I. INTRODUCTION

Microcomputer Systems:

Basic Computer Organization

Outline

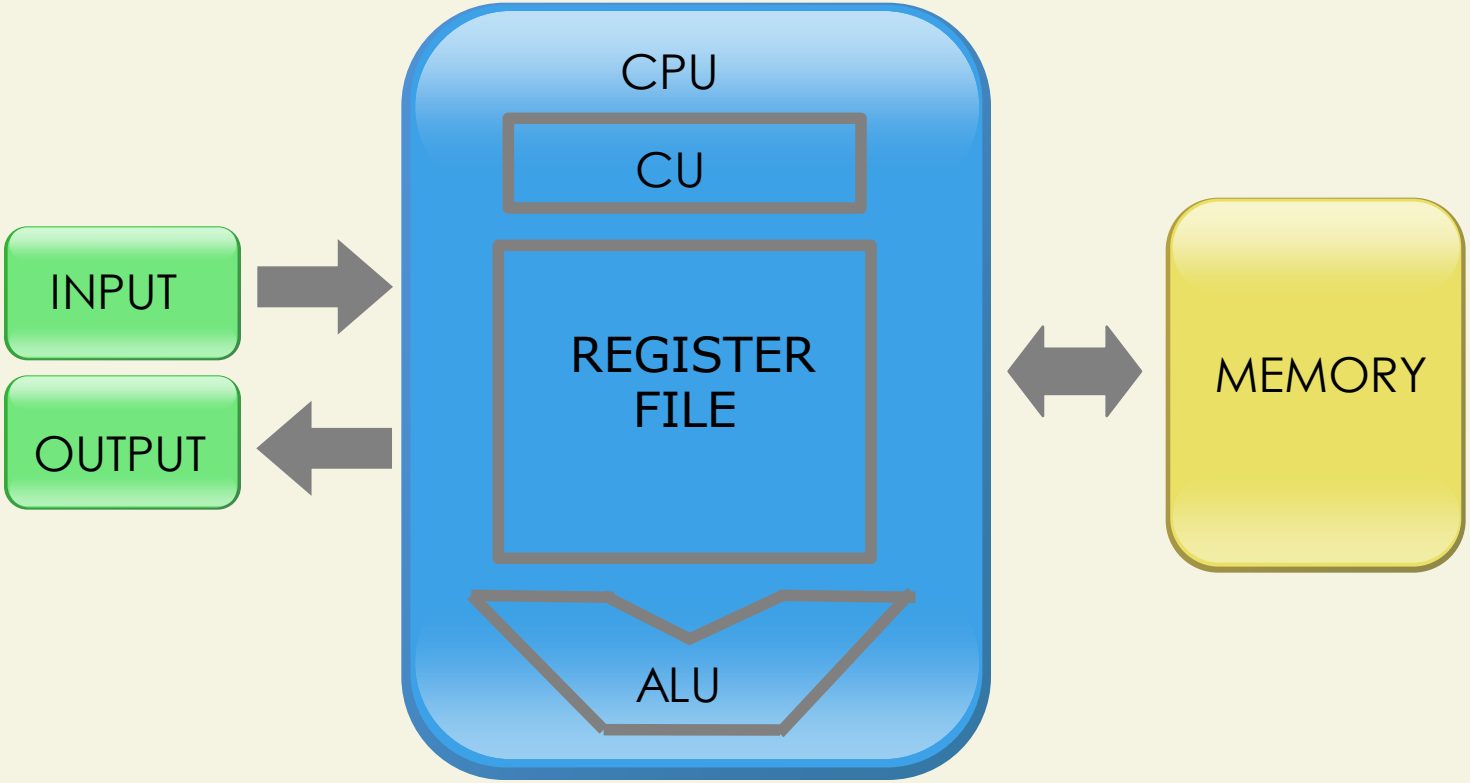
1. Basic Organization of a Microcomputer
2. Von-Neumann's Simple Computer
3. The Fetch-Decode-Execute Cycle

Objectives

At the end of the discussion, we should be able to

- describe the basic organization of microprocessor-based systems and the Von Neumann system, and
- discuss how a program instruction is executed.

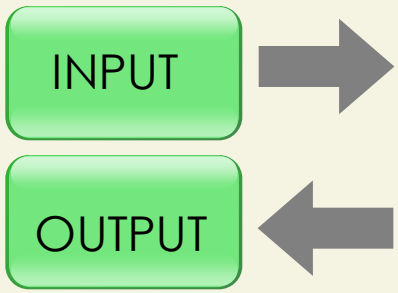
The Basic Organization of a Microcomputer



The Basic Organization of a Microcomputer

Input and Output

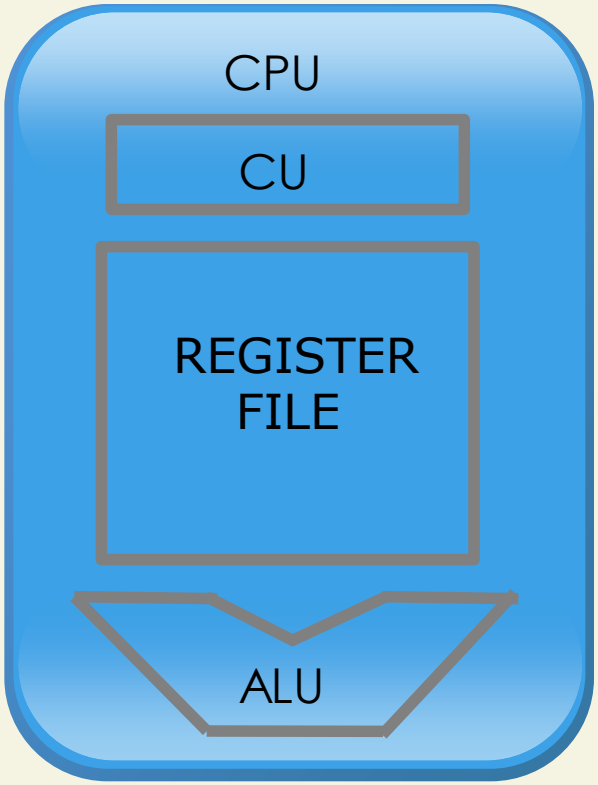
- the I/O devices connected to the bus
- bus –the collection of the computer's electrical lines where signals pass through
- the bus is generally divided into four types: the data, address, control, and power bus



The Basic Organization of a Microcomputer

CPU

- the Central Processing Unit; that is, the computer's processor
- composed of the CU, ALU, and Register File
- reads one instruction from memory at a time and executes it



The Basic Organization of a Microcomputer

CU

- the Control Unit
- the part of the CPU that sends control signals to the different parts of the system through the control bus

The Basic Organization of a Microcomputer

ALU

- the Arithmetic and Logic Unit
- a logic circuit in the CPU that is responsible for performing mathematical and logical operations.

The Basic Organization of a Microcomputer

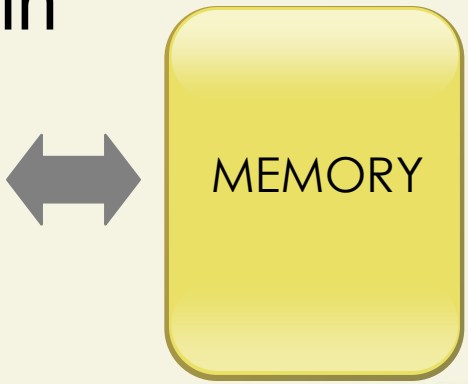
Register File

- the collection of registers inside the CPU
- a register – a set of flip-flops treated as a single unit
- flip-flop – a digital logic circuit capable of storing a single bit
- there are several registers in a computer system, some are for general purposes while others are called special-purpose registers

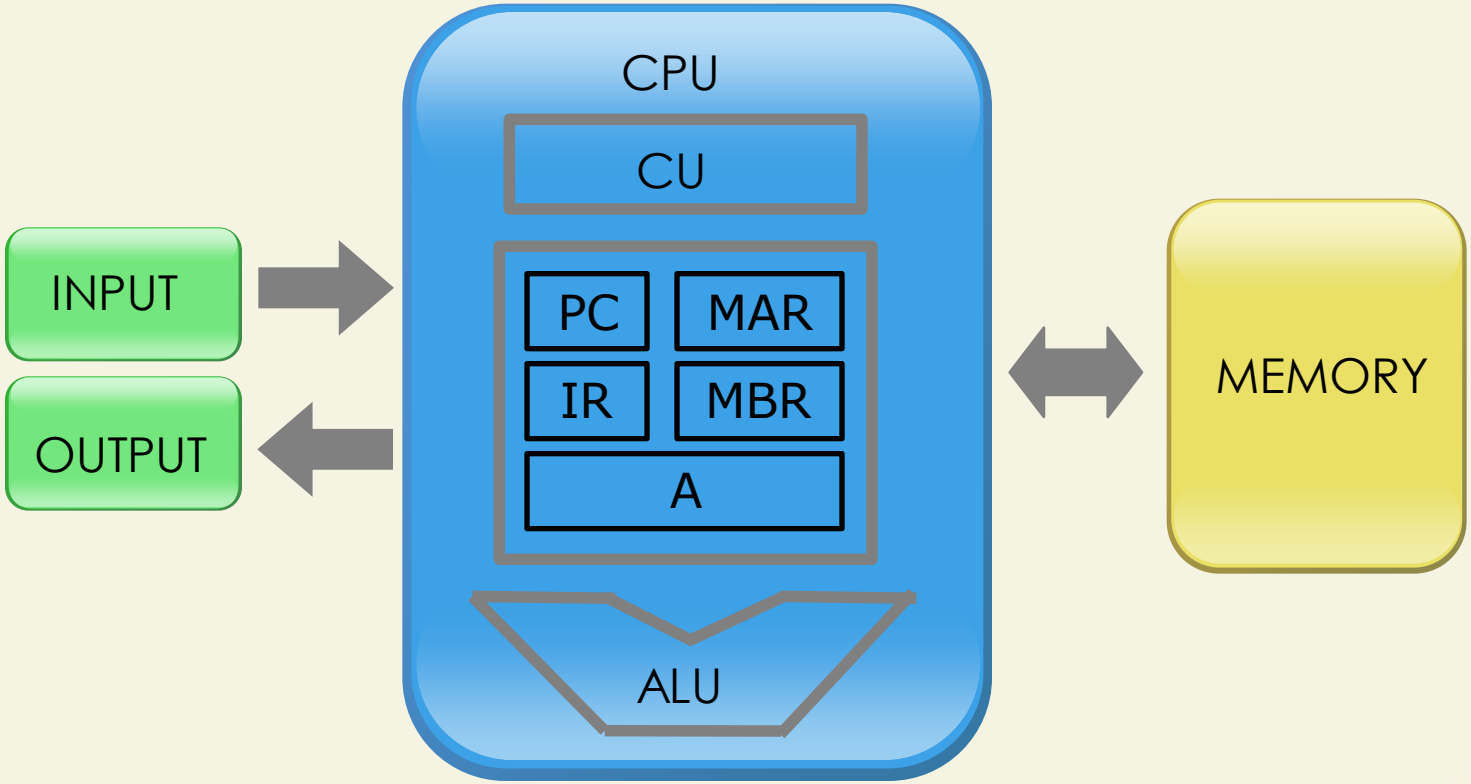
The Basic Organization of a Microcomputer

Memory

- the program-addressable storage from which instructions and other data may be loaded for subsequent execution or processing
- typically the memory is organized in chunks of 8 bits (called a byte)
- each chunk (byte) has an address



Von Neumann's Simple Computer



Von Neumann's Simple Computer

PC

- Program Counter – contains the address of the next instruction to be executed.

IR

- Instruction Register – contains the current instruction word.

Von Neumann's Simple Computer

MAR

- Memory Address Register –contains the memory address of data needed for an instruction's execution.

MBR

- Memory Buffer Register –contains data needed for an instruction's execution.

Von Neumann's Simple Computer

A

- Accumulator –the register used as temporary storage for data or for the result of arithmetic or logical operations.

The Fetch-Decode-Execute Cycle

1. Get the instruction from the memory using the address contained in PC.
2. Put the instruction into IR.
3. Increment the value in PC.
4. Decode the value in IR.
5. Execute the operation specified in the instruction.
6. Repeat step number 1.

Instructions

- Each instruction is stored in memory as a bunch of bits.
- The CPU decodes the bits to determine what should happen.
- For example, the instruction to add 2 numbers might look like this:

  10100110101001101010011010100110

- Instructions are from a language called machine language.

Machine Code

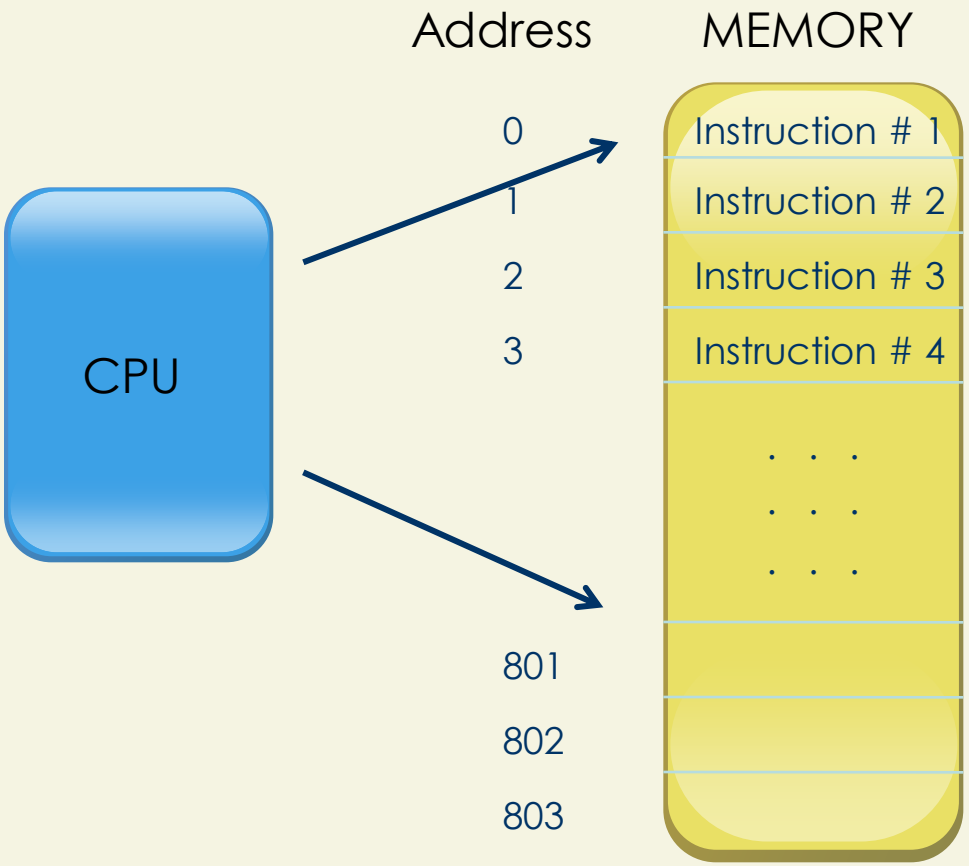
- An executable program is a sequence of these simple instructions.
- The sequence is stored in memory.
- The CPU processes the simple instructions sequentially.
- Some instructions can tell the CPU to jump to a new place in memory to get the next instruction.

Sample Program

Instruction

1. set memory[801] to hold 00000001
2. set memory[802] to hold 00000000
3. if memory[802] = 10 jump to instruction #8
4. increment memory[802]
5. set memory[803] to 2 times memory[801]
6. put memory[803] in to memory[801]
7. jump to instruction #3
8. print memory[801]

Illustration

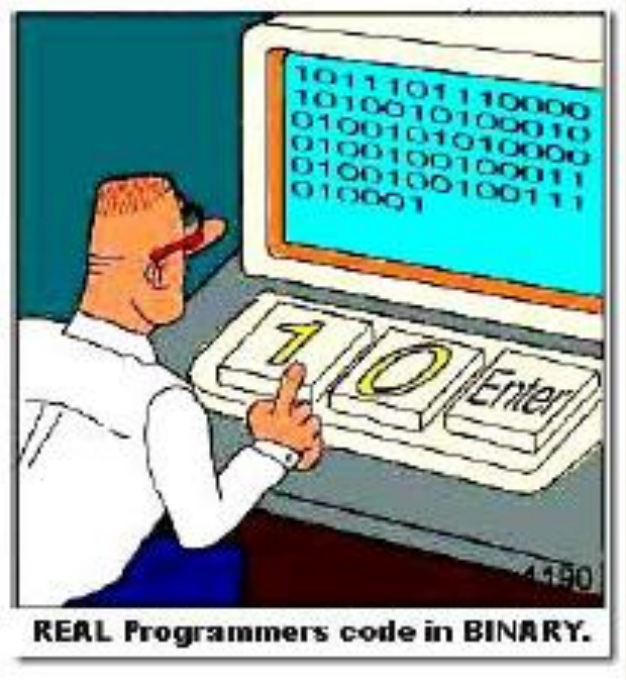


Human vs. Machine Programs

- The computer can only understand the bits (the encoded program)

  = Machine Language
- Humans don't like to deal with bits, so they developed English-like abbreviations for programs.

  = Assembly Language



I. INTRODUCTION

The Rationale of Using Low-level Language

Objectives

At the end of this section, we should be able to:

- Identify different levels of programming languages, and
- Discuss the rationale of using low-level languages.

Hierarchy of Programming Languages

Machine Language

- This is what the computer actually sees and deals with. Every command the computer sees is given as a number or sequence of numbers.

Hierarchy of Programming Languages

Assembly Language

- the same as machine language, except the command numbers have been replaced by letter sequences which are easier to memorize.
- middle-level language
- maps human-readable mnemonics to machine instructions
- allows machine-level programming without writing in machine language

Hierarchy of Programming Languages

Assembly Language

- For example, an x86 processor can execute the following binary instruction as expressed in machine language:

  Binary: 10110000 01100001

  Hexadecimal: B0 61

- The equivalent assembly language representation is easier to remember:

  MOV AL, #61h

Hierarchy of Programming Languages

High-Level Language

- High-level languages are there to make programming easier.
- Assembly language requires you to work with the machine itself. High-level languages allow you to describe the program in a more natural language.
- A single command in a high-level language usually is equivalent to several commands in an assembly language.

Reasons for not using Assembly

- Development time: it takes much longer to develop in assembly
- Maintainability: unstructured
- Portability: platform-dependent

Reasons for using Assembly

- To understand how CPUs and compilers work
- Developing compilers, debuggers and other development tools
- Hardware drivers, system code and low-level tasks such as bootloaders
- Embedded systems
- Reverse Engineering
- Address critical performance issues (Optimizing for speed or space)

The Rationale of Using Low-level Language

- By gaining a deeper understanding of how computers work at a lower level, one can often be more productive developing software in higher level language such as C.
- Learning to program in assembly language is an excellent way to achieve this goal.

An Application

- NBA Jam