CS1216 - Monsoon 2022 - Homework 5

Jivansh Sharma
UG 24 1020211193

21/11/2022

Collaborators: None

1. Consider an in-order 5-stage pipeline similar to the one discussed in class. First assume that the pipeline does not support bypassing (forwarding). What are the stall cycles introduced between the following pairs of back-to-back instructions? Then, solve the same problem while assuming support for bypassing. Show your work clearly i.e., show how each instruction goes through the 5 stages, indicate the point of production and point of consumption, show where consuming instruction is held back when there are stalls. Recall that a register read is performed in the second half of the D/R stage and a register write is performed in the first half of the RW stage

```
A)  lw $5, 64($7)
    add $4, $5, $3
```

```
Without Forwarding
```
Cycles: 1 2 3 4 5 6 7

```
lw would go in the following stages:
IF D/R ALU DM  RW

and add would go in the following stages:
IF STALL STALL D/R ALU DM RW
```

The add instruction consumes 2 stall cycles in D/R stage as it waits for lw to write to $5. lw finishes writing in first half of cycle 5 and add picks it up from the second half.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6 7

```
lw would go in the following stages:
IF D/R ALU DM  RW
```

```
and add would go in the following stages:
IF STALL D/R ALU DM RW
```

In this case, at the 4th cycle the address is at the ALU stage and loads it into the DM stage for add. Then in the 5th clock cycle, the operand is available.

```
B)  lw $1, 32($2)
    sw $1, 128($5)
```

```
Without Forwarding
```
Cycles: 1 2 3 4 5 6 7 8

```
lw would go in the following stages:
IF D/R ALU DM  RW

and sw would go in the following stages:
IF STALL STALL D/R ALU DM RW
```

The lw instruction consumes 2 stall cycles in D/R stage as it waits for lw to write to $1. lw finishes writing in first half of cycle 1 and add picks it up from the second half.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6 7

```
lw would go in the following stages:
IF D/R ALU DM  RW

and sw would go in the following stages:
IF STALL D/R ALU DM RW
```

In this case, at the 4th cycle the address is at the ALU stage and loads it into the DM stage for sw.

```
B)  lw $1, 32($2)
    sw $1, 128($5)
```

```
Without Forwarding
```
Cycles: 1 2 3 4 5 6 7 8

```
lw would go in the following stages:
IF D/R ALU DM  RW

and sw would go in the following stages:
IF STALL STALL D/R ALU DM RW
```

The lw instruction consumes 2 stall cycles in D/R stage as it waits for lw to write to $1. lw finishes writing in first half of cycle 5 and add picks it up from the second half.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6 7

```
lw would go in the following stages:
IF D/R ALU DM  RW

and sw would go in the following stages:
IF STALL D/R ALU DM RW
```

In this case, at the 4th cycle the address is at the ALU stage and loads it into the DM stage for sw.

```
C)  add $1, $2, $3
    sub $4, $2, $1

Without Forwarding
```
Cycles: 1 2 3 4 5 6 7 8

```
add would go in the following stages:
IF D/R ALU DM  RW

and sub would go in the following stages:
IF STALL STALL D/R ALU DM RW
```

The sub instruction consumes 2 stall cycles in D/R stage as it waits for add to write to $1. add finishes writing in first half of cycle 5 and sub picks it up from the second half.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6

```
add would go in the following stages:
IF D/R ALU DM  RW

and sub would go in the following stages:
IF D/R ALU DM RW
```

There is no stalling now, the shared value is computed in ALU stage of add instruction and passed to sub in cycle 4.

2. Consider an in-order pipeline that has the same stages as a 5 stage pipeline discussed in class. However, unlike the 5-stage pipeline discussed in class, a register read in this design takes an entire cycle and a register write takes an entire cycle (and not a half cycle).
Use the same sequence of instructions from Question 1, and show how these sequences proceed through the pipeline. Indicate the number of stall cycles that are experienced by each sequence. Show your work for both versions of the pipeline: one without bypassing, and another one with bypassing.

```
A)  lw $5, 64($7)
    add $4, $5, $3
```

```
Without Forwarding
```
Cycles: 1 2 3 4 5 6 7 8 9

```
lw would go in the following stages:
IF D/R ALU DM  RW

and add would go in the following stages:
IF STALL STALL STALL D/R ALU DM RW
```

The add instruction consumes 3 stall cycles in D/R stage as it waits for lw to write to $5. This now takes an additional cycle as compared to q1 hence the increase to 3 cycles.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6 7

```
lw would go in the following stages:
IF D/R ALU DM  RW

and add would go in the following stages:
IF STALL D/R ALU DM RW
```

In this case, at the 4th cycle the address is at the ALU stage and loads it into the DM stage for add. Then in the 5th clock cycle, the operand is available.

```
B)  lw $1, 32($2)
    sw $1, 128($5)
```

```
Without Forwarding
```
Cycles: 1 2 3 4 5 6 7 8 9

```
lw would go in the following stages:
IF D/R ALU DM  RW

and sw would go in the following stages:
IF STALL STALL STALL D/R ALU DM RW
```

The lw instruction consumes 3 stall cycles in D/R stage as it waits for lw to write to $1. lw finishes writing in cycle 5 and add sw picks it up from the cycle 6.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6 7

```
lw would go in the following stages:
IF D/R ALU DM  RW

and sw would go in the following stages:
IF STALL D/R ALU DM RW
```

In this case, at the 4th cycle the address is at the ALU stage and loads it into the DM stage for sw in the 5th cycle.

```
C)  add $1, $2, $3
    sub $4, $2, $1

Without Forwarding
```
Cycles: 1 2 3 4 5 6 7 8 9

```
add would go in the following stages:
IF D/R ALU DM  RW

and sub would go in the following stages:
IF STALL STALL STALL D/R ALU DM RW
```

The sub instruction consumes 3 stall cycles in D/R stage as it waits for add to write to $1. add finishes writing in cycle 5 and sub picks it up from the cycle 6.

```
With Forwarding
```
Cycles: 1 2 3 4 5 6

```
add would go in the following stages:
IF D/R ALU DM  RW

and sub would go in the following stages:
IF D/R ALU DM RW
```

There is no stalling now, the shared value is computed in ALU stage of add instruction (Cycle 3) and passed to sub in cycle 4.

4. Consider a program that executes a large number of instructions. Assume that the program does not suffer from stalls from data hazards or structural hazards. Assume that 40% of all instructions are branch instructions, and 65% of these branch instructions are Taken.
What is the average CPI for this program when it executes on each of the processors listed below? All of these processors implement a 12-stage pipeline and resolve a branch outcome at the end of the 4th stage. The 1st stage fetches an instruction, the 2nd stage does decode, the 3rd stage does register read, and the 4th stage does the computations for the branch. (30 points)

(a) The processor pauses the instruction fetch as soon as it fetches a branch. Instruction fetch is resumed after the branch outcome has been resolved.

For 12 stage pipeline, each would be 1 CPI. But in this case, CPI for branch instructions is different as the processor stalls the instruction fetch as soon as it fetches the branch instruction.

As the branch is resolved in 4th stage, the processor fetches new instructions 3 clock cycles after branch instruction is fetched so branch instruction takes 3 clock cycles to execute.

Avg. CPI = (0.4 * 3) + (0.12 * 1) $\implies CPI = 1.32$

(b) The processor always fetches instructions sequentially. If a branch is resolved as Taken, the incorrectly fetched instructions after the branch are squashed.

When a branch instruction is taken, previous stages are squashed. Branch when taken consumes 3 cycles, and all others take 1 cycle.

Avg. CPI = (0.4 * 0.65 * 3 * 0.12 * 1) + (0.4 * 0.35 * 1) $\implies CPI = 0.2236$

(c) The processor implements three branch delay slots. The compiler fills the branch delay slots with three instructions that come before the branch in the original code.

Branch stall is filled with an instruction before branch and processor stops fethching as soon as stall is fethced, therefore 2 cycles are required for branches.

Avg. CPI = (0.40 * 2) + (0.12 * 1) $\implies CPI = 0.92$

(d) The processor does not implement branch delay slots. Instead, it implements a hardware branch predictor that makes correct predictions for 90% of all branches. When an incorrect prediction is discovered, the incorrectly fetched instructions after the branch are squashed.

As hardware predictor makes correct prediction for 90% of branches,

1 clock cycle is required when correct prediction is made and 3 clock cycles are required for incorrect prediction.

Avg. CPI = (0.40 * 0.90 * 1) + (0.40 * 0.1 * 3 + 0.12 * 1) $\implies CPI = 0.6$

5. Assume that you have a 2 bit saturating counter (with the initial value of 11). Every time the branch is taken, the counter is incremented, and is decremented every time the branch is not taken.

(a) Show the change in the counter for the following sequence of 9 events: T,T,N,T,N,T,N,N,N (T stands for taken and N stands for not taken; these are the actual outcomes of the branch).

T	T	N	T	N	T	N	N	N
11	11	10	11	10	11	10	01	00

(b) What will the prediction be for the 10th sequence? Assume the predictions for the 2 bit predictor as discussed in class

At the end of 9th sequence, the counter is 00. As the counter is 00, the prediction is not taken. Therefore, the prediction for 10th sequence is not taken.