CSI 333 – Lecture 12
An Abstract View of a Computer
(A Short Lecture)
Notion of Abstraction

- Simplified view.
- Hides extraneous details.

**Examples:**

1. Depositing a check.
   - Customer's view: Adds money to an account.
   - Bank’s view: Many steps involved.

2. A function in C.
   - Abstraction for a part of the computation (e.g. sorting).
   - User need not worry about implementation details.

3. A programming language: An abstract way of representing data structures and algorithms.
A statement in C:

\[ p = 2 \times (q - r); \]

- Statement has three operations: \(*\), \(-\) and \(=\).

- Generally, a computer can only execute simple (unary or binary) operations in one step.

Translation of the above statement:

\[ t = q - r; \]
\[ p = 2 \times t; \]

More common form (closer to machine language):

```
sub t, q, r
mul p, 2, t
```
Machine Language: Set of instructions that a computer can execute directly.

Assembly Language: Allows mnemonic operations and symbolic addresses.

Execution of Programs by a Computer:

Machine Language Program:
- Sequence of machine language instructions.
- Data locations.
Interaction Between Memory and CPU

**Memory:**
- Stores both instructions and data.
- Controlled by CPU.

**Functions of the Central Processing Unit (CPU):**
- Responsible for executing instructions and modifying memory locations.
- Contains registers and hardware for performing arithmetic operations. (Also has a control unit.)
- Retrieves instructions (“Instruction fetch”) and data (“Operand fetch”) from memory.
- Interacts with memory by specifying an address and issuing a “Read” or “Write” command.
Interaction Between Memory and CPU (continued)

Functions of the CPU (continued):

- **Read**: Allows CPU to retrieve a value stored in memory.
- **Write**: Allows CPU to modify the value stored in a memory location.
- **Special register: Program Counter (PC)**
  - Contains the address of the next instruction to be executed.
  - Initially, contains the starting address of the program (i.e., address of the first instruction to be executed).

Program Execution:

```plaintext
do {
  (1) Fetch instruction.
  (2) Decode instruction.
  (3) Execute instruction.
} while (instruction is not "halt")
```
Fetch-Decode-Execute Loop

**Fetching an Instruction:**

1. Use PC to specify the address and obtain the instruction to be executed.
2. Increment PC.

**Decoding an Instruction:**

1. Determine the opcode.
2. Interpret the remaining parts of the instruction.
Executing an Instruction:

1. Fetch operands, if necessary. (May involve reading from memory.)
2. Carry out the operation. (May involve writing to memory.)

Types of Instructions:

- Arithmetic instruction.
- Control instruction.

Control Instruction:

- Involves a comparison (or test).
- Result of comparison determines the instruction to be executed next.
Example:

**Code Segment in C:**

```c
if (a != b)
    p = q + r;
d = 2 * q;
```

**Assembly Language Equivalent:**

```
beq a, b, next
add p, q, r
next: mul d, 2, q
```
Why Study Assembly Language?

- Some features of a machine can be accessed only through assembly language. (These features are used by system-specific software – e.g. operating system.)

- Programs that must fit within a small amount of memory are usually written in assembly language.

- Programs that must execute very quickly are usually written in assembly language.

- Knowledge of assembly language is essential in developing compilers.