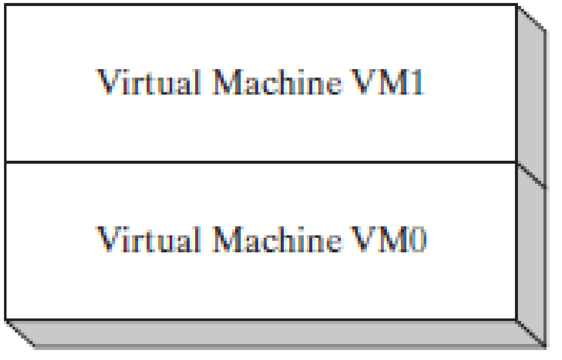
Virtual Machine Concept

- Virtual Machine Concept:
  - An effective way to explain how a computer's hardware and software are related is called the *virtual machine concept.*
- Specific Machine Levels

Virtual Machines

- Virtual machine is a software program that emulates the functions of some other physical or virtual computer.
- 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



Irvine, Kip R. Assembly Language for Intel-Based Computers 6/e, 2010.

Virtual Machines (Continue…)

- Programs written in L1 can run in two different ways:
  - Translation – L1 program is completely translated into an L0 program (which then runs on the computer hardware)
  - Interpretation – L0 program interprets and executes L1 instructions one by one

Translating Languages

English: Display the sum of A times B plus C.

C++: cout << (A * B + C);

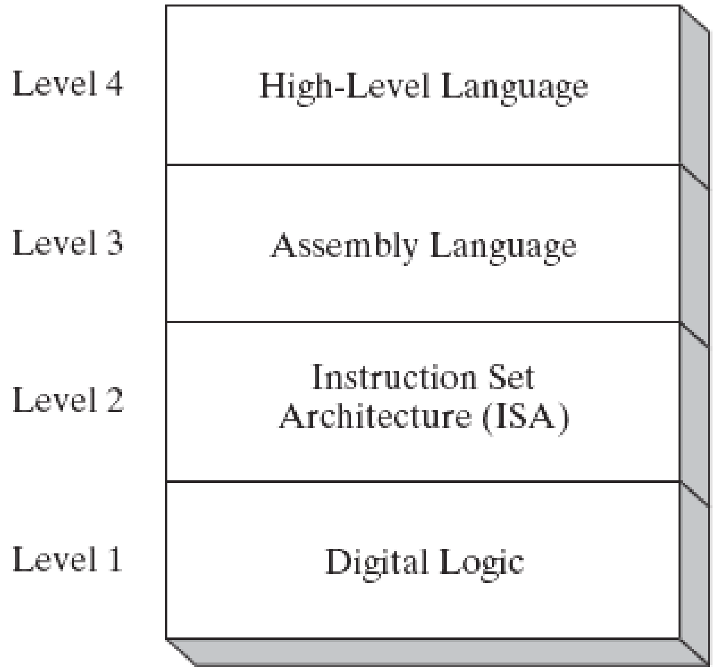
Assembly Language:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Specific Machine Levels

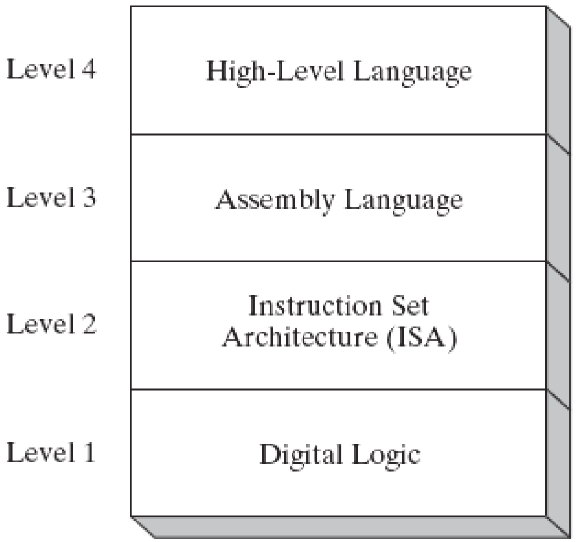


(descriptions of individual levels follow . . .)

High-Level Language

Level 4

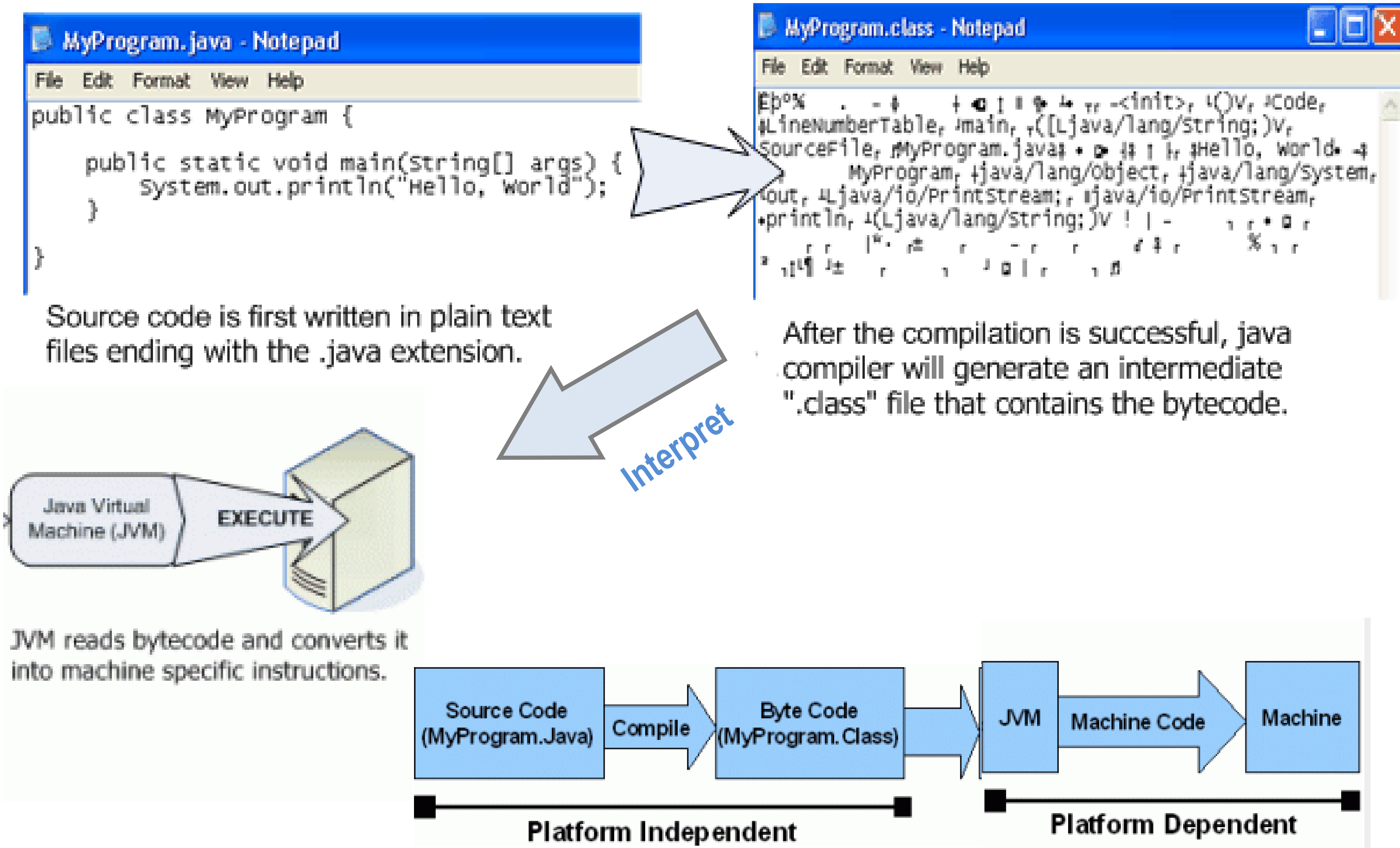
- Application-oriented languages
  - C++, Java, Pascal, Visual Basic . . .
- Programs compile into assembly language (Level 3)



Irvine, Kip R. Assembly Language for Intel-Based Computers 6/e, 2010.

High-Level Language

- 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 code.
- Java byte code is executed at runtime by a program known as a *Java virtual machine (JVM)*.

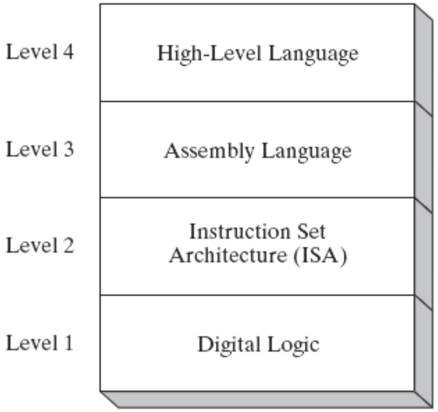


Java code / Bytecode is always the same on different OS.

- That makes java program as platform independent.

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

Assembly Language



Level 3

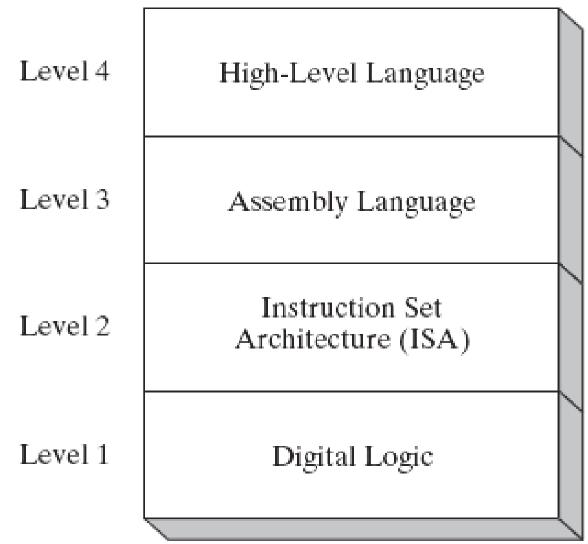
- Instruction mnemonics that have a one-to-one correspondence to machine language
- Programs are translated into Instruction Set Architecture Level - machine language (Level 2)

- 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

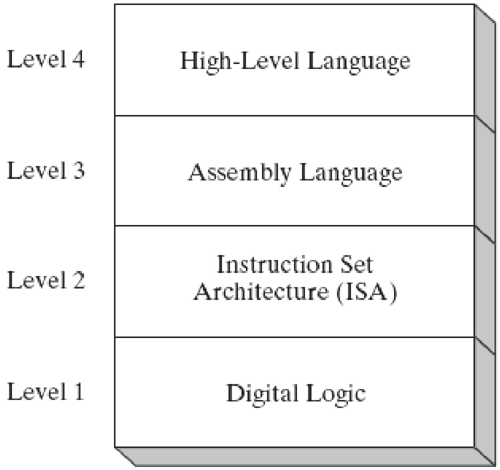
- Also known as conventional machine language
- Executed by Level 1 (Digital Logic)



Digital Logic

Level 1

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



Basic Microcomputer Design

Figure 2–1 Block Diagram of a Microcomputer.

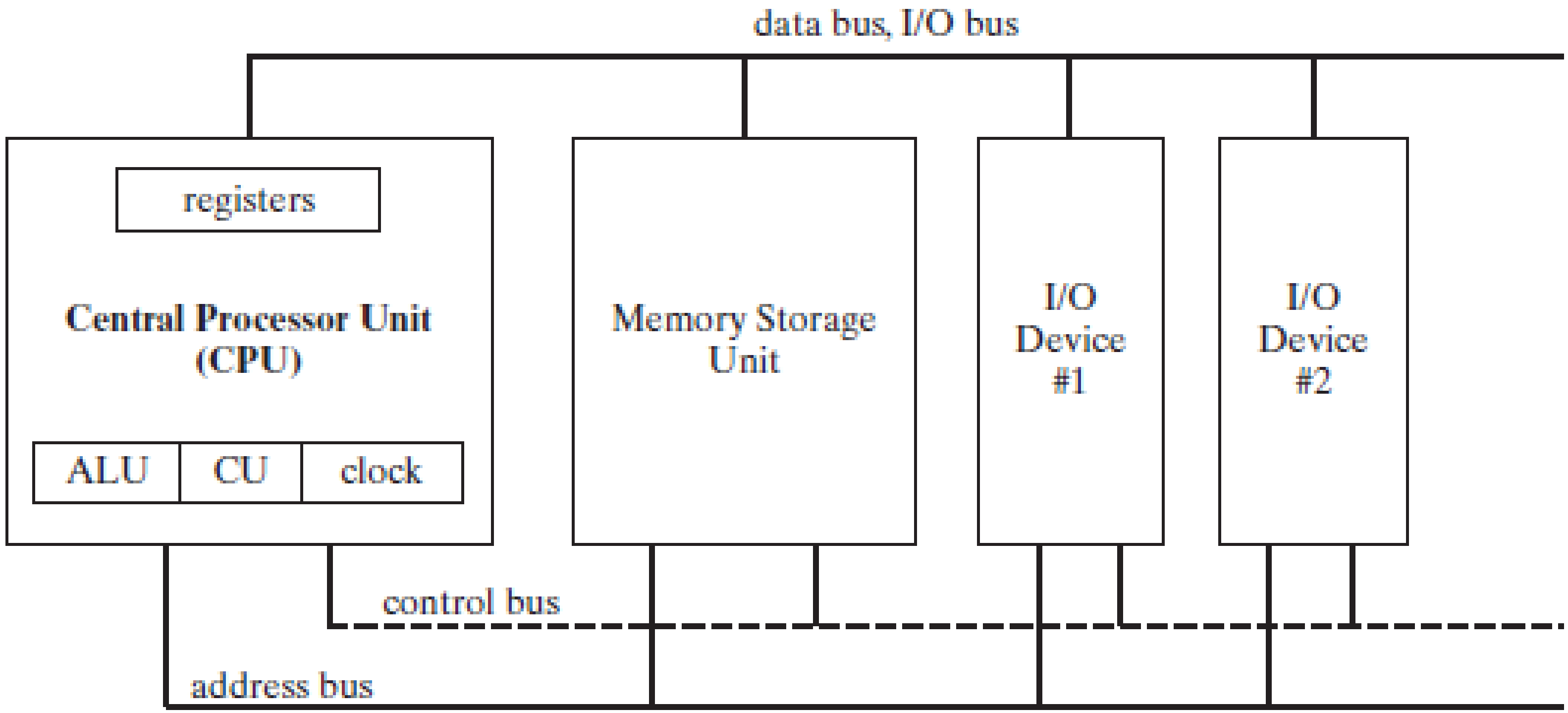
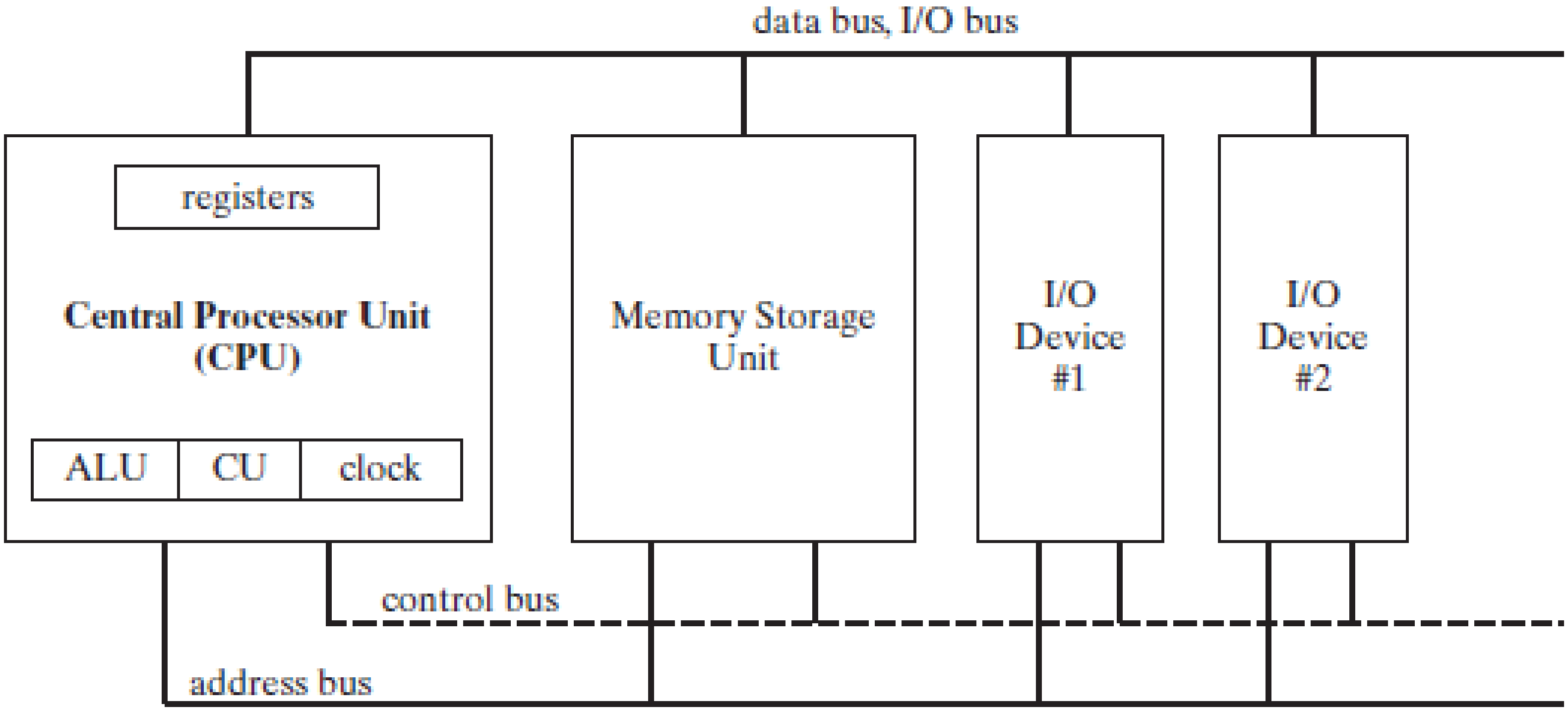


FIGURE 2–1 Block Diagram of a Microcomputer.



The *central processor unit* (CPU): Where calculations and logic operations are performed:

- 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.

Figure 2–1 Block Diagram of a Microcomputer.

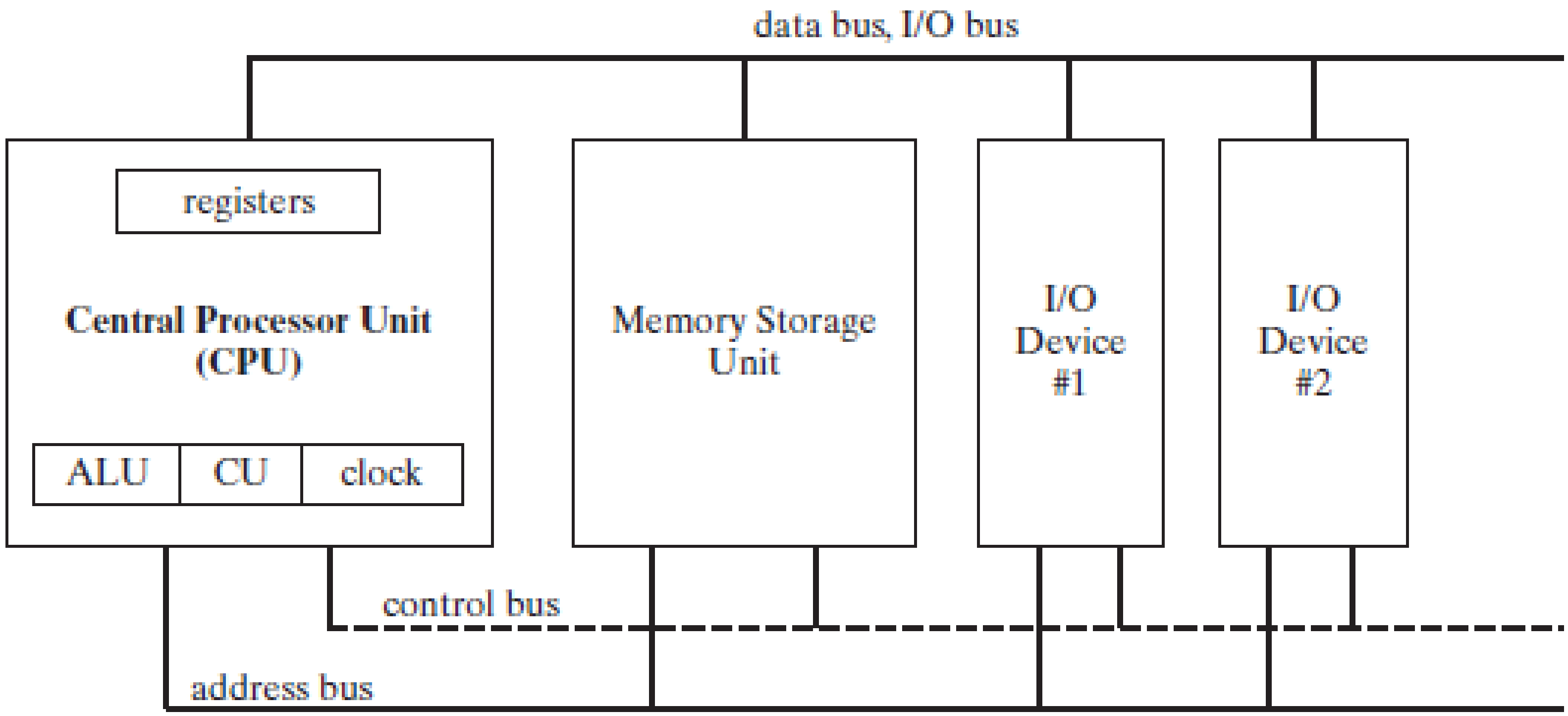
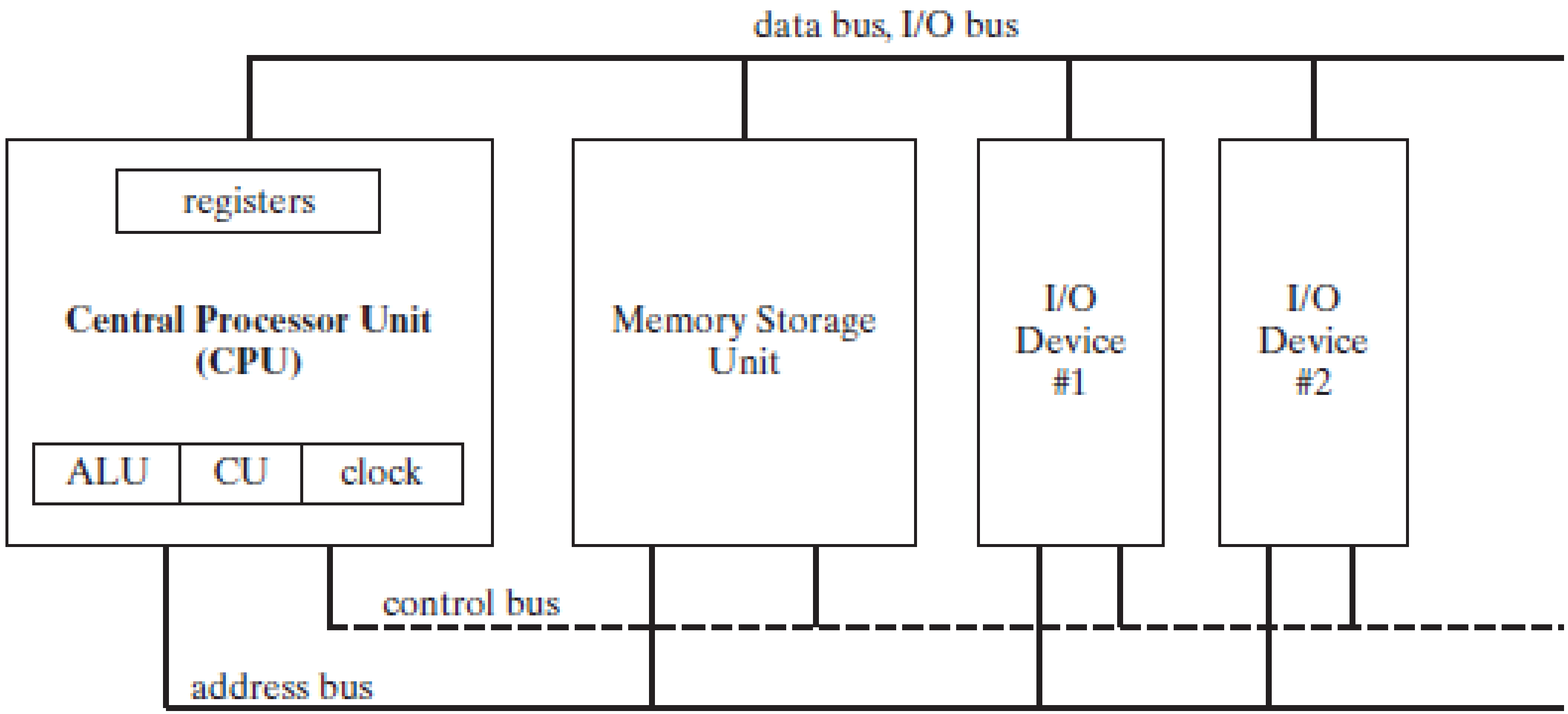


FIGURE 2–1 Block Diagram of a Microcomputer.



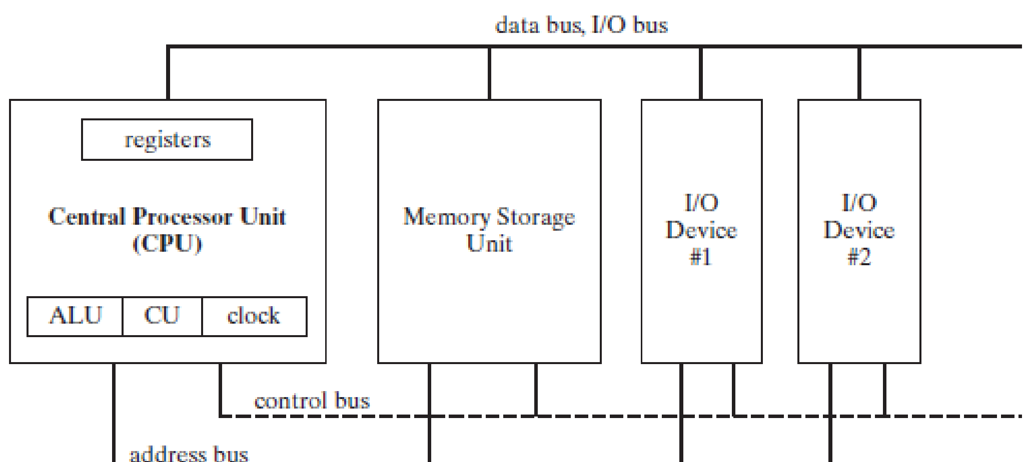
The storage unit receives requests for data from the CPU, transfers data from random access memory (RAM) to the CPU, and transfers data from the CPU into memory.

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.

FIGURE 2–1 Block Diagram of a Microcomputer.



Clock and Clock Cycles

- Clock synchronizes all CPU and BUS operations
  - Clock is used to trigger events
  - Clock cycles measure time of a single operation
- 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

- 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.

Instruction Execution Cycle

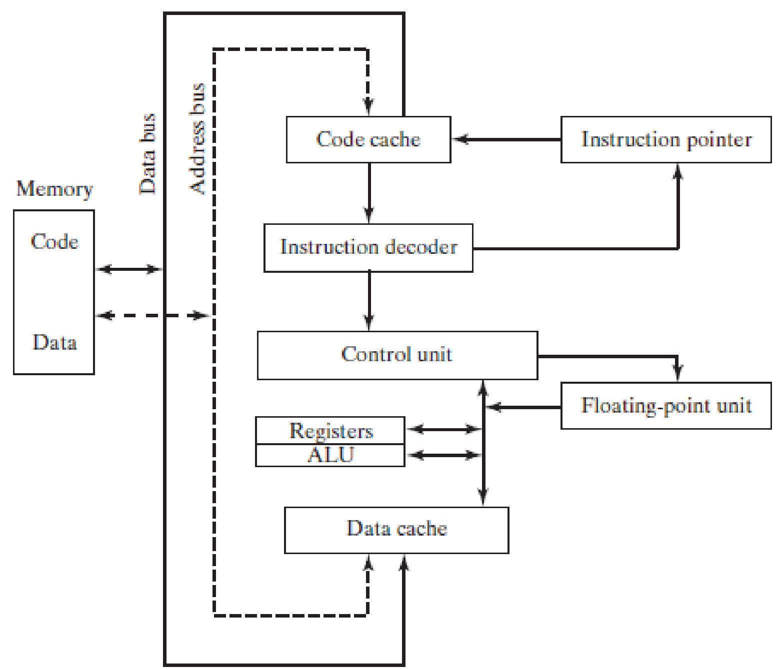
Here are the steps to execute it:

1. First, the CPU **fetch the instruction** from 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

- 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

Figure 2–2 Simplified CPU block diagram.



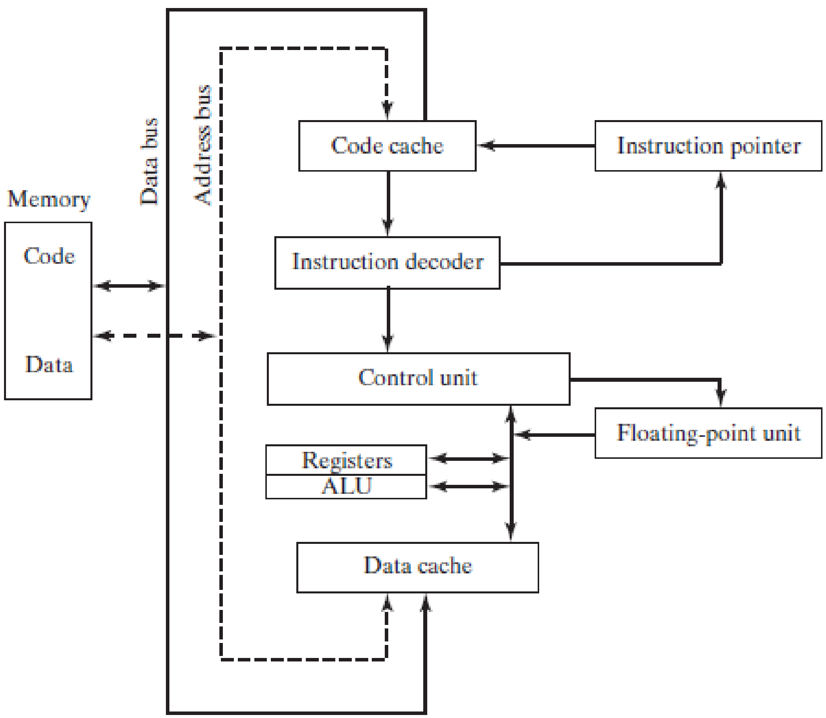
- The memory controller places the requested code on the data bus
  - Making the code available inside the code cache

Instruction Execution Cycle

- The instruction pointer's value determines which instruction will be executed next.
- The instruction is analyzed by the **instruction decoder**
  - Causing appropriate digital signals to be sent to the Control Unit
  - Control Unit coordinates with 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.

Figure 2–2 Simplified CPU block diagram.



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 few 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 (1 of 2)

- 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 want to access the same memory and instructions repeatedly
  - so cache keeps these values where they can be accessed quickly

Cache (2 of 2)

- When the CPU begins to execute a program, it loads the next thousand instructions (for example) into cache
  - The assumption is that these instructions will be needed in the near future.
- If it 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

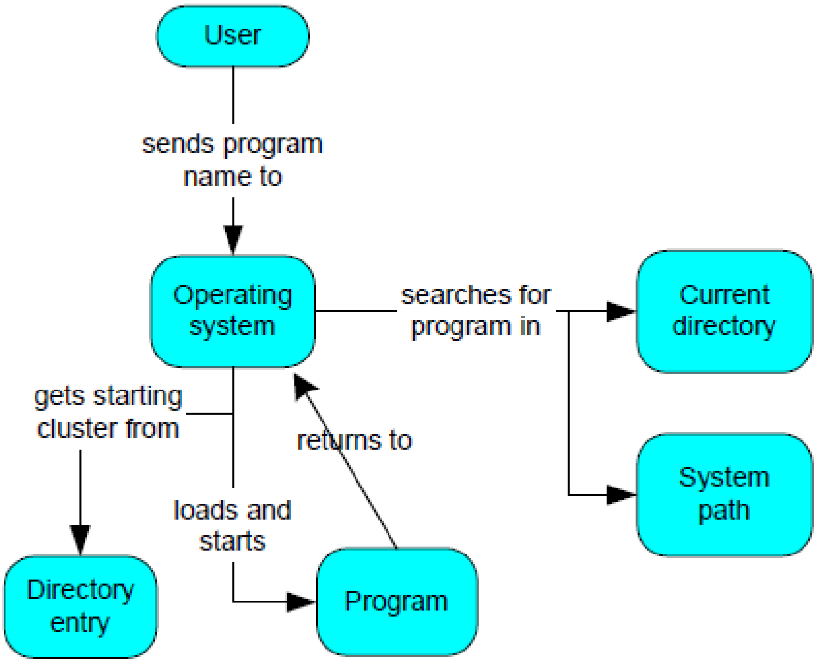
- Cache memory for the x86 family comes in two types.
  - *Level-1 cache* (or *primary cache*) is stored right on the CPU.
  - *Level-2 cache* (or *secondary cache*) is a little bit slower, and attached to the CPU by a high-speed data bus.

Why cache memory is faster than conventional RAM?

- It's because cache memory is constructed from a special type of memory chip called *static RAM*
  - It's expensive, but it does not have to be constantly refreshed in order to keep its contents
- Conventional memory, known as *dynamic RAM,* refreshed constantly
  - It's much slower, and cheaper

Loading and Executing a Program (1 of 3)

- The operating system (OS) searches for the program's filename in the current disk directory
  - If it cannot find the name there, it searches a predetermined list of directories (called *paths*) for the filename
  - If the OS fails to find the program filename, it issues an error message



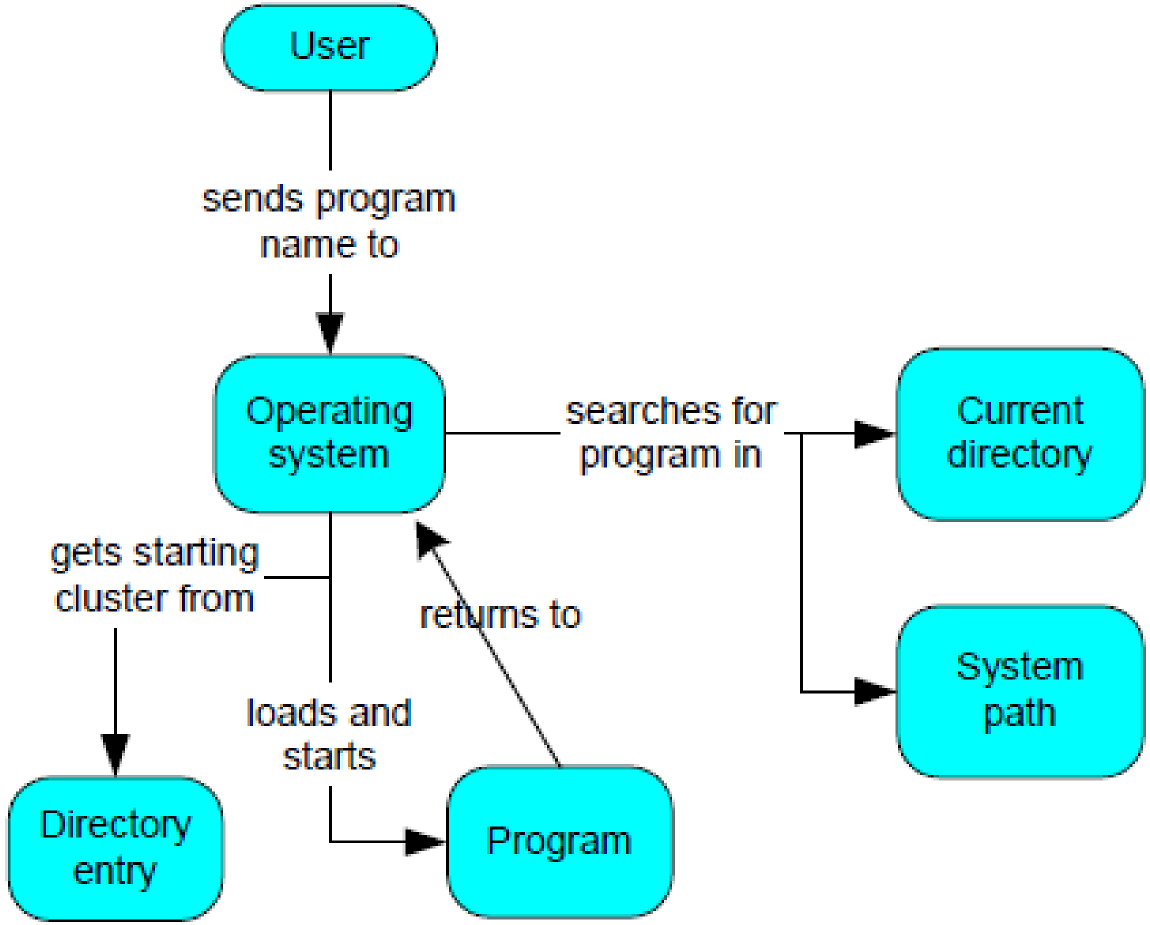
Loading and Executing a Program (2 of 3)

- If the program file is found, the OS retrieves basic information about the program's file from the disk directory
  - including the file size and
  - its physical location on the disk drive.
- The OS determines the next available location in memory and loads the program file into memory
  - It allocates a block of memory to the program and enters information about the program's size and location into a table (sometimes called a *descriptor table*).
  - The OS also adjust the values of pointers within the program so they contain addresses of program data.

Loading and Executing a Program (3 of 3)

- The OS begins execution of the program's first machine instruction (its entry point).
- As soon as the program begins running, it is called a *process*.
- The OS assigns the process an identification number (*process ID*), which is used to keep track of it while running.
- It is the OS's job to track the execution of the process and to respond to requests for system resources.
  - Examples of resources are memory, disk files, and input-output devices.
- When the process ends, it is removed from memory.

How a Program Runs



Mode of Operations

x86 processors have three primary modes of operation: protected mode, real-address mode, and system management mode.

- ❖ Real-Address mode (original mode provided by 8086)
  - ✧ Only 1 MB of memory can be addressed, from 0 to FFFFF (hex)
  - ✧ Programs can access any part of main memory
  - ✧ MS-DOS runs in real-address mode
- ❖ Implements the programming environment of the Intel 8086 processor
- ❖ This mode is available in Windows 98, and can be used to run an MS-DOS program that requires direct access to system memory and hardware devices
- ❖ Programs running in real-address mode can cause the operating system to crash (stop responding to commands)

Slide Credit: Dr. Aiman El-Maleh & Dr. Kip Irvine

Mode of Operations

- Protected mode (introduced with the 80386 processor)
  - Each program can address a maximum of 4 GB of memory
  - The operating system assigns memory to each running program
  - Programs are prevented from accessing each other's memory *(segments)*
  - Native mode used by Windows NT, 2000, XP, and Linux

Slide Credit: Dr. Aiman El-Maleh & Dr. Kip Irvine

Mode of Operations

Virtual 8086 mode: A sub-mode, *virtual-8086*, is a special case of protected mode

- Processor runs in protected mode, and creates a virtual 8086 machine with 1 MB of address space for each running program such as MS-DOS

  - If an MS-DOS program crashes or attempts to write data into the system memory area, it will not affect other programs running at the same time

  - Windows XP can execute multiple separate virtual-8086 sessions at the same time

Slide Credit: Dr. Aiman El-Maleh & Dr. Kip Irvine

Mode of Operations

❖ System Management Mode:

- Provides a mechanism for implementation power management and system security
  - Manage system safety functions, such as shutdown on high CPU temperature and turning the fans on and off
  - Handle system events like memory or chipset errors