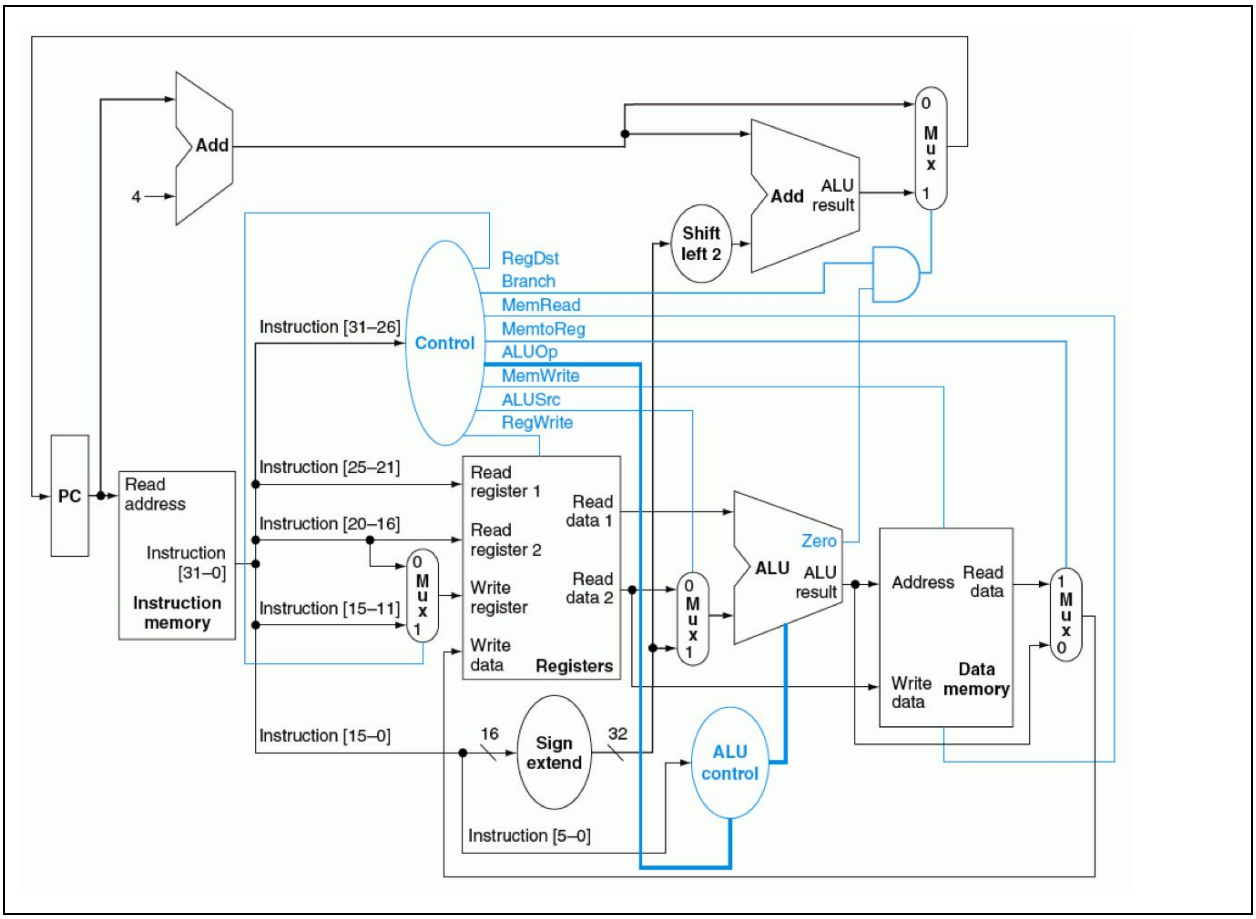
Lab 4 - Single Cycle Datapath

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).

The component connections are outlined in the class notes. A copy of the image is included below.



Deliverables

For the turn-in of this lab you should have a working single cycle datapath. The inputs to the top module are only a clk, and reset signal. The datapath should be programed by a .coe file which holds MIPS assembly instructions. The top module should also have output debug signals to monitor the datapath.

For this lab you are not required to build all the datapath components, but you are required to connect them together. The given files, containing most of what you will need, will be given by your TA. 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 in your project. By default the architecture's memory loads data from an init.coe file. The programing occurs when the reset signal is held high. A sample init.coe file is given, but does not fully test the datapath. You will have to extend it.

A more complete sample init2.coe file is given to test the datapath. It is recommended that you extend it with your own tests. For convenience, the assembly for the init2 file can be found here. ~~You will have to use your control files from lab 3 to manage the datapath.~~ A tar-ball with the given files can be found on iLearn.

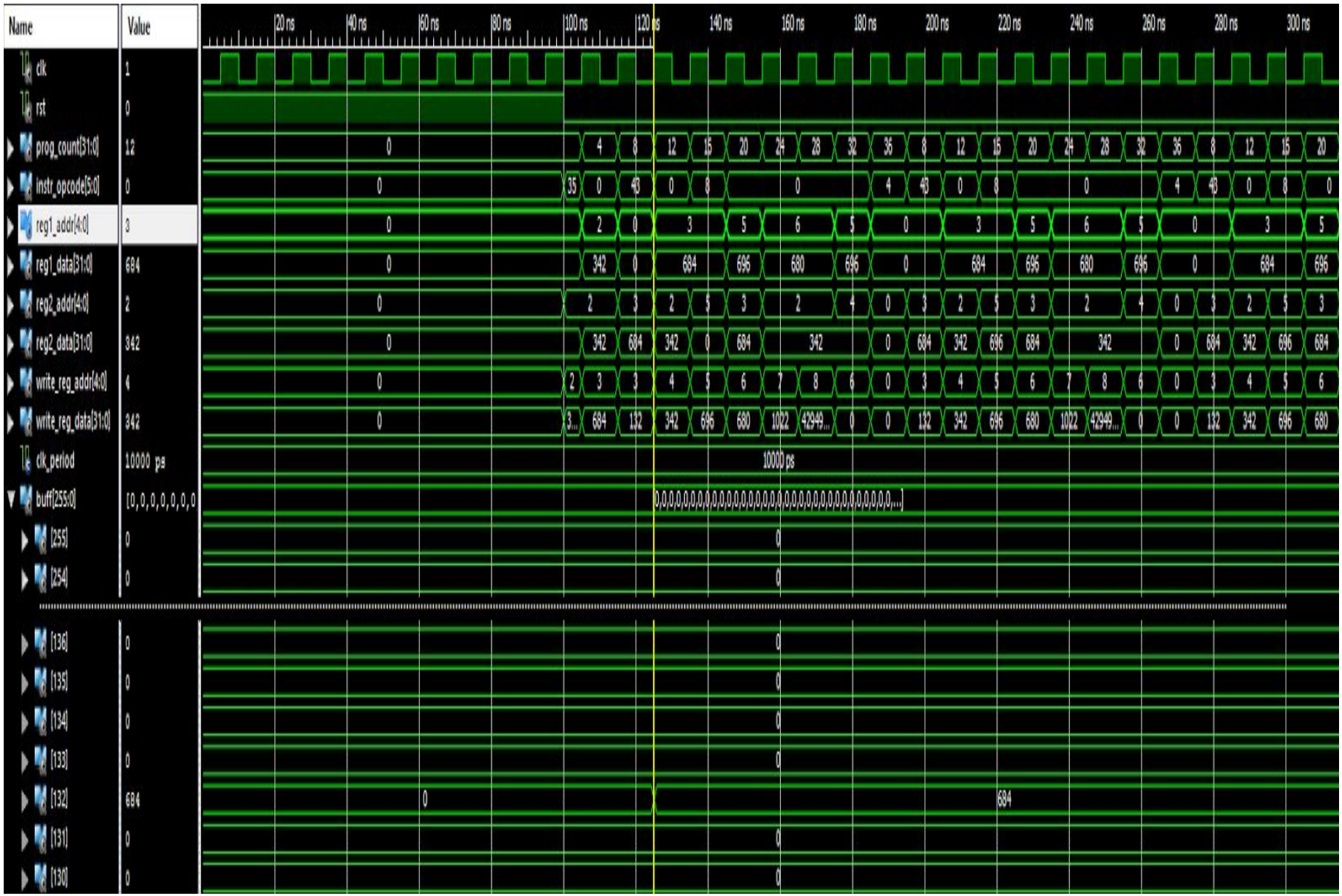
Turn-In:

Each group should turn in one tar file to iLearn. The contents of which should be:

- A README file
- All HDL files used in your design

Output

The expected waveform for the init2.coe file is shown. 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
120 ns
140 ns
160 ns
180 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
260 ns
280 ns
300 ns
10000 ps