CSCE 2214 Lab 09

Pre-Knowledge

In order to complete this lab you will need to understand the role of instruction memory and the program counter.

Objective

In this lab the student will leverage the already designed CPU and build upon a data memory and a program counter units to automatically feed in instructions. The student is also provided with a test bench that can be run in order to test the CPU.

Overview

In previous labs we have assembled a CPU up to the following level of functionality.

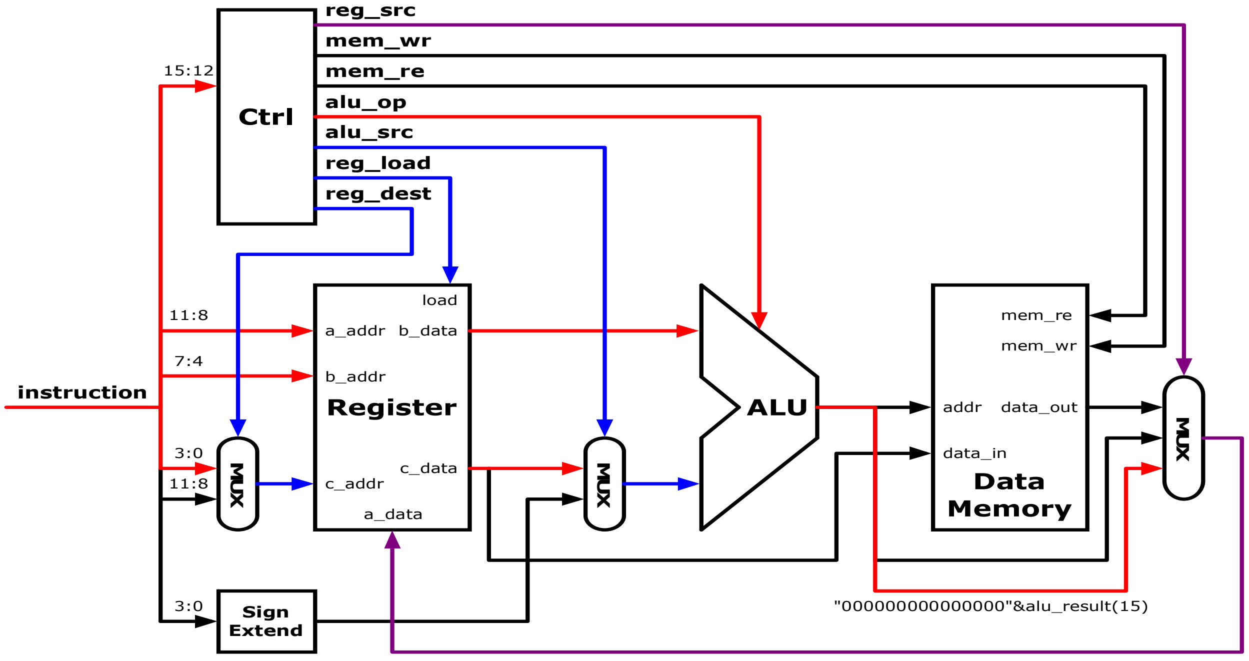

Fig.1 - CPU of lab 08

This design allows for a user to execute I-type, R-type, Load, Store, and SLT instructions with the possibility of using immediate values for these instructions. In this Lab we wish to add functionality for doing a range of instructions. These instructions were provided to our design using a test bench with hardcoded instruction values.

In this lab we wish to add a program counter and an instruction memory to our design. The design will appear as follows:

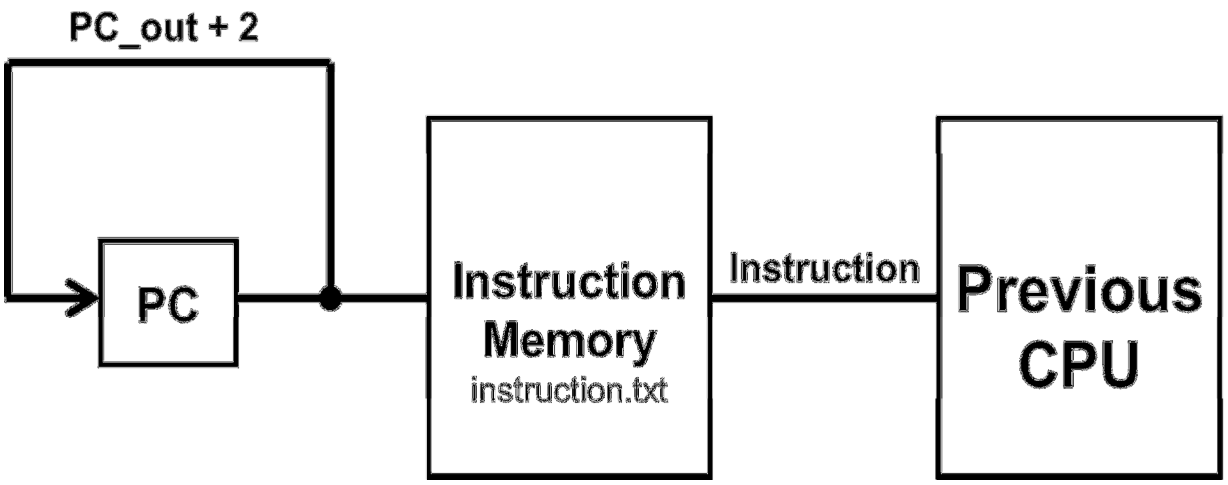

Fig.2 - CPU design with the Instruction Memory and the program counter

Lab Execution

Guidelines: For each lab there will be a report. This report will consist of a text that follows the report guideline, your code file, and screenshots of your working code. These files should be placed together in a single file, printed, and then turned into your TA. Your lab report is due the week after you finish lab. The lab is composed of a report worth 70 points (20 from the report and 50 for the vhdl files) and possibly a pre-lab worth 30 points. On labs that take place over multiple weeks there will be only a single lab report due the week after the lab is completed. Lab reports suffer a 10% per day overall penalty for late work.

Tools: Xilinx ISE. If you have forgotten how to use ISE please review lab 1.

VHDL Programming instructions: For this lab your instructions are to add the instruction memory component into your design and implement the program counter unit that will store the PC value. A test bench has been provided but does not need to be filled in. For information about components in previous labs please read that lab document. An overview of each of the new given components is below:

- Instruction Memory: This memory is implemented in the same way as Data memory (it is basically the same component). The only difference is that we will always only read (*mem_re* signal is always going to be 1) from instruction memory and never write to it.
    - **Instruction Memory will initialize from a file called *instruction.txt* (provided as additional material) which must be included in the same directory as the Memory.vhd file. This instruction file contains all of the instructions to run (so that they do not need to be hardcoded in a testbench).**
- Program Counter: This simple module acts as a register that stores the value of its input (PC + 2) on the rising edge of the clock. Basically, storing the value here simply means that at the rising edge of the clock, the input signal is going to be placed on the output line. Also this module should have a *reset* signal which helps to reset the PC value to zero whenever it is asserted low. **This module does**

not increment the value of the counter, it merely stores it. You will implement a way to increment this value in the CPU top-level entity. The following interface could be used for the VHDL implementation:

```
entity PC is
   port(
         clk : in std_logic;
         reset : in std_logic;
         PC_input : in std_logic_vector(15 downto 0);
         PC_output : out std_logic_vector(15 downto 0)
   );
end PC;
```

The PC value is incremented by 2 every clock cycle (and not by 4 as in general cases) because each instruction has been encoded inside the instruction file on 2 bytes (16 bits). Each line of the instruction file holds a half-instruction, starting with the Least Significant Byte (LSB).

- Finally, the top-level interface will be modified to only input the clock and the clear/reset signals. Normally this design does not need an output. However, to be able to monitor the operations inside the CPU, two output signals will be added to the interface:
  - *read_instruction*: a 16 bits signal that will be connected to the output of the Instruction memory module to display the instruction being executed.
  - *data_out*: the output of the Data memory mux.

Testing the Design:

- Your testbench file for the simulation will only reset off and on the whole system. Input stimuli will be provided through the instruction file.
- To test your design, the instructions listed in Table 1 will be used. First of all, you will make sure that by running your design initially (i.e. running the testbench without altering the instruction file) the simulation trace is identical to the results obtained in Lab08, as shown in Figure 3.

Table 1 – Testbench for the Simulation.

Instruction	op	r_d	r_s	r_t/immediate	Value (r_d)
ADDI R3, R0, 5					
ADDI R4, R0, 2					
SLT R11, R3,R4					
SW R3, 0(R0)					
SW R4, 4(R0)					
ADDI R6, R0, 4					
LW R7, 0(R6)					
LW R8, 0(R0)					
ADD R9, R7, R8					

SLT R10, R0,R1					
SLT R10, R1,R0					
OR R5, R10, R9					
SUBI R10, R5, 7					
SUB R11, R10, R7					
SW R11, 5(R8)					

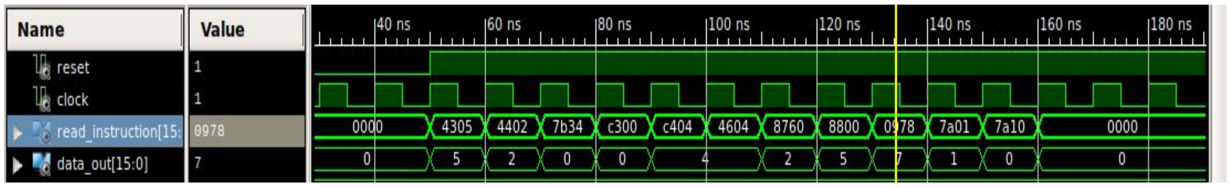

Fig.3 – Simulation trace of lab 08.

- Then, you will modify the instruction file to execute the new instructions (highlighted in the table). **After the modification, do not just run a new simulation. It is recommended to re-check the syntax of your testbench whenever you edit the Instruction file. This will quickly re-run the synthesis on your system and take into account the new instructions.**
- The value of the output result will be completed in Table 1 and included in the Lab report (fill in the table completely; which means each cell must have a value!!).