CSE 240A: Principles of Computer Architecture

Midterm Examination 1

Spring 2019

Name: Gauri Iyer

PID: A12786362

Email: gaiyer@ucsd.edu

1	25	
2	25	
3	25	
4	25	
Total	100	
Bonus	10	

This exam is open book and open notes. Personal calculators *are* allowed. Show your work and insert your answer in the space(s) provided. **Please provide details on how you reach a result.**

The exam totals 100 points. This midterm exam counts for 10% of your course grade. The bonus question counts for extra points beyond the 10% (the total grade of the class might exceed 100%).

(25 points) Q1. Computer A uses the MIPS ISA and has a 2 GHz clock frequency. Computer B uses the x86 ISA and has a 3 GHz clock frequency. On average, MIPS programs execute 1.5 times as many instructions as x86 programs. For program P1, computer A has a CPI of 2 and computer B has a CPI of 3. Which computer has faster execution time? What is the speedup?

Computer	ISA	Clk	CPI	# Instr
A	MIPS	2 GHz	2	1.5×
B	x86	3 GHz	3	1×

Execution time = # Instr * CPI * Clk

A = 1.5 * 2 * 2 = 6

B = 1 * 3 * 3 = 9

Computer A has a faster execution time.

Speedup = 9 / 6 = 1.5

(25 points) Q2. Assume that a typical program has the following instruction type breakdown. Assume the processor that this program will be running on has the following instruction latencies.

Instruction	Instr. Frequency	Latency (Cycles)
load	30%	4
store	10%	4
add	50%	2
multiply	8%	16
divide	2%	50

If you could pick one type of instruction to make twice as fast (half the latency) in the next-generation of this processor, which instruction type would you pick? Why?

load: .3 * 4 = .12
store: .1 * 4 = .04
add: .5 * 2 = 1
multiply: .08 * 16 = 1.28
divide: .02 * 50 = 1
base CPI = 3.44

I would pick the multiply instruction because the greatest improvement in performance will occur when the most costly instruction, multiply, is modified to be twice as fast. The base CPI is 3.44. Making multiplication twice as fast results in a new CPI of 2.8, which is 1.23x faster.

(25 points) Q3. Given the following instruction categories and execution latencies (assuming a non-pipelined CPU):

Instruction	Latency (Cycles)
add (addu, addiu)	4
load (lw)	10
multiply (mul)	20
branch (bne)	8

Given the following assembly code:

```
L3:  addu R7, R4, R3
     lw R7, (R7)
     addu R8, R5, R3
     lw R8, (R8)
     mul R7, R7, R8
     addu R2, R2, R7
     addiu R3, R3, #4
     bne R3, R6, L3
```

where registers are written as R(reg number), for example R1.

Answer following questions:

1) What is the CPI of the loop for one iteration (the bne instruction included)?

CPI = (add + load + add + load + mul + add + add + bne) / 8
= (4(add) + 2(load) + mul + bne) / 8
= (4(4) + 2(10) + 20 + 8) / 8
= 64 / 8
= 8

2) Which of the following optimizations would produce the biggest CPI improvement for one iteration (the bne instruction included)?

- Implement prefetching to reduce the latency of loads from 10 cycles to 7 cycles.
- Implement branch prediction to reduce the latency of branch instructions from 8 cycles to 2 cycles.

Option 1:

CPI = (4(add) + 2(load) + mul + bne) / 8

= (4(4) + 2(7) + 20 + 8) / 8

= 58 / 8

= 7.25

Option 2:

CPI = (4(add) + 2(load) + mul + bne) / 8

= (4(4) + 2(10) + 20 + 2) / 8

= 58 / 8

= 7.25

Both optimizations result in the same new CPI: 7.25. They would be produce equal improvements in performance.

(25 points) Q4. Classify the following attributes of a machine as either a property of its microarchitecture or ISA:

1) The machine does not have a subtract instruction.
2) The ALU of the machine does not have a subtract unit.
3) The machine does not have condition codes.
4) A 5-bit immediate can be specified in an ADD instruction.
5) It takes n cycles to execute an ADD instruction.
6) There are 8 general purpose registers.
7) A 2-to-1 mux feeds one of the inputs to ALU.
8) The register file has one input and two output ports

1) ISA
2) microarchitecture
3) ISA
4) ISA
5) microarchitecture
6) ISA
7) microarchitecture
8) microarchitecture

(10 points) Bonus Question. Consider the following high-level language code segments:

(a)
```
uint8_t a[100]; // a is allocated in memory
for (i = 0; i < 100; i ++){
     a[i] = 10;
}
```

(b)
```
int a[100]; // a is allocated in memory
for (i = 0; i < 100; i ++){
     a[i] = 10;
}
```

(c)
```
int *p; // p is allocated in memory
p = (int *)malloc(sizeof(int));
*p = 100;
```

Assume that in the first two code segments, a register contains the address of the start of the array, and in the last snippet, a register contains the address of the pointer p. Note that the "auto-increment" mode increments the base register by *one byte* a time. For each of the above three high-level language code segments, which of the addressing modes listed in the slides, do you think, would lead to the minimum number of instructions?
(Note: No addressing mode might fit perfectly. You might require other instructions for address computation)

(a) auto-increment

(b) index-base

(c) memory-indirect