Module III (10 Hours)

Computer abstractions and technology - Introduction, Computer architecture -8 Design features, Application program - layers of abstraction, Five key components of a computer, Technologies for building processors and memory, Performance, Instruction set principles – Introduction, Classifying instruction set architectures, Memory addressing, Encoding an instruction set.

- J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.
- J. Hennessy and D. Patterson, “Computer Architecture, A quantitative approach”, 5th Edition.
- Hamacher, Vranesic & Zaky, “Computer Organization” (5th Ed), McGraw Hill.

Introduction

- Computers have led to a third revolution for civilization, with the information revolution taking its place alongside the agricultural and the industrial revolutions.

Introduction

- The computer revolution continues.
- Applications that were economically infeasible suddenly become practical.
- In the recent past, the following applications were "Computer science fiction"
  - Computers in automobile
  - Cell phones
  - World Wide Web
  - Search engines

Introduction

- Classes of Computing Applications and Their Characteristics
  - Personal computers (PCs)
    - Emphasize delivery of good performance to single users at low cost
  - Servers
    - Servers are oriented to carrying large workloads, which may consist of either single complex applications—usually a scientific or engineering application—or handling many small jobs, such as would occur in building a large web server.
    - Provide for greater computing, storage, and input/output capacity.
  - Embedded
    - These computers are the largest class of computers and span the widest range of applications and performance.
    - Embedded computers include the microprocessors found in our car, the computers in a television set, and the networks of processors that control a modern airplane or cargo ship

PostPC Era

Personal Mobile Device (PMD):

- PMDs are battery operated with wireless connectivity to the Internet and typically cost hundreds of dollars, and, like PCs, users can download software ("apps") to run on them.
- Unlike PCs, they no longer have a keyboard and mouse, and are more likely to rely on a touch-sensitive screen or even speech input.
- Today's PMD is a smart phone or a tablet computer, but tomorrow it may include electronic glasses.

PostPC Era

Cloud Computing (Replacing Traditional Servers)

- Cloud Computing, which relies upon giant datacenters that are known as Warehouse Scale Computers (WSCs).
- Companies like Amazon and Google build these WSCs containing 100,000 servers and then let companies rent portions of them so that they can provide software services to PMDs without having to build WSCs of their own.
- Software as a Service (SaaS) deployed via the cloud is revolutionizing the software industry just as PMDs and WSCs are revolutionizing the hardware industry.
- Today's software developers will often have a portion of their application that runs on the PMD and a portion that runs in the Cloud.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design.

1. Design for Moore's Law
   - The one constant for computer designers is rapid change, which is driven largely by Moore's Law.
   - It states that integrated circuit resources double every 18–24 months.
   - Moore's Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel.
   - Use an "up and to the right" Moore's Law graph to represent designing for rapid change.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design

2. Use Abstraction to Simplify Design
   - A major productivity technique for hardware and software is to use abstractions to represent the design at different levels of representation;
   - Lower-level details are hidden to offer a simpler model at higher levels.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design

3. Make the Common Case Fast
   - Making the common case fast will tend to enhance performance better than optimizing the rare case.
   - Ironically, the common case is often simpler than the rare case and hence is often easier to enhance.
   - This common sense advice implies that you know what the common case is, which is only possible with careful experimentation and measurement.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design

4. Performance via Parallelism
   - Computer architects have offered designs that get more performance by performing operations in parallel.
   - We use multiple jet engines of a plane as the icon for parallel performance.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design

5. Performance via Pipelining

- A particular pattern of parallelism is so prevalent in computer architecture that it merits its own name: pipelining.
- For example, before fire engines, a "bucket brigade" would respond to a fire. The townsfolk form a human chain to carry a water source to fire, as they could much more quickly move buckets up the chain instead of individuals running back and forth.
- Our pipeline icon is a sequence of pipes, with each section representing one stage of the pipeline.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design

6. Performance via Prediction

- It can be better to ask for forgiveness than to ask for permission, the final great idea is prediction.
- In some cases, it can be faster on average to guess and start working rather than wait until you know for sure, assuming that the mechanism to recover from a misprediction is not too expensive and your prediction is relatively accurate.
- We use the fortune-teller's crystal ball as the prediction icon.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design

7. Hierarchy of Memories

- Programmers want memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory is often the majority of computer cost.
- Architects can address these conflicting demands with a hierarchy of memories, with the fastest, smallest, and most expensive memory(cache) per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom.
- The triangle shape indicates speed, cost, and size: the closer to the top, the faster and more expensive per bit the memory; the wider the base of the layer, the bigger the memory.

Eight Great Ideas in Computer Architecture

Eight great ideas that computer architects have been invented in the last 60 years of computer design.

8. Dependability via Redundancy

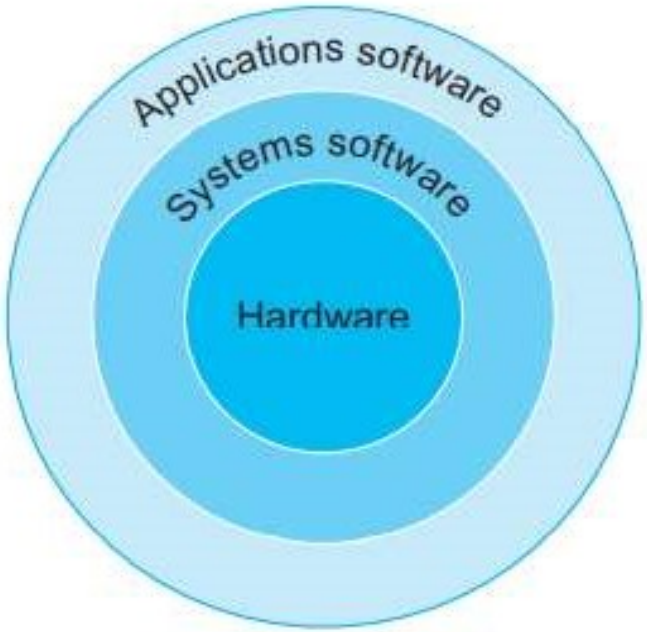
- Computers not only need to be fast; they need to be dependable.
- Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures.
- We use the tractor-trailer as the icon, since the dual tires on each side of its rear axels allow the truck to continue driving even when one tire fails.

Application Program

- A typical application, such as a word processor or a large database system, may consist of millions of lines of code and rely on sophisticated software libraries that implement complex functions in support of the application.
- The hardware in a computer can only execute extremely simple low-level instructions.
- To go from a complex application to the simple instructions involves several layers of software that interpret or translate high-level operations into simple computer instructions, an example of the great idea of abstraction.

Application Program

- Layers of software are organized primarily in a hierarchical fashion, with applications being the outermost ring and a variety of systems software sitting between the hardware and applications software.



Systems software: Software that provides services that are commonly useful, including operating systems, compilers, loaders and assemblers.

FIGURE 1 A simplified view of hardware and software as hierarchical layers, shown as concentric circles with hardware in the center and applications software outermost. In complex applications, there are often multiple layers of application software as well. For example, a database system may run on top of the systems software hosting an application, which in turn runs on top of the database.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

Application Program

- There are many types of systems software, but two types of systems software are central to every computer system today: an operating system and a compiler.

Operating System

- An operating system interfaces between a user's program and the hardware and provides a variety of services and supervisory functions.
- Among the most important functions are:
  - Handling basic input and output operations
  - Allocating storage and memory
  - Providing for protected sharing of the computer among multiple applications using it simultaneously.
- Examples of operating systems in use today are Linux, iOS, and Windows.

Application Program

Compiler

- Compilers perform a vital function: the translation of a program written in a high-level language, such as C, C++, Java, or Visual Basic into instructions that the hardware can execute.
- Given the sophistication of modern programming languages and the simplicity of the instructions executed by the hardware, the translation from a high-level language program to hardware instructions is complex.

Operating system: Supervising program that manages the resources of a computer for the benefit of the programs that run on that computer.

Compiler: A program that translates high-level language statements into assembly language statements

From a High-Level Language to the Language of Hardware

- Computer language as numbers in base 2, or binary numbers, each "letter" as a binary digit or bit.
- Computers are slaves to our commands, which are called instructions.
- Instructions, which are just collections of bits that the computer understands and obeys, can be thought of as numbers.
- For example, the bits 1000110010100000 tell one computer to add two numbers.

> Instruction: A command that computer hardware understands and obeys.

From a High-Level Language to the Language of Hardware

- Pioneers invented programs to translate from symbolic notation to binary. The first of these programs was named an assembler.
- This program translates a symbolic version of an instruction into the binary version.
- For example, the programmer would write add A,B and the assembler would translate this notation into 1000110010100000
- The name coined for this symbolic language, is assembly language. In contrast, the binary language that the machine understands is the machine language.

Assembler: A program that translates a symbolic version of instructions into the binary version.

Assembly language: A symbolic representation of machine instructions.

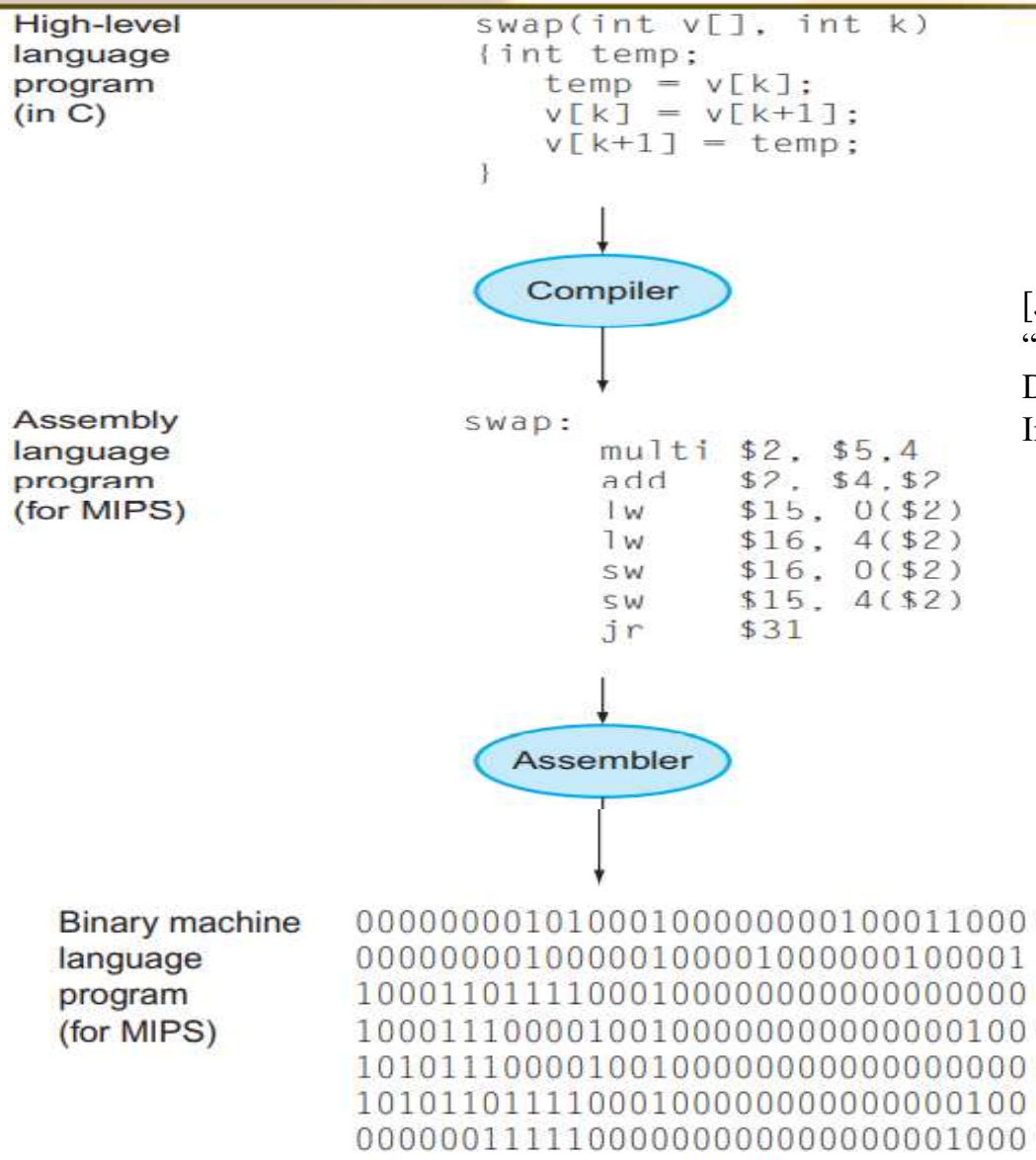
Machine language: A binary representation of machine instructions

From a High-Level Language to the Language of Hardware

- A program could be written to translate a more powerful language into computer instructions was one of the great breakthroughs in the early days of computing.
- Programmers today owe their productivity—and their sanity—to the creation of high-level programming languages and compilers that translate programs in such languages into instructions.
- Figure 2 shows the relationships among these programs and languages, which are examples of the power of abstraction.

High-level programming language: A portable language such as C, C++, Java, or Visual Basic that is composed of words and algebraic notation that can be translated by a compiler into assembly language.

From a High-Level Language to the Language of Hardware



[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

FIGURE 2 C program compiled into assembly language and then assembled into binary machine language. Although the translation from high-level language to binary machine language is shown in two steps, some compilers cut out the middleman and produce binary machine language directly.

Back

From a High-Level Language to the Language of Hardware

- A compiler enables a programmer to write the high-level language expression: A + B
- The compiler would compile it into the assembly language statement: add A,B
- The assembler would translate this statement into the binary instructions that tell the computer to add the two numbers A and B: 1000110010100000

From a High-Level Language to the Language of Hardware

- High-level programming languages offer several important benefits.
  - First, they allow the programmer to think in a more natural language, using English words and algebraic notation, resulting in programs that look much more like text than like tables of cryptic symbols
  - The second advantage of programming languages is improved programmer productivity.
  - Conciseness (short and clear) is another advantage of high level languages over assembly language.
  - The final advantage is that programming languages allow programs to be independent of the computer on which they were developed, since compilers and assemblers can translate high-level language programs to the binary instructions of any computer.