Smith College
Computer Science

CSC231—Assembly

Week #7 — Fall 2017

Dominique Thiébaut
dthiebaut@smith.edu

Exercise

```
x       dd     0
most    dd     0
least   dd     0

        call   _getInput    ; eax <- user input
        mov    dword[x], eax

; set most to contain all 0s except most
; significant bit of x.  Set least to
; contain all 1s except least significant
; nybble of x
```

Logic Design

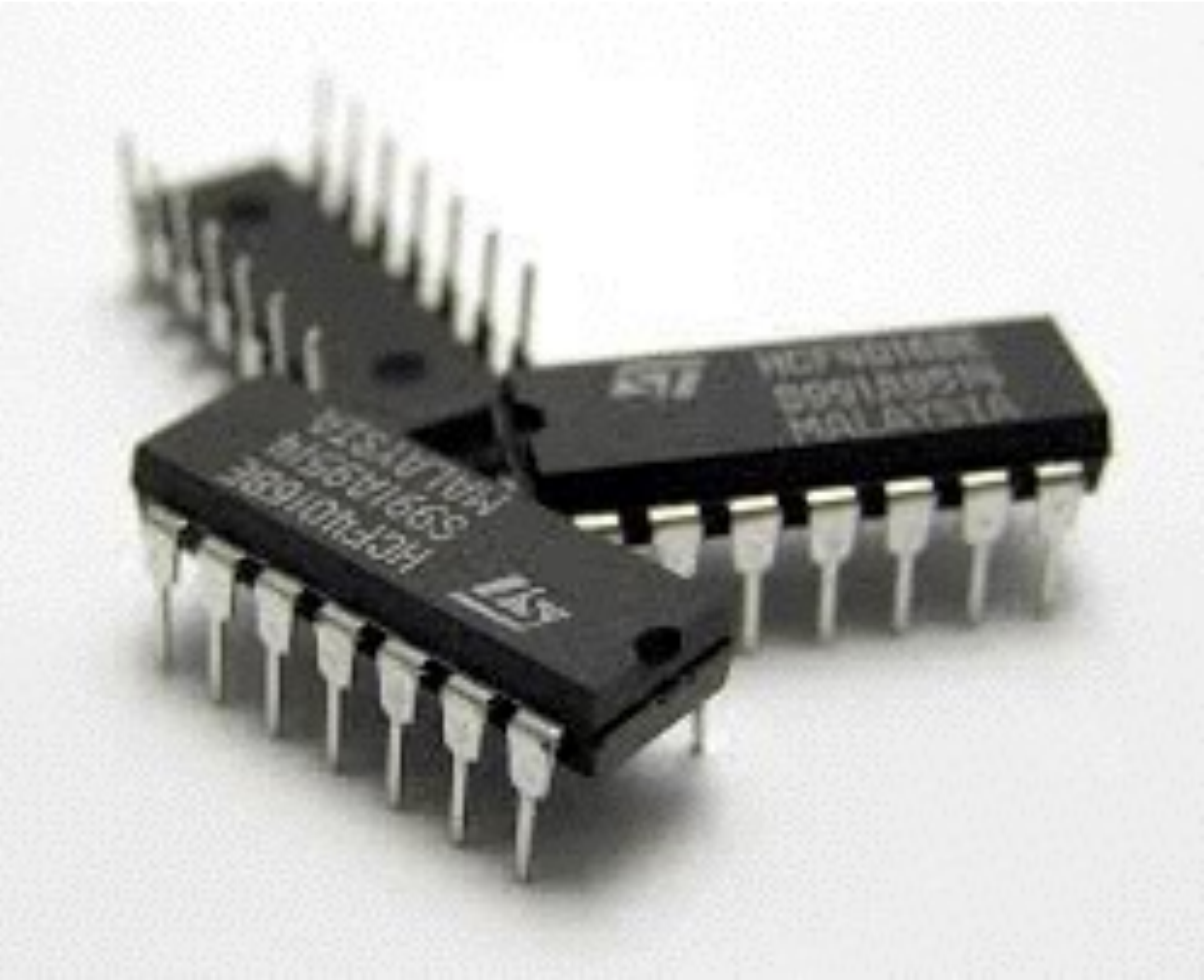
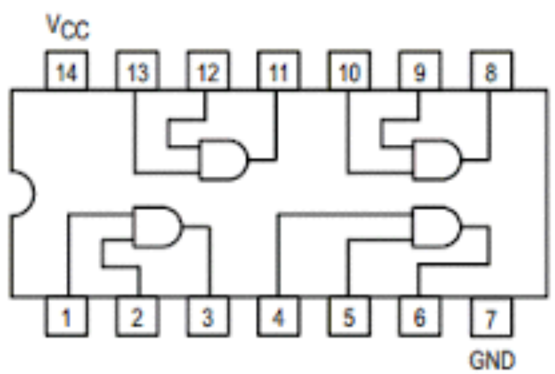
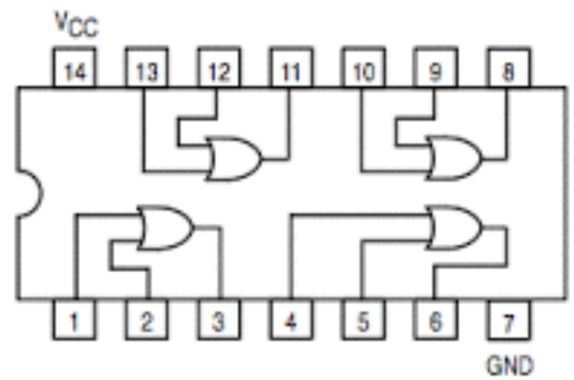


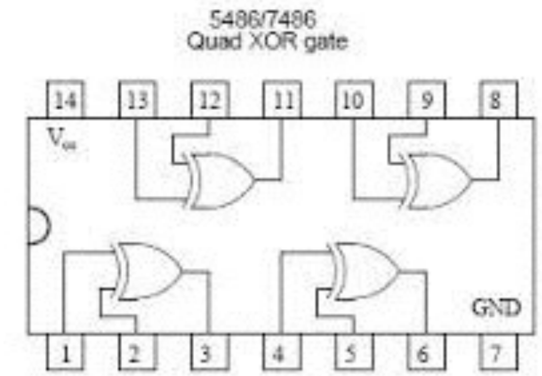
Image from http://www.willegal.net/appleii/images/motherboard.jpg

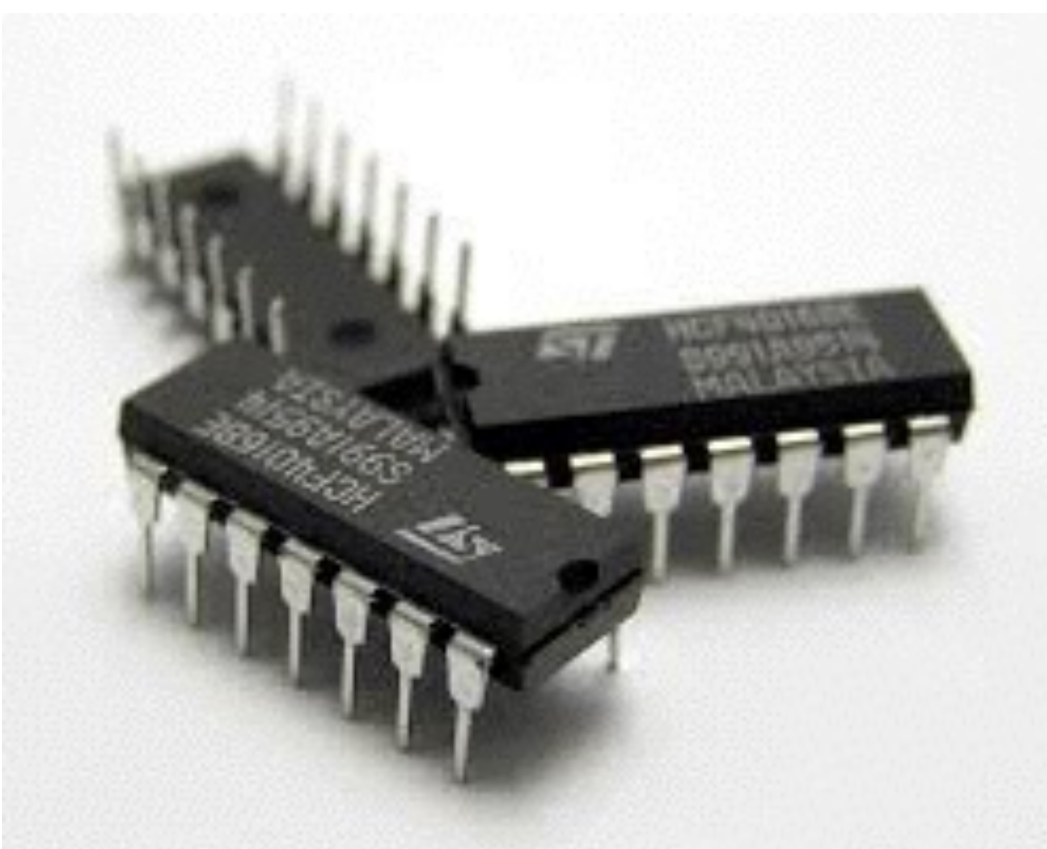
Apple II Motherboard

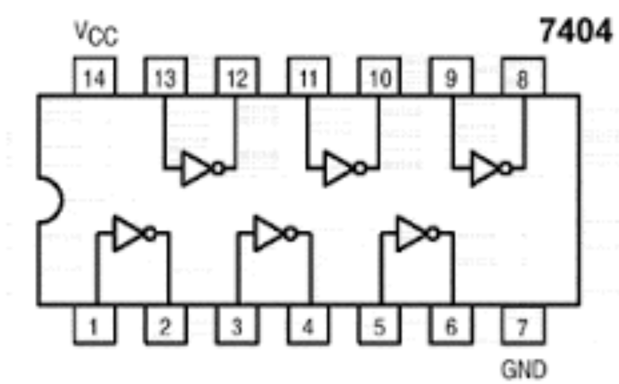
Logic Design

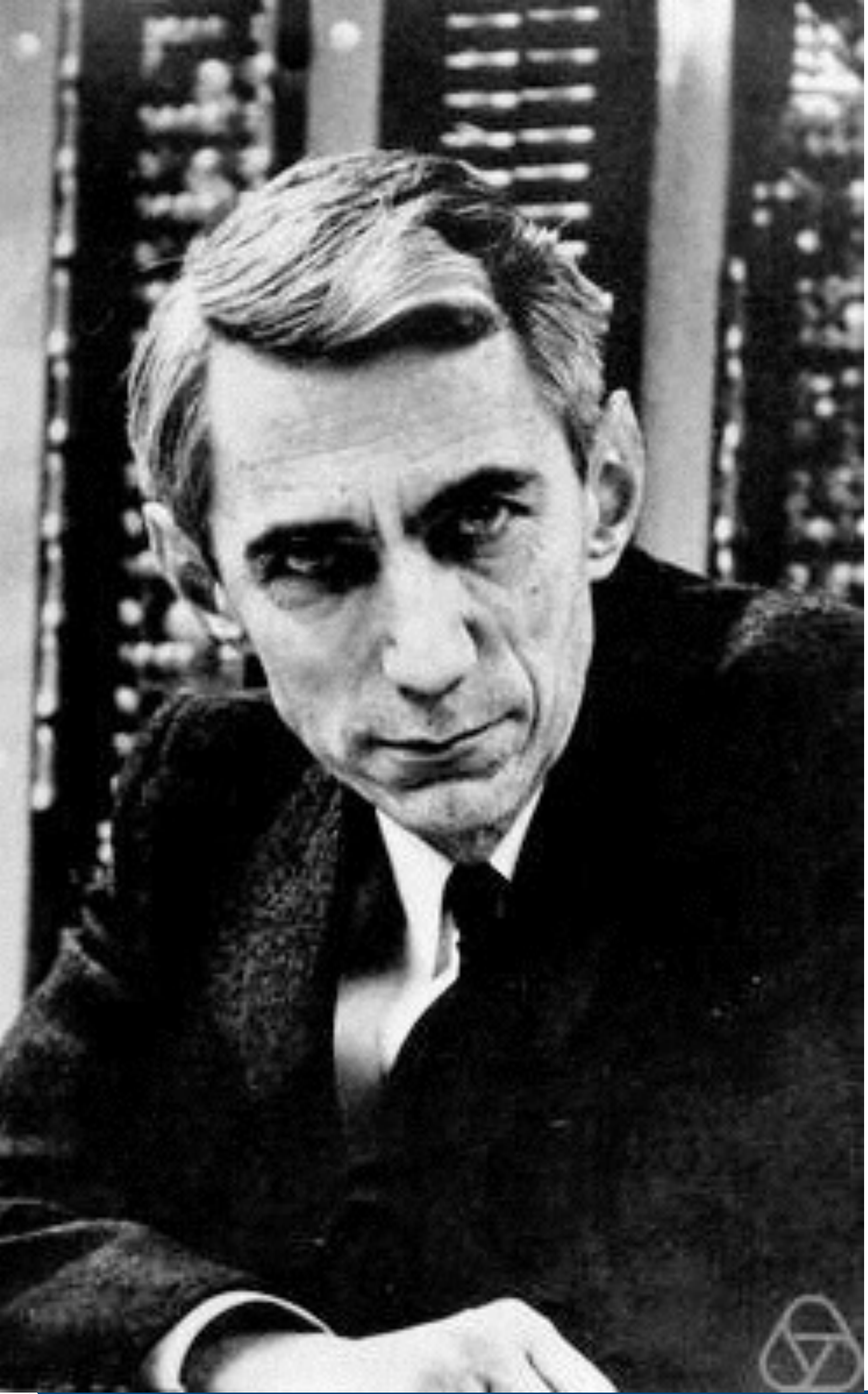










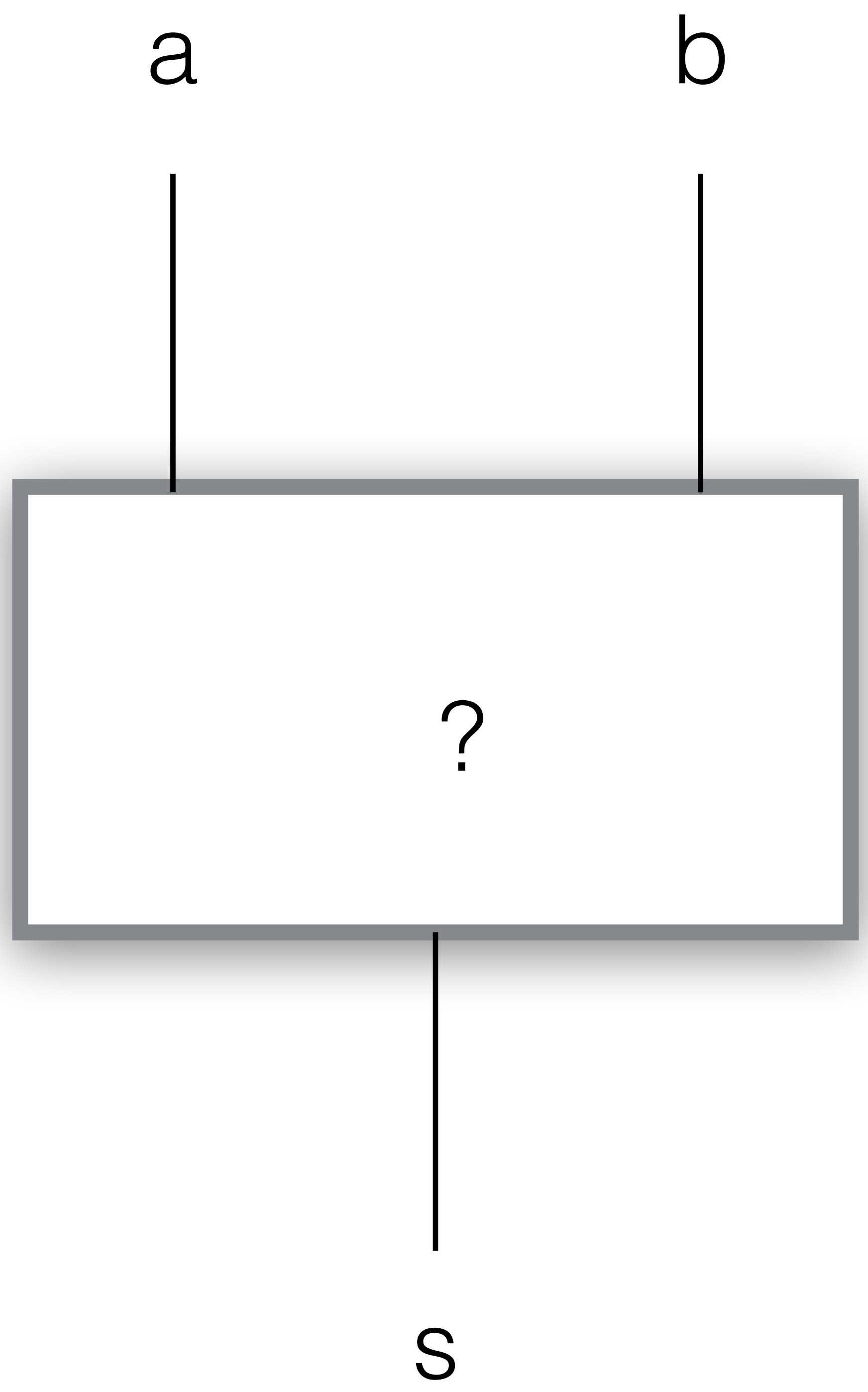


A Bit of History

- Claude Shannon
- 21 years old
- 1937
- MIT Master's Thesis
- All arithmetic operations on binary numbers can be performed using boolean logic operators

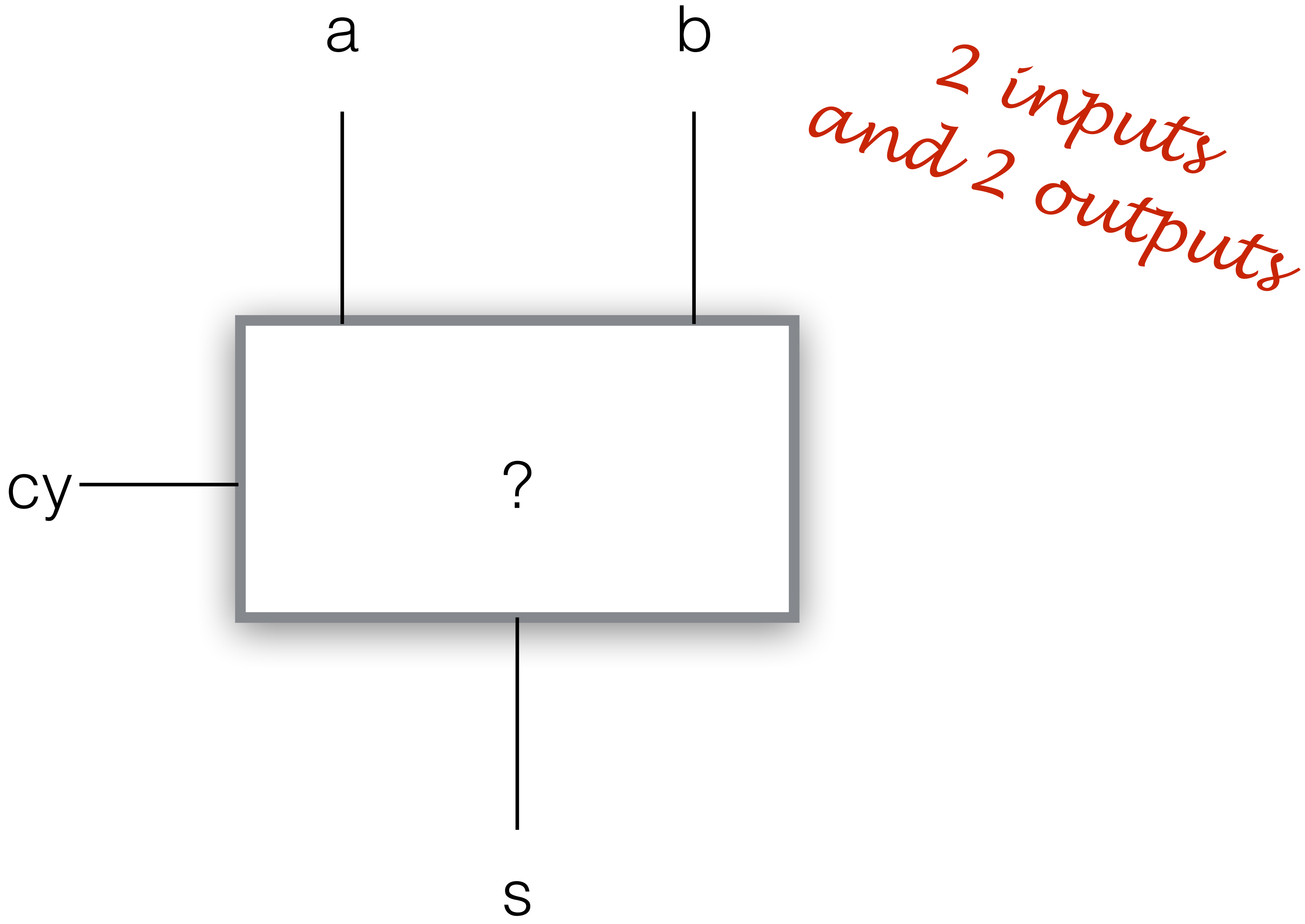
https://en.wikipedia.org/wiki/Claude_Shannon

Designing a 2-bit Adder with Logic Gates

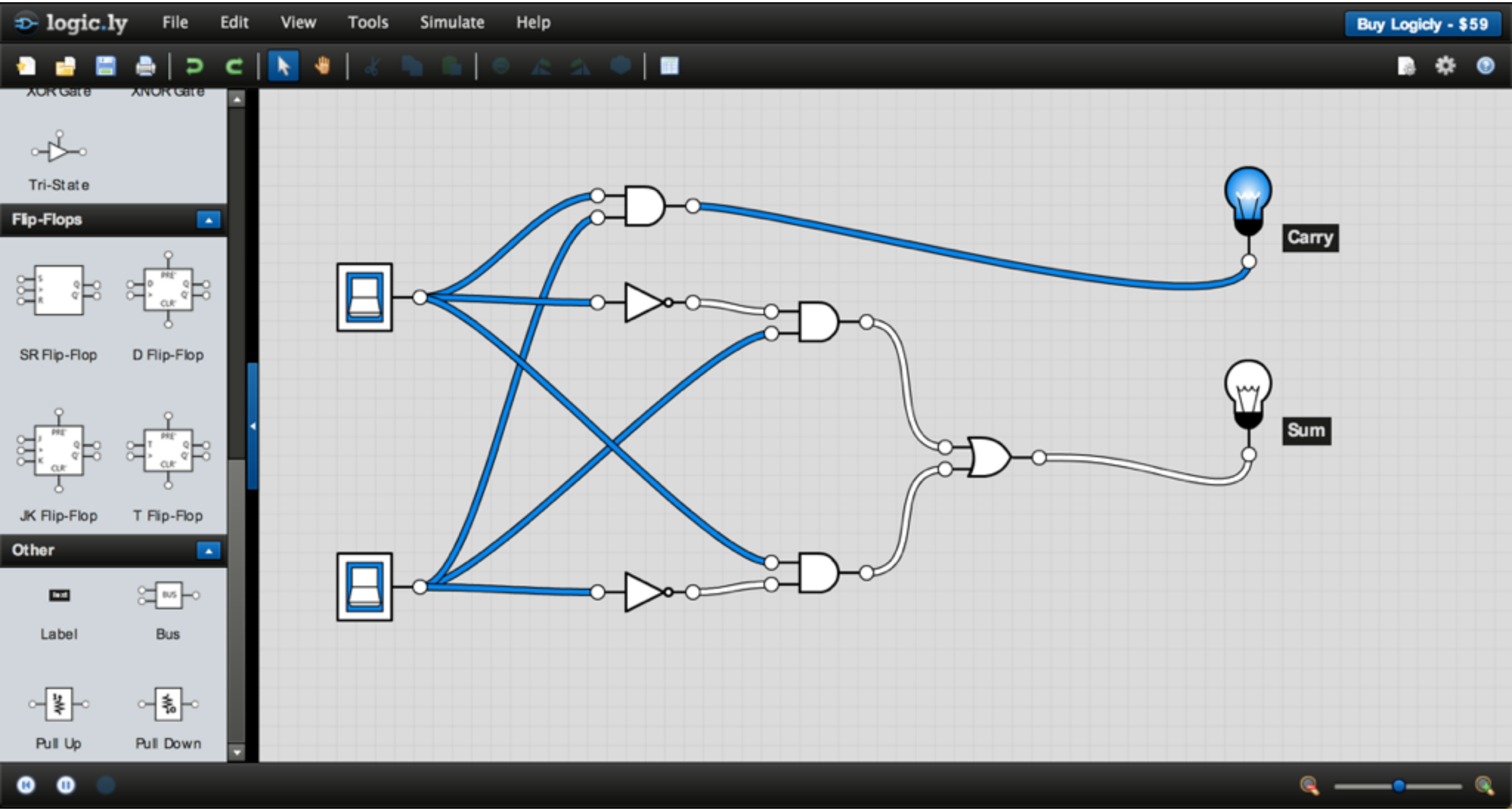

a
b
?
s

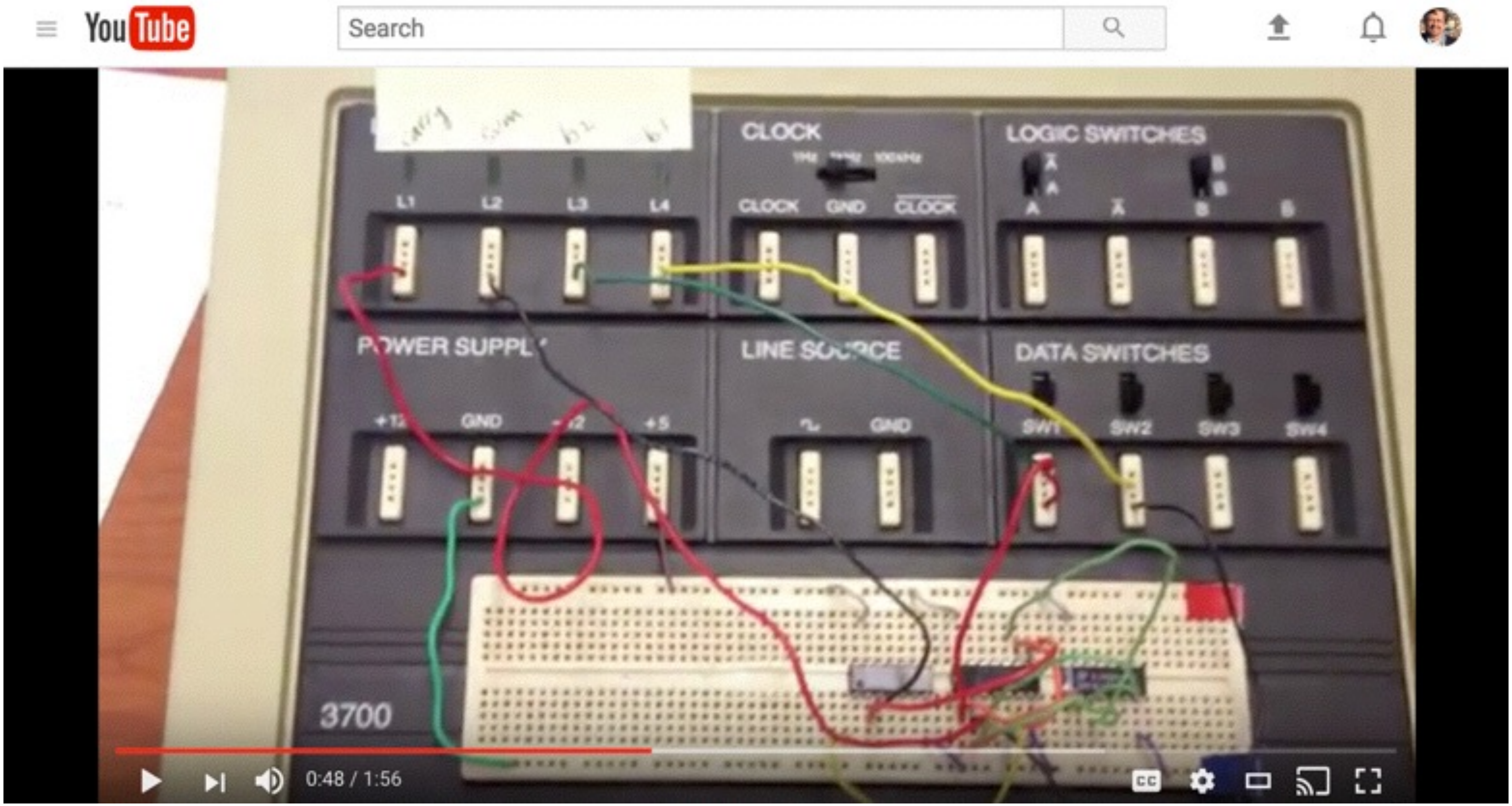
0 + 0	0 + 1	1 + 0	1 + 1
= .	= .	= .	= .

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline =\ . \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline =\ . \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline =\ . \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline =\ . \end{array}$$


a
b
cy
?
s
2 inputs
and 2 outputs

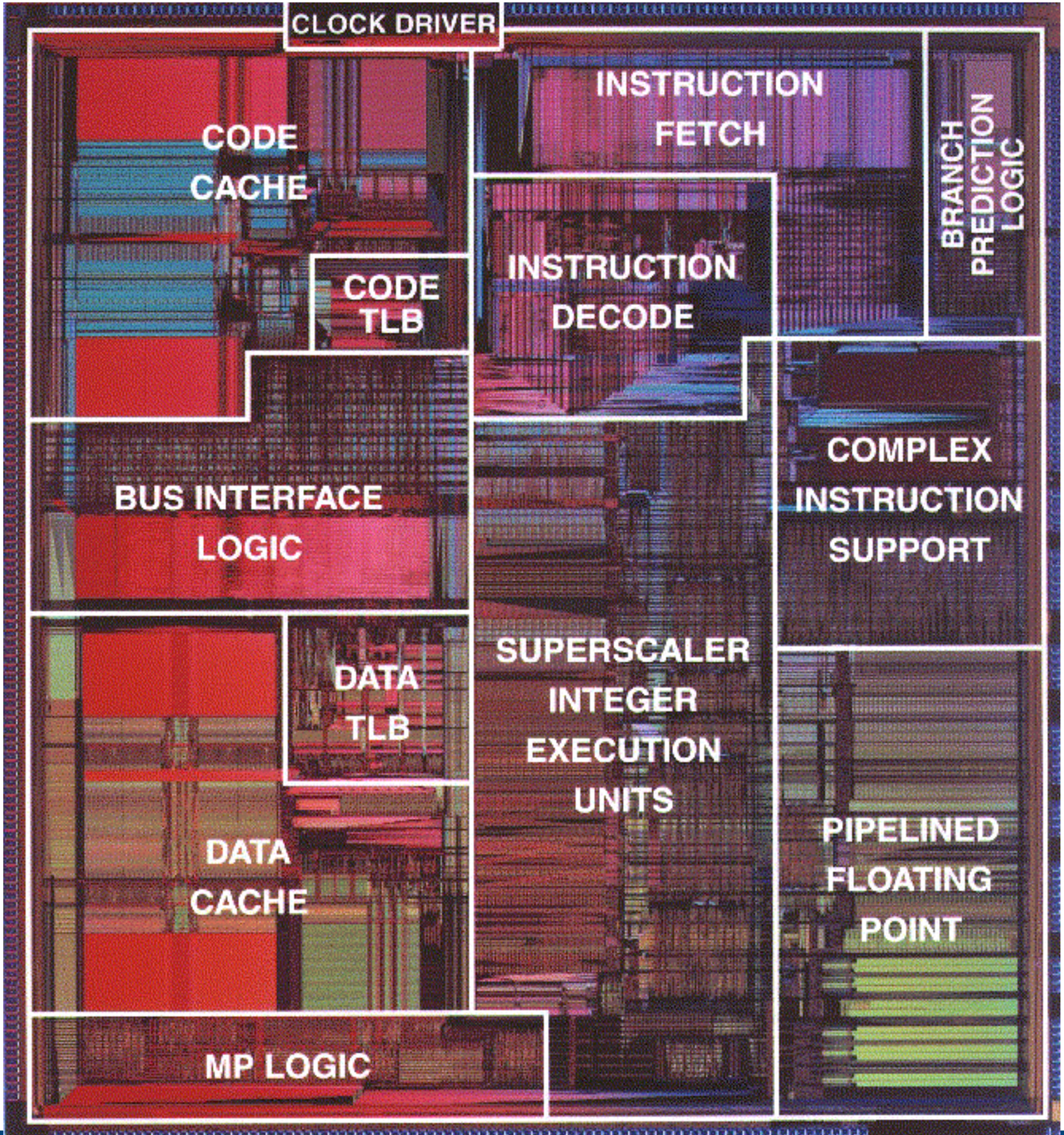
https://logic.ly/demo/



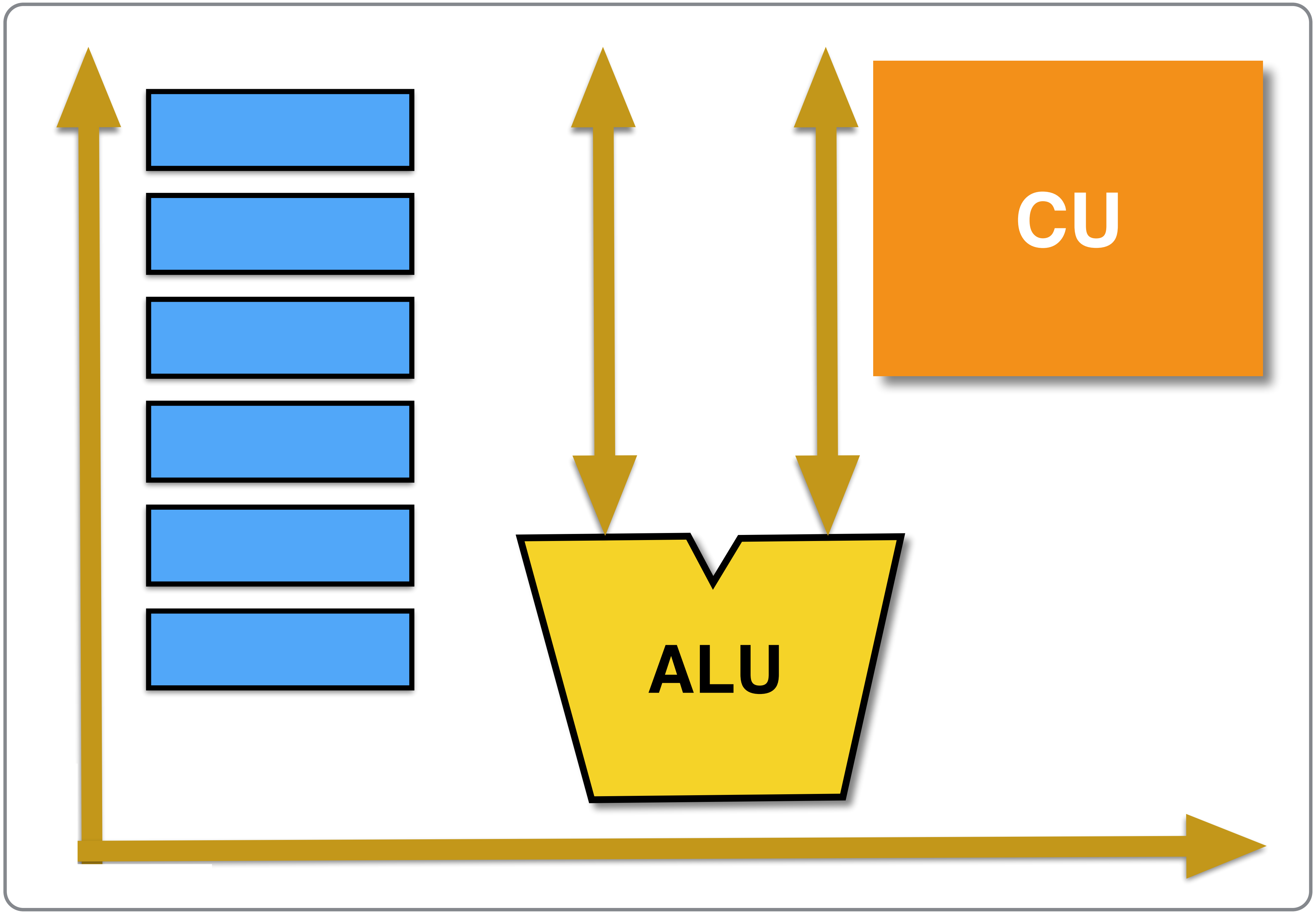


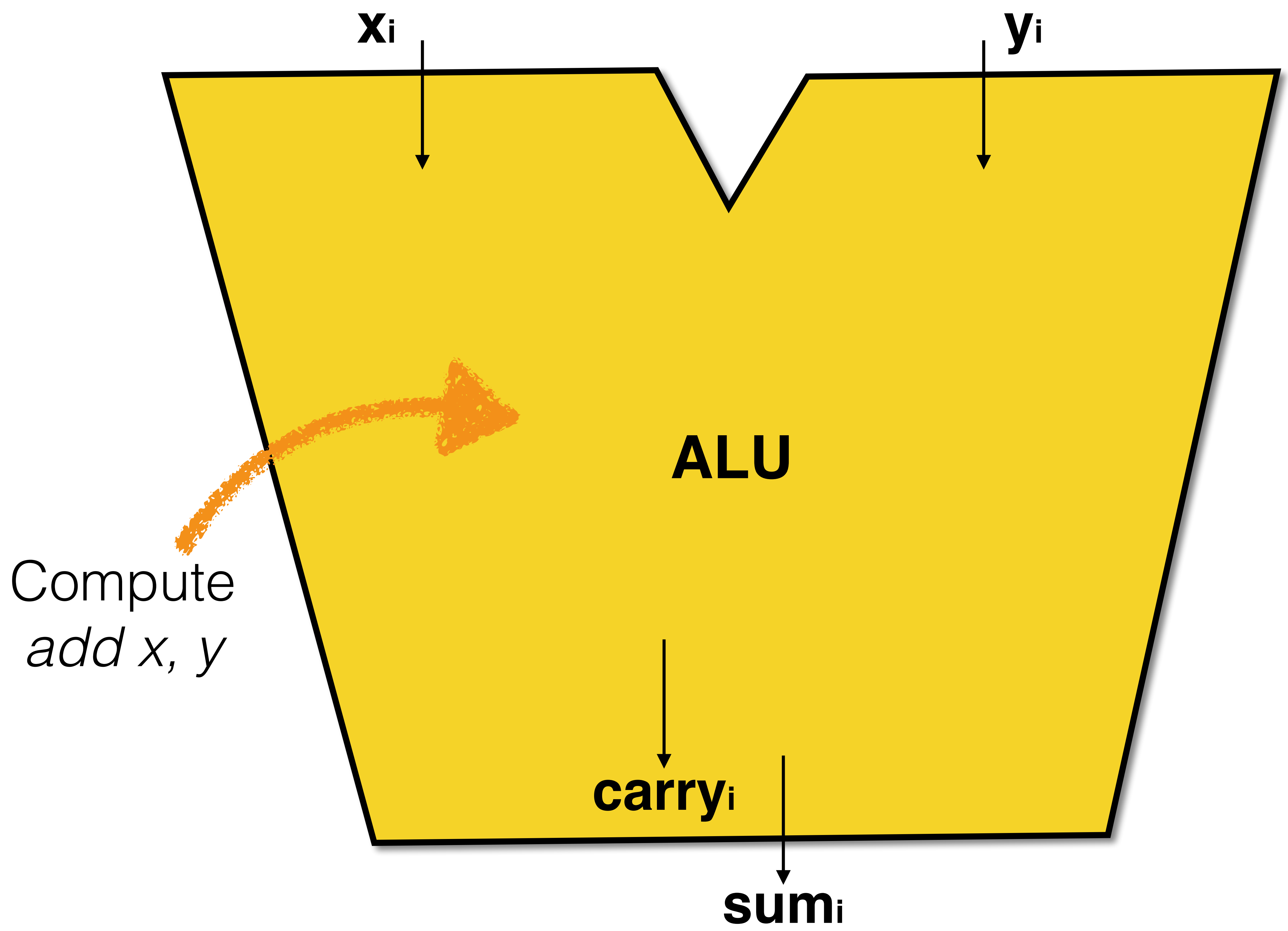
https://www.youtube.com/watch?v=xTQDIiSWK_k

NEGATIVE NUMBERS

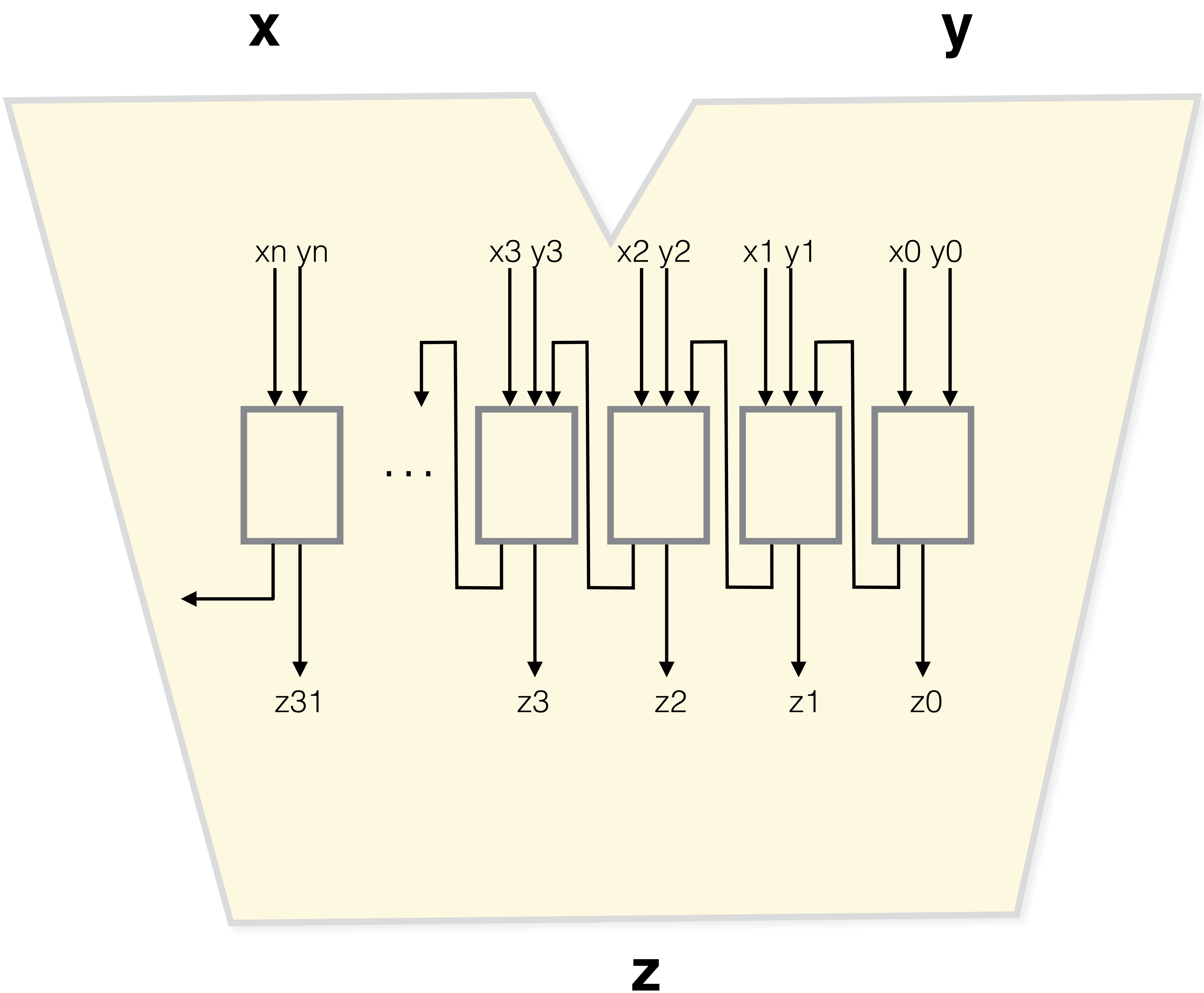

CLOCK DRIVER
CODE CACHE
INSTRUCTION FETCH
BRANCH PREDICTION LOGIC
CODE TLB
INSTRUCTION DECODE
COMPLEX INSTRUCTION SUPPORT
BUS INTERFACE LOGIC
DATA TLB
SUPERSCALER INTEGER EXECUTION UNITS
DATA CACHE
PIPELINED FLOATING POINT
MP LOGIC

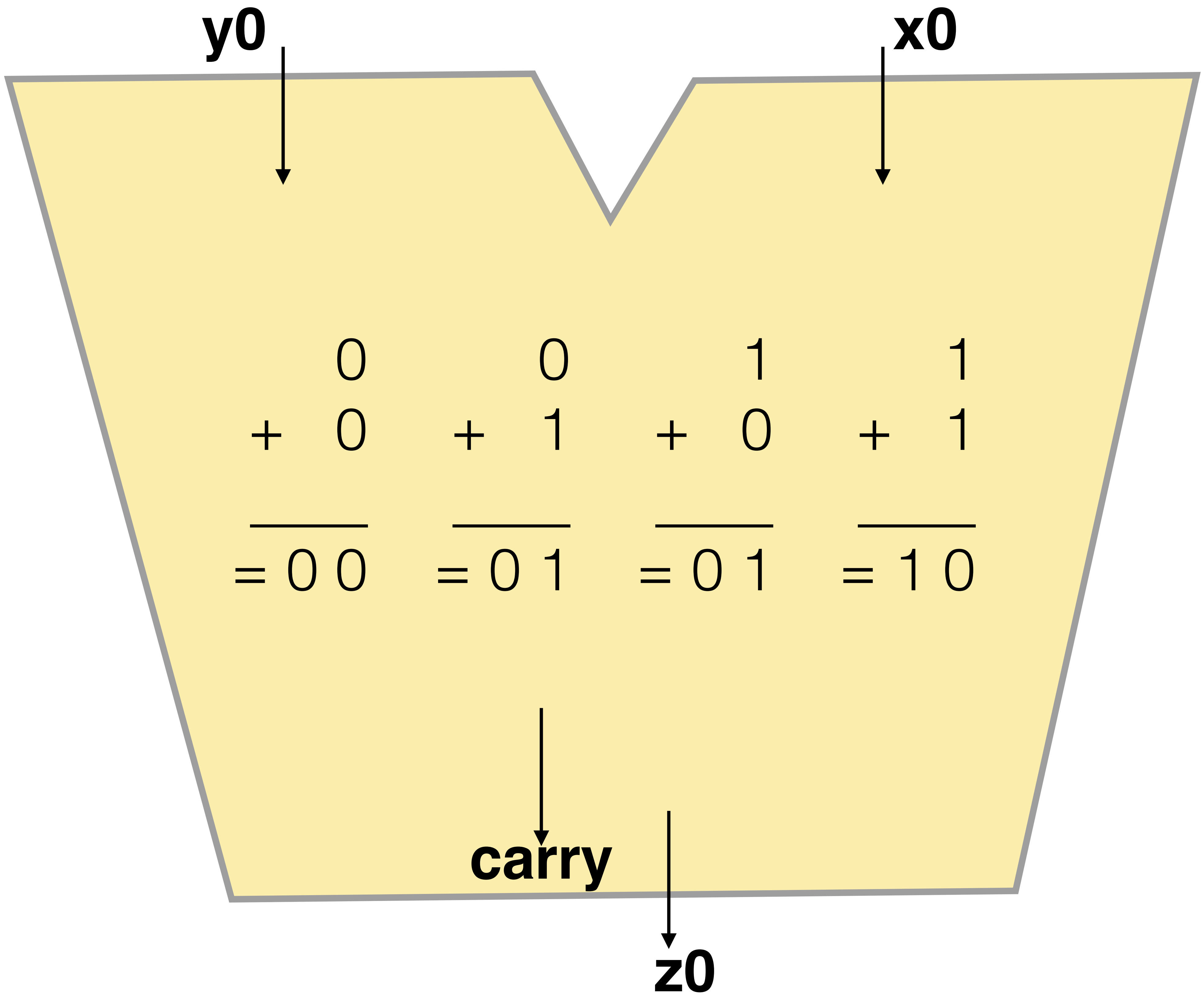
Processor





x_i
y_i
ALU
Compute
add x, y
$carry_i$
sum_i


x
y
xn yn
x3 y3
x2 y2
x1 y1
x0 y0
. . .
z31
z3
z2
z1
z0
z

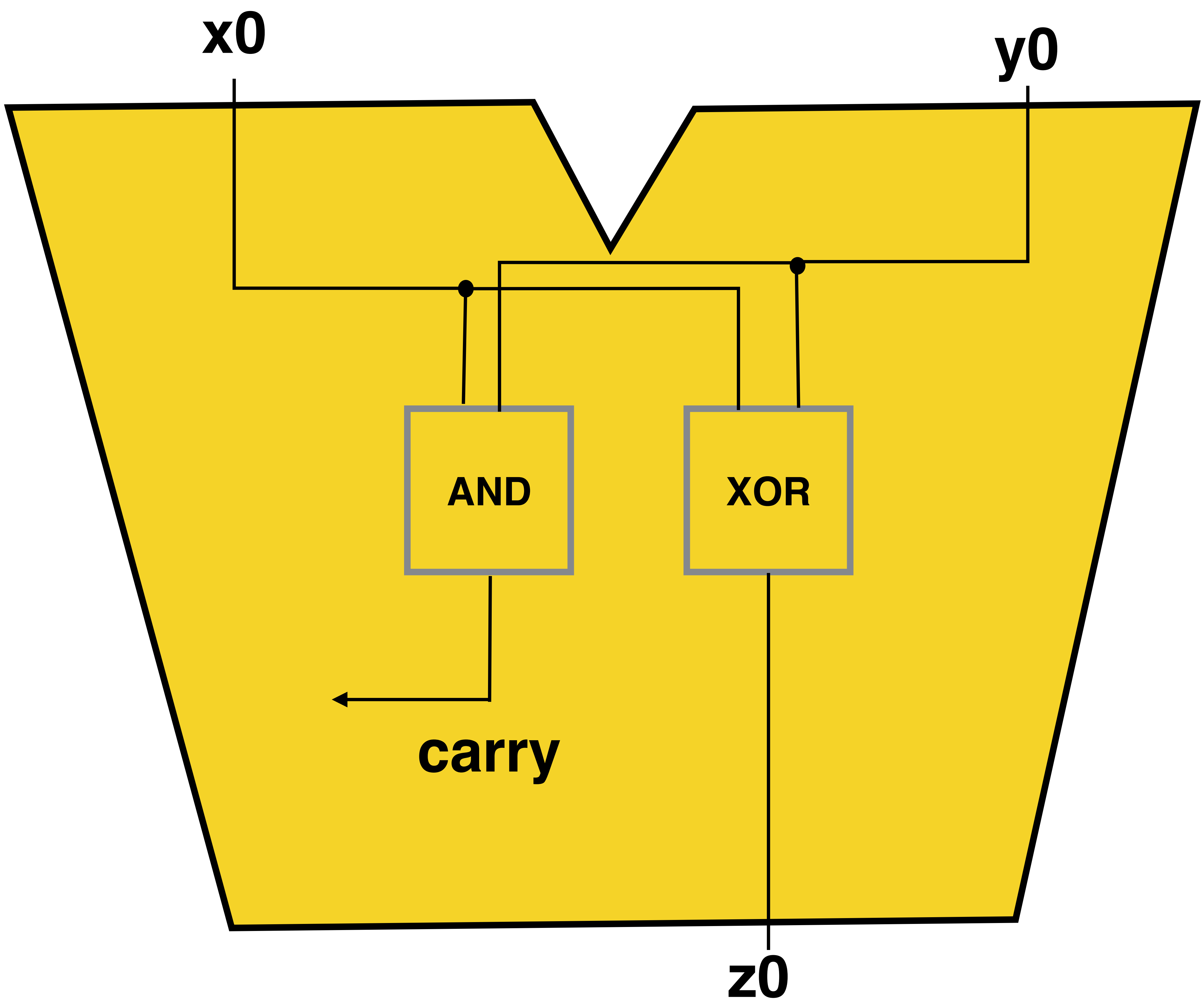

y0
x0
0 + 0 = 0 0
0 + 1 = 0 1
1 + 0 = 0 1
1 + 1 = 1 0
carry
z0

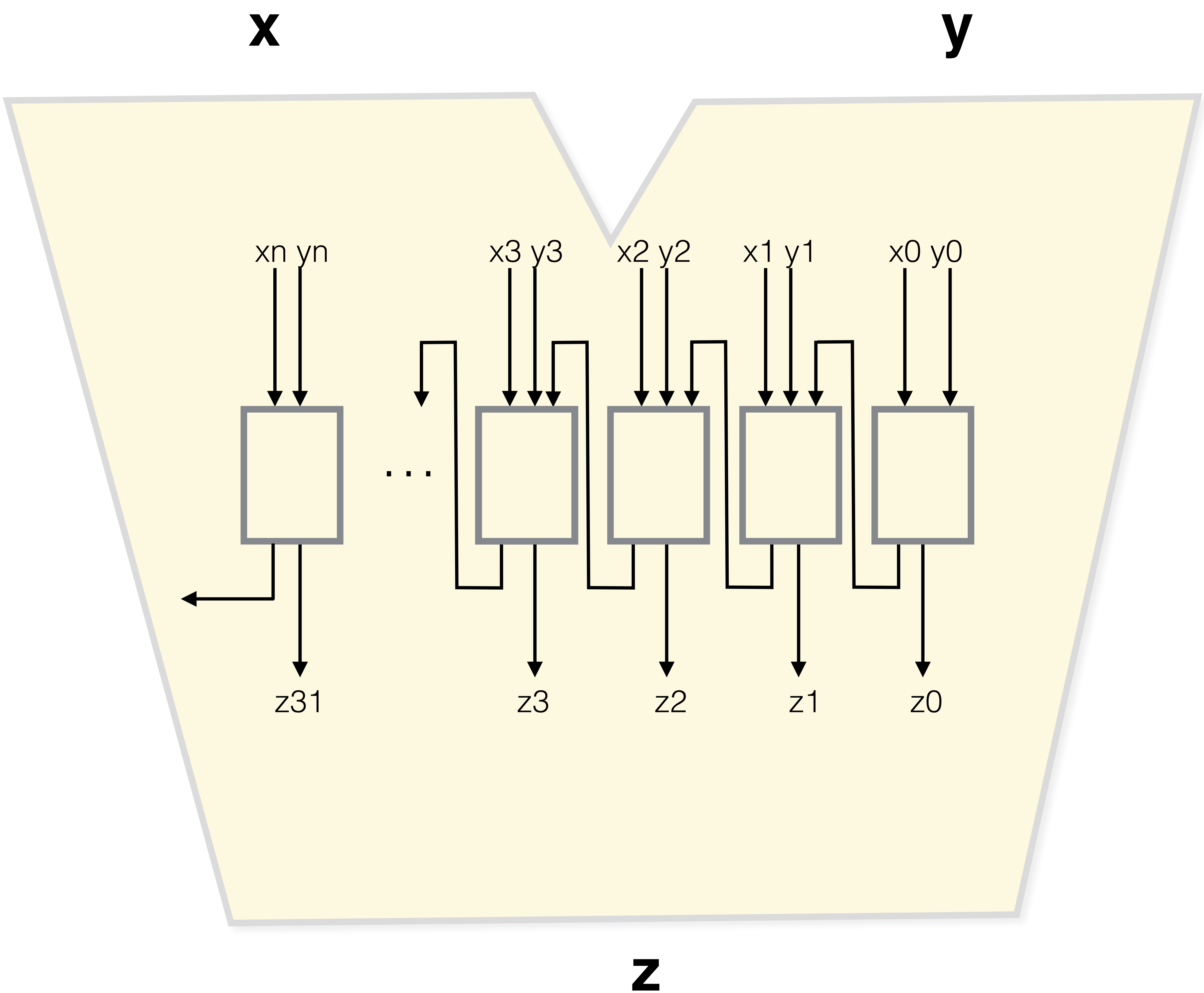
x0	y0	Carry	z0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

x0	y0	Carry	z0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Carry = x0 **and** y0

z0 = x0 **xor** y0


x0
y0
AND
XOR
carry
z0


x
y
xn yn
x3 y3
x2 y2
x1 y1
x0 y0
. . .
z31
z3
z2
z1
z0
z

Moral of the Story:
Addition is performed
by logic operations
using *natural* binary numbers…

(*unsigned arithmetic*)

How can we represent signed binary numbers when all we have are bits (0/1)?

Whichever system we use should work with the binary adder in the ALU…

4-bit Nybble

Binary	Hex	Unsigned Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

4-bit Nybble

Sign Bit

Binary	Hex	Unsigned Decimal
0 000	0	0
0 001	1	1
0 010	2	2
0 011	3	3
0 100	4	4
0 101	5	5
0 110	6	6
0 111	7	7
1 000	8	8
1 001	9	9
1 010	A	10
1 011	B	11
1 100	C	12
1 101	D	13
1 110	E	14
1 111	F	15

4-bit Nybble

Binary	Hex	Unsigned Decimal	
0 000	0	0	Positive Numbers
0 001	1	1	
0 010	2	2	
0 011	3	3	
0 100	4	4	
0 101	5	5	
0 110	6	6	
0 111	7	7	
1 000	8	8	Negative Numbers
1 001	9	9	
1 010	A	10	
1 011	B	11	
1 100	C	12	
1 101	D	13	
1 110	E	14	
1 111	F	15	

Signed Magnitude **Number *System***

Signed Magnitude Rule: To find the opposite of a number, just flip its MSB

0 101
(+5)

1 101
(-5)

and vice versa…

4-bit Nybble

Binary	Hex	Unsigned Decimal	**Signed Magnitude**
0 000	0	0	**+0**
0 001	1	1	**+1**
0 010	2	2	**+2**
0 011	3	3	**+3**
0 100	4	4	**+4**
0 101	5	5	**+5**
0 110	6	6	**+6**
0 111	7	7	**+7**
1 000	8	8	**-0**
1 001	9	9	**-1**
1 010	A	10	**-2**
1 011	B	11	**-3**
1 100	C	12	**-4**
1 101	D	13	**-5**
1 110	E	14	**-6**
1 111	F	15	**-7**

Does this System work With the ALU Adder?

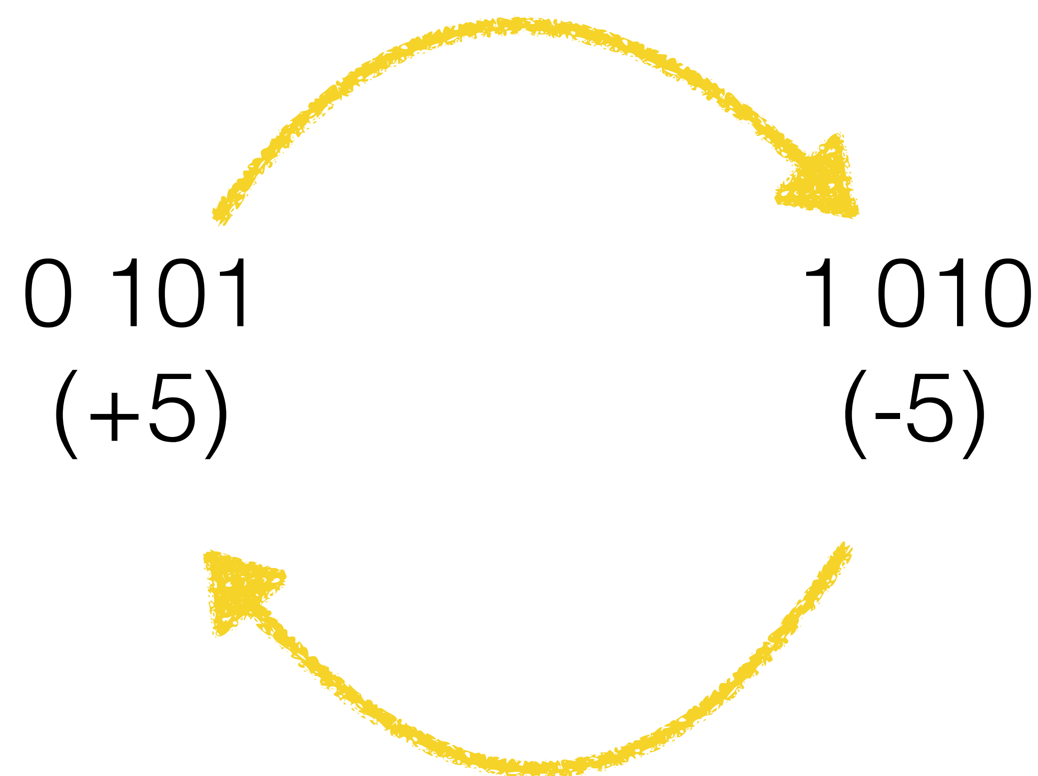
Binary	Hex	Unsigned Decimal	**Signed Magnitud**
0 000	0	0	**+0**
0 001	1	1	**+1**
0 010	2	2	**+2**
0 011	3	3	**+3**
0 100	4	4	**+4**
0 101	5	5	**+5**
0 110	6	6	**+6**
0 111	7	7	**+7**
1 000	8	8	**-0**
1 001	9	9	**-1**
1 010	A	10	**-2**
1 011	B	11	**-3**
1 100	C	12	**-4**
1 101	D	13	**-5**
1 110	E	14	**-6**
1 111	F	15	**-7**

```
    3
+  -3
------
=   0
```

```
    4
+  -1
------
=   3
```

1's Complement Number *System*

1's Complement Rule: To find the opposite of a number, just flip all its bits



and vice versa…

4-bit Nybble

Binary	Hex	Unsigned Decimal	**1's Complement**
0 000	0	0	**+0**
0 001	1	1	**+1**
0 010	2	2	**+2**
0 011	3	3	**+3**
0 100	4	4	**+4**
0 101	5	5	**+5**
0 110	6	6	**+6**
0 111	7	7	**+7**
1 000	8	8	**-7**
1 001	9	9	**-6**
1 010	A	10	**-5**
1 011	B	11	**-4**
1 100	C	12	**-3**
1 101	D	13	**-2**
1 110	E	14	**-1**
1 111	F	15	**-0**

Does this System work With the ALU Adder?

Binary	Hex	Unsigned Decimal	1's Complement
0 000	0	0	**+0**
0 001	1	1	**+1**
0 010	2	2	**+2**
0 011	3	3	**+3**
0 100	4	4	**+4**
0 101	5	5	**+5**
0 110	6	6	**+6**
0 111	7	7	**+7**
1 000	8	8	**-7**
1 001	9	9	**-6**
1 010	A	10	**-5**
1 011	B	11	**-4**
1 100	C	12	**-3**
1 101	D	13	**-2**
1 110	E	14	**-1**
1 111	F	15	**-0**

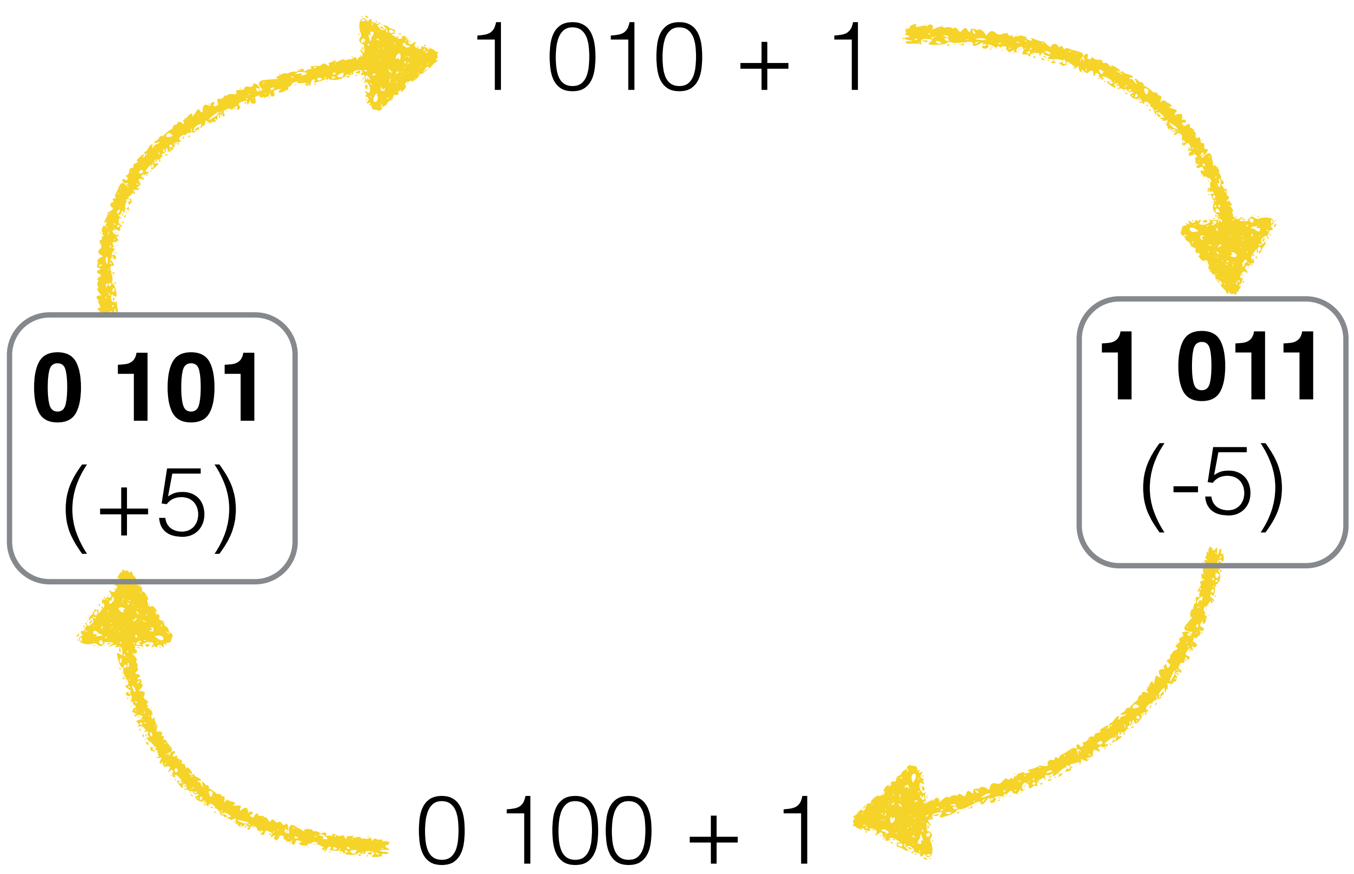
```
    3
+  -3
------
=   0
```

```
    4
+  -1
------
=   3
```

```
    5
+  -3
------
=   2
```

2's Complement
Number *System*

2's Complement Rule: To find the opposite of a number, just flip all its bits, and add 1



and vice versa…

4-bit Nybble

Binary	Hex	Unsigned Decimal	**2's Complement**
0 000	0	0	**+0**
0 001	1	1	**+1**
0 010	2	2	**+2**
0 011	3	3	**+3**
0 100	4	4	**+4**
0 101	5	5	**+5**
0 110	6	6	**+6**
0 111	7	7	**+7**
1 000	8	8	**-8**
1 001	9	9	**-7**
1 010	A	10	**-6**
1 011	B	11	**-5**
1 100	C	12	**-4**
1 101	D	13	**-3**
1 110	E	14	**-2**
1 111	F	15	**-1**

Does this System work With the ALU Adder?

Binary	Hex	Unsigned Decimal	2's Complement
0 000	0	0	**+0**
0 001	1	1	**+1**
0 010	2	2	**+2**
0 011	3	3	**+3**
0 100	4	4	**+4**
0 101	5	5	**+5**
0 110	6	6	**+6**
0 111	7	7	**+7**
1 000	8	8	**-8**
1 001	9	9	**-7**
1 010	A	10	**-6**
1 011	B	11	**-5**
1 100	C	12	**-4**
1 101	D	13	**-3**
1 110	E	14	**-3**
1 111	F	15	**-1**

3
+ -3
——————-
= 0

4
+ -1
——————-
= 3

5
+ -3
——————-
= 2

Interesting Property

- What is the binary representation of **-1** as a byte?
- What is the binary representation of **-1** as a word?
- What is the binary representation of **-1** as a dword?

```java
int x = 0x7fffffff - 5;

for ( int i=0; i<10; i++ )
   System.out.println( x++ );
```

getcopy Loop0x7fffffff.java

What did you just learn about Java ints?

neg

neg oprnd

```
neg      op8
neg      op16
neg      op32

op: mem, reg
```

To get the 2's complement of an int

```
alpha db    1
beta  dw    4
x     dd    0xF06

      neg    byte[alpha]
      mov    ax,1234
      neg    ax

      neg    dword[x]
```

Range of 2's Comp't. ints

Binary	Hex	Unsigned Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Type	Minimum	Maximum	# Bytes
unsigned byte	0	255	1
signed byte	-128	127	1
unsigned short	0	65,535	2
signed short	-32,768	32,767	2
unsigned int	0	4,294,967,295	4
signed int	-2,147,483,648	2,147,483,647	4
signed long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	8

Binary	unsigned	signed
0000 0000	0	+0
0000 0001	1	+1
0000 0010	2	+2
…		
…		
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	-128
1000 0001	129	-127
1000 0010	130	-126
…		
…		
1111 1110	254	-2
1111 1111	255	-1

Binary	unsigned	signed
0000 0000	0	+0
0000 0001	1	+1
0000 0010	2	+2
…		
…		
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	-128
1000 0001	129	-127
1000 0010	130	-126
…		
…		
1111 1110	254	-2
1111 1111	255	-1

Binary	unsigned	signed
0000 0000	0	+0
0000 0001	1	+1
0000 0010	2	+2
…		
…		
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	-128
1000 0001	129	-127
1000 0010	130	-126
…		
…		
1111 1110	254	-2
1111 1111	255	-1

$= -2^7$

Exercises on Signed Numbers

http://www.science.smith.edu/dftwiki/index.php/CSC231_Exercises_on_Signed_Numbers

Useful On-Line Tool

http://www.exploringbinary.com/twos-complement-converter/