CS 31: Introduction to Computer Systems

02: Introduction & Data Representation

01-23-2025

SWARTHMORE COLLEGE

Announcements

- Register your clicker! https://forms.gle/YBFvNWPTXgiySMHx5
- Submit Lab 0!
- Reading quizzes count from next week!
- Edstem: Turn on notifications so you get email when we post

Reading Quiz

- Note the red border!
- 1 minute per question
- No talking, no laptops, phones during the quiz

<u>Check your frequency:</u>

- Iclicker2: frequency AA
- Iclicker+: green light next to selection

For new devices this should be okay,
For used you may need to reset frequency

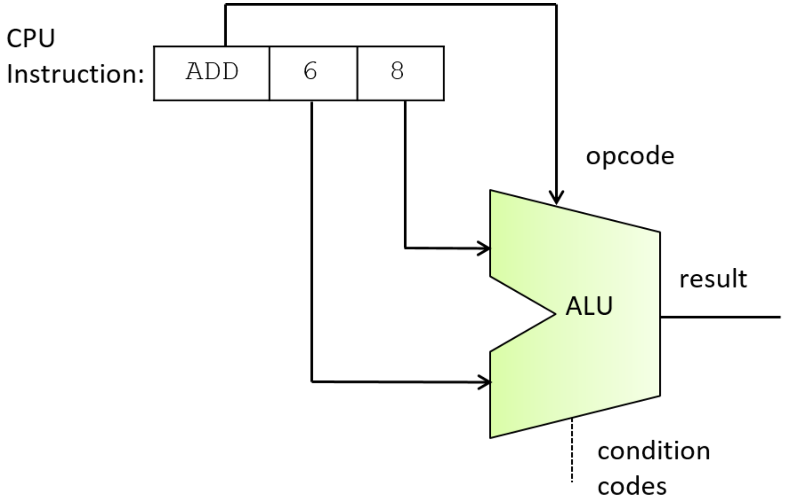
Reset:

1. hold down power button until blue light flashes (2secs)
2. Press the frequency code: AA
   vote status light will indicate success

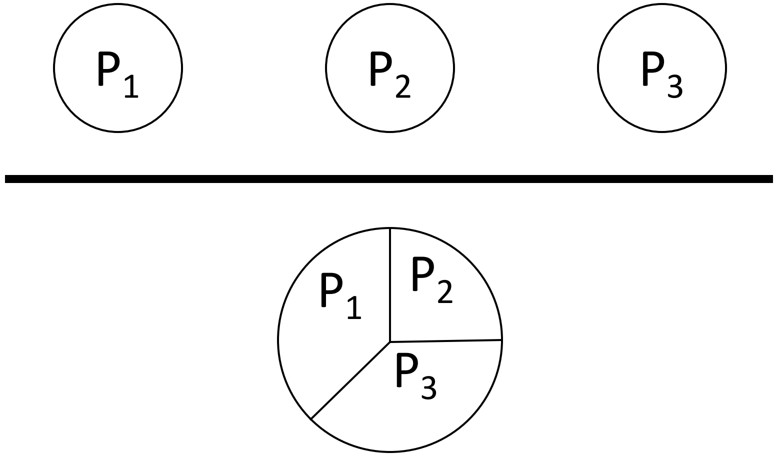
What is a computer system?

Hardware (HW) & Special Systems Software (OS) that work together to run application programs

- HW executes program instructions
- OS that manages the computer HW
- OS also provides abstractions to the programs/users



Computer Hardware



Operating System
CPU | Memory | Storage

C program:

```
// example C program
void main() {
  int authenticate;
  scanf("Enter your
  username and pwd:");
}
```

Program

What is a computer system?

Hardware (HW) & Special Systems Software (OS) that work together to run application programs

What are the goals of our system? Correctness

- Is x^2 >= 0?
  - Floating point values: Yes!
  - Integers
    - 40000 * 40000 = 1600000000
    - 50000 * 50000 = ??

example: courtesy Eleanor Birrell

What we will learn this week

1. Binary Representation of program data types ex. 6, -4, ‘a’
   - C data types and sizes, bit, byte, word
   - signed and unsigned representation
2. Operations on binary data
   - Addition and Subtraction on integer types. (e.g.: 6 + 12   15 – 5   -9 + 12)
   - Some other operations on bits
   - Bit shifting, bit-wise OR, AND and NOT

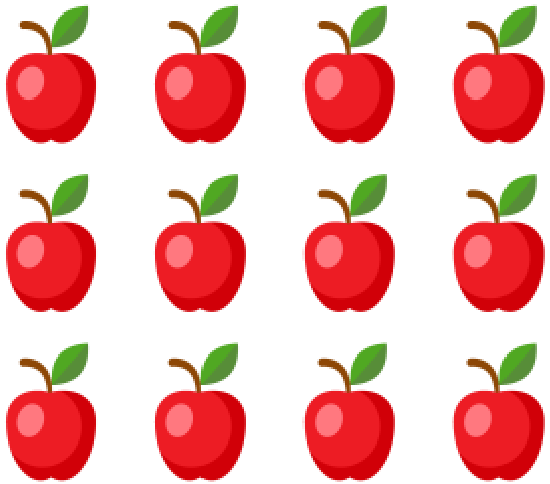
Number Representation

How many apples are there?

A. 12

B. 1100

C. c



Number Representation

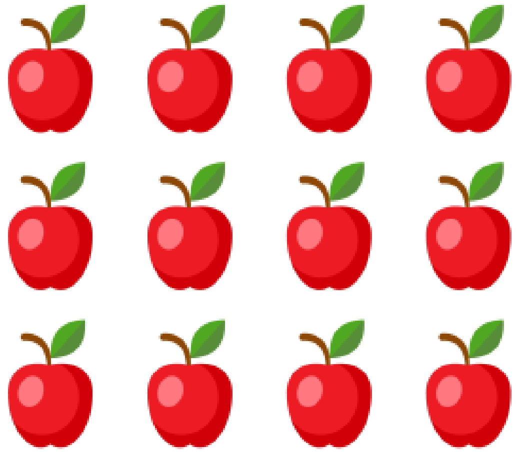
How many apples are there?

A. 12 (decimal, base 10)

B. 0b1100 (binary, base 2)

C. 0xc (hexadecimal, base 16)

D. all of these



We are using different number systems to represent the concept of twelve

- to be clear about which representation:
  - prefix binary with 0b
  - prefix hex with 0x

E.g.: Without a prefix what does “10” refer to?

decimal: 10, binary: 0b10 = 2 hex: 0x10 = 16!

Different Representations

- Binary: base 2 digits [0,1]
- Decimal: base 10 digits [0, 1, …, 9]
- Hexadecimal: base 16 digits [0, …,9,a,b,c,d,e,f]

Relationship between Binary and Hexadecimal: 16 is 2^4

- each hex digit is unique permutation of 4 binary digits

0000: 0	0001:1	0010:2	0011:3	0100:4	0101:5	0110:6	0111:7
1000: 8	1001:9	1010:a	1011:b	1100:c	1101:d	1110:e	1111:f

Why hex? Shorthand for binary that is easier for humans to read

0011111011111010 -> 0011 1110 1111 1010 -> 0x 3 e f a

Positional Notation: Decimal Base 10

A number, written as the sequence of digits

$d_n d_{n-1} \ldots d_2 d_1 d_0$

where d is in {0,1,2,3,4,5,6,7,8,9},

represents the value:

$$[d_n * 10^n] + [d_{n-1} * 10^{n-1}] + \ldots + [d_2 * 10^2] + [d_1 * 10^1] + [d_0 * 10^0]$$

64025 =

$6 * 10^4 \quad + \quad 4 * 10^3 \quad + \quad 0 * 10^2 \quad + \quad 2 * 10^1 \quad + \quad 5 * 10^0$

$60000 \quad + \quad 4000 \quad + \quad 0 \quad + \quad 20 \quad + \quad 5$

Binary: Base 2

Used by computers: Indicated by prefixing number with **0b**

A number, written as the sequence of digits in {0,1}

$$[d_n * 2^n] + [d_{n-1} * 2^{n-1}] + \ldots + [d_2 * 2^2] + [d_1 * 2^1] + [d_0 * 2^0]$$

- 10101: $1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

$= 16 + 0 + 4 + 0 + 1 = 21$

What is the value of 0x1B7 in decimal?

A. 397
B. 409
C. 419
D. 437
E. 439

$[d_n * 16^n] + [d_{n-1} * 16^{n-1}] + ... +$

$[d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$

$16^2 = 256$

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

Converting between Hex and Binary

Hex to binary:

expand each hex digit into its 4 binary digits:

0xa12f: a 1 2 f

1010 0001 0010 1111

0b1010000100101111

Binary to hex:

group into sets of 4 digits

convert each set of 4 to a single hex digit:

0b1001010100001111: 1001 0101 0000 1111

9 5 0 f

0x950f

High-level Takeaway

- You can represent the same value in a variety of number systems / bases.

- It’s **all** stored as binary in the computer.
  - Presence/absence of voltage.

What is the value of 0x1B7 in decimal?

A. 397
B. 409
C. 419
D. 437
E. 439

$[d_n * 16^n] + [d_{n-1} * 16^{n-1}] + ... +$

$[d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$

$16^2 = 256$

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

Converting Decimal -> Binary

- Two methods:
  - division by two remainder
  - powers of two and subtraction

Method 1: decimal value D, binary result b (b_i is ith bit):

Example: Converting 105 from Binary to Decimal

idea: D	example: D = 105	$b_0 = 1$	
D = D/2	D = 52	$b_1 = 0$	
D = D/2	D = 26	$b_2 = 0$	
D = D/2	D = 13	$b_3 = 1$	
D = D/2	D = 6	$b_4 = 0$	
D = D/2	D = 3	$b_5 = 1$	
D = D/2	D = 1	$b_6 = 1$	
D = 0 (done)	D = 0	$b_7 = 0$	

```
i = 0
while (D > 0)
    if D is odd
      set bi to 1
    if D is even
      set bi to 0
    i++
    D = D/2
```

105 = 01101001

Method 2: Subtraction by powers of 2

- $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$

To convert <u>105</u>:

- Find largest power of two that's less than 105 (64)
- Subtract 64 (105 – 64 = <u>41</u>), put a 1 in d_6
- Subtract 32 (41 – 32 = <u>9</u>), put a 1 in d_5
- Skip 16, it's larger than 9, put a 0 in d_4
- Subtract 8 (9 – 8 = <u>1</u>), put a 1 in d_3
- Skip 4 and 2, put a 0 in d_2 and d_1
- Subtract 1 (1 – 1 = <u>0</u>), put a 1 in d_0 (Done)

1	1	0	1	0	0	1
d_6	d_5	d_4	d_3	d_2	d_1	d_0

What is the value of 357 in binary?

8 7654 3210

→ digit position

A. 1 0110 0011
B. 1 0110 0101
C. 1 0110 1001
D. 1 0111 0101
E. 1 1010 0101

$2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16,$

$2^5 = 32, \quad 2^6 = 64, \quad 2^7 = 128, \quad 2^8 = 256$

What is the value of 357 in binary?

8 7654 3210 → digit position

A. 1 0110 0011

B. 1 0110 0101

C. 1 0110 1001

D. 1 0111 0101

E. 1 1010 0101

357 – 256 = 101

101 – 64 = 37

37 – 32 = 5

5 – 4 = 1

$\underline{1}$	$\underline{0}$	$\underline{1}$	$\underline{1}$	$\underline{0}$	$\underline{0}$	$\underline{1}$	$\underline{0}$	$\underline{1}$
d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

$2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$,

$2^5 = 32$, $2^6 = 64$, $2^7 = 128$, $2^8 = 256$

So far: Unsigned Integers

With N bits, can represent values: 0 to 2^n-1

We can always add 0's to the front of a number without changing it:

10110= 010110 = 00010110 = 0000010110

So far: Unsigned Integers

With N bits, can represent values: 0 to 2^n-1

- `1 byte:  char, unsigned char`
- `2 bytes: short, unsigned short`
- `4 bytes: int, unsigned int, float`
- `8 bytes: long long, unsigned long long, double`
- `4 or 8 bytes: long, unsigned long`

Unsigned Integers

- Suppose we had <u>one byte</u>
  - Can represent 2^8 (256) values
  - If unsigned (strictly non-negative): 0 – 255

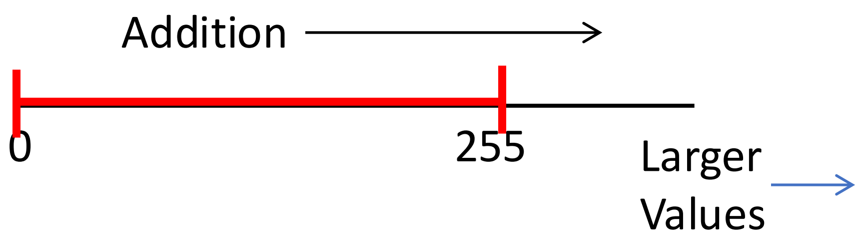
252 = 11111100

253 = 11111101

254 = 11111110

255 = 11111111

Traditional number line:



Unsigned Integers

Suppose we had one byte

- Can represent 2^8 (256) values
- If unsigned (strictly non-negative): 0 – 255

252 = 11111100

253 = 11111101

254 = 11111110

255 = 11111111

What if we add one more?

Car odometer “rolls over”.

Any time we are dealing with a finite storage space we cannot represent an infinite number of values!

Unsigned Integers

Suppose we had one byte

- Can represent 2^8 (256) values
- If unsigned (strictly non-negative): 0 – 255

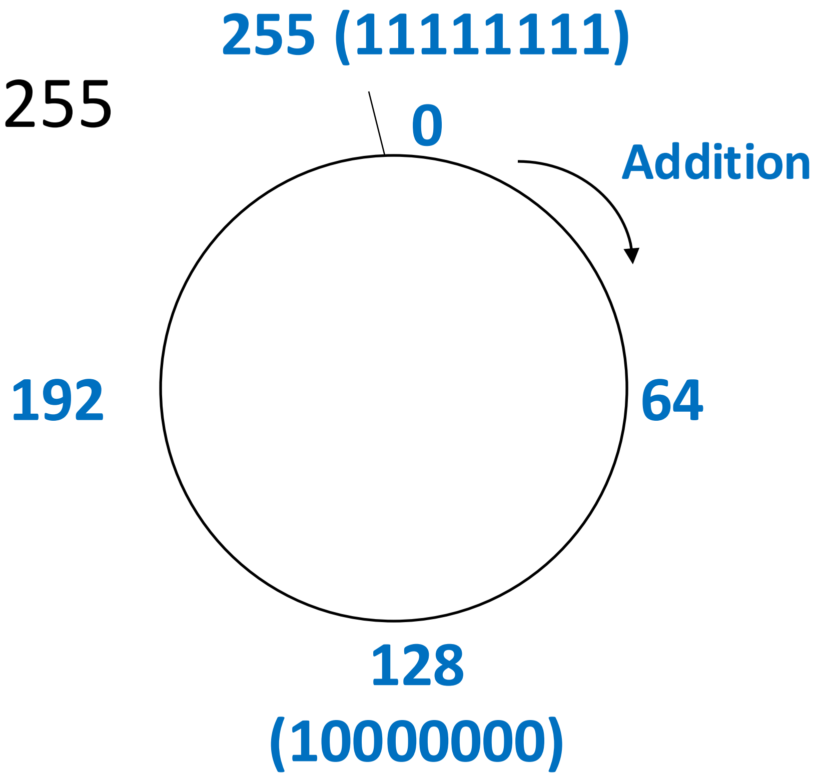
252 = 11111100

253 = 11111101

254 = 11111110

255 = 11111111

<u>What if we add one more?</u>



Modular arithmetic: Here, all values are modulo 256.

Unsigned Addition (4-bit)

Works just like grade school addition

1. Add corresponding digits, starting with d_0 digits
2. Carry if result is greater than or equal to the base

Let's try (6 + 4) in unsigned binary

$6 = 0110 \quad (2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0)$

$4 = 0100 \quad (2^3 \times 0 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 0)$

Unsigned Addition (4-bit)

Works just like grade school addition

1. Add corresponding digits, starting with d_0 digits
2. Carry if result is greater than or equal to the base

Let's try (6 + 4) in unsigned binary

```
  0b0110        6
+0b0100       +4
```

in binary 1+1 = 10
"0 carry the 1" out

Unsigned Addition (4-bit)

Works just like grade school addition

1. Add corresponding digits, starting with d_0 digits
2. Carry if result is greater than or equal to the base

Let's try (6 + 4) in unsigned binary

```
   1
  0b0110      6
 +0b0100     +4
  0b1010     10
```

in binary 1+1 = 10
"0 carry the 1" out

Unsigned Addition (4-bit)

- Addition works like grade school addition:

```
   1
   0110       6          1100     12
+  0100    +  4       +  1010    +10
   1010      10       1 0110       6
^no carry out         ^carry out
```

in binary 1+1 = 10
“0 carry the 1” out

Overflow!

Four bits give us range: 0 - 15

Carry out is indicative of something having gone wrong when adding unsigned values

Let's try some more examples (note down if you get a carry out)

```
   0100      4          1111     15
+  0100   +  4       +  0001   +  1
```

in binary 1+1 = 10
"0 carry the 1" out

<u>Carry out is indicative of something having gone wrong when adding unsigned values</u>

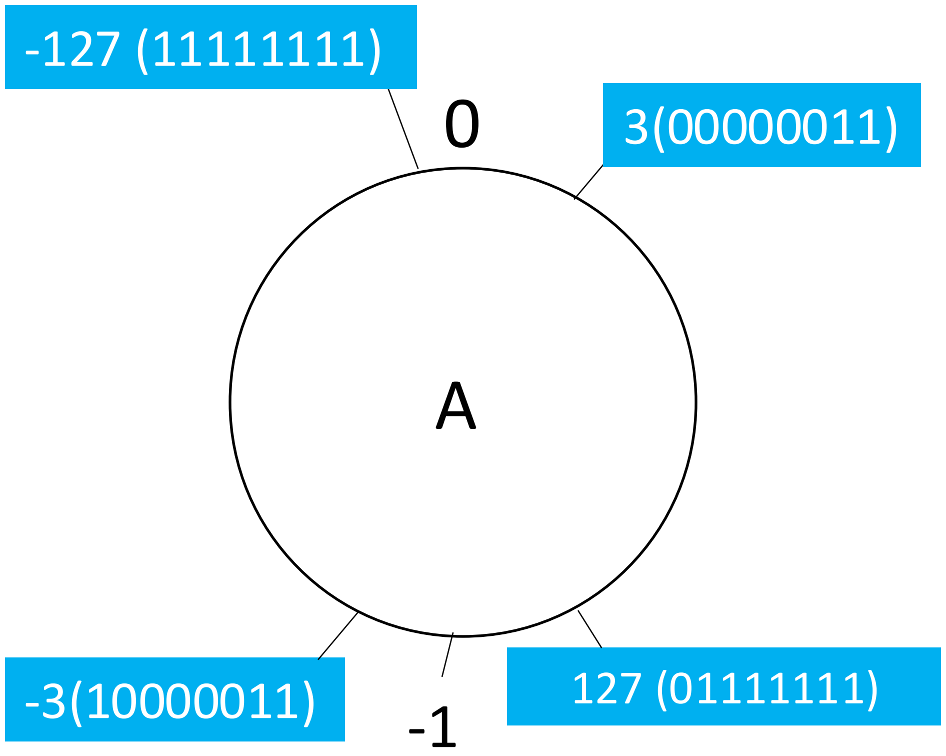
Let's try some more examples (note down if you get a carry out)

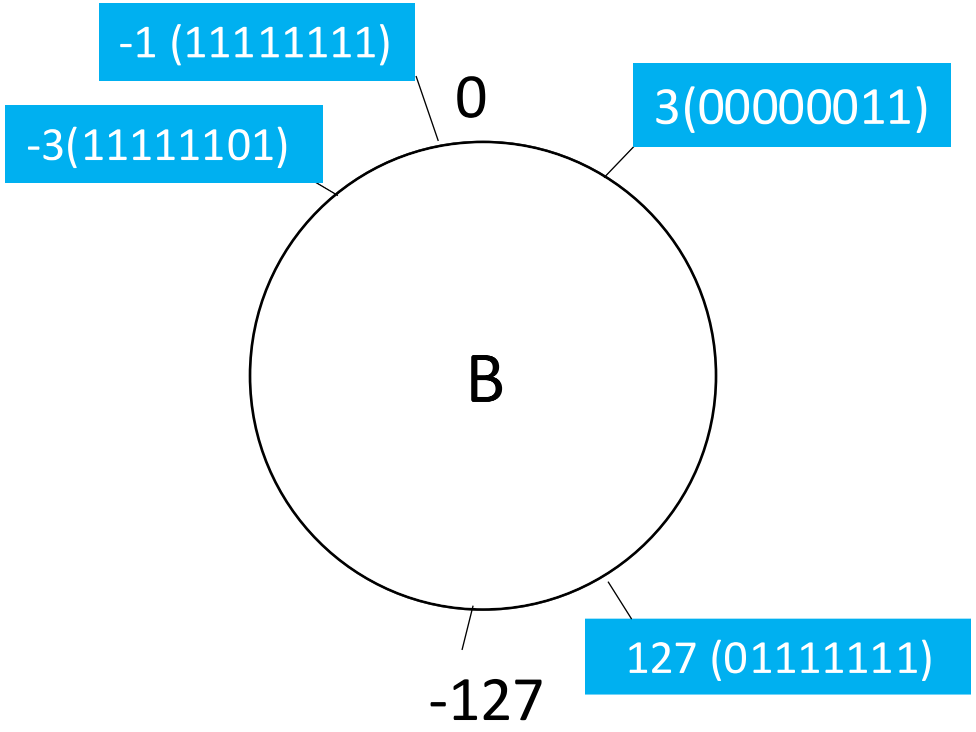
```
   0100      4         1111     15
+  0100   +  4      +  0001   +  1
--------  -----     -------   ----
0  1000      8      1  0000      0!
^no carry out       ^carry out
```

in binary 1+1 = 10
"0 carry the 1" out

<u>Carry out is indicative of something having gone wrong when adding unsigned values</u>

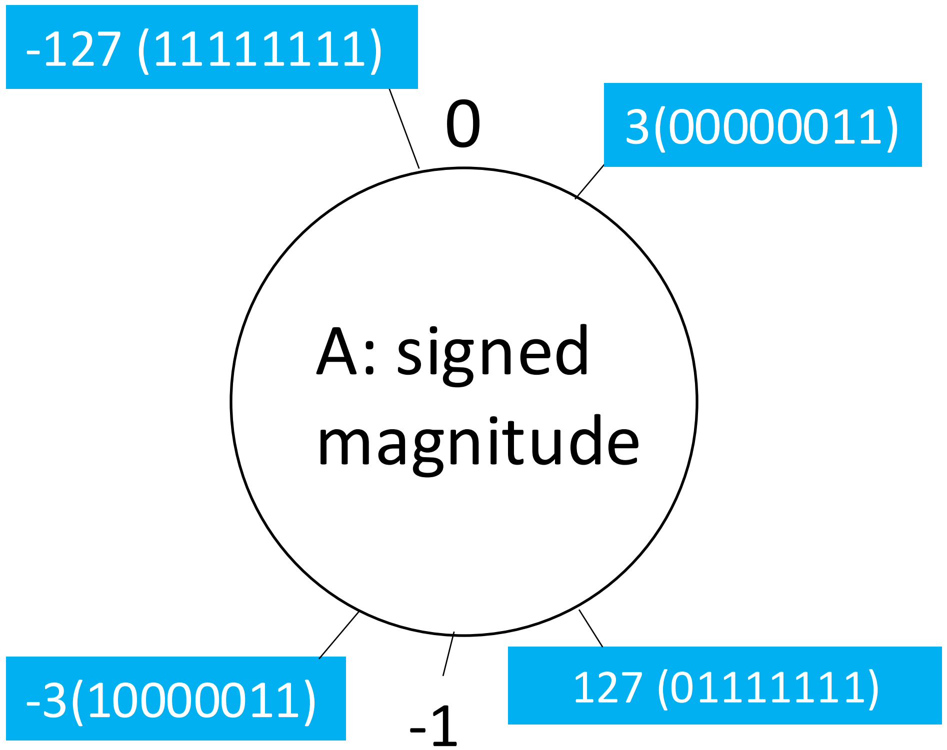
Suppose we want to support signed values (positive and negative) in 8 bits, where should we put -1 and -127 on the circle? Why?

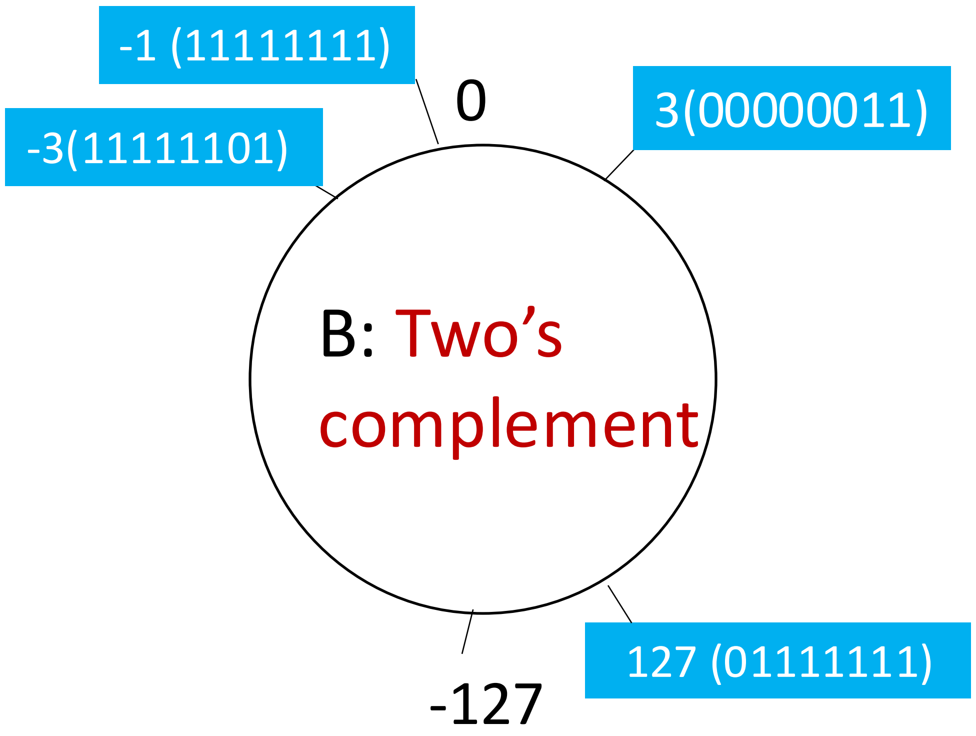




C: Put them somewhere else.

Suppose we want to support signed values (positive and negative) in 8 bits, where should we put -1 and -127 on the circle? Why?





C: Put them somewhere else.

Two's Complement Representation (for four bit values)

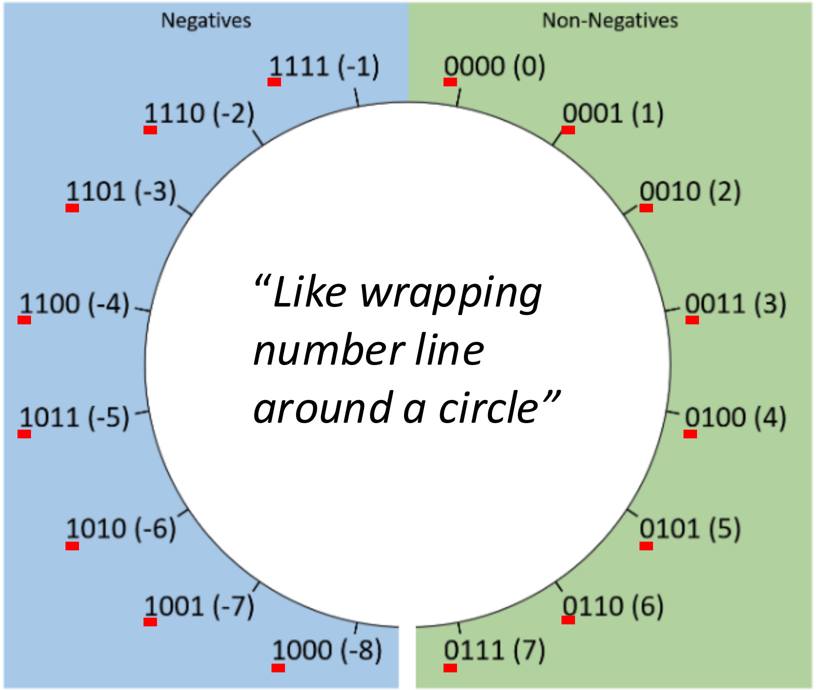
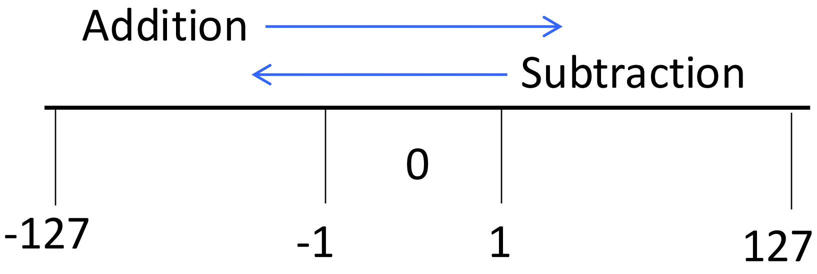


Figure 2. A logical layout of two's complement values for bit sequences of length four.

For an 8 bit range we can express 256 unique values:

- 128 non-negative values (0 to 127)
- 128 negative values (-1 to -128)

Borrow nice property from number line:



Only one instance of zero!
Implies: -1 and 1 on either side of it.

- Addition moves to the right
- Subtraction moves to the left.

Two's Complement

- Only one value for zero
- Adding positive and negative just like unsigned addition

```
  11111111     (-1)
+ 00000001     (1)
  --------
  00000000     (0)
```

- With N bits, can represent the range:
  - -2^{N-1} to $2^{N-1} - 1$

- Most significant (first) bit still designates positive (0) /negative (1)

- Negating a value is slightly more complicated:

  1 = 00000001,   -1 = 11111111

From now on, unless we explicitly say otherwise, we'll assume all integers are stored using two's complement! This is the standard!

Two's Compliment

Each two's compliment number is now:

$$[-2^{n-1} * d_{n-1}] \quad + \quad [2^{n-2} * d_{n-2}] \quad + \ldots + \quad [2^{1} * d_{1}] \quad + \quad [2^{0} * d_{0}]$$

Note the negative sign on just on the higher-order bit.
High-order bit is the sign-bit: encodes if number is negative or positive

(The other digits are unchanged and carry the same meaning as unsigned.)

If we interpret 11001 as a two's complement number, what is the value in decimal?

Each two's compliment number is now:

$$[-2^{n-1} * d_{n-1}] \quad + \quad [2^{n-2} * d_{n-2}] \quad + \ldots + \quad [2^1 * d_1] \quad + \quad [2^0 * d_0]$$

A. -2

B. -7

C. -9

D. -25

If we interpret 11001 as a two's complement number, what is the value in decimal?

Each two's compliment number is now:

$$[-2^{n-1} * d_{n-1}] \quad + \quad [2^{n-2} * d_{n-2}] \quad + \ldots + \quad [2^{1} * d_{1}] \quad + \quad [2^{0} * d_{0}]$$

A. -2

B. -7 $\quad -16 + 8 + 1 = -7$

C. -9

D. -25

“If we interpret…”

- What is the decimal value of 1100?

- …as unsigned, 4-bit value: 12 (%u)
- …as signed (two’s complement), 4-bit value: -4 (%d)

- …as an 8-bit value: 12
  (i.e., **0000**1100)

Let's try some more examples

High order bit is the sign bit, otherwise just like unsigned conversion. 4-bit and 8-bit examples:

4 bit numbers (4th bit is the sign bit)

0110

1110

8 bit numbers (8th bit is the sign bit)

00001010

11111111

Let's try some more examples

High order bit is the sign bit, otherwise just like unsigned conversion. 4-bit and 8-bit examples:

4 bit numbers (4th bit is the sign bit)

0110 = $-2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0 = 6$

1110 = $-2^3 \times 1 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0 = -2$

8 bit numbers (8th bit is the sign bit)

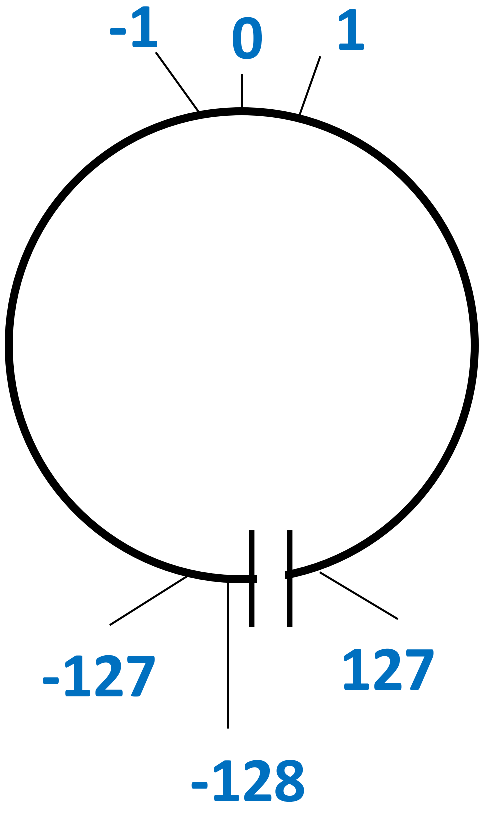
00001010= $-2^7 \times 0 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 0$
$+2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 = 10$

11111111= $-2^7 \times 1 + 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1$
$+2^3 \times 1 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1 = -1$

Two's Complement Negation

- To negate a value x, we want to find y such that $x + y = 0$.

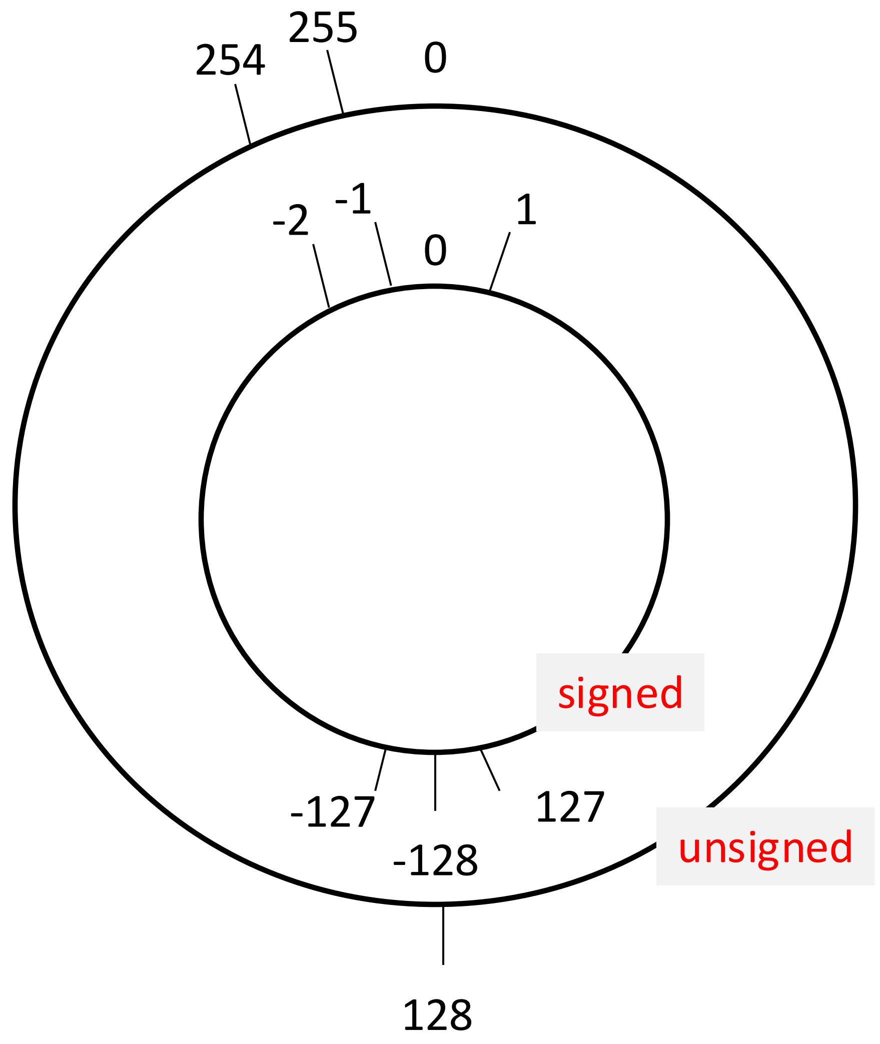
- For N bits, $y = 2^N - x$



Negation Example (8 bits)

- For N bits, $y = 2^N - x$
- Negate 00000010 (2)
  - $2^8 - 2 = 256 - 2 = 254$

- Our wheel only goes to 127!
  - Put -2 where 254 would be if wheel was unsigned.
  - 254 in binary is 11111110



Given 11111110, it’s 254 if interpreted as <u>unsigned</u> and -2 interpreted as <u>signed</u>.

Negation Shortcut

- A much easier, faster way to negate:
  - Flip the bits (0's become 1's, 1's become 0's)
  - Add 1

- Negate 00101110 (46)
  - 2^8 - 46 = 256 - 46 = 210
  - 210 in binary is 11010010

46: 00101110

Flip the bits: 11010001

Add 1

+ 1

-46: 11010010

Negation Two Ways

4-bit Examples			
x	-x	2^4 - x	Bit flip + 1
0000	0000	10000 – 0000 = 0000	1111 + 1 = 0000
0001	1111	10000 – 0001 = 1111	1110 + 1 = 1111
0010	110	10000 – 0010 = 1110	1101 + 1 = 1110
0111	1001	10000 – 0111 = 1001	1000 + 1 = 1001

Decimal to Two's Complement with 8-bit values

(high-order bit is the sign bit)

For positive values, use same algorithm as unsigned

```
For example, 6:   6 - 4 = 2 (4:2^2)
                2 – 2 = 0 (2:2^1):  00000110
```

For negative values:

1. convert the equivalent positive value to binary
2. then negate binary to get the negative representation

```
For example, -3:
          3: 00000011
     negate: 11111100+1 = 11111101 = -3
```

What is the 8-bit, two's complement representation for -7?

For negative values:

1. convert the equivalent positive value to binary
2. then negate binary to get the negative representation

A. 11111001
B. 00000111
C. 11111000
D. 11110011

What is the 8-bit, two's complement representation for -7?

For negative values:

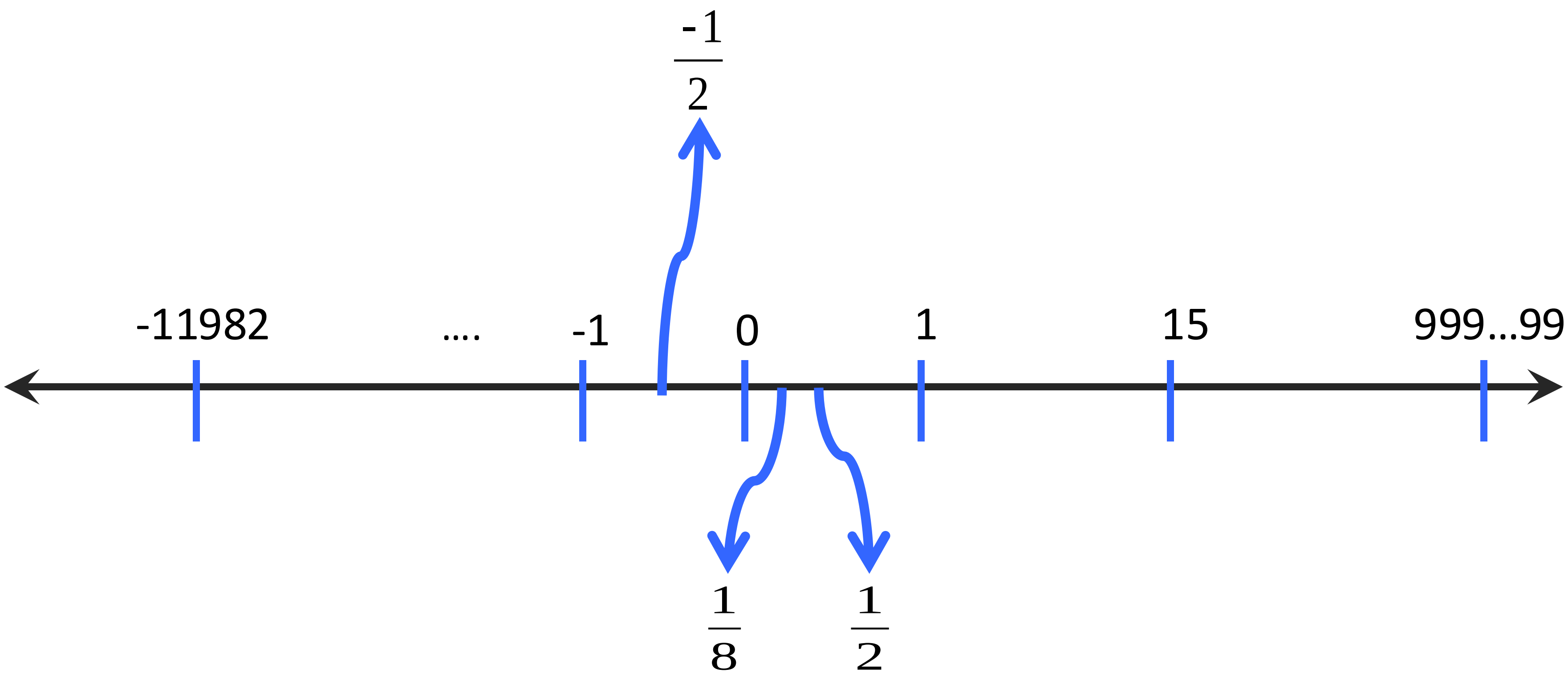
1. convert the equivalent positive value to binary
2. then negate binary to get the negative representation

A. 11111001
B. 00000111
C. 11111000
D. 11110011

-7 = (1) 7: 00000111
(2) negate: 11111000 + 1 = 11111001

Additional Info: Fractional binary numbers

How do we represent fractions in binary?



Additional Info: Floating Point Representation

1 bit for sign   sign | exponent | fraction |

8 bits for exponent

23 bits for precision

$$value = (-1)^{sign} * 1.fraction * 2^{(exponent-127)}$$

let's just plug in some values and try it out

```
0x40ac49ba: 0 10000001   01011000100100110111010
     sign = 0 exp = 129   fraction = 2902458
```

$$= 1*1.2902458*2^{2} = 5.16098$$

<u>You are not expected to memorize this</u>

Summary

- Images, Word Documents, Code, and Video can represented in bits.
- Byte or 8 bits is the smallest addressable unit
- N bits can represent 2^N unique values
- A number is written as a sequence of digits: in the decimal base system
  - [dn * 10 ^ n] + [dn-1 * 10 ^ n-1] + ... + [d2 * 10 ^ 2] + [d1 * 10 ^ 1] + [d0 * 10 ^ 0]
  - For any base system:
  - [dn * b ^ n] + [dn-1 * b ^ n-1] + ... + [d2 * b ^ 2] + [d1 * b ^ 1] + [d0 * b ^ 0]
- Hexadecimal values (represent 16 values): {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
  - Each hexadecimal value can be represented by 4 bits. (2^4=16)
- A finite storage space we cannot represent an infinite number of values. For e.g., the max unsigned 8 bit value is 255.
  - Trying to represent a value >255 will result in an overflow.
- Two’s Complement Representation: 128 non-negative values (0 to 127), and 128 negative values (-1 to -128).

Aside: Signed Magnitude Representation (for 4 bit values)

- One bit (usually left-most) signals:
  - 0 for positive
  - 1 for negative

For one byte:

1 = 00000001, -1 = 10000001

Pros: Negation (negative value of a number) is very simple!

For one byte:

0 = 00000000

What about 10000000?

Major con: Two ways to represent zero!