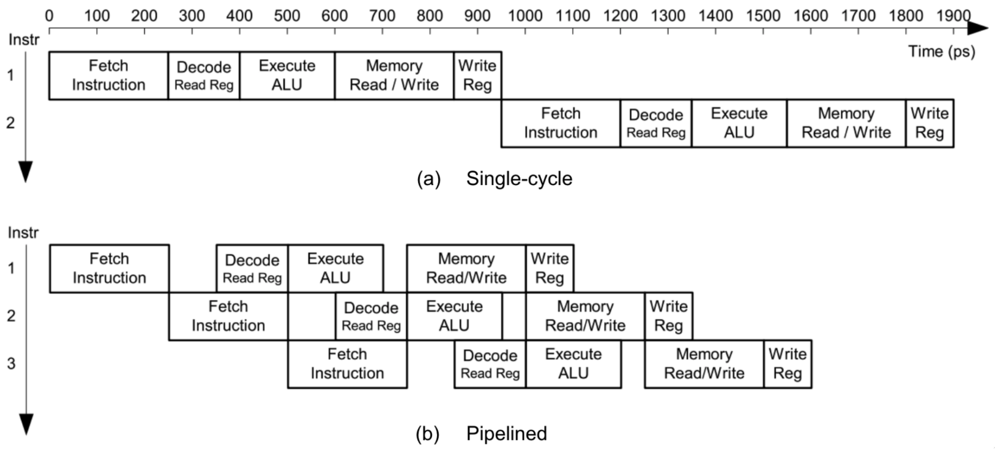
EECS 151/251A Homework 5

Due Friday, October 7th, 2022 11:59PM

Problem 1: Pipelined Design

Here is a diagram that shows timing of datapath stages for both single-cycle processor design, and 5-stage pipelined processor design.



(a) Answer the following two-choice questions about pipelined processor design.

- [① | ②] The main goal of pipelined processor design is to increase ① clock frequency | ② executed instructions per cycle.
- [T | F] Deeper pipelines with more stages can always yield higher performance, because we can execute more instructions in parallel.
- [T | F] The area and power cost is higher for pipelined design than single-cycle design.
- [T | F] Pipelined design generally has lower CPI (cycles per instruction) than single-cycle design.
- [T | F] Pipelined design has lower hardware utilization than single-cycle design, because each instruction can occupy only one stage at a time.
- [T | F] In a single-cycle design, none of the hazards – structural, data and control – can occur.

(b) Assume that the critical path in *Fetch* stage takes 150ps, *Decode* takes 200ps, *Execute* takes 50ps, *Memory* takes 150ps, and *Writeback* takes 100ps. What would be the minimum clock period achievable for this pipelined design? How much speedup do you achieve over the single-cycle design?

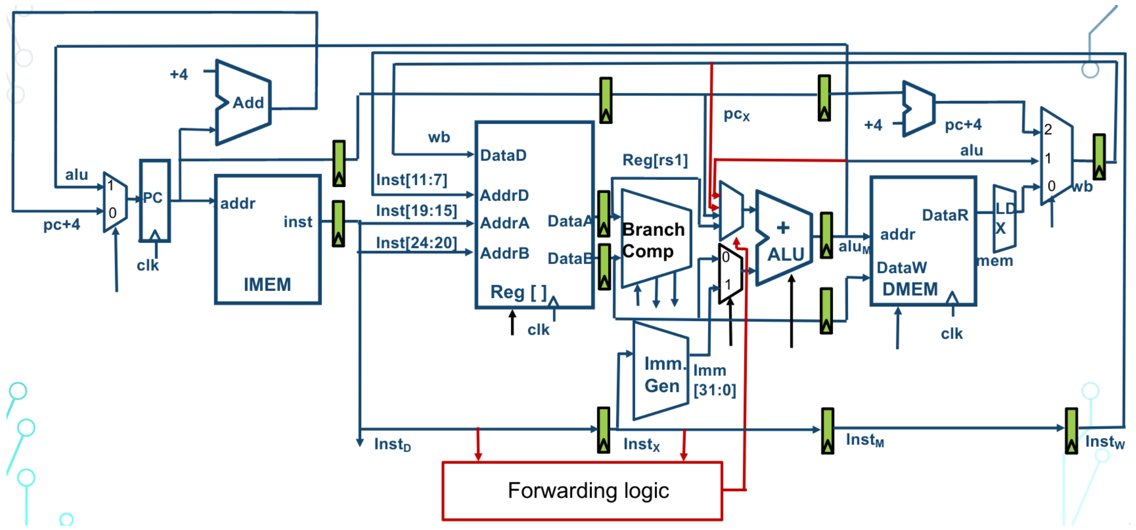
Note: The diagram is not drawn to scale. Also, ignore register clk-q delays or setup/hold time requirements.

(c) In your course project, you will be designing a 3-stage pipelined processor instead of a 5-stage one. Which stages would you merge together among the five F/D/X/M/W stages to achieve best performance? Express using formats like FDX/M/W, F/DXM/W, etc.

Problem 2: Data Hazard and Forwarding

This diagram shows the datapath for a 5-stage pipeline with forwarding logic implemented.

Assume that the write and the read to the same register can happen in a single cycle. Also, assume each stage has the same critical path as Problem 1: *Fetch* 150ps, *Decode* 200ps, *Execute* 50ps, *Memory* 150ps, and *Writeback* 100ps.



(a) First, assume no forwarding is implemented. How many cycles will the following RISC-V code take to execute? What is the CPI of this code? Show how you derived your result.

Note: Count all the cycles starting from when the first instruction enters the fetch stage, to when the last instruction leaves the writeback stage.

```
and x1, x2, x4
sub x3, x1, x5
or  x2, x4, x5
srl x1, x1, x3
add x4, x2, x1
```

(b) Now, all data forwarding logic including ALU-to-ALU and Mem-to-ALU are implemented, as shown in the datapath diagram. How do the answers of (a) change?

(c) Lastly, suppose we are employing a higher-frequency design where the register file is no longer able to write to and read from the same register at a single cycle. However, this faster register file reduces the critical path of the *Decode* (D) stage from 200 down to 100ps.

What are the number of cycles and CPI with this design? Also, taking into account the new critical path of the design, how does the actual time taken to execute this code change from (b)? Assume that we don't add any new forwarding logic to the datapath.

(d) In the datapath diagram, there is actually a missing input wire to the forwarding logic block that is needed to implement all data forwarding. Which signal should this wire be coming from?

(e) (251A Only) Calculate the number of cycles the following code takes to execute, assuming all aforementioned forwarding logic is implemented as in the diagram. Is the forwarding implementation in the diagram sufficient to eliminate all stalls? If not, how would you change the forwarding datapath to achieve this? Would there be any disadvantage in such implementation?

```
lw  x1, 4(x2)
and x3, x1, x3
```

Problem 3: Control Hazard

Refer back to the datapath diagram shown in Problem 2.

Assume that all data forwarding logic are implemented so that only control hazards are causing stalls in the pipeline.

(a) Here is a simple loop code in RISC-V.

```
    li   x1, 0
    li   x4, 0
    li   x2, 5
    blez x2, exit
loop:
    lw   x3, 0(x1)
    add  x4, x4, x3
    addi x1, x1, 4
    addi x2, x2, -1
    bnez x2, loop
exit:
    add  x4, x4, x4
```

i. If branches are always predicted not taken, how many branches are predicted correctly? How many are not? What is the total **stall cycles** introduced by branch mispredictions?

ii. Repeat i. for when the branches are always predicted taken.

(b) You wrote a branch predictor that can predict branches at 99% accuracy. It is a simple block whose inputs are the PC and branch outcome of the current branch instruction, and whose output is the predicted next PC. Which stage in the datapath would you place this block to be most effective? You don't need to specify how to wire things up correctly, just choose a stage among F/D/X/M/W.

(c) Remember that you have to design a 3-stage pipeline for your project. Could you merge the 5 stages into 3 in such a way that minimizes cycle cost of control hazards? Thinking back to what we did in Problem 1-(b), would this design yield performance improvement for the predict-all-taken case in (a)?

Note: Assume we don't move any components around the datapath, but simply remove pipeline registers to merge stages.