Mark xo,

\[ x_1 = 1 \]

// Load 53.5, 2

\[ \text{str} \ x_0, \text{C} \times 31, \#07 \]
\[ \text{or} \ x_0, x_0, x_1 \]
\[ \text{lsl} \ x_1, x_1, \#30 \]
\[ \text{add} \ i, x_1, x_3, \#1 \]
\[ \text{ldr} \ x_0, [x_31, \#07] \]

\[ \text{addr} \ 0 \times 458 / 546 \times 658 \]

The simplest program to fix this?

Instruction is at location Mem[0]. What's the instruction we meant to write SUB X0, X0, X4. The we meant to write: ADD X0, X1, X4, when we goofed, and wrote: ADD X0, X1, X4, when

---

Review Problem 8

SUBR 7x51

- ADDI
Compute the sum of the values 0...N-1

Conversion example
Direct Consideration

Pi-kind Semantics

Numbers for Everything

Machine Language

Assembly

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
Binary Numbers

Example: 0111010101 = (2)10

Binary: 011101 = 1*3 + 1*2 + 0*2 + 1*2^0 = (13)10

Decimal: 469 = 4*10^2 + 6*10^1 + 9*10^0

469

\[ \overline{1+4+16+64+128+256} \]

384

0 80 5
To interpret numbers, convert to positive version, then convert:

EX: \(-01101\)²

Negation: Flip all bits and add 1

Positive numbers & zero have leading 0, negative have leading 1

2's Complement Numbers
\[ x = -3 \]
\[ = -(01000001 + 1) \]
\[ = -10111111 \]
\[ = 10111111 \]

\[ \text{EX - Convert to 8-bit: 01101} = (13)_{10} \]
\[ q_3q_2q_1q_0 = q_3q_2q_1q_0 = \overline{b_3b_2b_1b_0} = \overline{b_3b_2b_1b_0} \]

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

Sign Extension
Can detect overflow in addition when highest bit has carry-in ≠ carry-out

Operations can create a number too large for the number of bits

Overflow
Full Adder

Checkered Board
Multi-Bit Addition
Adder/Subtractor

\[ A - B = A + B + 1 \]
\[ A + B = A + B + 0 \]
What does the number 1000112 represent?

Review Problem 9
Complex circuits require careful debugging.
endmodule
fulladd pos2 (cout(c[1]), s(s[1]), A(a[1]), B(b[1]), C(c[1]));
fulladd pos1 (cout(c[0]), s(s[0]), A(a[0]), B(b[0]), C(c[0]));
fulladd pos0 (cout(c[0]), s(s[0]), A(a[0]), B(b[0]), C(c[0]));

module greaterThan7 (Out, A, B);
endmodule

fulladd A1 (cout, s, A, B, C);

output Cout, s, input A, B;
module halfadd (cout, s, A, B);
endmodule

assign S = A&B'Cin;
assign Cout = (A&!B) | (A&Cin) | (B&Cin);
DO NOT RIP UP, DEBUGE!

Look at values at intermediate points in circuit.

Typically, trace backwards from bad outputs, forward from inputs.
Examine inputs and outputs to find earliest place where value is wrong.

Identity any incorrect behaviors.

All combinations of inputs for small circuits, subcircuits.
Test all behaviors.

Debugging Approach
ALU: Arithmetic Logic Unit

Computes arithmetic & logic functions based on controls

... ==, '>', overflow, ...
XOR, AND, NAND, OR, NOR
Add, subtract

Overflow
Negative
Zero

R

Control
Select
Function

A

64

B

64
Optional Shift by two: (A >> b^1)

Optional Shift by one: (A >> b^0)

Support Shift Operations: (A >> 001101)

Shifter