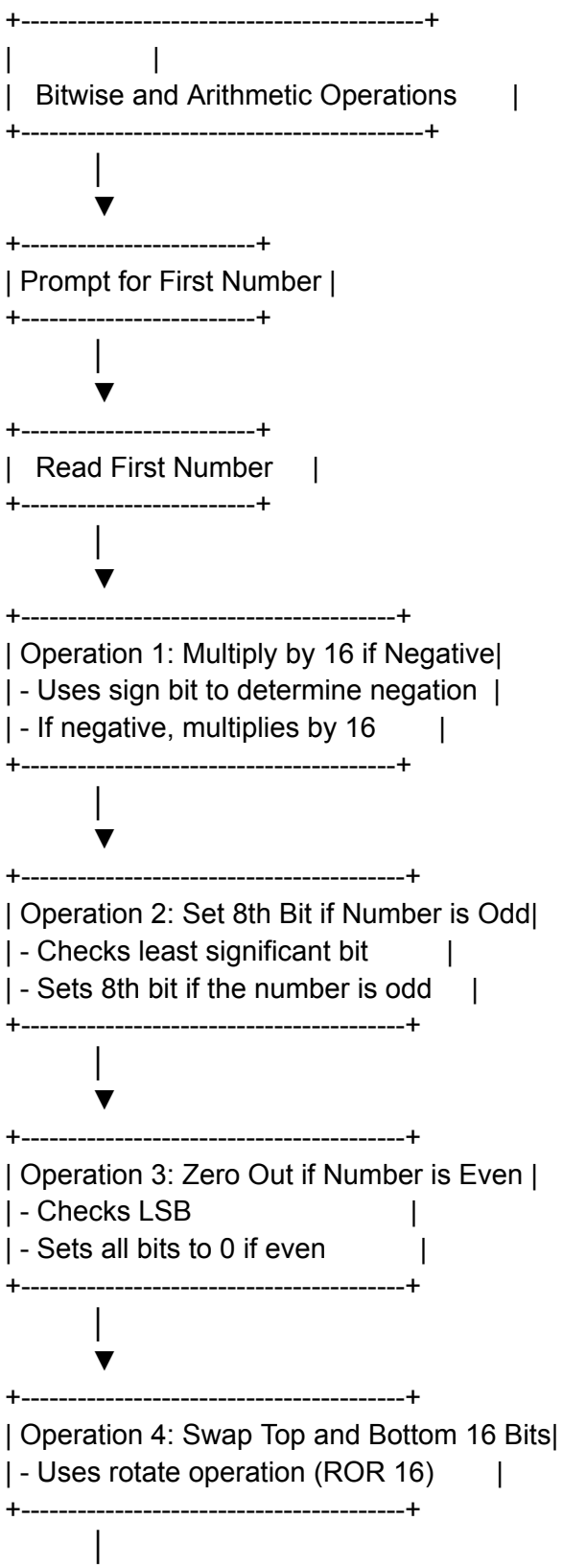
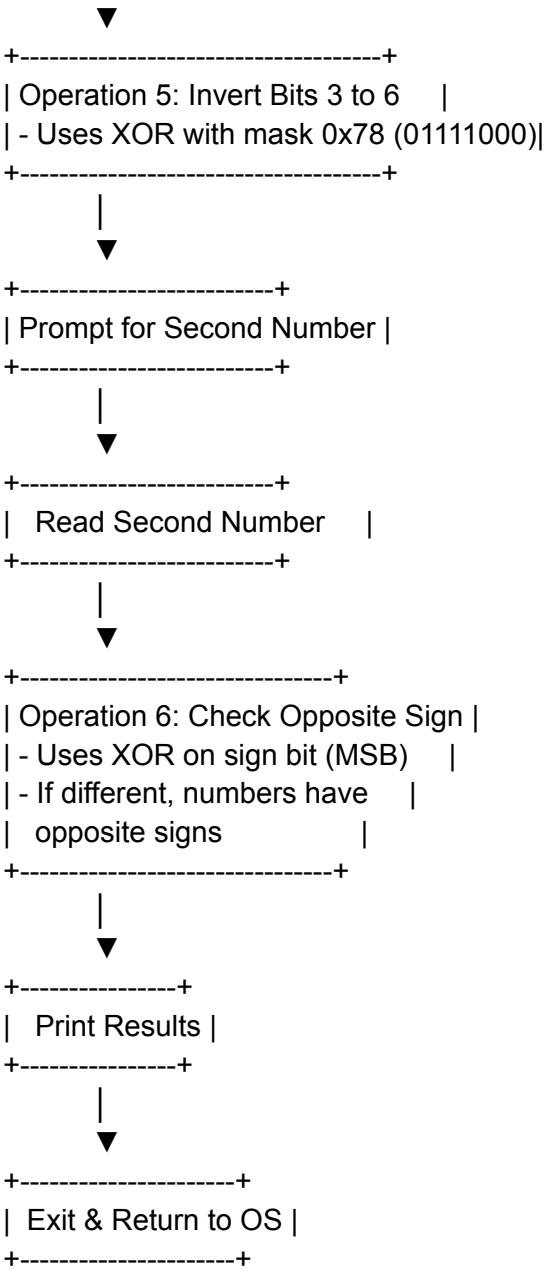
Planning:


Bitwise and Arithmetic Operations
Prompt for First Number
Read First Number
Operation 1: Multiply by 16 if Negative
- Uses sign bit to determine negation
- If negative, multiplies by 16
Operation 2: Set 8th Bit if Number is Odd
- Checks least significant bit
- Sets 8th bit if the number is odd
Operation 3: Zero Out if Number is Even
- Checks LSB
- Sets all bits to 0 if even
Operation 4: Swap Top and Bottom 16 Bits
- Uses rotate operation (ROR 16)



Bit Twiddling Operations

1. Multiply by 16 if the number is negative

- **Logic**:
  - Check if the number is negative by shifting right (SAR) by 31 bits.
  - If negative, compute `num * 15`, then add it to the original number (equivalent to multiplying by 16).
- **Example**: `-5 → -80` (bitwise computed as `-5 + (-5 * 15)`).

2. Turn on the 8th bit if the number is odd

- **Logic**:
  - Check if the number is odd by isolating bit 0 (`AND` with `1`).
  - If odd, shift this value left by 8 and OR it with the original number.
- **Example**: `19 (00010011) → 275 (00010011 | 00000000 00000001 00000000)`.

3. Zero out all bits when even

- **Logic**:
  - Check if the number is even (`AND` with `1` to isolate LSB).
  - Negate this bitwise result to create a mask (`0xFFFFFFFF` if odd, `0x00000000` if even).
  - Apply `AND` operation to zero out the number if even.
- **Example**: `4 (even) → 0`.

4. Swap the top 16 bits and the bottom 16 bits

- **Logic**:
  - Perform a **rotate right by 16** (ROR 16) to swap the top and bottom 16 bits.
- **Example**:
  - `4 (00000000 00000000 00000000 00000100)`
  - → `262144 (00000000 00000100 00000000 00000000)`.

5. Invert the bits 3-6

- **Logic**:
  - XOR the number with `0x78` (`01111000` in binary) to toggle bits 3-6.
- **Example**:
  - `4 (00000000 00000000 00000000 00000100)`
  - → `124 (00000000 00000000 00000000 01111100)`.

6. Determine if 2 numbers have opposite signs

- **Logic**:
  - XOR the two numbers and check if bit 31 (sign bit) is `1`.
  - If `1`, numbers have opposite signs, so print `1`; otherwise, print `0`.
- **Example**: `-6 and 4 → 1 (opposite signs)`

Test case:

```
Enter a number: 4
```

- x16 if negative: 4
- Turn on 8th bit if odd: 4
- Zero out if even: 0
- Swap top & bottom 16 bits: 262144
- Invert bits 3 - 6: 124
- Enter a second number: 5
- Opposite sign? 0
-
-

-

Follow Up Question

1) yeamin. 5

2) 4 hour

3) I thing it was easy

4) understanding of bitwise operations and their practical applications in assembly language.

5) yes. For fixing error.