CS2100 Computer Organisation
2020/21 Semester II
Assignment 1
SUGGESTION SOLUTIONS

PLEASE READ THE FOLLOWING INSTRUCTIONS VERY CAREFULLY.

1. **DEADLINE:** The deadline for this assignment is **MONDAY 15 FEBRUARY 2021, 1 PM**. The workbin 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. Keep your file short. You only need to submit your answers; you do not need to include the questions.

7. Answers may be typed or handwritten. If it is the latter, please ensure that it is neat and legible. Marks may be deducted for untidy work or illegible handwriting.

8. Upload your submission file (pdf format) to your personal submission folder in your respective tutorial group's folder, in the Assignment 1 Submissions folder in LumiNUS.

9. There are FOUR (4) QUESTIONS on SIX (6) printed pages including this page.

10. 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 matric number. (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 the lectures we saw the 2's complement and 1's complement number system which are based on the binary (base 2) system. Here we will explore other complement number systems under other bases.

a. Given a base B (≥ 2), derive a formula to calculate X', the n-digit B's complement of a number X. (1 mark)

$X' = B^n - X$

b. An additive inverse of a number a is a number b such that $a + b = 0$. Prove that the B's complement, n', of a number n, is the additive inverse of n. (2 marks)

$n' = B^n - n$
$n + n' = n + B^n - n = B^n$. The B^n is the carry over, which we discard, giving 0.
Hence n' is the additive inverse of n.

c. Find the four digit 10's complement representation of the decimal numbers 123 and -456. I.e. you should find the representation of the two numbers in 10's complement and not their negation.

(4 marks)

$123 = (0123)_{10s}$.
$-456 = 10^4 - 456 = 9544$. Therefore, $-456 = (9544)_{10s}$

d. Following the procedure for subtraction of two 2's complement numbers in base 2 (by converting it into an addition operation like what you did in tutorial 1), compute the following subtraction in 4-digit 10's complement. Show your working and give your final answer in 10's complement. (3 marks)

123 – 456

$123 = (0123)_{10s}$
$-456 = 10^4 - 456 = 9544$. Therefore, $-456 = (9544)_{10s}$.
123 – 456 = 123 + (-456)
In 10's complement: (2 marks for the addition below)

```
   0 1 2 3
 + 9 5 4 4
 ---------
   9 6 6 7
```

Answer: $(9667)_{10s}$ (which is -333 in decimal) (1 mark)

QUESTION 2. FLOATING POINT NUMBERS (12 MARKS)

a. When representing *n-bit* signed numbers in excess notation, briefly explain, using a 3-bit signed number system as example, why the excess chosen is usually 2^{n-1}. (2 marks)

It allows us to have a balance of negative and positive numbers. For example, using excess-4 in a 3-bit representation, we have 4 negative and 4 non-negative values.

Excess-4 code	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Value represented	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**

In the following parts we will use an 8-bit floating point representation with this format:

Sign	Exponent	Mantissa
1-bit	3-bit excess-4	5 bits normalized with 1 hidden bit (i.e. only 4 bits of the mantissa are stored)

b. What are the smallest and largest positive decimal numbers that can be represented? We assume that the entire range of exponent values can be used, i.e. there are no exponent values with special meanings. Write your answers in decimal. (2 marks)

Smallest: $1.0000 \times 2^{-4} = 0.0625$
Largest: $1.1111 \times 2^{3} = 1111.1 = 15.5$

c. Find the normalized representation for these two numbers in the 8-bit floating point representation above. You should calculate the mantissa to one bit more than you require and round up the result if necessary. Write your answer in hexadecimal. (4 marks)

X = 4.4
Y = –0.7

We work out to 6 bits because the mantissa has 5 bits (including hidden bit) and we want to know whether or not to round:

$X = 4.4 = 100.011_2 \approx 100.10_2 = 1.0010 \times 2^2$
Sign = 0, mantissa = 0b0010, exponent = 2 + 4 = 6 = 0b110
Representation = 0110 0010 = 0x62 (2 marks)

$Y = -0.7 = -0.101100_2 \approx -0.10110_2 = -1.0110 \times 2^{-1}$
Sign = 1, mantissa = 0b0110, exponent = -1 + 4 =3 = 0b011
Representation = 0b1011 0110 = 0xB6 (2 marks)

d. Are the two numbers represented perfectly in this floating-point representation? If no, what is the error in the representation for each number (i.e. the difference between the actual value and the floating point representation)? (3 marks)

No. (1 mark)
4.4 is represented by 100.1_2 = 4.5. There is an error of 0.1. (1 mark)
-0.7 is represented by -0.10110_2 = -0.6875. There is an error of 0.0125. (1 mark)

e. Is the accuracy of one of the two numbers better than the other? Explain. (1 mark)

Representation for Y is better. This is because with X some of the mantissa bits are used to represent the integer part leading to a loss of accuracy.

QUESTION 3. STRUCTURES IN C (8 MARKS)

In this question we will examine how C stores structures in memory.

a. Compile and run the enclosed q3.c program using gcc on PC, MacOS or Sunfire. The code included is at the end of this file for your reference. Complete the following tables: (3 marks)

Table 1: Component Sizes

Variable	Size in bytes
t1.num1	**4**
t1.ch	**1**
t1.num2	**4**

Note to TA: The addresses in Table 2 will vary, but each item should be 4 bytes apart. Note in rare cases t1.num2 is only one byte away from t1.ch. See part b.

Table 2: Component Addresses

Variable	Address
&t1.num1	**0x7ffee0fcd980**
&t1.ch	**0x7ffee0fcd984**
&t1.num2	**0x7ffee0fcd988**

Note to TA: The addresses in Table 3 will vary, but each array element should be 12 bytes apart. In rare cases they may be only 9 bytes apart. See part b.

Table 3: Array Element Addresses

Variable	Address
&t2[0]	**0x7ffee0fcd990**
&t2[1]	**0x7ffee0fcd99c**
&t2[2]	**0x7ffee0fcd9a8**

Marking scheme: 1 mark per table.

b. Look at the addresses of t1.ch and t1.num2 in Table 2. Do they differ by 1 byte or 4 bytes? Explain why, with reference to the size of t1.ch. (1 mark)

They differ by 4 bytes. The t1.ch component is 1 byte long and the compiler pads in 3 more bytes to maintain word alignment.

(Note to TA: There may be rare odd cases where the compiler does not pad the extra 3 bytes. Counter-check with the addresses of t1.ch and t1.num in Table 2 to check that this is the case. If so, you can accept explanations like 'They differ by 1 as t1.ch is 1 byte long', etc.)

c. The code fragment from q3.c sums the num2 component of the array t2 and stores the total in result.num2.

```
… <other code here> …

mystruct_t result = {0.0, ' ', 0};
int i;

… <other code here> …

for (i=0; i<6; i++) {
    result.num2 = result.num2 + t2[i].num2;
}
```

Assuming that $s0 contains the base address of t2, $s1 contains the address of result and $t0 is mapped to the variable i, rewrite the for-loop part of the code in MIPS assembly. Ensure that your code is properly commented or marks will be deducted. Note also that pseudo-instructions other than those required for any unaligned loads and stores are not allowed. (Hint: Look at Table 3 to see how far apart each element of t2 is in memory). (4 marks)

(Solution where t2.c is padded with 3 bytes):

Marking Scheme:

Correct access to t2[i].num2: 1 mark
Loop works correctly: 1 mark
Correctly increments to next array element: 2 marks

```
        addi $t0, $zero, 0       # Initialize i to 0
        addi $t1, $zero, 0       # Temporary sum for result.num1
        addi $t2, $s0, 0         # Copy base address of t2 into temporary register.

loop:   slti $t3, $t0, 6
        beq $t3, $zero, exit     # Exit if i==6
        lw $t4, 8($t2)           # Load t2[i].num2
        add $t1, $t1, $t4        # temp = temp + t2[i]
        addi $t2, $t2, 12        # The next structure is 12 bytes away.
        addi $t0, $t0, 1         # Increment i
        j loop
exit:   sw $t1, 8($s1)           # Store sum in result.num2
```

(Solution where t2.c is NOT padded with 3 bytes. Differences are underlined. Note use of unaligned load word ulw and unaligned store word usw. -1 mark total if students fail to use these.)

<u>Note: -1 mark if student uses lw/sw instead of ulw and usw.</u>

```
        addi $t0, $zero, 0       # Initialize i to 0
        addi $t1, $zero, 0       # Temporary sum for result.num1
        addi $t2, $s0, 0         # Copy base address of t2 into temporary register.

loop:   slti $t3, $t0, 6
        beq $t3, $zero, exit     # Exit if i==6
        ulw $t4, 5($t2)          # Load t2[i].num2
        add $t1, $t1, $t4        # temp = temp + t2[i]
        addi $t2, $t2, 9         # The next structure is 9 bytes away.
        addi $t0, $t0, 1         # Increment i
        j loop
exit:   usw $t1, 5($s1)          # Store sum in result.num2
```

QUESTION 4. ANALYSIS OF MIPS ASSEMBLY PROGRAM (7 MARKS)

Line	Labels	Instructions
1		`addi $1, $0, 0`
2		`add  $3, $20, $1`
3	`loop:`	`lw   $4, 0($3)`
4		`srl  $5, $4, 16`
5		`sll  $6, $4, 16`
6		`or   $4, $5, $6`
7		`sw   $4, 0($3)`
8		`addi $1, $1, 1`
9		`slt  $5, $1, $21`
10		`beq  $5, $zero, loop`

The program above goes through the elements of an integer array whose base address is in register $20, and the number of elements is in register $21. It does something with the data in the array and writes it back.

a. Unfortunately, this program does not work as intended. Rewrite the program so that it works properly, changing only what is necessary. You may also add instructions but these should be kept to the absolute minimum. You may assume that there are no errors in lines 4 to 6 inclusive and you are NOT allowed to change these lines.

(3 marks)

Solution #1:

Line	Labels	Instructions
1		`addi $1, $0, 0`
2		`add  $3, $20, $1`
3	`loop:`	`lw   $4, 0($3)`
4		`srl  $5, $4, 16`
5		`sll  $6, $4, 16`
6		`or   $4, $5, $6`
7		`sw   $4, 0($3)`
8		`addi $1, $1, 1`
ex		***`add $3, $3, 4`***
9		`slt  $5, $1, $21`
10		***`bne`*** `$5, $zero, loop`

Solution #2:

Line	Labels	Instructions
1		`addi $1, $0, 0`
ex	***`loop:`***	***`sll $12, $1, 2`***
2		***`add $3, $20, $12`***
3		`lw $4, 0($3)`
4		`srl $5, $4, 16`
5		`sll $6, $4, 16`
6		`or $4, $5, $6`

7		`sw $4, 0($3)`
8		`addi $1, $1, 1`
9		`slt $5, $1, $21`
10		***`bne`*** `$5, $zero, loop`

ex = extra instruction added

b. How many instructions are executed in the CORRECTED program if \$21 = 5? (2 marks)

For solution #1: 2 + (5 × 9) = 47
For solution #2: 1 + (5 × 10) = 51
Grade according to what the student wrote for part (a).

c. Describe in one sentence what the CORRECTED program does. (2 marks)
It swaps the upper and lower 16 bits of each element in the array.

Code for q3.c
This is provided only for your reference. You do not need to type it in.

```
#include <stdio.h>

typedef struct {
    float num1;
    char ch;
    int num2;
} mystruct_t;

int main() {

    mystruct_t t1, t2[6];
    mystruct_t result = {0.0, ' ', 0};
    int i;

    printf("Structure component sizes:\n");
    printf("Size of t1.num1 = %lu bytes.\n", sizeof(t1.num1));
    printf("Size of t1.ch = %lu bytes.\n", sizeof(t1.ch));
    printf("Size of t1.num2 = %lu bytes.\n\n", sizeof(t1.num2));

    printf("Structure component addresses:\n");
    printf("Address of t1.num1 = %p.\n", &t1.num1);
    printf("Address of t1.ch = %p.\n", &t1.ch);
    printf("Address of t1.num2 = %p.\n\n", &t1.num2);

    printf("Array element addresses:\n");
    printf("Address of t2[0] = %p.\n", t2);
    printf("Address of t2[1] = %p.\n", &t2[1]);
    printf("Address of t2[2] = %p.\n", &t2[2]);

    for (i=0; i<6; i++)
        result.num2 = result.num2 + t2[i].num2;

    return 0;
}
```