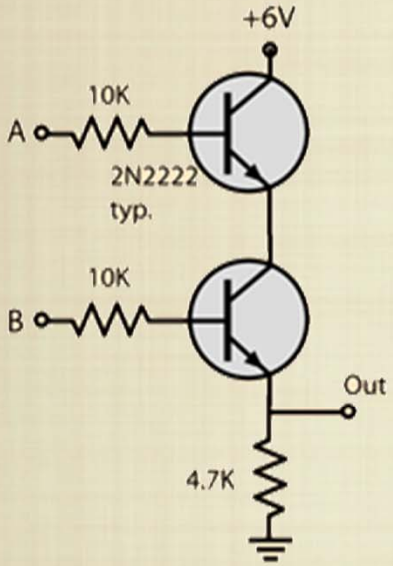
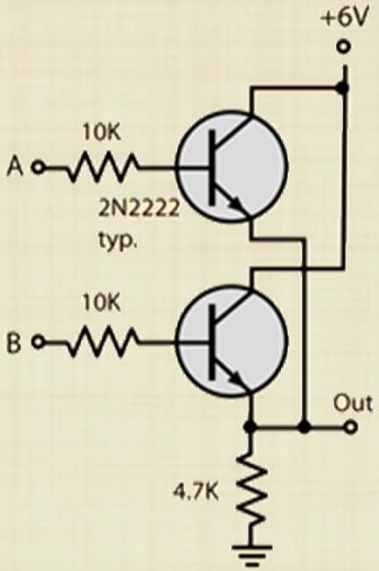
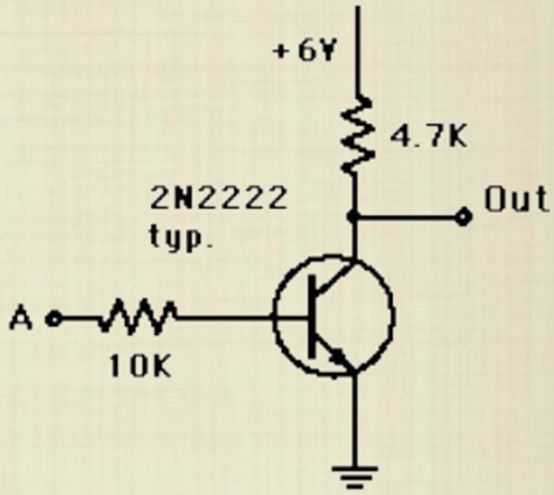
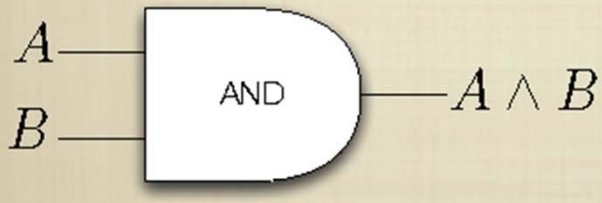
Digital Logic and Transistors

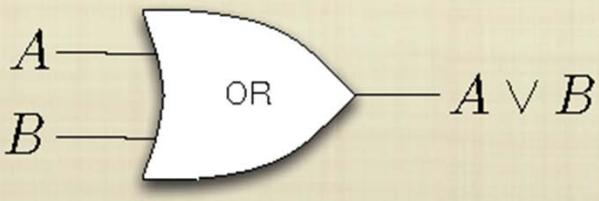
The invention of the transistor made binary logic the cheapest and most effective way to implement logic gates.

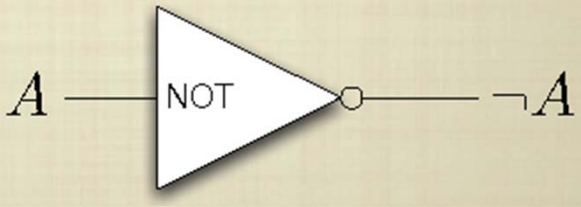






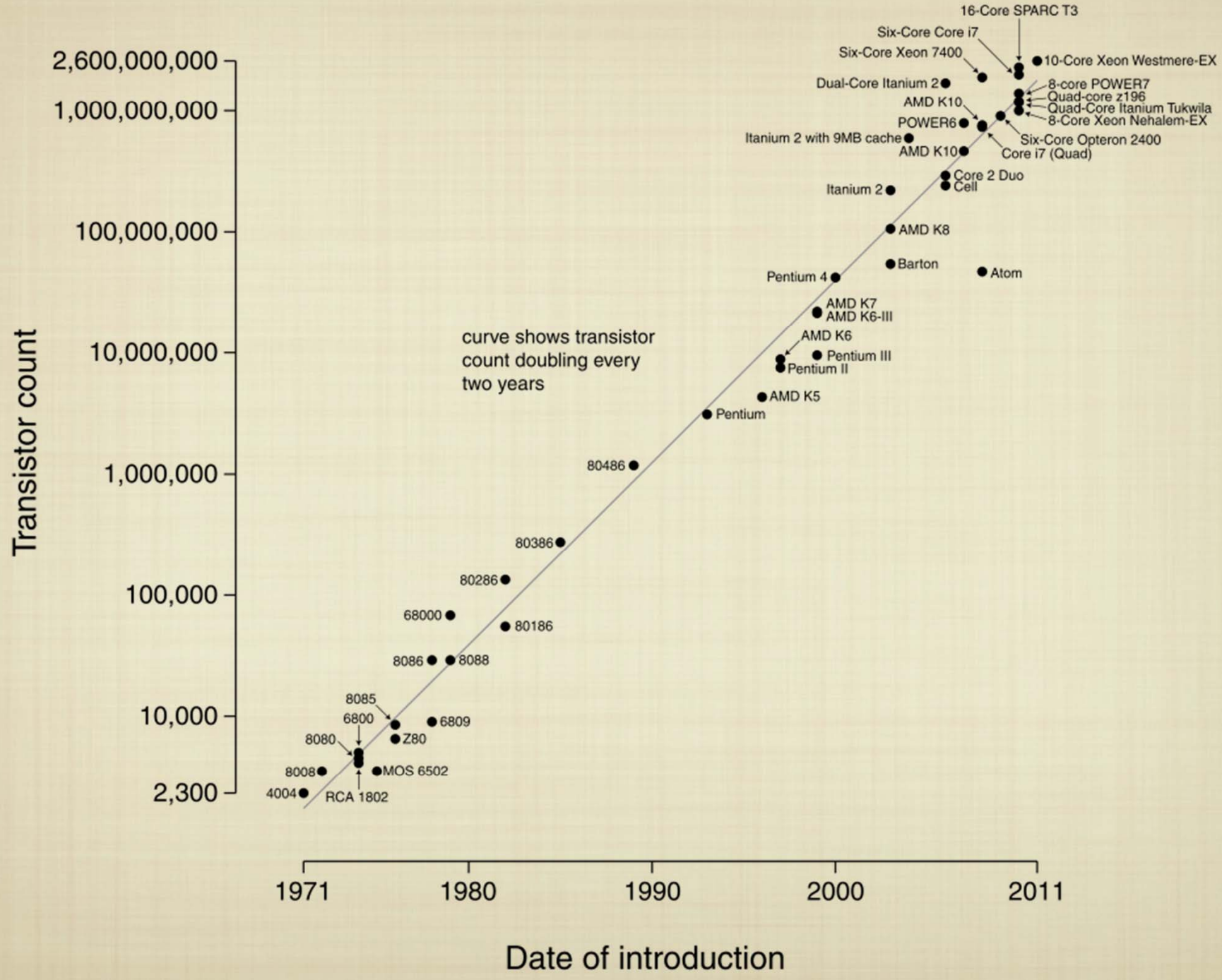






Both inputs are “on”. At least one input is “on”. The input is “off”.

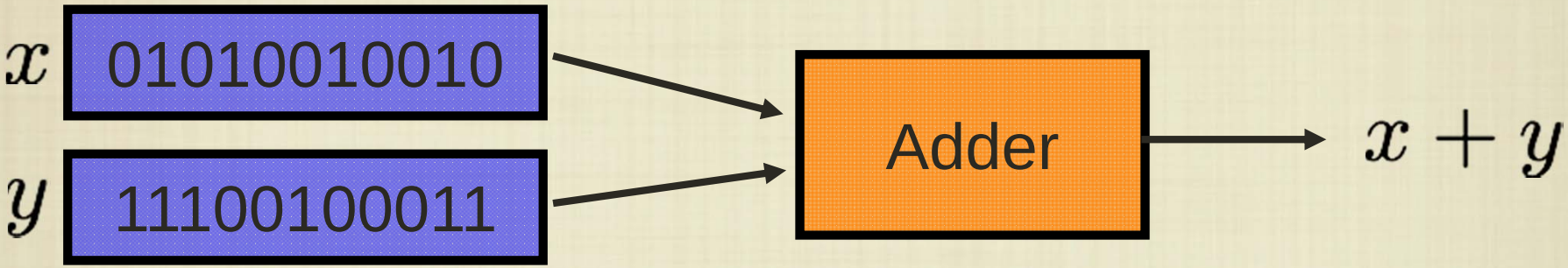
Microprocessor Transistor Counts 1971-2011 & Moore's Law



What is an “Instruction”?

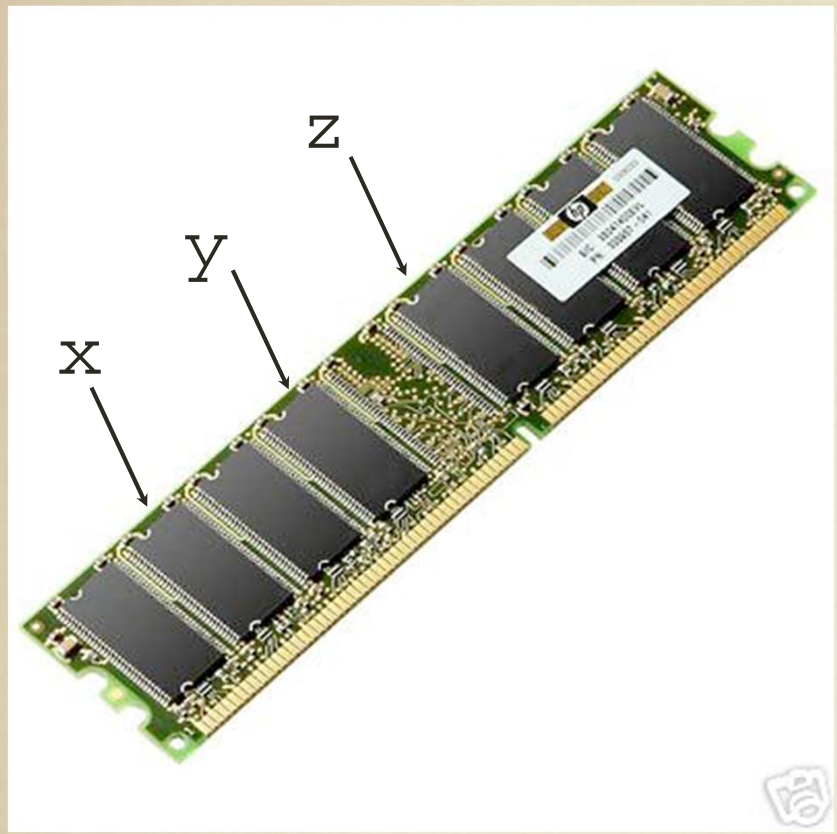
Let’s consider the simple operation of adding two numbers.

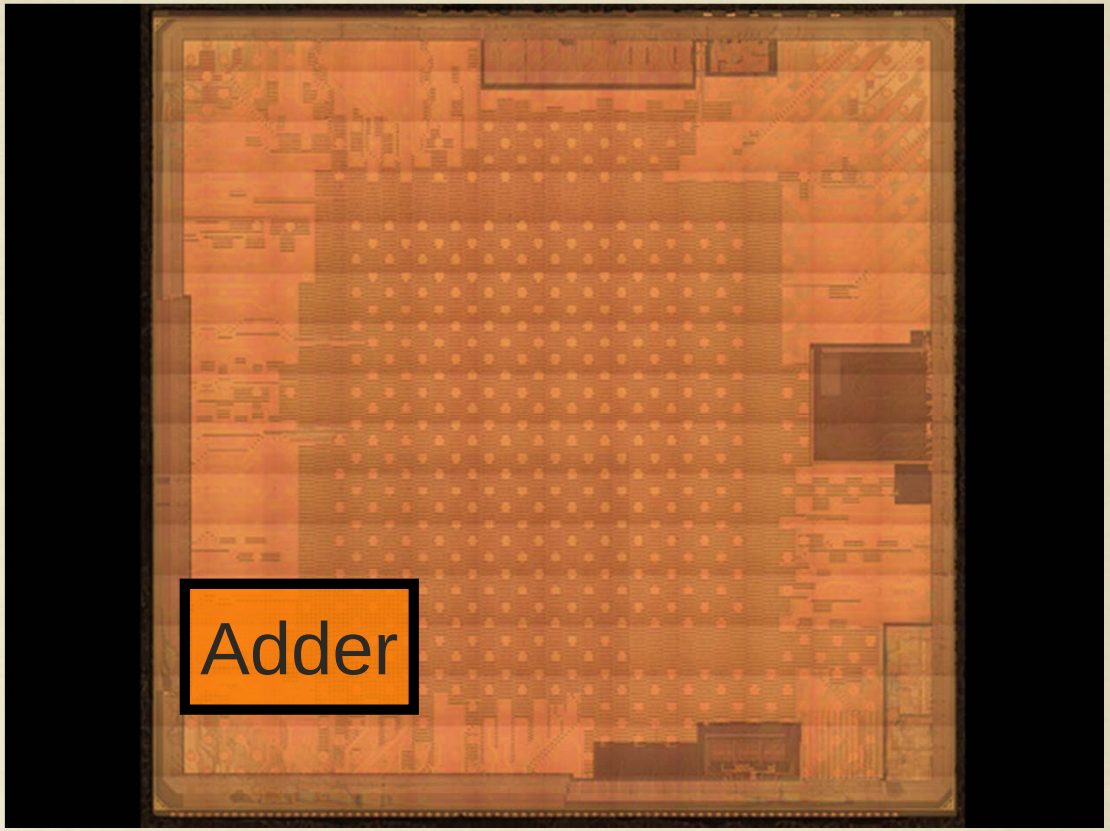
```
add x, y, z
```



Why are numbers represented in binary? How would we add binary numbers?

Adding on a CPU



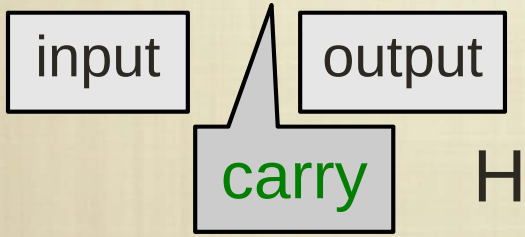


Every instruction in a CPU is composed of logic gates. With current technology, gates are about 200 nm - roughly 300 times smaller than the diameter of a human hair.

Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:

1-bit numbers	2-bit numbers
0 + 0 = 00	00 + 00 = 00
0 + 1 = 01	00 + 01 = 01
1 + 0 = 01	00 + 11 = 11
1 + 1 = 10	01 + 00 = 01
	...



How many entries?

How many possible sums can two *n*-bit numbers have?

Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:

<u>1-bit numbers</u>

0 + 0 = 0 0

0 + 1 = 0 1

1 + 0 = 0 1

1 + 1 = 1 0

input

carry

output

x	y
0	0
0	1
1	0
1	1

input

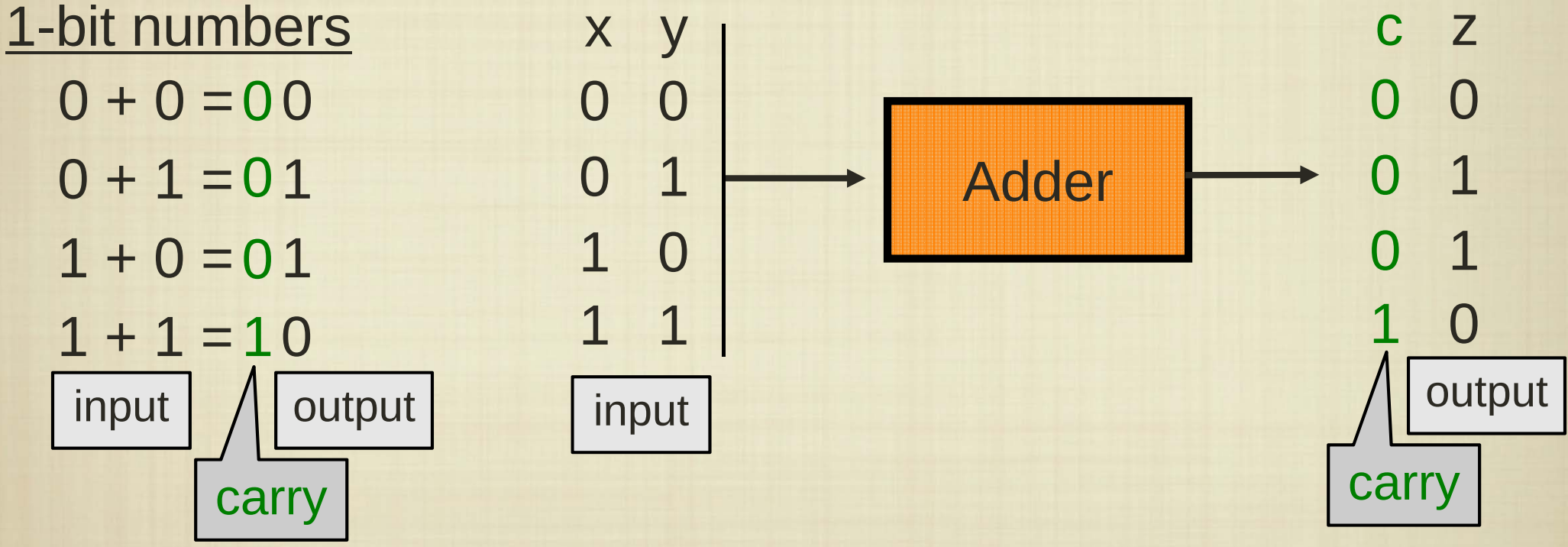
c	z
0	0
0	1
0	1
1	0

carry

output

Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:



Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:

1-bit numbers

0 + 0 = 0 0

0 + 1 = 0 1

1 + 0 = 0 1

1 + 1 = 1 0

input

output

carry

x	y	c	z
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

input

output

carry

c = ?

z = ?

Can we calculate z and c using logic gates?

Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:

1-bit numbers

0 + 0 = 0 0

0 + 1 = 0 1

1 + 0 = 0 1

1 + 1 = 1 0

input

output

carry

x	y	c	z
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

input

output

carry

$c = x \wedge y$

$z = ?$

Can we calculate z and c using logic gates?

Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:

1-bit numbers

0 + 0 = 0 0

0 + 1 = 0 1

1 + 0 = 0 1

1 + 1 = 1 0

input output carry

x	y	c	z
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

input output carry

$c = x \wedge y$

$z = x \oplus y$

$= (x \vee y) \wedge \neg(x \wedge y)$

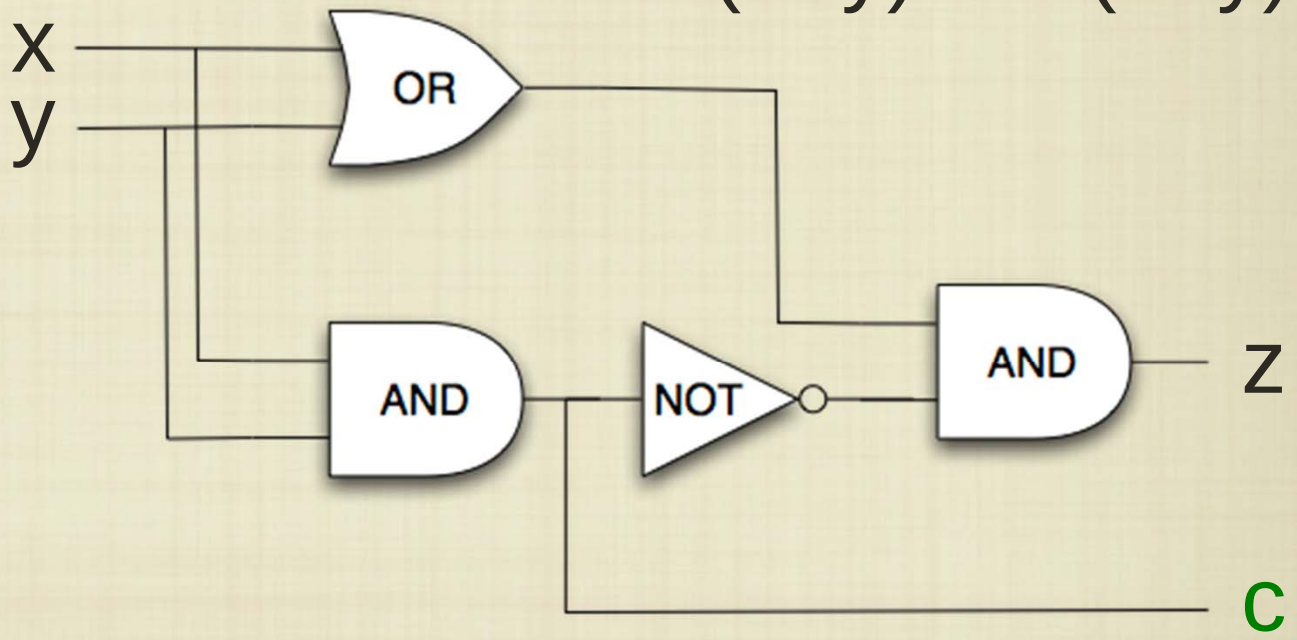
Can we calculate z and c using logic gates?

Circuits and Boolean Logic

What does an adding circuit look like? Let's consider adding in binary:

$c = x \wedge y$

$z = (x \vee y) \wedge \neg(x \wedge y)$



1-bit numbers

0 + 0 = 0 0

0 + 1 = 0 1

1 + 0 = 0 1

1 + 1 = 1 0

input | output | carry

A one-bit adder needs 4 gates. How do we add numbers with more bits?

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c              0
x  01010010010
y  11100100011
-----------------
z
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c              00
x  01010010010
y  11100100011
-----------------
z              1
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c            100
x  01010010010
y  11100100011
-----------------
z             01
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c            0100
x  01010010010
y  11100100011
-------------------
z            101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c          00100
x  01010010010
y  11100100011
-----------------
z          0101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c        000100
x  01010010010
y  11100100011
-----------------
z          10101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c       0000100
x  01010010010
y  11100100011
-----------------
z        110101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c     00000100
x  01010010010
y  11100100011
-----------------
z      0110101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c    000000100
x  01010010010
y  11100100011
-----------------
z     10110101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c   0000000100
x  01010010010
y  11100100011
-------------
z    110110101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

```
c 10000000100
x 01010010010
y 11100100011
-------------
z  0110110101
```

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

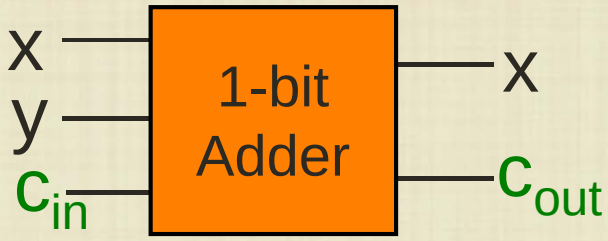
```
c  10000000100
x  01010010010
y  11100100011
-------------
z 100110110101
```

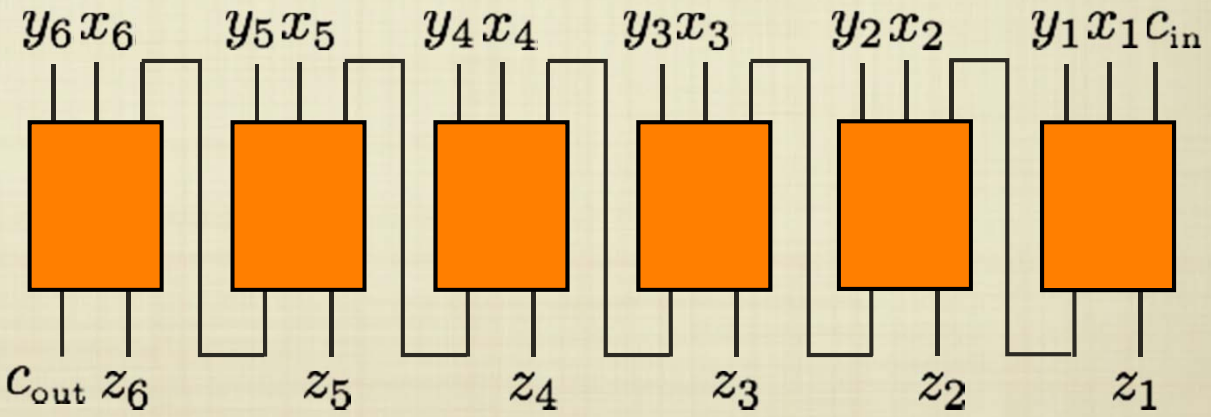
x	y	c_{in}	c_{out}	z
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Adder Logic

We need to modify our 1-bit adder slightly to use it in series:

x	y	c_{in}	c_{out}	z
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1





Note that computation time corresponds to circuit “depth”.