# CSE 30321 – Computer Architecture I – Fall 2009

**Homework 07 – Pipelined Processors and Multi-core Programming**

**Assigned:** November 5, 2009 **– Due:** November 12, 2009

This assignment can be done in groups of 1, 2, or 3. The assignment is worth 85 points

## Problem 1: (25 points)

Background:

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 take the following amounts of time to complete:

| IF | ID | EX | MEM | WB |
|---|---|---|---|---|
| 300 ps | 225 ps | 250 ps | 325 ps | 275 ps |

Questions:

Part A: (5 points)

- How long will it take to completely process a single add instruction in a pipelined processor?
- What about in a non-pipelined, multi-cycle processor?
- Comment on the difference in execution times.

Part B: (5 points)

- How long will it take to completely process 100 independent add instructions in a pipelined processor?
- What about in a non-pipelined, multi-cycle processor?
- Comment on the difference in execution times.

Part C: (5 points)
To try to improve performance further, you have decided to break up 2 of the above stages into 2 shorter stages.

- What steps would you recommend breaking up?
- How would this affect your answer to part B?

Note:
The remaining problems in this question assume that instructions executed by the processor are broken down as follows:

| ALU | beq | lw | sw |
|---|---|---|---|
| 30% | 25% | 30% | 15% |

Part D: (5 points)
Assuming there are no stalls or hazards (and ignoring time to fill or drain the pipeline) what percentage of CCs require a reference to memory?

Part E: (5 points)
Assuming there are no stalls or hazards (and ignoring time to fill or drain the pipeline) what percentage of time do instructions need to access the register file in the same CC?

## Problem 2: (25 points)

Background:

In this question, we examine how data dependencies affect execution in the basic 5-stage pipeline discussed in class. Problems in this exercise refer to the following sequence of instructions:

```
sw $10, 0($5)
add $8, $5, $6
lw $9, 0($8)
sub $10, $9, $2
add $20, $21, $22
```

Questions:

Part A: (10 points)

Assume there is full forwarding. Show how the above sequence of instructions would flow through the pipeline. Indicate when data hazards occur.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **sw $10, 0($5)** | | | | | | | | | | | | | | | |
| **add $8, $5, $6** | | | | | | | | | | | | | | | |
| **lw $9, 0($8)** | | | | | | | | | | | | | | | |
| **sub $10, $9, $2** | | | | | | | | | | | | | | | |
| **add $20, $21, $22** | | | | | | | | | | | | | | | |

Note:

- Refer to the following table to answer the remaining 2 questions.

| Without forwarding | With full forwarding |
|---|---|
| 350 ps | 400 ps |

Part B: (10 points)

What is the execution time of this instruction sequence without forwarding and with full forwarding? (A register *may not* be read and written in the same CC.) What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

Part C: (5 points)

Is there anything that you or the compiler could do to further improve performance of the design *with* full forwarding?

## Problem 3: (15 points)

Background:
This problem will help you understand how certain code sequences can degrade the performance of pipelined code.

Question:
Identify all of the data dependencies in the following code. Which dependencies are data hazards that can be resolved via forwarding? Which dependencies are data hazards that will cause a stall?

```
add   $10, $11, $12
lw    $13, 4($10)
sw    $13, 8($10)
subi  $13, $13, 4
sw    $13, 12($10)
```

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **add $10, $11, $12** | | | | | | | | | | | | | | | | |
| **lw $13, 4($10)** | | | | | | | | | | | | | | | | |
| **sw $13, 8($10)** | | | | | | | | | | | | | | | | |
| **subi $13, $13, 4** | | | | | | | | | | | | | | | | |
| **sw $13, 12($10)** | | | | | | | | | | | | | | | | |

## Problem 4: (20 points)

Background:

The two parts of this question will help you see how instructions progress through a pipeline. Moreover, you will see (graphically) how structural enhancements to the pipeline can affect performance.

Questions:

For the sequence of instructions shown below, show how they would progress through the pipeline. For these two questions, assume the following:

- Stalls are indicated by placing the code of the state where the hazard would be discovered in the succeeding square
- We will assume a standard, 5-stage pipeline:
  - IF = Instruction Fetch
  - ID = Instruction Decode
  - EX = Execute
  - M = Memory Access
  - WB = Write Back
- Assume that each stage of the pipeline takes just 1 CC to finish

Part A: (5 points)

- Assume that forwarding **has not** been implemented.
- Assume that you **cannot** read and write a register in the same clock cycle

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw $5, 8($2) | | | | | | | | | | | | | | | | | | | | | | | | |
| add $5, $4, $5 | | | | | | | | | | | | | | | | | | | | | | | | |
| sub $8, $2, $7 | | | | | | | | | | | | | | | | | | | | | | | | |
| or $3, $8 ,$9 | | | | | | | | | | | | | | | | | | | | | | | | |
| sw $9, 8($3) | | | | | | | | | | | | | | | | | | | | | | | | |

Part B: (5 points)

- Assume that forwarding **has** been implemented.
- Assume that you **can** read and write a register in the same clock cycle
-

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw $5, 8($2) | | | | | | | | | | | | | | | | | | | | | | | | |
| add $5, $4, $5 | | | | | | | | | | | | | | | | | | | | | | | | |
| sub $8, $2, $7 | | | | | | | | | | | | | | | | | | | | | | | | |
| or $3, $8 ,$9 | | | | | | | | | | | | | | | | | | | | | | | | |
| sw $9, 8($3) | | | | | | | | | | | | | | | | | | | | | | | | |

Part C: (10 points)

- **NOTE DIFFERENT INSTURUCTION SEQUENCE!**
- Assume that forwarding **has** been implemented.
- Assume that you **can** read and write a register in the same clock cycle
- Assume that a given register "contains its number"
    - E.g. $1 = 1, $2 = 2, $3 = 3, $4 = 4, $5 = 5, $6 = 6, etc.
- You may assume that each branch is "predicted" correctly.

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y: add $5,$1,$1 | | | | | | | | | | | | | | | | | | | | | | | | |
| beq $5,$2,X | | | | | | | | | | | | | | | | | | | | | | | | |
| add $5,$5,$5 | | | | | | | | | | | | | | | | | | | | | | | | |
| add $6,$2,$2 | | | | | | | | | | | | | | | | | | | | | | | | |
| X: beq $4,$6,Y | | | | | | | | | | | | | | | | | | | | | | | | |
| sll $7,$1,4 | | | | | | | | | | | | | | | | | | | | | | | | |

What if branch prediction was **not** perfect? (You don't have to show a pipe trace, just briefly explain what might happen / what you might have to do.)