CMPUT 229 - Computer Organization and Architecture I
Midterm Exam (part #1)— Fall 2012 - Section A1

Prof. José Nelson Amaral
Computing Science Department
University of Alberta

Name:

CMPUT 229 Honor Code

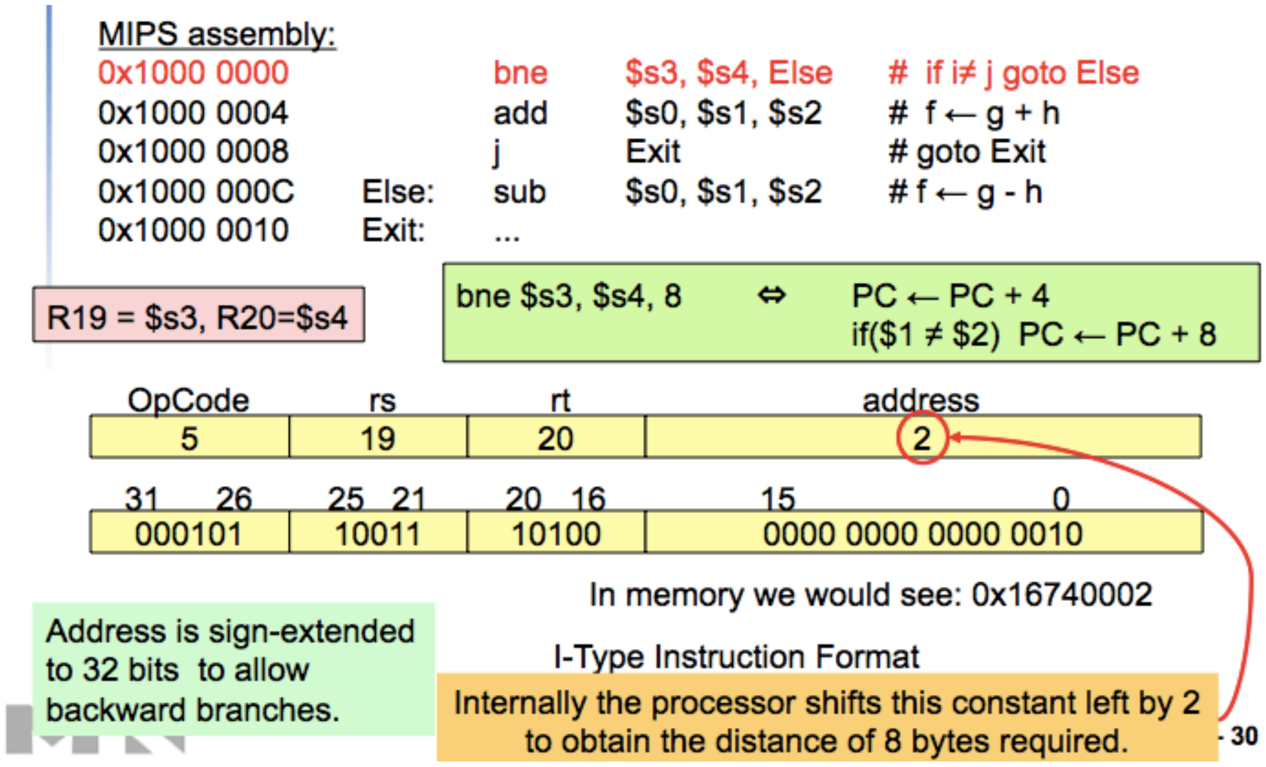
By turning in the exam solution for grading, I certify that I have worked all the solutions on my own, that I have not copied or transcribed solutions from a classmate, someone outside the class, or from any other source. I also certify that I have not facilitated or allowed any of my classmates to copy my own solutions. I am aware that the violation of this honour code constitutes a breach of the trust granted me by the teaching staff, compromises my reputation, and subjects me to the penalties prescribed in Section 26.1 of the University of Alberta 2012/2013 Calendar.

Edmonton, October 19, 2012.

Question 1 (10 points):

The figure below displays the slide used in class to explain the format of a branch-not-equal instruction. The table below lists two branch instructions that were fetched by a processor. It shows the memory address from which the instruction was fetched and the hexadecimal representation of the fetched instruction. For each instruction, indicate the address of the next instruction executed in case of a branch-taken and in case of a branch-not-taken outcome.

Fetching Address	Fetched Instruction	Address of Next Instruction Executed	
		Branch Not Taken	Branch Taken
`0x2000 4FDC`	`0x16F9 FFFC`	`0x2000 4FE0`	`0x2000 4FD0`
`0x2000 2000`	`0x1410 00CE`	`0x2000 2004`	`0x2000 233C`



MIPS assembly:

```
0x1000 0000          bne   $s3, $s4, Else   # if i≠ j goto Else
0x1000 0004          add   $s0, $s1, $s2    # f ← g + h
0x1000 0008          j     Exit             # goto Exit
0x1000 000C   Else:  sub   $s0, $s1, $s2    # f ← g - h
0x1000 0010   Exit:  ...
```

R19 = $s3, R20=$s4

bne $s3, $s4, 8 ⇔ PC ← PC + 4
if($1 ≠ $2) PC ← PC + 8

OpCode	rs	rt	address
5	19	20	2
31 26	25 21	20 16	15 0
000101	10011	10100	0000 0000 0000 0010

In memory we would see: 0x16740002

I-Type Instruction Format

Address is sign-extended to 32 bits to allow backward branches.

Internally the processor shifts this constant left by 2 to obtain the distance of 8 bytes required.

address	=	`0x00CE`
after shift	=	`0x0338`
after sign extension	=	`0x0000 0338`
Original PC	=	`0x2000 2000`
	+	`4`
	+	`0x0000 0338`
Target Address	=	`0x2000 233C`

address	=	`0xFFFC`
after shift	=	`0xFFF0`
after sign extension	=	`0xFFFF FFF0`
Original PC	=	`0x2000 4FDC`
	+	`4`
	+	`0xFFFF FFF0`
Target Address	=	`0x2000 4FD0`

Question 2 (20 points): You have been hired by a very important supplier of computer servers for Internet Service Providers (ISPs). The CPI of the most popular program run by ISPs in this machine is dominated by a class of instructions C. Instructions belonging to class C are executed 40% of the time and, on average, each instruction of class C takes 3 cycles to execute. For the remaining 60% of the instructions in this application the CPI is 0.5 cycles per instruction. Lets call the execution time of this original machine T_{orig}.

A processor-architecture team in your company has proposed a design change to the processor that will reduce the average number of clocks executed by C instructions to 2 clock cycles, but will increase the clock cycle by 20%. Lets call the execution time of this machine T_{proc}.

The head of your compiler team proposed that, with a larger development team, they could implement compiler optimizations that would reduce the number of C instructions executed by 60% without affecting the number of execution of instructions of other classes. Lets call the execution time of this machine T_{comp}

Note: You do not need the instruction count or the clock cycle of the original machine to solve the problem, but if you wish to have these numbers you can assume that the popular program executed 10000 instructions and that the clock cycle in the original machine was 100 *ps*.

Given budget constraints, you only can afford to approve one of these changes.

a. **(8 points)** Is the machine proposed by the processor-architecture team faster or slower than the original machine? By how much?

Let:

I_X:	number of instructions executed in machine X
T_X:	execution time in machine X
C_X:	Clock cycle in machine X
CPI_X:	CPI in machine X

$X = orig$ for original machine;
$X = proc$ for the machine where the processor is modified and
$X = comp$ for the machine where the compiler is modified.

$$T_X = I_X \times CPI_X \times C_X \quad (1)$$

Because of the wording in the question, there are two possible interpretations to this question:

- **V1:** 40% of the instructions executed in the original machine are of class C with a CPI of 3 $\frac{cycles}{instruction}$, and 60% of the instructions executed are of other types with a CPI of 0.5 $\frac{cycles}{instruction}$.
- **V2:** Instructions of class C execute 40% of the time in the original machine and instructions of other classes execute 60% of the time in the original machine.

For **V1:**

$$CPI_{orig} = 0.6 \times 0.5 + 0.4 \times 3 = 1.5\frac{\text{clocks}}{\text{instruction}} \quad (2)$$

$$CPI_{proc} = 0.6 \times 0.5 + 0.4 \times 2 = 1.1\frac{\text{clocks}}{\text{instruction}} \quad (3)$$

$$T_{orig} = I_{orig} \times CPI_{orig} \times C_{orig} \quad (4)$$
$$T_{orig} = 1.5 \times I_{orig} \times C_{orig} \quad (5)$$

$$T_{proc} = I_{proc} \times CPI_{proc} \times C_{proc} \quad (6)$$
$$T_{proc} = I_{orig} \times 1.1 \times 1.2 \times C_{orig} \quad (7)$$
$$= 1.32 \times I_{orig} \times C_{orig} \quad (8)$$

Therefore the machine proposed by the processor-architecture team is $\frac{1.5}{1.32} = 1.14$ times faster than the original machine.

For **V2:**

$$T_{proc} = 1.2 \times (0.4 \times \frac{2}{3} + 0.6) \times T_{orig} \quad (9)$$
$$= 0.96 \times T_{orig} \quad (10)$$
$$(11)$$

Therefore the machine proposed by the processor-architecture team is $\frac{1.0}{0.96} = 1.04$ times faster than the original machine.

b. **(8 points)** Is the machine proposed by the compiler team faster or slower than the original machine? By how much?

For **V1**:

Assuming that 10000 instructions were executed in the original machine. Therefore there were 4000 instructions of class C and 6000 instructions from other classes. The compiler reduced the number of instructions of class C by 60%, therefore now there are $0.4 \times 4000 = 1600$ instructions of the class C. The CPI for non-class C instructions was 0.5 $\frac{\text{clocks}}{\text{instruction}}$ and that value does not change. Therefore:

$$CPI_{comp} = \frac{\#\text{ clock cycles}}{\text{Instruction Count}} = \frac{6000 \times 0.5 + 1600 \times 3}{6000 + 1600} = 1.026 \frac{\text{clocks}}{\text{instruction}} \quad (12)$$
$$T_{comp} = (0.6 + 0.4 \times 0.4) \times I_{orig} \times CPI_{comp} \times C_{orig} \quad (13)$$
$$T_{comp} = 0.760 \times I_{orig} \times 1.026 \times C_{orig} \quad (14)$$
$$T_{comp} = 0.78 \times I_{orig} \times C_{orig} \quad (15)$$

The design proposed by the compiler team is $\frac{1.5}{0.78} = 1.92$ times faster than the original machine.

For **V2:**

$$T_{comp} = (0.4 \times 0.4 + 0.6) \times T_{orig} \quad (16)$$
$$= 0.76 \times T_{orig} \quad (17)$$

The design proposed by the compiler team is $\frac{1.0}{0.76} = 1.3$ times faster than the original machine.

c. **(4 points)** How much faster is the better design compared with the other proposal?

For **V1**: The design proposed by the compiler team is $\frac{1.32}{0.78} = 1.7$ times faster than the design proposed by the processor-architecture team.

For **V2**: The design proposed by the compiler team is $\frac{0.96}{0.76} = 1.26$ times faster than the design proposed by the processor-architecture team.

Question 3 (20 points): Assume that `p`, `q`, `i`, `j` are integers and that their values are stored in `$s0`, `$s1`, `$s2` and `$s3`, respectively. Assume that `A` and `B` are vectors of integers. Assume that `r` and `s` are pointers declared as:

```
int *r;
int **s;
```

Assume that `r`, `s`, the base address of array `A`, and the base address of array `B` are all in the stack frame of the current function.

In the table below, indicate the minimum number of `load word (lw)` and `store word (sw)` instructions are necessary to execute each of the C statements.

C statement	**load words (lw)**	**store words (sw)** assembly code	

`p = **s;`	3	0	`lw $t0 ??($fp) # s`
			`lw $t1 0($t0)`
			`lw $s0 0($t1)`

`p = q*q;`	0	0	`mul $s0 $s1 $s1`

`q = *r;`	2	0	`lw $t0 ??($fp) # r`
			`lw $s1 0($t0)`

`A[i] = p+q;`	1	1	`add $t0 $s0 $s1`
			`lw $t1 ??($fp) # A`
			`sll $t2 $s2 2`
			`add $t3 $t1 $t2`
			`sw $t0 0($t3)`

`**s = *r;`	4	1	`lw $t0 ??($fp) # r`
			`lw $t1 0($t0)`
			`lw $t2 ??($fp) # s`
			`lw $t3 0($t2)`
			`sw $t1 0($t3)`

C statement	**load words (lw)**	**store words (sw)**	assembly code

`B[i] = A[j];`	3	1	`sll $t0 $s3 2`
			`lw $t1 ??($fp) # A`
			`add $t2 $t1 $t0`
			`lw $t3 0($t2)`
			`lw $t4 ??($fp) # B`
			`sll $t5 $s2 2`
			`add $t6 $t4 $t5`
			`sw $t3 0($t6)`

`p = A[i] + A[j];`	3	0	`sll $t0 $s2 2`
			`lw $t1 ??($fp) # A`
			`add $t2 $t1 $t0`
			`lw $t3 0($t2)`
			`sll $t4 $s3 2`
			`add $t5 $t1 $t4`
			`lw $t6 0($t5)`
			`add $s0, $$t3 $t6`

`A[B[j]] = p+q;`	3	1	`sll $t0 $s3 2`
			`lw $t1 ??($fp) # B`
			`add $t2 $t1 $t0`
			`lw $t3 0($t2)`
			`sll $t4 $t3 2`
			`lw $t5 ??($fp) # A`
			`add $t6 $t4 $t5`
			`add $t7 $s0 $s1`
			`sw $t7 0($t6)`

`p = q + A[B[j]];`	4	0	`sll $t0 $s3 2`
			`lw $t1 ??($fp) # B`
			`add $t2 $t1 $t0`
			`lw $t3 0($t2)`
			`sll $t4 $t3 2`
			`lw $t5 ??($fp) # A`
			`add $t6 $t4 $t5`
			`lw $t7 0($t6)`
			`add $s0 $s1 $t7`