# CS M151B Homework 5

Charles Zhang

February 8, 2022

## Problem 4.16

**In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:**

| `IF` | `ID` | `EX` | `MEM` | `WB` |
|---|---|---|---|---|
| 250ps | 350ps | 150ps | 300ps | 200ps |

**Also, assume that instructions executed by the processor are broken down as follows:**

| ALU/Logic | Jump/Branch | Load | Store |
|---|---|---|---|
| 45% | 20% | 20% | 15% |

**a) What is the clock cycle time in a pipelined and non-pipelined processor?**

In a pipelined processor, the clock cycle time is determined by the pipeline stage with the longest latency. Therefore, the `ID` stage is the critical path, and the clock cycle time is:

$$\boxed{\text{CT} = 350\text{ps}}$$

In a non-pipelined processor, the clock cycle time is determined by the instruction with the longest latency. In this case, loads would have the longest latency, as they need to go through every stage in the pipeline, leading to a clock cycle time of:

$$\text{CT} = 250\text{ps} + 350\text{ps} + 150\text{ps} + 300\text{ps} + 200\text{ps}$$

$$\boxed{\text{CT} = 1250\text{ps}}$$

**b) What is the total latency of an `lw` instruction in a pipelined and non-pipelined processor?**

In both processors, an `lw` instruction would need to be processed by all stages of the pipeline. Therefore, the total latency of `lw` in both cases would be:

$$\text{CT} = 250\text{ps} + 350\text{ps} + 150\text{ps} + 300\text{ps} + 200\text{ps}$$

$$\boxed{\text{CT} = 1250\text{ps}}$$

**c) If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?**

If this were the case, the `ID` stage would be split, as it is the current critical path of the pipelined processor. This would lead to it being split into 2 stages, each with a latency of 175ps. The new critical path would be the `MEM` stage, resulting in a clock cycle time of:

$$\boxed{\text{CT} = 300\text{ps}}$$

**d) Assuming there are no stalls or hazards, what is the utilization of the data memory?**

The instructions that make use of data memory are the loads and stores. As a result, the data memory's utilization is:

$$20\% + 15\% = \boxed{35\%}$$

**e) Assuming there are no stalls or hazards, what is the utilization of the write-register port of the `Registers` unit?**

The instructions that make use of the write-register port are ALU/Logic and loads. As a result, the write-register port's utilization is:

$$45\% + 20\% = \boxed{65\%}$$

# Problem 4.27

**Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:**

```
add $s3, $s1, $s0
lw  $s2, 4($s3)
lw  $s1, 0($s4)
or  $s2, $s3, $s2
sw  $s2, 0($s3)
```

**a) If there is no forwarding or hazard detection, insert `NOP`s to ensure correct execution.**

```
add $s3, $s1, $s0
NOP
NOP
lw  $s2, 4($s3)
lw  $s1, 0($s4)
NOP
or  $s2, $s3, $s2
NOP
NOP
sw  $s2, 0($s3)
```

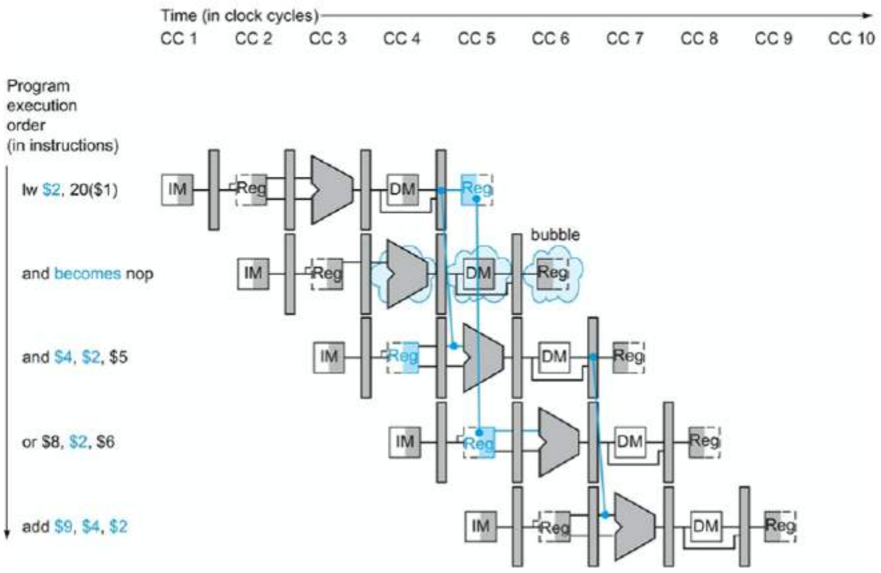
**b) Now, change and/or rearrange the code to minimize the number of `NOP`s needed. You can assume register `$t0` can be used to hold temporary values in your modified code.**

```
add $s3, $s1, $s0
lw  $s1, 0($s4)
NOP
lw  $s2, 4($s3)
NOP
or  $s2, $s3, $s2
NOP
NOP
sw  $s2, 0($s3)
```

**c) If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?**

The code executes properly, as there is no load-use case that the hazard detection unit would be needed to solve. There would be no need for `NOP`s due to the introduction of the forwarding logic.

**d) If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in the following figure:**



| | CC1 | CC2 | CC3 | CC4 | CC5 | CC6 | CC7 |
|---|---|---|---|---|---|---|---|
| HDU | No Stall | No Stall | No Stall | Stall | No Stall | No Stall | No Stall |
| FU | `00` | `00` | `00` | `00` | `ForwardA = 01` | `00` | `ForwardA = 01` |

**e) If there is no forwarding, what new input and output signals do we need for the hazard detection unit depicted above? Using this instruction sequence as an example, explain why each signal is needed.**

In this situation, we would need to start by taking in `ID/EX.RegWrite` and `ID/EX.RegisterRd`. This would tell us if the instruction being executed is writing to a register, and what register it was writing to. We would also need to take in `IF/ID.RegisterRs` and `IF/ID.RegisterRt`, telling us what registers the next instruction is going to be reading from. With these signals, we can now tell if one is dependent on the results of the other, such as the RAW dependence exhibited by the `lw` followed by the `add` above. Here, we would simply inject a bubble into the pipeline, just like we did for the forwarding case, but we would do it one stage earlier, so we would need an output signal that could bubble at the `ID` stage.

**f) For the new hazard detection unit from e), specify which output signals it asserts in each of the first five cycles during the execution of this code.**

| CC1 | CC2 | CC3 | CC4 | CC5 |
|---|---|---|---|---|
| No Stall | No Stall | Stall | Stall | No Stall |

## Problem 4.28

**The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:**

| R-type | `beqz/bnez` | `jal` | `lw` | `sw` |
|---|---|---|---|---|
| 40% | 25% | 5% | 25% | 5% |

**Also, assume the following branch predictor accuracies:**

| Always-Taken | Always-Not-Taken | 2-Bit |
|---|---|---|
| 45% | 55% | 85% |

**a) Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the `ID` stage and applied in the `EX` stage that there are no data hazards, and that no delay slots are used.**

We know that, since the branch outcome is applied in the `EX` stage, we lose 2 cycles on a mispredicted branch. Therefore:

$$\%\text{ of mispredicted branches} = 0.25 \times (1 - 0.45) = 13.75\%$$

$$\Delta\text{CPI} = 0.1375 \times 2 = \boxed{0.275}$$

**b) Repeat a) for the "always-not-taken" predictor.**

$$\%\text{ of mispredicted branches} = 0.25 \times (1 - 0.55) = 11.25\%$$

$$\Delta\text{CPI} = 0.1125 \times 2 = \boxed{0.225}$$

**c) Repeat a) for the 2-bit predictor.**

$$\%\text{ of mispredicted branches} = 0.25 \times (1 - 0.85) = 3.75\%$$

$$\Delta\text{CPI} = 0.0375 \times 2 = \boxed{0.075}$$

**d) With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions to some ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.**

In this situation, the frequency of `beqz` and `bnez` instructions would be cut in half, but the prediction accuracies and cycle delay wouldn't change. Therefore:

$$\text{\% of mispredicted branches} = 0.125 \times (1 - 0.85) = 1.875\%$$

$$\Delta\text{CPI} = 0.01875 \times 2 = 0.0375$$

Since there is a 1-to-1 conversion of branch instruction to ALU instruction, we know that the instruction count of the program remains the same. In addition, since none of the pipeline was changed, we know the cycle time must be exactly the same as well. The speedup can therefore be represented as follows, where `x` is the CPI without mispredictions:

$$\boxed{\frac{P_f}{P_i} = \frac{x + 0.075}{x + 0.0375}}$$

**e) With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.**

In this situation, we have the same change in CPI as in d). In addition, the pipeline still remains unchanged, so the cycle time must be the same. However, we know that the instruction count increases by 12.5% of the original instruction count, as half of the branch instructions turn into 2 ALU instructions. Therefore, the speedup can be determined by:

$$\boxed{\frac{P_f}{P_i} = \frac{x + 0.075}{1.125(x + 0.0375)}}$$

**f) Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-bit predictor on the remaining 20% of the branch instructions?**

$$0.85 = (0.8 \times 1) + (0.2 \times x)$$

$$\boxed{x = 25\%}$$