

# lecture 4

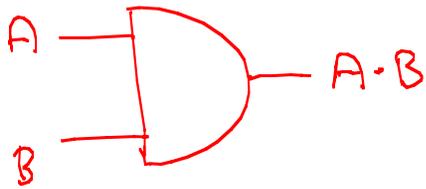
## Combinational logic 2

- ROM
- arithmetic circuits
- arithmetic logic unit (ALU)

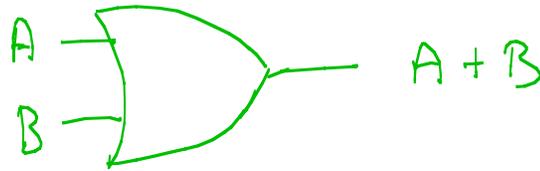
January 20, 2016

# Last lecture: truth tables, logic gates & circuits

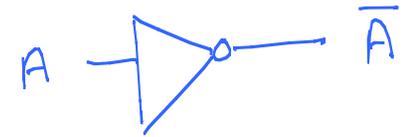
AND



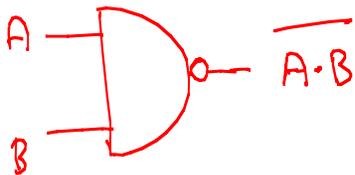
OR



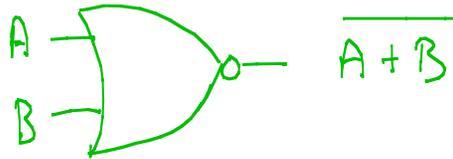
NOT



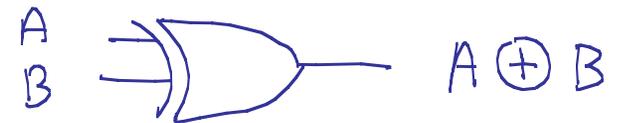
NAND



NOR

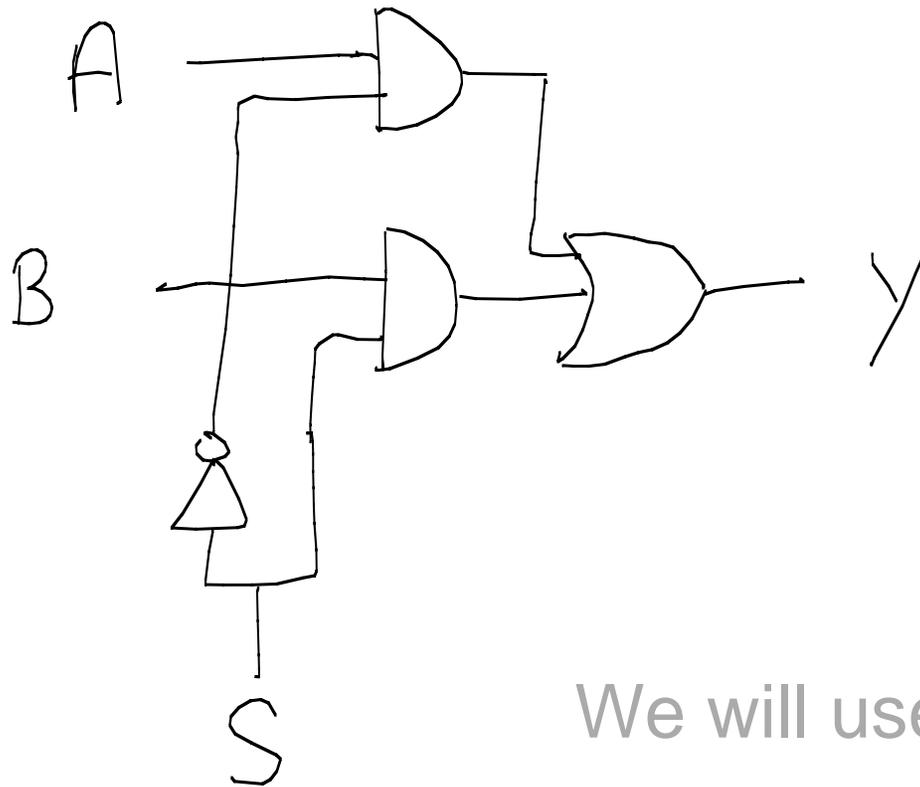


XOR



# Recall multiplexor (selector)

$$Y = \overline{S} \cdot A + S \cdot B$$



if S  
Y = B  
else  
Y = A

We will use this several times later.

# "Read-only Memory"

(leftover topic from last lecture)

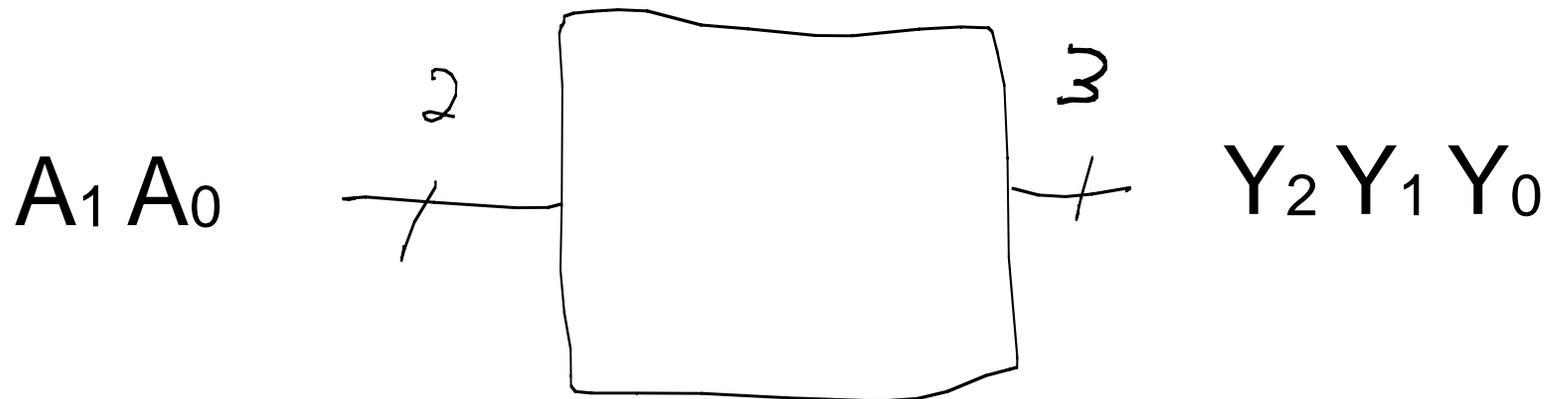
$A_1$	$A_0$	$Y_2$	$Y_1$	$Y_0$
0	0	0	1	1
0	1	0	0	1
1	0	0	0	0
1	1	1	0	0

address

data

Sometimes we can think of a circuit as a "hardwired" memory (read only).

Note: the order of the  $A_1$   $A_0$  variables matters.



# Recall: binary arithmetic

$$\begin{array}{rcccc} & C_{n-1} & \dots & C_2 & C_1 & C_0 \\ & A_{n-1} & \dots & A_2 & A_1 & A_0 \\ + & B_{n-1} & \dots & B_2 & B_1 & B_0 \\ \hline S_{n-1} & \dots & S_2 & S_1 & S_0 & \end{array}$$

## Notes:

- $C_0 = 0$
- $A, B$  could represent signed or unsigned numbers

Let's build an "adder" circuit.

$$\begin{array}{r}
 C_{n-1} \dots C_2 C_1 \\
 A_{n-1} \dots A_2 A_1 A_0 \\
 B_{n-1} \dots B_2 B_1 B_0 \\
 \hline
 S_{n-1} \dots S_2 S_1 S_0
 \end{array}$$

$A_0$	$B_0$	$S_0$	$C_1$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

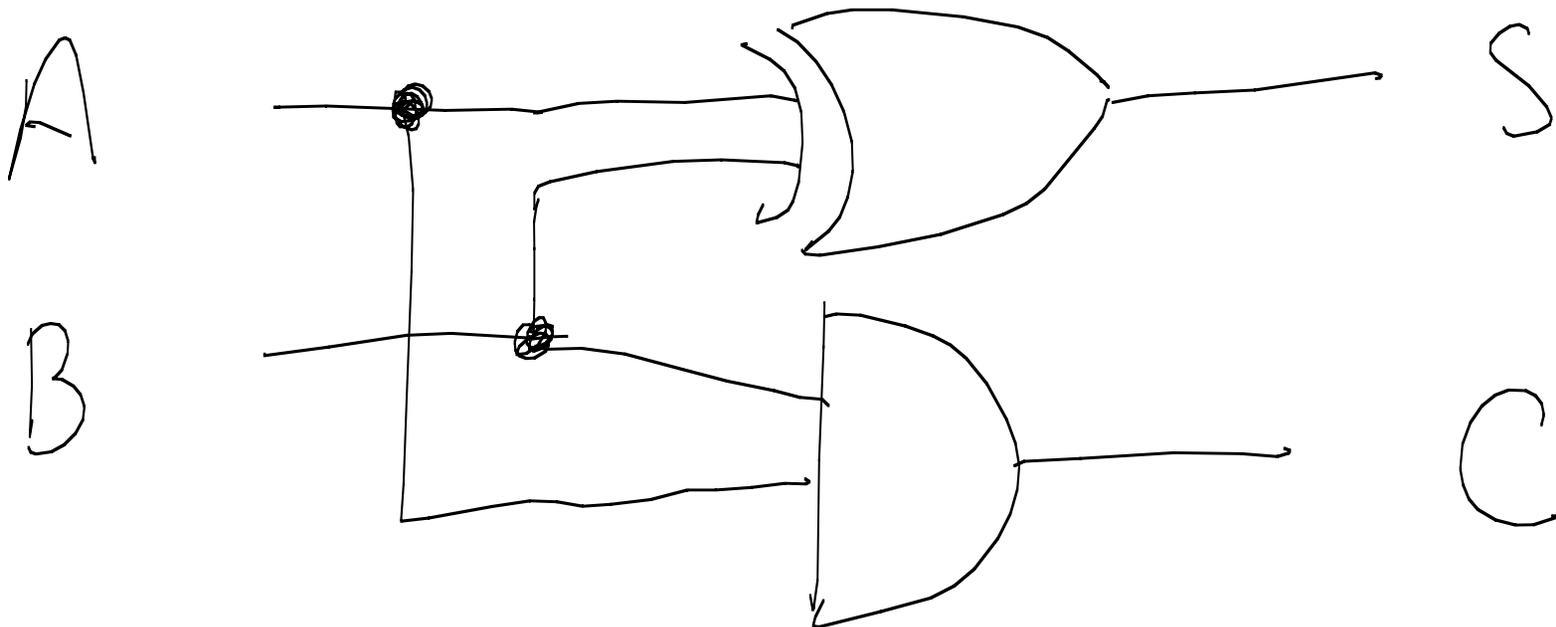
$$S_0 =$$

$$C_1 =$$

# Half Adder

$$S = A \oplus B$$

$$C = A \cdot B$$



$C_{n-1} \dots C_2 C_1 C_0$

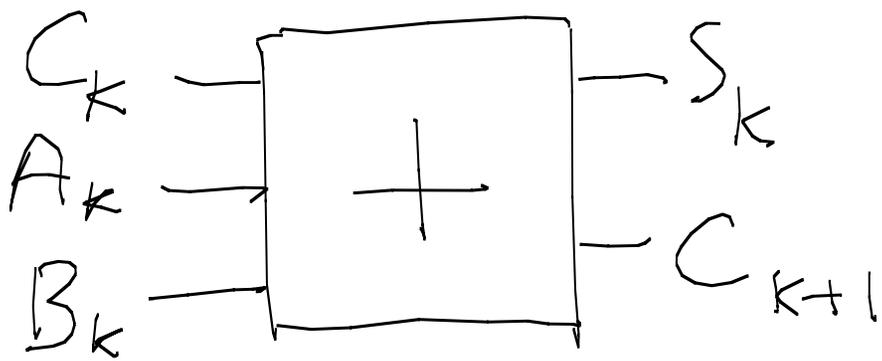
$A_{n-1} \dots A_2 A_1 A_0$

$B_{n-1} \dots B_2 B_1 B_0$

---

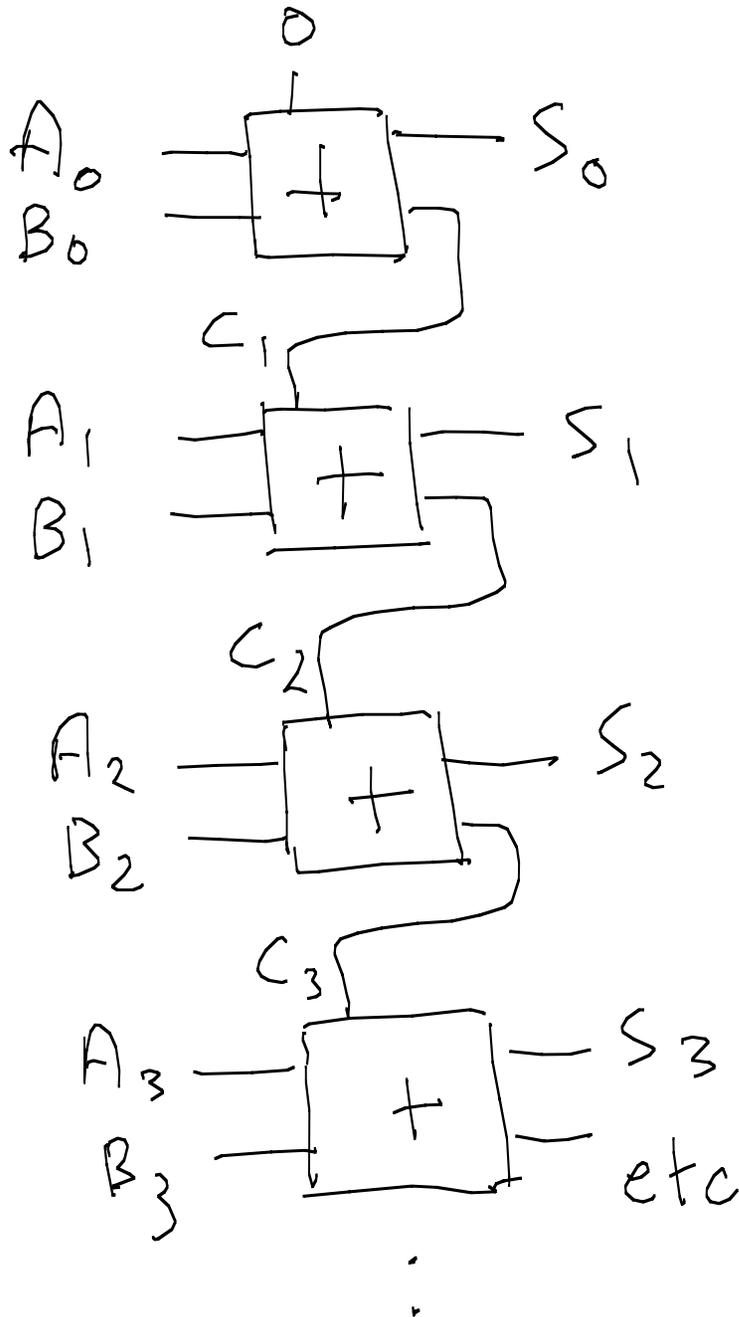
$S_{n-1} \dots S_2 S_1 S_0$

### full adder



$A_k$	$B_k$	$C_k$	$S_k$	$C_{k+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

# Ripple Adder



If  $n = 32$ , then we can have a long delay as carries propagate through the circuit. We'll return to this later.

$$\begin{array}{r}
 C_{n-1} \dots C_2 C_1 C_0 \\
 A_{n-1} \dots A_2 A_1 A_0 \\
 + B_{n-1} \dots B_2 B_1 B_0 \\
 \hline
 S_{n-1} \dots S_2 S_1 S_0
 \end{array}$$

As I mentioned before.... the *interpretation* of the S bit string depends on whether the A and B bit strings are signed or unsigned.

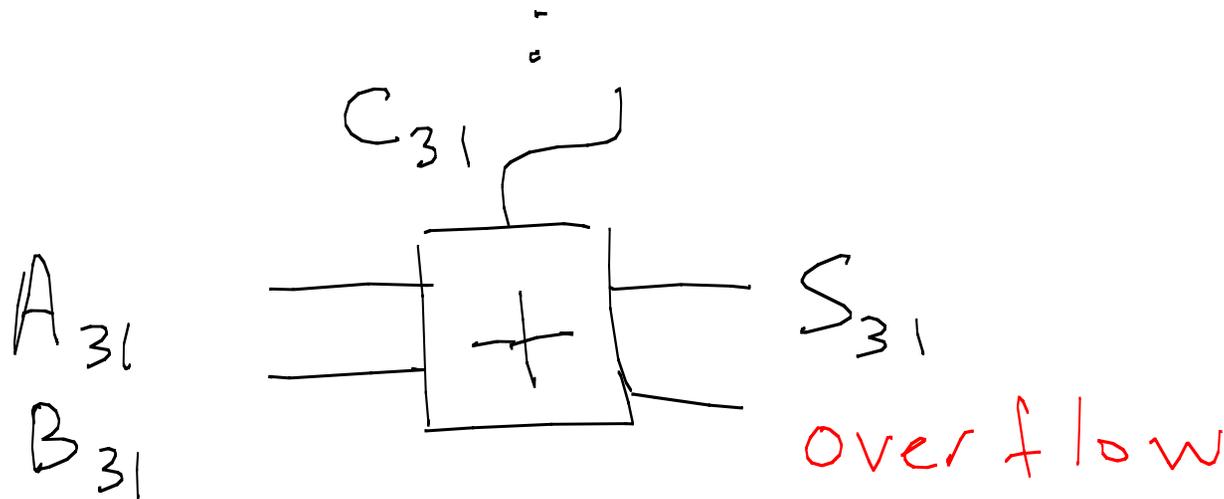
However, the full adder circuit does not depend on whether A and B are signed or unsigned.

# Overflow

We still might want to know if we have "overflowed" :

- e.g. - if the sum of two positive numbers yields a negative
- if the sum of two negative numbers yields a positive

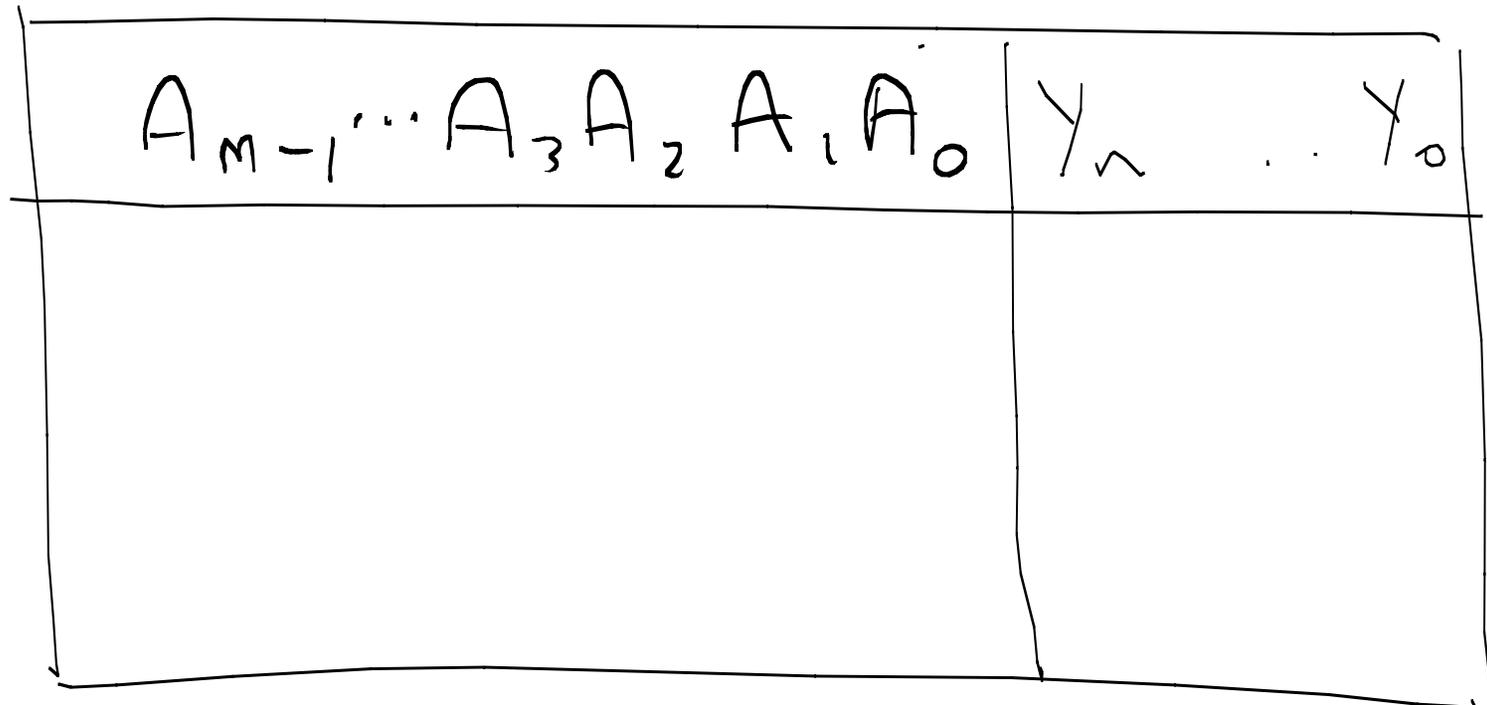
How can we detect these two cases ? (see Exercises 2)



# TODO TODAY

- encoder
- decoder
- n-bit multiplexor
- fast adder
- ALU

# Encoder

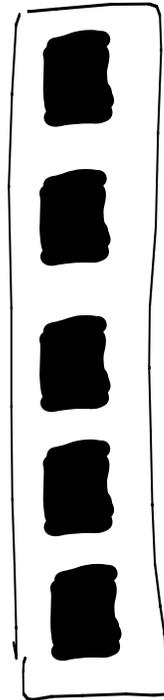


many bits

code  
(few bits)

# Encoder Example 1

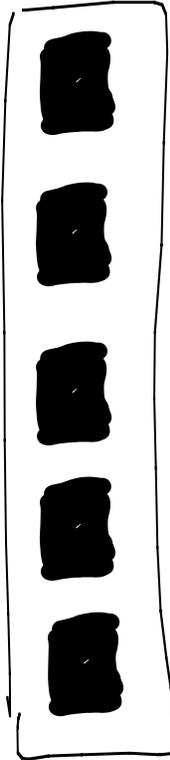
panel with  
five buttons



$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	1	0	0	0	0	1	1
1	0	0	0	0	1	0	0

This assumes only one button can be pressed at any time.

panel with  
five buttons

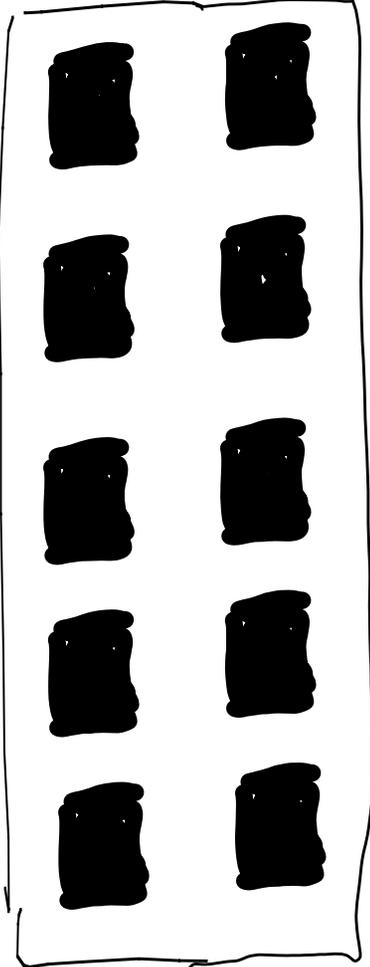


$b_4$	$b_3$	$b_2$	$b_1$	$b_0$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	0
X	X	X	X	1	0	0	1
X	X	X	1	0	0	1	0
X	X	1	0	0	0	1	1
X	1	0	0	0	1	0	0
1	0	0	0	0	1	0	1

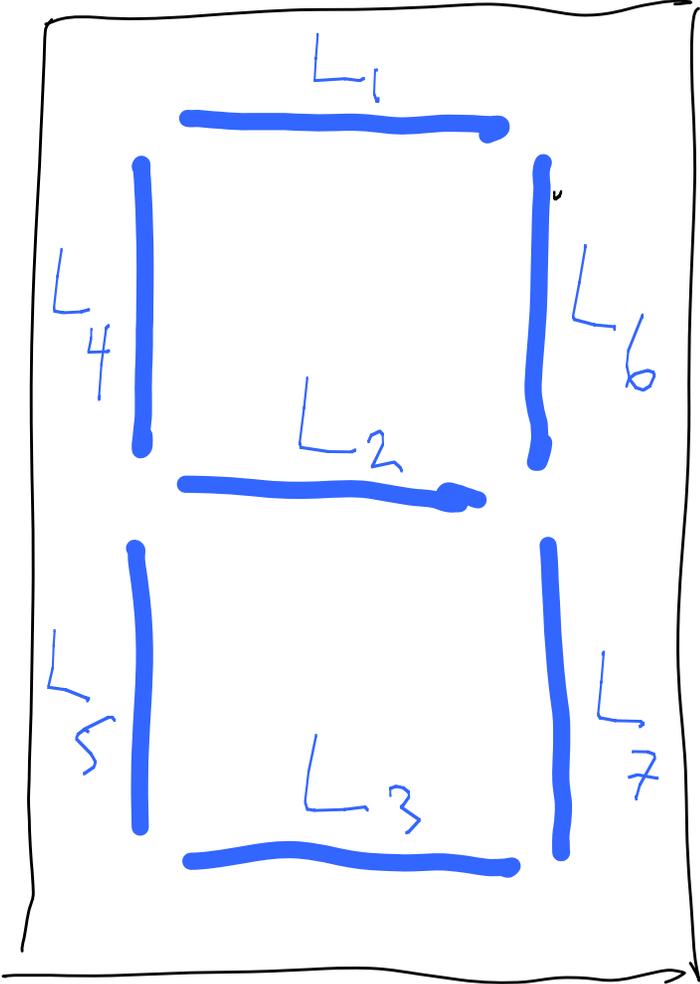
This allows two buttons to be pressed at the same time (and encodes the one with the highest index).

# Encoder Example 2

panel with ten buttons



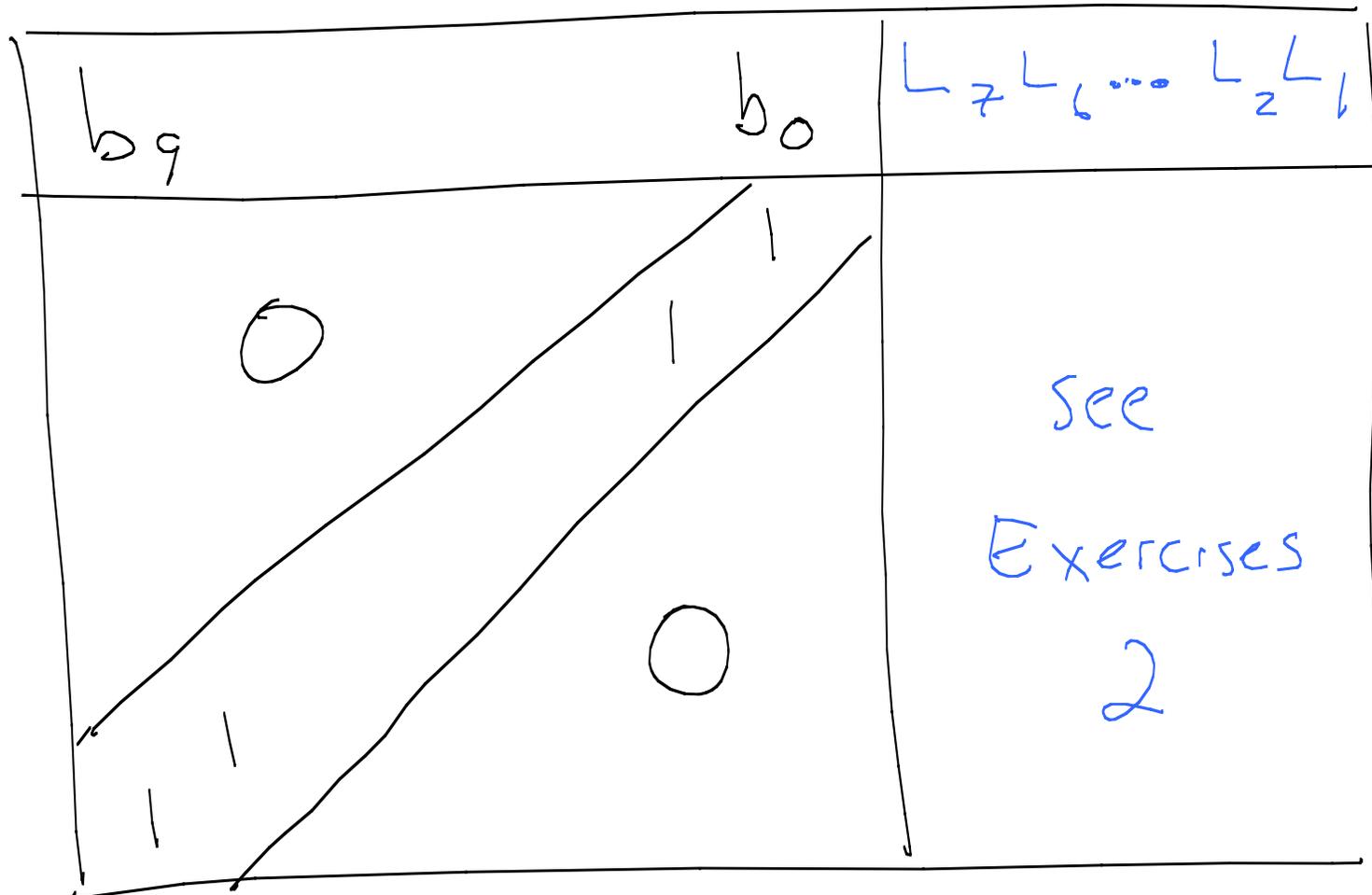
b 9 0 0 b 0



display e.g. on a digital watch, calculator, etc.

buttons

lights



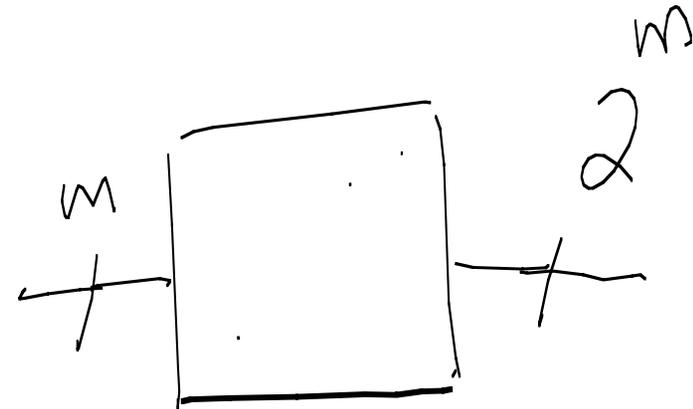
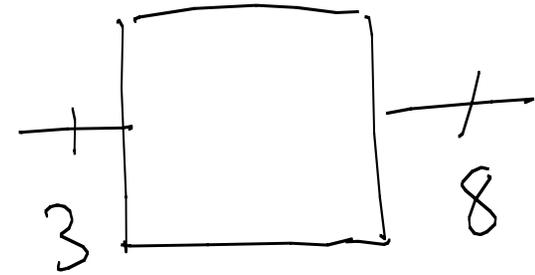
Each light  $L_i$  is turned on (1) by some set of button presses.

Each button  $b_k$  turns on (1) some set of lights.

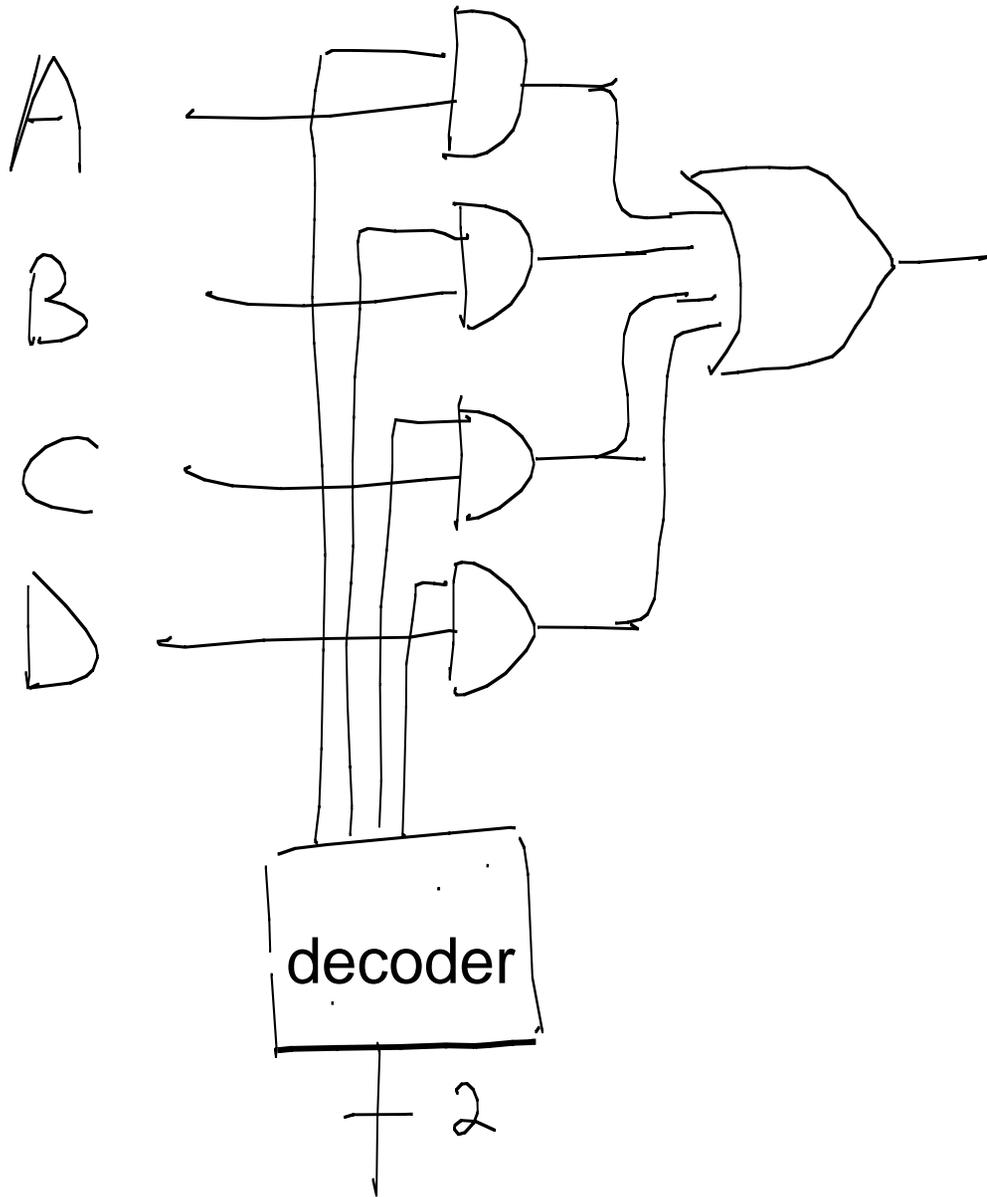
# Decoder

$b_2$	$b_1$	$b_0$	$Y_7$	...	$Y_2$	$Y_1$	$Y_0$
0	0	0	1				0
0	0	1		1			
0	1	0			1		
0	1	1				1	
1	0	0					1
1	0	1					
1	1	0					
1	1	1					

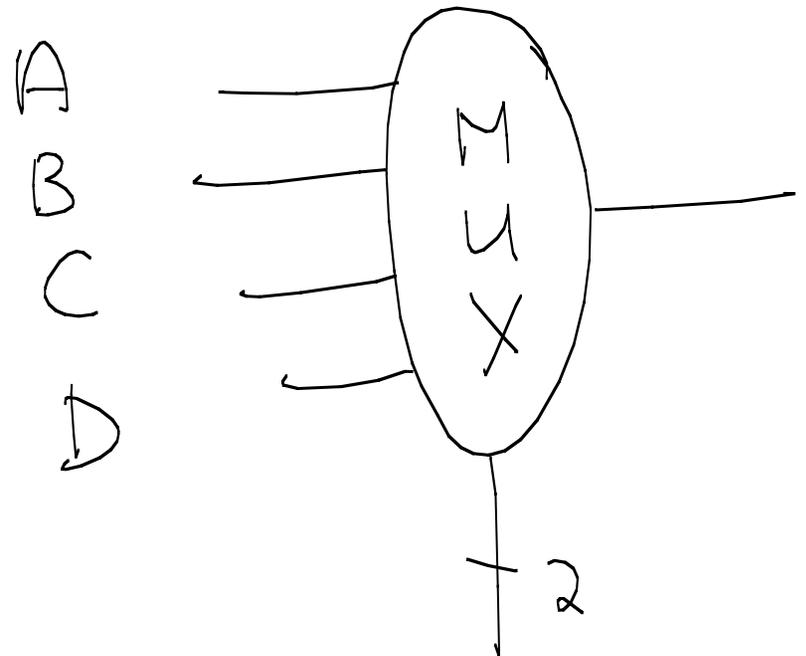
code word (in this example, it specifies which output is 1)



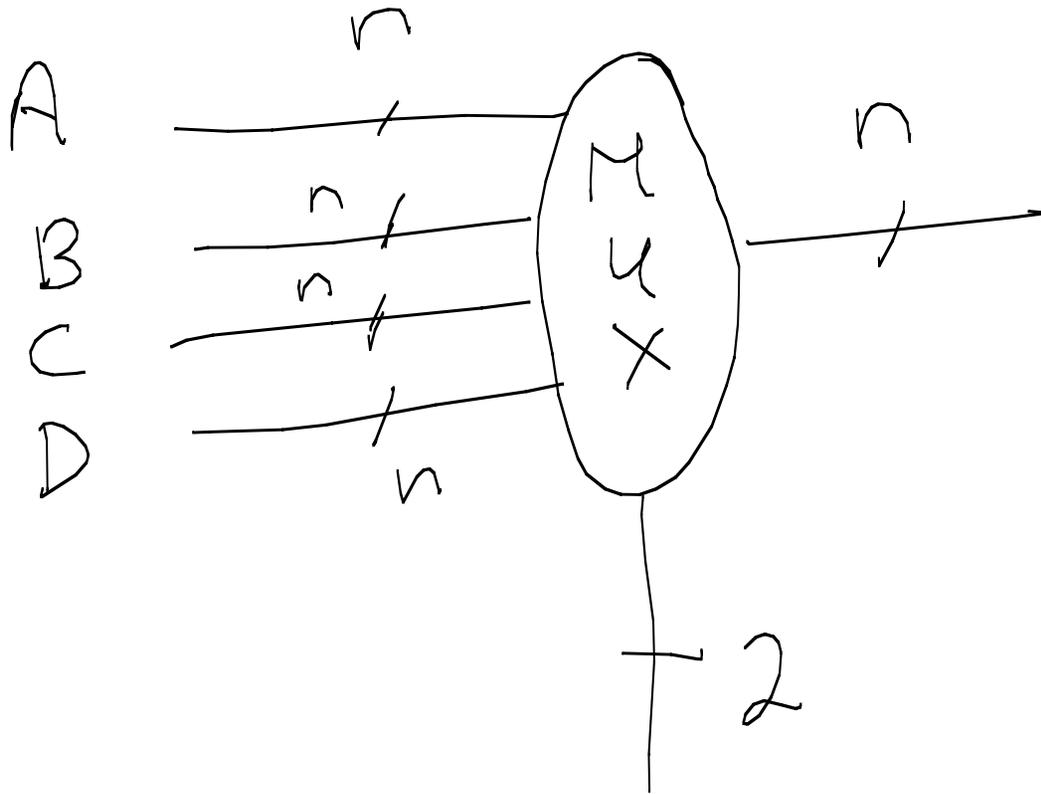
# 2-bit multiplexor



Notation:

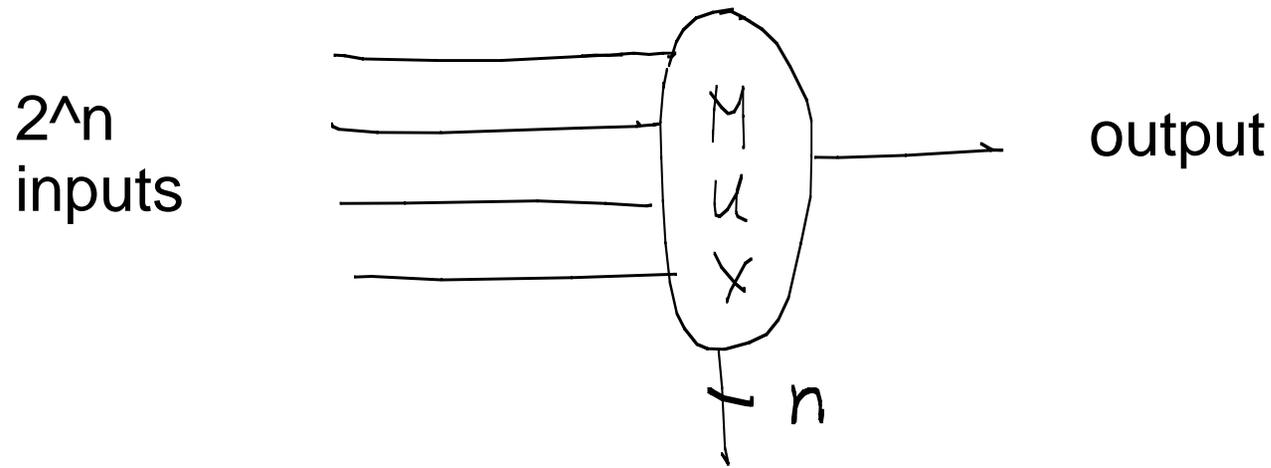


# More general example (2-bit multiplexor)



Selects from four  $n$ -bit inputs. For each  $A_i, B_i, C_i, D_i$ , we replicate the circuit on the previous slide, but use the same decoder circuit.

# n-bit multiplexor

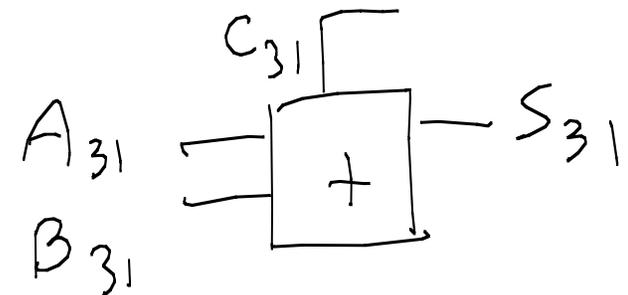
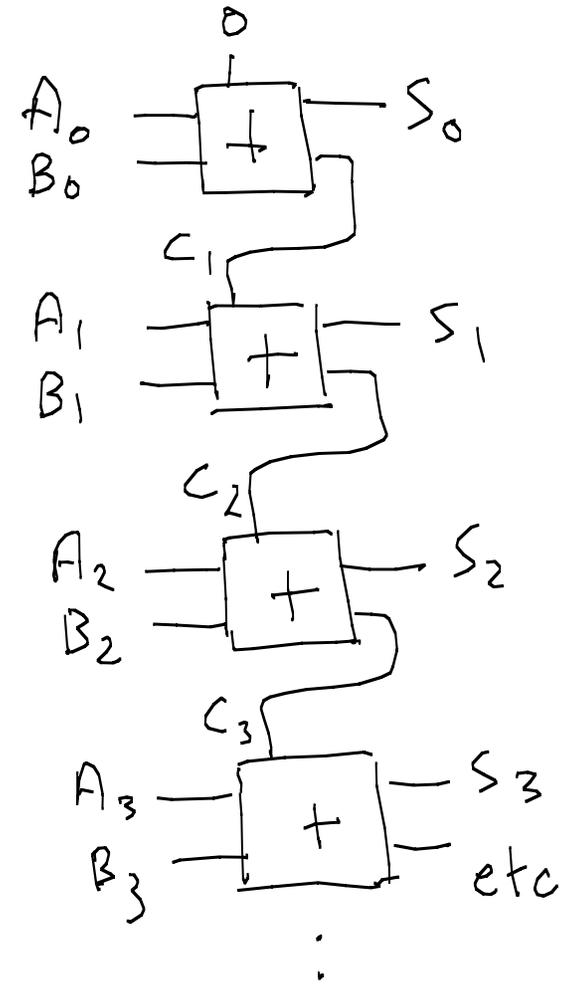


which  
input?

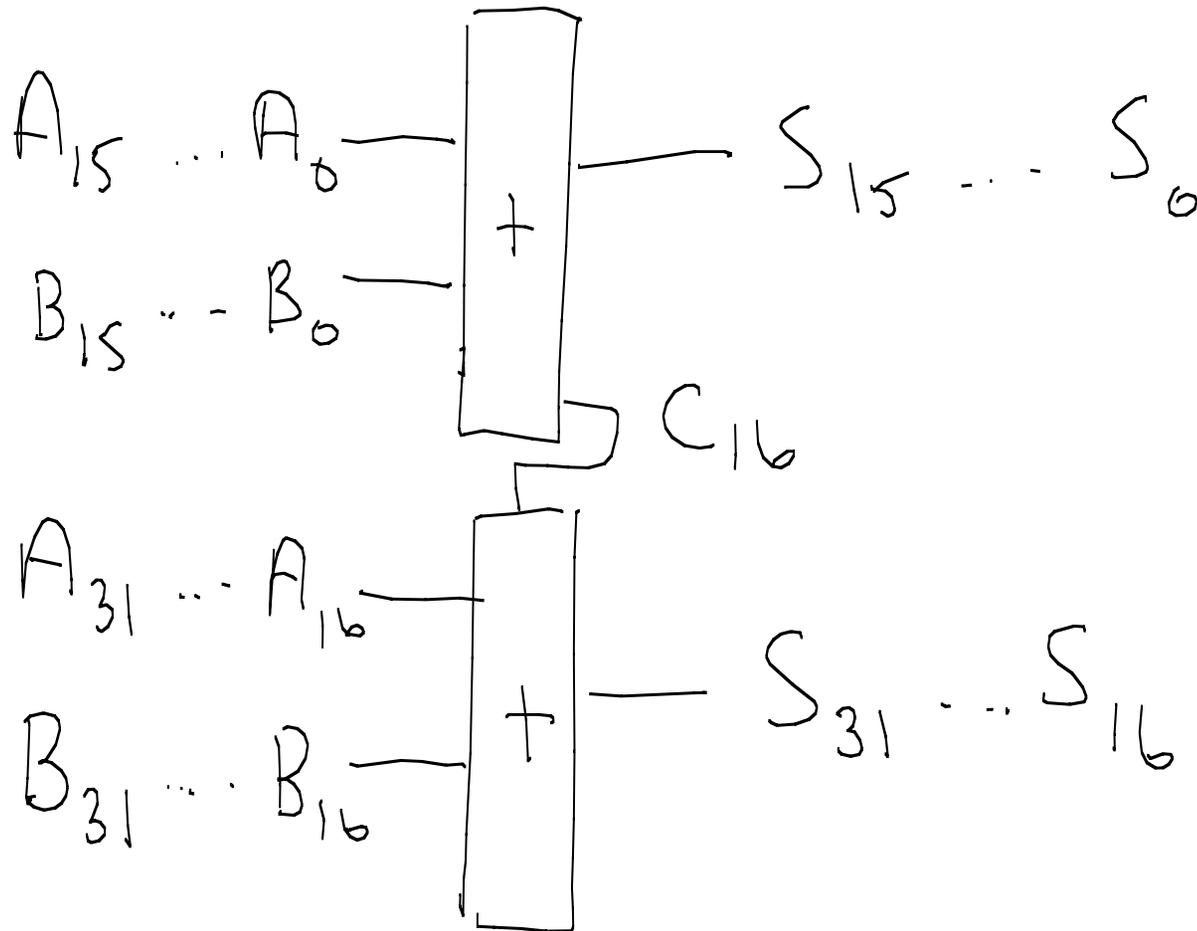
We will next look at some examples of how multiplexors are used.

Recall the ripple adder.

The main problem is that it is slow.



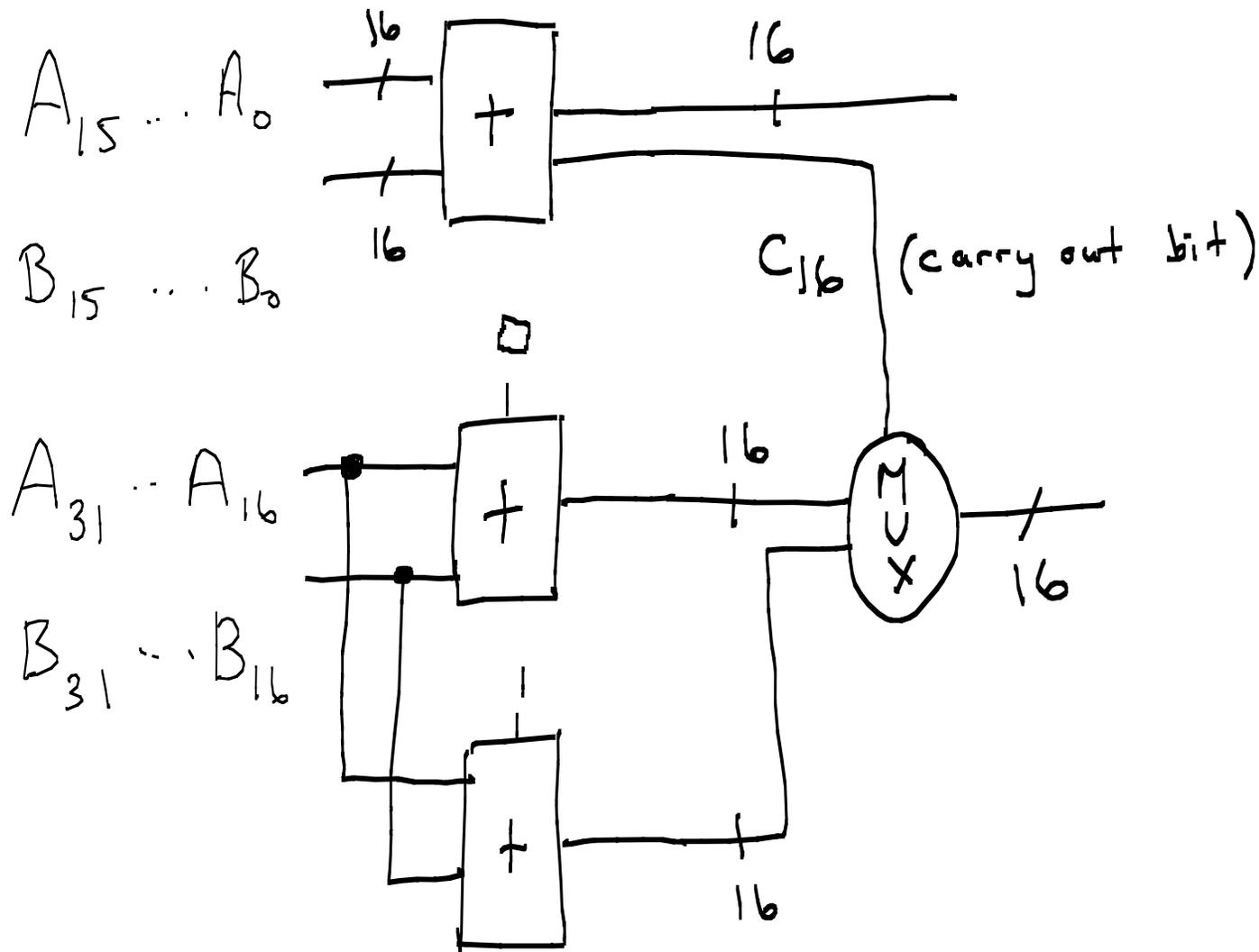
# How to speed up the adder ?



Instead of one 32 bit adder, think of two 16 bit adders.

We can compute the result of each, in half the time. (However, if  $C_{16} = 1$ , then we have to wait for it to ripple through.)

# Fast Adder



Tradeoffs: we chop the time in half (almost, why?) but it increases the number of gates by more than 50% (why?). Note we can repeat this idea (recursion).

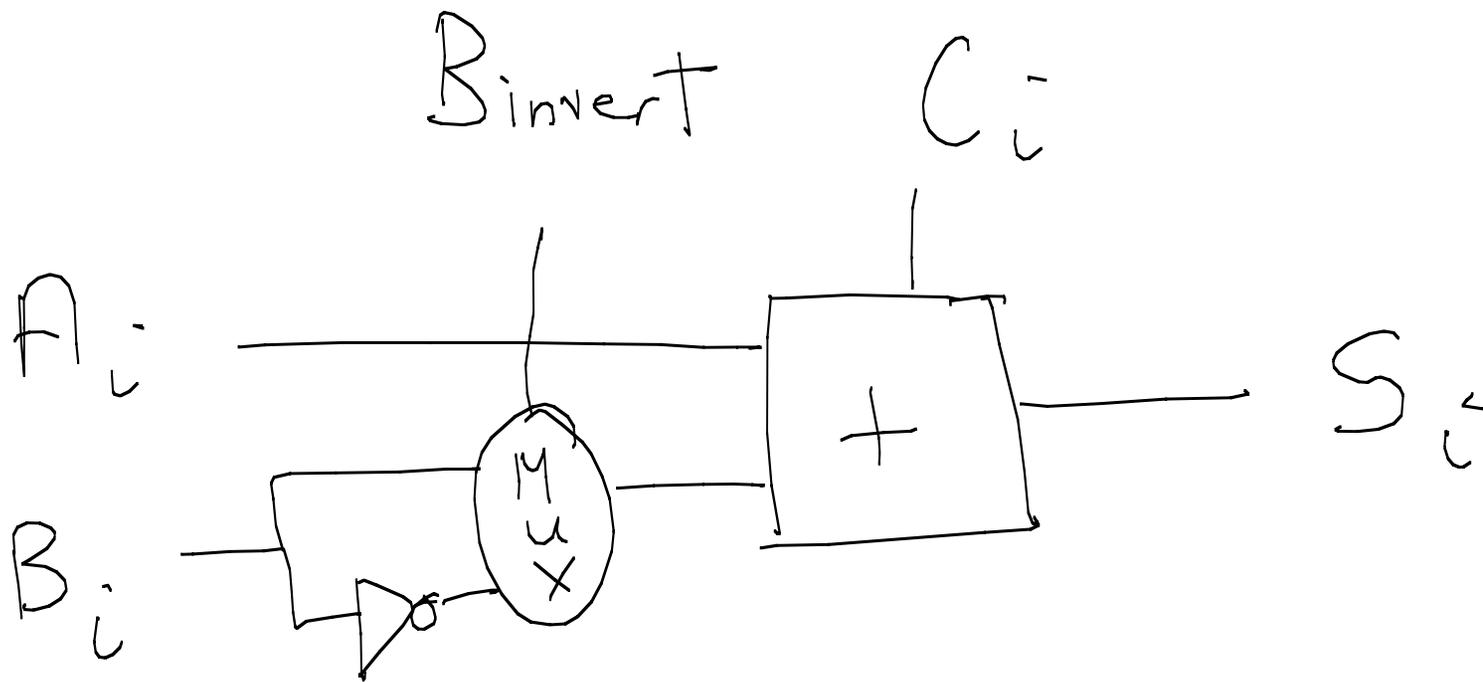
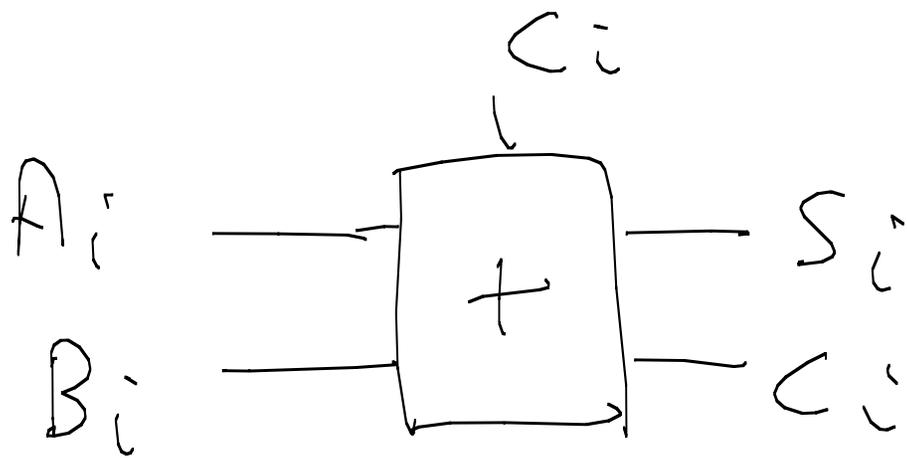
# Subtraction

$$\begin{array}{r} A_{n-1} \dots A_2 A_1 A_0 \\ - B_{n-1} \dots B_2 B_1 B_0 \\ \hline S_{n-1} \dots S_2 S_1 S_0 \end{array}$$

$$x - y = x + (-y)$$

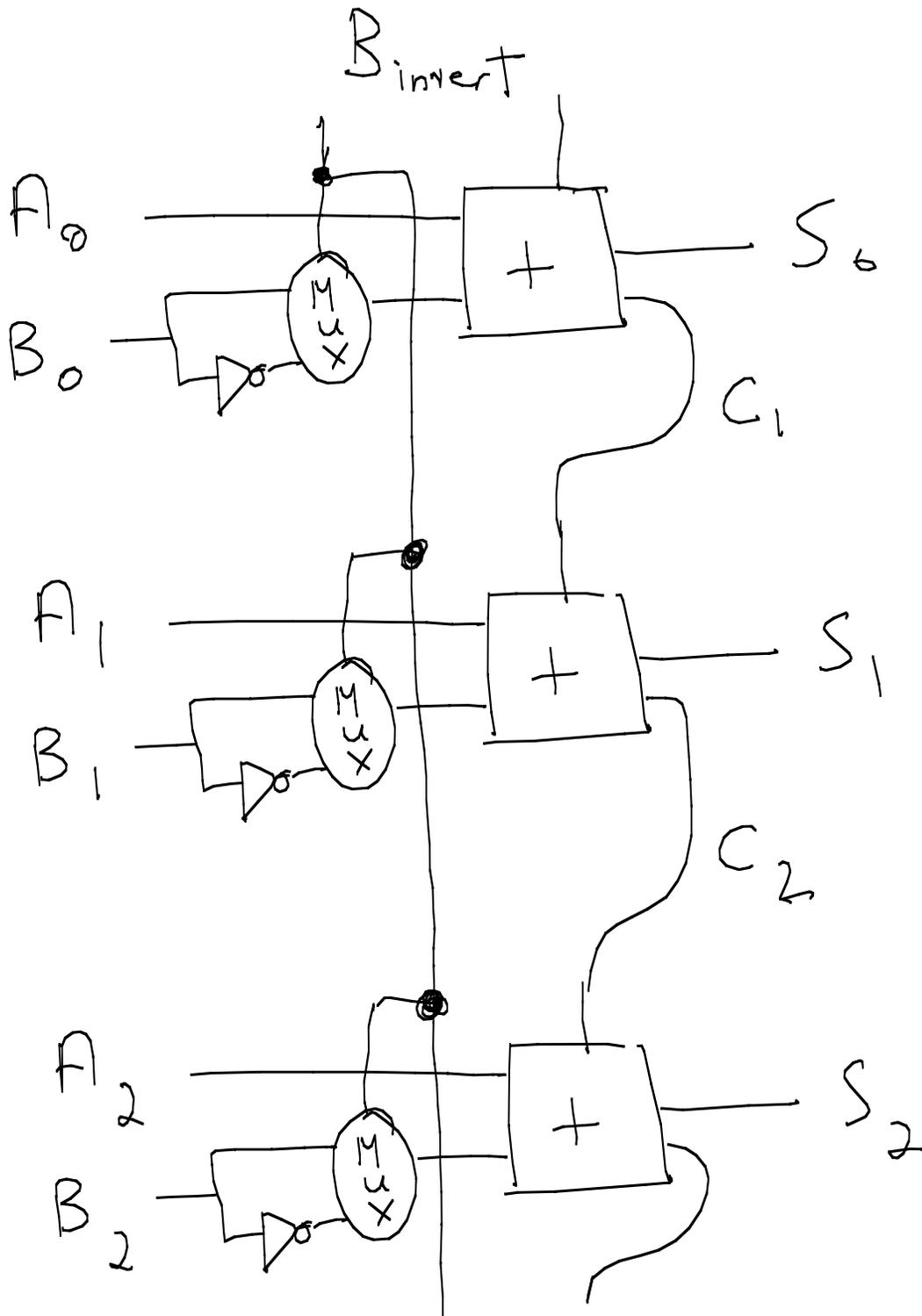
↑

Invert bits and add 1.



Invert bits and add 1.

When  $B_{\text{invert}}$  is 1, this adds 1 by setting  $C_0$  to 1.

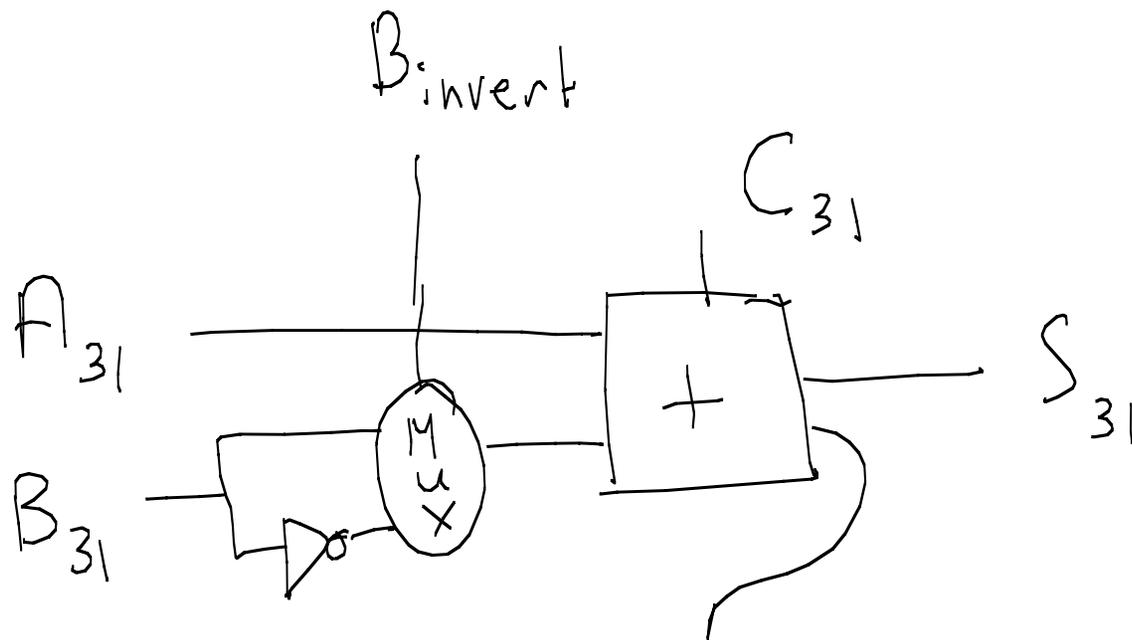


$A_{n-1} \dots A_2 A_1 A_0$

$B_{n-1} \dots B_2 B_1 B_0$

$S_{n-1} \dots S_2 S_1 S_0$

$n = 32$

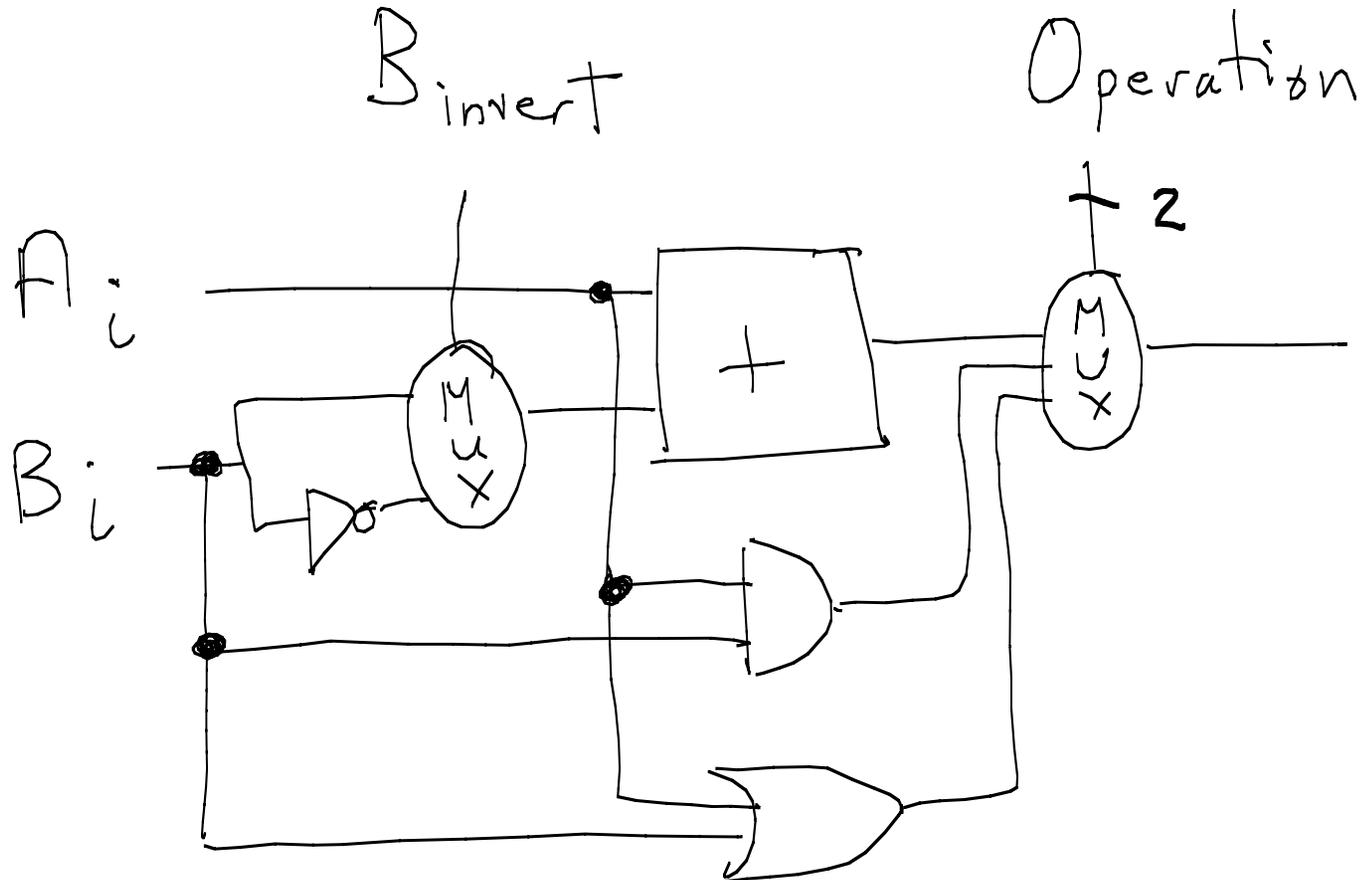


overflow

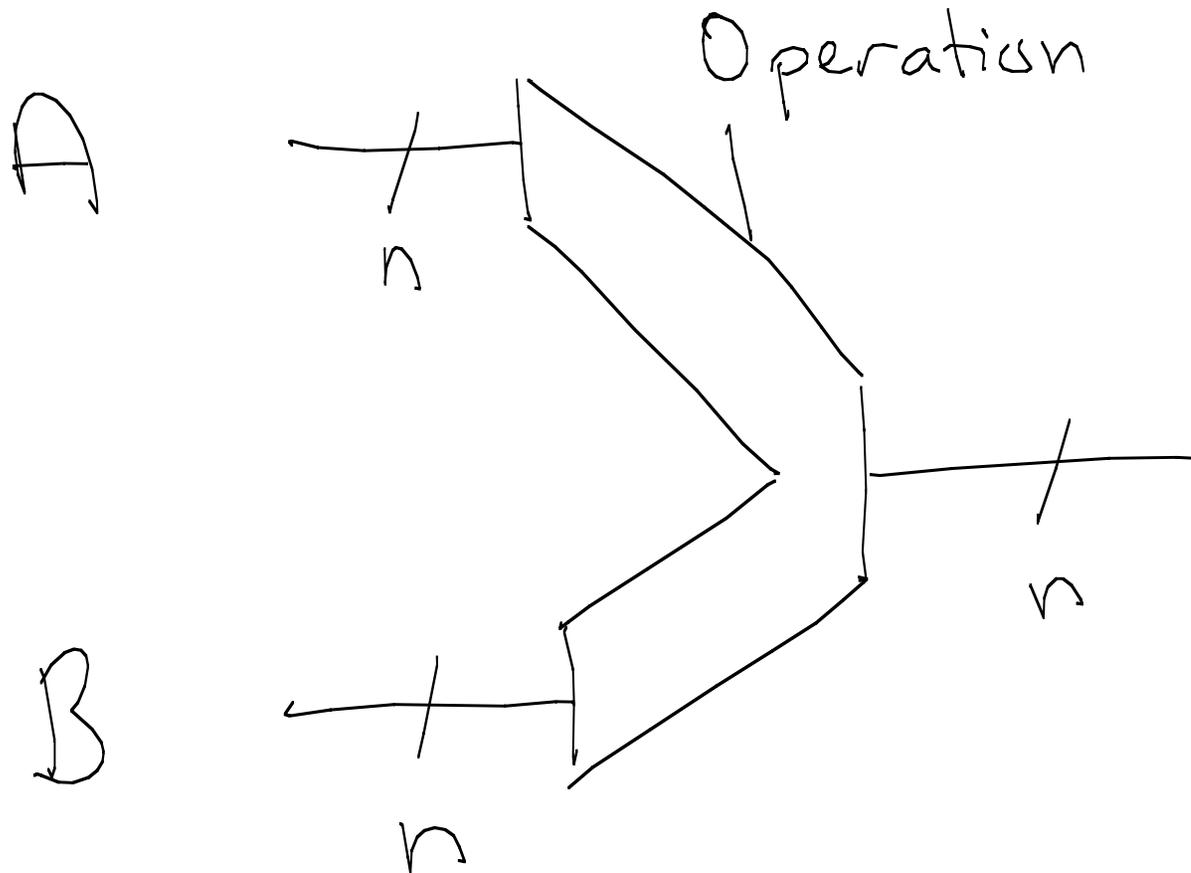
$B_{invert}$	$A_{31}$	$B_{31}$	$S_{31}$	overflow
0				
1				

See Exercises 2

Let's include a bitwise AND and OR.



# Arithmetic Logic Unit (ALU)



# Announcements

- 193 registered (172 seats in room)
- Quiz 1

The solutions, grading scheme, and grades were posted yesterday. You should know this b/c you should be subscribed to news on my courses. I will not be posting stuff on Facebook.

The TAs were instructed not to take off points for trivial errors if it was clear you understood what you were doing. If your Quiz was not graded according to this guideline, let me know by resubmitting it with a yellow sticky explaining the problem. **Please do not make such requests for marginal judgment calls: similar to professional sports ("the ruling on the field stands").** And, I reserve the right to regrade the whole exam, if I disagree with you.