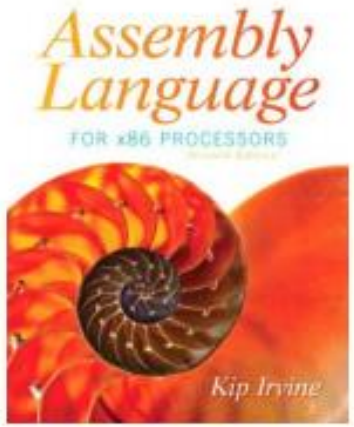
EE-2003
Computer Organization & Assembly Language

Recommended Text Book (s)



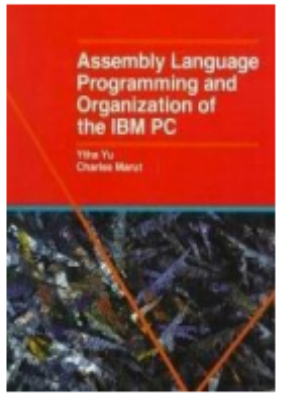
Assembly Language

FOR X86 PROCESSORS

Seventh Edition

Kip R. Irvine

Reference Text Book



Assembly Language Programming and Organization of the IBM PC

Ytha Yu, Charles Marut

MARKS DISTRIBUTION

Assessment	Weightage
Quizzes	10%
Assignments	10%
Midterm Exams	30%
Final	50%

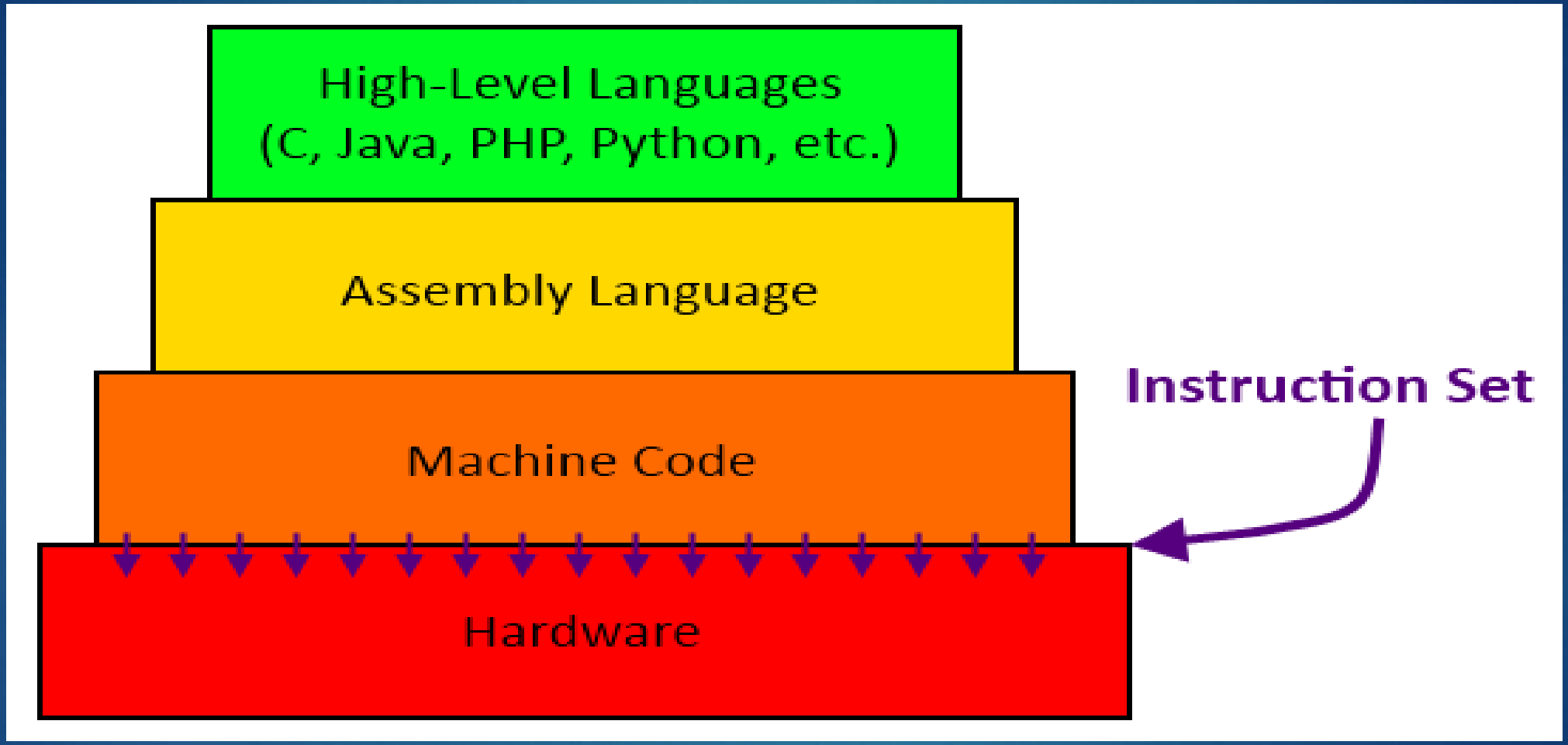
COURSE OBJECTIVES

- Covering the basics of computer organization with emphasis on the lower level abstraction of a computer system
- Programming Methodology of low-level languages, the assembly language.
- Accessing computer hardware directly
- Overview of a user-visible architecture (of Intel 80x86 processors)
- Intel 80x86 instruction set, assembler directives, macro, etc.
- Device handlers
- How is it possible to interface high-level language and low-level language modules

Computer Organization

- **Computer architecture** is concerned with the way hardware components to operate and the way they are connected together to form the computer system.
- **Computer organization** is concerned with the structure and behavior of a computer system as seen by the user.
- The computer organization is concerned with the **structure** and **behavior** of digital computers.
- Working of Internal Parts of Computer. i.e. RAM, CACHE etc.
- Computer organization describes how a task is done by the computer.

Hierarchy of Languages



High Level Languages

- A high-level language (HLL) is a programming language such as C, JAVA, or PYTHON.
- It enables a programmer to write programs that are more or less independent of a particular type of computer. **(Machine Independent)**
- Such languages are considered high-level because they are closer to human languages and further from machine languages.
- A single statement in C++ expands into multiple assembly language or machine instructions. (***one-to-many* relationship**)

Assembly Language

- An assembly language is a low-level **programming language** designed for a specific type of **processor**. ***(Machine Specific)***
- Assembly language consists of statements written with short mnemonics such as ADD, MOV, SUB, and CALL.
- Each assembly language instruction corresponds to a single machine-language instruction. (***one-to-one*** **relationship)**

Translating Languages

English: D is assigned the sum of A times B plus 10.

High-Level Language: D = A * B + 10

A statement in a high-level language is translated typically into several machine-level instructions

Intel Assembly Language:

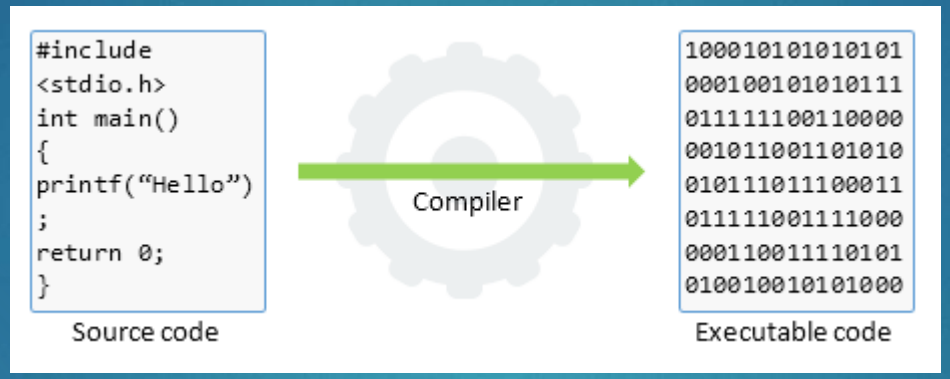
```
mov   eax, A
mul   B
add   eax, 10
mov   D, eax
```

Intel Machine Language:

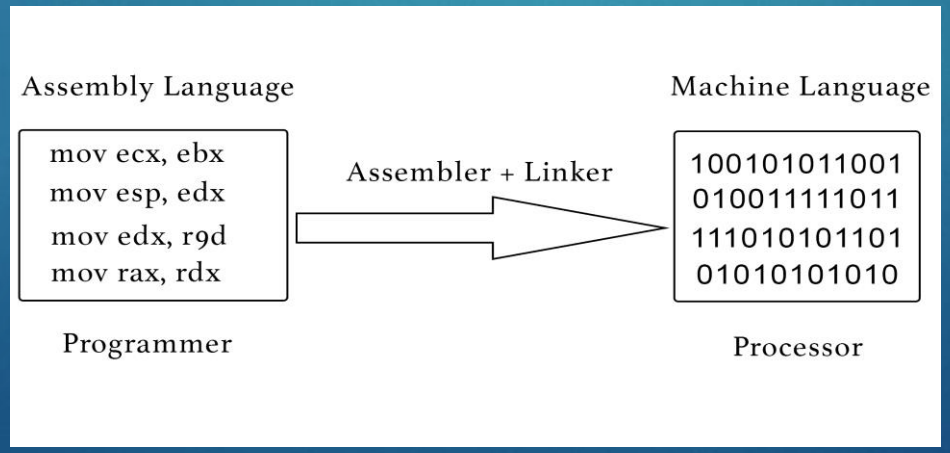
```
A1 00404000
F7 25 00404004
83 C0 0A
A3 00404008
```

Compiler and Assembler

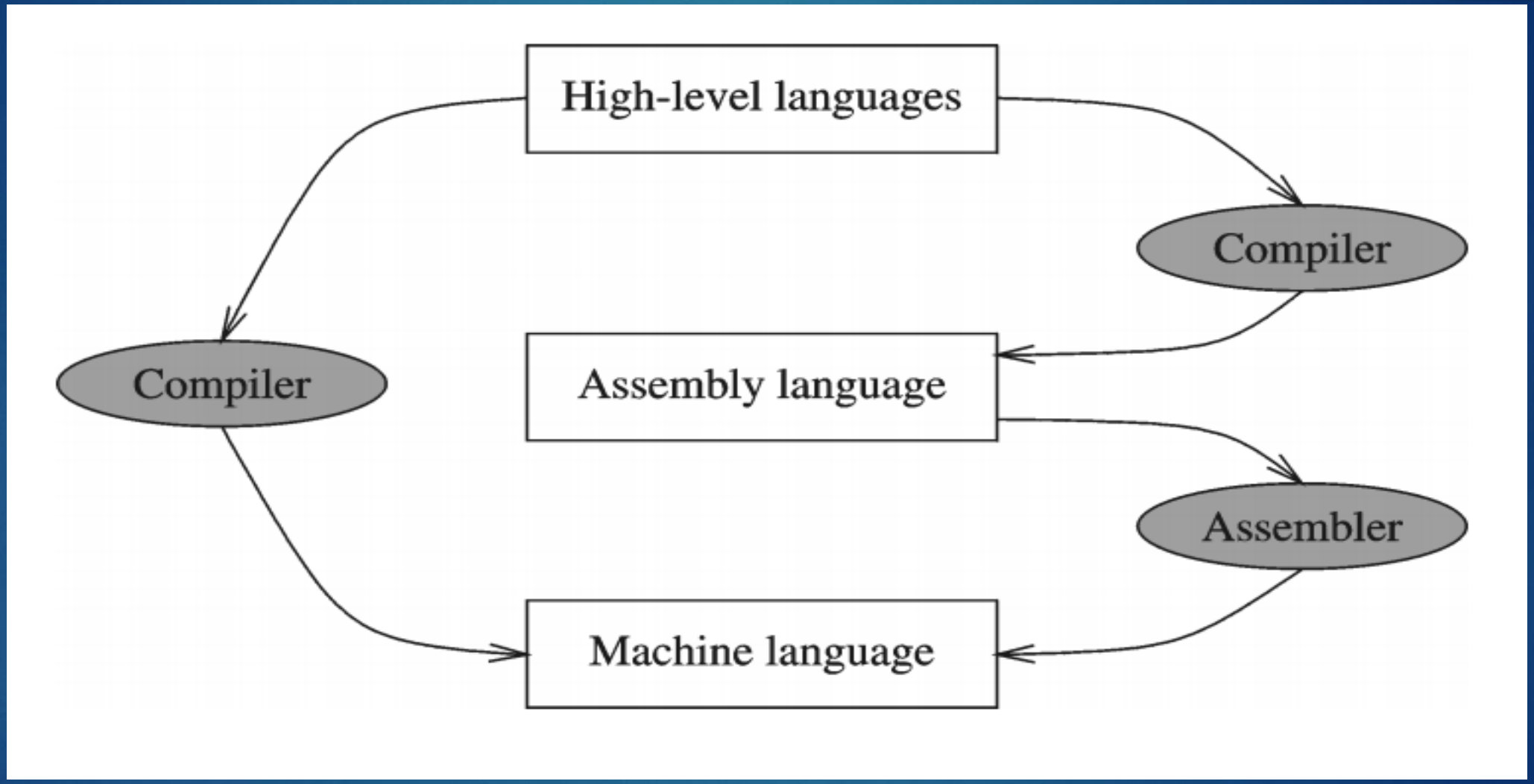
- Compilers translate high-level programs to machine code.(Directly/Indirectly).



- Assemblers translate assembly language to machine code.



Compiler and Assembler



Is Assembly Language Portable?

Is Assembly Language Portable?

- A language whose source programs can be compiled and run on a wide variety of computer systems is said to be portable.
- A C++ program, for example, should compile and run on just about any computer, unless it makes specific references to library functions that exist under a single operating system.
- Assembly language is ***not portable*** because it is designed for a specific processor family.

Assembly Language for x86 Processors

- Intel stands for "**Int**egrated **El**ectronics"
- AMD stands for Advanced Micro Devices.
- Assembly Language for x86 Processors focuses on programming microprocessors compatible with the **Intel IA-32** and **AMD x86** processors running under Microsoft Windows.

Advantages of High-Level Languages

- Program development is faster
  - – High-level statements: fewer instructions to code
- Program maintenance is easier
  - – For the same above reasons
- Programs are portable
  - – Contain few machine-dependent details
    - Can be used with little or no modifications on different machines

Why Learn Assembly Language?

- **Complete control over a system's resources**

  - gateway to optimization in speed, offering great efficiency and performance.

- **Assembly language is transparent**

  -It has a small number of operations, but it is helpful in understanding the algorithms and other flow of controls. It makes the code less complex and easy debugging as well.

- For writing the compilers or device drivers, write some code in assembly language.

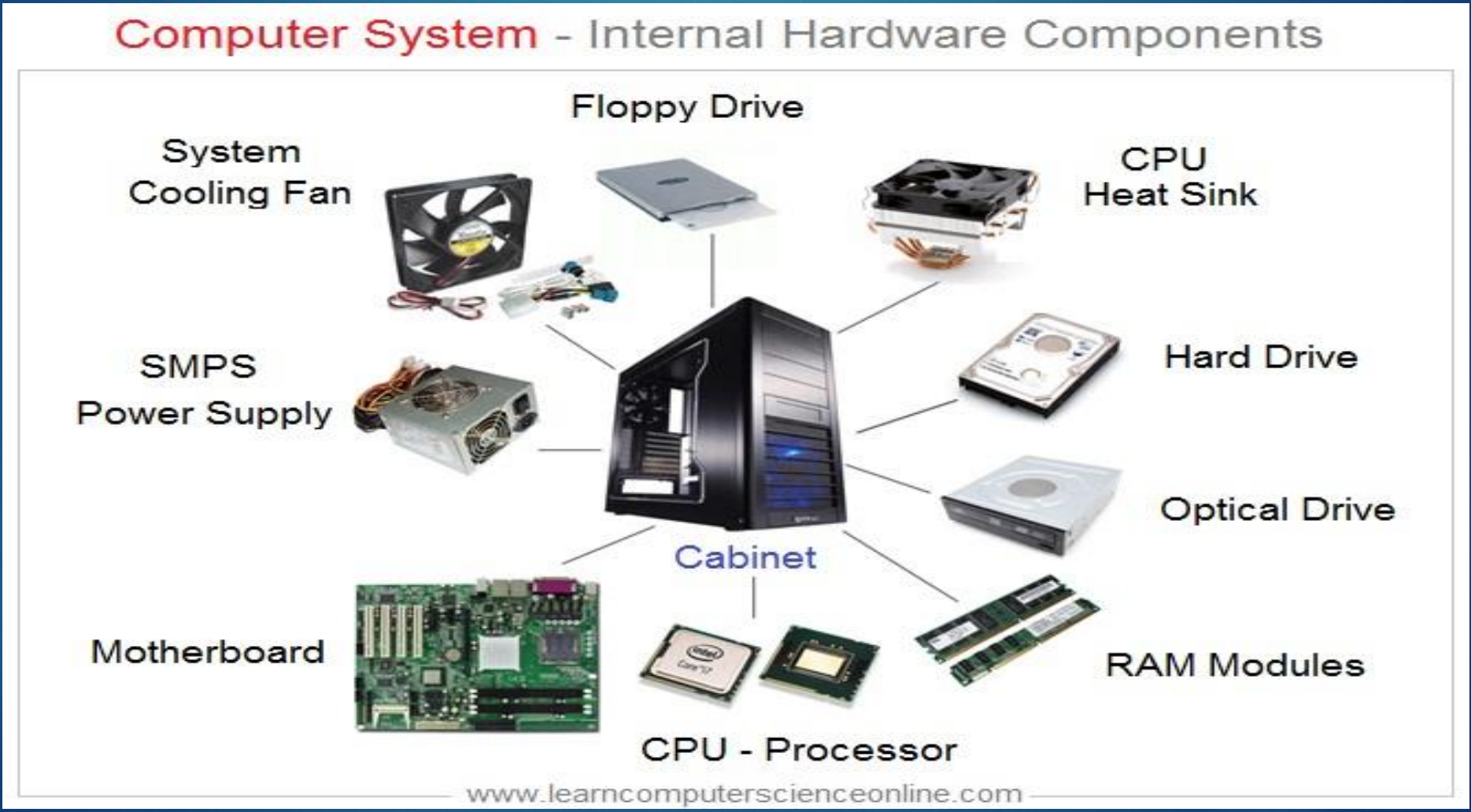
Applications of Assembly Language

- **Embedded System Programming**

- **Game Programmers → programs to be highly optimized for both space and runtime speed.**

Virtual Machine Concept

- Computers are Build with Physical Parts i.e. Hardware



Virtual Machine Concept

- SOFTWARES → Makes Computer easy to use.

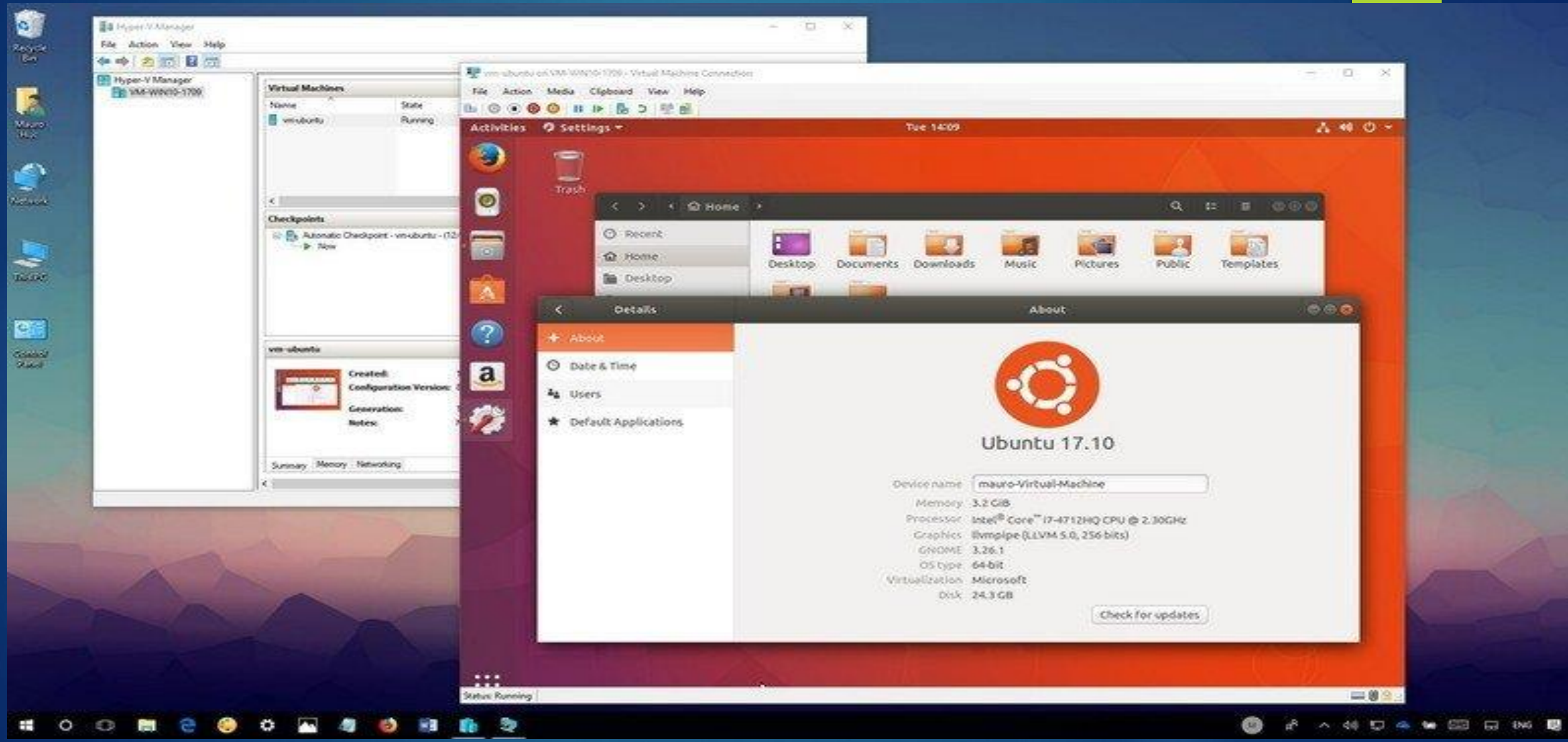
Virtual Machine Concept

- OPERATING SYSTEM SOFTWARES
- They are able to control Physical Components of Computer i.e. HARDWARE

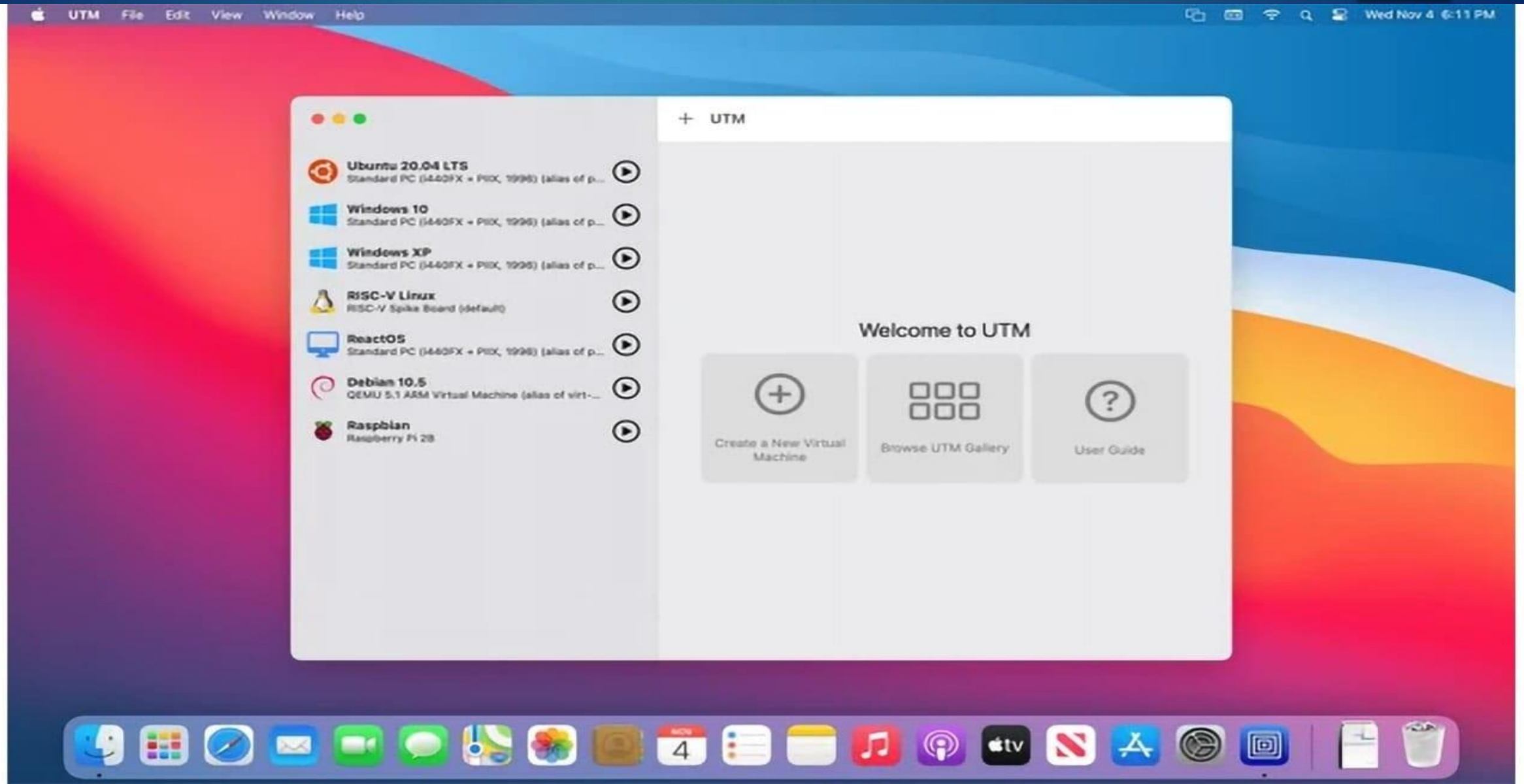
Virtual Machine Concept

- **VIRTUAL MACHINE MANAGER (HYPERVISOR)**
- **Software → allows to run → an Operating System → Within another Operating System.**
- 

Virtual Machine Concept



Virtual Machine Concept



Virtual Machine Concept

Virtual Machine Concept

- A virtual machine (VM) is an image file managed by the hypervisor that exhibits the behavior of a separate computer, capable of performing tasks such as running applications and programs like a separate computer.

- In other words, a VM is a software application that performs most functions of a physical computer, actually behaving as a separate computer system.

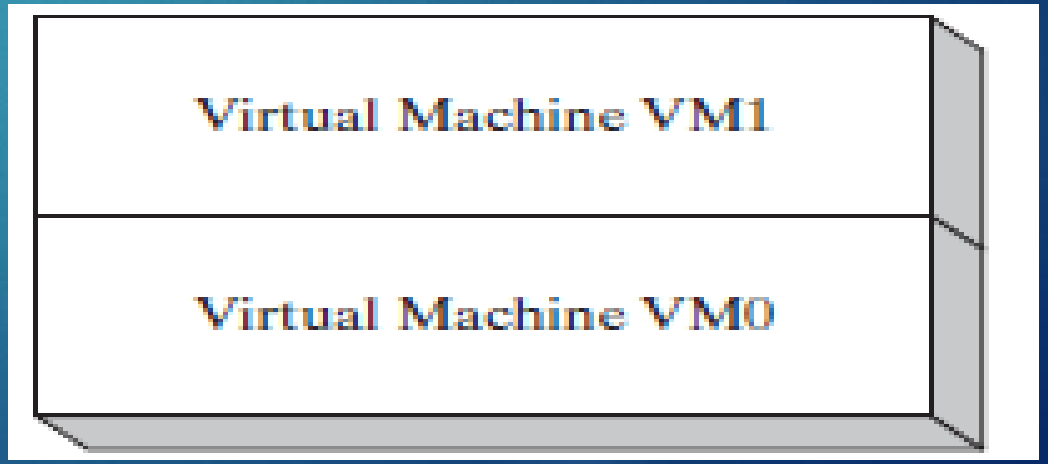
- A virtual machine, usually known as a guest, is created within another computing environment referred as a "host." Multiple virtual machines can exist within a single host at one time.

Virtual Machine Concept

Virtual Machine Concept

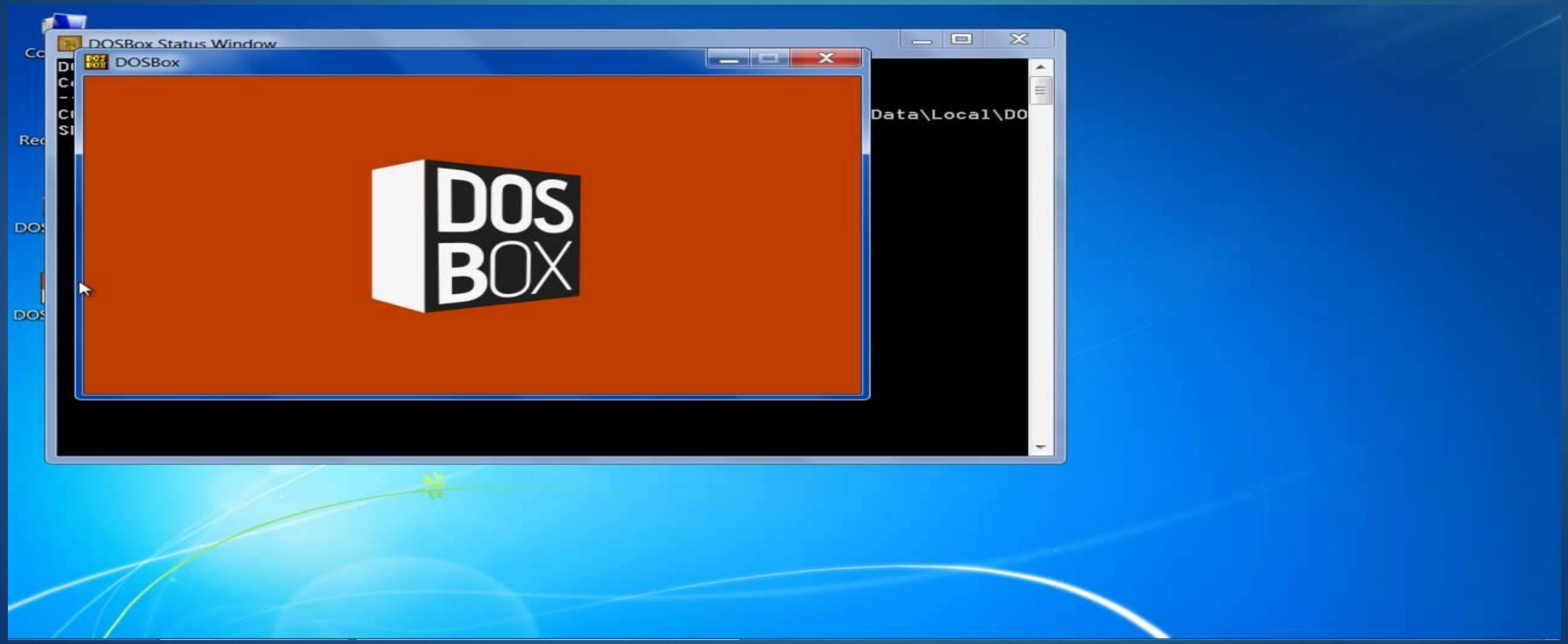
- Programming Language analogy:
  - Each computer has a native machine language (language L0) that runs directly on its hardware
  - A more human-friendly language is usually constructed above machine language, called Language L1

- The virtual machine **VM1**, can execute commands written in language L1.
- The virtual machine **VM0** can execute commands written in language L0



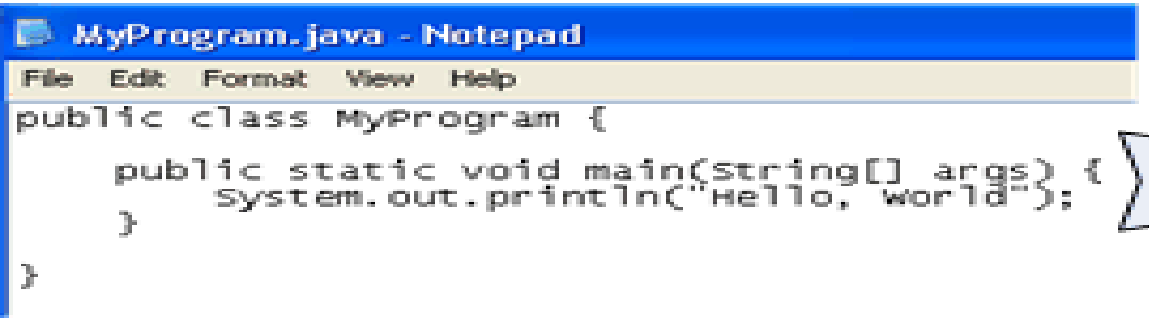
Virtual Machine Concept

- The virtual machine **VM0** can execute commands written in language L0

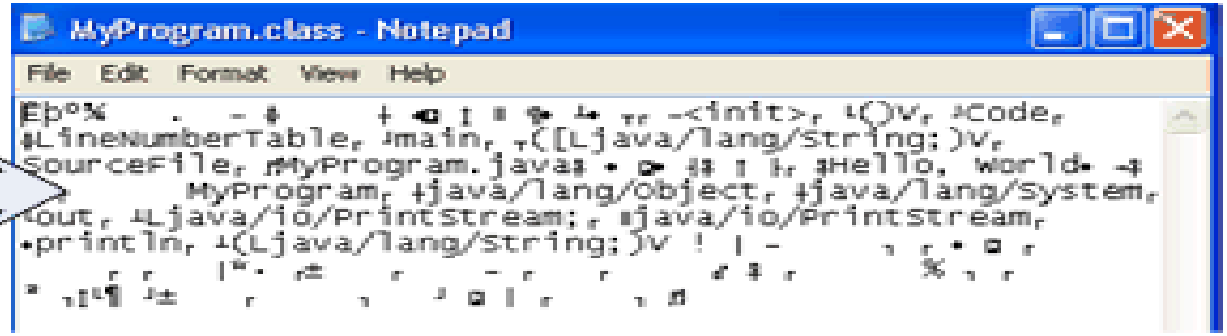


Virtual Machine Concept

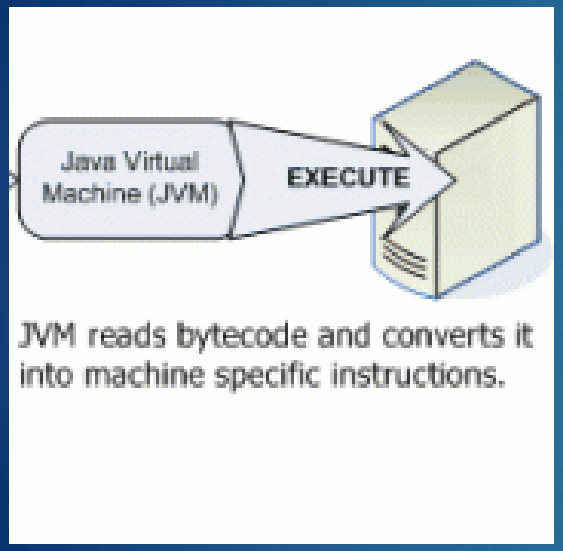
- The Java programming language is based on the virtual machine concept.
- A program written in the Java language is translated by a Java compiler into *Java byte code* - a low-level language quickly executed at runtime by a program known as a *Java virtual machine (JVM)*.



Source code is first written in plain text files ending with the .java extension.

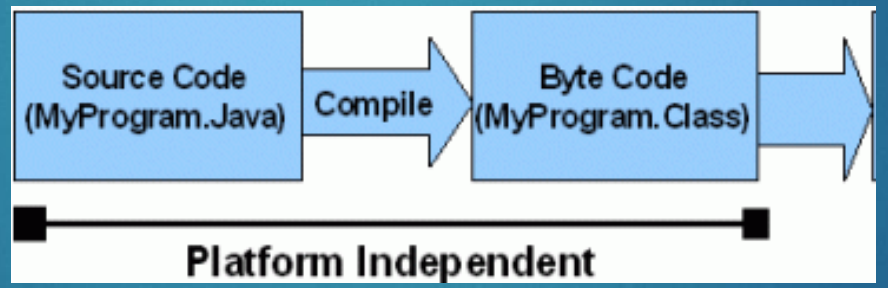


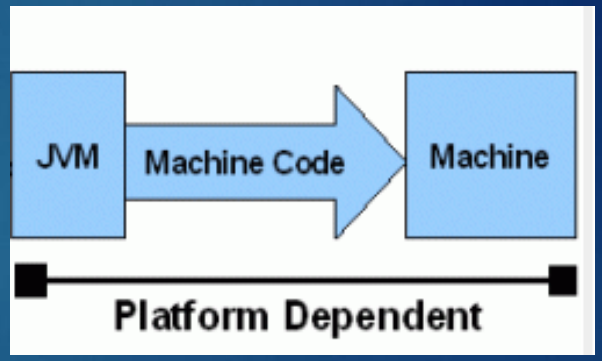
After the compilation is successful, java compiler will generate an intermediate ".class" file that contains the bytecode.



JVM reads bytecode and converts it into machine specific instructions.

Interpret

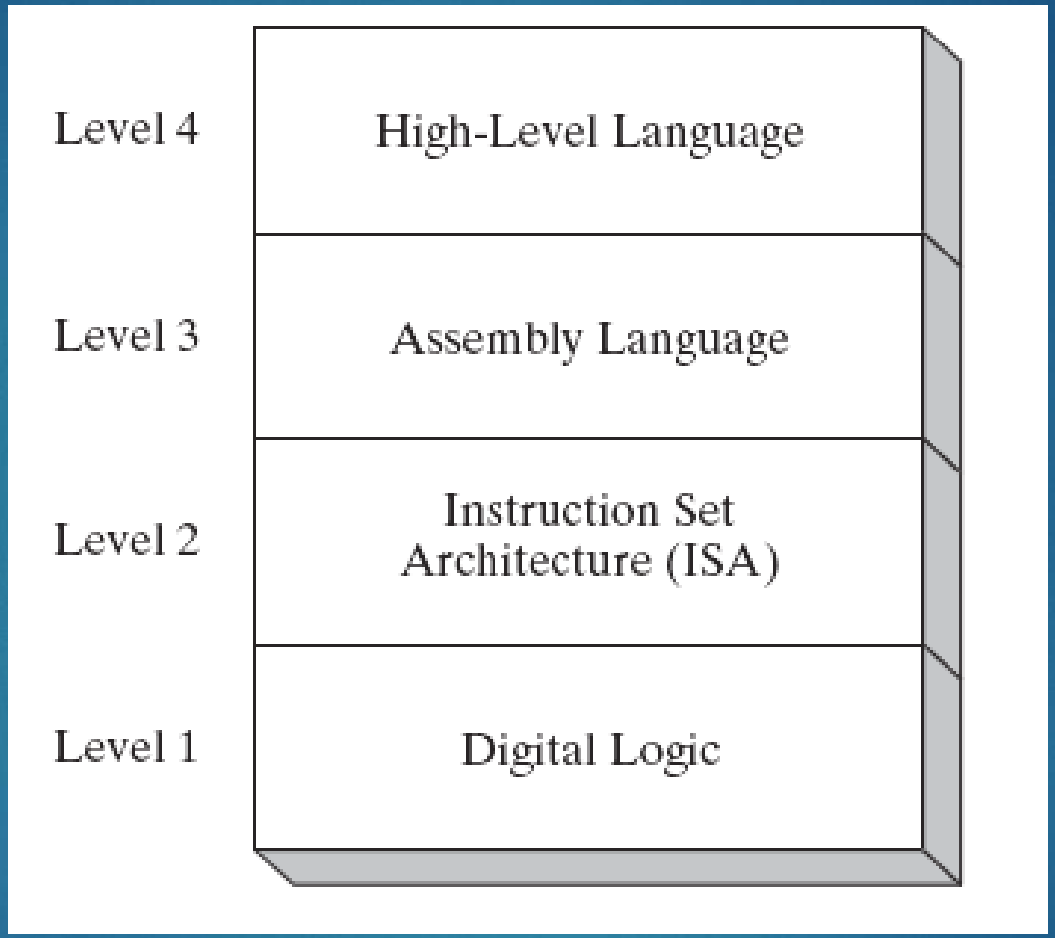




Java code / Bytecode is always the same on different OS. That makes java program as platform independent.

JVM is platform dependent that means there are different implementation of JVM on different OS.

Specific Machine Levels

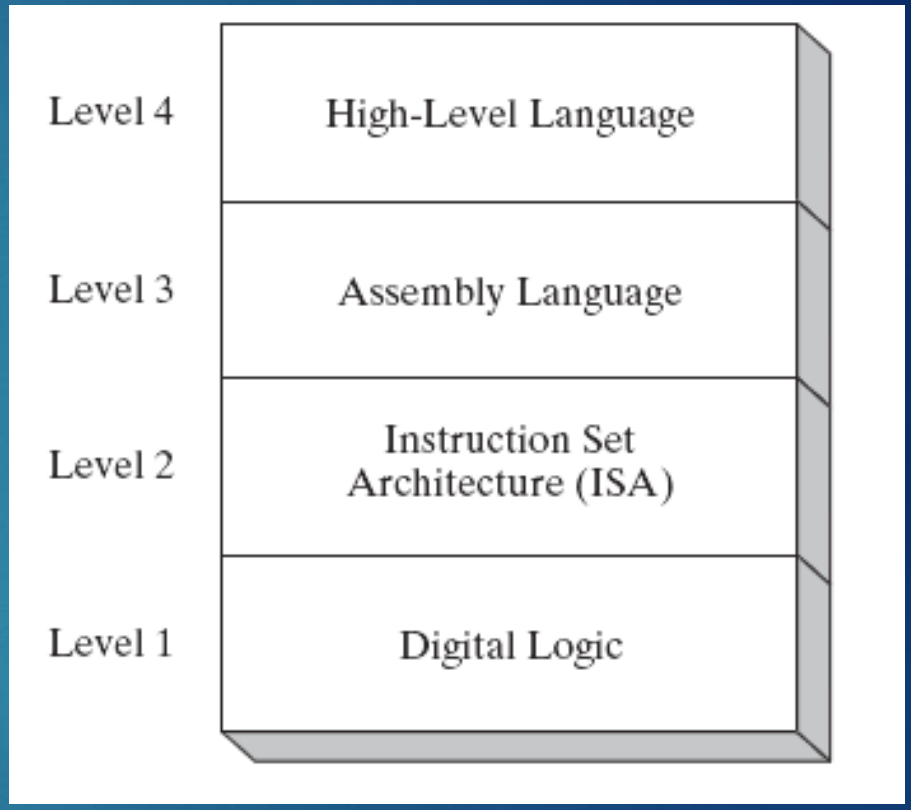


(descriptions of individual levels follow . . .)

High-Level Language

Level 4

- Application-oriented languages
  - C++, Java, Pascal, Visual Basic . . .
- Programs compile into assembly language (Level 3)

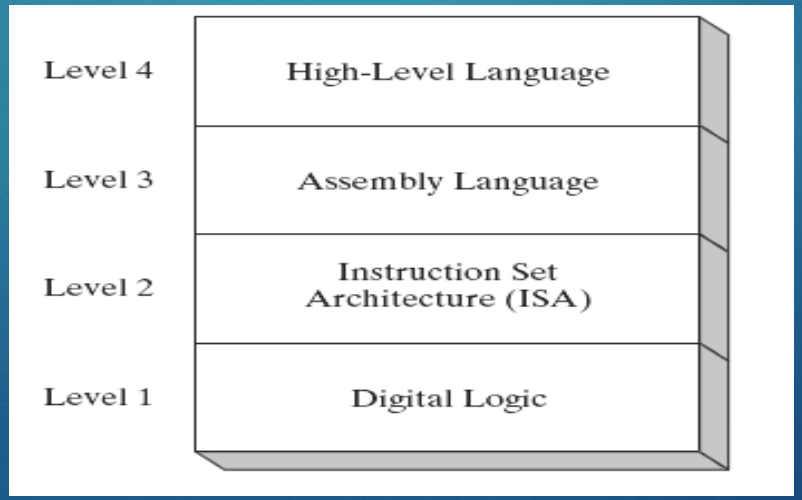


Assembly Language

Level 3

- Instruction mnemonics that have a one-to-one correspondence to machine language
- Programs are translated into Instruction Set Architecture Level - machine language (Level 2)

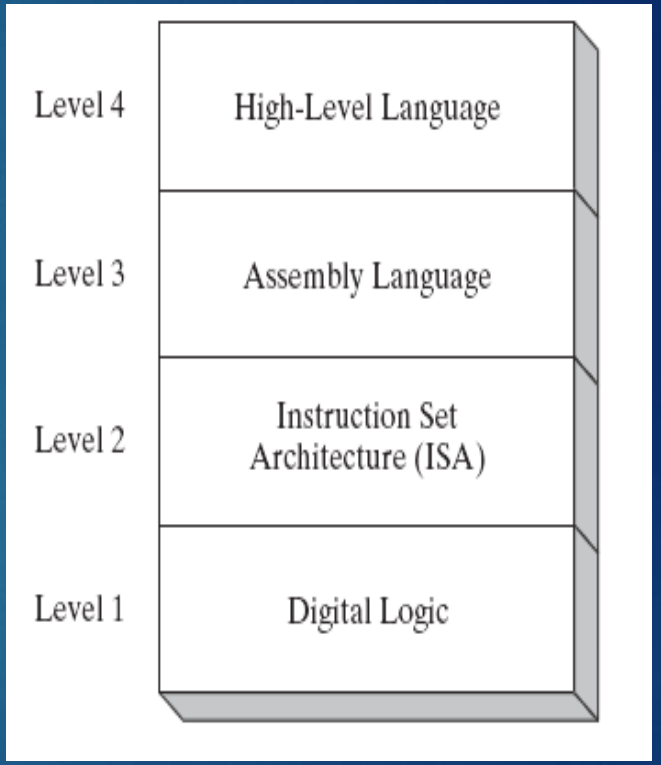
- The instructions in assembly language may directly match the computer's architecture or they may be translated during execution by a program inside the processor known as a *microcode interpreter*.



Instruction Set Architecture (ISA)

Level 2

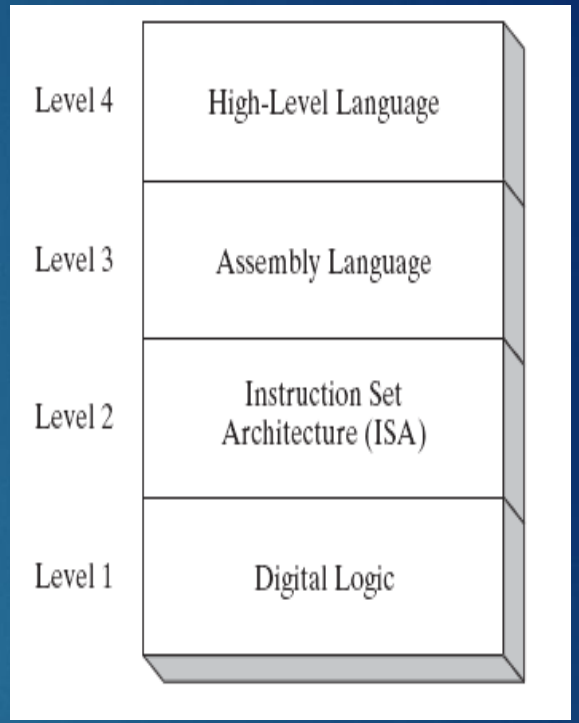
- Also known as conventional machine language.
- Executed by Level 1 (Digital Logic).



Digital Logic

Level 1

- CPU, constructed from digital logic gates
- System bus
- Memory
- Implemented using bipolar transistors

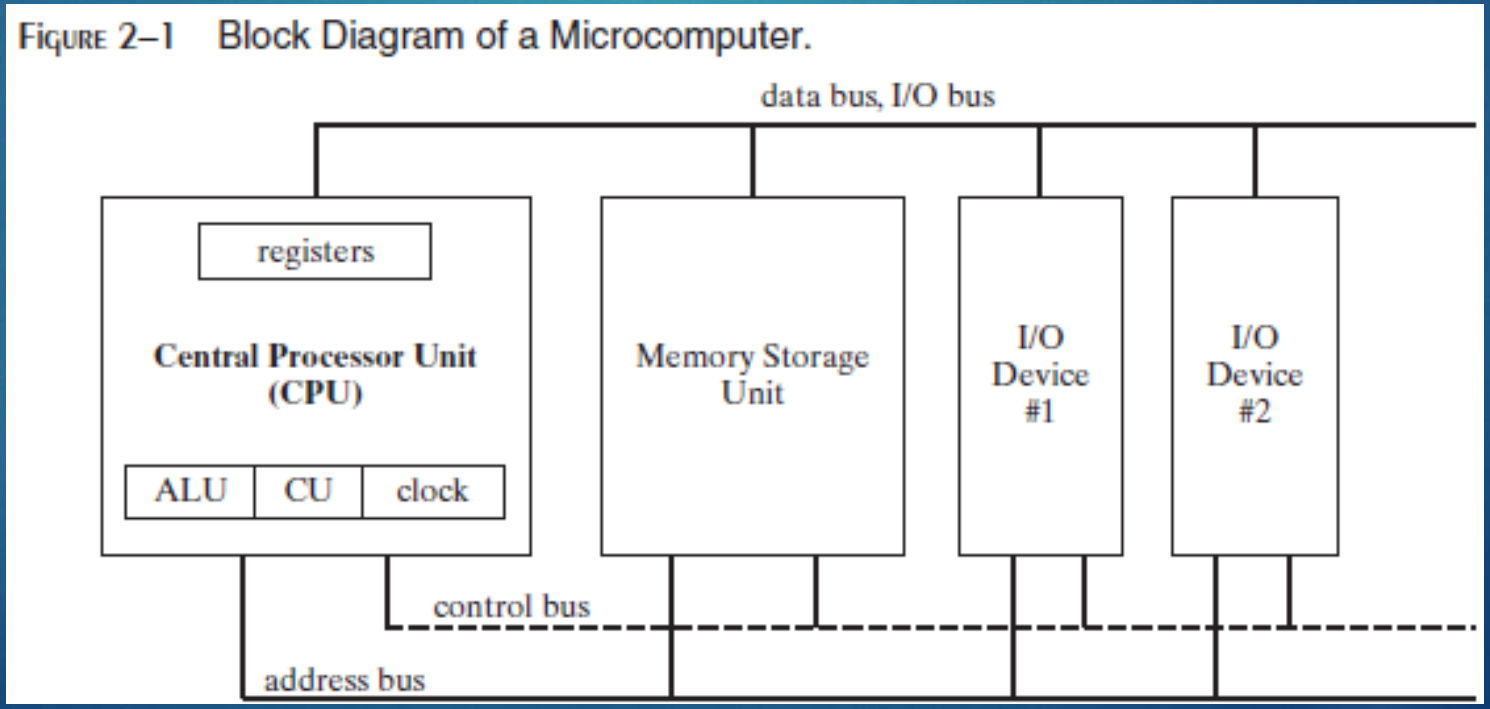


Basic Microcomputer Design

- The central processor unit (CPU), where calculations and logic operations take place, contains a limited number of storage locations named registers, a high-frequency clock, a control unit, and an arithmetic logic unit.
- The memory storage unit is where instructions and data are held while a computer program is running.
- The storage unit receives requests for data from the CPU, transfers data from random access memory (RAM) to the CPU, and transfers data from the CPU into memory.
- All processing of data takes place within the CPU, so programs residing in memory must be copied into the CPU before they can execute.

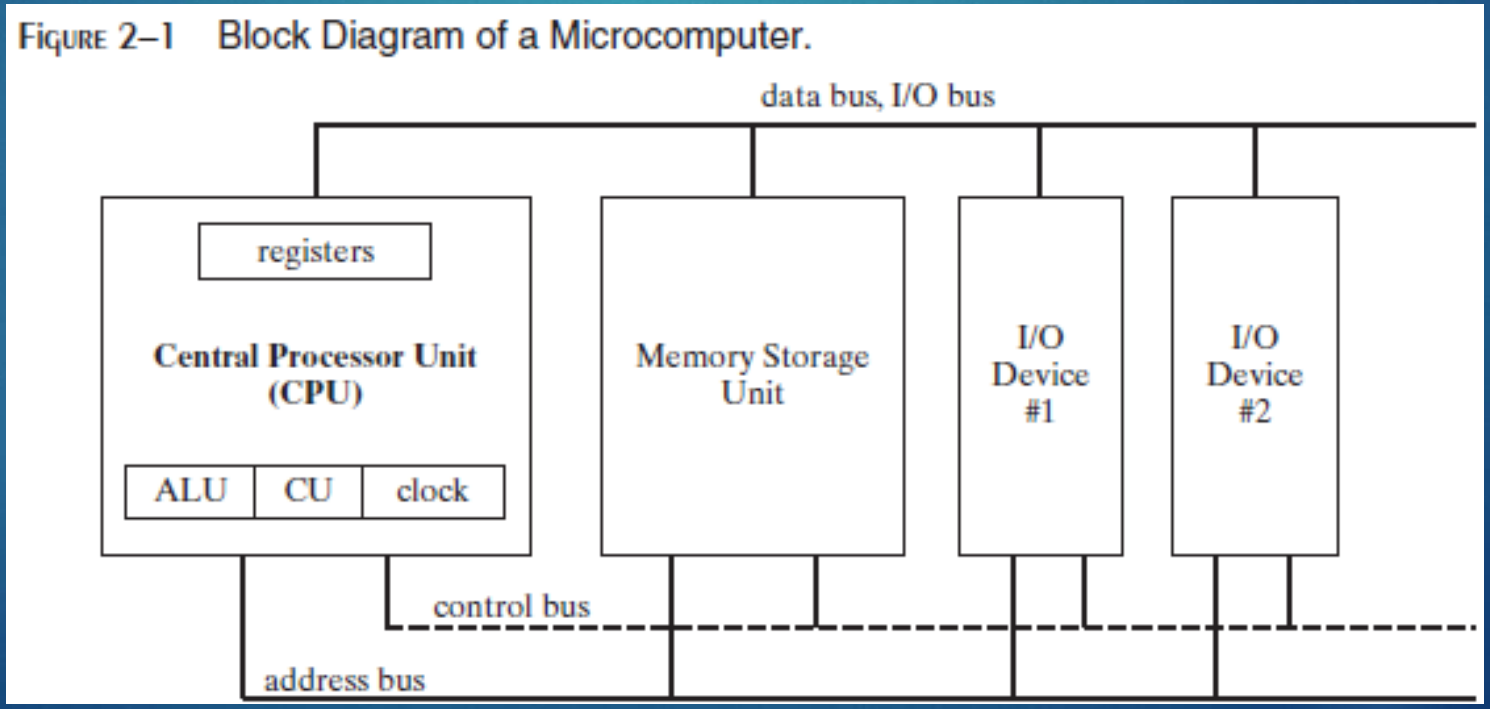
Basic Microcomputer Design

- The *central processor unit* (CPU), where calculations and logic operations take place, contains a limited number of storage locations named registers, a high-frequency clock, a control unit, and an arithmetic logic unit.
- The memory storage unit is where instructions and data are held while a computer program is running.



Basic Microcomputer Design

- The storage unit receives requests for data from the CPU, transfers data from random access memory (RAM) to the CPU, and transfers data from the CPU into memory.
- All processing of data takes place within the CPU, so programs residing in memory must be copied into the CPU before they can execute.



Buses

- A *bus* is a group of parallel wires that transfer data from one part of the computer to another.
- A computer system usually contains *four bus types*: **data, I/O, control, and address**.
- The data bus transfers instructions and data between the CPU and memory.
- The I/O bus transfers data between the CPU and the system input/output devices.
- The control bus uses binary signals to synchronize actions of all devices attached to the system bus.
- The *address bus* holds the addresses of instructions and data when the currently executing instruction transfers data between the CPU and memory.

Clock Cycles

- A machine instruction requires at least one clock cycle to execute.
  - Few require in excess of 50 clocks (the multiply instruction on the 8088 processor, for example).

- Instructions requiring memory access often have empty clock cycles called **wait states**.
  - Because of the differences in the speeds of the CPU, the system bus, and memory circuits.

Instruction Execution Cycle

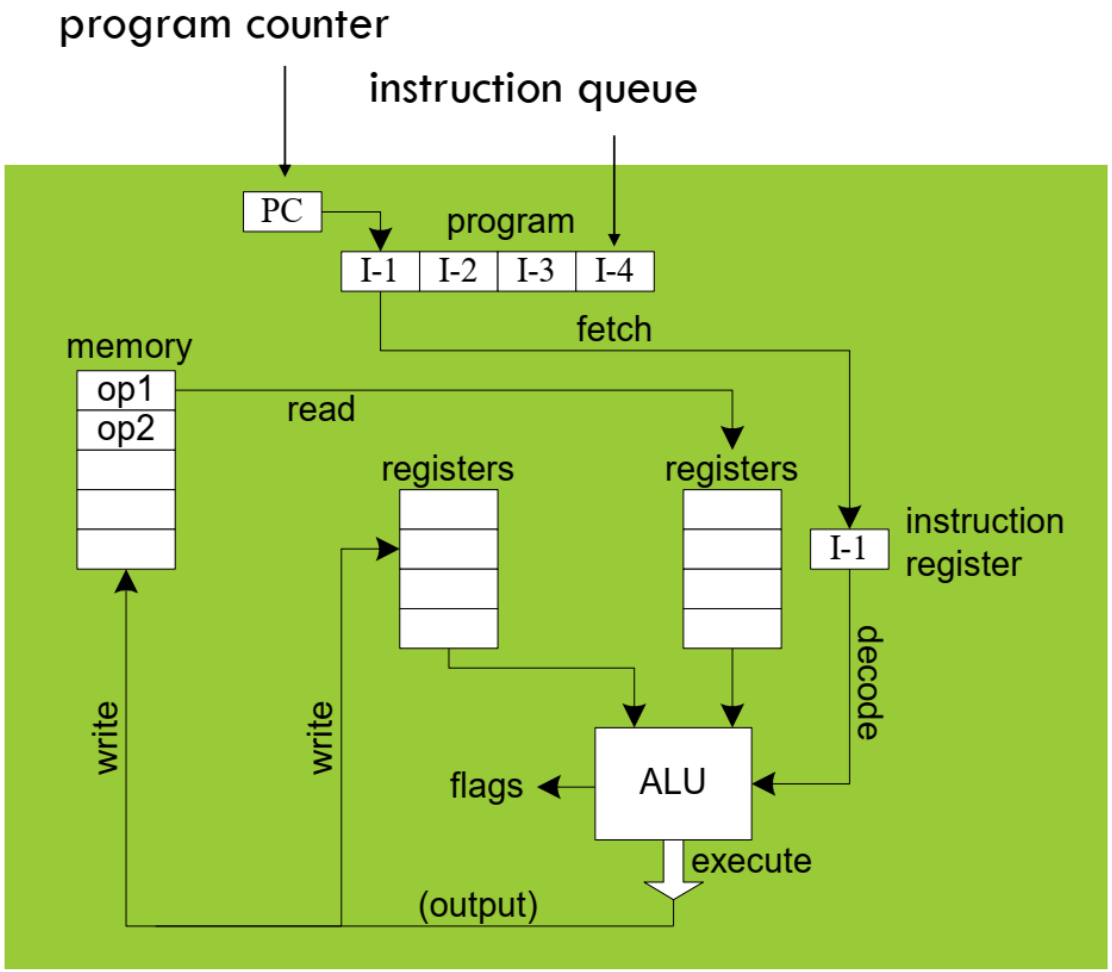
Fetch

Decode

Fetch operands

Execute

Store output

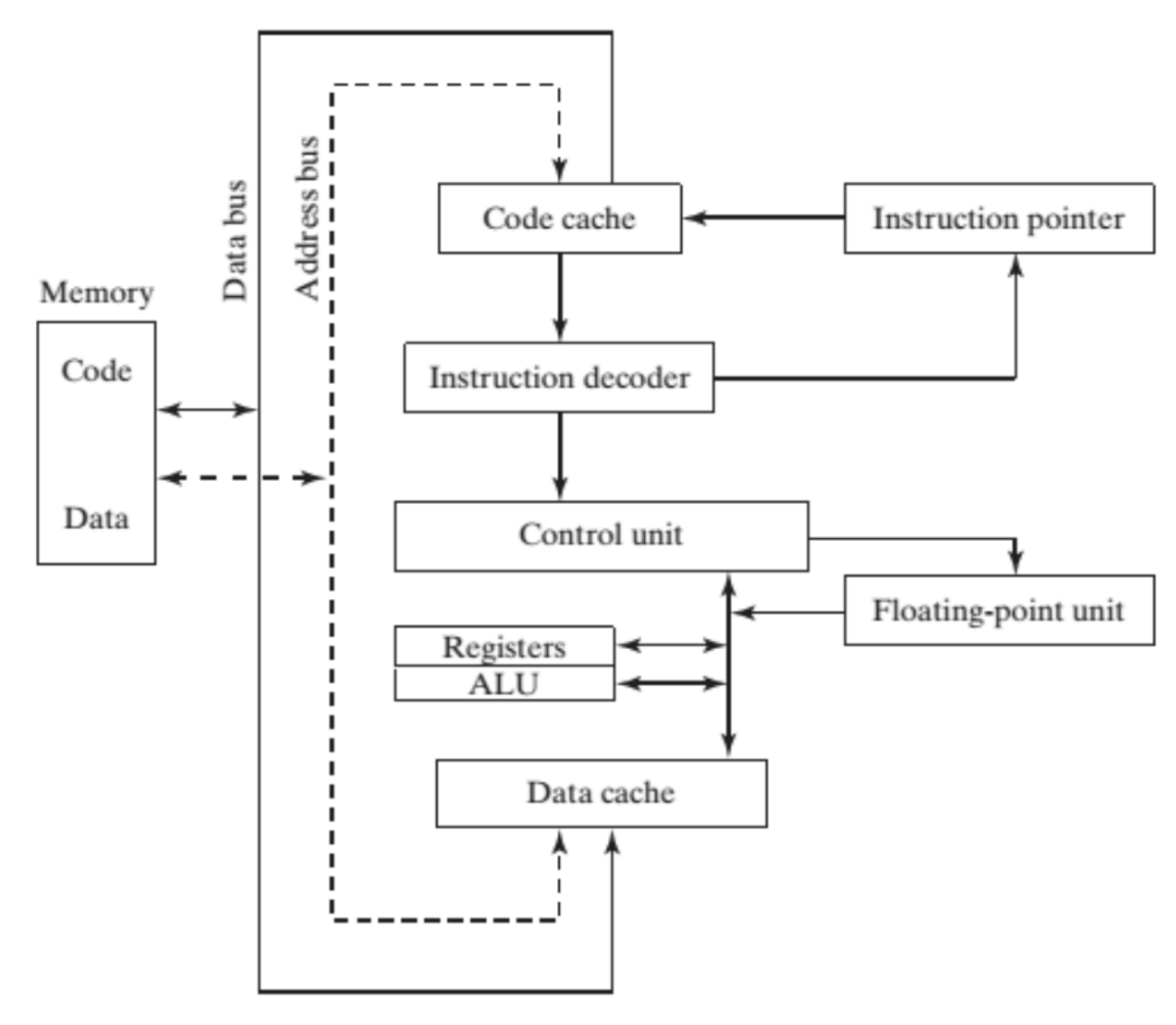


INSTRUCTION EXECUTION CYCLE

Instruction Execution Cycle

- The CPU go through a predefined sequence of steps to execute a machine instruction, called the instruction **execution cycle**.
- The **instruction pointer** (IP) register holds the address of the instruction we want to execute.
- Here are the steps to execute it:
  1. First, the CPU has to **fetch the instruction** from an area of memory called the *instruction queue*. It then increments the instruction pointer.
  2. Next, the CPU **decodes** the instruction by looking at its binary bit pattern.
     - This bit pattern might reveal that the instruction has operands (input values).
  3. If operands are involved, the CPU **fetches the operands** from registers and memory.
     - Sometimes, this involves address calculations.
  4. Next, the CPU **executes** the instruction, using any operand values it fetched during the earlier step. It also *updates a few status flags*, such as Zero, Carry, and Overflow.
  5. Finally, if an output operand was part of the instruction, the CPU **stores the result** of its execution in the operand.

Instruction Execution Cycle



Instruction Execution Cycle

- An **operand** is a value that is either an input or an output to an operation.
- For example, the expression Z = X + Y has **two input operands** (X and Y) and a single **output operand** (Z).
- In order **to read program instructions from memory**, an address is placed on the **address bus**.
- Next, the **memory controller** places the requested code on the data bus, making the code available inside the code cache.
- The **instruction pointer's** value determines which instruction will be executed next.
- The instruction is analyzed by the **instruction decoder**, causing the appropriate digital signals to be sent to the control unit, which coordinates the ALU and floating-point unit.
- **Control bus** carries signals that use the system clock to coordinate the transfer of data between different CPU components.

Reading from Memory

- As a rule, computers read memory much more slowly than they access internal registers.
- Reading a single value from memory involves four separate steps:

1. Place the address of the value you want to read on the address bus.
2. Assert (change the value of) the processor's RD (read) pin.
3. Wait one clock cycle for the memory chips to respond.
4. Copy the data from the data bus into the destination operand.

- Each of these steps generally requires a single clock cycle.

Cache

- CPU designers figured out that **computer memory creates a speed bottleneck** because most programs have to **access variables**.
- To reduce the amount of time spent in reading and writing memory the **most recently used instructions and data** are stored in high-speed memory called **cache**.
- The idea is that a program is more likely to want **to access the same memory and instructions repeatedly**, so cache keeps these values where they can be accessed quickly.
- When the CPU begins to execute a program, it **loads the next thousand instructions** (for example) into cache, on the assumption that these instructions will be needed in the near future.
- If there happens to be a **loop** in that block of code, the same instructions will be in cache.
- When the processor is able to find its data in cache memory, we call that a **cache hit**.
- On the other hand, if the CPU tries to find something in cache and it's not there, we call that a **cache miss**.

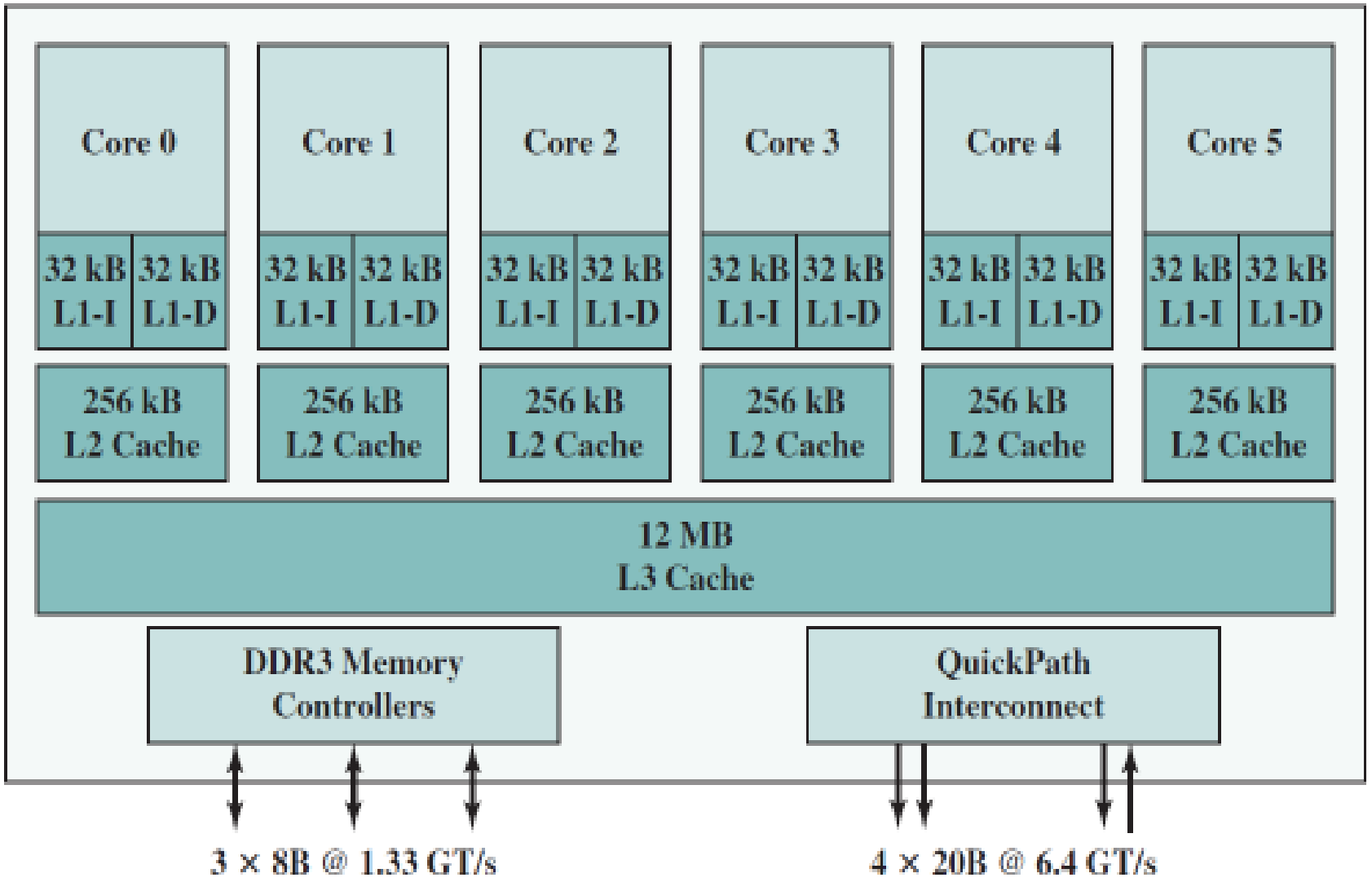
X86 family Cache types

- Cache memory for the x86 family comes in two types.

1. Level-1 cache (or primary cache) is stored right on the CPU.

2. Level-2 cache (or secondary cache) is a little bit slower, and attached to the CPU by a high-speed data bus.

The block diagram of Intel Core i7-990X is shown below. Explain the working of processor and how cache coherency is maintained? Also compute the aggregate speed of DDR3 Memory Controller and Quick Path Interconnect?



Intel Core i7-990X Block Diagram