COMP 230: Computer Architecture and Organization September 29, 2017

Exam 1

Instructions:

- This exam is open book and notes. However, you are not allowed to use laptops, cell phones, calculators, or other electronics.
- If you do not show your work, do not expect partial credit for incorrect answers.
- If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution.
- In all cases, clearly state any assumptions you make in your answers.

Name	

Question	**Points Possible**	**Grade**
1	30	
2	18	
3	20	
4	32	
Total	100	

1. Your company's program consists of 1 billion (1×10^9) instructions: 40% are integer arithmetic, 30% are conditionals (branches), and 30% are memory accesses. Your processor runs at 4 GHz and the manufacturer has given you the following information about the instructions:

Instruction Type	CPI for the processor
Integer Math	6.0
Conditionals	2.0
Memory Access	10.0

(a) What is the average CPI of your program?

$$\text{CPI} = 0.5(5) + 0.2(2.5) + 0.3(10) = 2.5 + 0.5 + 3 = \boxed{6\,\text{cycle/s}}$$

(b) What is the CPU execution time?

$$\begin{aligned}\text{Total Execution Time} &= \frac{1 \times 10^9\,\text{instruction}}{\text{program}} \times \frac{6\,\text{cycle}}{\text{instruction}} \times \frac{1\,\text{s}}{4 \times 10^9\,\text{cycle}} \\ &= \frac{6}{4}\text{s} = \boxed{1.5\,\text{s}}\end{aligned}$$

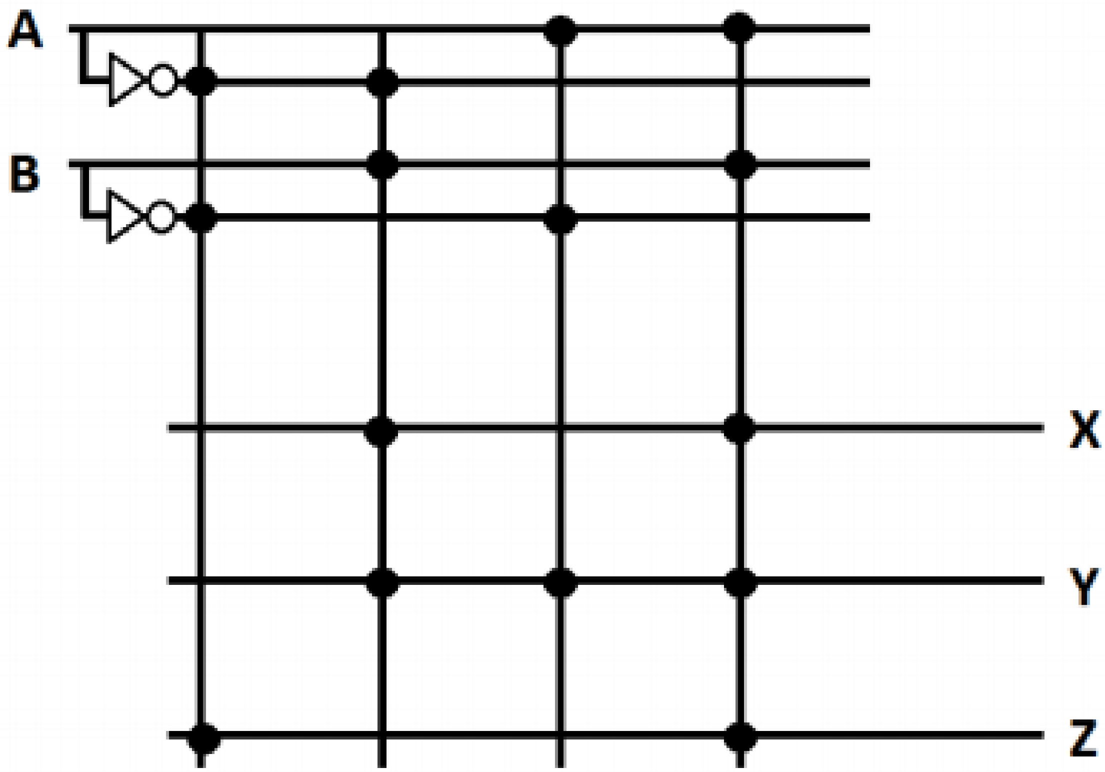
(c) How much total time is spent on conditional instructions?

$$\begin{aligned}\text{Total Execution Time} &= \frac{0.3 \times 1 \times 10^9\,\text{instruction}}{\text{program}} \times \frac{2\,\text{cycle}}{\text{instruction}} \times \frac{1\,\text{s}}{4 \times 10^9\,\text{cycle}} \\ &= \frac{0.6}{4}\text{s} = \boxed{0.15\,\text{s}}\end{aligned}$$

(d) The CPU manufacturer claims they have a new architecture feature called a "branch predictor" which improves the CPI for conditionals by a factor of 5. What is the improved execution time for this new processor?

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}} = \frac{0.15\,\text{s}}{5} + 1.35\,\text{s} = 0.03 + 1.35\text{s} = \boxed{1.38\,\text{s}}$$

2. Consider the following PLA:



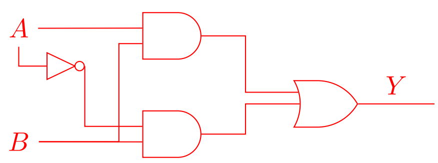
(a) Draw a truth table for the inputs A and B and the outputs X, Y, and Z.

A	B	X	Y	Z
0	0	0	0	1
0	1	1	1	0
1	0	0	1	0
1	1	1	1	1

(b) Write a logic function for X.

$X = (\overline{A} \cdot B) + (A \cdot B)$

(c) Draw a wire (gate) diagram for X.



3. Consider the hexadecimal instruction `014f 802a`.

(a) Convert the instruction into (binary) machine code.

0	1	4	f	8	0	2	a
0000	0001	0100	1111	1000	0000	0010	1010

`op` is $000000 = 0$, so this is I-format. `funct` is $101010 = 42 = 2a_{16}$, so this is `slt`.

op	rs	rt	rd	shamt	funct
000000	01010	01111	10000	00000	101010
0	10 = `$t2`	15 = `$t7`	16 = `$s0`	0	$42 = 2a_{16}$

(b) Convert the instruction into MIPS assembly.

`slt $s0, $t2, $t7`

4. Consider the following C code that operates on an array of 32-bit integers:

```
int calc(int size, int data[]) {
  for (int i{0}; i < size; i++) {
    data[i] = foo(i) * 8;
  }
  return data[size - 1];
}
```

The procedure `foo` is just a helper procedure — don't worry about what it does.

(a) Complete the procedure prologue code:

calc:	addi	`$sp`,	`$sp`,	-16	# Reserve stack space
	sw	`$s0`,	0(`$sp`)		# Use to store `size`
	sw	`$s1`,	4(`$sp`)		# Store addr. of `data`
	sw	`$s2`,	8(`$sp`)		# Store `i`
	sw	`$ra`,	12(`$sp`)		# Something else we need to save...

(b) Complete the procedure body code:

```
        add    $s0,    $a0,    $zero    # Move size to $s0
        add    $s1,    $a1,    $zero    # Move addr. of data to $s1
        add    $s2,    $zero,  $zero    # Initialize i for loop
loop:   slt    $t0,    $s2,    $s0
        beq    $t0,    $zero,  exit
        add    $a0,    $s2,    $zero    # Setup call to foo
        jal    foo                      # Call foo
        sll    $v0,    $v0,    3        # Multiply by 8
        sll    $t0,    $s2,    2
        add    $t0,    $t0,    $s1
        sw     $v0,    $t0              # Store to data[i]
        addi   $s2,    $s2,    1        # Loop increment
        j      loop
exit:   addi   $t0,    $s0,    -1       # Setup for data[size - 1]
        sll    $t0,    $t0,    2
        add    $t0,    $t0,    $s1
        lw     $v0,    0($t0)           # Set result of calc
```