Lab 04 - Single Cycle Datapath

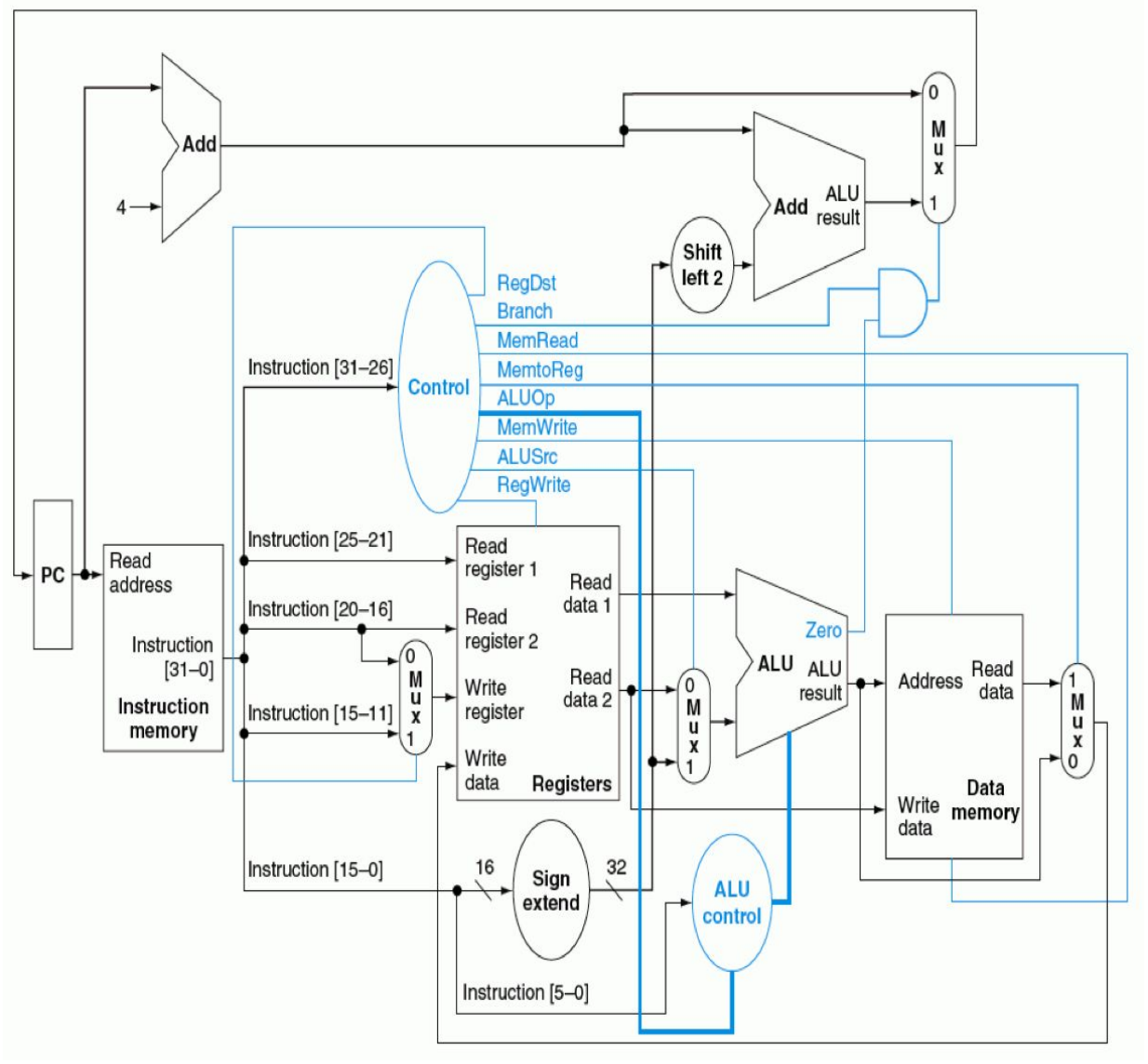
Introduction

In this lab you will be building a single cycle version of the MIPS datapath. The datapath is composed of many components interconnected. They include an ALU, Registers, Memory, and most importantly the Program Counter (PC). The program counter is the only clocked component within this design, and specifies the memory address of the current instruction. Every cycle the PC will be moved to the next instructions location. **The MIPS architecture is BYTE ADDRESSABLE**. Remember this when handling the PC, and the memory (which is WORD ADDRESSABLE).

Pre-lab

You will need to submit your test file (`init.coe`) on ilearn (check online for due dates) prior to coming to lab. Your `init.coe` should demonstrate that you have read through the lab specifications and understand the goal of this lab. You will need to consider the boundary cases. You do not need to begin designing yet, but this testbench will be helpful during the lab while you are designing.

The component connections (shown below) are outlined in the CS161 notes.



Deliverables

For the turn-in of this lab you should have a working **single cycle datapath**. The inputs to the top module (`processor.v`) are only a `clk`, and `rst` signals. The datapath should be programmed by a "`.coe`" file which holds MIPS assembly instructions. The top module should also have output debug signals to monitor the datapath, you should still connect these signals when you demo.

For this lab, you are not required to build all the datapath components but you are required to connect them together in the datapath template provided (`datapath.v`). You will be connecting this datapath and your `alu_control.v` and `control.v` from Lab 03 in the top level modules (`processor.v`). All of the files can be found here. If you need more functionality you will have to build the components yourself. To use the given components you only need to copy the given verilog files into your project. By default the architecture's memory loads data from an `init.coe` file. The programming occurs when the `rst` signal is held high. A sample `init.coe` file is given, but does not fully test the datapath. You will have to extend it.. For convenience, the assembly for the `init.coe` file can be found here.

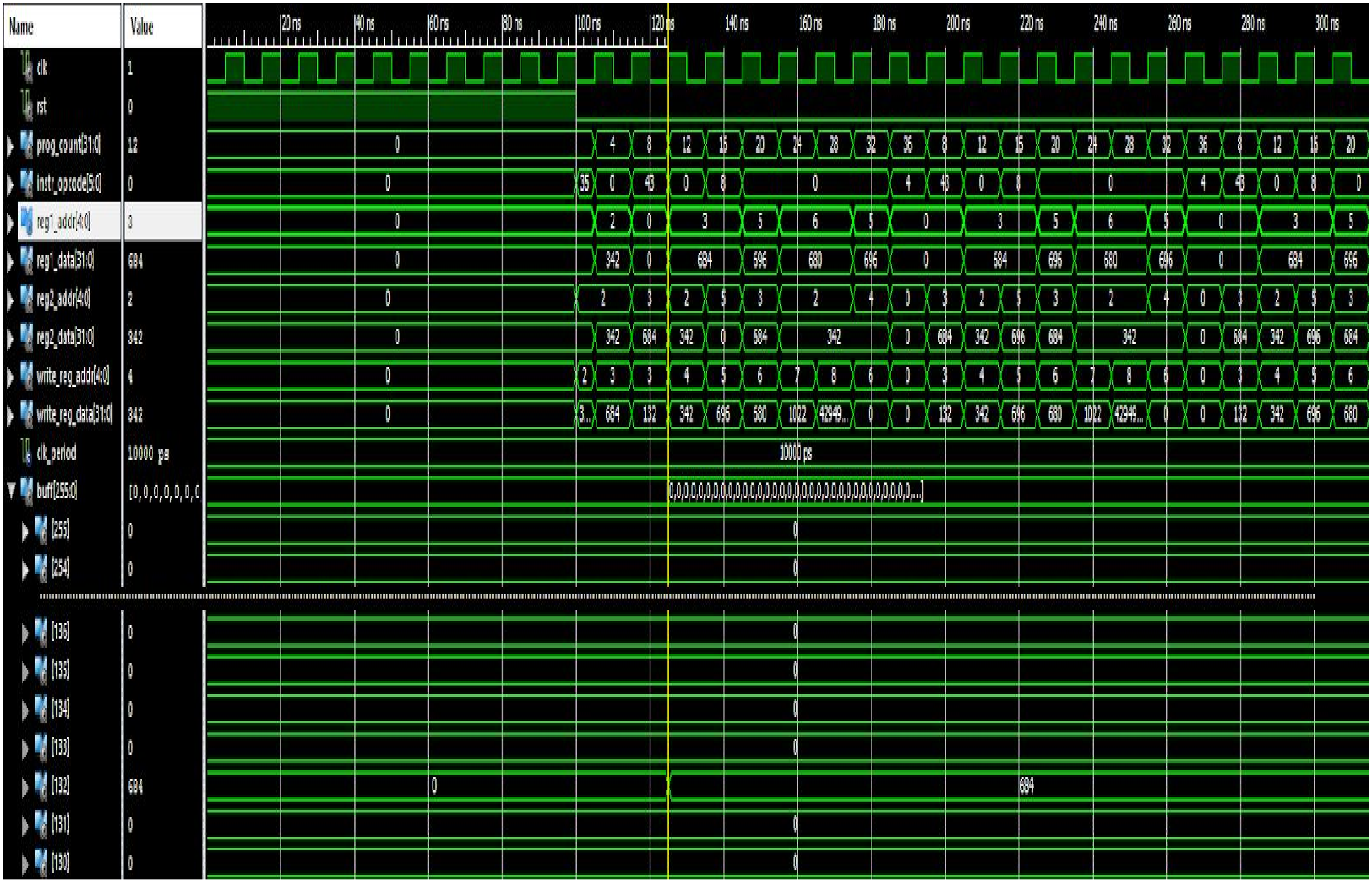
Turn-In

Each student should turn in one tar file to iLearn. Your Verilog code can be (and should be) identical to the other members of your group. The contents of which should be:

- A README file
- All Verilog (and .coe/.asm) files used in your design and for testing

Output

The expected waveform for the `init.coe` file is shown below. Notice the PC increments by 4, the code runs a loop so the waveform can continue indefinitely and if you add the memory buffer to the waveform you can scroll to see that the store properly saves 684 to address 132.


Name
Value
clk 1
rst 0
prog_count[31:0] 12
instr_opcode[5:0] 0
reg1_addr[4:0] 3
reg1_data[31:0] 684
reg2_addr[4:0] 2
reg2_data[31:0] 342
write_reg_addr[4:0] 4
write_reg_data[31:0] 342
clk_period 10000 ps
buff[255:0] [0,0,0,0,0,0,0
[255] 0
[254] 0
[136] 0
[135] 0
[134] 0
[133] 0
[132] 684
[131] 0
[130] 0
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
260 ns
280 ns
300 ns