

Computer Architecture Midterm Recap

23 October, 2025

Recap by @scruffyscarf

Agenda

1. Introduction to Computer Architecture
2. Fundamental Ideas of Computer Architecture
3. Architectures, Performance Metrics and Logic Gates
4. Logic Circuits: Adder and Subtractor. Circuit properties
5. Hardware Building Blocks
6. Binary Multiplier, Latches, FSM for Latches
7. Flip-Flop, Synchronous and Asynchronous Circuits

Introduction to Computer Architecture

What is a computer?

1. A machine that is programmed to automatically perform sequences of arithmetic/logic instructions with some input data and to output result
2. An electronic device that operates under the control of a program stored in its memory
3. A device that accepts binary input and manipulates it for some binary result following a sequence of instructions

What is a Computer Architecture?

- Hardware organization of computer systems
- Hardware/Software interaction principles
- Performance-related computer aspects

Computer Architecture explains

- Key computer components and programs execution principles
- Hardware circuit design
- Efficient programs implementation for specific hardware
- Computer hardware customization for specific programs
- Computers principles relevant for other areas

Central Processing Unit (CPU)

- Head and brain of the computer
- Performs execution of various commands

CPU:

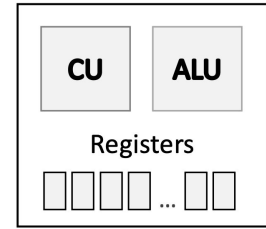
Central
Processing
Unit

Widely-known CPU architectures

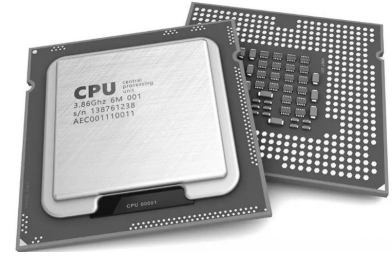
- Intel x86
- RISC-V (Reduced Instruction Set Computer)
- AMD64 (Advanced Micro Devices)
- ARM (Advanced RISC Machines)

CPUs differ in various aspects

- Speed of instructions execution
- Supported instructions set
- Energy consumption
- L1 cache storage capacity



CPU components



- Control Unit (CU) - picks the next instruction to execute from memory
- Arithmetic-Logic Unit (ALU) - performs arithmetic and logic operations
- Registers - store data required by an executed instruction

Hardware Components

System Memory

contains program instructions and data



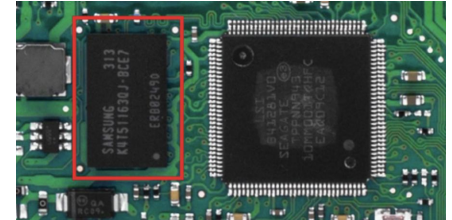
Communication Bus

exchanges data between computer components



Caches

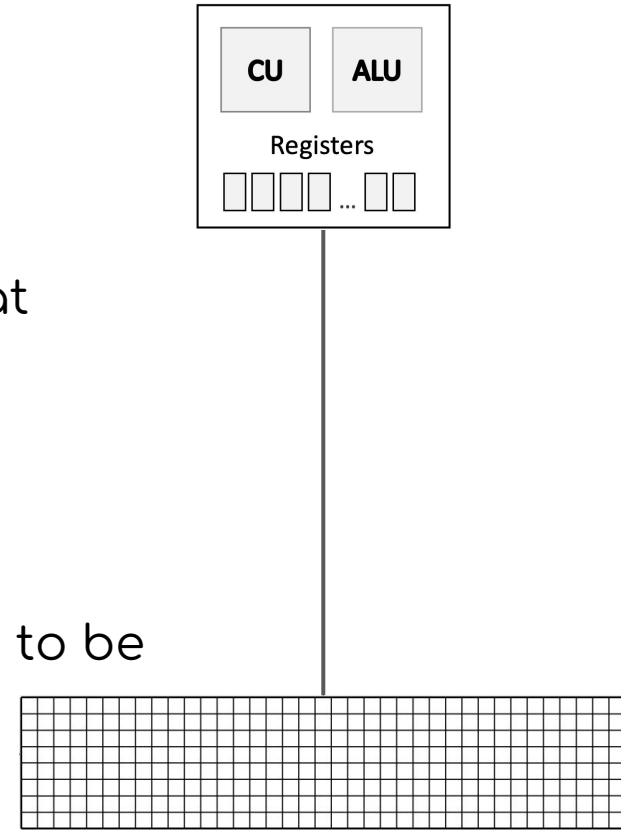
intermediate data exchange point

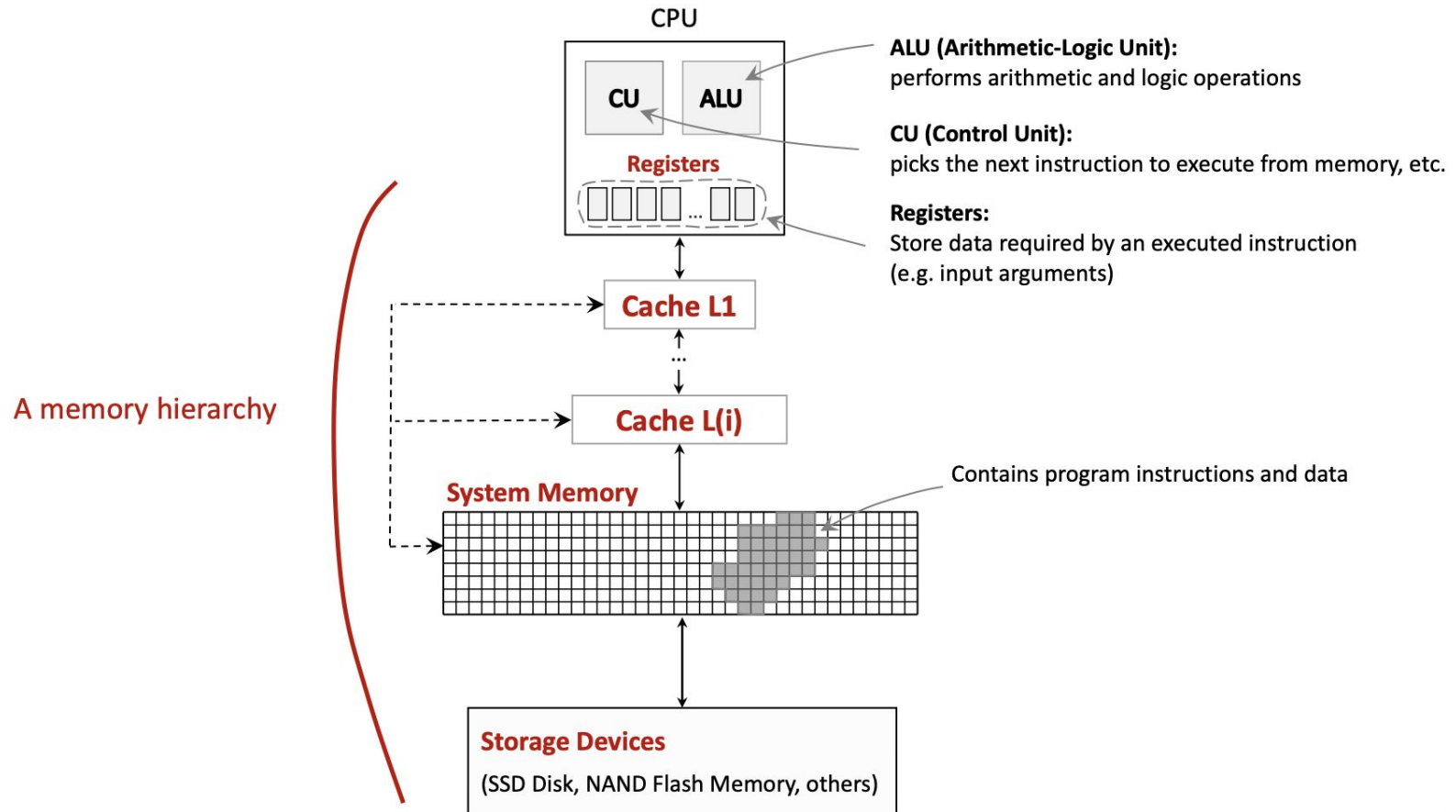


Wall problem

When the processor's speed outpaces the rate at which data can be transferred to and from the memory system.

As a result, the processor must wait for the data to be fetched from memory, which slows down its performance and limits its speed.





Different levels of CPU caches

- L1 cache

 - embedded into a CPU chip

 - the fastest

 - the smallest storage capacity

- L(i) cache

 - either on or outside of a CPU chip

 - slower, but has a larger storage capacity than L(i-1) cache



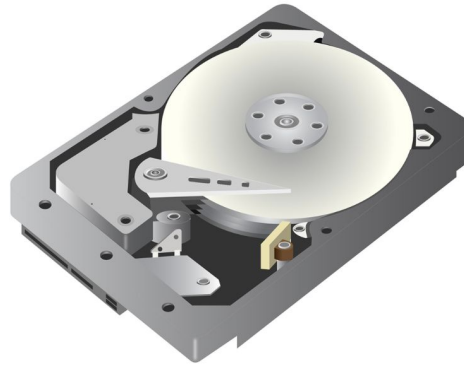
I/O Peripheral Devices

- Keyboard
- Printer
- Remote storage devices
- etc

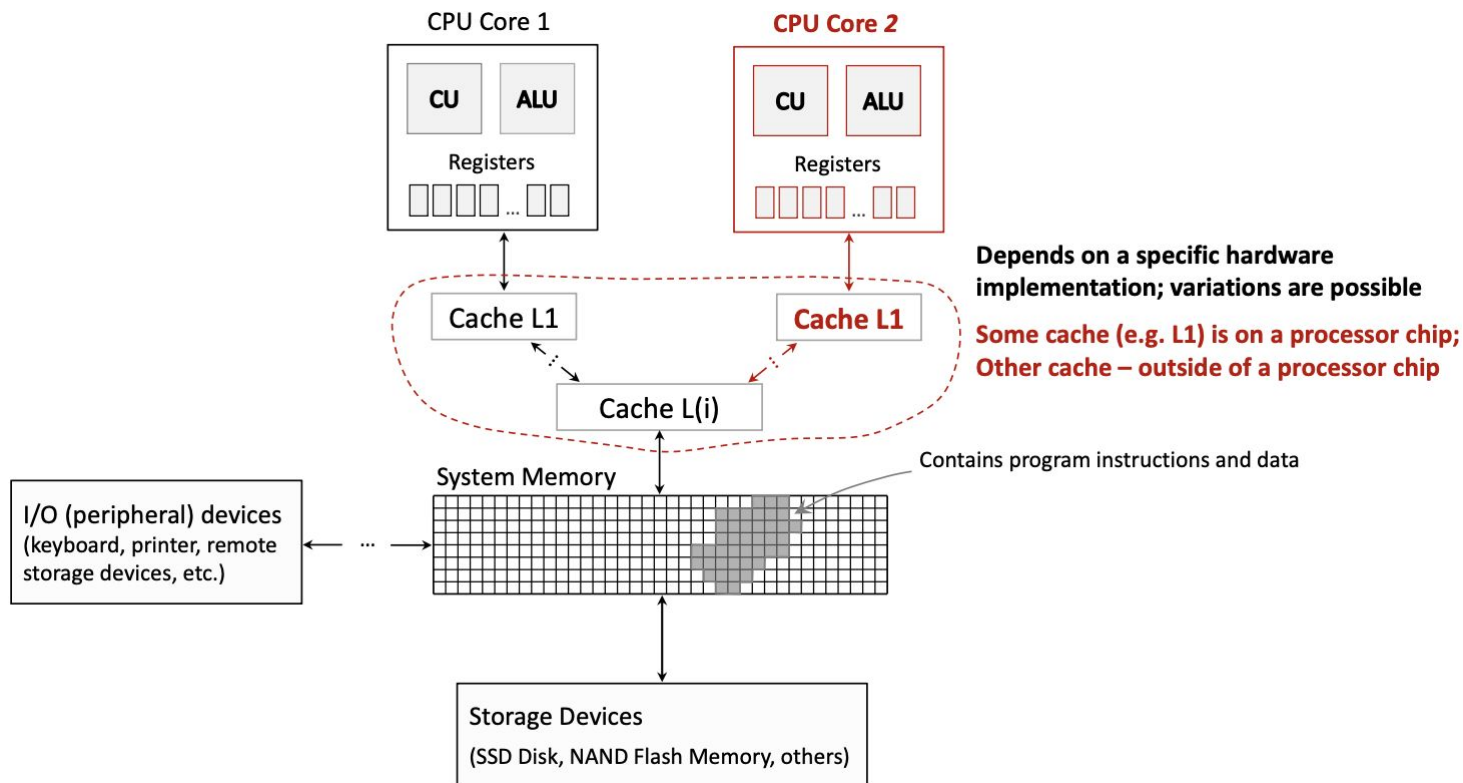


Storage Devices

- Solid State Drive (SSD)
- Hard Disk Drive (HDD)



Multicore (multiprocessor) system



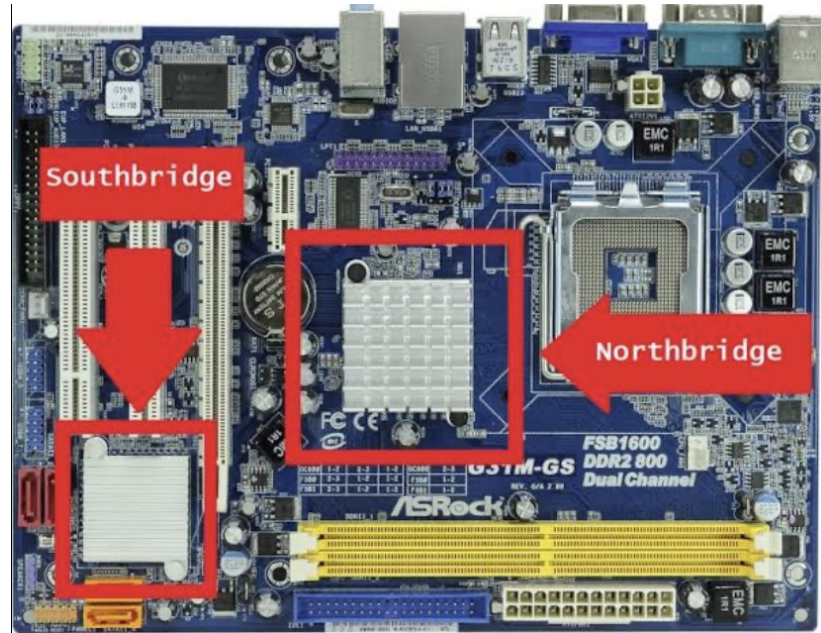
Some Performance Metrics

- CPU speed (~2-3 GHz; speedup - ~5 GHz)
- CPU speedup (~5 GHz)
- System memory storage (up to ~16-64 GBs)
- System memory access speed (~10-20 GBs/sec)
- Storage capacity (~TBs)
- Storage access speed (up to ~50-300 MBs/sec)

Northbridge vs Southbridge

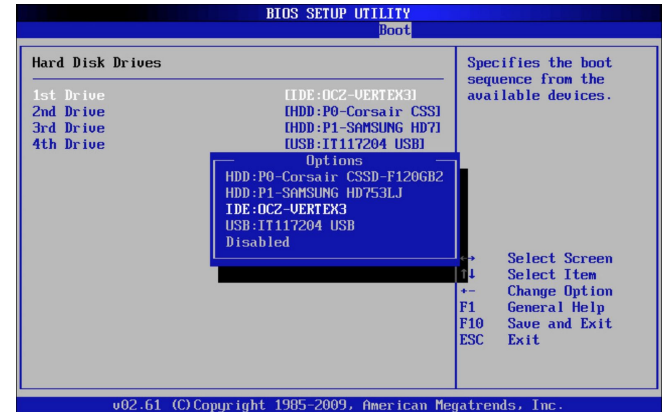
Northbridge handles the communication between the CPU, memory, and graphics card

Southbridge manages the input/output operations and connects devices like drives, Universal Serial Bus (USB) devices



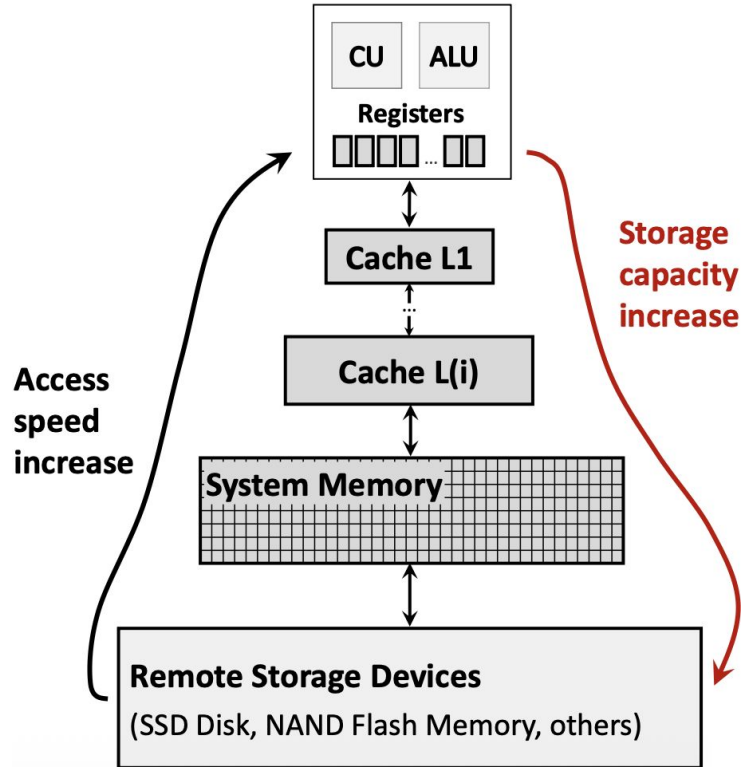
Basic Input/Output System (BIOS)

- Performs hardware testing when the computer is turned on to ensure that it is working properly and to detect errors.
- Initializes hardware, including processor, memory, disk drives, graphics card, and other components
- Turns on earlier than anyone else

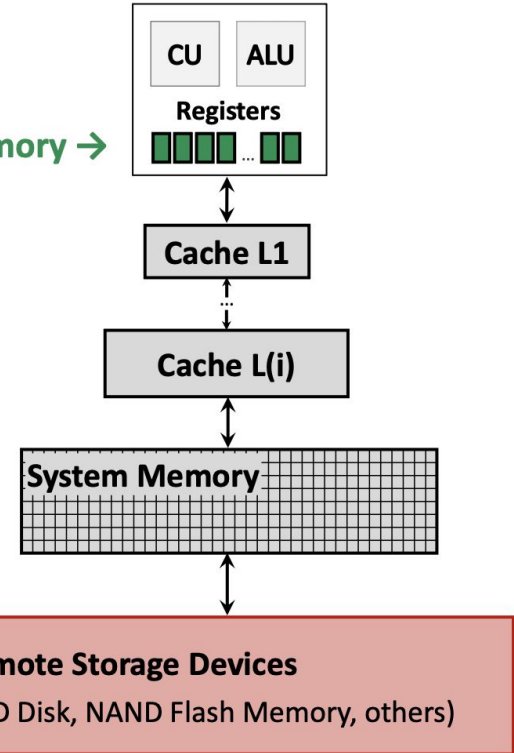


Fundamental Ideas of Computer Architectures

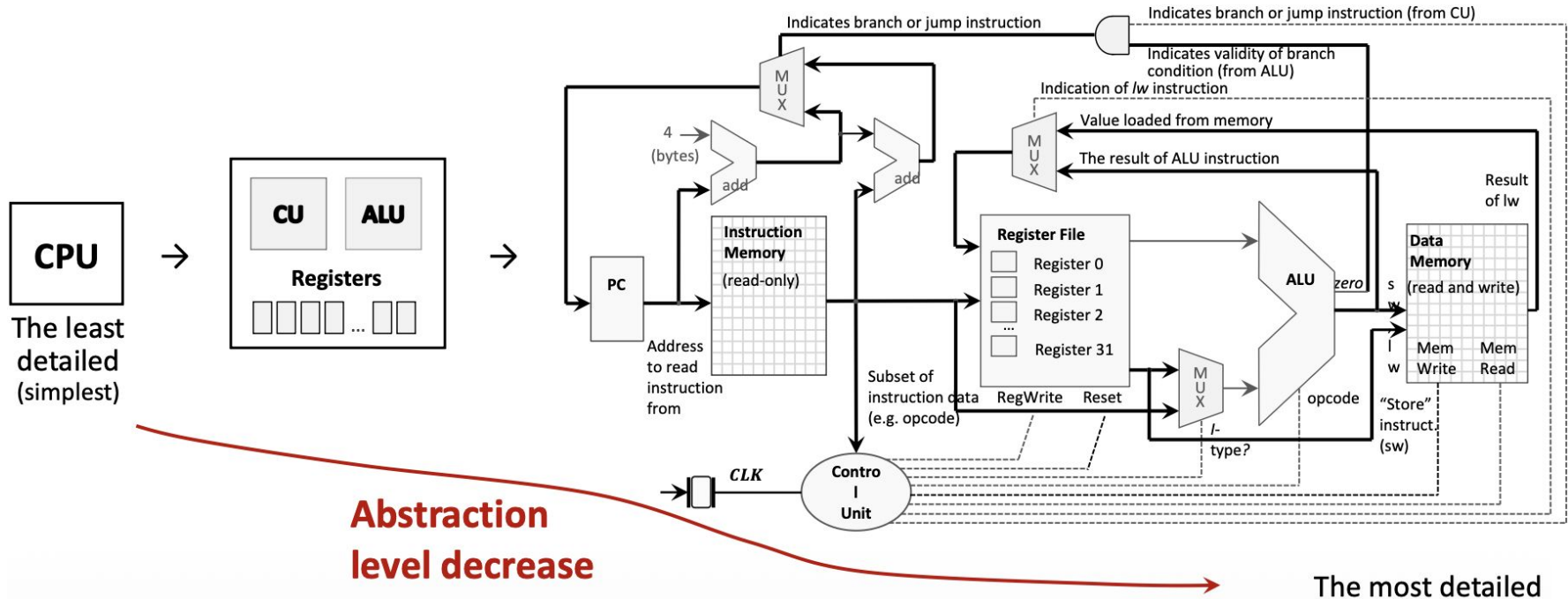
1. Hierarchy of memories



The fastest memory →

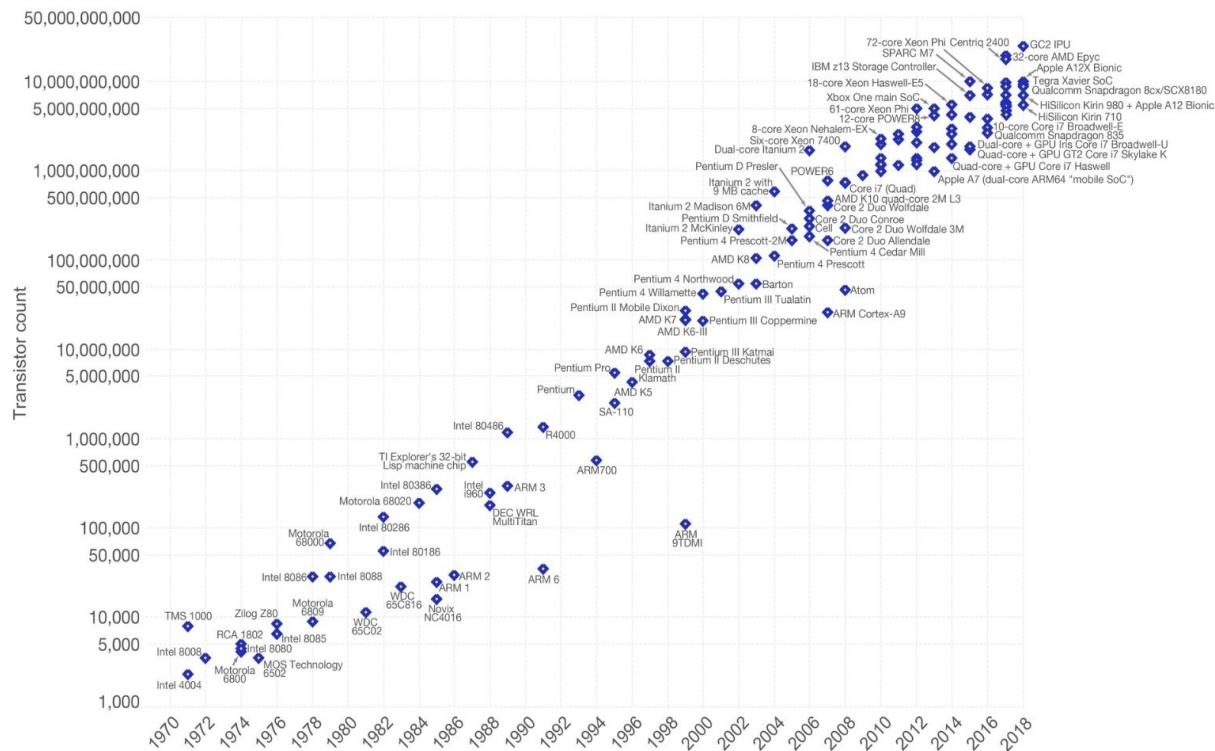
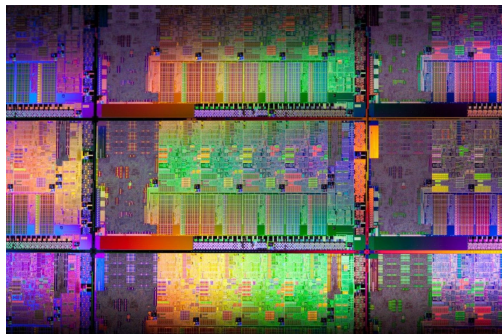


2. Design simplification via abstraction use



3. Moore's Law

The CPU execution speed doubles every 18-24 months (due to an increasing number of transistors on a CPU chip)

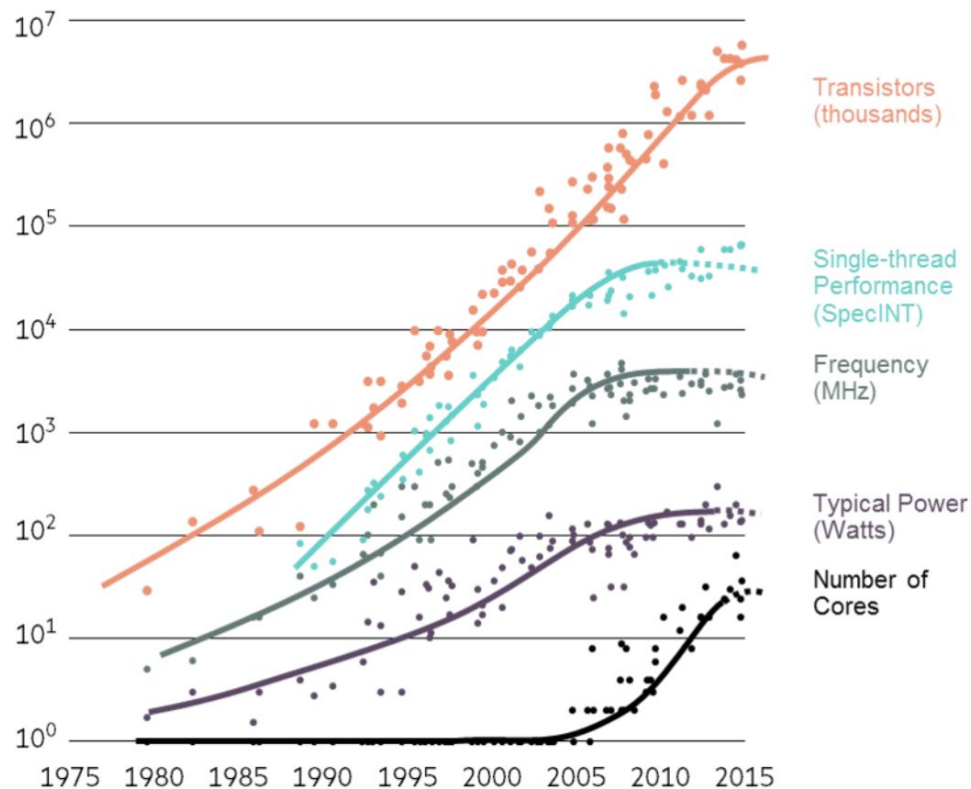


Moore's law stagnation

- Heat dissipation problem
- The limitation of the speed of light

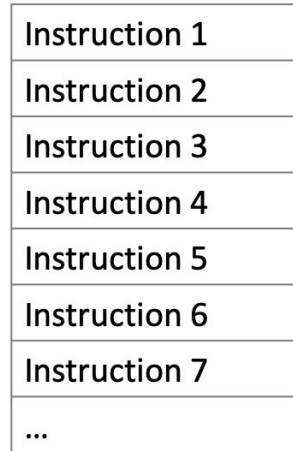
Conclusion:

There is a strong need for multiprocessor/multicore systems



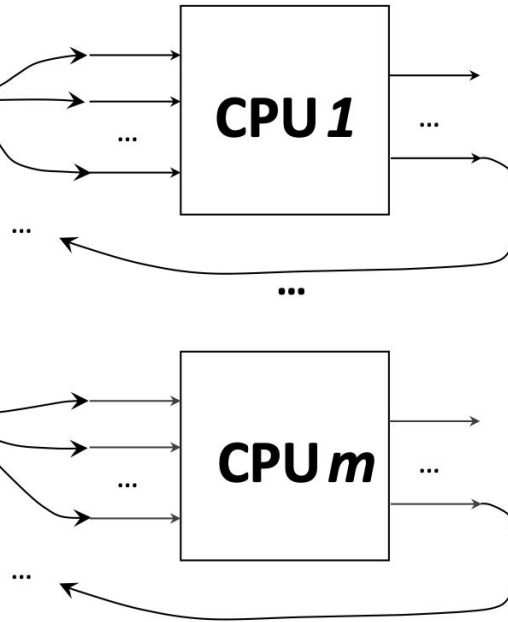
4. Performance via parallelism

Computer program
(a set of instructions to be executed);
loaded into system memory



Simultaneous
execution
(restrictions apply)

Multiprocessor (or multicore) system:



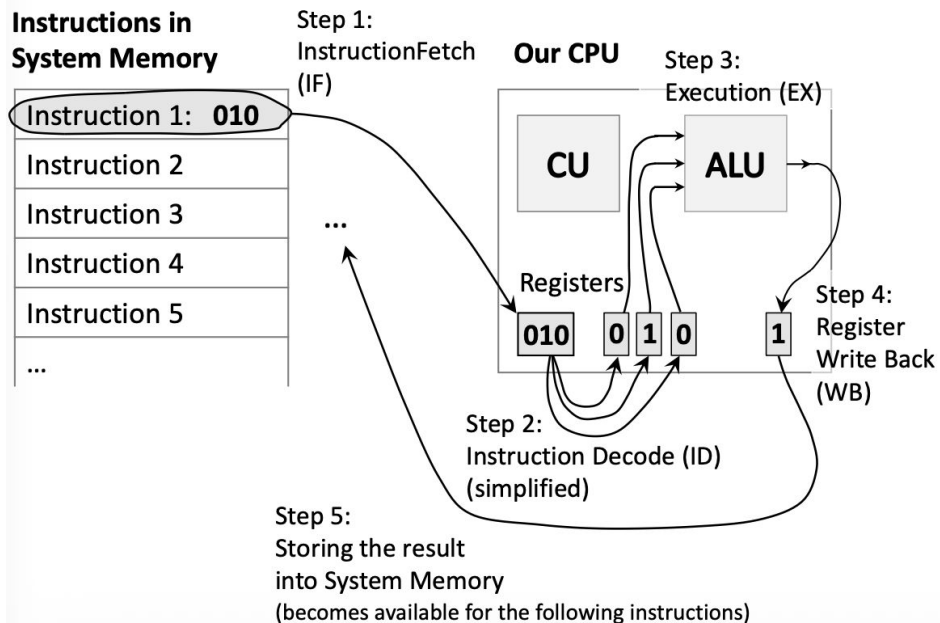
M instructions can be
executed at a time,
simultaneously

Multiprocessing aims at
speeding-up program
execution, or simultaneous
execution of multiple
programs

The major problem:
Some instructions depend on
the execution result of
previous instructions, and thus,
have to wait for their
completion

5. Performance via pipelining

Each instruction is executed in several steps:



Idea of pipelining:

The same CPU executes several instructions simultaneously, but with each instruction being at a different execution stage

Clock cycle \ Instr. No.	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

IF: Instruction Fetch

ID: Instruction Decode

EX: Execution

MEM: Memory access (optional stage)

WB: register Write Back

What is the difference of “pipelining” from the “performance via parallelism” idea?

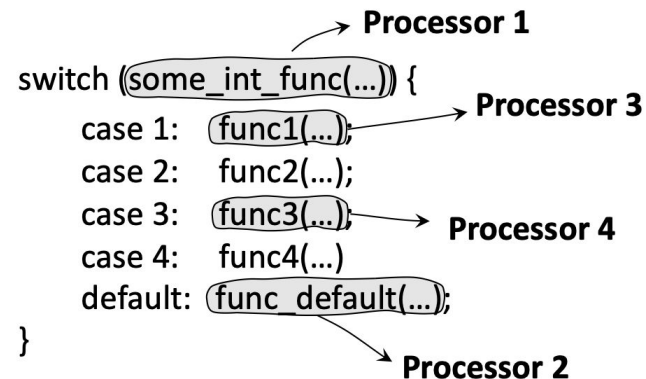
6. Performance via speculation (prediction of program flow)

Estimate the execution time or other performance factors (such as cache misses of a program on a given computer)

Illustration with a code sample

Assumptions:

- 4 processors are available;
- No dependencies between functions



For this sample code, we aim at predicting (“speculating”) 3 cases with the highest probability to occur

7. Dependability via redundancy

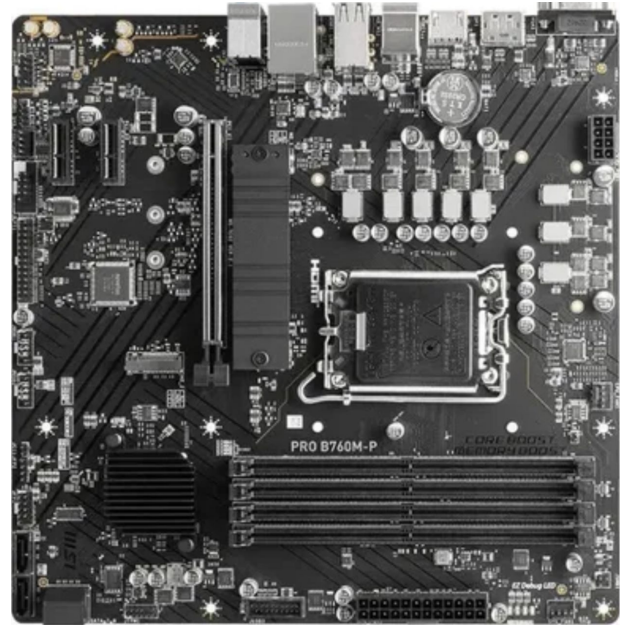
Using some redundant components (e.g. CPU or memory unit) to increase the reliability of a computer platform.

8. Make the common case fast

The computer platform should be optimized for the most common use case expected.

Making the common case fast will tend to enhance performance better than optimizing the rare case.

The common case is often simpler than rare case and hence is often easier to enhance.

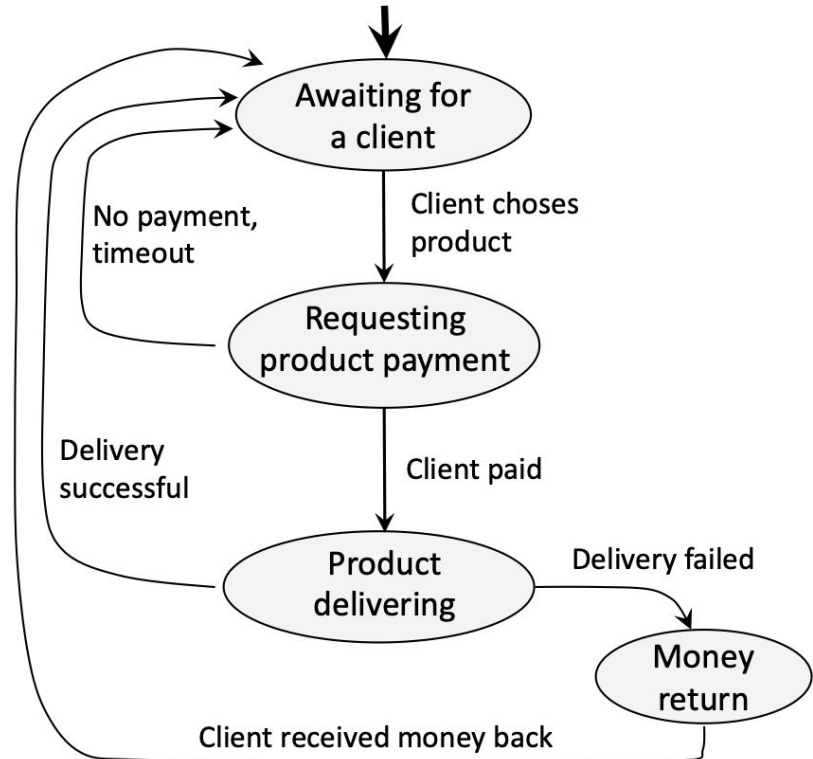


9. Finite State Machine (FSM)

Finite set of states and transitions between them

A convenient way to model systems behaviour

Sample FSM for a vending machine:



Architectures, Performance Metrics, and Logic gates

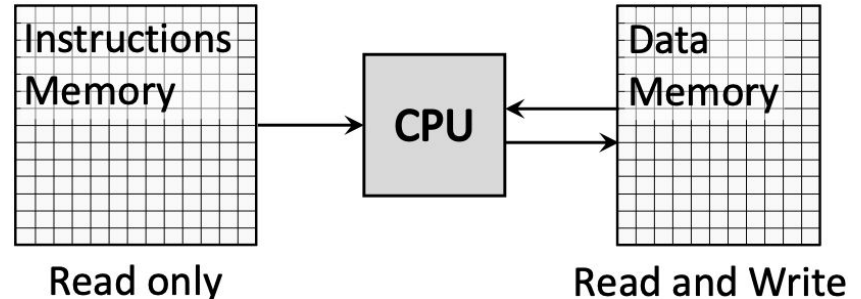
Harvard Architecture

Principle: Instruction and data are in different physical memory units

Pros: Lower performance overheads

Cons: Complex hardware implementation

Usage: Embedded systems



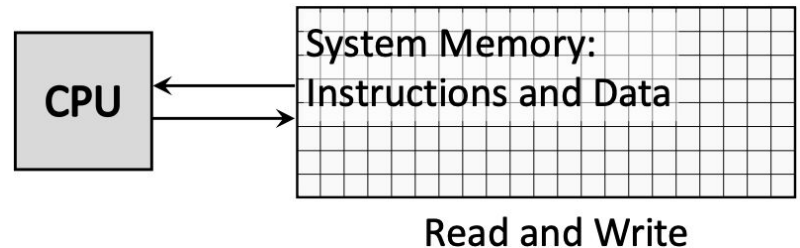
Von-Neumann Architecture

Principle: Instruction and data stored in System Memory

Pros: Simpler memory management, Simpler hardware implementation

Cons: Higher performance overhead

Usage: All general-purpose computers



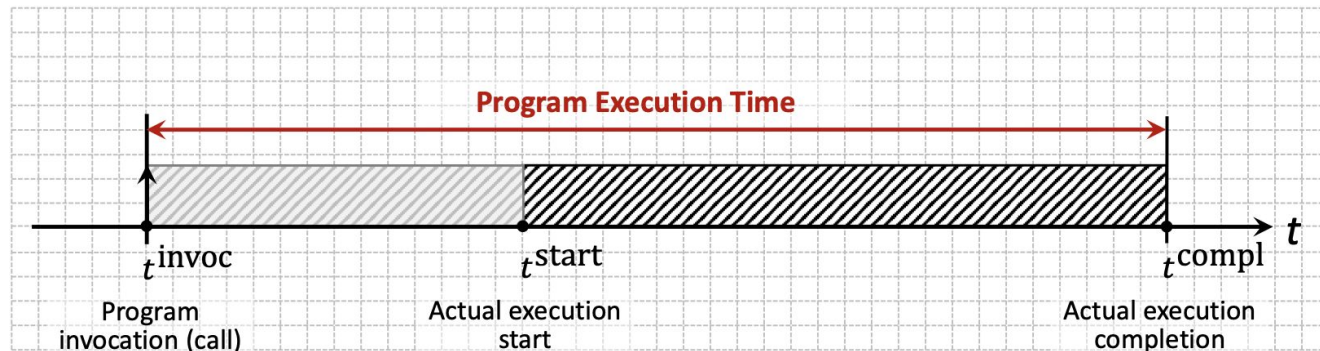
Performance Metrics

Latency (Delay, Response Time)

Delay between a program invocation and an actual execution start time

Execution Time

Time between a program invocation and its completion

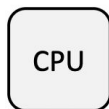


Performance Metrics

Bandwidth

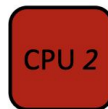
Theoretical bound on the number of instructions to execute concurrently

Uniprocessor



Not more than
1 job at a time
(bandwidth is 1 job)

Multiprocessor with bandwidth m :



Not more than m jobs to
execute concurrently

Throughput

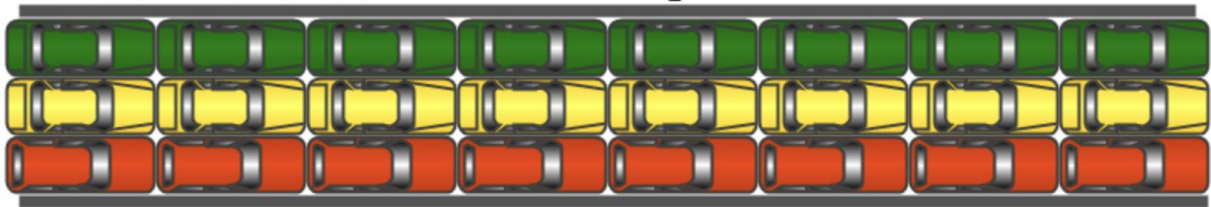
Average number of concurrently executing instructions over some time

Utilization

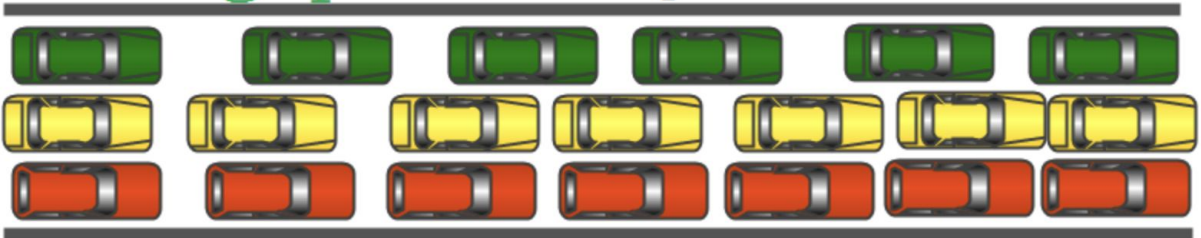
Time percentage of an effective component use

Bandwidth vs Throughput

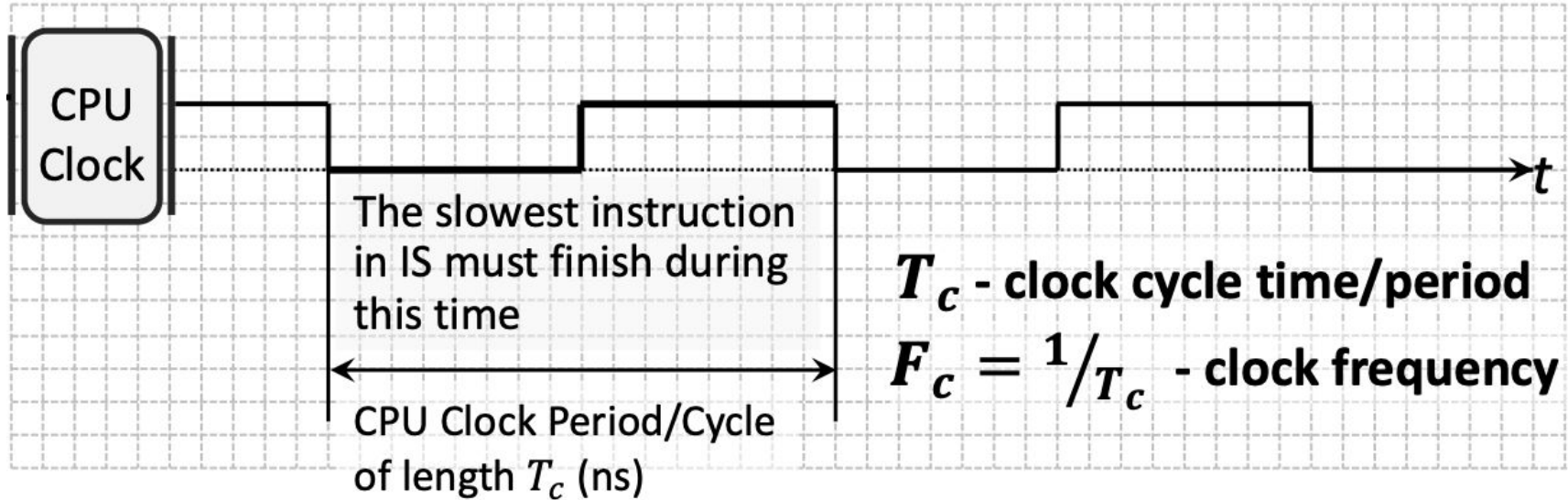
Bandwidth 24 Cars per second



Throughput 20 Cars per second

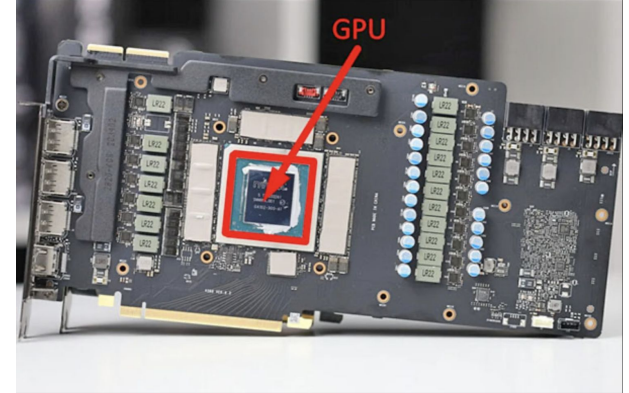
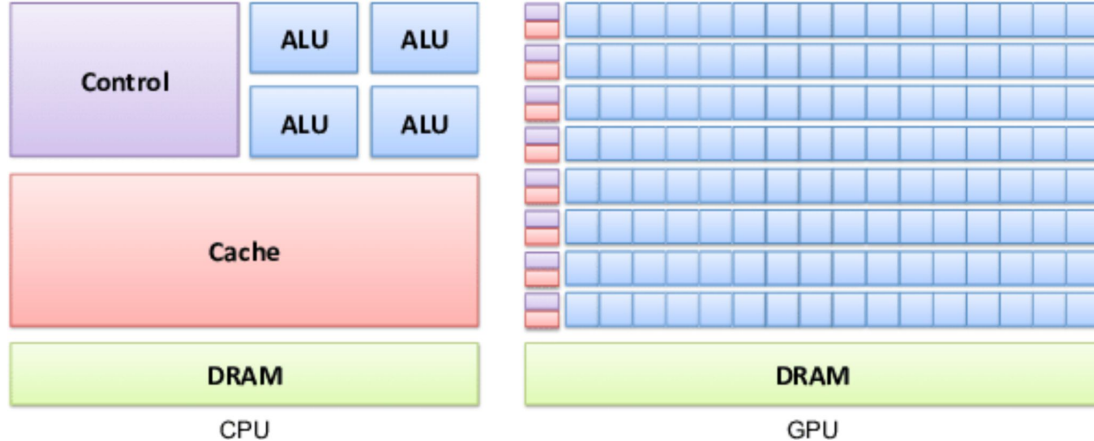


CPU Clock and Frequency



CPU vs GPU

Graphical Processing Unit (GPU) - the component of the computer that is responsible for image and video processing



Average Performance Characteristics

Device		Operation Speed	Storage Capacity
CPU (4-6 cores)		2.0-3.5 GHz Up to 5.0 GHz (Turbo mode)	32 or 64 Bits per register
GPU (700-2000 cores)		1.5-2.0 GHz Simple IS; parallel execution ~100 times faster than CPU	32 Bits per register
L1 Cache		The same as CPU speed; ~100 times faster than RAM	32-128 KB
L2 Cache		~25 faster than RAM	256-512 KB
L3 Cache		~2 faster than RAM	32-64 MB
System Memory		18-36 GB/sec	8-32 GB
GPU Memory		300-500 GB/sec (due to high throughput)	8-16 GB
Bus	RAM (DDR4)	18-36 GB/sec (64 lines)	N/A
	PCIe (4.0)	8 GB/sec (4 lines)	
	USB (3.0)	0.5 GB/sec (1 line)	
Storage	SSD	0.5-2 GB/sec	250-500 GB
	HDD	0.1-0.15 GB/sec	100-150 GB

Amdahl's Law

$$\text{speedup} = \frac{t^{1 \text{ proc}}}{t^{m \text{ procs}}} = \frac{1}{x + \frac{1-x}{m}}$$

Provides a theoretical upper bound on the execution speedup due to multicores use

Sample 1:

Given 5 cores

Given 40 instructions (100 %):

10 independent instruction (25%)

30 dependent instructions (75%)

$$\text{speedup} = \frac{1}{0.75 + \frac{1-0.75}{5}} = 1.25$$

Sample 2:

Given 8 cores

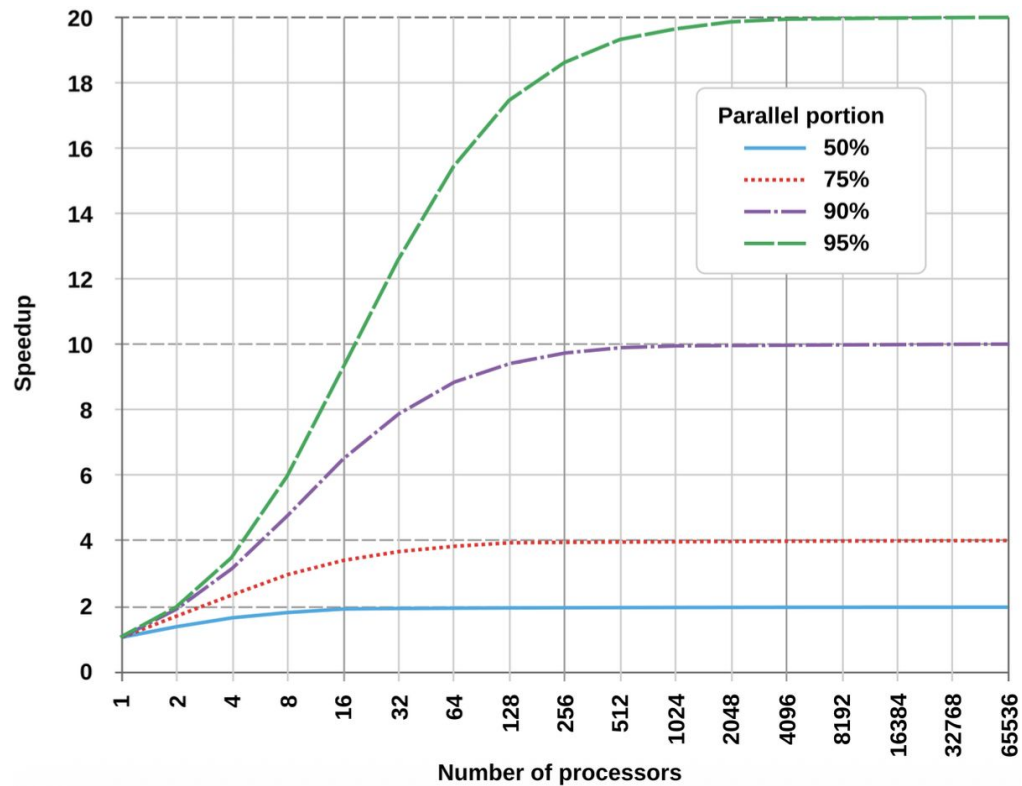
Given 50 instructions (100 %):

0 independent instruction (0%)

50 dependent instructions (100%)

$$\text{speedup} = \frac{1}{1 + \frac{1-1}{8}} = 1$$

Amdahl's Law



Boolean function

A function taking one or multiple binary inputs and producing exactly one binary output

Sample boolean variables:

$$A, B, C, D, F \in \{0,1\}$$

Functions in one notation:

$$F(A, B) = A * B$$

$$F(A, B, C) = A * B + C$$

$$F(A, B, C, D) = A * (B + C) + D$$

Functions in another notation:

$$F(A, B) = \text{AND}(A, B)$$

$$F(A, B, C) = \text{OR}(\text{AND}(A, B), C)$$

$$F(A, B, C, D) = \text{OR}(\text{AND}(A, \text{OR}(B, C)), D)$$

Truth Table

A table specifying the output of a Boolean function for every combination of input values

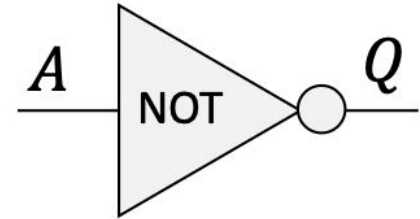
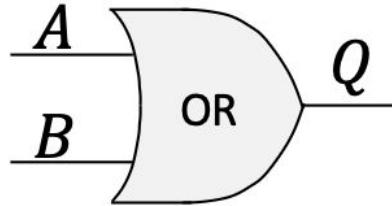
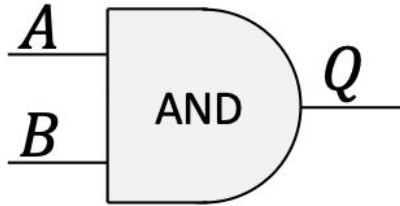
Truth table for function $F = A * B$:

A	B	$F = A * B$
0	0	0
1	0	0
0	1	0
1	1	1

Logic Gate

An abstract or physical device implementing a Boolean function

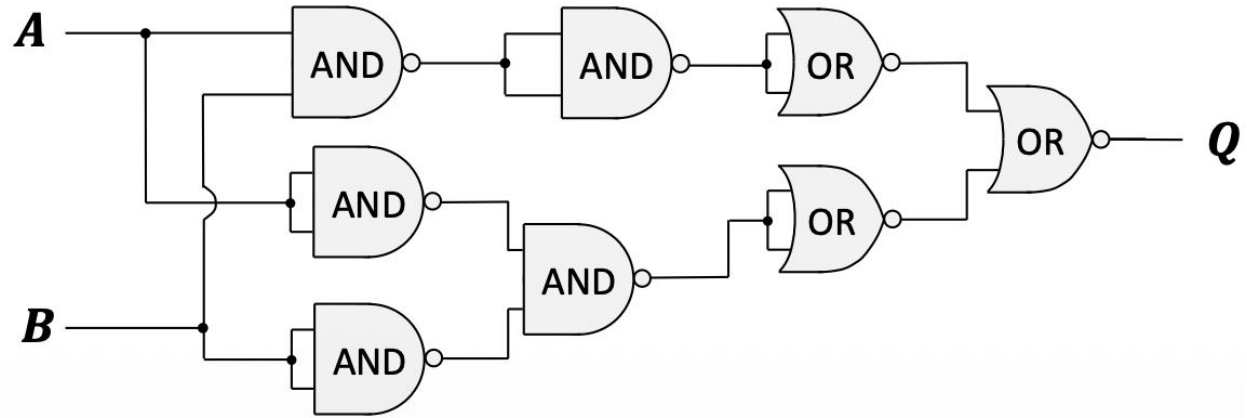
Sample logic gates:



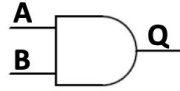
Logic Circuit

Composition of logic gates, to implement a certain boolean function

Sample logic circuit:

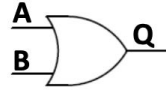


Basic Logic Gates



AND

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



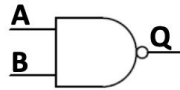
OR

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



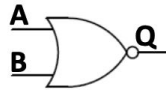
XOR

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



NAND

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



NOR

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

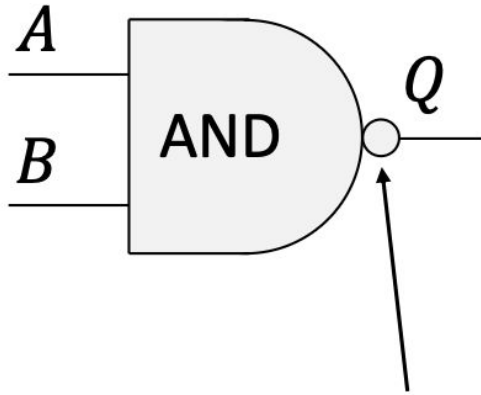


XNOR

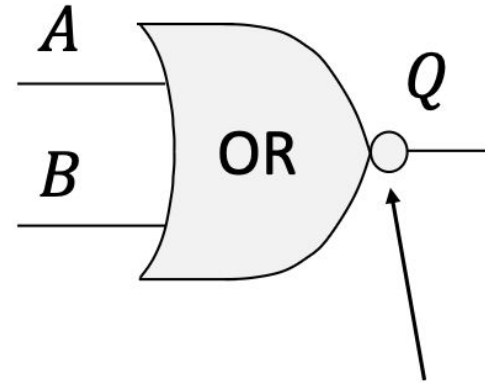
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Universal Logic Gates

Any other logic gate can be implemented through one of these

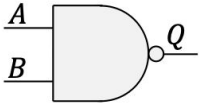
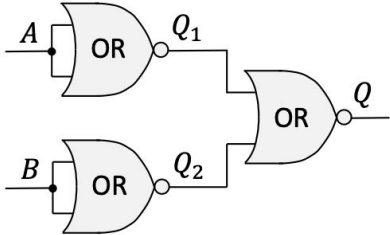
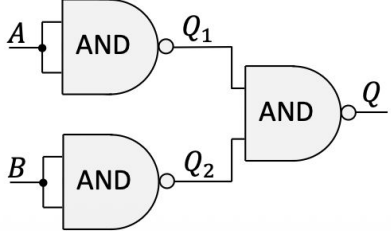
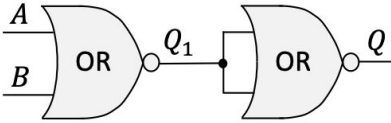


NAND = AND + NOT



NOR = OR + NOT

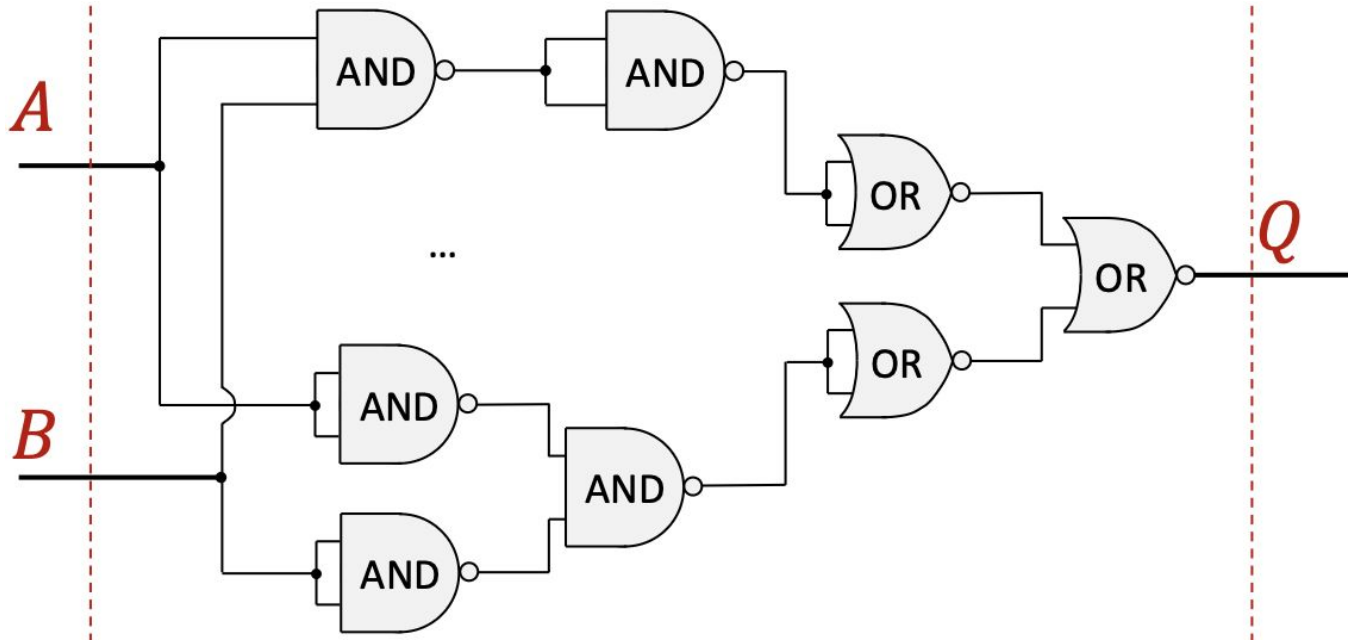
Universal Logic Gates Sample Use Cases

Universal Gate Original Gate	NAND	NOR
AND		
OR		

Logic Circuits: Adder and Subtractor. Circuit properties

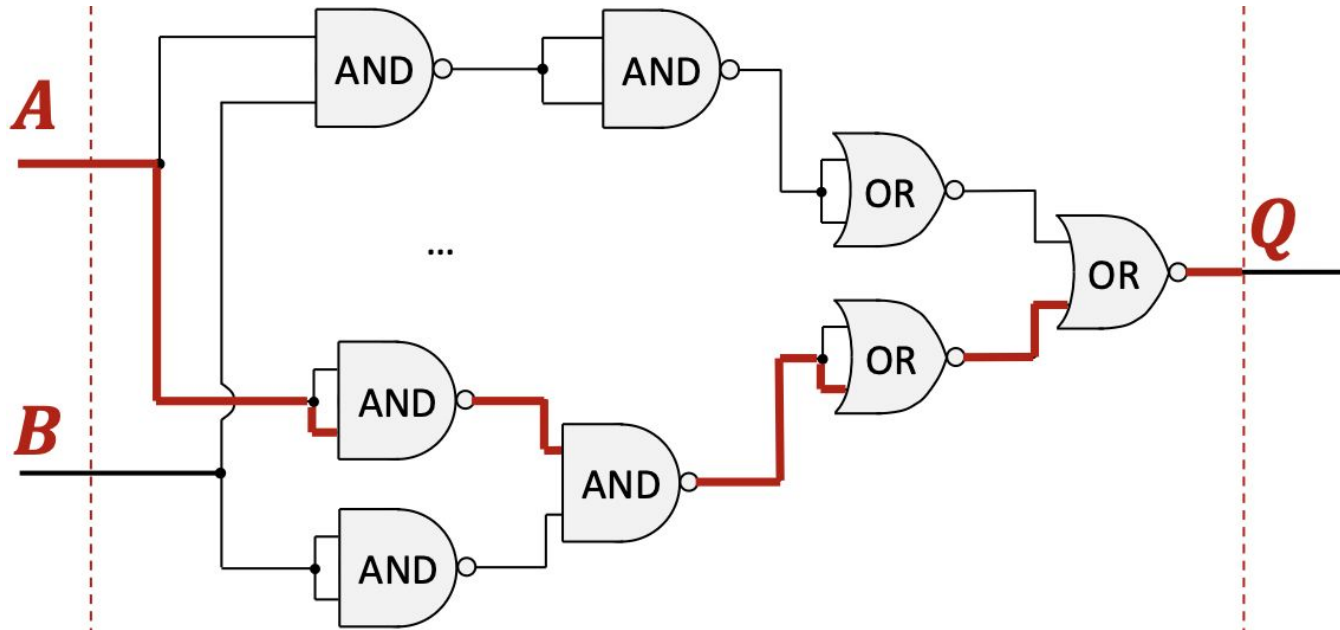
Propagation (Time) Delay

A time delay between the updates of input and output signals



Critical Path

The path taking the longest time



Binary Values Summation

Most Significant Bits (MSB)

Least Significant Bits (LSB)

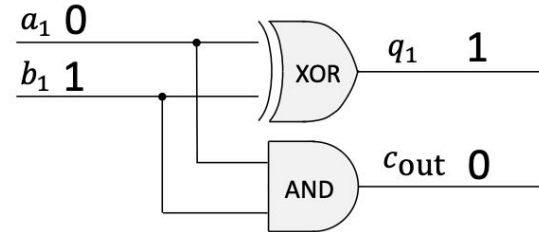
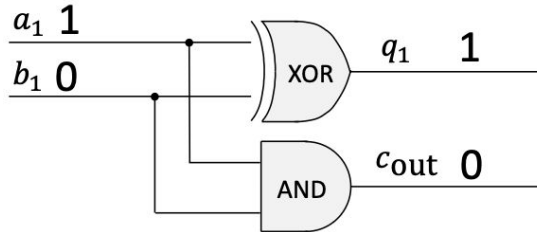
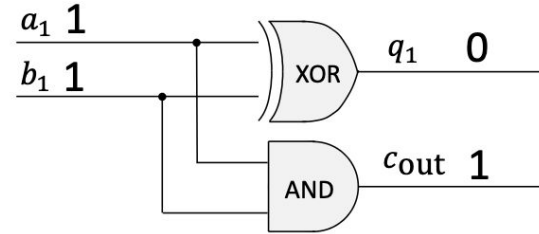
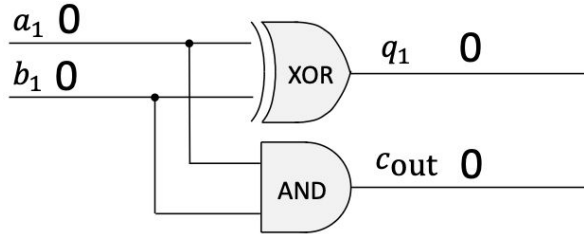
$$\begin{array}{r} + \quad \begin{array}{|c|} \hline 0 \\ \hline 0 \end{array} \quad 0 \quad 1 \quad \begin{array}{|c|} \hline 1 \\ \hline 0 \end{array} \\ \hline \end{array}$$



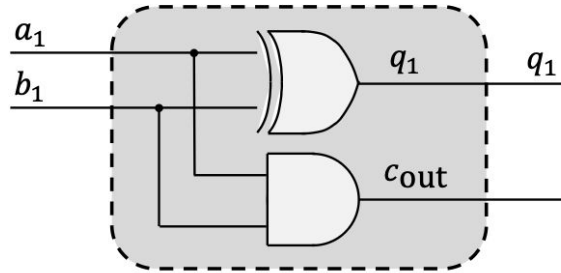
Computation starts with the least significant bits;
Excess is carried-out

Half-Adder Circuit

- No carry-in handling
- Least Significant Bits (LSB) summation only

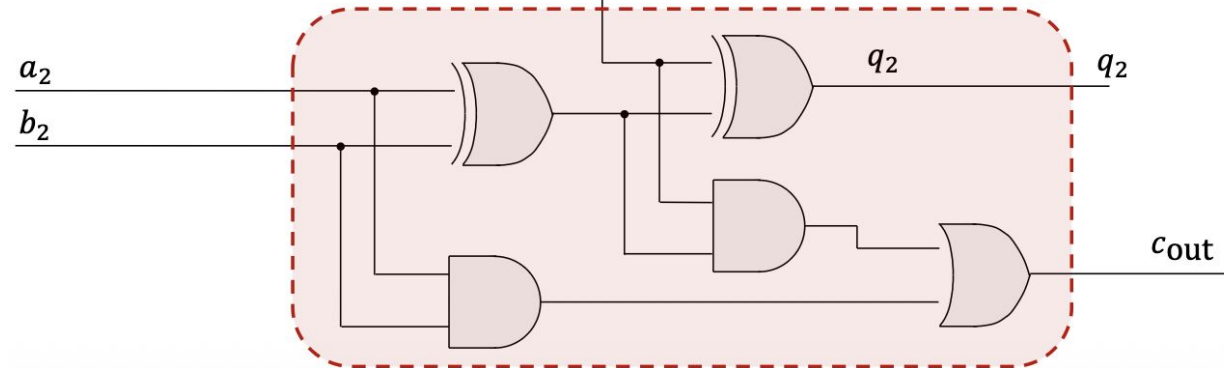


Half-Adder vs Full-Adder



Half-Adder:

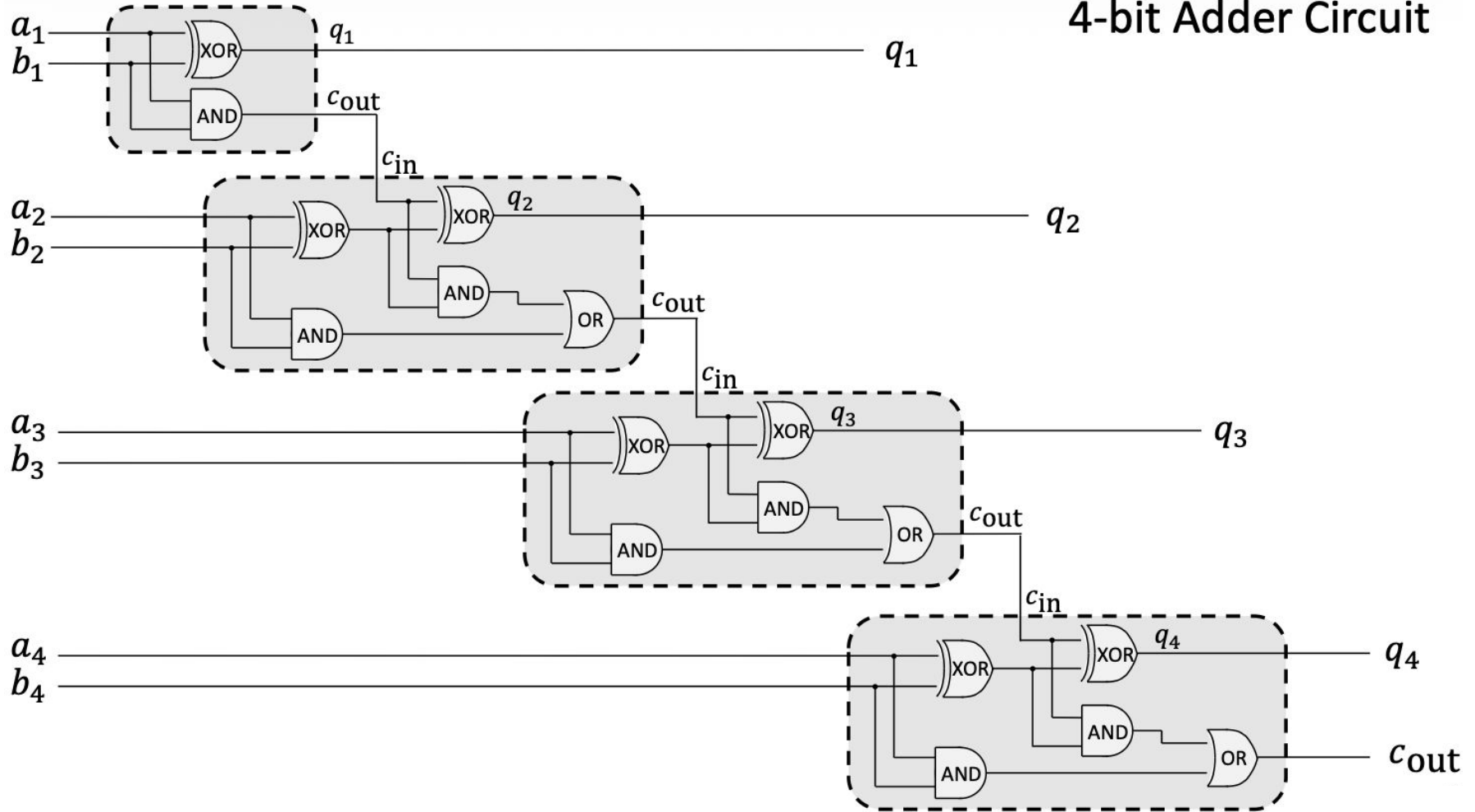
- No carry-in handling
- Least Significant Bits (LSB) summation only



Full-Adder:

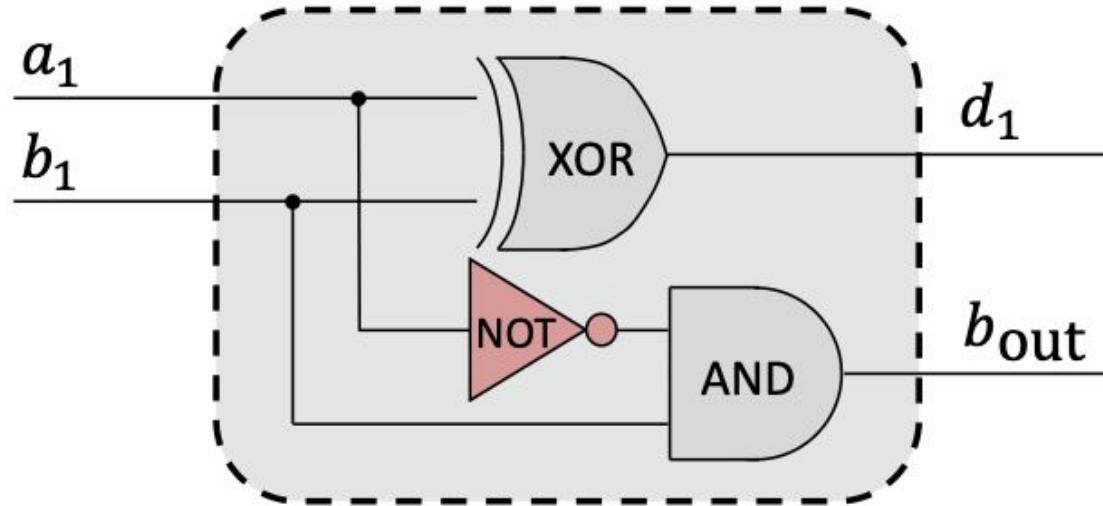
- Carry-in handling

4-bit Adder Circuit

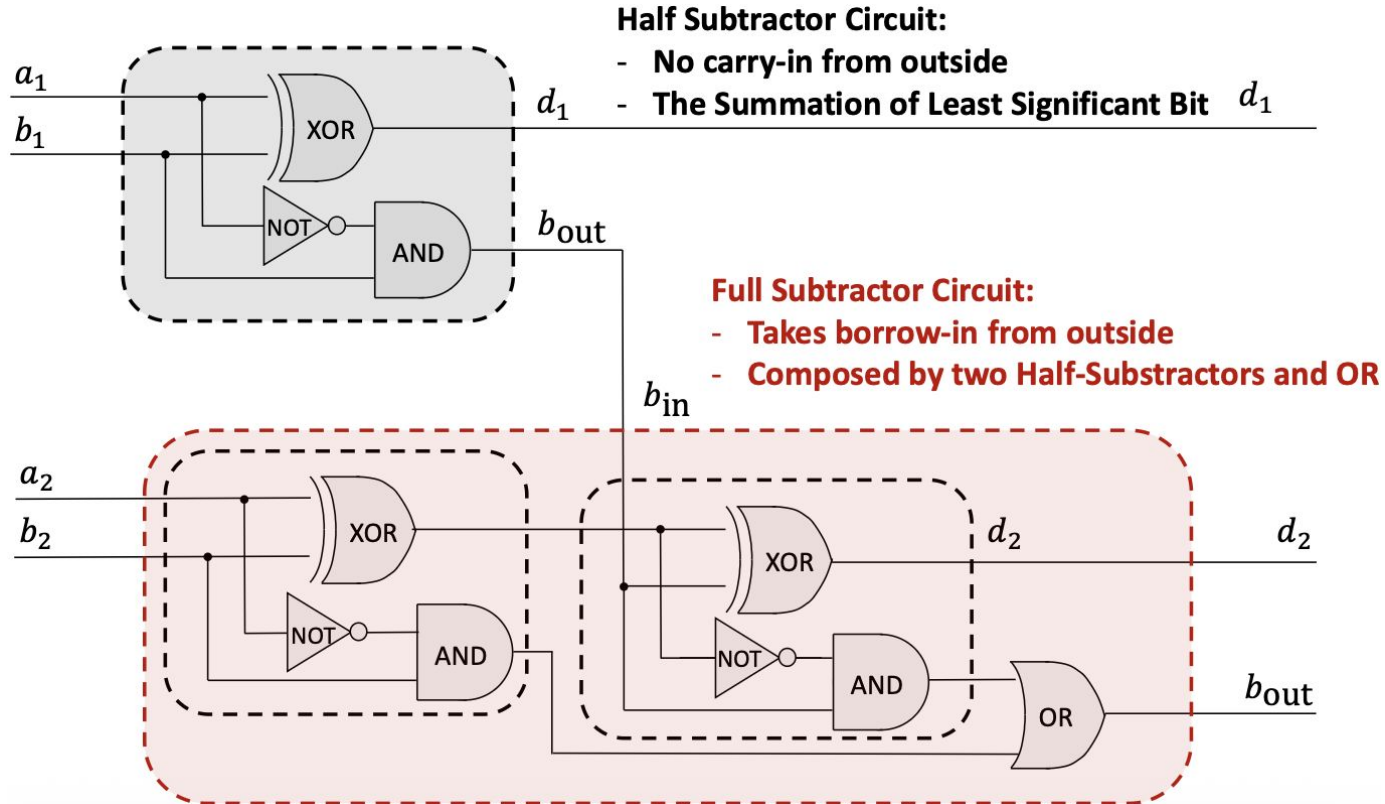


Half-Subtractor Circuit

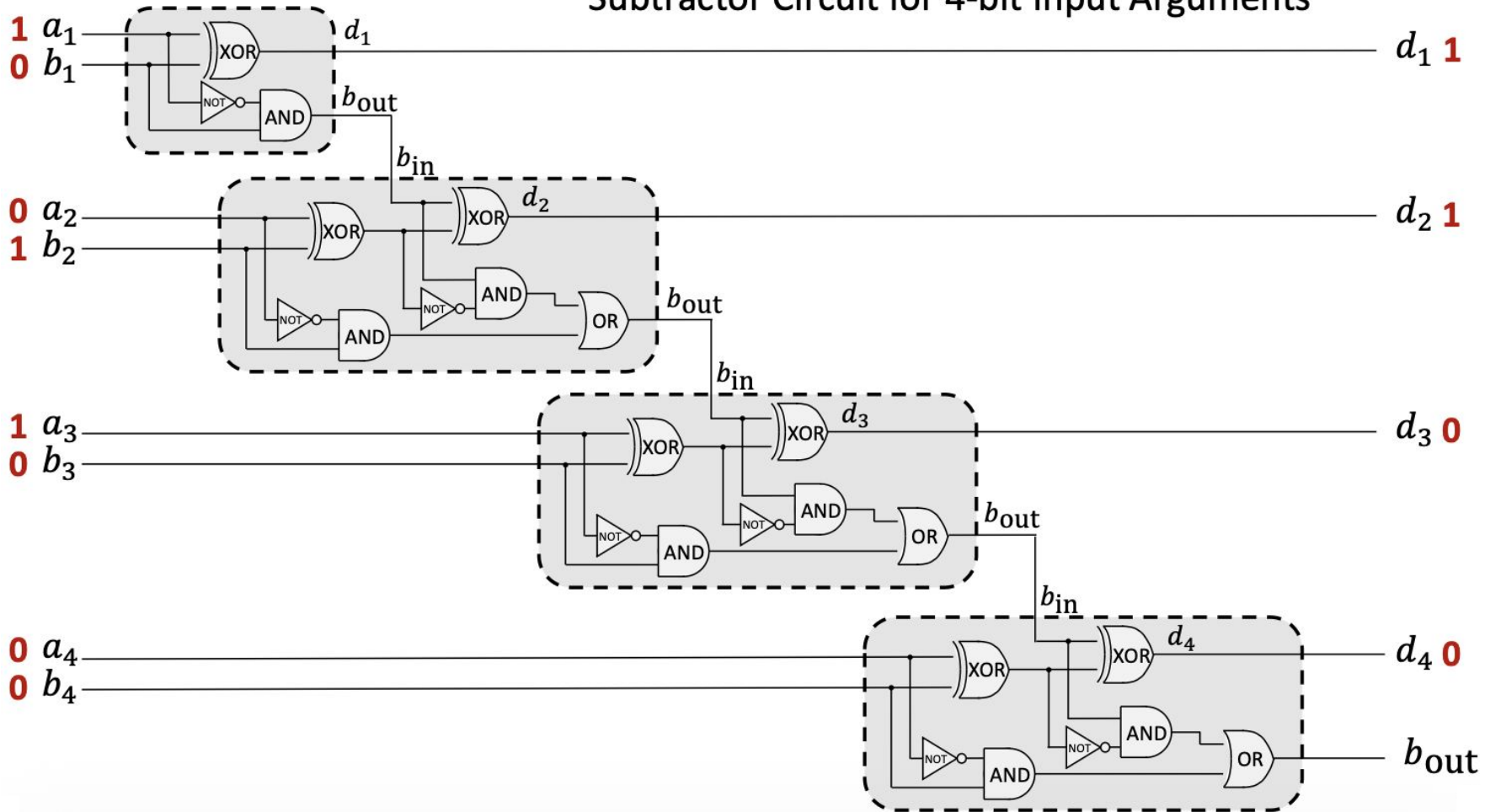
- No borrow-in handling
- Subtraction of Least Significant Bit



Half-Subtractor vs Full-Subtractor



Subtractor Circuit for 4-bit Input Arguments



Characteristics of Combinational Logic Circuits

- An output signal depends just on “current” input signals
- Comprised of logic gates, such as AND, OR, etc., and wires connecting them
- Output signal is updated after the change of input signals, subject to propagation delay
- Use no memory units, such as registers
- Typically, no clock signal is present (exceptions might apply)

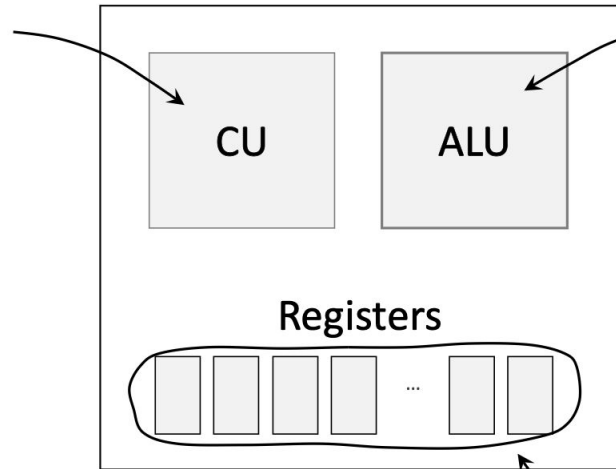
Pros: Fast

Cons: No support of inputs synchronization

Types of Logic Circuits for CPU Components

Not a pure combinational circuit

(for example, might have IR – instruction register)



Combinational logic circuit
for arithmetical and logic
operations

Fast - advantage

No synchronization support
- disadvantage

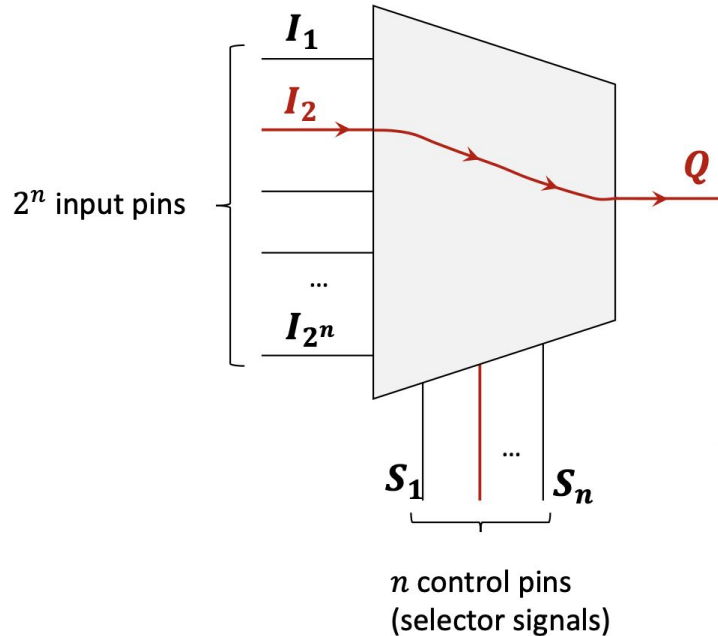
Sequential logic circuits (clocked)

- 1) Capture inputs (might arrive not simultaneously);
- 2) Store inputs until a certain time instant;
- 3) Output all inputs simultaneously to ALU

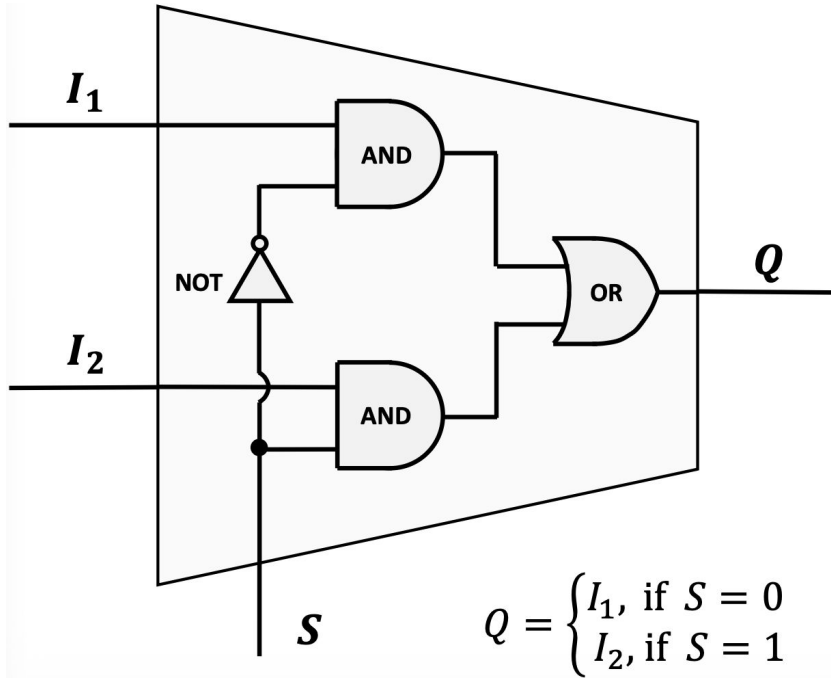
Hardware Building Blocks

Multiplexer

Based on combination of control pin signals, sets its output to one of its inputs



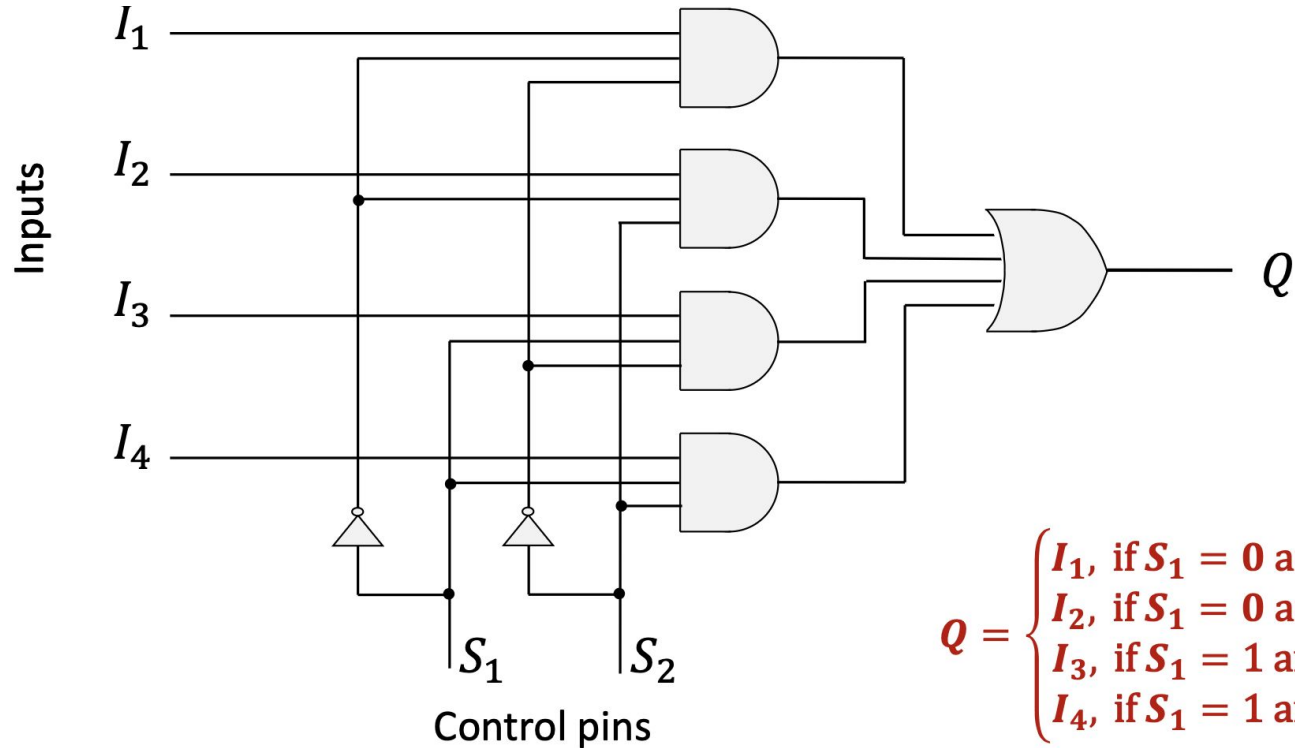
2-to-1 Multiplexer Implementation



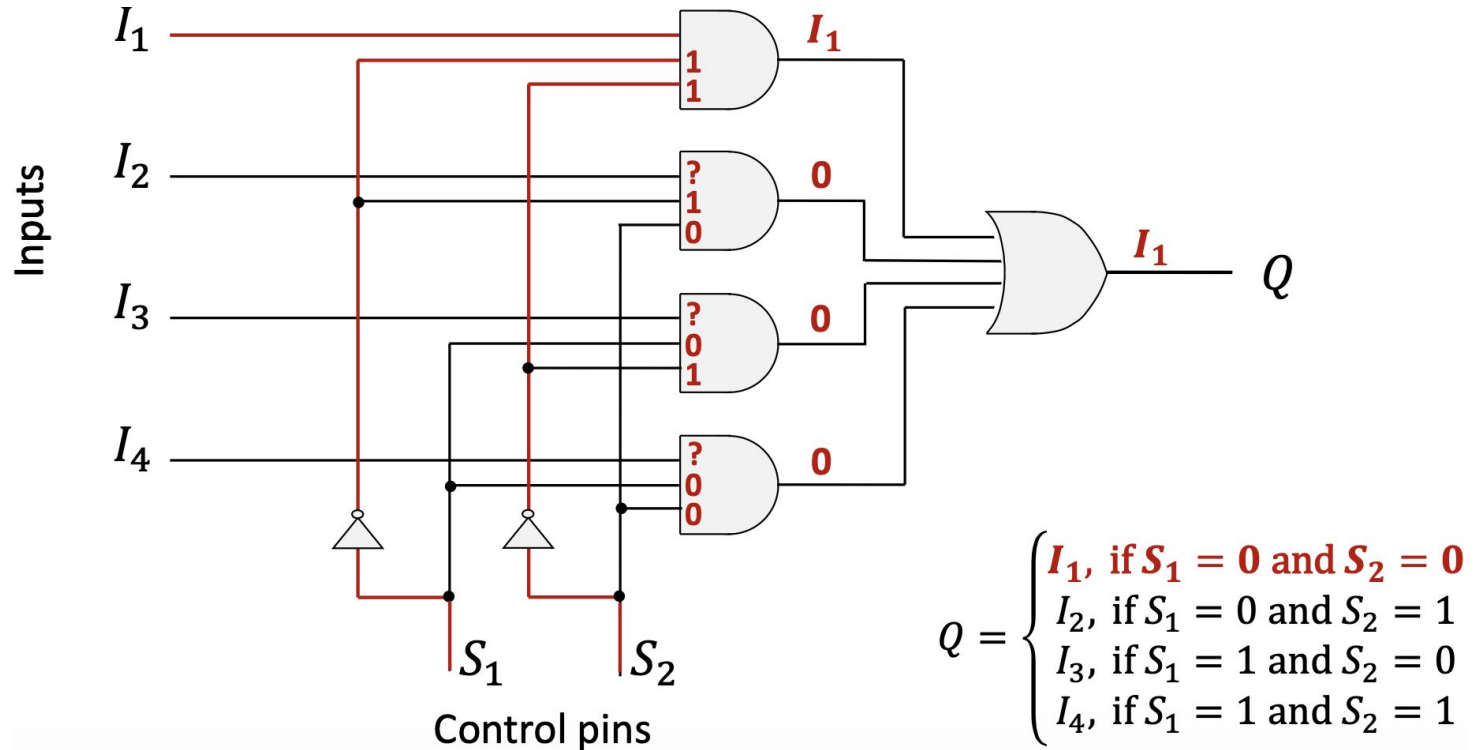
Verification of correctness:

I_1	I_2	S	Q
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

4-to-1 Multiplexer Implementation

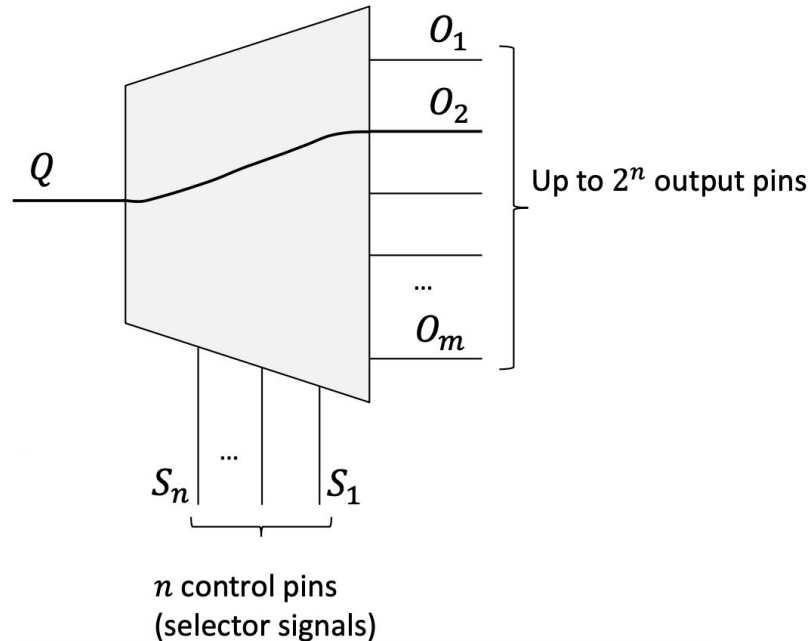


4-to-1 Multiplexer Implementation

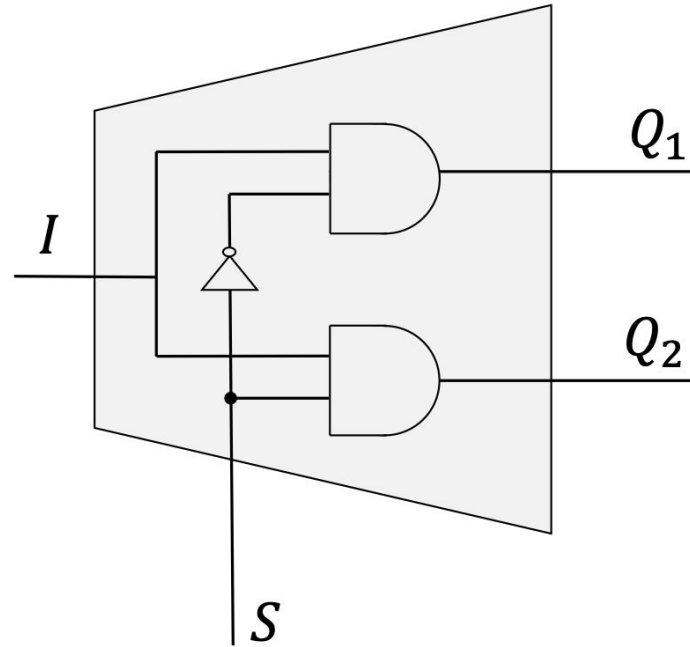


Demultiplexer ("Flipped" Multiplexer)

Based on combination of control pin signals, sets its input to one of its outputs



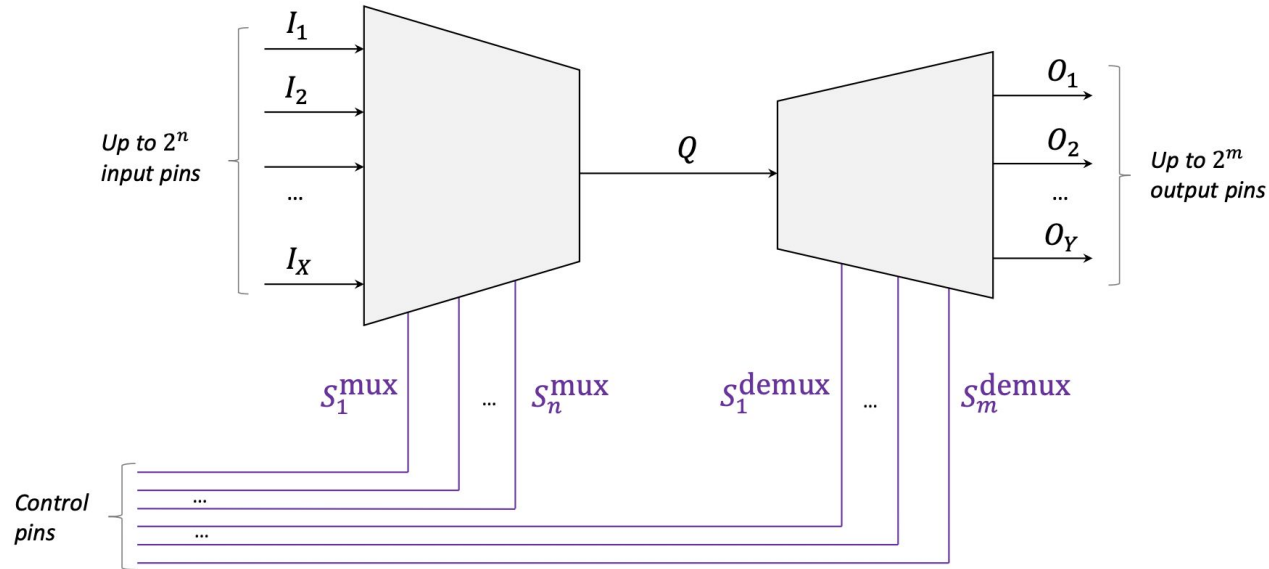
1-to-2 Demultiplexer Implementation



$$Q_1 = I \text{ and } Q_2 = 0, \text{ if } S = 1$$
$$Q_1 = 0 \text{ and } Q_2 = I, \text{ if } S = 0$$

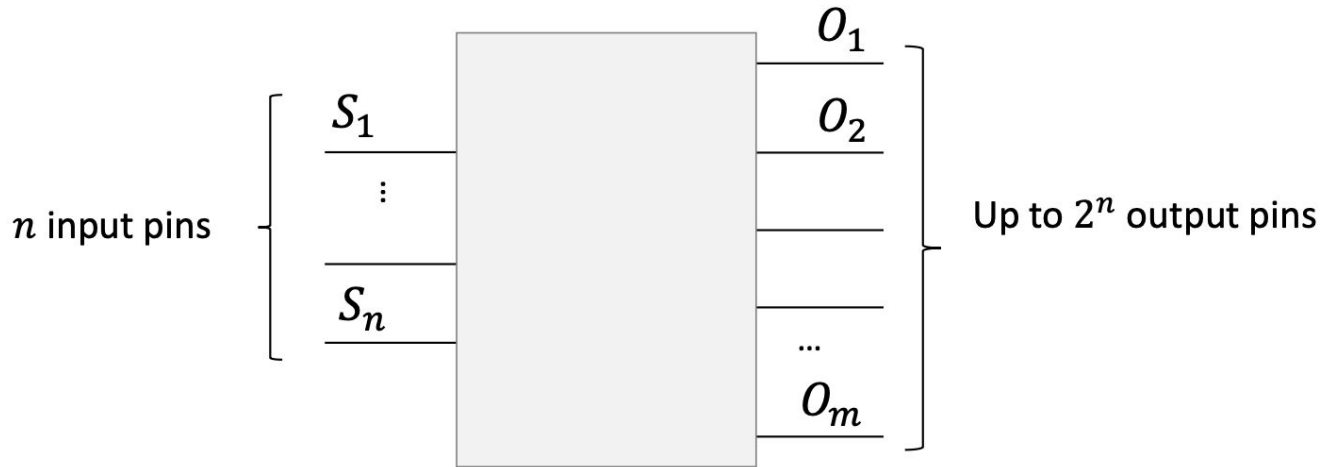
X-to-Y Multiplexer

Chooses one of its X input pins, and outputs its value at one of its Y output pins, while all other output pins are reset to 0s

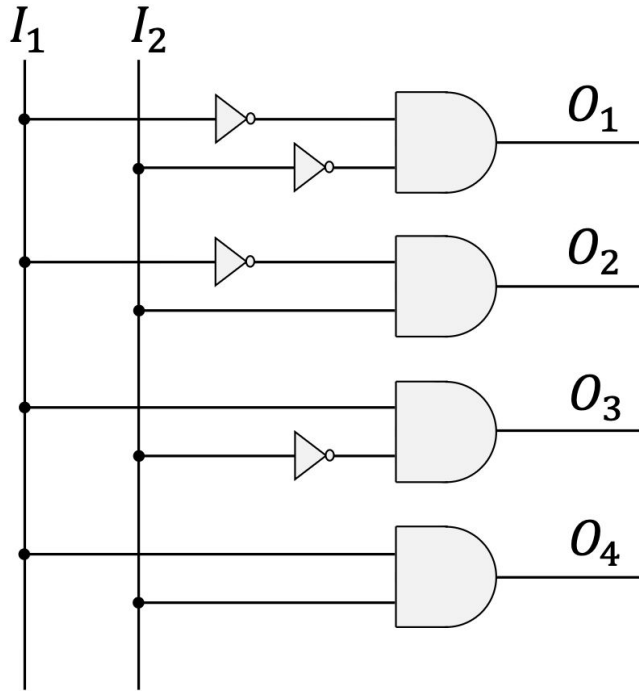


Decoder

- All output pins are set to "0", except
- The only one pin is set to "1", which corresponds to a given combination of inputs



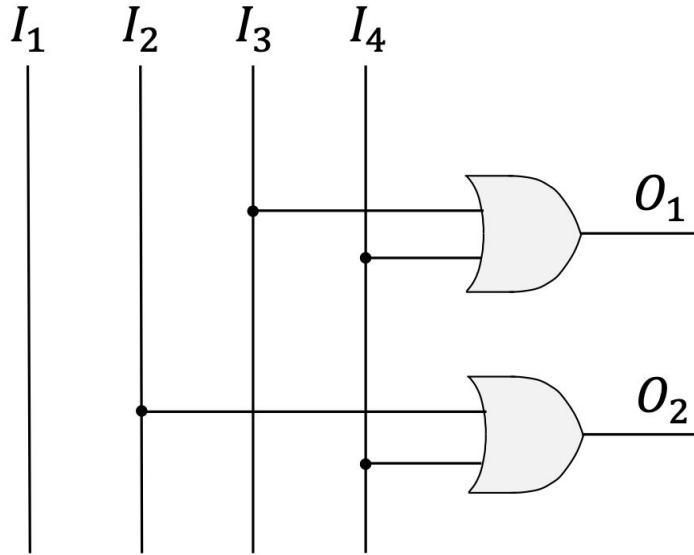
2-to-4 Decoder Implementation



I_1	I_2	O_1	O_2	O_3	O_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Note: These encoder and decoder circuits have symmetric truth tables.

4-to-2 Encoder Implementation

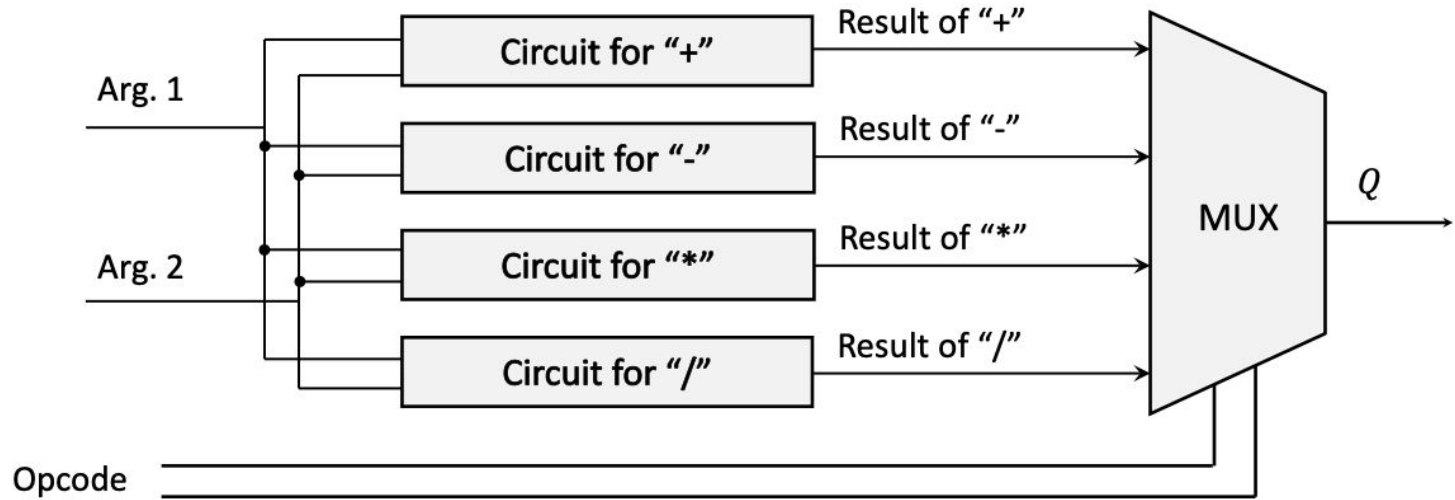


I_1	I_2	I_3	I_4	O_1	O_2
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

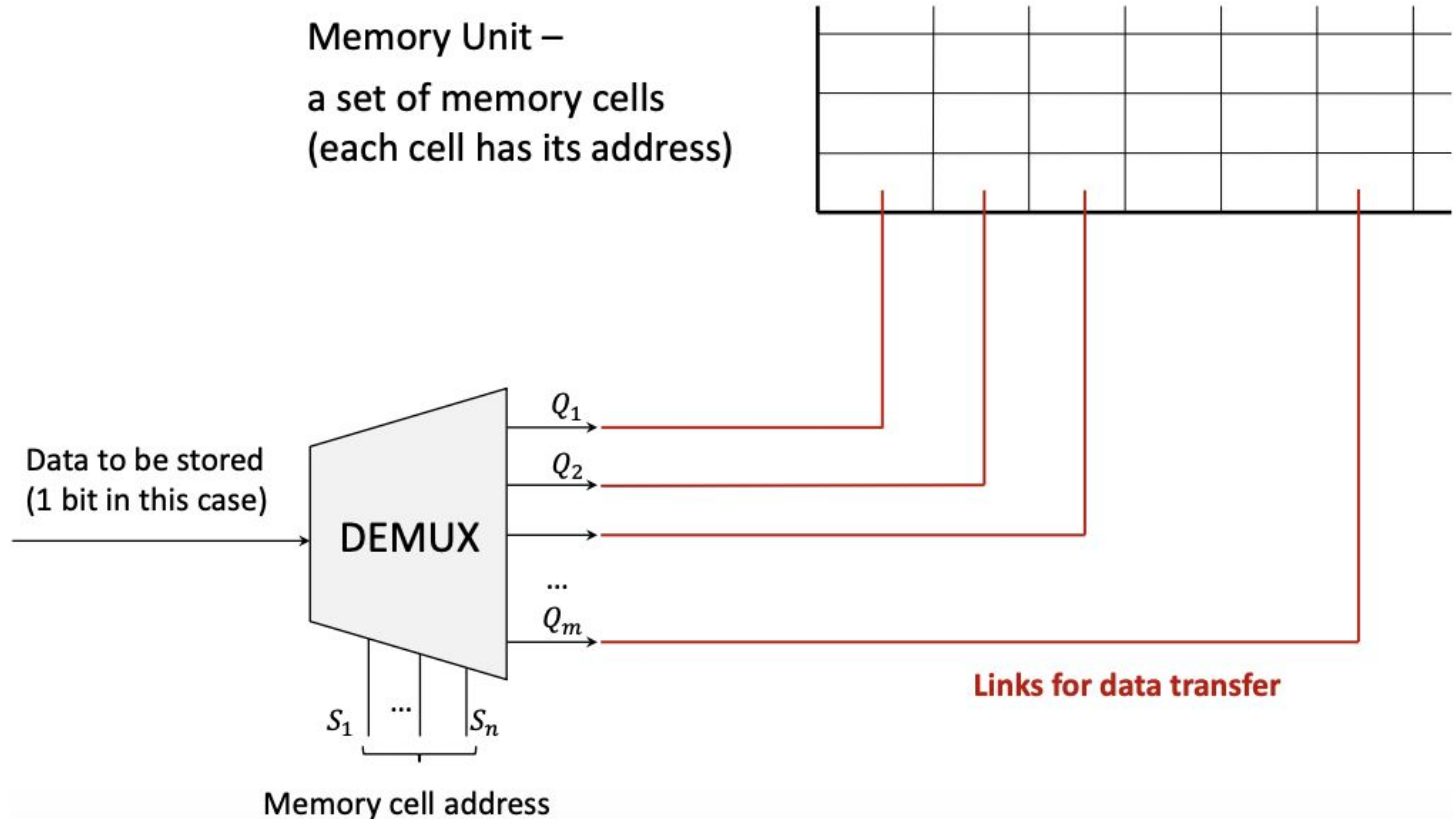
Note: These encoder and decoder circuits have symmetric truth tables.

Multiplexer for ALU Implementation

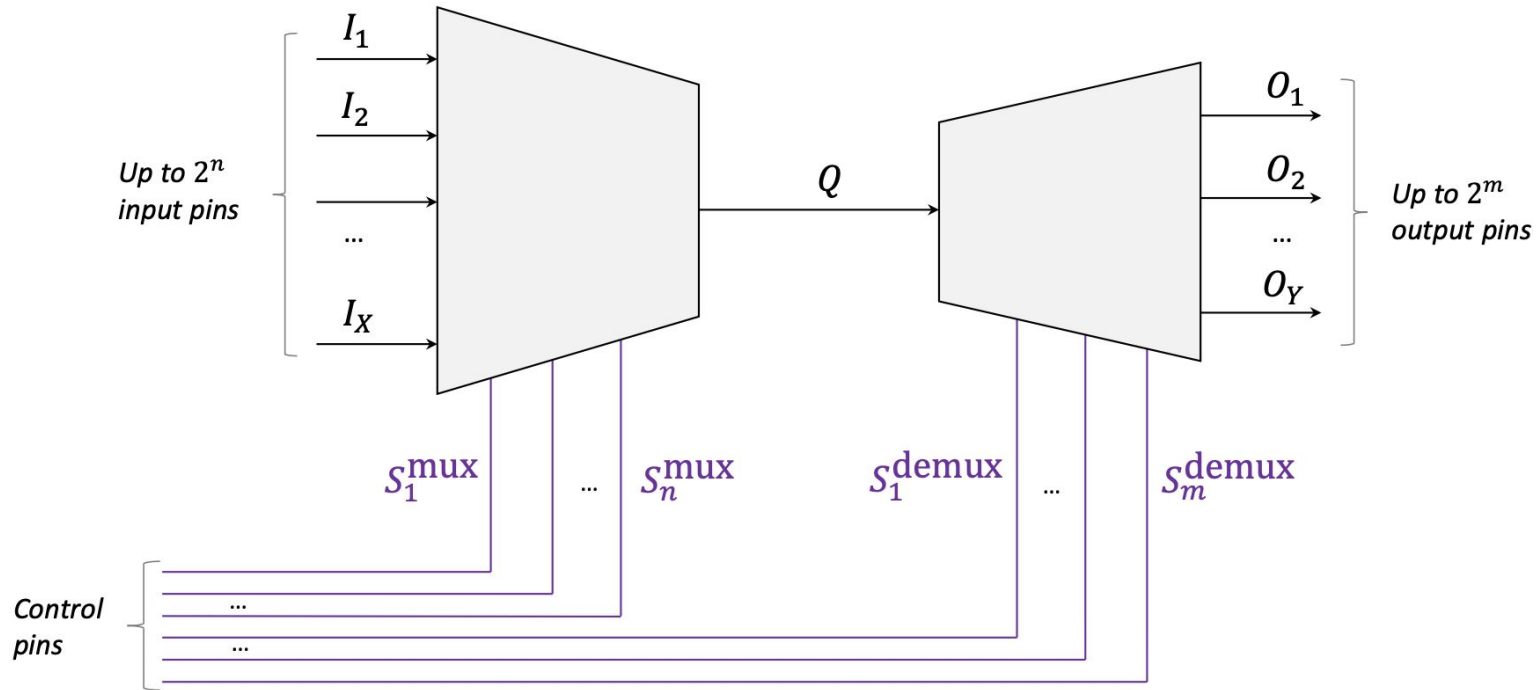
Let our ALU support 4 instructions: "+", "-", "*", "/"



Demultiplexer for Memory Unit Write



X-to-Y Muxplexer for Multiple Bit Storage



Binary Multiplier, Latches, FSM for Latches

Binary Multipliers

Single bits case:

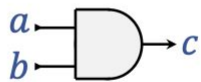
$$a \times b = c$$

$$0 \times 0 = 0$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 1 = 1$$



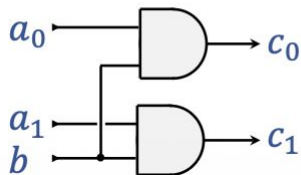
2-by-1 bits case:

$$a_1 a_0 \times b = c_1 c_0$$

$$a_1 a_0 \times 1 = a_1 a_0$$

$$a_1 a_0 \times 0 = 00$$

Every bit of $a_2 a_1$ is multiplied by b :



2-by-2 bits case:

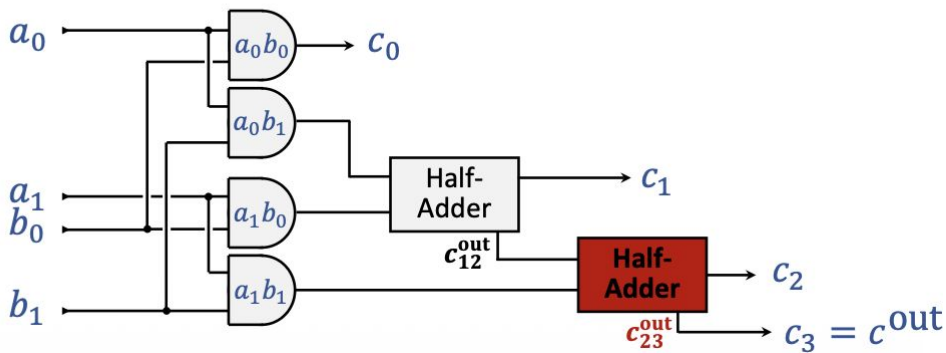
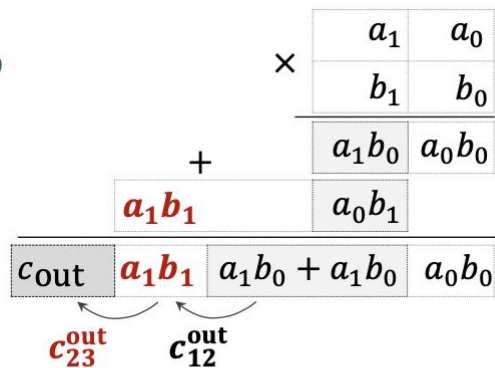
$$a_1 a_0 \times b_1 b_0 = c_3 c_2 c_1 c_0$$

$$11 \times 11 = 1001$$

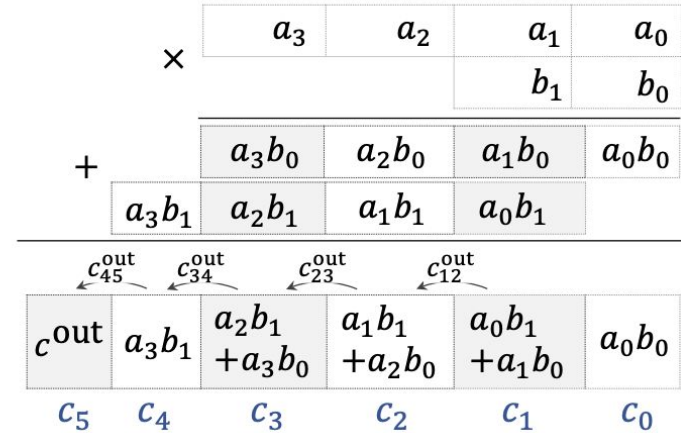
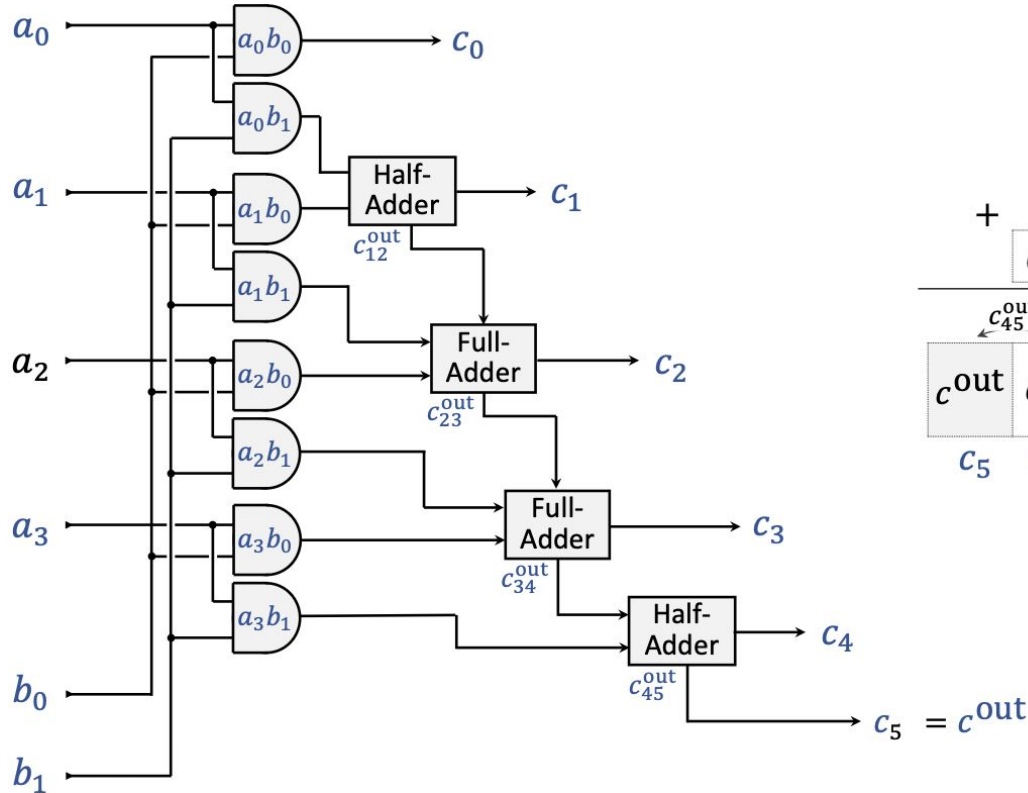
$$10 \times 01 = 010$$

$$00 \times 11 = 00$$

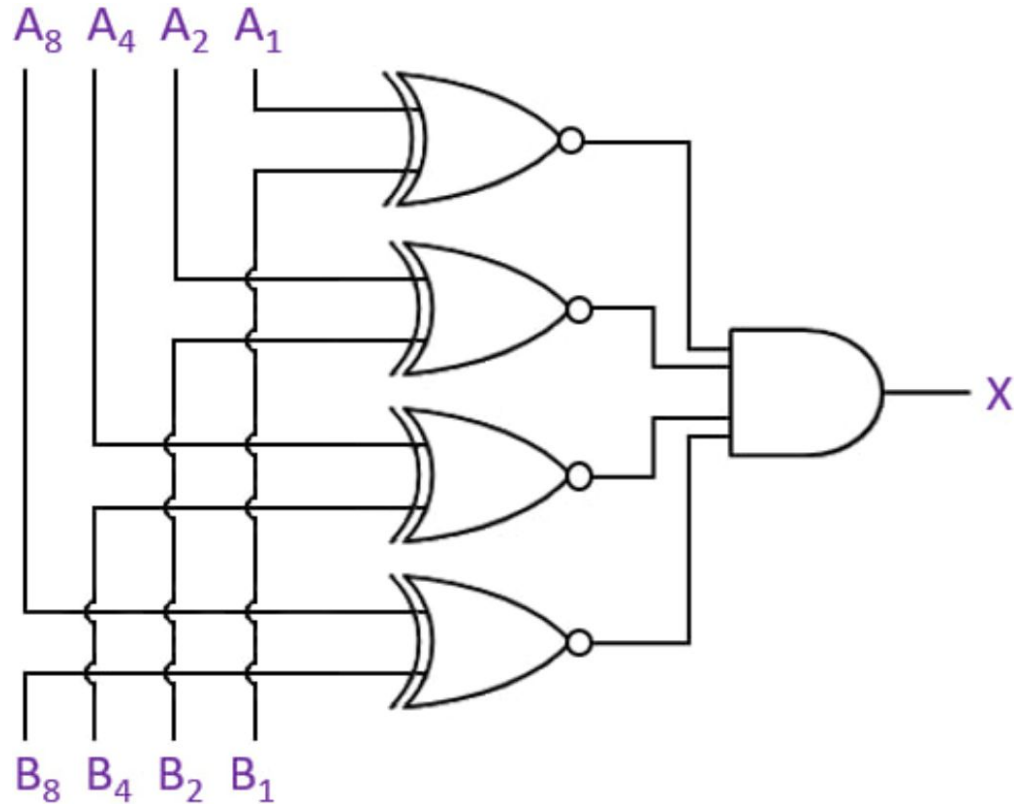
...



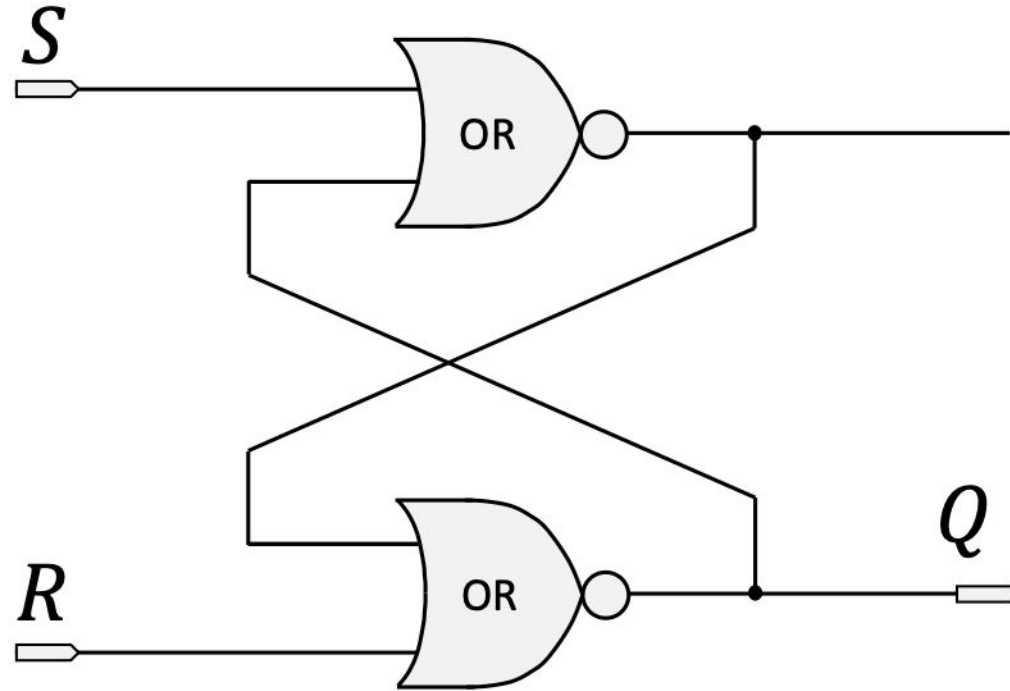
4-by-2 Bits Multiplication Construction



4-by-4 Bits Equality Checker

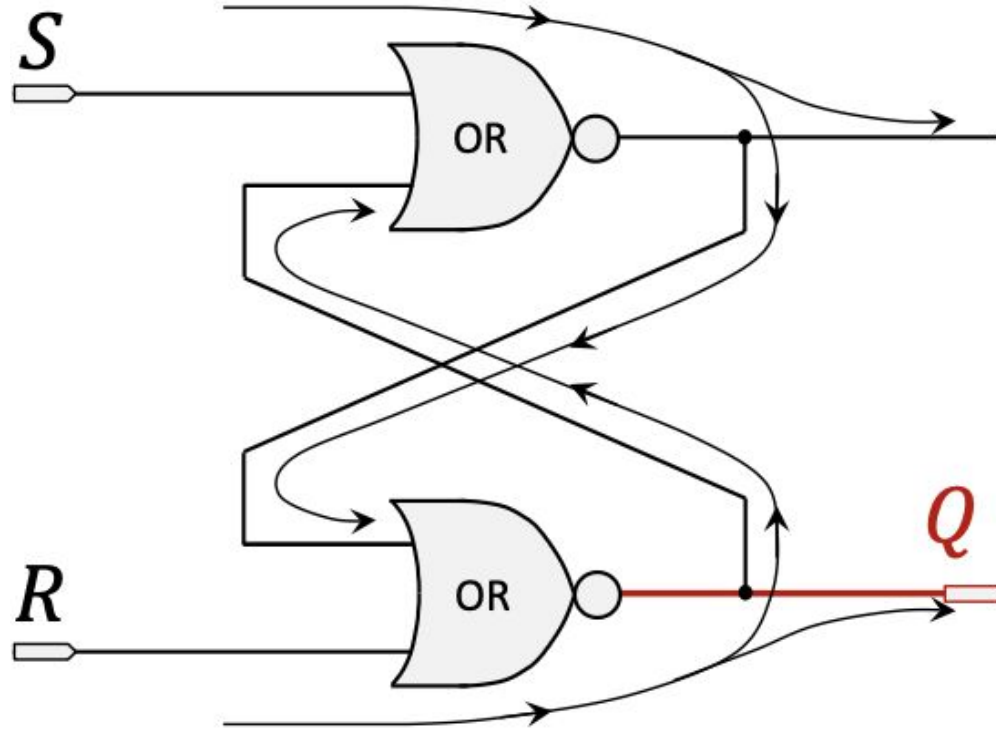


S/R Latch



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

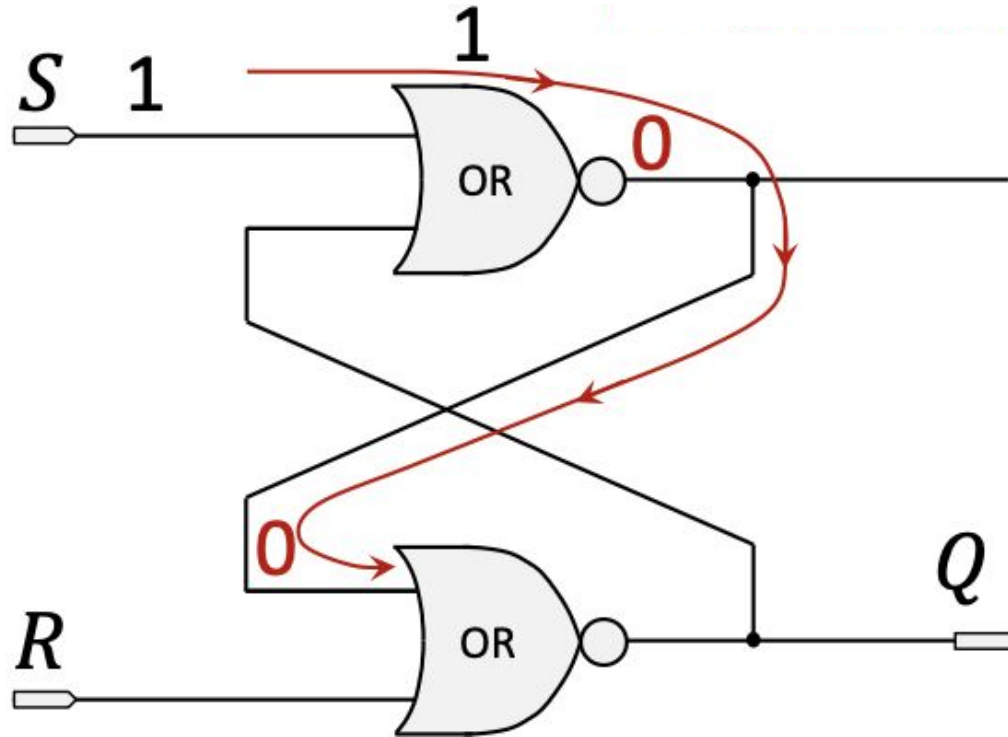
S/R Latch



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

S/R Latch

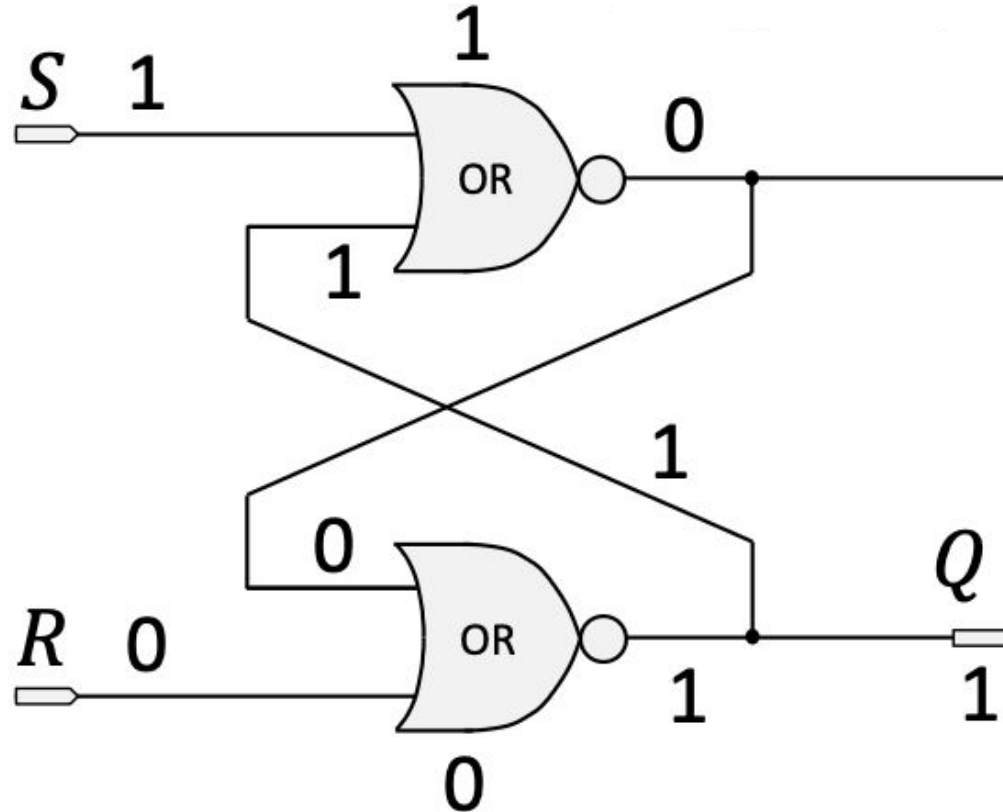
Set $S = 1$



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

S/R Latch

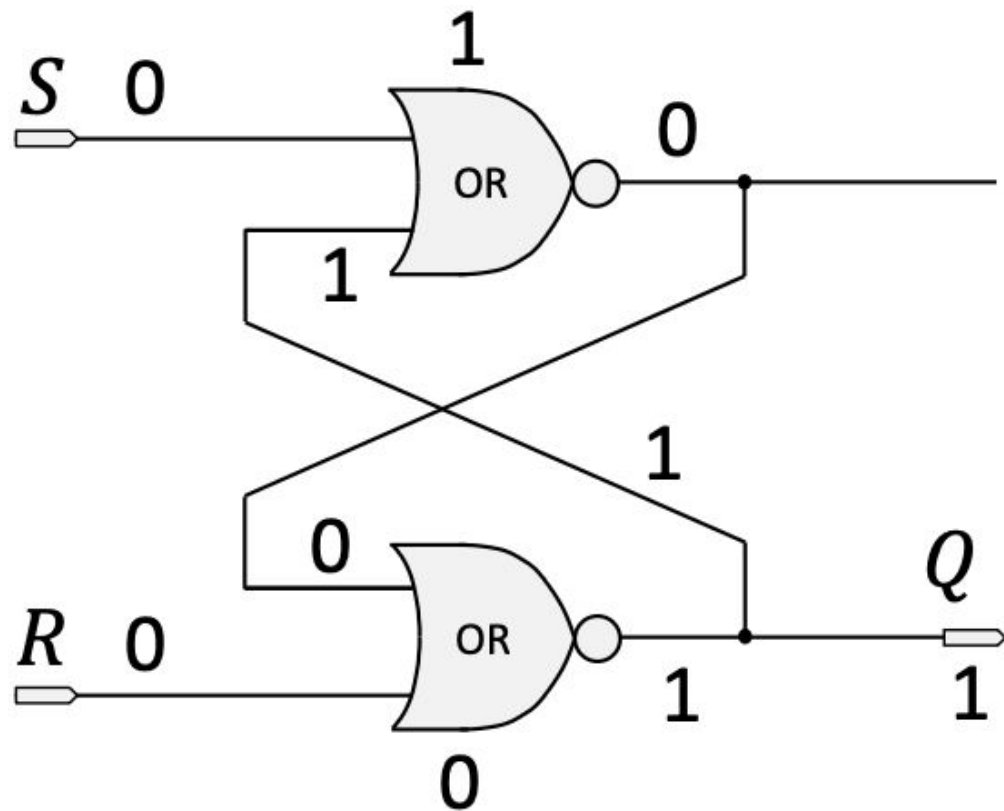
Set R = 0



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

S/R Latch

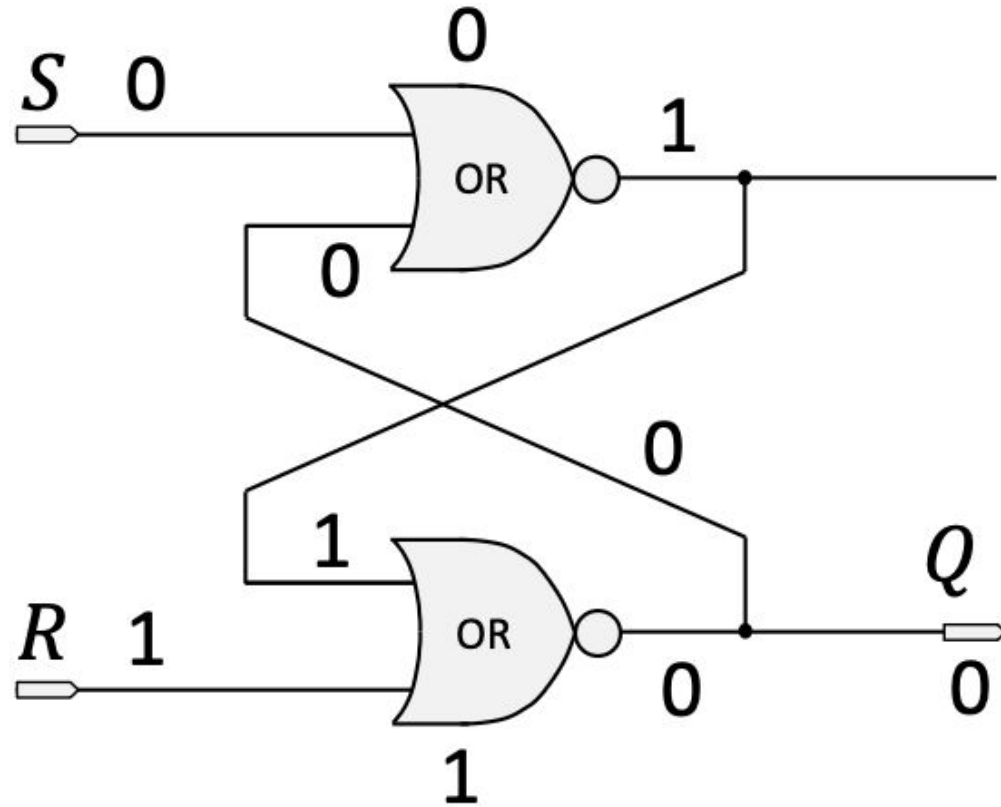
Set S = 0



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

S/R Latch

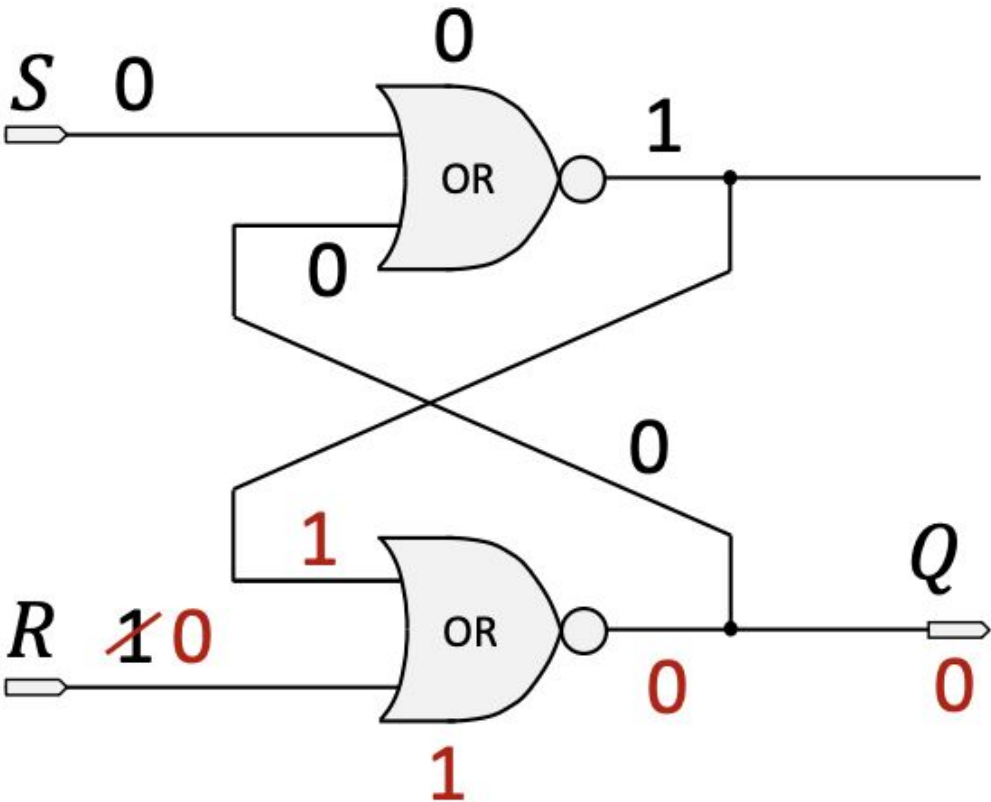
Set R = 1



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

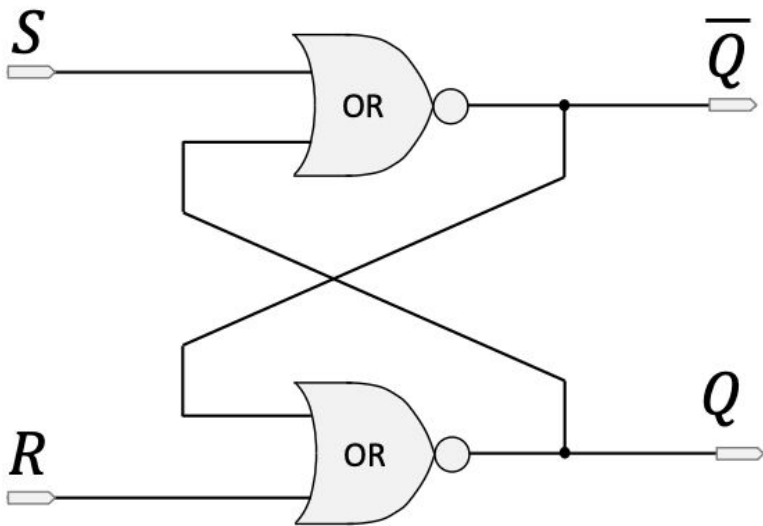
S/R Latch

Set R = 0



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

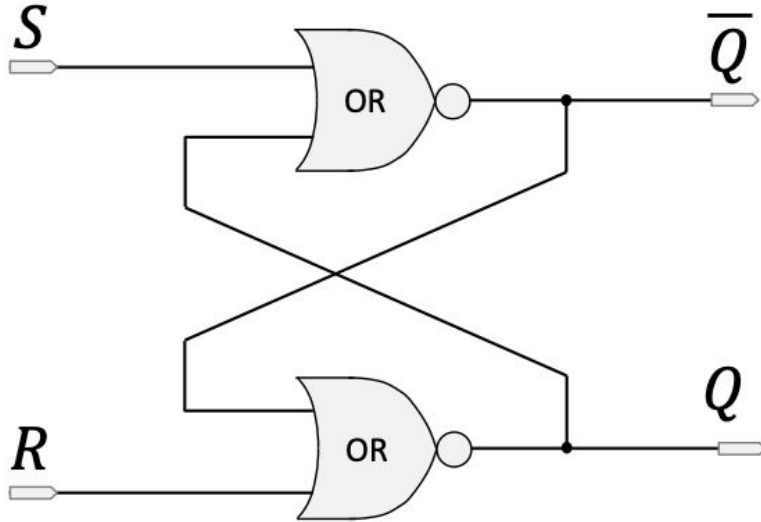
Circuit Behavior Analysis



A	B	OR	NOR
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

S	R	Q
1	0	1
0	1	0
0	0	Q^{prev}
1	1	Illegal inputs

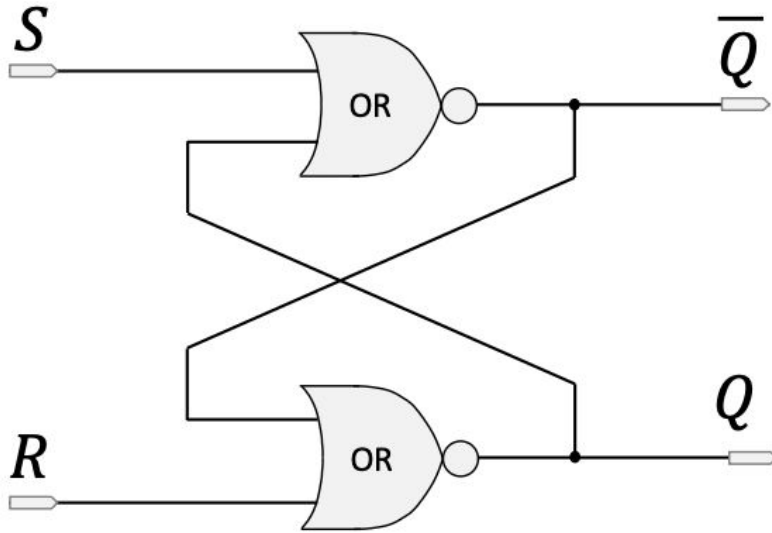
Bistable Multivibrator



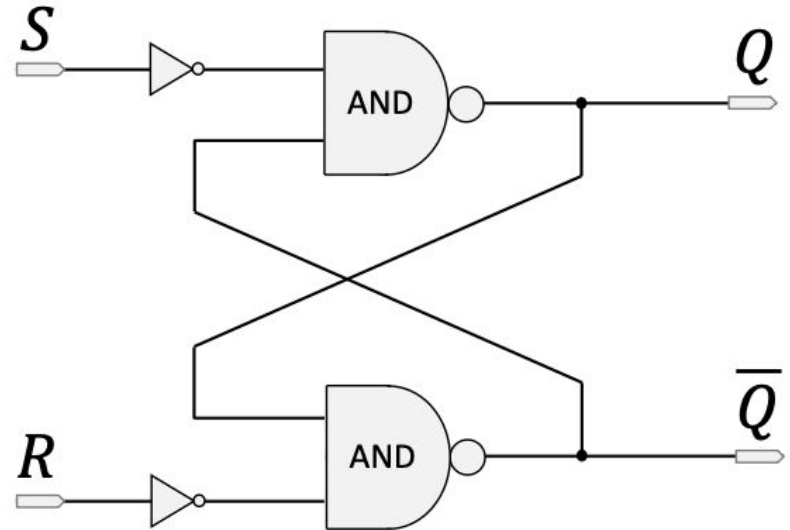
- Stable in either state
- Flipped from one state to the other by an external trigger pulse
- Used to store one bit of data;
- Used in volatile computer memory

S/R Latch through Universal Logic Gate

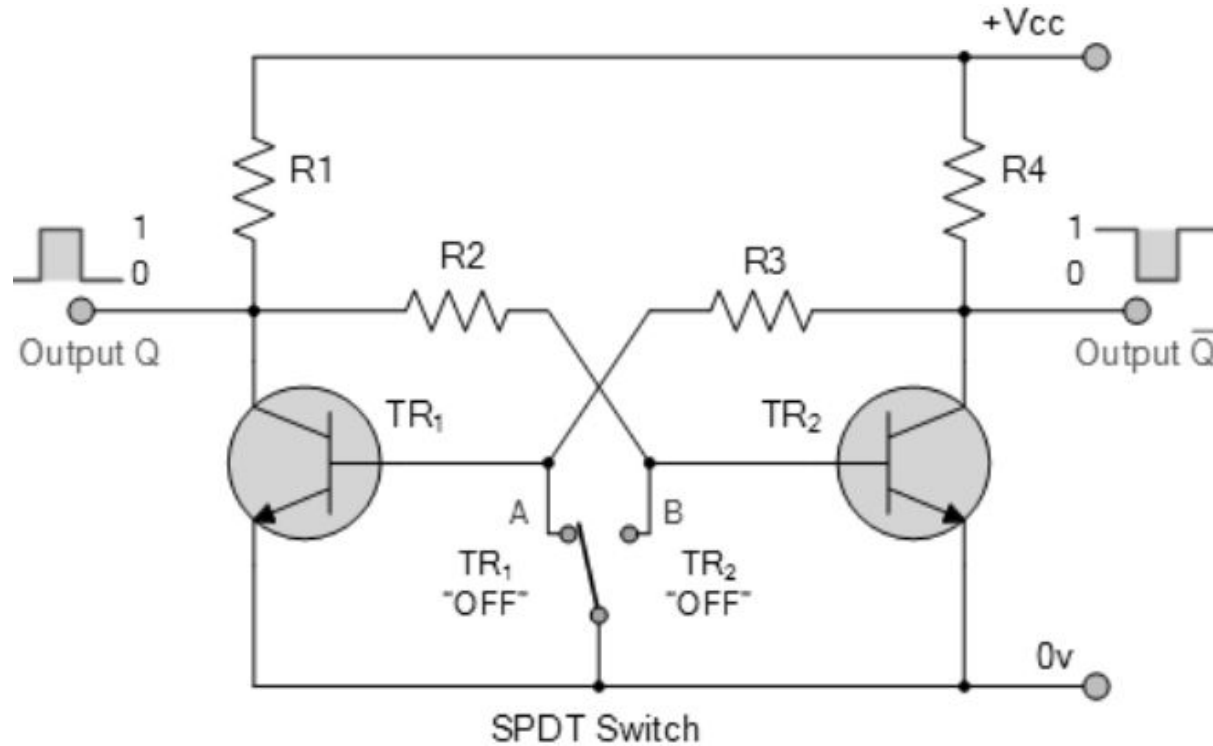
1) Implementation by using NOR logic gates:



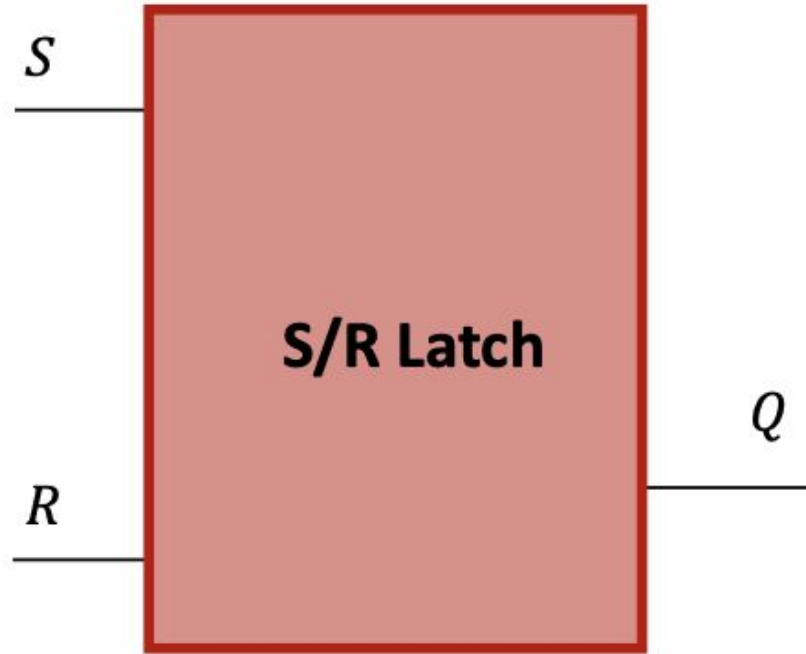
2) by using NAND logic gates:



S/R Latch Hardware Implementation



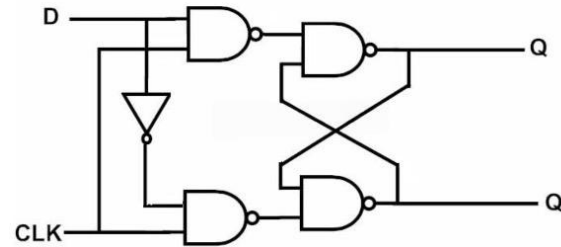
S/R Latch Symbolic Representation



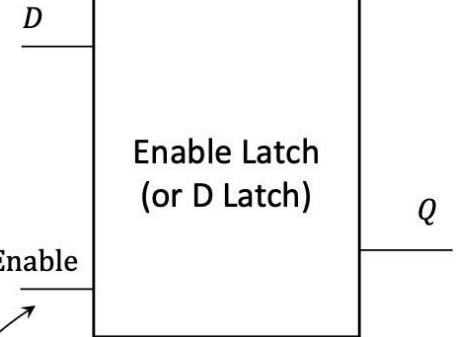
Enable Latch (D Latch)

- All input combinations are allowed
- Simpler usage

**Enable Latch implementation;
clock serves as "Enable" pin**



1 bit of "data" (0 or 1) to be stored

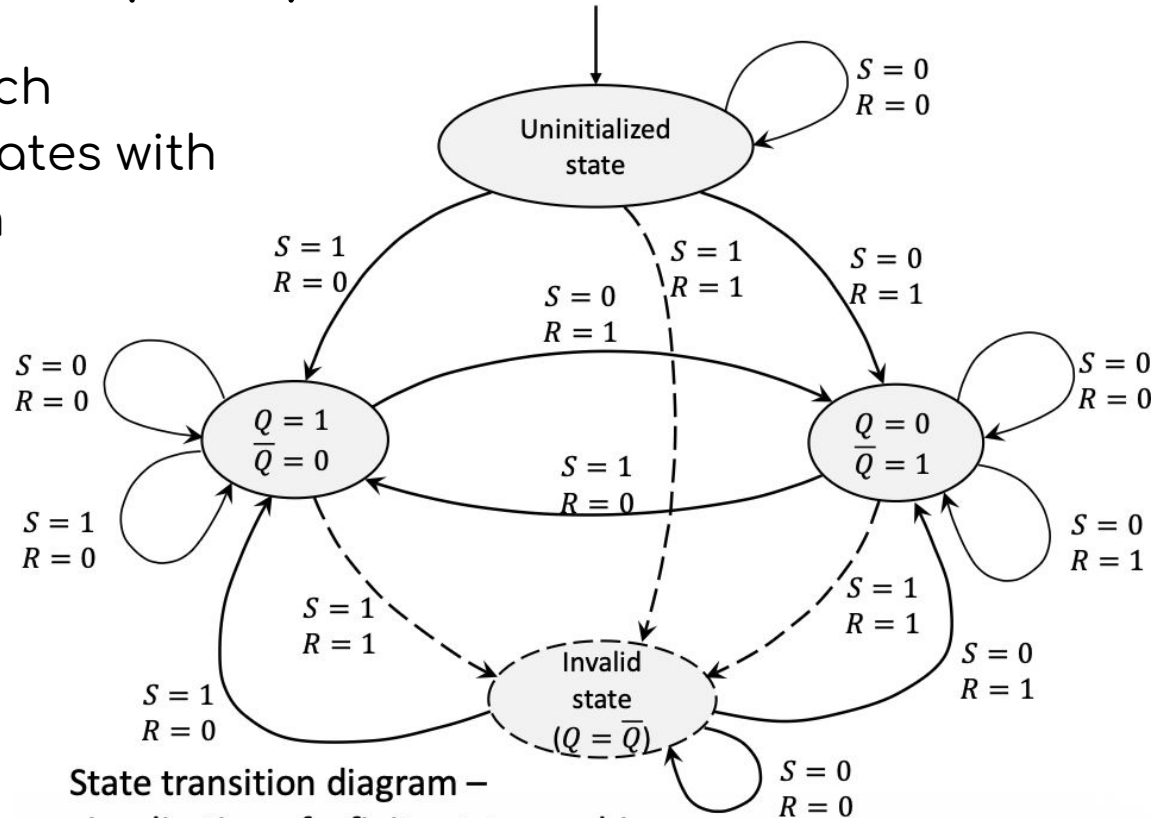


Enables data storage:

- if "1": D is stored;
- If "0": D is ignored

Finite State Machine (FSM)

An abstract machine which represents all possible states with transitions between them



State transition diagram –
visualization of a finite state machine

Flip-Flop, Synchronous and Asynchronous Circuits

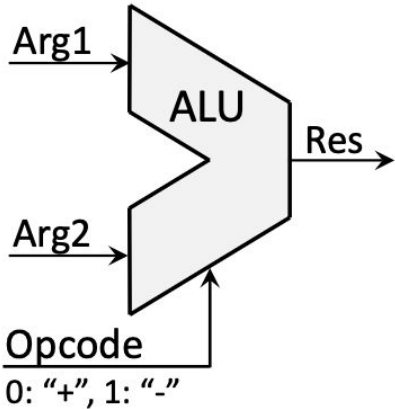
Inputs Synchronization: Motivation

Expected execution:

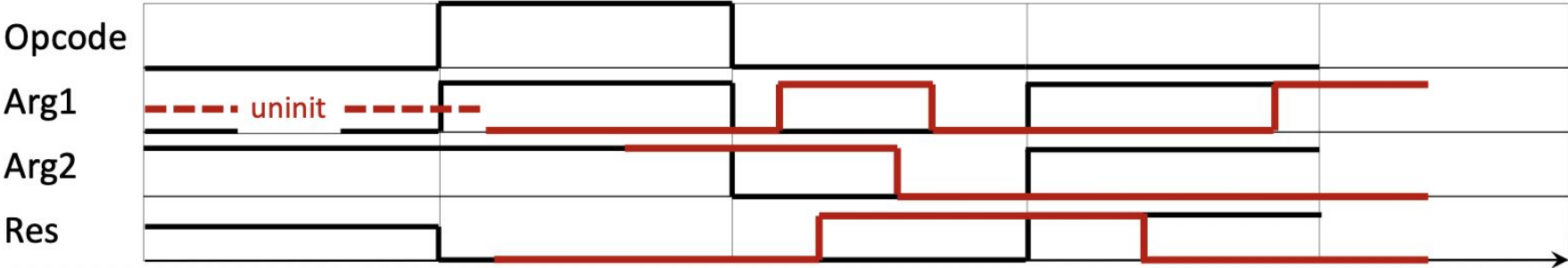
Opcode	Arg1	Arg2	Res (Expected)
0	0	1	1
1	1	1	0
0	0	0	0
0	1	1	10

Reality (inputs arrive async.):

Opcode	Arg1	Arg2	Res (Reality)
0	uninit	1	uninit
1	0	1	01
0	1	1	10
0	0	1	1



Inputs synchronization is needed!

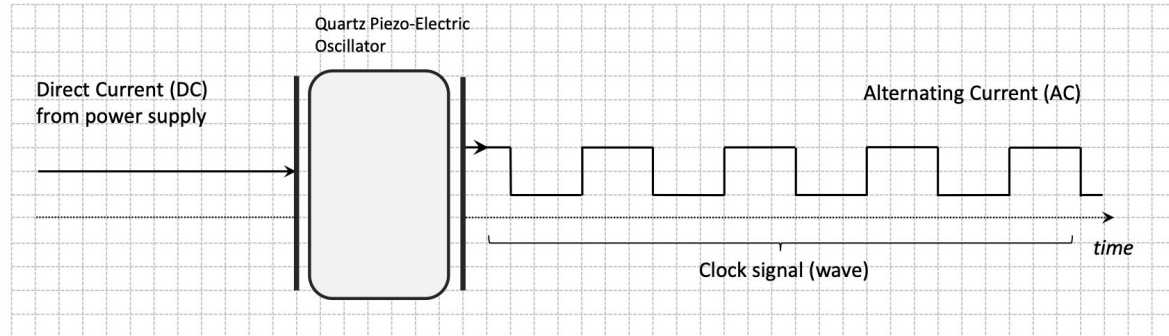


Clock Generator (Oscillator)

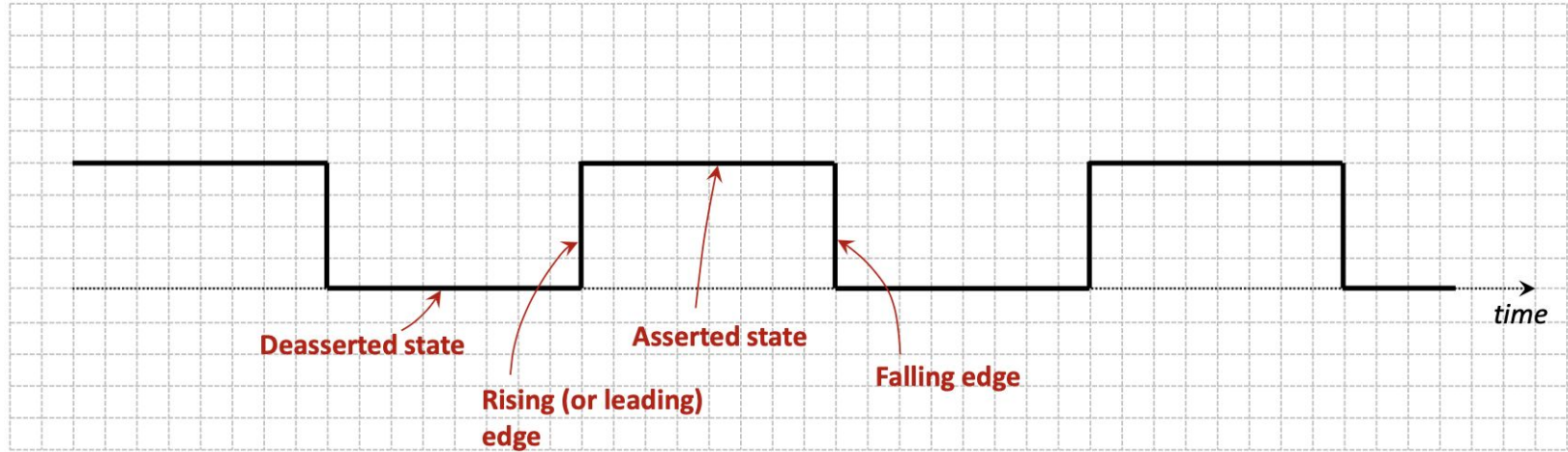
An electronic device, producing a periodic clock signal

Clock oscillator – a physical device, that converts Direct Current (DC) from a power supply into an alternating periodic signal

Oscillation – a repetitive variation, typically in time, of some measure between two or several values



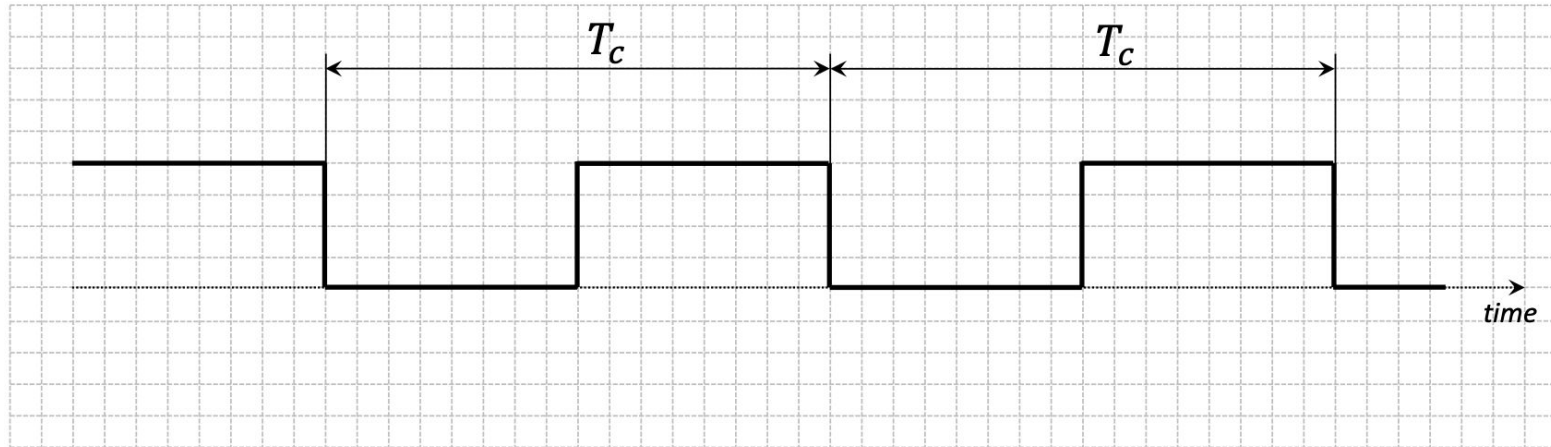
Clock Signal Terminology



Characteristics of Rectangular Clock Signal

T_c - clock cycle time, or period, or cycle length

$F_c = 1/T_c$ - clock frequency



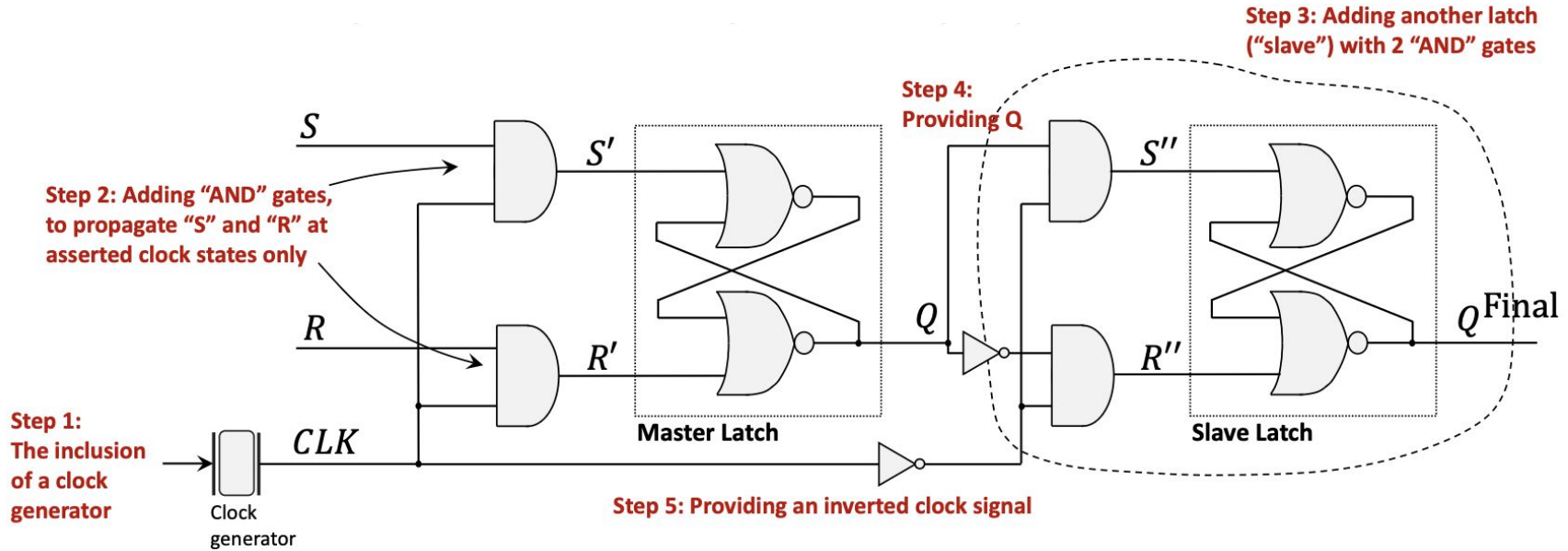
Clock frequency sets the tempo for the processor to execute instructions

Common characteristics for modern general-purpose CPUs:

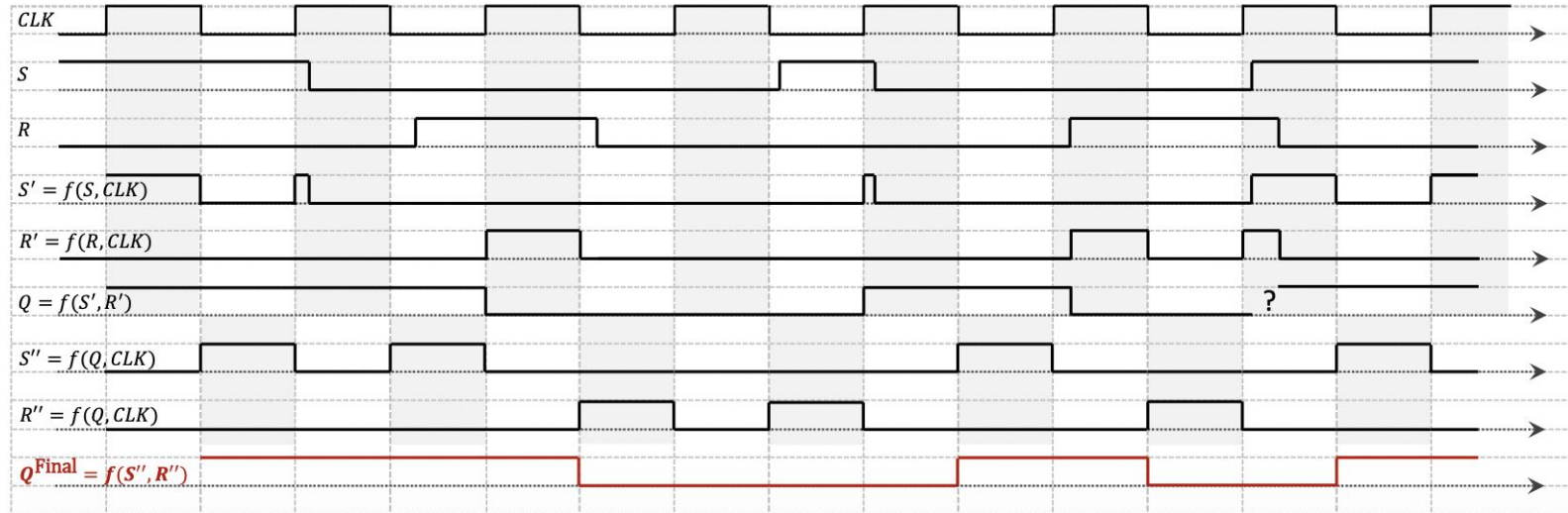
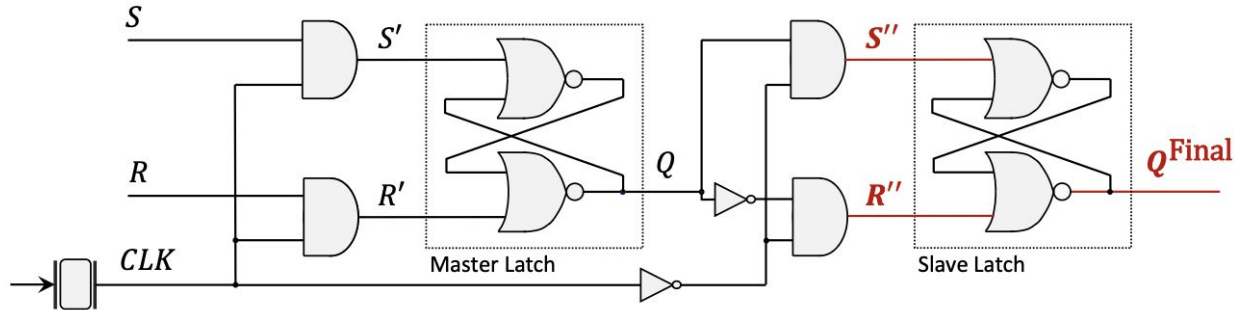
T_c - a few nanoseconds, $\sim 10^{-9}$ sec

F_c - a few gigahertz, 2-3 GHz

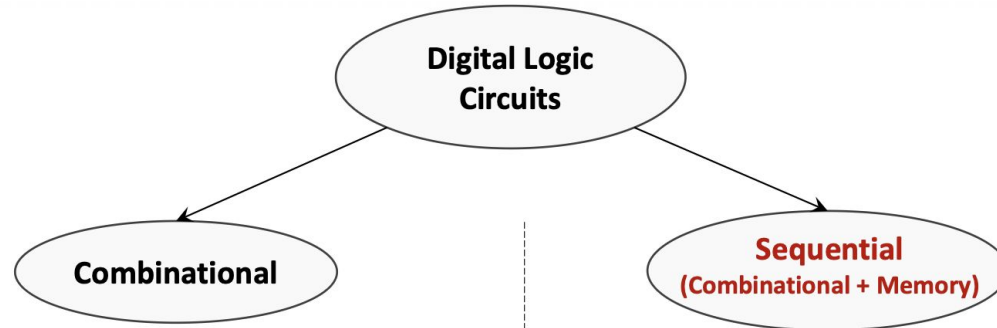
Transformation of an Asynchronous Latch into a Synchronous Flip-Flop



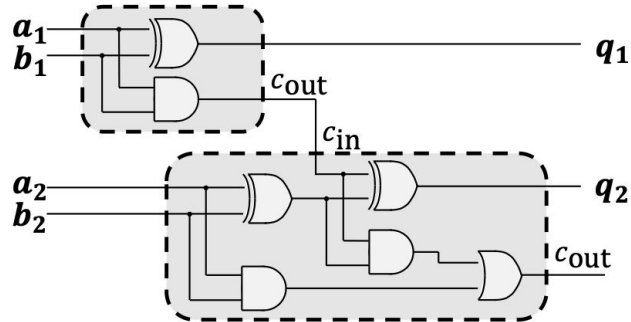
Flip-Flop flow



Digital Circuits Classification

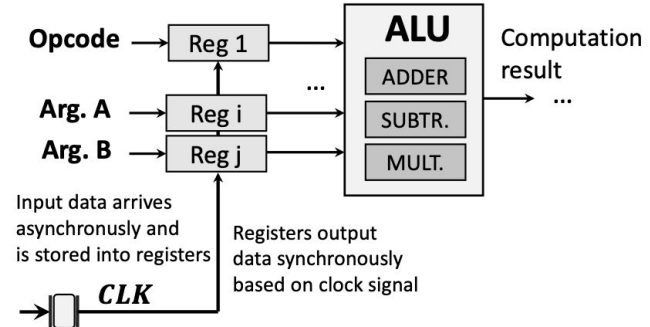


Example: Adder Circuit



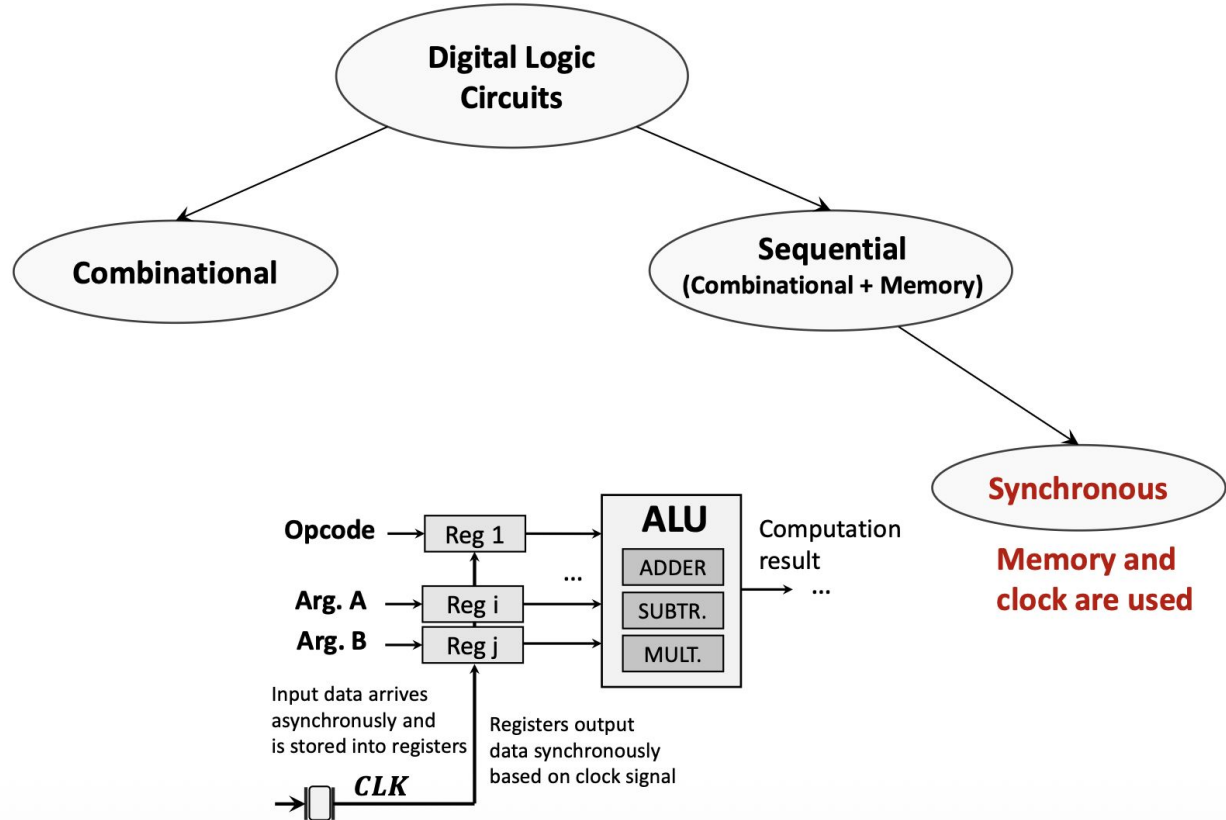
Uses **NO** memory or clock elements

Example: ALU + CPU Registers

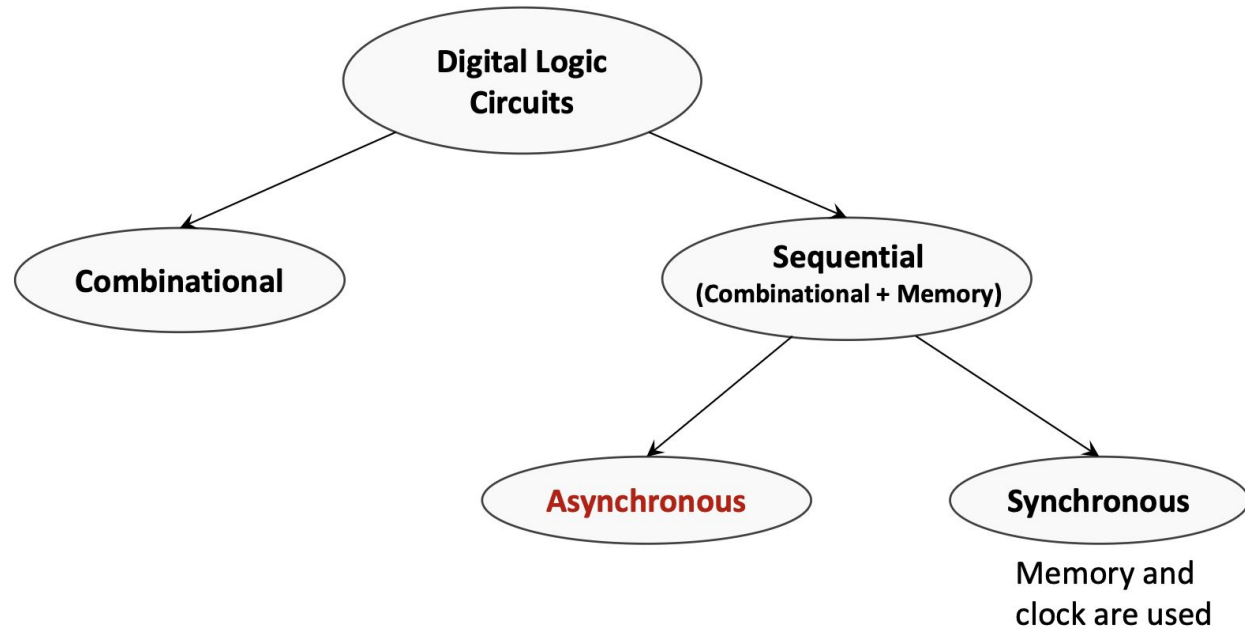


Uses memory or clock elements

Digital Circuits Classification



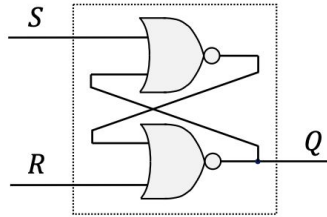
Digital Circuits Classification



Asynchronous sequential circuits, also known as self-timed or ripple-clock circuits, are digital circuits that do not use a clock signal to determine the timing of their operations. Instead, the state of the circuit changes in response to changes in the inputs.

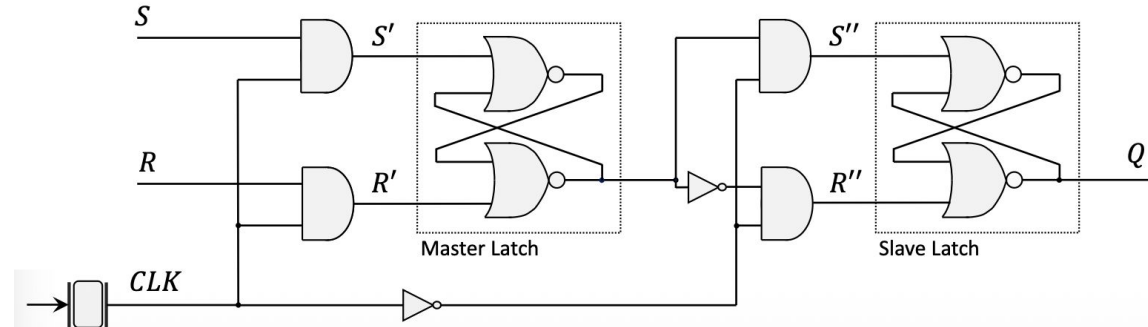
Asynchronous and Synchronous Circuits

S/R latch – asynchronous circuit: updates of output signal Q are not synchronised with a clock signal

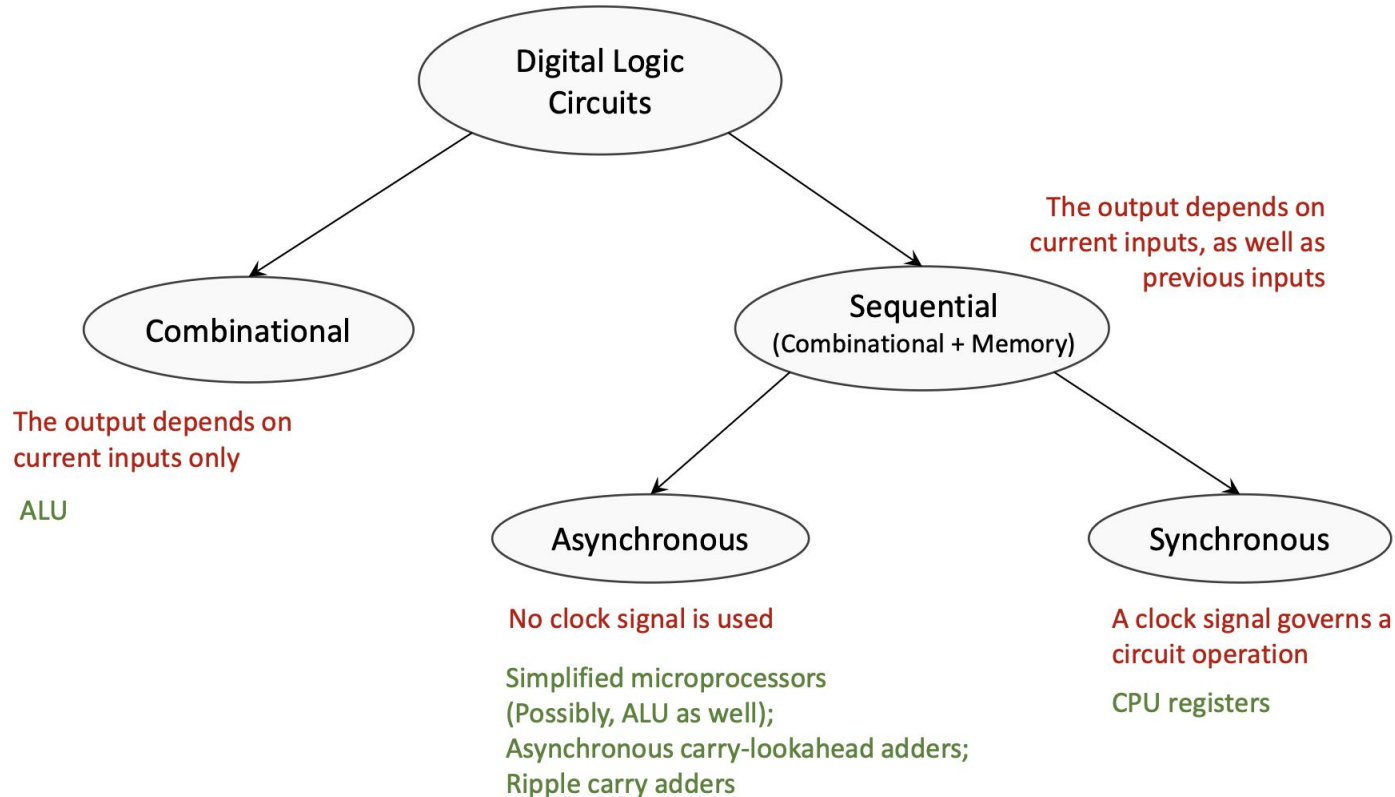


Asynchronous sequential circuits, also known as self-timed or ripple-clock circuits, are digital circuits that do not use a clock signal to determine the timing of their operations. Instead, the state of the circuit changes in response to changes in the inputs.

Flip-Flop – synchronous circuit: updates of output signal Q are synchronised with a clock signal edge



Digital Circuits Classification Use Cases



Synchronous vs. Asynchronous Circuits

Characteristic	Asynchronous (like S/R latch)	Synchronous (like Flip-Flop)
Output Update Trigger	Any change of one of the inputs	A clock signal, together with other input signals
Clock Presence	No clock present	Clock governs the entire circuit activity
Reliability (for result correctness)	Less reliable (prone to incorrect result, subject to propagation delays)	Reliable (guarantees a correct result, independently of propagation delays)
Memory Element Presence	Not used	Used
Operation Speed	Faster (no clock signal for synchronisation delay is used)	Slower (due to overpessimistic synchronisation delays)
Power Consumption	Lower	Higher (e.g. due to the presence of flip-flops, consuming power for data storage)
Logical Complexity, Size (the number of elements)	Simpler , smaller (Note: an asynchronous circuit might behave as a synchronous, but at the price of a higher hardware implementation complexity)	More complex , larger
Sample Use Cases	Arithmetic-Logic Unit (ALU); Small fast peripheral circuits supporting CPU operation	Register files; Most of the circuits containing memory elements

That's all!

Good Luck!

23 October, 2025

Recap by @scruffyscarf