CS-225 Fundamentals of Computer Systems

Assignment 1 September 25, 2025

Deadline: October 2, 2025 11:55PM

Total Marks: 100

Instructions

- **Blatant copying** from other students or the internet is strictly **forbidden**. If you have questions, post on the forum/Slack or contact your TA/Instructor.
- **Submission:** Submit a single PDF on LMS in the correct tab. Hand-written work may be scanned and uploaded. No late days/extensions. Include your **Name** and **Roll No.** on the front page.
- **Show working:** Unless explicitly stated otherwise, you must show steps for each conversion/derivation.
- **Use Slack and office hours:** Make use of the platform in case you have any confusions. The TAs are there to help you.
- **Solving digitally:** Typed assignments are not allowed, but those done using a stylus + tablet/iPad are allowed.
- **Use of AI:** You can use AI/LLMs to get guidance and help, but answers copied down from any generative AI will result in a strict action (DC).

Question 1 [5 + 5 = 10 marks]

(a) Convert the following decimal numbers into **8-bit two's complement** binary form:

(i) -45

(ii) $+117$

(iii) -100

(b) Given the 8-bit two's complement value `0x9A`, do the following:

(i) Sign-extend it to **12 bits**.

(ii) Truncate it to **5 bits**.

(iii) Briefly explain the difference in representation and its effect on the stored value.

Question 2 [4 + 4 = 8 marks]

(a) Let $x =$ `0b11010110` be an **unsigned** 8-bit integer. Compute:

(i) $x \ll 2$

(ii) $x \gg 3$

(iii) $(\sim x) \gg 4$ (assume **arithmetic** right shift with two's complement)

(b) Let $y = -23$ in 8-bit two's complement form.

(i) Represent y in 8-bit binary.

(ii) Show the result of a **logical** right shift by 2 and an **arithmetic** right shift by 2.

(iii) Briefly explain why the results differ.

Question 3 [4 + 3 + 3 = 10 marks]

(a) Addition, subtraction, and multiplication are the building blocks of anything remotely computational. Say goodbye to your days of being a pro at mental math unless you can at least do it in binary! Fill the table for **signed 8-bit** addition (`TAdd8`). Indicate the *True Sum* (8 or 9 bits if needed), the truncated `TAdd8` result (8 bits), and whether **overflow** occurred (Yes/No).

U (8-bit)	V (8-bit)	True Sum (8/9-bit)	TAdd8 (8-bit), Overflow?
`10110111`	`01011001`		
`11010110`	`11101110`		

(b) Now to really make you work for it! Below are two decimal numbers. Convert each to binary and compute $A - B$ (show working):

$$A = (793)_{10}, \quad B = (364)_{10}.$$

Note: Consider A and B as 12-bit numbers

(c) And now for the finale! The hardest thing you'll do in your CS degree: multipying integers. Convert each operand (in hex) to binary and multiply (show working). Record the **8-bit** product (truncate if necessary).

U (8-bit, hex)	V (8-bit, hex)	Binary Working	UMult (8-bit)
`0xB4`	`0x31`		
`0x4F`	`0x9E`		

Question 4 [10 + 11 = 21 marks]

Do you miss the cute homeworks you got in school? Where you had to draw smiley faces on worksheets, or help a lost rabbit find its way home through a maze, and life was gloriously uncomplicated? Well, your TAs have decided to once again break every rule of "serious assignments" to bring that energy back :)

This time, however, things have gotten... strange.

Your TA Sameer has been wandering around campus in a state of complete confusion. He swears he borrowed something extremely important, but he cannot for the life of him remember what it was, or from

whom. The only clue he has is that it was "big, heavy, and oddly suspiciously round."

Abdullah insists that he lent Sameer a calculator. Hussain, on the other hand, is absolutely certain that Sameer borrowed his airpods (which, according to Hussain, he definitely needs for "deep academic focus"). But Sameer isn't convinced either way - he just keeps mumbling that the missing item is the source of his lost mental peace.

Now it's up to you to solve the puzzle and help Sameer recover what he borrowed — before Abdullah files an FIR for theft and Hussain starts charging rental fees. Is it something that can even be "borrowed"?

(a) Use **4-bit** arithmetic (all results masked to 4 bits). Each expression evaluates to a 4-bit value which maps to a letter. Mapping:

$$\text{Hex } 0xA - 0xF \to \text{A-F}, \quad 0x2 \to \text{T},\ 0x0 \to \text{O},\ 0x4 \to \text{L},\ 0x3 \to \text{N},$$
$$0x7 \to \text{R},\ 0x5 \to \text{M},\ 0x6 \to \text{G},\ 0x8 \to \text{U},\ 0x9 \to \text{W}.$$

1. (`0x8` | `0x1`) & `0xF`
2. (`0xF` $\gg$ 1) ^ `0xD`
3. ((`0x7` $\ll$ 1) ^ `0xC`) & (`0x6` ^ `0x4`)
4. ($\sim$ `0x0`) ^ (`0x2` $\gg$ 1)
5. (`0xE` $\gg$ 1) | (`0x2` & `0x3`)
6. (`0xA` $\gg$ 1) & (`0x7` | `0x0`)
7. ($\sim$ `0x0`) & (`0xF` ^ `0x1`)
8. ($\sim$ `0xB`) & (`0x7` | `0x0`)
9. (`0x3` & `0x4`) ^ (`0x1` & `0x2`)
10. $\sim$ `0xC`

Answer:

(b) Use the mapping (note this mapping is *different* from part (a)):
$P \to$ `0x3` $\to$ `0011`, $Q \to$ `0xA` $\to$ `1010`,
$R \to$ `0x6` $\to$ `0110`, $S \to$ `0x9` $\to$ `1001`,
$T \to$ `0x0` $\to$ `0000`, $U \to$ `0xF` $\to$ `1111`.

Fill in each blank with one of { &, |, ^, $\sim$ } so that the expression evaluates to the given hex value (assume 4-bit results): (note: options 2-6 carry 2 marks each)

1. P __ R $\to$ `0x2` (1 mark)
2. Q __ S $\to$ `0xB`
3. R __ U $\to$ `0x9`
4. __(P __ T) $\to$ `0xC`
5. __(Q __ S) $\to$ `0x7`
6. __(S __ T) $\to$ `0x6`

Question 5 [5 + 5 + 10 = 20 marks]

(a) Convert $A = (42.42)_{10}$ to **IEEE-754 single-precision (32-bit)** floating-point. Give the final answer in **hexadecimal** (`0x....`). (5)
(b) Convert $B =$ `0xC2480001` (IEEE-754 single precision) to **decimal**. (5)
(c) Compute $C = A - B$ using IEEE-754 arithmetic. Give C in **hexadecimal**. Verify by converting A, B, and C to decimal. (10)

IEEE-754 Single Precision (32-bit) Layout

Sign (1 bit) **Exponent (8 bits)** **Fraction (23 bits)**
Bias = 127. Normalized value: $(-1)^s \times (1.f) \times 2^{e-127}$ for $1 \leq e \leq 254$.

Question 6 [20 marks]

(i)

(a) Write the decimal number 24 in **binary** as a **32-bit big-endian int**. (1)

Address 0x00	Address 0x01	Address 0x02	Address 0x03

(b) Write the decimal number 24 in **hexadecimal** as a **32-bit little-endian int**. (1)

Address 0x00	Address 0x01	Address 0x02	Address 0x03

(c) For each of the following, state if the big-endian and little-endian **representations** are the same or different (assume 32-bit `int`). (9; 1.5 each)

(i) The minimum **unsigned** number.
(ii) The maximum **unsigned** number.
(iii) +1 in two's complement.
(iv) −1 in two's complement.
(v) The maximum two's complement number.
(vi) The minimum two's complement number.

(d) Give (in **hex**) a 32-bit value that represents the same **unsigned** number in big-endian or little-endian and is *not* one of the choices above. (2)

(ii)

```
char S[6] = "15213";   // null-terminated string
int  N    = 0x12345678;
```

(a) Show the contents of memory starting at `0x2F00` for the string `S`. Indicate each byte in hex, including the null terminator. Assume a **little-endian** machine. (3)

Address	Byte (hex)
0x2F00	
0x2F01	
0x2F02	
0x2F03	
0x2F04	
0x2F05	

(b) Show the contents of memory starting at `0x2F06` for the `int N`. Indicate each byte in hex. Assume **little-endian**. (2)

Address	Byte (hex)
0x2F06	
0x2F07	
0x2F08	
0x2F09	

(c) Is there any difference in how the string `S` and the `int N` are stored? Explain briefly. (2)

Question 7 [5 + 6 = 11 marks]

PART 1 (5 marks)
Consider the following function. It takes two parameters as input (an integer array `arr` and the number of elements in the array `size`).

(a) Explain how the function correctly counts odd numbers, even when the array contains negative numbers. (Hint: explain using two's complement binary representation of negative numbers and LSB). /2 marks

```
int countOdds(int arr[], int size)
{
    int total_count = 0;
    for (int i=0; i<size; i++)
    {
        if (arr[i] & 1)
        {
            total_count++;
        }
    }
    return total_count;
}
```

(b) Explain how you would modify the code so it only counts positive odd numbers. /1 mark
(c) Now write an alternate implementation using bitwise operations for counting positive odd numbers without using the modulo operation of course. (Only provide the modified condition in this part). /2 marks

PART 2 (6 marks)

(a) Write a function that checks if a number is a power of 2 using bitwise operators. You're not allowed loops or %. (Hint: powers of 2 have only one bit set as 1, e.g., 0010 is 2^1, 0100 is 2^2, 1000 is 2^3 and so on). /4 marks

```
// make sure to code in C++ only, and take care of syntax
int isPowerOfTwo(int x) {
    // your code
```

(b) Consider the function below. What is the value of `sum` when we pass `x1 = 2147483647` and `x2 = 1`? Explain why we get this output. /2 marks

```
int addTwo(int x1, int x2) {
    int sum = x1 + x2;
    return sum;
}
```

Good luck!