CS2100 Computer Organisation
AY2022/23 Semester 2
Assignment 1

PLEASE READ THE FOLLOWING INSTRUCTIONS VERY CAREFULLY.

1. The **deadline** for this assignment is **12 NOON, FRIDAY 24 FEBRUARY 2023**. The submission folders will close shortly after 12 noon and no submissions will be allowed after that. You WILL receive 0 mark if you do not submit on time.
2. **We will be enforcing the 0-mark for late submission rule very strictly! It does not matter how hard you worked you WILL get ZERO (0) if you submit even 1 second after the folder closes! No appeals will be entertained. Therefore, plan to submit at least 2 hours earlier (by 10 am, 24 February 2023).**
3. You should complete this assignment **on your own** without collaborating or 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.
4. Use the answer book assign1_ansbk.docx to enter your answers for Questions 1, 2 and 4, then save as a PDF called AxxxxxxxY.pdf, where AxxxxxxY is your STUDENT NUMBER. Do not mix this up with your email address which begins with E instead of A. ZIP up your AxxxxxxY.pdf file together with your q3.c from Question 3 into **ONE ZIP FILE** named AxxxxxxxY.zip. Failure to follow these instructions will result in loss of marks.
5. Please keep your submission **short**. You only need to submit your answers; you do not need to include the questions.
6. If you submit multiple copies, one copy will be graded. This is not necessarily the last copy submitted.
7. Answers may be typed or handwritten. In case of the latter, please ensure that you use **dark ink** and your handwriting is **neat and legible**. Marks may be deducted for untidy work or illegible handwriting.
8. Submit your assignment on Canvas Assignments under "Term Assignment 1" in the folder created specifically for your tutorial group.
9. There are **FOUR (4)** questions on **SIX (6)** printed pages in this assignment.
10. The questions are worth **40 MARKS** in total.
11. If you have any queries on this assignment please post them on the Canvas forums.

=== END OF INSTRUCTIONS ===

Question 0. Submission instructions (-2 marks)
You do not need to answer this question. Your tutor will deduct the marks accordingly.

(a) You did not name your files appropriately, i.e., **AxxxxxxxY.zip** or **AxxxxxxxY.pdf** [-1 mark]

(b) You do not have your **student ID, name** or **tutorial group number** written at the top of the first page of your submitted answer book. [-1 mark]

Question 1. Basic Number Systems (10 marks)

We will begin this assignment with a warmup question that directly applies what is taught in the lectures. Complete all parts of this question.

(a) Convert decimal value **-19.04492188** into 32-bit IEEE-754 floating-point format, writing your answer in hexadecimal. [2 marks]

(b) Convert **0x3EED0000**, expressed as a 32-bit IEEE-754 format floating point number into decimal. [2 marks]

(c) Consider a fixed-point representation using 16 bits in 2's complement, with 10 bits for the integer portion and 6 bits for the fraction portion.

(i) What is the largest possible positive number that you can represent? Write your answer in decimal. [1 mark]

(ii) What is the smallest possible positive number that you can represent? Write your answer in decimal. Note that 0 is not a positive number. [1 mark]

(iii) What is the most negative number that you can represent? [1 mark]

(iv) Represent -193.1 in this number system. Write your answer in hexadecimal. To do this, remove the binary point and encode as a single 16-bit number. E.g. if your answer is 0010110010.101010_{2s}, remove the binary point to get 0010110010101010_{2s}, break into groups of 4 to get $0010\ 1100\ 1010\ 1010_{2s}$, then encode to 0x2CAA. [1 mark]

(v) Convert the answer you've given to part (iv) back into decimal. [1 mark]

(vi) What is the error in representing -193.1 in this number system? [1 mark]

Question 2. Alternate Number Systems (10 marks)

We should all now be very familiar with 2's- and 1's-complement number systems, as well as working with numbers in base 2, 8, 10 and 16. We will now look at alternative number systems.

(a) In general, the n-digit B's complement in base-B of a number x is found using the formula $B^n - x$. Find the following complements in the specified base and number of digits.[3 marks]

(i) The 3-digit 7's complement of 312_7.

(ii) The 4-digit 5's complement of 1431_5.

(iii) The 4-digit 10's complement of 783_{10}.

(b) The negation (or additive inverse) of a number x is defined to be some number y such that $x + y = 0$. Using this definition, show why the n-digit B's complement of a number x in base B is its negation. [2 marks]

(c) Find the results of the following additions in the respective bases shown: [3 marks]

(i) $132_5 + 114_5$

(ii) $214_7 + 135_7$

(iii) $415_8 + 515_8$

(d) Given $418_x + 765_x = 1284_x$, find the mystery base x. [2 marks]

Question 3. C Programming (10 marks)

In this question we will implement routines to add and subtract numbers defined in the fixed-point format in Question 1c, where we have a fixed-point number system represented as a 16-bit 2's complement number, with 10 bits for the integer part and 6 bits for the fraction part. In your assignment ZIP file, you will find three C files: **q3.c**, **q3.h** and **q3test.c**.

You should implement the required routines in **q3.c**.

Since the lengths of the "int" and "short" datatypes are system dependent, in this question we will instead use the int16_t datatype, which is a datatype that is guaranteed to be 16-bit long.

The tables below show the various functions in **q3.c**. Note that you **do not** have to implement the **print_bits** function.

Function Name	**Description**
print_bits	Prints out the supplied int16_t number in binary. **You do not need to implement this.**
float_to_fixed	Takes a float input and converts it into a 16-bit 2's complement fixed-point number as described in Question 1c, with 10 bits for the integer portion and 6 bits for the fraction portion. **For the fraction part, truncate your conversion to binary after 6 bits. DO NOT convert to 7 bits to check for rounding.**
fixed_to_float	Takes an int16_t argument representing a 16-bit 2's complement fixed-point number as described in Question 1c. Returns the floating point equivalent.
negate	Takes an int16_t argument representing a 16-bit 2's complement fixed-point number as described in Question 1c. Returns a 2's complement negation of the argument.

Function Name	Description
add	Given two int16_t numbers x and y representing the fixed-point representation in Question 1c, return x + y. Note that to obtain any credit at all for this part you must implement the addition using only the int16_t numbers. You must not convert to floating point to add the numbers.
subtract	Given two int16_t numbers x and y representing the fixed-point representation in Question 1c, return x – y. Note that to obtain any credit at all for this part you must implement the subtraction using only the int16_t numbers. You must not convert to floating point to subtract the numbers.

The **q3.h** file contains the headers for the functions above, and the **q3test.c** file contains a main function to test your **q3.c** library. The **q3test.c** file will read in two floating point numbers from the keyboard, then convert them to fixed point, print them out, do an add and a subtract, then print out the results together with errors in the representation.

DO NOT MODIFY ANYTHING IN q3.h NOR IN q3test.c.

To compile the program:

gcc q3.c q3test.c -o q3test

To run:

./q3test

To aid you in your testing the following screenshots show some sample runs. You should get results very similar to what is shown here:

```
Enter x and y: 33.4 -12.3


x: 33.400002 fixed_x: 2137 float_x: 33.390625 Error: 0.009377
y: -12.300000 fixed_y: -787 float_y: -12.296875 Error: -0.003125

x = 0000100001011001 y = 1111110011101101 x + y = 0000010101000110 x - y = 0000101101101100


x: 33.400002 y: -12.300000 x + y: 21.100002 fixed_x + fixed_y = 21.093750 fixed_err = 0.006252


x: 33.400002 y: -12.300000 x - y: 45.700001 fixed_x - fixed_y = 45.687500 fixed_err = 0.012501
```

x = 33.4, y = -12.3

```
Enter x and y: -10.34 -30.586


x: -10.340000 fixed_x: -661 float_x: -10.328125 Error: -0.011875
y: -30.586000 fixed_y: -1957 float_y: -30.578125 Error: -0.007875

x = 1111110101101011 y = 1111100001011011 x + y = 1111010111000110 x - y = 0000010100010000


x: -10.340000 y: -30.586000 x + y: -40.926003 fixed_x + fixed_y = -40.906250 fixed_err = -0.019753


x: -10.340000 y: -30.586000 x - y: 20.246000 fixed_x - fixed_y = 20.250000 fixed_err = -0.004000
```

x=-10.34, y = -30.586

```
Enter x and y: 1.3847 -1.3847


x: 1.384700 fixed_x: 88 float_x: 1.375000 Error: 0.009700
y: -1.384700 fixed_y: -88 float_y: -1.375000 Error: -0.009700

x = 0000000001011000 y = 1111111110101000 x + y = 0000000000000000 x - y = 0000000010110000


x: 1.384700 y: -1.384700 x + y: 0.000000 fixed_x + fixed_y = 0.000000 fixed_err = 0.000000


x: 1.384700 y: -1.384700 x - y: 2.769400 fixed_x - fixed_y = 2.750000 fixed_err = 0.019400
```

x= 1.3847, y = -1.3847

Submission

Zip your **q3.c** file ONLY together with the PDF file containing the answers for the rest of this assignment into a single ZIP file called AxxxxxxxY.zip, where AxxxxxxxY is your student ID.

Marking Scheme

Your program will be tested on five test cases worth 2 marks each. If you pass all 5 test cases you will get the full 10 marks.

For the “negate, “add” and “subtract” functions you will lose 3 marks each if you do not negate, add or subtract using strictly the fixed-point representation in the int16_t arguments.

For example, if you convert the arguments first to floating point numbers in “add”, then add them together, then convert the result back to an int16_t number, you will lose 3 marks. Likewise too with “negate” and “subtract”.

Question 4. MIPS (10 marks)

The following MIPS code reads an integer array ***A***, whose base address is stored in **$s0**, and computes some answer in **$s1** which is associated with the integer variable **ans**.

```
        addi $s1, $zero, 0      # Inst1:
        addi $t0, $s0, 16       # Inst2:
                                #
  Here: addi $t0, $t0, -4       # Inst3:
        lw   $t1, 0($t0)        # Inst4:
        slti $t2, $t1, 8        # Inst5:
        bne  $t2, $zero, Skip   # Inst6:
        addi $s1, $s1, 1        # Inst7:
        sll  $s1, $s1, 1        # Inst8:
  Skip: beq  $t0, $s0, End      # Inst9:
        j    Here               # Inst10
  End:
```

For the following questions, you do NOT need to show working.

For parts (a) and (b), assuming that array *A* contains the following 16 integers:

8, 2, 10, 23, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6. (i.e., *A*[0] = 8, *A*[15] = 6.)

(a) How many elements of array ***A*** are read into **$t1** in the given code? [1 mark]

(b) What is the final value of **$s1** after running the code? [1 mark]

(c) For the instruction **addi $t0, $t0, -4**, write out its hexadecimal representation. [2 marks]

(d) For the only R-format instruction in the above code, write out its hexadecimal representation. [2 marks]

(e) For the instruction **bne $t2, $zero, Skip**, write out its hexadecimal representation. [2 marks]

(f) Assuming that the first instruction **addi $s1, $zero, 0** is at address **0x0040002C**, write out the hexadecimal encoding of the **j Here** instruction? [2 marks]

=== END OF PAPER ===