CS2100 Computer Organization
AY2025/26 Semester 2
Assignment 1 [with Answers]
Total Marks: 40

Instructions for Students:

1. There are **4** questions in this assignment, with a total of 40 marks.

2. This assignment is due on **Monday, 16th February 2026, 1 pm**. Late submissions will incur penalties as spelt out on Canvas > Pages: "a 10% penalty for submissions up to 3 hours late (i.e. submitting between 1.01pm and 4:00pm for a 1pm deadline), a 20% penalty for submissions up to 6 hours late, a 30% penalty for submissions up to 9 hours late, and no marks will be given for assignments submitted more than 9 hours late."

3. Enter your answers into Canvas > Assignments > Assignment 1. You are allowed [3] attempts. Only your last submission (not your best) will be graded. You are not to request us to ignore your last submission (including a late submission), nor can you choose the version you want us to grade.

4. You could refer to the document assign1_25s2_qns.pdf at Canvas > Files > Assignments > Assignment 1 in which all the questions are presented in a single file for your convenience. You will look at the questions in this file and answer them in the Canvas.

5. Please read and follow the instructions in each question on how to format your answers. This is important as your answers will be auto-graded, so any answer that departs from the specified format will be graded as incorrect.

6. You should do these assignments on your own. Do not discuss the assignment questions with others.

7. Please post on QnA "Assignment " topic if you have any queries on this assignment.

Question 1: Number System Conversions and Precision (12 marks)

(a) Base Conversion from Binary (6 marks)
Convert the following binary number into the bases specified below:
1010110.101_2
(i) Convert the number to decimal.
(ii) Convert the number to hexadecimal.
(iii) Convert the number to base-5.
For fractional parts, your final answers should be rounded to the nearest 3 digits after the radix point.

(i) To decimal
Integer part:

$$1010110_2 = 64 + 16 + 4 + 2 = 86$$

Fraction part:

$$0.101_2 = \frac{1}{2} + \frac{1}{8} = 0.625$$

So, $1010110.101_2 = 86.625_{10}$
Answer: 86.625_{10}

(ii) To hexadecimal
Group into 4 bits:

- Integer: $1010110_2 \rightarrow 01010110 = 56$
- Fraction: $.101_2 \rightarrow .1010_2 = A$

Answer: 0x56.A00

(iii) To base-5
Convert $86.625_{10}$to base-5.
Integer part: $86_{10}\rightarrow$ repeated division by 5 $\rightarrow 321_5$
Fraction part: multiply by 5 repeatedly:

- $0.625 \times 5 = 3.125\rightarrow$ digit 3, remainder 0.125
- $0.125 \times 5 = 0.625\rightarrow$ digit 0, remainder 0.625
  This pattern repeats: 30 30 30…

So, $86.625_{10} = 321.303_5$ (since .303… continues as $321.303030\ldots_5$)
Answer: 321.303_5

(b) Base Conversion from Decimal (6 marks)
Convert the following decimal number into the bases specified below:
142.6875_{10}
(i) Convert the number to binary.
(ii) Convert the number to ternary (base-3).
(iii) Convert the number to hexadecimal.
For fractional parts, your final answers should be rounded to the nearest 3 digits after the radix point.

(i) To binary
Integer part: $142_{10} = 10001110_2$
Fraction part: multiply by 2 repeatedly:

- $0.6875 \times 2 = 1.375 \rightarrow 1$, remainder 0.375
- $0.375 \times 2 = 0.75 \rightarrow 0$, remainder 0.75
- $0.75 \times 2 = 1.5 \rightarrow 1$, remainder 0.5

- $0.5 \times 2 = 1.0 \rightarrow 1$, remainder 0

So the fraction part is 0.1011_2
The actual binary representation is: $142.6875_{10} = 10001110.1011_2$
To represent in 3 fractional bits, round 0.1011_2 to 3 bits: since next bit is $1 \rightarrow 0.101$ rounds up to 0.110
Answer (3 fractional bits): 10001110.110_2

(ii) To ternary (base-3)
Integer part: $142_{10} = 12021_3$
Fraction part (0.6875): multiply by 3:

- $0.6875 \times 3 = 2.0625 \rightarrow$ digit 2
- $0.0625 \times 3 = 0.1875 \rightarrow$ digit 0
- $0.1875 \times 3 = 0.5625 \rightarrow$ digit 0
- $0.5625 \times 3 = 1.6875 \rightarrow$ digit 1

Then it repeats (cycle 2001).

To represent in 3 fractional digits, round $0.20012001\ldots_3$ to 3 digits:
The fourth digit after the radix point is 1. In an odd base, this is the middle digit, contributing exactly half of one unit in the previous place. Consequently, the value lies exactly midway between 0.200_3 and 0.201_3. Without a specified rounding rule, either representation is acceptable.

$$142.6875_{10} = 12021.200_3 \; or \; 12021.201_3$$

Answer (3 fractional digits): 12021.200_3 or 12021.201_3

(iii) To hexadecimal
Integer part: $142_{10} = 0x8E$
Fraction part: $0.6875 = 0.1011 \rightarrow$ digit B
Answer: 0x8E.B00 (3 fractional hex digits)

Question 2: Custom Number System Design and Comparison (8 marks)

A computer architect proposes a custom signed number system with the following properties:

- 6 digits and Base-5
- The most significant digit (MSD) is used as a sign digit: 0 is positive and 1 is negative
- The remaining digits represent the magnitude in base-5

(a) Range of the Number System (3 marks)
(i) The largest positive number representable in this system (in decimal) is [_____]
(ii) The most negative number representable in this system (in decimal) is [_____]
(iii) The total number of distinct numerical values representable in this system is [_____]

(i) Largest positive number
The largest magnitude occurs when all 5 magnitude digits are 4:

$$44444_{5(SM)} = 5^5 - 1 = 3124_{10}$$

With sign digit = 0 (positive), the largest positive number = 3124

(ii) Most negative number
The magnitude is the same as above, but with sign digit = 1 (negative):
Most negative number = −3124

(iii) Total number of distinct values representable

- Number of possible magnitude values = $5^5 = 3125$
- Number of sign options = 2 (positive and negative)

$$\text{Total representable values} = 2 \times 5^5 = 6250$$

(Note: this counts representations; +0 and −0 are both representable.)
Total number of distinct values representable = 6250 – 1 = 6249

Final Answers:
(i) 3124
(ii) −3124
(iii) 6249

(b) Representation of a Given Number (2 marks)
The representation of the decimal number −224 in this custom number system is [_____].

First convert the magnitude 224_{10} to base-5
Divide repeatedly by 5:

- $224 \div 5 = 44$ remainder 4
- $44 \div 5 = 8$ remainder 4
- $8 \div 5 = 1$ remainder 3
- $1 \div 5 = 0$ remainder 1

So,

$$224_{10} = 1344_5$$

Since the magnitude field has 5 digits, pad with a leading zero:

$$224_{10} = 01344_5$$

Now add the sign digit. The sign digit for negative numbers = **1**.
So, the full 6-digit base-5 SM representation is: $101344_{5(SM)}$
Answer is $101344_{5(SM)}$

(c) Comparison with Two's Complement (3 marks)
(i) The minimum value representable using 5-bit two's complement (in decimal) is [_____]
(ii) Is the number -224_{10} is representable in 5-bit two's complement? [Yes/No]
(iii) One advantage of two's complement over the custom base-5 system above is: [_____]
(Answer in one short phrase)

(i) The minimum value representable using 5-bit two's complement (in decimal) is −16.
(Range of 5-bit two's complement is −16 to +15.)

(ii) Is the number −224 is representable in 5-bit two's complement?: No, this is because −224 lies outside the range of values representable by 5-bit two's complement.

(iii) One advantage of two's complement over the custom base-5 system above is:

- Simpler arithmetic
- single zero
- no separate sign handling
- efficient hardware implementation

Final Answers:

- **Minimum value:** −16
- **Representable?** No
- **Advantage:** simpler arithmetic or any other correct answer.

Question 3: Floating-Point Representation and Precision (4 marks)

This question focuses on how real numbers are represented (approximately) using the IEEE 754 single-precision floating-point format.

(a) Decimal to IEEE 754 (2 marks)
Consider the decimal number: +0.1875
(i) The normalized binary form of this number is [_____]$_2$ × 2^[__]
(ii) The exponent field (8 bits, after applying bias) is [_____]$_2$
(iii) The final IEEE 754 single-precision representation (hexadecimal) is 0x[_____]

(i) Normalized binary form:
Shift the binary point to get a leading 1:

$$0.0011_2 = 1.1 \times 2^{-3}$$

where, the exponent is **−3.**

(ii) Exponent field (8 bits, after bias):
IEEE 754 single precision bias = 127

$$-3 + 127 = 124$$
$$124_{10} = 01111100_2$$

So the exponent field is: 01111100

(iii) Final IEEE 754 representation (hexadecimal)
Sign bit (1): 0 (positive)
Exponent (8): 01111100
Mantissa(23): take bits after the leading 1 → 10000000000000000000000

Full 32-bit representation: **0 | 01111100 | 10000000000000000000000**
In hexadecimal: **0x3E400000**

Final Answers:
(i) 1.1 × 2⁻³
(ii) 01111100
(iii) 0x3E400000

(b) IEEE 754 to Decimal (2 marks)
Given the IEEE 754 single-precision floating-point number: 0xC1A00000
(i) The sign of the number is [positive / negative]
(ii) The actual exponent value (after removing bias) is [_____]
(iii) The binary mantissa (including the implicit leading 1) is [_____]$_2$
(iv) The decimal value of the number is [_____]$_{10}$

Convert 0xC1A00000 to 32-bit binary form is: 11000001101000000000000000000000
Split into IEEE 754 fields

Field	Bits
Sign	1
Exponent	10000011
Mantissa	01000000000000000000000

Sign of the number: Sign bit = 1, so negative.

Actual exponent value (after removing bias):
- Exponent field: $10000011_2 = 131_{10}$
- Actual exponent $= 131 - 127 = 4$

Binary mantissa (including implicit leading 1):
Mantissa bits are stored without the leading 1.
Including implicit leading 1: $1.01000000000000000000000_2$
Answer: $1.01000000000000000000000_2$
Decimal value of the number is $-(1.01_2 \times 2^4)$ = $-10100_2 = -20_{10}$

Final Answers:
(i) negative
(ii) 4
(iii) $1.01000000000000000000000_2$ or 1.01_2
(iv) -20_{10}

Question 4: Analyse the given MIPS Code and answer the following questions (16 marks)

```
1   # $s0 - address of memory that stores base address of the array
2   # $s1 - address of memory that holds the array size
3
4   lw $a0, 0($s0)              # Load base address of the array
5   lw $a1, 0($s1)              # Load array size
6   addi $v0, $0, 0              # Clear $v0 for sum
7   addi $t0, $0, 0              # Index i = 0
8 label3:
9   beq $t0, $a1, label1        # If i == length, exit loop
10  lw $t1, 0($a0)              # Load array[i]
11  andi $t2, $t1, 1            # Check Least Significant Bit
12  bne $t2, $zero, label2      # If result != 0, it's odd → skip
13  add $v0, $v0, $t1           # Add to sum if even
14 label2:
15  addi $a0, $a0, 4            # Pointer = Pointer + 4 bytes
16  addi $t0, $t0, 1            # i++
17  j label3
18 label1:
19  add $t4, $v0, $zero         # Save sum to $t4 (final result)
20 exit:
```

(a) Explain the purpose of the given MIPS code when `$s0 = 0x2000` and `$s1 = 0x2004`. Assume that the address `0x2000` has the base address of an integer array, and the address `0x2004` stores the size of the array as 7. The elements of the integer array can take values from 0 to 99. What operation does the given MIPS code perform on the integer array?
(1 mark)

The given MIPS code iterates through an integer array whose base address is stored at 0x2000 and whose size is stored at 0x2004. For each element, it determines whether the value is even or odd by examining the least significant bit (LSB). The code then computes the sum of all even-valued elements in the array.

(b) What does register `$t4` contain at the end of execution?
(1 mark)

The register $t4 contains the sum of all even-valued elements in the array.

(c) Given the following memory contents, calculate the **total number of instructions executed** by the MIPS code. **(3 marks)**

Memory Address	Values
0x2000	0x3000
0x2004	7
...	...
0x3000	4
0x3004	7
0x3008	10
0x300C	3
0x3010	8
0x3014	6
0x3018	9

The total number of instructions executed is **59**.
This includes:

- 4 setup instructions
- the loop body executed for 7 elements, 4 of which are even each taking up 8 instructions, and 3 of which are odd each taking up 7 instructions, hence a total of (4x8)+(3x7)=53 instructions
- one final beq when i == length.
- one final instruction that copies the sum into $t4.

So 4+53+1+1 = 59 instructions.

(d) Assuming that the array size is *n* (where *n* is a positive integer). What are the maximum and minimum possible number of times the branch instruction `beq $t0, $a1, label1` (line 9) is taken? A branch instruction is taken if its branch condition is true, otherwise it is not taken. **(2 marks)**

The maximum and minimum possible number of times the beq instruction at line 9 is taken are **1**.

Explanation: The branch condition (`$t0 == $a1`) becomes true only once, when the loop index t0 reaches the array size a1. At that point, the branch is taken to exit the loop. In all earlier iterations, the condition is false, so the branch is not taken.

(e) Assuming that the array size is *n* (where *n* is a positive integer). What are the maximum and minimum possible number of times the branch instruction `bne $t2, $zero, label2` (line 12) is taken? A branch instruction is taken if its branch condition is true, otherwise it is not taken. **(2 marks)**

The minimum number of times the bne instruction at line 12 is taken is **0**, which occurs when the array contains only **even** values.
The maximum number of times the bne instruction at line 12 is taken is **n**, which occurs when the array contains only **odd** values.

Explanation: the bne instruction is taken only when the array element is odd number.

(f) Modify the given MIPS code so that it **terminates the loop immediately** when the value **99** is encountered in the array. The program should then exit and store the current accumulated result in register $t4. You should make a **minimal number of changes** to the original code and at the same time keep your code efficient. Clearly indicate the number of instruction(s) that must be added and/or removed with explicit reference to the line numbers in the original code. Example: remove 2 instructions in line 1 and 2 and add 3 instructions between lines 3 and 4. **(3 marks)**

Minimal and efficient changes:

- Remove: 0 instructions
- Add: 2 instructions
- Where to add: Insert the below instruction, before line 8

```
addi $t3, $zero, 99      # load constant 99
```

- Insert the below instruction, between lines 10 and 11

```
beq $t1, $t3, label1    # if array[i]==99, exit loop early
```

(g) Instruction Encoding: Encode the instructions in line 9 and line 10 from the original code into 32-bit binary machine code. Clearly show the opcode and relevant fields used in the encoding. You need to represent the final machine code in hexadecimal. **(4 marks)**
(i) Instruction at line 9: `beq $t0, $a1, label1`
(ii) Instruction at line 10: `lw $t1, 0($a0)`

(i) Instruction at line 9: `beq $t0, $a1, label1`

This is an I-type instruction. The branch offset is calculated as below:

- beq is at line 9
- PC + 4 points to instruction at line 10
- Target label label1 is after line 18
- Number of instructions between line 10 and line 19 = 7, hence offset = 7

Field	Value
Opcode (beq)	000100
rs ($t0)	01000
rt ($a1)	00101
Immediate (offset)	0000000000000111

Binary encoding: 000100 01000 00101 0000000000000111
Hexadecimal encoding: 0x11050007

(ii) Instruction at line 10: `lw $t1, 0($a0)`

This is an I-type instruction.

Field	Value
Opcode (lw)	100011
rs ($a0)	00100
rt ($t1)	01001
Immediate	0000000000000000

Binary encoding: 100011 00100 01001 0000000000000000
Hexadecimal encoding: 0x8C890000