

CSC258 Winter 2016

Lecture 3

Announcements

- Check your lab marks and quiz marks on MarkUs
- If you cannot login, email me your UTORID and name
- About borrowing DE2 boards outside the labs: we a bit short on the number of boards available, so we don't really have spare boards to be circulated outside the labs. So to make things simple and fair, no board borrowing outside the lab.
- If you attend the same lab multiple times, you lowest mark will be recorded.

Feedback 1

About the quiz: "Putting in questions which has to do with the current lecture is a bad move, ..., it favours people who can understand the lecture on the dot and/or are in the right mood for it, which is unfair for people who would put in the work after the lecture..."

- Valid point, so from this week on I will only quiz on content from previous weeks.

Feedback 2

“Could you post the slides before the lecture?”

- I probably won't do this because
 - The slides are typically not ready until a very short time before the lecture. I'm reluctant to post something that is partially done.
 - Some parts of the slides are supposed to “emotional moments” which would make you remember certain things better. Seeing the slides before would spoil it.
 - So I estimated that the loss is bigger than the gain if I post them beforehand.

Feedback 3

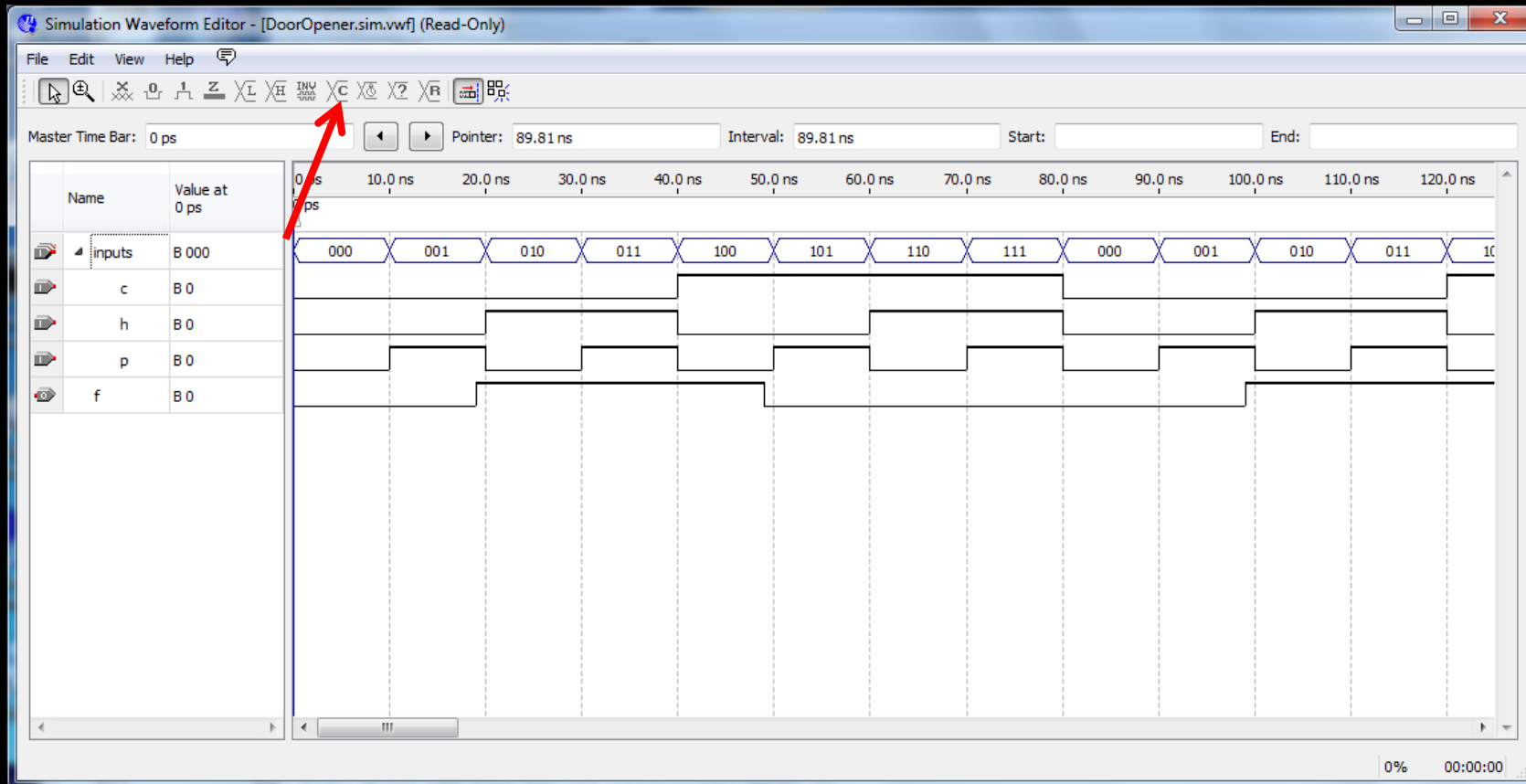
“Go slower!”

“I really like the pace of the lecture!”

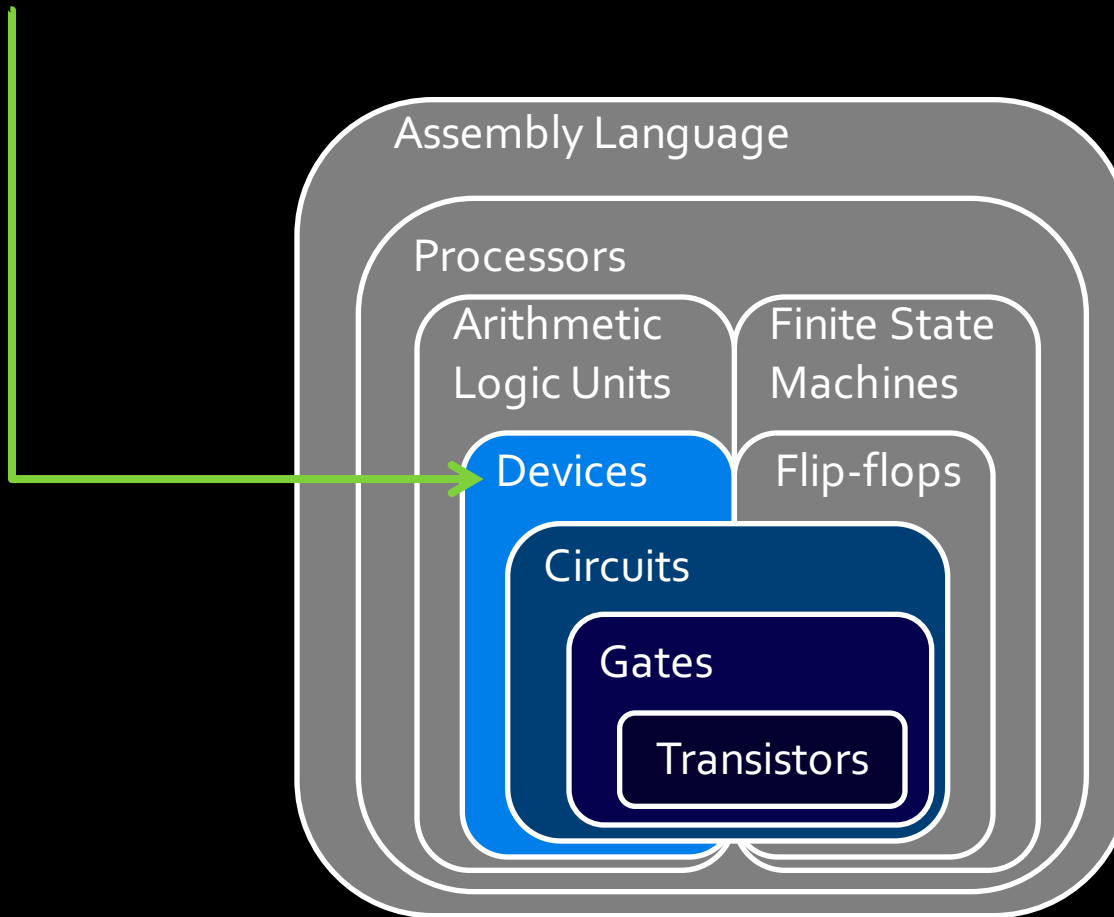
- I’ll try to keep it slow given that I can still teach all the important concepts.
- I need to hear more from you to correctly understand whether it is too fast or too slow.

A Quartus Trick

In the waveform editor, use the “**counter**” feature to generate all possible input combinations, i.e., for grouped inputs ABC, generate 0, 1, 2, 3, ..., 7, which is basically 000, 001, 010, 010, 011, ..., 111, all possible combinations.



We are here

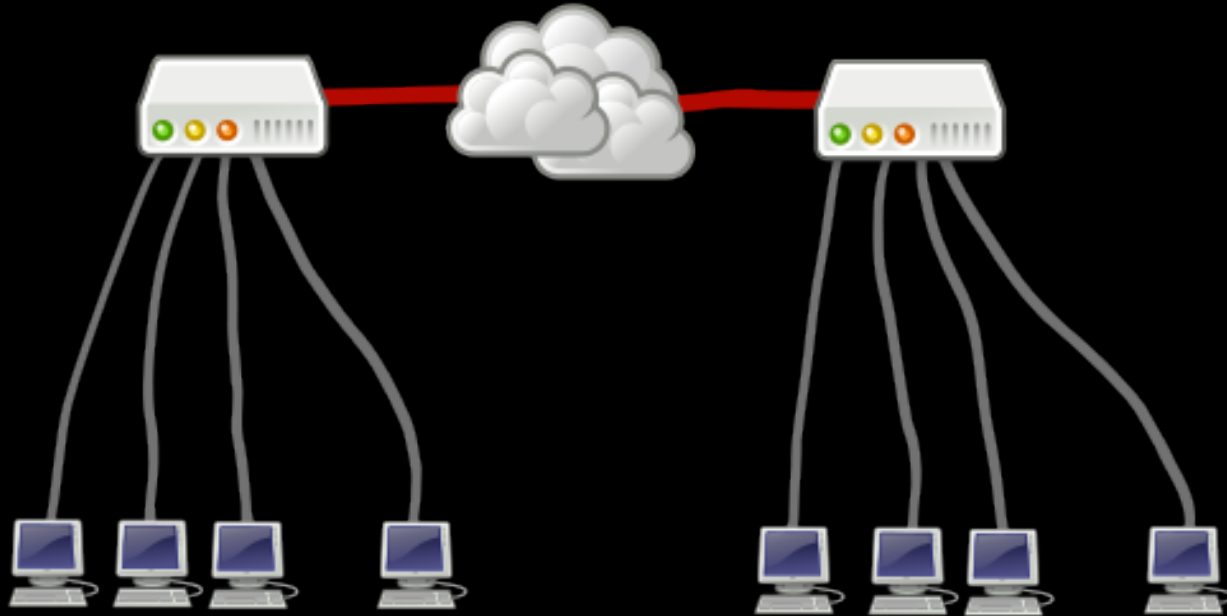


Logical Devices

Building up from gates...

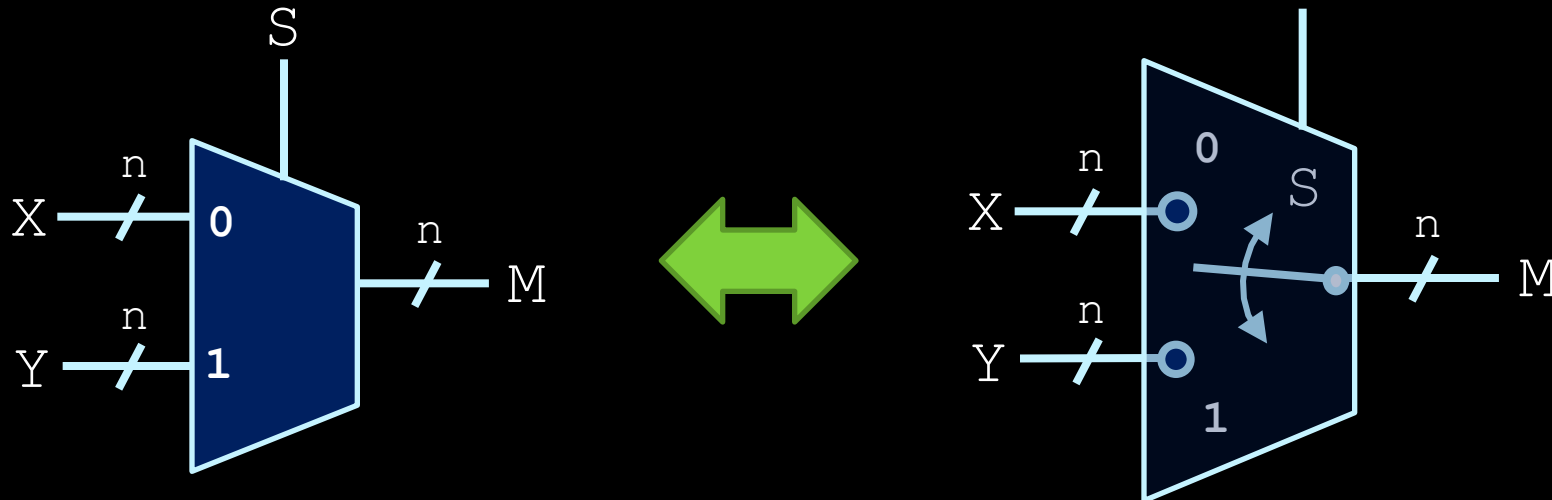
- Some common and more complex structures:
 - Multiplexers (MUX)
 - Adders (half and full)
 - Subtractors
 - Decoders
 - Seven-segment decoders
 - Comparators

Multiplexers



Logical devices

- Certain structures are common to many circuits, and have block elements of their own.
 - e.g. Multiplexers (short form: **mux**)
 - Behaviour: Output is X if S is 0, and Y if S is 1, i.e., S selects which input can go through

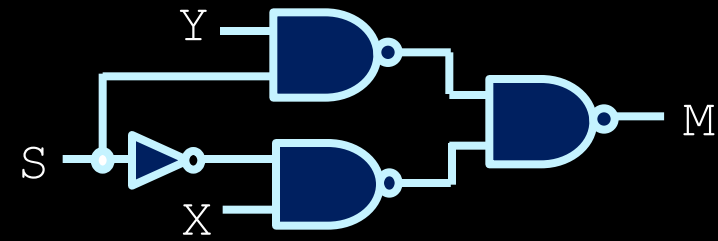
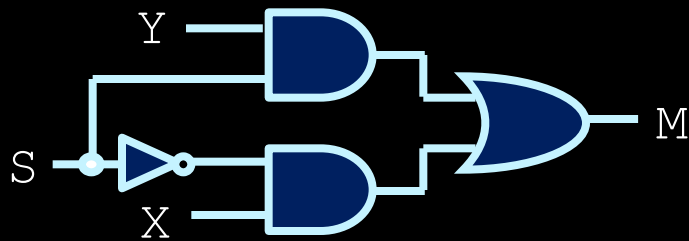


Multiplexer design

X	Y	S	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

	$\overline{Y} \cdot \overline{S}$	$\overline{Y} \cdot S$	$Y \cdot S$	$Y \cdot \overline{S}$
\overline{X}	0	0	1	0
X	1	0	1	1

$$M = Y \cdot S + X \cdot \overline{S}$$



Multiplexer uses

- Muxes are very useful whenever you need to select from multiple input values.
 - Example:
 - Surveillance video monitors,
 - Digital cable boxes,
 - routers.



Adder circuits



Adders

- Also known as binary adders.
 - Small circuit devices that add two **1-bit** number.
 - Combined together to create **iterative combinational circuits** – add **multiple-bit** numbers
- Types of adders:
 - Half adders (HA)
 - Full adders (FA)
 - Ripple Carry Adder
 - Carry-Look-Ahead Adder (CLA)

Review of Binary Math

Review of Binary Math

- Each digit of a decimal number represents a power of 10:

$$258 = 2 \times 10^2 + 5 \times 10^1 + 8 \times 10^0$$

- Each digit of a binary number represents a power of 2:

$$\begin{aligned} 01101_2 &= 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 13_{10} \end{aligned}$$

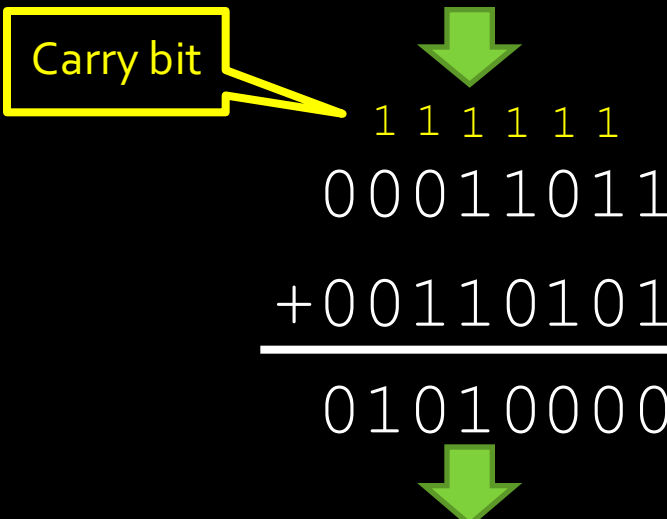
Unsigned binary addition

▪ $27 + 53$

$27 = 00011011$

$53 = 00110101$

Carry bit



1 1 1 1 1 1
00011011
+00110101

01010000

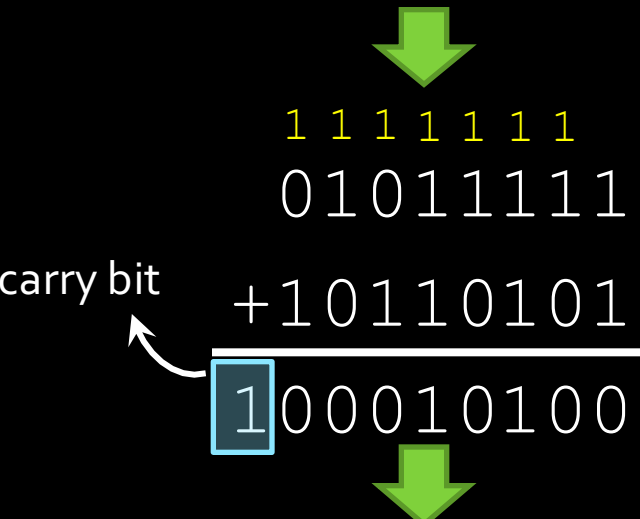
01010000

▪ $95 + 181$

01011111

+10110101

carry bit



1 1 1 1 1 1 1
01011111
+10110101

1 00010100

00010100

Half Adder

Input: two 1-bit numbers

Output: 1-bit sum and 1-bit carry

Half Adders

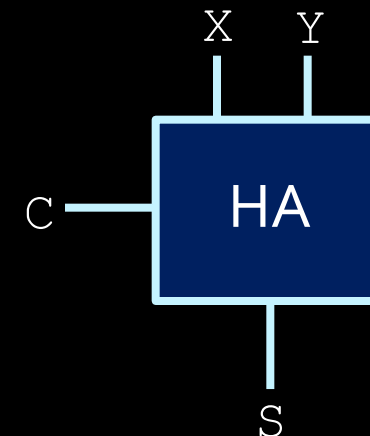
- A 2-input, 1-bit width binary adder that performs the following computations:

X	0	0	1	1
+Y	+0	+1	+0	+1
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
CS	00	01	01	10

$$C = X \cdot Y$$

$$S = X \oplus Y$$

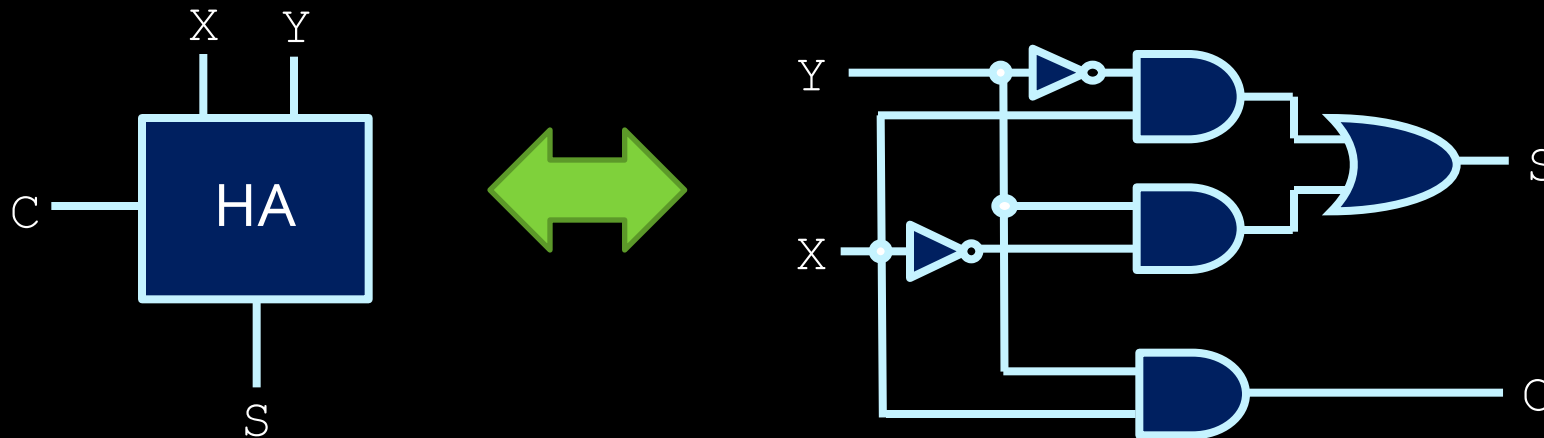
- A half adder adds two bits to produce a two-bit sum.
- The sum is expressed as a sum bit S and a carry bit C.



Half Adder Implementation

- Equations and circuits for half adder units are easy to define (even without Karnaugh maps)

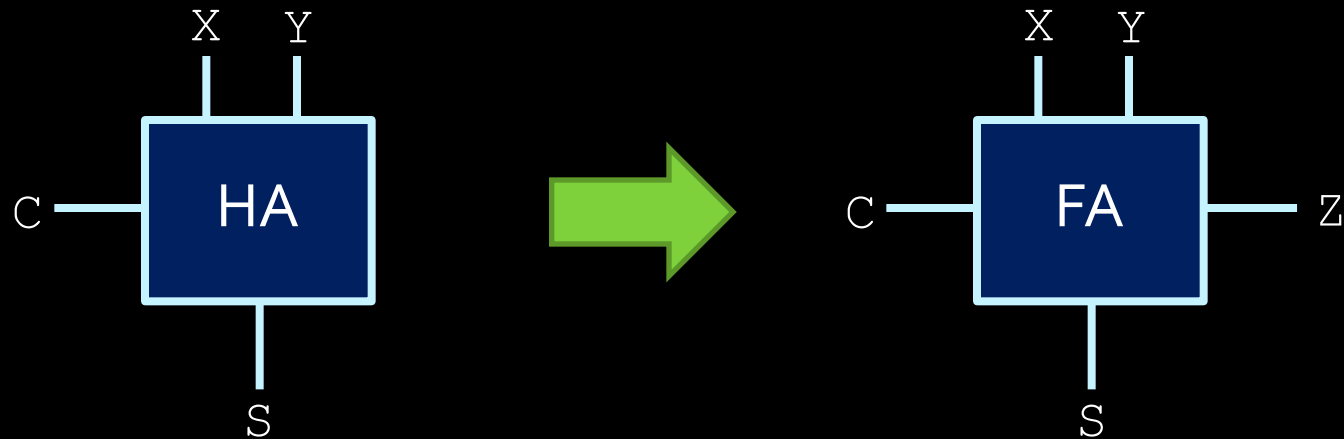
$$C = X \cdot Y \quad S = X \cdot \bar{Y} + \bar{X} \cdot Y \\ = X \oplus Y$$



A half adder **outputs** a carry-bit,
but does not take a carry-bit as **input**.

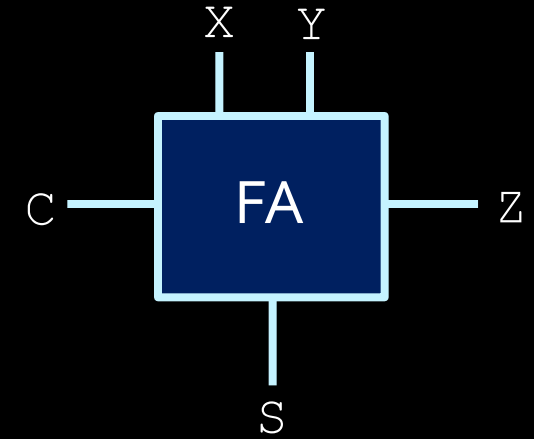
Full Adder

takes a carry bit as **input**



Full Adders

- Similar to half-adders, but with another input Z , which represents a **carry-in bit**.
 - C and Z are sometimes labeled as C_{out} and C_{in} .
- When Z is 0, the unit behaves exactly like...
 - a half adder.
- When Z is 1:



X	0	0	1	1
+Y	+0	+1	+0	+1
+Z	+1	+1	+1	+1
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
CS	01	10	10	11

Full Adder Design

X	Y	Z	C	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

C	$\overline{Y} \cdot \overline{Z}$	$\overline{Y} \cdot Z$	$Y \cdot Z$	$Y \cdot \overline{Z}$
\overline{X}	0	0	1	0
X	0	1	1	1

S	$\overline{Y} \cdot \overline{Z}$	$\overline{Y} \cdot Z$	$Y \cdot Z$	$Y \cdot \overline{Z}$
\overline{X}	0	1	0	1
X	1	0	1	0

$$C = X \cdot Y + X \cdot Z + Y \cdot Z$$

$$S = X \oplus Y \oplus Z$$

$$C = X \cdot Y + (X \oplus Y) \cdot Z$$

For gate reuse ($X \oplus Y$)
considering both C and S

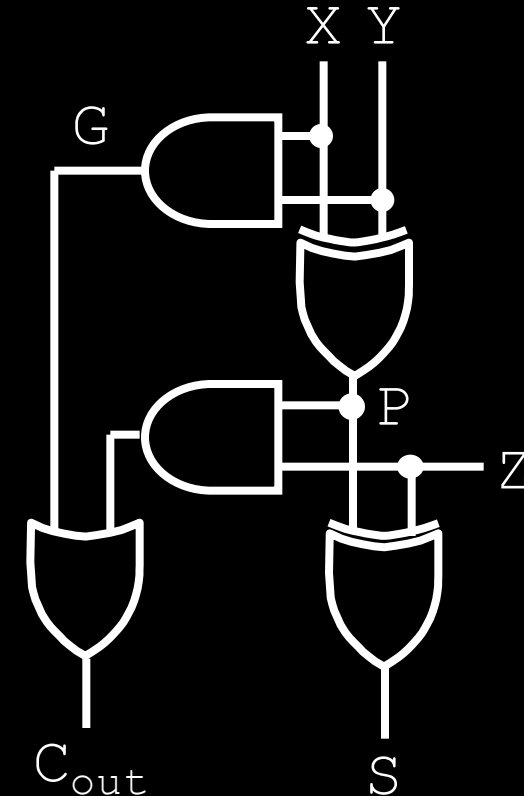
Full Adder Design

$$S = X \oplus Y \oplus Z$$

- The C term can also be rewritten as:

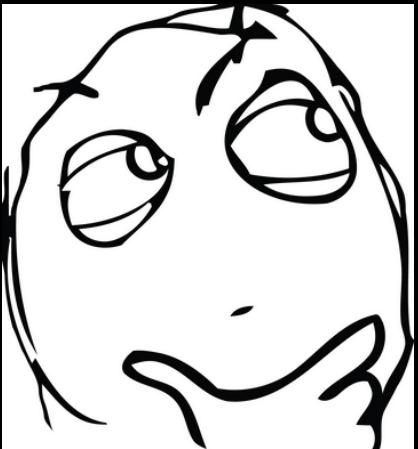
$$C = X \cdot Y + (X \oplus Y) \cdot Z$$

- Two terms come from this:
 - ▣ $X \cdot Y = \text{carry generate (G)}$.
 - Whether X and Y generate a carry bit
 - ▣ $X \oplus Y = \text{carry propagate (P)}$.
 - Whether Z will be propagated to Cout
- Results in this circuit →

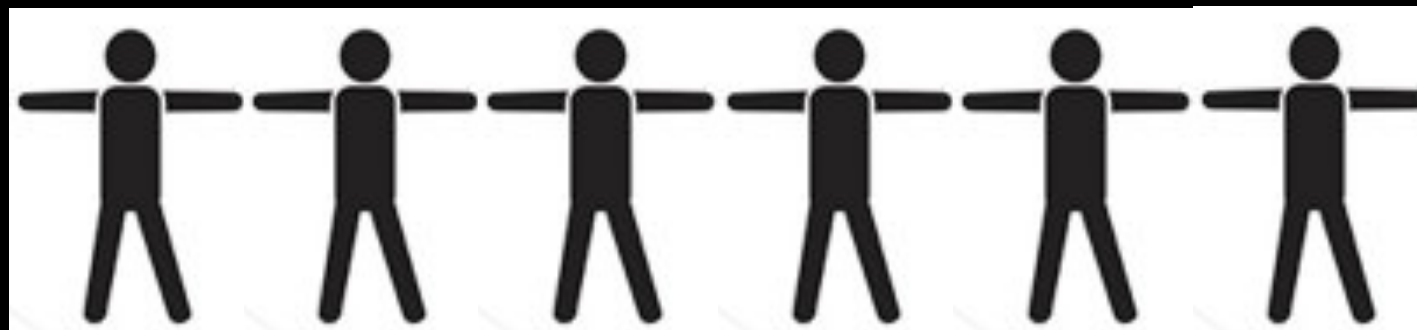
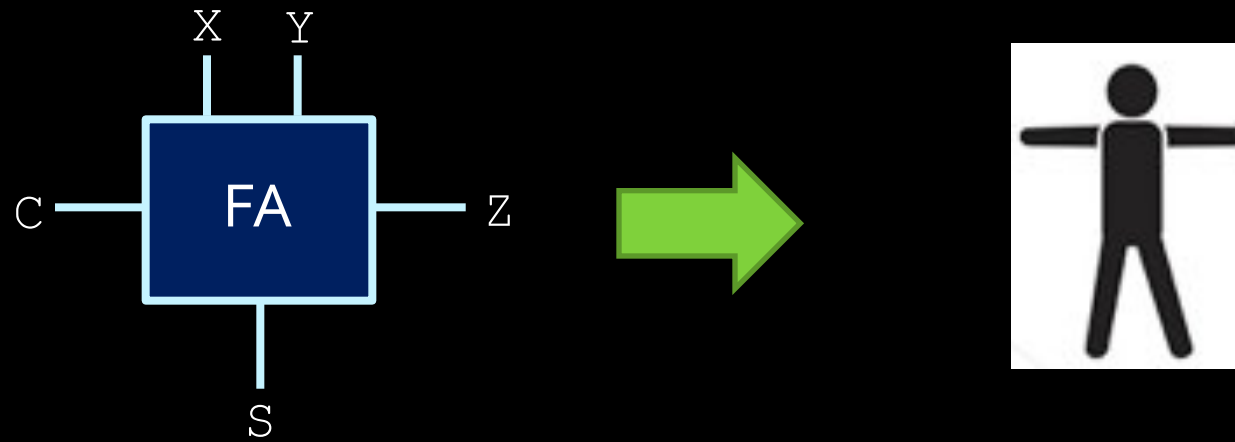
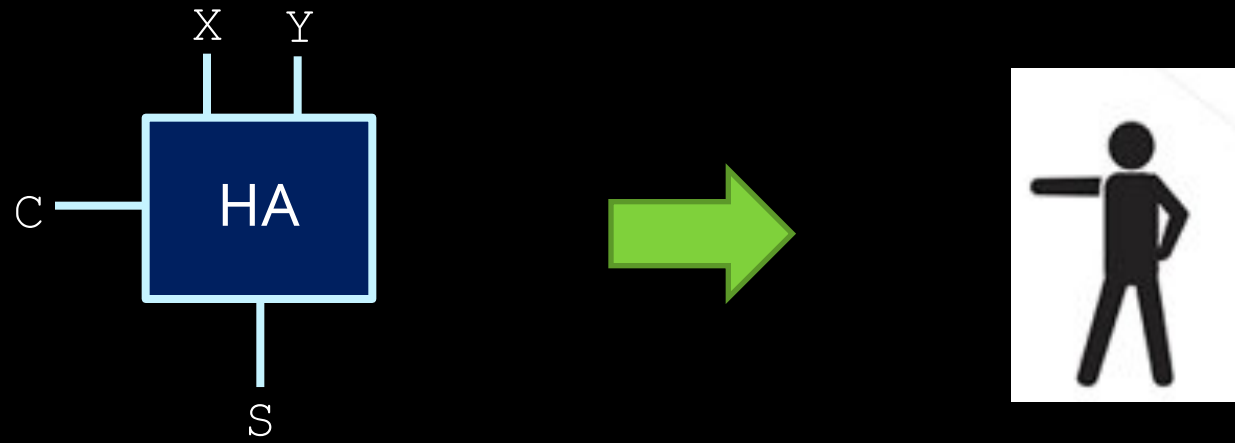


Now we can add one bit properly, but most of the numbers we use have more than one bits.

- int, unsigned int: 32 bits (architecture-dependent)
- short int, unsigned short int: 16 bits
- long long int, unsigned long long int: 64 bit
- char, unsigned char: 8 bits



How do we add multiple-bit numbers?



Each full adder takes in a carry bit and outputs a carry bit.

Each full adder can take in a carry bit which is output by another full adder.

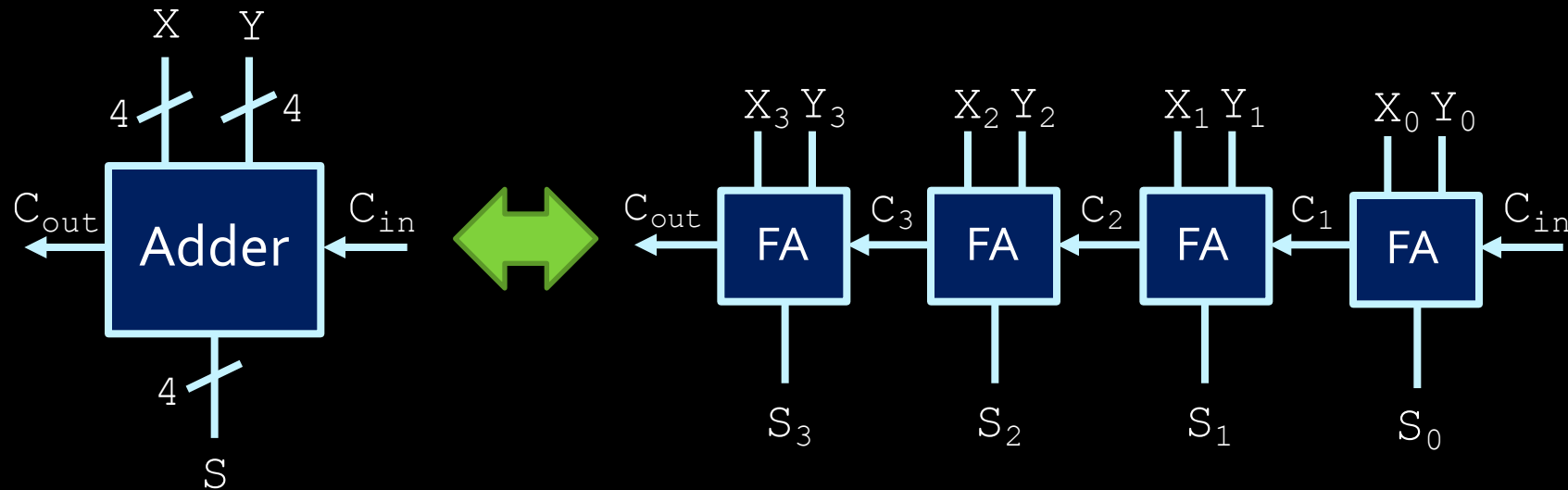
That is, they can be **chained** up.

Ripple-Carry Binary Adder

Full adders chained up,
for **multiple-bit** addition

Ripple-Carry Binary Adder

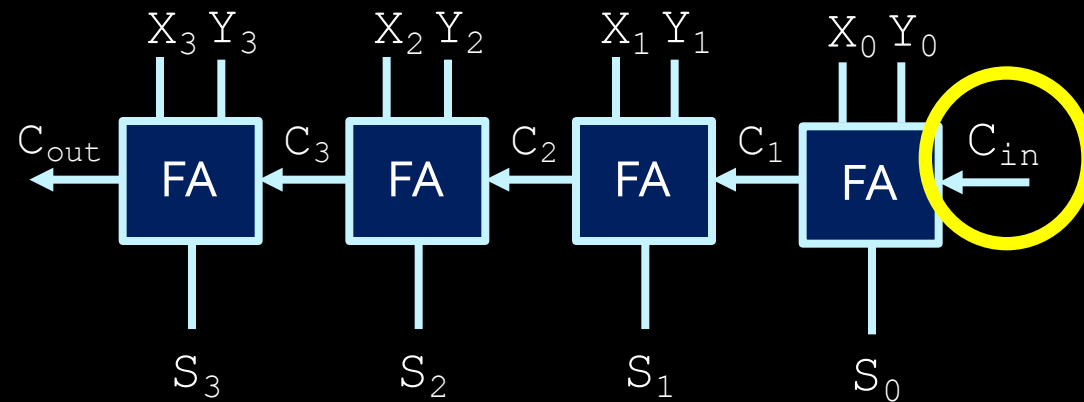
- Full adder units are chained together in order to perform operations on signal **vectors**.



$S_3S_2S_1S_0$ is the sum of $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$

The role of C_{in}

- Why can't we just have a half-adder for the smallest (right-most) bit?
- Because if we can use it to do **subtraction**!



Let's play a game...

1. Find a partner
2. Each of you pick a number between 0 and 31
3. Convert both numbers to binary
4. Invert each digit of the smaller number
5. Add up the big binary number and the inverted small binary number
6. Add 1 to the result, keep the lowest 5 digits
7. Convert the result to a decimal number

What do you get?

**You just did subtraction
without doing subtraction!**



Subtractors

- Subtractors are an extension of adders.
 - Basically, perform addition on a negative number.
- Before we can do subtraction, need to understand negative binary numbers.
- Two types:
 - **Unsigned** = a separate bit exists for the sign; data bits store the positive version of the number.
 - **Signed** = all bits are used to store a **2's complement** negative number.

Two's complement

- Need to know how to get **1's complement**:
 - Given number X with n bits, take $(2^n - 1) - X$
 - Negates each individual bit (bitwise NOT).

01001101	→	10110010
11111111	→	00000000

- **2's complement** = (1's complement + 1)

01001101	→	10110011
11111111	→	00000001

} Know
this!

- Note: Adding a 2's complement number to the original number produces a result of zero.

$$(\text{2's complement of } A) + A = 0.$$

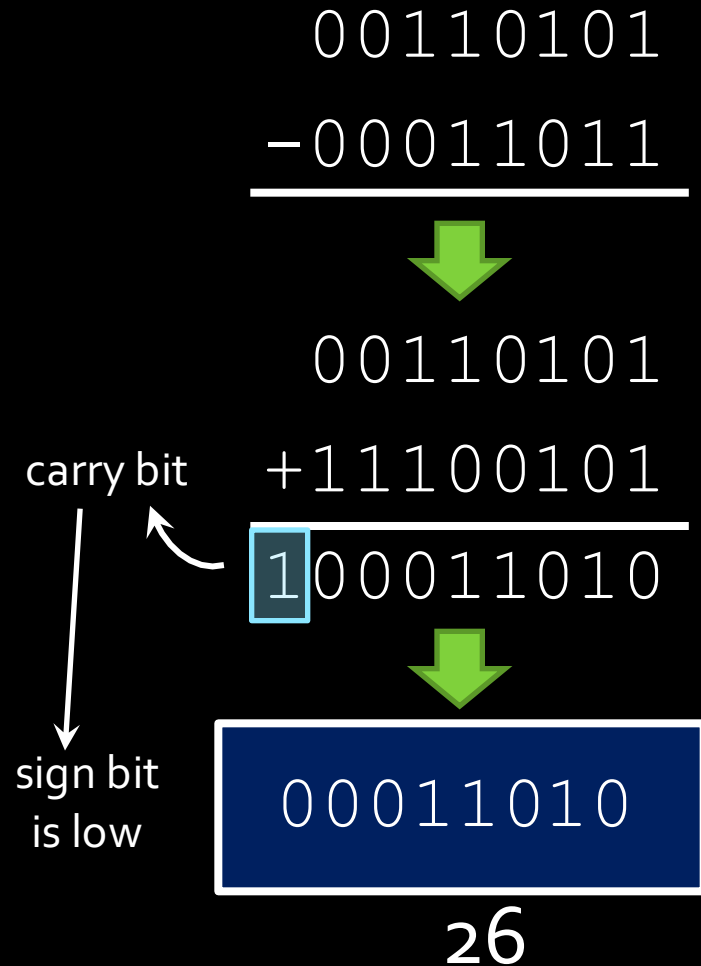
The 2's complement of A is like $-A$

Unsigned subtraction

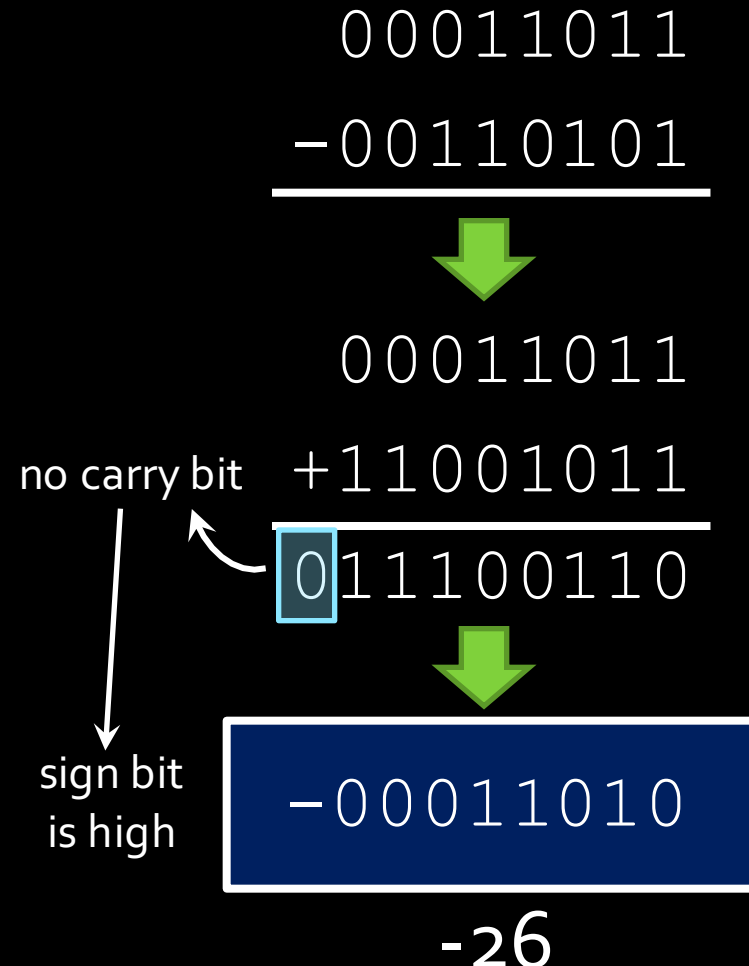
- General algorithm for **A - B**:
 1. Get the **2's complement** of **B** ($-B$)
 2. Add that value to **A**
 3. If there is an end carry (C_{out} is high), the final result is positive and does not change.
 4. If there is no end carry (C_{out} is low), get the 2's complement of the result ($B-A$) and add a **negative sign** to it, or set the sign bit high ($-(B-A) = A-B$).

Unsigned subtraction example

▪ $53 - 27$



▪ $27 - 53$



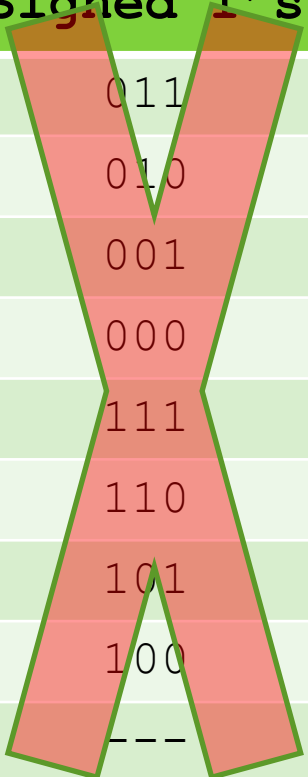
Signed subtraction (easier)

- Store negative numbers in 2's complement notation.
 - Subtraction can then be performed by using the binary **adder** circuit with negative numbers.
 - To compute $A - B$, just do $A + (-B)$
 - Need to get $-B$ first (the 2's complement of B)

Signed representations

Why 2's complement is better than 1's complement?

Decimal	Signed 1's	Signed 2's
3	011	011
2	010	010
1	001	001
0	000	000
-0	111	---
-1	110	111
-2	101	110
-3	100	101
-4	---	100



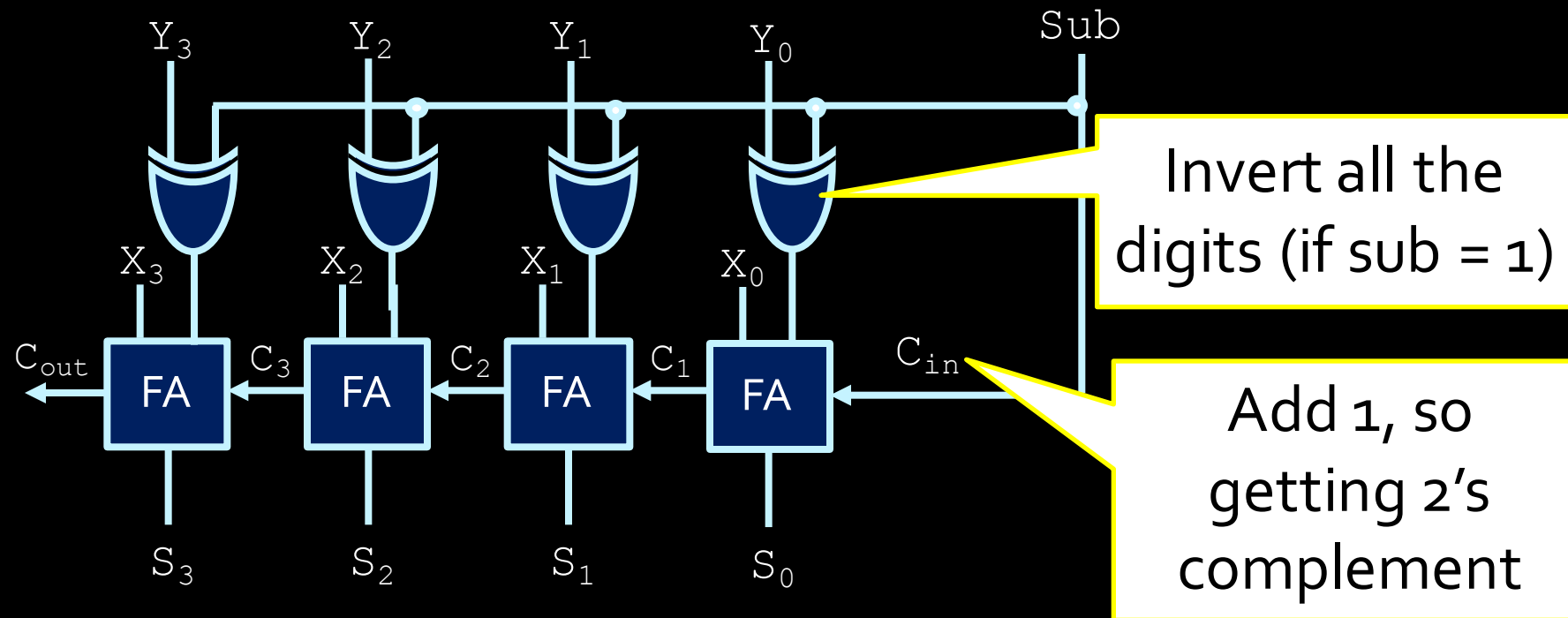
Trivia about sign numbers

- The **largest positive** 8-bit signed integer?
 - $01111111 = 127$ (0 followed by all 1)
- The **smallest negative** 8-bit signed integer?
 - $10000000 = -128$ (1 followed by all 0)
- The binary form 8-bit signed integer **-1**?
 - 11111111 (all one)
- For n-bit signed number there are 2^n possible values
 - 2^{n-1} are negative numbers (e.g. 8 bit, -1 to -128)
 - $2^{n-1}-1$ are positive number (e.g. 8 bit, 1 to 127)
 - and a zero



-128: 10000000 (signed)

Subtraction circuit



- If $\text{sub} = 0$, $S = X + Y$
- If $\text{sub} = 1$, $S = X - Y$

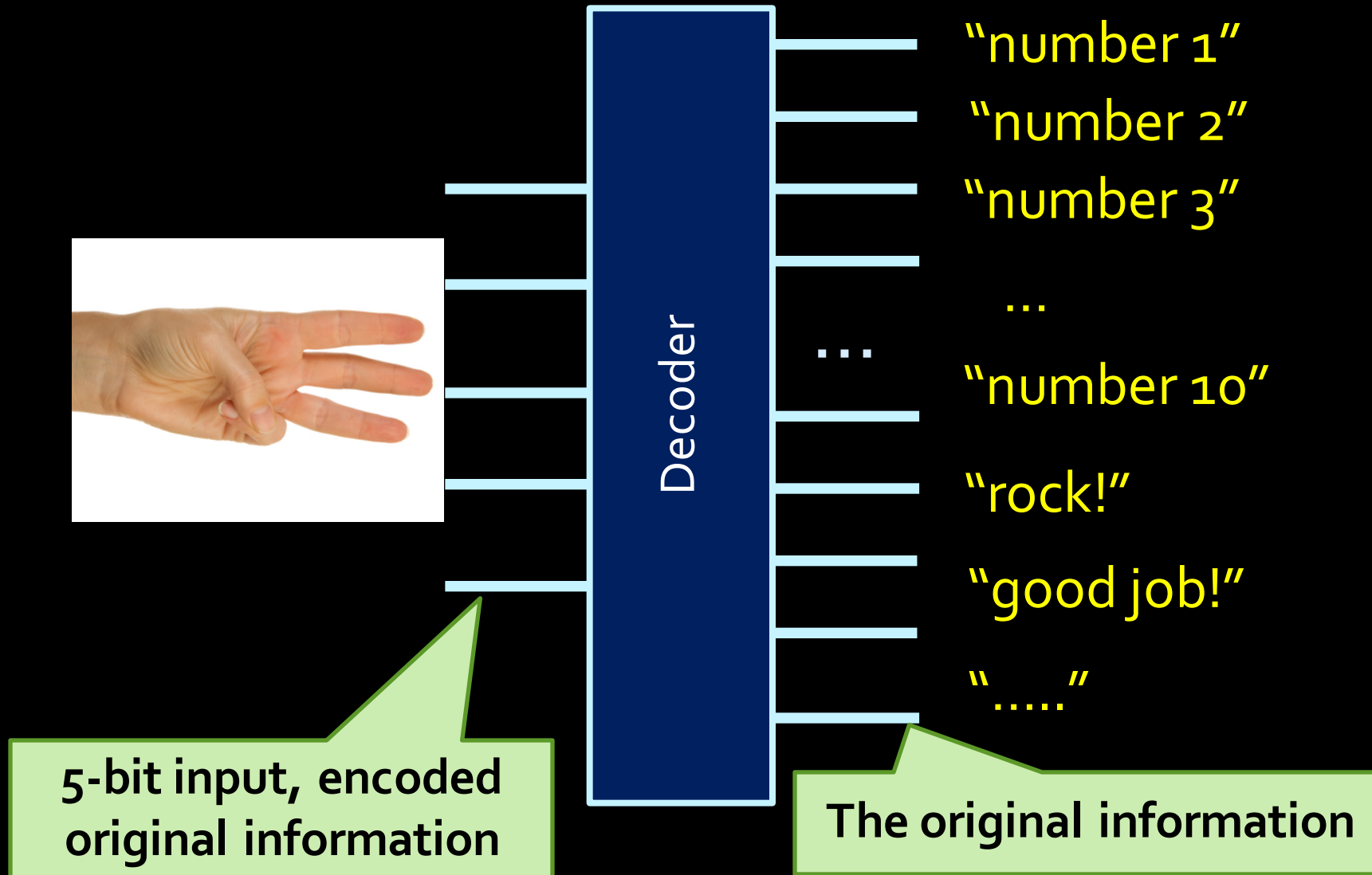
One circuit, both adder or subtractor



Decoders

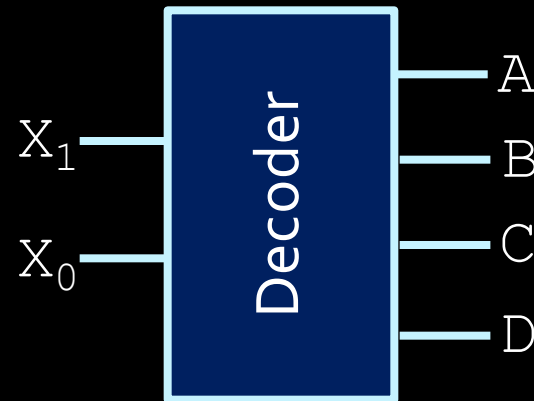


What is a decoder?



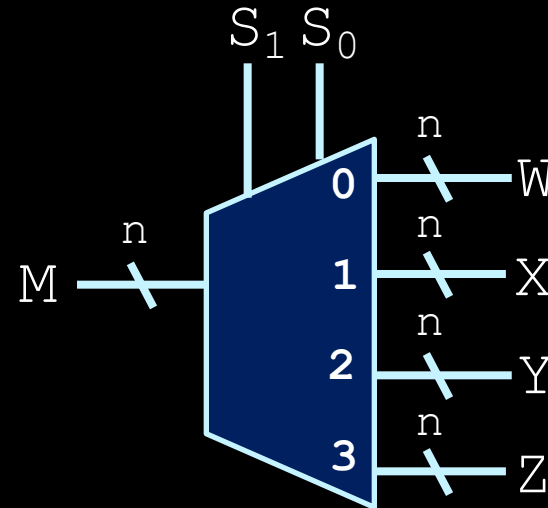
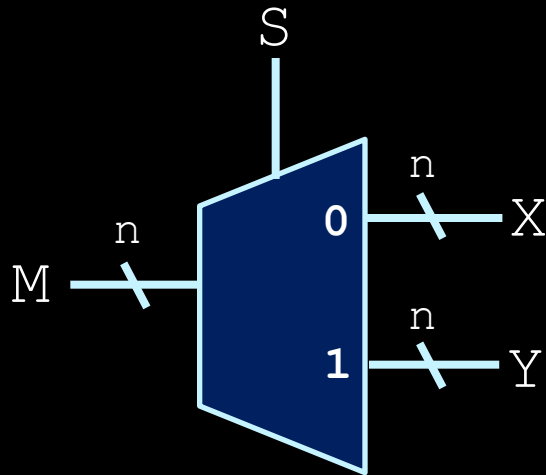
Decoders

- Decoders are essentially translators.
 - Translate from the output of one circuit to the input of another.
- Example: Binary signal splitter
 - Activates one of four output lines, based on a two-digit binary number.



Demultiplexers

- Related to decoders: demultiplexers.
 - Does multiplexer operation, in reverse.



Multiplexer:

Choose one from multiple inputs as output

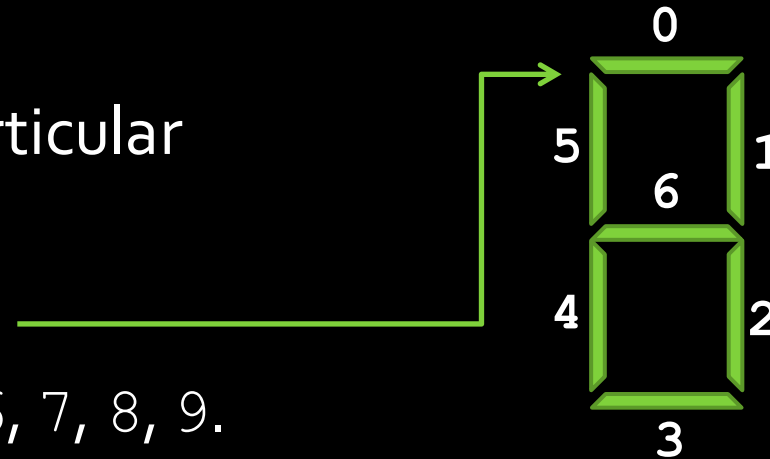
Demultiplexer:

One input chooses from multiple outputs

7-segment decoder



- Common and useful decoder application.
 - Translate from a 4-digit binary number to the seven segments of a digital display.
 - Each output segment has a particular logic that defines it.
 - Example: Segment 0
 - Activate for values: 0, 2, 3, 5, 6, 7, 8, 9.
 - In binary: 0000, 0010, 0011, 0101, 0110, 0111, 1000, 1001.
 - First step: Build the truth table and K-map.



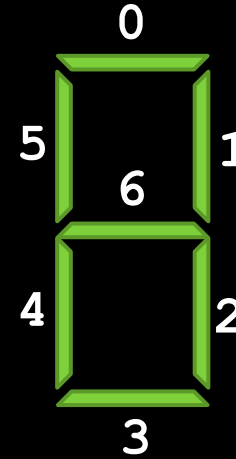
Note

What we talk about here is NOT the same as what we do in Lab 2

- In labs we translate numbers 0, 1, 3, 4, 5, 6 to displayed letters such as (K, E, L, V, I, N)
 - This is specially defined for the lab
- Here we are talking about translating 0, 1, 2, 3, 4,..., to displayed 0, 1, 2, 3, 4, ...
 - This is more common use

7-segment decoder

- For 7-seg decoders, turning a segment on involves driving it low.
 - i.e. Assuming a 4-digit binary number, segment 0 is low whenever input number is 0000, 0010, 0011, 0101, 0110, 0111, 1000 or 1001, and high whenever input number is 0001 or 0100.
 - This create a truth table and map like the following...



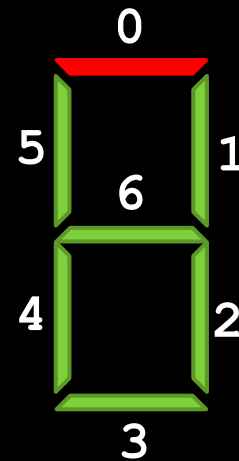
7-segment decoder

x_3	x_2	x_1	x_0	HEX ₀
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

6 rows missing!
1010 ~ 1111

	$\bar{x}_1 \cdot \bar{x}_0$	$\bar{x}_1 \cdot x_0$	$x_1 \cdot x_0$	$x_1 \cdot \bar{x}_0$
$\bar{x}_3 \cdot \bar{x}_2$	0	1	0	0
$\bar{x}_3 \cdot x_2$	1	0	0	0
$x_3 \cdot x_2$	x	x	x	x
$x_3 \cdot \bar{x}_2$	0	0	x	x

- $HEX_0 = \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot \bar{x}_0$
- But what about input values from 1010 to 1111?



“Don’t care” values

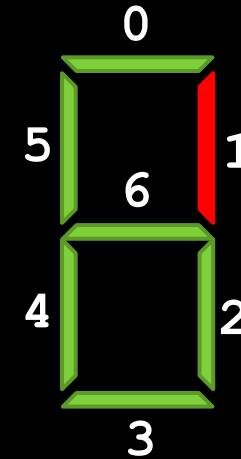
- Some input values will never happen, so their output values do not have to be defined.
 - Recorded as 'X' in the Karnaugh map.
- These values can be assigned to whatever values you want, when constructing the final circuit.

$$\text{HEX0} = \overline{x}_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 \\ + x_2 \cdot \overline{x}_1 \cdot \overline{x}_0$$

	$\overline{x}_1 \cdot \overline{x}_0$	$\overline{x}_1 \cdot x_0$	$x_1 \cdot x_0$	$x_1 \cdot \overline{x}_0$
$\overline{x}_3 \cdot \overline{x}_2$	0	1	0	0
$\overline{x}_3 \cdot x_2$	1	0	0	0
$x_3 \cdot x_2$	X	X	X	X
$x_3 \cdot \overline{x}_2$	0	0	X	X

Boxes can cover “x”s, or not, whichever you like.

Again for segment 1



x_3	x_2	x_1	x_0	HEX ₁
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

	$\bar{x}_1 \cdot \bar{x}_0$	$\bar{x}_1 \cdot x_0$	$x_1 \cdot x_0$	$x_1 \cdot \bar{x}_0$
$\bar{x}_3 \cdot \bar{x}_2$	0	0	0	0
$\bar{x}_3 \cdot x_2$	0	1	0	1
$x_3 \cdot x_2$	x	x	x	x
$x_3 \cdot \bar{x}_2$	0	0	x	x

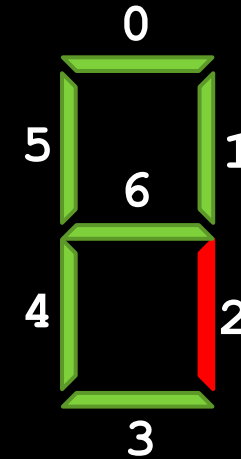
$$\text{HEX1} = x_2 \cdot \bar{x}_1 \cdot x_0 + x_2 \cdot x_1 \cdot \bar{x}_0$$

Again for segment 2

x_3	x_2	x_1	x_0	HEX ₂
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

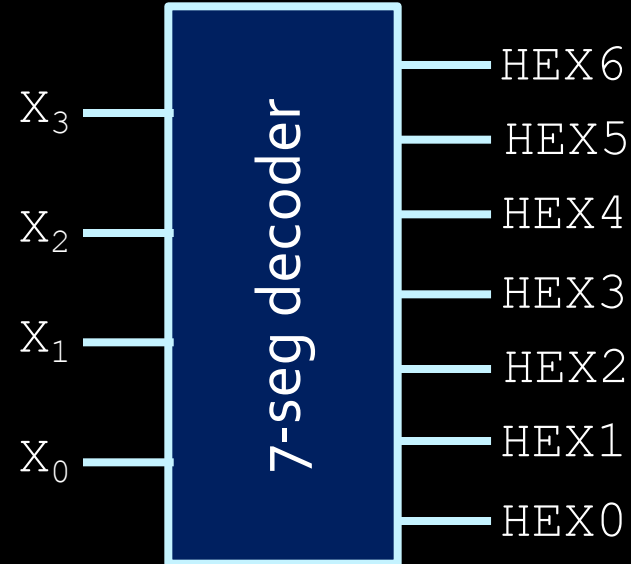
	$\bar{x}_1 \cdot \bar{x}_0$	$\bar{x}_1 \cdot x_0$	$x_1 \cdot x_0$	$x_1 \cdot \bar{x}_0$
$\bar{x}_3 \cdot \bar{x}_2$	0	0	0	1
$\bar{x}_3 \cdot x_2$	0	0	0	0
$x_3 \cdot x_2$	x	x	x	x
$x_3 \cdot \bar{x}_2$	0	0	x	x

$$\text{HEX2} = \bar{x}_2 \cdot x_1 \cdot \bar{x}_0$$



The final 7-seg decoder

- Decoders all look the same, except for the inputs and outputs.
- Unlike other devices, the implementation differs from decoder to decoder.





Comparators



Comparators

- A circuit that takes in two input vectors, and determines if the first is greater than, less than or equal to the second.
- How does one make that in a circuit?



Basic Comparators

- Consider two binary numbers A and B, where A and B are one bit long.
- The circuits for this would be:

□ $A=B$:

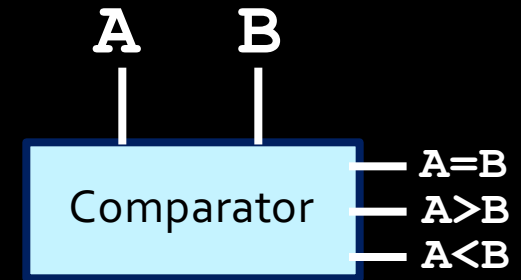
$$A \cdot B + \bar{A} \cdot \bar{B}$$

□ $A>B$:

$$A \cdot \bar{B}$$

□ $A<B$:

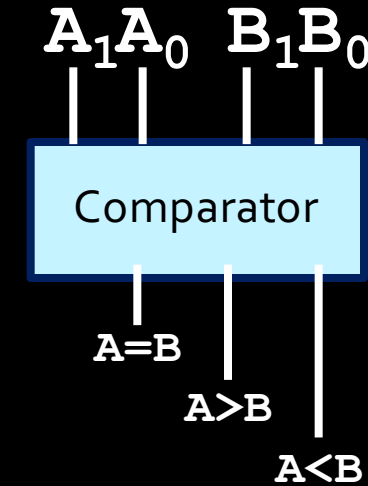
$$\bar{A} \cdot B$$



A	B
0	0
0	1
1	0
1	1

Basic Comparators

- What if A and B are two bits long?
- The terms for this circuit for have to expand to reflect the second signal.
- For example:



□ $A==B$:

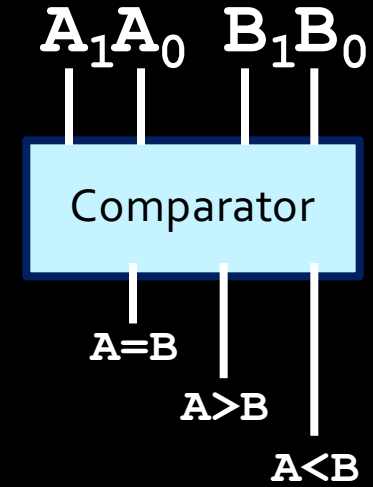
$$(A_1 \cdot B_1 + \bar{A}_1 \cdot \bar{B}_1) \cdot (A_0 \cdot B_0 + \bar{A}_0 \cdot \bar{B}_0)$$

Make sure that the values
of bit 1 are the same

Make sure that the values
of bit 0 are the same

Basic Comparators

- What about checking if A is greater or less than B?



□ $A>B$:

$$A_1 \cdot \bar{B}_1 + (A_1 \cdot B_1 + \bar{A}_1 \cdot \bar{B}_1) \cdot (A_0 \cdot \bar{B}_0)$$

Check if first bit satisfies condition

If not, check that the first bits are equal...

...and then do the 1-bit comparison

□ $A<B$:

$$\bar{A}_1 \cdot B_1 + (A_1 \cdot B_1 + \bar{A}_1 \cdot \bar{B}_1) \cdot (\bar{A}_0 \cdot B_0)$$

$A > B$ if and only if $A_1 > B_1$ or ($A_1 = B_1$ and $A_0 > B_0$)

General Comparators

- The general circuit for comparators requires you to define equations for each case.
- Case #1: Equality
 - If inputs A and B are equal, then all bits must be the same.
 - Define X_i for any digit i :
 - (equality for digit i)
 - Equality between A and B is defined as:

$$X_i = A_i \cdot B_i + \overline{A_i} \cdot \overline{B_i}$$

$$A==B : X_0 \cdot X_1 \cdot \dots \cdot X_n$$

Comparators

- Case #2: $A > B$

- The first non-matching bits occur at bit i , where $A_i=1$ and $B_i=0$. All higher bits match.
- Using the definition for X_i from before:

$$A > B = A_n \cdot \bar{B}_n + X_n \cdot A_{n-1} \cdot \bar{B}_{n-1} + \dots + A_0 \cdot \bar{B}_0 \cdot \prod_{k=1}^n X_k$$

- Case #3: $A < B$

- The first non-matching bits occur at bit i , where $A_i=0$ and $B_i=1$. Again, all higher bits match.

$$A < B = \bar{A}_n \cdot B_n + X_n \cdot \bar{A}_{n-1} \cdot B_{n-1} + \dots + \bar{A}_0 \cdot B_0 \cdot \prod_{k=1}^n X_k$$

Comparator truth table

- Given two input vectors of size $n=2$, output of circuit is shown at right.

Inputs				Outputs		
A_1	A_0	B_1	B_0	$A < B$	$A = B$	$A > B$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Comparator example (cont'd)

$A < B :$

	$\overline{B}_0 \cdot \overline{B}_1$	$B_0 \cdot \overline{B}_1$	$B_0 \cdot B_1$	$\overline{B}_0 \cdot B_1$
$\overline{A}_0 \cdot \overline{A}_1$	0	1	1	1
$A_0 \cdot \overline{A}_1$	0	0	1	1
$A_0 \cdot A_1$	0	0	0	0
$\overline{A}_0 \cdot A_1$	0	0	1	0

$$LT = B_1 \cdot \overline{A}_1 + B_0 \cdot B_1 \cdot \overline{A}_0 + B_0 \cdot \overline{A}_0 \cdot \overline{A}_1$$

Comparator example (cont'd)

$A=B :$

	$\overline{B}_0 \cdot \overline{B}_1$	$B_0 \cdot \overline{B}_1$	$B_0 \cdot B_1$	$\overline{B}_0 \cdot B_1$
$\overline{A}_0 \cdot \overline{A}_1$	1	0	0	0
$A_0 \cdot \overline{A}_1$	0	1	0	0
$A_0 \cdot A_1$	0	0	1	0
$\overline{A}_0 \cdot A_1$	0	0	0	1

$$EQ = \overline{B}_0 \cdot \overline{B}_1 \cdot \overline{A}_0 \cdot \overline{A}_1 + B_0 \cdot \overline{B}_1 \cdot A_0 \cdot \overline{A}_1 + \\ B_0 \cdot B_1 \cdot A_0 \cdot A_1 + \overline{B}_0 \cdot B_1 \cdot \overline{A}_0 \cdot A_1$$

Comparator example (cont'd)

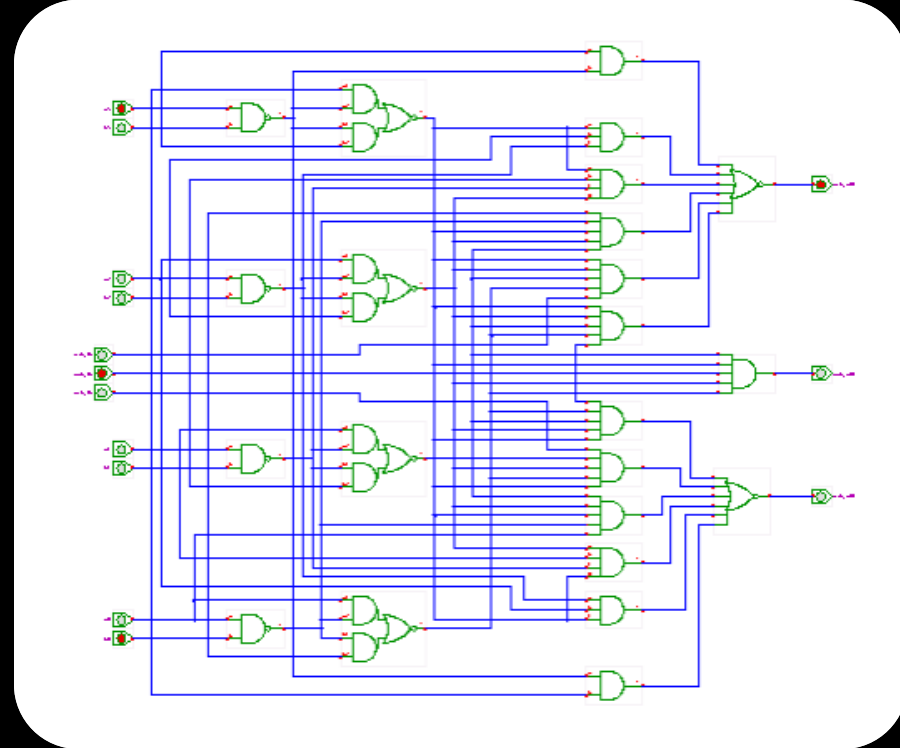
$A > B :$

	$\overline{B}_0 \cdot \overline{B}_1$	$B_0 \cdot \overline{B}_1$	$B_0 \cdot B_1$	$\overline{B}_0 \cdot B_1$
$\overline{A}_0 \cdot \overline{A}_1$	0	0	0	0
$A_0 \cdot \overline{A}_1$	1	0	0	0
$A_0 \cdot A_1$	1	1	0	1
$\overline{A}_0 \cdot A_1$	1	1	0	0

$$GT = \overline{B}_1 \cdot A_1 + \overline{B}_0 \cdot \overline{B}_1 \cdot A_1 + \overline{B}_0 \cdot A_0 \cdot A_1$$

Comparing larger numbers

- As numbers get larger, the comparator circuit gets more complex.
- At a certain level, it can be easier sometimes to just process the result of a subtraction operation instead.
 - Easier, less circuitry, just not faster.



Today we learned

Nothing new really. Used the circuit creation procedure to create some commonly useful circuits

- Mux and demux
- Adder and subtractor
- Decoder
- Comparator

Next week:

- Sequential circuits: circuits that have **memories**.

QUIZ
Time !

Question 1

Given that minterm $A'B'CD'$ is m_2 (0010)

$AB'C'D$ is m_9 (1001)

Question 2

Consider the **optimal** set of boxes that result in the most reduced circuit expression of the following K-map.

How many boxes are needed?

3

What is the size of the smallest box?

4

	$\bar{C} \cdot \bar{D}$	$\bar{C} \cdot D$	$C \cdot D$	$C \cdot \bar{D}$
$\bar{A} \cdot \bar{B}$	1	1	0	1
$\bar{A} \cdot B$	0	1	1	0
$A \cdot B$	0	1	1	0
$A \cdot \bar{B}$	1	1	0	1

Question 3

Convert the following Sum-Of-Minterms to the **equivalent** Product-Of-Maxterms expression.

$$AB + A'B$$

$$\text{Ans: } (A + B)(A' + B)$$

Draw the truth table, the minterms corresponds to the rows with output 1, to get POM, look at the rows with output 0.

In this question, rows with output 1 are 11 and 01, rows with output 0 are 00 (maxterm $A+B$) and 10 (maxterm $A' + B$)