CS 152
Computer Systems Architecture
Winter 2013

Homework 3

Due Tuesday, February 24th, 2015, EEE 11:59PM

Name	Student ID
Ford Tang	46564602

IMPORTANT NOTES:

1. **No late submissions.**
2. **Please show your work. Remember that bottom line answers without proper explanations are worth ZERO points.**
3. **Remember that you are solely responsible for the answers to the questions, therefore, please refrain from copying from your class peers.**

For Grading Purposes Only:

Q1	Q2	Q3	Q4	Q5	Q6
10	10	10	10	10	10

Q7	Q8	Q9	Q10
10	10	10	10

Total Score
100

Problem 1 (10 points)

True and False Questions. Give reasons for your answers.

(a) Allowing jumps, branches, and ALU instructions to take fewer clock cycles than the five required by the load instruction will increase pipeline performance under all circumstances.

False, if there is a load or store instruction in the pipe, then those instructions will need to finish before jumping or branching.

(b) Trying to allow some instructions to take fewer cycles does not help, since the throughput is determined by the clock cycle. (It is true that the number of pipe stages per instruction affects latency, not throughput.)

True, since more stages will require longer time for the instructions to finish, but once the pipeline is filled, throughput will be every cycle (approximately).

(c) Allowing jumps, branches, and ALU operations to take fewer cycles only helps when no loads or stores are in the pipeline, so the benefits are small.

True, otherwise the load and store instructions will need to finish before jumping or branching.

(d) Since branches and jumps can take fewer cycles, there is some opportunity for improvement in pipelining performance. (You cannot make ALU instructions take fewer cycles because of the write-back of the result.)

True, because there's always room for improvement.

(e) We could improve performance if instead of trying to make instructions take fewer cycles, we explore making the pipeline longer, so that instructions take more cycles, but the cycles are shorter.

True, this will result in a higher output.

Problem 2 (10 points)

Consider the pipelined implementation (without forwarding and/or stalling) of the MIPS microprocessor.

(a) Explain how this pipelined implementation deals with I-type conditional branch instructions in the case the branch is not taken and in the case the branch is taken.
(b) Explain why data dependency hazards may occur in this implementation.
(c) List all possible instruction sequences that may exhibit data dependency hazards.
(d) For each sequence in (c), give a software equivalent that does not suffer from the data dependency limitation.

(a) If branch is not taken, then pipeline continues processing instructions sequentially.
If branch is taken, then program counter is updated accordingly, then the program counter and IF/ID register is stalled for 3 cycles to flush the pipeline to accept the new instruction from the jump.
(b) If there is a store or load instruction before the branch instruction, the new instruction from the branch address may be dependent on the data from the store/load instruction.
(c) ALU -> ALU
ALU -> Store/Load
Store/Load -> ALU
Store/Load -> Store/Load
(d) ALU -> ALU:

```
add $3, $1, $2
add $3, $1, $2
```

ALU -> Store/Load:

```
add $3, $1, $2
SW $3, 0($1)
```

Store/Load -> ALU:

```
SW $3, 0($1)
add $3, $1, $2
```

Store/Load -> Store/Load:

```
LW $3, 0($1)
SW $3, 0($1)
```

Problem 3 (10 points)

Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding?

```
add   $2, $5, $4
add   $5, $2, $4
sw    $5, 100($2)
lw    $4, 0($5)
add   $3, $2, $4
```

Data dependency on $2, $5, $4

Resolved via Forwarding:
add $5, $2, $4
sw $5, 100($2)
lw $4, 0($5)
add $3, $2, $4

Problem 4 (10 points)

Assume a 5 stage pipelined MIPS processor with stages IF, ID, EX, MEM and WB. LOAD and STORE are the only instructions accessing memory. Branches are resolved at ID stage.

(a) Give a code sequence that has data hazard which can be solved by forwarding.
(b) Give a code sequence that has data hazard that cannot be solved by forwarding. Indicate stall cycles required.
(c) Explain branch hazards. Why do branch hazards degrade the performance?

(a) add $3, $1, $2
add $1, $3, $2

(b) add $3, $1, $2
j next ; Stall 3 cycles
next:
add $1, 3, $2

(c) Branch hazards occur if a branch is taken. The next instruction after the branch may require the data from the previous instruction, if that is the case, stalls must occur to insure that the data is written back into the registers ready for the instruction at the branch. This will degrade performance since stalls are necessary.

Problem 5 (10 Points)

Consider the following MIPS code sequence.

```
        add $2, $2, $2
        add $5, $5, $5
L:      lw $8, 1000 ($5)
        sub $5, $2, $8
        addi $2, $2, -4
        beq $2, $0, L
        sw $5, 500 ($2)
```

Assume that there is **no forwarding unit** (including register file forwarding) but instead there is a data hazard detection unit that introduces the stalls needed to avoid data hazards. Suppose the processor uses **Assume Branch Not Taken** strategy and branches are resolved in the ID stage. Illustrate the execution of the given code.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Add	F	D	E	M	W															
Add		F	D	E	M	W														
Lw			F	D			E	M	W											
Sub				F			D			E	M	W								
Addi							F			D	E	M	W							
Beq										F	D			E	M	W				
Sw											F			D	E	M	W			

[Note] F: Fetch, D: Decode, E: Execution, M: Memory Access, W: Register Write Back

Problem 6 (10 points)

Using the same sequence of instructions as in Problem 5, now suppose the situation **beq** will be taken and branches are resolved in the ID stage (Processor uses **Assume Branch Not Taken** strategy). Show the execution of the given code around the loop, starting with the execution of a **beq** instruction and ending with the next execution of a **beq**. **(10 points)**

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Beq	F	D	E	M	W															
Sw		F	D	E	M	W														
Lw				F	D	E	M	W												
Sub					F	D			E	M	W									
Addi						F			D	E	M	W								
Beq									F	D			E	M						

[Note] F: Fetch, D: Decode, E: Execution, M: Memory Access, W: Register Write Back

Problem 7 (10 points)

Using the same instruction sequence as in Problem 5, redraw a diagram that describes the execution of the given code assuming that the processor has a **forwarding unit**.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Add	F	D	E	M	W															
Add		F	D	E	M	W														
Lw			F	D	E	M	W													
Sub				F	D	E	M	W												
Addi					F	D	E	M	W											
Beq						F	D	E	M	W										
Sw							F	D	E	M	W									

[Note] F: Fetch, D: Decode, E: Execution, M: Memory Access, W: Register Write Back

Problem 8 (10 points)

An unpipelined processor has a cycle time of 30ns. What is the cycle time of a pipelined version of the processor with 5 evenly divided pipeline stages if each pipeline latch has a latency of 1ns? What if the processor is divided into 50 pipeline stages? What is the minimum cycle time we can hope to get if we could add as many pipeline stages as desired?

30ns / 5 stages + 1 ns = 7 ns / stage

30ns / 50 stages + 1 ns= 1.6 ns / stage

Minimum cycle time will be 1 ns

Problem 9 (10 points)

Consider the following code segment within a loop body:

```
if ( n % 2 == 0 )  // branch 1
        a++;
if ( n % 10 == 0 )// branch 2
        b++;
```

Assume that the following list of 10 values of n is to be processed by 10 iterations of this loop:
18, 29, 30, 41, 52, 60, 79, 80, 91, 100

List the predictions for the following branch prediction schemes and calculate the prediction accuracies for each scheme.

(a) Always taken.
(b) Always not taken.
(a) 1-bit predictor, initialized to predict taken.
(b) 2-bit predictor, initialized to weakly predict taken.

(a) Prediction: T T | T T | T T | T T | T T | T T | T T | T T | T T | T T
Actual: T F | F F | T T | F F | T F | T T | F F | T T | F F | T T
Accuracy: 50%

(b) Prediction: F F | F F | F F | F F | F F | F F | F F | F F | F F | F F
Actual: T F | F F | T T | F F | T F | T T | F F | T T | F F | T T
Accuracy: 50%

(c) Prediction: T T | F F | F T | T F | F T | F T | T F | F T | T F | F T
Actual: T F | F F | T T | F F | T F | T T | F F | T T | F F | T T
Accuracy: 50%

(d) Prediction: T T | T F | F F | T F | F F | F F | T F | F F | T F | F F
Actual: T F | F F | T T | F F | T F | T T | F F | T T | F F | T T
Accuracy: 30%

Problem 10 (10 points)

Consider a processor with a delayed branch that has three delay slots. Three compilers compiler A, compiler B and compiler C, could run on this processor. Compiler A can fill the first delay slot 60% of the time and the second delay slot 40% of the time and the third delay slot 20% of the time (filling delay slot is independent). Compiler B can fully fill all the three delay slots. Compiler C leaves all the slots empty. Assuming that branches account for 20% of all instructions and arithmetic/logic operations for the remaining 80% of the instructions for any program, what is the improvement of CPI with compiler B compared to CPI with compiler A and compared to CPI with compiler C? Assume that CPI of arithmetic/logic operations is 1.

Compiler A: 4 cycles - 0.6 - 0.4 - 0.2 = 2.8
0.2 x 2.8 + 0.8 x 1 = 1.36

Compiler B: 4 cycles - 1 - 1 - 1 = 1
0.2 x 1 + 0.8 x 1 = 1

Compiler C: 4 cycles = 4
0.2 x 4 + 0.8 x 1 = 1.6

(A-B)/A = (1.36-1)/1.36 = 0.2647058823529412
(A-C)/A = (1.36 - 1.6)/1.36 = -0.1764705882352941