Part 4–Flow Control Instructions [25 marks]

Now that you have a working CPU which supports `add`, `sub`, `and`, `or`, `mov`, and `loadi` instructions, it's time to add microarchitectural support for the flow control instructions `j` and `beq`. For this, you will need to modify your top-level *cpu* and *alu* modules.

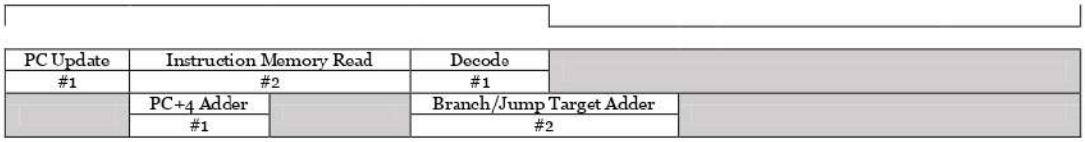
The *alu* needs an additional output port (`ZERO`) to indicate whether the result of a subtract operation is zero or not, in order to implement the `beq` instruction.

The *cpu* needs a new adder which can be used to compute the branch/jump target address based on the next PC value and the offset provided by the branch/jump instruction. We will assume that the new adder has a latency of two time units (#2), which will work in parallel to the ALU. The control functionality within the top-level *cpu* module needs to be modified to:

- manipulate the Program Counter using the immediate jump/branch target offsets provided in `j` and `beq` instructions (you will need to use adders and multiplexers);
- and generate the additional control signals required.

The timing diagrams for `j` and `beq` instructions are shown in Figure 5 below.

`j`:



`beq`:

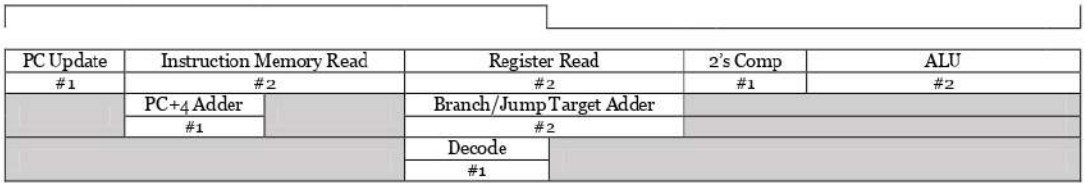


Figure 5: Timing Details for the Datapath

Before you go and modify your code, first **draw a complete block diagram** of the datapath and control by extending Figure 3 with the added components. Make sure to keep copies of your original files before modifying.

1. Modify the top-level *cpu* module and *alu* module to support the `j` and `beq` instructions. Include **a lot of comments**.

2. Write a testbench and thoroughly test your upgraded design. Hardcode your software program (instruction sequence) within the testbench file.

3. Submit a compressed file ***groupXX_lab5_part4.zip*** containing your Verilog files with the upgraded top-level *cpu module, alu* and *reg_file* modules and any other Verilog files with any sub modules of your design, testbench, complete block diagram, and screenshots of timing diagrams clearly showing the synchronized operation of the datapath and control signals for the two new instructions.

Make sure you **add a lot of comments** when coding.

Note that any form of plagiarism will result in zero marks for the entire lab.

Part 5–Extended ISA [20 marks]

There's a part 5?!? Well, only if you are up for the challenge, extend your processor to support two or more of the following instructions for maximum of **bonus 20 marks**:

`mult 4 1 2` (multiply value in register 1 by value in register 2, and place the result in register 4)
`sll 4 1 0x02` (apply logical shift left 2 times on value in register 1, and place the result in register 4)
`srl 4 1 0x02` (apply logical shift right 2 times on value in register 1, and place the result in register 4)
`sra 4 1 0x02` (apply arithmetic shift right 2 times on value in register 1, and place the result in register 4)
`ror 4 1 0x02` (apply rotate right 2 times on value in register 1, and place the result in register 4)
`bne 0x02 1 2` (if values in registers 1 and 2 are not equal, branch 2 instructions forward)

Note that you should use the same 3-bit `ALUOP` signal as in the previous part. Since you can only have 8 functional units inside the ALU with the 3-bit ALUOP control signal, you need to find a way for the new instructions to share functional units (*hint: can* `sll` *and* `srl` *share a single functional unit?*). You are discouraged to use additional control signals, unless absolutely necessary.

You must make reasonable assumptions for the latencies of any added hardware, and make sure that the new instructions can still complete within one clock cycle (8 time units).

As this is a bonus exercise, you must implement any new functional unit modules without using simple operations (e.g. using operator << for left shifting will not be acceptable).

You must provide a clear description of the instruction encodings, assigned opcodes, timing, and changes made to the datapath+control as a separate report. Two instructions correctly implemented with proper documentation will earn you 4 bonus marks. Thereafter, you can earn 4 more marks per additional instruction with proper documentation. Submit a compressed file ***groupXX_lab5_part5.zip*** containing all files of your design along with a testbench and your report.

Have fun coding. May the force be with you!