Building an ALU (Part 1):

Today's lecture

- **We start building our computer!**
  - We'll start with the arithmetic/logic unit (ALU)

- **Adding single bits**
  - Half Adders and Full Adder
- **Multi-bit Arithmetic**
  - Hierarchical design
  - Subtraction
- **Building a Logic Unit**
  - Multiplexors

Arithmetic Logic Units (ALUs)

- **The computation in a computer processor takes place in the arithmetic logic unit. This unit performs:**
  - Arithmetic operations
    - e.g., addition and subtraction
  - Bit-wise logical operations
    - e.g., AND, OR, NOT, XOR

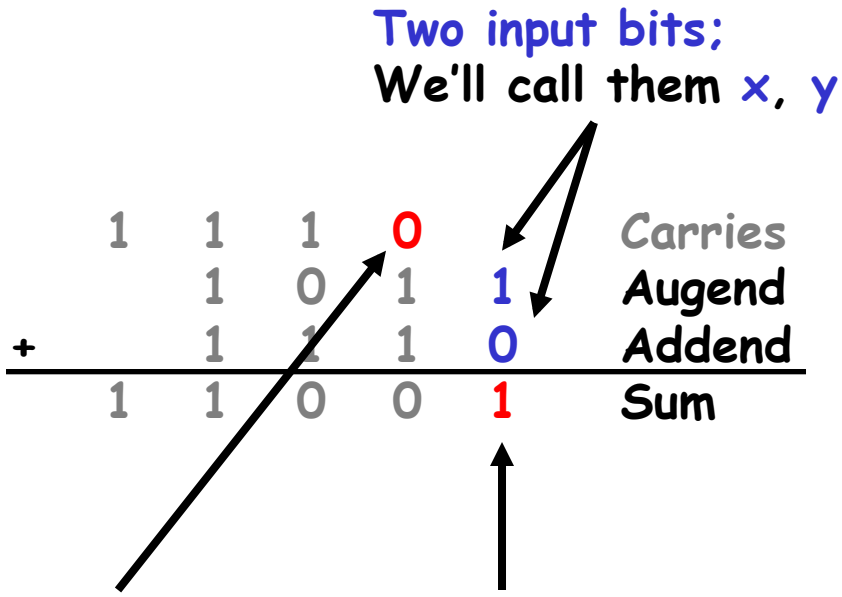
- **Typically these operations are performed on multi-bit words**
  - The MIPS-subset processor we will build uses 32-bit words

In Lab 3 you will build a 32-bit ALU with the above operations

Binary Addition Review

	1	1	1	0		Carries
		1	0	1	1	Augend
+		1	1	1	0	Addend
	1	1	0	0	1	Sum

First Bit Position

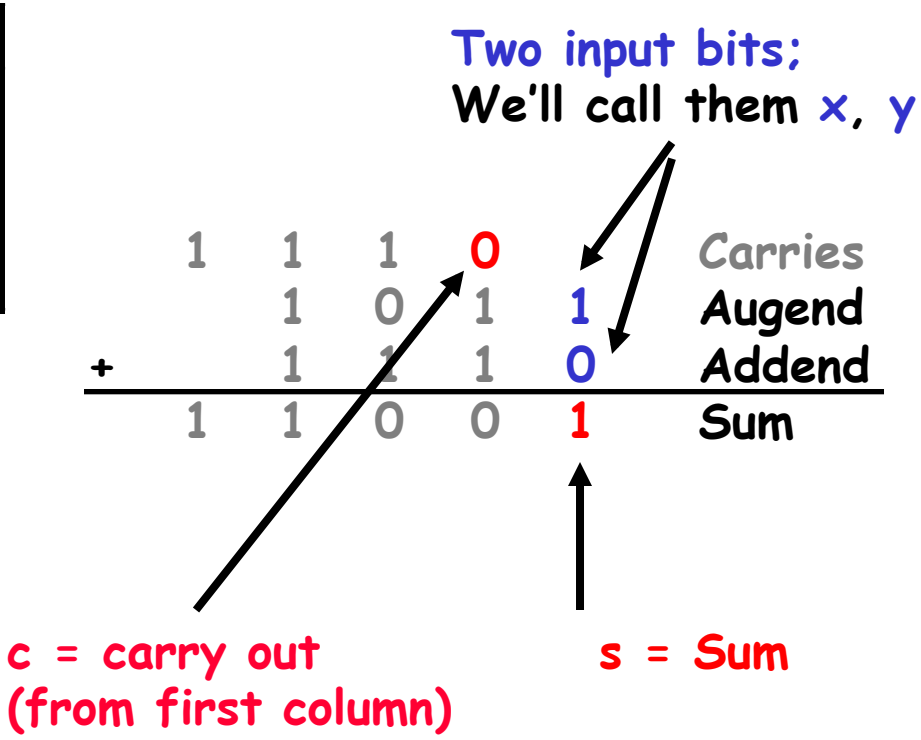


Two output bits: c = carry out (from first column)

s = Sum

First Bit Position's Truth Table

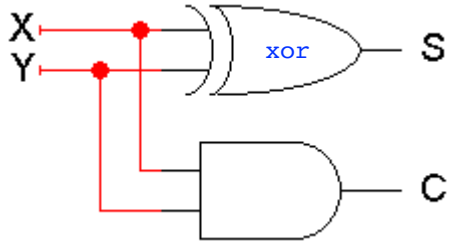
X	Y	C	S
0	0		
0	1		
1	0		
1	1		

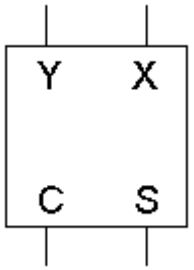


Half Adder

- **Adds two input bits to produce a sum and carry out.**

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





C = XY

S = X'Y + XY'

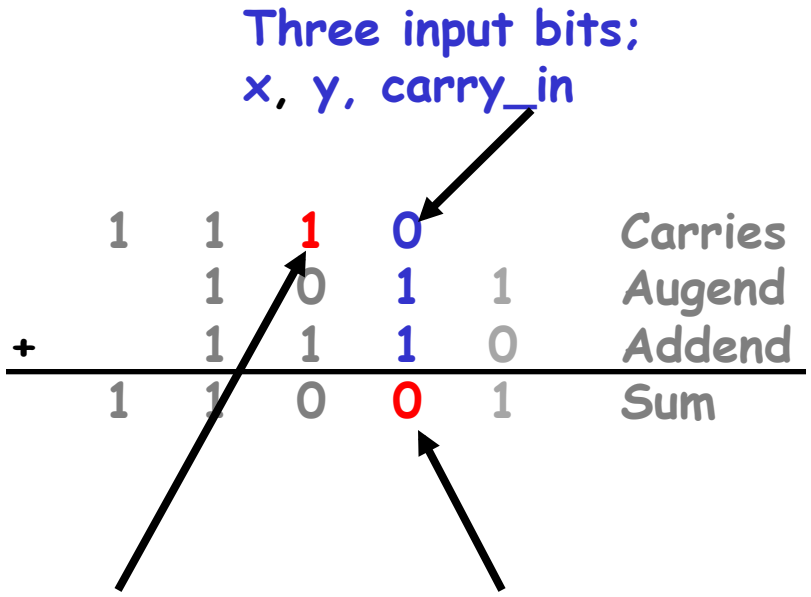
= X ⊕ Y

XOR

- ***carry out is worth twice as much as the sum bit***

Second Bit Position

- (and every subsequent position)



Second Bit Position's Truth Table

- **Adding 3 bits together to get a two bit number**

0	+ 0	+ 0	= 00		
0	+ 0	+ 1	= 01		
0	+ 1	+ 0	= 01		
0	+ 1	+ 1	= 10		
1	+ 0	+ 0	= 01		
1	+ 0	+ 1	= 10		
1	+ 1	+ 0	= 10		
1	+ 1	+ 1	= 11		

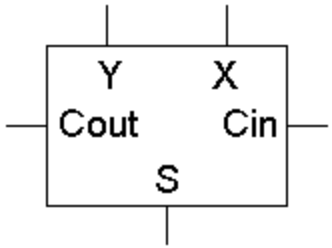
X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder

- **Adds three input bits to produce a sum and carry out.**

$S = X \oplus Y \oplus C_{in}$

$C_{out} = XY + (X \oplus Y)C_{in}$



X	Y	C_{in}	C_{out}	S
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

Full Adder Circuit

- A Full Adder can be built with two half adders

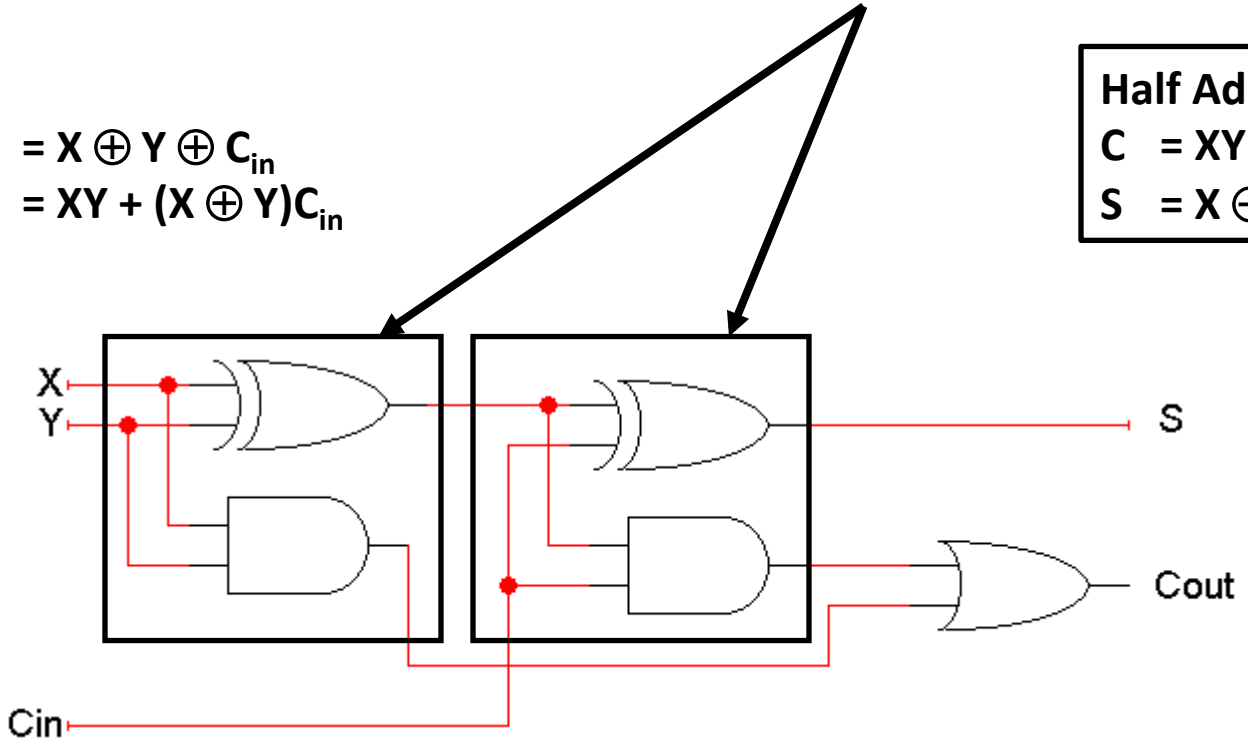
$S = X \oplus Y \oplus C_{in}$

$C_{out} = XY + (X \oplus Y)C_{in}$

Half Adder Equations

$C = XY$

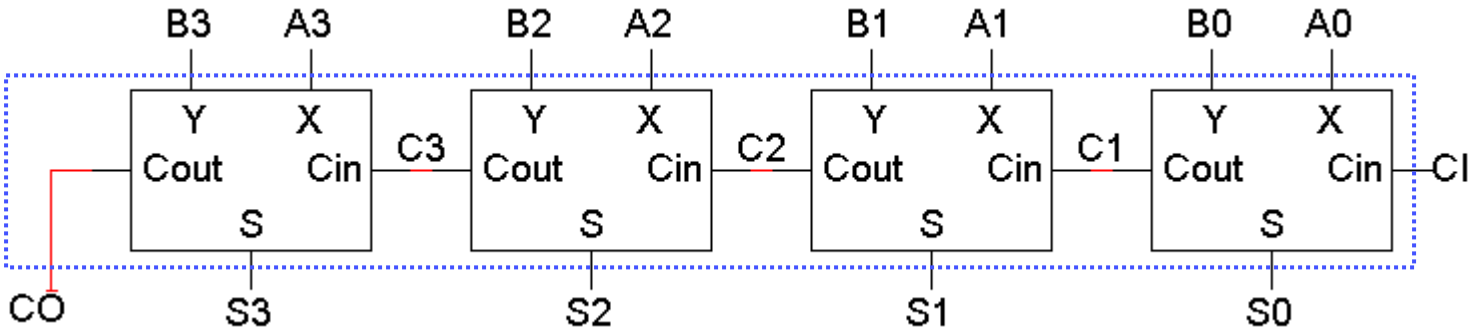
$S = X \oplus Y$



Building multi-bit adders

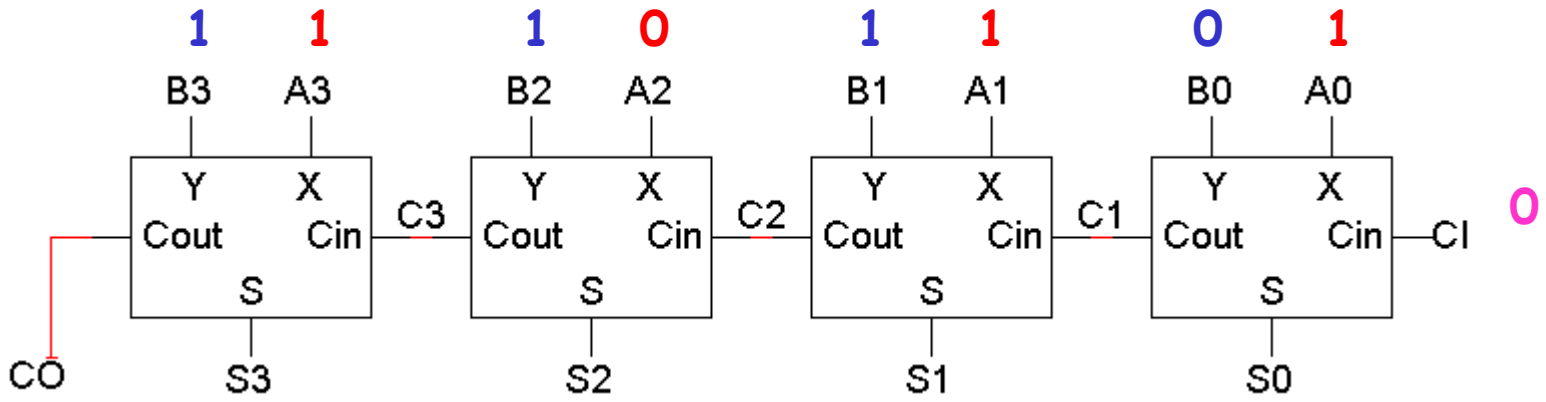
- **Recall our discussion about hierarchical design**
  - *(The stop lights to prevent train collisions…)*
- **We're going to build multi-bit adders by chaining full adders**
- **Example: 4-bit adder**

CO	C_3	C_2	C_1		*Carries*
	A_3	A_2	A_1	A_0	*Augend*
+	B_3	B_2	B_1	B_0	*Addend*
	S_3	S_2	S_1	S_0	*Sum*



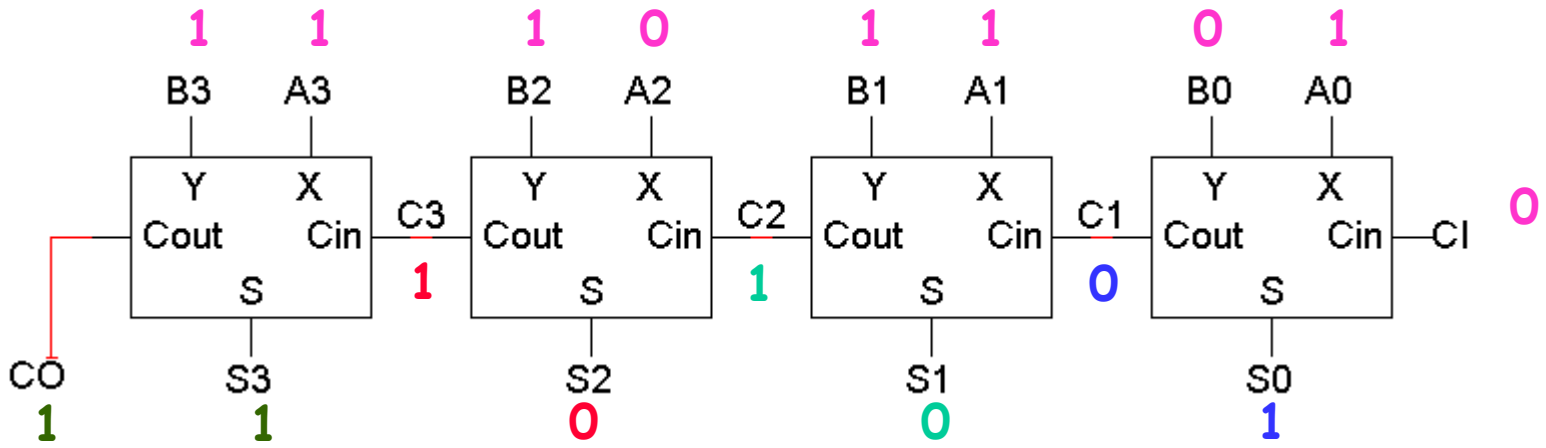
An example of 4-bit addition

- Let's try our initial example: A=1011 (eleven), B=1110 (fourteen).



An example of 4-bit addition

- Let's try our initial example: A=1011 (eleven), B=1110 (fourteen).



1. Fill in all the inputs, including CI=0
2. The circuit produces C1 and S0 (1 + 0 + 0 = 01)
3. Use C1 to find C2 and S1 (1 + 1 + 0 = 10)
4. Use C2 to compute C3 and S2 (0 + 1 + 1 = 10)
5. Use C3 to compute CO and S3 (1 + 1 + 1 = 11)

Woohoo! The final answer is 11001 (twenty-five) *if we consider it a 5-bit output*.

Implementing Subtraction

- **We said last time that we implement subtraction**
  - By negating the second input and then adding

$$A - B = A + (-B)$$

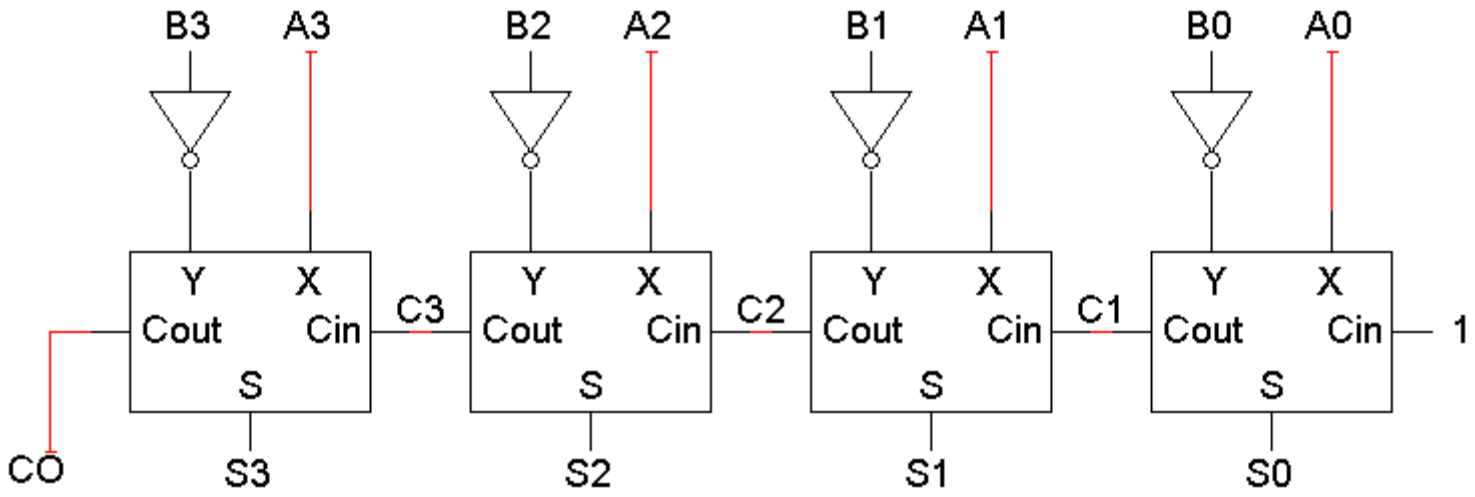
- **Negating in 2's complement is**
  - Inverting the bits and adding one

$$-B = \sim B + 1$$

- **Substituting in:**

$$A - B = A + (-B) =$$

Implementing Subtraction, cont.

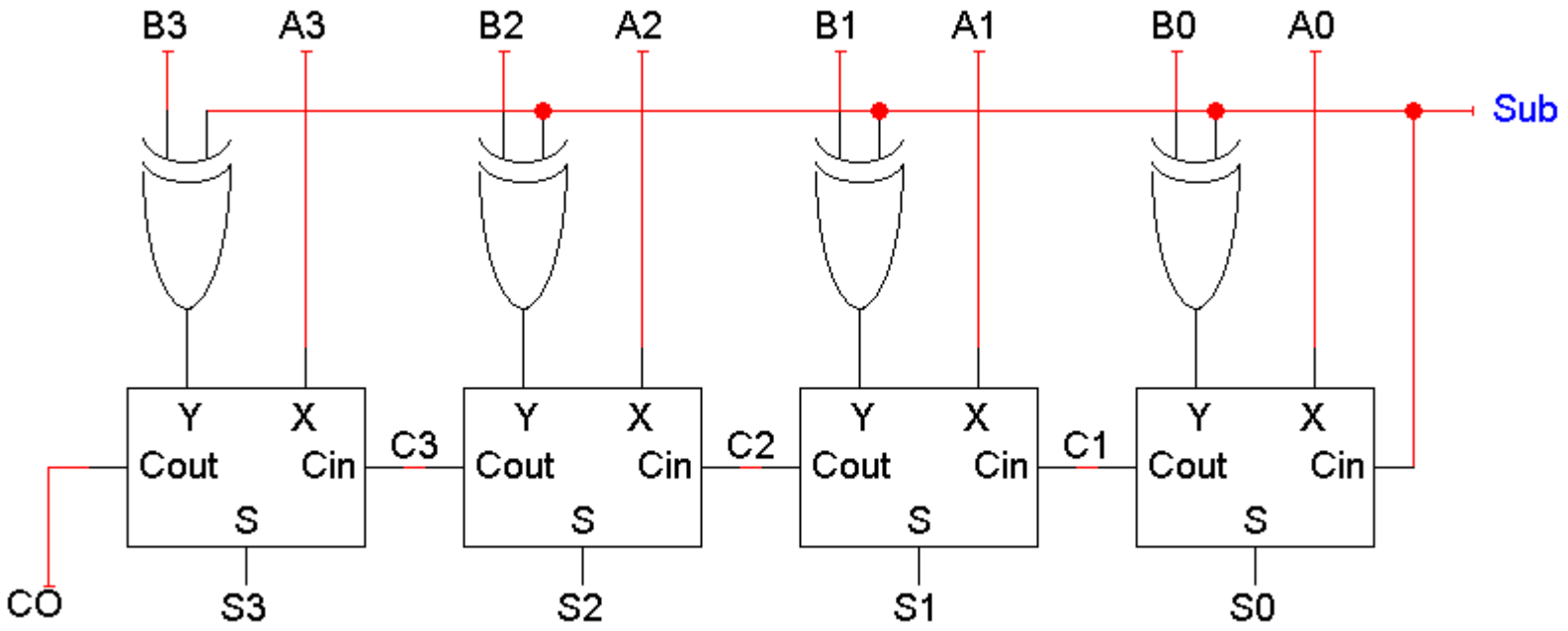


Addition + Subtraction in one circuit

- XOR gates let us selectively complement the B input.

$$X \oplus 0 = X \qquad X \oplus 1 = X'$$

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A – B.



Data vs. Control

- **We'll delineate two groups of signals in the hardware**

- **Datapath**
  - *These generally carry the numbers we're crunching*
  - *E.g., the X and Y inputs and the output S*

- **Control**
  - *These generally control how data flows and what operations are performed*
  - *E.g., the SUB signal.*

Logical Operations

- **In addition to ADD and SUBTRACT, we want our ALU to perform bit-wise AND, OR, NOT, and XOR.**
- **This should be straight forward.**
  - We have gates that perform each of these operations.

X

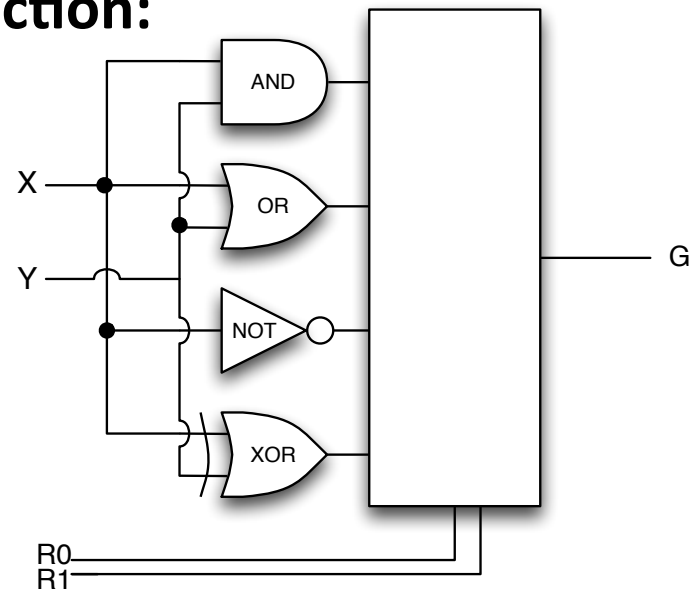
Y

Selecting the desired logical operation

- **We need a control signal to specify the desired operation:**
  - We'll call that sign R
  - 4 operations means R is 2 bits

R_1	R_0	Output	
0	0	$G_i = X_i Y_i$	AND
0	1	$G_i = X_i + Y_i$	OR
1	0	$G_i = X_i'$	NOR
1	1	$G_i = X_i \oplus Y_i$	XOR

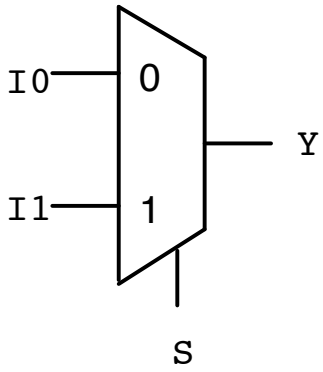
- **We need a circuit to perform the selection:**



Multiplexors

- **A multiplexor is a circuit that (logically) selects one of its inputs to connect to its output**

- **Consider a 2-to-1 multiplexor. It has:**
  - 2 data inputs (I_0, I_1)
  - a 1-bit control input (S)
  - 1 data output (Y)



S	Y
0	I0
1	I1

- **The control input selects which data input is output:**

$$Y = S'I_0 + SI_1$$

Multiplexors, cont.

- **In general, a multiplexor (mux) has:**
  - 2^N data inputs ($I_0 - I_{2^N-1}$)
  - an *N*-bit control input (S)
  - 1 data output (Y)
- **If $S = K$ then $Y = I_K$**

- **Examples:**
  - 4-to-1 mux: 4 data inputs, 2-bit control input
    - $Y = S_0'S_1'I_0 + S_0S_1'I_1 + S_0'S_1I_2 + S_0S_1I_3$
  - 16-to-1 mux: 16 data inputs, 4-bit control input

Complete 1-bit Logic Unit

R_1	R_0	Output
0	0	$G_i = X_i Y_i$
0	1	$G_i = X_i + Y_i$
1	0	$G_i = X_i'$
1	1	$G_i = X_i \oplus Y_i$