Computer Organization and Architecture

COSC 2425

Lecture – 2

Aug 24th, 2022

Acknowledgement: Slides from Edgar Gabriel & Kevin Long

Review: Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff

- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

Review: Classes of Computers

- Supercomputers
  – High-end scientific and engineering calculations
  – Highest capability but represent a small fraction of the overall computer market

- Embedded computers
  – Hidden as components of systems
  – Stringent power/performance/cost constraints

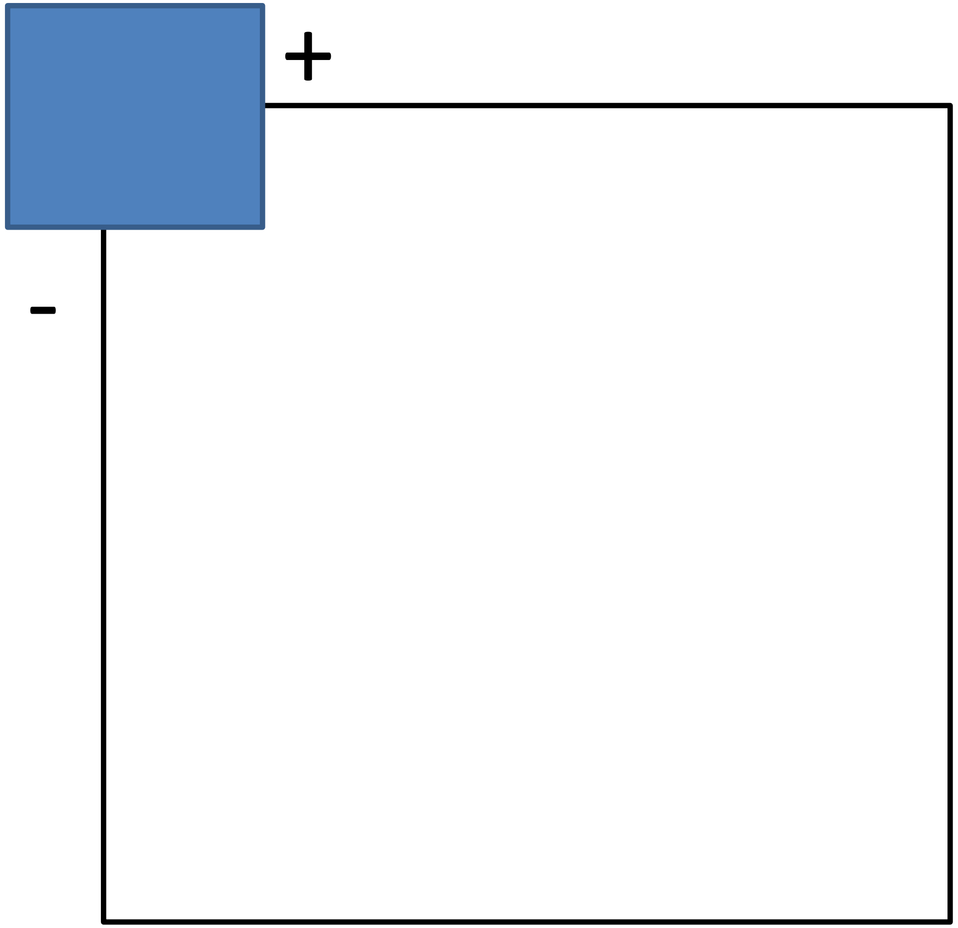
Review: How do computers work?

Today: 20 emails → Represent numbers
Yesterday: 10 emails
Total: 30 emails → Perform addition

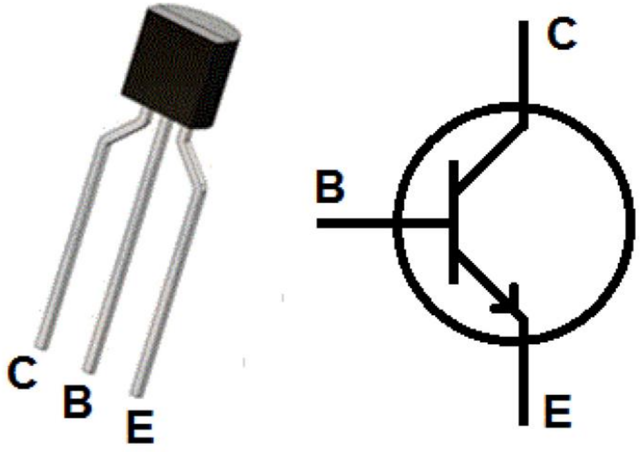
Email1 => subject: Hi John → Represent text

What is the basic idea behind a computer?
How does it do this?

Review: Control the flow of current using transistors

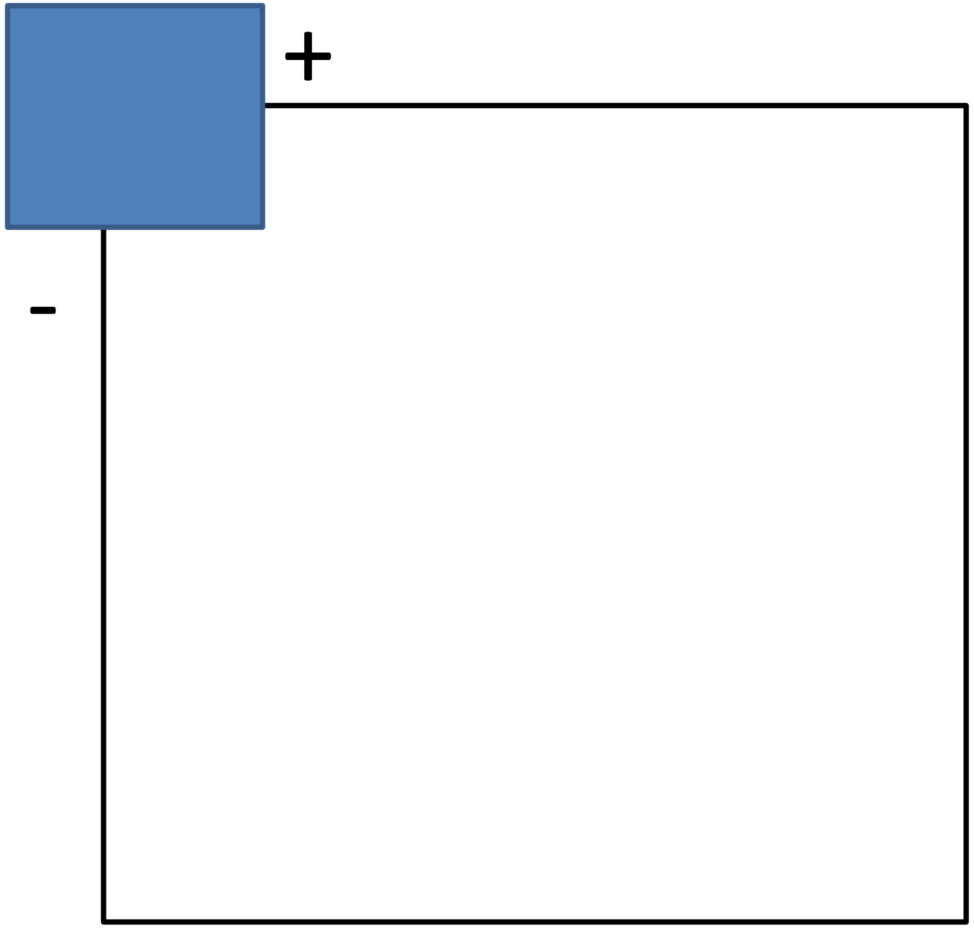


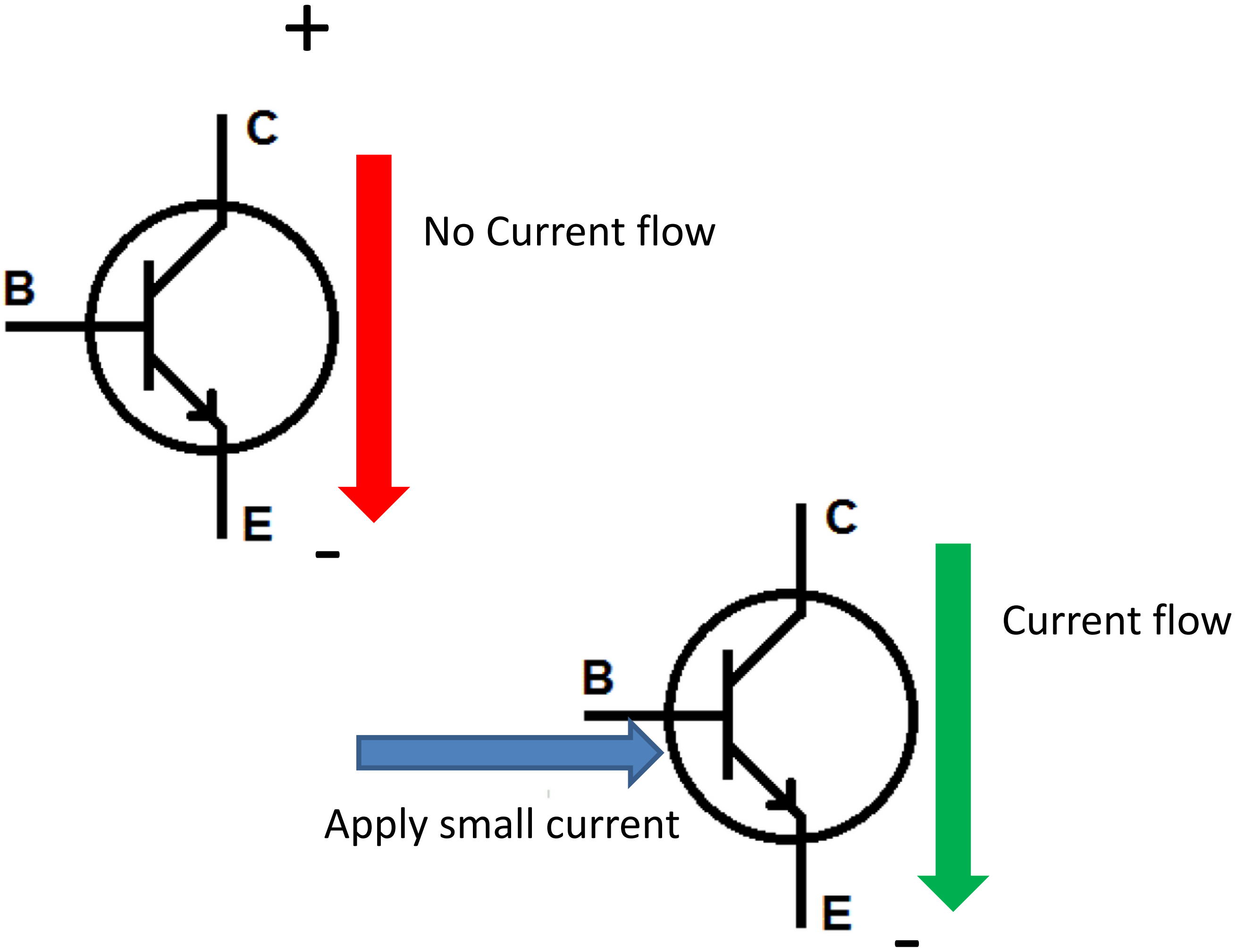
NPN Transistor



Transistor　　Symbol

Review: Control the flow of current using transistors





Review: Binary Representation of Numbers

Binary Number -> use two symbols for representation (0 & 1)

101101

Review: Binary Representation of Numbers

Binary Number -> use two symbols for representation (0 & 1)

101101

index						
index	5	4	3	2	1	0
	1	0	1	1	0	1
	1×2^5 +	0×2^4 +	1×2^3 +	1×2^2 +	0×2^1 +	1×2^0
	1 X 32	0 X 16	1 X 8	1X 4	0 X 2	1 X 1

= 45

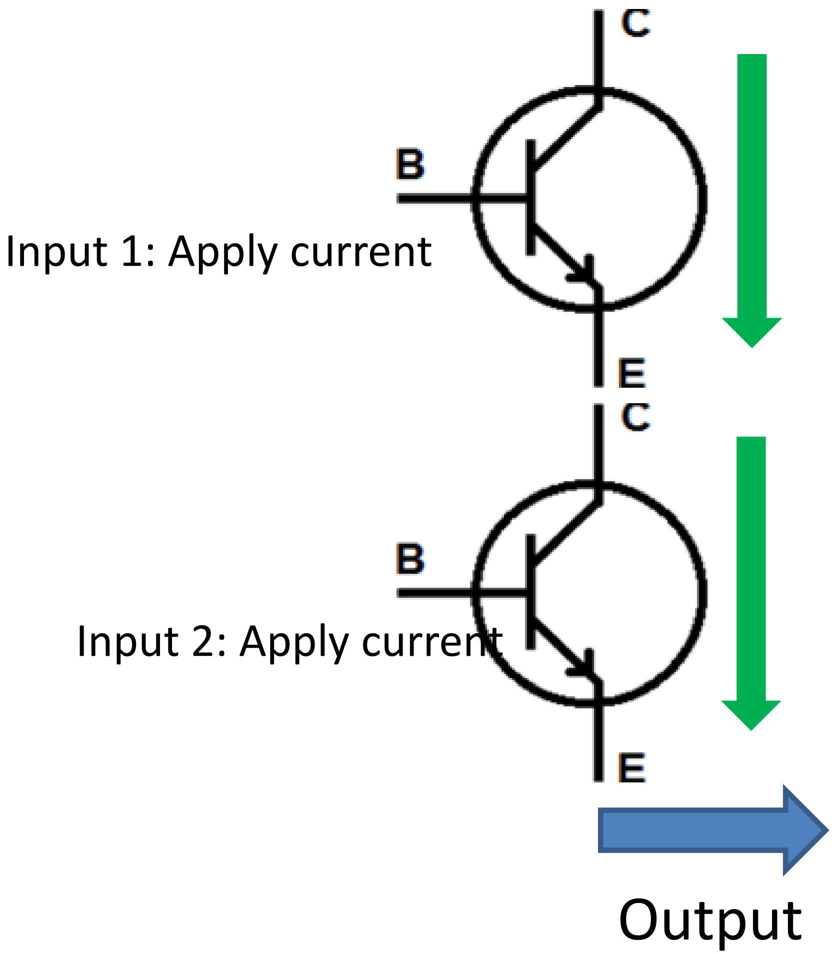
Review: Represent Letters

- Come up with an arbitrary convention to associate with numbers, and then use binary representation.
- A => 65 => 01000001
- B => 66 => 01000010
- C => 67 => 01000011
- D => 68 => 01000100
- Represent words CAB => 01000011 01000001 01000010

Review: Computations

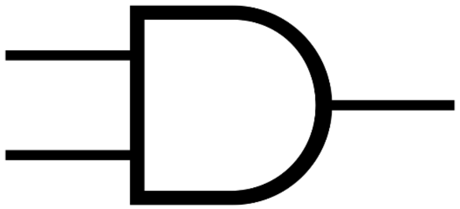
- Can we do Arithmetic (Addition)?
  - Need to take at least two inputs, and operate on them.
  - We will build logic gates to accomplish this.
  - How do we build logic gates?
    - Cleverly place transistors to create circuits.

Review: Two transistors in parallel: Logic Gate



Input 1	Input 2	Output
0	0	0
1	0	0
0	1	0
1	1	1

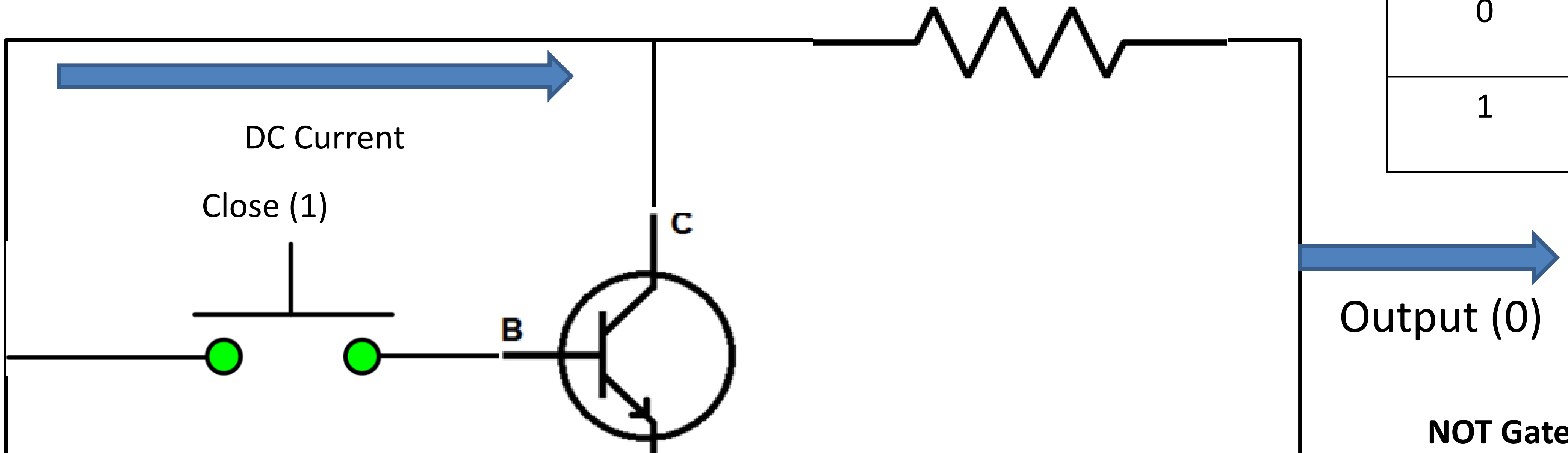
AND Gate



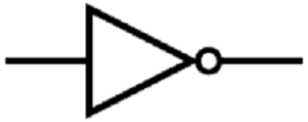
Symbol

Review: Another logic gate

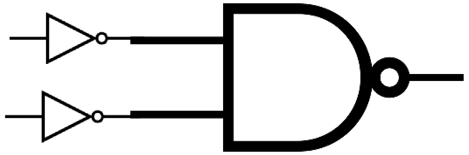
Input	Output
0	1
1	0



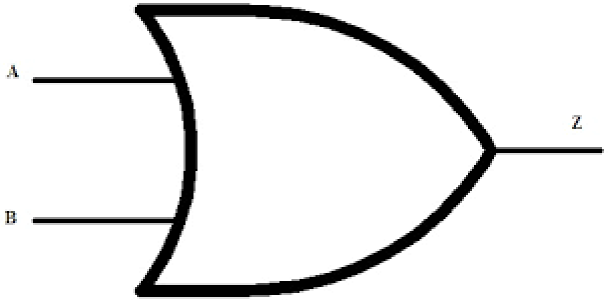
NOT Gate



Review: OR Gate



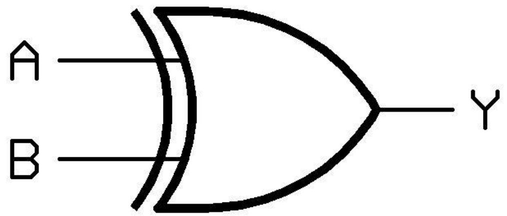
Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1



OR Gate

Review: ADD Numbers (Simplified)

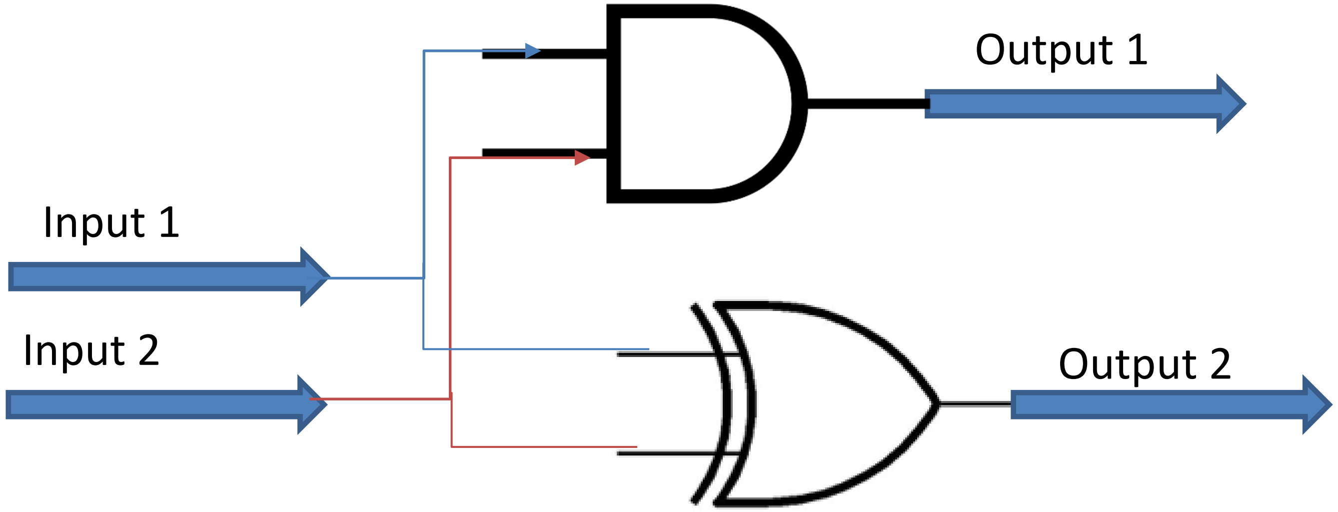
- Add two one-bit numbers, and produce two-bit results
- Input can be 0 or 1



XOR Gate

Input 1	Input 2	Output 2
0	0	0
1	0	1
0	1	1
1	1	0

Review: Design an Adder



XOR Gate

It looks similar to OR Gate.
Except the last row is inverted where both inputs are 1, the result in inverted.

Input 1	Input 2	Output 1	Output 2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Binary

Review: Many Such Circuits Exists

- There are many such circuits available to
  - Load data, store data, add, subtract and perform logical ops on data.

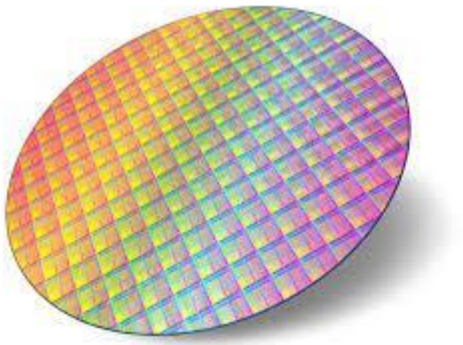
Review: Questions

1. What operations can hardware perform? How to instruct computer to perform a certain operation? How are negative numbers/exponentials represented?
2. How do we perform addition, multiplication, division?
3. How do we improve the speed of the computer? Can we do things in parallel (compute while loading next data, etc.)
4. Where is data stored? How can we make it efficient?
5. Can we perform computations in parallel to improve performance?
6. How do we define performance?

Review: Third Generation: Integrated Circuits

- 1958 – the invention of the integrated circuit
- Exploits the fact that transistors can be fabricated from a semiconductor such as silicon

- Many transistors can be produced at the same time on a single wafer of silicon
- Transistors can be connected with a processor metallization to form circuits



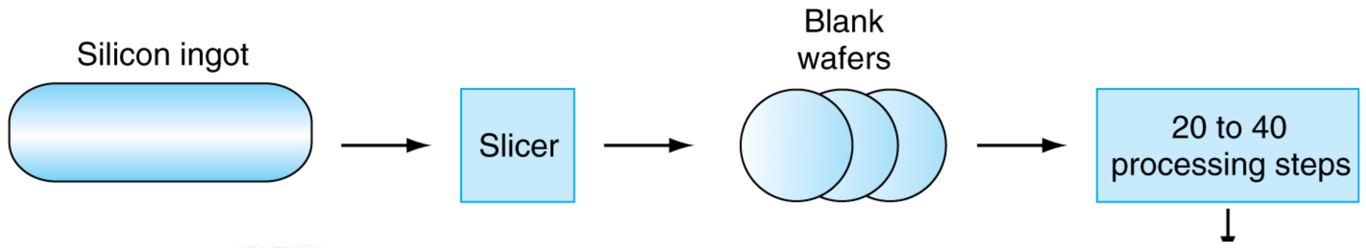
Semiconductor Technology

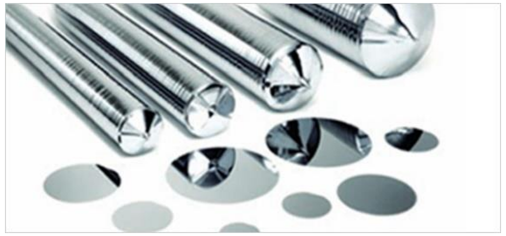
- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

Manufacturing ICs

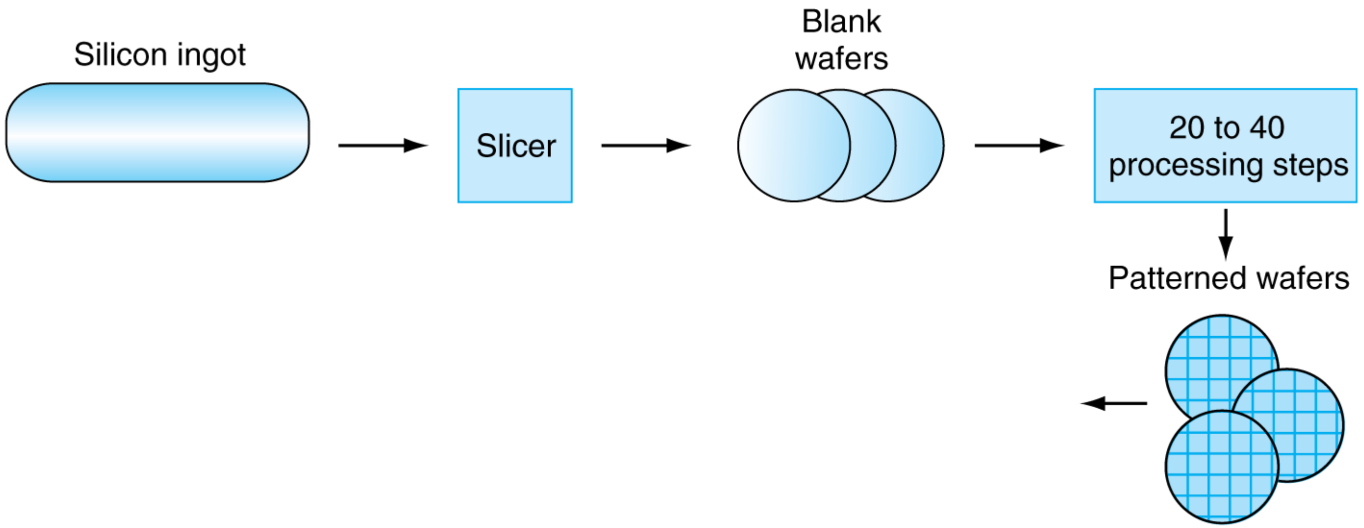
Silicon ingot

Manufacturing ICs



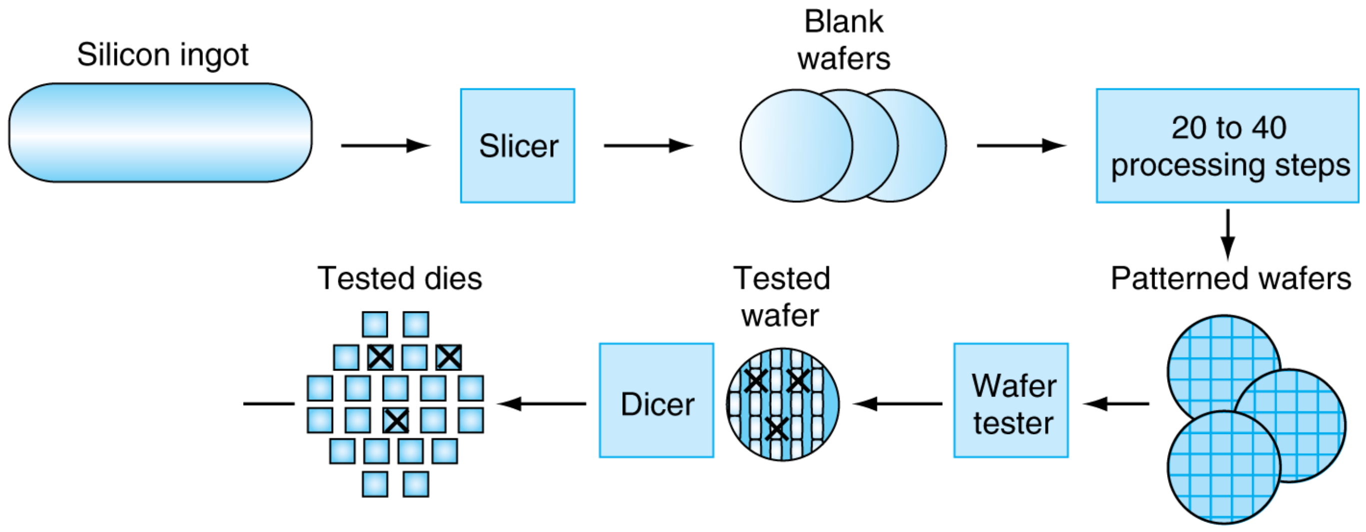


Manufacturing ICs



Chemical Process/Add impurities to make switches (transistors

Manufacturing ICs



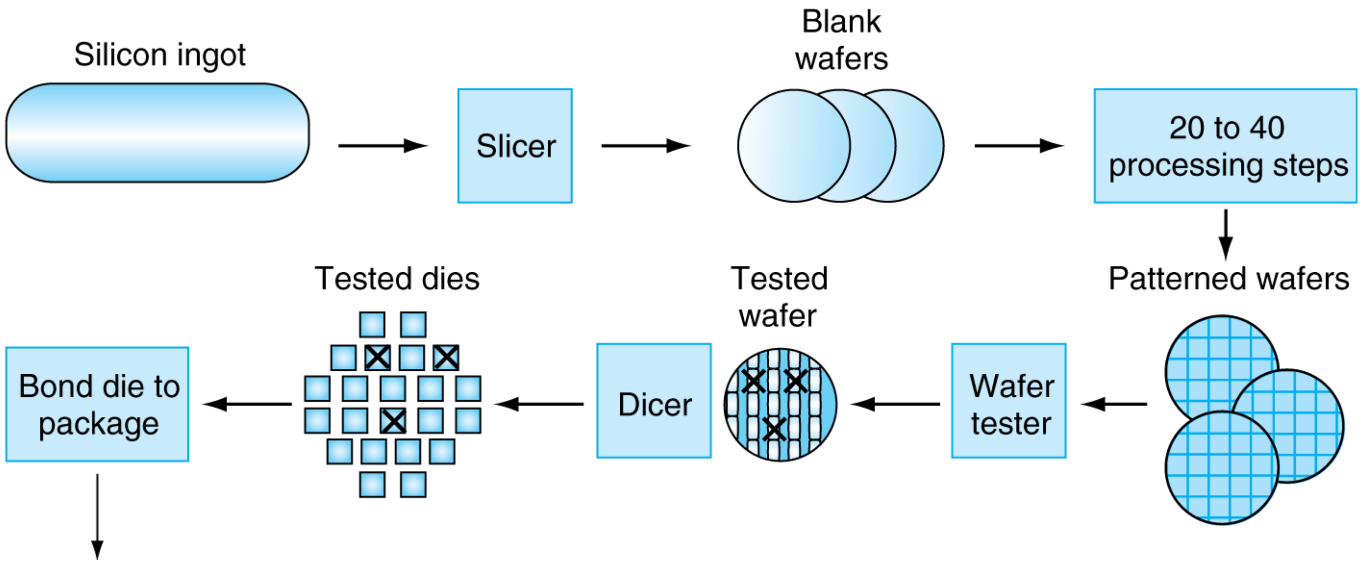
Chemical Process/Add impurities to make switches (transistors

Delicate Process at microscopic level, potential for failure

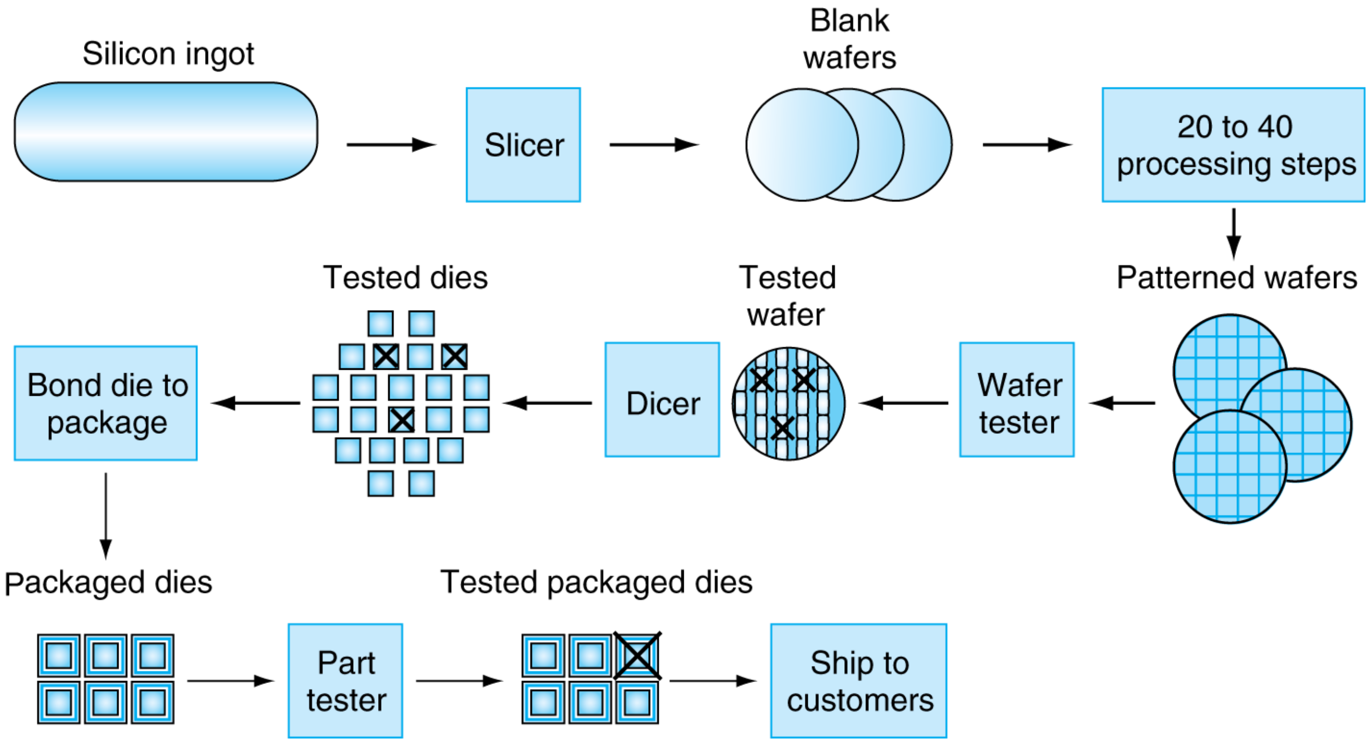
Multiple chips/dies from each wafer

Manufacturing ICs

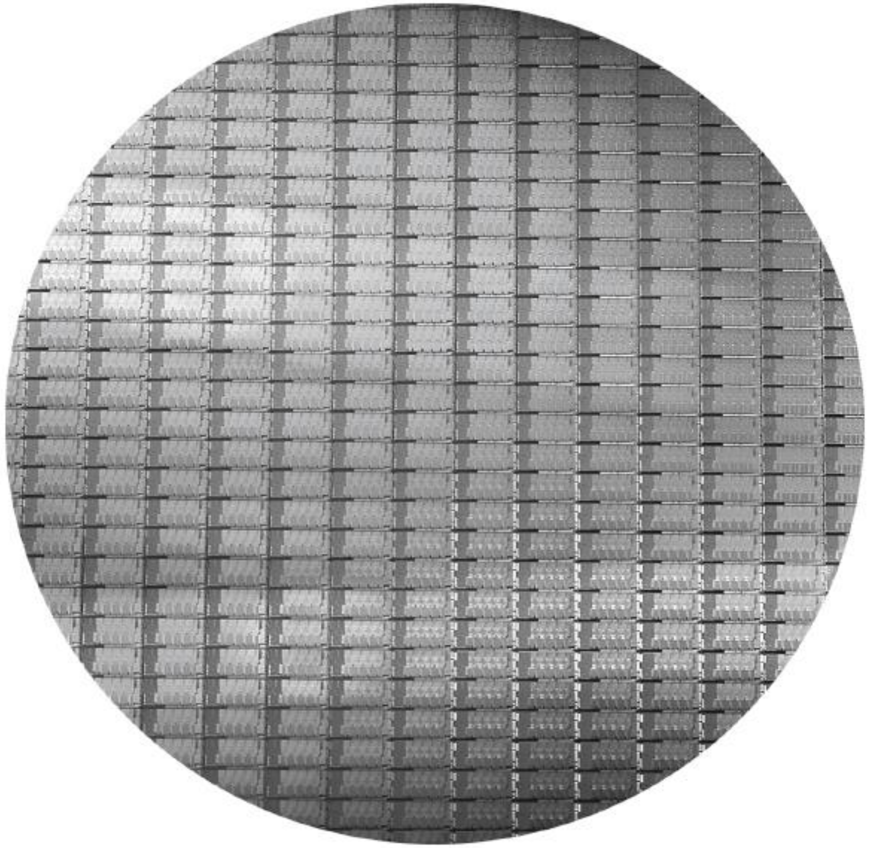
Circuitry to create logic gates/



Manufacturing ICs



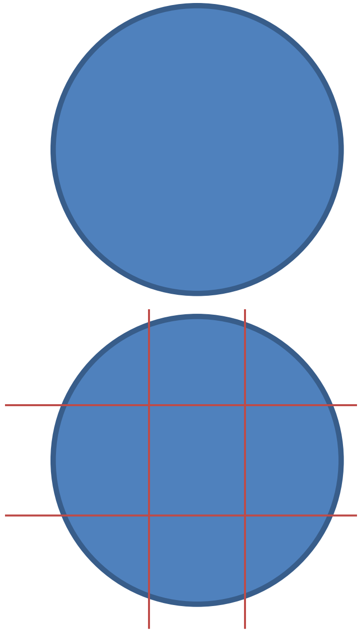
Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

Integrated Circuit Cost

Cost to make one chip/die

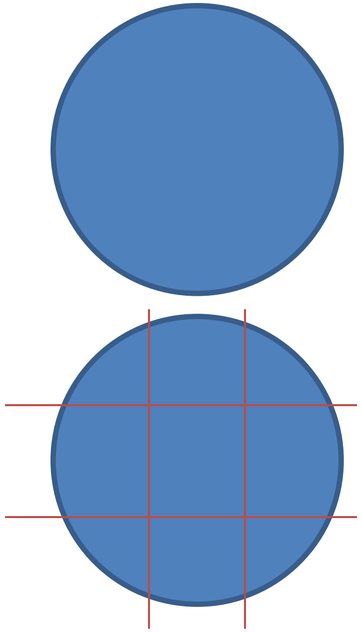


$90 Cost per wafer

9 chips/dies

? Cost per die

Integrated Circuit Cost

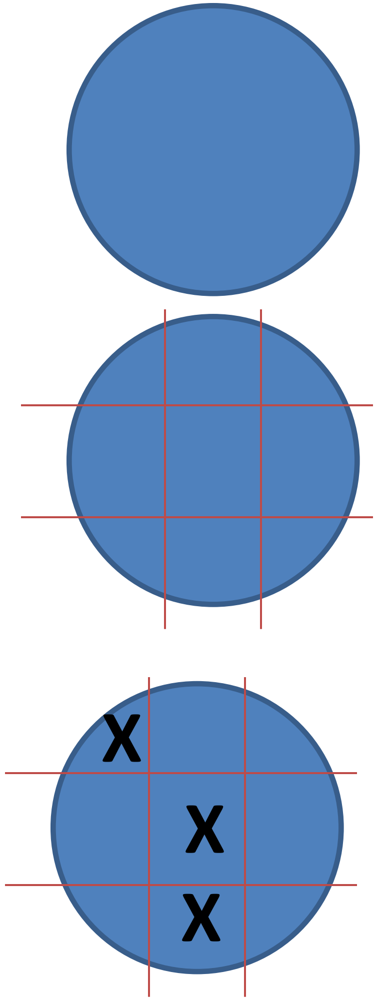


$90 Cost per wafer

9 chips/dies

$10 $Cost\ per\ die = \frac{Cost\ per\ wafer\ (90)}{dies\ per\ wafer\ (9)}$

Integrated Circuit Cost



$90 Cost per wafer

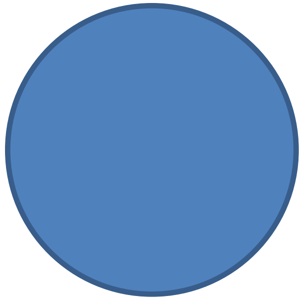
9 chips/dies

$10 $$Cost\ per\ die = \frac{Cost\ per\ wafer\ (90)}{dies\ per\ wafer\ (9)}$$

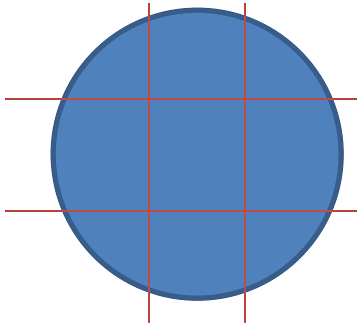
9 chips/dies

?

Integrated Circuit Cost



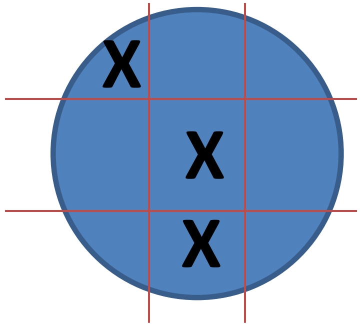
\$90 Cost per wafer



9 chips/dies

\$10

$$Cost\ per\ die\ = \frac{Cost\ per\ wafer\ (90)}{dies\ per\ wafer\ (9)}$$

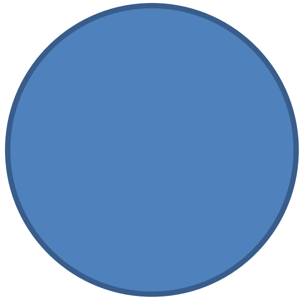


9 chips/dies

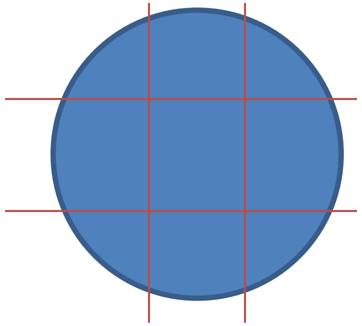
?

$$yield\ = \frac{Working\ dies (6)}{dies\ per\ wafer\ (9)} = 0.66$$

Integrated Circuit Cost

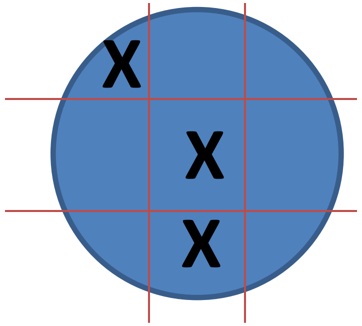


$90 Cost per wafer



9 chips/dies

$10 $$Cost\ per\ die\ = \frac{Cost\ per\ wafer\ (90)}{dies\ per\ wafer\ (9)} = 10$$



9 chips/dies

?

$$yield\ = \frac{Working\ dies(6)}{dies\ per\ wafer\ (9)} = 0.66$$

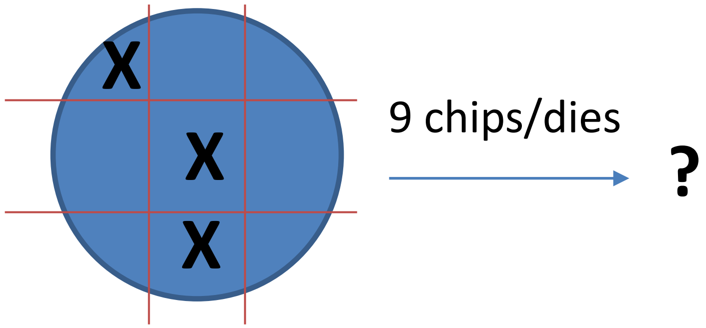
$$Cost\ per\ die\ = \frac{Cost\ per\ wafer\ (90)}{dies\ per\ wafer\ (9) * yield\ (0.66)} = 15.15$$

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

Integrated Circuit Cost



$$yield = \frac{Working\ dies(6)}{dies\ per\ wafer\ (9)} = 0.66$$

Yield for this wafer.
Much more complicated to compute yield for manufacturing process.
Many statistical models are proposed to estimate yield.

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

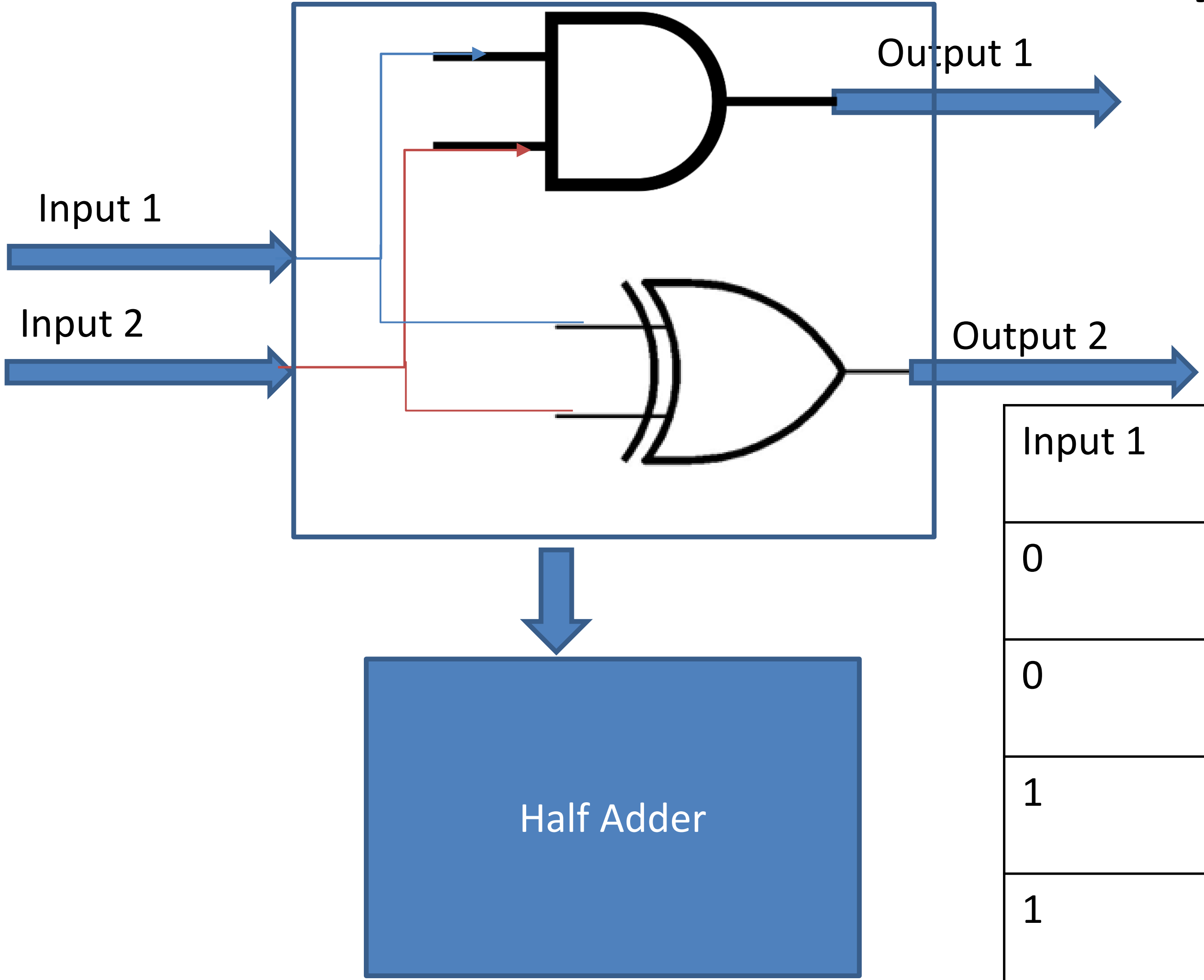
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

Questions

1. What operations can hardware perform? How to instruct computer to perform a certain operation? How are negative numbers/exponentials represented?
2. How do we perform addition, multiplication, division?
3. How do we improve the speed of the computer? Can we do things in parallel (compute while loading next data, etc.)
4. Where is data stored? How can we make it efficient?
5. Can we perform computations in parallel to improve performance?
6. How do we define performance?

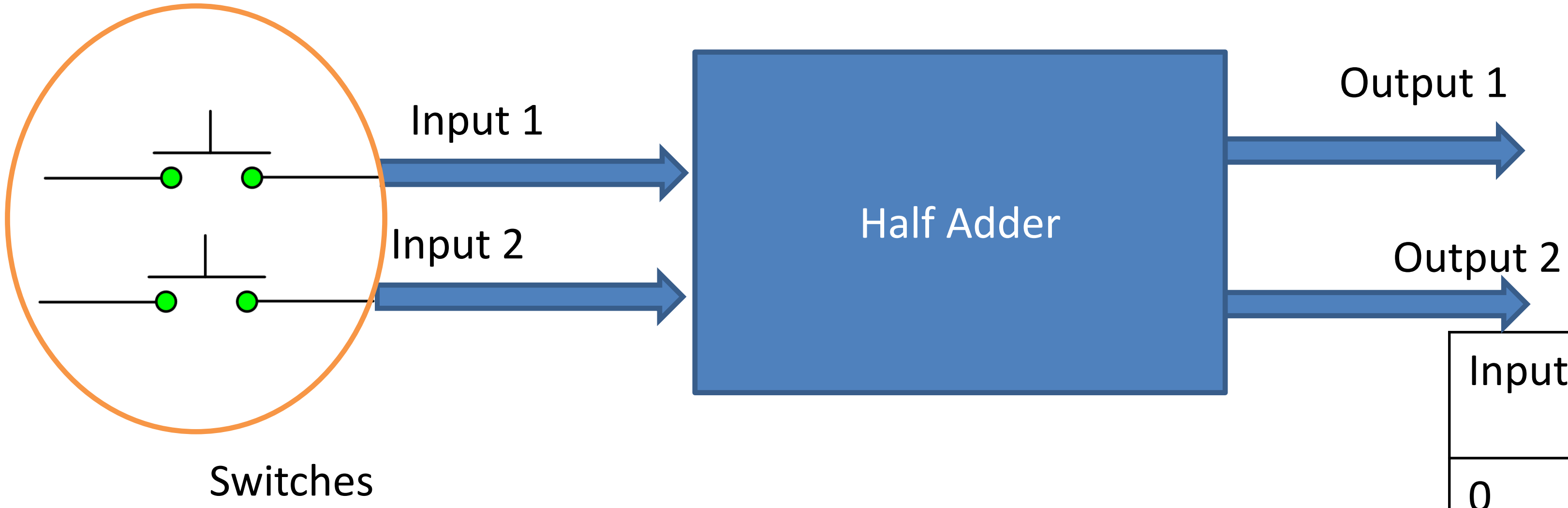
Half-Adder with manual input



Input 1	Input 2	Output 1	Output 2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Binary

Half-Adder with manual input

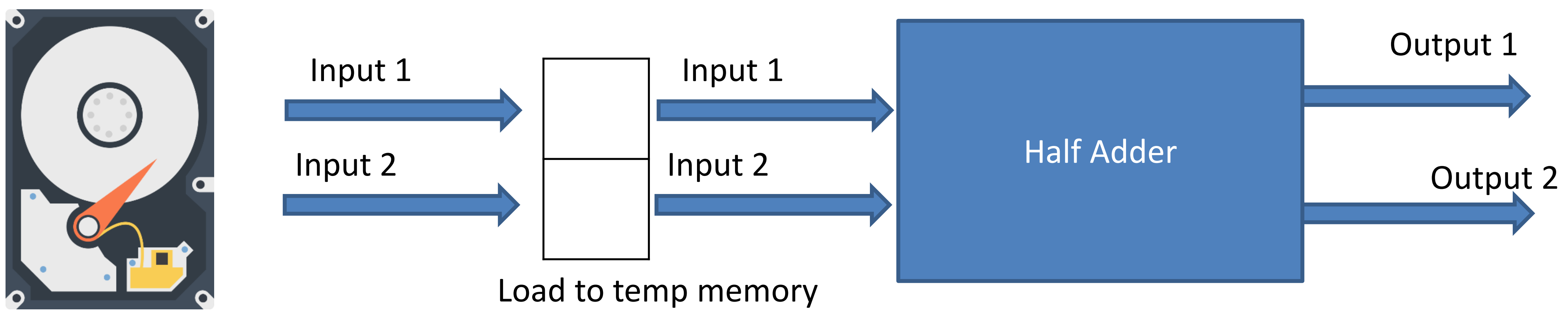


Input 1	Input 2	Output 1	Output 2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Binary

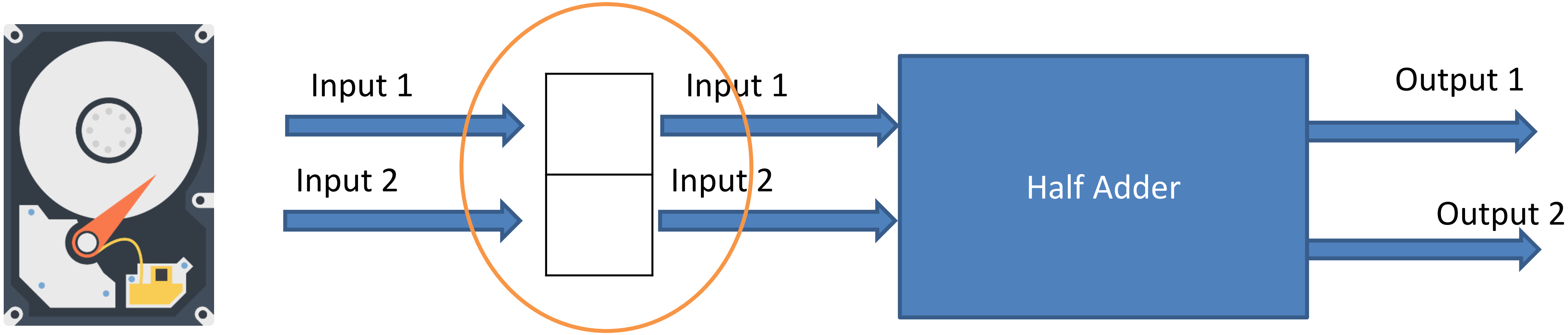
Half-Adder with manual input

Stored in memory (E.g. HDD)



Half-Adder with manual input

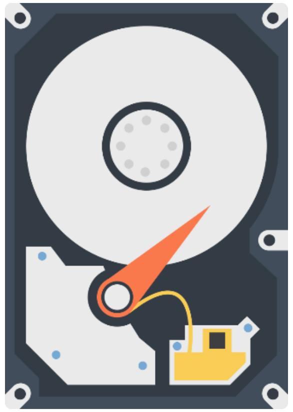
Stored in memory (E.g. HDD)

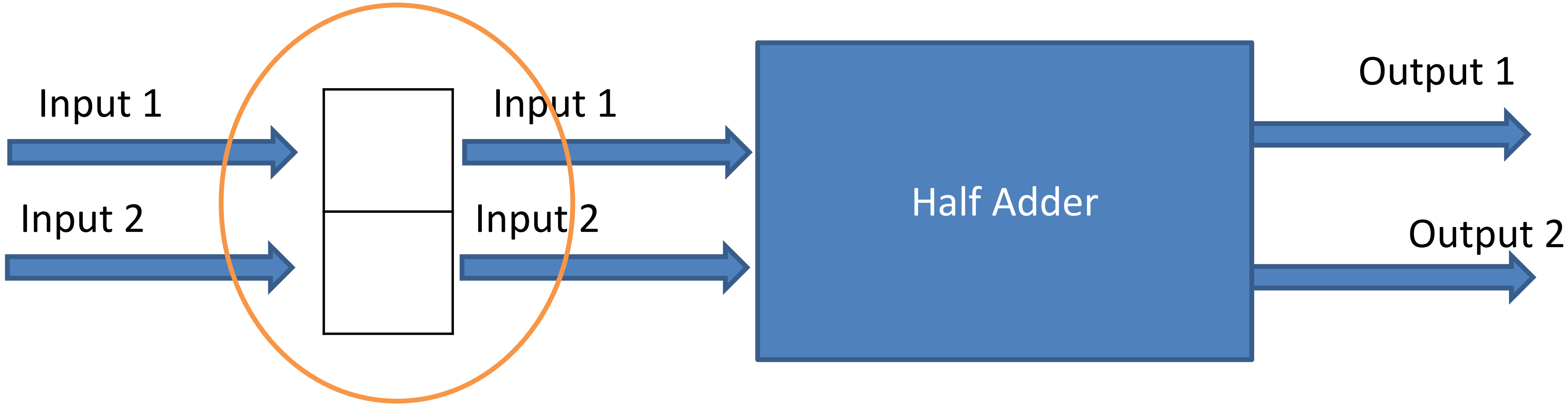


What is memory?
How do you make a computer remember values (0, 1)?

Half-Adder with manual input

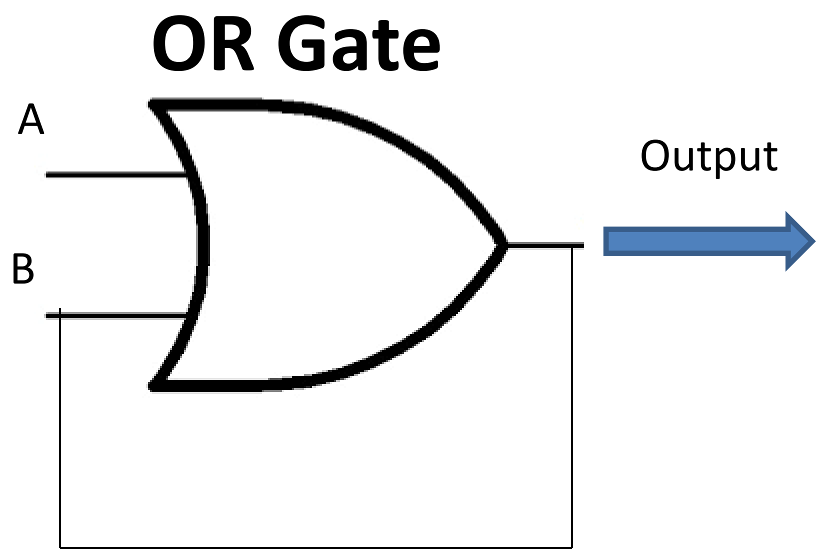
Stored in memory (E.g. HDD)





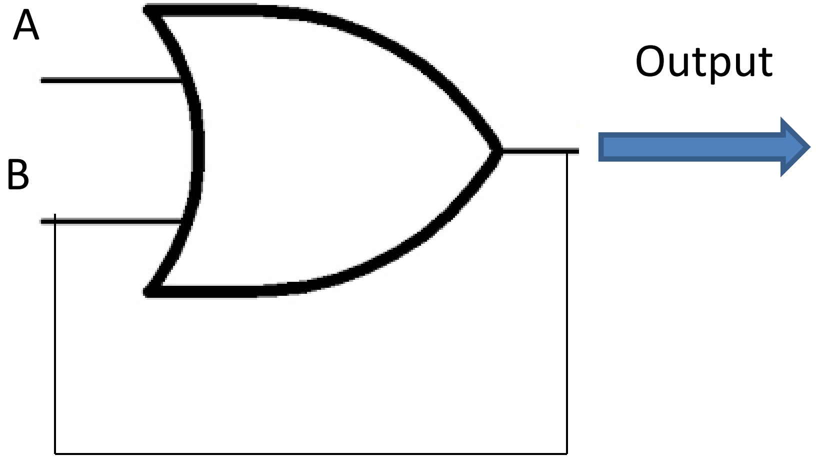
What is memory?
How do you make a computer remember values (0, 1)?
Also using Logic Gates.

Remembering 1's



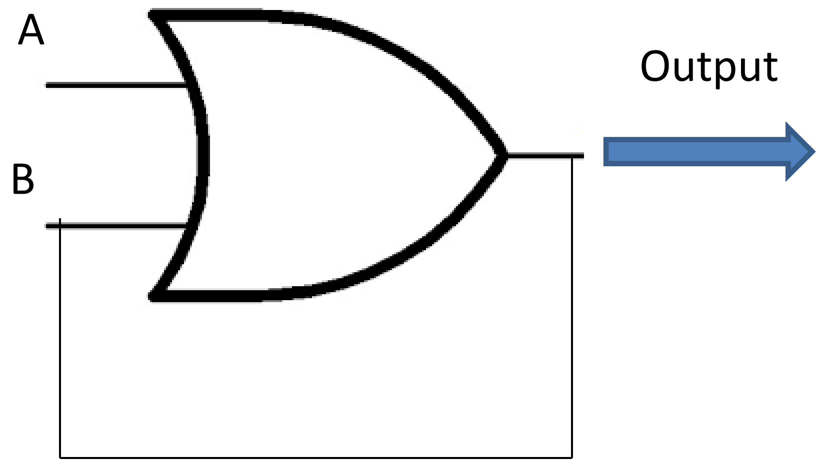
A	B	Output
0	0	0
1	0	1

Remembering 1's



A	B	Output
0	0	0
1	0	1
0	X	?

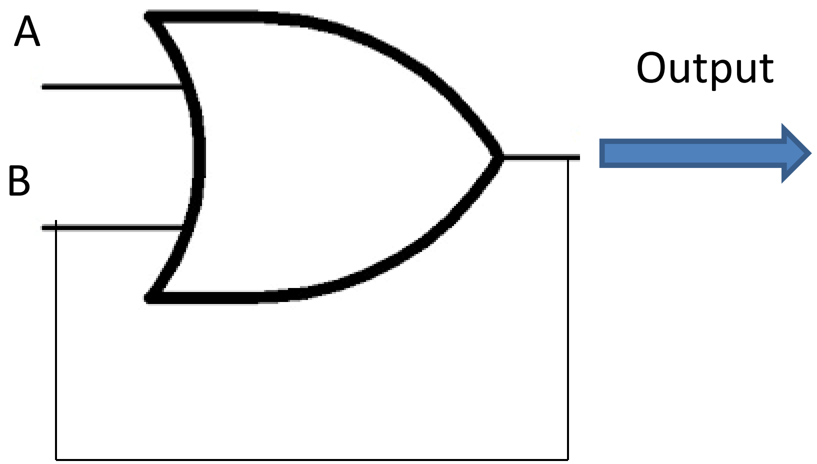
Remembering 1’s

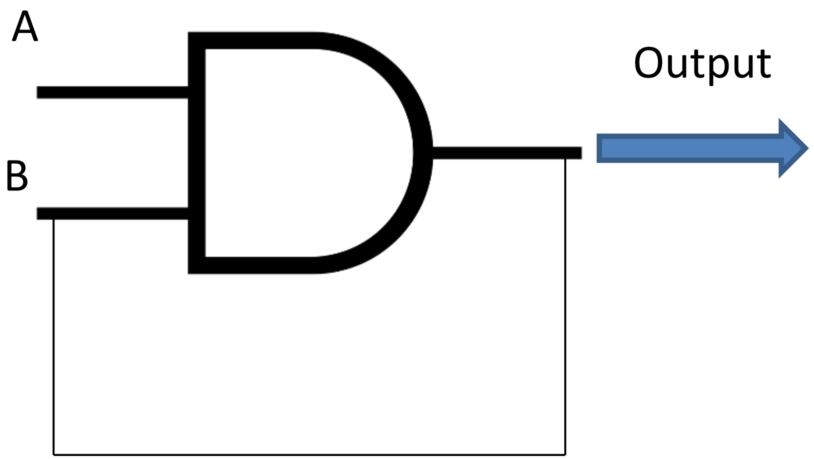


A	B	Output
0	0	0
1	0	1
0	X	1
1	X	1

Once input, it remembers a **one** for ever.

Remembering 1's and 0's





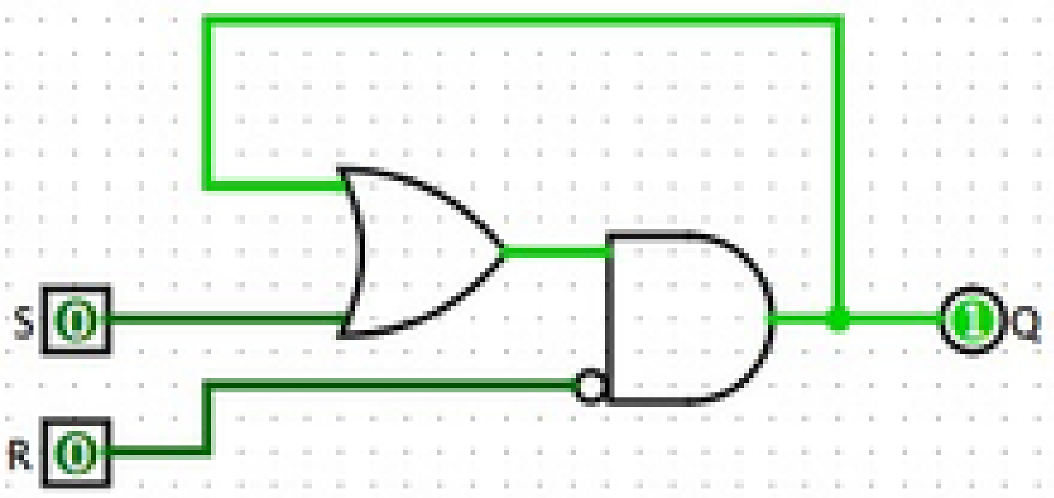
A	B	Output
0	0	0
1	0	1
0	X	1
1	X	1

A	B	Output
1	1	1
0	1	0
1	X	0
0	X	0

Combine Both: SR And-OR Latch

Set

Reset



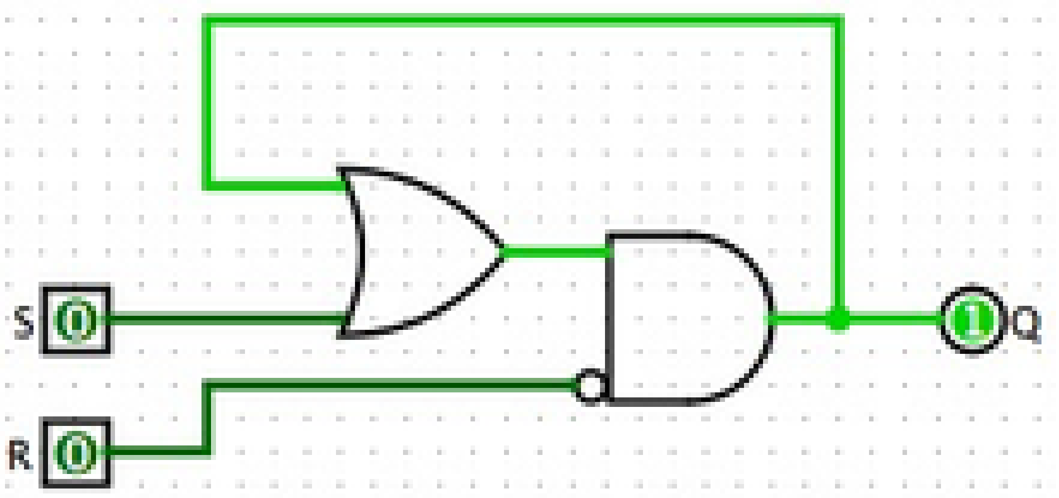
S	R	Output
1	0	?

https://upload.wikimedia.org/wikipedia/commons/e/e0/RS-and-or-flip-flop.png

Combine Both: SR And-OR Latch

Set

Reset



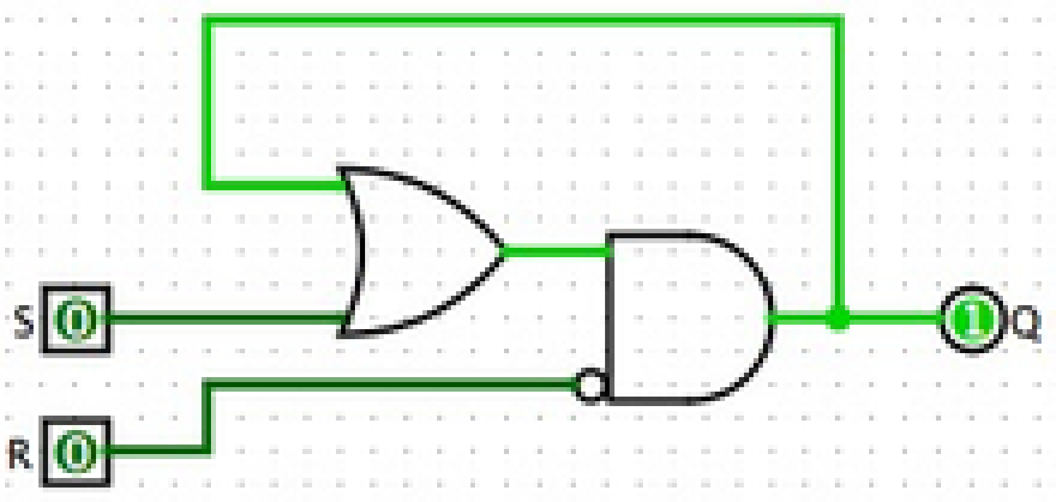
S	R	Output
1	0	1
0	0	?

https://upload.wikimedia.org/wikipedia/commons/e/e0/RS-and-or-flip-flop.png

Combine Both: SR And-OR Latch

Set

Reset



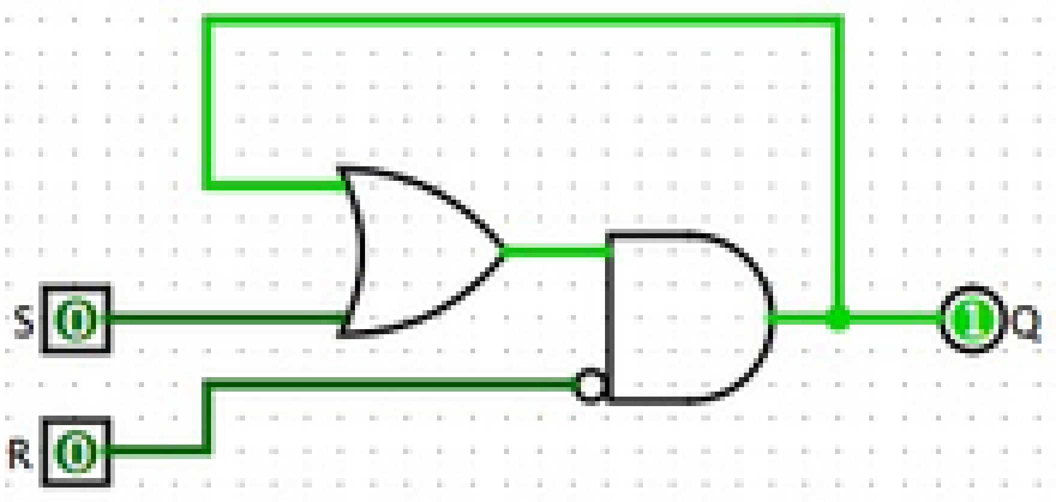
S	R	Output
1	0	1
0	0	1
0	1	?

https://upload.wikimedia.org/wikipedia/commons/e/e0/RS-and-or-flip-flop.png

Combine Both: SR And-OR **Latch**

Set

Reset



S	R	Output
1	0	1
0	0	1
0	1	0
0	0	0

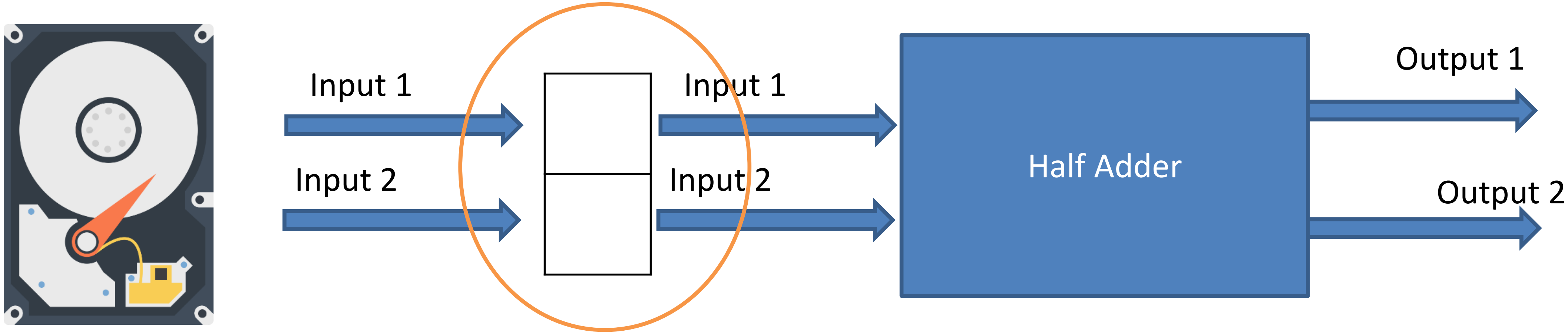
https://upload.wikimedia.org/wikipedia/commons/e/e0/RS-and-or-flip-flop.png

Gated Latch

I	W	O
0	0	0
1	0	0 (No update)
0	**1**	0 (Same as I)
1	**1**	1 (Same as I)
0	0	1 (No update)
1	0	1 (No update)

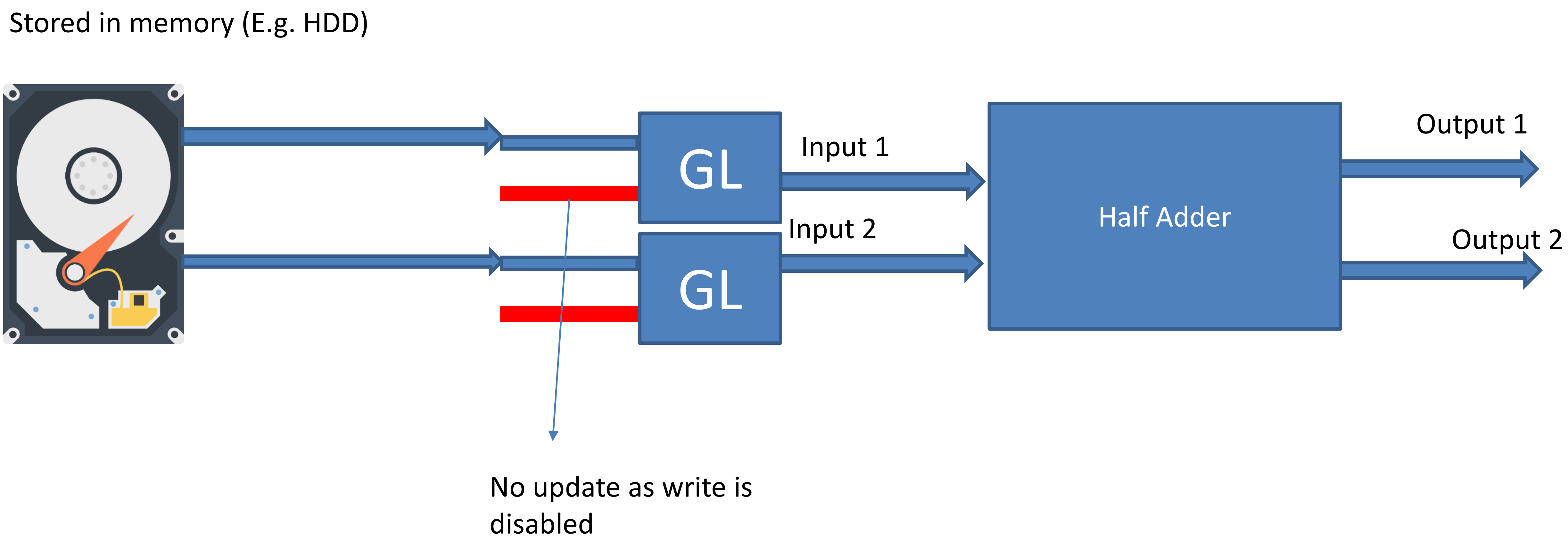
Half-Adder with manual input

Stored in memory (E.g. HDD)



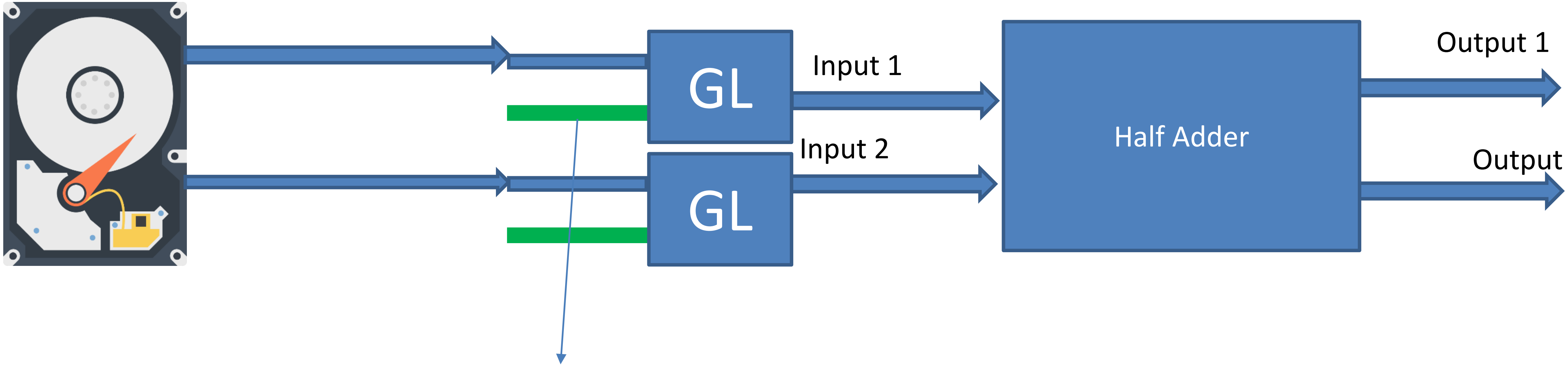
What is memory?
How do you make a computer remember values (0, 1)?
Also using Logic Gates.

Half-Adder with manual input



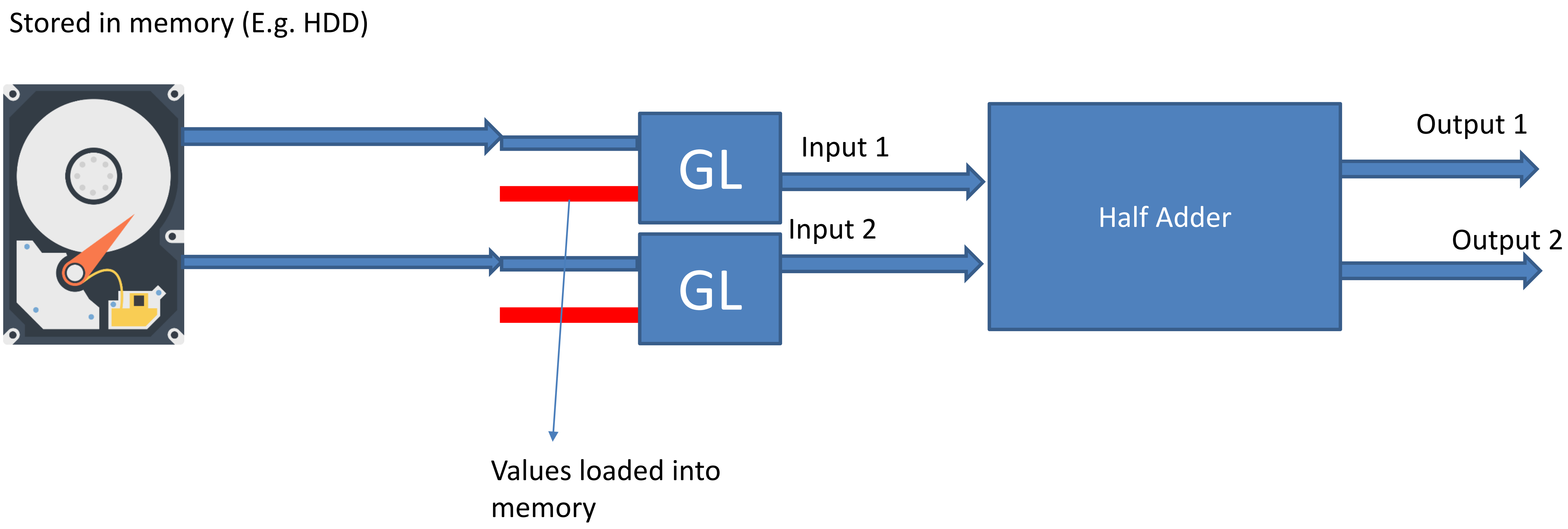
Half-Adder with manual input

Stored in memory (E.g. HDD)

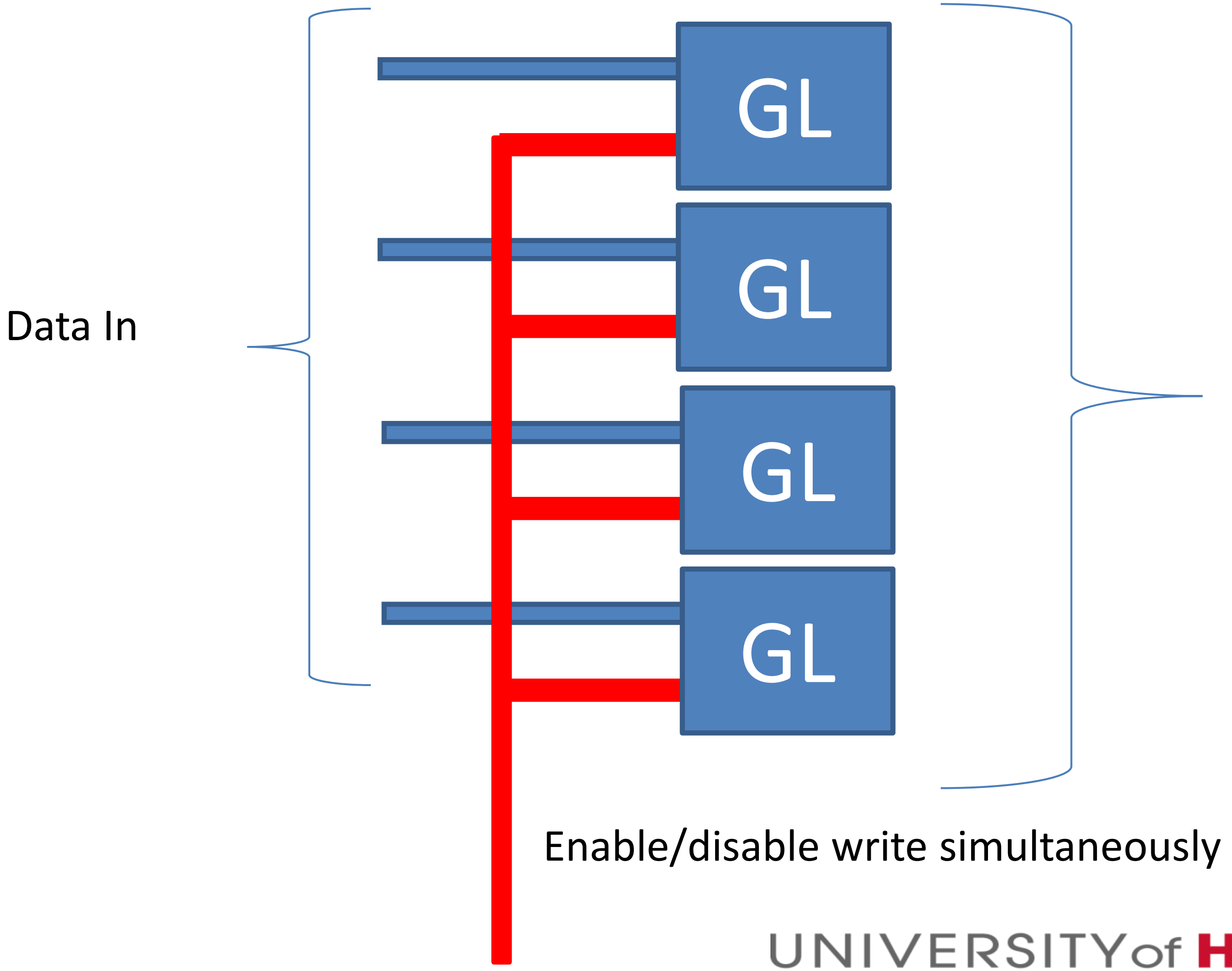


Values updated as
write is enabled

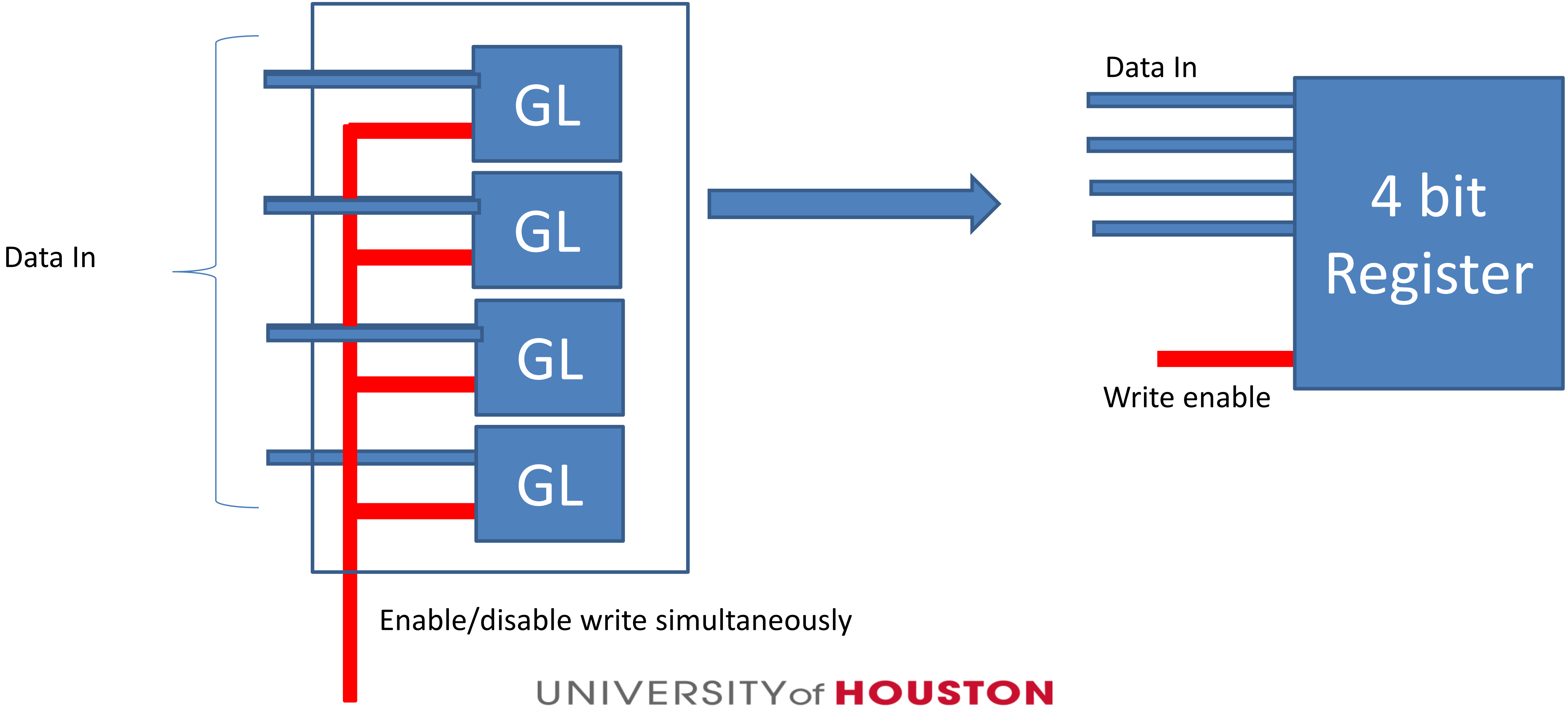
Half-Adder with manual input



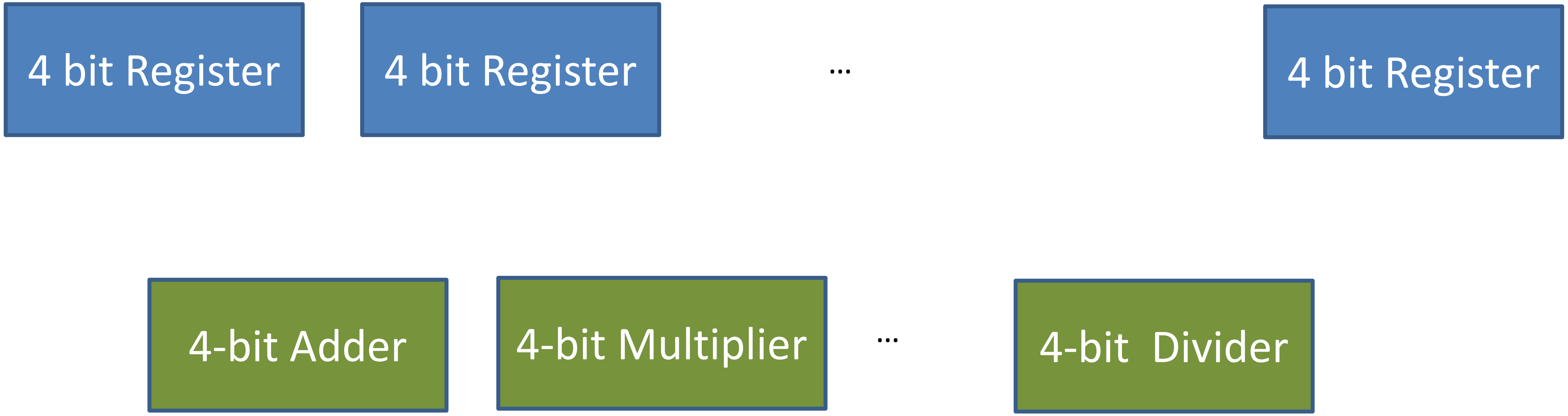
Register (4 bit) – A group of latches



Register (4 bit) – A group of latches



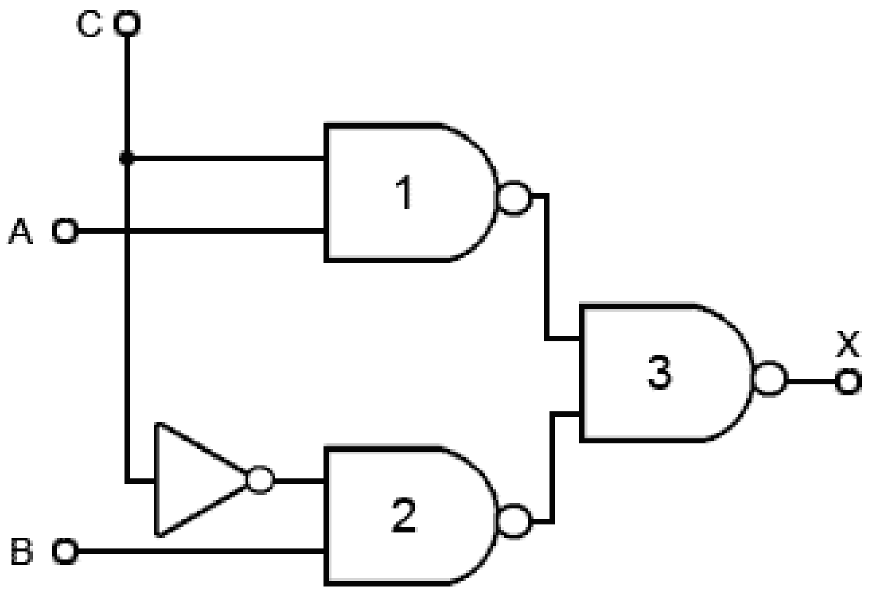
Instruction



Instruction

How to instruct computer to Add/Multiply/Divide?

Multiplexer (Data selector)

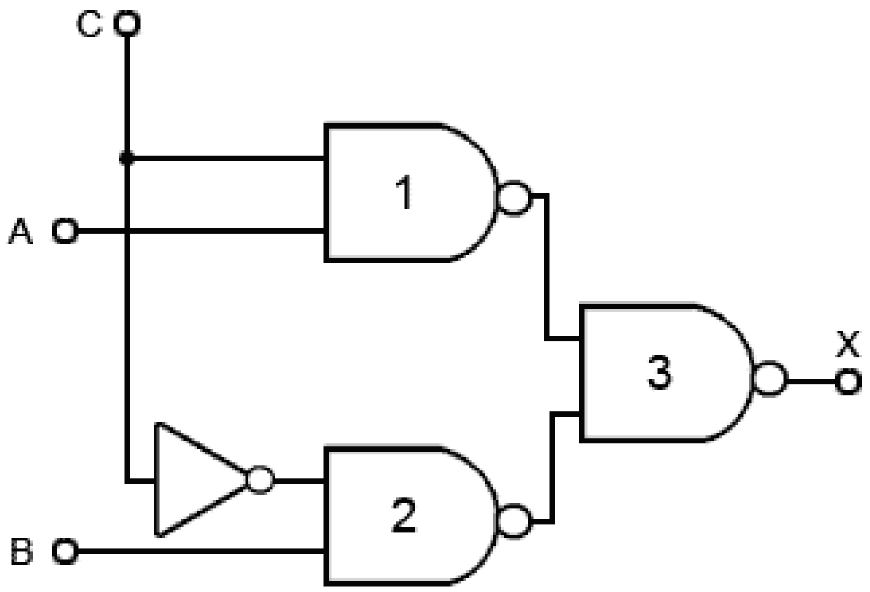


A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

https://learnabout-electronics.org/Digital/dig42.php

Multiplexer (Data selector)



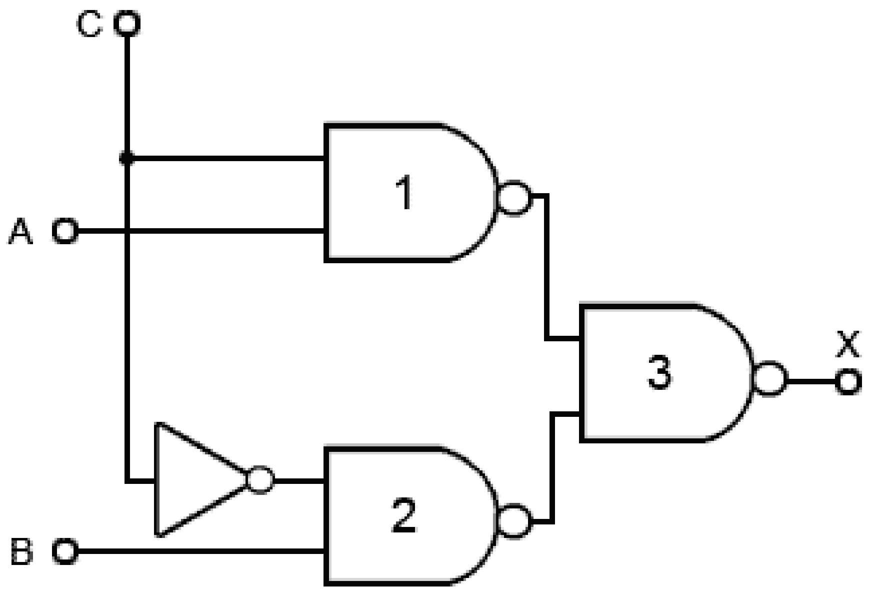
C = 0

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

https://learnabout-electronics.org/Digital/dig42.php

Multiplexer (Data selector)



C = 1

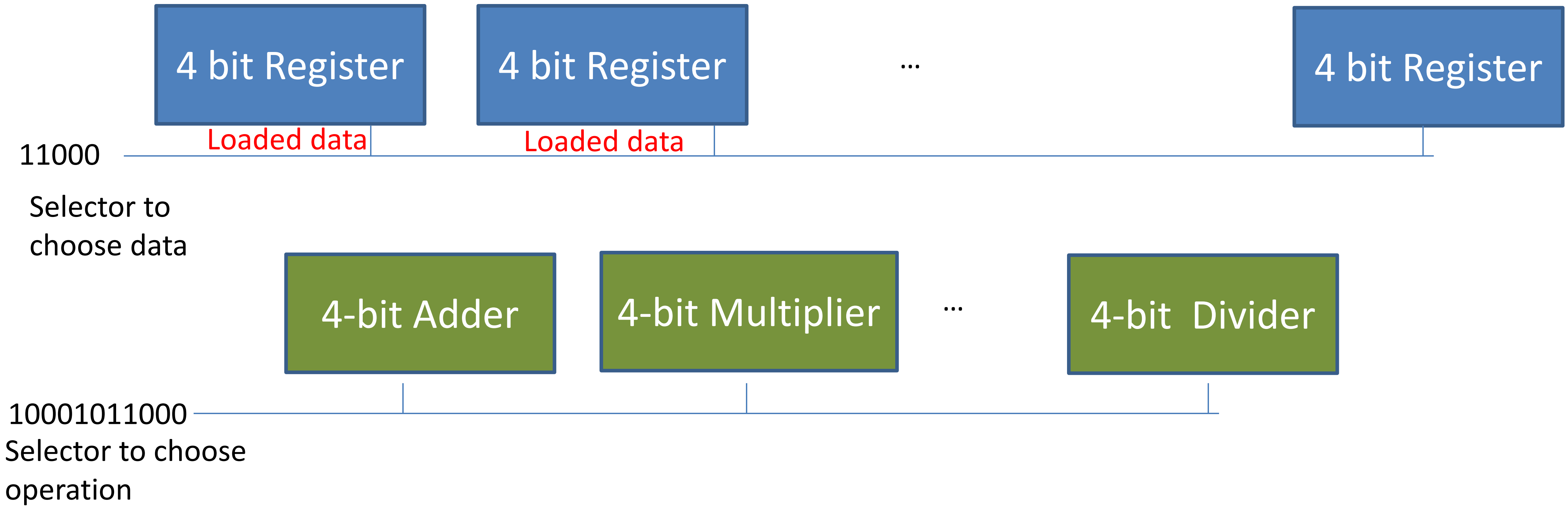
Selector

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

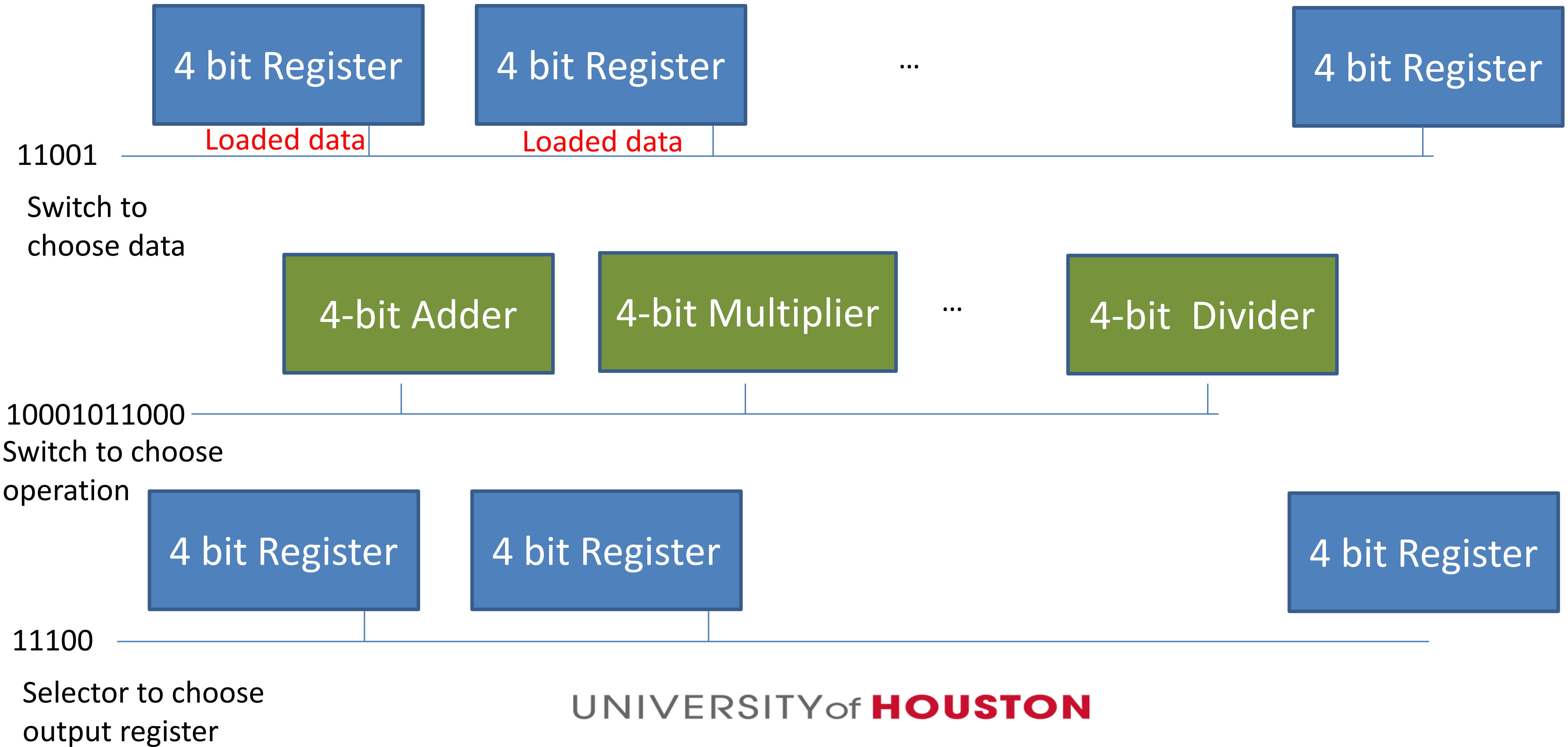
C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

https://learnabout-electronics.org/Digital/dig42.php

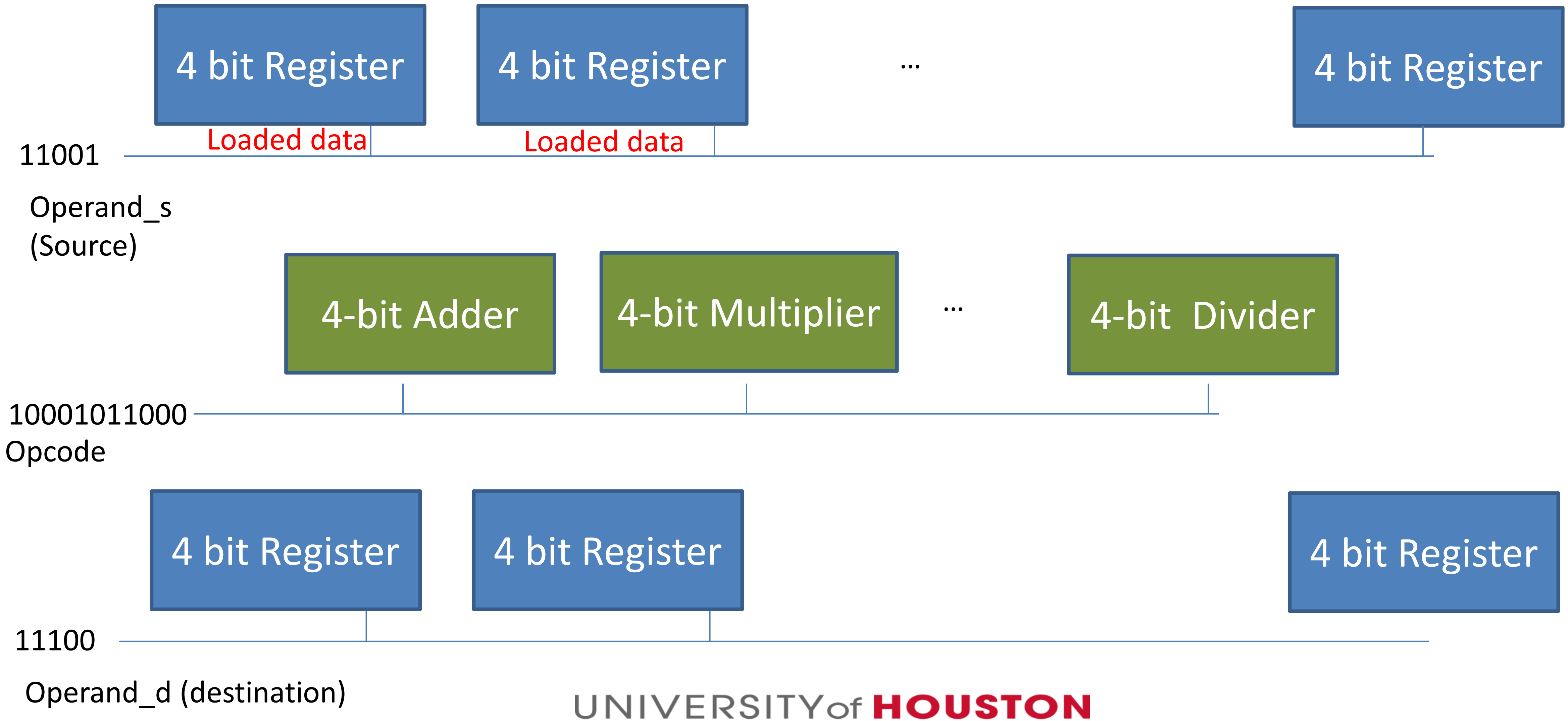
Instruction



Instruction



Instruction



Instruction Example

Opcode	Operand_s1	Operand_s2	Operand_d
10001011000	11001	11010	11100

Instruction : 10001011000 11001 11010 11100

Instruction are represented in binary form. Stored in memory.
The only language a computer understand.
Byte code, machine code, …

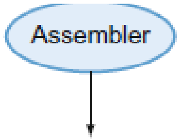
Levels of Program Code

- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

Binary machine language program (for ARMv8)

```
00000000101000100000000100011000
0000000010000010000100000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

Levels of Program Code



- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

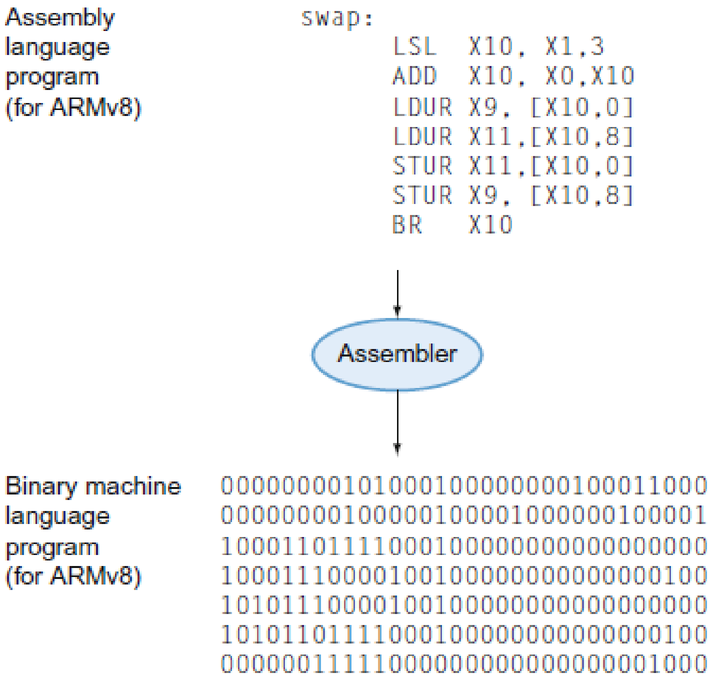
Binary machine language program (for ARMv8)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

Levels of Program Code

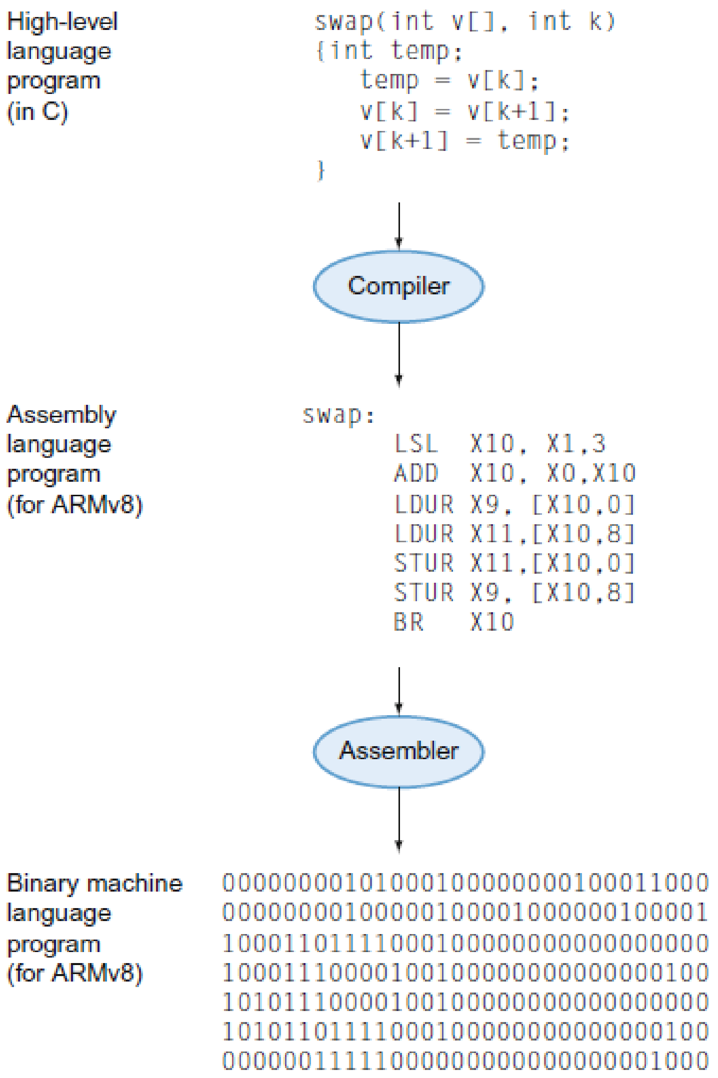
- Assembly language
  - Textual representation of instructions

- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



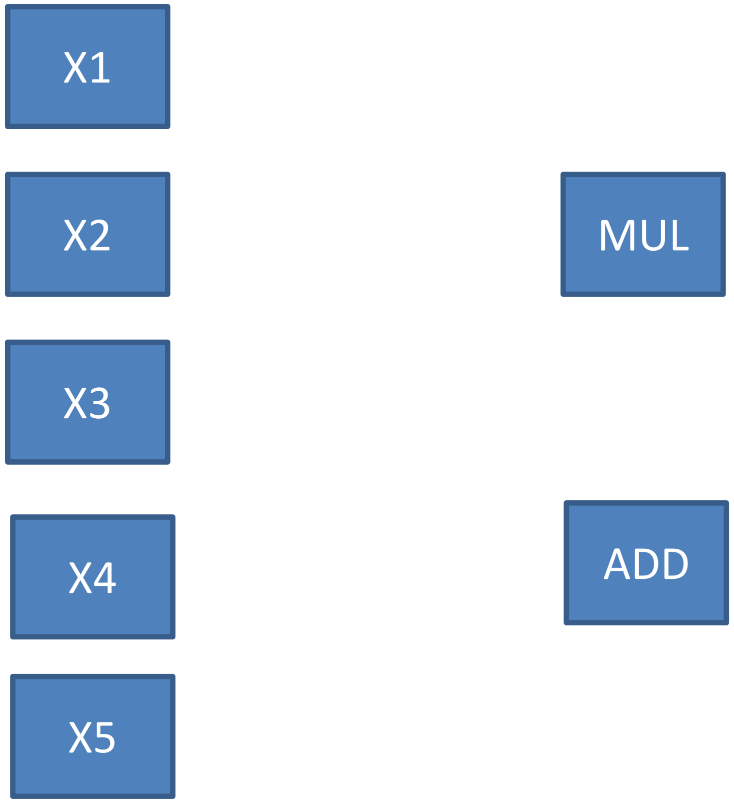
Computer Architecture: **Great Ideas**

- Use ***abstraction*** to simplify design

Execution in Sequence

- CPU executes instructions in sequence.

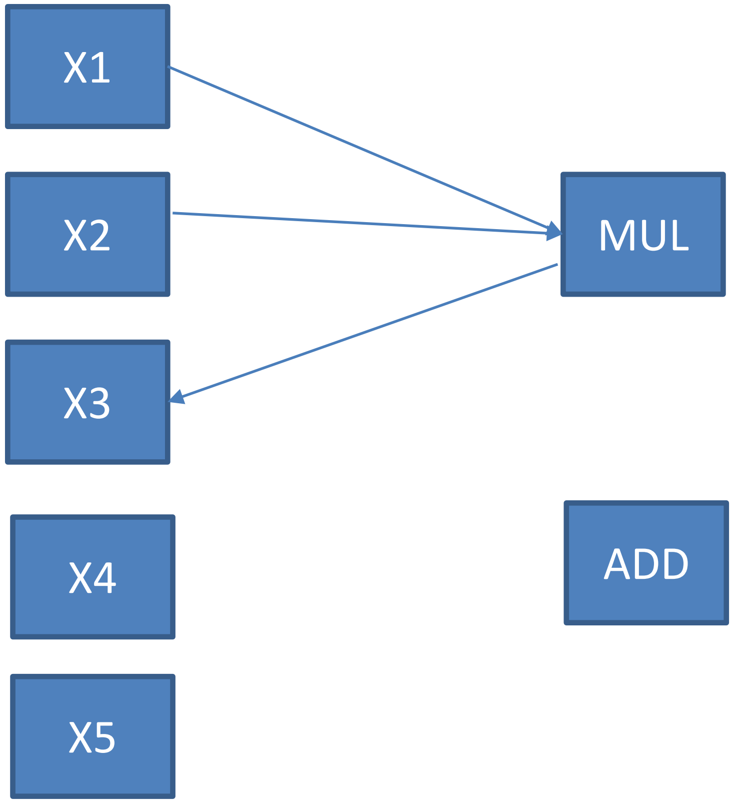
MUL X3, X1, X2
ADD X5, X3, X4



Execution in Sequence

- CPU executes instructions in sequence.

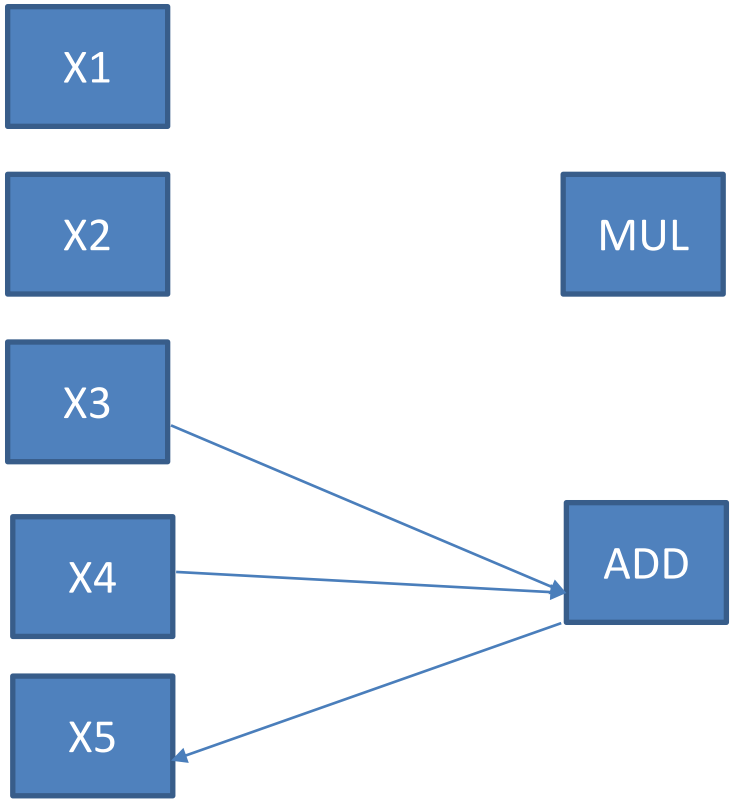
MUL X3, X1, X2
ADD X5, X3, X4



Execution in Sequence

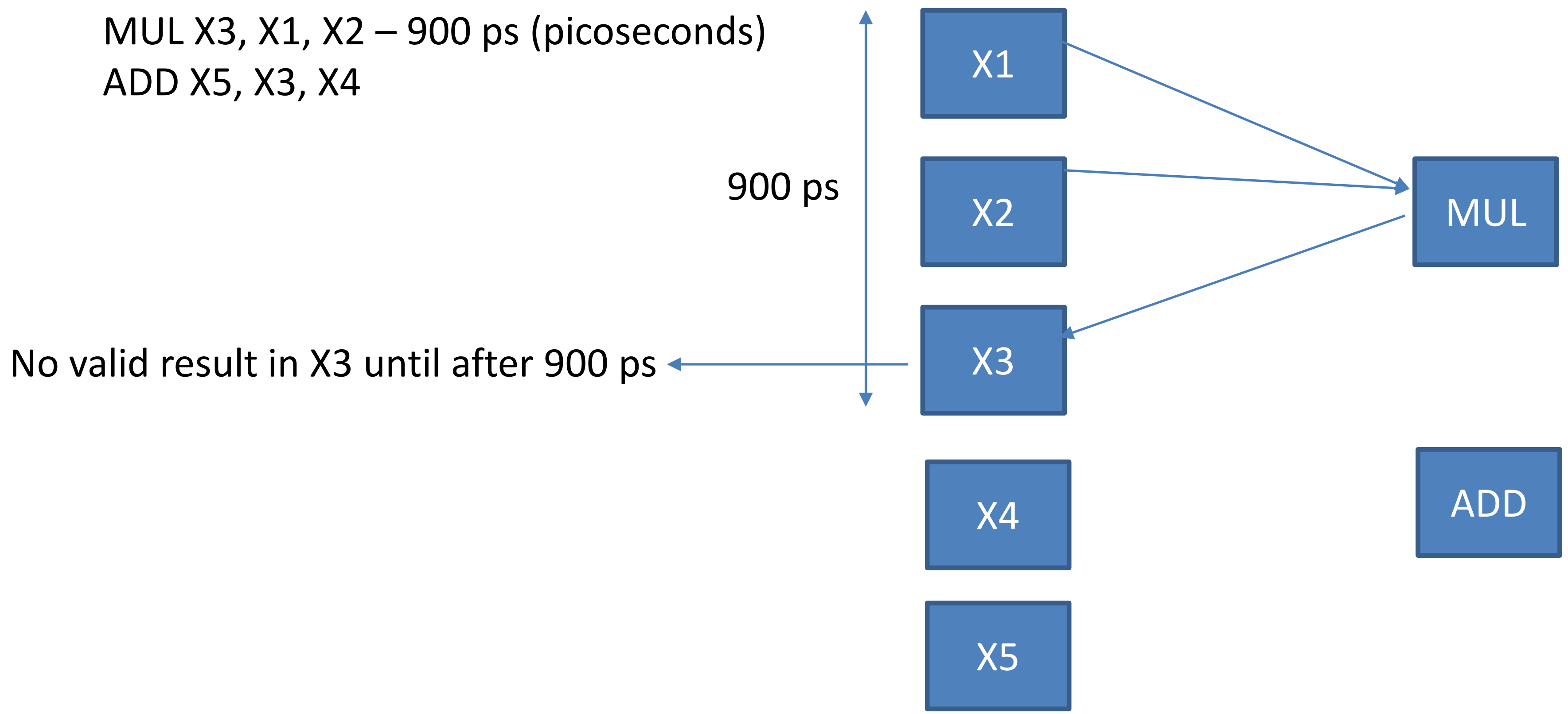
- CPU executes instructions in sequence.

MUL X3, X1, X2
ADD X5, X3, X4



Execution in Sequence

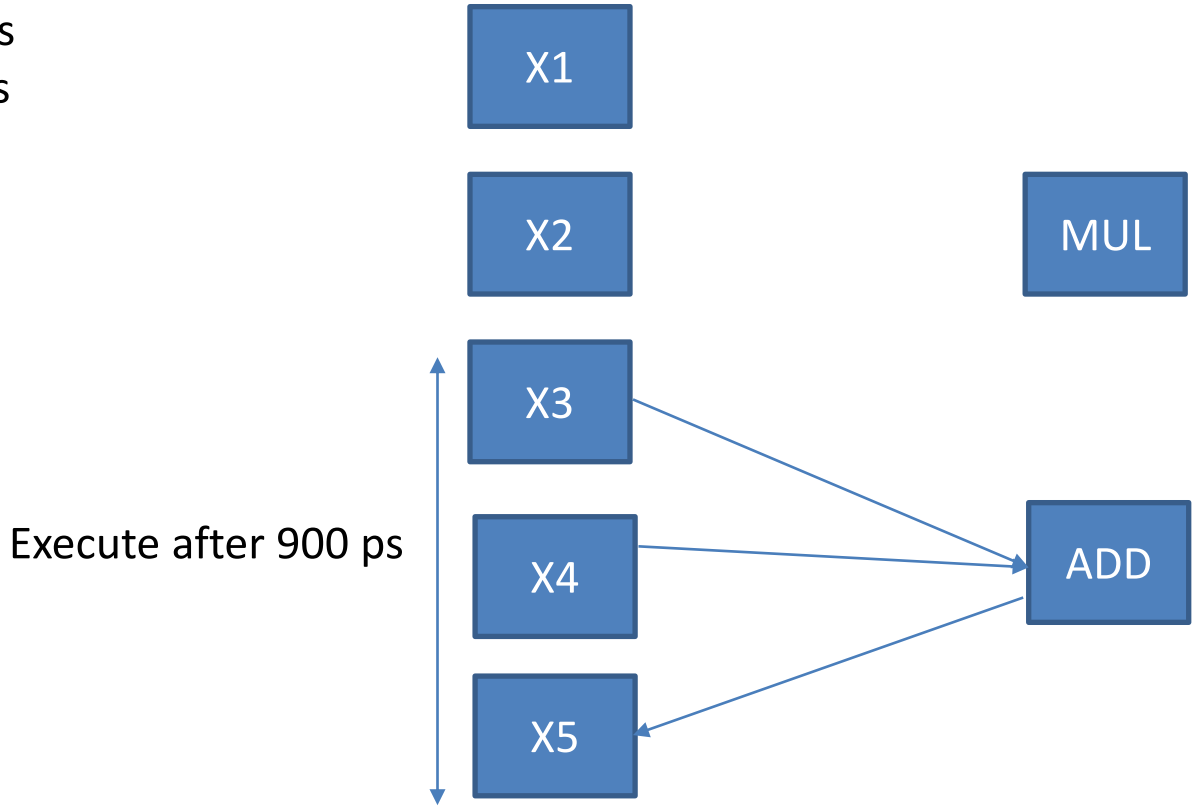
- CPU executes instructions in sequence.



Execution in Sequence

- CPU executes instructions in sequence.

MUL X3, X1, X2 – 900 ps
ADD X5, X3, X4 – 200 ps



Execution in Sequence

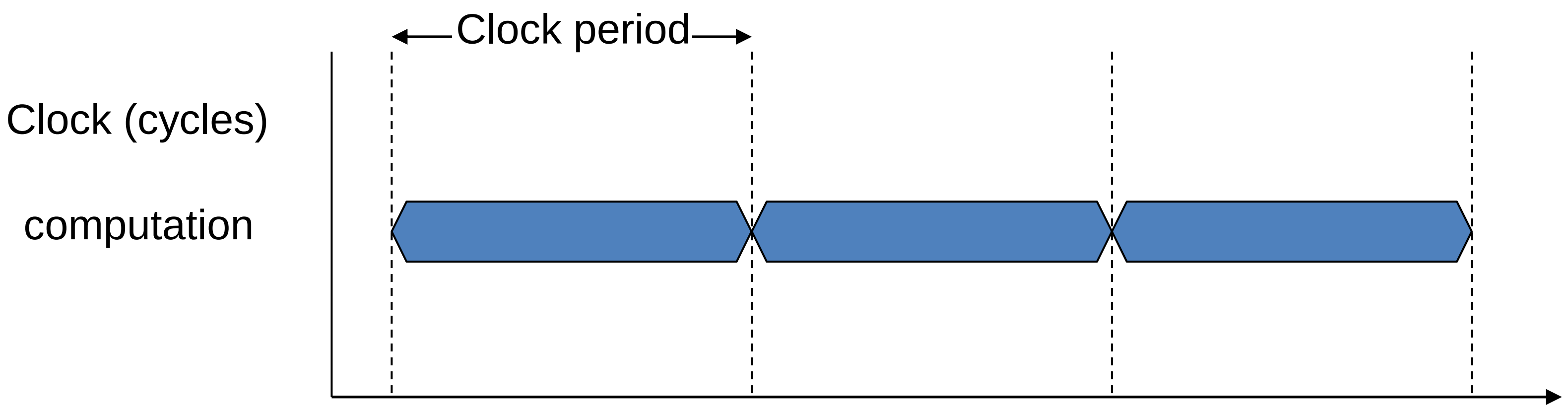
- CPU executes instructions in sequence.

Operation	Time (ps)
Add	200
Mul	900
Div (Max)	1200

- Hundreds of instructions.
  - Too complicated to compute how much to wait.
- Choose the largest values as the **clock period** for all instructions.
- All instructions are executed for that period of time.

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.250ns = 250×10^{-12}s

CPU Clocking

- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate in Hertz): cycles per second

$$= \frac{1}{(250 \times 10^{-12})} Hz = \left(\frac{1000}{250}\right) * 10^{9} Hz = 4 * 10^{9} Hz$$

$$= 4 * 10^{6} \boldsymbol{K}Hz = 4 * 10^{3} \boldsymbol{M}Hz = \boldsymbol{4\ GHz}$$

CPU Clocking

- MUL X3, X1, X2

  (wait 1200ps)

- ADD X5, X3, X4

  (wait 1200ps)

- Next Instruction

CPU Clocking

- MUL X3, X1, X2

  (wait 1200ps)
- ADD X5, X3, X4

  (wait 1200ps)
- Next Instruction

How does a computer know 1200 ps has passed?

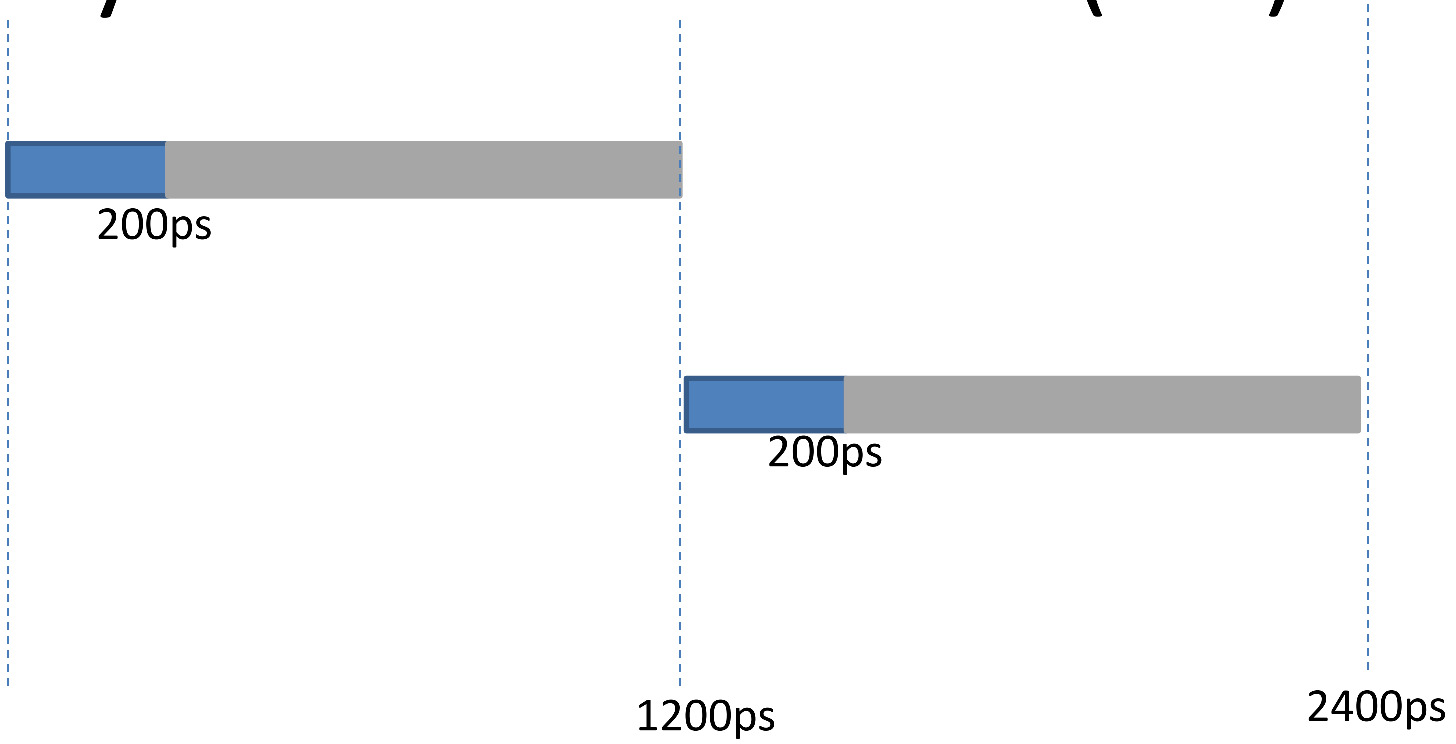
Clock Cycles Per Instruction (CPI)

- ADD X3, X1, X2

  (takes 200ps)

- ADD X5, X3, X4

  (takes 200ps)



MUL X3, X1, X2 → 1200ps

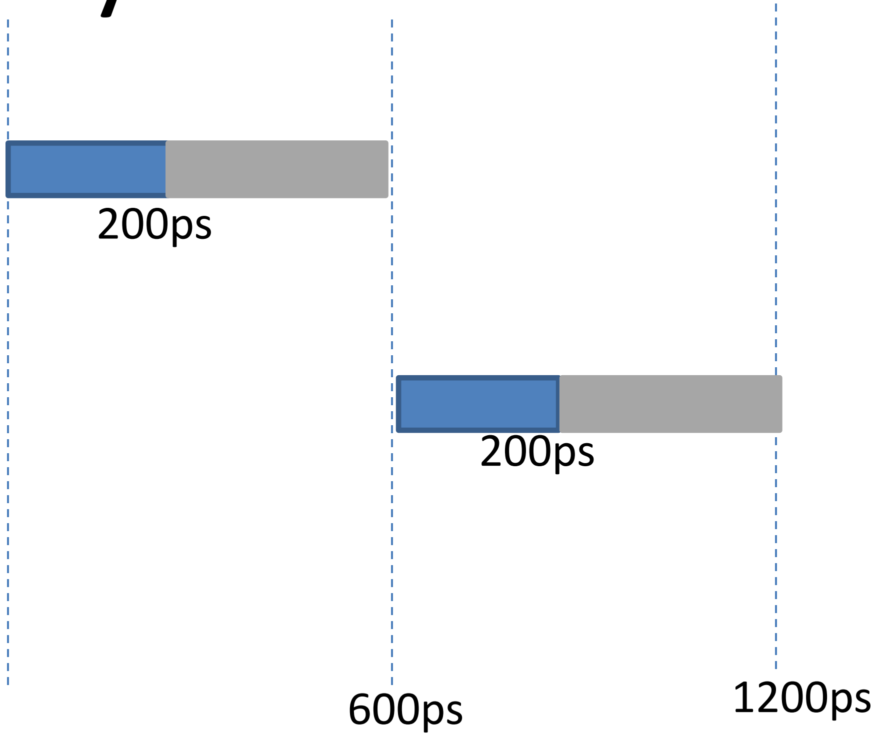
Clock Cycles Per Instruction (CPI)

- ADD X3, X1, X2

  (takes 200ps)

- ADD X5, X3, X4

  (takes 200ps)



DIV X3, X1, X2 → 600ps X 2Clock cycles

Clock Cycles Per Instruction (CPI)

- All take **at-least** one clock cycle.
- Some instructions can take more than one clock cycle.
- If the instruction set has only **three** instructions

Operation	Clock Cycles
Add	1
Mul	2
Div	12

Average Clock Cycle Per Instruction (CPI) = $\frac{\mathbf{1+2+12}}{\mathbf{3}} = \mathbf{5}$

Performance

Defining Performance

- Which airplane has the best performance?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)
Boeing 777	375	4630	610
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

Defining Performance

- Which airplane has the best performance?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y" or
- "Speedup of X over Y " is

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B
  - Speedup of A over B is 1.5

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Time

Instruction	Clock Cycles	Cycle Time (ps)
1. MUL X3, X1, X2	2	600
2. ADD X5, X3, X4	1	600
3. Div X7, X3 X6	12	600
CPU Time	**15 (Total Clock Cycles)**	**15 * 600 = 9000 ps = 9 ns**

CPU Time = Total Clock Cycles X Clock Cycle Time

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

MUL X3, X1, X2 ➔ 1200ps ➔ ⬇600ps X ⬆2Clock cycles

Clock Cycles Per Instruction (CPI)

- All take **at-least** one clock cycle.
- Some instructions can take more than one clock cycle.
- If the instruction set has only **three** instructions

Operation	Clock Cycles
Add	1
Mul	2
Div	12

Average Clock Cycle Per Instruction (CPI) = $\mathbf{\frac{1+2+12}{3} = 5}$

CPU Time using CPI

Instruction
1. MUL X3, X1, X2
2. ADD X5, X3, X4
3. Div X7, X3 X6

CPI = 5

Instruction Count = 3

Clock Cycle Time = 600ps

CPU Time = 5 *3 * 600 = 9 ns

CPU Time = (Instruction Count X CPI) X Clock Cycle Time

Total Clock Cycles

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed