ECE4700J Computer Architecture

Summer 2022

HW #2

Due: **2:59am (Beijing Time) June 13th, 2022**
Please Submit a single ***PDF*** *file on Canvas*

Note: for those who are unfamiliar with RISC-V ISA, a reference card has been attached at the end of this document.

Q1 (15%): Now we have a typical 5 stage pipeline (instruction fetch (IF) | decode(D) | execute (EX) | memory access (M) | writeback (WB)) RISC-V processor. You are given the assembly code shown below.

```
lw x1, 0(x2)
add x5, x4, x1
```

1) (5%) Assuming an **asynchronous** read data memory, how many cycles will this set of instructions take to execute? Identify any data hazards.
2) (10%) If you could add another forwarding path from the output RD of the data memory, how many cycles will these instructions take to execute? What could be a disadvantage of forwarding from the output of the data memory versus from the pipeline register clocking RD?

Sample Solutions:

1) 8 cycles or 9 cycles (depends on implementations). RAW hazard cause by x1 access.
2) 6 cycles or 7 cycles (depends on implementations).
The downside would be that the critical path becomes even longer, and this can potentially downgrade the performance benefits.

Q2 (40%): 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	ld	sd
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

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

1) (10%) 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.
2) (5%) Repeat 1) for "always-not-taken" predictor.
3) (5%) Repeat 1) for "2-Bit" predictor.
4) (10%)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.
5) (10%) 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?

Sample Solutions:

1) An incorrectly predicted branch will cause three instructions to be flushed: the instructions currently in the IF, ID, and EX stages. (At this point, the branch instruction reaches the MEM stage and updates the PC with the correct next instruction.) In other words, 55% of the branches will result in the flushing of two or three instructions (depends on the implementation), giving us a CPI of

$1 + (1 - 0.45)(0.25)3 = 1.4125$ (3 stall cycles)

Or

$1 + (1 - 0.45)(0.25)2 = 1.275$ (2 stall cycles)

2)
1 + (.25)(1 - .55)3 = 1.3375 (3 stall cycles)

Or

1 + (1 - 0.55)(0.25)2 = 1.225 (2 stall cycles)

3)

1 + (.25)(1 - .85)3 = 1.1125 (3 stall cycles)

Or

1 + (1 - 0.85)(0.25)2 = 1.075 (2 stall cycles)

4)

Changing half of the branch instructions to an ALU instruction reduces the percentage of instructions that are branches from 25% to 12.5%. Because predicted and mispredicted branches are replaced equally, the misprediction rate remains 15%.

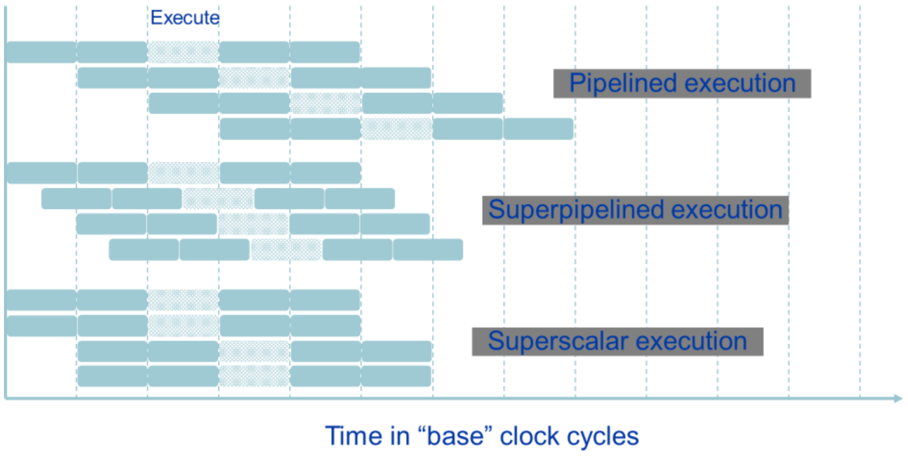
Thus, the new CPU is 1 + (.125) (1 - .85) (3) = 1.05625. This represents a speedup of 1.1125 / 1.05625 = 1.0533 (3 stall cycles)

Or

1 + (.125) (1 - .85) (2) = 1.0375
Speedup is 1.075/1.0375 = 1.036

5) We know that 80% of branches are always predicted correctly and the overall accuracy is 0.85. Thus, 0.8*1 + 0.2*x = 0.85. Solving for x shows that x = 0.25

Q3 (10%): Differentiate in your own words between the concepts of pipelining, superpipielining and instruction level parallelism such as superscalar.



Sample Solutions:

The above slide gives a clear view of differences among three implementations.

A pipelined processor executes 1 instruction per "base" clock cycle, while the superscalar and superpipelined processors can execute more. In superscalar, parallel instructions are processed in each pipeline stage. In the case of superpipelined processor, its clock frequency is further divided while the superscalar processor's clock frequency is the same as the pipelined processor.

Q4 (15%): Add NOP instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

```
addi   x11, x12, 5
add    x13, x11, x12
addi   x14, x11, 15
add    x15, x13, x12
```

Sample Solutions:

```
addi x11, x12 , 5
nop
nop
add x13, x12 , x12
addi x14, x11 , 15
```

```
6 nop
7 add x15, x13 , x12
```

Or

```
1 addi x11, x12 , 5
2 nop
3 nop
4 nop
5 add x13, x12 , x12
6 addi x14, x11 , 15
7 nop
8 nop
9 add x15, x13 , x12
```

Q5 (20%): Consider the fragment of RISC-V assembly below:

```
sd    x29, 12(x16)
ld    x29, 8(x16)
sub   x17, x15, x14
beqz  x17, label
add   x15, x11, x14
sub   x15, x30, x14
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

1) (10%) What is a structural hazard? Draw a pipeline diagram to show where the code above will stall.
2) (10%) In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

Sample Solutions:

1) Structrural hazard happens when a required resource is busy

```
Stalls are marked with **

sd  x29, 12(x16)   IF ID EX ME WB
ld  x29, 8(x16)       IF ID EX ME WB
sub x17, x15, x14        IF ID EX ME WB
bez x17, label              ** ** IF ID EX ME WB
add x15, x11, x14                    IF ID EX ME WB
sub x15,x30,x14                         IF ID EX ME WB
```

2) Reordering code won’t help. Every instruction must be fetched; thus, every data access causes a stall. Reordering code will just change the pair of instructions that are in conflict.