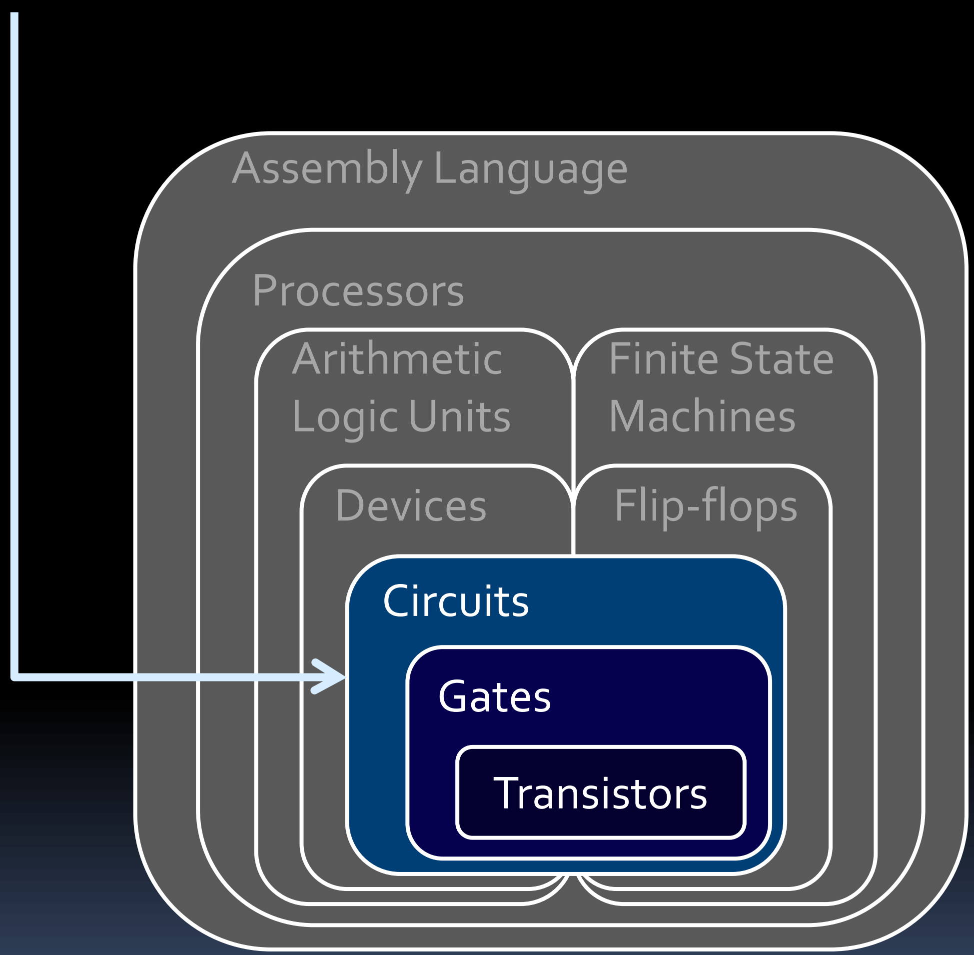
Week 3:
Logical Devices

Last week


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Karnaugh map review

	$\overline{B}\cdot\overline{C}$	$\overline{B}\cdot C$	$B\cdot C$	$B\cdot\overline{C}$
$\overline{A}$	0	0	1	0
A	1	0	1	1

- K-maps provide an illustration of a circuit's minterms (or maxterms), and a guide to how neighbouring terms may be combined.

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

Karnaugh map example

- Create a circuit with four inputs (A, B, C, D), and two outputs (X, Y):
  - The output X is high whenever two or more of the inputs are high.
  - The output Y is high when three or more of the inputs are high.

A	B	C	D	X	Y
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Karnaugh map example

X:

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

$$X = A\cdot B + C\cdot D + B\cdot D + B\cdot C + A\cdot D + A\cdot C$$

Alternative for X: Maxterms

X:

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

Alternative for X: Maxterms

X:

	$C+D$	$C+\overline{D}$	$\overline{C}+\overline{D}$	$\overline{C}+D$
$A+B$	0	0	1	0
$A+\overline{B}$	0	1	1	1
$\overline{A}+\overline{B}$	1	1	1	1
$\overline{A}+B$	0	1	1	1

$$X = (A+C+D)\cdot(B+C+D)\cdot(A+B+C)\cdot(A+B+D)$$

Karnaugh map example

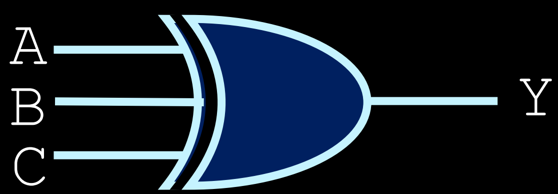
Y:

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

$$Y = A\cdot B\cdot D + B\cdot C\cdot D + A\cdot B\cdot C + A\cdot C\cdot D$$

Karnaugh map review

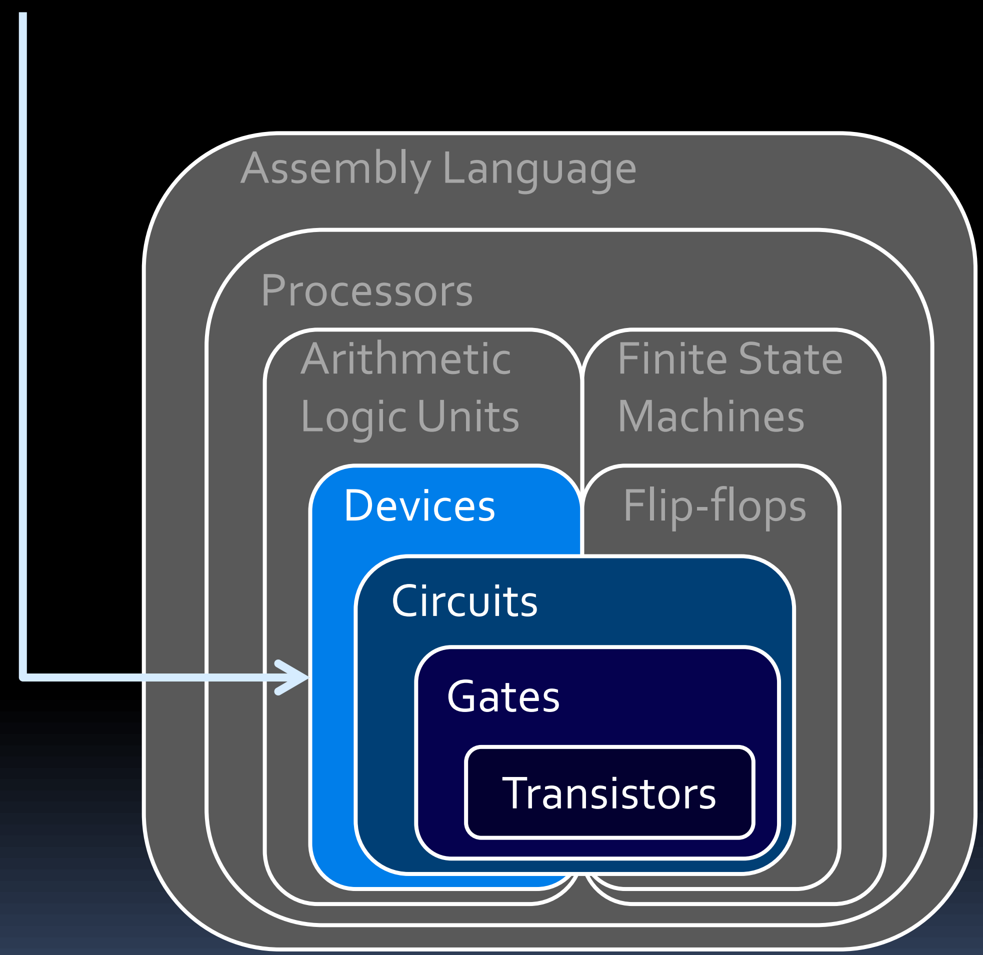
- <u>Note:</u> There are cases where no combinations are possible. K-maps cannot help these cases.
- <u>Example:</u> Multi-input XOR gates.



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

$$Y = \bar{A}\cdot\bar{B}\cdot C + A\cdot\bar{B}\cdot\bar{C} + \bar{A}\cdot B\cdot\bar{C} + A\cdot B\cdot C$$

We are here


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Building up from gates…

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

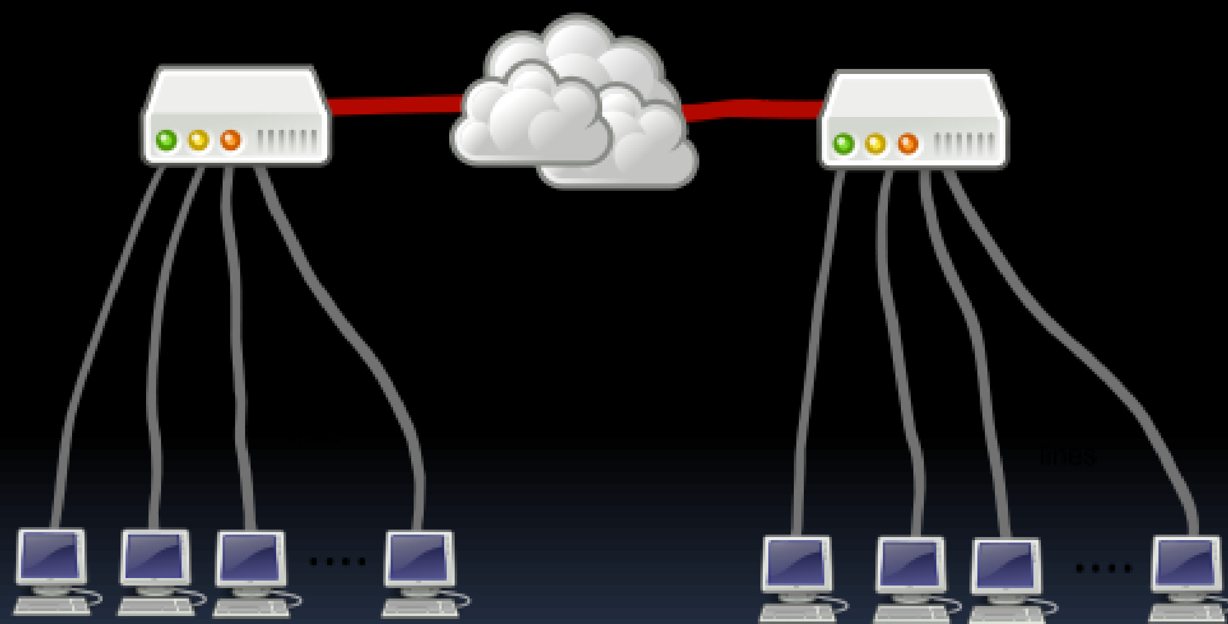
These are all combinational circuits

Combinational Circuits

- *Combinational Circuits* are any circuits where the outputs rely strictly on the inputs.
  - Everything we've done so far and what we'll do today is all combinational logic.

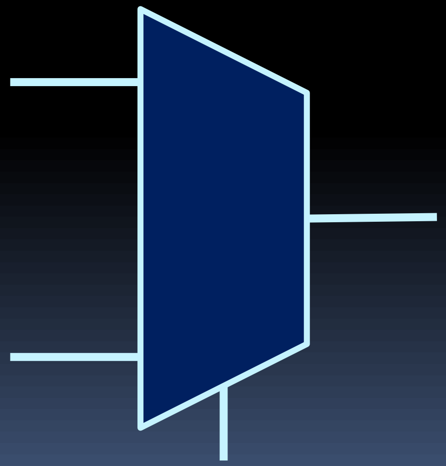
- Another category is *sequential circuits* that we will learn in the next few weeks.

Multiplexers



Mux Symbol

- Some circuits are so common to they have their own drawing.
- One of them is the multiplexor, or mux.

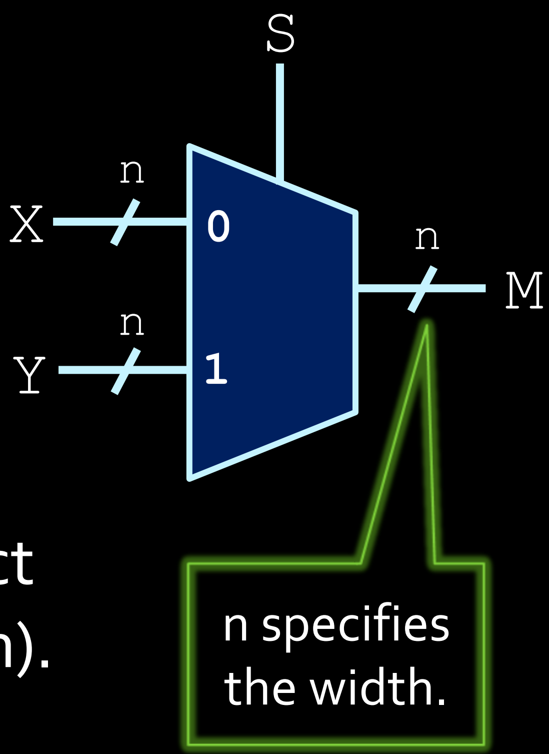


Multiplexer, or mux

- Switches between inputs:
  - Select one of multiple inputs.
  - Connect that input to the single output.
- A 2-to-1 mux will output `X` if `S` is `0`, and will output `Y` if `S` is `1`.

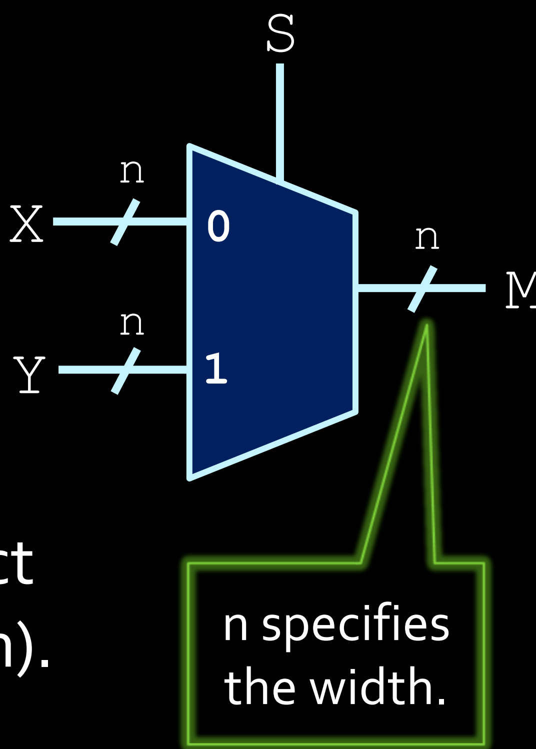
Multiplexer

- S is called the select input.
- X and Y are the data inputs.
- X and Y can have n data bits.
    - Note the number of select bits is distinct from the number of data bits (the width).

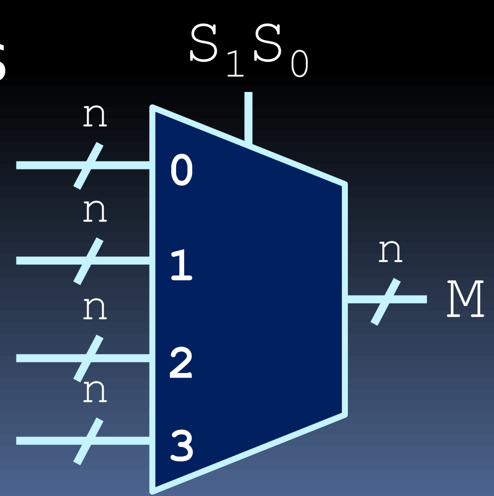


Multiplexer

- S is called the select input.
- X and Y are the data inputs.
- X and Y can have n data bits.
  - Note the number of select bits is distinct from the number of data bits (the width).



- A 4-to-1 mux would have 2 select bits
  - And as many data bits as we want!
- 8-to-1 mux → 3 select bits.



Multiplexer uses

- Muxes are very useful whenever you need to select from multiple input values.
- Your TV has at least one! You can select different input sources.
- More examples:
  - surveillance video monitors
  - digital cable boxes
  - routers.

Multiplexer design

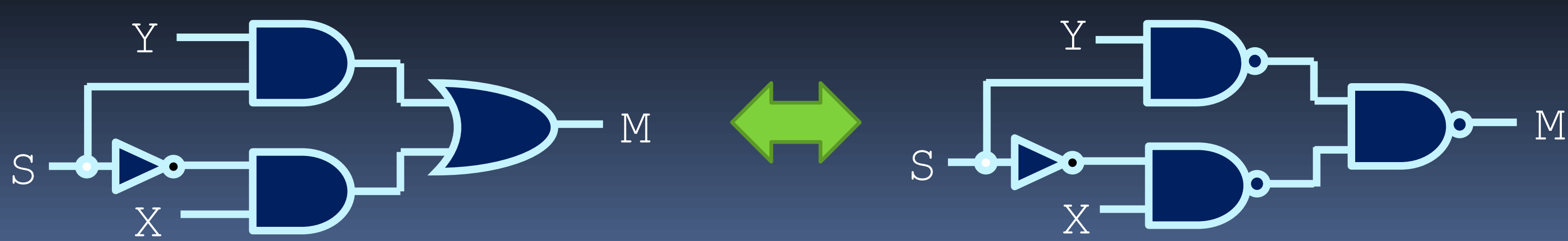
X	Y	S	M
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

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

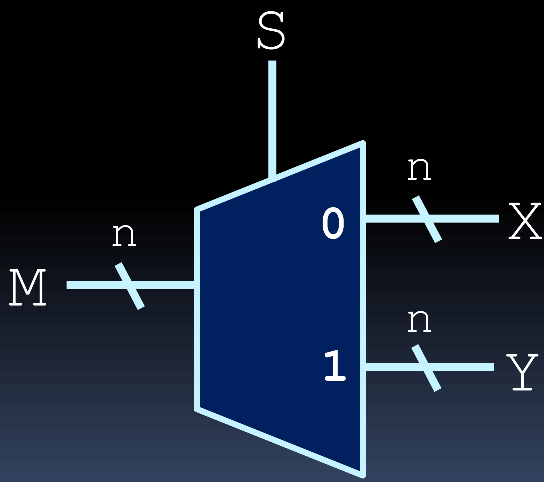
	$\overline{Y}\cdot\overline{S}$	$\overline{Y}\cdot S$	$Y\cdot S$	$Y\cdot\overline{S}$
$\overline{X}$	0	0	1	0
X	1	0	1	1

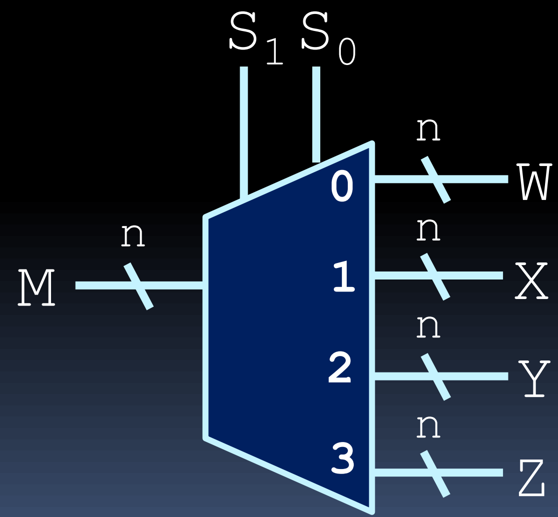
$$M = Y\cdot S + X\cdot\overline{S}$$



Demultiplexers

- Does multiplexer operation, in reverse:
  - Mux: one of multiple inputs → a single output
  - Demux: single input → one of multiple outputs.

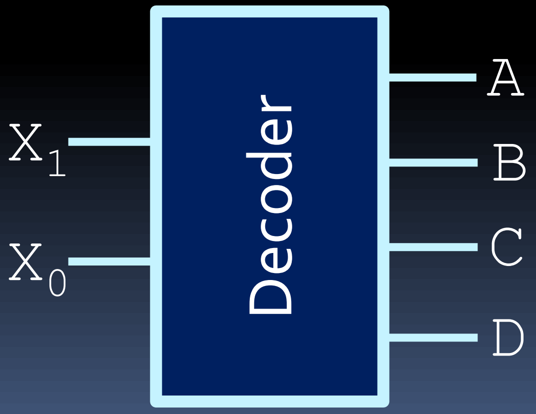




Decoders

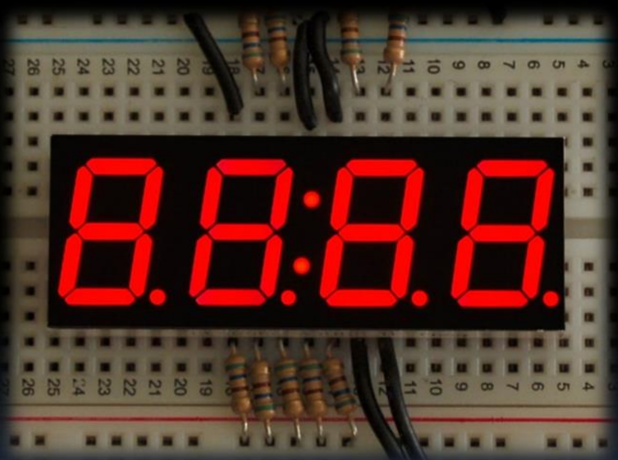
Decoders

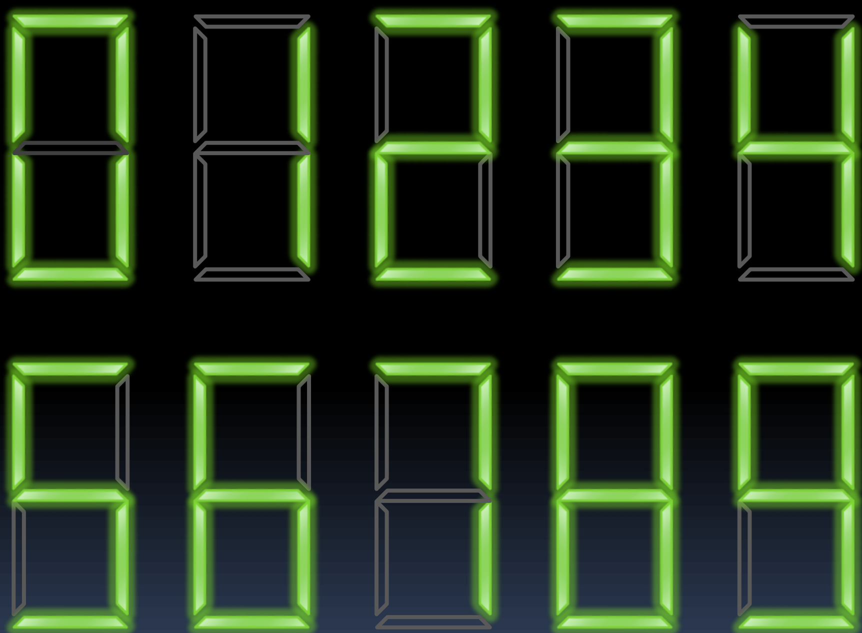
- Decoders are essentially binary translators.
  - Translate from the output of one circuit to the input of another.
  - Think of them as providing a mapping from a binary number to another encoding.
- <u>Example:</u> one-hot decoder
  - Activates one of four output lines, based on a two-digit binary number. (binary → "one-hot")



7-segment decoder

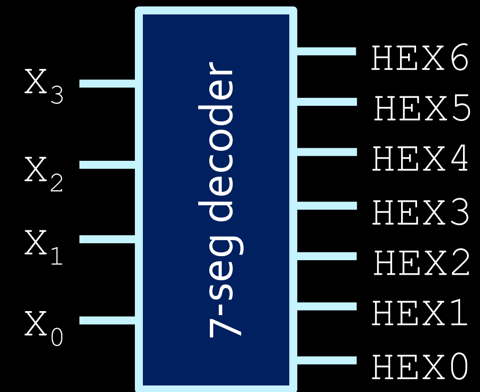
- Use 7 LEDs to show digits and even letters.

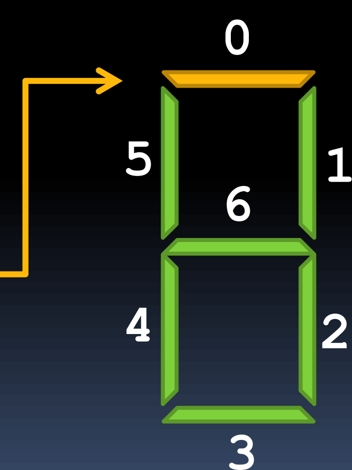




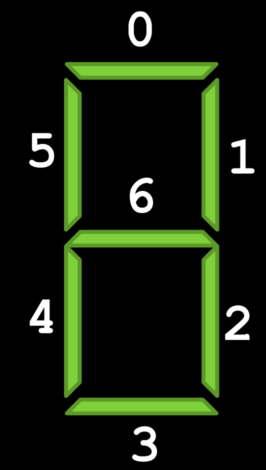
7-segment decoder

- Common and useful decoder.
  - Translate from a 4-digit binary number to the seven segments of a digital display.
  - Each segment controlled by wire.
  - For each output segment, we'll create Boolean logic for when to turn it on.
  - <u>Example:</u> Segment #0
    - Activate for inputs: 0, 2, 3, 5, 6, 7, 8, 9.
    - In binary: 0000, 0010, 0011, 0101, 0110, 0111, 1000, 1001.





7-segment decoder



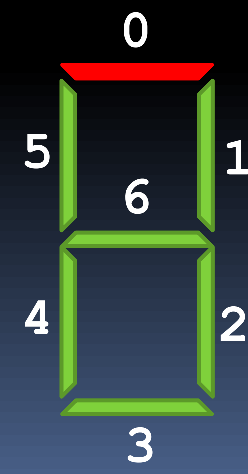
- Segments are "active-low", meaning they are on when the wire is low.
- <u>Example:</u> Displaying digits `0-9`
  - Assume input is a 4-digit binary number
  - Segment `0` (top segment) is low whenever the input values are `0000`, `0010`, `0011`, `0101`, `0110`, `0111`, `1000` or `1001`, and high whenever input number is `0001` or `0100`.
- <u>First step:</u> Build the truth table and K-map.

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

	$\overline{X_1} \cdot \overline{X_0}$	$\overline{X_1} \cdot X_0$	$X_1 \cdot X_0$	$X_1 \cdot \overline{X_0}$
$\overline{X_3} \cdot \overline{X_2}$	0	1	0	0
$\overline{X_3} \cdot X_2$	1	0	0	0
$X_3 \cdot X_2$	?	?	?	?
$X_3 \cdot \overline{X_2}$	0	0	?	?

- $HEX0 = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot X_0 + \overline{X_3} \cdot X_2 \cdot \overline{X_1} \cdot \overline{X_0}$
- But what about input values from 1010 to 1111?



“Don’t care” values

- Input values that will never happen or are not meaningful in a given design, and so their output values do not have to be defined.
  - Recorded as ‘X’ in truth-tables and K-Maps.

- In the K-maps we can think of these “don’t care” values as either 0 or 1 depending on what helps us simplify our circuit.
  - Note you do NOT change the X with a 0 or 1, you just include (or not include it) it in a grouping as needed.

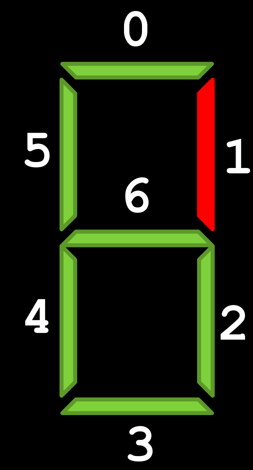
"Don't care" values

- New equation for `HEX0`:

	$\overline{x_1} \cdot \overline{x_0}$	$\overline{x_1} \cdot x_0$	$x_1 \cdot x_0$	$x_1 \cdot \overline{x_0}$
$\overline{x_3} \cdot \overline{x_2}$	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

$$\mathbf{HEX0} = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0}$$

Again for segment 1

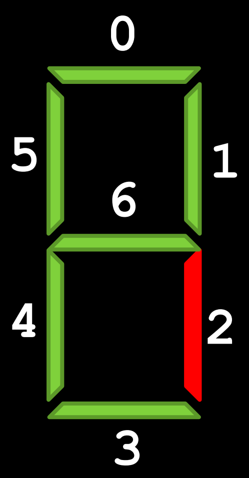


x_3	x_2	x_1	x_0	HEX_1
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

	$\overline{x_1}\cdot\overline{x_0}$	$\overline{x_1}\cdot x_0$	$x_1\cdot x_0$	$x_1\cdot\overline{x_0}$
$\overline{x_3}\cdot\overline{x_2}$	0	0	0	0
$\overline{x_3}\cdot x_2$	0	1	0	1
$x_3\cdot x_2$	x	x	x	x
$x_3\cdot\overline{x_2}$	0	0	x	x

$$HEX1 = x_2\cdot\overline{x_1}\cdot x_0 + x_2\cdot x_1\cdot\overline{x_0}$$

Again for segment 2



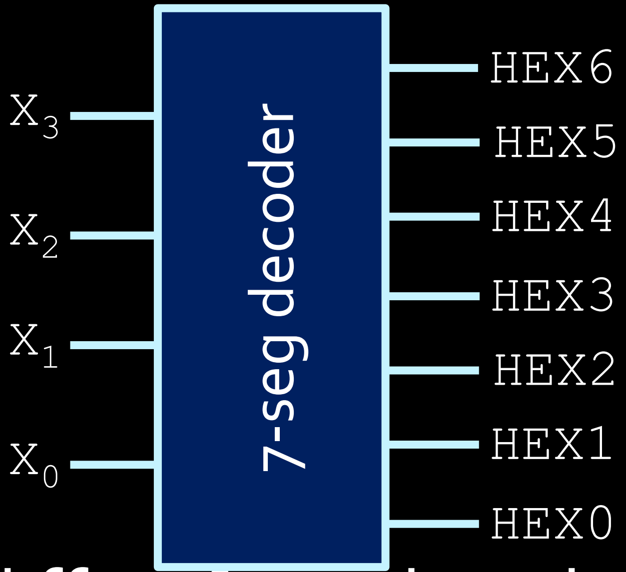
x_3	x_2	x_1	x_0	HEX_2
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0

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

$$HEX2 = \overline{x}_2 \cdot x_1 \cdot \overline{x}_0$$

The final 7-seg decoder

- There are many kinds of decoders.
- They all look the same, except for the inputs and outputs.



- Of course, the internals differs from decoder to decoder.
  - Most devices (e.g., mux) the internals are always the same…

Another "don't care" example

- Climate control fan:
  - The fan should turn on ($F=1$) if the temperature is hot ($H=1$) or if the temperature is cold ($C=1$), depending on whether the unit is set to A/C ($A=1$) or heating ($A=0$).

H	C	A	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

	$\overline{H}\cdot\overline{C}$	$\overline{H}\cdot C$	$H\cdot C$	$H\cdot\overline{C}$
$\overline{A}$	0	1	X	0
A	0	0	X	1

$$F = A\cdot H + \overline{A}\cdot C$$

Adder circuits

Adders

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

Review of Binary Math

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

$$258 = 2\times10^2 + 5\times10^1 + 8\times10^0$$

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

$$01101_2 = 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$$
$$= 13_{10}$$

Unsigned binary addition

- 27 + 53

27 = 00011011

53 = 00110101

```
 1 1 1 1 1 1
  00011011
 +00110101
 ---------
  01010000
```

80_{10}

01010000

Unsigned binary addition

- 27 + 53

27 = 00011011

53 = 00110101

```
   1 1 1 1 1 1
  00011011
 +00110101
 ---------
  01010000
```

80_{10}

01010000

- 95 + 181

```
  01011111
 +10110101
```

```
 1 1 1 1 1 1 1 1
  01011111
 +10110101
 ---------
 100010100
```

carry out bit

20_{10} ??

00010100

With 8 bits we can only represent unsigned numbers 0 to 255 !

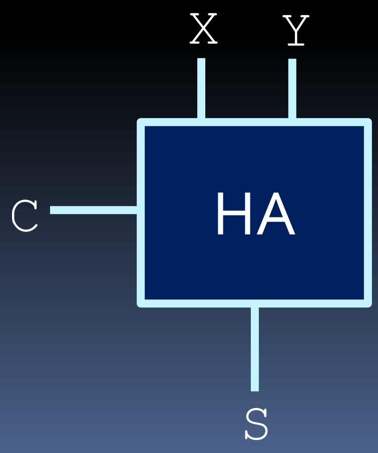
Half Adders

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

X	0	0	1	1
+Y	+0	+1	+0	+1
CS	00	01	01	10

This is a truth table!

- 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.

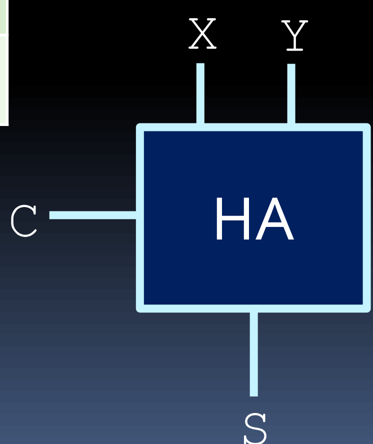


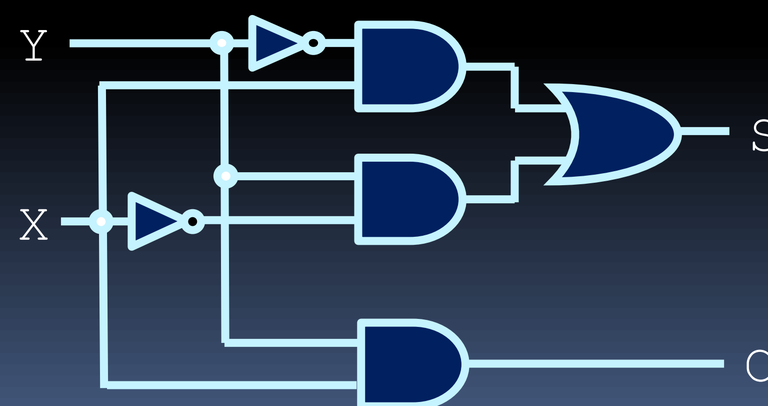
Half Adder Implementation

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

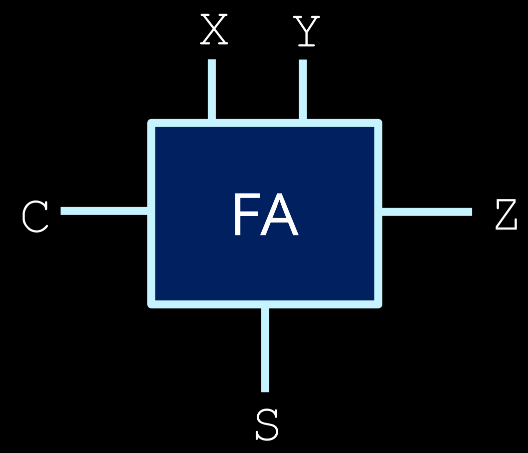
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = X \cdot Y \qquad S = X \cdot \overline{Y} + \overline{X} \cdot Y$$
$$= X \oplus Y$$





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
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	$\bar{Y}\cdot\bar{Z}$	$\bar{Y}\cdot Z$	$Y\cdot Z$	$Y\cdot\bar{Z}$
$\bar{X}$	0	0	1	0
X	0	1	1	1

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

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

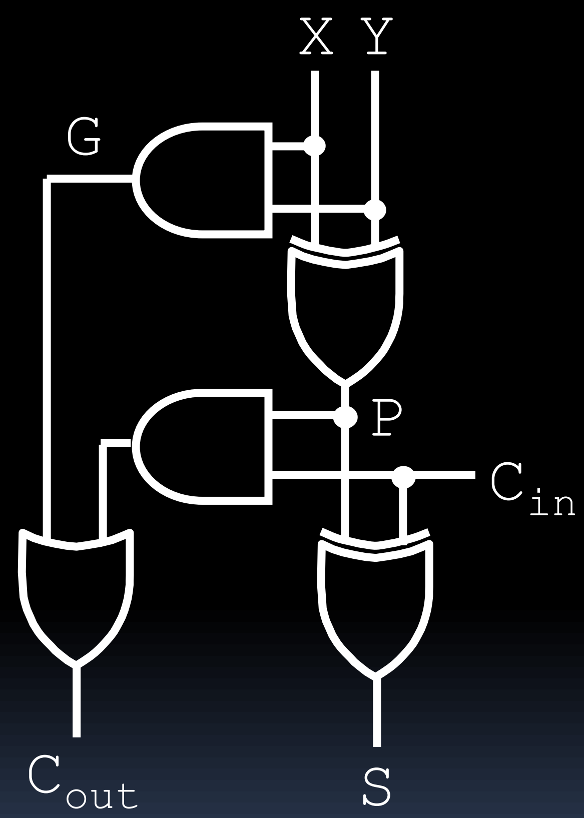
$$S = X \oplus Y \oplus Z$$

Full Adder Design

- The C term can also be rewritten as:

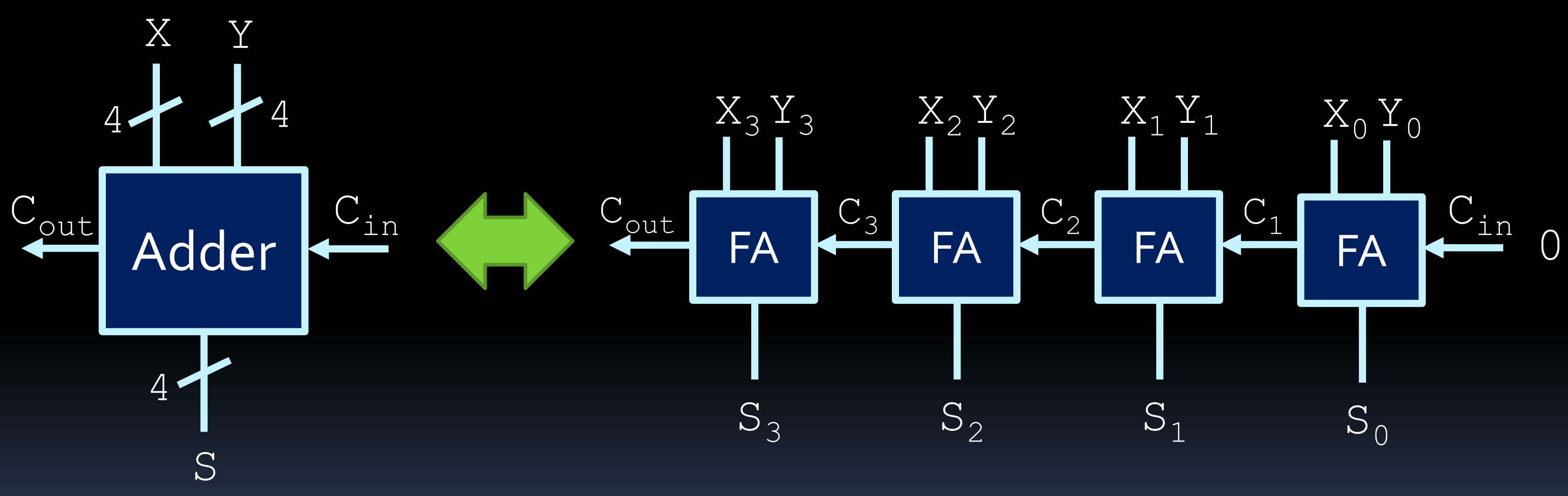
$$C = X \cdot Y + (X \oplus Y) \cdot Z$$

- Two terms come from this:
  - $X \cdot Y$ = carry generate (G).
  - $X \oplus Y$ = carry propagate (P).
- Results in this circuit →



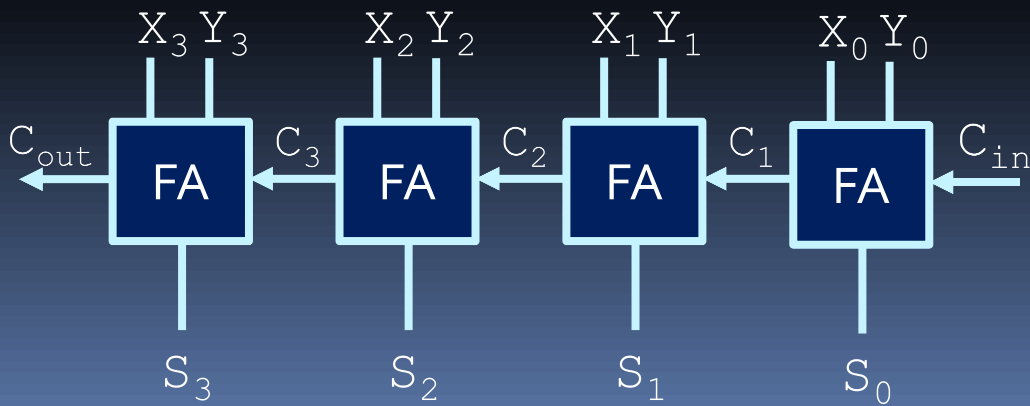
Ripple-Carry Binary Adder

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



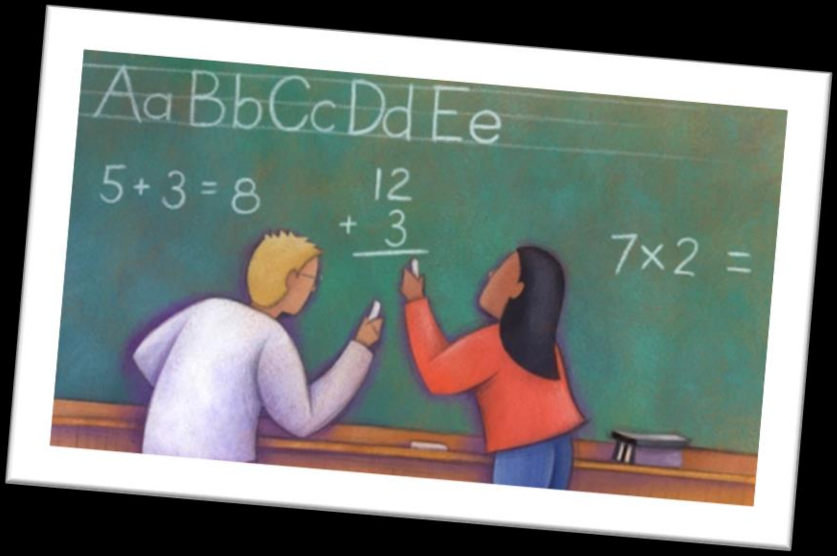
The role of C_{in}

- Why did we use a full-adder for the right-most (smallest) bit? can't we just have a half-adder?
- We could, if we were only interested in addition. But the last bit allows us to do subtraction as well!
  - Time for a little fun with subtraction!



Let's Play a Game!

1. Choose two five-digit binary numbers.
2. Take the smaller number and invert its digits.



3. Add this inverted number to the larger one.
4. Add one to the result.
5. Check what the result is…

Subtractors

- Subtractors are an extension of adders.
  - Basically, perform addition on a negative number.
- To do subtraction, we need to understand representation of negative binary numbers.
- Unsigned numbers
  - Data bits store the positive version of the number.
- Sign-and-magnitude:
  - Use a separate bit for the sign (the sign bit).
- Signed (2's complement):
  - Store a negative number using all bits.
  - More common, and what we use for this course.

Negative Binary Numbers

- Unsigned number:
  - All bits are data bits.
  - Data bits store a positive number.
- Example:
  Represent 46 as a 6-bit unsigned number:

bit value	2^5	2^4	2^3	2^2	2^1	2^0
	1	0	1	1	1	0

Negative Binary Numbers

- Sign and magnitude:
  - Need to set a side a bit to represent the sign
  - Data bits store the positive version of the number.
- Example:
  Represent -18 as a 5-bit plus 1 sign bit:

sign bit
1

bit value	2^4	2^3	2^2	2^1	2^0
	1	0	0	1	0

Negative Binary Numbers

- Signed (two's complement)
  - All bits are data bits.
  - Most significant bit (MST) has negative value.
- Example:
  Represent -18 as a 6-bit signed number:

bit value	-2^5	2^4	2^3	2^2	2^1	2^0
	1	0	1	1	1	0

This bit is worth -32

Convert to Two's Complement

- First get 1's complement:
  - Invert individual bits (bitwise NOT).
  - Given number `X` with `n` bits, this gives $(2^n-1)-X$

```
01001101    →    10110010
11111111    →    00000000
```

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

```
01001101    →    10110011
11111111    →    00000001
```

Know this!

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

Signed representations

Decimal	Unsigned	Signed 2's
7	111	---
6	110	---
5	101	---
4	100	---
3	011	011
2	010	010
1	001	001
0	000	000
-1	---	111
-2	---	110
-3	---	101
-4	---	100

Practice 2's complement!

- Assume 4-bits signed representation, write the following decimal numbers in binary:
  - 2 => `0010`
  - -1 => `1111`
  - 0 => `0000`
  - 8 => **Not possible to represent in 4 digits!**
  - -8 => `1000`
- What is max positive number? => 7 (or $2^{4-1} - 1$)
- What is min negative number? => -8 (or -2^{4-1})

Shortcuts for signed numbers

- When thinking of signed binary numbers, there are a few useful tricks to remember:
  - The largest positive binary number is a zero followed by all ones.
  - The binary value for -1 has ones in all the digits.
  - The most negative binary number is a one followed by all zeroes.

- There are 2^n possible values that can be stored in an n-digit binary number.
  - 2^{n-1} are negative, $2^{n-1}-1$ are positive, and one is zero.

  - For example, given an 8-bit binary number:
    - There are 256 possible values
    - One of those values is zero
    - 128 are negative values (11111111 to 10000000)
    - 127 are positive values (00000001 to 01111111)

-1 to -128

1 to 127

Signed subtraction

- Negative numbers are generally stored in 2's complement notation.
  - Reminder: 1's complement → bits are the bitwise NOT of the equivalent positive value.
  - 2's complement → 1's complement value plus one; results in zero when added to equivalent positive value.
- Subtraction can then be performed by using the binary adder circuit with negative numbers.

At the core of subtraction

- Subtraction of a number is simply the addition of its negative value.
- This the negative value is found using the 2's complement process.
  - $7 – 3 = 7 + (-3)$
  - $-3 – 2 = -3 + (-2)$

Signed Subtraction example

- 7 − 3

```
  0111
 -0011
 -----
  0111
 +1101
 -----
 10100
```

discarded

$0100 = 4_{10}$

- -3 − 2

```
  1101
 -0010
 -----
  1101
 +1110
 -----
 11011
```

discarded

$1011 = -5_{10}$

What about bigger numbers

- 53 – 27

```
  00110101
 -00011011
 ---------
  00110101
 +11100101
 ---------
 100011010
```

discarded

$\boxed{1}00011010$

$00011010 = 26_{10}$

- 27 – 53

```
  00011011
 -00110101
 ---------
  00011011
 +11001011
 ---------
 011100110
```

discarded

$\boxed{0}11100110$

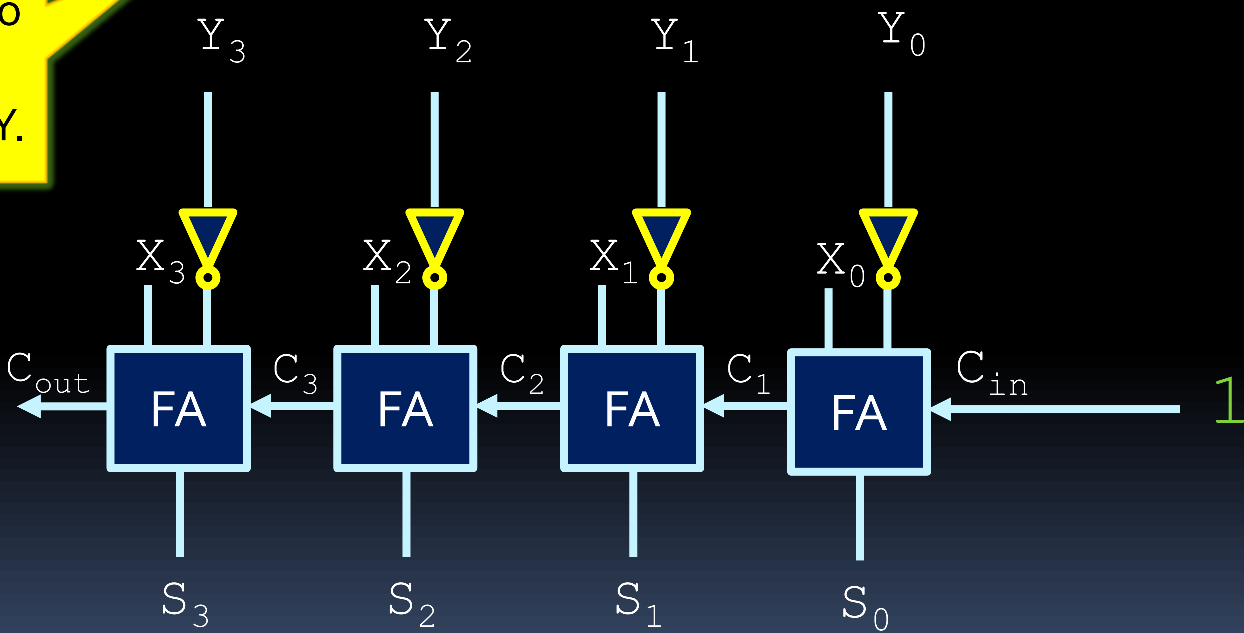
$11100110 = -26_{10}$

Subtraction circuit

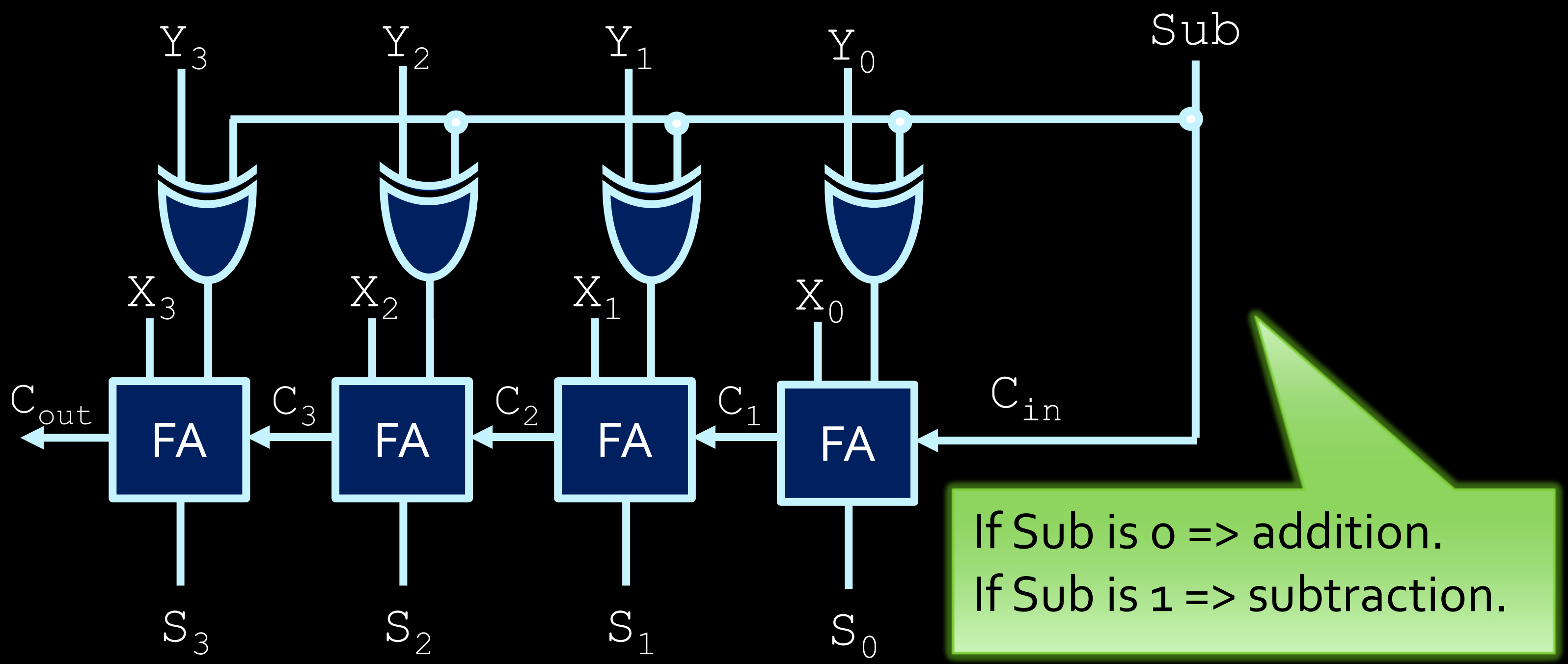
- 4-bit subtractor: X – Y
  - → X plus 2's complement of Y
  - → X plus 1's complement of Y plus 1

Feed 1 as Carry-In in the least significant FA.

Use NOT gates to get the 1's complement of Y.



Addition/Subtraction circuit



- The full adder circuit can be expanded to incorporate the subtraction operation
  - Remember: 2's complement = 1's complement + 1
  - We connect Sub to Cin

Food for Thought

- What happens if we add these two positive signed binary numbers `0110` + `0011` (i.e., `6` + `3`) ?
  - The result is `1001`.
  - But that is a negative number (−`7`)! ☹
- What happens if we add the two negative numbers `1000` + `1111` (i.e., −`8` + (−`1`))?
  - The result is `0111` with a carry-out. ☹
- We need to know when the result might be wrong.
  - This is usually indicated in hardware by the Overflow flag!
  - More about this when we'll talk about processors.

Subtracting unsigned numbers

- General algorithm for X - Y:
  (for sign-and-magnitude representation)
  1. Get the 2's complement of the subtrahend Y (the term being subtracted).
  2. Add that value to the minuend X (the term being subtracted from).
  3. If there is an end carry (C_{out} is high), the final result is positive and does not change (set sign bit of output to 0).
  4. If there is no end carry (C_{out} is low), get the 2's complement of the result and set the sign bit of output to 1.

Unsigned subtraction example

- 53 – 27

```
  00110101
 -00011011
```

- 27 – 53

```
  00011011
 -00110101
```

Unsigned subtraction example

- 53 – 27

```
  00110101
 -00011011
 ---------
  00110101
 +11100101
 ---------
 100011010
```

carry bit

sign bit is low

00011010

- 27 – 53

```
  00011011
 -00110101
 ---------
  00011011
 +11001011
 ---------
 011100110
```

no carry bit

sign bit is high

-00011010