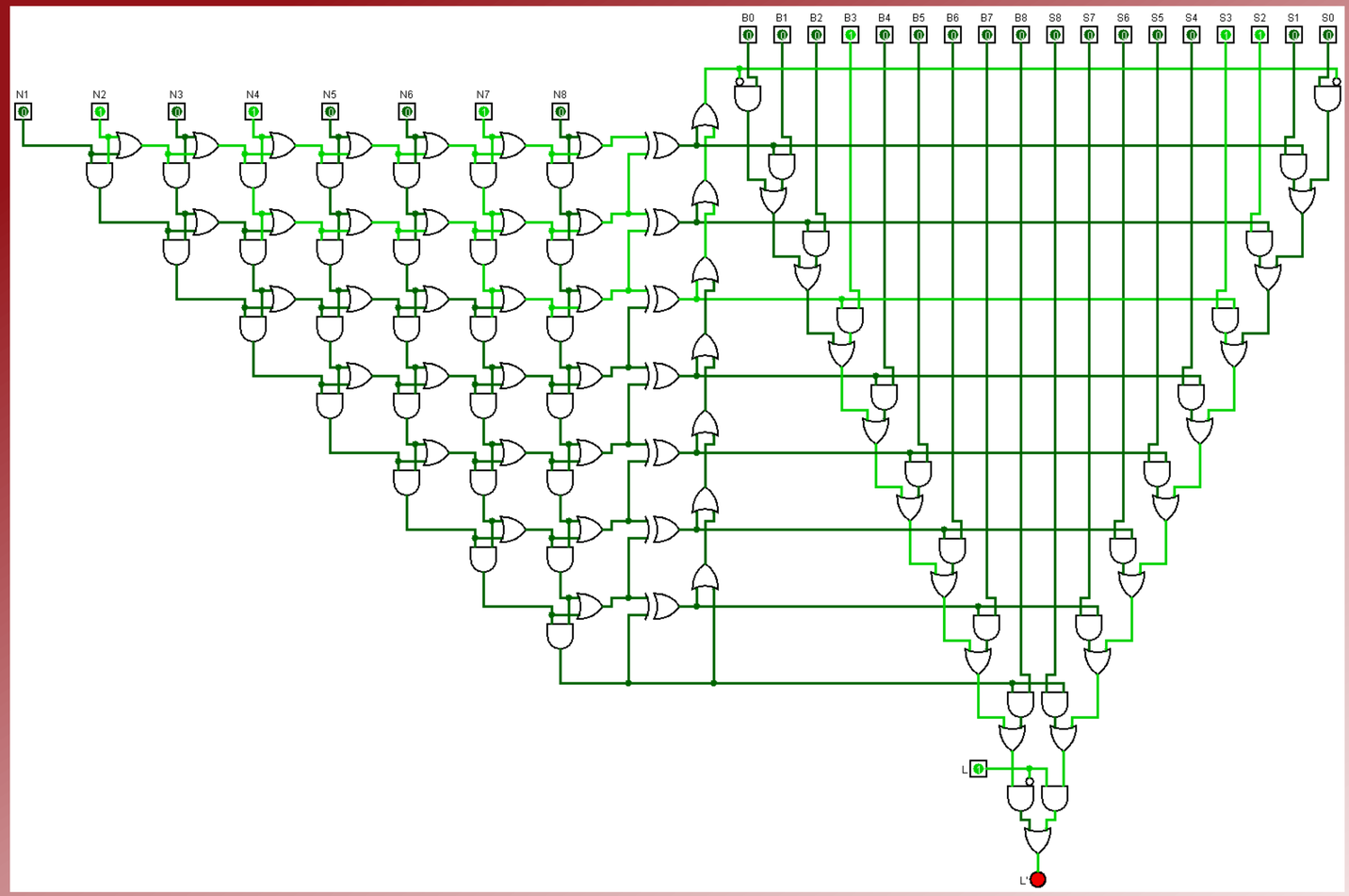


Lecture 6: Number Representation

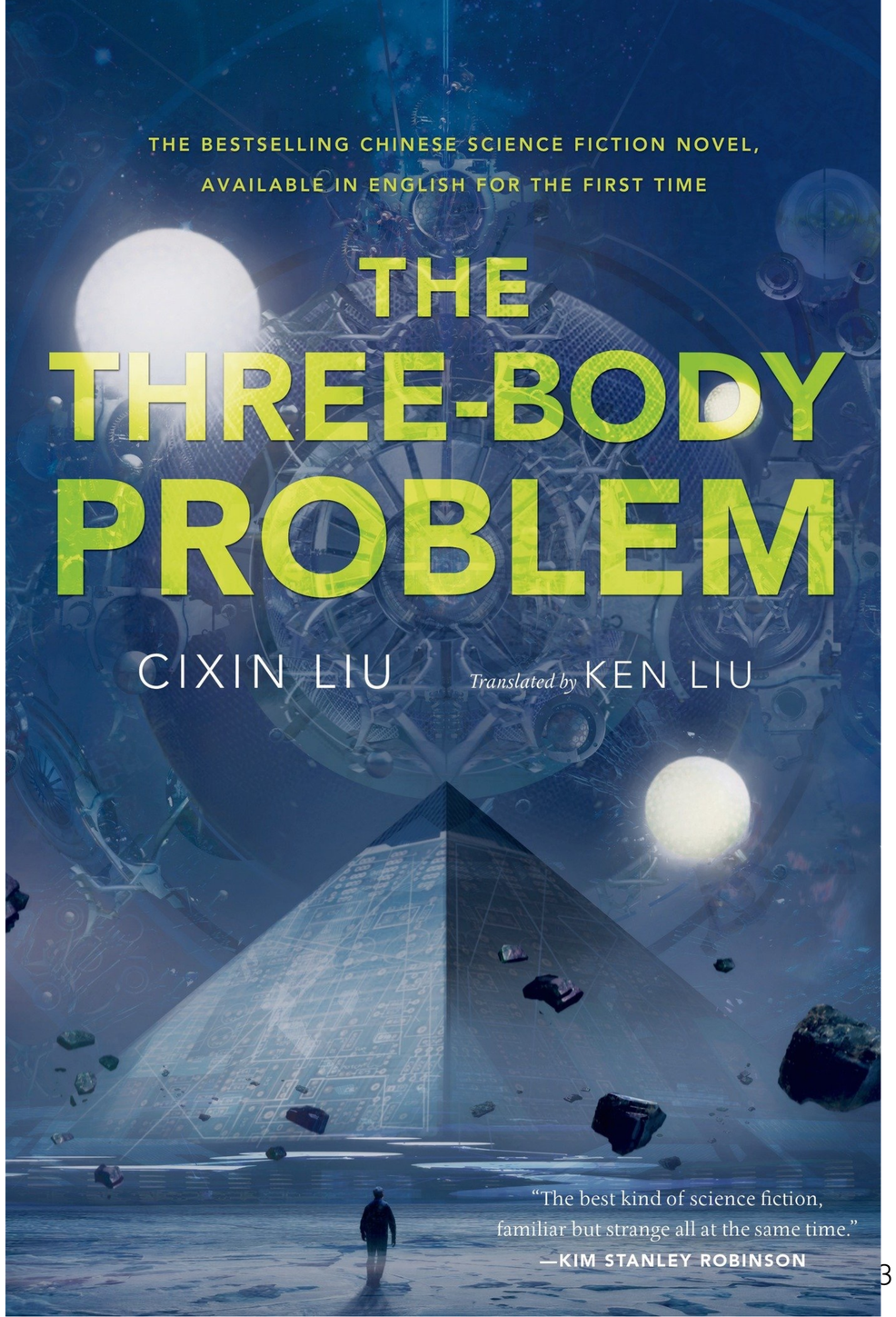
ENAE 380 Flight Software Systems
September 16, 2024

Topics … Why do we care?

Logic Gates

Binary Arithmetic

Hexadecimal



Set against the backdrop of China's Cultural Revolution, a secret military project sends signals into space to establish contact with aliens. An alien civilization on the brink of destruction captures the signal and plans to invade Earth. Meanwhile, on Earth, different camps start forming, planning to either welcome the superior beings and help them take over a world seen as corrupt, or to fight against the invasion. The result is a science fiction masterpiece of enormous scope and vision.

Amazon synopsis
(not a personal
endorsement)

“I don’t know your names,” Von Neumann said, tapping the shoulders of two of the soldiers. “The two of you will be responsible for signal input, so I’ll call you ‘Input 1’ and ‘Input 2.’” He pointed to the last soldier. “You will be responsible for signal output, so I’ll call you ‘Output.’” He shoved the soldiers to where he wanted them to stand. “Form a triangle. Like this. Output is the apex. Input 1 and Input 2 form the base.”

“You could have just told them to stand in the Wedge Attack Formation,” Qin Shi Huang said, glancing at Von Neumann contemptuously.

Newton took out six small flags: three white, three black. Von Neumann handed them out to the three soldiers so that each held a black flag and a white flag. “White represents 0; black represents 1. Good. Now, listen to me. Output, you turn around and look at Input 1 and Input 2. If they both raise black flags, you raise a black flag as well. Under all other circumstances, you raise the white flag.”

“I think you should use some other color,” Qin Shi Huang said. “White means surrender.”

The excited Von Neumann ignored him. He shouted orders at the three soldiers. “Begin operation! Input 1 and Input 2, you can raise whichever flag you want. Good. Raise! Good. Raise again! Raise!”

Input 1 and Input 2 raised their flags three times. The first time they were black-black, the second time white-black, and the third time black-white. Output reacted correctly each time, raising the black flag once and the white one twice.

“Very good. Your Imperial Majesty, your soldiers are very smart.”

“Even an idiot would be capable of that. Tell me, what are they really doing?” Qin Shi Huang looked baffled.

“The three soldiers form a computing component. It’s a type of gate, an AND gate.” Von Neumann paused to let the emperor digest this information.

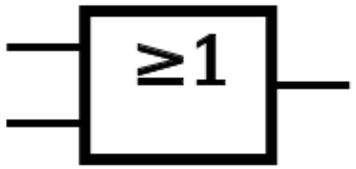
AND Gate

A	B	out
0	*0*	*0*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*

Von Neumann turned to the three soldiers again. “Let’s form another component. You, Output: if you see either Input 1 or Input 2 raise a black flag, you raise the black flag. There are three situations where that will be true: black-black, white-black, black-white. When it’s white-white, you raise the white flag. Understand? Good lad, you’re really clever. You’re the key to the correct functioning of the gate. Work hard, and the emperor will reward you! Let’s begin operation. Raise! Good, raise again! Raise again! Perfect. Your Imperial Majesty, this component is called an OR gate.”

OR Gate

A	B	out
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*1*

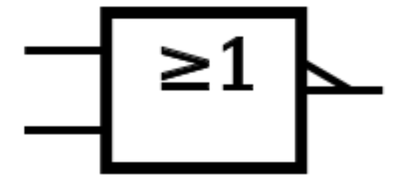


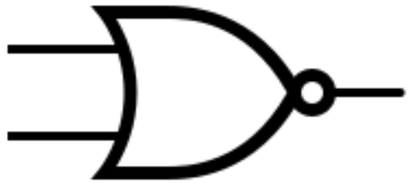
NAND Gate

A	B	out
0	*0*	*1*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

NOR Gate

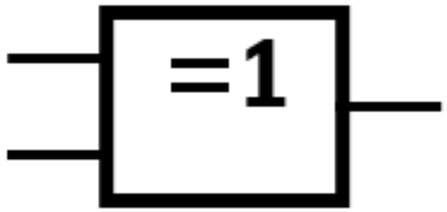
A	B	out
0	*0*	*1*
0	*1*	*0*
1	*0*	*0*
1	*1*	*0*





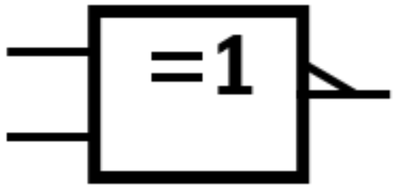
XOR Gate

A	B	out
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*



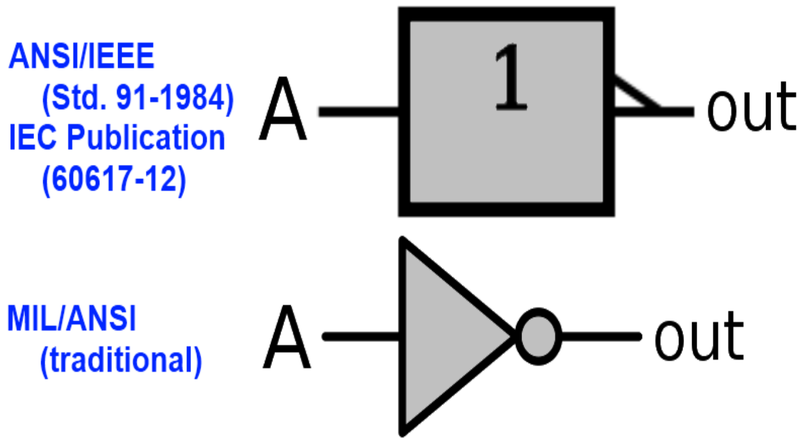
XNOR Gate

A	B	out
0	*0*	*1*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*



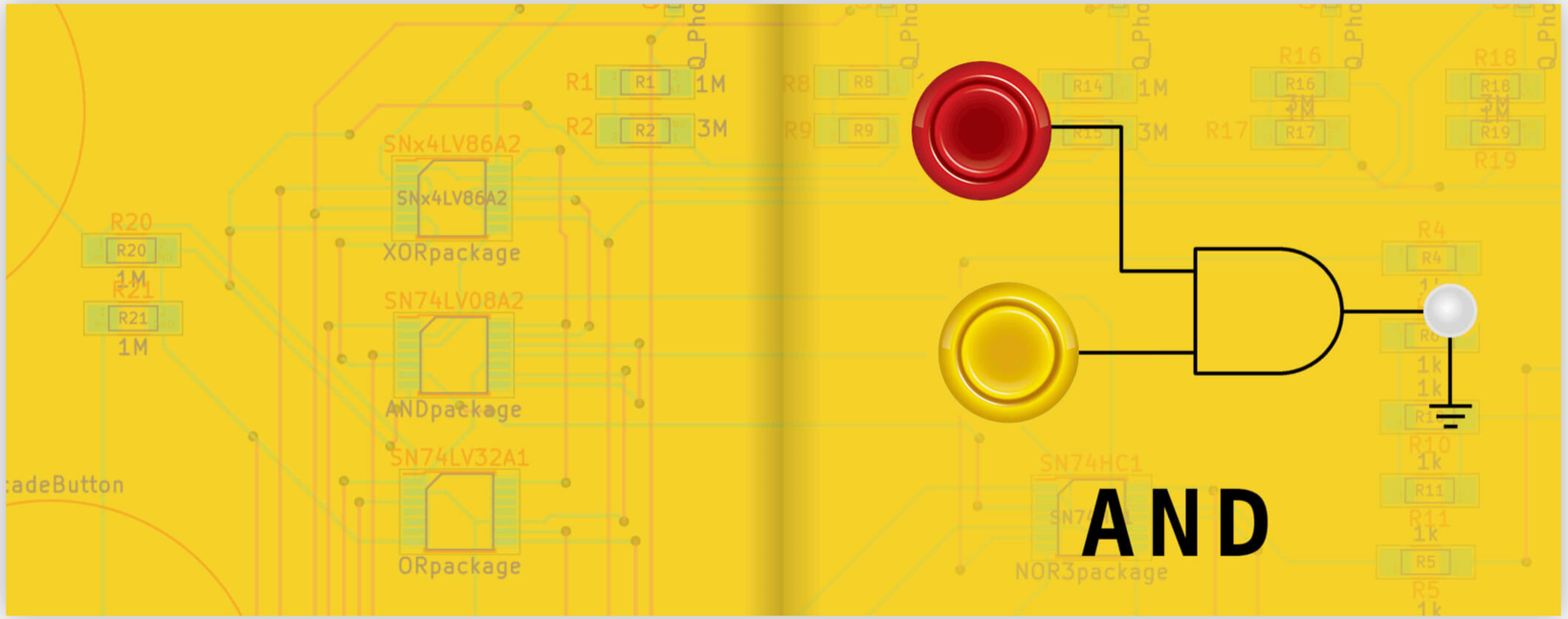
NOT Gate

A	out
0	*1*
1	*0*



https://computerengineeringforbabies.com/

Demo the Book:

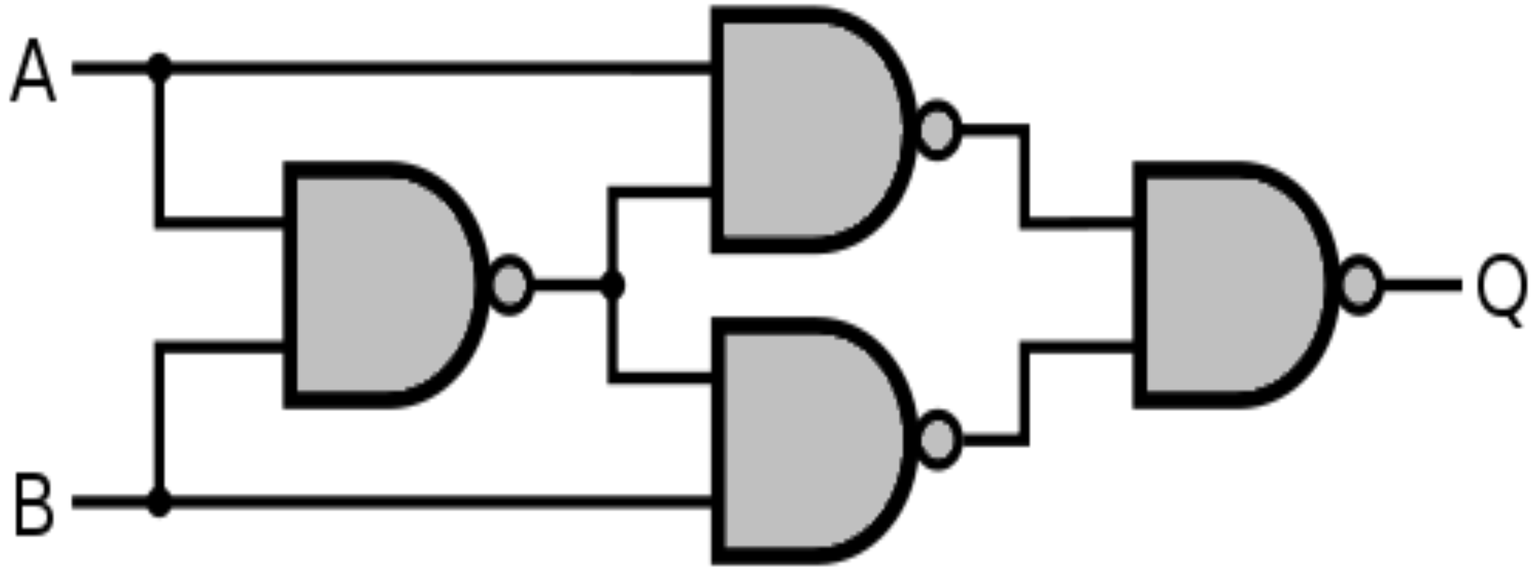


Click the button(s) to activate the LED.

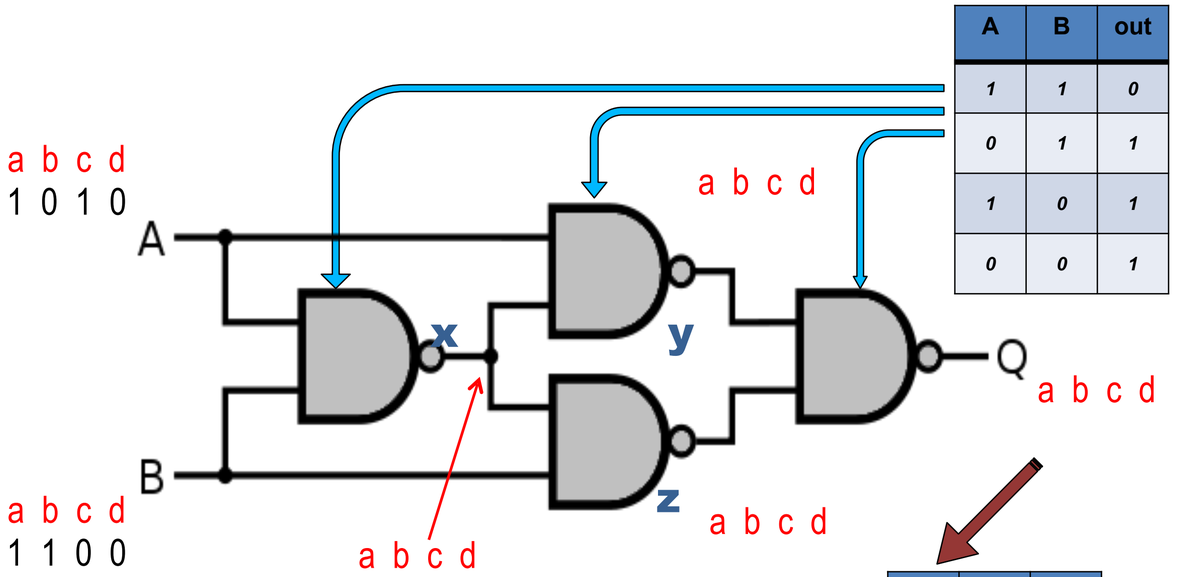
Use arrow keys to turn the page.

XOR Circuit

- **XOR** gate derived from just NAND gates
  - Not XOR, inverse of the XOR



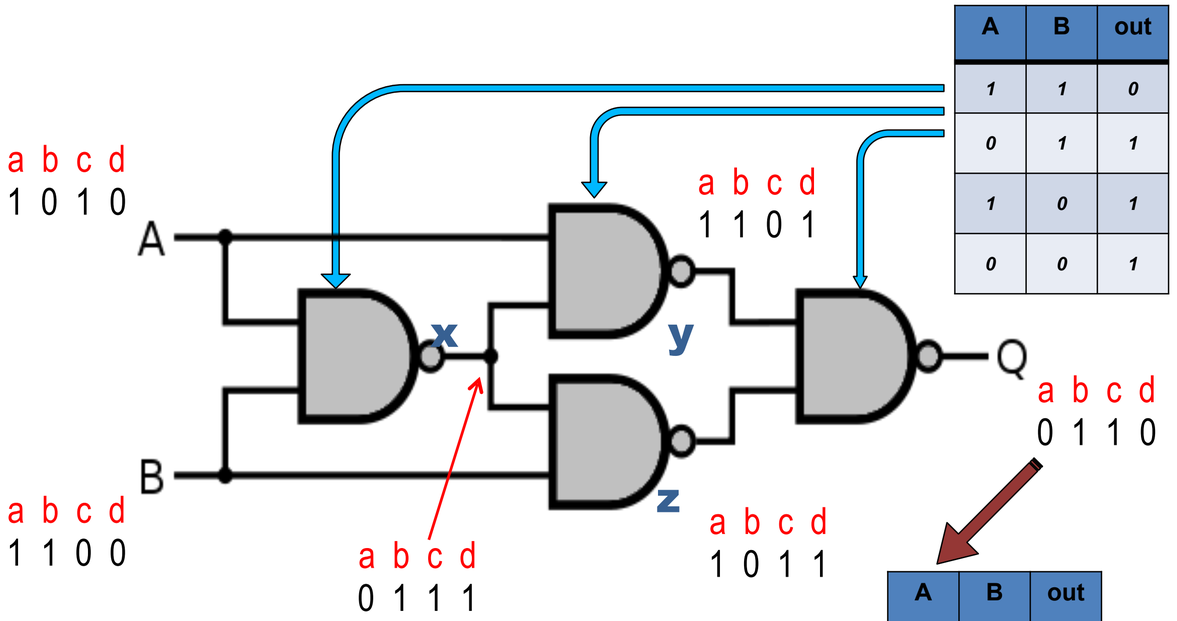
XOR Circuit



A	B	out
1	1	0
0	1	1
1	0	1
0	0	1

A	B	out
1	1	0
0	1	1
1	0	1
0	0	0

XOR Circuit



A	B	out
1	1	0
0	1	1
1	0	1
0	0	1

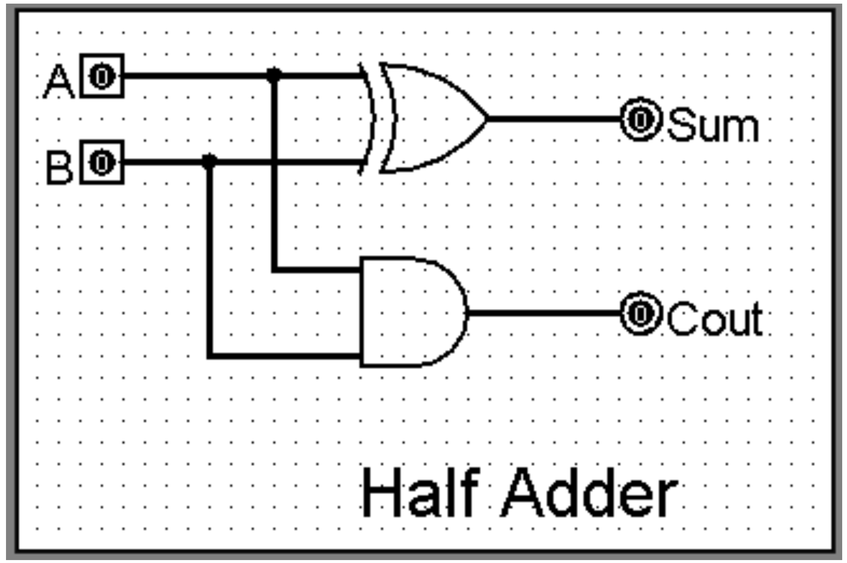
A	B	out
1	1	0
0	1	1
1	0	1
0	0	0

Half-Adder

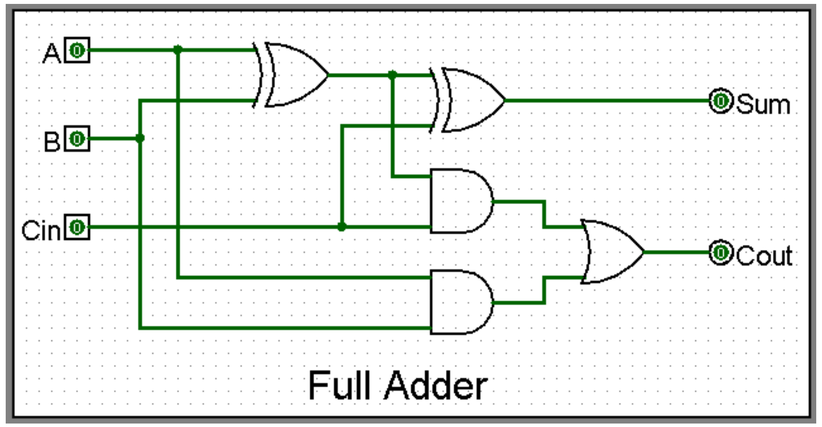
Half Adder Circuit

- Combinatorial logic circuits to perform binary addition.
- Add two 3-bit numbers
  - 110 + 011 = (1) 0 0 1
  - When we perform two k-bit numbers, expect answer to be k-bits. But, results can be k+1 bits.
  - Need a “carry” bit

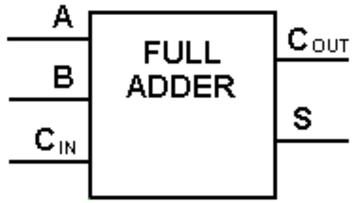
A	B	Sum	Cout
0	*0*	*0*	*0*
0	*1*	*1*	*0*
1	*0*	*1*	*0*
1	*1*	*0*	*1*



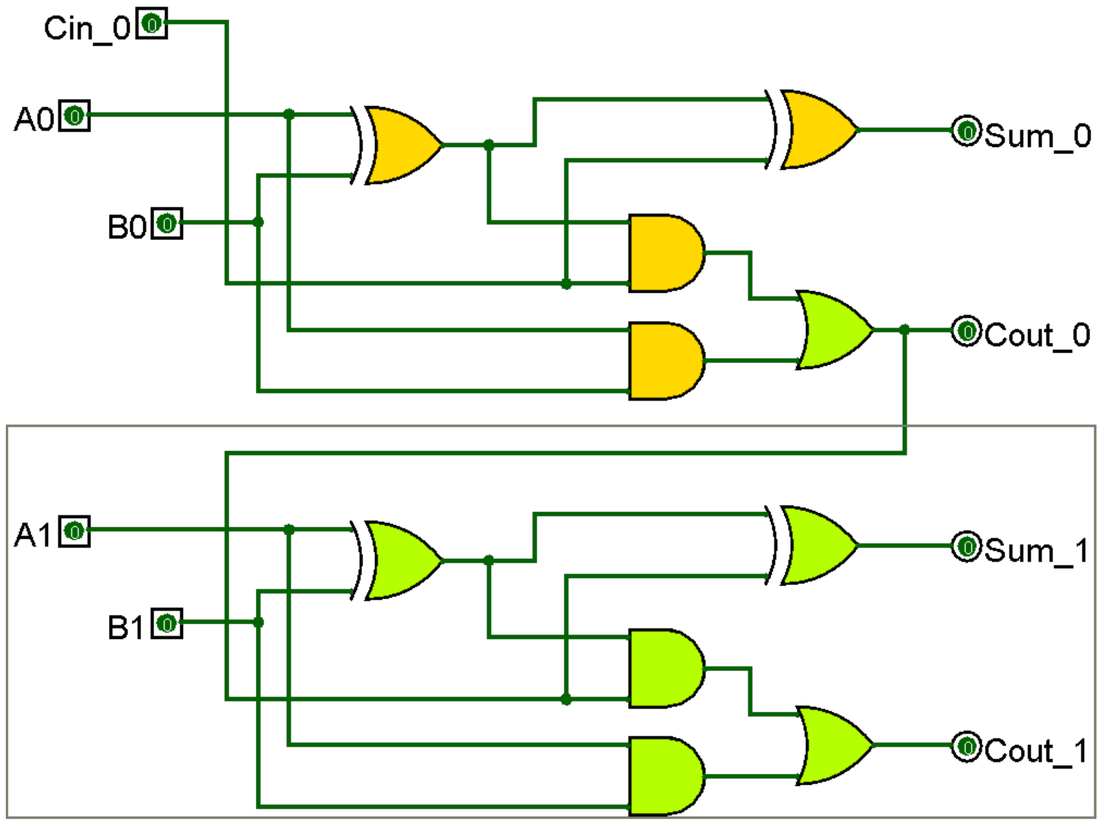
Full Adder Circuit



A	B	Cin	Cout	Sum
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*

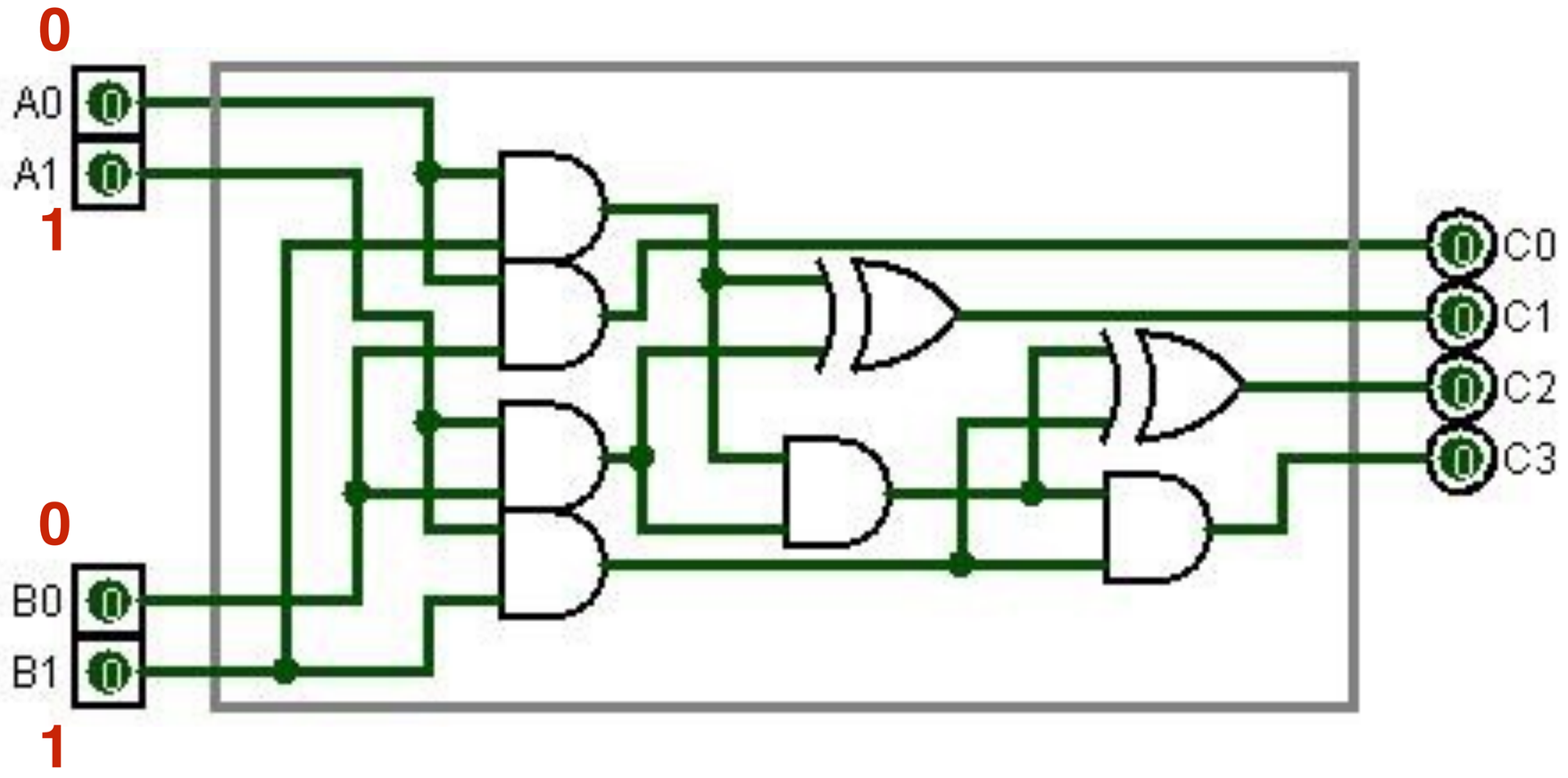


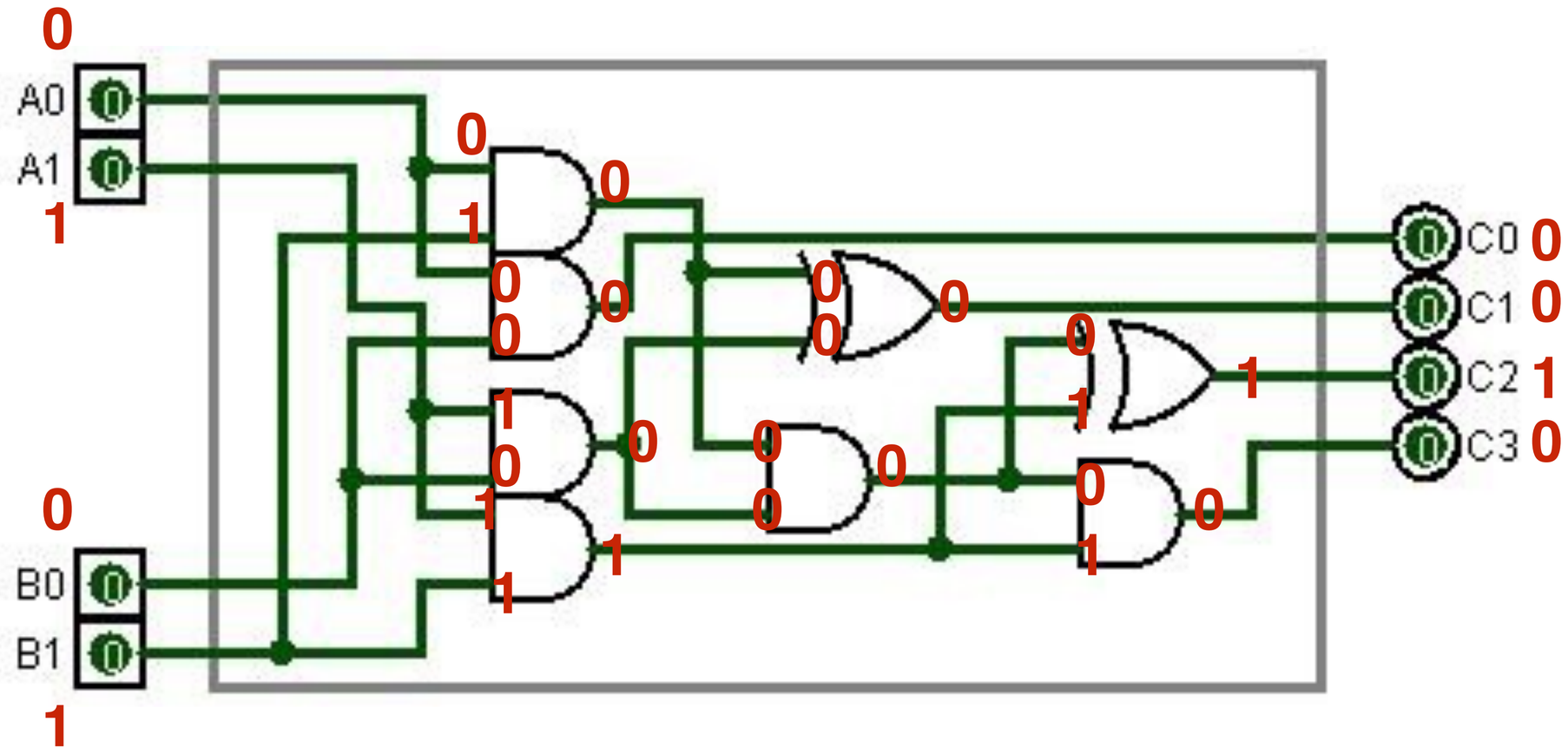
Two-Bit Full Adder Circuit



```
    1011
  x 1110
  ======
    0000
   1011
  1011
+ 1011
=========
 10011010
```

Multiplier


0
A0
A1
1
0
B0
B1
1
C0
C1
C2
C3


A0
A1
B0
B1
C0
C1
C2
C3

Hexadecimal can save your life.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

Positional Notation

- Today we use base-10 notation (decimal)
  - Motivated by?…
- Exponentiation
  - 465
    - Base 10
      - $4x10^2 + 6x10^1 + 5x10^0$
    - Base 7

```
243
```

    - Base 12

```
653
```

Human Language

- Base 12: many factors (2, 3, 4, 6)
  - Dozen, Gross, time
  - 12 pence in 1 shilling, 20 shillings in 1 pound
- Base 8 used in Yuki tribe in northern California
  - Space between fingers
- Base 20
  - Mayans, North American Tribes
  - Gauls (France)
    - 60, 70, 80, 90
      - soixante, soixante-dix, quatre-vingt, quatre-vint-dix
      - sixty, sixty-ten, four twenties, four twenties and ten

Bases

- Binary numbers are base 2.
  - Each digit in a string of digits has two times the numerical power as it's neighbor to the right.
- Decimal numbers are base 10.
  - Each digit in a string of digits has ten times the numerical power as it's neighbor to the right.
- Hexadecimal numbers are base 16.
  - Each digit in a string of digits has sixteen times the numerical power as it's neighbor to the right.
- Octal numbers are base 8.
  - Each digit in a string of digits has eight times the numerical power as it's neighbor to the right.

Number Base Notation

- Can be confusing when discussing different numerical bases
  - Subscript 362_{10}
  - Decimal numbers never start with zero
  - Hex numbers begin with 0x
  - Binary numbers begin with 0b

Why do we use hexadecimal?

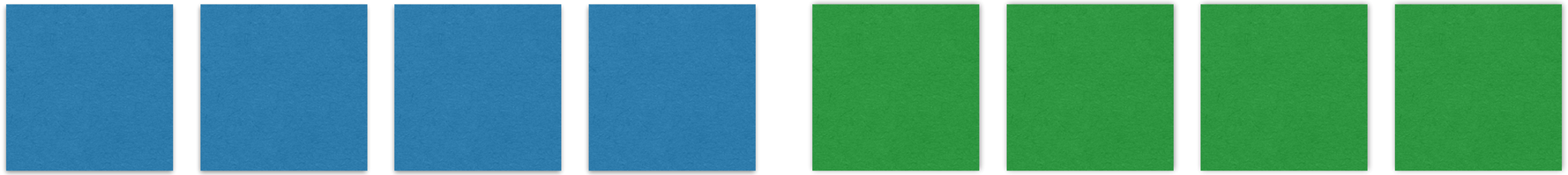
Because engineers are lazy.

1 byte = 8 bits

0000 0000 - 1111 1111

(0 to 255)

Top 4 bits, low 4 bits



4 bits gives a range of 16, so we use base 16 system

Easier to express binary value to someone as “A” than “1010”

2 hex values represent 1 byte

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

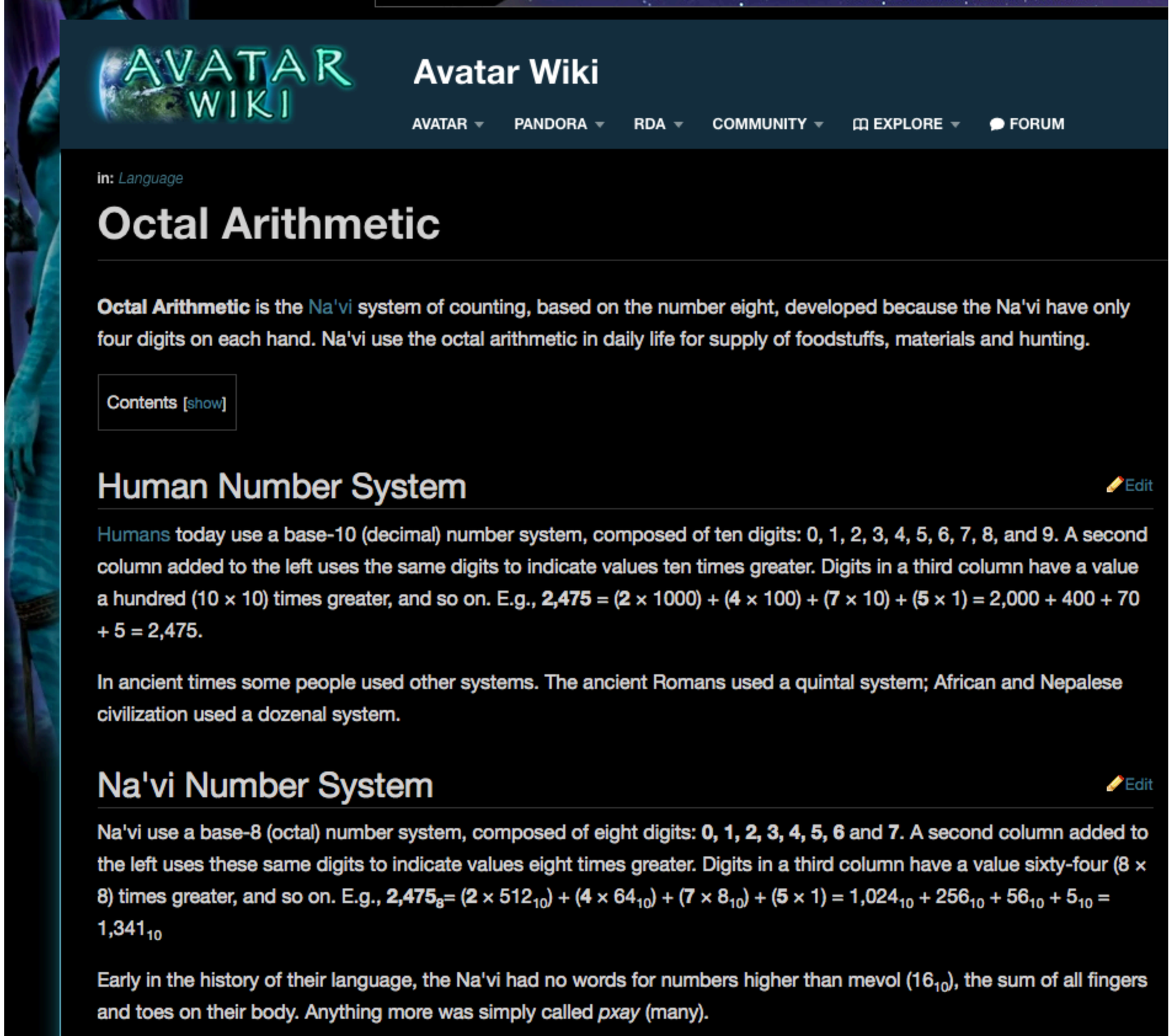
What is 0xCE in binary?

Side Note: Octal

- Single octal digit has 3 bits
  - 311_8 is 11 001 001
- Used in UNIX for permissions due to 3 bit nature
  - R-W-X for a file (user, group, world)
    - chmod 744
    - 111 100 100

Value	Permission
7	rwx
6	rw-
5	r-x
4	r--
3	-wx
2	-w-
1	--x
0	---

Side Side Note: Avatar



in: *Language*

Octal Arithmetic

Octal Arithmetic is the Na'vi system of counting, based on the number eight, developed because the Na'vi have only four digits on each hand. Na'vi use the octal arithmetic in daily life for supply of foodstuffs, materials and hunting.

Contents [show]

Human Number System

Edit

Humans today use a base-10 (decimal) number system, composed of ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. A second column added to the left uses the same digits to indicate values ten times greater. Digits in a third column have a value a hundred (10 × 10) times greater, and so on. E.g., **2,475** = (**2** × 1000) + (**4** × 100) + (**7** × 10) + (**5** × 1) = 2,000 + 400 + 70 + 5 = 2,475.

In ancient times some people used other systems. The ancient Romans used a quintal system; African and Nepalese civilization used a dozenal system.

Na'vi Number System

Edit

Na'vi use a base-8 (octal) number system, composed of eight digits: **0, 1, 2, 3, 4, 5, 6** and **7**. A second column added to the left uses these same digits to indicate values eight times greater. Digits in a third column have a value sixty-four (8 × 8) times greater, and so on. E.g., $\mathbf{2,475}_{8} = (\mathbf{2} \times 512_{10}) + (\mathbf{4} \times 64_{10}) + (\mathbf{7} \times 8_{10}) + (\mathbf{5} \times 1) = 1{,}024_{10} + 256_{10} + 56_{10} + 5_{10} = 1{,}341_{10}$

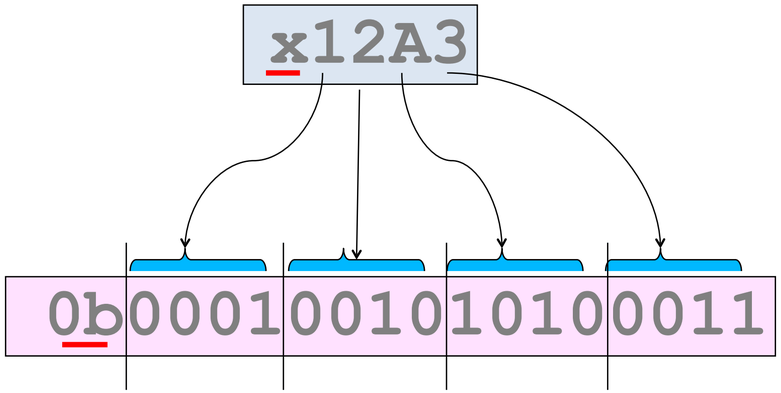
Early in the history of their language, the Na'vi had no words for numbers higher than mevol (16_{10}), the sum of all fingers and toes on their body. Anything more was simply called *pxay* (many).

Comparison

DECIMAL	HEXADECIMAL	OCTAL	BINARY
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Hexadecimal to Binary

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

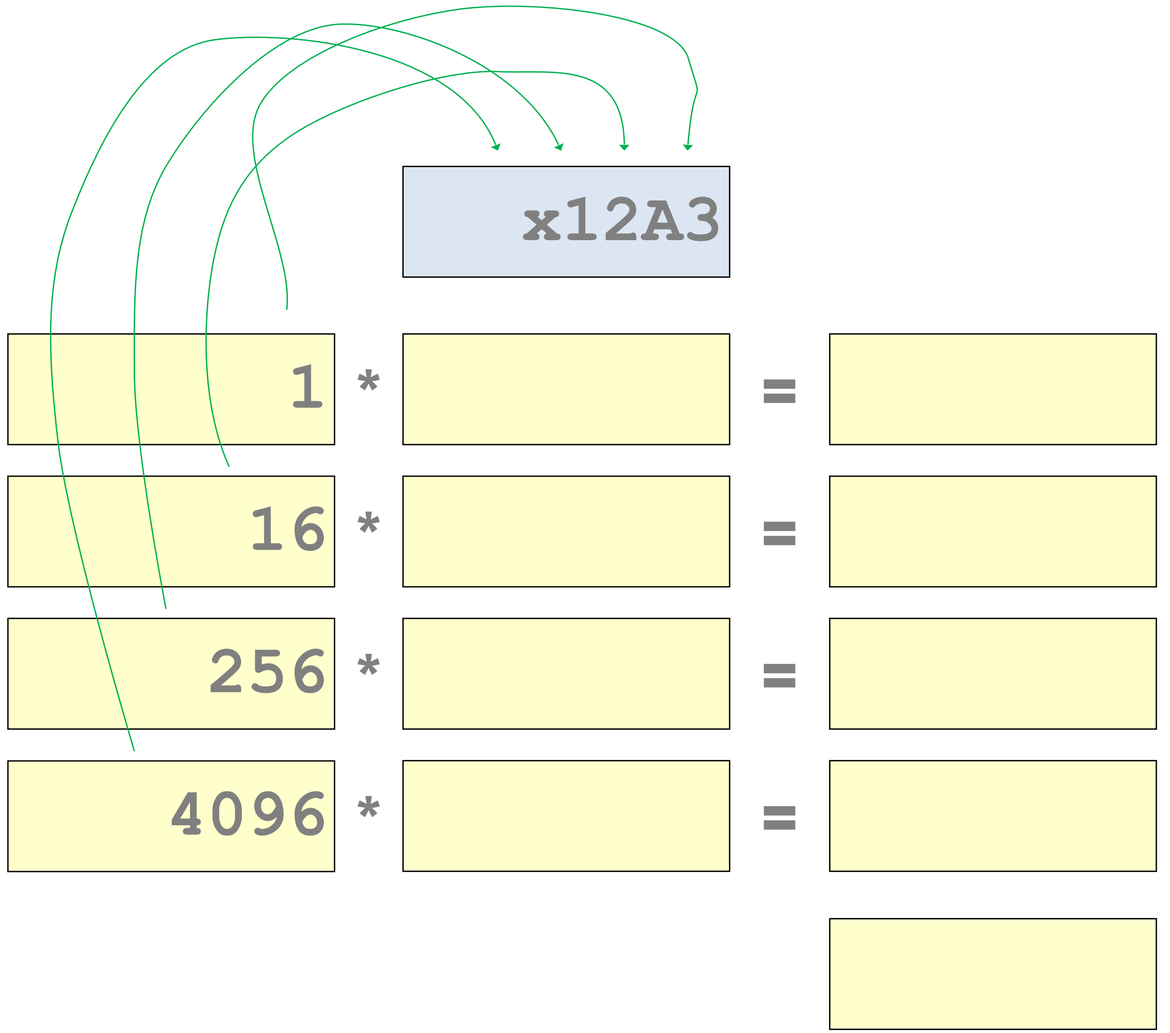


x8C3F

0b1000 1100 0011 1111

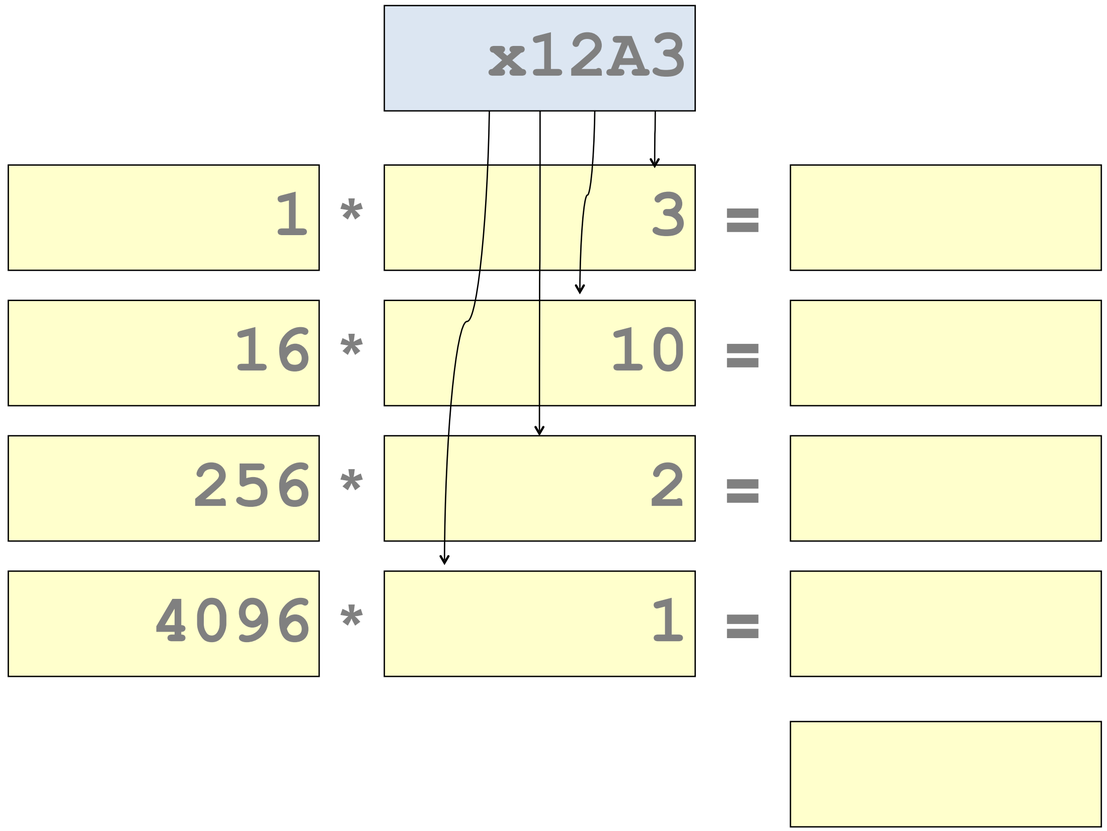
Converting Hex to Decimal

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



Converting Hex to Decimal

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



Converting Hex to Decimal

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

x12A3

1	*	3	=	3
16	*	10	=	160
256	*	2	=	512
4096	*	1	=	4096
				4771

Converting Dec to Hex

Hexadecimal Column	Decimal value
0x00000001	1
0x00000010	16
0x00000100	256
0x00001000	4,096
0x00010000	65,536
0x00100000	1,048,576
0x01000000	16,777,216
0x10000000	268,435,456

7845_{10}

Converting Dec to Hex

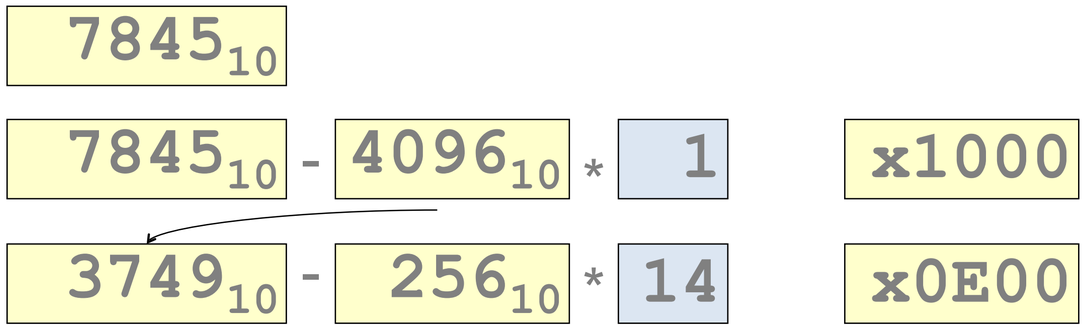
Hexadecimal Column	Decimal value
0x00000001	1
0x00000010	16
0x00000100	256
0x00001000	4,096
0x00010000	65,536
0x00100000	1,048,576
0x01000000	16,777,216
0x10000000	268,435,456

7845_{10}

7845_{10} - 4096_{10} * 1 x1000

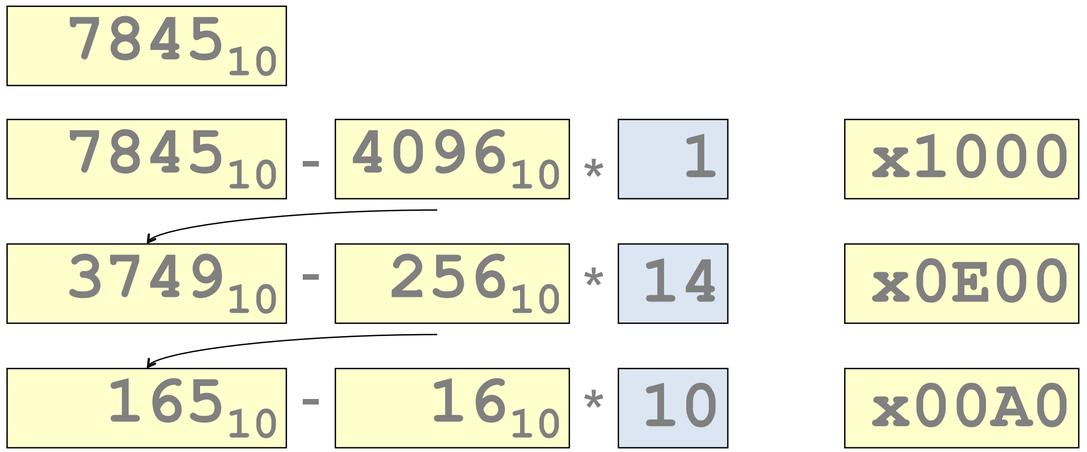
Converting Dec to Hex

Hexadecimal Column	Decimal value
0x00000001	1
0x00000010	16
0x00000100	256
0x00001000	4,096
0x00010000	65,536
0x00100000	1,048,576
0x01000000	16,777,216
0x10000000	268,435,456



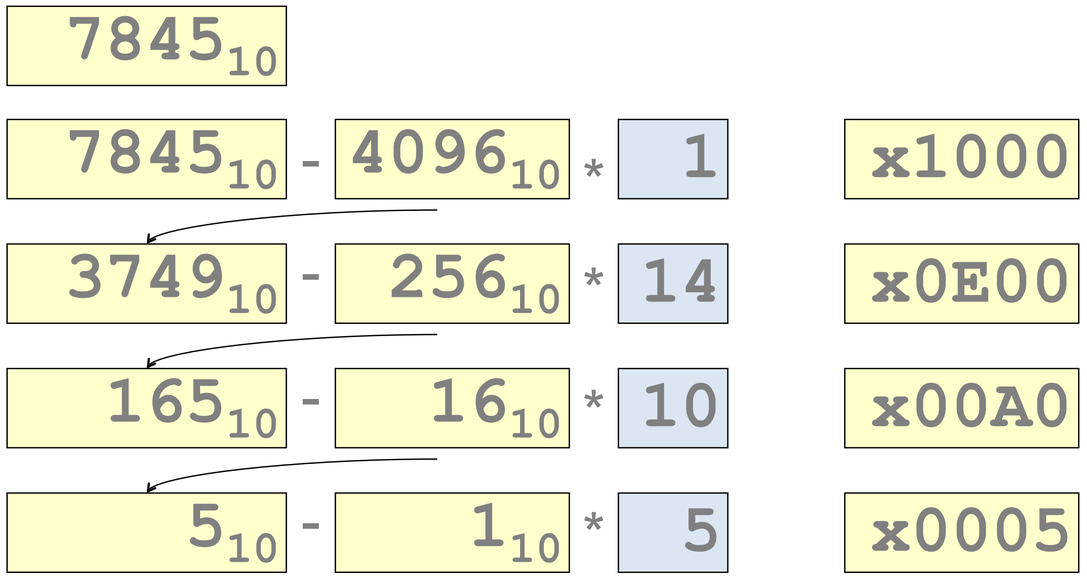
Converting Dec to Hex

Hexadecimal Column	Decimal value
0x00000001	1
0x00000010	16
0x00000100	256
0x00001000	4,096
0x00010000	65,536
0x00100000	1,048,576
0x01000000	16,777,216
0x10000000	268,435,456



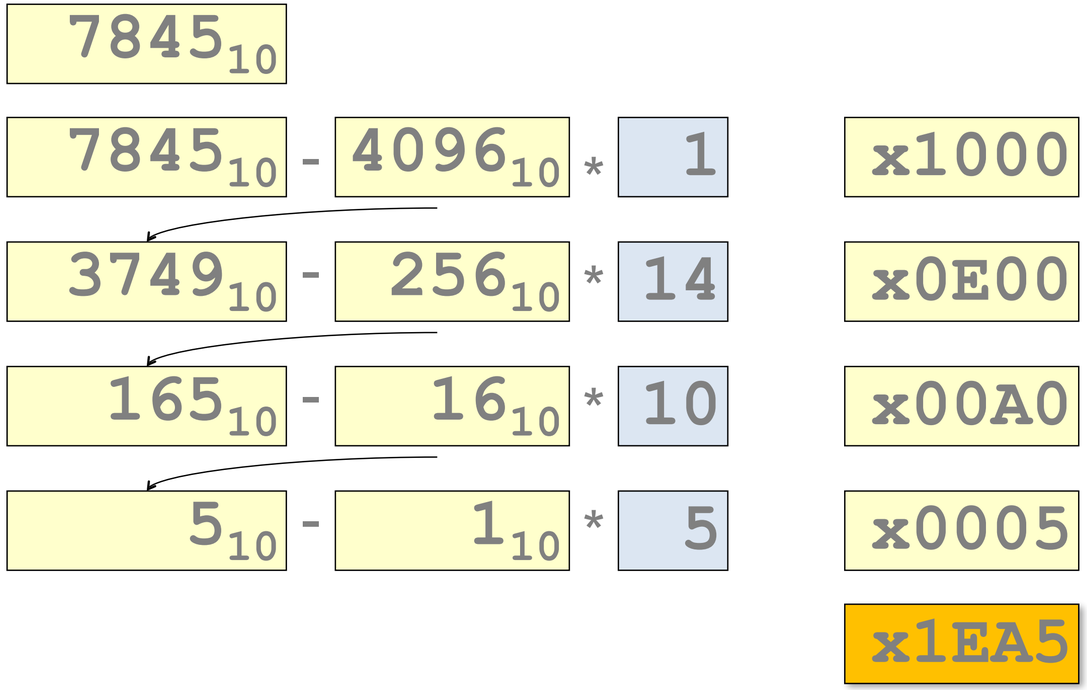
Converting Dec to Hex

Hexadecimal Column	Decimal value
0x00000001	1
0x00000010	16
0x00000100	256
0x00001000	4,096
0x00010000	65,536
0x00100000	1,048,576
0x01000000	16,777,216
0x10000000	268,435,456



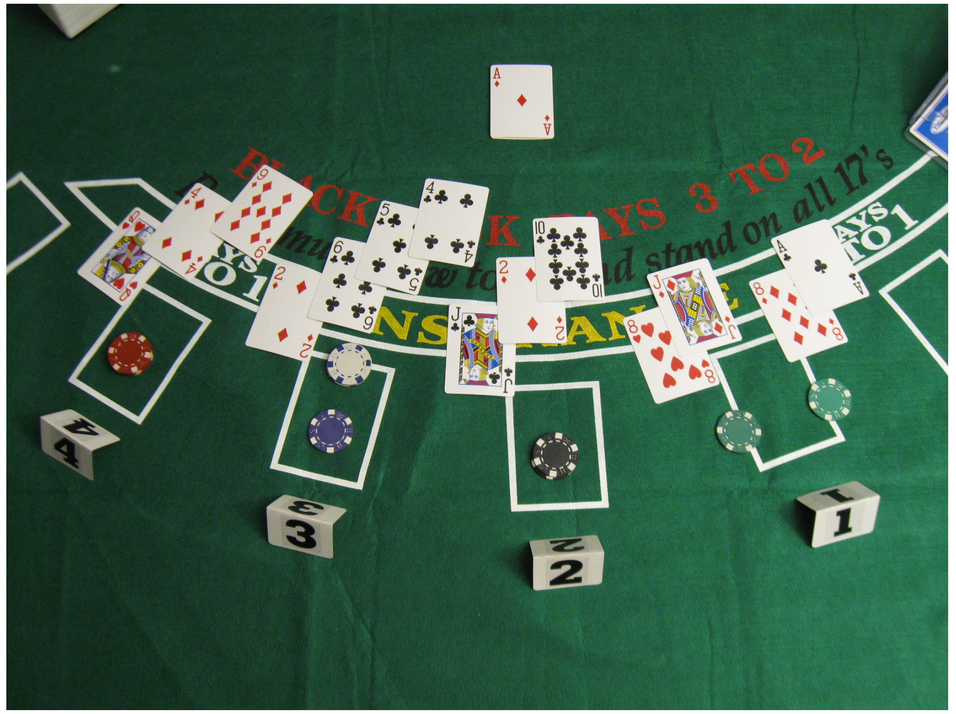
Converting Dec to Hex

Hexadecimal Column	Decimal value
0x00000001	1
0x00000010	16
0x00000100	256
0x00001000	4,096
0x00010000	65,536
0x00100000	1,048,576
0x01000000	16,777,216
0x10000000	268,435,456



Hexadecimal Blackjack

- Each participant attempts to beat the dealer by getting a count as close to **31** as possible, without going over 31.
- Dealer and player receive two cards each. Dealer has one card face down.
- Player can keep his hand (stand) or take more cards from deck (hit)
- When all players are done, the dealer turns over their card. Dealer must hit if value is lower than 28, otherwise the dealer will stand.



Hexadecimal Blackjack

```
import random
import numpy as np

deck =np.array([0,"ACE",2,3,4,5,6,7,8,9,"A","B","C","D","E","F"])
fulldeck = np.repeat(deck,4)

# print fulldeck

var = 1
while var==1:
    num = input("New card? (y/n) ")
    if num=='y':
        card = random.choice(fulldeck)
        print("New card value is: " + str(card))
    else:
        var = 0

print("Good bye!")

```

Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

```
  x18A6
+ x2913
-------
    ?
```

Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

```
  x18A6
+ x2913
-------
     9
```

Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

```
  x18A6
+ x2913
-------
   B9
```

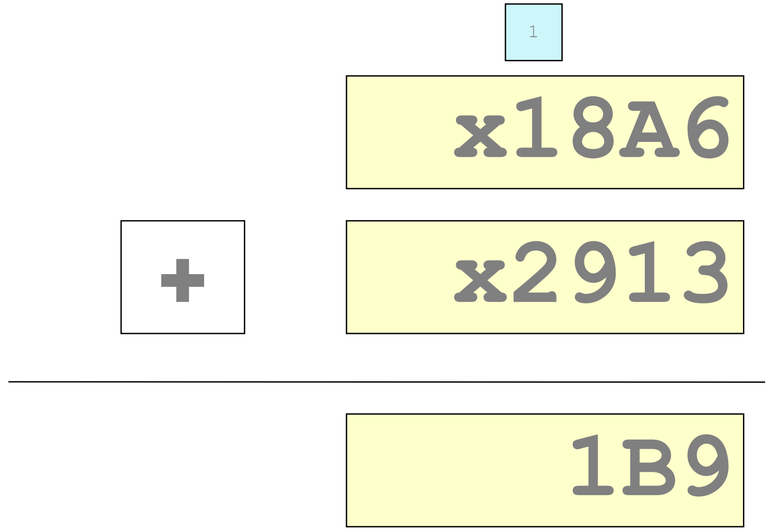
Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

```
  x18A6
+ x2913
-------
  1B9
```

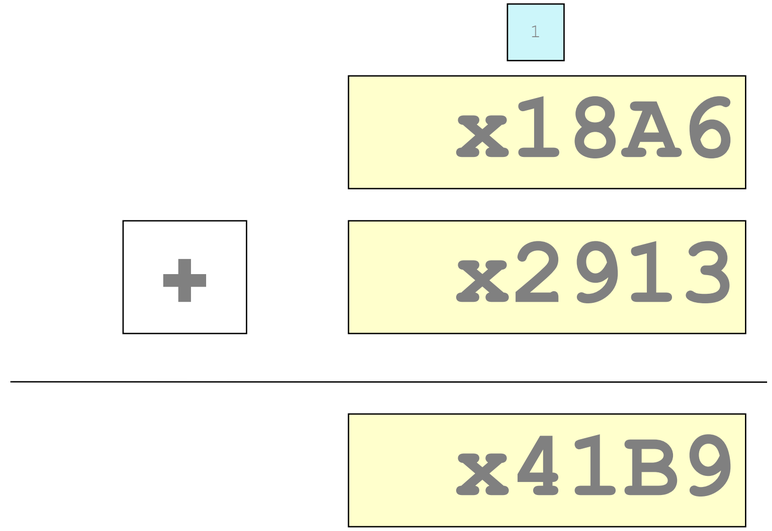
Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



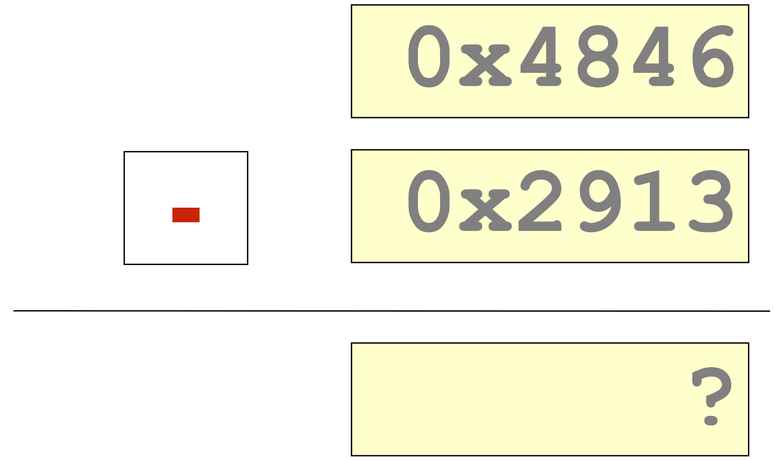
Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



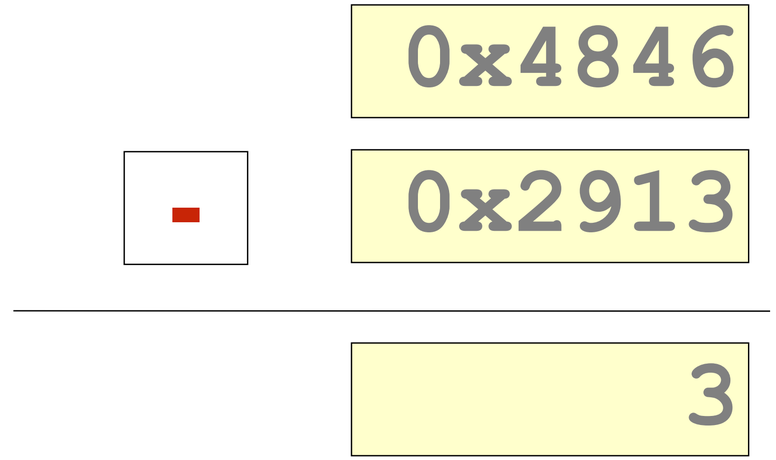
Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



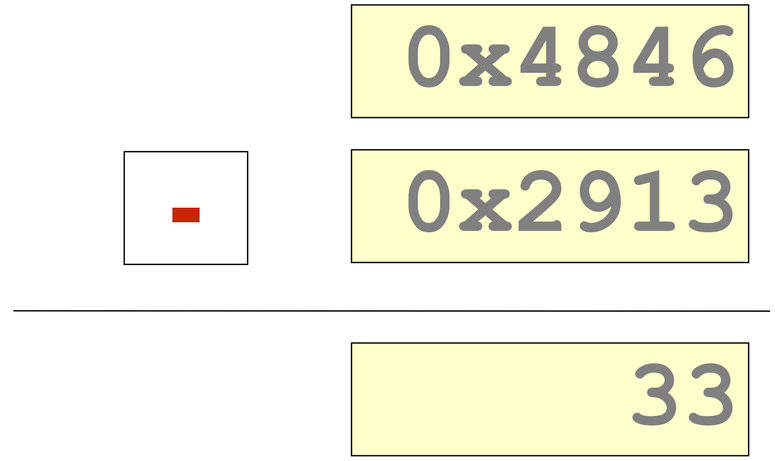
Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



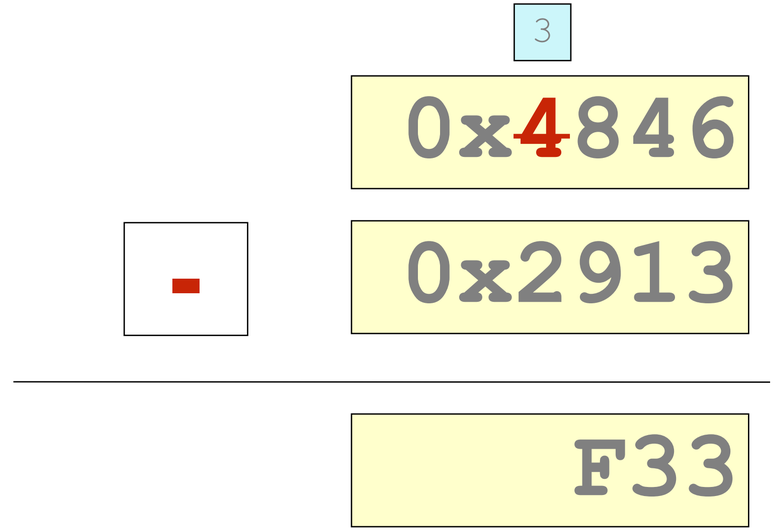
Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



Hexadecimal Arithmetic

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F