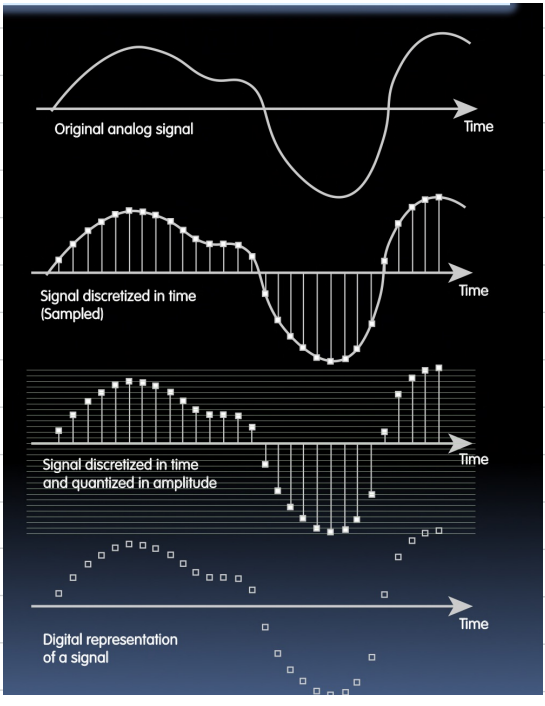
# Number Representation

- Real world is analog!
- To import analog information, we must do two things
  - Sample
  - Quantize




Binary Decimal Hex

## Numeral

A symbol or name that stands for a number

*e.g., 4 , four , quatro , IV , IIII , …*

…and **Digits** are symbols that make numerals

## Number

The "idea" in our minds…there is only ONE of these

*e.g., the concept of "4"*

## Base 10 (Ten) #s, Decimals

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Example:

$3271 = 3271_{10} = (3\times10^3 + 2\times10^2 + 7\times10^1 + 1\times10^0)$

## Base 2 (Two) #s, Binary (to Decimal)

Digits: 0, 1 (binary digits → bits)

Example: "1101" in binary? ("0b1101")

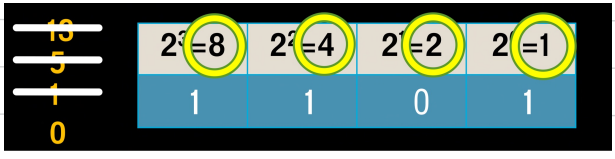
$1101_2 = (1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0)$

$= 8 + 4 + 0 + 1$

$= 13$

## Convert from Decimal to Binary

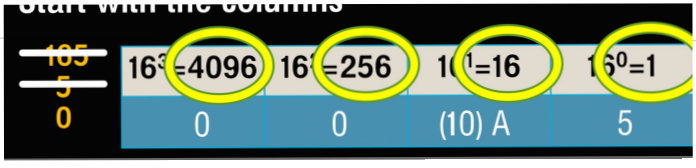
- E.g., 13 to binary
- Start with the column




- Left to right, is (column) ≤ number n?
  - If yes, put how many of that column fit in n, subtract col* that many from n, keeping going.
  - If not, put 0 and keep going. (and stop at 0)

## Convert from Decimal to Hexadecimal

- E.g., 165 to hexadecimal?
- Start with the columns



- Left to right, is (column) ≤ number n?
  - If yes, put how many of that column fit in n, subtract col* that many from n, keeping going.
  - If not, put 0 and keep going. (and stop at 0)

## Convert Binary ⟷ Hexadecimal

- Binary → Hex
  - Always left-pad with 0s to make full 4-bit values, then look up!
  - E.g. 0b11110 to Hex?

- 0b11110 → 0b00011110
- Then look up: 0x1E

- Hex → Binary
  - Just look up, drop leading 0s

- 0x1E→0b00011110→0b11110

| D | H | B |
|---|---|---|
| 00 | 0 | 0000 |
| 01 | 1 | 0001 |
| 02 | 2 | 0010 |
| 03 | 3 | 0011 |
| 04 | 4 | 0100 |
| 05 | 5 | 0101 |
| 06 | 6 | 0110 |
| 07 | 7 | 0111 |
| 08 | 8 | 1000 |
| 09 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

## Decimal vs Hexadecimal vs Binary

- 4 Bits
  - 1 "Nibble"
  - 1 Hex Digit = 16 things
- 8 Bits
  - 1 "Byte"
  - 2 Hex Digits = 256 things
  - Color is usually

0-255 Red,
0-255 Green,
0-255 Blue.
#D0367F=

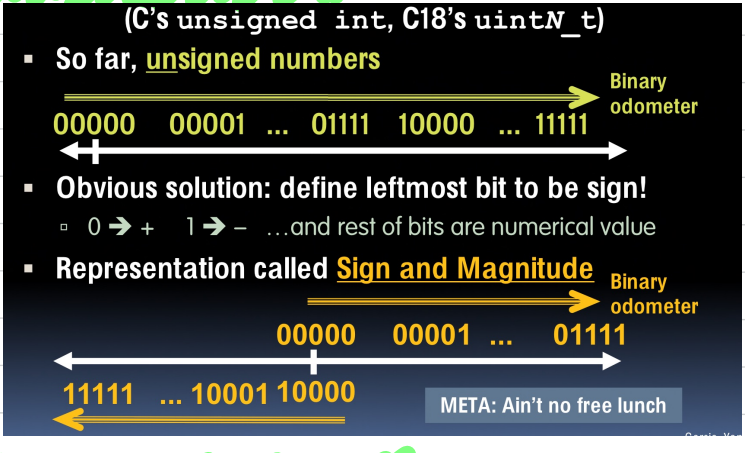
# Number Representations

- add, subtract, multiply, divide, compare

## What if too big?

- Binary bit patterns are simply representatives of numbers. Abstraction!
  - Strictly speaking they are called "numerals"
- Numerals really have an ∞ number of digits
  - with almost all being some [00...0 or 11...1] except for a few of the rightmost digits
- If result of add (or -, *, /) cannot be represented by these rightmost HW bits, we say overflow occurred

## How to Represent Negative Numbers?




## Shortcoming of Sign and Magnitude

- Arithmetic circuit complicated
  - Special steps depending on if signs are the same or not
- two zeros
- Incrementing "binary odometer", sometimes increases values, and sometimes decreases!
- Therefore sign and magnitude used only in signal processors

## Another try: complement the bits

- Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
- Called One's Complement
- Note: positive numbers have leading 0s, negative numbers have leadings 1s.


- What is -00000 ? Answer: 11111
- How many positive numbers in N bits?
- How many negative numbers?

## Shortcomings of One's Complement

- Arithmetic still somewhat complicated
- Still two zeros
  - 0x00000000 = $+0_{ten}$
  - 0xFFFFFFFF = $-0_{ten}$
- Although used for a while on some computer products, one's complement was eventually abandoned because another solution was better.

# Two's Complement & Bias Encoding

- **Problem is the negative mappings "overlap" with the positive ones (the two 0s). Want to shift the negative mappings left by one.**
  - Solution! For negative numbers, complement, then add 1 to the result
- **As with sign and magnitude, & one's compl. leading 0s → positive, leading 1s → negative**
  - 000000...xxx is ≥ 0, 111111...xxx is < 0
  - except 1…1111 is -1, not -0 (as in sign & mag.)
- **This representation is Two's Complement**
  - This makes the hardware simple!

**(C's `int`, C18's `intN_t`, aka a "signed integer")**

# Two's Complement Formula

- **Can represent positive and negative numbers in terms of the bit value times a power of 2:**

$d_{31} \times -(2^{31}) + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$

- **Example: $1101_{two}$ in a nibble?**

$= 1 \times -(2^3) + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= -2^3 + 2^2 + 0 + 2^0$

$= -8 + 4 + 0 + 1$

$= -8 + 5$

$= -3_{ten}$

**Example: -3 to +3 to -3 (again, in a nibble):**

$x : 1101_{two}$ −3
$x' : 0010_{two}$
$+1 : 0011_{two}$ 3
$()' : 1100_{two}$
$+1 : 1101_{two}$ −3

# In Summary

- We represent "things" in computers as particular bit patterns: N bits ⇒ $2^N$ things
- These 5 integer encodings have different benefits; 1s complement and sign/mag have most problems.
- unsigned (C18's `uintN_t`) :

00000 00001 ... 01111 10000 ... 11111

- Two's complement (C18's `intN_t`) universal, learn!

00000 00001 ... 01111

10000 ... 11110 11111

- Overflow: numbers ∞; computers finite, errors!