CS2100 Computer Organization
AY2023/24 Semester 2
Assignment 1 [with ANSWER]

1. Tertiary (Base-3) Number System (Total: 8 marks)

(a) **Tertiary to Decimal Conversion** **(3 marks)**

You are given this unsigned tertiary (base-3) value in a tertiary number system with 8 digits for the integer part and 3 digits for the fraction part: N_3=21012202.111_3.
Convert N_3 to its decimal equivalent, correct to 4 decimal places.

Answer and Explanation: Multiply each digit by 3 raised to its position power:
$(2\times3^7)+(1\times3^6)+(0\times3^5)+(1\times3^4)+(2\times3^3)+(2\times3^2)+(0\times3^1)+(2\times3^0)+(1\times 3^{-1}) +(1\times 3^{-2}) +(1\times 3^{-3})$

Let's calculate this:
(2×2187)+(1×729)+(0×243)+(1×81)+(2×27)+(2×9)+(0×3)+(2×1)+(1x0.33333)+(1x0.11111)+(1x0.03704)
=4374+729+0+81+54+18+0+2+0.33333+0.11111+0.03704
=5258.48148
=5258.4815

Thus, the decimal equivalent of 21012202.111_3 is **5258.4815_{10}**

(b) **Conversion to binary** **(3 marks)**

Convert the tertiary number N_3=02210001.121_3 to its binary equivalent, correct to 4 binary places.

Answer and Explanation: First, convert 02210001.121_3 to decimal, following the process described in part (a), and then convert that decimal to binary.

$(0\times3^7)+(2\times3^6)+(2\times3^5)+(1\times3^4)+(0\times3^3)+(0\times3^2)+(0\times3^1)+(1\times3^0)+((1\text{x}3^{-1})+(2\text{x}3^{-2})+(1\text{x}3^{-3})$
=0+(2×729)+(2×243)+81+0+0+0+1+0.33333+(2× 0.11111)+0.03703
=2026.59258_{10}

Converting 2026.59258_{10} to Binary:
2026 in binary is calculated by repeatedly dividing by 2 and noting the remainders: 11111101010_2
0.59258 in binary is calculated by repeatedly multiplying by 2 and noting the carry: 0.100101_2.
Rounding to 4 binary places, we have 0.1001_2.
(If we use 0.5926 instead, we will still get 0.1001_2.)

Thus, the binary equivalent of 02210001.121_3 is **11111101010.1001_2**

(c) **Range of Representable Numbers** **(2 marks)**

Determine the range of numbers representable in a 4-digit tertiary number system in unsigned format.

Answer and Explanation: The smallest number is **0000_3 (0 in decimal)**, and the largest is **2222_3 (80 in decimal)**, offering a range of 0 to 80. In total, a 4-digit, base-3 number system can represent 3^4 = 81 numbers.

2. Base-4 Number System (Total: 4 marks)

(a) **3's complement and 4's complement:** **(2 marks)**

Calculate the 3's complement and 4's complement for a given 8-digit base-4 number on signed numbers, N_4=32103203.

Answers:

3's Complement:

- Subtract each digit from 3 to get the 3's complement: 32103203 -> 01230130
- Thus, the 3's complement of N_4=32103203 is **01230130**

4's Complement:

- First, subtract each digit from 3 to get the 3's complement: 32103203 -> 01230130
- Then, add 1 to the entire number in a base-4 manner: 01230130+1=01230131
- Thus, the 4's complement of N_4=32103203 is **01230131**

(b) **Range of Representable Numbers** **(2 marks)**

Determine the range of numbers representable in a 4-digit base-4 number system in (i) unsigned and (ii) signed (4's complement) format.

Answers:

Unsigned format: In an unsigned format, each digit can represent values from 0 to 3. For a 4-digit number in base-4, the range is straightforward:

- Minimum Value: The smallest number is represented by 0000_4, which is 0 in decimal.
- Maximum Value: The largest number is 3333_4, where each digit is the maximum value (3) in base-4. To convert 3333_4 to decimal, we calculate $3\times4^3+3\times4^2+3\times4^1+3\times4^0=255$ in decimal.

So, the range of unsigned 4-digit base-4 numbers is from **0 to 255** in decimal.

Signed (4's complement) format:

Positive values are represented by 0000_{4s} to 1333_{4s}. Negative values are represented by 2000_{4s} to 3333_{4s}.

Largest value (most positive): $1333_{4s} = (1\times4^3)+(3\times4^2)+(3\times4^1)+(3\times4^0) = 64+(3\times16)+(3\times4)+3 = 127$.

Smallest value (most negative): $2000_{4s} = -(2\times4^3) = -128$.

So, the range of signed 4-digit signed 4's complement format is from **–128 to 127**.

3. Excess-N Number Representation (Total: 6 marks)

(a) Convert the decimal number 25 to 8-bit excess-128 form. Explain the steps involved in the conversion process. **(2 marks)**

Answer:

- **Step 1:** Start with the decimal number 25.
- **Step 2:** Add the bias (128) to the number: 25+128=153.
- **Step 3:** Convert 153 to binary: 153=10011001.
- The excess-128 binary representation of 25 is $10011001_{\text{Excess-128}}$.

(b) Given an 8-bit binary number in excess-128 format, $10010110_{\text{Excess-128}}$, convert it back to its original decimal value. Describe the process used for conversion. **(2 marks)**

Answer:

- **Step 1:** Begin with the binary number 10010110.
- **Step 2:** Convert it to decimal: 10010110=150.
- **Step 3:** Subtract the bias (128) from the decimal representation: 150−128=22.
- The original decimal number before applying excess-128 encoding was $\mathbf{22_{10}}$.

(c) Range of Representable Numbers: Determine the range of decimal numbers that can be represented in an 8-bit excess-128 system. Explain how the excess-N system affects the representable range of numbers compared to the standard unsigned binary representation. **(2 marks)**

Answer:

- In an 8-bit system, the maximum binary number is 11111111, which is 255 in decimal.
- When using excess-128, the bias is subtracted from the binary representation, meaning the range of representable numbers goes from 0−128=−128 to 255−128=127.
- The excess-128 system allows representation of both positive and negative numbers in an 8-bit system, ranging from **−128 to 127**, unlike the standard unsigned binary representation, which ranges from **0 to 255**.

4. IEEE 754 Format (Total: 6 marks)

Consider the single-precision IEEE 754 format for this question.

(a) **Decimal to IEEE 754 Conversion:** Convert the decimal number −118.625 to its IEEE 754 single-precision floating-point representation. Write your answer in hexadecimal. Outline the steps involved, including normalization, binary conversion, exponent adjustment, and final encoding. **(3 marks)**

Answer:

1. **Sign:** Since the number is negative, the sign bit is 1.
2. **Normalization:** Convert −118.625 to binary: −1110110.101.
3. **Normalization (continued):** Normalize the binary number to 1.110110101×2^6
4. **Exponent:** The exponent is 6, and after adjusting with the bias (127 for single-precision), we get 133, which is 10000101.
5. **Mantissa:** The mantissa is the normalized value without the leading 1, filled to 23 bits: 11011010100000000000000.
6. **Final Encoding:** Combine the sign, exponent, and mantissa: 11000010111011010100000000000000.
7. **Convert to hexadecimal**: 1100/0010/1110/1101/0100/0000/0000/0000 = $\mathbf{C2ED4000_{16}}$.

(b) **IEEE 754 to Decimal Conversion:** Given the IEEE 754 single-precision floating-point number represented by the binary string 11000010111010000000000000000000, convert this binary string back into its decimal form. Describe the decoding process, including how to interpret the sign, exponent, and mantissa (fraction) fields. **(3 marks)**

Answer:

1. **Split** into sign-exponent-mantissa: 1 10000101 11010000000000000000000
2. **Sign:** The sign bit is 1, indicating a negative number.
3. **Exponent:** 10000101 is 133. Subtracting the bias (127) gives 6.
4. **Mantissa:** 11010000000000000000000. Including the implicit leading 1 gives 1.1101.
5. **Binary to Decimal:** Convert $-1.1101_2 \times 2^6$ to decimal: $-1.1101_2 \times 2^6 = -1110100_2 =$ **-116**.

5. MIPS (Total: 17 marks)

Study the MIPS program below. Mem[x] and Mem[y] are non-negative integers less than 10000. Assume that registers 4 – 15 are set to 0 prior to the execution of this program.

```
main:
    la    $8, x
    la    $9, y
    lw    $4, 0($8)
    lw    $5, 0($9)
    move  $11, $0
    beq   $5, $0, exit

loop:
    andi  $10, $5, 1
    neg   $10, $10
    and   $10, $10, $4
    add   $11, $10, $11
    srl   $5, $5, 1
    sll   $4, $4, 1
    bgei  $5, 1, loop

exit:
```

(a) There are four pseudo-instructions present in this MIPS program:

- `la` (load address, see lab 3)
- `move` (move value of one register to another)
- `bgei` (branch if greater than or equal to)
- `neg` (negate, in 2s complement).

Provide equivalent MIPS instruction(s) for the following pseudo-instructions:

- `move $dst, $src` (1 instruction)
- `bgei $src, imm, label` (2 consecutive instructions)
- `neg $dst, $src` (1 instruction)

Assume that there will be no overflow/underflow. Your equivalent MIPS instruction(s) should contain exactly the number of instructions specified. Use $t0 if you need a temporary register, and use $zero for the zero register. **(3 marks)**

Answers:

- `move $dst, $src`
  - **`add $dst, $src, $zero`** OR **`add $dst, $zero, $src`**
  - OR **`addi $dst, $src, 0`**
  - OR **`or $dst, $src, $src`** OR **`and $dst, $src, $src`**
  - OR **`or $dst, $src, $zero`**
- `bgei $src, imm, label`
  - **`slti $t0, $src, imm`**
  - **`beq  $t0, $zero, label`**
- `neg $dst, $src`
  - **`sub $dst, $zero, $src`**

(b) If Mem[x] = 2100 and Mem[y] = 24 at the start of the program, what is the value of register $11 at the end of the program? Write your answer in hexadecimal. **(2 marks)**

Answer: **0xC4E0**

As this program calculates the product between Mem[x] and Mem[y], we should get 50400_{10} = $C4E0_{16}$.

(c) In one sentence, explain the relationship between Mem[x], Mem[y], and the value of register $11 at the end of the program. **(2 marks)**

Answer: **The value in the $11 register is the product of Mem[x] and Mem[y].**

(d) Given Mem[x] = 2024 and Mem[y] = 2100 at the start of the program, determine the total number of times the `beq`/`bgei` instructions result in a branch during the execution of the program. **(2 marks)**

Answer: **11**

The beq instruction only branches if y = 0, which is not applicable in our case.

Note that the bgei instruction branches when $5 > 0. The contents in register $5 gets right-shifted (`srl`) once every loop. Therefore, the number of branches is one less than the number of digits in the binary representation of y, i.e. ($\log_2(y) - 1$) times. Since $2100_{10} = 1000\ 0011\ 0100_2$ which has 12 digits, this instruction branches 11 times.

Another way to get this answer is to consider the value of $5 every time the bgei instruction is encountered. In the first loop, $5 = 1050; in the second loop $5 = 525; in the third loop $5 = 262; and so on, counting the number of loops until $5 = 0 and the bgei does not branch.

(e) What is the minimum and maximum number of instructions which could be executed in this program? Assume that any of the pseudo-instructions used counts as only one MIPS instruction. **(2 marks)**

Answer: **minimum 6, maximum 104**

The minimum number of instructions executed would happen when y = 0, which results in the early branch to the exit label resulting in 6 instructions.

The maximum number of instructions executed would happen when the binary representation of y has the greatest number of digits. Since y < 10000, the largest possible value for y is $10\ 0111\ 0000\ 1111_2$, which would result in 14 iterations of the loop. This would result in 6 + (14 x 7) = 104 instructions being executed.

(f) Encode the instructions on lines 4, 7, 10, 12, 13, and 14 in hexadecimal. Assume that each pseudo-instruction used above counts as only one MIPS instruction. **(6 marks)**

Answer:
```
lw   $4,  0($8)       => 0x8D040000
beq  $5,  $0,  exit   => 0x10A00007
andi $10, $5,  1      => 0x30AA0001
and  $10, $10, $4     => 0x01445024
add  $11, $10, $11    => 0x014B5820
srl  $5,  $5,  1      => 0x00052842
```

Explanation:

```
    rt imm(rs)
lw $4, 0($8) => 0x8D040000
opcode = 23_16 = 0b10 0011
rs = $8 = 0b01000
rt = $4 = 0b00100
imm = 0 = 0b0000 0000 0000 0000
0b1000 1101 0000 0100 0000 0000 0000 0000 = 0x8D040000

      rs  rt  imm
beq   $5, $0, exit => 0x10A00007
opcode = 0x04 = 0b00 0100
rs = $5 = 0b00101
rt = $0 = 0b00000
imm = 7 = 0b0000 0000 0000 0111
0b0001 0000 1010 0000 0000 0000 0000 0111 = 0x10A00007

      rt   rs  imm
andi  $10, $5, 1     => 0x30AA0001
opcode = 0x0C = 0b00 1100
rs = $5 = 0b00101
rt = $10 = 0b01010
imm = 0b0000 0000 0000 0001
0b0011 0000 1010 1010 0000 0000 0000 0001 = 0x30AA0001

      rd   rs   rt
and   $10, $10, $4 => 0x01445024
opcode = 0 = 0b00 0000
funct = 0x24 = 0b10 0100
rs = rd = $10 = 0b01010
rt = $4 = 0b00100
shamt = 0b00000
0b0000 0001 0100 0100 0101 0000 0010 0100 = 0x01445024
  opcode rs    rt     rd     shamt funct

      rd   rs   rt
add   $11, $10, $11 => 0x014B5820
opcode = 0 = 0b00 0000
funct = 0x20 = 0b10 0000
rt = rd = $11 = 0b01011
rs = $10 = 0b01010
shamt = 0b00000
0b0000 0001 0100 1011 0101 1000 0010 0000 = 0x014B5820
  opcode rs    rt     rd     shamt funct

      rt  rd  imm
srl   $5, $5, 1     => 0x00052842
opcode = 0 = 0b00 0000
funct = 0x02 = 0b00 0010
rt = rd = $5 = 0b00101
rs = 0 for shift instructions
shamt = 0b00001
0b0000 0000 0000 0101 0010 1000 0100 0010 = 0x00052842
  opcode rs    rt     rd     shamt funct
```