COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr

CHAPTER

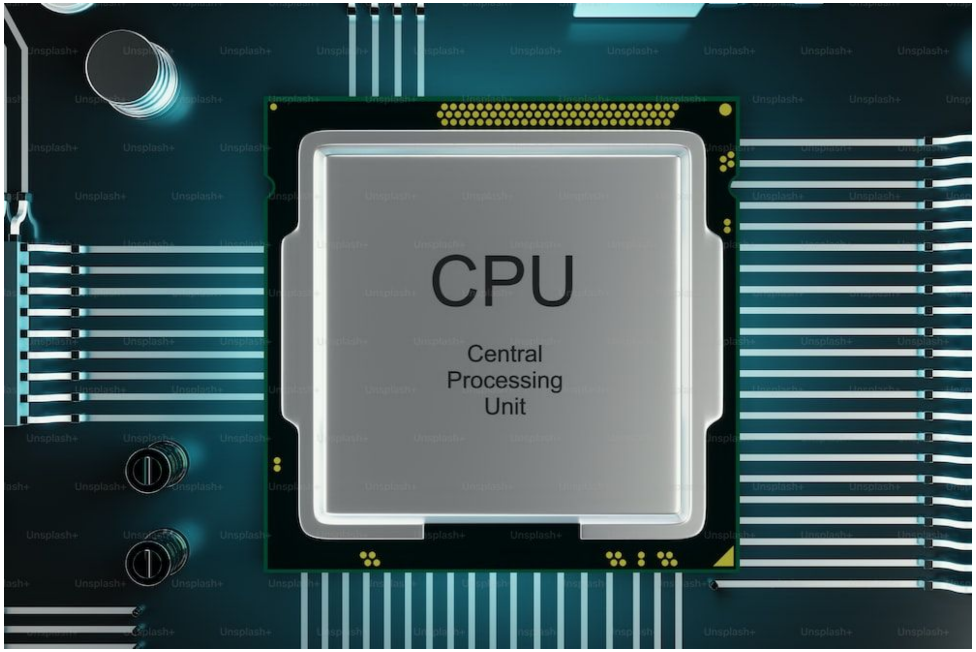
2 Data Manipulation

In this chapter we will learn how a computer manipulates data and communicates with peripheral devices such as printers and keyboards. In doing so, we will explore the basics of computer architecture and learn how computers are programmed by means of encoded instructions, called machine language instructions.

Computer Architecture

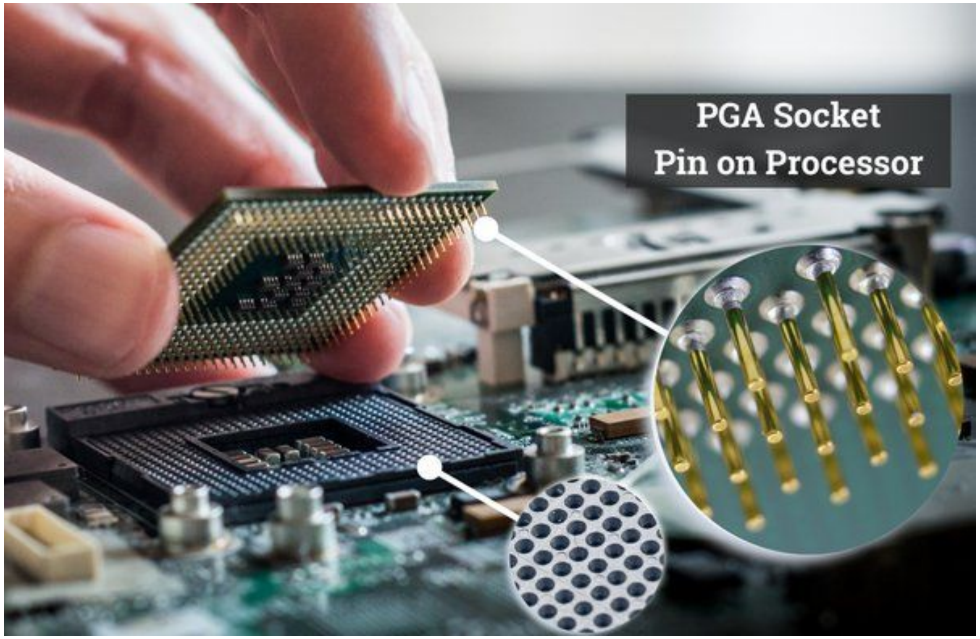
The circuitry in a computer that controls the manipulation of data is called the **central processing unit**, or **CPU**

- (often referred to as merely the processor).

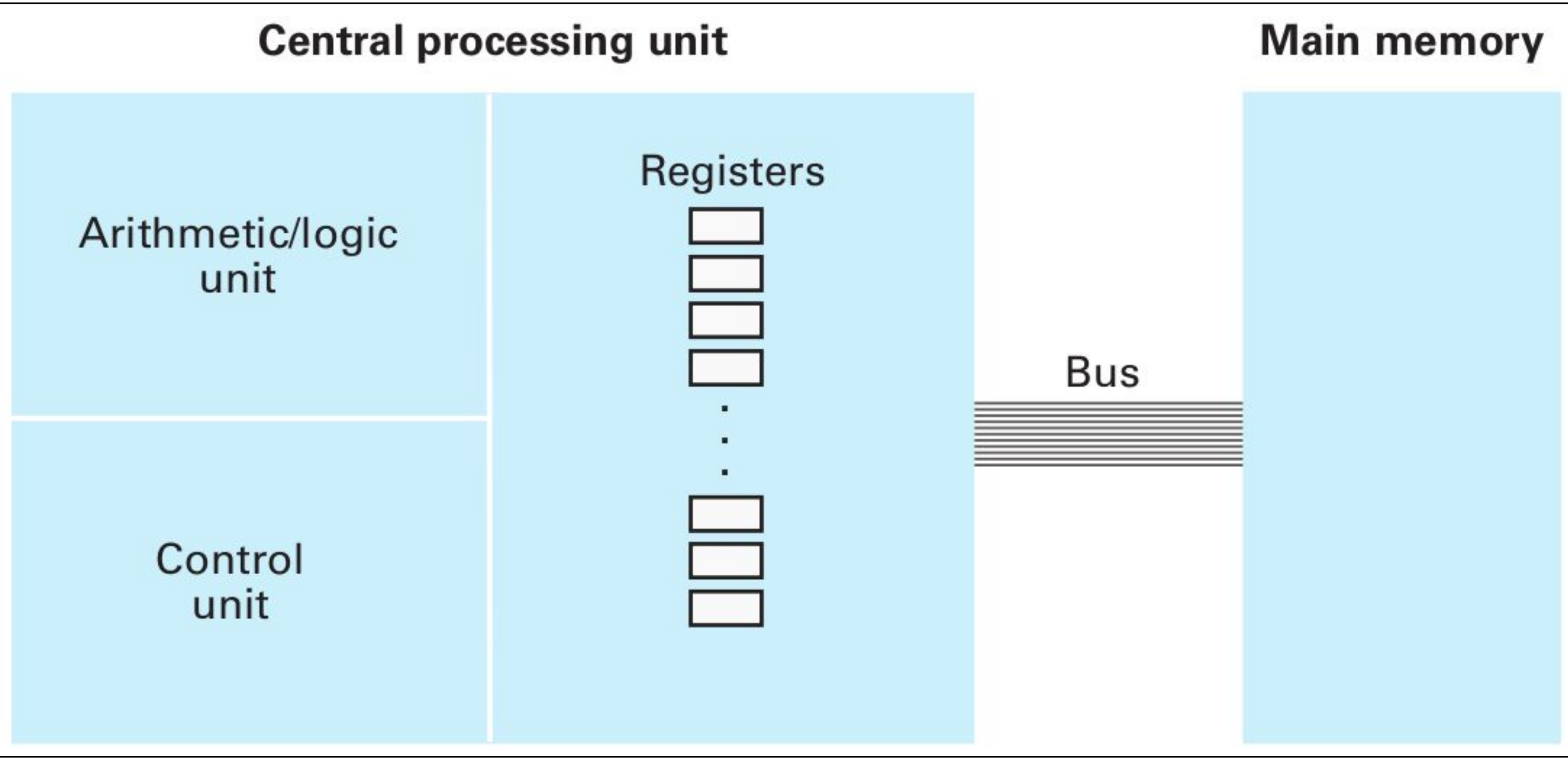


Computer Architecture

The CPUs found in today’s desktop computers and notebooks are packaged as **small flat squares** (approximately two inches by two inches) whose connecting pins plug into a socket mounted on the machine’s main circuit board (called the **motherboard**).

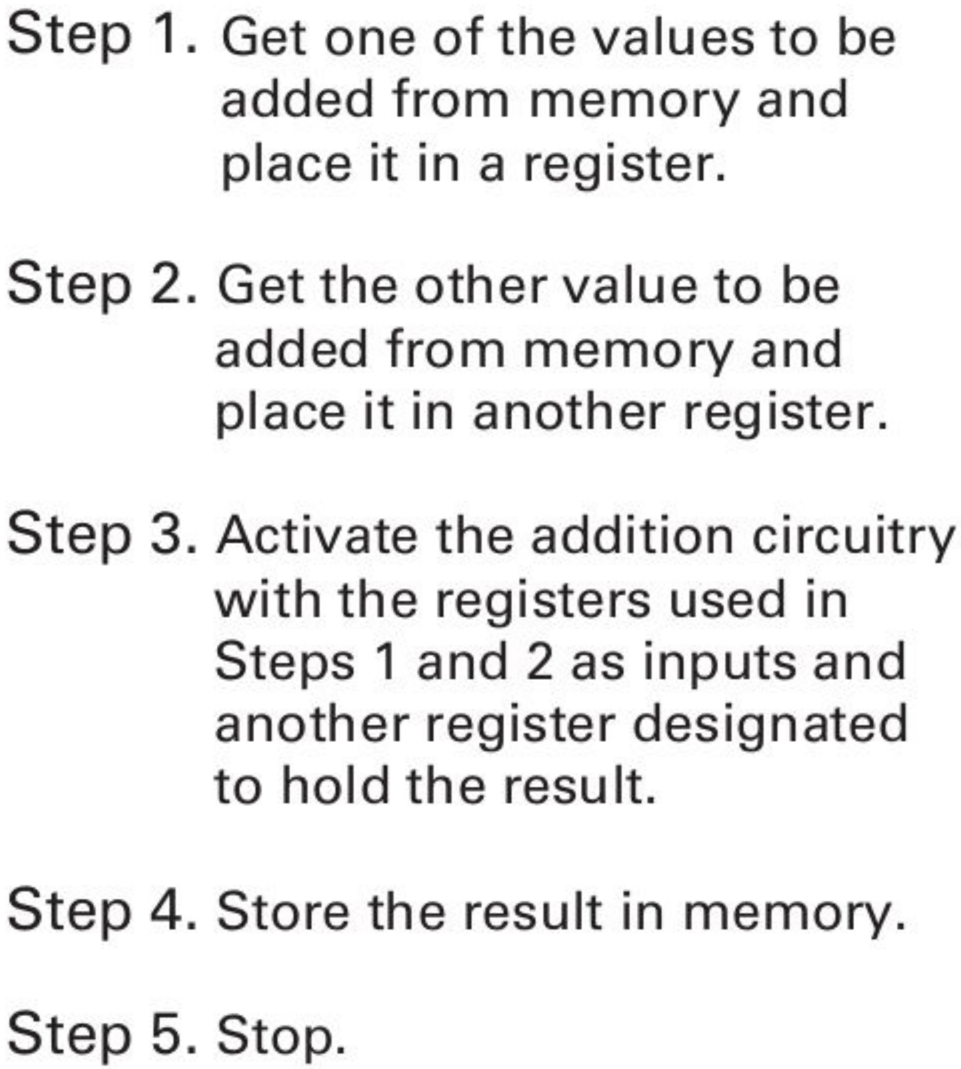


CPU Basics


Central processing unit
Main memory
Arithmetic/logic unit
Registers
Control unit
Bus

CPU Basics

Algorithm of adding values stored in memory



Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Cache Memory

It is instructive to compare the memory facilities within a computer in relation to their functionality. Registers are used to hold the data immediately applicable to the operation at hand; main memory is used to hold data that will be needed in the near future; and mass storage is used to hold data that will likely not be needed in the immediate future. Many machines are designed with an additional memory level, called cache memory. **Cache memory** is a portion (perhaps several hundred KB) of high-speed memory located within the CPU itself. In this special memory area, the machine attempts to keep a copy of that portion of main memory that is of current interest. In this setting, data transfers that normally would be made between registers and main memory are made between registers and cache memory. Any changes made to cache memory are then transferred collectively to main memory at a more opportune time. The result is a CPU that can execute its machine cycle more rapidly because it is not delayed by main memory communication.

Stored-program concept

A program, just like data, can be encoded and stored in main memory.

If the control unit is designed to extract the program from memory, decode the instructions, and execute them, the program that the machine follows can be changed merely by changing the contents of the computer’s memory instead of rewiring the CPU.

The idea of storing a computer’s program in its main memory is called the **stored-program concept** and has become the standard approach used today.

Machine Language

CPUs are designed to recognize instructions encoded as bit patterns.

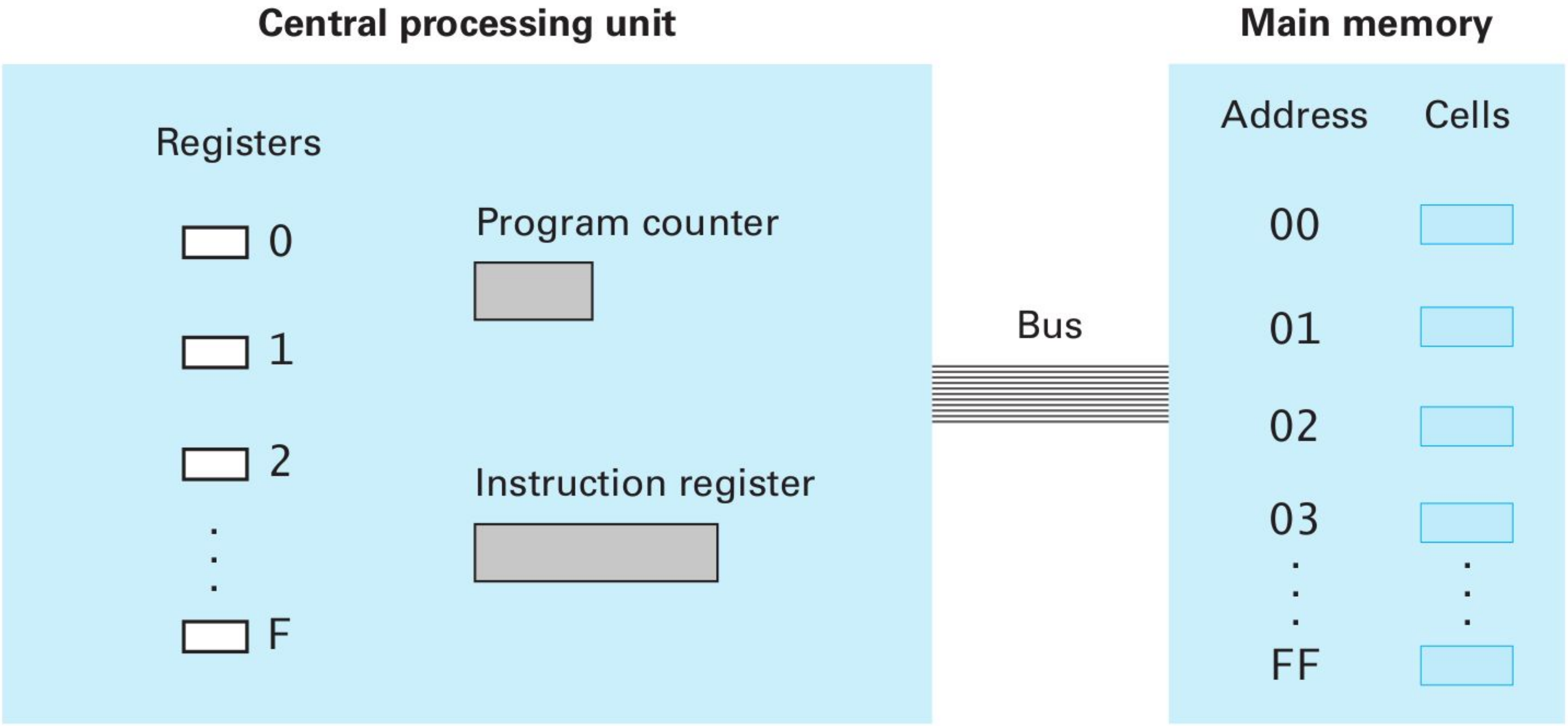
- This collection of instructions along with the encoding system is called the machine language.
- An instruction expressed in this language is called a machine-level instruction.

Illustrative Machine Language

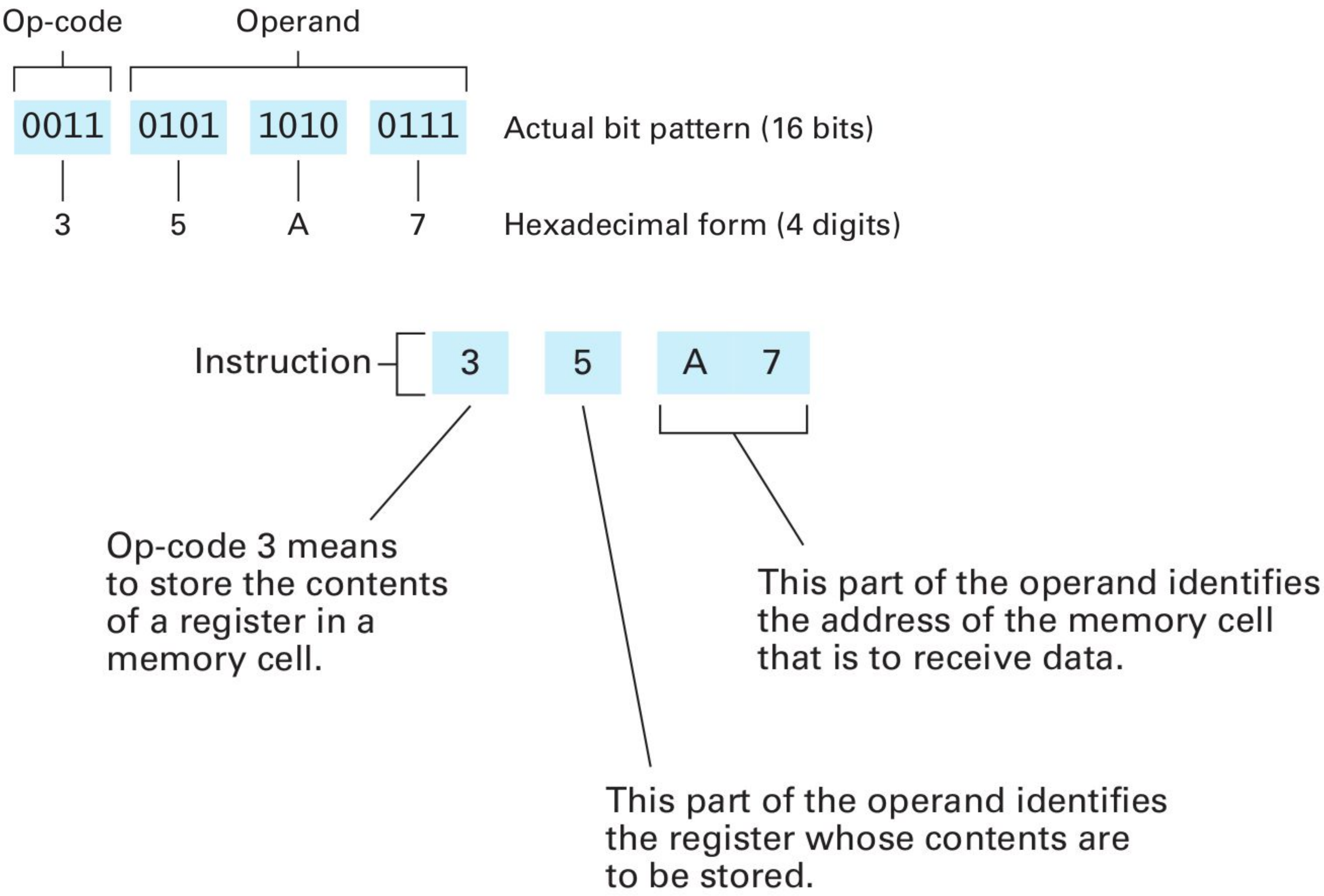
A machine’s instructions can be categorized into three groupings:

- the data transfer group,
  - STORE, LOAD, I/O instructions
- the arithmetic/logic group, and
  - ADD, SHıFT, ROTATE
  - AND, OR, and XOR
- the control group
  - JUMP

Illustrative Machine Language



Decoding the instruction 35A7



Op-code 3 means to store the contents of a register in a memory cell.

This part of the operand identifies the address of the memory cell that is to receive data.

This part of the operand identifies the register whose contents are to be stored.

STORE the bit pattern found in register 5 in the memory cell whose address is A7.

	Encoded instructions	Translation
Step 1. Get one of the values to be added from memory and place it in a register.	156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
Step 2. Get the other value to be added from memory and place it in another register.	166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.	5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
Step 4. Store the result in memory.	306E	Store the contents of register 0 in the memory cell at address 6E.
Step 5. Stop.	C000	Halt.

Program Execution

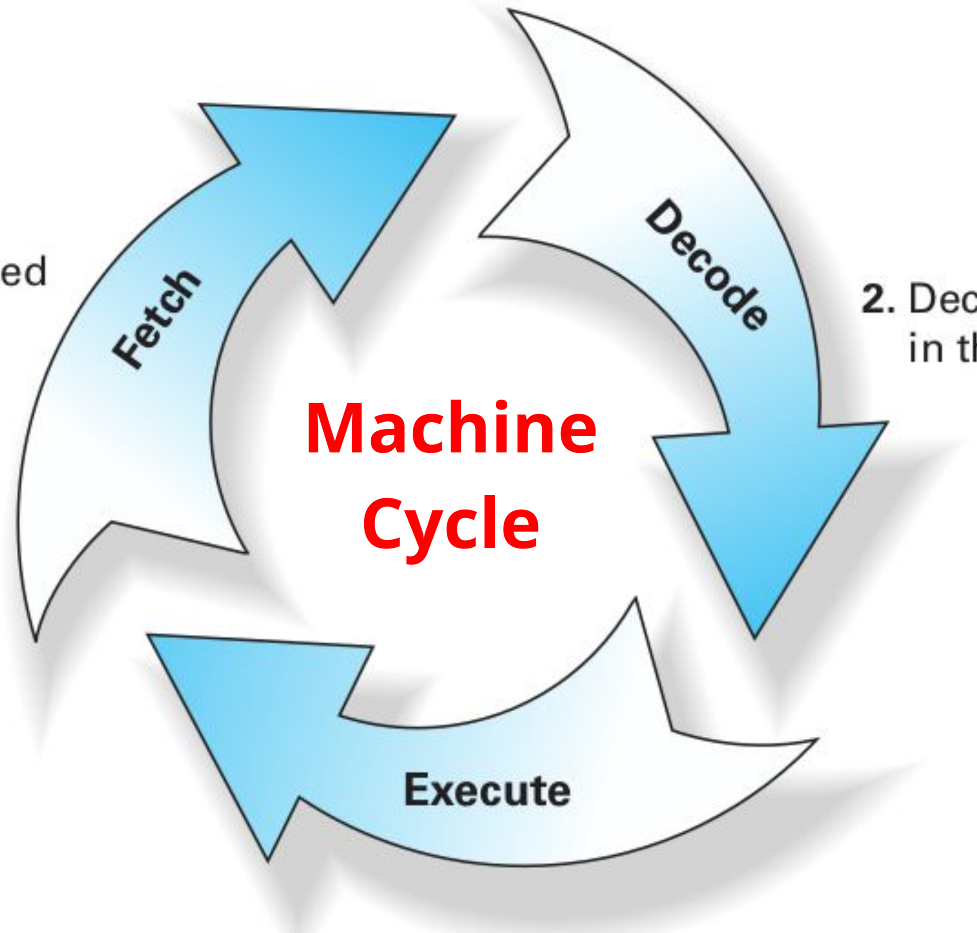
To understand how the overall execution process takes place, it is necessary to consider two of the special purpose registers within the CPU:

- **Instruction register**: used to hold the instruction being executed, thereby serving as the machine's way of keeping track of where it is in the program
- **Program counter**: contains the address of the next instruction to be executed

The CPU performs its job by continually repeating an algorithm that guides it through a three-step process known as the **machine cycle**.

Program Execution

1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.

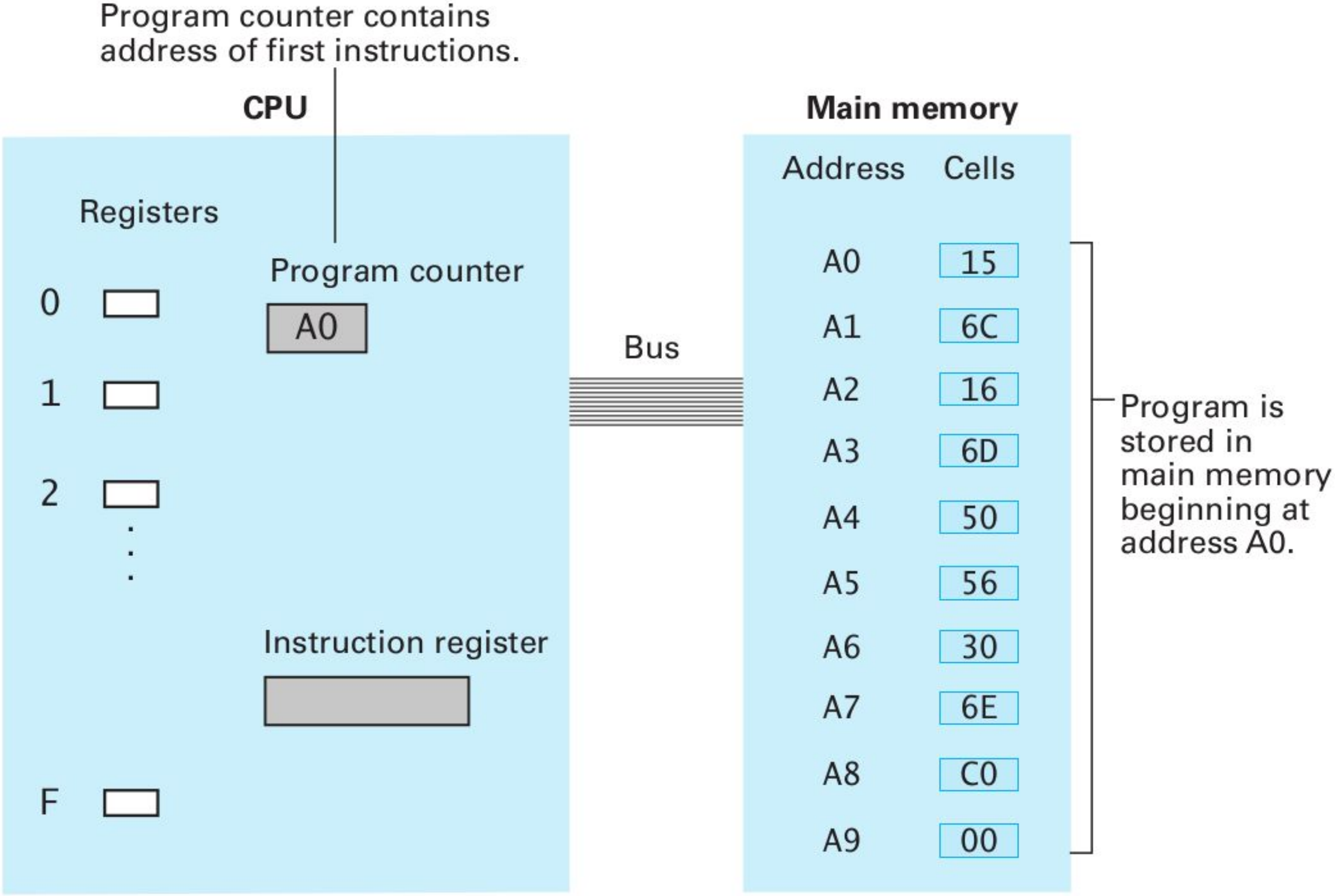


2. Decode the bit pattern in the instruction register.

3. Perform the action required by the instruction in the instruction register.

Program Execution

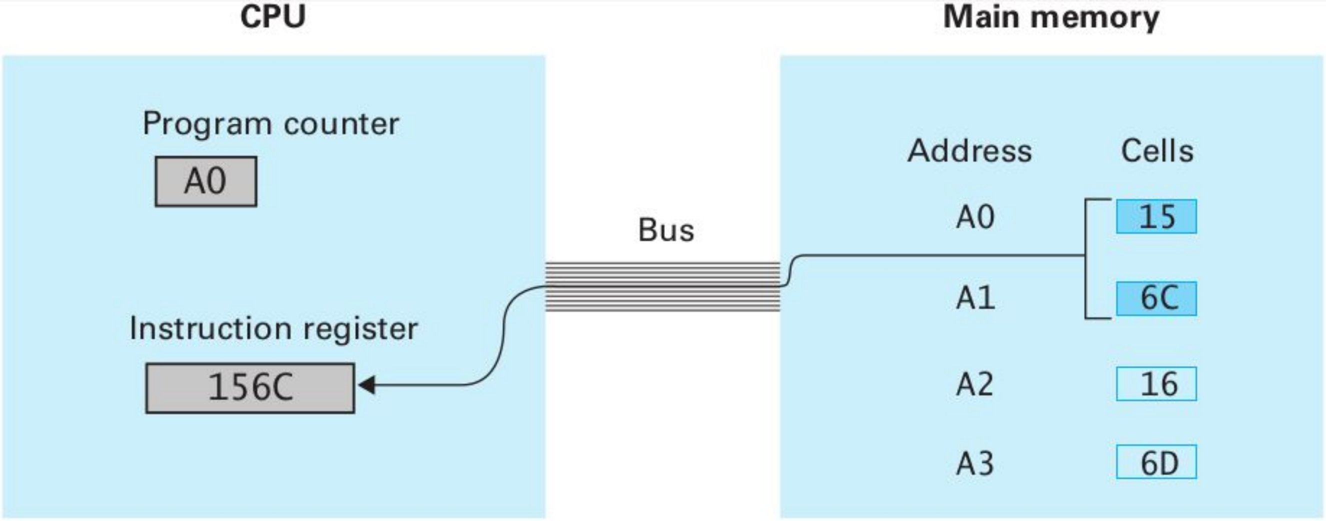
The program stored in main memory ready for execution



Program counter contains address of first instructions.

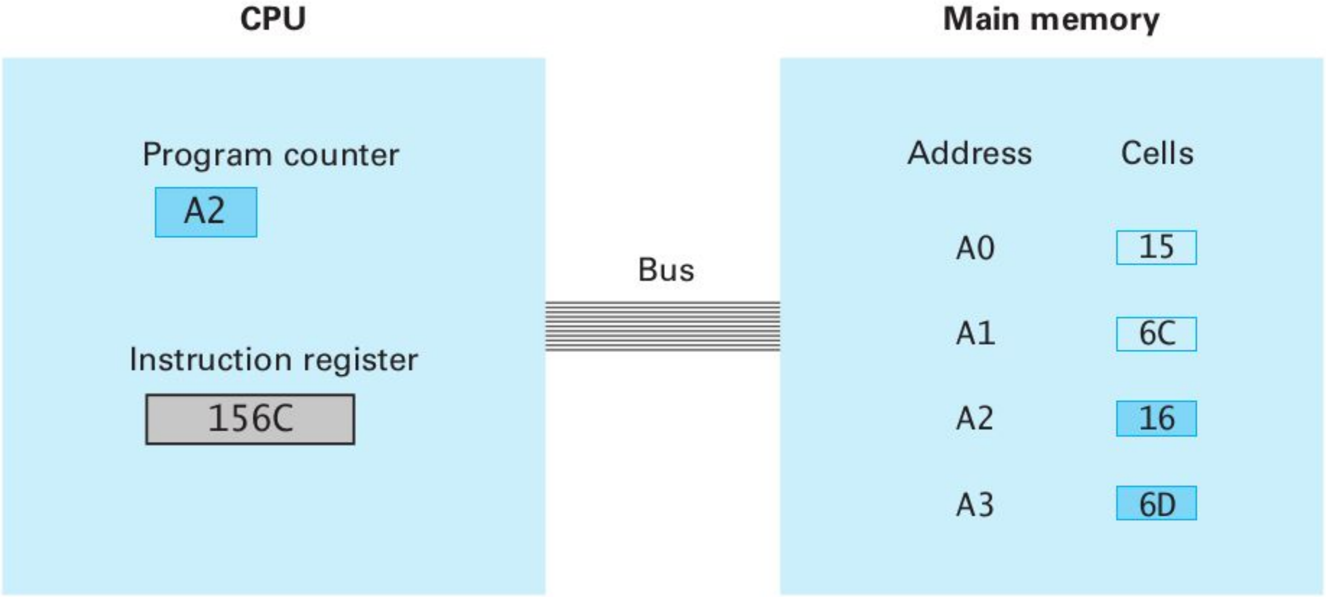
Address	Cells
A0	15
A1	6C
A2	16
A3	6D
A4	50
A5	56
A6	30
A7	6E
A8	C0
A9	00

Program is stored in main memory beginning at address A0.



Program Counter: A2
Instruction Register: 156C

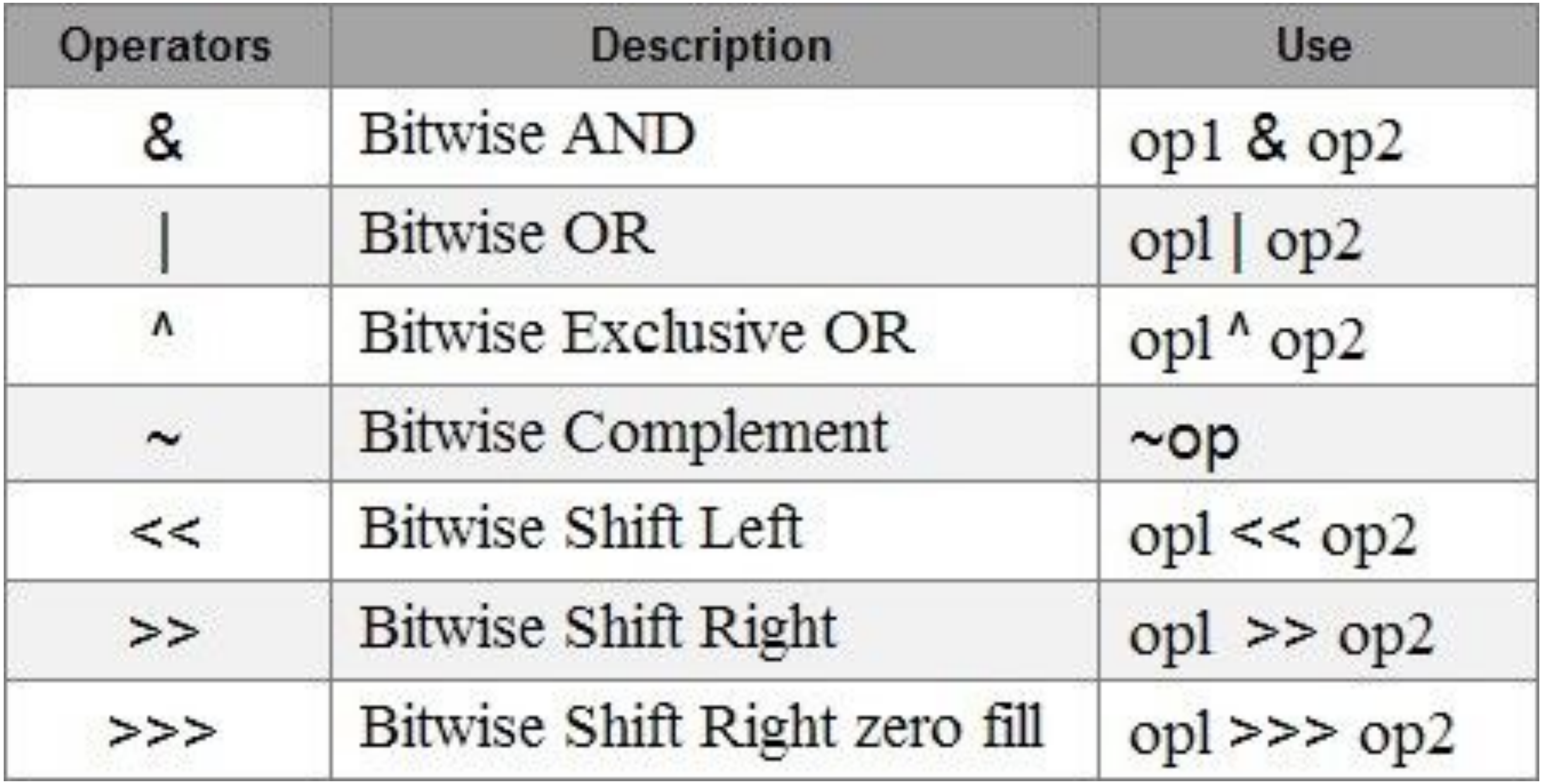
a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.



Program Counter: A4
Instruction Register: 166D

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

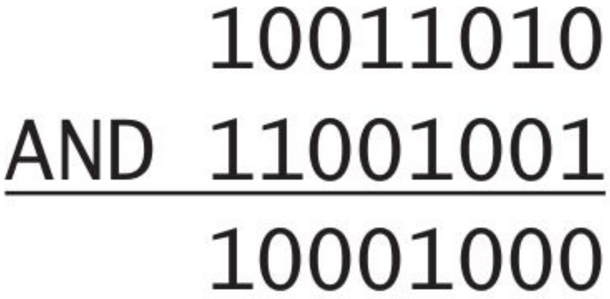
Arithmetic/Logic Instructions



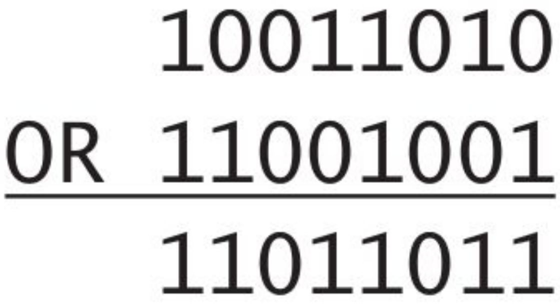
Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	opl | op2
^	Bitwise Exclusive OR	opl ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	opl << op2
>>	Bitwise Shift Right	opl >> op2
>>>	Bitwise Shift Right zero fill	opl >>> op2

Arithmetic/Logic Instructions

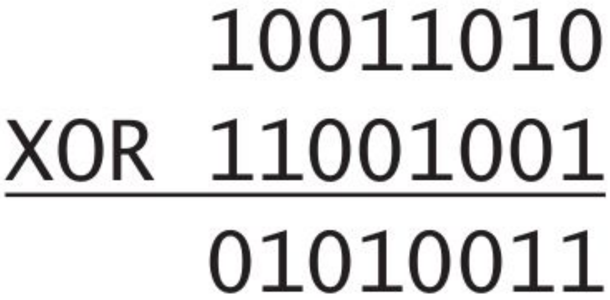
Logic Operations



```
     10011010
AND  11001001
     10001000
```



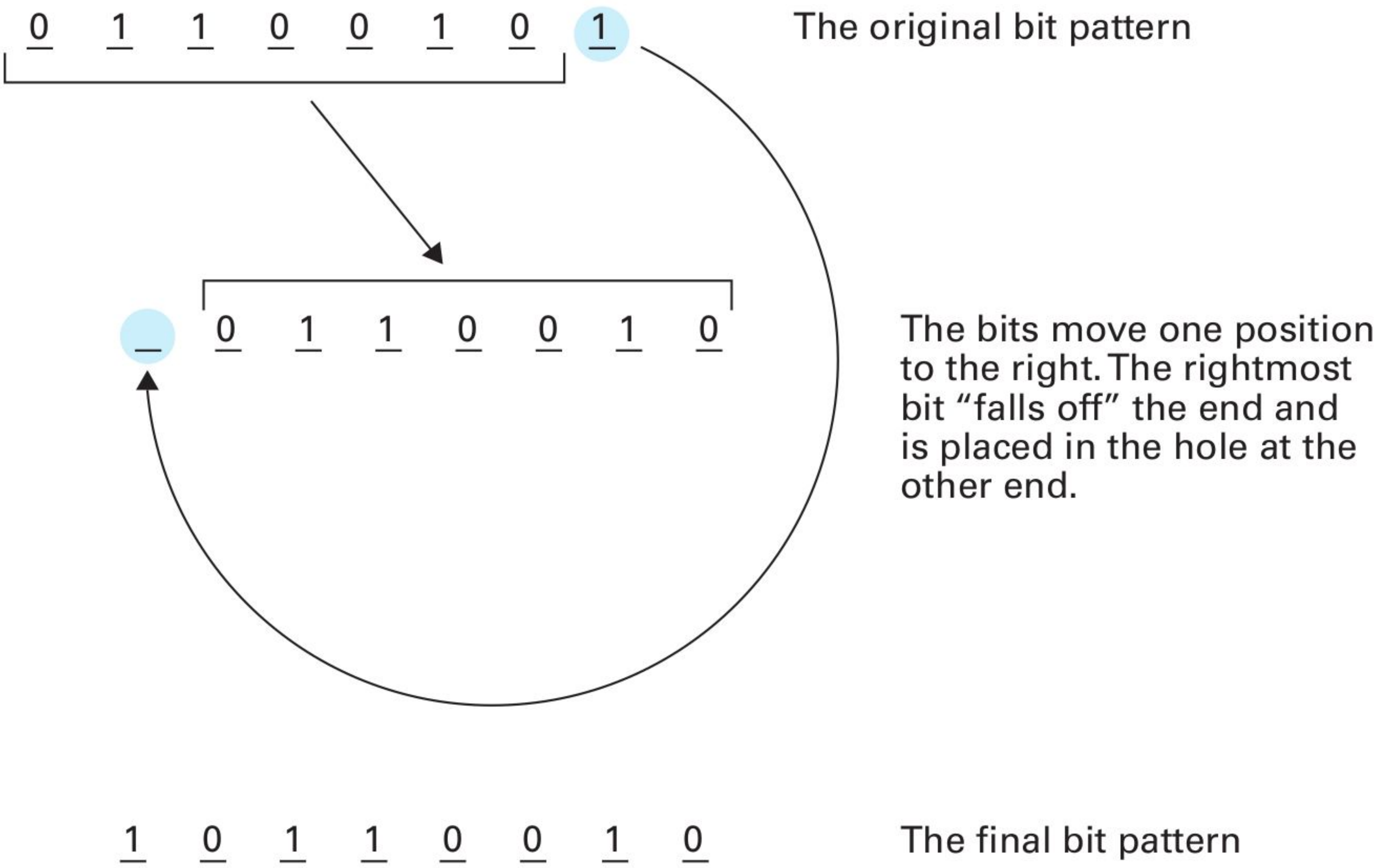
```
    10011010
OR  11001001
    11011011
```



```
     10011010
XOR  11001001
     01010011
```

Arithmetic/Logic Instructions

Rotation



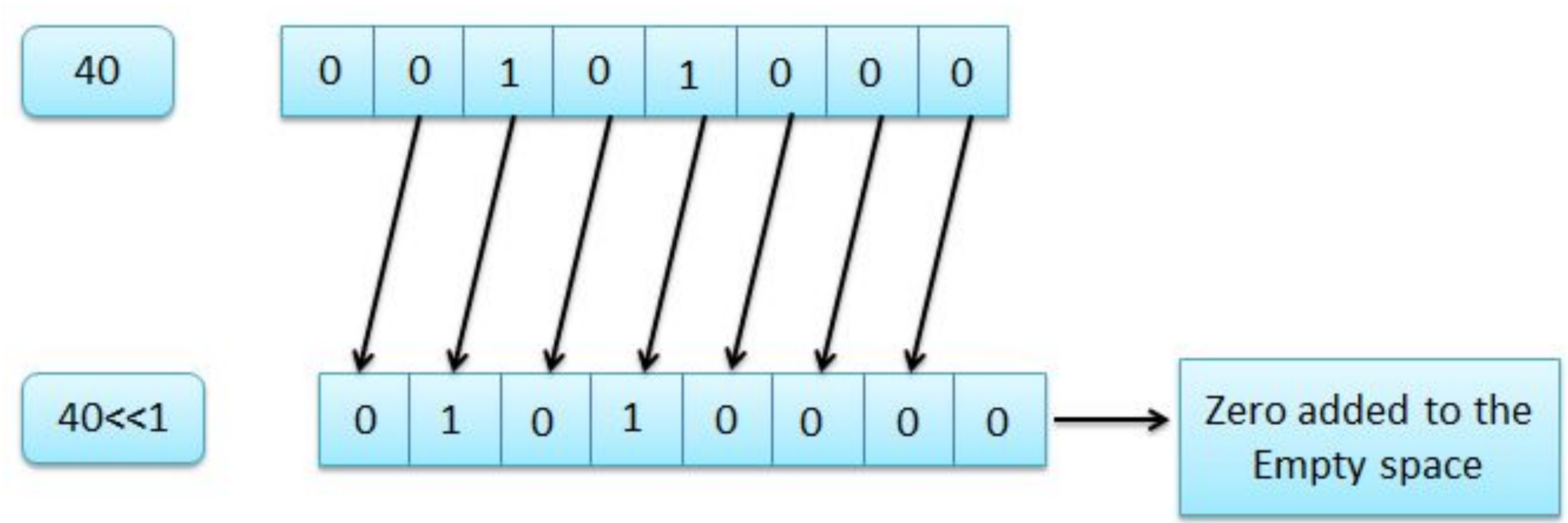
The original bit pattern

The bits move one position to the right. The rightmost bit “falls off” the end and is placed in the hole at the other end.

The final bit pattern

Arithmetic/Logic Instructions

Shift



40

0 0 1 0 1 0 0 0

40<<1

0 1 0 1 0 0 0 0

Zero added to the Empty space

Communicating with Other Devices

Communication between a computer and other devices is normally handled through an intermediary apparatus known as a **controller**.

The controller connects via cables to peripheral devices within the computer case or perhaps to a connector, called a **port**, on the back of the computer where external devices can be attached.

A controller translates messages and data back and forth between forms compatible with the internal characteristics of the computer and those of the peripheral device to which it is attached.

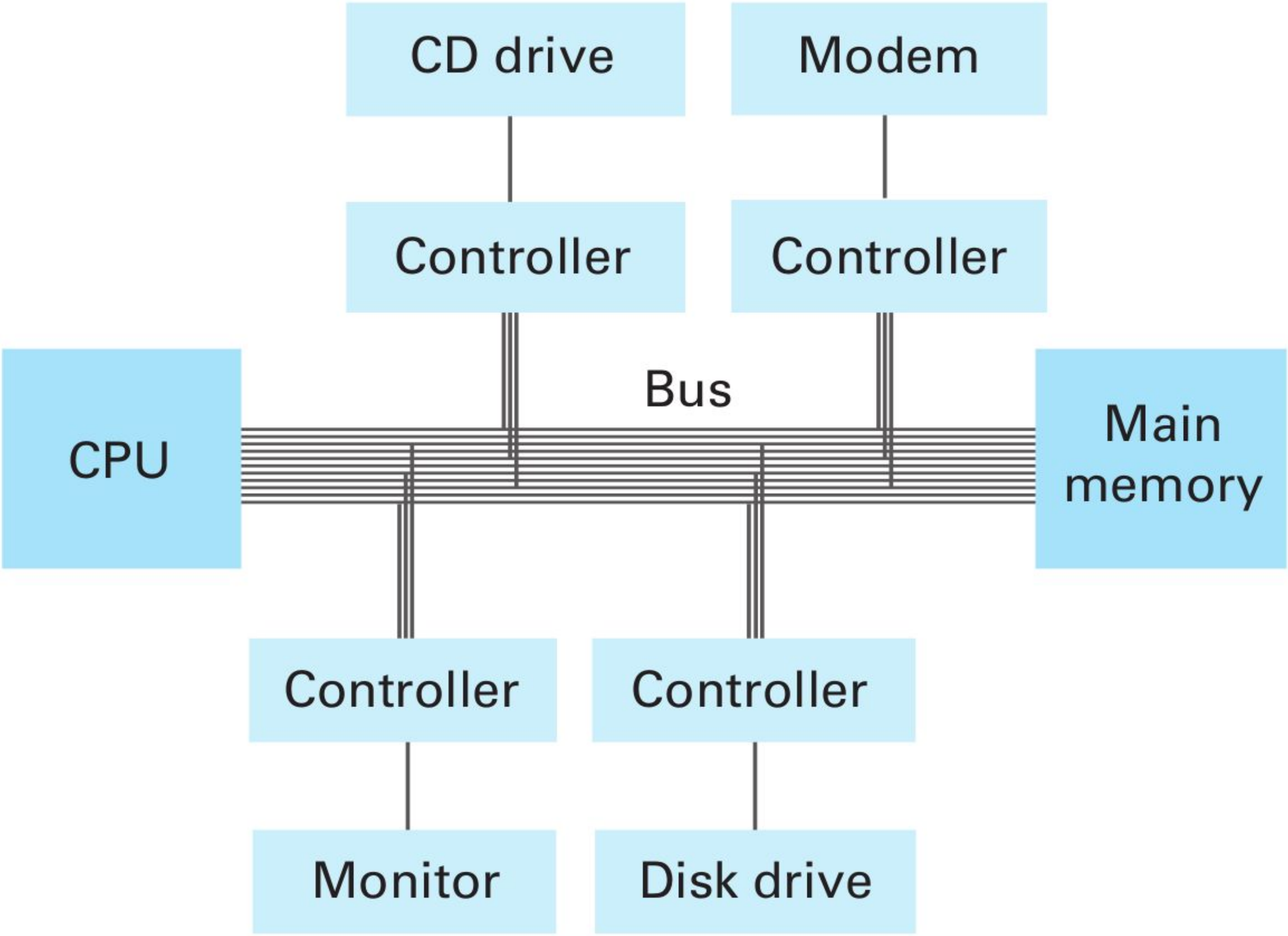
Communicating with Other Devices

Originally, each controller was designed for a particular type of device

Recently, there are standards such as the universal serial bus (USB) and FireWire,

- A single controller is able to handle a variety of devices.
    - Mice, printers, scanners, mass storage devices, digital cameras, and smartphones

Communicating with Other Devices



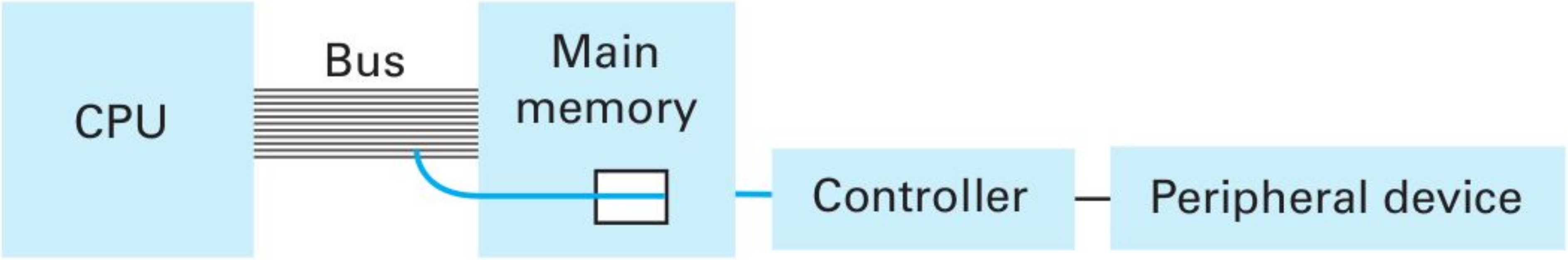
Communicating with Other Devices

To send a bit pattern to a controller,

- the bit pattern is first constructed in one of the CPU’s general-purpose registers.
- Then an instruction similar to a STORE instruction is executed by the CPU to “store” the bit pattern in the controller.

Likewise, to receive a bit pattern from a controller,

- an instruction similar to a LOAD instruction is used.



Communication Rates

The rate at which bits are transferred from one computing component to another is measured in **bits per second** (**bps**).

Common units include Kbps (kilo-bps, equal to one thousand bps), Mbps (mega-bps, equal to one million bps), and Gbps (giga-bps, equal to one billion bps).

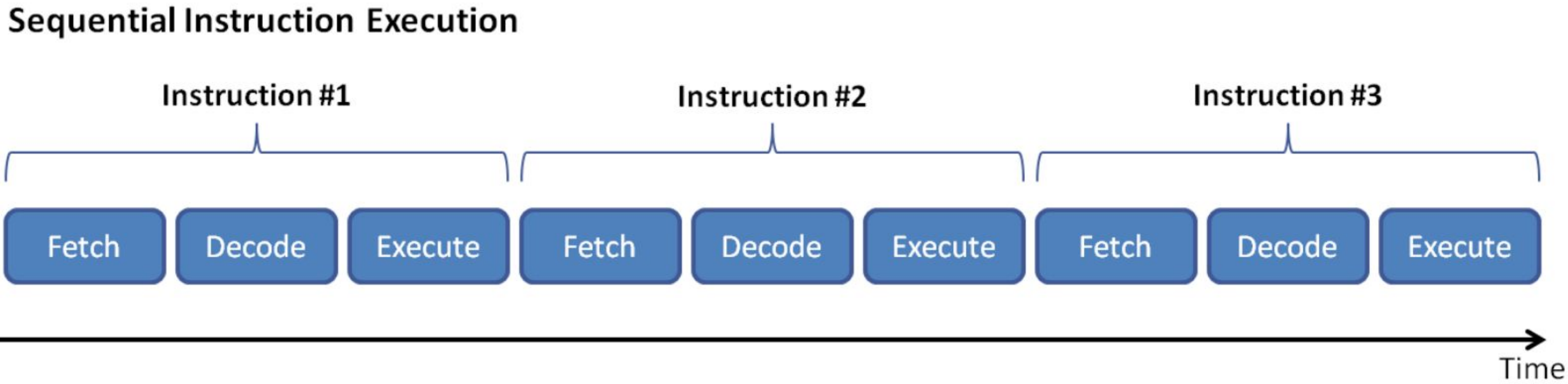
- E.g. For short distance communication, USB 2.0 and FireWire provide transfer rates of several hundred Mbps, which is sufficient for most multimedia applications.

Pipelining

Increasing execution speed is not the only way to improve a computer's performance.

The real goal is to improve the machine's throughput,

- which refers to the total amount of work the machine can accomplish in a given amount of time

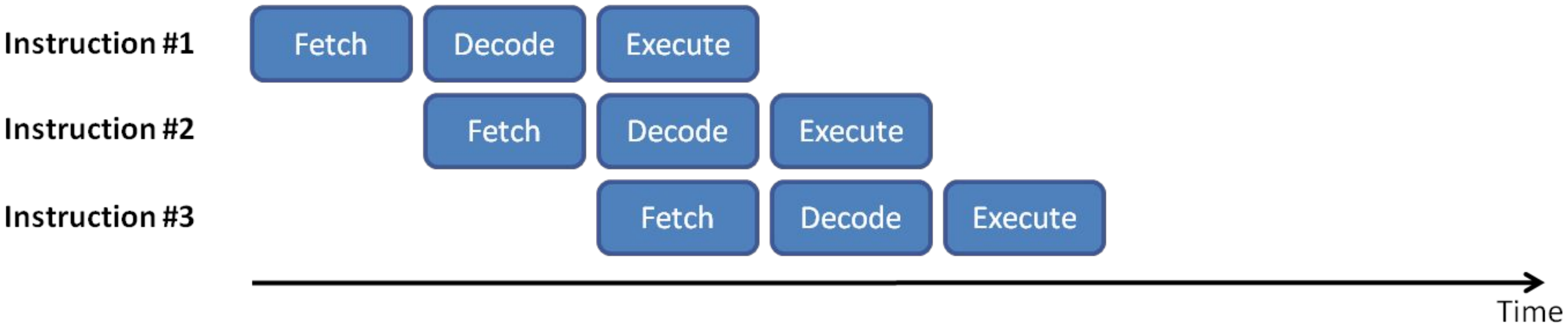


Pipelining

Pipelining: the technique of allowing the steps in the machine cycle to overlap.

More than one instruction can be in “the pipe” at any one time, each at a different stage of being processed.

Pipelined Instruction Execution



Instruction #1: Fetch, Decode, Execute

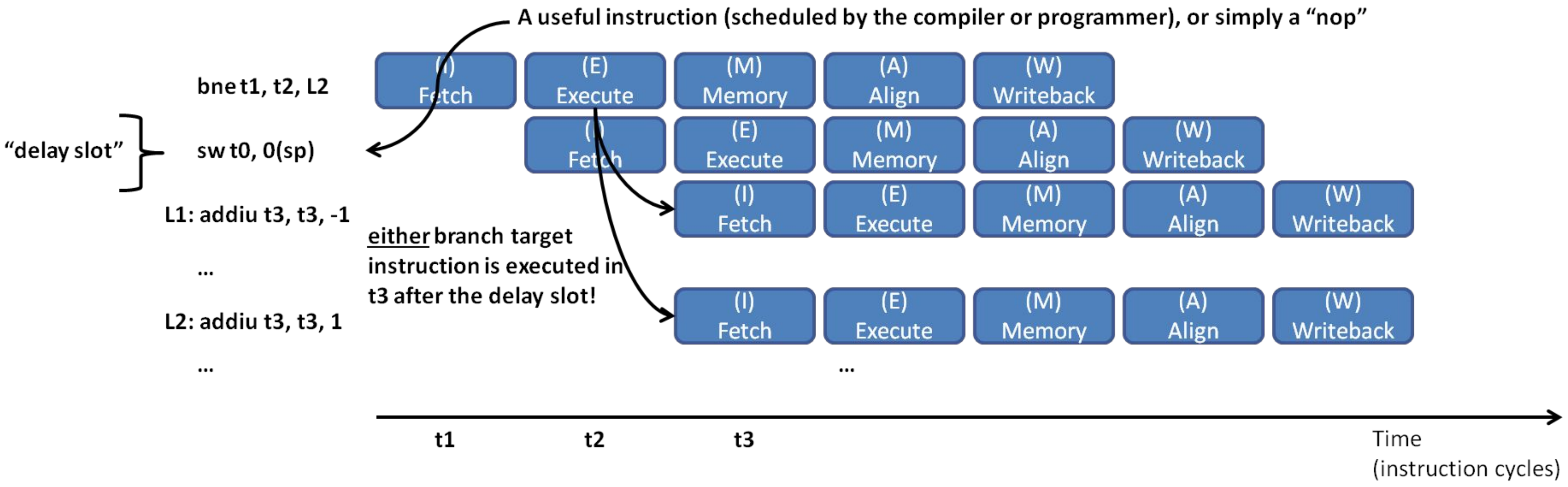
Instruction #2: Fetch, Decode, Execute

Instruction #3: Fetch, Decode, Execute

Pipelining

😓 Branches such as JUMP instructions

One solution: delayed branch



A useful instruction (scheduled by the compiler or programmer), or simply a "nop"

bne t1, t2, L2

"delay slot" sw t0, 0(sp)

L1: addiu t3, t3, -1

...

L2: addiu t3, t3, 1

...

either branch target instruction is executed in t3 after the delay slot!

(I) Fetch | (E) Execute | (M) Memory | (A) Align | (W) Writeback

(I) Fetch | (E) Execute | (M) Memory | (A) Align | (W) Writeback

(I) Fetch | (E) Execute | (M) Memory | (A) Align | (W) Writeback

...

(I) Fetch | (E) Execute | (M) Memory | (A) Align | (W) Writeback

t1 t2 t3

Time (instruction cycles)

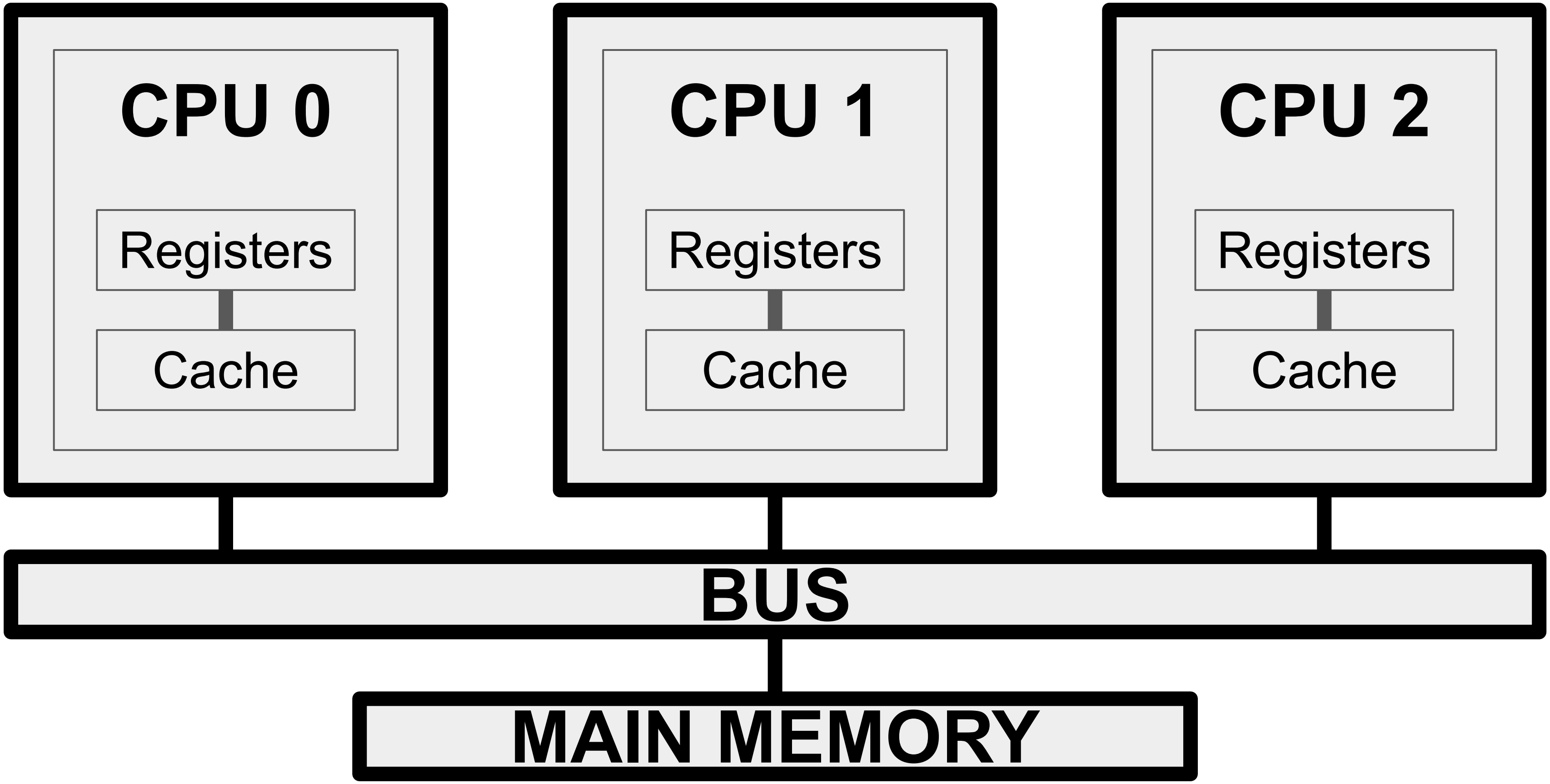
Pipelining ↠ Multiprocessor Machines

Modern pipelined machine designs are often capable of fetching several instructions at the same time and actually executing more than one instruction at a time when those instructions do not rely on each other.

Pipelining can be viewed as a **first step toward parallel processing**, which is the performance of several activities at the same time.

Multiprocessor Machines

True parallel processing requires **more than one processing unit**, resulting in computers known as **multiprocessor** or multi-core machines.



COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr