CS2100 Computer Organisation
AY2021/22 Semester I
Assignment 1

PLEASE READ THE FOLLOWING INSTRUCTIONS VERY CAREFULLY.

1. **DEADLINE:**
   The deadline for this assignment is **WEDNESDAY 15 SEPTEMBER 2021, 1 PM**. The submission folder will close shortly after 1 pm and no submissions will be allowed after that. You WILL receive 0 if you do not submit on time.

2. You should complete this assignment on your own without discussing with anyone. If you have been found to have cheated in this assignment, you will receive an F grade for this module as well as face other disciplinary sanctions. Take this warning seriously.

3. Please submit as **ONE SINGLE PDF FILE** called **AxxxxxxY.pdf**, where **AxxxxxxY** is your student ID (sometimes called "matric number). (The other number that begins with 'e' – example, e0123456, is your NUSNET-id and not your student ID.)

4. Ensure that your submission contains your **tutorial group number, name and student ID.**

5. Failure to follow steps 2 and 3 will result in a **3-mark penalty.**

6. We have provided a template file for your submission, in two formats: .docx and .pdf. If you use the former, remember to generate a pdf file as we accept only pdf format. If you intend to handwrite your answers, please follow the format in the template and ensure that your handwriting is neat and legible, and scan your answers into a pdf file. Marks may be deducted for untidy or illegible handwriting.

7. Upload your submission file (pdf format) to LumiNUS > Files > Assignment 1 submission > Your tutorial group > Your personal submission folder.

8. There are FIVE (5) QUESTIONS on FIVE (5) printed pages including this page.

9. The questions are worth **40 marks** in total.

QUESTION 0. SUBMISSION INSTRUCTIONS (3 MARKS)

a. Ensure that you name your file <AxxxxxxxY>.pdf, where AxxxxxxxY is your student ID. (1 mark)

b. Ensure that you submit your assignment as a **single PDF file**. (1 mark)

c. Ensure that your assignment submission has your tutorial group number, student ID and name. (1 mark)

QUESTION 1. COMPLEMENT NUMBER SYSTEMS (10 MARKS)

In conventional digital computers, two voltage levels are used to represent data in base 2, or binary. So, we use 0 volts (0v) to represent '0', and 5v to represent '1'. This allows for a good electrical separation between 0 and 1 and thus minimizes errors – '0's being accidentally flipped to '1's and vice-versa.

With reliability improvements in technology, we consider a hypothetical computer that uses four voltage levels (0v, 1.25v, 2.5v, 3.75v and 5v) to represent data in base 4.

a. Write down the formula to negate an *n* digit base-4 integer *m* using radix complement (i.e. 4's complement). (2 marks)

Answer: $(-m) = 4^n - m$

b. Convert the following decimal (base-10) integers to unsigned base 2: (i) 1149; (ii) 412. Remember to write the subscript 2 in your answers. (2 marks)

Answers: 10001111101_2; 110011100_2

c. Convert the following decimal (base-10) integers to unsigned base 4: (i) 1149; (ii) 412. Remember to write the subscript 4 in your answers. (2 marks)

Answers: 101331_4; 12130_4

d. Use your formula in part (a) to find -1149 and -412 in 6-digit 4's complement representation. (2 marks)

$-(101331)_4 = -(101331)_{4s} = (232003)_{4s}$,
$-(012130)_4 = -(012130)_{4s} = (321210)_{4s}$

e. Find 1149 – 412 using addition in 6-digit 4's complement system. Show that your answer is correct by converting the result to decimal. (2 marks)

$101331_{4s} + 321210_{4s} = 023201_{4s}$
In decimal, $023201_{4s} = 2x4^4 + 3x4^3 + 2x4^2 + 1x4^0$
$= 737_{10} = 1149_{10} - 412_{10}$

	1	0	1	3	1
+	3	2	1	2	1	0
~~1~~	0	2	3	2	0	1

QUESTION 2. REAL NUMBERS (11 MARKS)

In this question we look at how we can represent real numbers in fixed-point and floating-point notations and compare the two representations. You may assume that all words are 16 bits long and 2's complement system is used.

a. We are a fixed-point representation using these 16-bit words with *m* bits used for the integer portion. Assuming that all bits are used in this representation, derive the expressions for the following values: (3 marks)

i. The most positive integer (i.e. the largest positive integer). $\sum_{i=0}^{m-2} 2^i = 2^{m-1} - 1$

ii. The most negative integer (i.e. the smallest negative integer). Answer: **-2^{m-1}**

iii. The smallest positive value. Answer: **$2^{-(16-m)}$**

b. Use your answers to part (a) to complete this table. You may write your answers in powers of 2 if necessary: (3 marks)

m	Most positive integer	Most negative integer	Smallest positive value
4	**7**	**-8**	**2^{-12}**

Working: Most positive integer: $(0111.000000000000)_{2s}$ = **7**
Most negative integer: $(1000.000000000000)_{2s}$ = **-8**
Smallest positive value: $(0000.000000000001)_{2s}$ = **2^{-12}**

c. Suppose we create a floating-point number system that resembles the IEEE 754 floating-point representation but with the following format:

Sign	Exponent	Normalised mantissa
1 bit	7 bits, excess 63	8 bits

Complete the following table. Write each answer as a decimal value in 3 decimal places multiplied by a power of 2 (eg: 1.234×2^{15} – the value should be in normalized form as shown). (3 marks)

Most positive value	Most negative value	Smallest positive value
1.996×2^{64}	**-1.996×2^{64}**	**1×2^{-63}**

d. Based on your answers for parts (b) and (c) (and possibly using Google), give one advantage and one disadvantage of the floating-point number representations over the fixed-point representation. (2 marks)

Advantage: Much better range and accuracy.
Disadvantage: More complex representation.

(Note: Other answers are acceptable as long as at least one makes reference to the results in parts b and c)

QUESTION 3. C and Assembly Programming (8 MARKS)

In tutorial 2 question 5 we gave the program below to compute even parity for a 31-bit number in register $s0, with bit 31 (MSB) of $s0 holding this parity bit. As before we assume that bit 31 of $s0 is always 0 at the beginning.

a. We can change this program to compute odd parity just by inserting a SINGLE instruction at label "e". Write down that instruction. (2 marks)

```
      add  $t0, $s0, $zero      # make a copy of $s0 in $t0
      lui  $t1, 0x8000
lp:   beq  $t0, $zero, e
      andi $t2, $t0, 1
      beq  $t2, $zero, s
      xor  $s0, $s0, $t1
s:    srl  $t0, $t0, 1
      j    lp
e:
# SINGLE instruction to be inserted here.
```

Single instruction**: xor $s0, $s0, $t1.**
This flips the parity bit so the original even parity code now becomes odd parity.

b. How many instructions in total are executed in the worst case in our modified program? (2 marks)

# of instructions before the loop: 2
# of instructions inside loop: 6
In the worst case, the loop will iterate 31 times (i.e. beq $t0, $zero, e is not taken).
# of instructions executed in the loop: 31 x 6 = 186.
On the 32nd time the beq at lp: is taken, and the instruction at e: is executed = 2 instructions.
Total = 2 + 186 + 1 + 1 = 190 instructions.

c. Write the C equivalent of our modified code. Call the variable that was mapped to $s0 in our MIPS code "data" of type "int". You may assume that "data" already contains the data and its MSB is 0. You may name other variables you use any appropriate names you want. (4 marks)

```
int tmp = data;
int mask = 0x80000000;
```

```
while (tmp) {
        if (tmp & 1)  {
                data ^= mask;
        }
        tmp = tmp >> 1;
}
// Loop above computes even parity. Flip the parity bit to make it odd.
data ^= mask;
```

QUESTION 4. INSTRUCTION ENCODING (8 MARKS)

The following MIPS assembly program processes an array. Answer the questions that follow.

	addi $4, $3, 40
	addi $5, $3, 0
loop:	lw $6, 0($5)
	addi $6, $6, 1
	sw $6, 0($5)
	addi $5, $5, 4
	slt $6, $5, $4
	bne $6, $zero, loop

a. Describe in one sentence what this program does, including where the base address of the array is stored, and how many elements there are in the array. (3 marks)

This program adds one to each element of a 10-element array whose base address is in $3.

b. How many instructions are executed in this program? (1 mark)

10 * 6 + 2 = 62

c. Write down the hexadecimal encoding for the instructions shown in **underlined bold**. You do not need to show your working. You do not need to encode the remaining 4 instructions. (4 marks)

Label	Instruction	Hexadecimal Encoding
	addi $4, $3, 40	**0x20640028**
	addi $5, $3, 0	
loop:	lw $6, 0($5)	
	addi $6, $6, 1	
	sw $6, 0($5)	**0xACA60000**
	addi $5, $5, 4	
	slt $6, $5, $4	**0x00A4302A**
	bne $6, $zero, loop	**0x14C0FFFA**

Working:

addi $4, $3, 40

Opcode	rs	rt	Immed
001000	00011	00100	0000000000101000

$0010\ 0000\ 0110\ 0100\ 0000\ 0000\ 0010\ 1000_2$

0x2 0 6 4 0 0 2 8

sw $6, 0($5)

Opcode	rs	rt	Immed
101011	00101	00110	0000000000000000

$1010\ 1100\ 1010\ 0110\ 0000\ 0000\ 0000\ 0000_2$

0xA C A 6 0 0 0 0

slt $6, $5, $4

Opcode	rs	rt	rd	shamt	func
000000	00101	00100	00110	00000	101010

$0000\ 0000\ 1010\ 0100\ 0011\ 0000\ 0010\ 1010_2$

0x0 0 A 4 3 0 2 A

bne $6, $zero, loop

The loop label is 6 instructions away from the instruction after bne. Thus immediate is -6.

$-(6) = -(0000\ 0000\ 0000\ 0110)_{2s} = (1111\ 1111\ 1111\ 1010)_{2s}$

Opcode	rs	rt	Immed
000101	00110	000000	1111111111111010

$0001\ 0100\ 1100\ 0000\ 1111\ 1111\ 1111\ 1010$

0x1 4 C 0 F F F A