LOGIC OPERATIONS

Logic operations

- We have already seen keywords `or`, `and`, `not` used in Python
  - Had a specific purpose – Boolean expressions. For example:

```
if x >= 0 and x < 10:
    print("x is a single digit")
```

- Python has a set of operators for bitwise computations:
  - **&** : bitwise AND
  - | : bitwise OR
  - ~ : bitwise NOT
  - >>: shift to the right
  - <<: shift to the left

A *bit* of intuition…

```
21 << 1        5 & 6        5 | 6
 42              ?            ?

21 >> 1
 10
```

A *bit* of intuition…

Sometimes the bits are *almost* visible:

```
10101          101   110       101   110
21 << 1        5  &  6         5  |  6

10101                                000
21 >> 1                              ~0
```

A *bit* of intuition…

Sometimes the bits are *almost* visible:

```
10101          101   110       101   110
21 << 1        5  &  6         5  |  6
   101010         100             111

10101                          000
21 >> 1                        ~0
   1010                          111
```

However ~0 gives -1 ???

```
>>> x = 0
>>> ~x
-1
```

We will see a thorough discussion on this later. Here is a quick illustration of the idea.

Binary	Decimal
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Take an example of 3-bit binary number. We can represent 8 different values.

If we want to represent negative numbers, we would typically (for good reasons!) to have –(n+1) to n. In this case, -4 to 3. In this system, the binary pattern of -1 is 111, which is ~0.

Truth tables

input		output
x	y	AND(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

AND outputs 1 only if **ALL** inputs are 1

AND

input		output
x	y	OR(x,y)
0	0	0
0	1	1
1	0	1
1	1	1

OR outputs 1 if **ANY** input is 1

OR

input	output
x	NOT(x)
0	1
1	0

NOT reverses its input

NOT

All computation

… consists of functions of bits, or ***boolean values***

Boolean inputs **x,y,…** can *only* be **0** or **1** (False or True).

Boolean functions can output *only* **0** or **1** (False, True).

inputs		output
x	y	fn(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

Lots of bits!

inputs			output
x	y	z	fn(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Truth table

LOGIC GATES AND CIRCUITS

Reviewing…

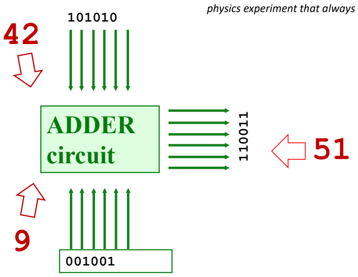
- We have:
  - explored a wide range of data types
  - learned how different encodings are used for different types
  - learned that, at the core of all data stored in the computer are bits
  - observed different operations that can be performed on these bits (AND, OR, NOT)
- We have one **BIG QUESTION** remaining…

  HOW IS COMPUTATION ACTUALLY CARRIED OUT?

In a computer, each bit is represented as a voltage
(**1** is +5v and **0** is 0v)

Computation is simply the deliberate combination of those voltages!

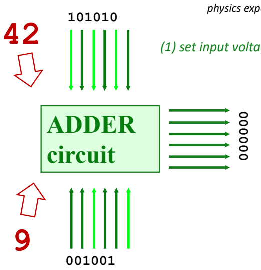
Feynman: *Computation is just a physics experiment that always works!*



In a computer, each bit is represented as a voltage (**1** is +5v and **0** is 0v)

Computation is simply the deliberate combination of those voltages!

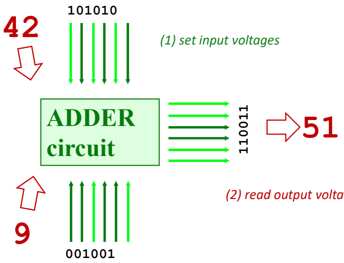
Feynman: *Computation is just a physics experiment that always works!*



(1) set input voltages

In a computer, each bit is represented as a voltage (**1** is +5v and **0** is 0v)

Computation is simply the deliberate combination of those voltages!

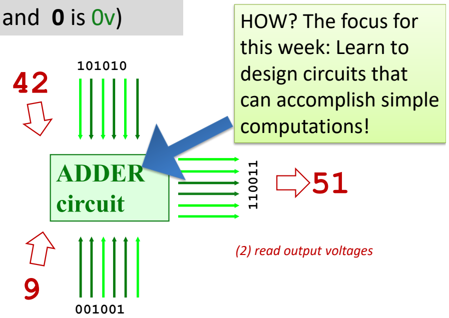


(1) set input voltages

(2) read output voltages

In a computer, each bit is represented as a voltage (**1** is +5v and **0** is 0v)

Computation is simply the deliberate combination of those voltages!



HOW? The focus for this week: Learn to design circuits that can accomplish simple computations!

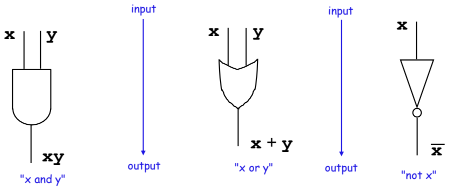
(2) read output voltages

We need only three building circuits to compute anything at all

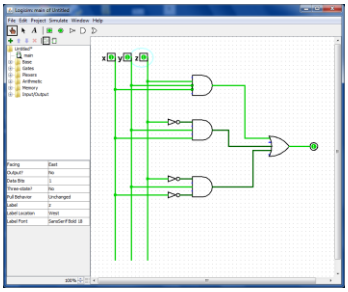
input		output
x	y	AND(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

input		output
x	y	OR(x,y)
0	0	0
0	1	1
1	0	1
1	1	1

input	output
x	NOT(x)
0	1
1	0



Circuits from logic gates... ?



What are all of the inputs that make this circuit output 1? Note the three input gates, both OR and AND gates.

Logisim

- HW 5 – Use **Logisim** (a free circuit simulation package) to design circuits to perform simple computations
- Hard? Well, let's recall our claim – we only need AND, OR and NOT to compute anything at all…

Constructive Proof !

i Specify a **truth table** defining ***any*** function you want

input		output
x	y	fn(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

ii For each input row whose output needs to be 1, build an **AND** circuit that outputs 1 *only for that specific input*!

iii **OR** them all together

Formula !

$\overline{x}y + x\overline{y}$

minterm

Minterm Expansion Principle – algorithm for building expressions from truth tables

Minterm expansion

readily converts into logic gates…

Constructive Proof !

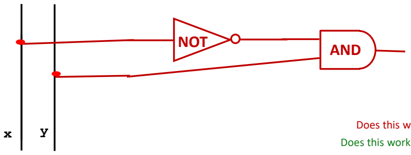
i Specify a **truth table** defining ***any*** function you want

input		output
x	y	fn(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

ii For each input row whose output needs to be 1, build an **AND** circuit that outputs 1 *only for that specific input*!

iii **OR** them all together

$\overline{x}y + x\overline{y}$



Does this work for the red row?

Does this work for any other rows?

Constructive Proof !

i Specify a **truth table** defining ***any*** function you want

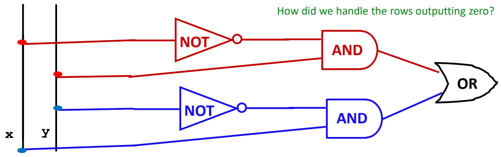
input		output
x	y	fn(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

ii For each input row whose output needs to be 1, build an **AND** circuit that outputs 1 *only for that specific input*!

iii **OR** them all together

$\overline{x}y + x\overline{y}$

How did we handle the rows outputting zero?



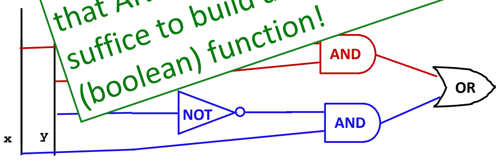
Constructive Proof !

i Specify a **truth table** defining ***any*** function you want

input		output
x	y	fn(x,y)
0	0	
0	1	
1		
1		

ii For each input row whose output needs to be 1, build an AN… 1, build … outputs 1 … *input*!

…ng zero?



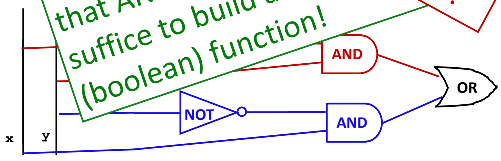
This is a constructive proof that AND, OR, and NOT suffice to build any (boolean) function!

Constructive Proof !

i Specify a **truth tabl**… ***any*** function you wan…

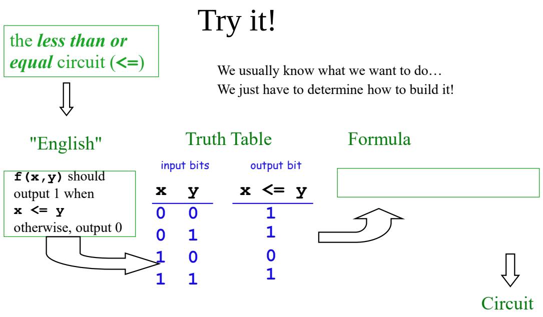
input		output
x	y	fn(x,y)
0	0	
0	1	
1		
1		

…or each input row whose … 1, build … outputs 1 … *input*!



This is a construc… that AND, OR, and N… suffice to build any (boolean) function!

But… **ALL** boolean functions are just binary!



Try it!

the *less than or equal* circuit (<=)

We usually know what we want to do…
We just have to determine how to build it!

"English"

f(x,y) should output 1 when **x <= y** otherwise, output 0

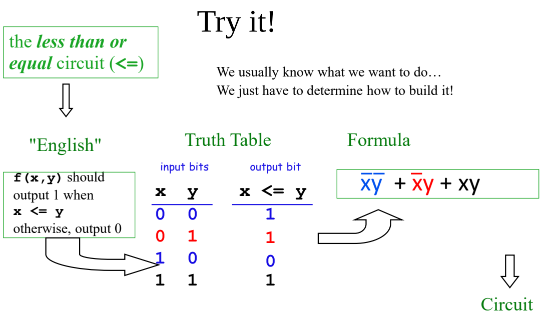
Truth Table

input bits		output bit
x	**y**	**x <= y**
0	0	1
0	1	1
1	0	0
1	1	1

Formula

Circuit

EXAMPLE



Try it!

the *less than or equal* circuit (<=)

We usually know what we want to do…
We just have to determine how to build it!

"English"

f(x,y) should output 1 when **x <= y** otherwise, output 0

Truth Table

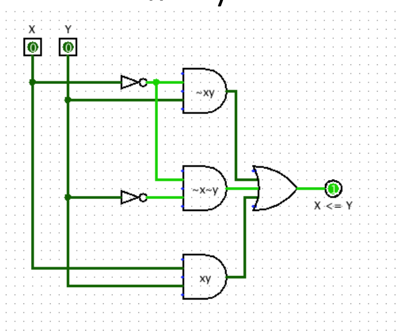
input bits		output bit
x	**y**	**x <= y**
0	0	1
0	1	1
1	0	0
1	1	1

Formula

$\overline{x}\overline{y} + \overline{x}y + xy$

Circuit

x <= y


X
Y
0
0
~xy
~x~y
xy
1
X <= Y