Computers and Programming

Chapter 01

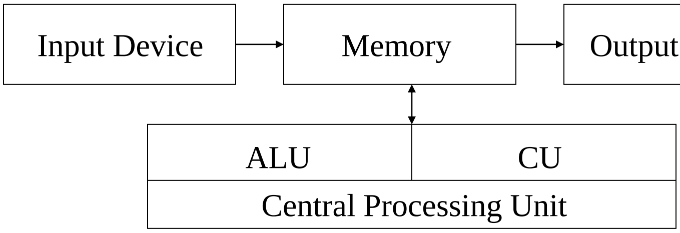
CMPE-112 *Programming Fundamentals*

Lecture Plan

- Hardware
- Software
- Programming Process
  - Problem Definition
  - Program Design
  - Program Coding
  - Compilation & Execution
  - Testing & Debugging
  - Program Documentation
- C language overview

Hardware: main components

- Five principal components in a computer are:
  - Arithmetic-logic unit (ALU)
  - Control unit (CU)
  - Memory
  - Input device (keyboard, mouse, floppy disk, etc)
  - Output device
- Central Processing Unit (CPU) = ALU + CU


Input Device

Memory

Output Device

ALU

CU

Central Processing Unit

Central Processing Unit

- Arithmetic-Logic Unit (ALU)
  - Performs arithmetic operations
  - Conducts comparisons of data
  - Main components: adders, multipliers, counters and comparators
- Control Unit (CU)
  - Fetches an instruction from the memory
  - Interprets the instruction
  - Loads the data into ALU
  - Executes the instruction
  - Stores the result back into memory
  - Directs and coordinates all other computer units

Memory

- Stores
  - Instructions
  - Intermediate data and final results of instructions
- Contains of cells (storage locations)
- Cell has a label from 0 upwards – its address in the memory
- Each location is called a **word** and consists of **bits**
- Bit – the abbreviation for a **binary digit** – can contain either a 0 or 1
- Eight adjacent bits form a **byte**
- A *word* usually consists of 8, 16 or 32 bits, i.e. 1, 2 or 4 bytes
- One cell can hold only *one piece* of information

Input and Output Devices

- Provide communication between users and computers
- Interchange information between computers
- Input devices store data **into** computer memory
  - Keyboard
  - Mouse
  - Light pen
- Output devices retrieve results **from** computer memory
  - Video Monitor
  - Printer
- Input/Output Devices
  - Hard and floppy disks
  - Tapes
  - Modems

Software

- System software direct the internal operation of a computer
  - Control input and output devices
  - Manage storage areas within the computer
- Application software solve user-oriented problems
  - Produce a student time-table
  - Calculate salary
  - Prepare a letter
  - Manage data bases

Programming languages

- Machine language
- Assembly language
- High-level languages (BASIC, FORTRAN, Pascal, C)

Programming Process (I)

- Problem definition
  - What must the program do?
  - What outputs are required and in what form?
  - What inputs are available and in what form?

Example: *Find a maximum of two numbers*

- Input two numbers, compare them and print the maximum value
- Inputs and outputs are decimal numbers
- Inputs are entered from the keyboard
- Result is shown on the monitor

Programming Process (II)

- *Program Design* involves creating an **algorithm** – sequence of steps, by which a computer can produce the required outputs from the available inputs

Top-down design

- The main problem is split into subtasks
- Then each subtask is divided into simpler subtasks, etc. unless it is clear how to solve all such subtasks

Programming Process (III)

- *Program Coding* means expressing the algorithm developed for solving a problem, in a programming language
- Example of source code is on the right

```
#include <stdio.h>

int main()
{
    int number1, number2;
    int maximum;

    printf("Please, enter two numbers: ");
    scanf("%d %d", &number1, &number2);

    if (number1 >= number2)
           maximum = number1;
    else
           maximum = number2;

    printf("\nMaximum value is %1d\n\n“,
    maximum);

    return 0;
}
```

Programming Process (IV)

- *Program Compilation* – translation of a program written in a high-level programming language into machine language instructions
- Compilation step converts a source program into an intermediate form, called *object code*
- Linking step is necessary to combine this object code with other code to produce an *executable program*
- The advantage of this two-step approach:
  - Source of the large program may be split into more than one file
  - These files are worked on and compiled separately
  - Object code may be stored in libraries and used for many programs
  - Then they will be combined into one executable code

Programming Process (V)

- Program Testing & Debugging
  - Initially, almost all programs may contain a few errors, or *bugs*
  - Testing is necessary to find out if the program produces a correct result. Usually it is performed with sample data
  - Debugging is the process of locating and removing errors
- Common types of errors
  - Compile-time errors arise from misuse of syntax rules (e.g. a ketword is misspelled). They are detected by compilers
  - Run-time, or execution-time errors are revealed when the program is executed (for instance, division by zero)
  - Logical errors are NOT detected automatically. They arise in the design of the algorithm. *Tracing* and/or *dumping* is necessary to detect and remove them

Programming Process (VI)

- *Program Documentation* involves describing the program in details so that it can be used and/or modified much later after it is created
- Some tips for documenting a program
  - Use a meaningful name for variables and constants

    S = D / T;

    Speed = Distance / Time;
  - Comment all pieces of code. Comments may include

    Programmer name

    Name of source file

    Dates of creating and modifying the source; its version

    Description of every input and output variable, etc

C language overview

- **C** is a general-purpose programming languages that was originally designed by Dennis Rithcie of ***Bell Laboratories*** and implemented there on a PDP-11 in 1972. It was first used as ***the system languages*** for the UNIX operating system
- **Ken Tomson**, the developer of UNIX, had been using both an assembler and a language named **B** to produce initial version of UNIX in 1970. **C** was invented to overcome the limitations of **B**
- By the early 1980s, the original **C** language had evolved into what is now known as traditional **C**. In late 1980s , **the American National Standards Institute(ANSI)** Committee created draft standards for what is known as **ANSI C** or **standard C**
- Today, **ANSI C** is mature, general-purpose programming language that is widely used available on many machines and in many operating systems

Why C ?

- **C is a small language**
  - It has fewer reserved words (keywords), powerful data types and control structures
- **C is native language of Unix**
  - Unix is major interactive OS on workstations, servers, mainframes and PC. Much of MS-DOS and OS/2, Windowing packages, database programs, graphics libraries are written in C
- **C is portable**
  - Code written on one machine can be easily moved to another
- **C is terse**
  - C has powerful set of operators; some of these operators allow the programmer to access the machine at the bit level
- **C is modular**
  - The heart of effective problem solving is problem decomposition. Taking a problem and breaking it into small, manageable pieces of code known as functions or modules, is a way to make the programming process easy
- **C is basis for C++ and Java**