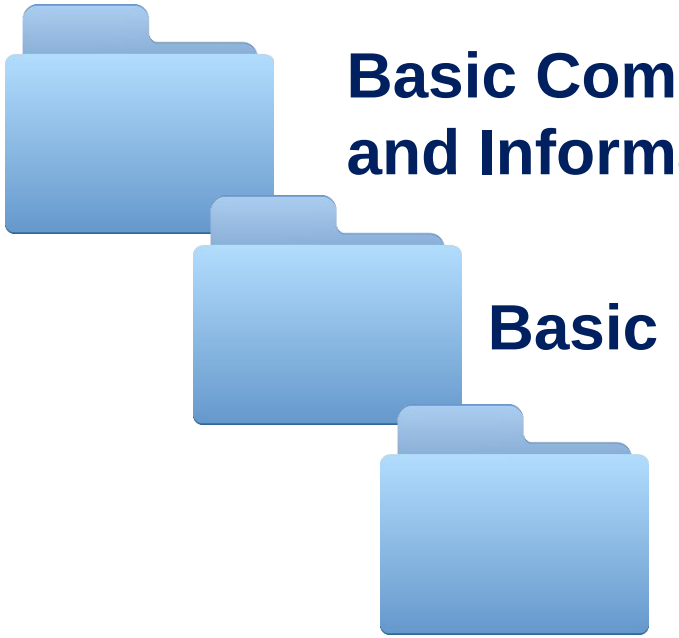
Basic Computer Operations

Lesson Outcomes

At the end of this lesson, you should be able to:

- Describe the basic organization and modules in a basic computer
- Describe the information representations used by a computer
- State the functional units in the Central Processing Unit (CPU) of a computer
- Describe the main functions of each units within the CPU
- Articulate the stages involves in typical CPU operations

Topic Outline



Basic Computer Organization and Information Representations

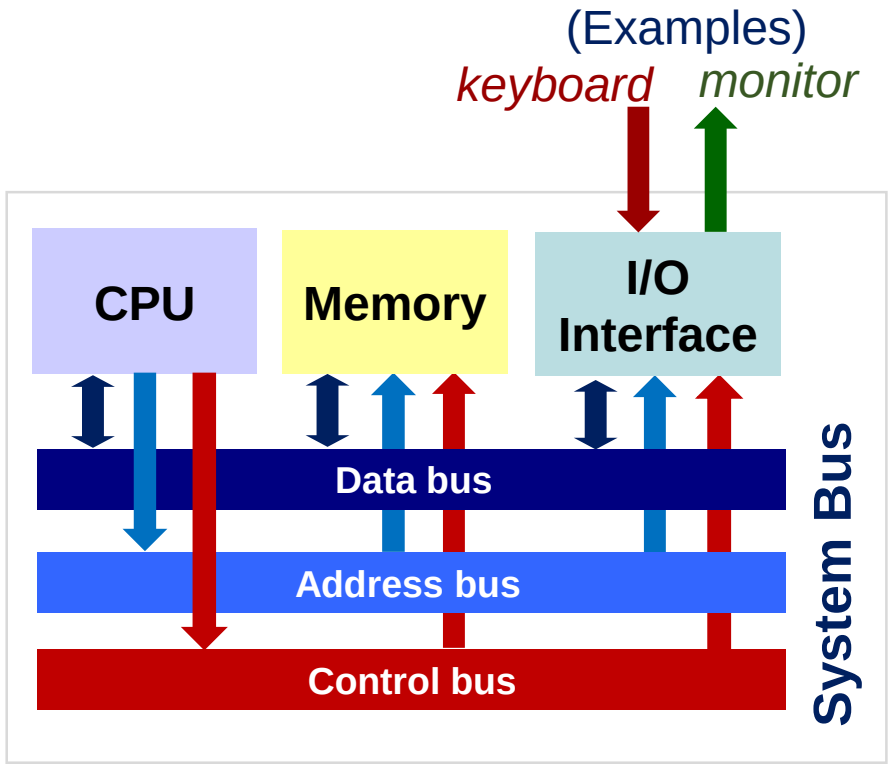
Basic Central Processing Unit (CPU) Structure

Basic CPU Operations

Basic Computer Organization and its Information Representations

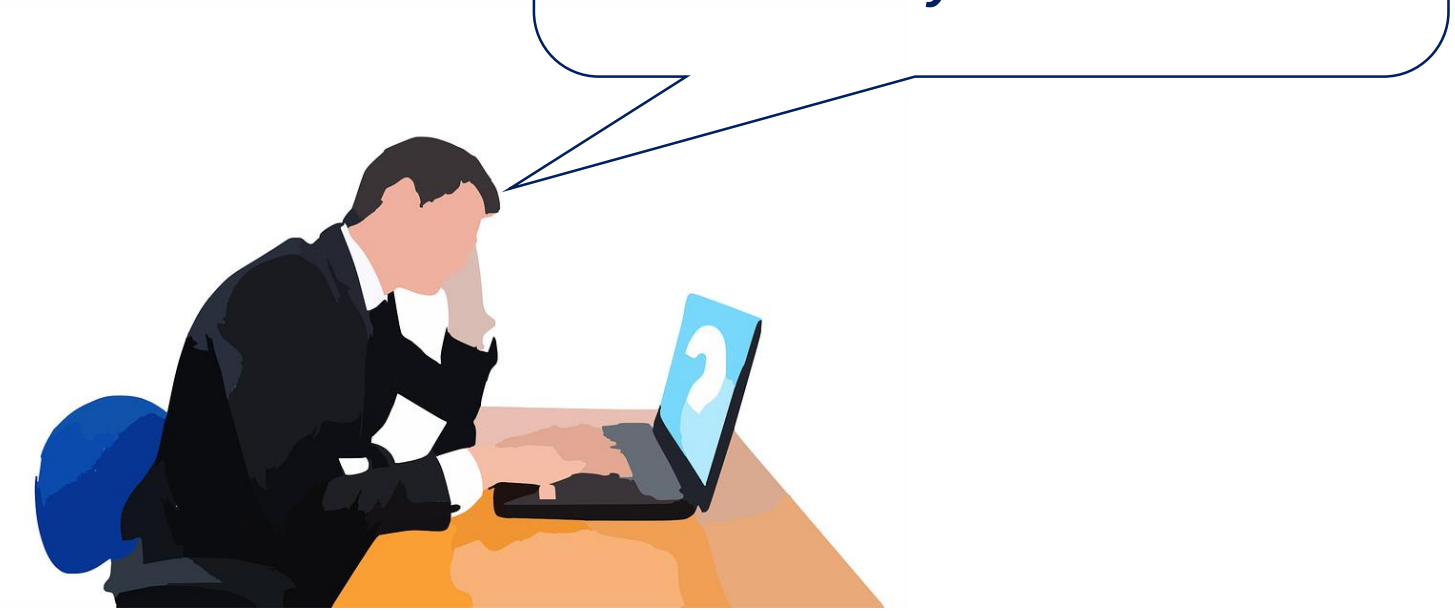
Basic Computer Organization

- **Central Processing Unit (CPU)**
  - processes information
  - performs operations based on information given
- **Memory**
  - stores information used by the computer
- **Input/ Output (I/O) Interface**
  - mechanism for transferring information to and from the outside world, such as to interact with users
- **System Bus**
  - providing the connection between the modules



*In practice, these modules are now commonly integrated together in one component and known as the **Microprocessor (μP)**.*

What are the types of information, and how are they represented in a computer system?



Types of Information

Consider the following arithmetic operation:

2 + 3

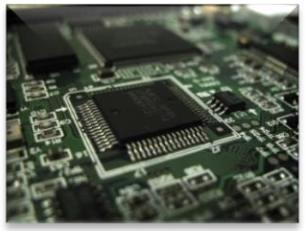
The given arithmetic contains two types of information:

- **Instruction**: to perform addition operation
- **Data**: using the two numbers with values of 2 and 3

In what form is the information presented to the computer?

Information Representation

Microprocessor in a computer is a digital device.



It consists of electronic components (e.g. transistor) that operate in **two states** (either turn-**on** or turn-**off**).

Conveyed in the form of **voltage levels**:

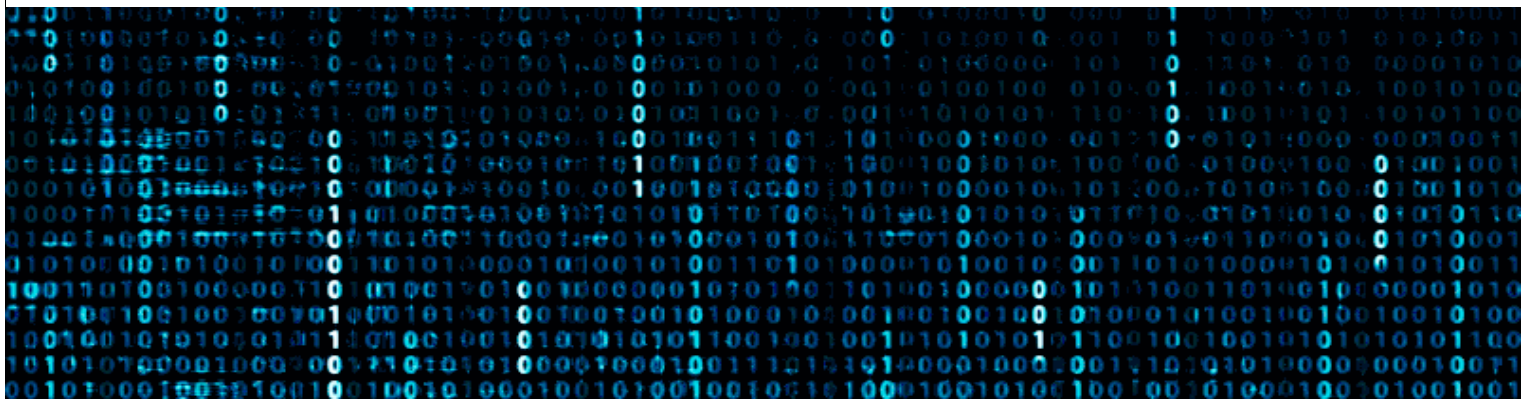
- **Low** voltage (0 volt)
- **High** voltage (depends on the power supply used by μP)

Represented by using **binary digit** (or one **bit** in short form):

- '**0**' for low voltage
- '**1**' for high voltage

Information representation using binary bits is known as in

Binary Format.



Binary Format

A datum (e.g. a number) that consists of one bit:

- can only have two values: either 0 or 1
- is not very useful in practice

Hence, multiple bits are combined together to give a wider range (or more variety of combinations).

Example of a number using 4 bits:

1010_2 (or written as 1010b)

The subscript '2' or 'b' means the number should be interpreted using a base-2 numeral system.

Value of Binary Number

Example of an 8-bit binary number: 01001101b

General expression for an n-bit binary number: $b_{n-1}\ldots b_k\ldots b_2 b_1 b_0$

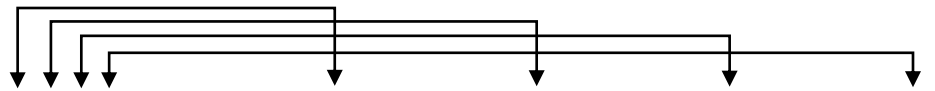
Each bit has a weightage corresponding to its bit position.

The decimal (base-10) value of the n-bit (unsigned) number can hence be calculated as:

$$2^{n-1} \times b_{n-1} + \ldots + 2^k \times b_k + \ldots + 2^2 \times b_2 + 2^1 \times b_1 + 2^0 \times b_0$$

- the k^{th} bit will have a weightage of 2^k.

Example:
Decimal value of **0110b**



$$\mathbf{0110b} = 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0$$

$$= 0 + 4 + 2 + 0$$

$$\equiv \mathbf{6_{10}}$$

Hexadecimal Format

Base-2 binary format: most conveniently used by the computer but it is *not human-friendly*.

e.g. 1011010101101100b

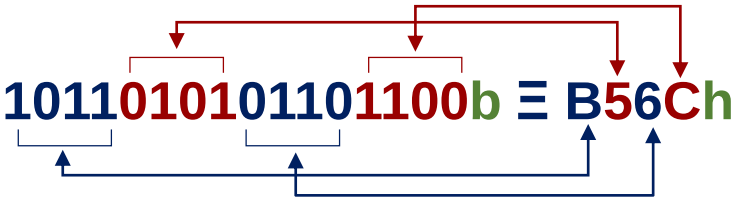
To make it easier for human to read (and write) binary data, **base-16 hexadecimal format** is normally used instead.

Hexadecimal symbols: **0, 1, 2, … 8, 9, A, B, C, D, E, F**

Each hexadecimal symbol (digit) ≡ four (4) binary bits

To convert between the two formats, the binary format data is separated into groups of four (4) bits.

e.g. 1011010101101100b ≡ B56Ch



What is the decimal value of the number represented by the 16-bit data?

Data Size/ Data Width

Conventions used to describe data size/ data width in microprocessor/ computer:

- one **Byte** = 8 bits (2 hex digits)
- one **Nibble** = 4 bits (1 hex digit)
- one **Word** = 16 bits or 32 bits, depending on the natural data size/ data width of the CPU architecture

***32-bit word* will be used in this course.**

- one **DWord** = 64 bits

Microprocessor Basic Structure

Outline

Simplified Model of a Microprocessor Structure that consists of the Three Modules connected by the System bus:

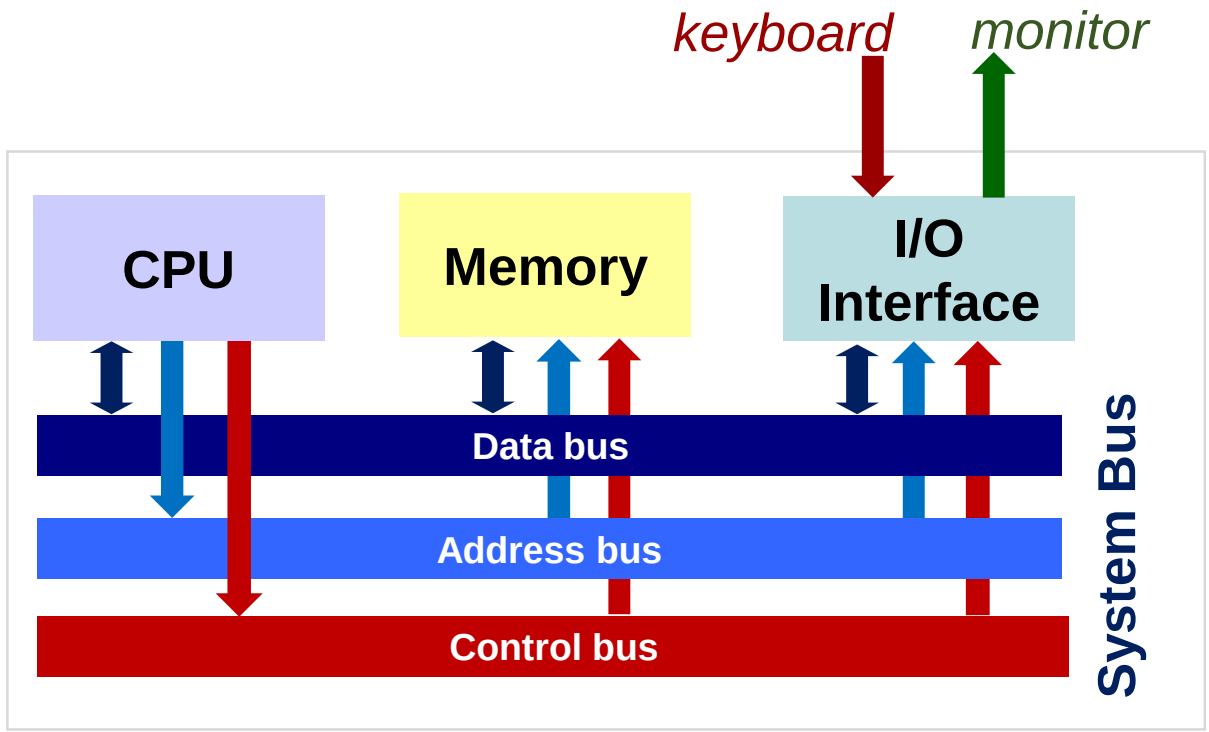
- *Memory*
- *CPU*
- *Input/ Output Interface*
- *System bus*

Basic Computer Organization

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Recap

- **Central Processing Unit (CPU)**
  - processes information
- **Memory**
  - stores information used by the computer
- **Input/ Output (I/O) Interface**
  - interact with users
- **System Bus**
  - providing the connection between the modules



Microprocessor

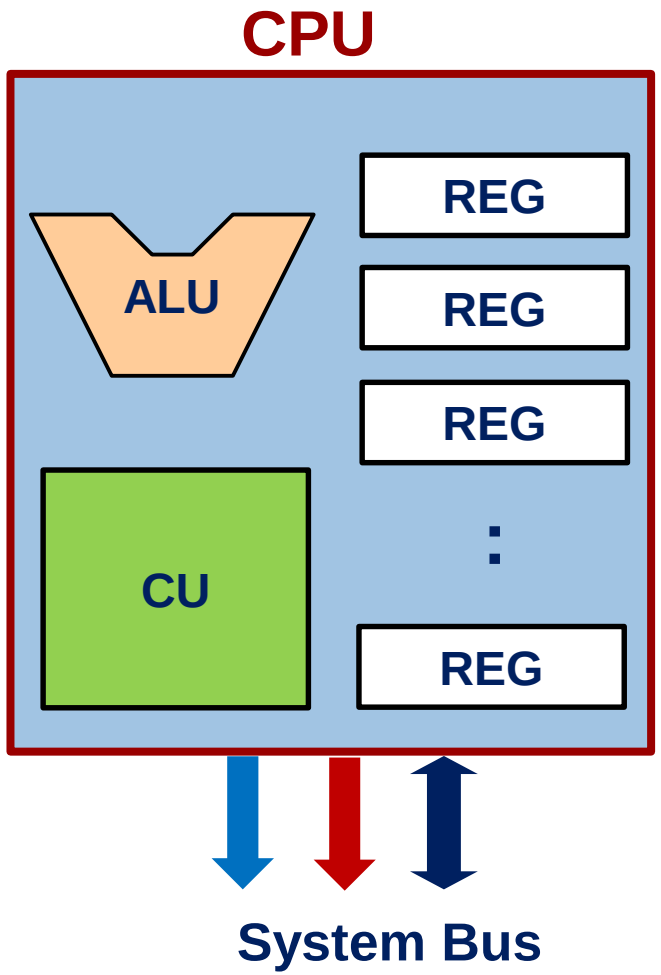
Basic CPU Internal Structure

Three main Functional Units within the CPU:

- **Control Unit (CU)**
  - controls and coordinates the overall operation of the CPU

- **Arithmetic/ Logic Unit (ALU)**
  - performs the arithmetic operations as well as Boolean logic functions

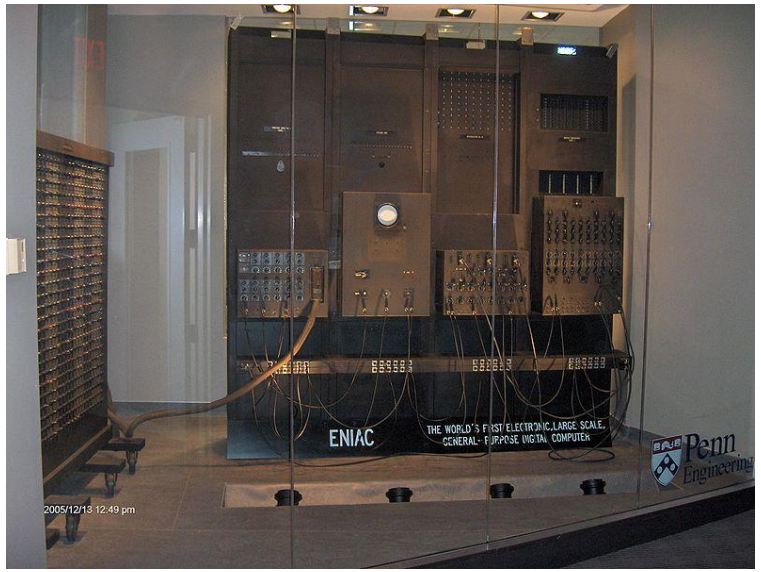
- **Register Array**
  - holds the various information used by CPU operations



These functional units are connected internally by wires that are extended to the external system bus.

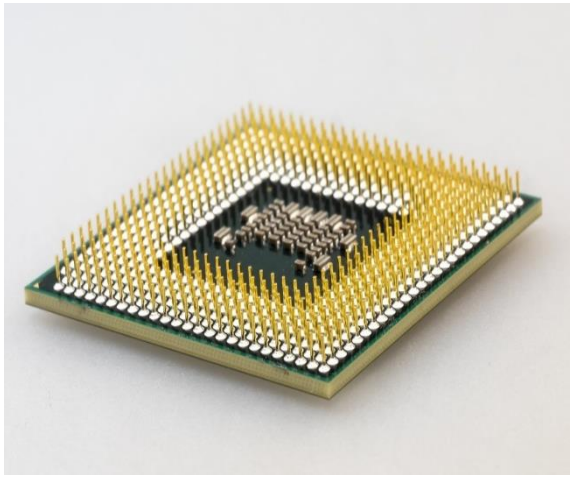
Aside: CPU Size

Early Computers' CPU



large units of electronics circuits that take up much space

Modern Computers' CPU



integrated circuit-based,
i.e., part of the microprocessor

Basic CPU Functional Units

CPU internal consists of the following:

1 Arithmetic and Logic Unit (ALU)

- deals with arithmetic operations like addition, subtraction, multiplication and division (if supported)
- also performs logic operations (i.e. Boolean operations)

AND, OR, NOT, XOR etc., and bit shifting/ rotation

2 Control Unit (CU)

- consists of decoders and logic circuits
- controls the overall operations of the various units and modules
- driven by a clock signal to ensure that everything happens at the correct instances and in proper sequence

Basic CPU Functional Units (Cont'd)

3 Register Array

- a small amount of very high speed internal storage used for frequently accessed data
- enables data to be stored and retrieved quickly

Some of these registers are used for more specific functions.

- *e.g. program counter, instruction register, stack pointer, memory address register, accumulator, etc.*

More on this later..

Data, Address and Control signals

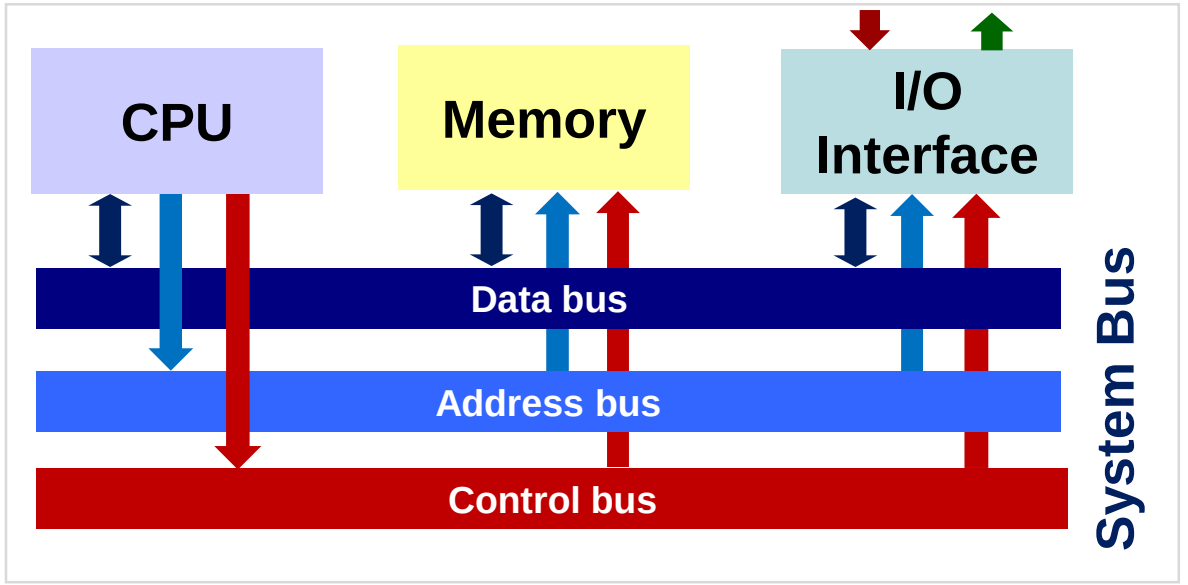
- comprises of signaling wires that are grouped into data signals, address signals, and control signals
- the signals connect the various internal functional units together
- extend to the external system bus for other modules (memory and I/O) of the µP

System Bus

The system bus consists of groups of parallel signals that are used to transfer information between the modules in the microprocessor.

Data Bus - conveys the information from one module to the other.

Control Bus - provides the control signals for the modules to work together, such as to determine the direction of data flow, and when each device can access the data bus and address bus.

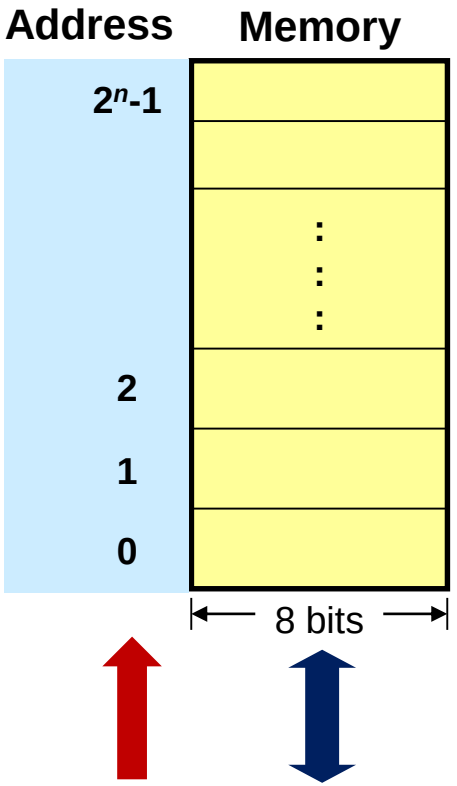


Address Bus used to convey the address information. The signals' pattern on the address bus lines determines the location of the source and destination of the data transfer.

Memory

Memory is used to store instructions and data for the CPU

- consists of high speed electronics components that store the information in binary bit format



Each location stores an 8-bit (byte) size data.

Each location is allocated a unique address.

- identified by specifying its binary pattern on the address bus

If data is more than 8 bits in size (e.g. a 32-bit word),

- consecutive locations are used
- the lowest byte address is used to access the word location

Memory Size

Memory size refers to the total number of locations that a memory module can support.

This is dependent on the number of bits (*n*) used in its address bus:

- **memory size = 2^n bytes**

When *n* = 10

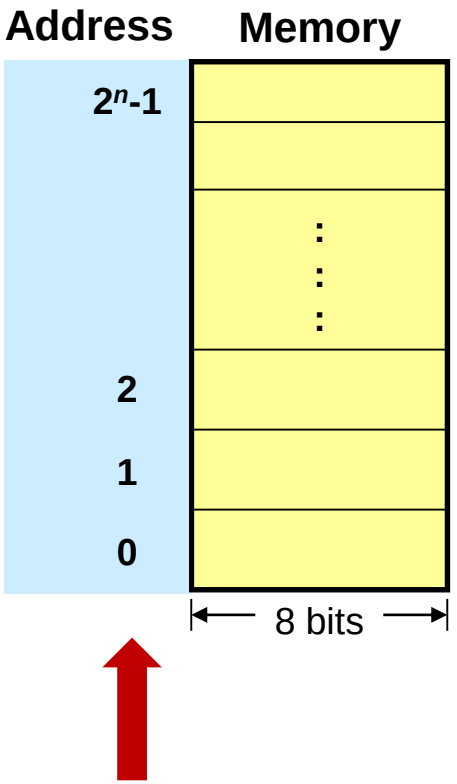
- **2^{10}** ≡ 1024 bytes.
- known as 1 KiloByte (KB)

When *n* = 20

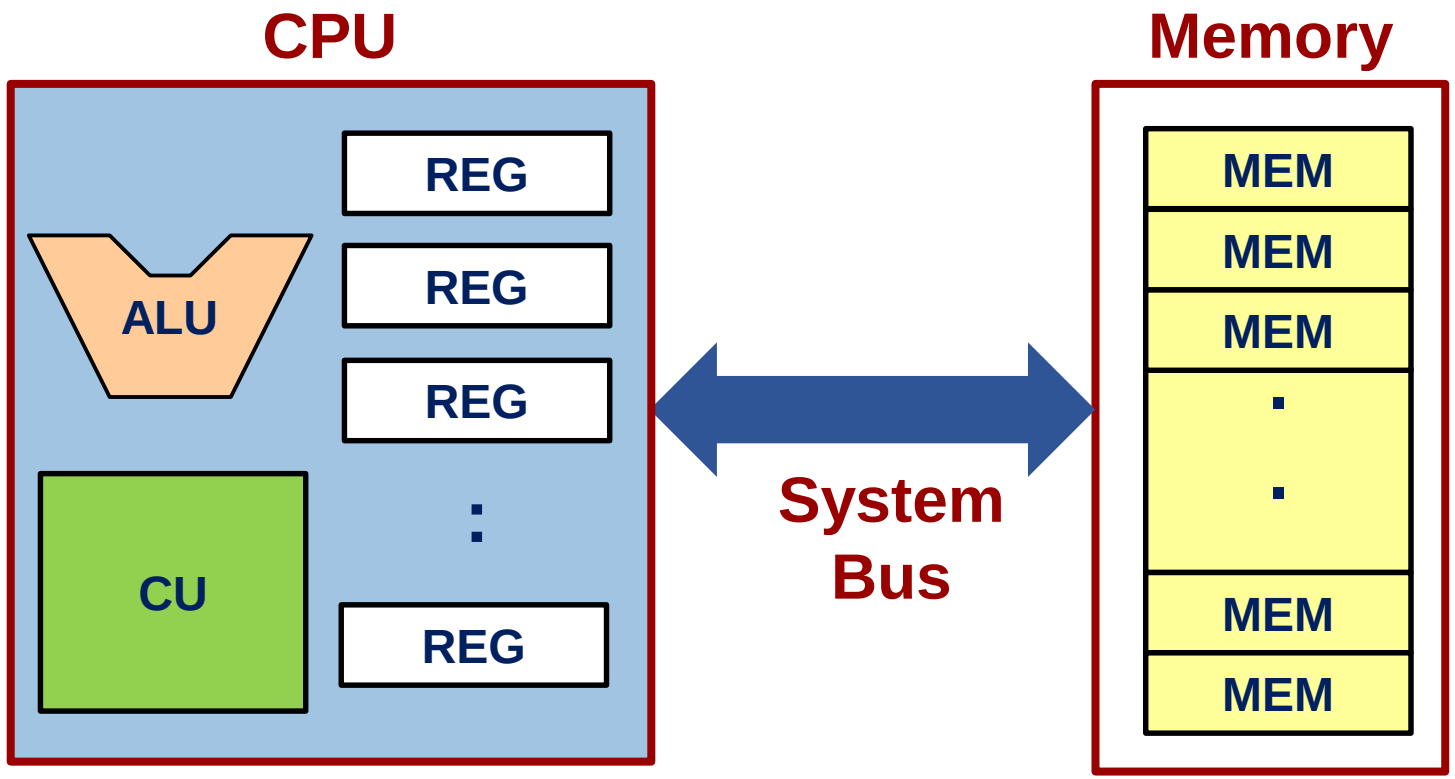
- **2^{20}** ≡ 1024 × 1KB = 1KB × 1KB is known as 1 MegaByte (MB)

When *n* = 30

- **2^{30}** ≡ 1KB × 1KB × 1KB is known as 1 GigaByte (GB)



CPU, Memory and System Bus



Program Execution

Outline

How Program Instructions are Executed

- *How various parts of a program are recognized*
- *How the CPU and memory work together when executing a program*
- *The role of the different functional units within the CPU module*

Program and Instructions

Operation of the computer is determined by a sequence of instructions that are given by the user

- collectively known as a **program**
- in more generic term – ***software***

  (as against the hardwired hardware circuitry that cannot be changed).

All instructions are encoded in binary format when loaded in the memory, together with the data.

Hence, the codes representing the instructions look just like those for numbers and data when stored in memory

- but they are interpreted as **commands** (rather than numeric data) when retrieved by the CPU.

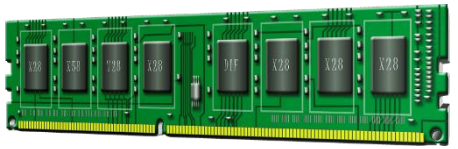
```
# 1. prompt user for the radius,
# 2. apply circumference&area formulae
# 3. print the results
import math
radiusString  = input("Enter the radius of your circle:")
radiusFloat = float(radiusString)
circumference = 2 * math.pi * radiusFloat
area = math.pi * radiusFloat * radiusFloat

print()       # print a line break
print( "The cirumference of your circle is:",circumference,\
            ", and the area is:" , area )
```

Aside: Instructions and Data

A program is typically permanently stored in a non-volatile storage device, such as hard disk or flash memory (e.g. SD Card).

- *first copy to the memory when it is ready to be executed*

SD Card


Random Access Memory


Adobe Flash

In addition to the instructions to be executed, a program will also need to have access to data that its instructions need to act upon.

- *e.g. Adobe Flash Player: a program that can be used to view a video clip stored as a data file*

When a program is loaded in the memory, the memory will hence contain the following information:

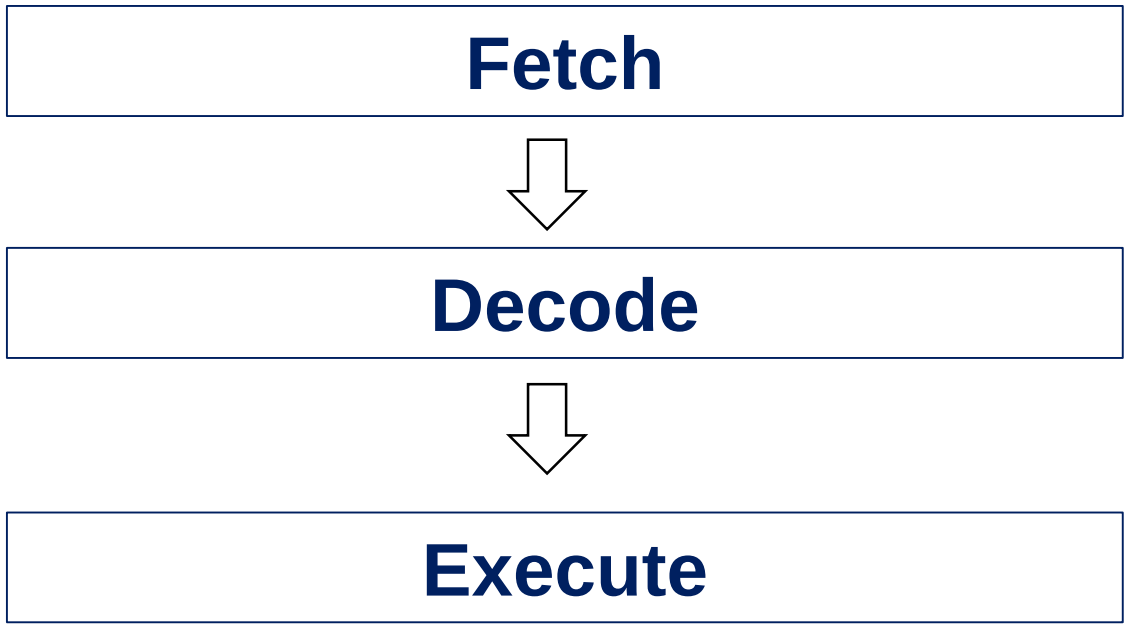
- **instructions** to be executed
- **data** needed by the relevant instructions

Program Execution

When a microprocessor executes a program

- it just executes the instructions in the given sequence.

Each instruction is in turn executed in a sequence of three basic steps:



Basic CPU Operations

Machine Instruction

When the microprocessor starts operations, instructions are fetched (i.e. retrieved) from memory successively into the CPU by the CPU's **Control Unit** for decoding.

Each instruction located in the memory is encoded in certain ***binary*** formats. (also written in hexadecimal format for ease of reading by human).

- *e.g. 01011101b (or 5Dh)*

This is known as the ***Machine Instruction***.

Hence, machine instructions stored in memory looks exactly like data.

So, how does the CPU know whether the content retrieved is an instruction or data (e.g. an unsigned integer)?

Instructions Executions

Valid machine instructions of a microprocessor family are encoded in specific binary bit patterns

- known as the instruction set of that microprocessor

CPU is designed to recognize its own instructions

- provided it fetches the instructions from the memory properly

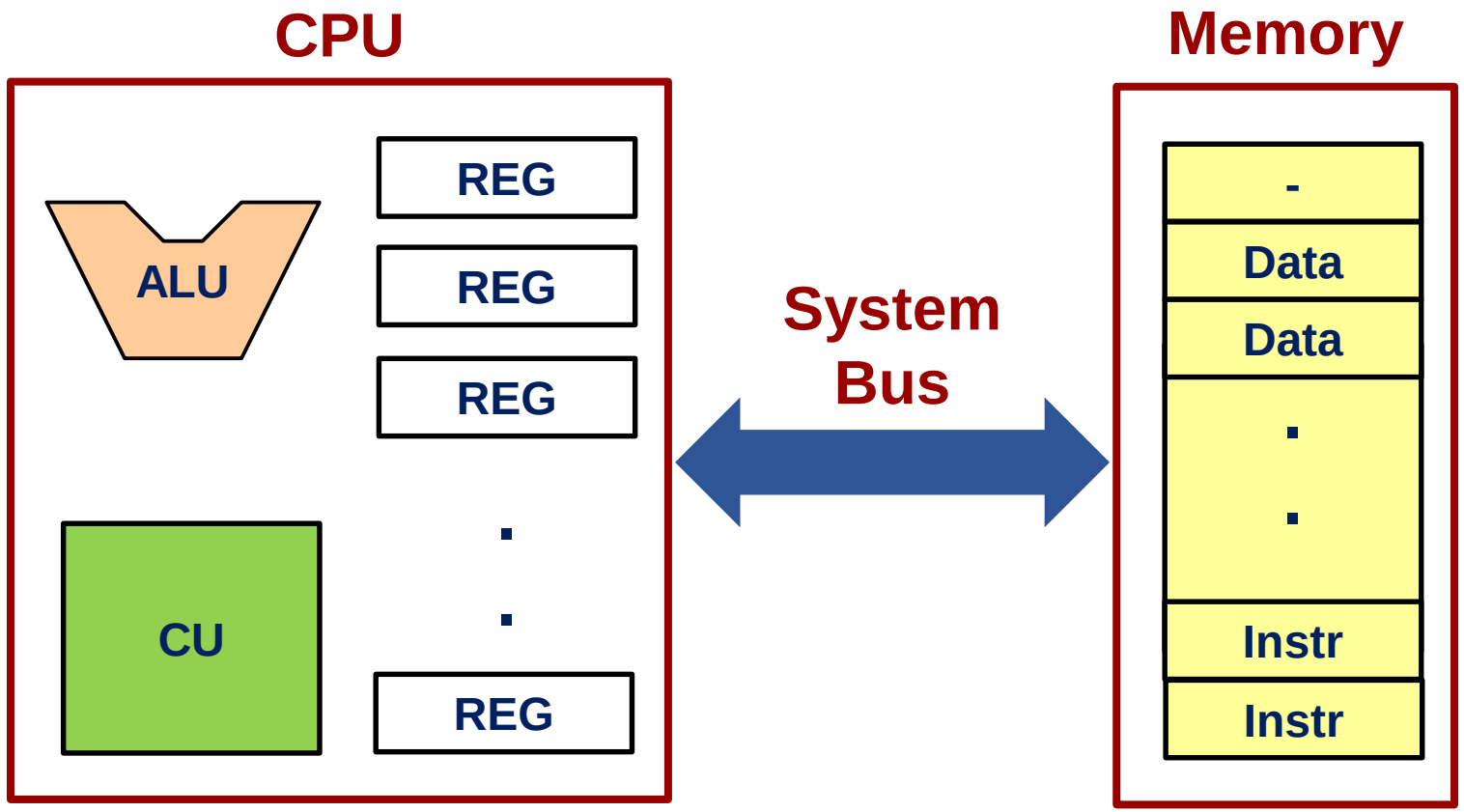
Once the CPU successfully decodes a valid instruction

- execution is then performed
  - such as applying an arithmetic operation on certain specified data

Stored-Program Model

To understand how the CPU works, the simple stored-program model will be used here

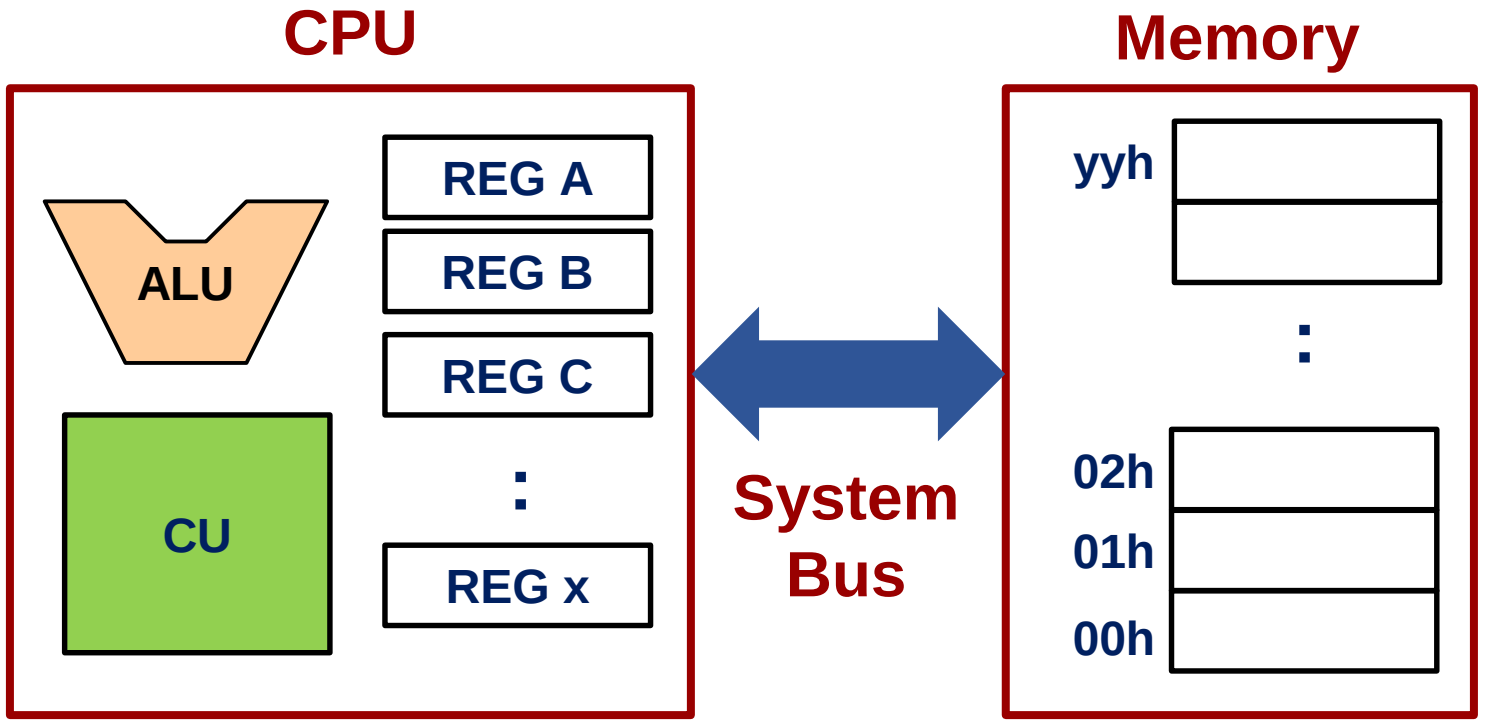
- where both instruction and data are stored in the memory



Registers and Memory

A '**Register**' is a very fast storage located within the CPU.

- Individual register within the CPU is identified by a label such as '**A**', '**B**', …; or '**R0**', '**R1**', …
- **Memory** location is identified by its (**hexadecimal**) address: '**00h**', '**01h**', …, '**38h**', …, etc.



Basic CPU Operation

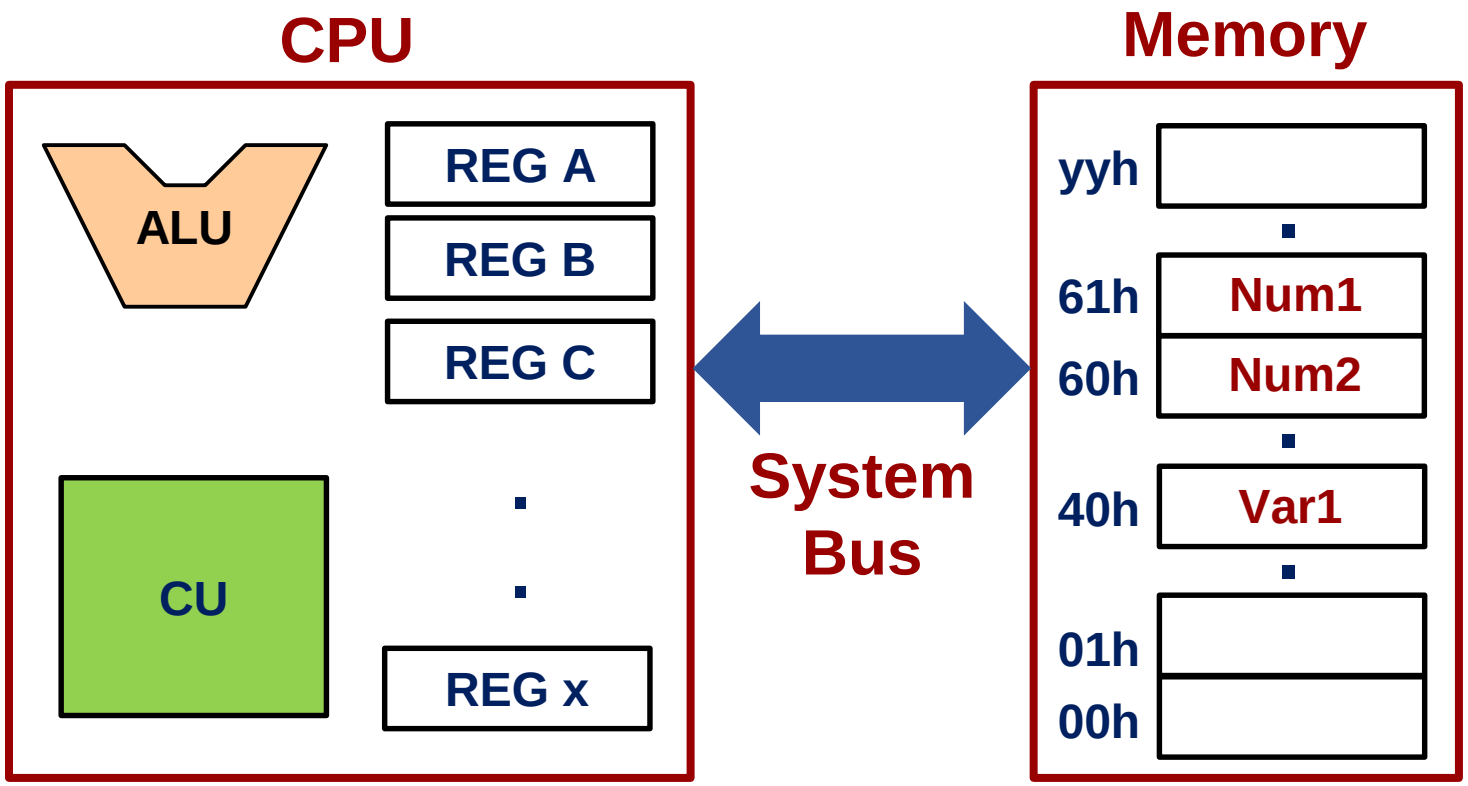
Example

Addition Instruction:

Var1 = Num1 + Num2

Assumptions:

- Num1 is stored in memory location 61h.
- Num2 is stored in memory location 60h.
- Var1 is stored in memory location 40h.



Basic CPU Operation (Cont'd)

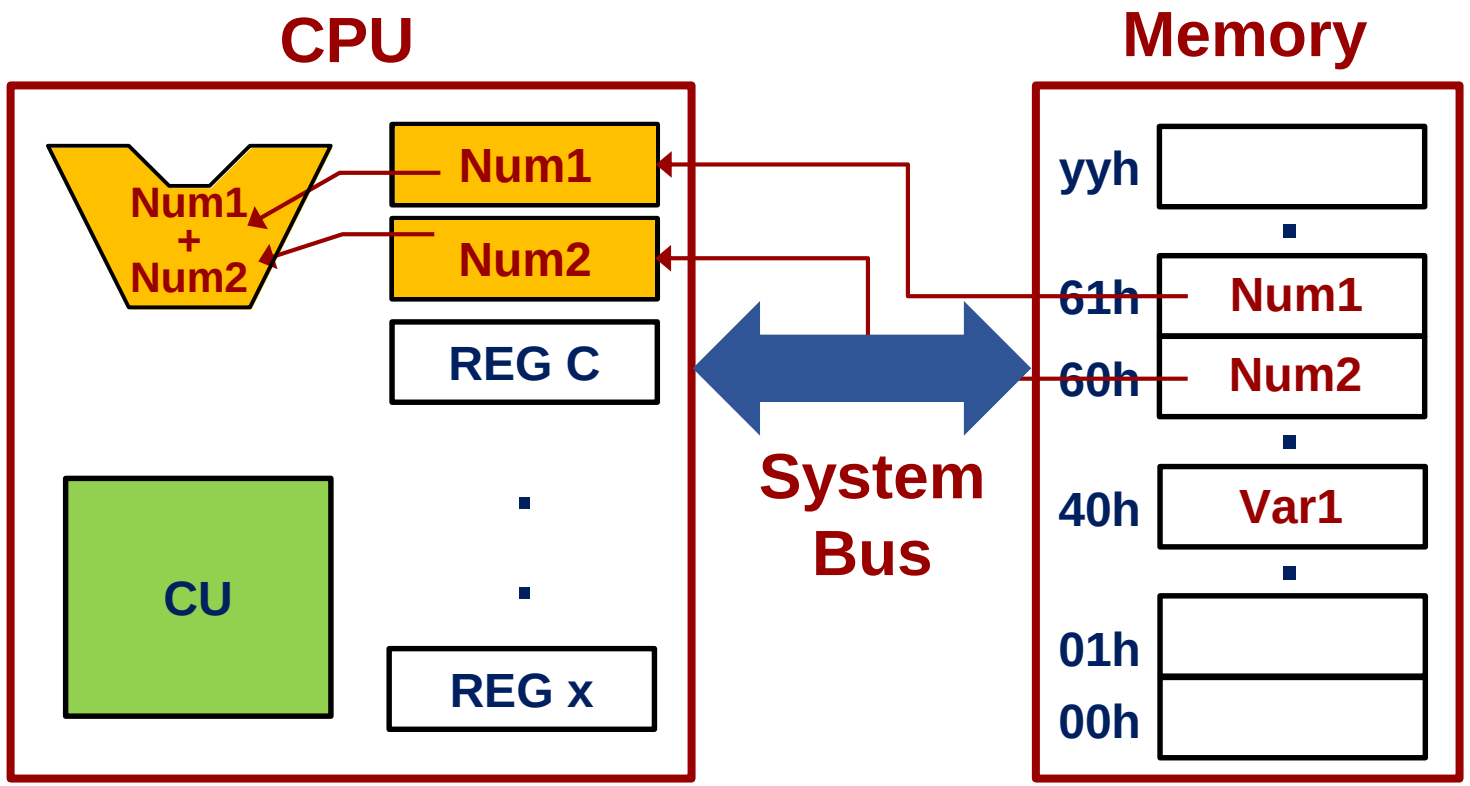
Example

Addition Instruction:

Var1 = Num1 + Num2

Execution: CU carries out the addition operation

- Num1 is copied to (**loaded** into) Register A
- Num2 is copied to Register B
- ALU executes the Addition operation: REG A + REG B



Basic CPU Operation (Cont'd)

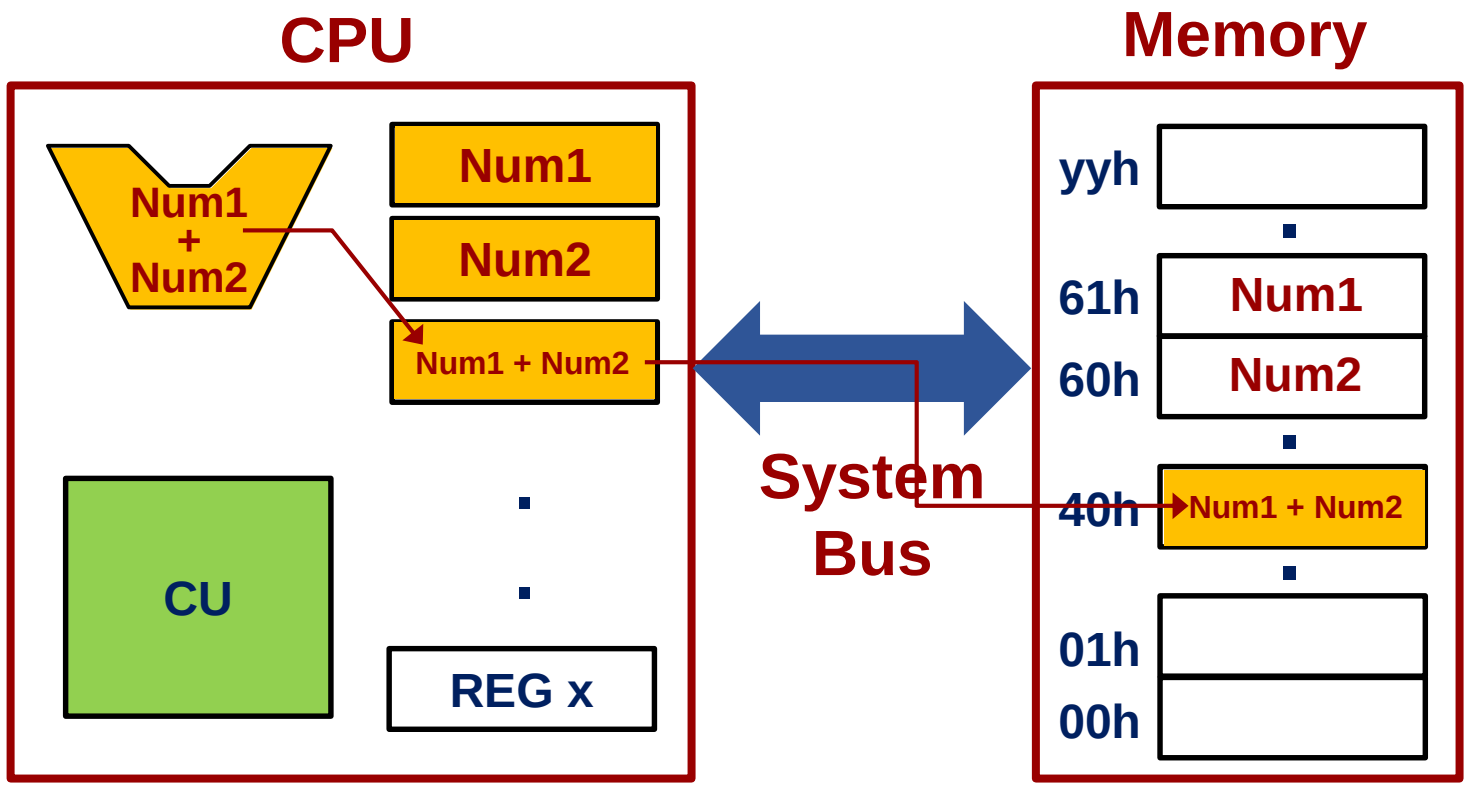
Example

Addition Instruction:

Var1 = Num1 + Num2

Execution: CU carries out the addition operation

- Result of ALU operation is moved into Register C.
- Content of Register C is copied (**stored**) into memory location 40h.



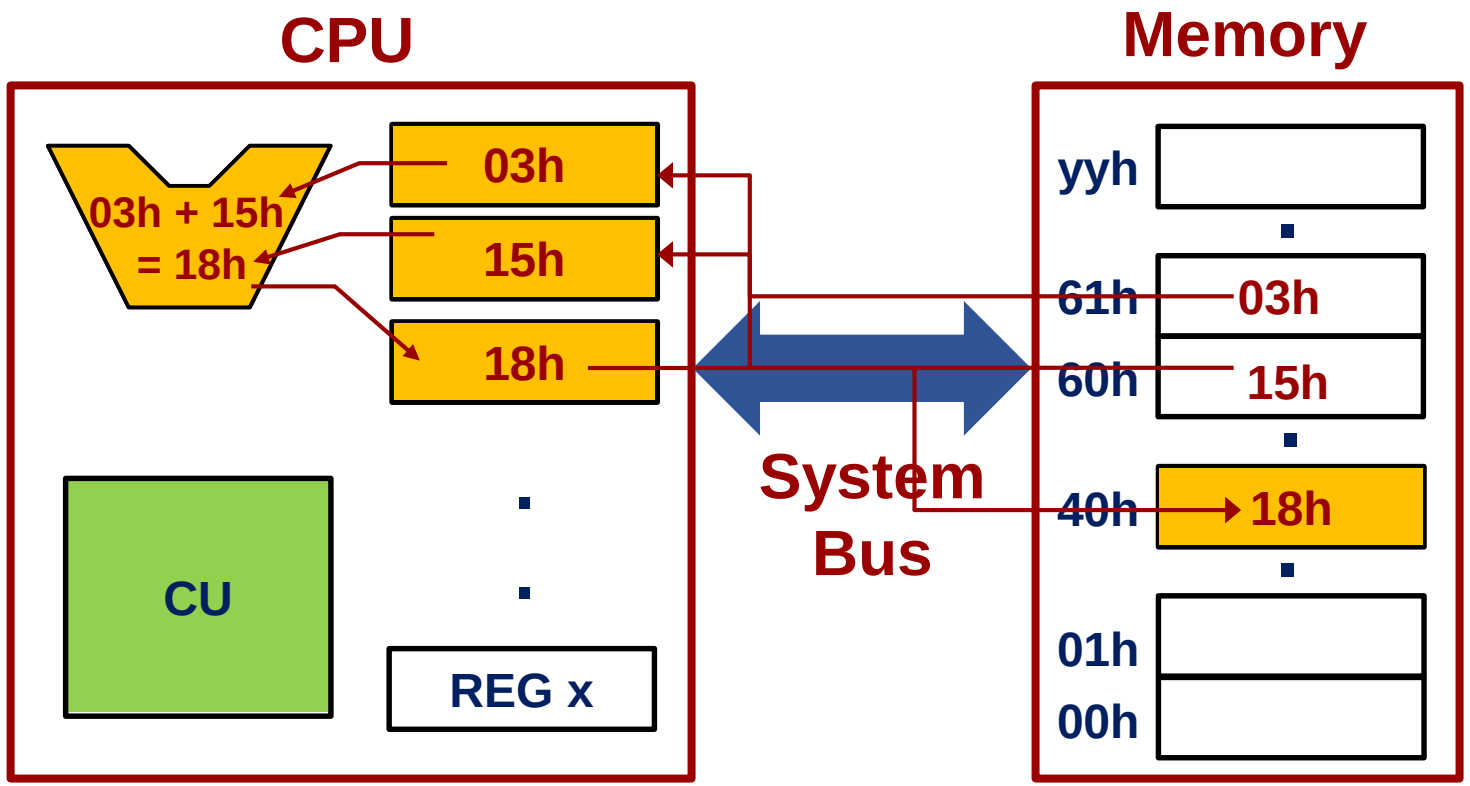
Basic Operation Example

Example

Let Num1 value = 03h,

Num2 value = 15h,

and Var1 = xxh
(i.e. it doesn't matter)



Instruction Execution

On power up, the program instructions will be typically first loaded into certain default memory locations (together with data).

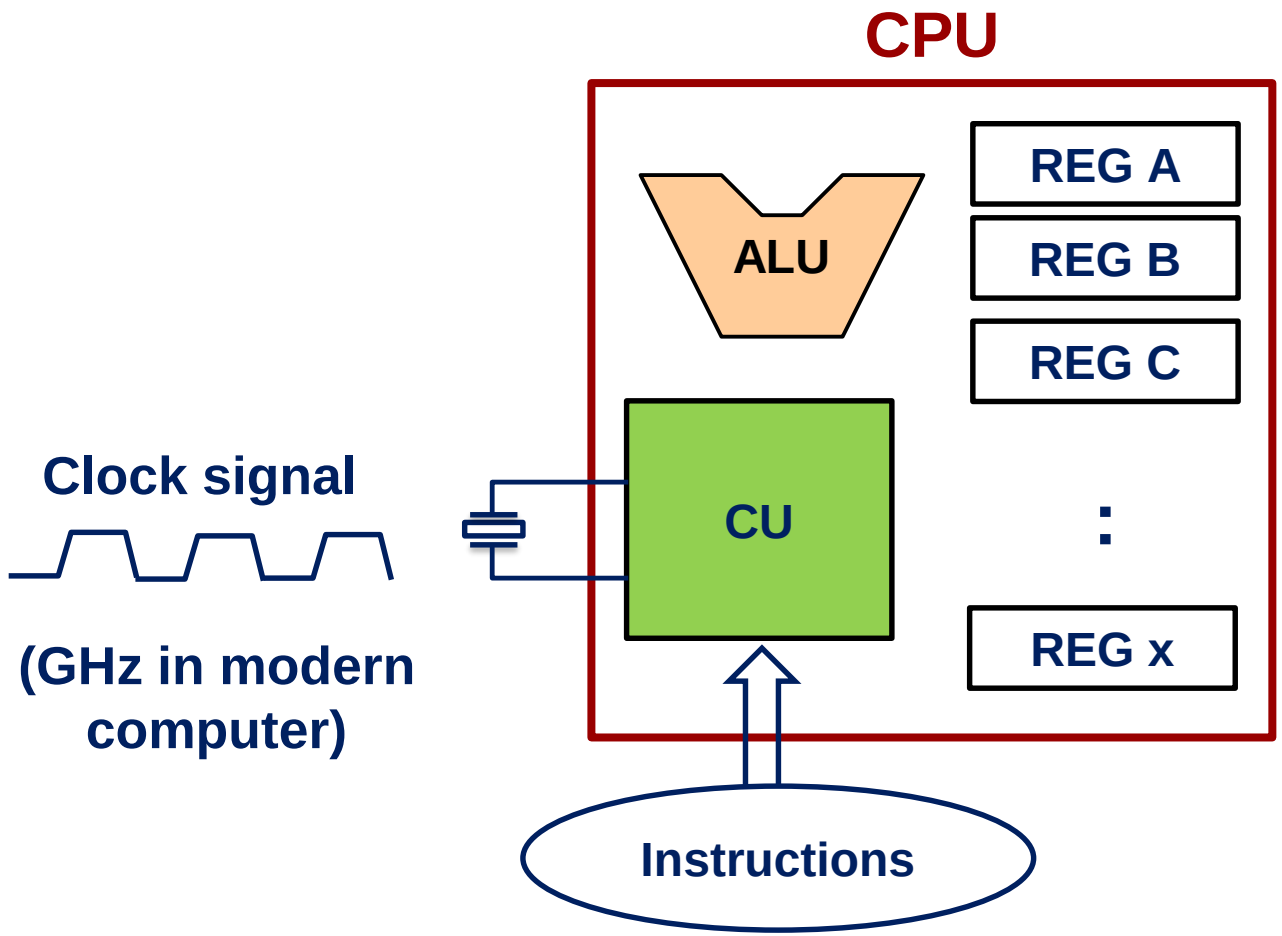
A **clocking signal** is also applied to the CPU (as well as to the rest of the microprocessor system).

Based on the rising/ falling edges of the clock, the CU will retrieve (fetch) the instruction from he default memory location.

The **CU of a CPU is designed to recognize its own instructions** and performs the corresponding operation coded.

How does the CU know where to find the instructions?

Control Unit (CU) Operations



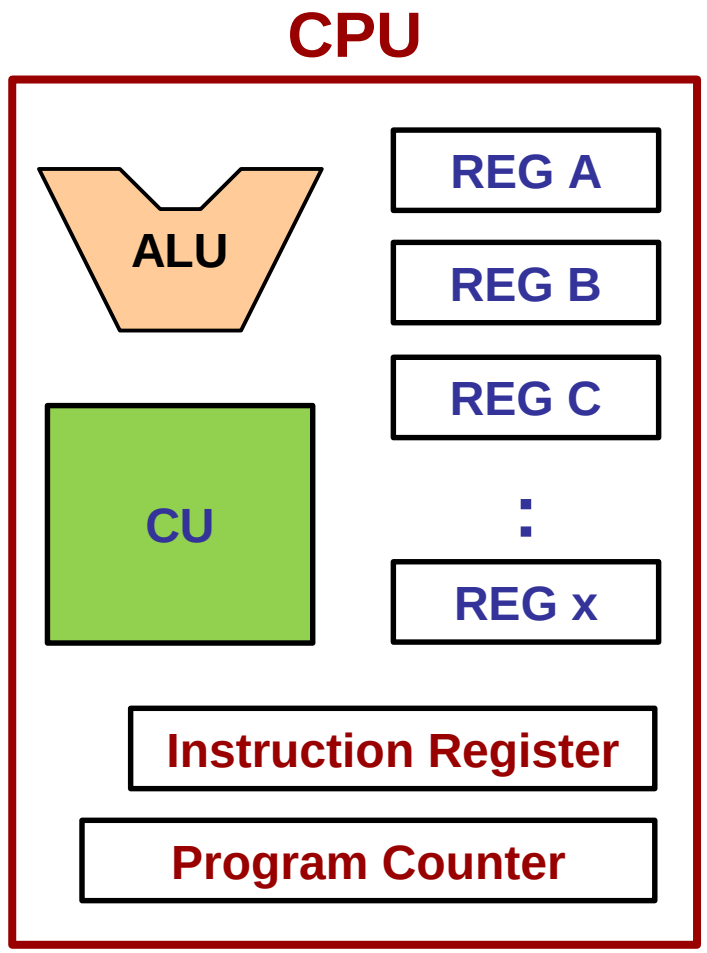
To be able to fetch and execute the instruction, several other registers will be needed.

PC and IR

CPU contains another two special function registers:

- **Program Counter (PC)**
  - tells CU where to find the instruction in the memory
- **Instruction Register (IR)**
  - holds the copy of the instruction to be decoded and executed by the CU

Practical CPU will contain several other registers, e.g., stack pointer, status register etc.

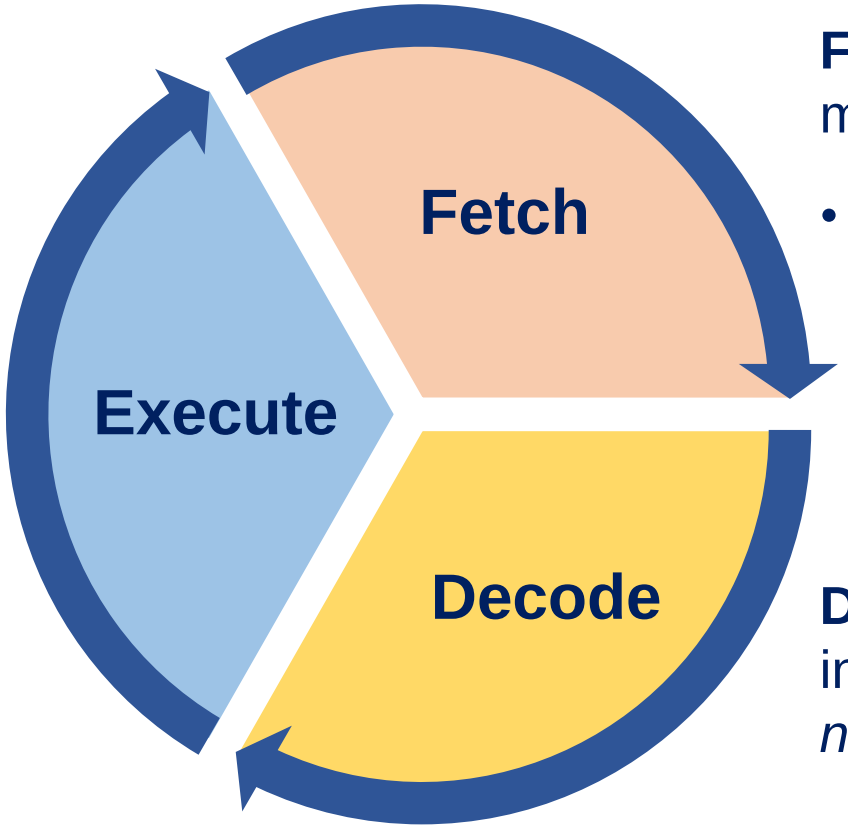


Fetch-Decode-Execute Cycle

Each operation cycle of the stored-program CPU typically involves three (3) basic steps:

Fetch the instruction from the memory into *IR*

- *using the address indicated by the PC*



Decode the machine instruction (by CU), *which is now stored in IR*

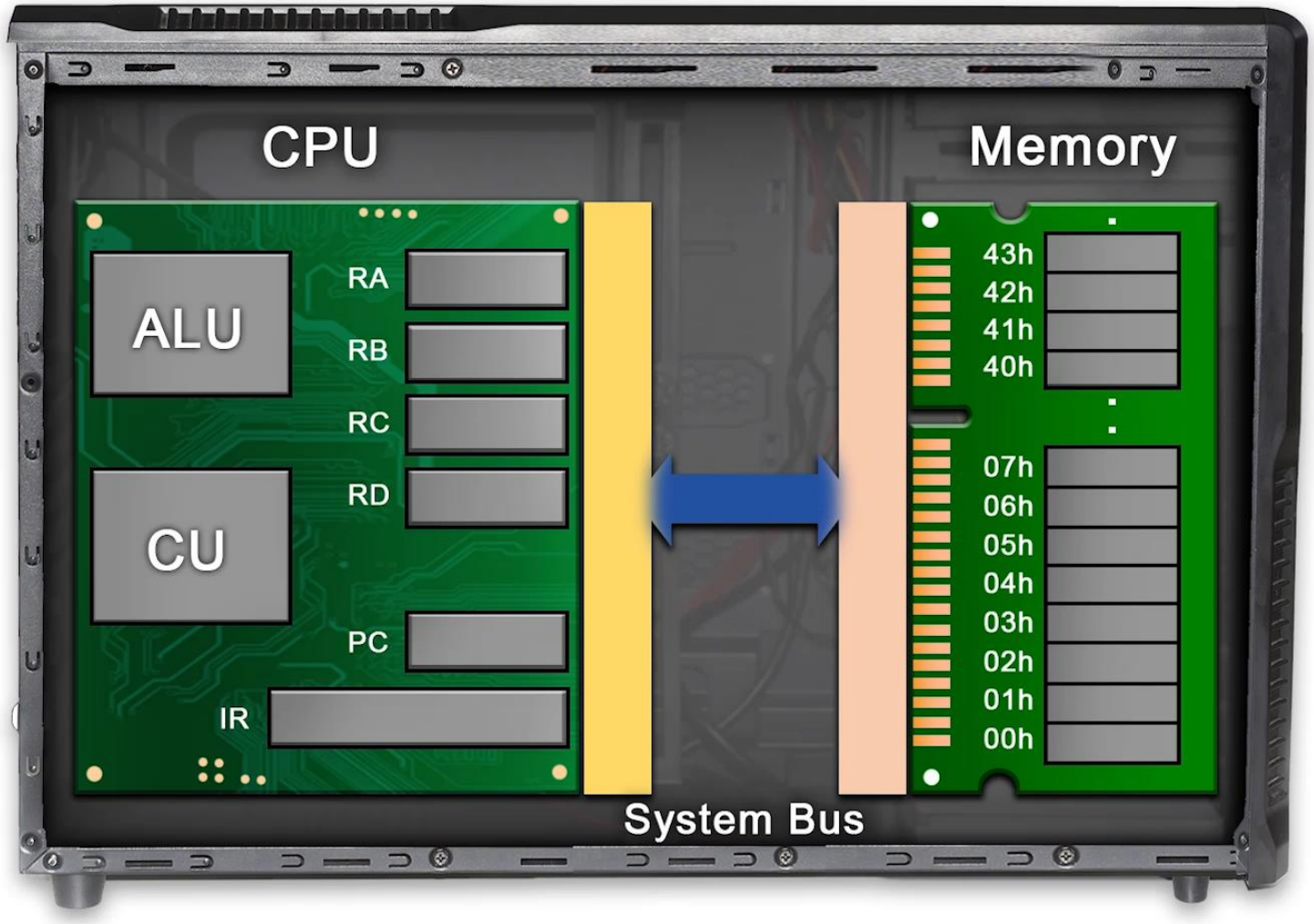
Execute the instruction

- *E.g., load the operands into the ALU and get the ALU to operate on them*

An Illustrative Example

After power-on,

- **memory is loaded with program and data**
- **Program Counter's content is set to '00h' (by design)**



An Illustrative Example (Cont'd)

Instruction Set

- The set of instructions understood by a CPU is specific to that particular CPU, which is determined by the designer of the CPU.
- Hence each CPU has its own "language", which is known as its instruction set.
- **Examples:**
  - *Typical PC and notebooks use the x86 instruction set.*
  - *Most tablets and smart phones use the ARM instruction set.*

Assumption for the Illustrative CPU Example:

"3A" is the code that means "Load Register A from memory".

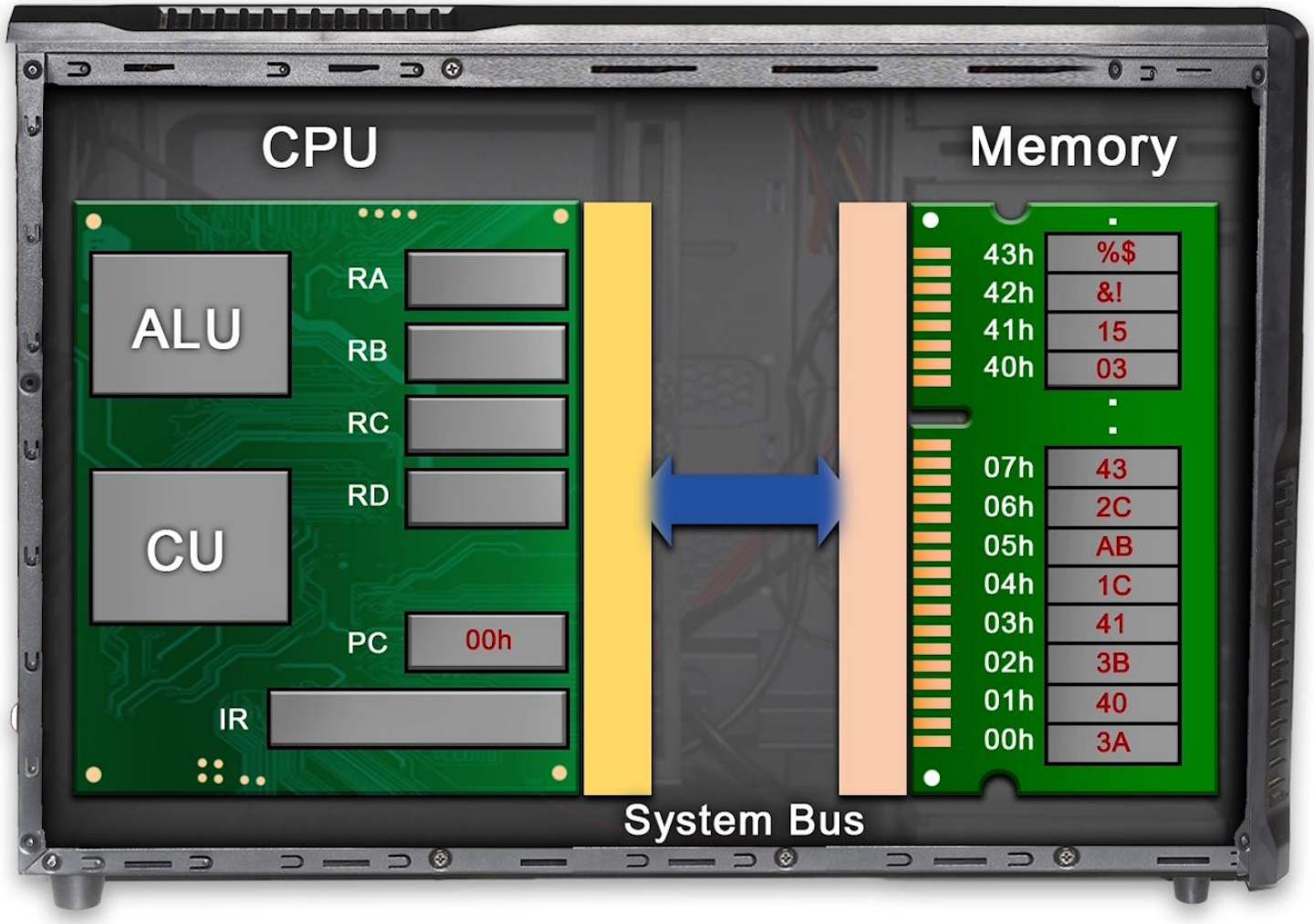
Note: This will have a totally different meaning for another CPU family (E.g. It could be interpreted as "multiply").

An Illustrative Example (Cont'd)

Back to Example

After power-on,

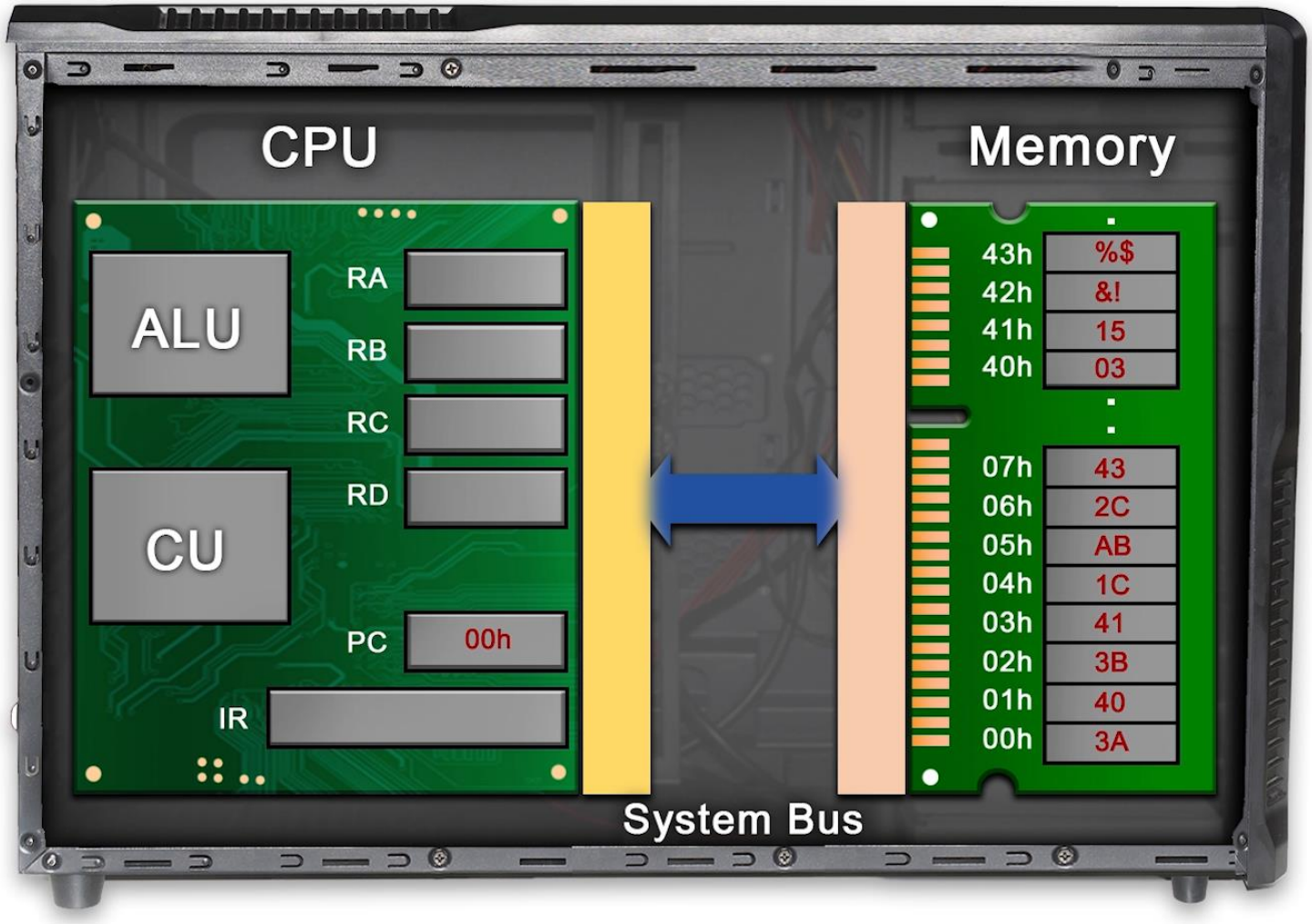
- **memory is loaded with the program and data**
- **PC content is set to '00h' (by design)**



An Illustrative Example (Cont'd)

Fetch 1

- Content of location 00h and 01h are loaded (or fetched) into IR (assume 16-bit size instruction).
- PC then automatically increments to '02h' (by design).

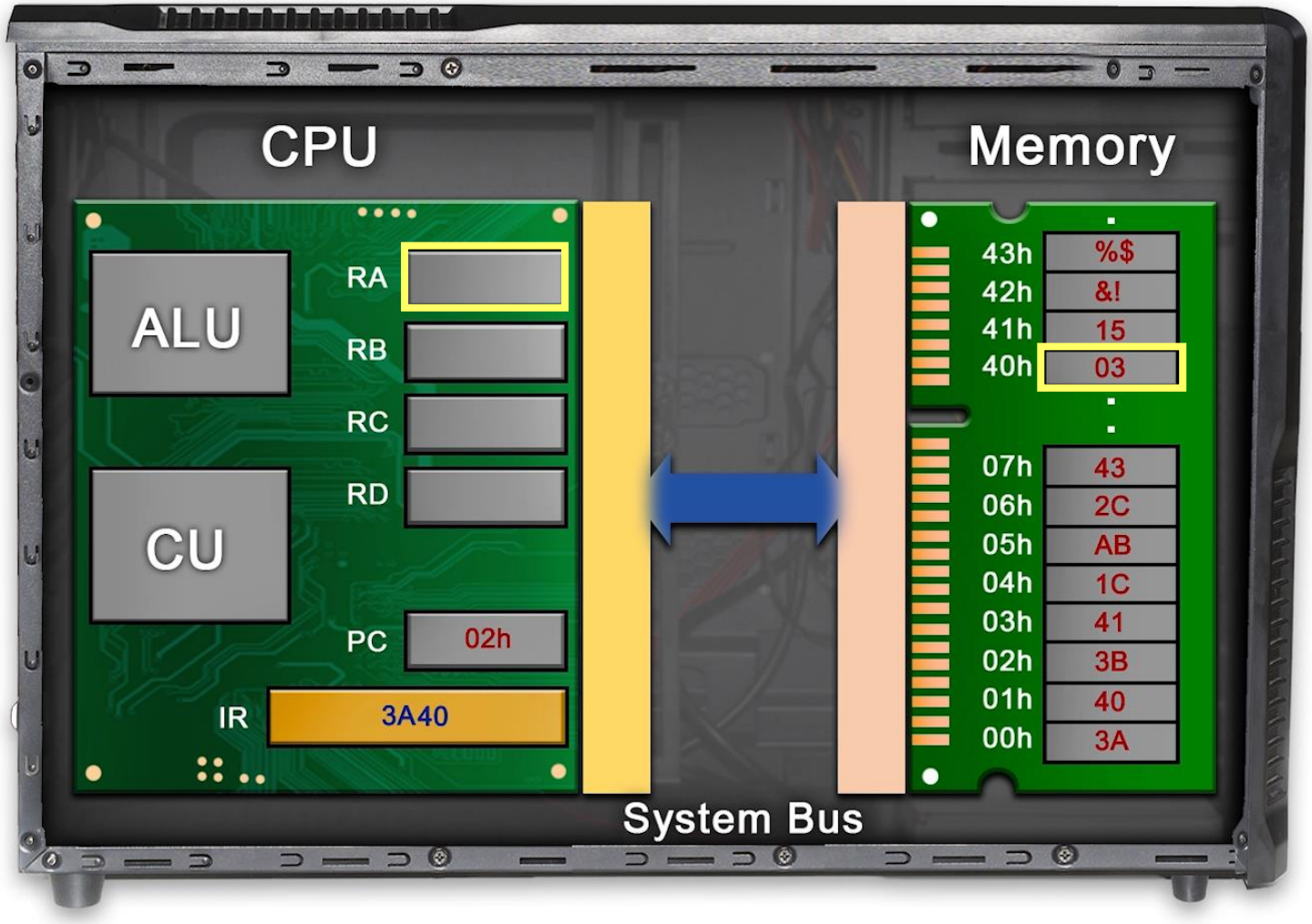


An Illustrative Example (Cont'd)

Decode 1

CU decodes the content IR:

- '3' and 'A' ⇒ Load register A (for this specific example)
- '40' ⇒ with content of memory location 40h

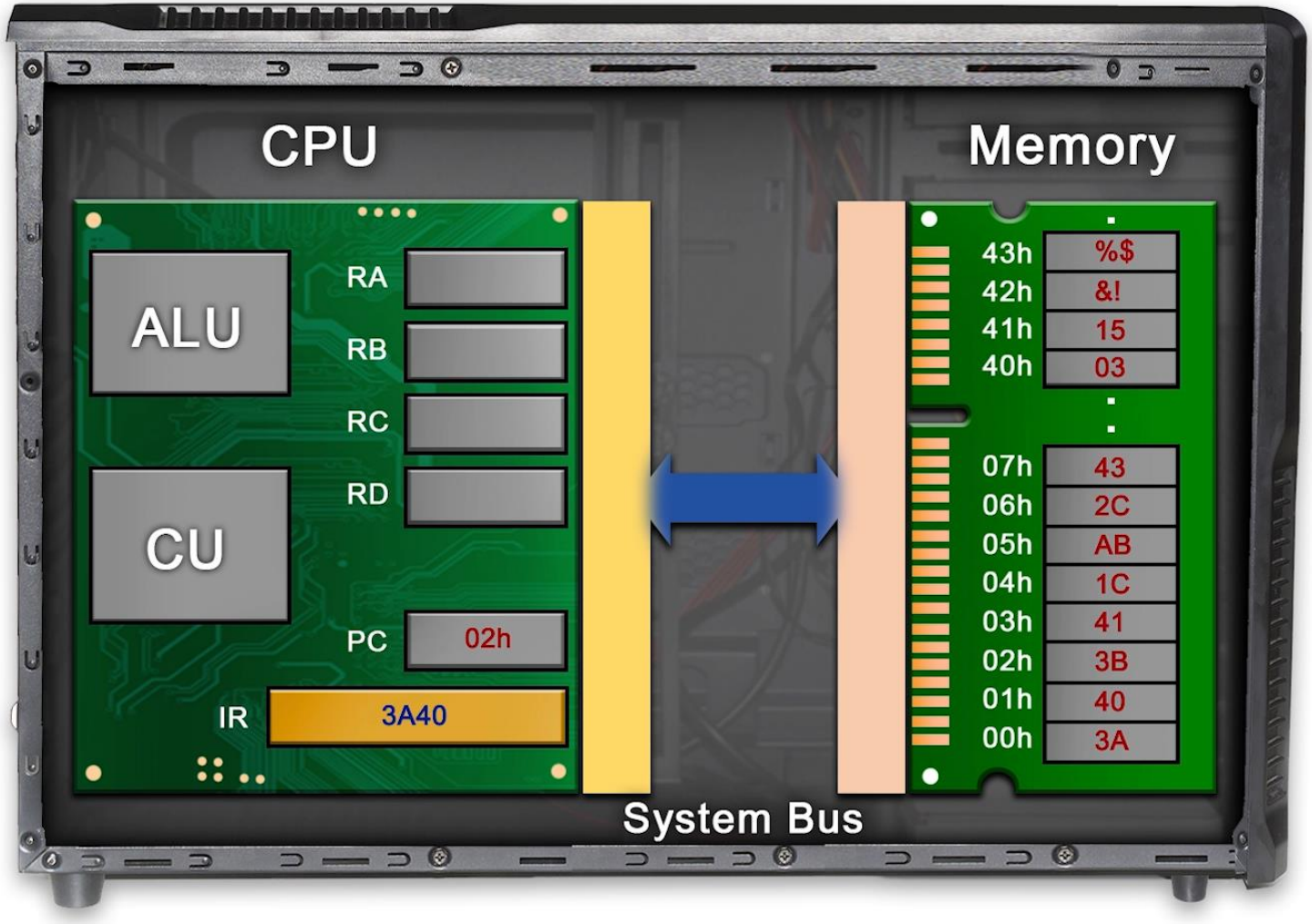


An Illustrative Example (Cont'd)

Execute 1

CU executes the instruction:

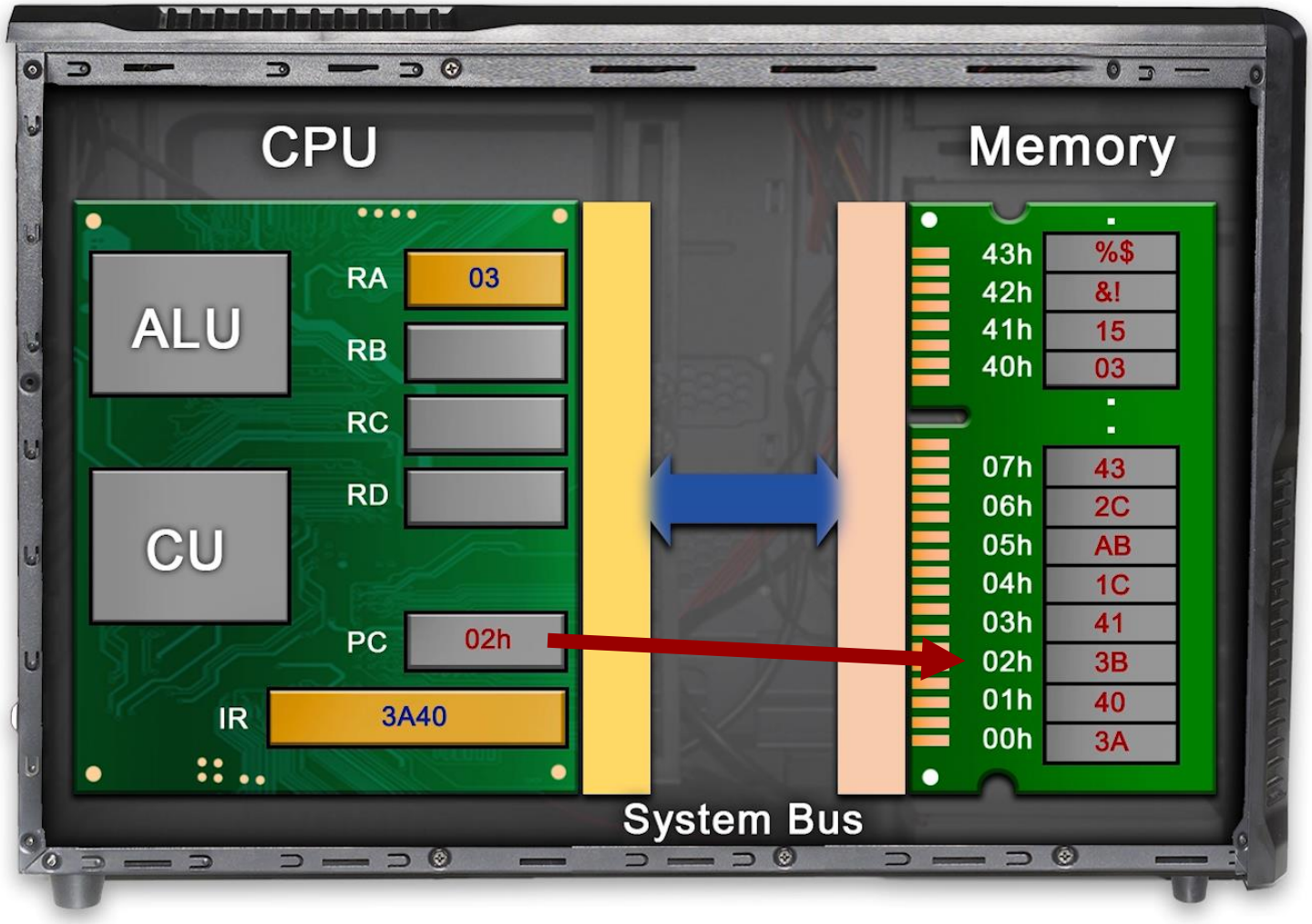
- Load register A with content of memory location 40h.



An Illustrative Example (Cont'd)

Fetch 2

- Content location 02h and 03h are loaded into IR.
- PC increment to '04h'.

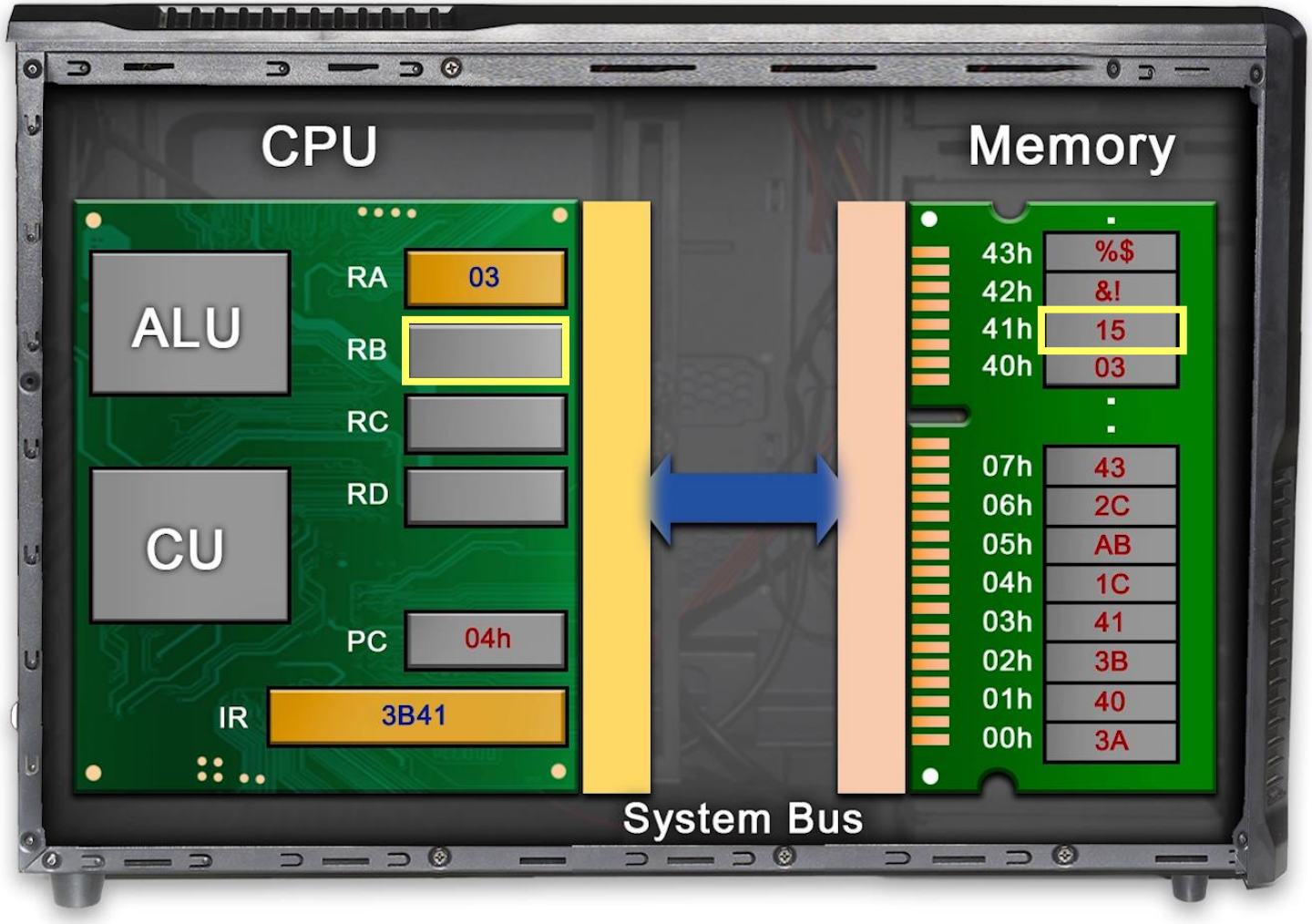


An Illustrative Example (Cont'd)

Decode 2

CU decodes the content IR:

- '3' and 'B' ⇒ Load register B (for this specific example)
- '41' ⇒ with content of memory location 41h

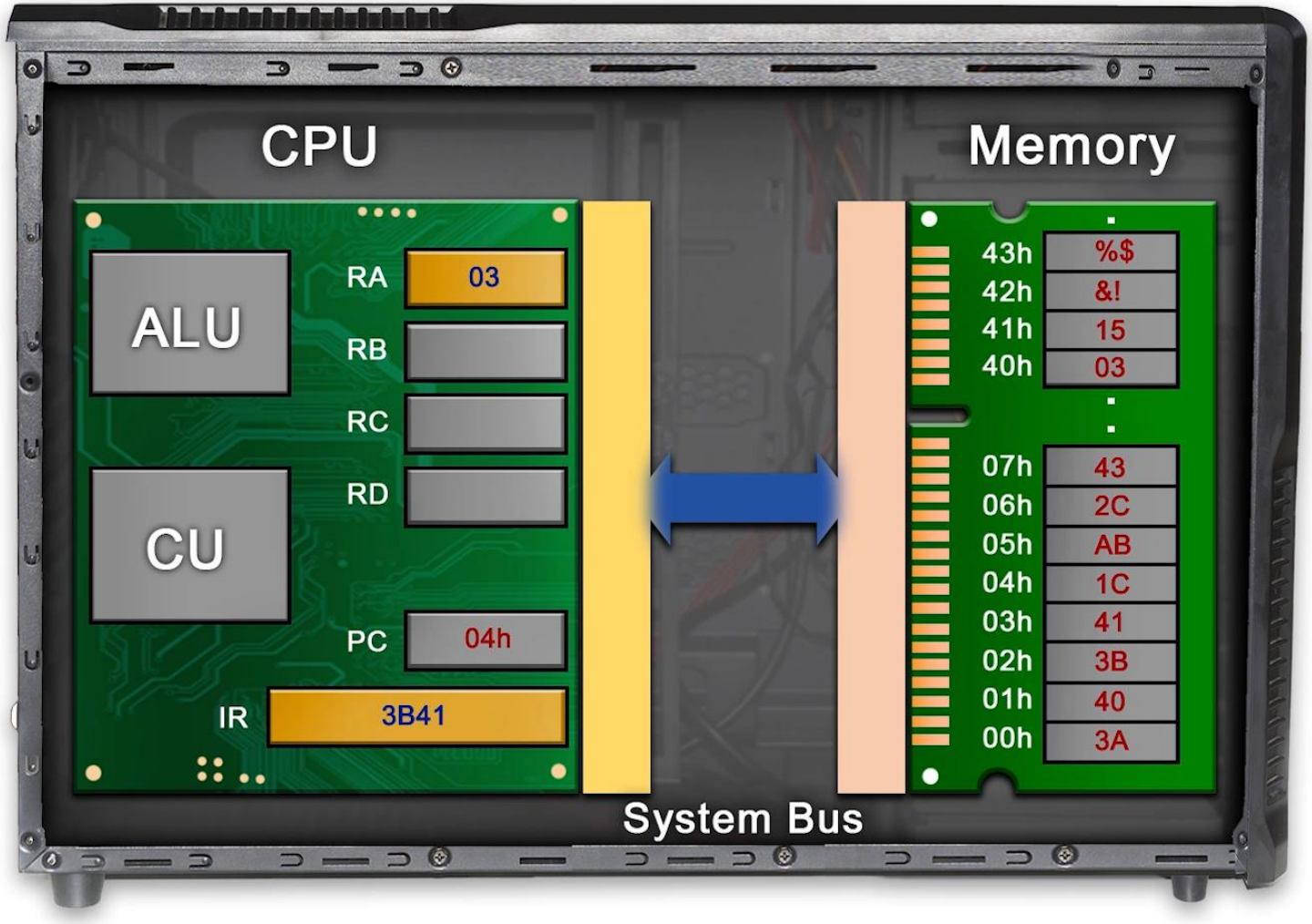


An Illustrative Example (Cont'd)

Execute 2

CU executes the instruction:

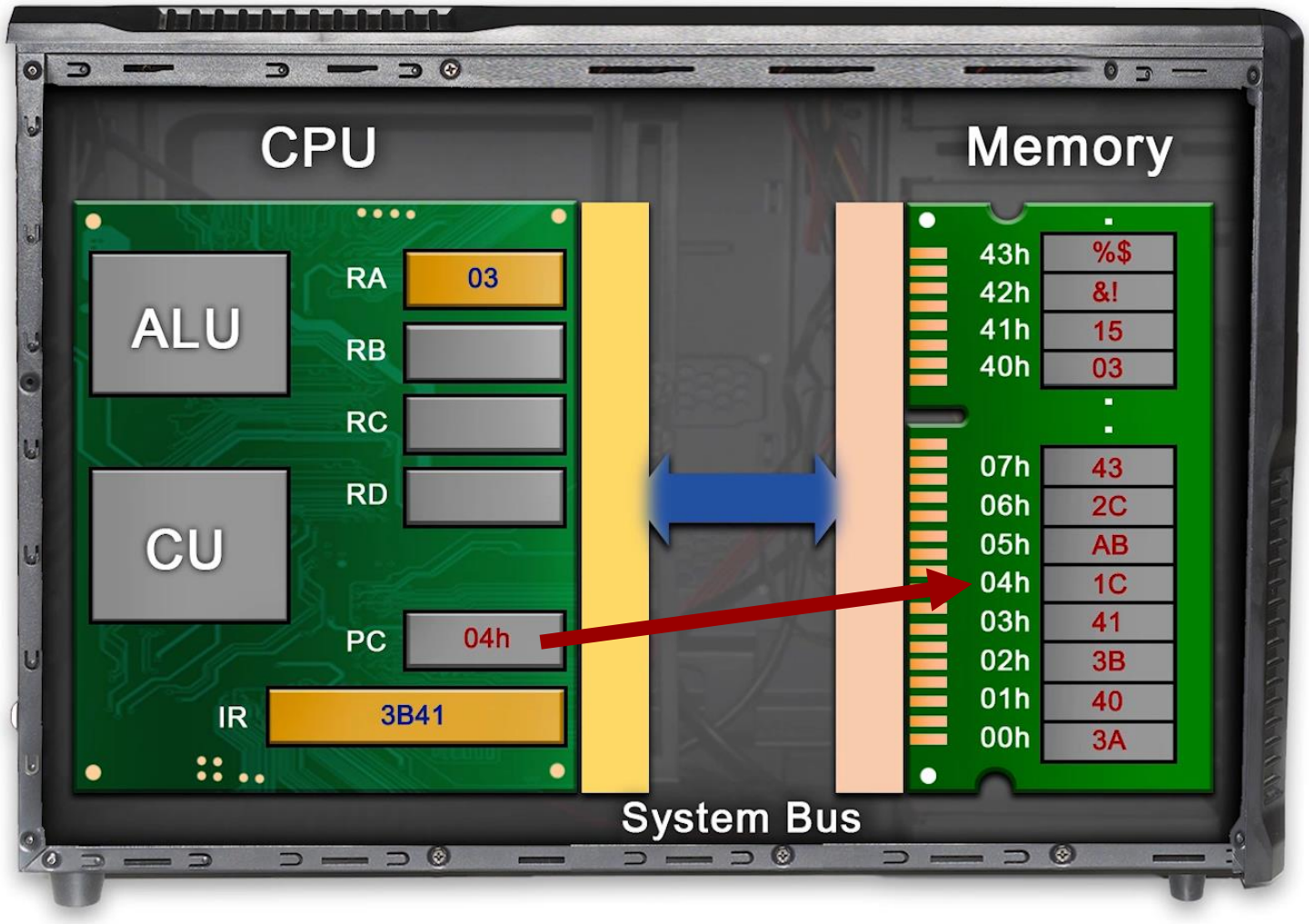
- Load register B with content of memory location 41h.



An Illustrative Example (Cont'd)

Fetch 3

- Content of location 04h and 05h are loaded to IR.
- PC increments to '06h'.

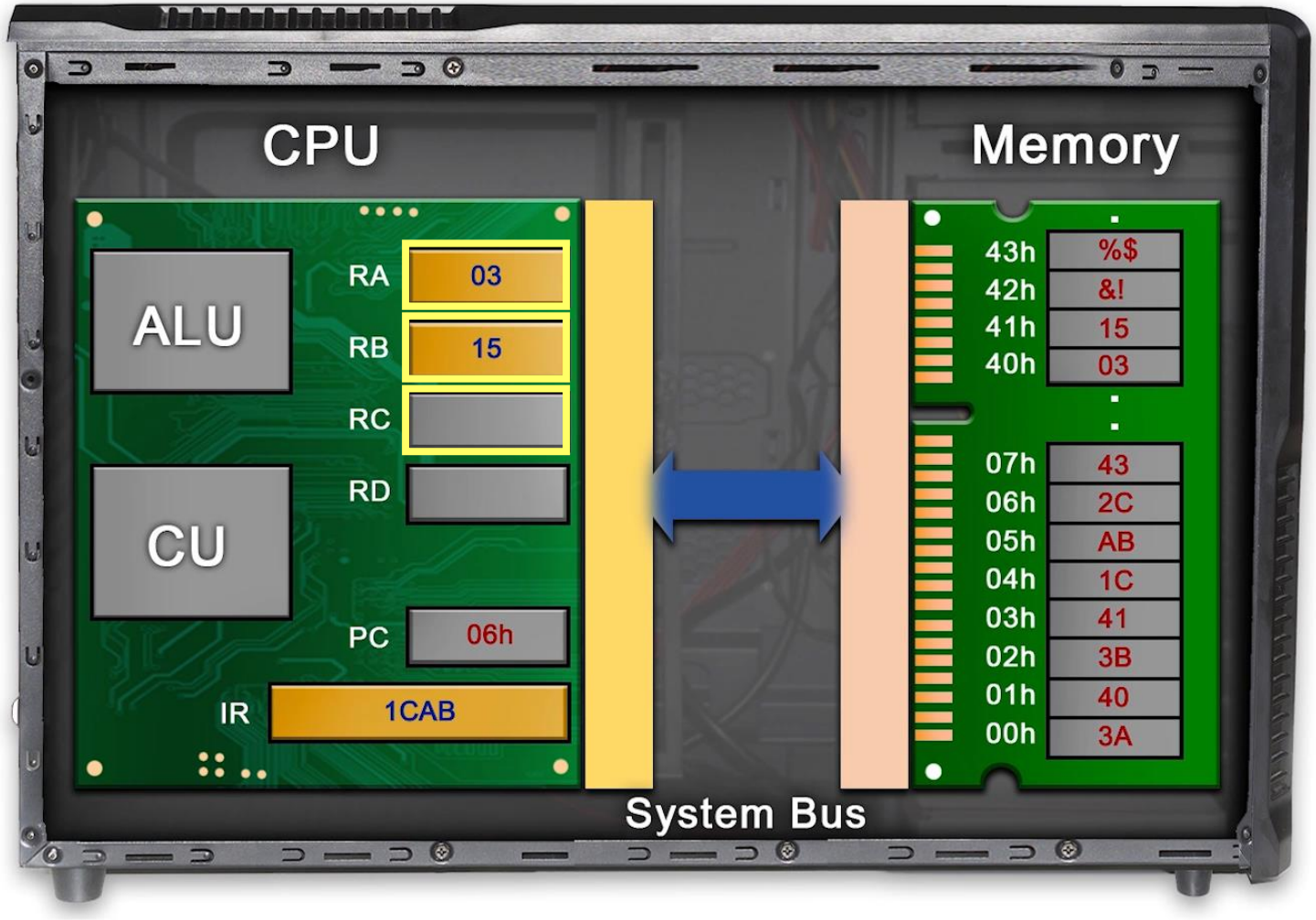


An Illustrative Example (Cont'd)

Decode 3

CU decodes the content IR:

- '1C' ⇒ add and save result in Register C
- 'AB' ⇒ Register A and Register B

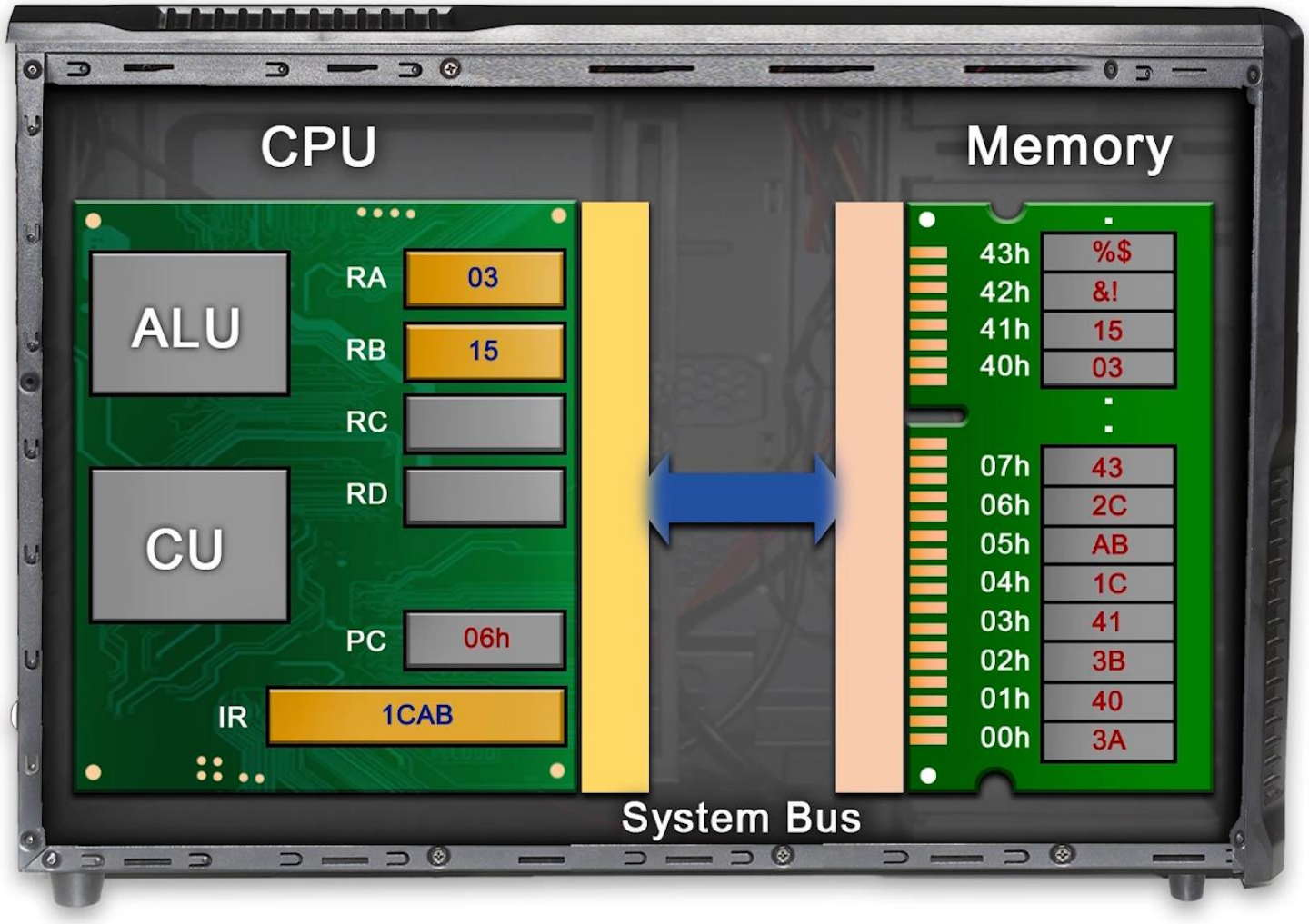


An Illustrative Example (Cont'd)

Execute 3

CU executes the instruction:

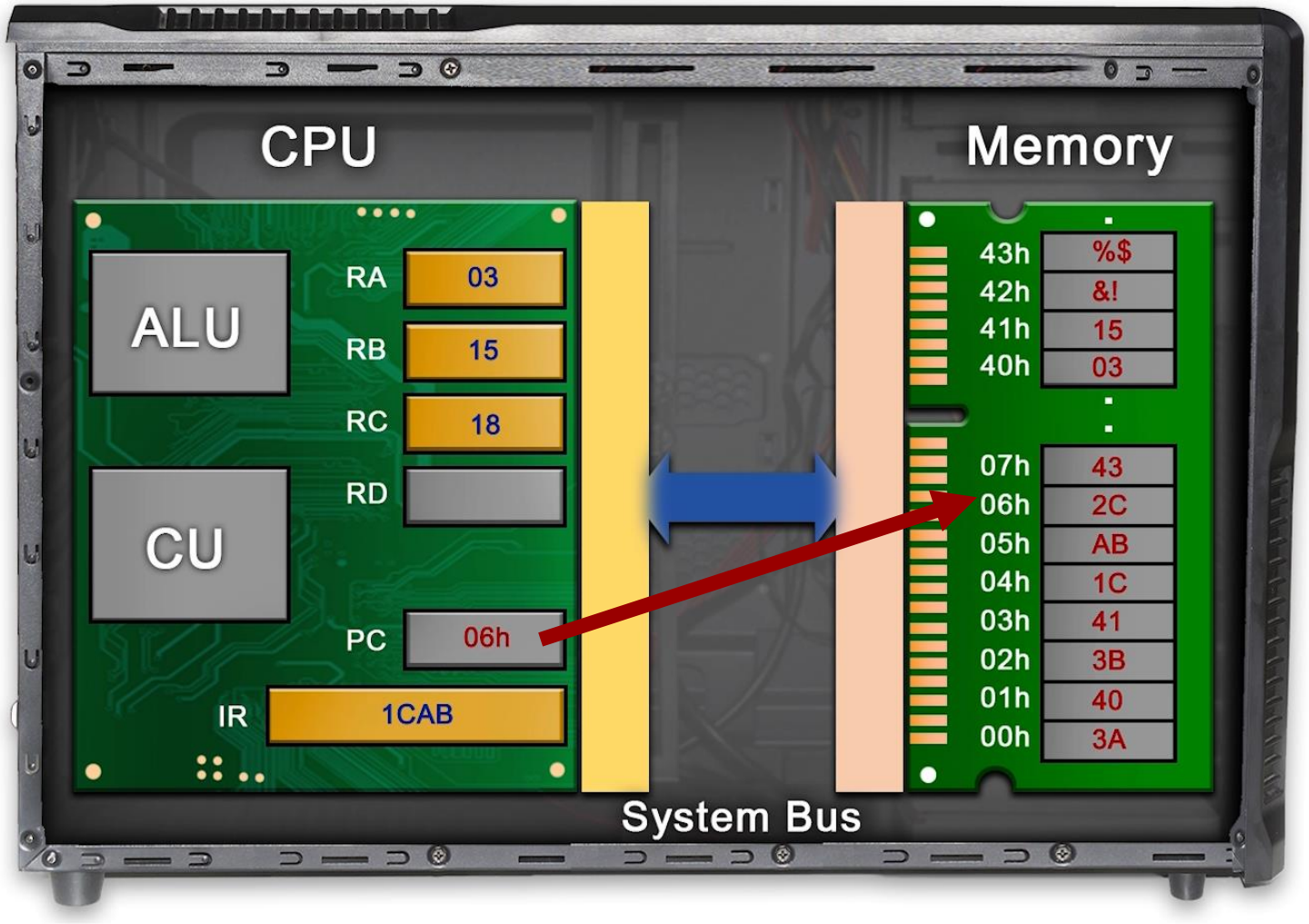
- Use ALU to add Register A and B, and save the result in Register C.



An Illustrative Example (Cont'd)

Fetch 4

- Content of location 06h and 07h are loaded to IR.
- PC increments to '08h'.

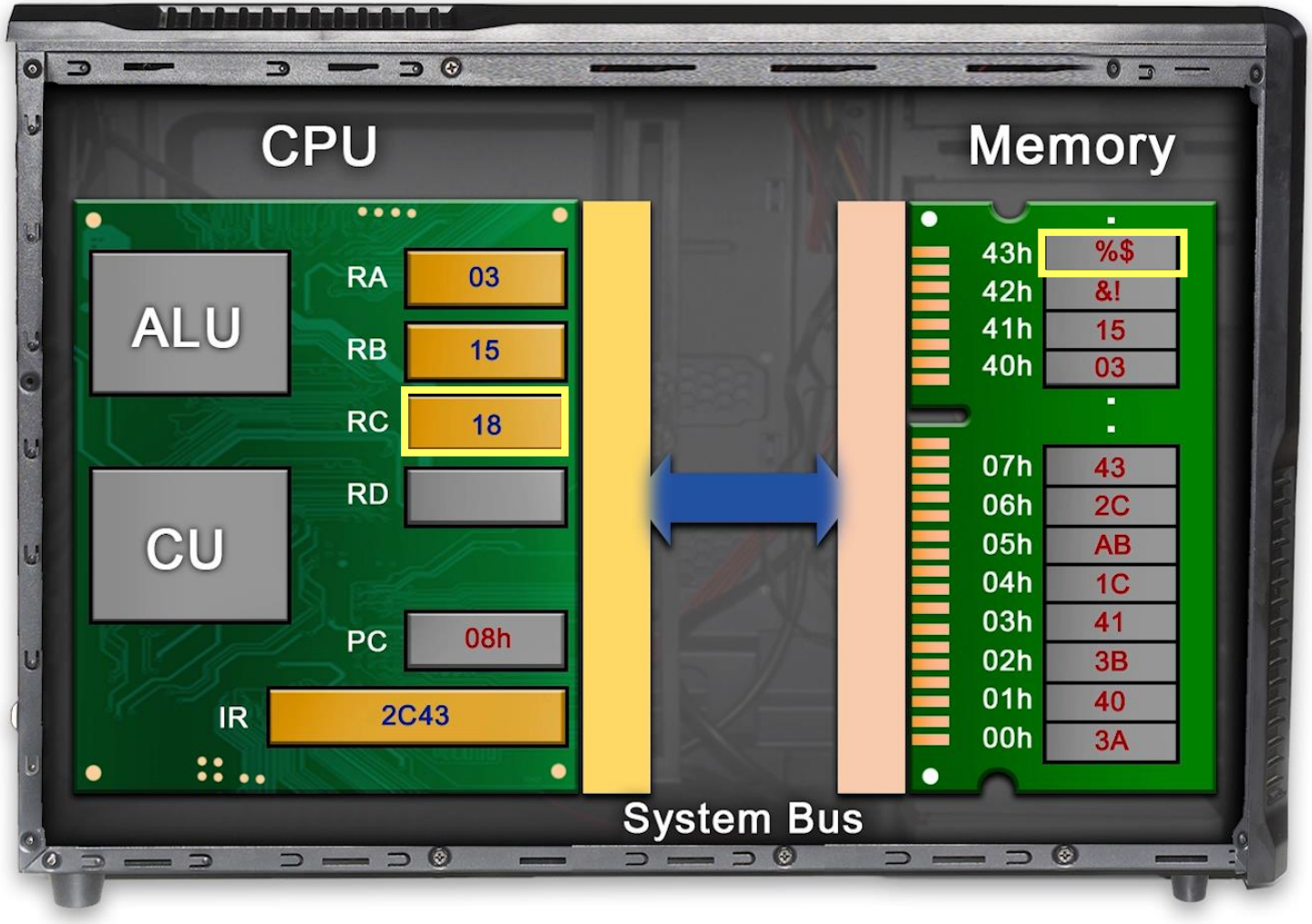


An Illustrative Example (Cont'd)

Decode 4

CU decodes the content IR:

- '2C' ⇒ store content of Register C
- '43' ⇒ memory location 43h

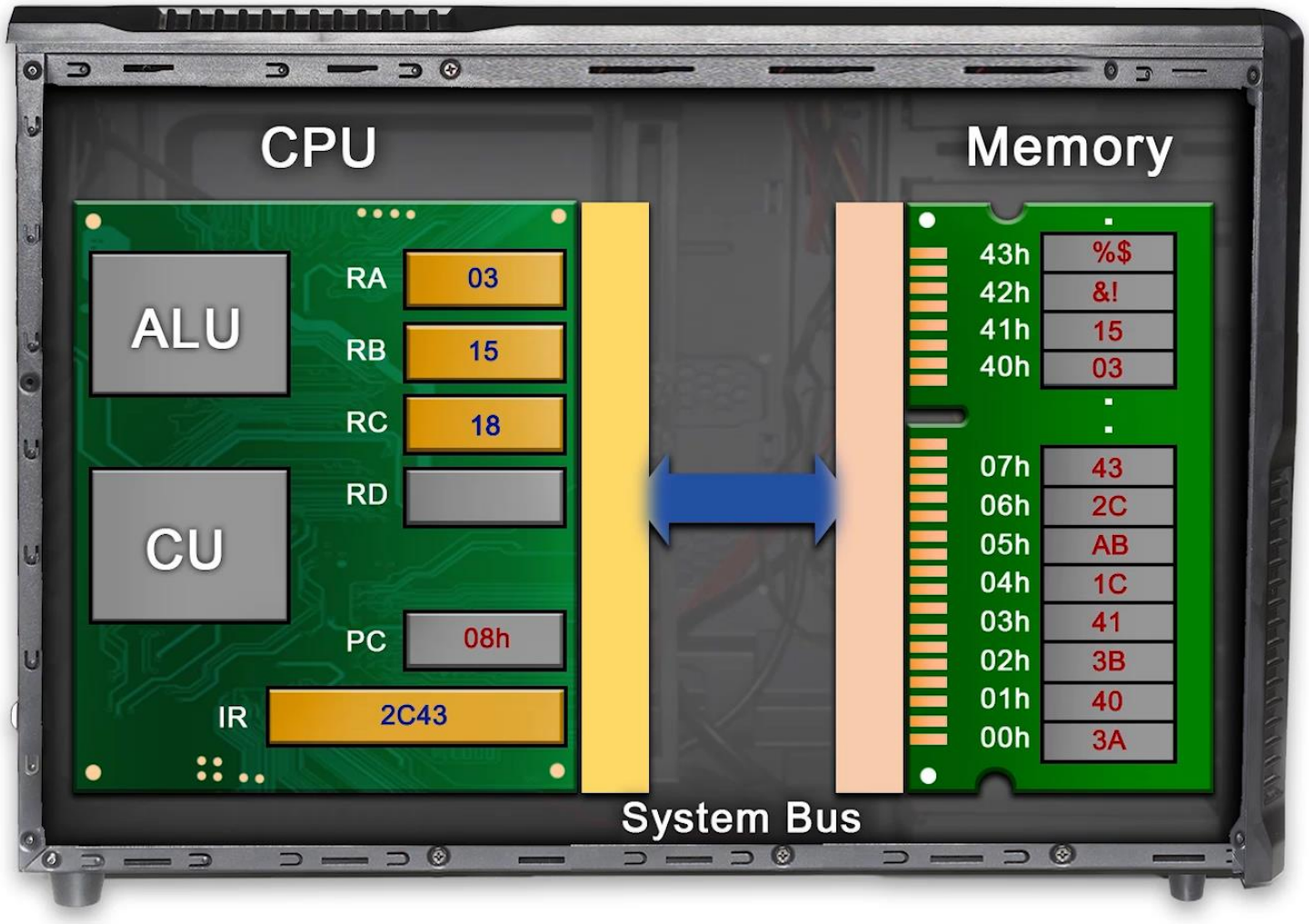


An Illustrative Example (Cont'd)

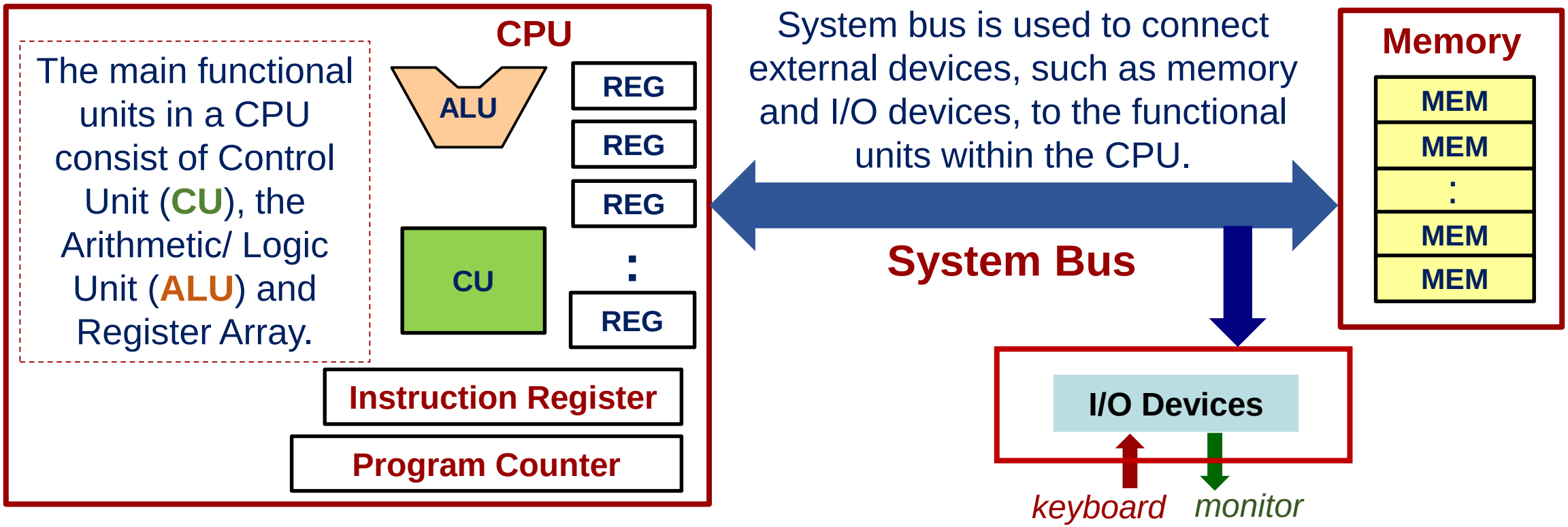
Execute 4

CU executes the instruction:

- Store content Register C in memory location 43h.



Summary



Execution of a program is performed

- with machine instructions encoded in binary format
- through the Fetch-Decode-Execute cycles by the CPU of a microprocessor