CS 152
Computer Systems Architecture
Winter 2015

Homework 2

Due Tuesday, February 3rd, 2015, EEE 11:59PM PDF

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 consulting with 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	15	15

Total Score
110

Problem 1 (10 points)

Consider two 32-bit floating-point numbers (X and Y) with a sign in bit position 31, a two's complement unbiased 8-bit exponent in positions 30-23 and a two's complement 23-bit mantissa in bits 0 through 22. The mantissa is a fraction less than one and there is no implied 1-bit in the normalized form.

The bit strings representing the 2 numbers are given in hexadecimal notation for convenience:

X = 00C0 0000 Y = 813F FFFF

X = 0 00000001 10000000000000000000000 Y = 1 00000010 01111111111111111111111

a. Calculate X + Y
b. Calculate X * Y

X = -1^0 x 0.10000000000000000000000 x 2^1

Y = -1^1 x 0.11111111111111111111111 x 2^2

X + Y = -1^0 x 1.00000000000000000000000 x 2^2

\+ -1^1 x 0.11111111111111111111111 x 2^2

= -1^1 x 1.11111111111111111111111 x 2^2

= -1^1 x 0.11111111111111111111111 x 2^1

= 10000000111111111111111111111111

X * Y = -1^0 x 1.00000000000000000000000 x 2^2

* -1^1 x 0.11111111111111111111111 x 2^2

= -1^1 x 0.11111111111111111111111 x 2^2

= 10000001001111111111111111111111

Problem 2 (10 points)

Consider a program P with the following mix of operations:

Floating-point add: 25%
Floating-point multiply: 10%
Floating-point divide: 10%
Integer instructions: 55%

To execute this program, also consider two machines M1 and M2 such that M1 has floating-point hardware while M2 does not.

M1 requires the following number of clock cycles for each instruction class:

Floating-point add: 5
Floating-point multiply: 7
Floating-point divide: 19
Integer instructions: 2

For M2, every integer instruction requires 2 clock cycles and we can emulate the floating-point operations using integer instructions. The number of instructions needed to implement each floating-point operation is as follows:

Floating-point add: 15
Floating-point multiply: 30
Floating-point divide: 50

The clock rate for M1 is 1GHz, and for M2 is 800 MHz.

Find which machine is faster in executing program P.

if P was 100 operations:

M1:	Floating-point add:	25 x 5 = 125
	Floating-point multiply:	10 x 7 = 70
	Floating-point divide:	10 x 19 = 190
	Integer instructions:	55 x 2 = 110
	Total cycles:	495
M2:	Floating-point add:	25 x 15 x 2 = 750
	Floating-point multiply:	10 x 30 x 2 = 600
	Floating-point divide:	10 x 50 x 2 = 1000
	Integer instructions:	55 x 2 = 110
	Total cycles:	2460

M1 is faster

Problem 3 (10 points)

Referring to figures 1 and 2 at the end of this assignment, single and multi-cycle implementations of a MIPS processor, describe what is done for each one of the 5 types of MIPS instructions defined by the 3 instructions formats (Namely R, I, J) using the Register Transfer Language (RTL). Please be sure to define each instruction format and write your RTL description so that it does correspond to your given formats.

R:
```
IR = Memory[PC];
PC = PC + 4;
RS = Reg[IR[25-21]];
RT = Reg[IR[20-16]];
RD = Reg[IR[15-11]];
ALUOut = RS op RT;
RD = ALUout;
```

I:
```
immediate:
IR = Memory[PC];
PC = PC + 4;
RS = Reg[IR[25-21]];
RT = Reg[IR[20-16]];
ALUOut = RS op RT;
RT = ALUOut;
conditional:
IR = Memory[PC];
PC = PC + 4;
RS = Reg[IR[25-21]];
RT = Reg[IR[20-16]];
ALUOut = PC + sign-extend(IR[15-0]);
if (RS - RT)
        PC = ALUout;
memory:
IR = Memory[PC];
PC = PC + 4;
RS = Reg[IR[25-21]];
RT = Reg[IR[20-16]];
ALUOut = RT + sign-extend(IR[15-0])
if (IR[31-26] == load)
        RS = Memory[ALUOut];
else if (IR[31-26] == store)
        Memory[ALUOut] = RS;
```

J:
```
IR = Memory[PC];
PC = PC + 4;
ALUOut = PC + sign-extend(IR[25-0]);
PC = ALUOut;
```

Problem 4 (10 points)

In estimating the performance of the single-cycle implementation, assume that only the major functional units have delays (assume the delay of multiplexers, control unit, PC access, etc. are negligible).

The operational timing for each of the major functional units are:

Memory units: 300 ps
ALU and Adders: 100 ps
Register file (read or write): 200 ps

Where ps stands for picoseconds (1 ps = 1 x 10^{-12} seconds)

For the single cycle implementation and for each of the 5 instruction classes, identify critical paths in the implementation and determine the time taken to complete each instruction.

R: Memory + Register + ALU = 300 + 200 + 100 = 600 ps
Load: Memory + Register + ALU + Memory = 300 + 200 + 100 + 300 = 900 ps
Store: Memory + Register + ALU + Memory = 300 + 200 + 100 + 300 = 900 ps
Branch: Memory + Register + ALU = 300 + 200 + 100 = 600 ps
J: Memory + ALU = 300 + 100 = 400 ps

Problem 5 (10 points)

Using the same operational timing from Problem 4 and incorporating the instruction mix:

Loads: 25%
Stores: 5%
ALU: 60%
Branches: 6%
Jumps: 4%

a. Consider a clock length where every instruction completes in one clock cycle, and the clock cycle is the same for all instructions. What is the clock cycle length for this set of instructions?
b. Now consider a variable length clock implementation, where each instruction executes in 1 clock cycle, but the clock cycle length is determined by the instruction. What is its average clock cycle length?
c. Find the performance ratio of the variable clock to the single clock.

a. 900 ps
b. .25 x 900 + .05 x 900 + .6 x 600 + .06 x 600 + .04 x 400 = 225 + 45 + 360 + 36 + 16 = 682 ps
c. 682/900 = 0.757778

Problem 6 (10 points)

Assume a multi-cycle architecture as shown in figure 2 and the same instruction mix from Problem 5, answer the following questions.

a. Determine the number of clock cycles it will take to execute a Load, Store, ALU operation, Branch, and Jump instruction. (Hint: You can also think of it as the number of steps it takes to execute each instruction)
b. Using results from part a, determine the CPI for this implementation and instruction mix?

a. Load = 5 cycles
Store = 4 cycles
ALU = 4 cycles
Branch = 3 cycles
Jump = 3 cycles

b. .25 x 5 + .05 x 4 + .6 x 4 + .06 x 3 + .04 x 3 = 1.25 + 0.2 + 2.4 + 0.18 + 0.12 = 4.15 CPI

Problem 7 (10 points)

In estimating the performance of the single cycle MIPS implementation, we assumed that only the major functional units had any delay (i.e., the delay of the multiplexers, the control units, PC access, sign extension unit, and wires are negligible).

Assume that the actual time taken by each component in the architecture is as follows: Instruction Memory fetch (200 ps), Register read (100 ps), ALU operation (200 ps), Data access (300 ps) and Register write (100 ps) where ps stands for pico-seconds (1 ps = 1 x 10^{-12} seconds).

Assume that we use different types of adders to implement addition for different functions. The delays of each type of adder are specified as follows:

ALU: 200 ps
Adder for PC+4: X ps
Adder for branch address computation: Y ps

Answer the following questions:

(a) What would the cycle time be if X=300 and Y=300?
(b) What would the cycle time be if X=500 and Y=500?
(c) What would the cycle time be if X=100 and Y=800?

a. Load = (X or IM) + register + ALU + Data = 300 + 100 + 200 + 300 = 900 ps
   Branch = (X or IM) + register + Y = 300 + 100 + 300 = 700 ps
   Cycle time is 900 ps
b. Load = (X or IM) + register + ALU + Data = 500 + 100 + 200 + 300 = 1100 ps
   Branch = (X or IM) + register + Y = 500 + 100 + 500 = 1100 ps
   Cycle time is 1100 ps
c. Load = (X or IM) + register + ALU + Data = 200 + 100 + 200 + 300 = 800 ps
   Branch = (X or IM) + register + Y = 200 + 100 + 800 = 1100 ps
   Cycle time is 1100 ps

Problem 8 (10 points)

Consider the following MIPS code:

```
Loop:  addi $t0, $t0, 1
       beq $t0, $s1, Exit
       add $t0, $t0, $t0
       add $t0, $t0, $t0
       add $t1, $a0, $t0
       lw  $t2, 0($t1)
       add $s0, $t2, $s0
       j   Loop
Exit:
```

Assume that the loop goes to label *Exit* in the 100th iteration after running completely for 99 times.

The following delays are:

Instructional and Data memories: 1.5 ns
Register File: 1ns
All other components: 0.5ns

Calculate the total time taken to execute the above program on both single cycle and multi-cycle implementations.

Single Cycle:
Line1 = PC + Instruction + Register + ALU = 0.5 + 1.5 + 1 + 0.5 = 3.5 ns
Line2 = PC + instruction + Register + MUX + ALU + MUX = 0.5 + 1.5 + 1 + 0.5 + 0.5 + 0.5 = 4.5 ns
Line3 = PC + Instruction + Register + ALU = 0.5 + 1.5 + 1 + 0.5 = 3.5 ns
Line4 = PC + Instruction + Register + ALU = 0.5 + 1.5 + 1 + 0.5 = 3.5 ns
Line5 = PC + Instruction + Register + ALU = 0.5 + 1.5 + 1 + 0.5 = 3.5 ns
Line6 = PC + Instruction + Register + MUX + ALU + Data + MUX = 0.5 + 1.5 + 1 + 0.5 + 0.5 + 1.5 + 0.5 = 5 ns
Line7 = PC + Instruction + Register + ALU = 0.5 + 1.5 + 1 + 0.5 = 3.5 ns
Line8 = PC + Instruction + Sign Extend + ALU + MUX = 0.5 + 1.5 + 0.5 + 0.5 + 0.5 = 3.5 ns
100 x (Line1 + Line2) + 99 x (Line3 + Line4 + Line5 + Line6 + Line7 + Line8) = 100 x 7 + 99 x 22.5 = 2927.5 ns
Multi-Cycle:
Line1 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU + Register + MUX = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 + 1 + 0.5 = 8 ns
Line2 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 = 6.5 ns
Line3 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU + Register + MUX = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 + 1 + 0.5 = 8 ns
Line4 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU + Register + MUX = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 + 1 + 0.5 = 8 ns
Line5 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU + Register + MUX = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 + 1 + 0.5 = 8 ns
Line6 = PC + MUX + Instruction + Register + MUX + Register + Register + Register + MUX + ALU + Register + Mux = 0.5 + 0.5 + 1.5 + 1 + 0.5 + 1 + 1 + 1 + 0.5 + 0.5 + 1 + 0.5 = 9.5 ns
Line7 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU + Register + MUX = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 + 1 + 0.5 = 8 ns
Line8 = PC + MUX + Instruction + Register + Register + Register + MUX + ALU + MUX = 0.5 + 0.5 + 1.5 + 1 + 1 + 1 + 0.5 + 0.5 + 0.5 = 7 ns
100 x (Line1 + Line2) + 99 x (Line3 + Line4 + Line5 + Line6 + Line7 + Line8) = 100 x 14.5 + 99 x 48.5 = 6251.5 ns

Problem 9 (15 points)

Consider a microcode controller with 5 control words.

a) How many bits are needed to address each control word?
b) What is the maximum number of bits allowed in each control word?
c) Can the same control function be performed by a combinational random logic circuit? If so, suggest a method to design such a combinational logic circuit.

a. 3 bits
b. 16 bits
c. Yes, you can use a large MUX.

Problem 10 (15 points)

Consider the multi-cycle implementation of the MIPS data path and the RTL description for instruction execution.

a. Derive the state transition diagram of a sequential machine capable of sequencing the execution of each of the MIPS instruction types.
b. Consider now a micro-programmed implementation for the sequential control of the multi-cycle implementation of the RISC data path. Explain how this approach can be used to implement the sequential machine in (**a**). (Hint: show that this control can sequence through the states of the state transition diagram that defines the multi-cycle approach).

a.

(start) Instruction Fetch -> Instruction Decode -> Memory Address -> Memory Access -> Memory Write -> start

(start) Instruction Fetch -> Instruction Decode -> Memory Address -> Memory Access -> start

(start) Instruction Fetch -> Instruction Decode -> Execution -> Completion -> start

(start) Instruction Fetch -> Instruction Decode -> Branch -> start

(start) Instruction Fetch -> Instruction Decode -> Jump -> start

b.

Using microprogramming, we can encode the current state and inputs to control the State machine.

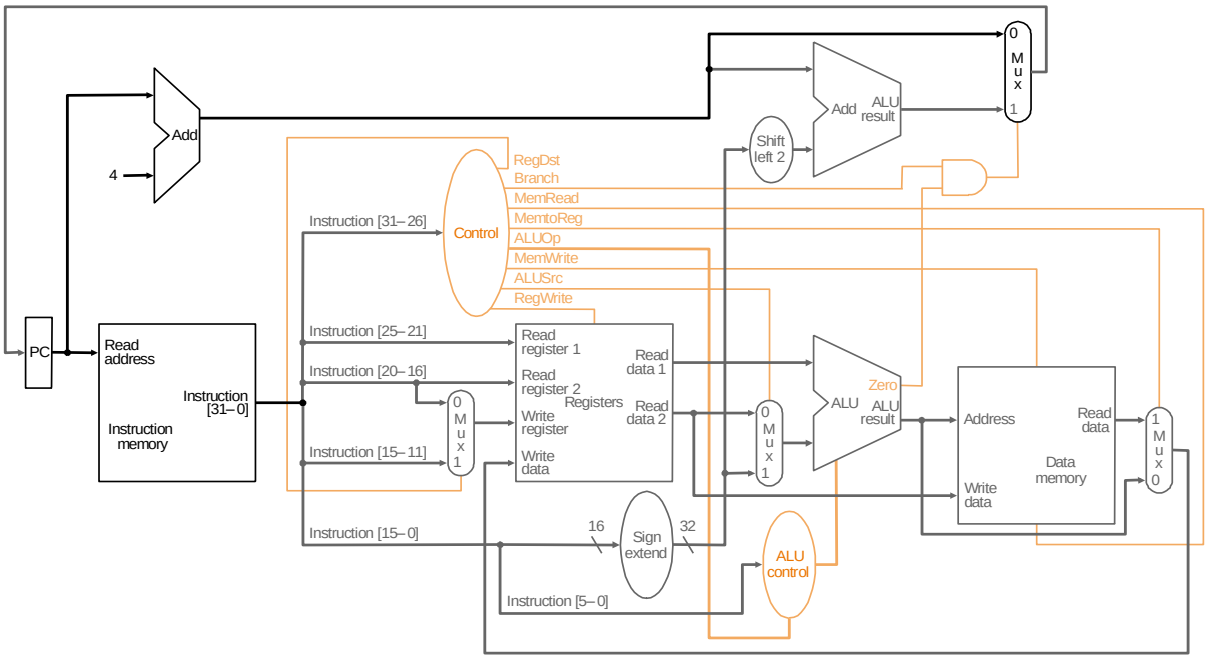

Figure 1: The MIPS processor single cycle implementation.

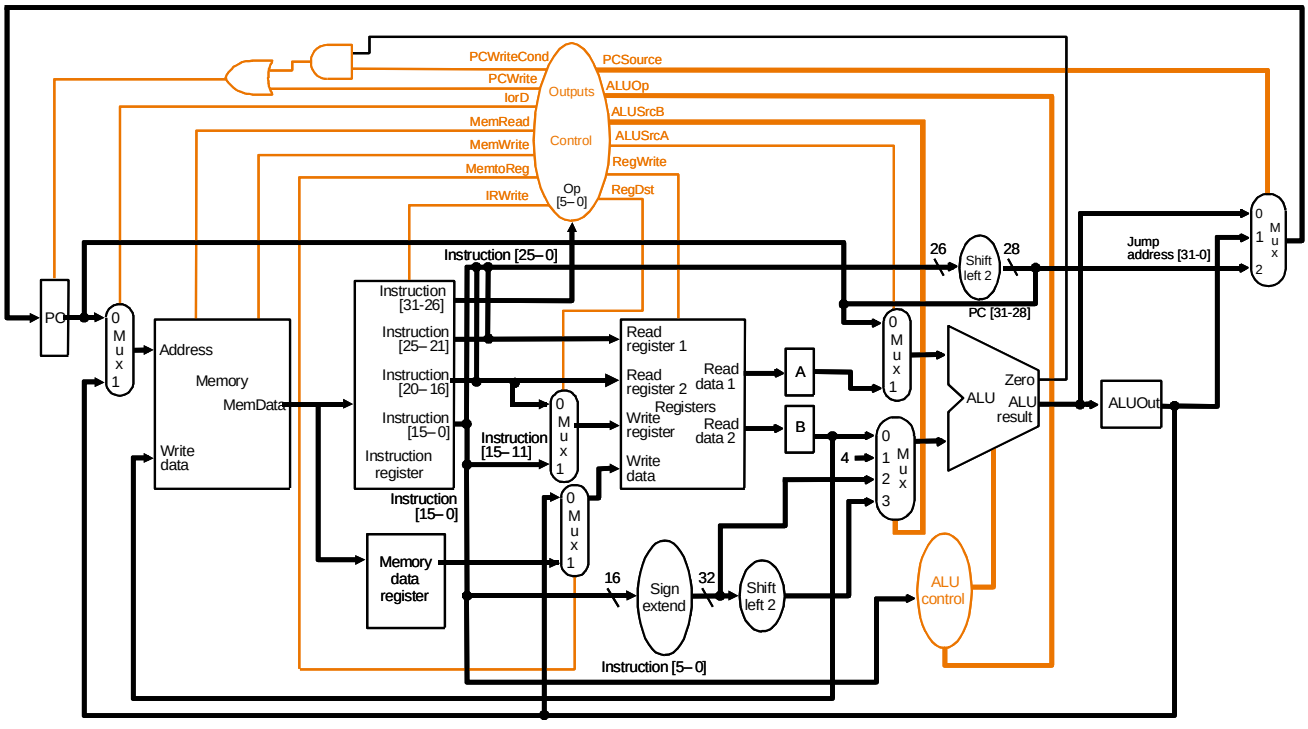

Figure 2: The MIPS processor multi-cycle implementation.