classical computing

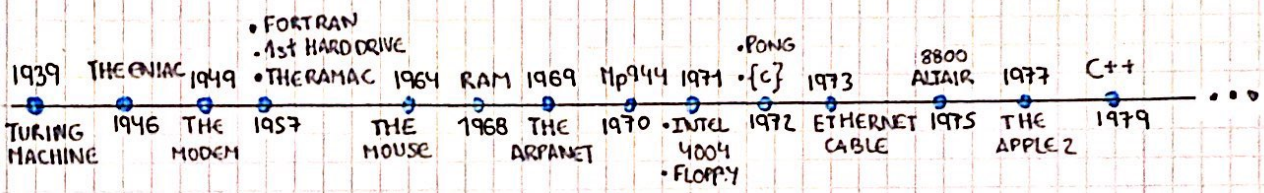
Lecture 1 ∈

by María Delgado

History of classical computing

- WHAT IS COMPUTATION
  - A mathematical calculation that maps inputs to an output based on a set of instruction.
    - something like this draw → Inputs → COMPUTATION [Instructions ... ...] → Outputs

- BIT-SIZED HISTORY OF COMPUTING



Base-representations

"Learning to think like a computer"

★Decimals

- Decimal number system is based on numerical digits 0-9
- Base determines how numbers get represented and how we perform arithmetic operations

Ex: $6 = 6$

$= (6 \cdot 10^0)$

$36 = 30 + 6$

$= (3 \cdot 10^1) + (6 \cdot 10^0)$

$536 = 500 + 30 + 6$

$= (5 \cdot 10^2) + (3 \cdot 10^1) + (6 \cdot 10^0)$

★Binary

- We can describe any number with BITS
- Base-2 is one of the most important bases for performing computation
- It is Binary, only 0 and 1
- Also referred to as a BIT
- We can still do operations, all of the operations in a classical computer happen by manipulating BITS

CONVERTING

1010 → decimal

$= (1 \cdot 10^3) + (0 \cdot 10^2) + (1 \cdot 10^1) + (0 \cdot 10^0)$

1010 → binary

$= (1 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (0 \cdot 2^0) = 8 + 2 = 10$

Base^0 ↗ Binary: $2^0 \approx 11 = 1 \cdot 2^1 + 1 \cdot 2^0 = 3$

↘ Decimal: $10^0 \approx 11 = 1 \cdot 10^1 + 1 \cdot 10^0$

BIts: arithmetic operators

"How computers compute"

★ BINARY ADDITION

- Similar to the decimal we are used to
- BITS carry over when the sum becomes larger than 2

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

★ MULTIPLYING BITS

- BINARY MULTIPLICATION
- It's like "normal" one (like the decimal)

0 · 0 = 0

0 · 1 = 0

1 · 0 = 0

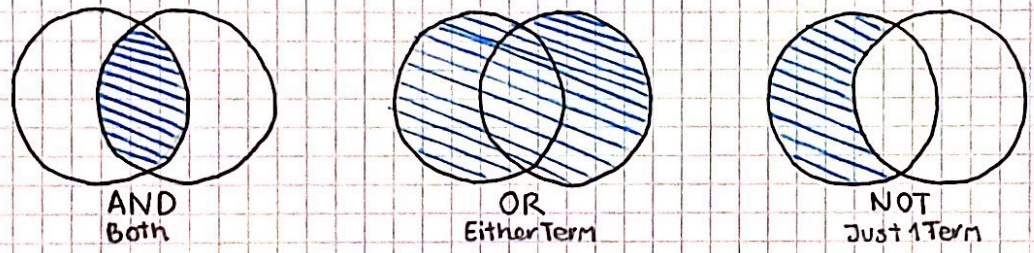
1 · 1 = 1

Why BITS

- Easier for Building Hardware
- Fast Operations

Boolean Logic

- Boolean logic: Maps Input bit(s) to output bit(s)



- Logic gates + Truth Tables
  - Logic: Maps Input bit(s) to output bit(s)
  - Truth Tables: Tells us the output of a logical operation based on its input

Gates 1 BIT

- Gates: NOT
  - Not: flips the bit

INPUT	OUTPUT
0	1
1	0

- Gates: FANOUT
  - Fanout: copies the bit

a	OUTPUT
0	0 0
1	1 1

Gates 2BIT

- Gates: AND
  - outputs 1 if both are 1, outputs 0 otherw.

a	b	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

- Gates: OR
  - outputs 1 if either of the input bits is 1 outputs 0 if neither of the inputs bits is 1

a	b	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

- Gates: XOR
  - Outputs 1 if either input bits are 1, but not both outputs 0 if neither or both bits are 1

a XOR b

a ⊗ b

a	b	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

universality

- Any computation operation can be made by using a combination of {NOT, AND, OR, FANOUT}

~ Gates: NAND → (NOT + AND)

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

c	OUT
0	1
1	0

a	b	OUT
0	0	1
0	1	1
1	0	1
1	1	0

"Let's make a Binary adder with the gates we have" a + b =

a	b	OUT_1	OUT_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

a	b	Bit_0
0	0	0
0	1	1
1	0	1
1	1	0

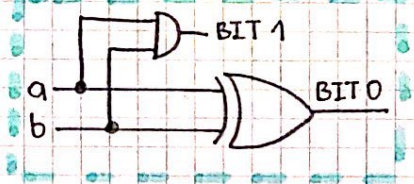
OUT_2

a	b	Bit_1
0	0	0
0	1	0
1	0	0
1	1	1

OUT_1

OUT_2 → XOR →

OUT_1 → AND →



Reversibility

- Given the output of a gate, we can determine what the inputs are

- Is NOT reversible? **YES**
  - output 1 → 0
  - output 0 → 1
- Is AND reversible? **NO**
  - output 1 → 1,1
  - output 0 → 0,0 / 0,1 / 1,0

→ REVERSIBLE GATE: preserves all the information

→ NON-REVERSIBLE GATE: loses some information