Addition and Subtraction in Hardware

- The same bit manipulations work for both unsigned and two's complement numbers!
  - Perform subtraction via adding the negated 2nd operand:
    $A - B = A + (-B) = A + (\sim B) + 1$

- 4-bit examples:

		Two's	Un
	0 0 1 0	+2	2
+	1 1 0 0	−4	12

		Two's	Un
	1 0 0 0	−8	8
+	0 1 0 0	+4	4

	0 1 1 0	+6	6
−	0 0 1 0	+2	2

	1 1 1 1	−1	15
−	1 1 1 0	−2	14

Half Adder (1 bit)

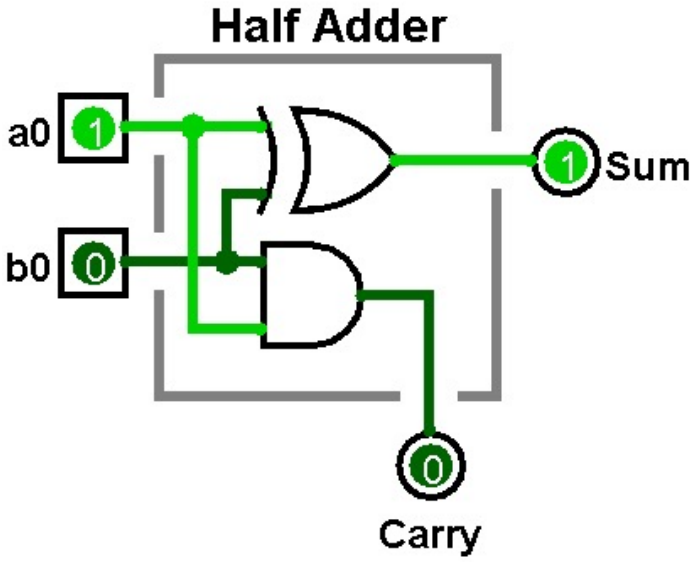
$$
\begin{array}{cccc|c}
 & a_3 & a_2 & a_1 & a_0 \\
+ & b_3 & b_2 & b_1 & b_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

Carry-out bit

a_0	b_0	c_1	s_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\text{Carry} = a_0 b_0$$

$$\text{Sum} = a_0 \oplus b_0$$



Full Adder (1 bit)

Possible carry-in c_1

$$\begin{array}{cccc|c} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline & s_3 & s_2 & s_1 & s_0 \end{array}$$

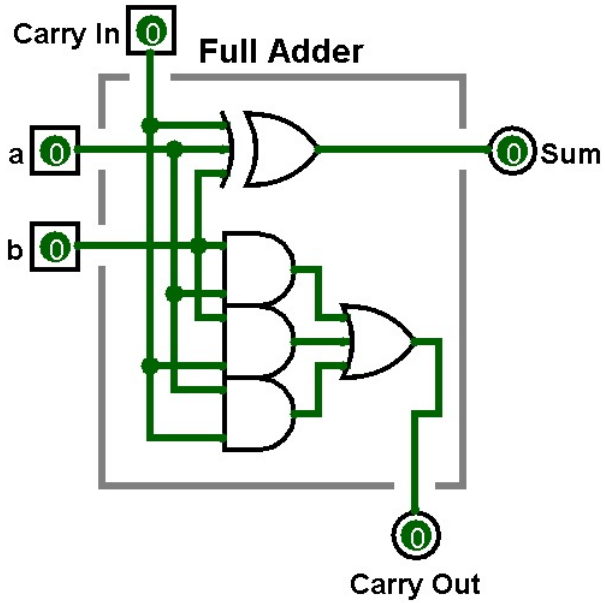
Carry-in ↘ ↙ Carry-out

c_i	a_i	b_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$\boldsymbol{s_i} = \mathrm{XOR}(a_i, b_i, c_i)$$

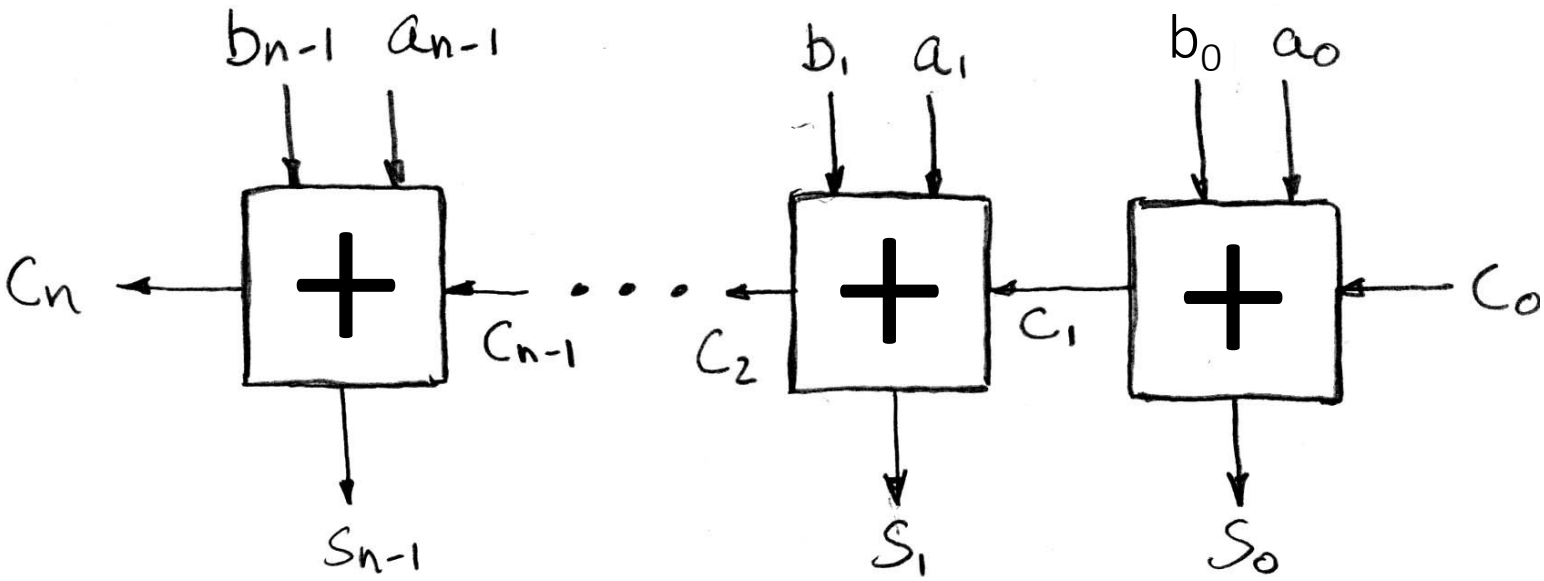
$$\boldsymbol{c_{i+1}} = \mathrm{MAJ}(a_i, b_i, c_i)$$

$$= a_i b_i + a_i c_i + b_i c_i$$



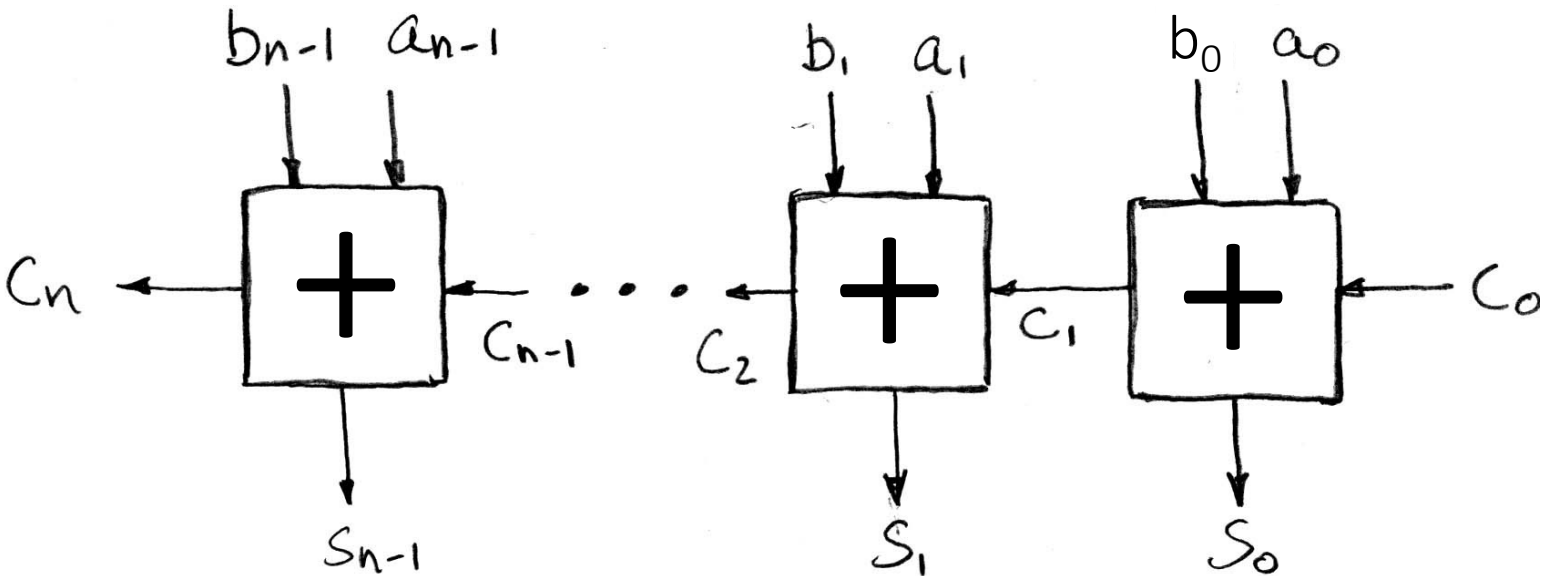
Multi-Bit Adder (N bits)

- Chain 1-bit adders by connecting $CarryOut_i$ to $CarryIn_{i+1}$:

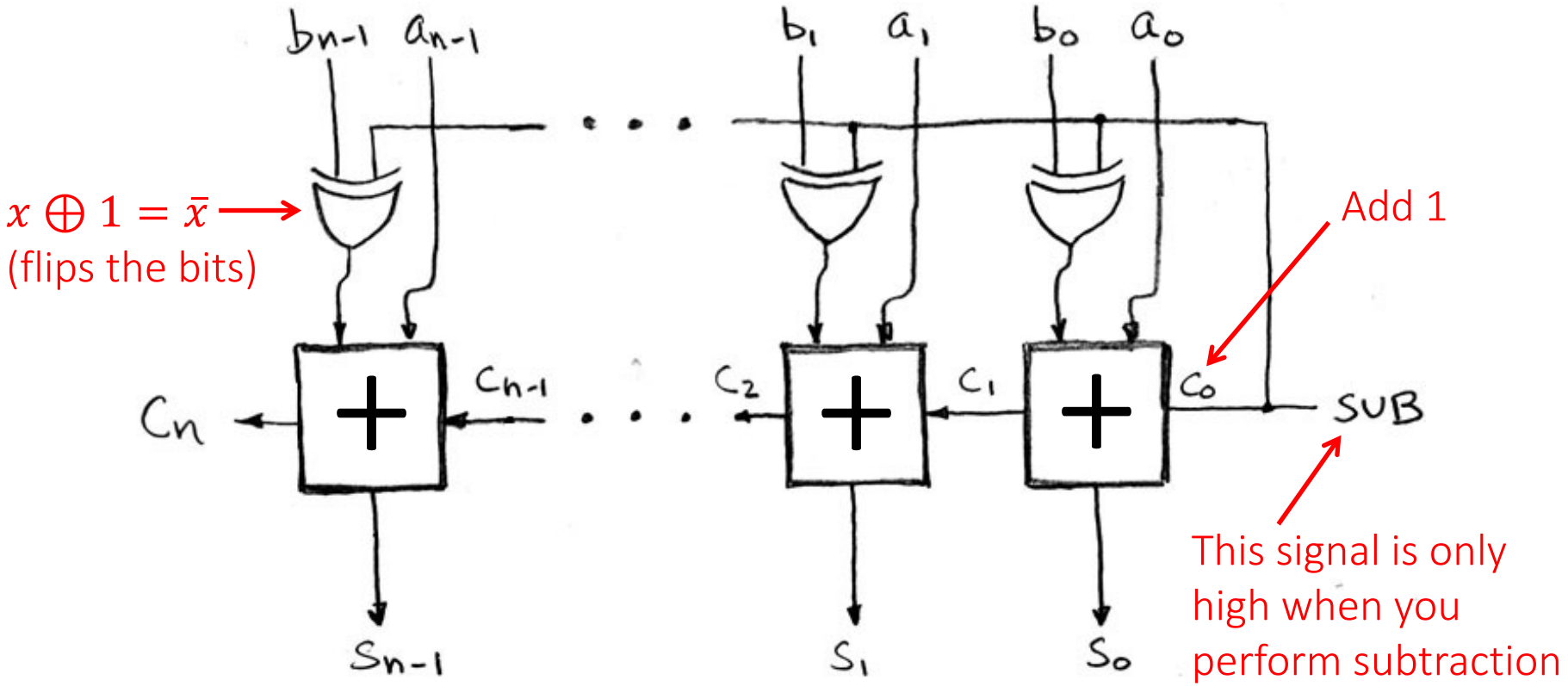


Subtraction?

- Can we use our multi-bit adder to do subtraction?
  - Flip the bits and add 1?
    - $X \oplus 1 = \overline{X}$
    - $CarryIn_0$ (using full adder in all positions)



Multi-bit Adder/Subtractor



Detecting Arithmetic Overflow

- Overflow: When a calculation produces a result that can't be represented in the current encoding scheme
  - Integer range limited by fixed width
  - Can occur in both the positive and negative directions

- Unsigned Overflow
  - Result of add/sub is > UMax or < Umin

- Signed Overflow
  - Result of add/sub is > TMax or < TMin
  - (+) + (+) = (−) or (−) + (−) = (+)

Signed Overflow Examples

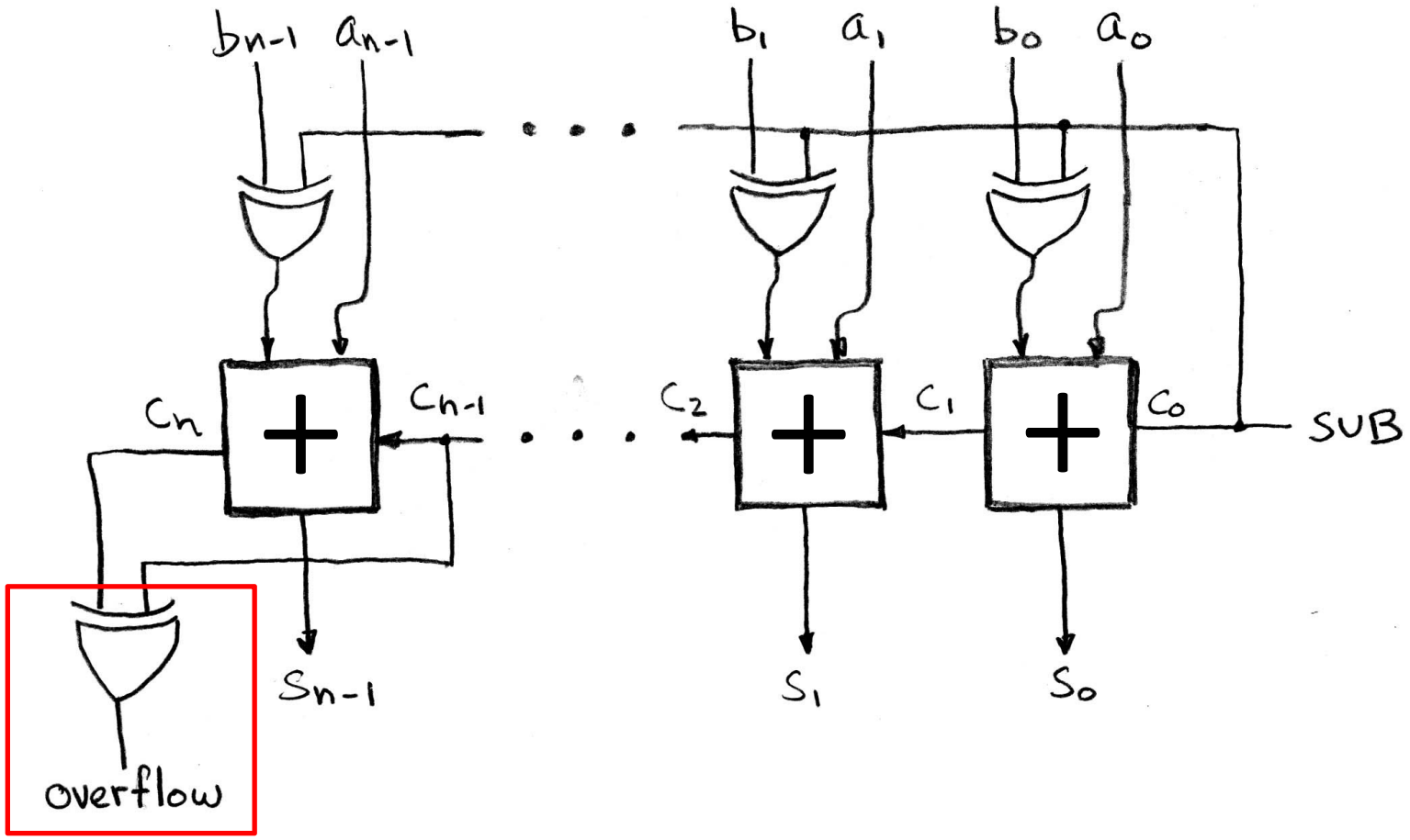
	Two's	
	0 1 0 1	+5
+	0 0 1 1	+3

	Two's	
	1 0 0 1	−7
+	1 1 1 0	−2

	Two's	
	0 1 0 1	+5
+	0 0 1 0	+2

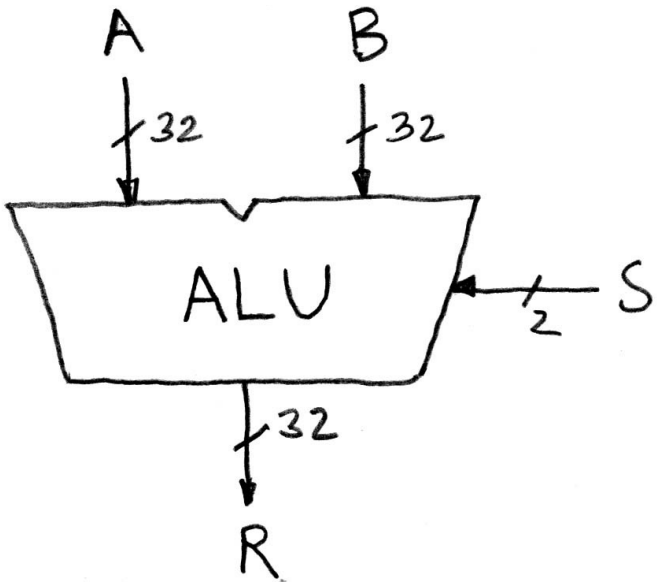
	Two's	
	1 1 0 1	−3
+	1 0 1 1	−5

Multi-bit Adder/Subtractor with Overflow



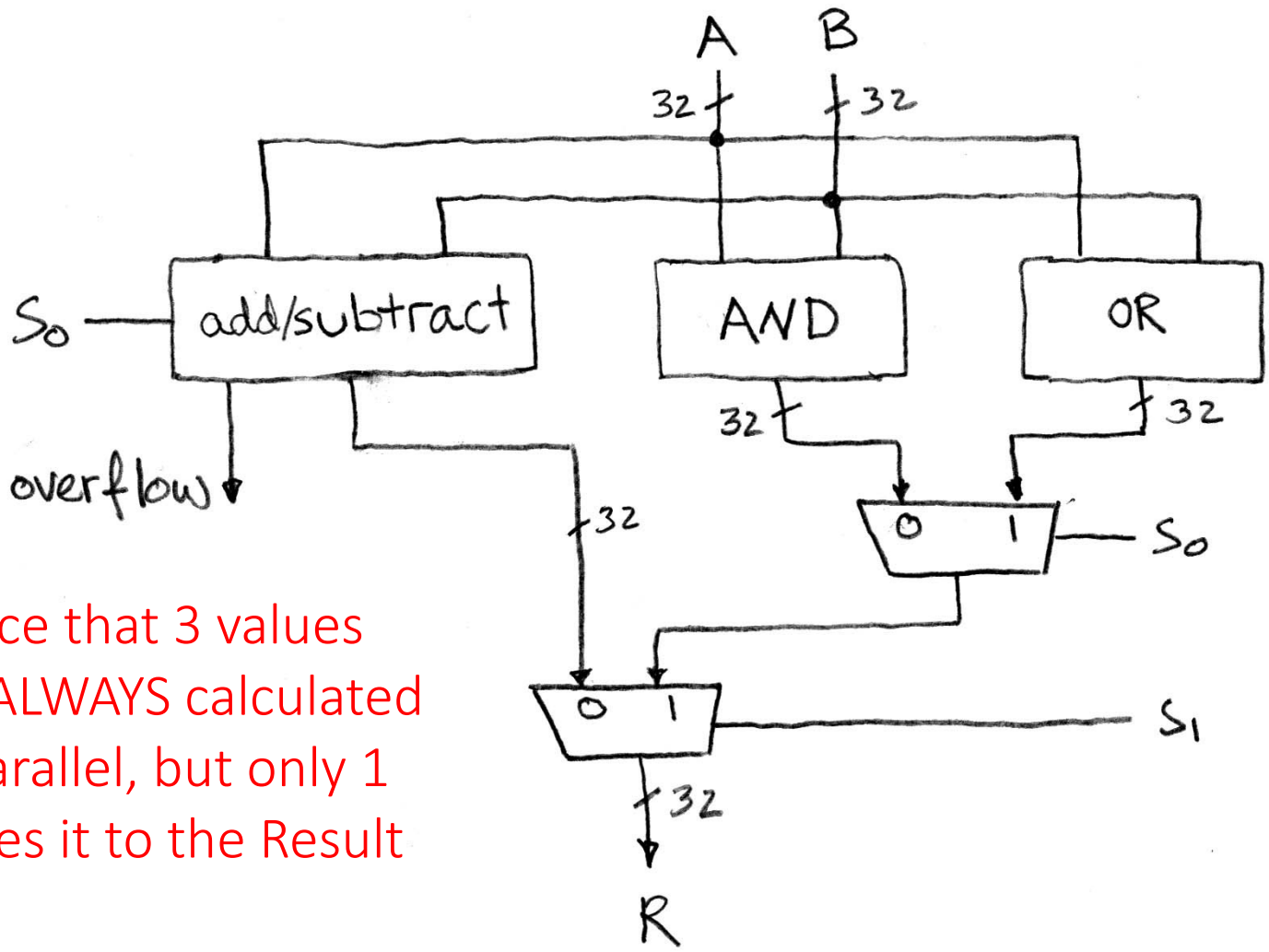
Arithmetic and Logic Unit (ALU)

- Processors contain a special logic block called the "Arithmetic and Logic Unit" (ALU)
  - Here's an easy one that does ADD, SUB, bitwise AND, and bitwise OR (for 32-bit numbers)
- Schematic:



when S=`00`, R = `A+B`
when S=`01`, R = `A-B`
when S=`10`, R = `A&B`
when S=`11`, R = `A|B`

Simple ALU Schematic



Notice that 3 values are ALWAYS calculated in parallel, but only 1 makes it to the Result

1-bit Adders in Verilog

- What's wrong with this?
  - Truncation!

```
module halfadd1 (s, a, b);
  output logic s;
  input  logic a, b;

  always_comb begin
    s = a + b;
  end
endmodule
```

- Fixed:
  - Use of `{sig, …, sig}` for *concatenation*

```
module halfadd2 (c, s, a, b);
  output logic c, s;
  input  logic a, b;

  always_comb begin
    {c, s} = a + b;
  end
endmodule
```

Ripple-Carry Adder in Verilog

```
module fulladd (cout, s, cin, a, b);
  output logic cout, s;
  input  logic cin, a, b;

  always_comb begin
    {cout, s} = cin + a + b;
  end
endmodule
```

- Chain full adders?

```
module add2 (cout, s, cin, a, b);
  output logic cout; output logic [1:0] s;
  input  logic cin;  input  logic [1:0] a, b;
  logic  c1;

  fulladd b1 (cout, s[1], c1,  a[1], b[1]);
  fulladd b0 (c1,   s[0], cin, a[0], b[0]);
endmodule
```

Add/Sub in Verilog (parameterized)

- Variable-width add/sub (with overflow, carry)

```
module addN #(parameter N=32) (OF, CF, S, sub, A, B);
  output logic          OF, CF;
  output logic [N-1:0] S;
  input  logic          sub;
  input  logic [N-1:0] A, B;
  logic  [N-1:0] D;     // possibly flipped B

  always_comb begin
    D = B ^ {N{sub}};  // replication operator
    {CF, S} = A + D + sub;
    OF = (~S[N-1] & A[N-1] & D[N-1]) |
         (S[N-1] & ~A[N-1] & ~D[N-1]);
  end
endmodule
```

- Here using `OF` = overflow flag, `CF` = carry flag
  - From condition flags in x86-64 processors
- `{n{m}}` is replication operator – repeats value `m`, `n` times

Add/Sub in Verilog (parameterized)

```
module addN_testbench ();
  parameter N = 4;
  logic          sub;
  logic [N-1:0] A, B;
  logic          OF, CF;
  logic [N-1:0] S;

  addN #(.N(N)) dut (.OF, .CF, .S, .sub, .A, .B);

  initial begin
    #100;  sub = 0;  A = 4'b0101;  B = 4'b0010;  //  5 +  2
    #100;  sub = 0;  A = 4'b1101;  B = 4'b1011;  // -3 + -5
    #100;  sub = 0;  A = 4'b0101;  B = 4'b0011;  //  5 +  3
    #100;  sub = 0;  A = 4'b1001;  B = 4'b1110;  // -7 + -2
    #100;  sub = 1;  A = 4'b0101;  B = 4'b1110;  //  5 -(-2)
    #100;  sub = 1;  A = 4'b1101;  B = 4'b0101;  // -3 -  5
    #100;  sub = 1;  A = 4'b0101;  B = 4'b1101;  //  5 -(-3)
    #100;  sub = 1;  A = 4'b1001;  B = 4'b0010;  // -7 -  2
    #100;
  end
endmodule
```