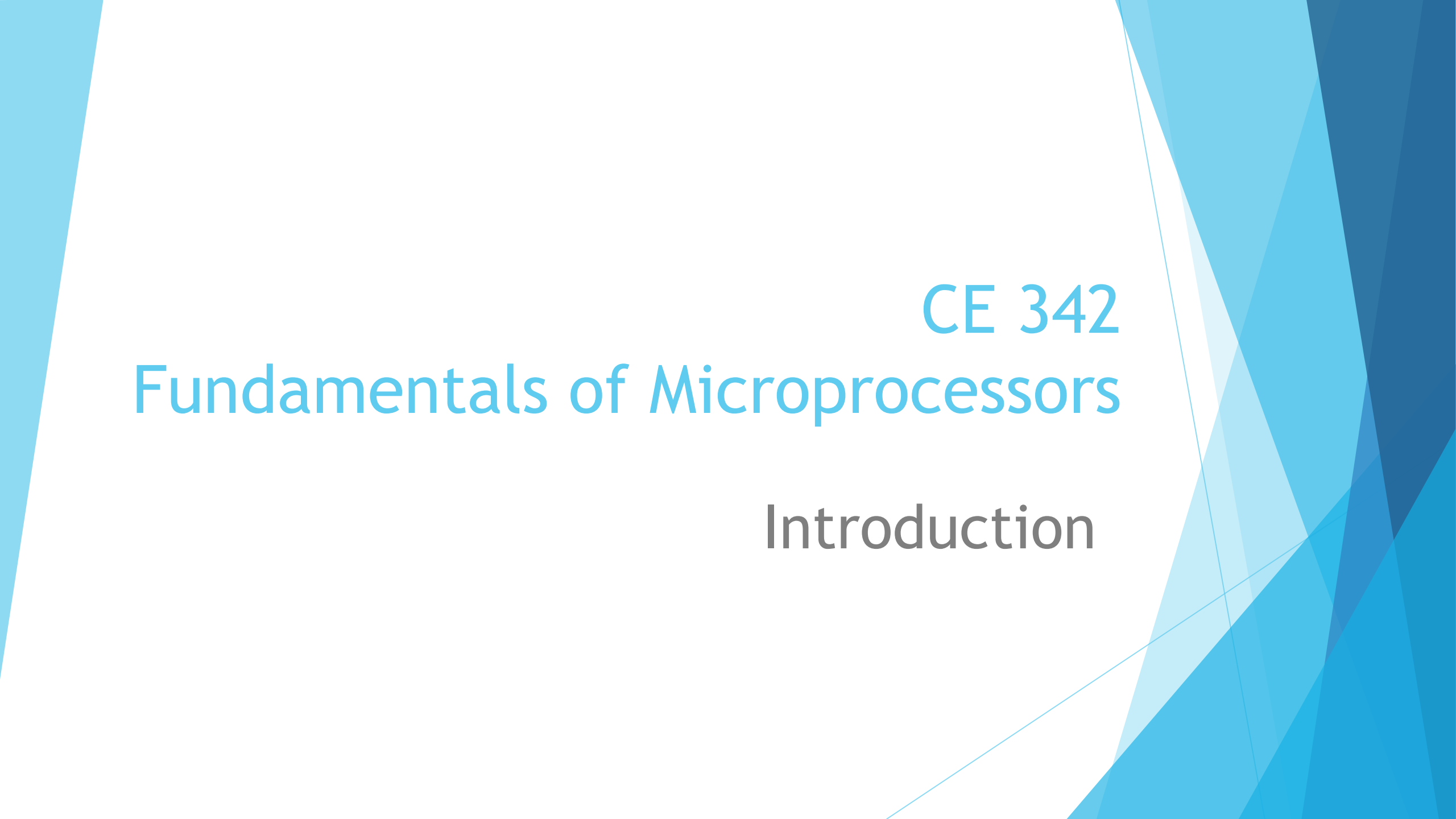


The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the left and right sides of the slide, framing the central text area.

CE 342

Fundamentals of Microprocessors

Introduction

Bits and Bytes

- ▶ First, we must consider how information can be stored inside computers.
- ▶ By “Information,” we mean numbers, text, images, sound, video, ..., etc.
- ▶ For today’s computers, information is encoded as patterns of 0’s and 1’s called bits (Binary digits).
- ▶ The reason we use only two symbols (0 and 1) for information encoding is because it’s *simple*, not because it’s powerful.
- ▶ A **Byte** is 8 bits, and a **Word** is 2 bytes.

Digital Data Representation

- ▶ Some possible meanings for a single bit:
 - ▶ Numeric value (1 or 0)
 - ▶ Boolean value (true or false)
 - ▶ Voltage (high or low)
- ▶ A bit can only represent one of two values, for more values, we need a long string of bits to represent them.

Boolean Operation

- ▶ Computers are made of small devices that can compute Boolean functions extremely fast.
- ▶ Boolean operations are operations that manipulate one or more 1/0 (or true/false) values.
- ▶ Some examples of Boolean operations:

■ AND: $\frac{\text{AND } 0}{0}$ $\frac{\text{AND } 1}{0}$ $\frac{\text{AND } 0}{1}$ $\frac{\text{AND } 1}{1}$

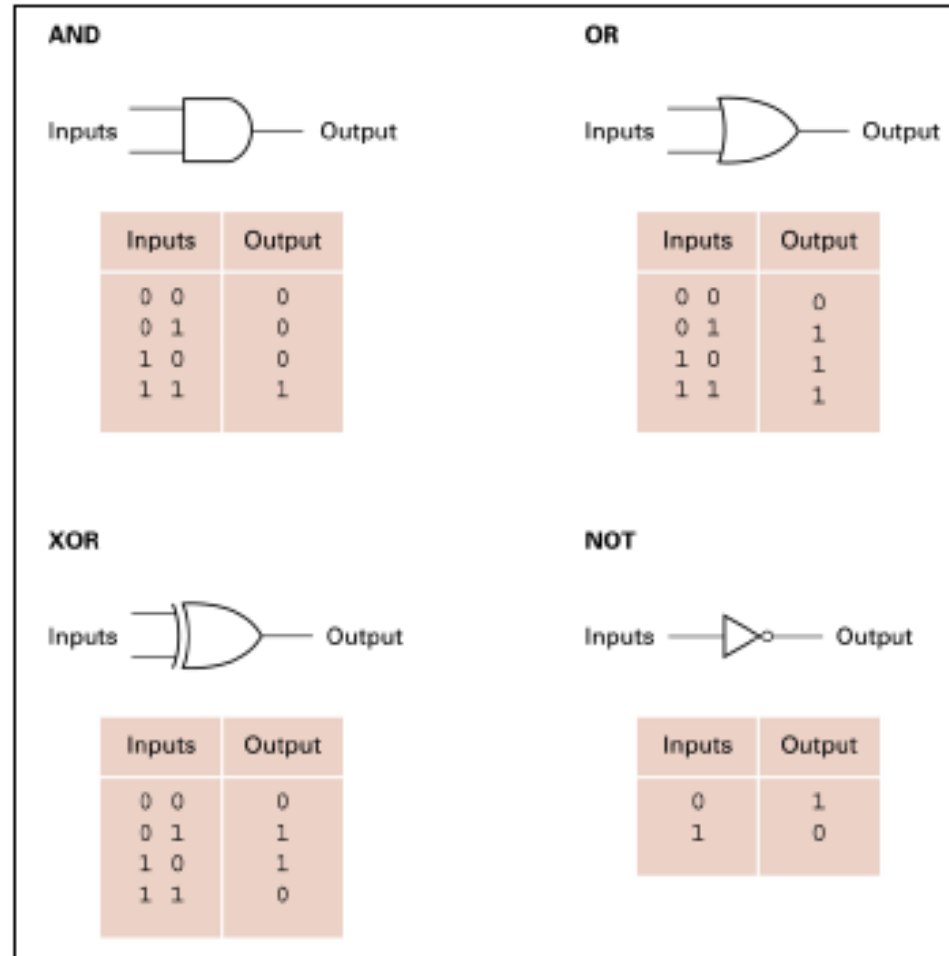
■ OR: $\frac{\text{OR } 0}{0}$ $\frac{\text{OR } 1}{1}$ $\frac{\text{OR } 0}{1}$ $\frac{\text{OR } 1}{1}$

■ XOR: $\frac{\text{XOR } 0}{0}$ $\frac{\text{XOR } 1}{1}$ $\frac{\text{XOR } 0}{1}$ $\frac{\text{XOR } 1}{0}$

■ NOT: $\frac{\text{NOT } 0}{1}$ $\frac{\text{NOT } 1}{0}$

Gates

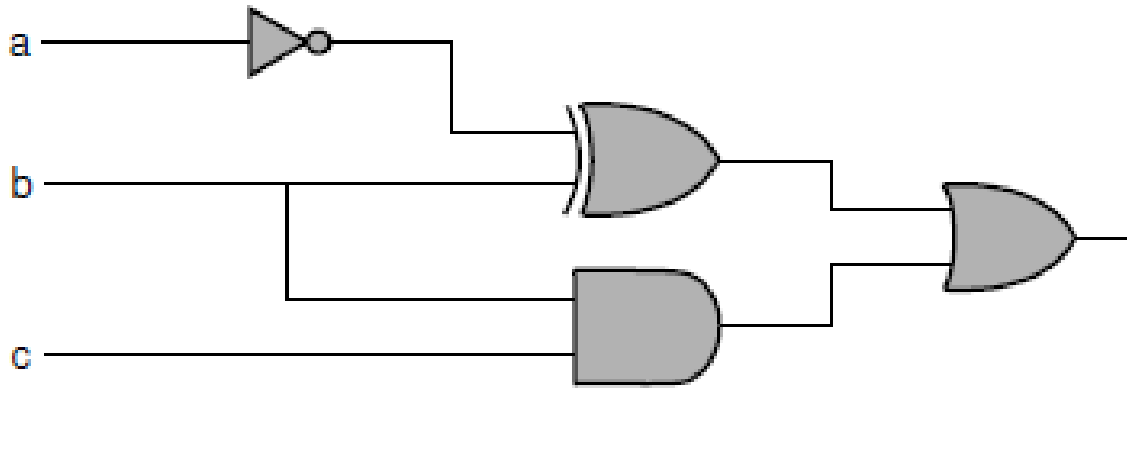
- ▶ The small devices that compute Boolean operations are called “gates”.
- ▶ Often implemented as electronic circuits.
- ▶ The building blocks from which computers are constructed.



Sample Combinational Circuit

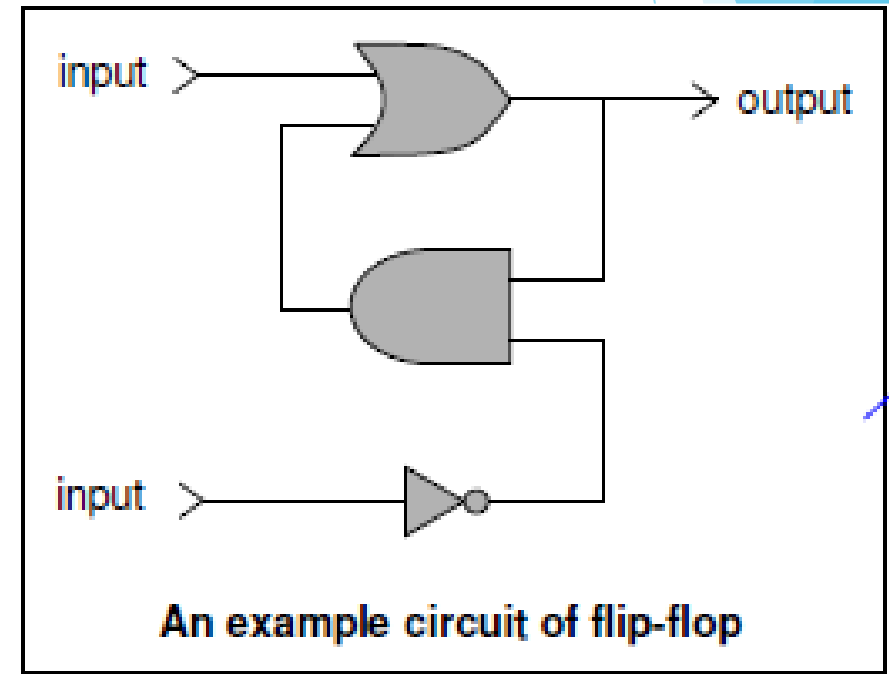
- ▶ A digital circuit's operation can be summarized by a *truth table*:

Input a, b, c	Output
000	
001	
010	0
011	1
100	0
101	0
110	1
111	1



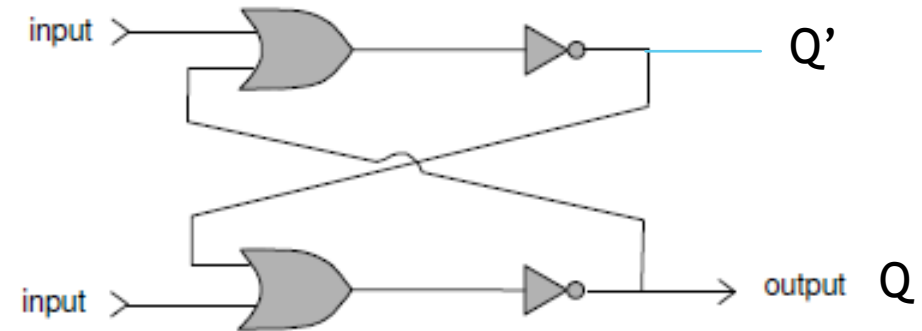
Flip Flops (Sequential Circuits)

- ▶ In addition to performing Boolean operations, gates can also be used to store information.
- ▶ A **flip-flop** is built from gates that stores one bit of data.
 - ▶ It has an input line which sets its stored value to 1.
 - ▶ It has an input line which sets its stored value to 0.
- ▶ When both inputs = 0, most recently stored value is preserved.



Another Example of a Flip Flop

- ▶ This is a different design.
- ▶ The inputs are usually referred to as R-S, J-K, or D.
- ▶ Outputs are referred to as Q and Q'.
- ▶ This is an RS Flip Flop, and its truth table is also given.
- ▶ Also, timing of the signals is important in these devices.



Inputs		Outputs	
S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	invalid	
1	1	invalid	

Radix-N Number Coding

- ▶ Bit numbers are coded in radix-2 (a.k.a. base-2) while human-friendly numbers are coded in radix-10.
- ▶ The conversion between radix-2 and radix-10 is not trivial, sometimes, it is easier to use a number coding system that is:
 - ▶ Easily readable by human,
 - ▶ Can be converted to/from radix-2 easily.
- ▶ Popular radix-N notation that fulfill these two points are radix-8 (octal notation) and radix-16 (hexadecimal notation).

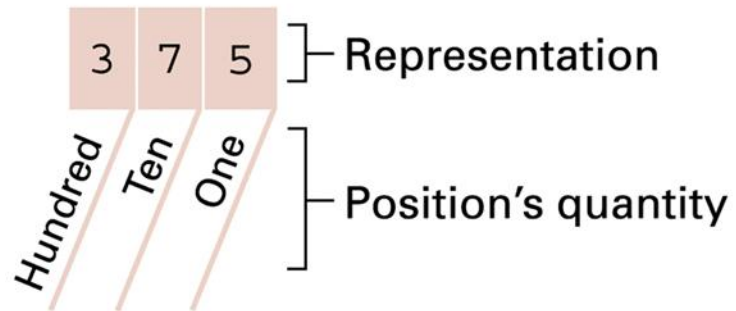
Different Number Systems

Binary	Octal	Hexadecimal	Decimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

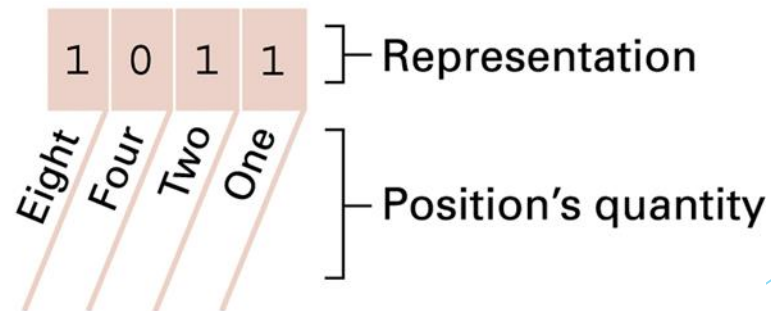
The Binary System

- ▶ The traditional decimal system is based on powers of **ten**.
- ▶ The Binary system is based on powers of **two**.

a. Base ten system



b. Base two system



Hexadecimal Notation

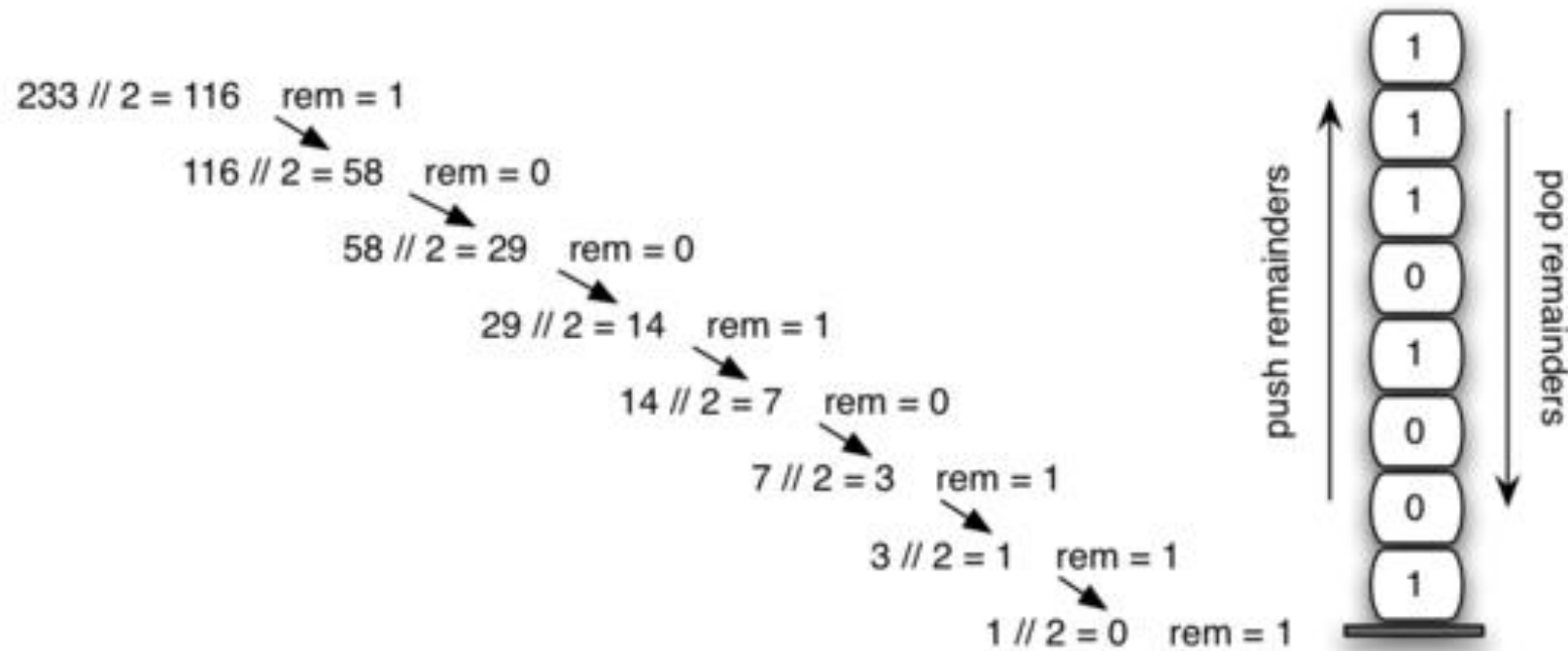
- ▶ **Hexadecimal notation:** A shorthand notation for long bit patterns.
 - ▶ Divides a pattern into groups of four bits each.
 - ▶ Represents each group by a single symbol.
- ▶ **Example:** 1010 0011 becomes **A3**.

Representing Numeric Values

- ▶ **Binary notation:** Uses bits to represent a number in base two.
- ▶ **Limitations of computer representations of numeric values:**
 - ▶ **Overflow** - happens when a value is too big to be represented.
 - ▶ **Truncation** - happens when a value is between two representable values.

Converting Decimal to Binary

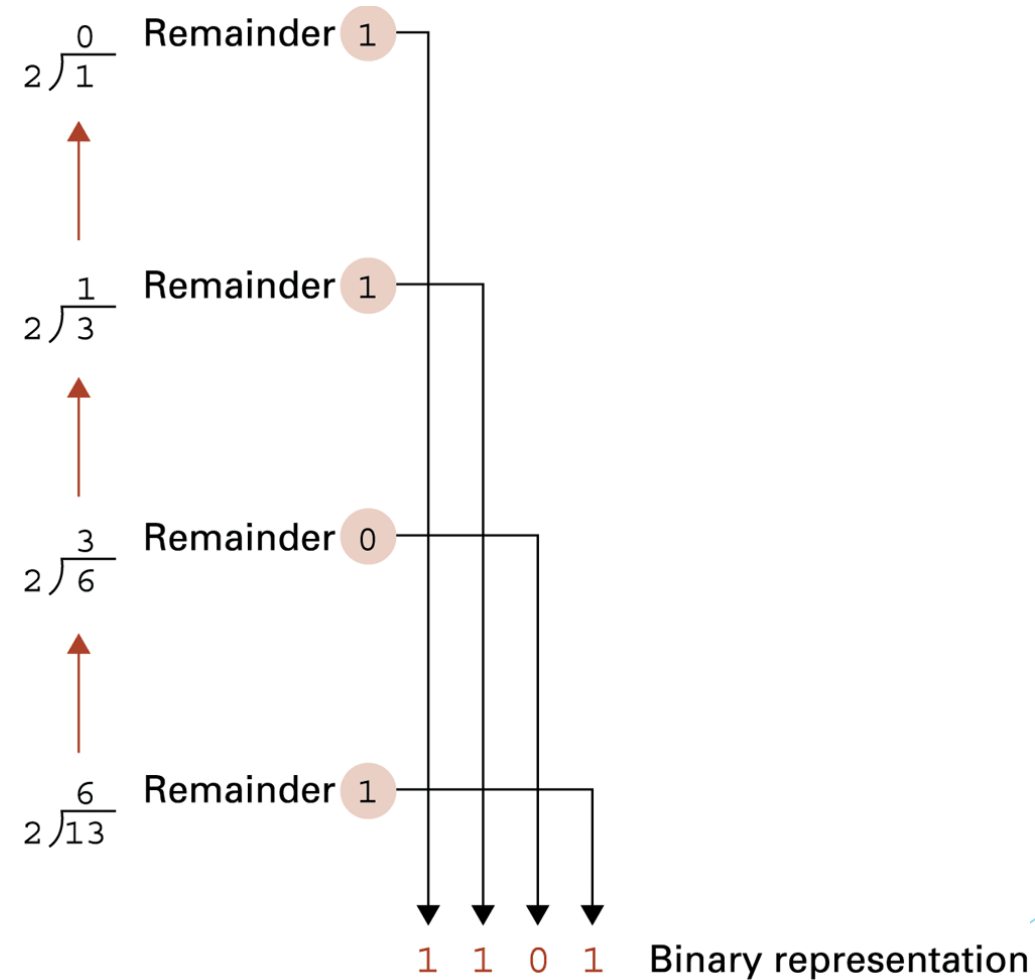
- ▶ We shall do the opposite of what we did when we converted binary to decimal ie. Divide



An algorithm for finding the binary representation of a positive integer

- Step 1.** Divide the value by two and record the remainder.
- Step 2.** As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.
- Step 3.** Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Applying the algorithm to obtain the binary representation of thirteen



The binary Arithmetic - Addition facts

$$\begin{array}{r} 0 \\ + 0 \\ \hline 0 \end{array}$$

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

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

$$\begin{array}{r} 1 \\ + 1 \\ \hline 10 \end{array}$$

An overflow!

Storing Integers

- ▶ **Two's complement notation:** The most popular means of representing integer values.
- ▶ **Excess notation:** Another means of representing integer values.
- ▶ Both can suffer from overflow errors.

2's complement

- ▶ Invert each bit in the number and add 1.

- ▶ Example:

12 = 1100 (in binary)

- ▶ Invert each bit:

0011

- ▶ Add 1:

$$\begin{array}{r} 0011 \\ + \quad \underline{\quad} 1 \underline{\quad} \\ \hline 0100 \end{array} \quad (2's \text{ complement of } 12 = 4)$$

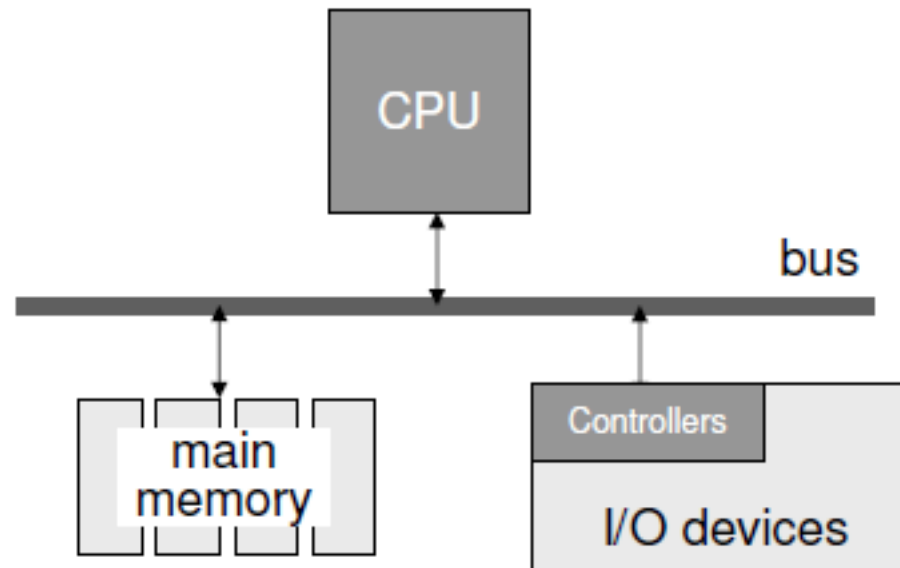
Representing Text

- ▶ Each character (letter, punctuation, etc.) is assigned a unique bit pattern.
 - ▶ **ASCII:** Uses patterns of 7-bits to represent most symbols used in written English text.
 - ▶ **Unicode:** Uses patterns of 16-bits to represent the major symbols used in languages worldwide.
 - ▶ **ISO standard:** Uses patterns of 32-bits to represent most symbols used in languages worldwide.

01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

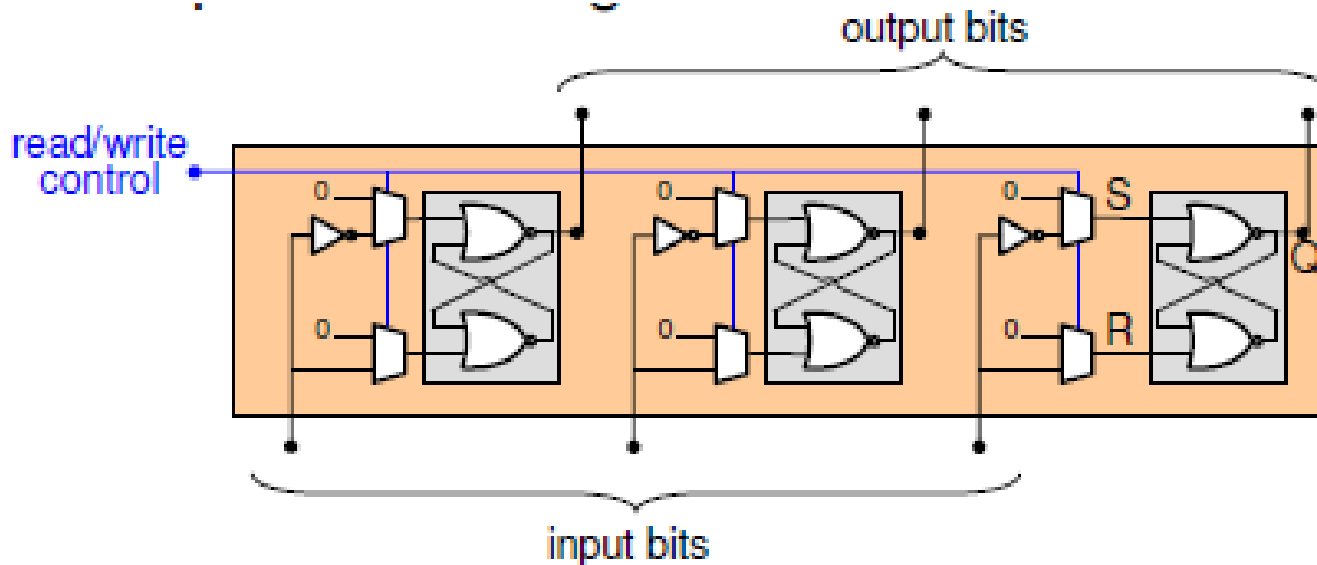
Computer Architecture

- ▶ Central Processing Unit (CPU) contains
 - ▶ Arithmetic/Logic Unit (ALU)
 - ▶ Control Unit
 - ▶ Registers
 - ▶ Cache Memory
- ▶ Bus
- ▶ Main Memory
- ▶ I/O devices



Register

- ▶ A register is a special memory cell inside the CPU that can store an n -bit data.
- ▶ For modern CPUs, n is usually 8, 16, 32, or 64.
- ▶ Registers are used to store intermediate computation results.
- ▶ An n -bit register is composed of a group of n flip-flops.
- ▶ **Example 3-bit register:**

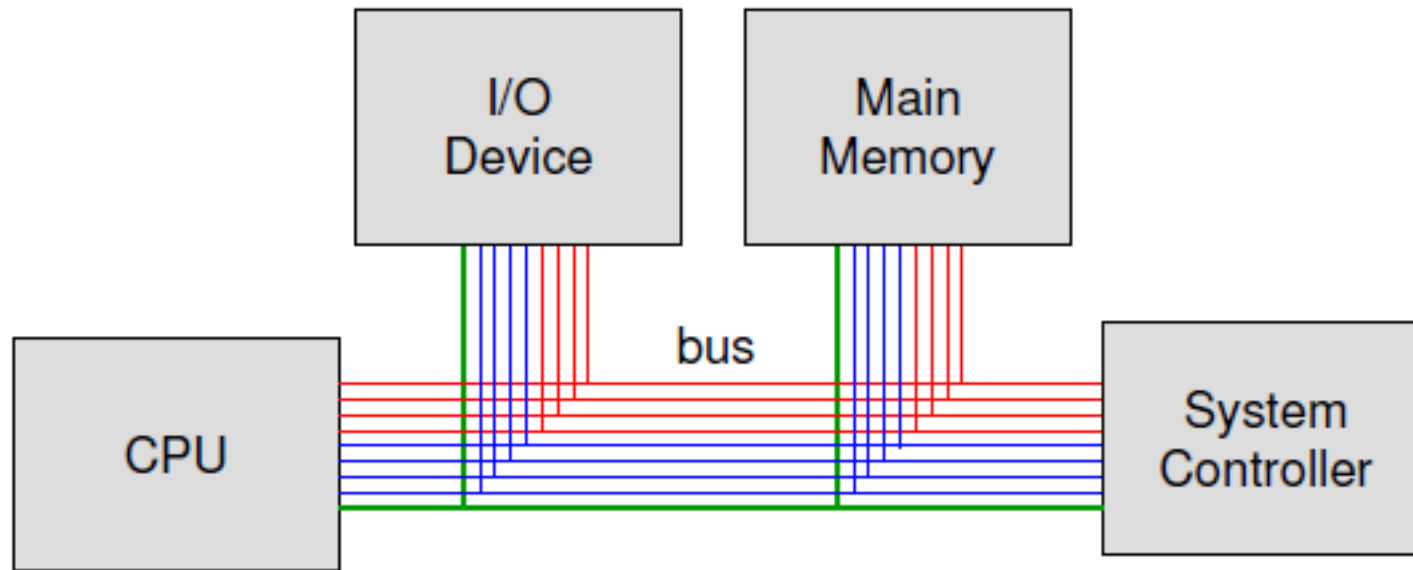


The truth table of the flip-flop is

S	R	Q
0	0	Previous value
1	0	0
0	1	1
1	1	Don't care

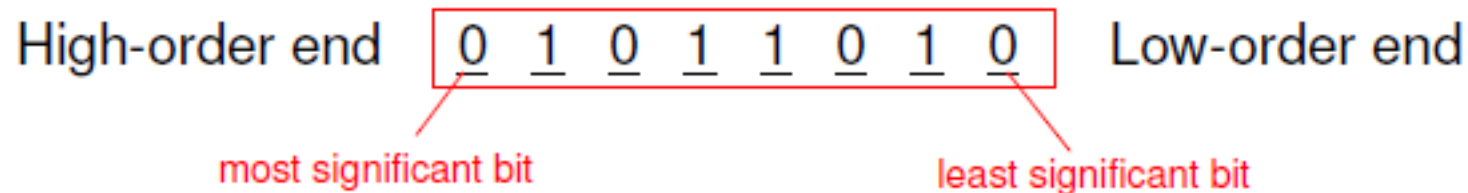
Bus

- ▶ A bus is a collection of wires to connect a computer's CPU, memory, and I/O devices.
- ▶ Information (0's and 1's) are transmitted among the CPU, memory, and devices via the bus following some handshaking rules.



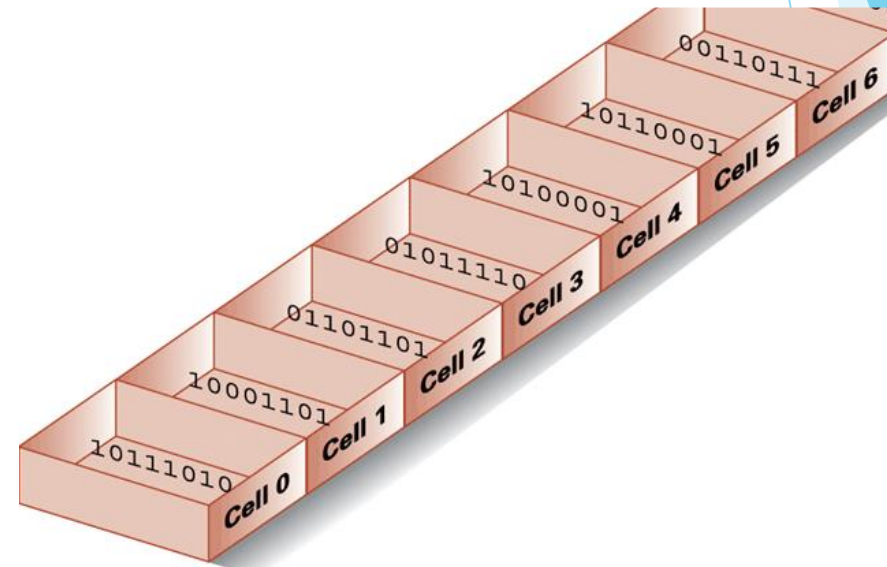
Main Memory Logical Units - Cells

- ▶ Data are stored in memory cells in a computer system; each cell is a memory access unit.
- ▶ The smallest memory access unit of the CPU is called a *byte*, usually equals 8 *bits* (If you want to emphasize that it is composed of 8 bits, you can call it an “octet”).
- ▶ Bits in a byte has orders:
 - ▶ **Most significant bit:** the bit at the left (high-order) end of the conceptual row of bits in a memory cell.
 - ▶ **Least significant bit:** the bit at the right (low-order) end of the conceptual row of bits in a memory cell.



Main Memory Addresses

- ▶ **Address:** A “name” that uniquely identifies one cell in the computer’s main memory.
 - ▶ The names are given in numerical format.
 - ▶ These numbers are assigned consecutively starting at zero.
 - ▶ Numbering the cells in this manner associates an order with the memory cells.
- ▶ **A Random Access Memory (RAM)** is a memory device where any cell can be accessed independently.
- ▶ **Dynamic Memory (DRAM):** RAM composed of a volatile memory.

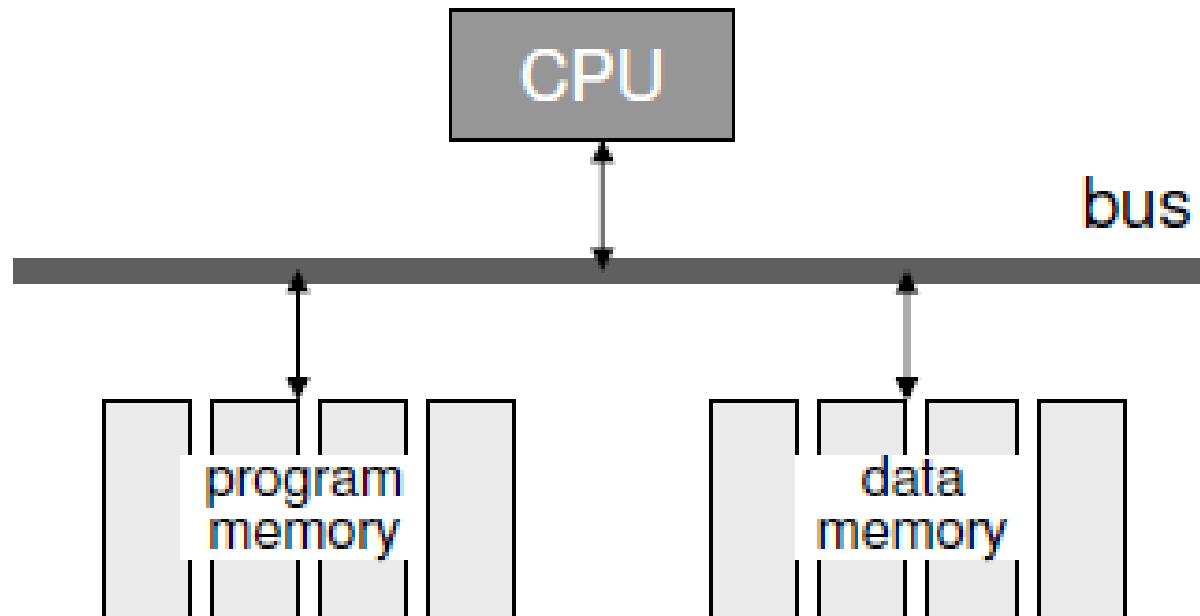


Measuring Memory Capacity

- ▶ **1 Kilobyte:** 2^{10} bytes = 1024 bytes
 - ▶ Example: 3 KB = 3×1024 bytes
- ▶ **1 Megabyte:** 2^{20} bytes = 1,048,576 bytes
 - ▶ Example: 3 MB = $3 \times 1,048,576$ bytes
- ▶ **1 Gigabyte:** 2^{30} bytes = 1,073,741,824 bytes
 - ▶ Example: 4 GB = $4 \times 1,073,741,824$ bytes

Stored Program Concept

- ▶ A program is a sequence of *instructions* that implements an algorithm.
- ▶ A program is just a special type of data and can be stored in main memory.
 - ▶ Program memory and data memory can be shared or separate; e.g., separate memory architecture:



What Do Instructions Look Like

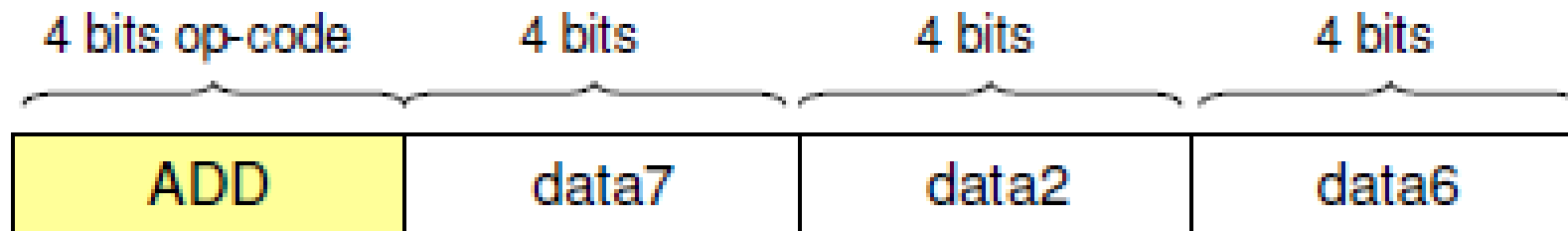
- ▶ When you ask other people to do something, you use human languages (Turkish, English, etc.).
- ▶ However, human languages are too cumbersome and too ambiguous for computers to ***decode*** and ***execute!***
- ▶ Machine languages must be:
 - ▶ **Concise** - easy to decode.
 - ▶ **Precise** - each instruction can be executed in only one way.

Machine Language

- ▶ A machine instruction is an instruction coded as a bit pattern directly recognizable by the CPU.
- ▶ A machine language is the set of all instructions recognizable by a CPU.
 - ▶ Each CPU has its own machine language, called the Instruction Set Architecture (ISA) of the CPU.
- ▶ The bit-pattern of a machine instruction can be divided into two parts: **op-code field** and **operand field**.

Structure of a Machine Instruction

- ▶ Op-code field
 - ▶ Specifies which machine operation to execute.
 - ▶ One per instruction.
- ▶ Operand field (a.k.a. addressing mode field)
 - ▶ Data and addresses related to this operation.
 - ▶ Number of operands varies depending on op-code.
- ▶ Example: the instruction that asks CPU to add **data2** and **data6** and store the result in **data7** can be coded in a 16-bit number as follows:



Machine Language and Architecture

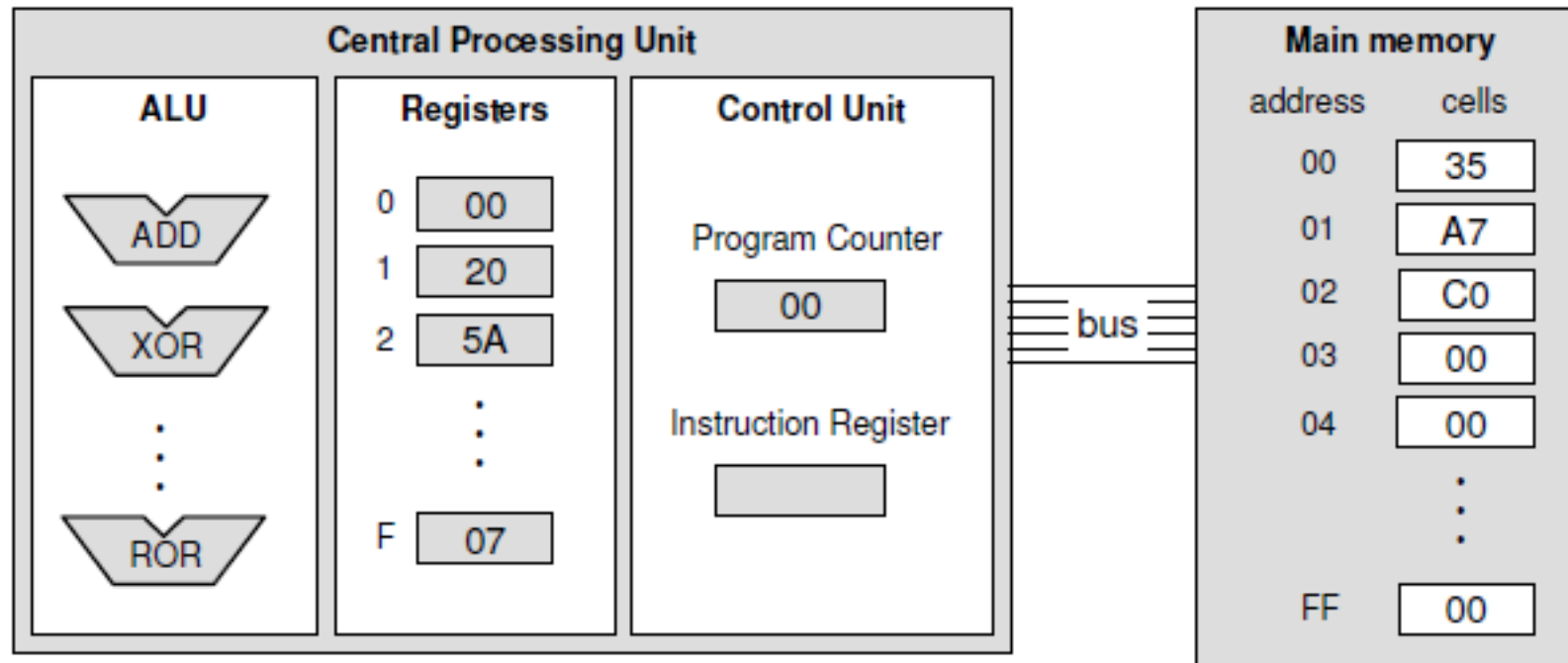
- ▶ Machine language reflects the architecture of the CPU, and there are two major types of design philosophy: **RISC** and **CISC**.
- ▶ **Reduced Instruction Set Computer (RISC):**
 - ▶ CPU only executes a small set of simple instructions.
 - ▶ However, CPU executes these simple instructions real fast.
 - ▶ Usually use **fixed-length** coding of instructions.
- ▶ **Complex Instruction Set Computer (CISC):**
 - ▶ CPU executes many complex and powerful instructions
 - ▶ Complex instructions takes longer to execute, but an algorithm implemented in CISC machine language requires less instructions than that of RISC's.
 - ▶ Usually use **variable-length** coding of instructions.

Machine Instruction Types

- ▶ The instructions of a machine language can be classified into three groups:
 - ▶ **Data Transfer:** copy data between CPU registers and/or main memory cells.
 - ▶ **Arithmetic/Logic Operation:** use existing data values (stored in registers or main memory cells) to compute a new data value.
 - ▶ **Control:** direct the execution flow of the program.

Example: A Simple CPU

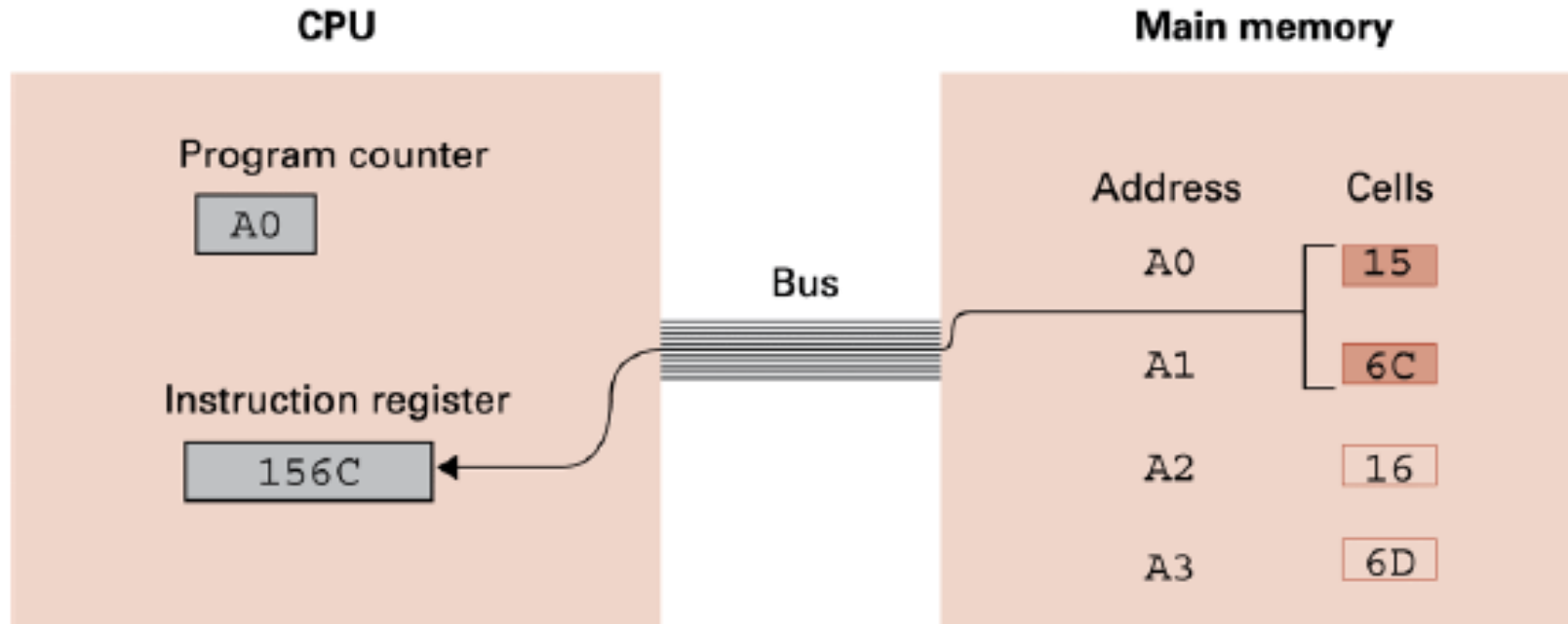
- ▶ A simple CPU architecture



Program Execution

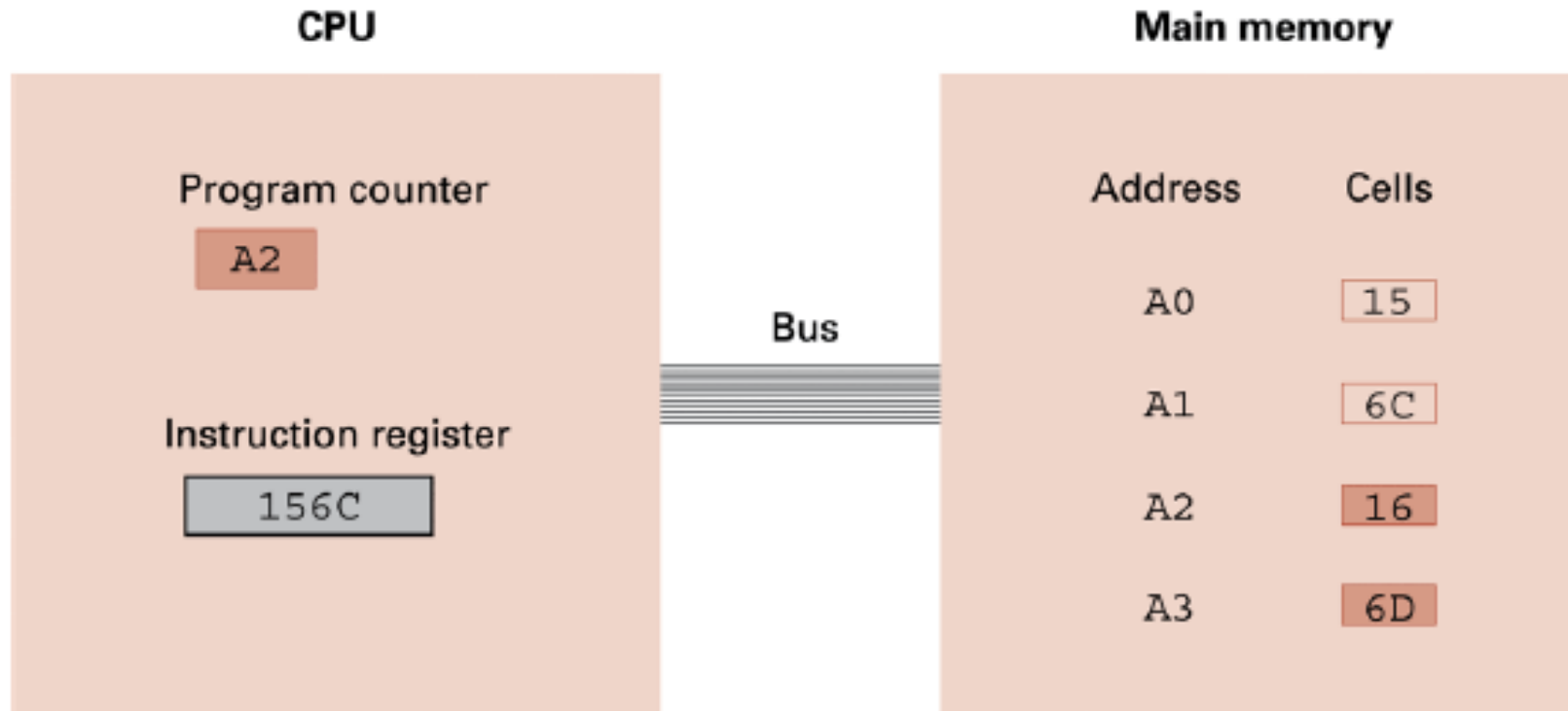
- ▶ Controlled by two special-purpose registers
 - ▶ **Program counter**: address of the next instruction.
 - ▶ **Instruction register**: holds the current instruction.
- ▶ Each machine cycle of program execution is composed of three steps:
 - ▶ **Fetch** - copies memory cells addressed by the program counter to the instruction register and increment the program counter to the next instruction in main memory.
 - ▶ **Decode** - decodes the bit pattern in the instruction register to determine the operation required and the related operands.
 - ▶ **Execute** - performs the operation specified by the instruction.

Fetch Step (1/2)



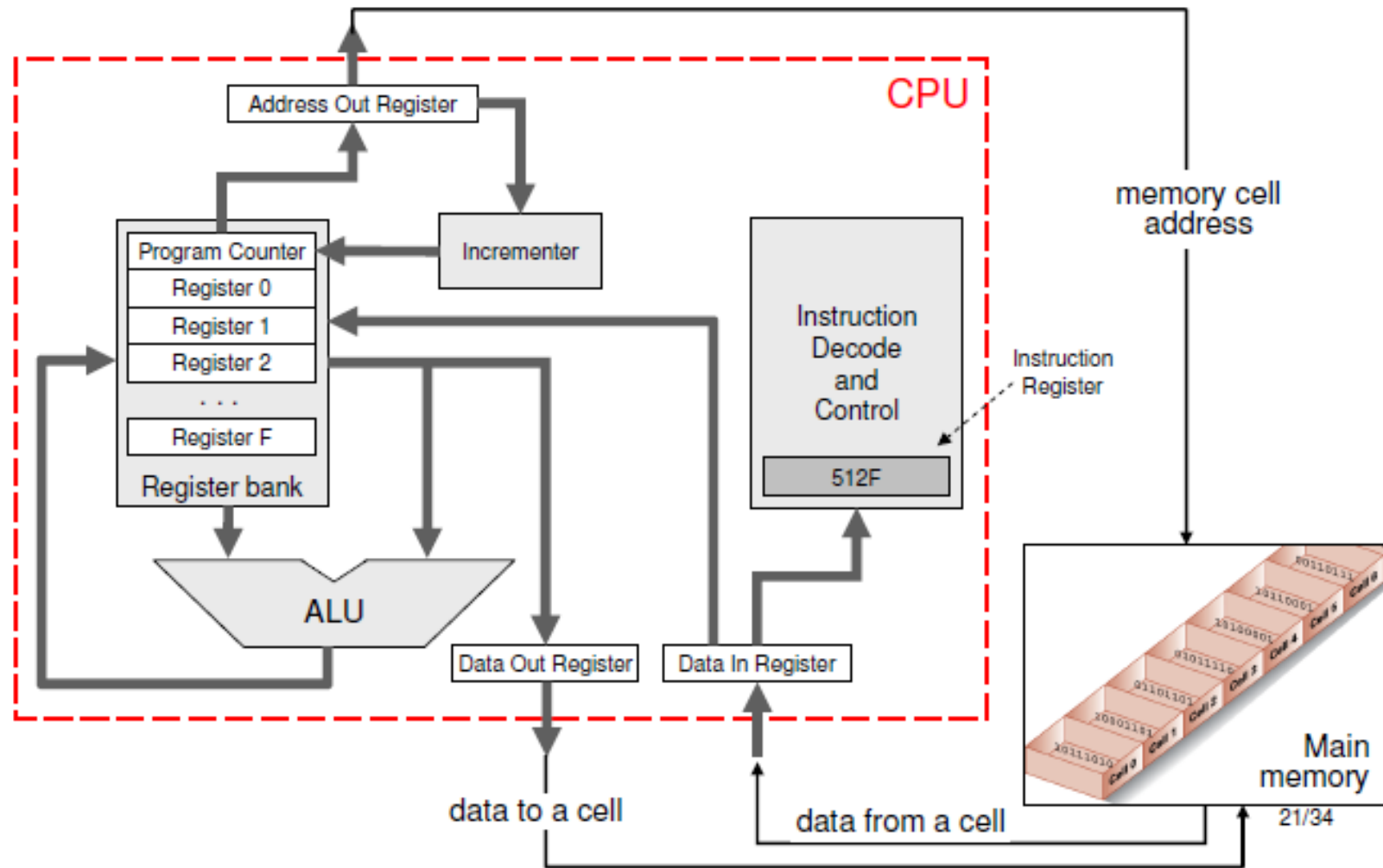
At the beginning of the fetch step, the instruction starting at address **A0** is fetched from the memory and placed in the instruction register.

Fetch Step (2/2)



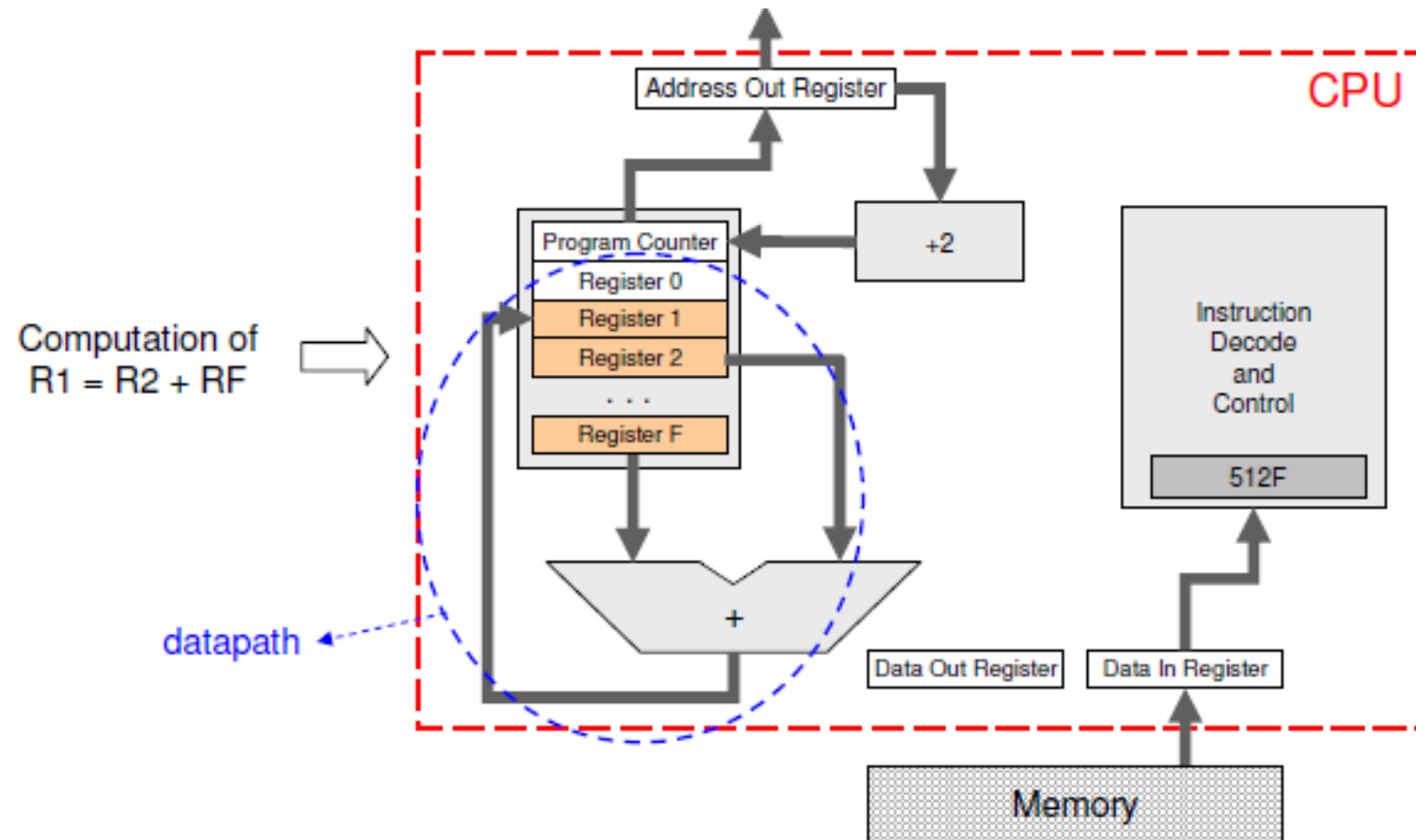
b. Then the program counter is incremented so that it points to the next instruction.

Internal Structure of a Realistic CPU



Execution of an Instruction

After decoding of an “**add**” instruction, the *data path* of a CPU may become as follows:

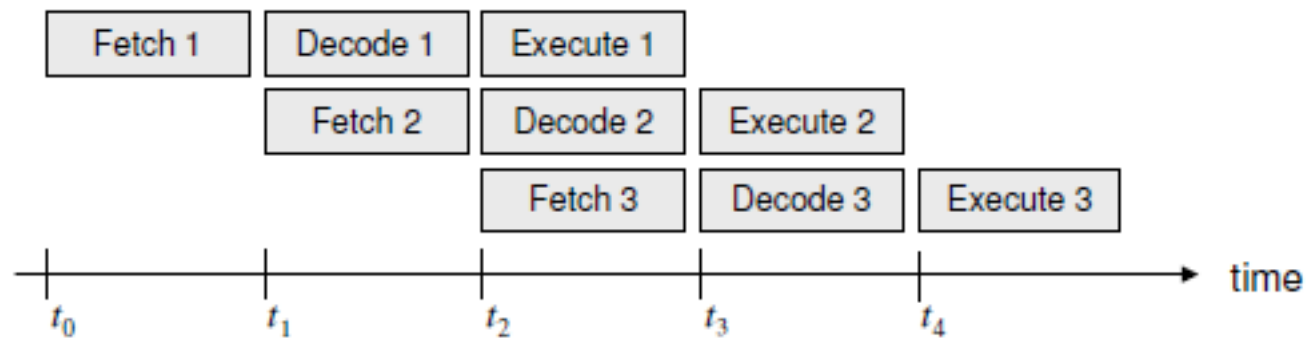


Arithmetic/Logic Operations

- ▶ The Arithmetic/Logic Unit (ALU) of a CPU is the muscle that performs data manipulation.
- ▶ Three types of operations are supported by ALU:
 - ▶ **Arithmetic:** add, subtract, multiply, divide, etc.
 - ▶ **Logic:** AND, OR, XOR, NOT, etc.
 - ▶ **Rotate and Shift:** rotate (a.k.a. circular shift), logical shift, arithmetic shift, etc.

Improving CPU Architecture

- ▶ **Pipelining:** overlap the steps of the CPU operation cycles.



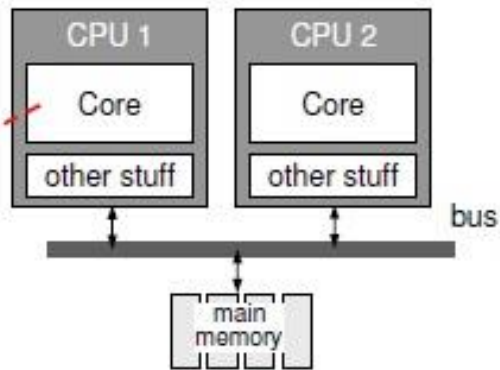
- ▶ **Parallel processing:** execute multiple operations simultaneously.
 - ▶ It can be performed within a CPU or across CPUs.

Multiprocessor Systems

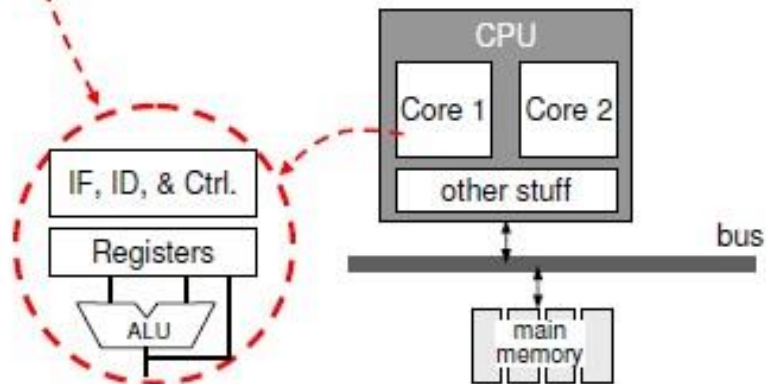
- ▶ A single processing unit execute one instruction a time, which is called Single-Instruction stream Single-Data stream (SISD) architecture.
- ▶ If multiple processing units (multi-CPU, multi-core, or multi-ALU) are connected to the main memory, we have parallel processing architecture:
 - ▶ Multiple-Instructions stream Multiple-Data stream (MIMD): different instructions are issued at the same time to operate on different data.
 - ▶ Single-Instruction stream Multiple-Data stream (SIMD): one instruction is issued to operate on different data.

Multi-CPU, Multi-Core, Multi-ALU

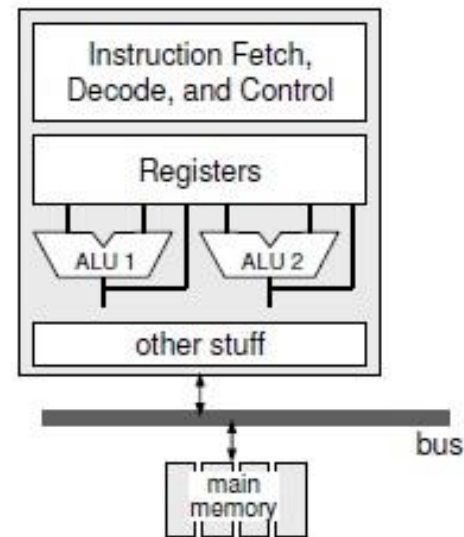
❑ Multi-CPU Architecture:



❑ Multi-Core Architecture:



❑ Multi-ALU Architecture:



Note: "other stuff" could be interface logic, cache, MMU, timer, ... etc.