CS2100 Computer Organization
AY2024/25 Semester 2
Assignment 1 [with ANSWER]
Total Marks: 40

Q1. Conversion Between Number Systems (2 x 6 = 12 marks)

(a) Convert the binary number 1101011.011 to decimal, hexadecimal, and base-6.

Answers:

To decimal: $(2^6 x1)+(2^5 x1)+0+(2^3 x1)+0+(2^1 x1)+(2^0 x1)+0+(2^{-2} x1)+(2^{-3} x1)$
$= 107.375$

The decimal representation of the given binary number is 107.375

To hexadecimal: Convert the integer part and fraction part to hexadecimal by grouping into 4 bits. Pad with 0's if necessary.
Integer part after padding 0's at the front = $\underline{0}110\ 1011 \rightarrow$ 6B
Fraction part after padding 0's at the end = $011\underline{0} \rightarrow$ 6
Hexadecimal value of the given binary number is 0x6B.6

To Base-6: Convert 1101011.011 in binary to decimal, which is 107.375. Now convert 107.375 to base-6 representation by repeatedly dividing the integer part by 6 and multiplying the fraction part by 6 and keeping remainders.
To convert 1101011.011 (binary) to base-6:

1. Convert Binary to Decimal: 1101011.011=107.375
2. Convert Decimal to Base-6:
   - Integer Part (107): Repeatedly divide by 6 and record remainders.
   - Fractional Part (0.375): Multiply by 6 and extract integer parts.

Thus, the base-6 representation of 107.375 is 255.213 (base-6)

(b) Convert the decimal number 255.3705 to binary, ternary, and hexadecimal. Your answer should be most accurate to 3 digits in the fraction part.

Answers:

To Binary: To convert 255.3705 in decimal to binary:

1. Integer part (255) : Repeatedly divide by 2 and record remainders.
2. Fraction part (3705) : Repeatedly multiple by 2 and extract the integer parts. For the 4th bit, you will have remainder as 1 so round-off the 3rd bit to 1, i.e., 0.0101 → 0.011.

The decimal **255.3705** will be **11111111.0101 = 11111111.011 (base 2)**
Marking scheme: Deduct 1 mark if answer is 11111111.010 (base 2)

To Ternary:

1. Integer part (255) : Repeatedly divide by 3 and record remainders.
2. Integer part (3705) : Repeatedly multiply by 3 and extract the integer parts.

The decimal **255.3705** will be **100110.1010 = 100110.101 (base 3)**

To Hexadecimal: Convert the decimal 255.375 from decimal to binary first and then convert the decimal to hexadecimal.

1. **Convert Decimal to Binary: 255.3705 = 11111111.011 (base-2).**
2. **Convert Binary to Hex:**
   Integer Part (255): Repeatedly divide by 16 and record remainders.
   $255_{10}=FF_{16}$

   Fractional Part (0.3705): Repeatedly multiply by 16 and record remainders.
   $0.3705_{10} \approx 0.5EE_{16}$ (rounded from $0.5ED9_{16}$)

Thus, the hexadecimal representation of **255.3705_{10}** is 0x**$FF.5EE_{16}$**

If students got the answer from binary, then their hexadecimal will be **FF.6 (base 16)**. Give them partial mark for this answer.

Q2. Understanding Encodings in Different Systems (2 x 3 = 6 marks)
(a) Represent the decimal number -128 using 8-bit signed-magnitude, 8-bit 2's complement and 8-bit excess-128 encoding. If you can't represent any of them, state the reason.

Answers:

Signed-magnitude:
The value -128 can't be represented using this notation with 8-bits. The range of 8-bit signed-magnitude representation is from -127 to 127. The value -128 can't be represented because the magnitude of 128 will take 8-bits and there is no space for the sign bit.

2's complement:
+128 in binary: 1000 0000
Flip the bits and add 1: 0111 1111 + 1 = $(1000\ 0000)_{2s}$
Final 8-bit 2's Complement Representation: $(1000\ 0000)_{2s}$
Note: This is a special case where -128 is the lowest possible value in 8-bit two's complement.

Excess-128 encoding:
In Excess-128, we add 128 to the actual number before converting to binary: $-128+128=0$
0 in 8-bit binary: 0000 0000.
Final 8-bit Excess-128 Representation: **$0000\ 0000_{E-128}$**

(b) A system uses 4-bit base 4 digits (quaternary). What is the range of unsigned integers (in decimals) that can be represented in this system? Also, convert the base-4 number 3210.1 to binary.

Answers:
In a 4-bit base-4 (quaternary) system, each digit can take values 0, 1, 2, or 3.

The number of possible values for a 4-digit base-4 number is: 4^4=256
Range: 0000_4 (decimal: 0) to 3333_4 (decimal: 255)

To Binary: Each base-4 digit can be directly converted to binary (since 4=2^2, each base-4 digit maps to 2 binary bits i.e., 00 - 0, 01 - 1, 10 - 2 and 11 - 3).
Therefore, the base-4 number 3210.1_4 converts to binary as: **3210.1_4= 11 10 01 00.01$_2$**

Q3. **IEEE 754 Floating Point Numbers (2 x 3 = 6 marks)**
(a) Calculate the IEEE 754 single-precision floating-point representation of the decimal number -0.15625 and explain the steps involved in converting it from decimal to IEEE 754 format.

Answer:
Binary representation of -0.15625 is -0.00101_2
Normalized Binary representation of -0.15625 is -1.01 x 2^{-3}
Sign bit (1 bit): The original number is negative, so the sign bit is 1.
Mantissa (23 bits): From normalized binary we have 1.01, from which the 01 will become our mantissa, while adding 21 bits of 0's at the end which results to: 01000000000000000000000.
Exponent (8 bits):

- The exponent we calculated was -3, in order to compute actual representation on a IEEE 754, add a bias of +127. So: -3 + 127 = 124.
- This needs to be converted into an 8 bit binary: 124 to binary would be 01111100 which would be our exponent.

Final IEEE 754 representation is : 0xBE200000

1	01111100	01000000000000000000000

(b) Calculate the decimal equivalent of the IEEE 754 single-precision floating point number 0x42480000 and explain the steps involved in converting it from IEEE 754 format to decimal.

Answer:
First convert hexadecimal to binary: 0x42480000=0100 0010 0100 1000 0000 0000 0000 00002
Split the bits to IEEE 754 format:

Sign	Exponent	Mantissa
0	10000100	10010000000000000000000

Sign bit: 0 → Positive number

Exponent:
Convert the binary into decimal which is 132
To get the actual exponent, subtract the bias 127 which is 132-127 = 5
Mantissa: Assuming the leading implicit 1 and the normalized binary representation is 1.1001_2.
The final decimal number is calculated as: **$+1.1001 \times 2^5 = +50.0_{10}$**

Q4. Analyse the following MIPS code and answer the below questions. **(16 marks)**
Note: bge is a pseudo-instruction that branch-on-greater-than-or-equal. You will treat this as single instruction while answering the following questions.

```
#t0 - element to be found
#t2 - base address of array
#t3 - size of the int array

        addi $t4, $zero, 0      #instruction 1
l1:
        bge $t4, $t3, f2        #instruction 2
        sll $t5, $t4, 2         #instruction 3
        add $t6, $t2, $t5       #instruction 4
        lw $t7, 0($t6)          #instruction 5
        beq $t7, $t0, f1        #instruction 6
        addi $t4, $t4, 1        #instruction 7
        j l1                    #instruction 8
f1:
        addi $t1, $zero, 1      #instruction 9
        j exit                  #instruction 10
f2:
        addi $t1, $zero, 0      #instruction 11
exit:
```

(a) [2 Marks] Explain the purpose of the given MIPS code when \$t0 = 5, \$t2 = 0x1000, and \$t3 = 100. Assume the address 0x1000 contains the base address of an integer array. The element of the array takes values from 1 to 100. What operation does the code perform on the integer array stored at base address 0x1000?

Answer: The given MIPS code performs a linear search on an integer array stored at memory address 0x1000. The program iterates through the array to check if the value stored in \$t0 (which is 5) exists in the array.

(b) [3 Marks] For the given MIPS code and initial values, calculate the total number of instructions executed in the worst-case scenario.

Answer: The worst-case scenario is where the element is not found in the integer array. In this case the code searches the entire array and finally sets the $t1 to 0. The number of instructions is calculated as follows:
i1 is executed once - 1 instruction
i2 to i8 are executed 100 times - 7 x 100 = 700 instructions
i2 executed one extra time to check when $t4 = $t3 and takes branch - 1 instruction
i9 and i10 will not be executed
i11 will be executed once - 1 instruction
Total = 1+700+1+1 = 703 instructions

(c) [4 Marks] In the same scenario, determine how many times the branch instructions "bge $t4, $t3, f2" and "beq $t7, $t0, f1" are taken in both best-case and worst-case situations. You need to calculate how many times each branch instructions are ~~calculated~~ taken for best case and worst case separately.

Answer: (using when branch instructions are "**taken**")

Best-case scenario: When the element is found in the 0th location of the array. The bge instruction is not taken and only beq instruction takes the branch. Therefore, bge taken **0** time, beq taken **1** time.
Worst-case scenario: When the element is not found in the array. The bge instruction is taken only once when the $t4=$t3. The beq instruction is not taken.
Therefore bge taken **1** time, beq taken **0** time.

Answer: (using when branch instructions are "**executed**")

Best-case scenario: When the element is found in 0th location, then the bge and beq instructions are executed only once.
Worst-case scenario: When the element is not found in the array. The bge instruction is executed 101 times and the beg instruction is executed 100 times.

(d) [3 Marks] If you want to modify the given MIPS code to count the number of times $t0 appears in the array, what are the minimal changes required? Identify the specific instructions that need to be modified or added while keeping the structure of the original code intact.

Answer:
You need to initialize $t1 to 0.
When an element is found, you need to increase $t1 by 1 and then jump to the beginning of the loop l1. At the end of the loop, you need to branch to exit when all the elements in the array are checked. Below is one solution, there can be more than one solution.

```
t0 - element to be found
```

```
t2 - base address of array
t3 - size of the int array

    addi $t1, $zero, 0
    addi $t4, $zero, 0
    l1:
        bge $t4, $t3, exit
        sll $t5, $t4, 2
        add $t6, $t2, $t5
        lw $t7, 0($t6)
        beq $t7, $t0, f1
        addi $t4, $t4, 1
        j l1
    f1:
        addi $t1, $t1, 1
        addi $t4, $t4, 1
        j l1
    ~~f2:~~
        ~~addi $t1, $zero, 0~~
    exit:
```

(e) [4 Marks] From the original MIPS code, provide the hexadecimal encoding of the following instructions. You can assume the address of the first instruction is 0x00400020:

(i) sll $t5, $t4, 2
(ii) j l1

Answers:

(i) sll $t5, $t4, 2

Opcode: 000000 (since sll is an R-type instruction)
rs: 00000 (since sll does not use rs, it's set to zero)
rt ($t4): 01100 (register $12)
rd ($t5): 01101 (register $13)
shamt: 00010 (shift amount is 2)
funct: 000000 (function code for sll)
Binary Representation: (R format)

000000	00000	01100	01101	00010	000000

Hexadecimal Representation: 0x000C6880

(ii) j l1

Opcode: 000010 (6 bits)

Address of instruction pointed by l1: 0x00400024
The 4-bits of PC is 0000
Address: 0000 0100 0000 0000 0000 0010 01 (26 bits)

Binary Representation: (J format)

000010	0000 0100 0000 0000 0000 0010 01

Hexadecimal Representation: 0x08100009