ECEC 355: Computer Organization & Architecture
Term Project: Building a Single Cycle MIPS Datapath
due Friday, March 17, 2017

Please form groups of 2 or 3 students to work on this project. If you wish to work on it by yourself, that is acceptable too.

Project Description

Using your hardware design knowledge from ECEC 302, you are being asked to develop a complete VHDL description of the MIPS-32 CPU as given in the below figure and to simulate it by executing a set of programs to demonstrate correct functionality of your processor. To simulate your VHDL code, use ModelSim, which is available on all ECE computer labs. Alternatively, you may also download the Student Edition for free from the vendor website[1].

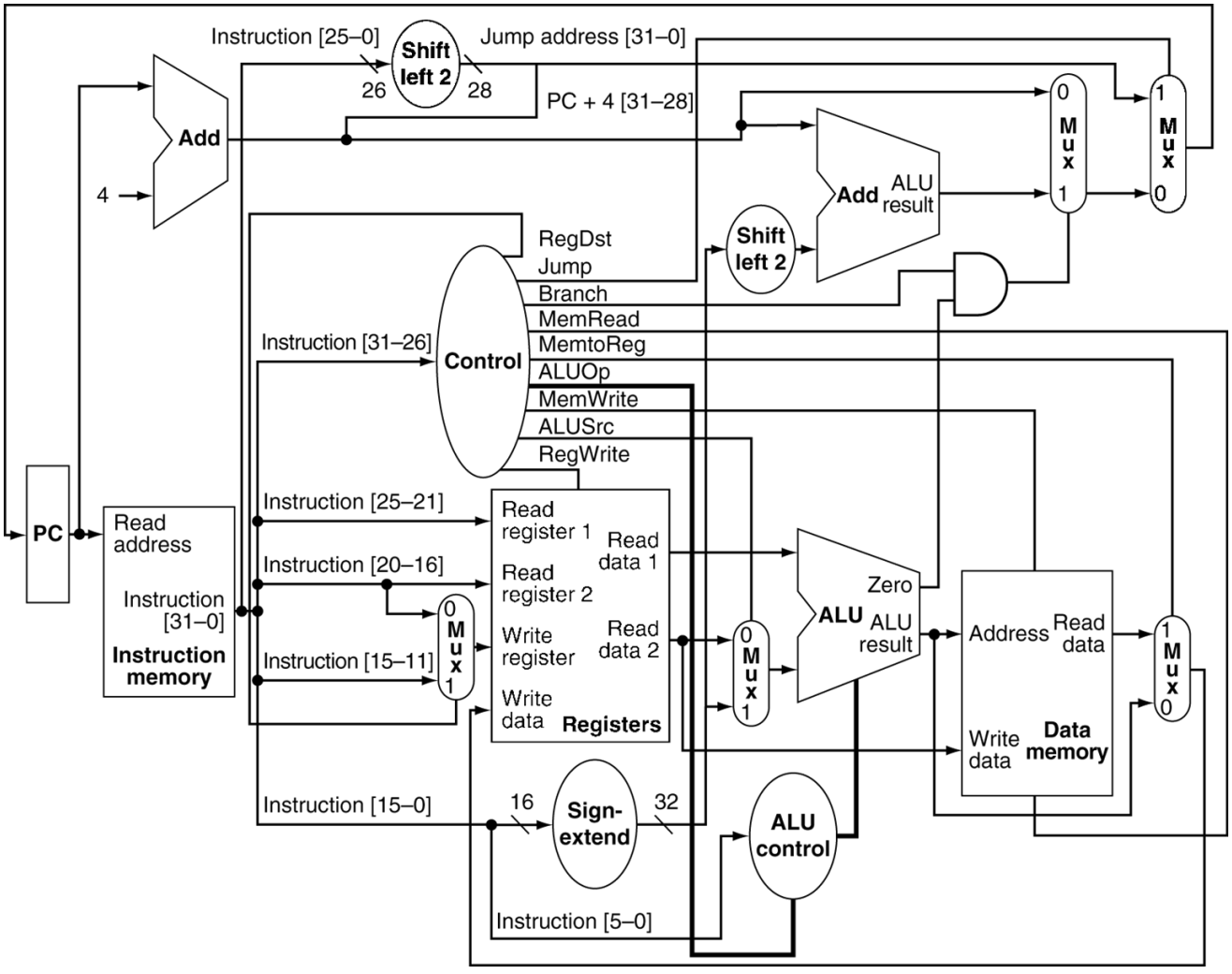

Figure 1: Single cycle MIPS processor.

[1] http://www.mentor.com/company/higher_ed/modelsim-student-edition

In order to give you a head start, the code for data memory is given to you. It is up to the group to ensure all other components are properly designed. *All components must follow the design principles introduced in Chapter 4 (The Processor) of your textbook (e.g. Control signals are correctly identified, ALU control works correctly etc.).* Do not try to "simplify" the design just because the codes below won't use it, since I might change your code to see if it still works as intended! In the end, these instructions must work correctly: all R-type instructions (except `addu`, `subu`, and `jr`), `beq`, `sw`, `lw`, `addi`, `subi`, `andi`, and `j`. Note that, for Part C, you may need to use `sll` or `srl` instructions. The figure shown in page 1 does not have the all the components needed to implement these two instructions. If you want to implement them in hardware, you may do so by suggesting ways of improving the processor. On the other hand, you may also choose a compiler based approach that allows you to not use `sll` or `srl`—therefore, not needing any additional hardware modules. Whichever method you choose, make sure to mention it in your solution and your report.

Refer to the following points when you're building your datapath:

- Assume the data memory and the instruction memory are both 128 bytes.
- All memory locations and registers should be set to 0 unless otherwise indicated.
- All input and output ports should be correctly labeled and the data width must be correct (e.g. 32-bit instructions)
- Pay attention to MUX input port indices. Your port index must match the ones shown in the figure above.
- The data registers should have the ability to read data on the clock rising edge and write data on the clock falling edge. This will enable us to complete the full instruction in one clock cycle.
- Your processor must have an appropriate clock signal (you may set this to 10 MHz).
- Each clock period should fetch and complete exactly one instruction from the instruction memory.

Part A: A Simple Set of Code

Once you complete the above tasks, convert the following MIPS code to machine code and load the instruction memory accordingly starting from address 0.

```
        lw  $s0, 0($t0)
        lw  $s1, 4($t0)
        beq $s0, $s1, L
        add $s3, $s4, $s5
        j   EXIT
L:      sub $s3, $s4, $s5
EXIT: sw  $s3, 8($t0)
```

- Initialize PC to start from 0.
- Load the following values into the registers: $t0 = 0, $s4 = 15, $s5 = 4
- Load the following values in data memory: Address 0 = 7, Address 4 = 2

Upon completion of your code segment, what is the final state of your data memory?

Part B: Different Values

Rerun the same code as above but this time setting the data at memory address 0 to a 4 (i.e. it should match the data at address 4). Upon completion of your code segment, what is the final state of your data memory?

Part C: Covert C Code to MIPS Machine Language

For this part of the project, you are given a C code segment. Convert the code into MIPS Assembly code and then to MIPS machine language. Load your instruction memory with the machine language you generated and run your simulation.

Before running your code, make sure to perform the following initializations:

- Initialize PC to start from 0.
- All registers are 0.
- Load the following values in data memory…
  - Address 0: 1
  - Address 4: 2
  - Address 8: 3
  - Address 12: 4
  - Address 16: 5
  - Address 20: 6
  - Address 24: 7
- Assume the base address for array A is 0.
- Assume the base address for array B is 48.

```
x = 10;
y = 20;
i = 0;
while(y >= x){
      B[i] = A[i] + x + y;
      x = x - 1;
      y = y - 3;
      i = i + 1;
}
```

Answer the following questions:

- What are the final values saved in the registers and the data memory?
- How many total cycles it took for your code to complete?

Deliverables

At the end of the project, you will be asked to demonstrate the functionality of your processor and the instructions you inputted. You will be graded on the correctness of your work. The following will be the graded components of this project:

- Group demonstration of parts A, B, and C (details TBD).
- Submitted code for all parts on BBLearn.
- Your report explaining the design principles of your processor. Make sure to include…
  - Any challenges you experienced
  - MIPS codes that you used for Part C
  - Screenshots of your waveform from ModelSim showing the full execution of your instructions (you may need to zoom out to make it fit in the window)
  - Answers to all the questions in all three parts of the project

Submissions will be via BBLearn. Please submit a ZIP folder that contains ALL the code for your processor, and a separate PDF file for your report (i.e. two files being submitted).

Extra Credit

Expand the functionality of your single-cycle MIPS processor to also support the bne (branch-on-not-equal) instruction. You may use any method you would like. Make sure to include in your final report details about your implementation (i.e. how your instruction works and what changes you decided to make).