Required Problems:

1(d), 1(e), 1(f), 1(g), 1(h), 2(b), 2(c), 2(d), 2(e), 2(f)

Show your work for all problems! If you do not show your work, you will not get any credit.

Please include your name and your CS username on your homework (it will make it easier to grade).

Turning It In

You have two options:

- Turn in a hard copy at lecture
- Turn in an electronic copy using turnin; assignment name cs252_f16_hw06
  
  If you need to use a late day, you may add _late to the end of the assignment name. However, I urge you to save your Late Days for projects - don’t waste them on Homework!

In either case, a typed version (printed out or turned in) is appreciated, but not required.

Problem 1 - Timing in a Single-Cycle Processor

In this problem, consider the single-cycle processor shown on the next page. Assume that the components in the processor’s data path have the following latencies (ps = pico seconds = 10^{-12} seconds). Assume that the other components in the processor’s datapath have negligible latencies (i.e. use 0 ps for their times). For simplicity, assume that writing back to registers takes 0 time. In truth, it takes time - but that time is already accounted for as part of the register-read time.

<table>
<thead>
<tr>
<th>Instruction Memory</th>
<th>200 ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (other than ALU)</td>
<td>70 ps</td>
</tr>
<tr>
<td>Mux (any size)</td>
<td>20 ps</td>
</tr>
<tr>
<td>ALU</td>
<td>90 ps</td>
</tr>
<tr>
<td>Regs</td>
<td>90 ps</td>
</tr>
<tr>
<td>Data memory</td>
<td>250 ps</td>
</tr>
<tr>
<td>Sign-extend</td>
<td>15 ps</td>
</tr>
<tr>
<td>Shift-left-2</td>
<td>10 ps</td>
</tr>
</tbody>
</table>
1(a)
Suppose that the processor only fetched instructions, but never executed them (but the Program Counter was updated each cycle). What would the cycle time be?

1(b)
Suppose that the processor only executes conditional PC-relative branches (that is, `beq` or similar). What would the cycle time be?

1(c)
This question has three parts - focusing on the two inputs, and the output, from the ALU input 2 MUX (controlled by the `ALUSrc` control wire).
Suppose that the processor needs to execute two types of instructions: `add` and `addi`. The `add` instruction sets `ALUSrc=0`. Counting from the beginning of the cycle, how long will it be before input 0 to the MUX has arrived?
Second, the `addi` instruction sets `ALUSrc=1`. Again, counting from the beginning of the cycle, how long will it be before input 1 to the MUX has arrived?
Third, how long will it be before we know that the output from the MUX has taken its final value (give the worst case answer)?

1(d) - Turn in this one
Suppose that the processor only executes `add` instructions. What would the cycle time be? (Suggestion: Use the answers from parts (a) and (c) to help you figure this out.)

1(e) - Turn in this one
Suppose that the processor must be able to execute both `add` and `addi` instructions. Refer back to part (c), and explain why this will not require that the cycle time be any longer than your answer from part (d).

1(f) - Turn in this one
Suppose that the processor only executed `lw` instructions. What would the cycle time be?

1(g) - Turn in this one
Suppose that the processor only executes `add` instructions. What elements are in the current datapath which could be removed (list all of them, including the PC update calculation)? (Do not modify the ALU.) Also, you may ignore changes to the control logic and the bits that it produces.
Then calculate an updated cycle time for the `add` instruction, after removing the unnecessary elements. How much would the clock cycle be reduced?

1(h) - Turn in this one
Following on from the previous part, now assume that you also replace the ALU with a simple Adder. What would the cycle time be now?

Problem 2 - Timing in a Pipelined Processor
In this problem, consider the pipelined processor shown on the next page. Assume that the components in the processor’s data path have the same latencies as from Problem 1 above.
2(a)
Give the time required to complete the IF pipeline stage. List all components and their latencies, and make clear when/if one component must wait for another.

2(b) - Turn in this one
Give the time required to complete the ID pipeline stage. List all components and their latencies, and make clear when/if one component must wait for another.

2(c) - Turn in this one
Give the time required to complete the EX pipeline stage. List all components and their latencies, and make clear when/if one component must wait for another.

2(d) - Turn in this one
Give the time required to complete the MEM pipeline stage. List all components and their latencies, and make clear when/if one component must wait for another.

2(e) - Turn in this one
Combine your answers from parts a-d. What is the cycle time for this processor, once pipelining registers are added? (Assume that the pipeline registers have 0 latency.)

Compare this to your answer from problem 1(f) above. Does the pipelined processor have a faster or slower clock?

2(f) - Turn in this one
Combine your answers from parts a-d; also account for the MemToReg MUX in the WB phase (which you haven’t considered before in the pipeline processor). What is the total latency for a single instruction (that is, how long does it take to traverse all of the stages of the pipeline)?

Compare this to your answer from problems 1(d) and 1(f) above. How does pipelining affect latency?
EXAMPLES

Example: Problem 1(a)
If we only fetch instructions and update the Program Counter, we would need to use the following elements:

- Instruction Memory (200 ps)
- Add, for PC+4 (70 ps), plus the Branch Destination MUX (20 ps)

Since these two things can happen in parallel, the total time is 200 ps per clock cycle.

Not required for your answer, just an interesting note: 200 ps per clock is equivalent to a 5 GHz processor.

Example: Problem 1(b)
To execute beq, we would use the following components:

- Instruction Memory (200 ps)
- Add, for PC+4 (70 ps)
- Add, for PC+4+branchOffset (70 ps)
- Register file (90 ps)
- ALUSrc MUX (20 ps)
- ALU (90 ps)
- Sign-extend (15 ps) and shift-left-2 (10 ps)
- PC destination MUX (20 ps)

There are several paths to check. First, we note that the PC+4 Adder completes in 70 ps; this provides one of the inputs to the branchOffset Adder. The other input comes through a much longer path: Instruction Memory, sign-extend, and shift-left-2, which is 200+15+10=225 ps. So the two inputs are known at 225 ps, and this Adder produces its result at 295 ps. Thus, the PC destination MUX has both of its inputs (PC+4 and PC+4+branchOffset) at 295 ps.

The control for this MUX is determined first by reading the instruction and two registers; one of the two registers goes through a MUX, and then the ALU executes. This takes a total of 200+90+20+90=400 ps; so the control input for the PC destination MUX is ready at 400 ps.

Thus, the PC destination MUX gives its answer at 420 ps; this is the total cycle time.
Example: Problem 1(c)
In the `add` instruction, the value is taken from the register, meaning that the following components are active:

- Instruction memory (200 ps)
- Registers (90 ps)

Thus, this value reaches the MUX at 290 ps.

In the `addi` instruction, the value is taken from the immediate field, meaning that the following components are active:

- Instruction Memory (200 ps)
- Sign extend logic (15 ps)

Thus, this value reaches the MUX at 215 ps.

We have two paths to the inputs of the MUX (290 ps and 215 ps). But either way, this MUX takes 20 ps to calculate a result. Therefore, we can assume that this MUX completes 20 ps after the last input arrives - that is, $290 + 20 = 310$ ps.

Example: Problem 2(a)
The IF stage includes:

- Instruction Memory (200 ps), which doesn’t wait for anything
  (Technically, it waits for the Program Counter, but we’ve been told to ignore that in this model.)
- Adder (PC+4) (70 ps), which doesn’t wait for anything.
- Branch MUX (20 ps), which waits for the Adder.

Thus, the stage completes in 200 ps, which is $\max(200,90)$. 