Processor Design Semester Project:

You are a computer architect at Qualcomm. Your team has been asked to design a simple but complete processor prototype that will be used as a reference architecture for prototyping Internet of Things(IoT) applications.

Task 1: Data Systems

Objective: Implement conversion logic and data constraints for the entire system.

Motivation: "Before we build the actual Arithmetic and Logical Unit, we need a language. Our processor needs to speak Binary, but as humans, we need to interface with Hex and Decimal."

Release Date: 02-12-2026
Due Date: 02-22-2026
Programming Language:

1. Python(Preferable)
2. Give instructions to run code in any other language you feel comfortable using

Development Platform: you can use any IDE, make sure to include your name in the code comments at beginning of each code file - you can submit in your github accounts, share link with the submission

Design & Validation Tasks

Section1: You must implement and validate the following:

1. A 32-bit signed decimal input parser
2. Decimal → Binary (Two's Complement) conversion logic
3. Binary → Hexadecimal conversion logic
4. Binary → Decimal conversion logic
5. Overflow detection for out-of-range inputs
6. Saturation logic to prevent wrap-around
7. Configurable output format selection

Section2: Unit tests covering:

1. Positive values
2. Zero
3. Negative values
4. Boundary cases (min/max 32-bit values)

Expected Inputs & Outputs:

Inputs:

1. One decimal integer per invocation

2. One output format selector (decimal, binary, or hex)

Outputs:

1. Converted value in requested format
2. Overflow / saturation status (if applicable)

What we will learn after this task (Key Takeaways):

1. **Number systems:** Decimal, Binary, and Hexadecimal
2. **Signed arithmetic:** Two's Complement, carry vs. overflow, and bit-width limits

Functional Requirements (Task 1: Data Systems):

FR1 — Input (What the user provides)

1. *Input value:* one decimal signed integer (e.g., "123", "-45", "0")
2. *Output format selector:* one of {DEC, BIN, HEX}

Assumption: user input is always decimal. No binary/hex input is accepted.

FR2 — Processor Data Model (What the processor supports)

1. The processor is a 32-bit signed integer machine
2. Internal representation is Two's Complement
3. Valid representable range:
    a. MIN_INT32 = -2^{31} = −2,147,483,648
    b. MAX_INT32 = $2^{31} - 1$ = 2,147,483,647

FR3 — Conversion (What must be produced internally)

Given an input decimal x:

1. Convert x into a 32-bit binary string (Two's Complement)
2. The binary string must be exactly 32 bits
3. Store/operate internally only on this 32-bit representation

FR4 — Range / Overflow Detection (No math needed)

During conversion, detect if the input does not fit in signed 32-bit:

1. *Overflow condition:* `x < MIN_INT32` or `x > MAX_INT32`
2. *The module must output an overflow flag:* `overflow = 1` if overflow occurred, else `overflow = 0`

FR5 — Overflow Handling Policy (Saturation)

If overflow occurs, apply saturation (clamping), not wrap-around:

1. If `x > MAX_INT32` → clamp internal value to `MAX_INT32`
2. If `x < MIN_INT32` → clamp internal value to `MIN_INT32`

FR6 — Output Formatting (What the user can print)

From the final internal 32-bit value (after saturation if needed), produce output in the selected format:

1. DEC: signed decimal integer (e.g., `-12`)
2. BIN: 32-bit binary string (e.g., `111111...0100`)
3. HEX: 8-digit hexadecimal string (e.g., `0xFFFFFFF4` or `FFFFFFF4`—choose one and stay consistent)

FR7 — Status Output (What must be returned every time)

Every invocation must return:

1. `value_out` (in chosen format)
2. `overflow` flag (0/1)
3. `saturated` flag (0/1)

FR8 — Required Tests (Coverage expectations)

Your test suite must include at least:

1. A positive number (e.g., `123`)
2. Zero (`0`)
3. A negative number (e.g., `-123`)
4. Boundary values: `MAX_INT32, MIN_INT32`
5. Overflow values: `(MAX_INT32+1), (MIN_INT32-1)`