CS 31: Introduction to Computer Systems

02: Binary Representation

January 23

SWARTHMORE COLLEGE

Announcements

- Sign up for Piazza!
- Let me know about exam conflicts!
- Register your clicker!
- Faculty Talk Tomorrow on Newtorks & Security: 11.30 – 12.30pm (Free Pizza!)

Reading Quiz

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

Today

- Number systems and conversion

- Data types and storage:
  - Sizes
  - Representation
  - Signedness

Abstraction

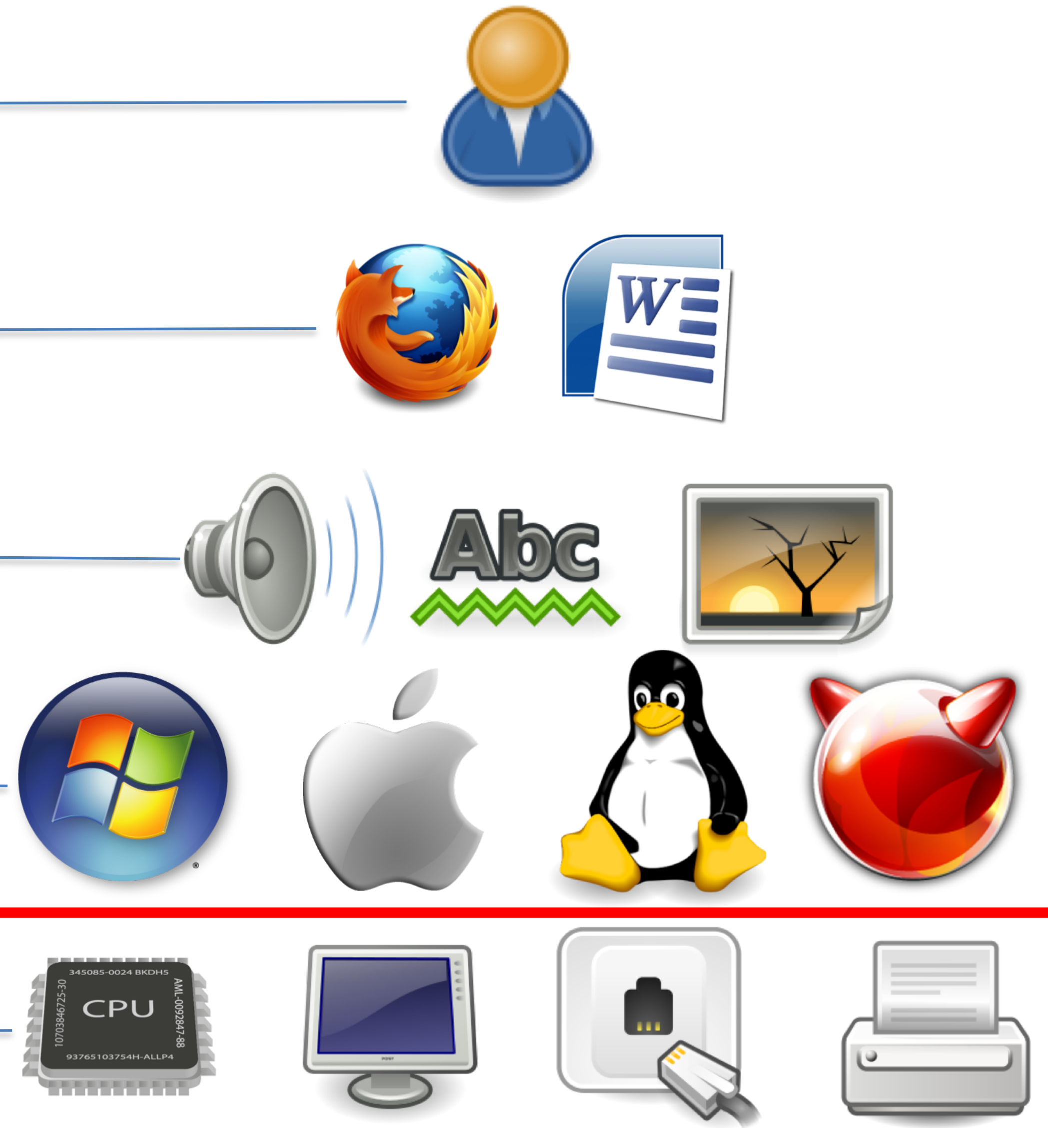
User / Programmer
Wants low complexity

Applications
Specific functionality

Software library
Reusable functionality

Operating system
Manage resources

Complex devices
Compute & I/O



Bits and Bytes

- Bit: a 0 or 1 value (binary)
  - HW represents as two different voltages
  - 1: the presence of voltage (high voltage)
  - 0: the absence of voltage (low voltage)

- Byte: 8 bits, the smallest addressable unit

  Memory: 01010101 10101010 00001111 …

Binary Digits: (BITs)

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

Files

Sequence of bytes… nothing more, nothing less

How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.

How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. <u>512</u>
E. Some other number of values.

How many values?

1 bit: 0 1

2 bits: 0 0 0 1 1 0 1 1

3 bits: 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1

4 bits: 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 16 values

0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1

1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1

1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1

N bits: 2^N values

Let's start with what we know

- Digits 0-9
- <u>Positional numbering</u>
- Digits are composed to make larger numbers
- Known as the <u>Base 10</u> representation

Decimal number system (Base 10)

Sequence of digits in range [0, 9]

6 4 0 2 5

Digit #0: 1's place (least significant digit)

Digit #1: 10's place

⋮

And so on

What is the significance of the N^{th} digit number in this number system? What does it contribute to the overall value?

64025

Digit #4: d_4 Digit #0: d_0

A. $d_N * 1$
B. $d_N * 10$
C. $d_N * 10^N$
D. $d_N * N^{10}$
E. $d_N * 10^{d_N}$

What is the significance of the N^{th} digit number in this number system? What does it contribute to the overall value?

64025

Digit #4: d_4 Digit #0: d_0

A. $d_N * 1$
B. $d_N * 10$
C. $d_N * 10^N$
D. $d_N * N^{10}$
E. $d_N * 10^{d_N}$

Consider the meaning of d_3 (the value 4) above. What is it contributing to the total value?

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

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 =

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

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

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 + 4000 + 0 + 20 + 5

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 =

$6 * 10^4$	+	$4 * 10^3$	+	$0 * 10^2$	+	$2 * 10^1$	+	$5 * 10^0$
60000	+	4000	+	0	+	20	+	5

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 + 4000 + 0 + 20 + 5

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 + 4000 + 0 + 20 + 5

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 + 4000 + 0 + 20 + 5

Generalizing

The meaning of a digit depends on its position in a number.

A number, written as the sequence of digits

$d_n d_{n-1} \ldots d_2 d_1 d_0$ in base b represents the value:

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

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

Binary: Base 2

Used by computers to store digital values.

- Indicated by prefixing number with **0b**

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}, represents the value:

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

What is the value of 0b110101 in decimal?

- 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}, represents the value:

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

A. 26
B. 53
C. 61
D. 106
E. 128

What is the value of 0b110101 in decimal?

- 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}, represents the value:

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

A. 26
B. <u>53</u>
C. 61
D. 106
E. 128

Binary Digits: (BITS)

7 6 5 4 3 2 1 0

Most significant bit → <u>1</u>000111<u>1</u> ← Least significant bit

Representation: $1 \times 2^7 + 0 \times 2^6$ ..... $+ 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

10001111 = 143

Other (common) number systems.

- Base 10: decimal
- Base 2: binary

- Base 16: hexadecimal
- Base 8: octal
- Base 64

Hexadecimal: Base 16

Indicated by prefixing number with **0x**

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,A,B,C,D,E,F},

represents the value:

$$[d_n * 16^n] + [d_{n-1} * 16^{n-1}] + \ldots + [d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$$

What is the value of 0x1B7 in decimal?

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

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

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

$16^2 = 256$

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

What is the value of 0x1B7 in decimal?

A. 397
B. 409
C. 419
D. 437
E. <u>439</u>

$$[d_n * 16^n] + [d_{n-1} * 16^{n-1}] + \ldots +$$
$$[d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$$

$16^2 = 256$

$1*16^2 + 11*16^1 + 7*16^0 = 439$

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

Important Point…

- 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.

Common number systems.

- Base 2: How data is stored in hardware.
- Base 10: Preferred by people.
- Base 8: Used to represent file permissions.
- Base 16: Convenient for representing memory addresses.
- Base 64: Commonly used on the Internet, (e.g. email attachments).

Different ways of visualizing the same information!

Hexadecimal: Base 16

- Fewer digits to represent same value
  - Same amount of information!
- Like binary, base is power of 2
- Each digit is a “nibble”, or half a byte.

Each hex digit is a "nibble"

- One hex digit: 16 possible values (0-9, A-F)
- $16 = 2^4$, so each hex digit has exactly four bits worth of information.
- We can map each hex digit to a four-bit binary value. (helps for converting between bases)

Each hex digit is a "nibble"

Example value: 0x1B7

Four-bit value: 1

Four-bit value: B (decimal 11)

Four-bit value: 7

In binary:	0001	1011	0111
	1	B	7

Hexadecimal Representation

- Bit patterns as base-16 numbers
- Convert binary to hexadecimal: by splitting into groups of 4 bits each.

Example:

$$11\ 1100\ 1010\ 1101\ 1011\ 0011_2 = 3CADB3_{16}$$

Bin	11	1100	1010	1101	1011	0011
Hex	3	C	A	D	B	3

Converting Decimal -> Binary

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

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

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

Example: Converting 105

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

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

Example: Converting 105

`example:  D = 105`

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

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

Example: Converting 105

```
example:  D = 105        b0 = 1
```

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

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

Example: Converting 105

```
example:  D = 105        b0 = 1
  D/2     D =  52        b1 = 0
```

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

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

Example: Converting 105

```
example:  D = 105       b0 = 1
  D/2     D =  52       b1 = 0
  D/2     D =  26       b2 = 0
  D/2     D =  13       b3 = 1
```

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

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

Example: Converting 105

```
example:  D = 105        b0 = 1
  D/2     D =  52        b1 = 0
  D/2     D =  26        b2 = 0
  D/2     D =  13        b3 = 1
  D/2     D =   6        b4 = 0
  D/2     D =   3        b5 = 1
```

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

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

Example: Converting 105

```
example:  D = 105      b0 = 1
  D/2     D =  52      b1 = 0
  D/2     D =  26      b2 = 0
  D/2     D =  13      b3 = 1
  D/2     D =   6      b4 = 0
  D/2     D =   3      b5 = 1
  D/2     D =   1      b6 = 1
  D/2     D =   0      b7 = 0
```

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

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

Example: Converting 105

```
example:  D = 105        b0 = 1
  D/2     D =  52        b1 = 0
  D/2     D =  26        b2 = 0
  D/2     D =  13        b3 = 1
  D/2     D =   6        b4 = 0
  D/2     D =   3        b5 = 1
  D/2     D =   1        b6 = 1
  D/2     D =   0        b7 = 0

          105   =   01101001
```

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

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

What is the value of 357 in binary?

A. 101100011
B. 101100101
C. 101101001
D. 101110101
E. 110100101

$2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$,
$2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$,
$2^8 = 256$

What is the value of 357 in binary?

A. 101100011
B. <u>101100101</u>
C. 101101001
D. 101110101
E. 110100101

357 – 256 = 101
101 – 64 = 37
37 – 32 = 5
5 – 4 = 1

1	0	1	1	0	0	1	0	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

<u>With N bits, can represent values: 0 to 2^n-1</u>

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

10110= <u>0</u>10110 = <u>00</u>010110 = <u>0000</u>010110

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

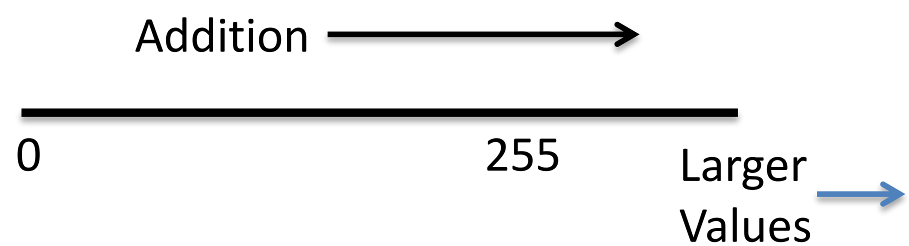
252 = 11111100

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

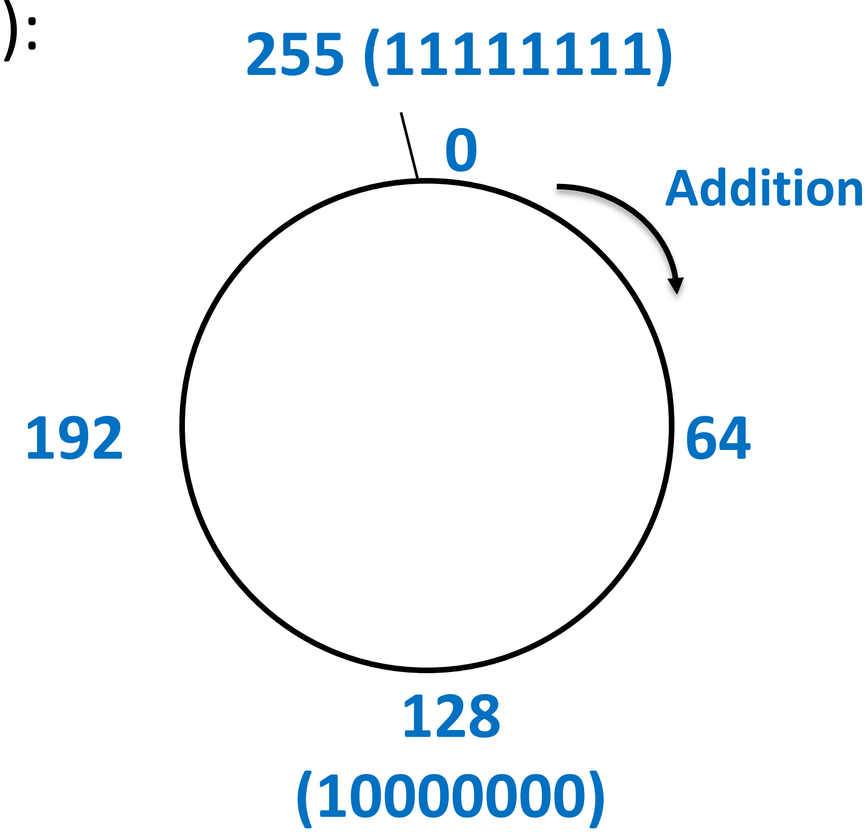
  252 = 11111100

  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)

- Addition works like grade school addition:

```
   1
   0110      6
+  0100   +  4
--------  -----
   1010     10
```

Four bits give us range: 0 - 15

Unsigned Addition (4-bit)

- Addition works like grade school addition:

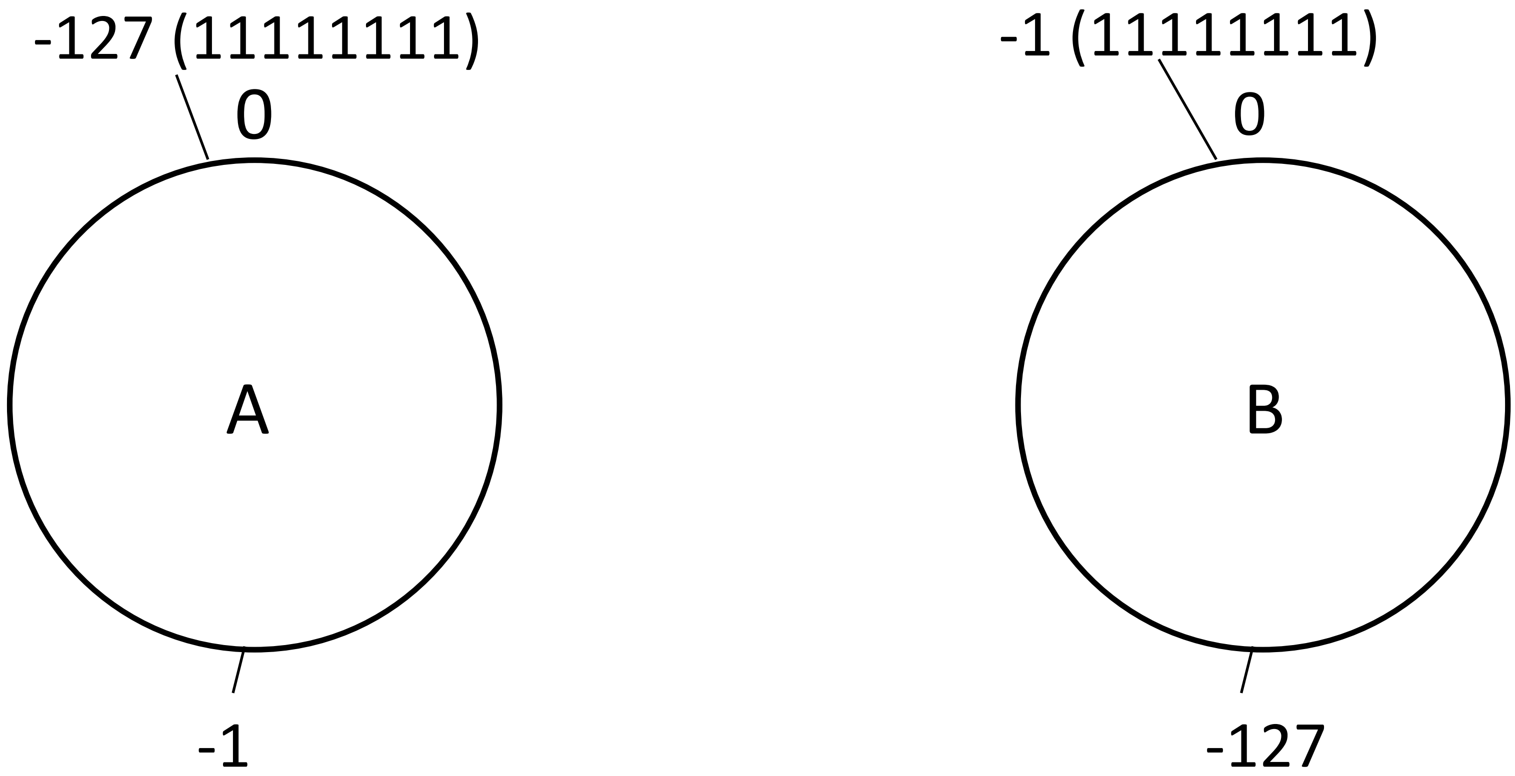
```
   1
   0110      6          1100     12
+  0100    + 4        + 1010    +10
--------   ----       --------  ----
   1010     10        1 0110      6
^no carry out         ^carry out
```

Four bits give us range: 0 - 15

Overflow!

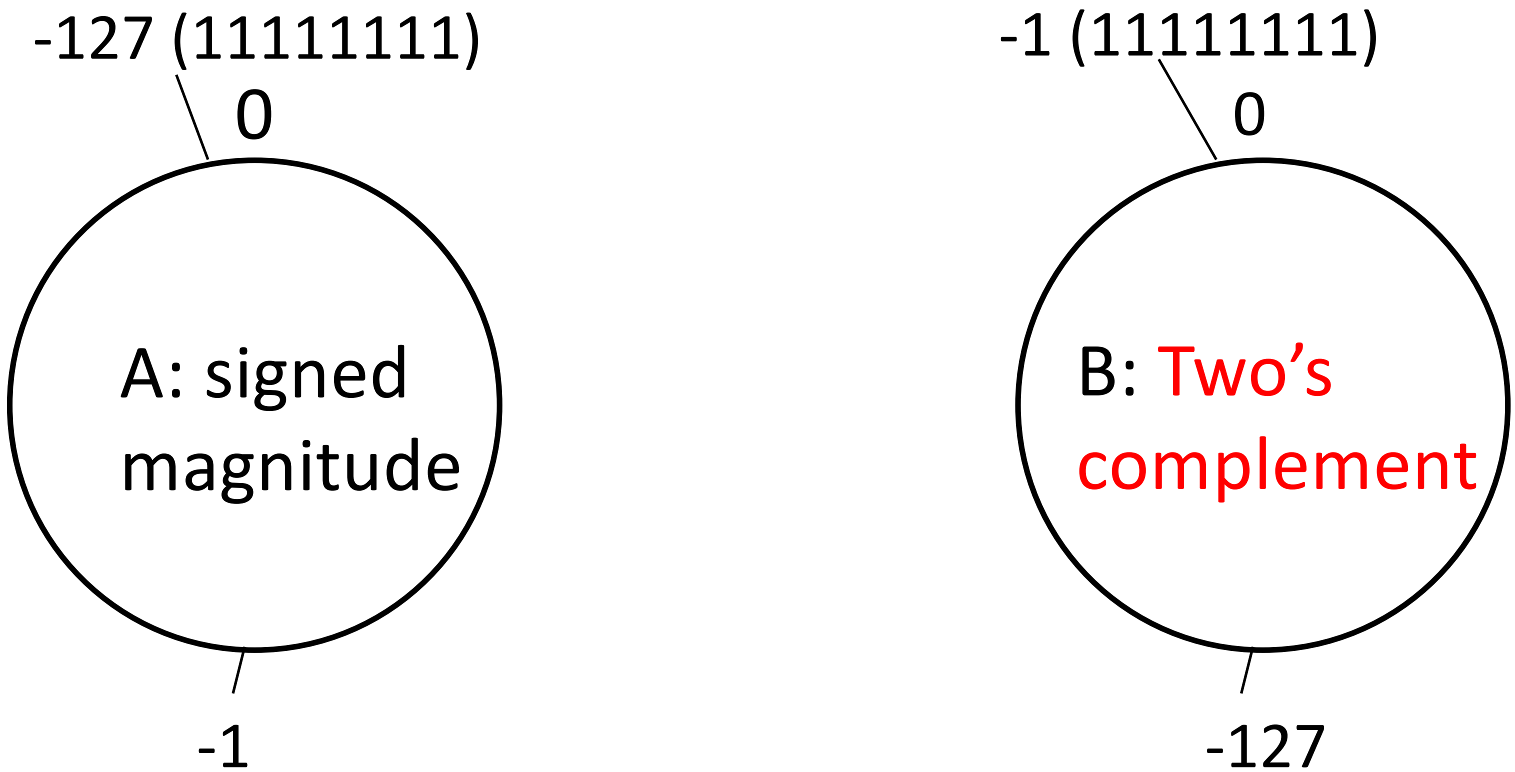
Carry out is indicative of something having gone wrong when adding unsigned values

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.

Signed Magnitude Representation (for 4 bit values)

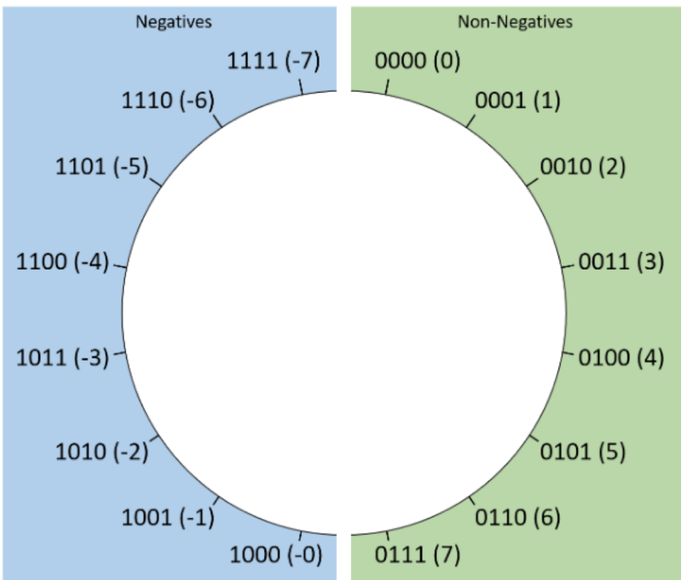


Figure 1. A logical layout of signed magnitude values for bit sequences of length four.

- 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.

Two's Complement Representation
(for four bit values)

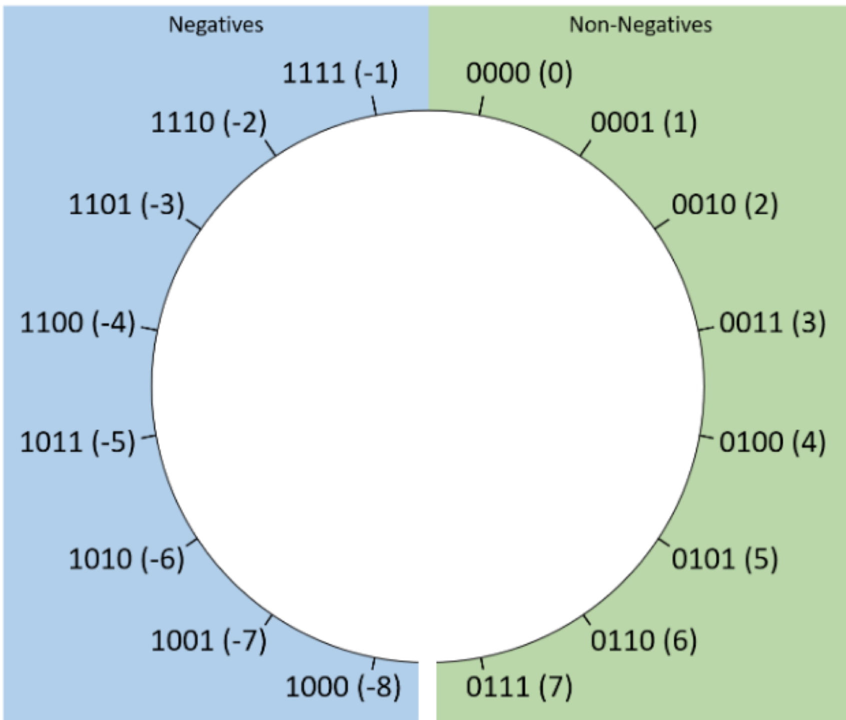
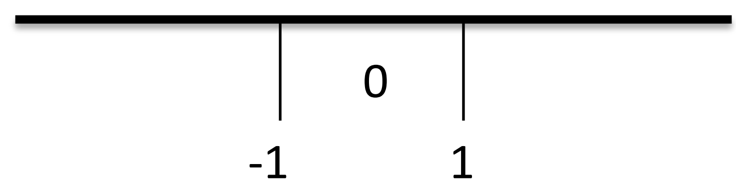


Figure 2. A logical layout of two's complement values for bit sequences of length four.

- Borrow nice property from number line:



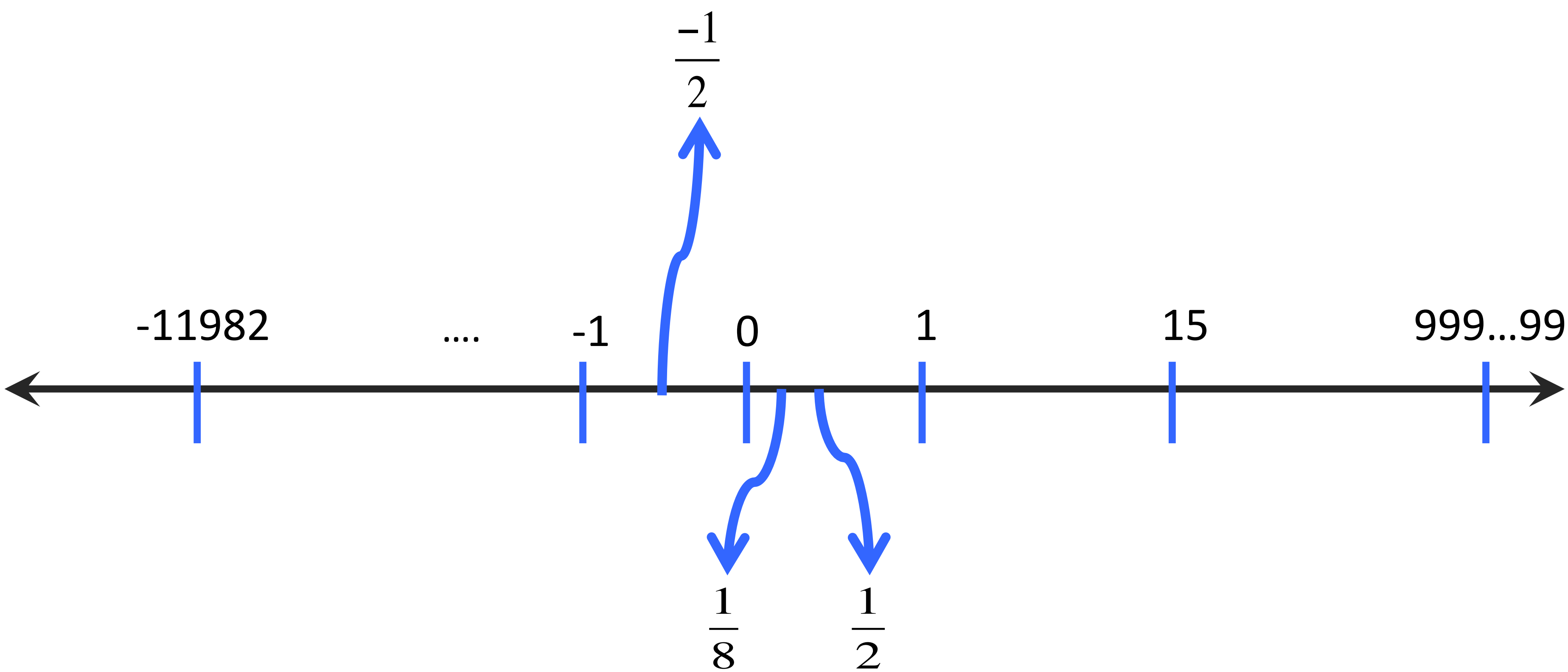
Only one instance of zero!
Implies: -1 and 1 on either side of it.

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

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

Additional Info: Fractional binary numbers

How do we represent fractions in binary?



Additional Info: Representing Signed Float Values

- One option (used for floats, <u>NOT integers</u>)
  - Let the first bit represent the sign
  - 0 means positive
  - 1 means negative

- For example:
  - <u>0</u>101 -> 5
  - <u>1</u>101 -> -5

- Problem with this scheme?

Additional Info: Floating Point Representation

1 bit for sign sign | exponent | fraction |
8 bits for exponent
23 bits for precision

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

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>I don’t expect you 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).