Project Title: Arithmetic and Logical Operations in a Simple ALU

Objective:
The goal of this project is to simulate a more comprehensive Arithmetic Logic Unit (ALU) by implementing advanced arithmetic and logical operations in C programming. This extended project builds on the foundational concepts of binary operations, bit manipulation, and modular design, equipping students with a deeper understanding of how low-level operations are handled in hardware.

Project Description:
In this project, you will develop a program in **C/C++** that reads and processes arithmetic and bitwise operations from a text file. The operations will include basic arithmetic (addition, subtraction) and bitwise operations (AND, OR, XOR, NOT, left shift, and right shift). The program will take the operation and operands from the input file, perform the corresponding operation, then output the result.

You will also implement error handling to ensure that the program correctly handles invalid operations, such as bit shifts that exceed the bit size, or unsupported operation codes. This project is designed to work with 32**-bit unsigned integers** only.

Functional Requirements:
Your program will read from an input text file with each line representing a single operation. Each line contains the following:

1. **Operation Code (Opcode)** - An integer that specifies the operation to be performed:
    1. ADD (addition)
    2. SUB (subtraction)
    3. AND (bitwise AND)
    4. OR (bitwise OR)
    5. XOR (bitwise XOR)
    6. NOT (bitwise NOT)
    7. LSL (logical shift left)
    8. LSR (logical shift right)
    9. EQ (Equal to) - checks if the operands are equal
    10. LT (Less than) - checks if the first operand is less than the second operand
    11. GT (Greater than) - checks if the first operand is greater than the second operand
2. **Operands** - Depending on the operation:
    - For **binary operations** (e.g., ADD, SUB, AND, OR, XOR) and compare operations (EQ, LT, GT), two operands (unsigned integers) are required.
    - For **bitwise NOT**, only one operand is required.
    - For **bitwise shift operations** (left or right), two operands are required: the number to shift and the number of positions.

- The operands will be in hexadecimal format starting with 0x. You might find it useful to use the uint32t in C/C++ to store the number as an unsigned 32-bit number.
- You are open to use any C++ library or resources to read or display hexadecimal value as output. (Hint: check std::hex)

3. Output format -

- The output of the operations (binary, bitwise) will be displayed as unsigned 32 bits hexadecimal format. (Check std::hex).
- If the output is out of range (more than 32 bits), only display the 32 bits. (Hint: use uint32t in C/C++ to store the number)
- Use bitwise operators (‘&’, ‘|’, ‘^’, ‘~’, ‘<<‘, ‘>>’) to perform the operations.
- In case of the compare operator (ET, GT, LT) write either true or false.
- Check invalid cases:
  - Unsupported Operation (invalid opcode).
  - Invalid Operand Count (missing or extra operands).
    - If the number of operands for each opcode is not correct (e.g. ADD 0xFF10 is invalid because ADD requires 2 operands or NOT 0xFF 0x1 is invalid because NOT only require 1 operand)
  - The second operand for the LSL and LSR always needs to be a positive number as this means how many times you are performing the shift operation.
  - The number of time you can perform the shift operation should not exceed the number of bits in the number being operated on. For our case, the values are stored as 32 bits so you not shift it 32 or more times.

Sample input: Check the given input text file.

```
ADD 0x1234 0x8765
SUB 0x32 0x14
AND 0xD 0x9
OR 0xAFF011
XOR 0x15 0xF
NOT 0xFFFFFFFF
LSR 0xAAA558 -0x1
LSL 0x40 0x21
Invalid 0xC 0x5
EQ 0xA 0xA
LT 0xA 0xF
GT 0xF 0xA
EQ 0xA
AND 0xA 0xF 0x1
ADD 0x1 0xFFFFFFFF
SUB 0xFFFFFFFF 0x20
5 0x20000 0x9
```

ADD 0x72DF9901 0x2E0B484A
LSL 0xFFFFFFFF 0x2
GT 0xA 0xF

Output format:

ADD 0x1234 0x8765 : <result>
SUB 0x32 0x14 : <result>
AND 0xD 0x9 : <result>
OR 0xAFF011 : <result>
XOR 0x15 0xF : <result>
NOT 0xFFFFFFFFF : <result>
LSR 0xAAA558 -0x1 : <result>
LSL 0x40 0x21 : <result>
Invalid 0xC 0x5 : <result>
EQ 0xA 0xA : <result>
LT 0xA 0xF : <result>
GT 0xF 0xA : <result>
EQ 0xA : <result>
AND 0xA 0xF 0x1 : <result>
ADD 0x1 0xFFFFFFFF : <result>
SUB 0xFFFFFFFF 0x20 : <result>
5 0x20000 0x9 : <result>
ADD 0x72DF9901 0x2E0B484A : <result>
LSL 0xFFFFFFFF 0x2 : <result>
GT 0xA 0xF : <result>

Instead of ***<result>***, write down the result for the specific operation.
For invalid cases, output a corresponding error message:

- **Unsupported Operation** for unsupported opcodes.
- **Shift Value Exceeds Bit Size** for invalid shifts.
- **Negative shift count** for negative number of shift count
- **Invalid Operand Count** when the number of operands does not match the operation.

Deliverables:

1. **Source Code:**
   - Submit a well-documented C/C++ program file. If you have multiple files such as C/C++ and header files, then provide a makefile with your submission to execute your work.
   - Make sure to add your name in your main.cpp file as commented out.
2. **Report**:
   - Describe how to run your code, your code's working process.
   - Explain the logic and functionality of each operation.
   - Include test cases for each operation and their expected outputs.
   - If there is any invalid cases, make sure to explain the reason.

3. Provide the input text that was given in the same folder.
4. Provide a screenshot of your output in the terminal.

Put everything in a zip file and submit the zip file. Rename the zip file with this format: *first-name_last-name_project1*

(Sometimes the code might not run from the TA's end as you can use different CPP library or compiler version. If it is running from your end don't worry. In that case make sure to contact me or the TAs as soon as you can to update your points)

No extensions will be given except as permitted by the course Syllabus. 30 minutes of grace will be given after the due time, assuming the students may face internet/submission/device issues.
However, after the grace period, all the submissions will be treated as LATE SUBMISSIONS.

IF YOUR SUBMISSION IS LATE BY 1 DAY, YOU WILL HAVE A 20% PENALTY. LATE SUBMISSIONS AFTER THAT WILL BE ACCEPTED FOR AT MOST 50% CREDIT.

In case of late submissions, the 30-minute grace in due time will not be given.

Extra credit (10 points):

If you want to perform multiplication or division by a power of 2, which operator from the given list would you use to calculate the result? Explain why this operator is the most suitable for such operations. Additionally, specify the input format based on the provided input conventions and illustrate with an example. Highlight this explanation in your project report documentation so the TA can easily locate and grade it. For example, if you want to multiply 0x0056 by 4 (2 power 2), how are you going to do this? Make sure to explain it for both multiplication and division.

Grading Criteria:

- Correctness: (50)
  - The program performs all required operations accurately.
  - 2.5 points for each input line
- Modularity and Code Design: (20)
  - Clear, reusable, and well-organized code structure.
  - Are you reading the input text file correctly? Are you displaying the output correctly with proper hexadecimal format?
- Data type and format Handling: (10)
  - Properly handling the data type that means you have to work with unsigned 32 bits of data, properly handling the 0x prefix for the hexadecimal values

- Documentation: (20)
  - Clear comments in code and a concise project report. In the report, you have to explain all the operations and inputs. Make sure to explain your result output.
- Extra credit: (10)
  - Check the requirement mentioned before.