*Spring 2022 (Sections 6 & 7)*

# CSE-332/EEE-336:
# Computer Organization and Architecture

## Lecture#1: Introduction and Basic Concepts

**Mainul Hossain, Ph.D.**

**Assistant Professor**

**Department of Electrical and Electronic Engineering**

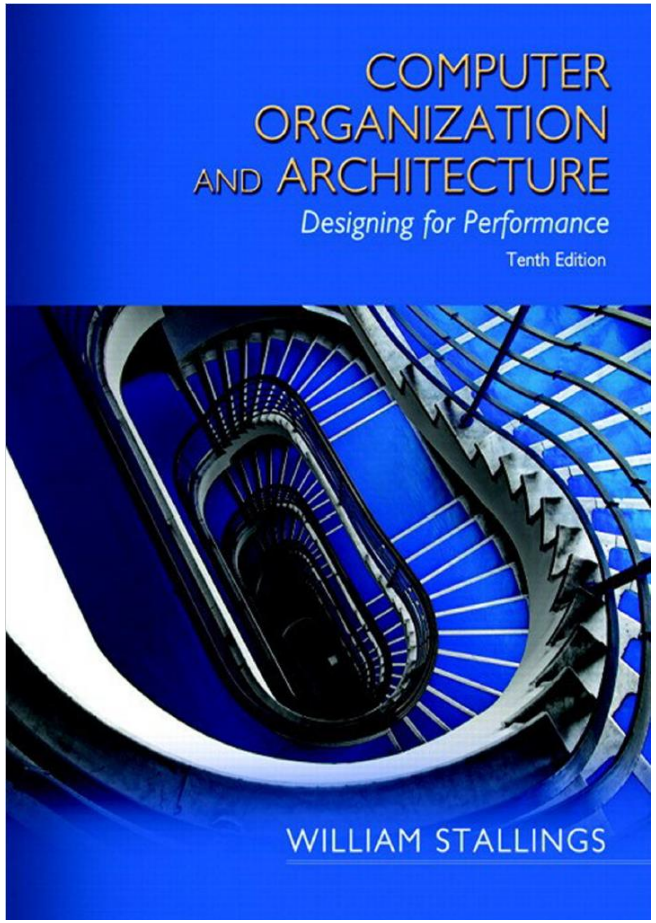
**University of Dhaka**

**mainul.eee@du.ac.bd**

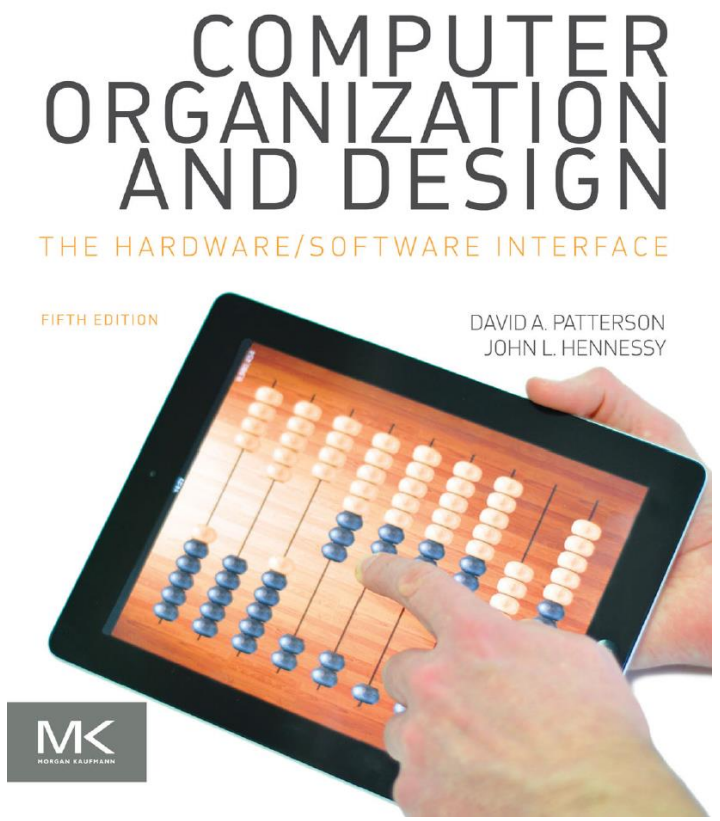
sites.google.com/du.ac.bd/mainulgroup

# Reading

Chapter-1




Chapter-1



# Computer Organization and Architecture

**Computer Organization:** How features are implemented

-**physical aspects** of computer systems (e.g. circuit design, control signals, memory types)

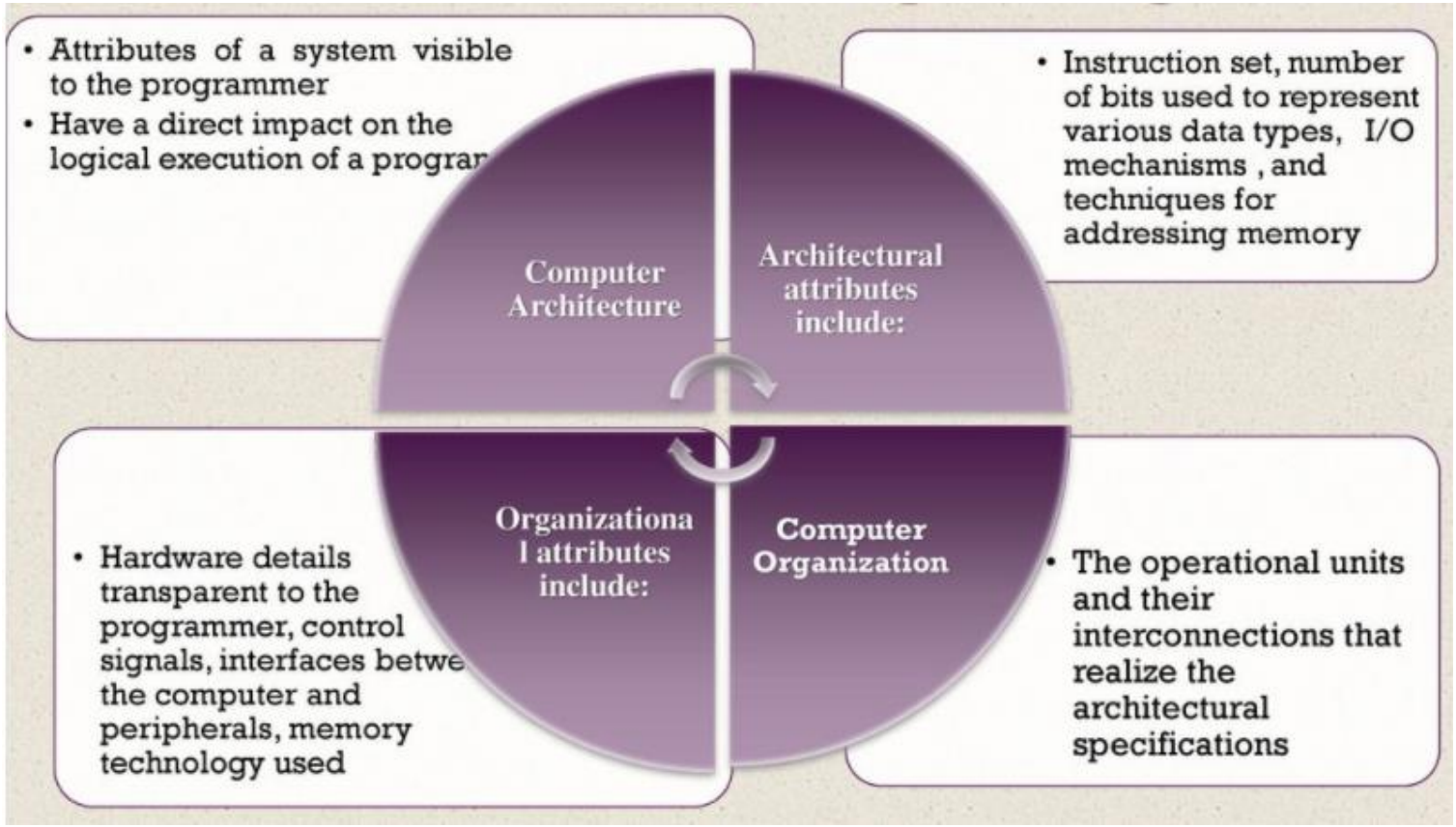
-Is there a hardware multiply unit or is it done by repeated addition?

**Architecture:** Attributes visible to the programmer

-**logical aspects** of systems as seen by the programmer

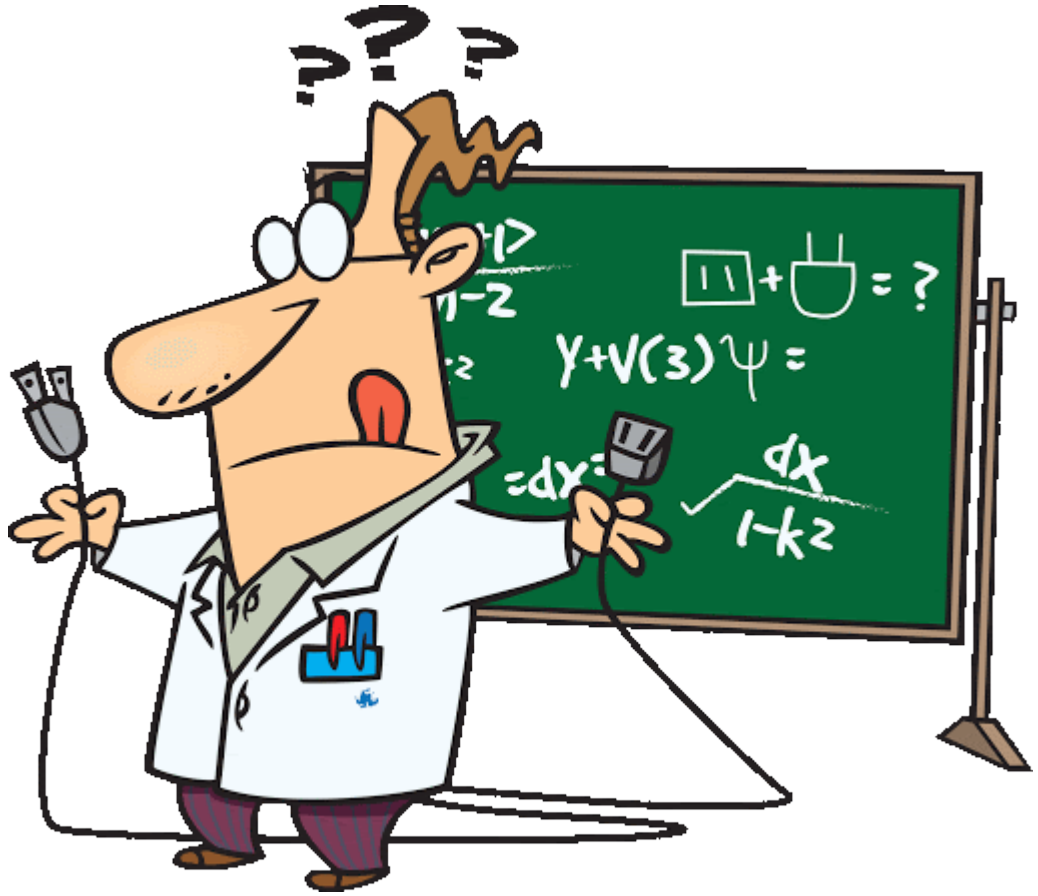
-Instruction set, number of bits used for data representation, I/O mechanism, addressing techniques

-Is there a multiply instruction?



# Why Study Computer Organization and Architecture?

- Design better programs, including system software such as compilers, operating systems, and device drivers.
- Optimize program behavior.
- Evaluate (benchmark) computer system performance.
- Understand time, space, and price tradeoffs.




- Being an architect is not easy
- You need to consider **many** things in designing a new system + have good intuition/insight into ideas/tradeoffs

- But, it is fun and can be very technically rewarding
- And, enables a great future
  - E.g., many scientific and everyday-life innovations would not have been possible without architectural innovation that enabled very high performance systems
  - E.g., your mobile phones

- This course will teach you how to become a good computer architect

# Why study Computer Organization and Architecture?

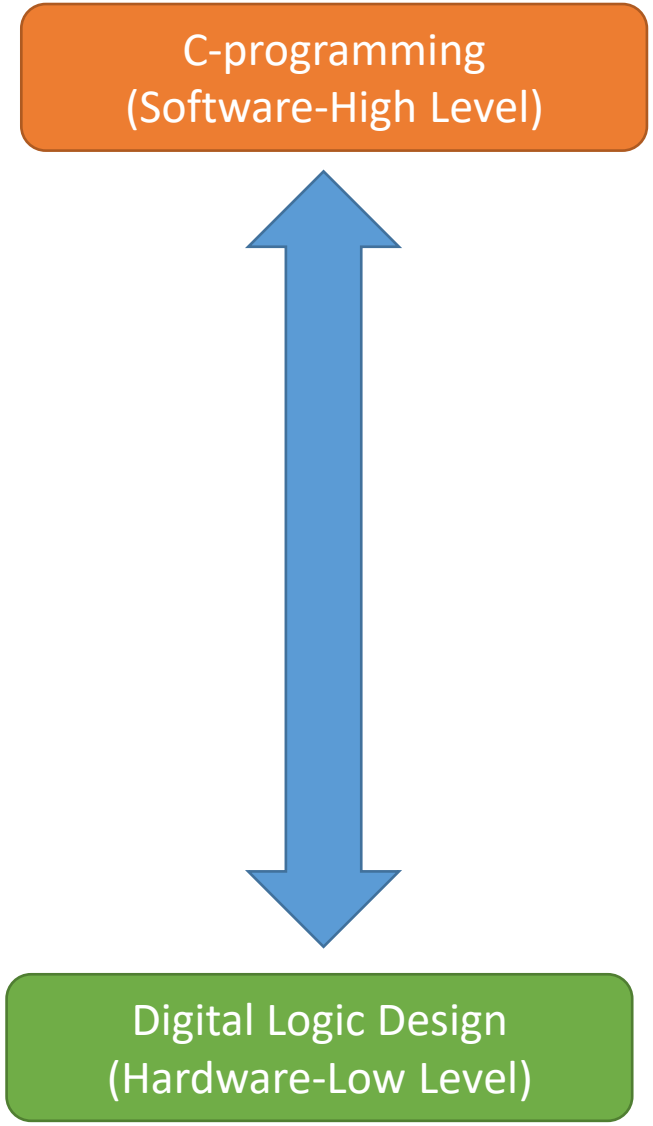
- Enable better systems: make computers faster, cheaper, smaller, more reliable, ...
  - By exploiting advances and changes in underlying technology/circuits
- Enable new applications
  - Life-like 3D visualization 20 years ago?
  - Virtual reality?
  - Personalized genomics? Personalized medicine?
- Enable better solutions to problems
  - Software innovation is built into trends and changes in computer architecture
    - > 50% performance improvement per year has enabled this innovation
- Understand why computers work the way they do

# What Will You Learn from this Course?

- Fundamental principles and tradeoffs in designing the hardware/software interface and major components of a modern programmable microprocessor
  - Focus on state-of-the-art (and some recent research and trends)
  - Trade-offs and how to make them
- How to design, implement, and evaluate a functional modern processor
  - Semester-long lab assignments
  - A combination of RTL implementation and higher-level simulation
  - Focus is functionality first (then, on "how to do even better")
- How to dig out information, think critically and broadly
- How to work even harder and more efficiently!

# Goals of this Course

- Two key goals of this course are
  - to understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer
  - to enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components



# Goals of This Course: A closer look

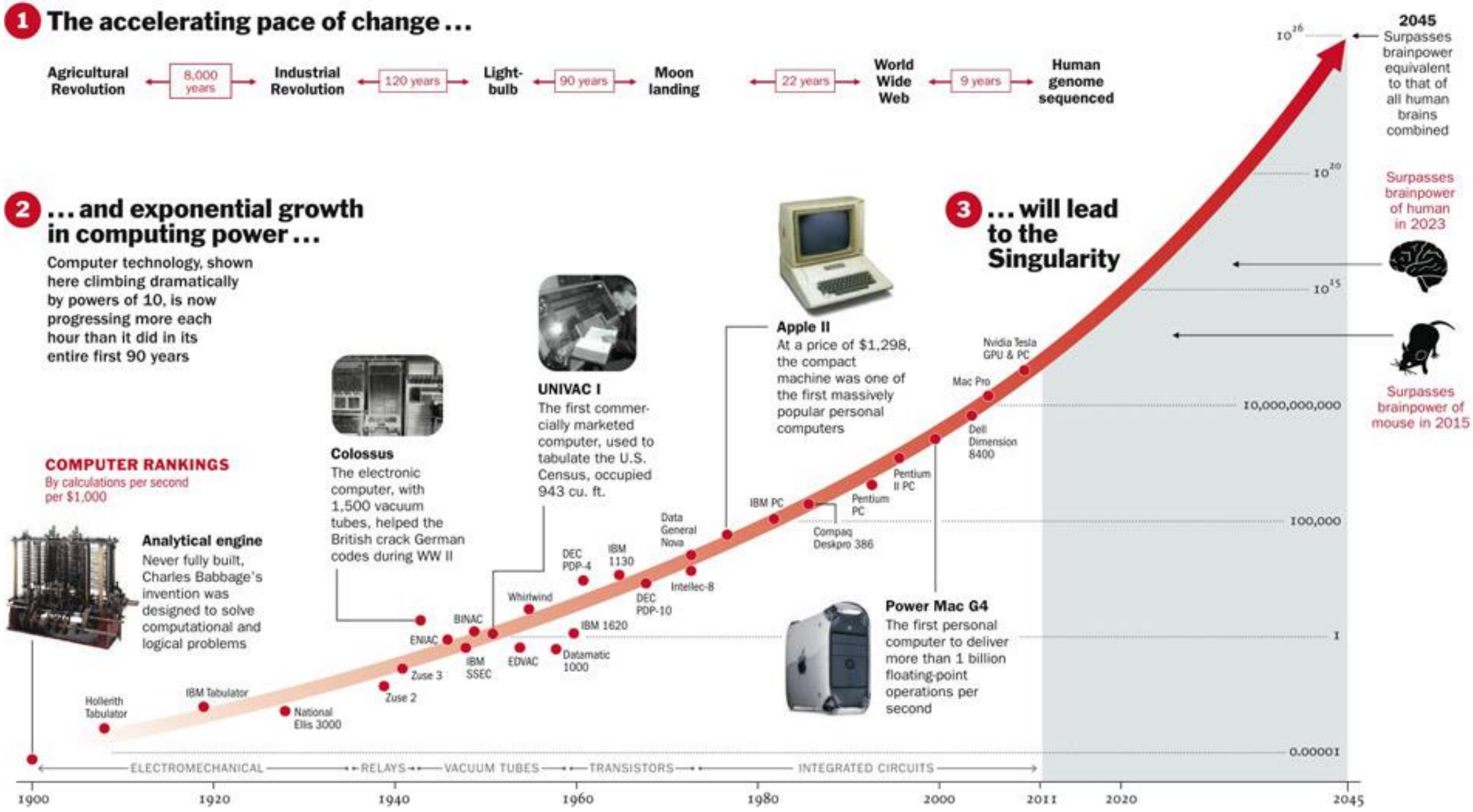
Goal 1: To familiarize those interested in computer system design with both fundamental operation principles and design trade-offs of processor, memory, and platform architectures in today's systems.

- Strong emphasis on fundamentals, design trade-offs, key current/future issues
- Strong emphasis on looking backward, forward, up and down

Goal 2: To provide the necessary background and experience to design, implement, and evaluate a modern processor

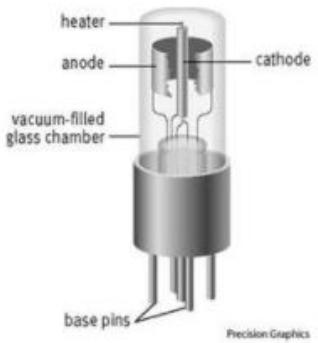
- Strong emphasis on functionality, hands-on design & implementation, and efficiency.
- Strong emphasis on making things work, realizing ideas

# Evolution of Computers

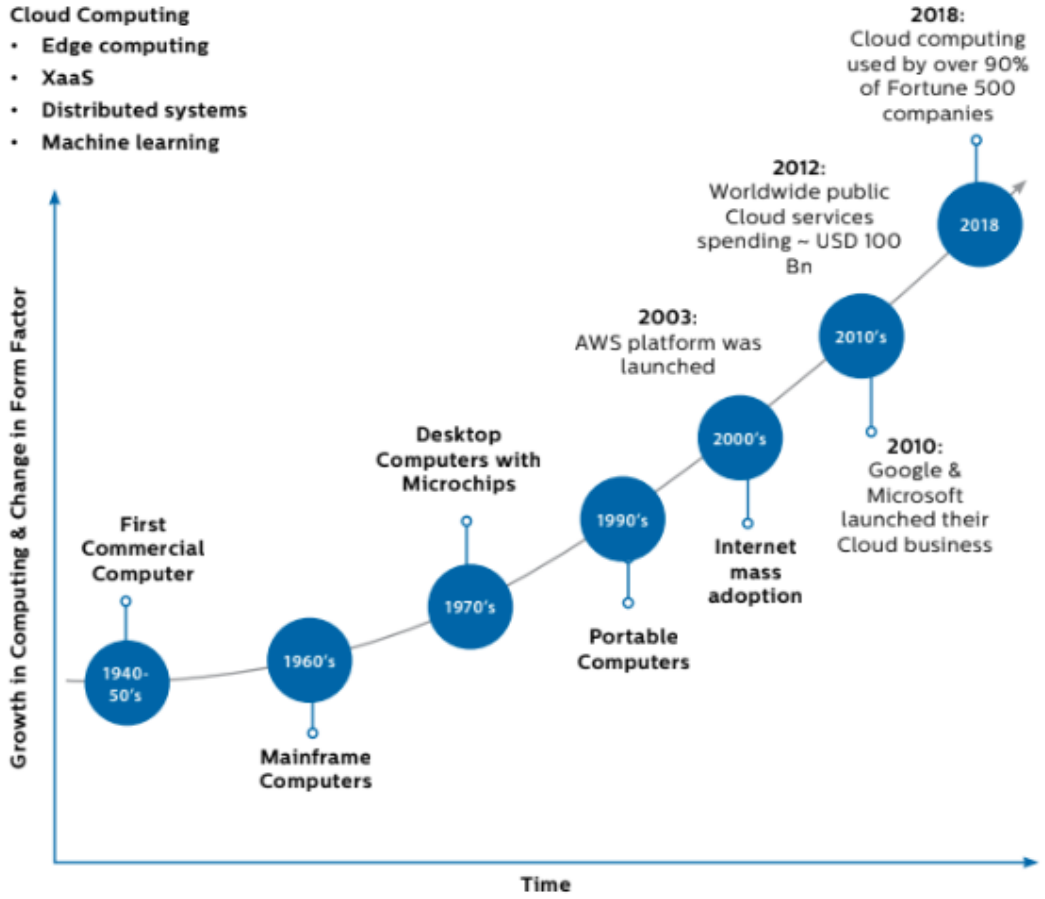


# Evolution of Computers

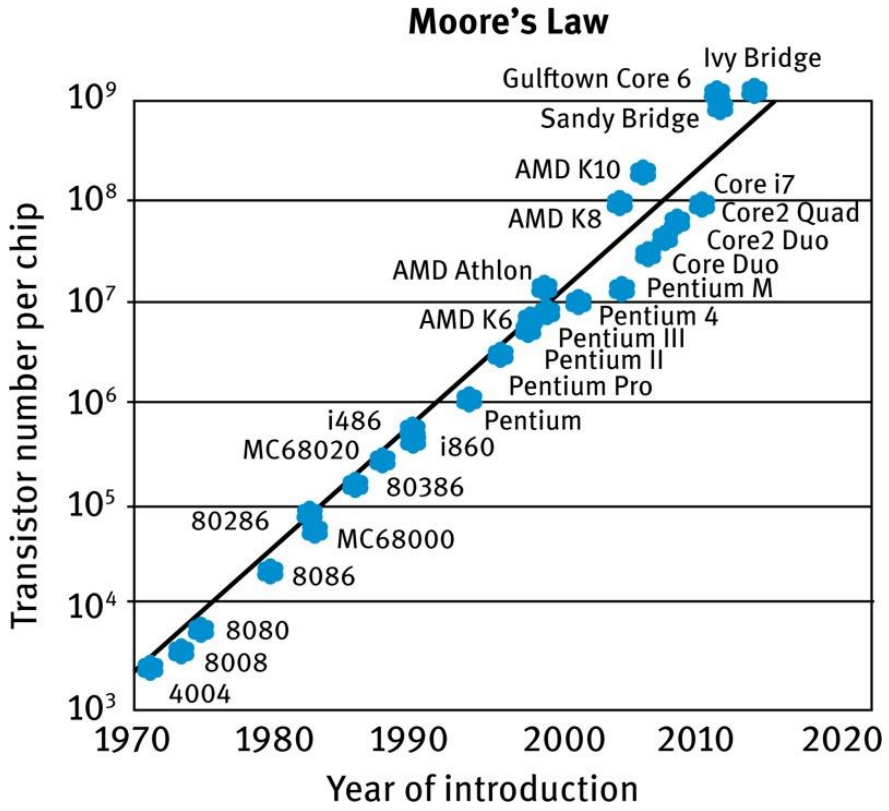
| Generation | Technology | Software apps/ Representing systems |
|---|---|---|
| First Generation (1945-54) | Vacuum tubes, relays, ALU | Machine/assembly languages ENIAC, IBM701 |
| Second Generation (1955-64) | Transistors, memories, IO processors | HLL, Batch processing IBM7090 |
| Third Generation (1965-74) | IC s SSI , MSI, Micro programming | Multiprogramming/Time sharing OS, Intel 8008 |
| Fourth Generation (1975-84) | LSI & VLSI | Multiprocessor OS IBM PC |
| Fifth Generation (1984-90) | VLSI, ULSI multiprocessors | Parallel computing IBM PC AT, Intel 486 |
| Sixth Generation (1990 onwards) | ULSI, VHSIC, high density packing, scalable architecture | Massively parallel processing/ Pentium, Sun Ultra Workstation |




Technology has seen rapid evolution since 1960s – From mainframes to Cloud computing and beyond



# The Key Enabler: Moore`s Law



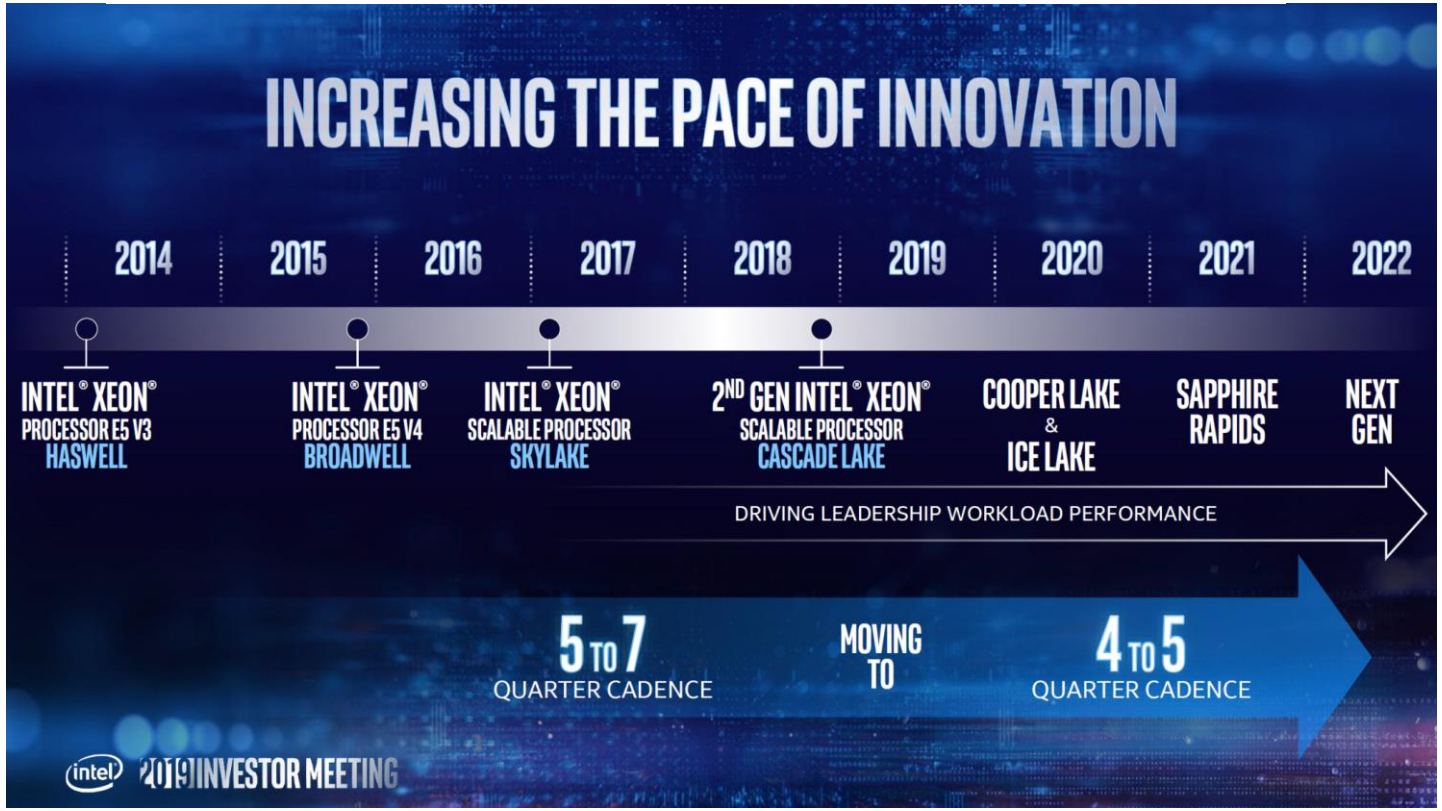

ANNOUNCEMENT : HARDWARE

## Intel's Cooper Lake, 56 Core / 112 Thread Socketed Xeon CPUs, Arriving in 2020 – LGA 4189 Socket, 8 Channel Memory & Support For 10nm Ice Lake-SP on Whitley Platform

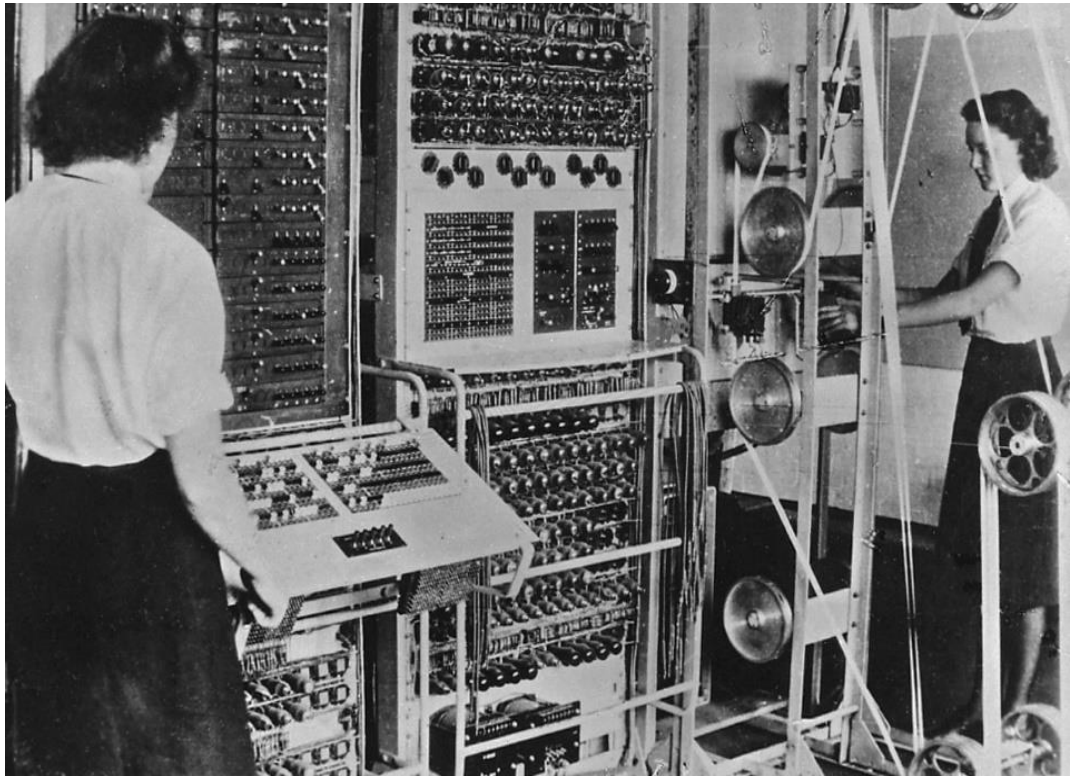
By Hassan Mujtaba

Aug 6, 2019 16:13 EDT

SHARE TWEET SUBMIT



# Early Computers: Colossus

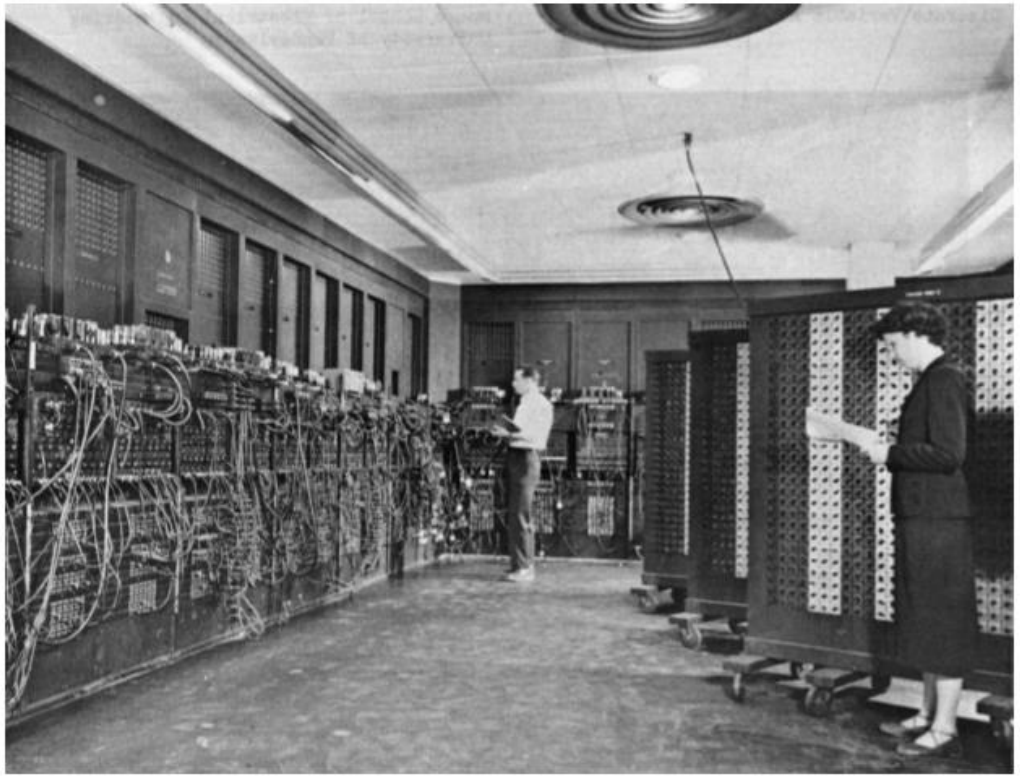


*Colossus was a highly specific machine, programmed rather than programmable. ENIAC was programmable, but a complete change of program could take days (no great problem given its original intended use). Neither machine had an effective random access memory that could work at electronic speeds, and (therefore) neither machine used the stored-program principle.*

**Colossus** was an **electronic digital computer**, built **during WWII** (1943–1945 ) from over 1700 valves (tubes). It was **used to break** the **codes** of the **German** Lorenz SZ-40 **cipher machine** that was **used** by the **German** High Command. **Colossus** is sometimes referred to as the **world's first fixed program, digital, electronic, computer**

*Colossus had relatively little direct impact on the evolution of computers. It was a highly secret operation, with its existence barely acknowledged for decades. There was no contemporary published material. Nearly all the machinery and documentation was destroyed immediately after the war. Its key contribution was that people closely involved, as they moved into civilian occupations, would have both a vision of future possibilities and useful practical experience.*

# Early Computers: ENIAC



Glen Beck and Betty Snyder program the ENIAC in BRL building 328. (Picture: U.S. Army)

*ENIAC was built at the Moore School of Electrical Engineering, at the University of Pennsylvania, Philadelphia, for the US Army's Ballistics Research Lab. It was first working (in secret) in 1945, and was unveiled to the public in February 1946. It was built under John W. Mauchly and J. Presper Eckert, and the team was joined in 1945 by John von Neumann.*

- ENIAC provided conditional jumps and was programmable, clearly distinguishing it from earlier calculators
- Programming was done manually by plugging cables and setting switches, and data was entered on punched cards. Programming for typical calculations required from half an hour to a whole day
- ENIAC was a general-purpose machine, limited primarily by a small amount of storage and tedious programming

**ENIAC = Electronic Numerical Integrator and Computer**

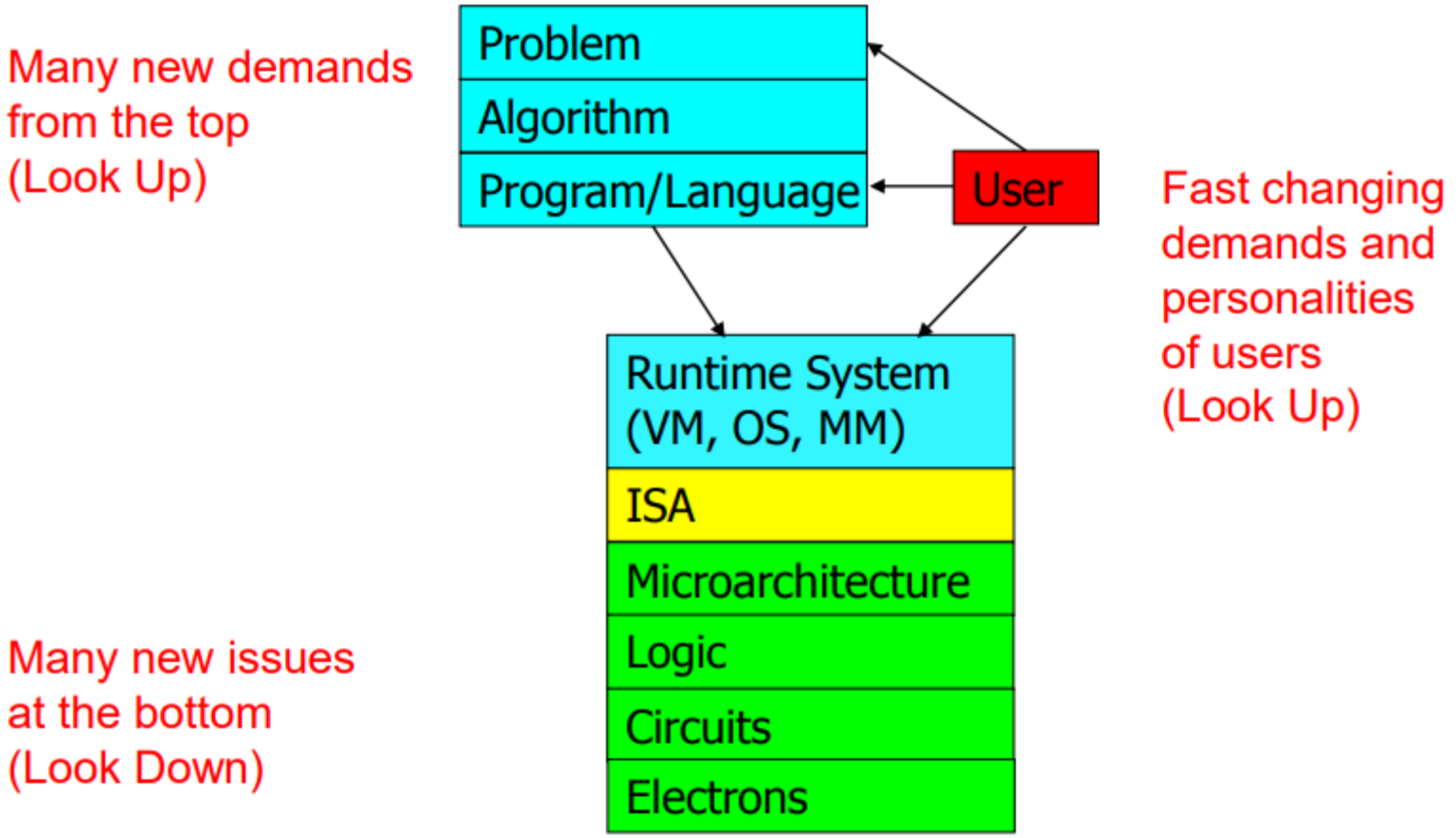
*ENIAC in contrast was central to the evolution of computers. It proved publicly that a large electronic digital machine was viable and useful. The need for an effective electronic random access store to make further progress was well understood -- but the means of achieving it seemed fraught with difficulty! In 1945 work was already in hand to design the successor to ENIAC, EDVAC. This would include an effective electronic Random Access Memory (well, nearly random access!) and the stored-program principle, i.e. the RAM would contain program as well as data, allowing for easy change of program and fast execution of programs. An early draft report was written by von Neumann, which resulted in the classic basic computer design being labelled the "**von Neumann computer**". (Sad as it is for joint pioneers, at least the public is grateful that it hasn't been known as e.g. the Mauchly-Eckert-von-Neumann computer -- or the Williams Tube as the Williams-Kilburn Tube!)*

# Computer Architecture Today-(1)

- Today is a very exciting time to study computer architecture
- Industry is in a large paradigm shift (to multi-core and beyond) – many different potential system designs possible
- Many difficult problems *motivating* and *caused by* the shift
  - Power/energy constraints → multi-core?
  - Complexity of design → multi-core?
  - Difficulties in technology scaling → new technologies?
  - Memory wall/gap
  - Reliability wall/issues
  - Programmability wall/problem
  - Huge hunger for data and new data-intensive applications
- No clear, definitive answers to these problems

# Computer Architecture Today-(2)

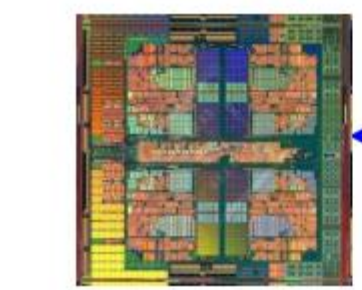
- These problems affect all parts of the computing stack – if we do not change the way we design systems



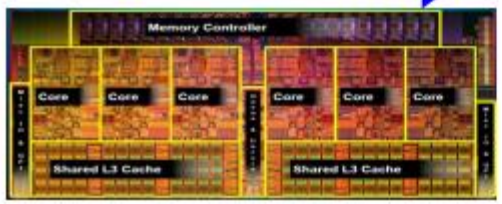

- No clear, definitive answers to these problems

# Computer Architecture Today-(3)

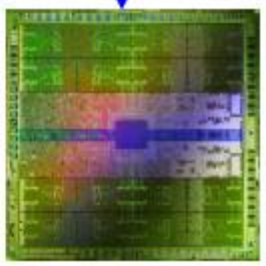
- Computing landscape is very different from 10-20 years ago
- Both UP (software and humanity trends) and DOWN (technologies and their issues), FORWARD and BACKWARD, and the resulting requirements and constraints



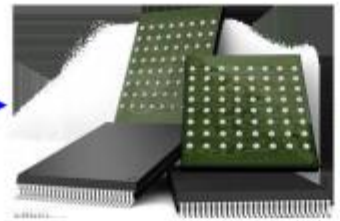
Heterogeneous Processors and Accelerators




Hybrid Main Memory



General Purpose GPUs

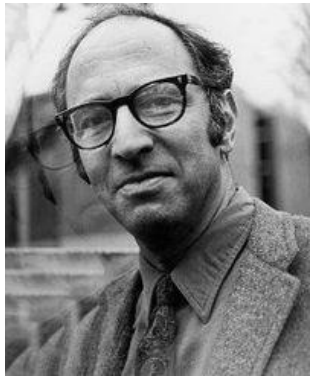


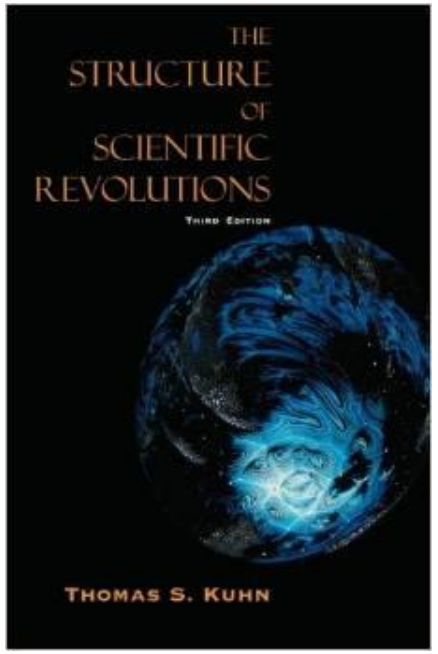
Persistent Memory/Storage

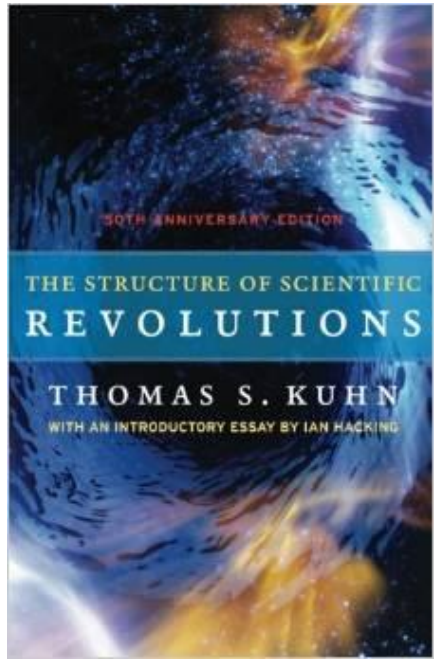
**Every component and its interfaces, as well as entire system designs are being re-examined**

# Computer Architecture Today-(4)

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)
- You can invent new paradigms for computation, communication, and storage
- Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined





# The Role of a Computer Architect

- Look backward (to the past)
  - Understand tradeoffs and designs, upsides/downsides, past workloads. Analyze and evaluate the past.
- Look forward (to the future)
  - Be the dreamer and create new designs. Listen to dreamers.
  - Push the state of the art. Evaluate new design choices.
- Look up (towards problems in the computing stack)
  - Understand important problems and their nature.
  - Develop architectures and ideas to solve important problems.
- Look down (towards device/circuit technology)
  - Understand the capabilities of the underlying technology.
  - Predict and adapt to the future of technology (you are designing for N years ahead). Enable the future technology.

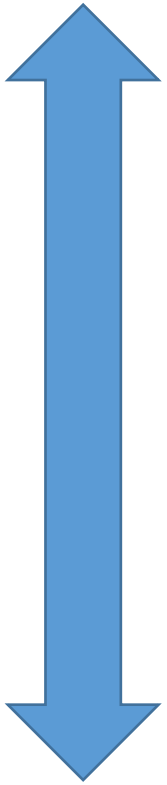
*Now, this is just a simulation of what the blocks will look like once they've assembled*

# The Role of a Computer Architect

"C" as a model of computation

Programmer's view of how a computer system works

C-programming
(Software-High Level)



- How does an assembly program end up executing as digital logic?
- What happens in-between?
- How is a computer designed using logic gates and wires to satisfy specific goals?

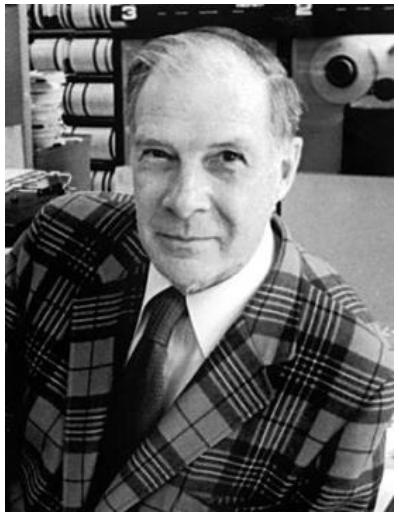
*Architect/microarchitect's view: How to design a computer that meets system design goals. Choices critically affect both the SW programmer and the HW designer*

HW designer's view of how a computer system works

Digital logic as a model of computation

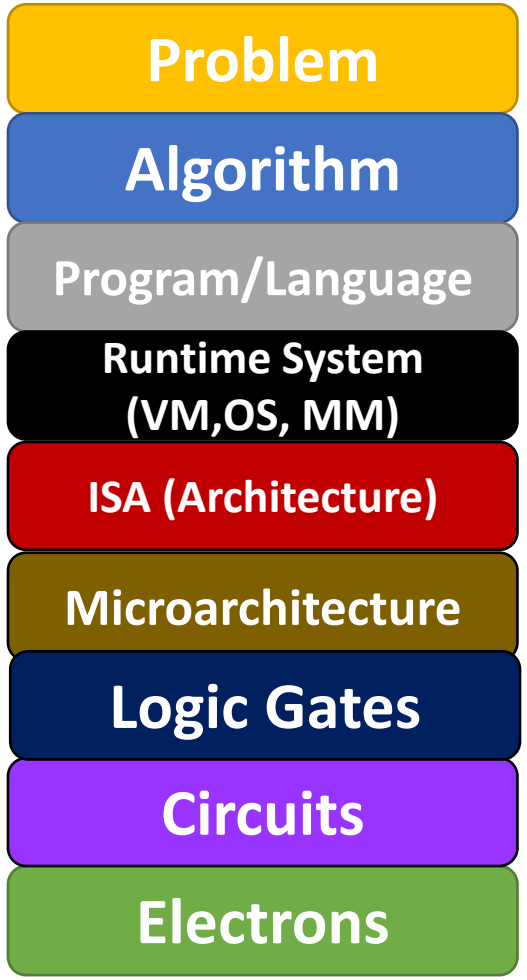
Digital Logic Design
(Hardware-Low Level)

# The Purpose of Computing



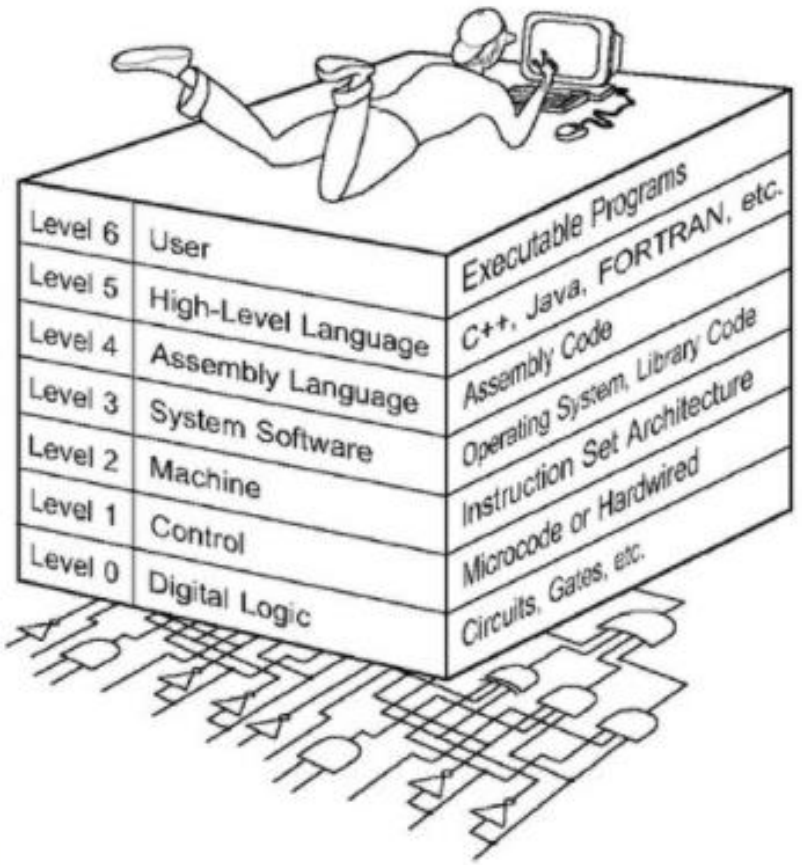
"The purpose of computing is insight" (*Richard Hamming*)
*We gain and generate insight by solving problems*
*How do we ensure problems are solved by electrons?*


- Hamming, "Error Detecting and Error Correcting Codes," Bell System Technical Journal 1950.
- Developed a theory of codes used for error detection and correction



# Computer Level Hierarchy

- Each virtual machine layer is an abstraction of the level below it.
- The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.
- Computer circuits ultimately carry out the work.



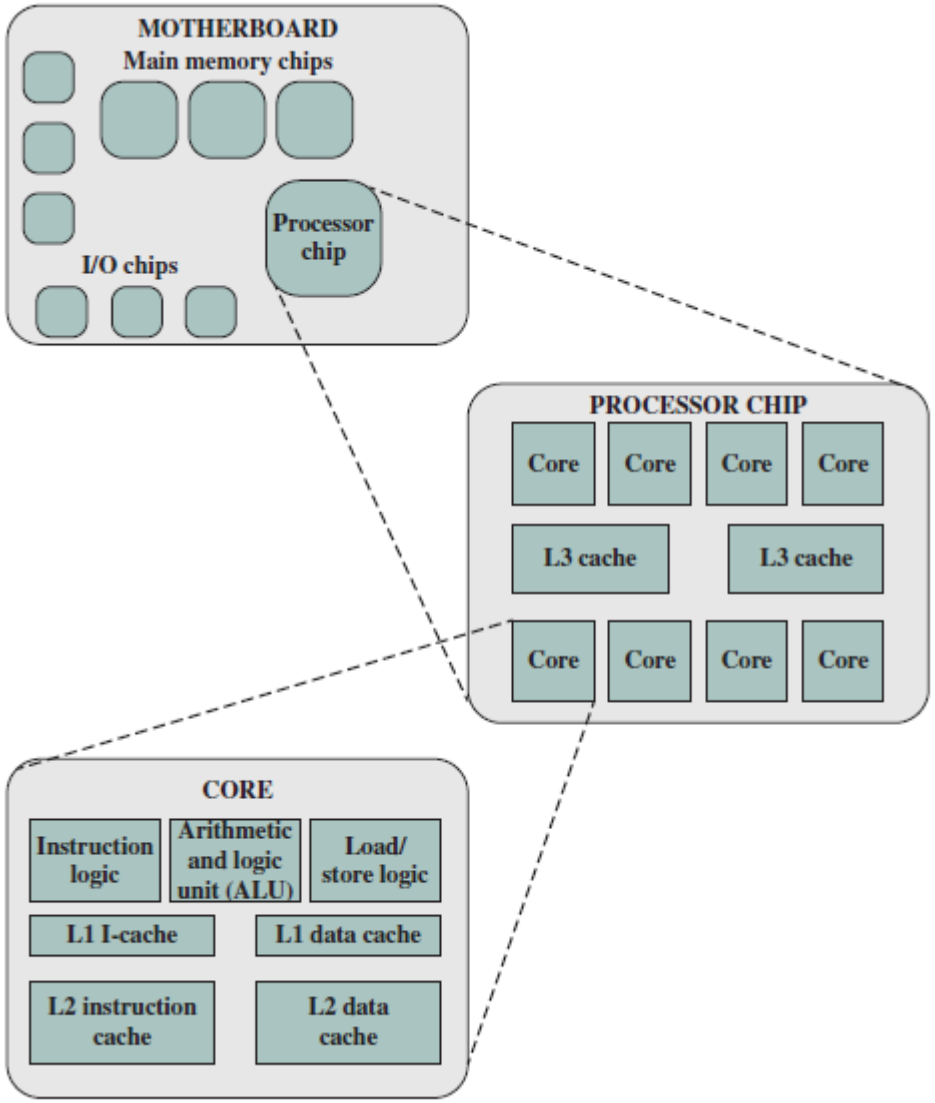
# The Power of Abstraction

- Levels of transformation create abstractions
  - Abstraction: A higher level only needs to know about the interface to the lower level, not how the lower level is implemented
  - E.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions
- Abstraction improves productivity
  - No need to worry about decisions made in underlying levels
  - E.g., programming in Java vs. C vs. assembly vs. binary vs. by specifying control signals of each transistor every cycle
- Then, why would you want to know what goes on underneath or above?

# Crossing the Abstraction Layers

- As long as everything goes well, not knowing what happens in the underlying level (or above) is not a problem.
- What if
  - The program you wrote is running slow?
  - The program you wrote does not run correctly?
  - The program you wrote consumes too much energy?
- What if
  - The hardware you designed is too hard to program?
  - The hardware you designed is too slow because it does not provide the right primitives to the software?
- What if
  - You want to design a much more efficient and higher performance system?

# Multi-Core Processors

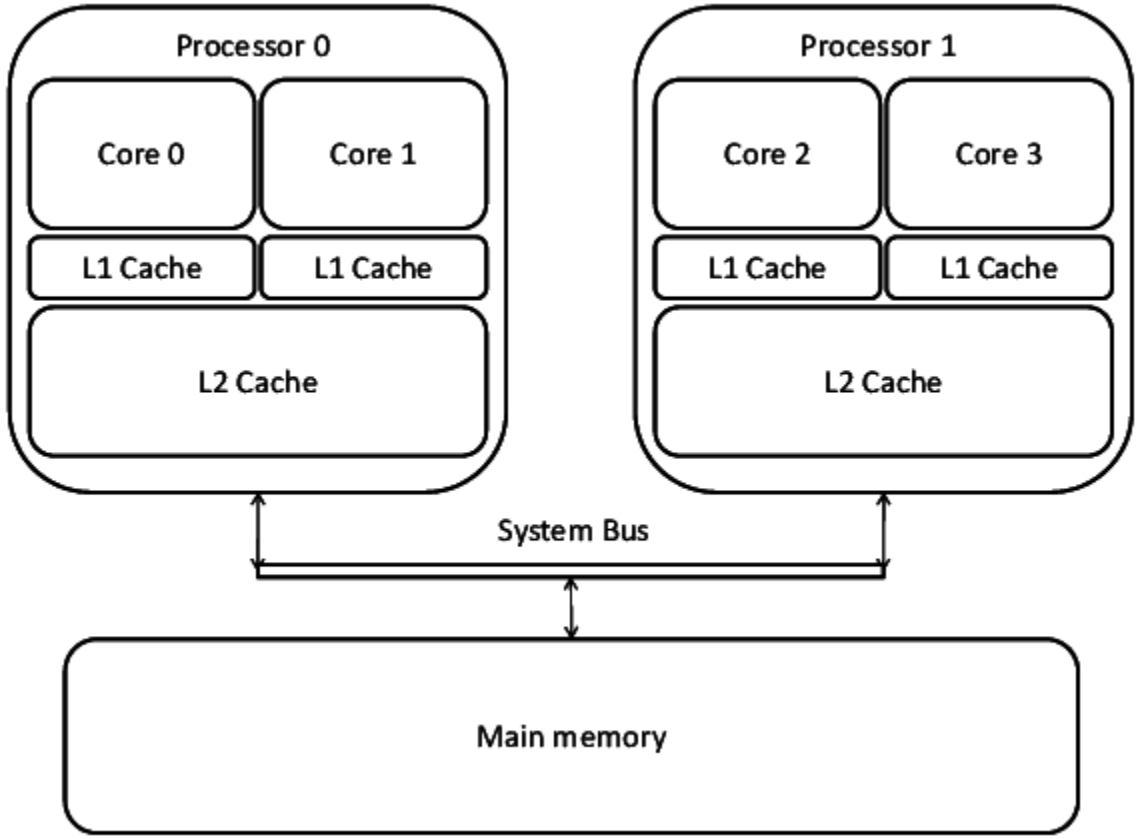



*MULTICORE COMPUTER STRUCTURE* As was mentioned, contemporary computers generally have multiple processors. When these processors all reside on a single chip, the term *multicore computer* is used, and each processing unit (consisting of a control unit, ALU, registers, and perhaps cache) is called a *core*. To clarify the terminology, this text will use the following definitions.

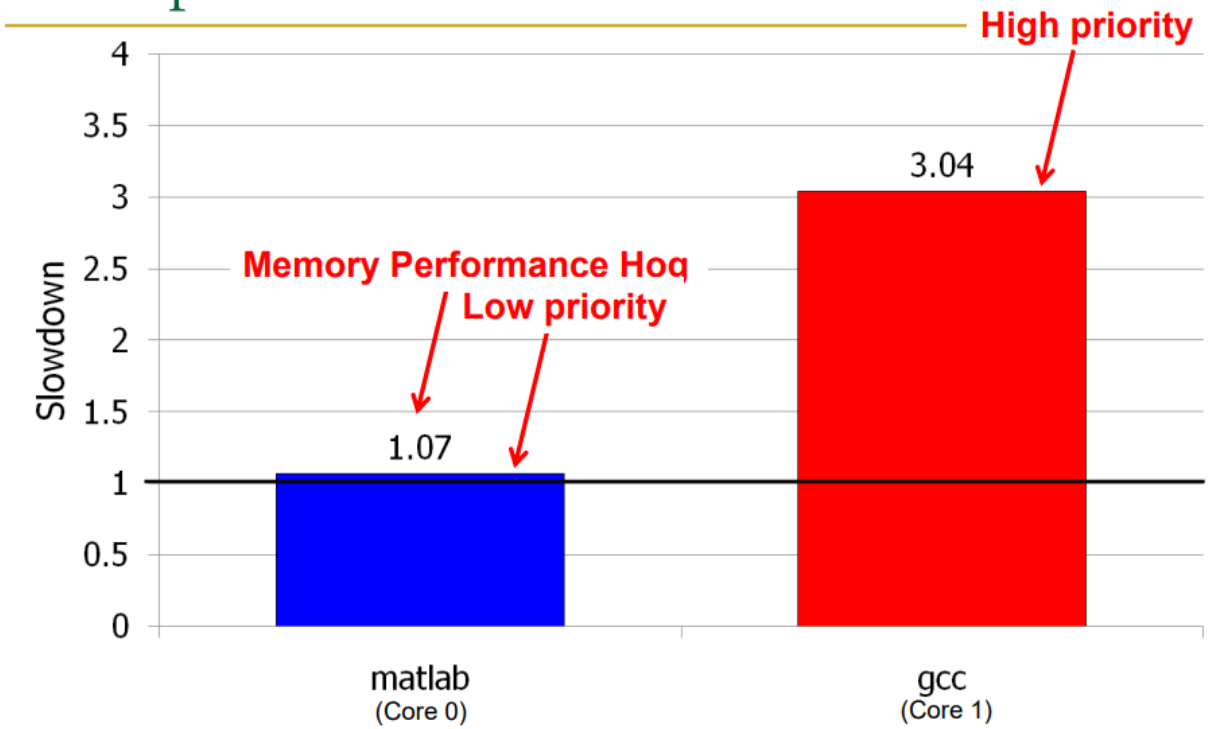
- **Central processing unit (CPU):** That portion of a computer that fetches and executes instructions. It consists of an ALU, a control unit, and registers. In a system with a single processing unit, it is often simply referred to as a *processor*.
- **Core:** An individual processing unit on a processor chip. A core may be equivalent in functionality to a CPU on a single-CPU system. Other specialized processing units, such as one optimized for vector and matrix operations, are also referred to as cores.
- **Processor:** A physical piece of silicon containing one or more cores. The processor is the computer component that interprets and executes instructions. If a processor contains multiple cores, it is referred to as a **multicore processor**.

***A cache memory is smaller and faster than main memory and is used to speed up memory access, by placing in the cache data from main memory, that is likely to be used in the near future. A greater performance improvement may be obtained by using multiple levels of cache, with level 1 (L1) closest to the core and additional levels (L2, L3, and so on) progressively farther from the core.***

# EXAMPLE: Multi-Core System

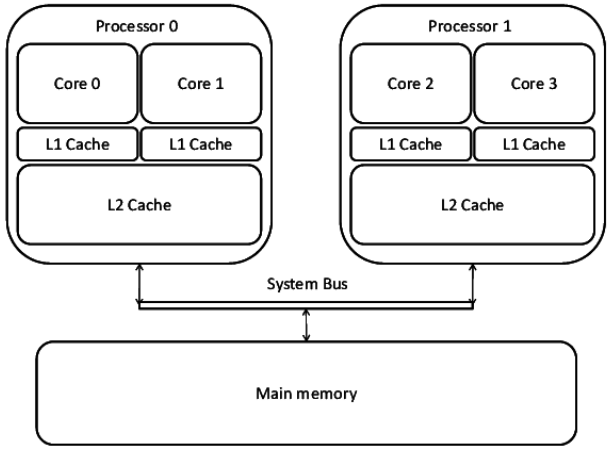


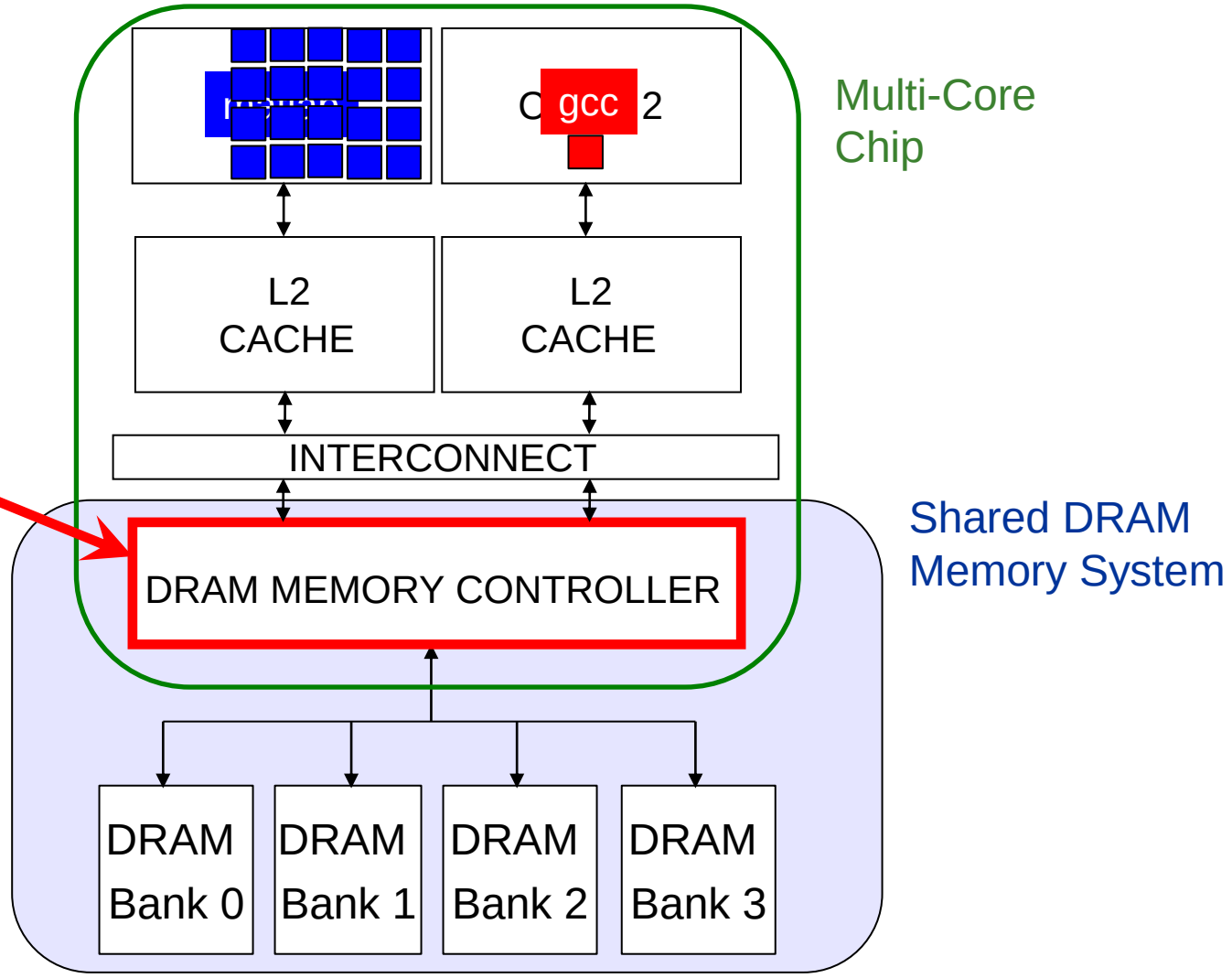

## Unexpected Slowdowns in Multi-Core




Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

# EXAMPLE: Multi-Core System



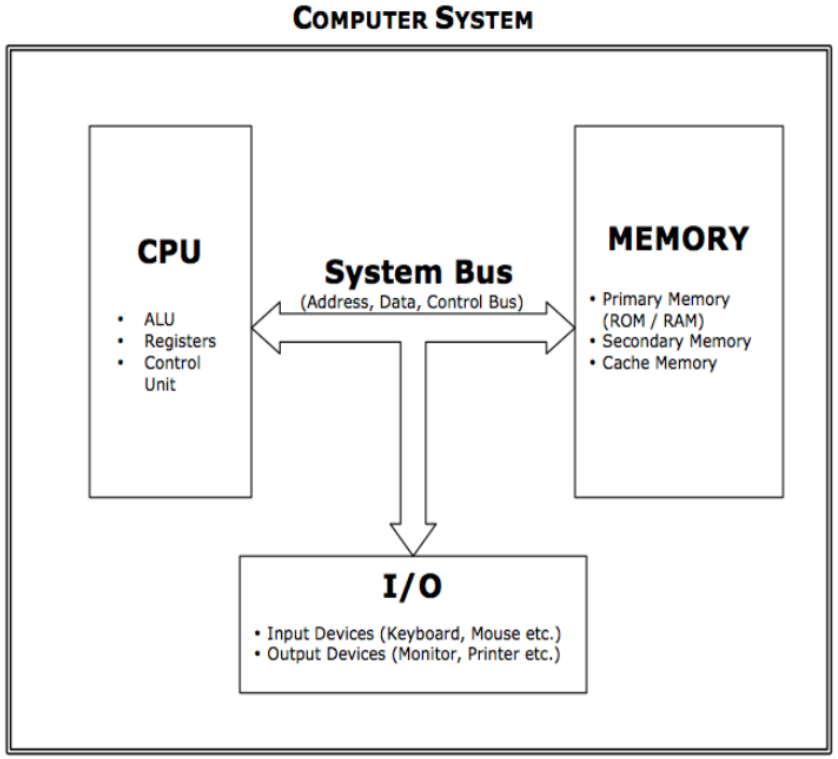




unfairness

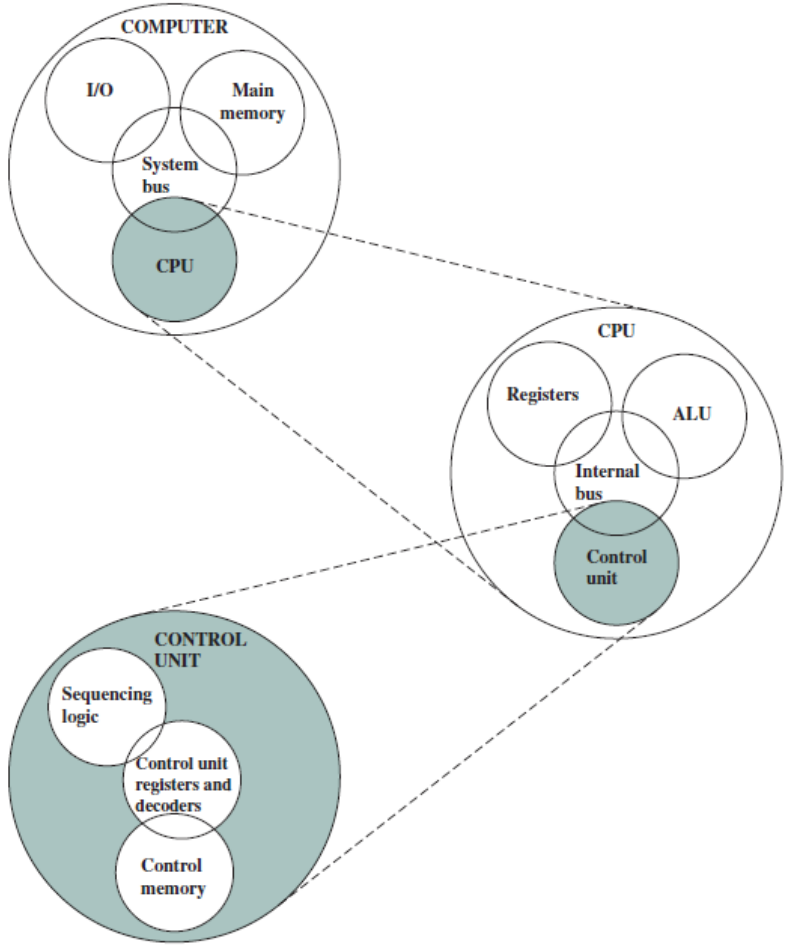
- Can you figure out why there is a disparity in slowdowns if you do not know how the system executes the programs?
- Can you fix the problem without knowing what is happening "underneath"?
- Multiple applications share the DRAM controller
- DRAM controllers designed to maximize DRAM data throughput
- DRAM scheduling policies are unfair to some applications

# Top Level Structure of a Computer

*The Job of a computer is to execute programs*



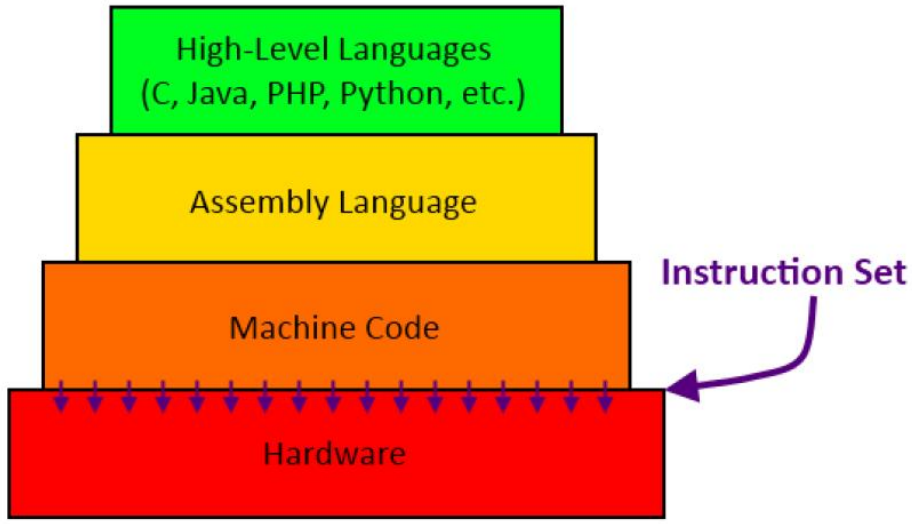

- **Central processing unit (CPU):** Controls the operation of the computer and performs its data processing functions; often simply referred to as **processor**.
- **Main memory:** Stores data.
- **I/O:** Moves data between the computer and its external environment.
- **System interconnection:** Some mechanism that provides for communication among CPU, main memory, and I/O. A common example of system interconnection is by means of a **system bus**, consisting of a number of conducting wires to which all the other components attach.



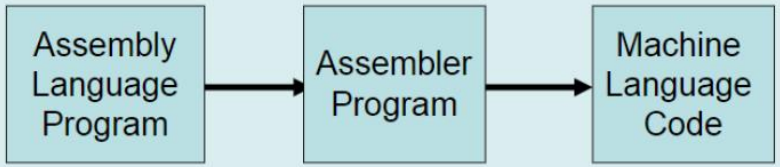

- **Control unit:** Controls the operation of the CPU and hence the computer.
- **Arithmetic and logic unit (ALU):** Performs the computer's data processing functions.
- **Registers:** Provides storage internal to the CPU.
- **CPU interconnection:** Some mechanism that provides for communication among the control unit, ALU, and registers.

# Programming Hierarchy

- A program is sequence of instructions
- A program can be written in
  - Low level language e.g. assembly language
  - High level language e.g. C, PASCAL
- Assembly language program is converted to machine code by an assembler
- High level language is translated into machine code by a compiler




- Microprocessor cannot understand a program written in Assembly language
- A program known as **Assembler** is used to convert a Assembly language program to machine language




- 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

# Hexadecimal Number Systems

| decimal | hexadecimal | binary |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

***Hex to Binary Number Conversion***

*Convert the hexadecimal $9DB5_{16}$ to its binary equivalent.*

1. *Separate the digits of the given hexadecimal, if more than 1 digit.*

*9 D B 5*

2. *Find the equivalent binary number for each digit of hex number, add 0's to the left if any of the binary number is shorter than 4 bits.*

*9 D B 5*

*1001 1101 1011 0101*

3. *Write the all groups binary numbers together, maintaining the same group order provides the equivalent binary for the given hexadecimal.*

*1001 1101 10110101*

***Result***

$9DB5_{16} = 1001\ 1101\ 10110101_{2}$

# Hexadecimal Number Systems

- They are much shorter than binary numbers, so easy to write and remember.
- Has a base of 16.
- Although any 16 digits may be used, everyone uses 0 to 9 and A to F.
- After reaching 9 in the hexadecimal system, we can continue counting as A, B, C, D,E,F

What is the largest number you can represent using four binary digits?

| 1 | 1 | 1 | 1 | $_2$ |
|---|---|---|---|---|
| $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
| = | = | = | = | |
| 8 | 4 | 2 | 1 | |

$8 + 4 + 2 + 1 = 15_{10}$

... the smallest number?

| 0 | 0 | 0 | 0 | $_2$ |
|---|---|---|---|---|
| $2^3$ | $2^2$ | $2^1$ | $2^0$ | |

$0 + 0 + 0 + 0 = 0_{10}$

What is the largest number you can represent using a single hexadecimal digit?

$F_{16} = 15_{10}$

... the smallest number?

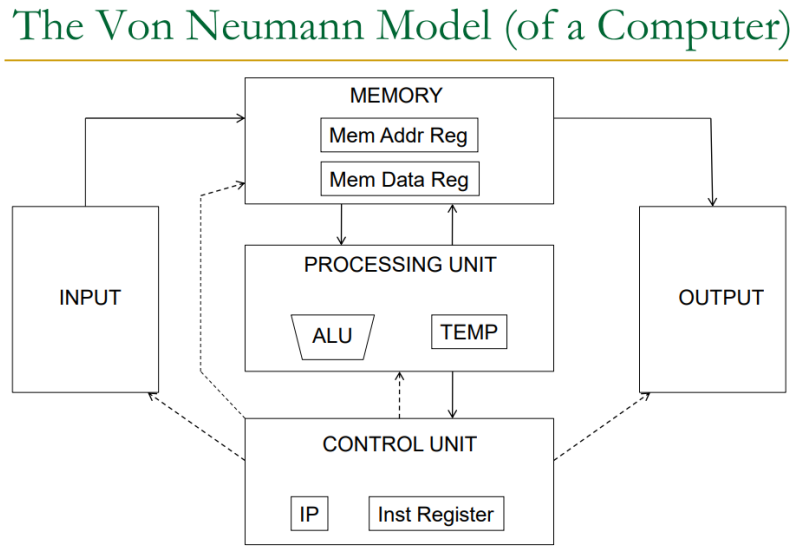
$0_{16} = 0_{10}$

**Base 16**
**Cheat Sheet**

$A_{16} = 10_{10}$
$B_{16} = 11_{10}$
$C_{16} = 12_{10}$
$D_{16} = 13_{10}$
$E_{16} = 14_{10}$
$F_{16} = 15_{10}$

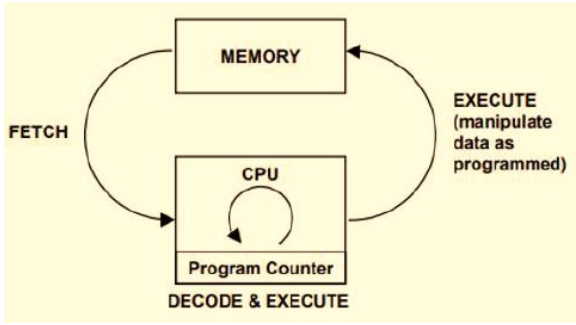
Note: You can represent the same range of values with a single hexadecimal digit that you can represent using four binary digits!

# The von Neumann Model of a Computer

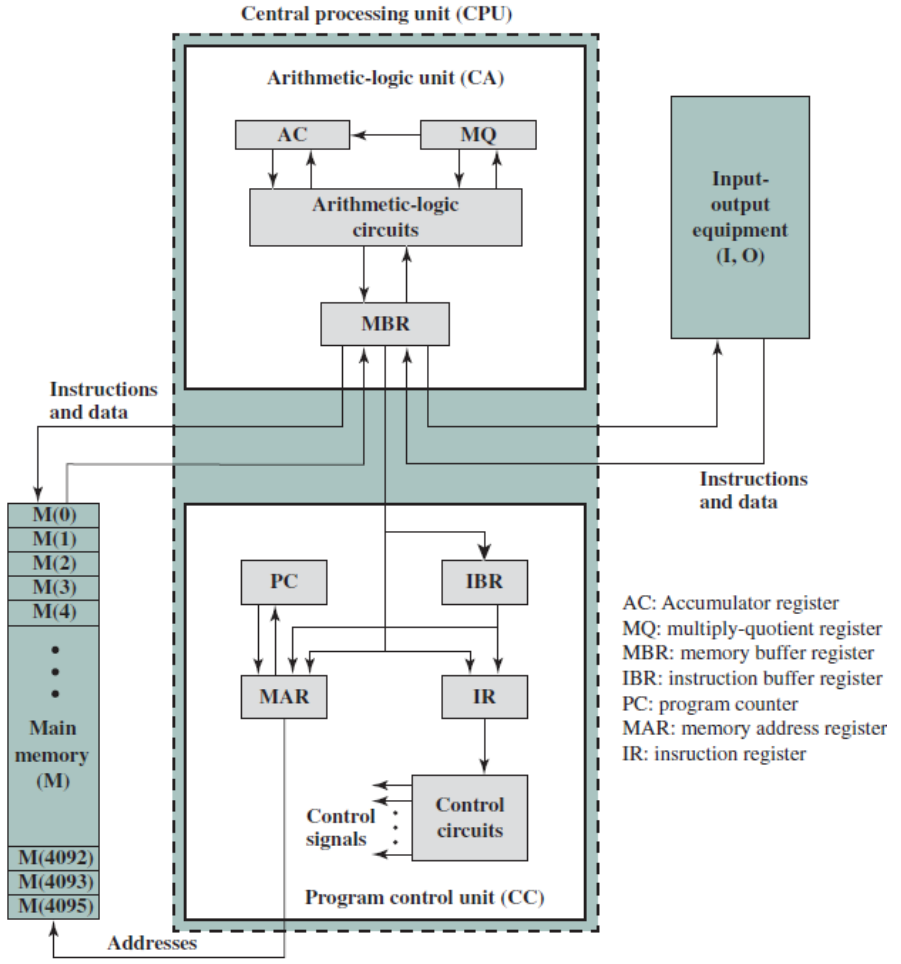



- Also called *stored program computer* (instructions in memory). Two key properties:
- Stored program
  - Instructions stored in a linear memory array
  - Memory is unified between instructions and data
    - The interpretation of a stored value depends on the control signals

When is a value interpreted as an instruction?

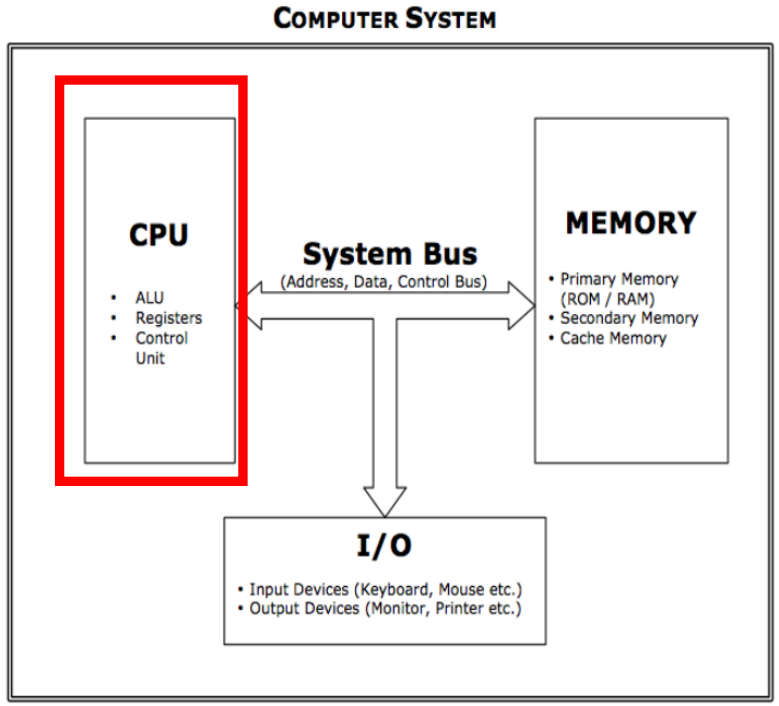



- Sequential instruction processing
  - One instruction processed (fetched, executed, and completed) at a time
  - Program counter (instruction pointer) identifies the current instr.
  - Program counter is advanced sequentially except for control transfer instructions




**Institute of Advanced Studies (IAS) Structure**

# The Central Processing Unit (CPU)

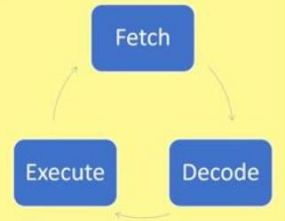



Processor

Control Unit (Fetch, Decode)

Execution Unit (ALU) (Execute)

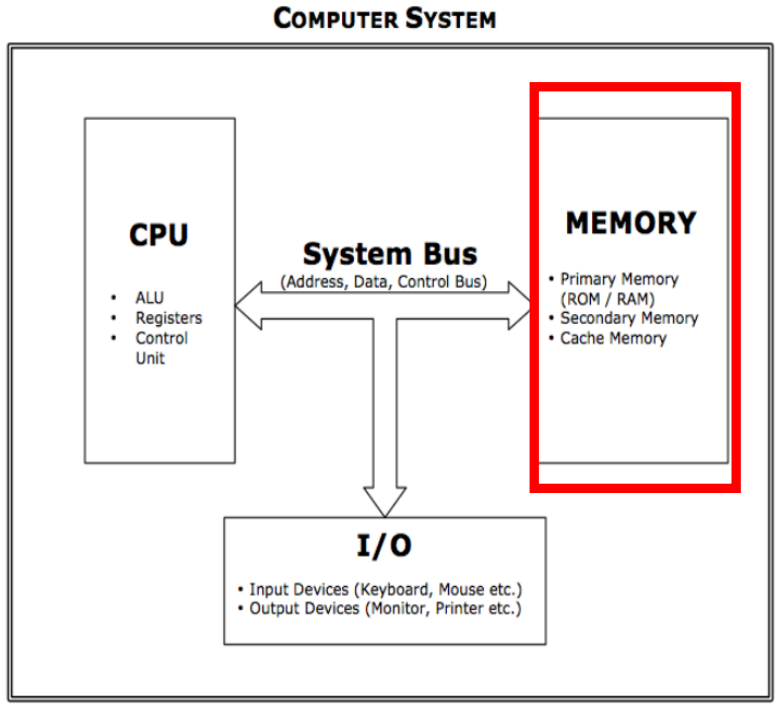
- Fetch-Decode-Execute
  - The processor is continually fetching new instructions from memory, decoding them, then executing them. This is called the **fetch-decode-execute cycle**.
- Fetch
  - The instruction is moved from memory to the CPU.
- Decode
  - The instruction is understood by the CPU.
- Execute
  - The instruction is carried out.




**CPU**

1) The Central Processing Unit is the **most important part** of the Computer.
2) It is also called the **microprocessor** or simply the **processor**.
3) It consists of the ALU, Registers, Control Unit etc.
4) All **programs are executed** in the CPU.
5) A program is a **set of instructions** stored in the memory.
6) The main function of the CPU is to **fetch, decode and execute** these instructions.
7) Instructions are **fetched from the memory** using the various **buses**.
8) Thereafter they are **decoded by the Control Unit** to analyze the Opcode.
9) Finally the instruction is **executed** to perform the **desired operation**.
10) This **execution** mainly involves the **ALU** and the **internal registers** of the processor.

# Memory




## MEMORY

1) The memory is used to **store information**
2) It mainly stores **programs and data**
3) Memory has various **locations**
4) Each location is identified by its **own unique address and contains some data**
5) The most basic form of memory is called **Primary Memory**, which consists of **RAM and ROM**
6) Then there are secondary storage devices such as **hard disk**
7) There are portable storage devices like **CD, DVD, Pen drives** etc.
8) Finally, there **is high-speed memory called Cache** memory implemented using **SRAM**
9) RAM is "**Volatile"** , that means contents of RAM are **lost** after power supply is withdrawn
10) All **other** memories are **Non-volatile** memories

# Memory: Basics

- Memory is an array of storage, each having capacity of 8 bits and holds machine code of instruction or data in binary format
- Each location is uniquely identified by a number/code, starting from '0', called Address, usually represented in Hexadecimal form
- Memory is byte addressable
- RAM temporarily holds machine codes (binary) of instructions/programs to be executed
- RAM also holds data (binary coded) required in a program
- ROM holds boot-up programs, BIOS

# Memory: Capacity

- Example: 1MB: Nearly 1 Million locations each having capacity of 1Byte
- Address starts at 0 and ends at 1 less than 1 Million, actually encoded in BINARY
- In Binary, first address requires 1 bit (0) and final addressable location requires 20 bits (all 1's: 11…11), since $\mathbf{2^{20} = 1M}$
- For ease of Decoder design, uniform address format is used for all the locations; Maximum number of bits!
- For convenience/ease of representation/programming/discussion, Hexadecimal number system is used to represent Memory address

# How a program & data are stored in Memory?

```
C++: cout << (A+B+C)
```

↓

```
Assembly Language:
mov eax, A
add eax, B
add eax, C
call WriteInt
```

→

| Machine codes |
|---|
| 11001101 |
| 10001101 |
| 11101101 |
| 11011101 |

Stored in RAM →

| ADDRESS | CONTENTS |
|---|---|
| 000100100101 (125H) | 11001101 (Machine code of Instruction-1) |
| 000100100110 (126H) | 10001101 (Machine code of Instruction-2) |
| 000100100111 (127H) | 11101101 (Machine code of Instruction-3) |
| 000100101000 (128H) | 11011101 (Machine code of Instruction-4) |
| | |
| 200H | 11000101 (DATA-1) |
| 201H | 11000001 (DATA-2) |
| 202H | 11000111 (DATA-3) |

- Starting address of the program is set by operating system/user depending on system. Here starting address of memory (RAM) is chosen randomly as 125H.
- Machine code of 1st Instruction is stored at 125H.
- Following instructions are stored in consecutive locations of RAM.
- Data, if used and if required to store in RAM, could be stored in a different segment but in consecutive locations of RAM

# Address of Memory (for 1MB)

| Address (20 bits in binary) | Content (8 bits) |
|---|---|
| 11111111111111111111B | 11001100 (machine code/data) |
| … | |
| 00000000000000000000B | 00110101(machine code/data) |

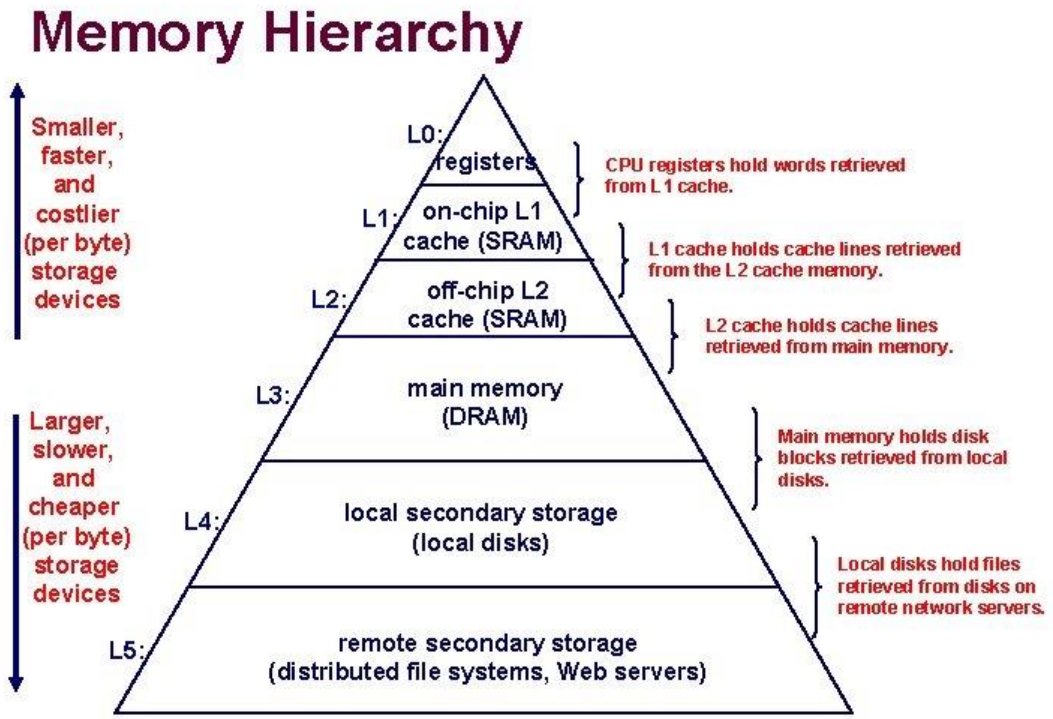
| Address (5 digits in hexadecimal) | Content(8 bits) |
|---|---|
| FFFFFH | 11001100 (machine code/data) |
| … | |
| 00000H | 00110101(machine code/data) |

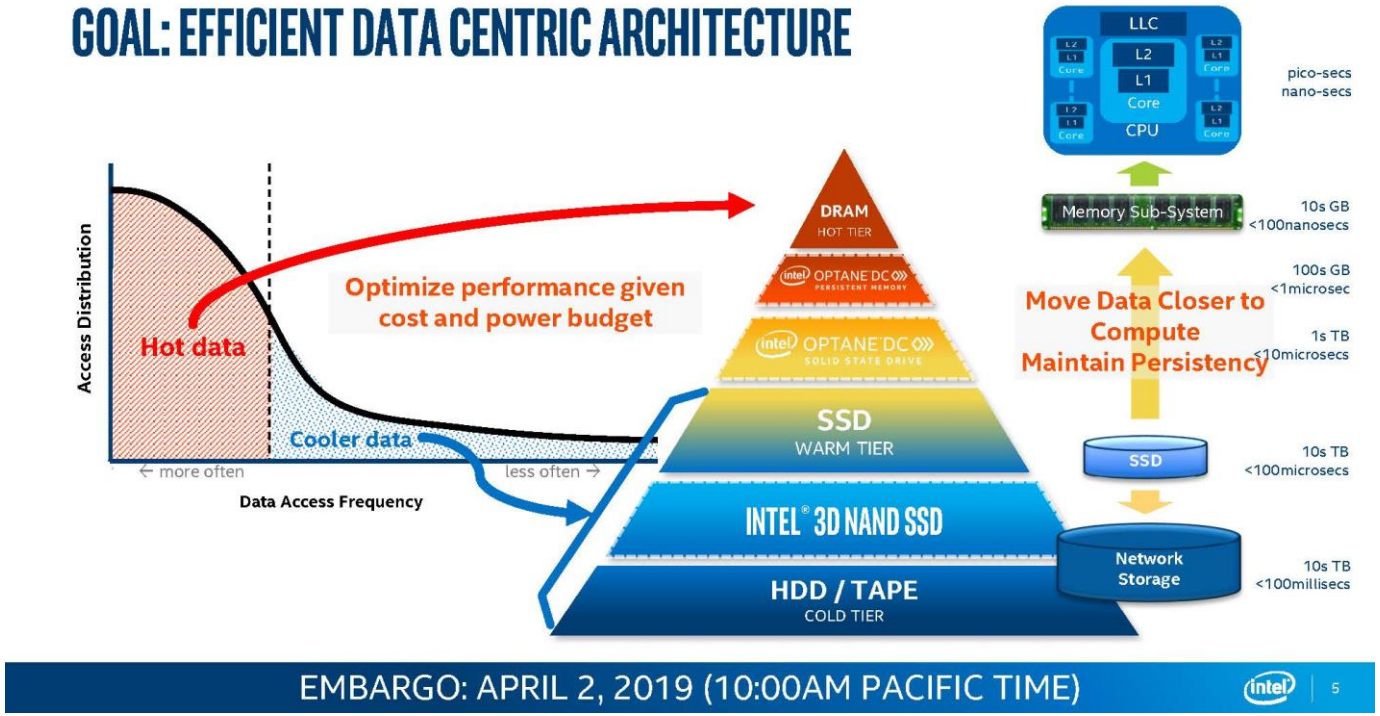
# Memory: Storage

## How a program is stored in Main Memory?

| Address of Main Memory | Contents |
|---|---|
| **500….H** | **Instruction-N (Machine Code)** |
| | …. |
| 50002H | Instruction-3(Machine Code) |
| 50001H | Instruction-2(Machine Code) |
| 50000H | Instruction-1 (Machine Code) |
| | |
| 300…H | Data-N (in binary) |
| … | … |
| 30001H | Data-1 (in binary) |
| 30000H | Data-1 (in binary) |

# Memory Hierarchy



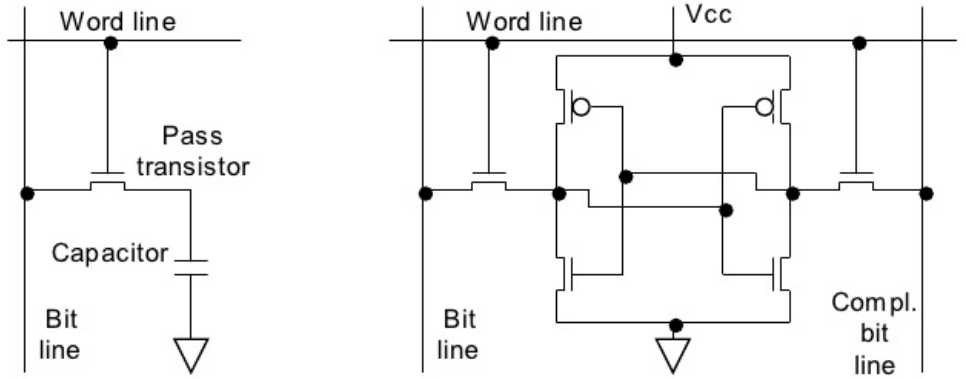



# Memory: RAM vs ROM

| | RAM | ROM |
|---|---|---|
| DEFINITION | a form of data storage that can be accessed randomly at any time, in any order and from any physical location. | a form of data storage that can not be easily altered or reprogrammed. |
| STANDS FOR | Random Access Memory | Read-only memory |
| USE | read data quickly to run applications. It allows reading and writing. | stores the program required to initially boot the computer. It only allows reading. |
| VOLATILITY | volatile (contents are lost when the device is powered off). | non-volatile (contents are retained even when the device is powered off). |
| TYPES | static RAM and dynamic RAM. | PROM, EPROM and EEPROM. |

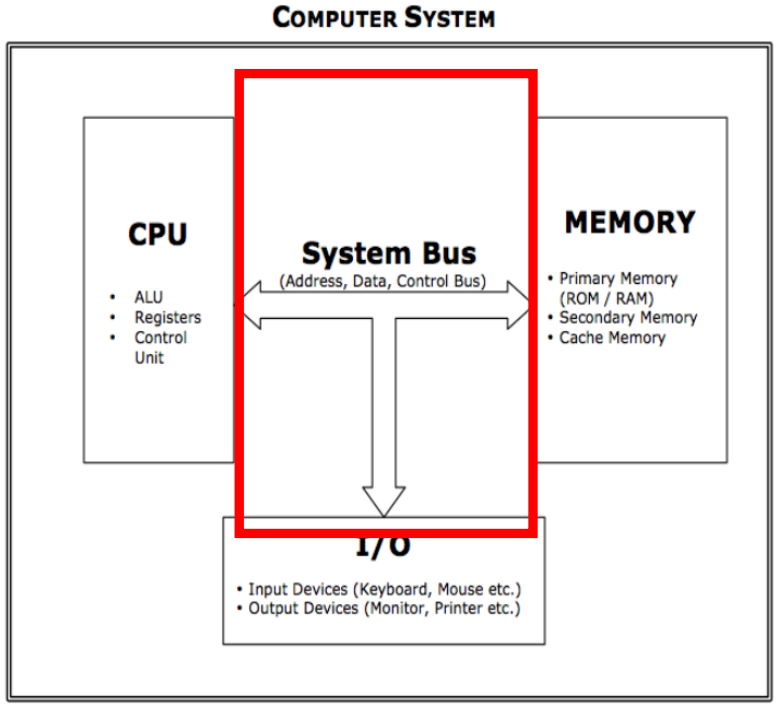
# Memory: SRAM vs DRAM

| SRAM | DRAM |
|---|---|
| Stores data till the power is supplied | Stores data only for few milliseconds even when power is supplied |
| Uses an arry of 6 transistors for each memory cell | Uses a single transistor and capaitor for each memory cell |
| Does not refreshes the memory Cell | Needs to refresh the memory cell after each reading of the capacitor |
| Data access is faster | Data acces is slower |
| Consume more power | Consume less power |
| Low density/less memory per chip | High density/more memory per chip |
| Cost per bit is high | Cost per bit is low |




(a) DRAM cell (b) Typical SRAM cell

# System Bus



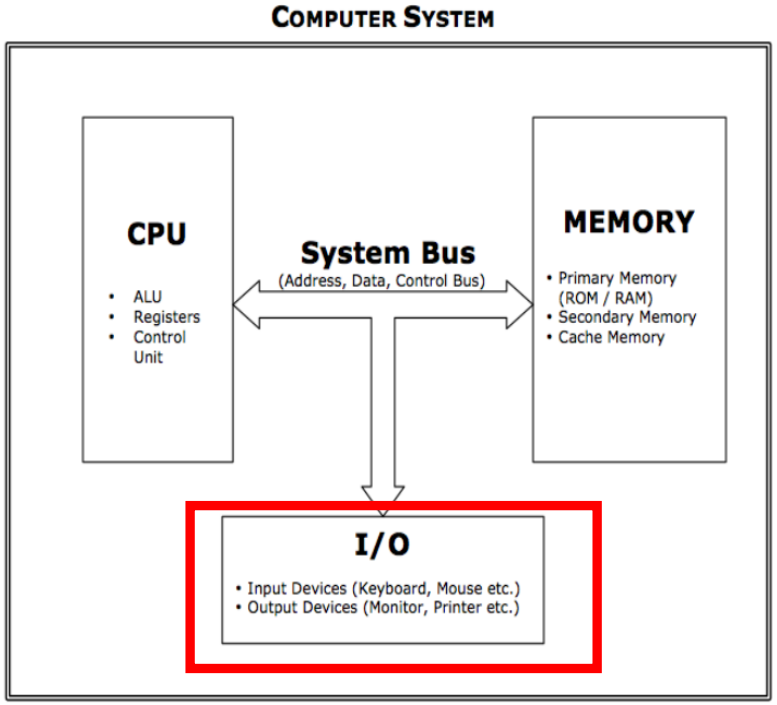

**System Bus**

1) A bus is a set of **interconnecting lines used to carry information**.
2) **Size** of a bus means its **number of lines**.
3) An 8-bit bus has eight lines carrying **one bit each**.
4) There are three types of buses.
5) **Address Bus: It carries the address for the operation.**
   During any operation, the address bus identifies the location where the operation is performed.
   The size of the address bus determines the amount of Primary Memory that can be connected.
   Example: If address bus is 16-bit, we can connect $2^{16}$ = 64 KB Memory.
   Bigger the address bus, bigger is the memory.
6) **Data Bus: It carries data to and from the processor.**
   The size of data bus determines how much data can be transferred in one operation (cycle).
   Bigger the data bus, faster the processor, as it can transfer more data in one cycle.
7) **Control Bus: It Carries control signals like RD, WR etc.**
   These signals determine the kind of operation that will be performed on the system bus.

# System Bus

- **Address Bus**
  - Carries the location in memory of a given item
  - Uni-directional (always supplied by the CPU)
  - Determines maximum amount of memory available to CPU
- **Data bus**
  - Carries data between CPU and memory or I/O devices
  - Bi-directional
  - Determines the width of the architecture
- **Control Bus**
  - Carries timing signals (and more) to synchronize CPU to external circuitry
  - Highly dependent on the specific CPU

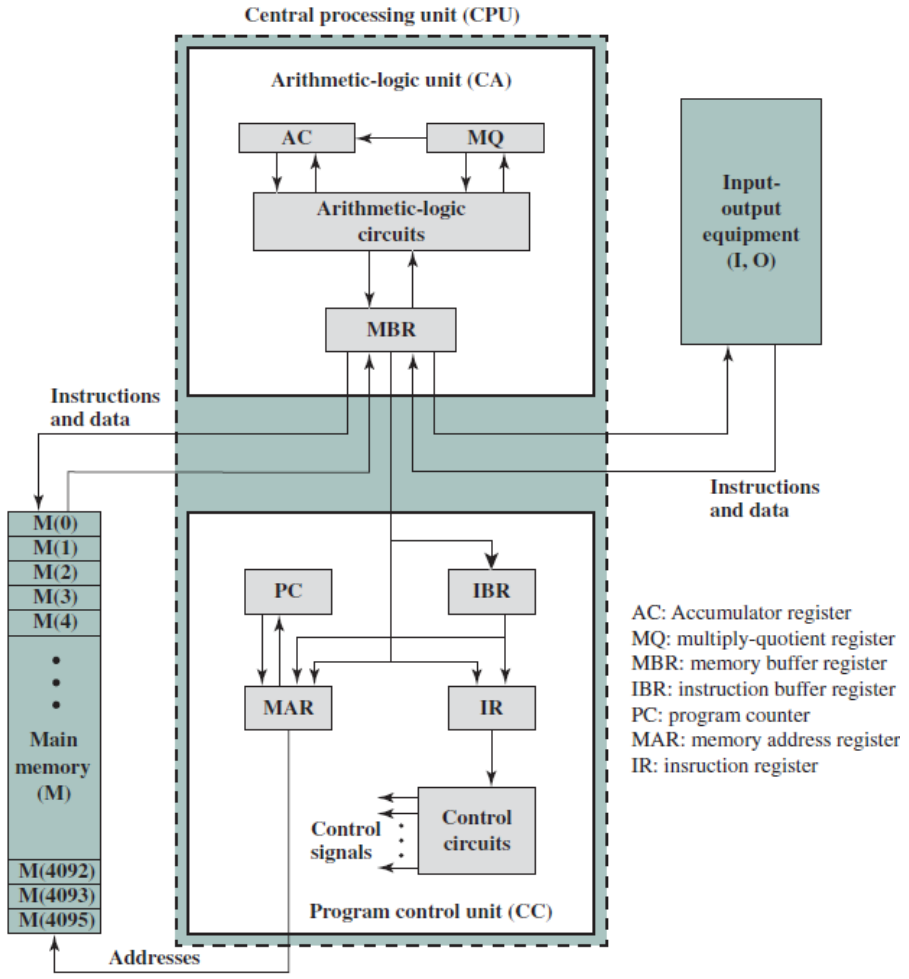
# Input/Output (I/O) Devices




## I/O

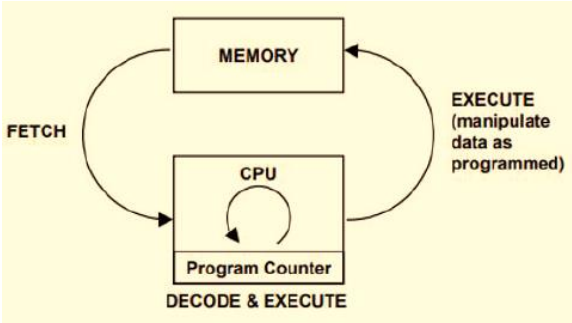
1) I/O devices are used for the **flow of information in and out** of the computer system.
2) **Input** devices such as **keyboard, mouse**, etc. are used to provide inputs into the computer.
3) They are used to **enter programs and data**.
4) **Output** devices such as **monitor and printer** are used to **generate results**.
5) Some devices such as a **touch-screen** can be used for **both input and output**.

# The IAS Structure




***With rare exceptions, all of today's computers have this same general structure and function and are thus referred to as von Neumann machines.***

***Registers*** *are small amounts of high-speed electronic storage contained within the* ***CPU****. They are used by the* ***processor*** *to store small amounts of data that are needed during processing.*

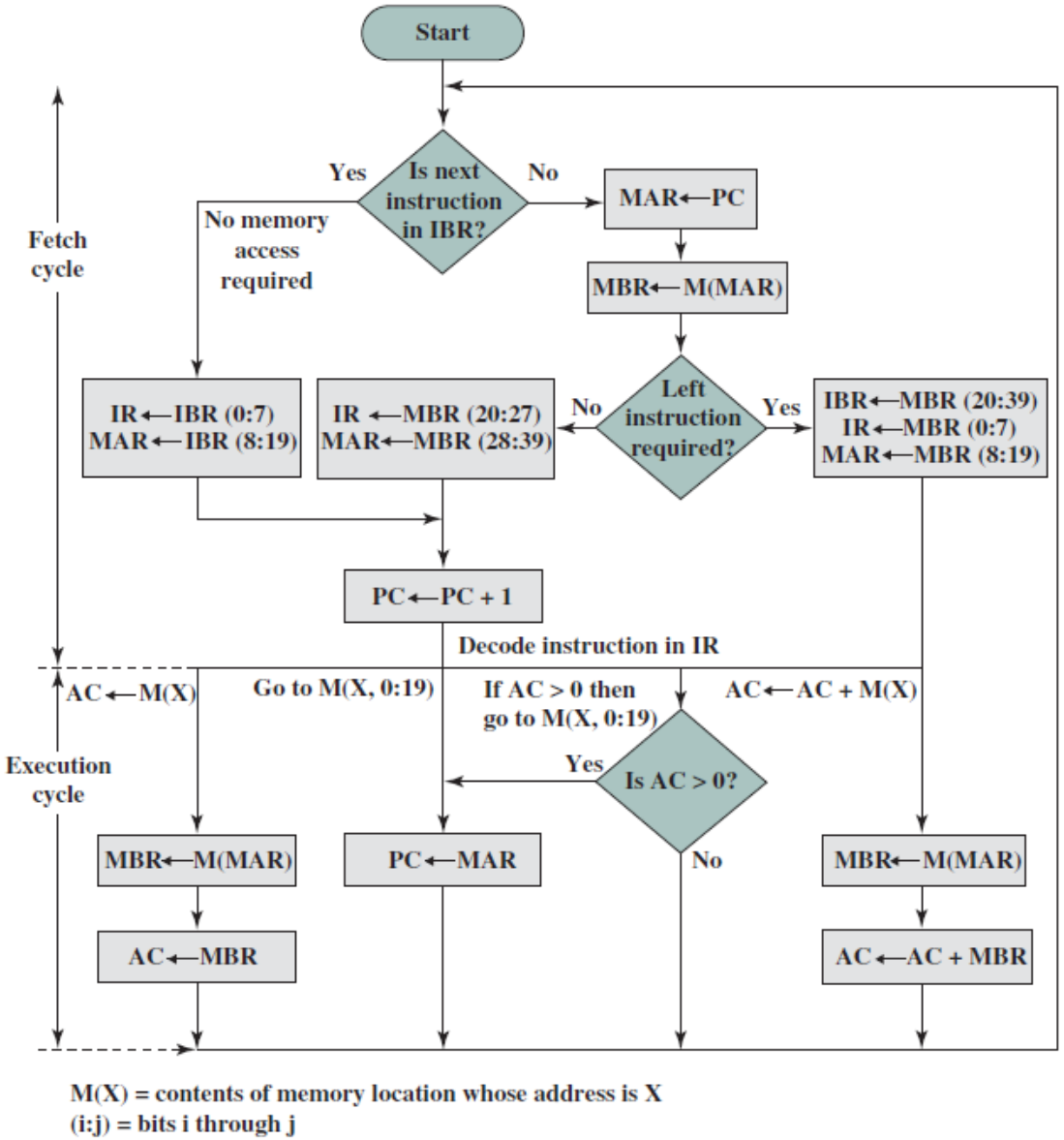



The control unit operates the IAS by fetching instructions from memory and executing them one at a time. We explain these operations

This figure reveals that both the control unit and the ALU contain storage locations, called *registers*, defined as follows:

- **Memory buffer register (MBR):** Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
- **Memory address register (MAR):** Specifies the address in memory of the word to be written from or read into the MBR.
- **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.
- **Instruction buffer register (IBR):** Employed to hold temporarily the right-hand instruction from a word in memory.
- **Program counter (PC):** Contains the address of the next instruction pair to be fetched from memory.
- **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations.

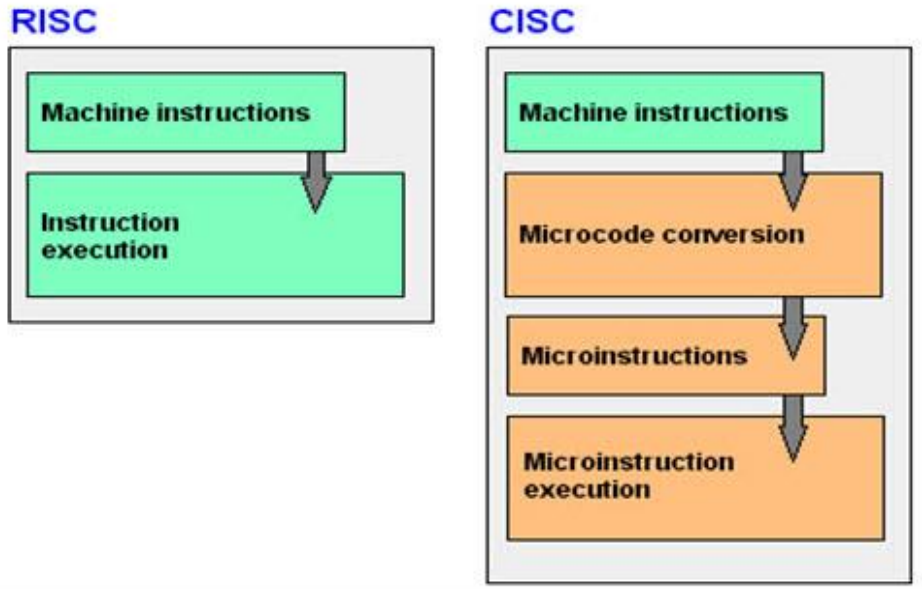
# Instruction Cycle of IAS




The IAS operates by repetitively performing an *instruction cycle*, as shown in Figure. Each instruction cycle consists of two subcycles. During the *fetch cycle*, the opcode of the next instruction is loaded into the IR and the address portion is loaded into the MAR. This instruction may be taken from the IBR, or it can be obtained from memory by loading a word into the MBR, and then down to the IBR,

# Instruction Set Architecture: RISC vs CISC

| Parameter | RISC | CISC |
|---|---|---|
| Instruction types | Simple | Complex |
| Number of instructions | Reduced (30-40) | Extended (100-200) |
| Duration of an instruction | One cycle | More cycles (4-120) |
| Instruction format | Fixed | Variable |
| Instruction execution | In parallel (pipeline) | Sequential |
| Addressing modes | Simple | Complex |
| Instructions accessing the memory | Two: Load and Store | Almost all from the set |
| Register set | multiple | unique |
| Complexity | In compiler | In CPU (micro-program) |




**Reduced Instruction Set Computer (RISC)**
**Complex Instruction Set Computer (CISC)**

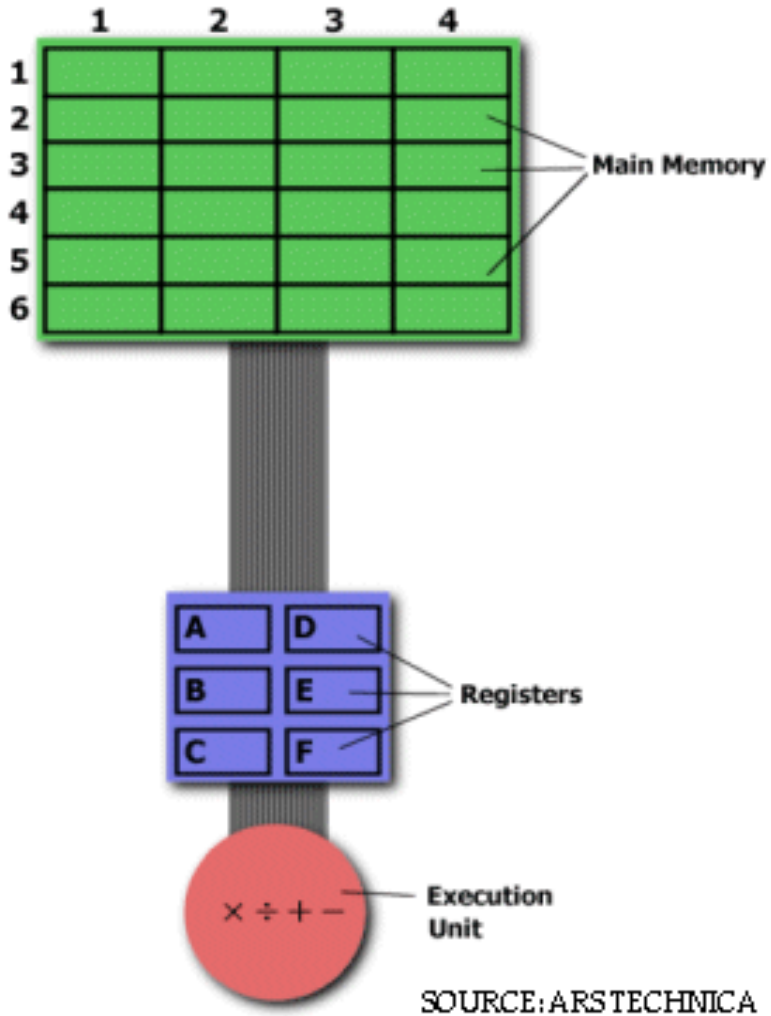
## The CISC Approach

The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT"). When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

MULT 2:3, 5:2

MULT is what is known as a "complex instruction." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions. It closely resembles a command in a higher level language. For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a * b."

One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly. Because the length of the code is relatively short, very little RAM is required to store instructions. The emphasis is put on building complex instructions directly into the hardware.



# The RISC Approach

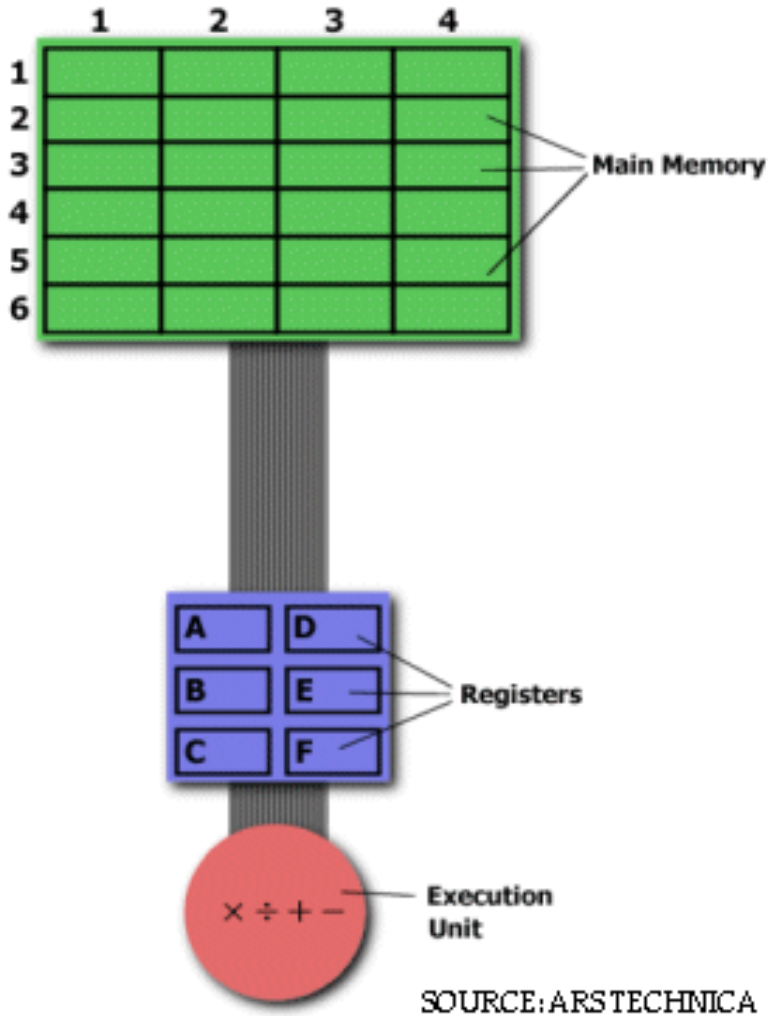
RISC processors only use simple instructions that can be executed within one clock cycle. Thus, the "MULT" command described above could be divided into three separate commands: "LOAD," which moves data from the memory bank to a register, "PROD," which finds the product of two operands located within the registers, and "STORE," which moves data from a register to the memory banks. In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

LOAD A, 2:3

LOAD B, 5:2

PROD A, B

STORE 2:3, A




At first, this may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement into code of this form.

# Instruction Set: 8086 Processor and IAS Machine

## 8086 Instruction Set Architecture

Instructions in 8086 can be of size 1 byte to 6 bytes.
The distribution of the bytes is as follows

| byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | opcode | | | | | | d | w | Opcode byte |
| 2 | mod | | reg | | | r/m | | | Addressing mode byte |
| 3 | [optional] | | | | | | | | low disp, addr, or data |
| 4 | [optional] | | | | | | | | high disp, addr, or data |
| 5 | [optional] | | | | | | | | low data |
| 6 | [optional] | | | | | | | | high data |

**Opcode Byte**

The first byte is called the "opcode byte".
It has a 6-bit opcode that indicates the operation to be performed.
It has two more bits "d" and "w"

**d: direction**
1 = data moves from operand specified by r/m to operand specified by reg.
0 = data moves from operand specified by reg to operand specified by r/m.

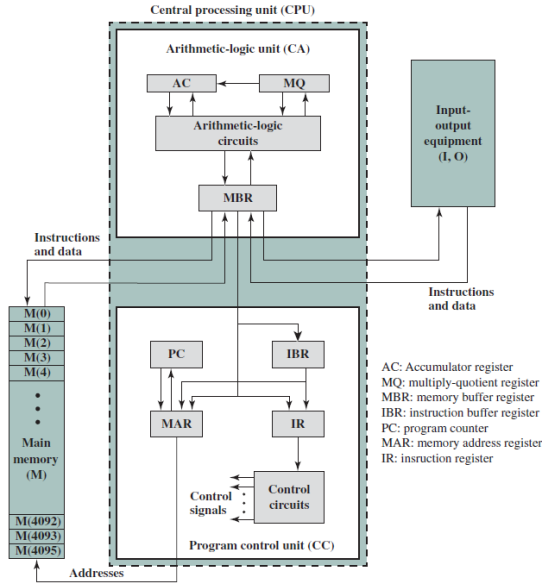
**w: word/ byte**
1: data is a word: 16-bits
0: data is a byte: 8-bits

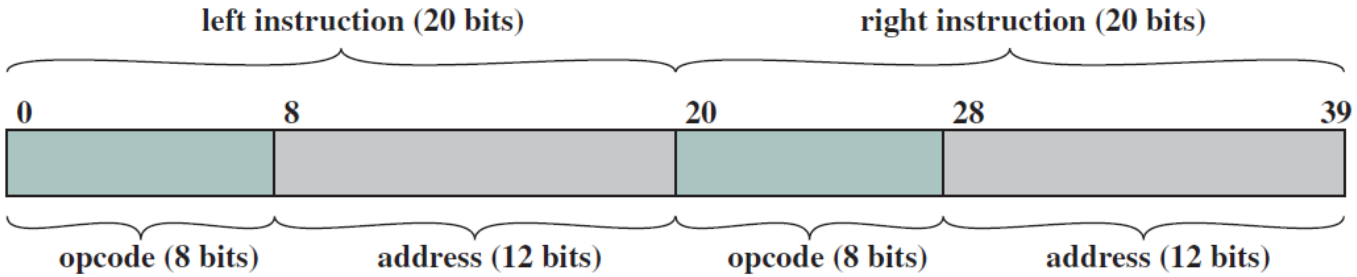
**Addressing Mode Byte**

**mod (2 bits):**
These are called "mode" bits. They decide how r/m is interpreted.

00: r/m is a memory operand, but no displacement
01: r/m is a memory operand, with 8-bit displacement
10: r/m is a memory operand, with 16-bit displacement
11: r/m is a register operand

## IAS Instruction Set Architecture




*the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC and the least significant in the MQ*

# The Instruction Format (8086)


Example: STD ; clear direction flag bit


Example: NOT AX ; complement content of AX


Example: MOV AX, BX; copy the contents of BX into AX

Example: Opcode of Instructions

| Operation | Mnemonic | Opcode |
|---|---|---|
| Addition | ADD | 001 |
| Subtraction | SUB | 010 |
| Data Transfer | MOV | 011 |
| Load Accumulator | LOAD | 100 |
| **Multiply** | **MUL** | **101** |
| Read from Input device | READ | 110 |
| Send to output device | STORE | 111 |

# The Instruction Format (8086)

## Machine Codes

- An instruction can be coded with 1 to 6 bytes
- Byte 1 contains three kinds of information
  - Opcode field (6 bits) specifies the operation (add, subtract, move)
  - Register Direction Bit (D bit) Tells the register operand in REG field in byte 2 is source or destination operand

    1: destination  0: source
  - Data Size Bit (W bit) Specifies whether the operation will be performed on 8-bit or 16-bit data

    0: 8 bits  1: 16 bits

| opcode | D | W | MOD | REG | R/M |
|---|---|---|---|---|---|

2-bit MOD field

| opcode | D | W | MOD | REG | R/M |
|---|---|---|---|---|---|
| 15 ------- 10 | 9 | 8 | 7 ---- 6 | 5 ---- 3 | 2 ---- 0 |

| CODE | EXPLANATION |
|---|---|
| 00 | Memory Mode, no displacement follows* |
| 01 | Memory Mode, 8-bit displacement follows |
| 10 | Memory Mode, 16-bit displacement follows |
| 11 | Register Mode (no displacement) |

*Except when R/M = 110, then 16-bit displacement follows

- Byte 2 has three fields
  - Mode field (MOD)
  - Register field (REG) used to identify the register for the first operand
  - Register/memory field (R/M)

| REG | W = 0 | W = 1 |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

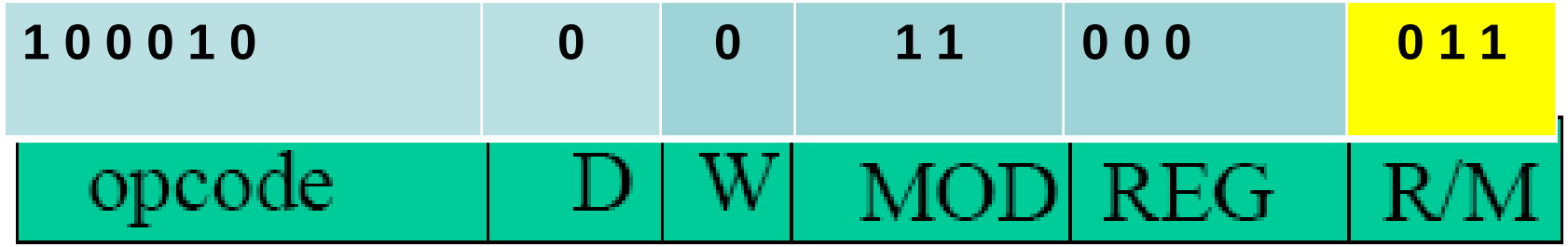
R/M field together specify the second operand

| opcode | D | W | MOD | REG | R/M |
|---|---|---|---|---|---|
| 15 ------- 10 | 9 | 8 | 7 ---- 6 | 5 ---- 3 | 2 ---- 0 |

| MOD = 11 | | | EFFECTIVE ADDRESS CALCULATION | | | |
|---|---|---|---|---|---|---|
| R/M | W = 0 | W = 1 | R/M | MOD = 00 | MOD = 01 | MOD = 10 |
| 000 | AL | AX | 000 | (BX) + (SI) | (BX) + (SI) + D8 | (BX) + (SI) + D16 |
| 001 | CL | CX | 001 | (BX) + (DI) | (BX) + (DI) + D8 | (BX) + (DI) + D16 |
| 010 | DL | DX | 010 | (BP) + (SI) | (BP) + (SI) + D8 | (BP) + (SI) + D16 |
| 011 | BL | BX | 011 | (BP) + (DI) | (BP) + (DI) + D8 | (BP) + (DI) + D16 |
| 100 | AH | SP | 100 | (SI) | (SI) + D8 | (SI) + D16 |
| 101 | CH | BP | 101 | (DI) | (DI) + D8 | (DI) + D16 |
| 110 | DH | SI | 110 | DIRECT ADDRESS | (BP) + D8 | (BP) + D16 |
| 111 | BH | DI | 111 | (BX) | (BX) + D8 | (BX) + D16 |

# The Instruction Format (8086)

**Example: MOV BL, AL**
**(machine code:** 10001000 11000011)
(Hexcode: 8 8 C 3**)**




Opcode = 100010 ⇨ MOV data transfer

D = 0 ⇨ AL is source operand

W = 0 ⇨ 8-bit data transfer

Therefore byte 1 is $10001000_2=88_{16}$

MOD = 11 ⇨ register mode

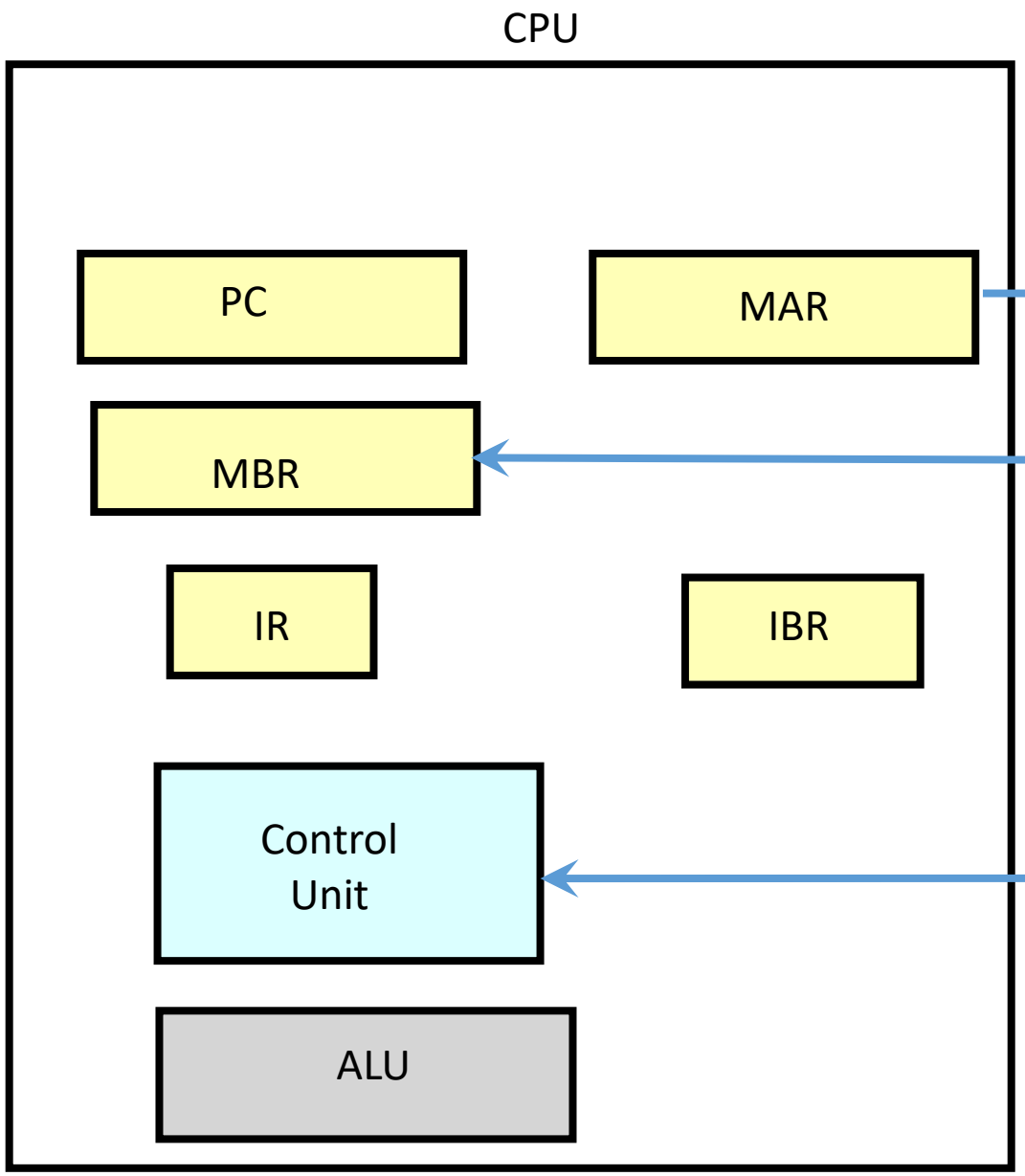
REG = 000 ⇨ code for AL

R/M = 011 ⇨ destination is BL

Therefore Byte 2 is $11000011_2=C3_{16}$

| Hex | Binary |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

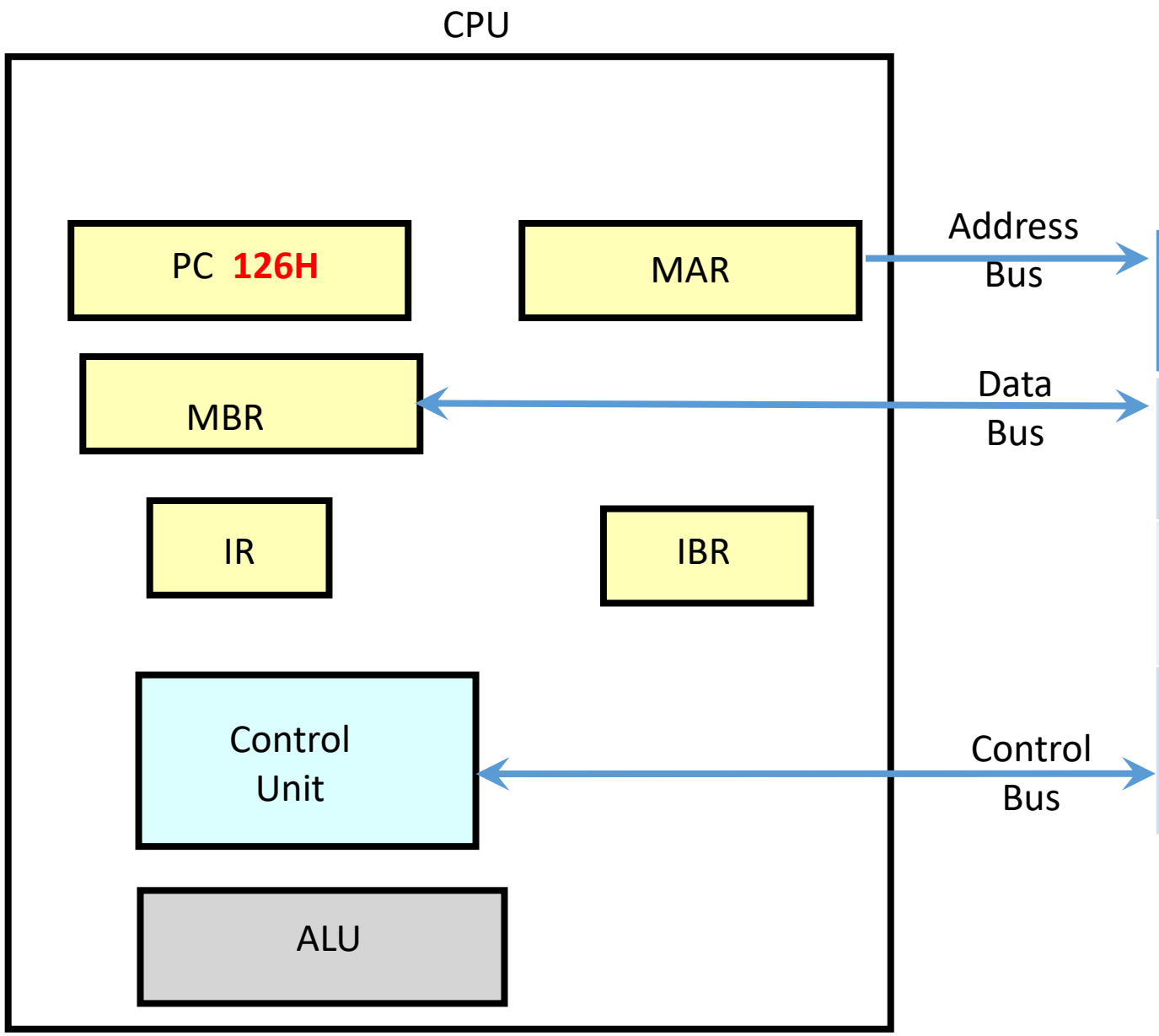
# How does IAS Machine work?




## Program in RAM(40 bits)

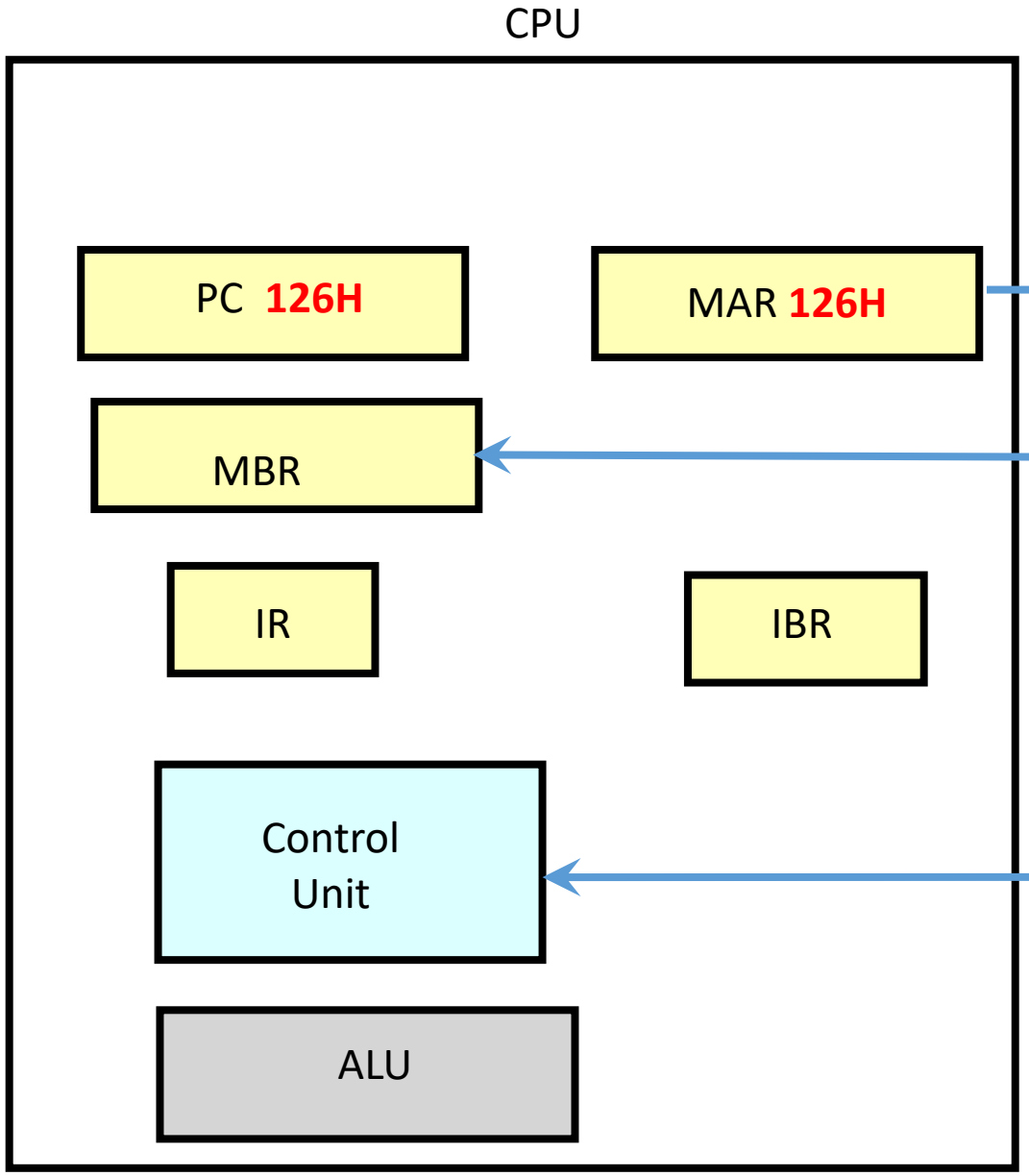
| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# STEP-1: PC is loaded with RAM address containing 1st & 2nd instructions of program




| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

STEP-2: MAR is loaded content of PC (with RAM address containing 1st & 2nd instructions of program)



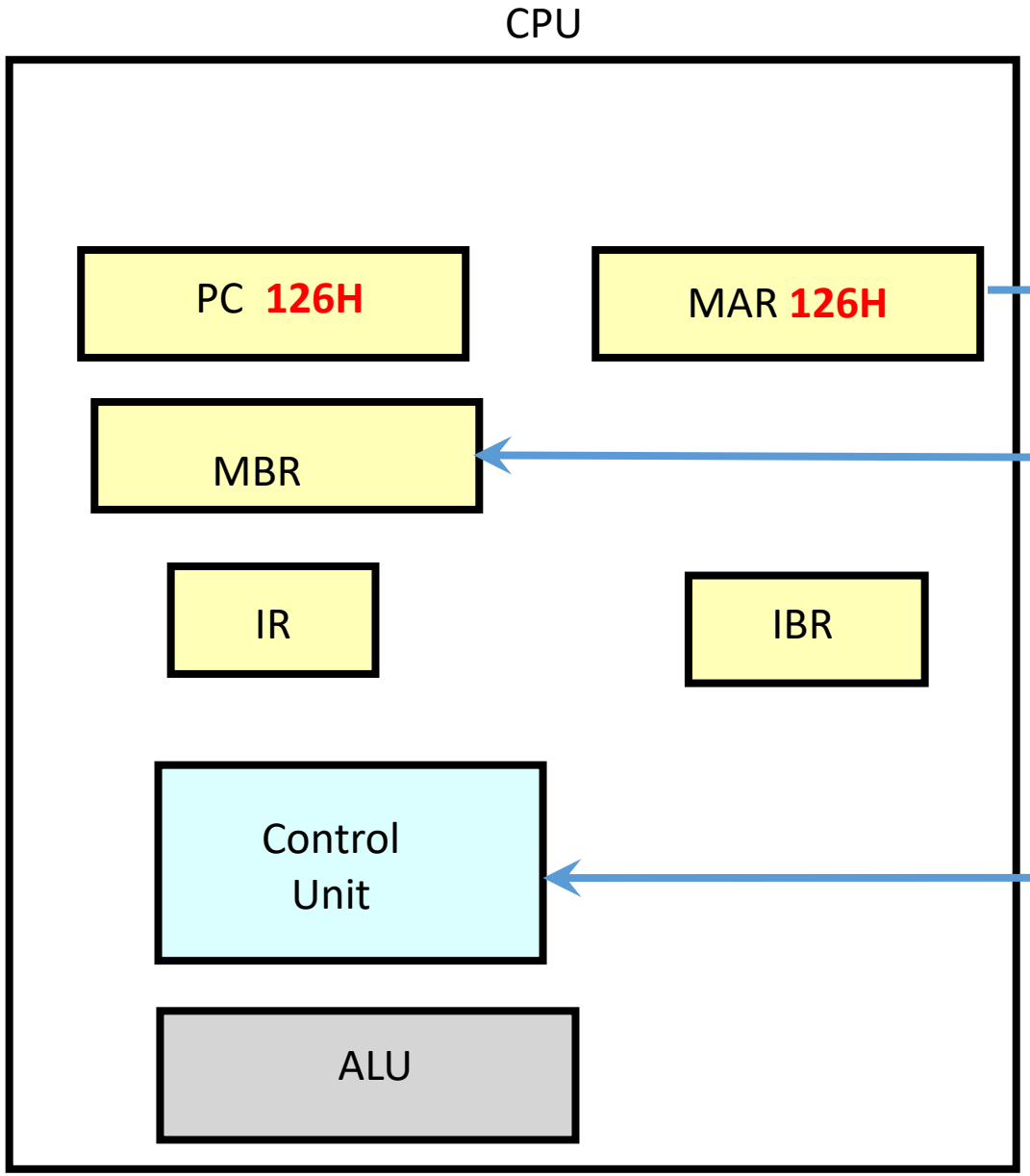

Address Bus

Data Bus

Control Bus

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# STEP-3: Address on Address bus, Memory location 126H selected



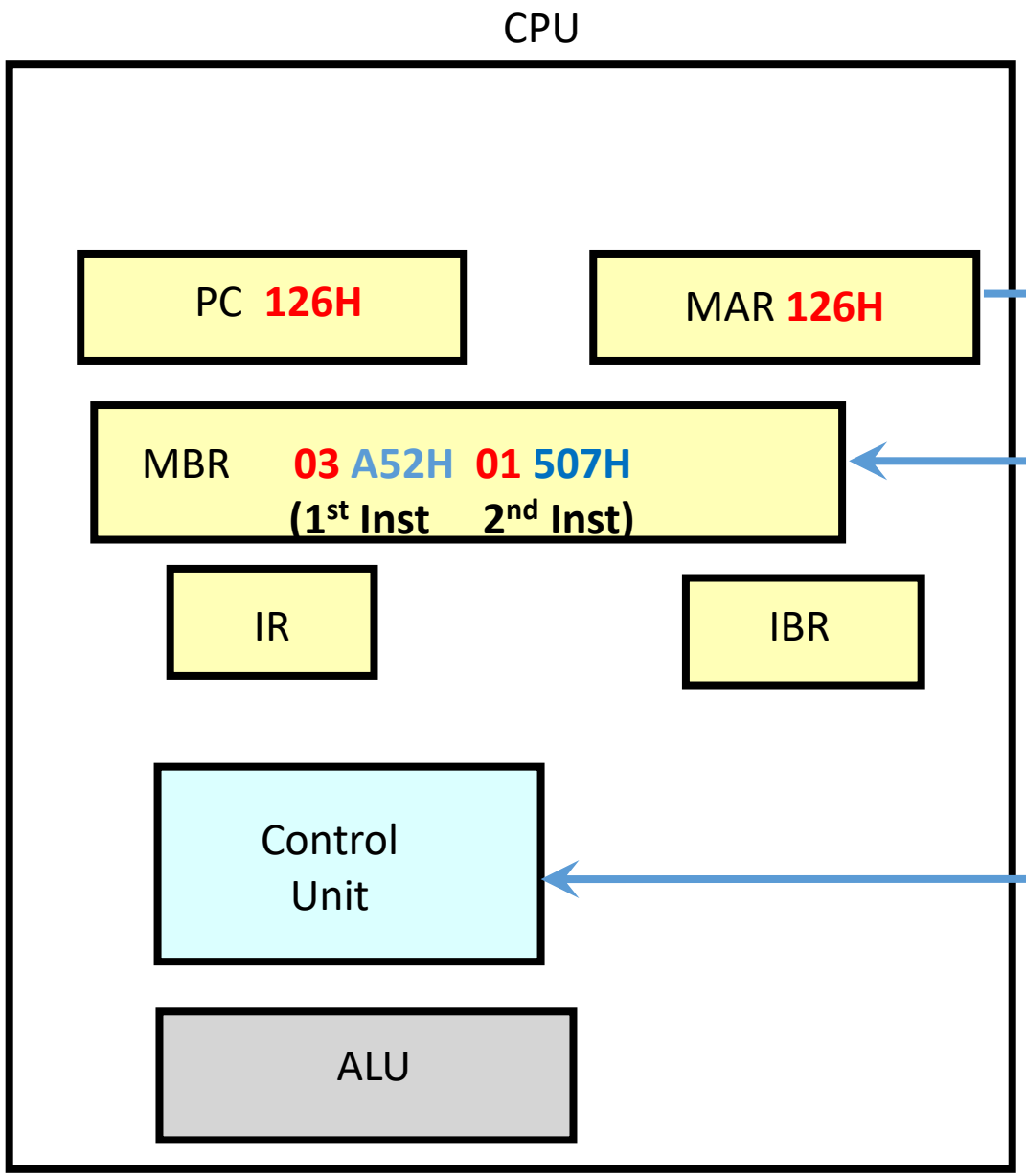

Address Bus 126H

Data Bus

Control Bus

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H (1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H (3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H (5th Inst 6th Inst) |

# STEP-4: Read Machine code of 1st and 2nd Instructions



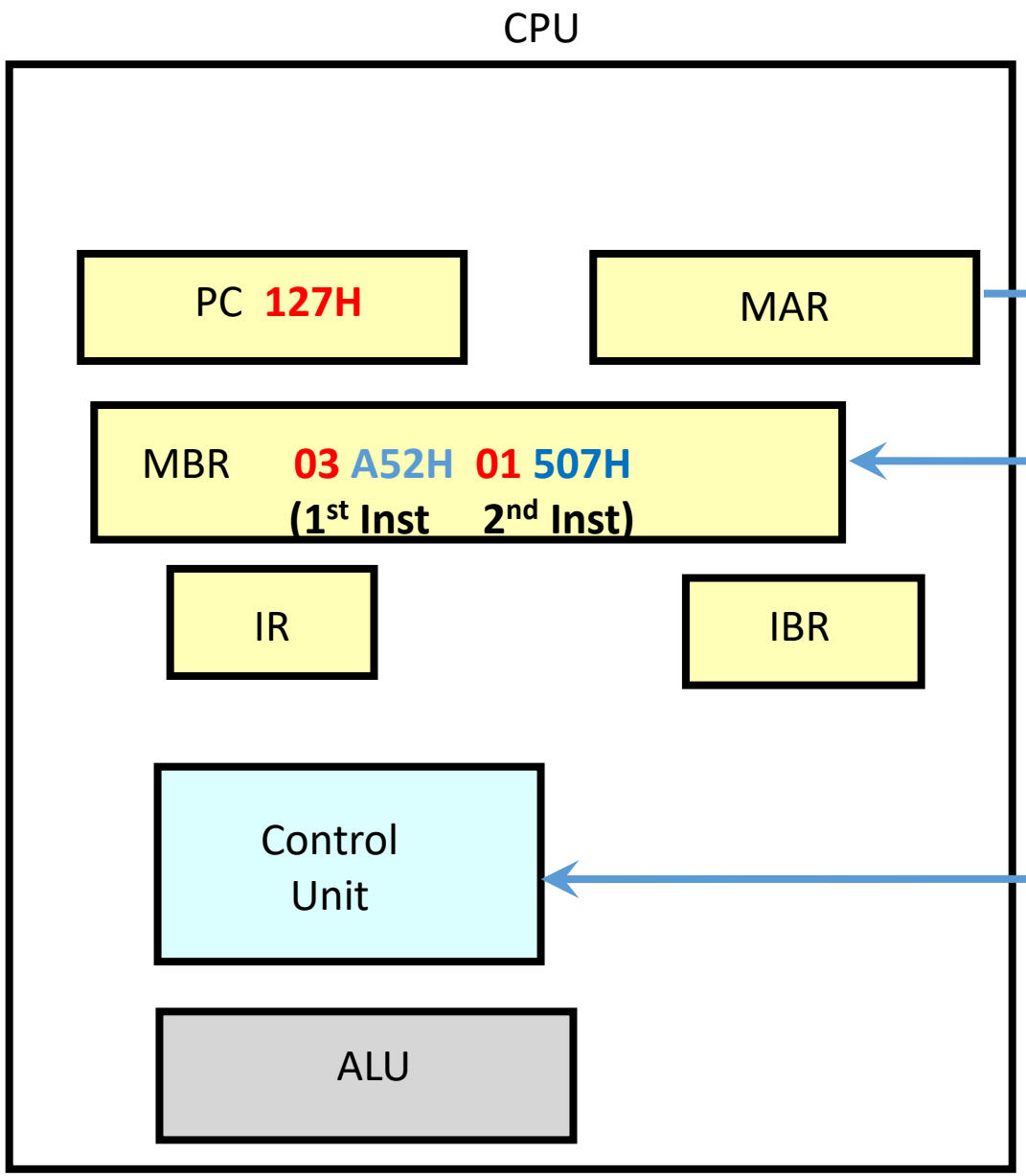

Address Bus 126H

Data Bus

Control Bus READ

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# STEP-5: PC is incremented to point next (3rd and 4th ) Instructions to be fetched



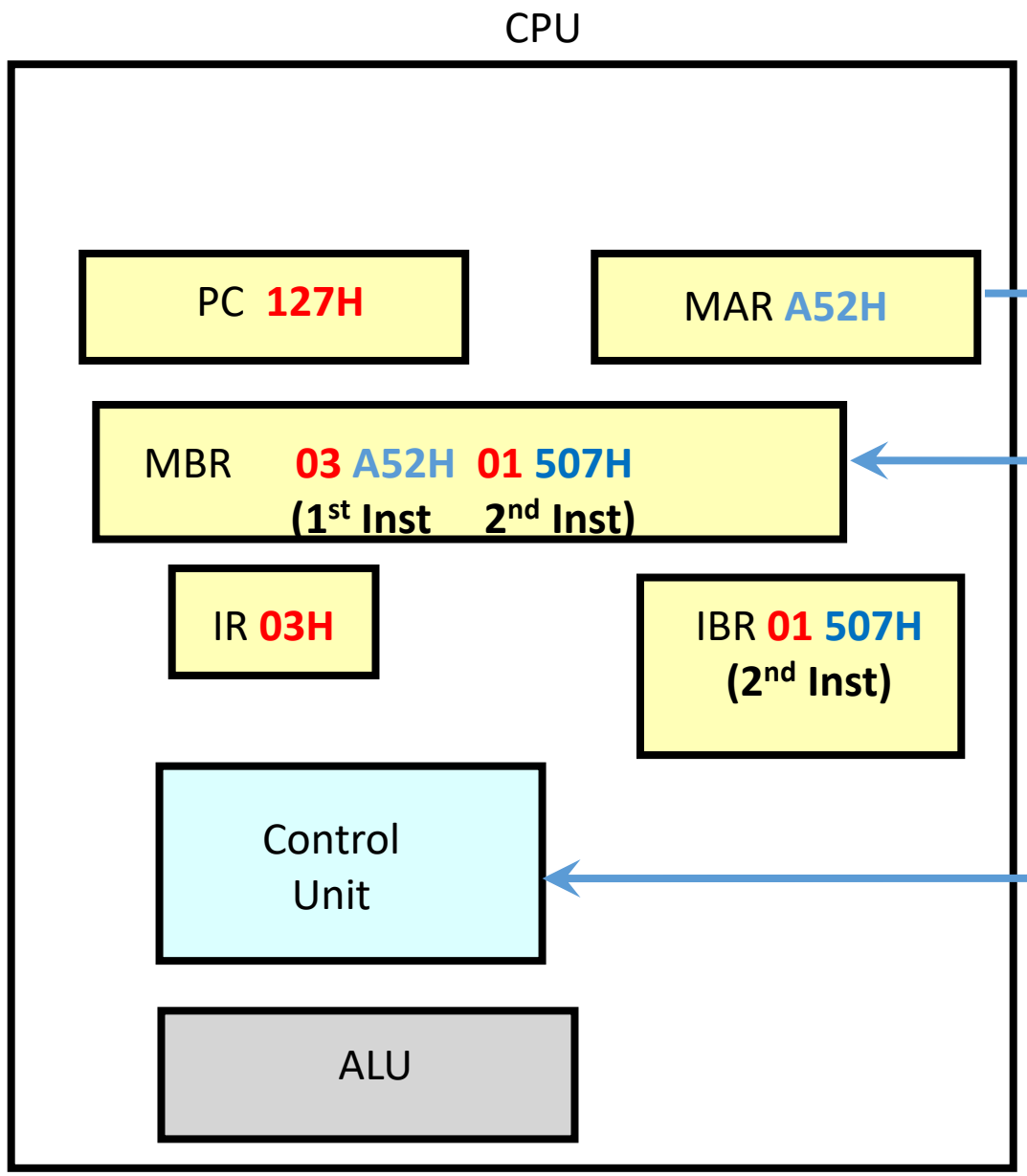

Address Bus

Data Bus

Control Bus

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# STEP-6: OPCODE of 1st Instructions into IR, Address part into MAR, 2nd Instruction into IBR



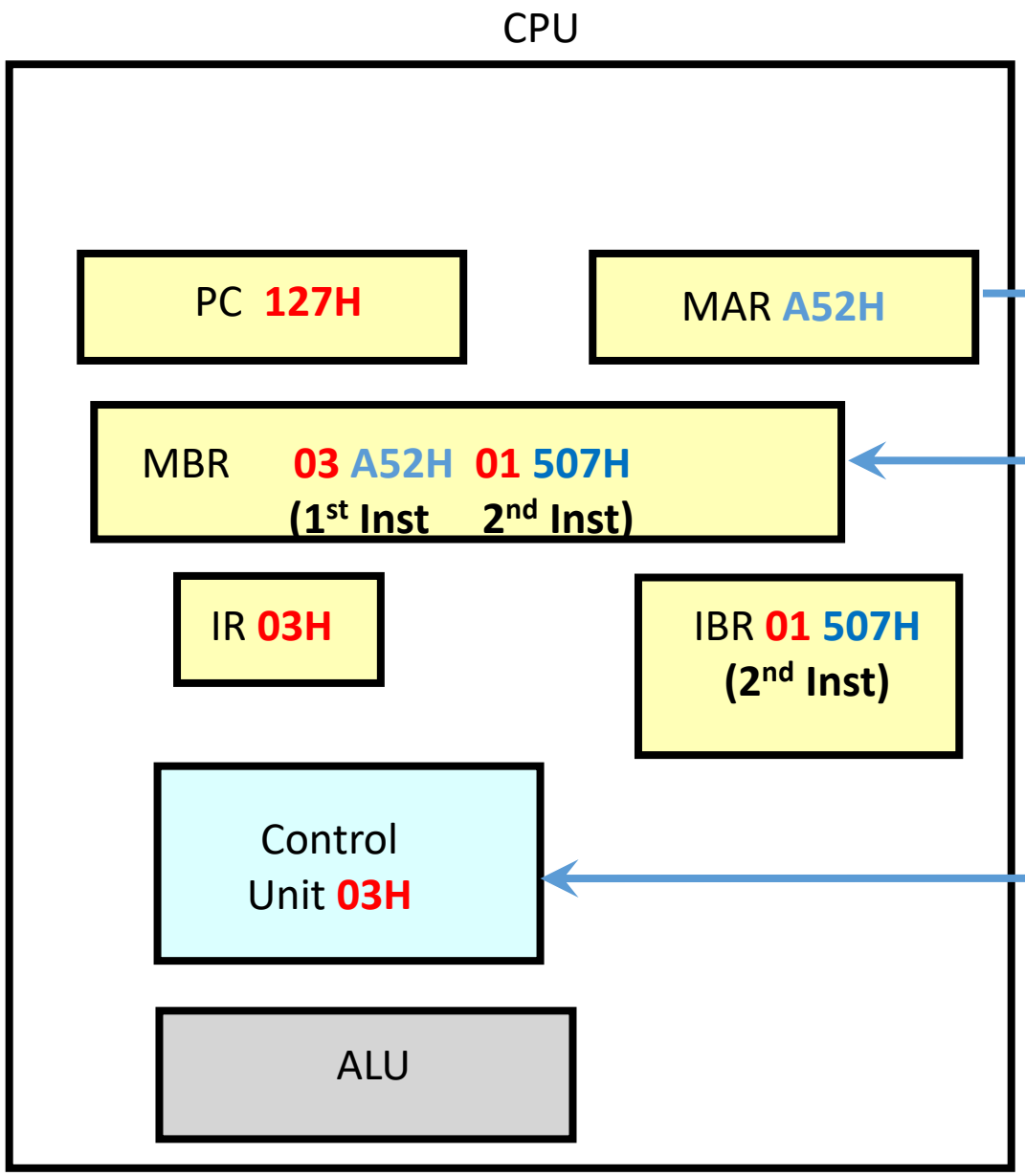

Address Bus 126H

Data Bus

Control Bus READ

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H (1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H (3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H (5th Inst 6th Inst) |

# STEP-7: 1st Instructions Decoded, Executed



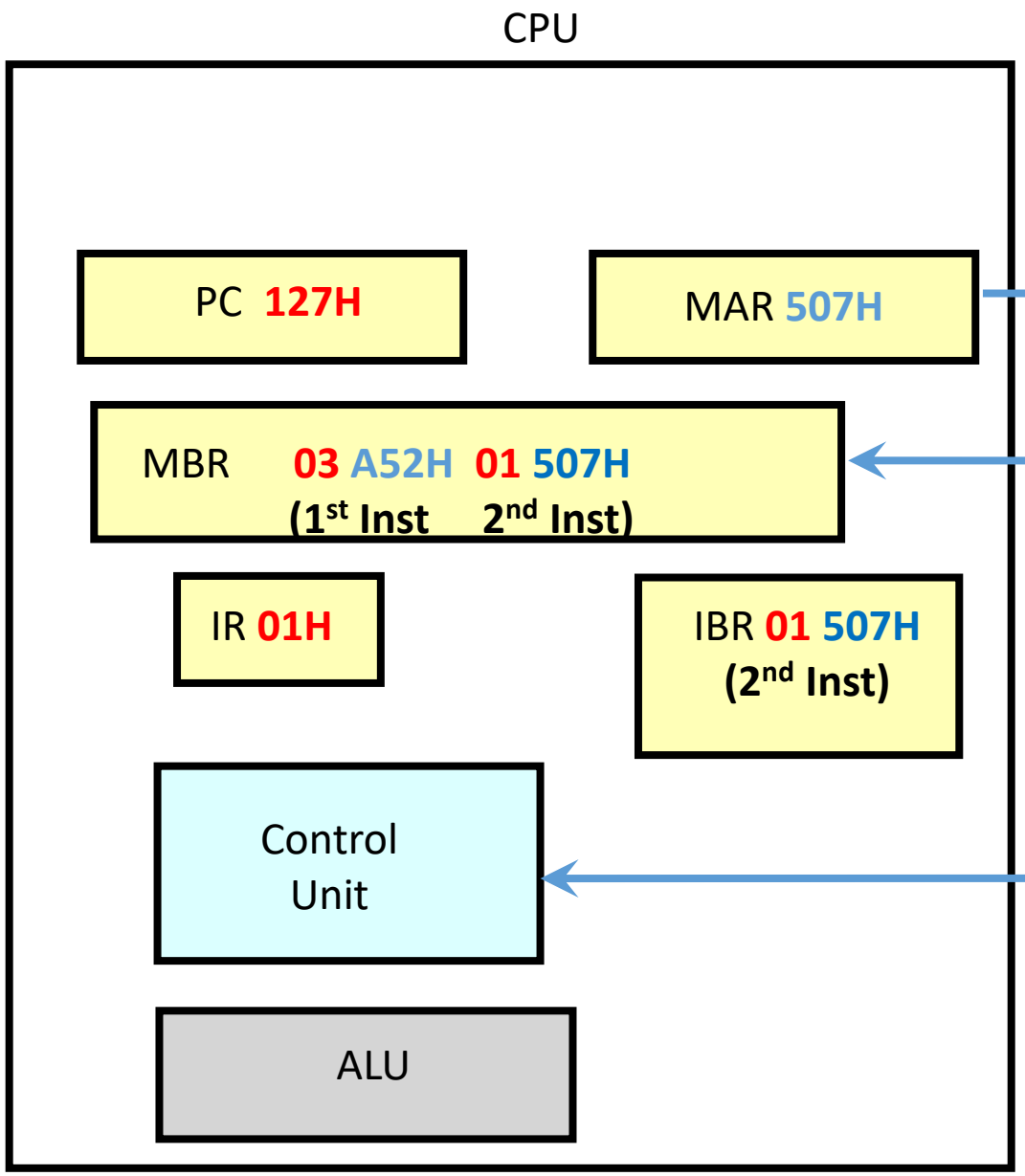

Address Bus A52H

Data Bus

Control Bus

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# STEP-8: 2nd Instructions processing begins: OPCODE into IR, Address into MAR



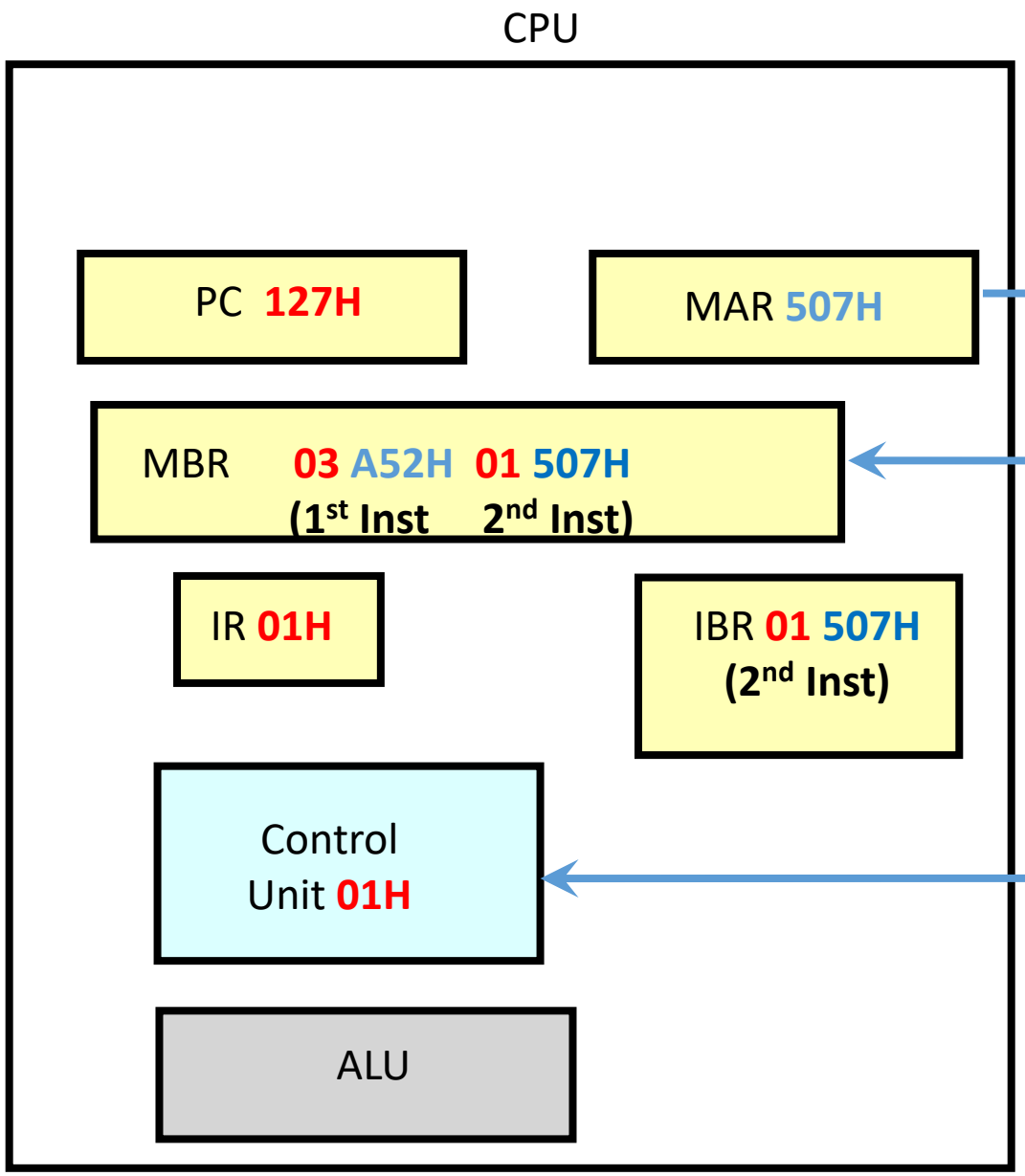

Address Bus 507H

Data Bus

Control Bus

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# STEP-9: 2nd Instructions Decoded, Executed



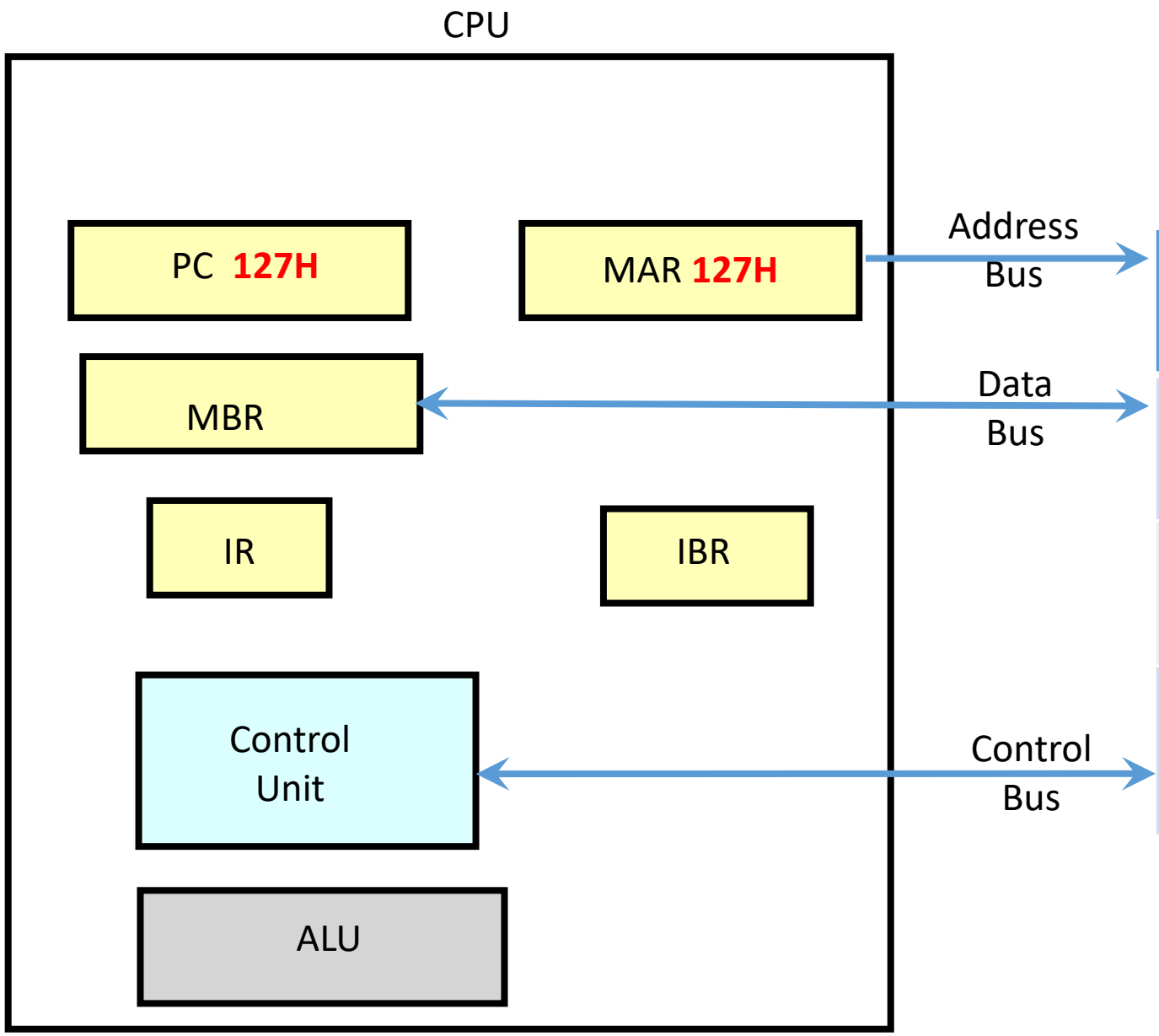

Address Bus 507H

Data Bus

Control Bus

| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

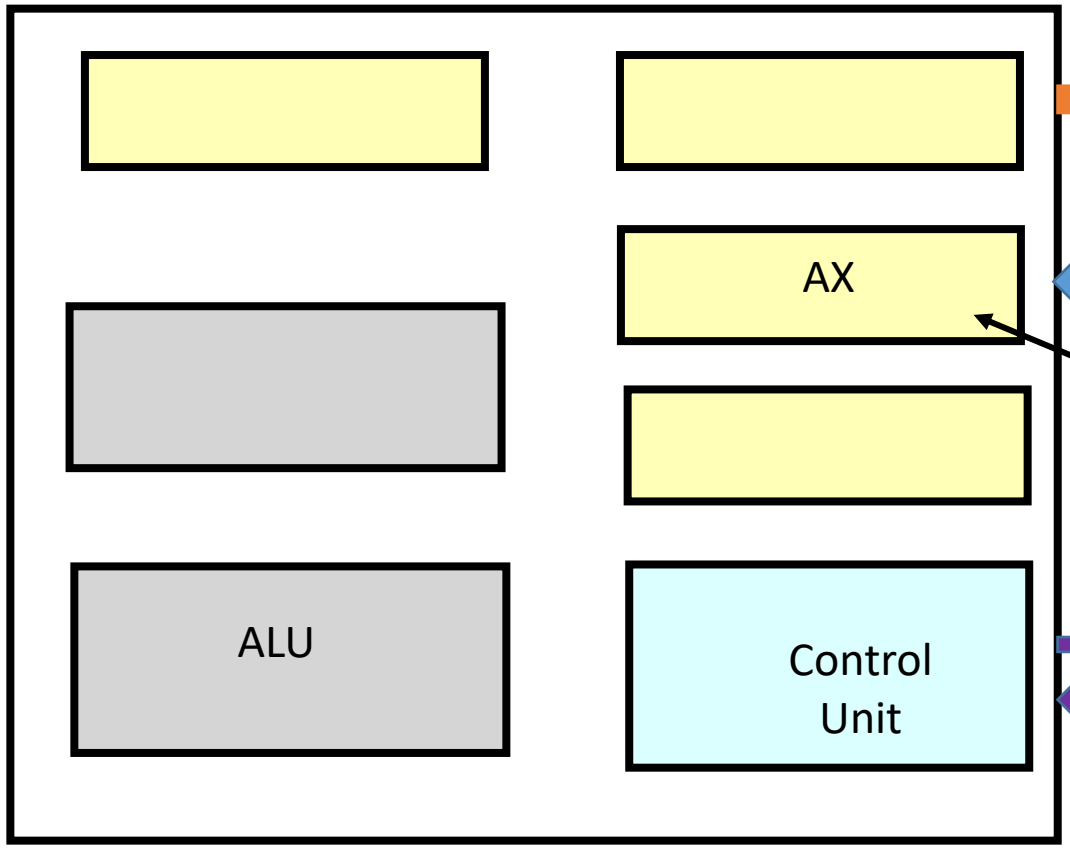
# STEPS: 2- 9 are repeated




| Address (12 bits) | Contents (40 bits) |
|---|---|
| 126H | 03 A52H 01 507H<br>(1st Inst 2nd Inst) |
| 127H | 02 A53H 04 508H<br>(3rd Inst 4th Inst) |
| 128H | 01 A54H 02 509H<br>(5th Inst 6th Inst) |

# General purpose

## CPU




Address Bus

Data Bus

Control Bus

## Memory

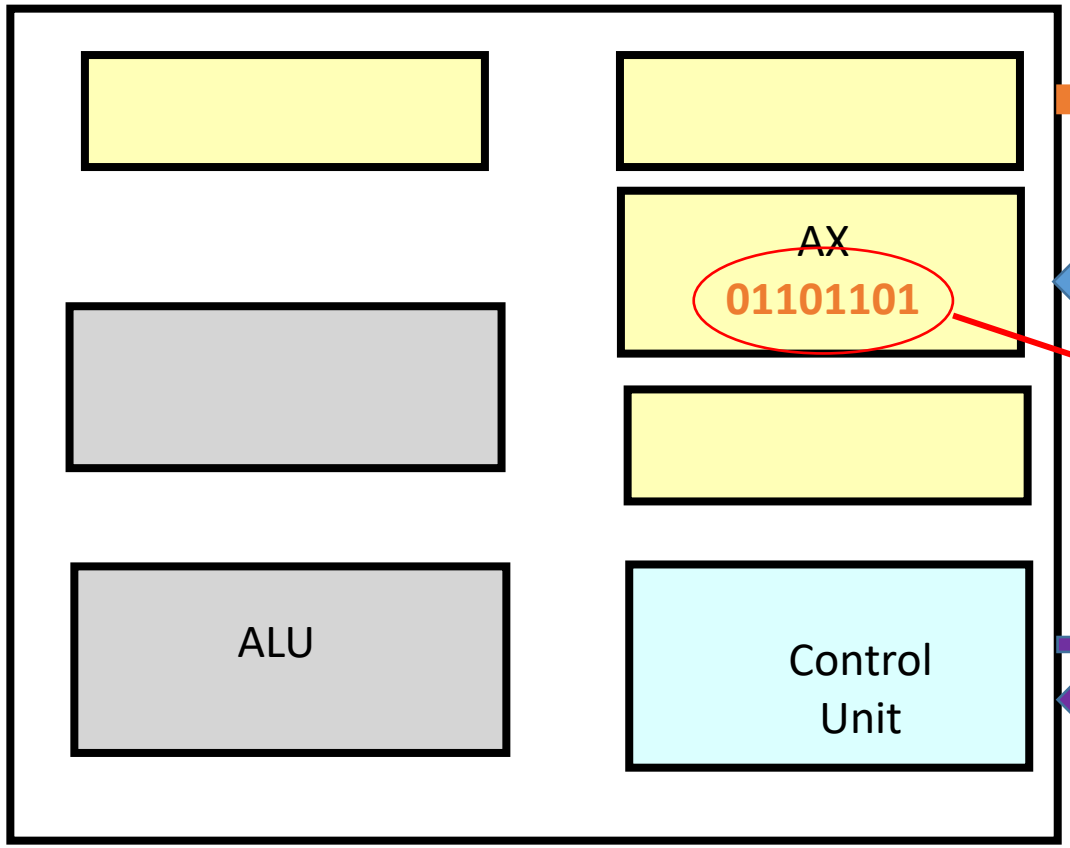
| ADDRESS | CONTENTS |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| **200H** | **11000101** (DATA-1) |
| **201H** | **11000001** (DATA-2) |
| **202H** | **11000111** (DATA-3) |

Example: **MOV AX, M1** ;M1= 200H

The content of memory location 200H is copied into AX register

# General purpose

## CPU




Address Bus

Data Bus

Control Bus

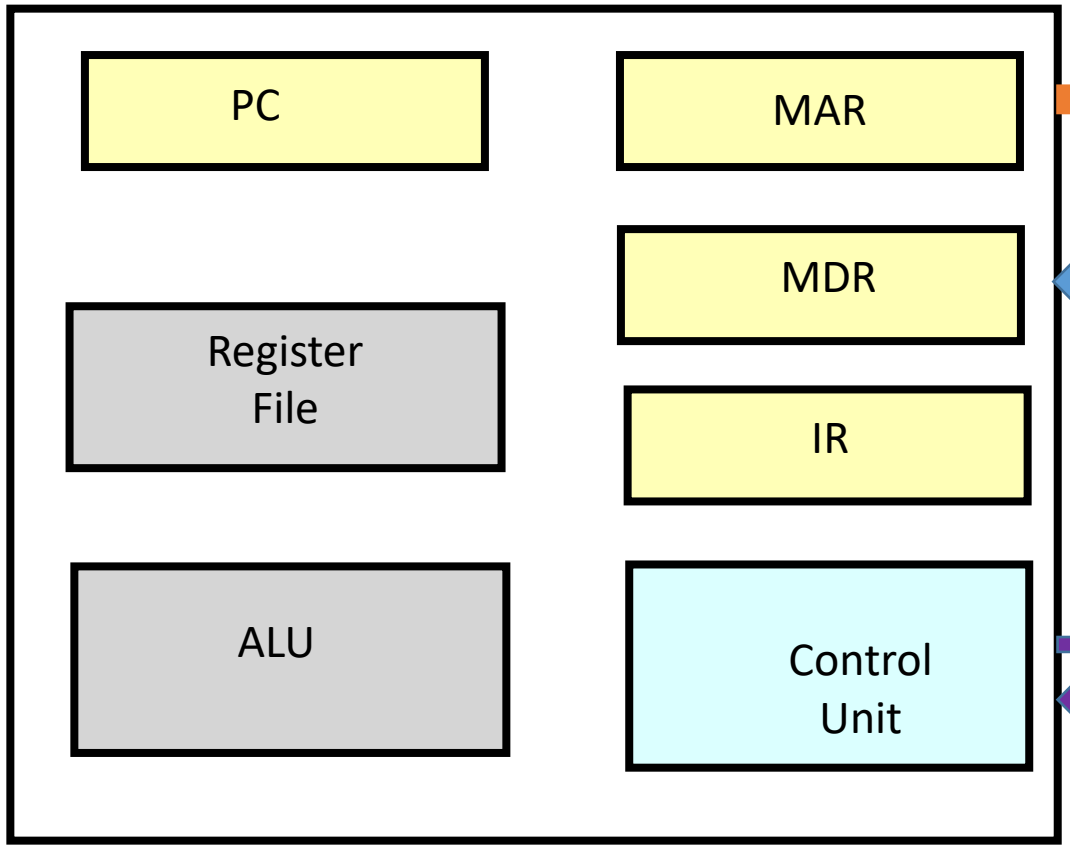
## Memory

| ADDRESS | CONTENTS |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| **200H** | **11000101** (DATA-1) |
| **201H** | ~~**11000001** (DATA-2)~~ **01101101** |
| **202H** | **11000111** (DATA-3) |

MOV M2, AX ;M2 = 201H
; [AX] = 01101101

# Computer Components

## CPU




Address Bus

Data Bus

Control Bus

## Memory

| ADDRESS | CONTENTS |
|---|---|
| 000100100101 (125H) | 11001101 (Machine code of Instruction-1) |
| 000100100110 (126H) | 10001101 (Machine code of Instruction-2) |
| 000100100111 (127H) | 11101101 (Machine code of Instruction-3) |
| 000100101000 (128H) | 11011101 (Machine code of Instruction-4) |
| | |
| | |
| 200H | 11000101 (DATA-1) |
| 201H | 11000001 (DATA-2) |
| 202H | 11000111 (DATA-3) |

PC-Program counter

MAR-Memory Address Register

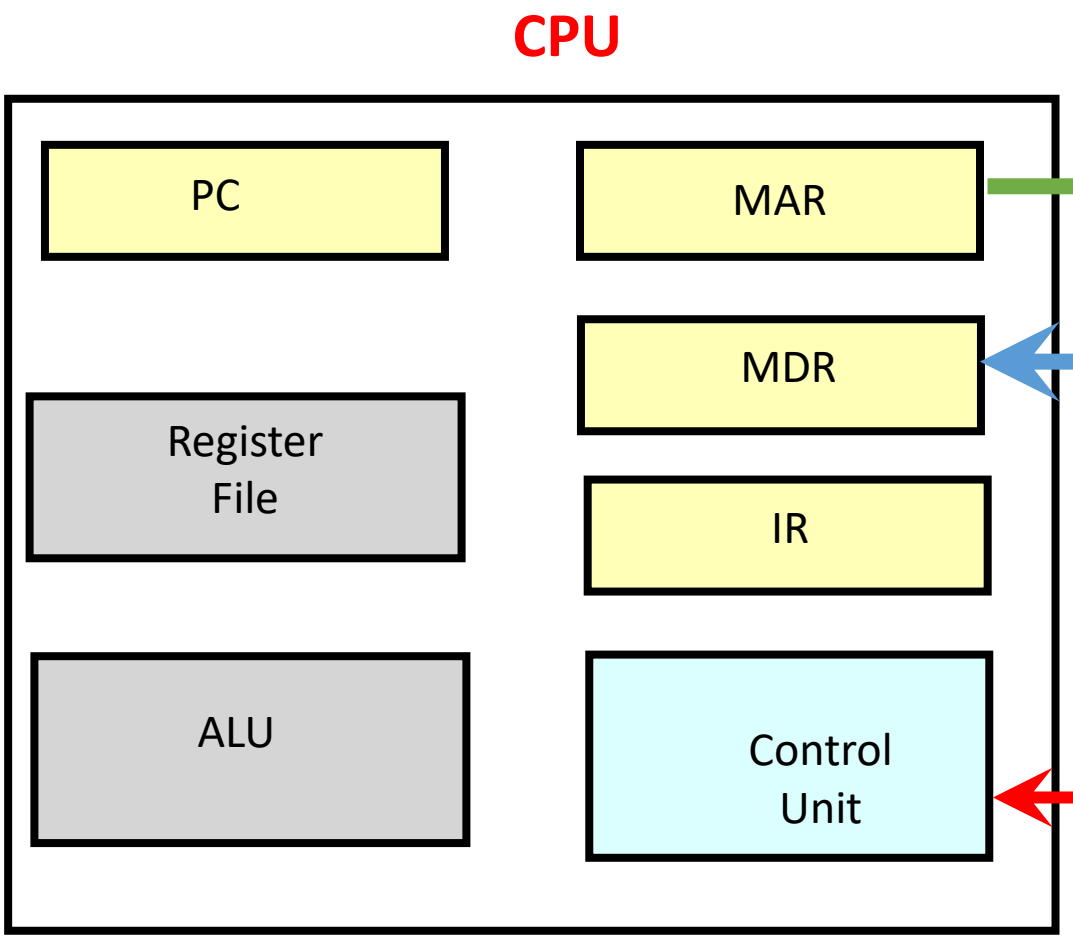
IR-Instruction Register

MDR-Memory Data Register

Control Unit decodes Instructions & Generates control signals

# How does computer work?

| User program |
| --- |
| SUB AX, BX ;0010101**11000011** |
| MOV CX, AX ;1000101**11001000** |
| MOV DX, 0 |
| |




Address Bus

Data Bus

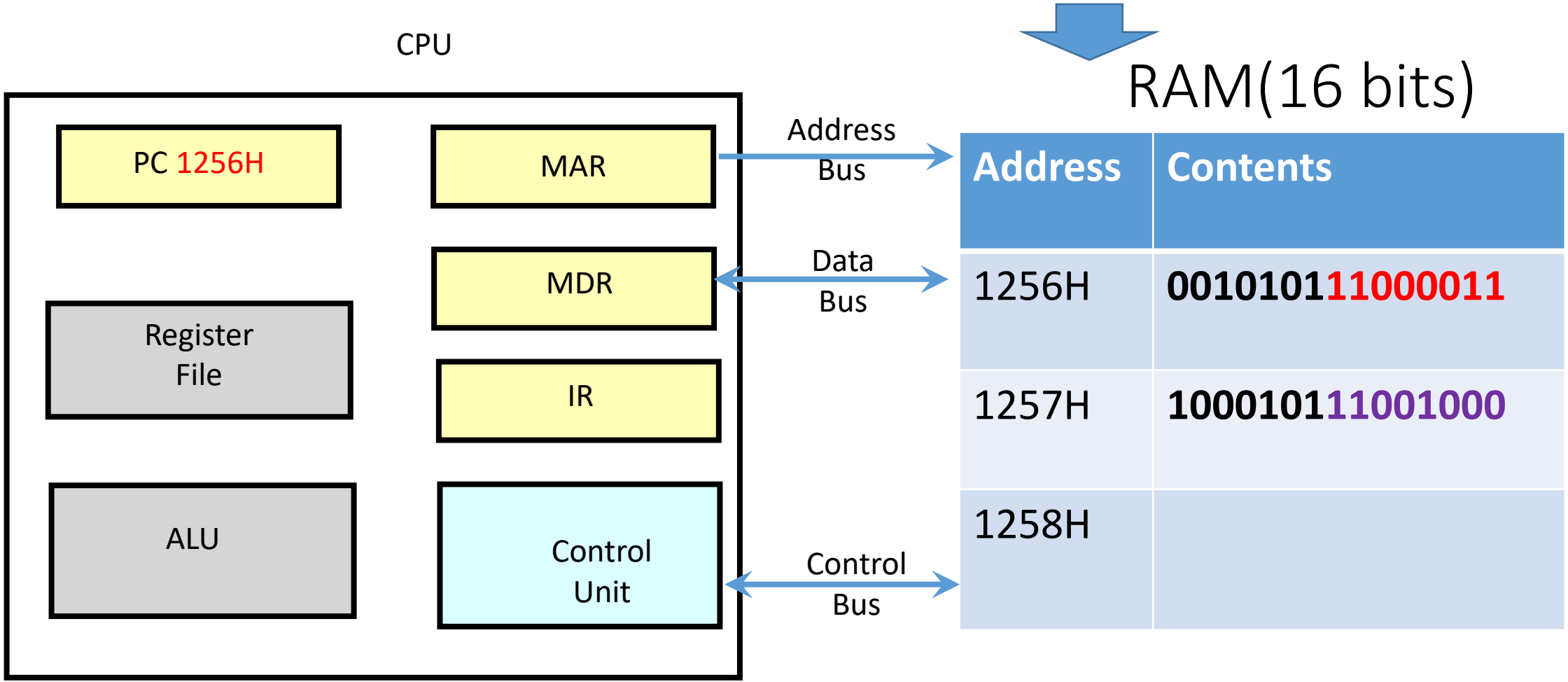
Control Bus

## Stored in RAM(16 bits)

| Address | Contents |
| --- | --- |
| 0001001001010110 (1256H) | **0010101****11000011** |
| 0001001001010111 (1257H) | **1000101****11001000** |
| | |

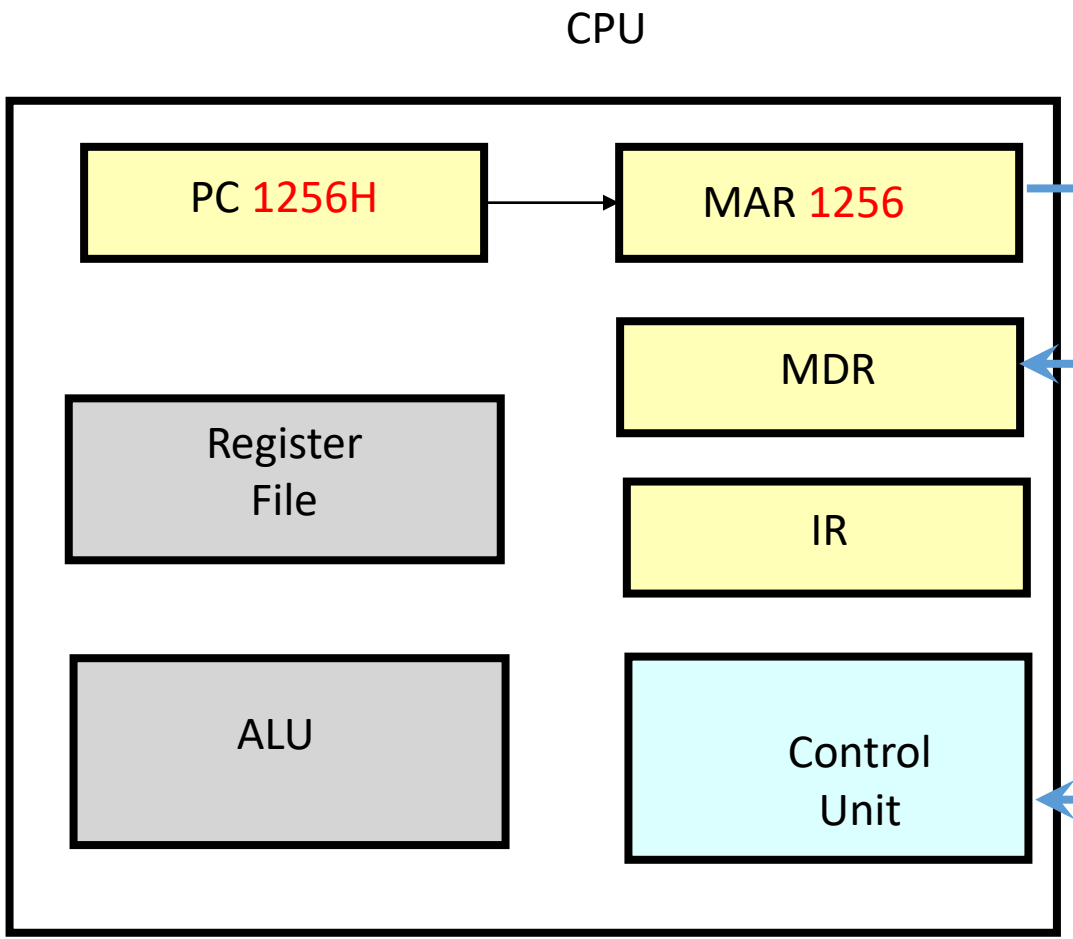
STEP-1: PC is loaded with address of 1st instruction of program

| User program | |
|---|---|
| SUB AX, BX | ;0010101**11000011** |
| MOV CX, AX | ;1000101**11001000** |
| MOV DX, 0 | |



# STEP-2: content of PC is loaded into MAR

| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |




Address Bus

Data Bus

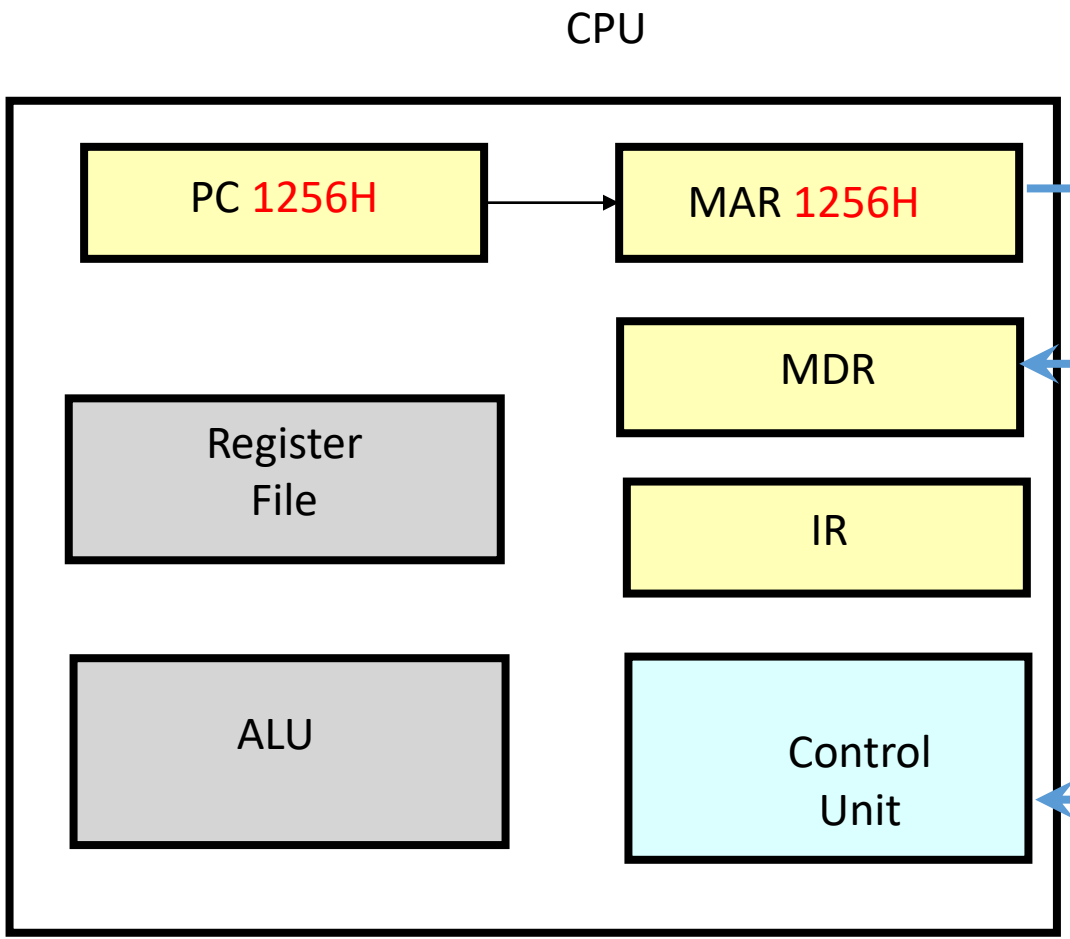
Control Bus

RAM(16 bits)

| Address | Contents |
| --- | --- |
| 1256H | **0010101**11000011 |
| 1257H | **1000101**11001000 |
| 1258H | **1001111**11001011 |

STEP-3: content of MAR is placed on Address bus and applied to Memory, as a result memory location 1256H is selected

| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |




Address Bus 1256H

Data Bus

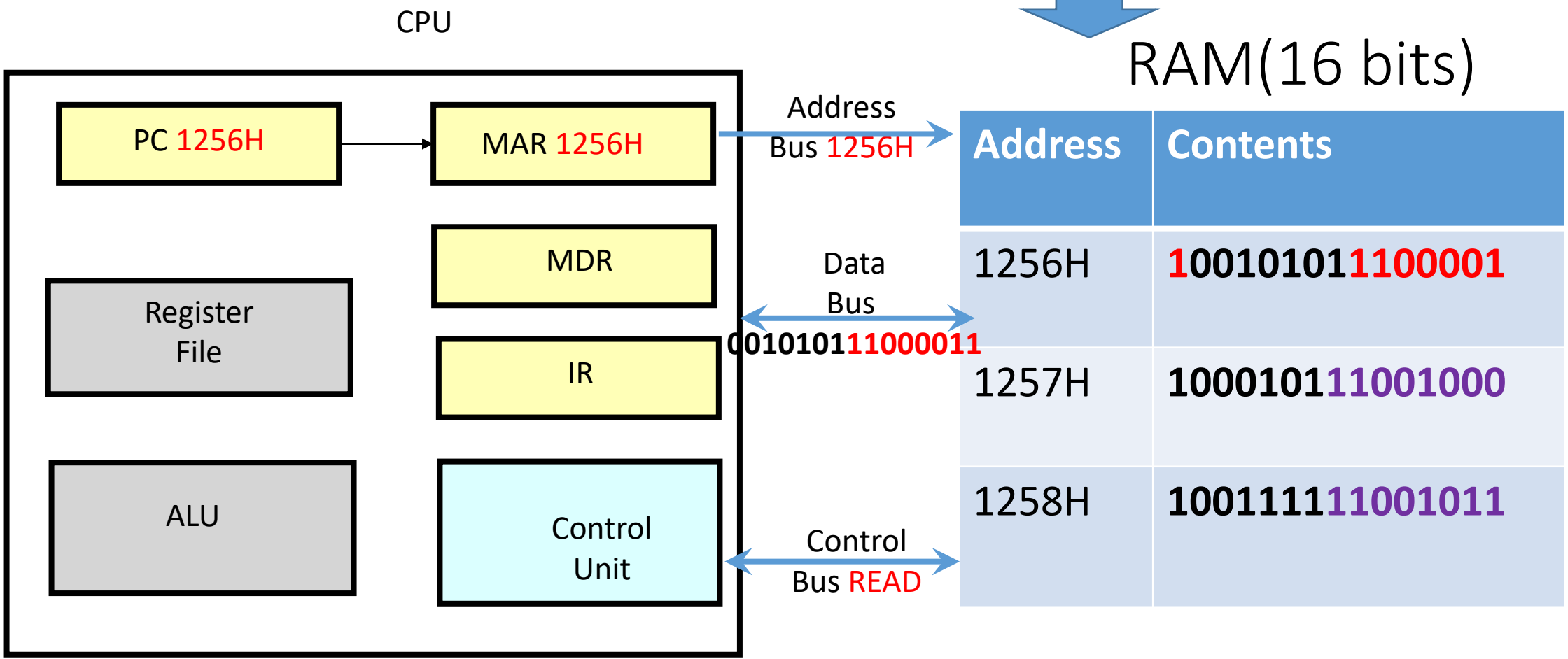
Control Bus

RAM(16 bits)

| Address | Contents |
| --- | --- |
| 1256H | 0010101**11000011** |
| 1257H | 1000101**11001000** |
| 1258H | 1001111**11001011** |

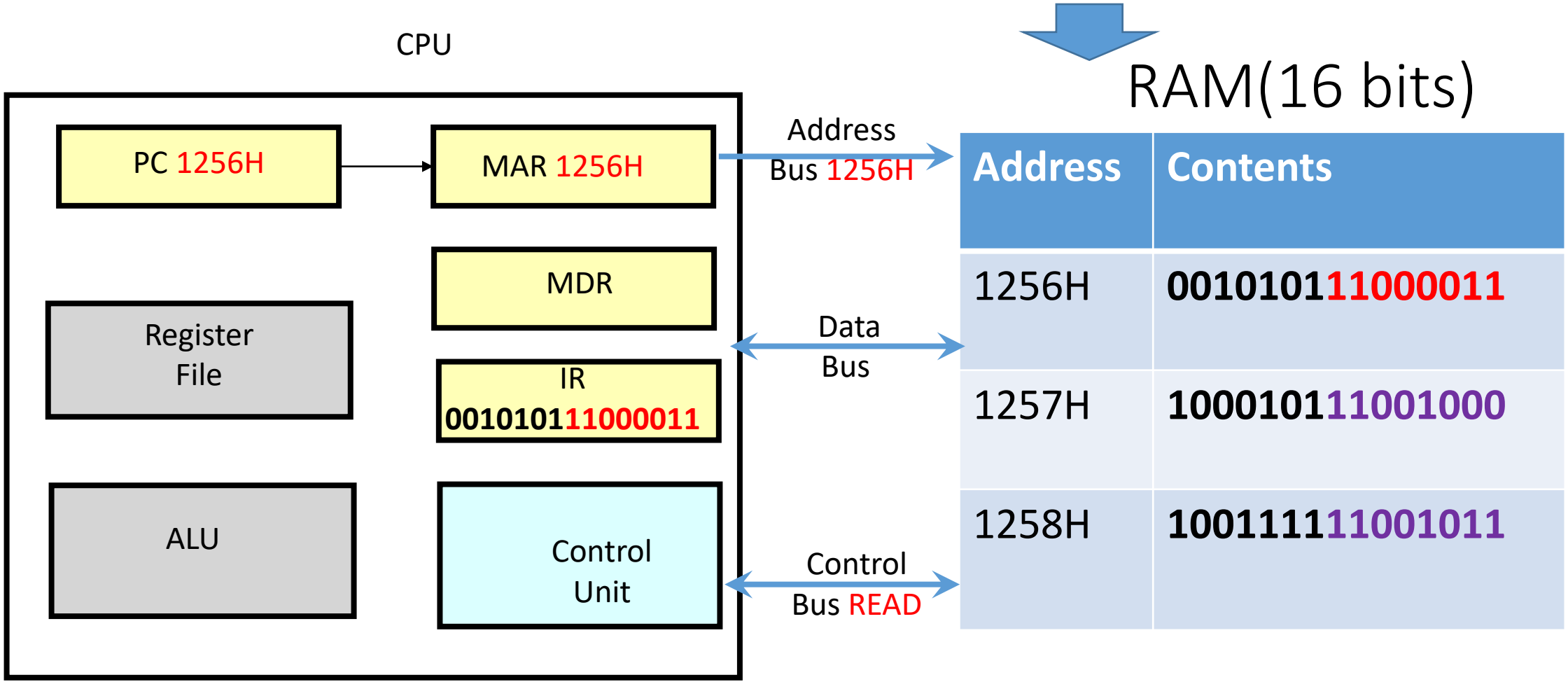
STEP-4: Control unit sends READ control signal to RAM, as a result machine code of 1st instruction is available on Data Bus

| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |



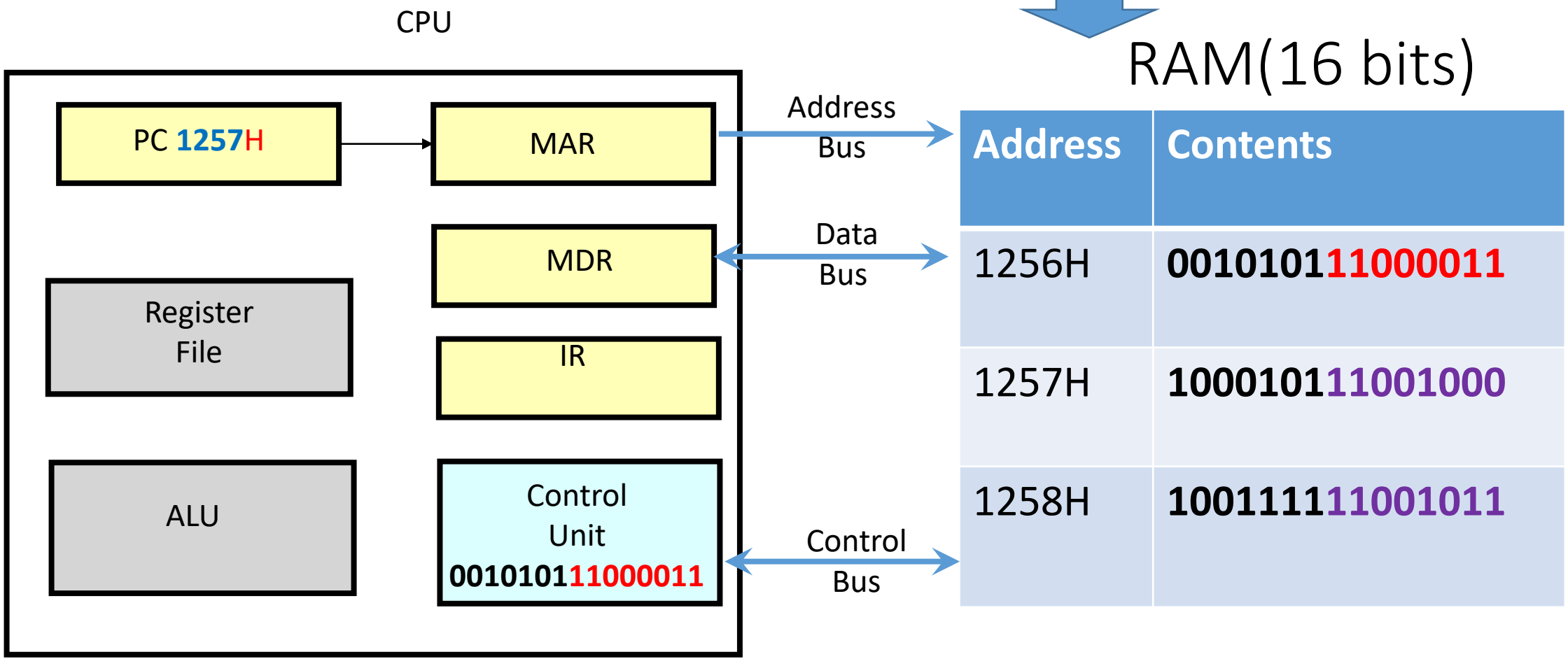
STEP-5: Machine code of 1st instruction is loaded into IR

| User program | |
|---|---|
| SUB AX, BX | ;0010101**11000011** |
| MOV CX, AX | ;1000101**11001000** |
| MOV DX, 0 | |



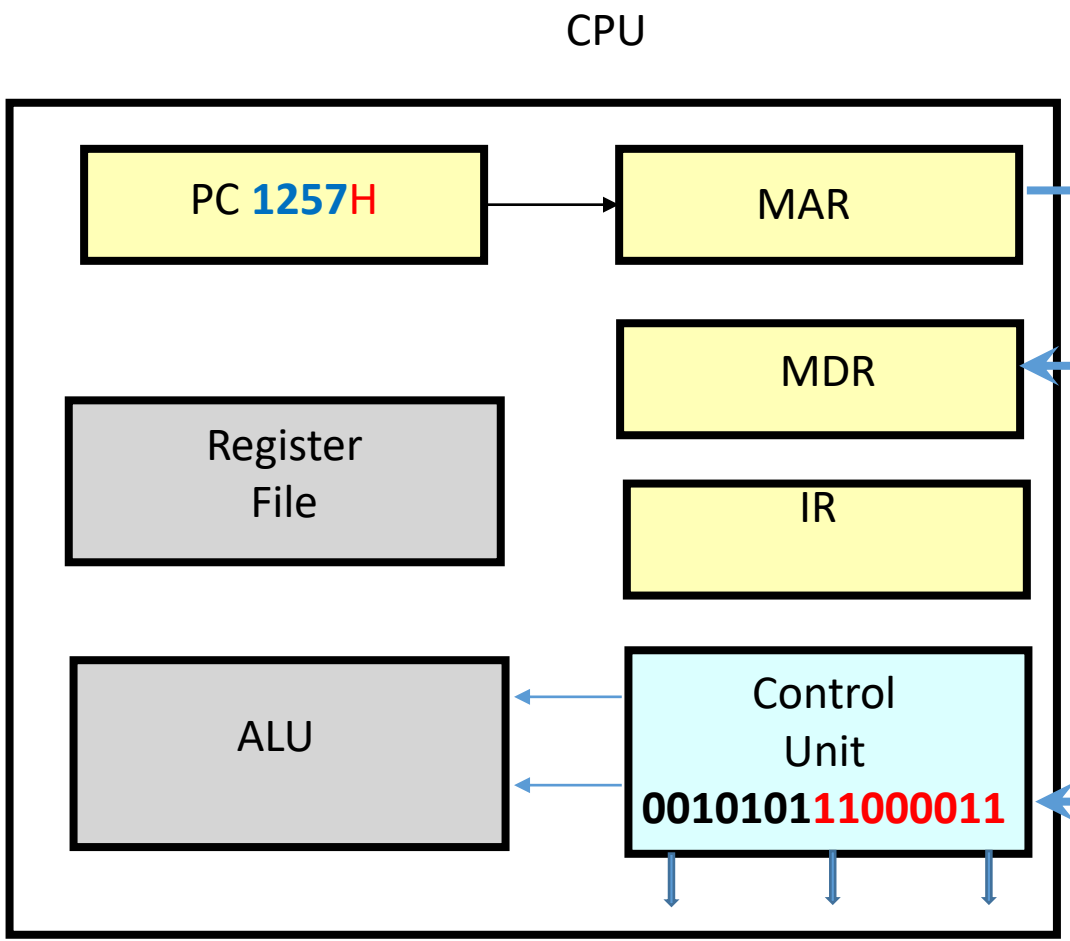
STEP-6: PC is incremented by 1 to point next instruction to be executed. Contents of IR is fed to Control Unit

| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |



STEP-7: 1st instruction is decoded at control unit: control signals are generated to activate ALU for specific operation as per instruction

| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |

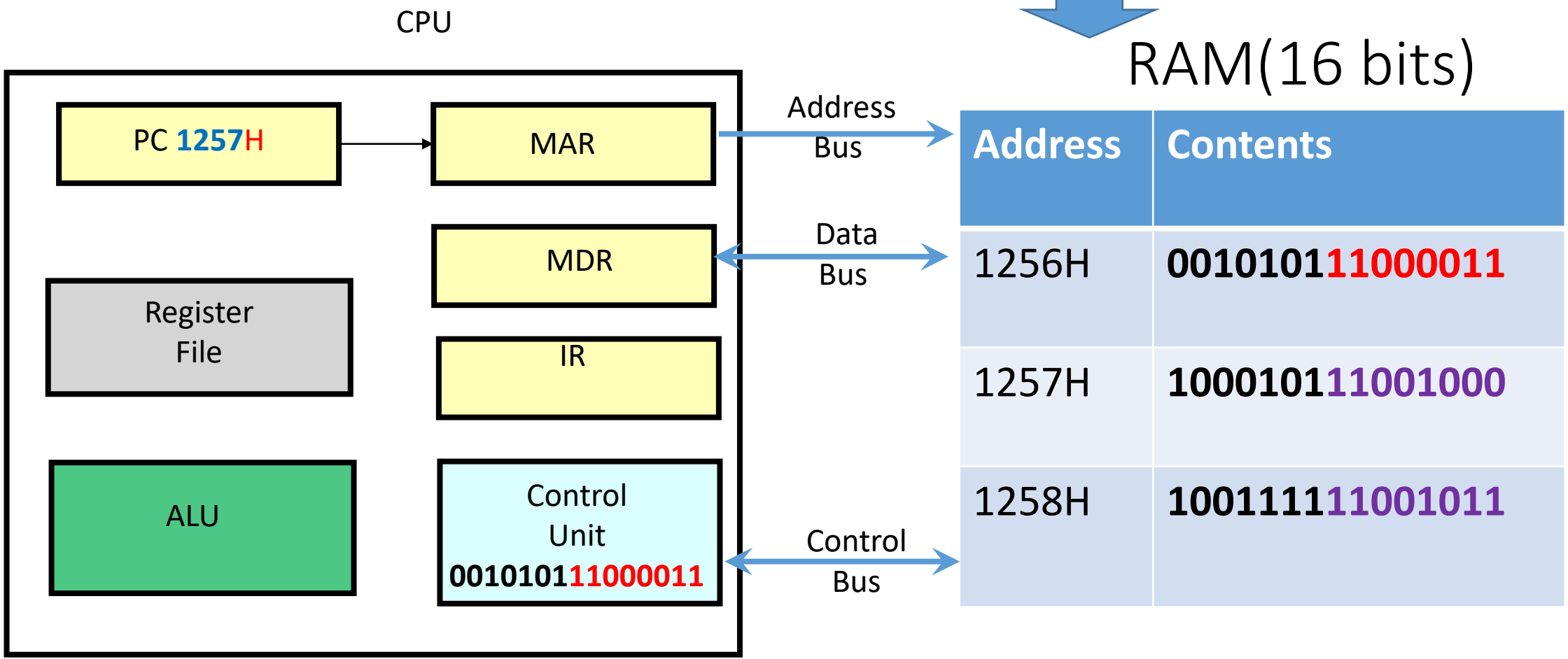



RAM(16 bits)

| Address | Contents |
| --- | --- |
| 1256H | **0010101 11000011** |
| 1257H | **1000101 11001000** |
| 1258H | **1001111 11001011** |

STEP-8: 1st instruction is Executed (that includes Data read from RAM followed by ALU operation, result stored as per instruction)

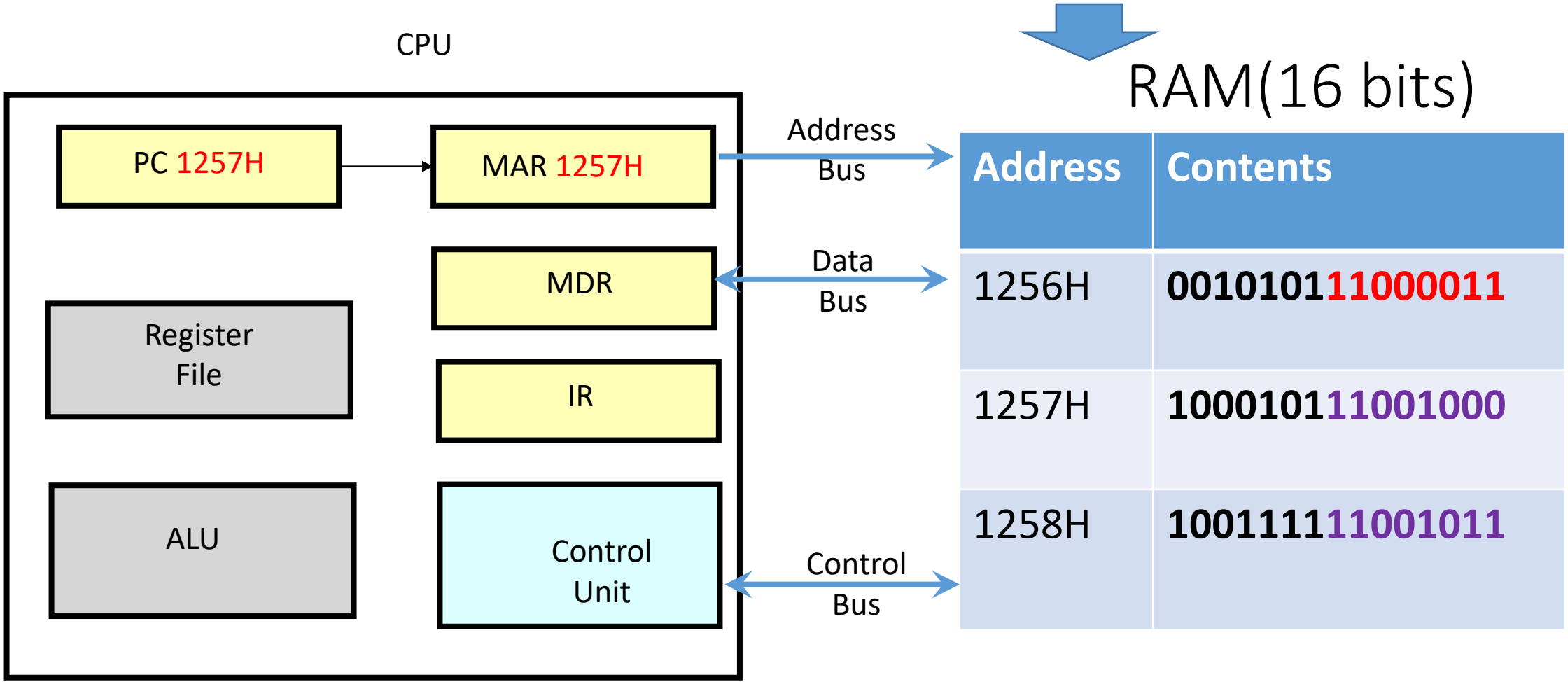
| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |



STEPS-2:8 repeated for next instruction

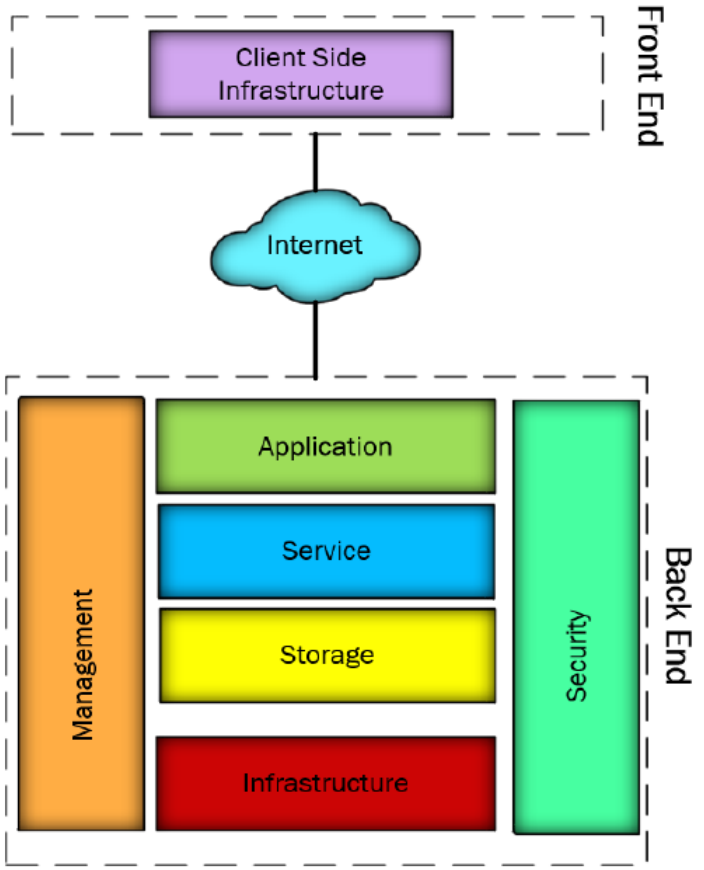
content of PC is loaded into MAR

| User program |
| --- |
| SUB AX, BX |
| MOV CX, AX |
| MOV DX, 0 |
| |



# Cloud Computing: Basics

**Cloud computing:** A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.



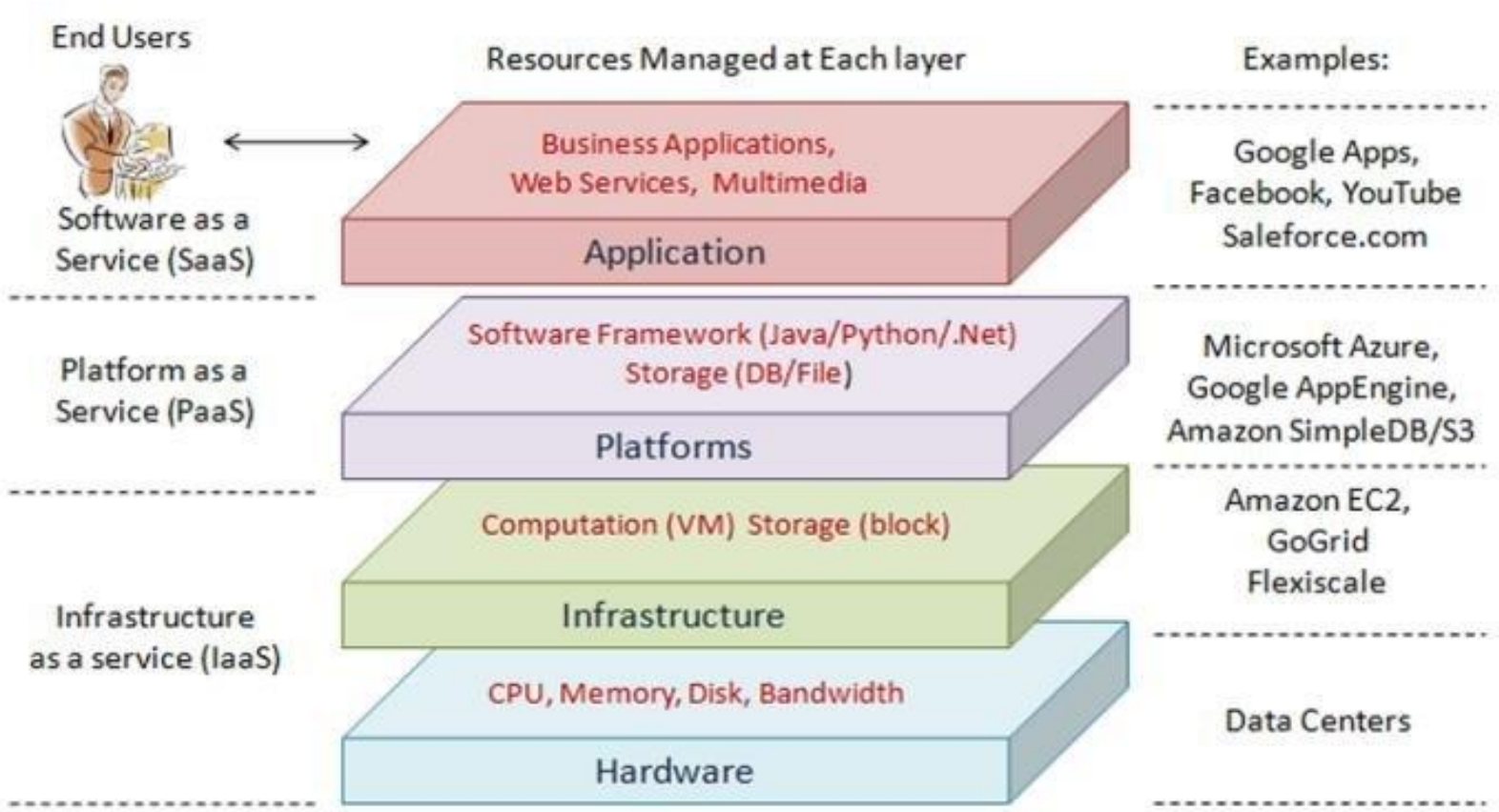
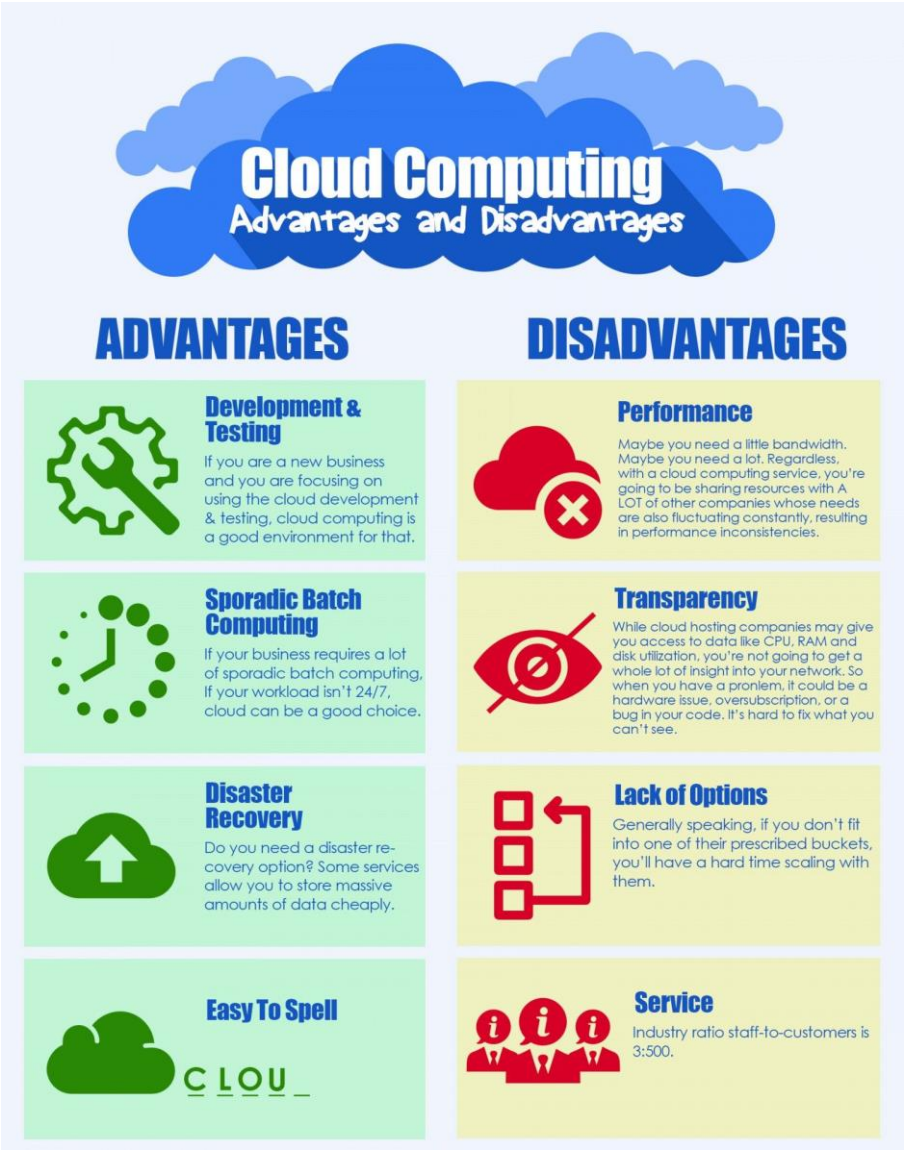
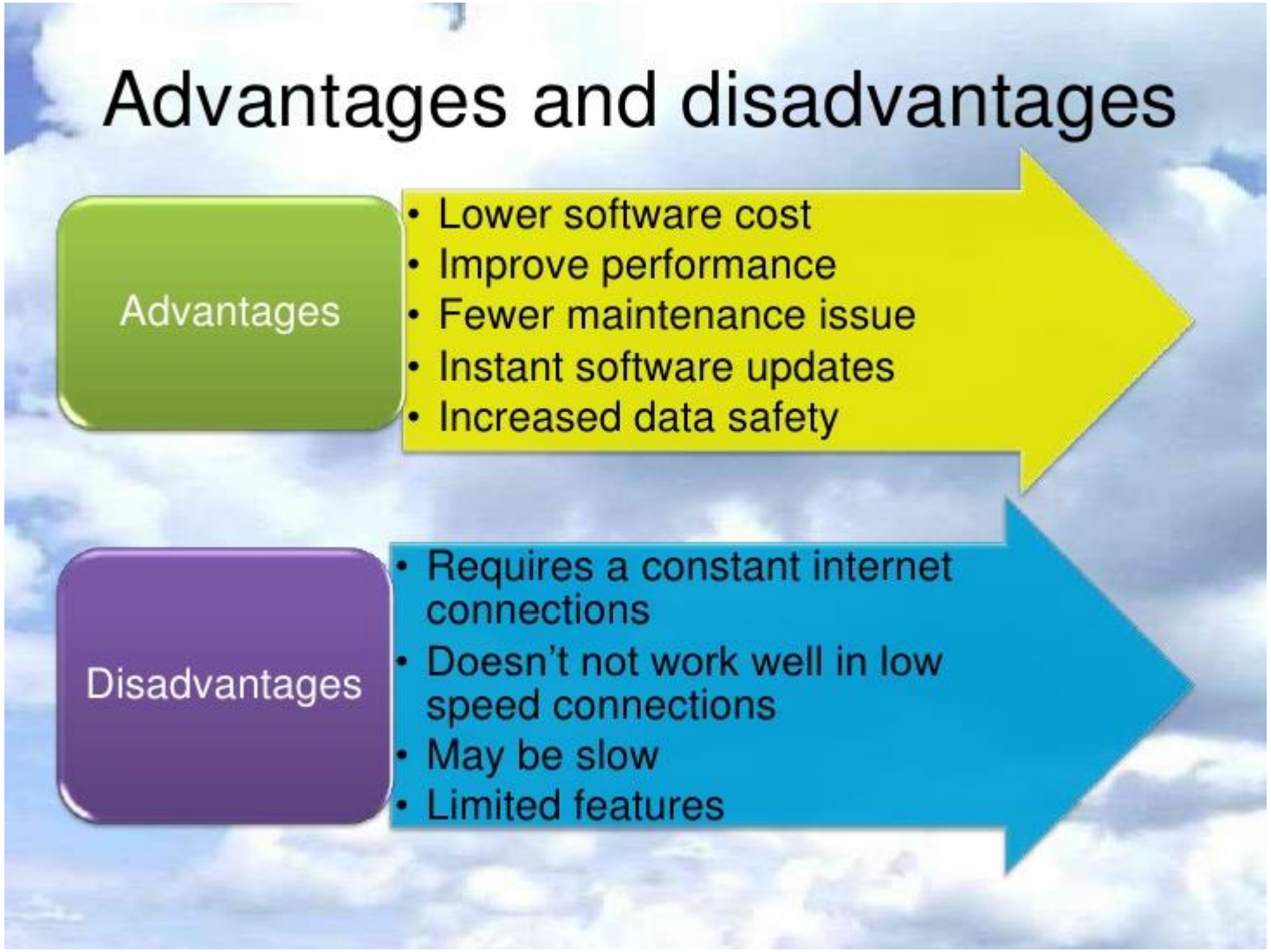




**Figure 2:** A generic Cloud Computing Architecture.

# Cloud Computing: Pros and Cons




See you next class!