simplest program to fix this?

Instruction is at location Mem[0]. What is the
we meant to write SUB X0, X1, X4. The
WPC: 0000 0110 0000 1000 1100 1000 0000 1000

Review Problem 8
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Rs</th>
<th>Rt</th>
<th>SHAMT</th>
<th>Rn</th>
<th>Rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **STRR** X0, [X31, X0, #3]  
- **ORR** X0, X0, X1 // OR ADD  
- **LST** X1, X1, #30  
- **ADDI** X1, X31, #1  
- **LDRR** X0, [X31, #0]  

**ADD is opcode 0x458, SUB is 0x658, so set bit 30:**

- **ADD** is a simplest program to fix this?

**Instruction is at location Mem[0]. What’s the we meant to write SUB X0, X1, X4. The we goter, and wrote: ADD X0, X1, X4, when Review Problem 8**
Compute the sum of the values 0, 1, ..., N-1

Conversion example
Directly control the CPU

Ex: Read from the keyboard

Assembly Language

Almost (but not always) intermediate to machine language

Moore's Law: for humans

Simple instructions
Introduce shifters, multipliers, etc.

Develop Arithmetic Logic Units (ALUs) to perform CPU functions.

Review binary numbers, 2's complement

Readings: 3.1-3.3, A.5

Computer Arithmetic
Example: \(01110101 \times (2)^0 = 69\)

Binary: \(01101 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (13)^10\)

Decimal: \(469 = 4 \times 10^2 + 6 \times 10^1 + 9 \times 10^0\)
2's Complement Numbers

To interpret numbers, convert to positive version, then convert:

- 01100 = 12
- 11010 = -11010
- 01100 + (8 + 4) = +12

Example: 

- (01101) = 10101 + 1 = 11010

Negation: Flip all bits and add 1.

Positive numbers & zero have leading 0. Negative have leading 1.
EX - Convert to 8-bit: 01101 = (13) \(_1^0\)

Conversion of n-bit to (n+m)-bit 2's complement: replicate the sign bit

\[ \overline{b_3b_2b_1b_0} = b_3\overline{b_2}\overline{b_1}b_0 = b_3b_2b_1b_0 \]

\[ \overline{11101} = (-3) \(_1^\text{10}\) \]

Sign Extension
### Arithmetic Operations

#### Binary

1. \[ \begin{array}{c}
        & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
    + & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
    \hline
    = & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
    \end{array} \]

2. \[ \begin{array}{c}
        & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
    + & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
    \hline
    = & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
    \end{array} \]

#### Decimal

1. \[ \begin{array}{c}
        3 & 6 & 8 & 4 & 8 \\
    + & 7 & 8 & 9 & 2 \\
    \hline
    = & 1 & 1 & 1 & 0 & 0 \\
    \end{array} \]

#### Binary

1. \[ A - B = A + (-B) = A + (B + 1) \]
Operations can create a number too large for the number of bits. Can detect overflow in addition when highest bit has carry-in ≠ carry-out.

n-bit 2's complement can hold \(-2^{(n-1)} \ldots 2^{(n-1)}-1\).