the sources is a 15 bit immediate value. Here is an example of R-type and I-type version of ADD instruction.

ADD RS, RD, RT // R_TYPE: rd ← rs + rt

ADDI RS, RD, IMM // I_TYPE: rd ← rs + immediate

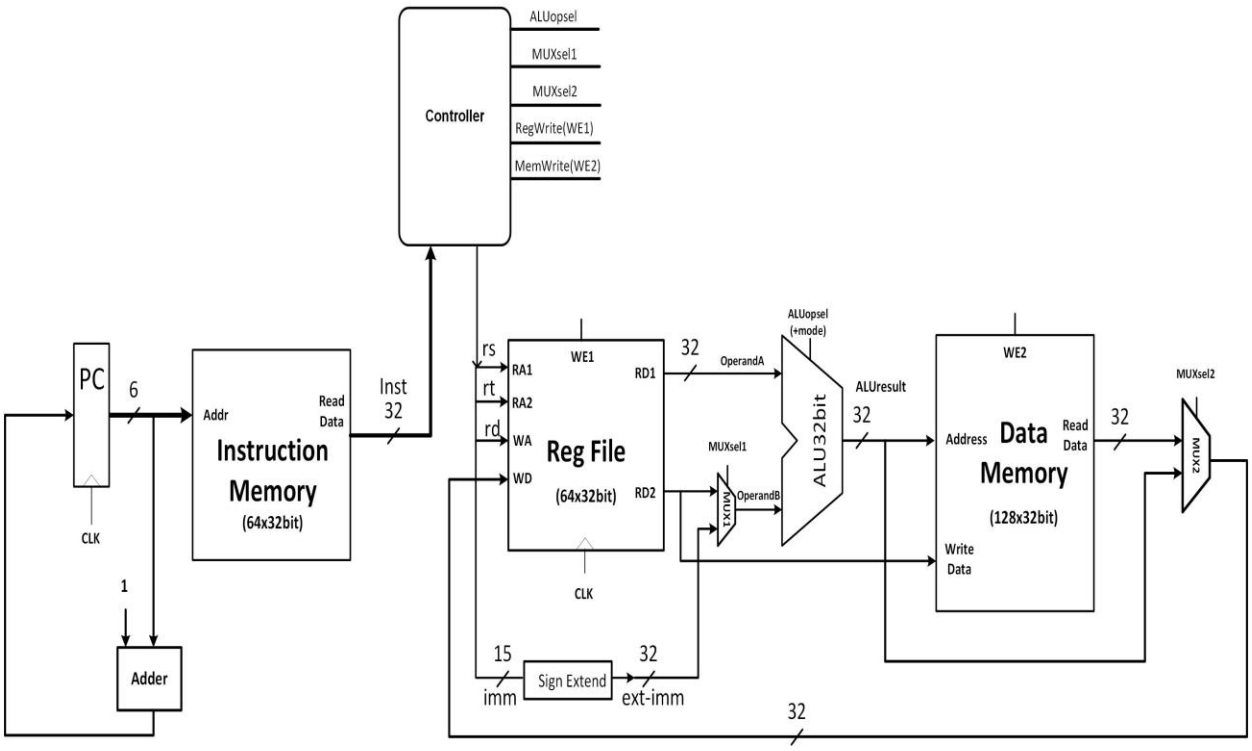
In the R-type instructions rs and rt in the instruction specify the address of the source registers while rd contains the address of the output register and the last 9 bits of instruction (imm) are don't care. On the other hand, in the I-type instructions only one source register is needed and the value of the second input

This simple processor has a simple instruction set including the following commands:

Instruction	Description	Function code	Comments
NOP	nothing	"1111"	
ADD/ADDI	rd ← rs + rt / rd ← rs + immediate	"0000"	
SUB/SUBI	rd ← rs - rt / rd ← rs - immediate	"0011"	
AND/ANDI	rd ← rs AND rt / rd ← rs AND immediate	"1000"	
OR/ORI	rd ← rs OR rt / rd ← rs OR immediate	"1001"	
NOT	rd ← NOT rs	"1011"	
XOR/XORI	rd ← rs XOR rt / rd ← rs XOR immediate	"1010"	
SLL/SLLI	rd ← shift left (rs) / rd ← shift left (rs)	"1101"	rt/ immediate value decides how many bits to shift
MOV/MOVI	rd ← rs / rd ← rs	"0010"	
LOAD	rd ← Data Memory(rs)	"0100"	
STORE	Data Memory(rs) ← rt	"0110"	

You are supposed to implement

- A controller/decoder which takes an instruction as input (from Instruction Memory) and then set the function types, register indexes, and all the other controlling signals in your design for choosing appropriate selector lines of the multiplexer.
- ALU component (Which was done in midterm project. If you had any bugs in your ALU, then fix them first). It has couple of inputs including two data bit lines and function type bit lines to implement the corresponding command. It also has 1 output data bit lines and 4 single bit lines including zero, sign, carry, and overflow which will be activated in case they are needed. Note: the ALU implemented in the midterm was 128-bit wide but it has to be changed to 32-bit to be compatible with other components in the processor. This change should be very simple as you have to change a single parameter in your ALU component.
- Register bank has four major inputs (three register index for address, and one data input) and two outputs. The register index inputs contain the address of rs, rt and rd registers, the outputs contain the values stored in memory cells of rs and rt. It also may have Write Enable (WE) an clk as inputs.
- Data Memory holds some of the data. This data comes from register file which means it is the result of some previous ALU computations. Load operation reads the value inside the cell with the address of rs of Data Memory and writes it back to Reg file. On the other hand, Store reads a value from register file and write it into Data Memory.
- Implement the interconnection among all ALU, Register file, memories and Controller/decoder components. You can also employ multiplexers if they needed.
- Remember all of this should happen in one cycle!



Notes:

- The instruction, at the power up, should be read from Instruction Memories first instruction.
- It reads the operand from register file or as immediate values.
- The result of operation is supposed to be stored inside register file (for most operations except load and store)
- For load and store operations the ALU result should be used as address for Data Memory. The ALU operation for while doing a load or store is just a simple move. It means rs becomes the address for data memory. During store operations rt becomes the write data for Data Memory. During load operation we will read one cell from Data Memory to pass to reg file.
- The 31L processor supports Data Memory with load and store operations. This data memory can be used to store large amount of information. In this project, we use a data memory that has 128 memory cell each 32 bits wide. It means that you only have to use first 7 bits of ALUresult to address cells in Data Memory.
- The immediate value has 15 bits while your whole design and registers are 32 bits, so you need a sign-extension component to modify your 15 bit immediate value into 32 bits. As a reminder a sign extension component increases the number of bits of a binary number while preserving the sign of number and its value. This is done by appending digits to the most significant location of the number.