Computer Organization & Architecture

Computer Organization

- Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system. A knowledge of computer organization can help one to understand the internal operations that are carried out by a computer while a program is being executed.

- The various components of a computer are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

Computer Organization

- We can therefore say that computer organization deals with the internal view of the computer and the roles that the internal components play during execution of a program.

Computer Architecture

- The architecture (also called instruction set architecture) of a computer is concerned with the structure and behavior of the computer as viewed by user such as an assembly or machine language programmer.

- An assembly or machine language programmer needs to be aware of the specific instructions supported by the processor (called the processor's instruction set), the instruction formats, the specific registers and their roles, the techniques for accessing the data stored in the memory, the way to perform input/output operations, etc.

Computer Architecture

- Thus, computer architecture is the external view of a computer that is essential to be properly understood by anyone who is likely to program a computer using machine or assembly languages.

Modules to be covered

- Introduction and Performance Evaluation
- Data representation and basic operations
- Instruction Set Architecture
- Processor Design
- Memory hierarchy
- Input/Output Organization
- Parallel Processing

Introduction and Performance Evaluation

Introduction

- Digital computers use the binary number system, which has two digits: 0 and 1.
- A binary digit is called a bit.
- Information is represented in digital computers in groups of bits.
- By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols, such as decimal digits or letters of the alphabet.

Introduction

- By judicious use of binary arrangements and by using various coding techniques, the groups of the bits are used to develop complete sets of instructions for performing various types of computations.

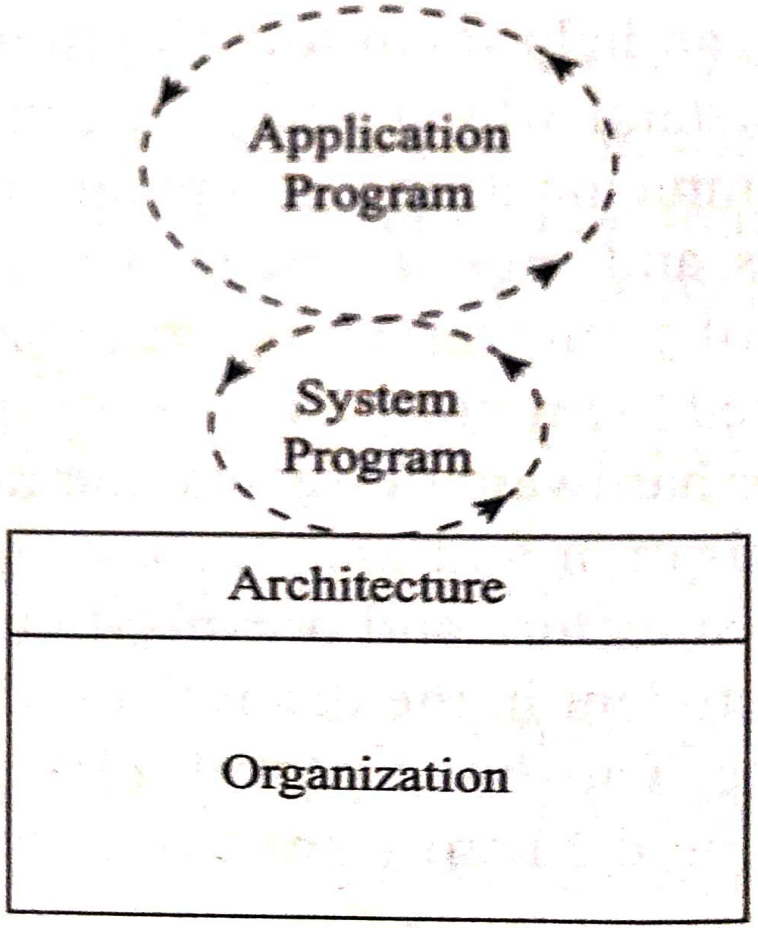
Why we need to Study Computer Organization and Architecture?

- To get an insight of how a computer executes programs internally.
- To write more effective programs.
- To understand why a program exhibits poor performance, even though it might be implementing a computationally efficient algorithm.

Why we need to Study Computer Organization and Architecture?

For system programmers and programmers of embedded computing systems, a good knowledge of computer architecture and organization is essential because they often need to program bare hardware without the support of any operating system.

Organization implements the architecture



Organization implements the architecture.

This figure shows the relation between computer organization , computer architecture, system programs , and application programs.

It also shows that system program directly interacts with the computer hardware.

The system programs are written for specific computer architectures.

The application programs on the other hand invoke the services offered by the system programs.

Computer design

- Computer organization is an implementation of the architecture of a computer .

- Starting from the specification of the architecture of a computer, and based on several constraints that may have to be satisfied, the organization is arrived through computer design.

- Computer design is concerned with the hardware design of the computer.

Computer design

- Once the architecture and other specifications of the computer are formulated it is the task of the designer to develop the hardware for the system.

- Computer design is concerned with the determination of what hardware to be used and how they are to be connected. This aspect of computer hardware is sometimes referred to as *computer implementation*.

Von Neumann Computers

- In the very early days of computing, one instruction at a time of a program was executed, with an operator setting up each instruction and also initiating the execution of each instruction .
- The operator would typically set up an instruction by flipping some switches on switch board, and then would set up the required data and control paths to the functional units by using patch cords.
- Certainly, executing a program was extremely cumbersome and required a lot of effort by the operators.
- At this juncture, in a revolutionary step, the concept of stored program computers was proposed by John Von Neumann in 1945.

Von Neumann Computers

- Von Neumann proposed that the instructions can be encoded and stored in the memory just like data.

- During the execution of a program, the stored instructions can be fetched from memory , and the fetched instruction can be decoded to set up the necessary data paths and generate the control signals.

Organization of Von Neumann computer

The organization of a Von Neumann computer is shown schematically in the below figure.

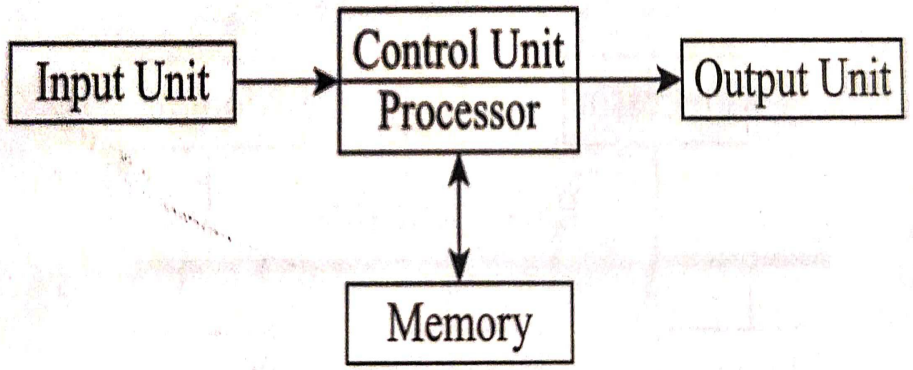


FIGURE 1.2 Organization of a Von Neumann computer.

The figure shows that a control unit is incorporated as a part of the processor.

Von Neumann Computers

- The control unit is an electronic circuitry that decodes each fetched instruction, and generates the necessary control signals that are transmitted to the functional units to carry out the required operations and also the control signals are responsible to automatically set up the required data paths.

- In essence, the stored program concept along with the control unit, eliminated the need for an operator to enter each instruction through switch settings, and also eliminated the need to set up data paths by inserting patch cords.

Von Neumann Computers

- Von Neumann's novel computer organization made the program execution much faster.

Von Neumann Computers

- Execution of programs became faster as manual entry of the instructions as well as setting up of the data and control paths to the functional units could be totally avoided. This gave rise to the concept of an *instruction cycle*.

- In an instruction cycle, an instruction is fetched from memory, analyzed, the data fetched, processed and finally the results are produced. In other words, the execution of an instruction is completed in an instruction cycle.

Shortcomings of Von Neumann computers.

- Towards the end of 1980s, the quest for faster computations made designers to notice a few shortcomings of the Von Neumann computing.

- A shortcoming is the existence of a single connection between the processor and the memory.
- This suggests that at a time only one memory access can occur.
- Hence, at a time either an instruction can be fetched or a data can be accessed. This appeared as the problem in parallel execution of instructions.

Shortcomings of Von Neumann computers.

- The term Von Neumann bottleneck is often used to indicate that both fetching an instruction and accessing (reading and writing) data over the same bus from memory by the processor at the same time in a Von Neumann computer is not possible , this is a bottleneck in achieving high-performance computations.

Basic Organization of a computer

- The important parts of a digital computer are the processor, main memory, hard disk (secondary memory), key board, monitor, and peripheral devices.

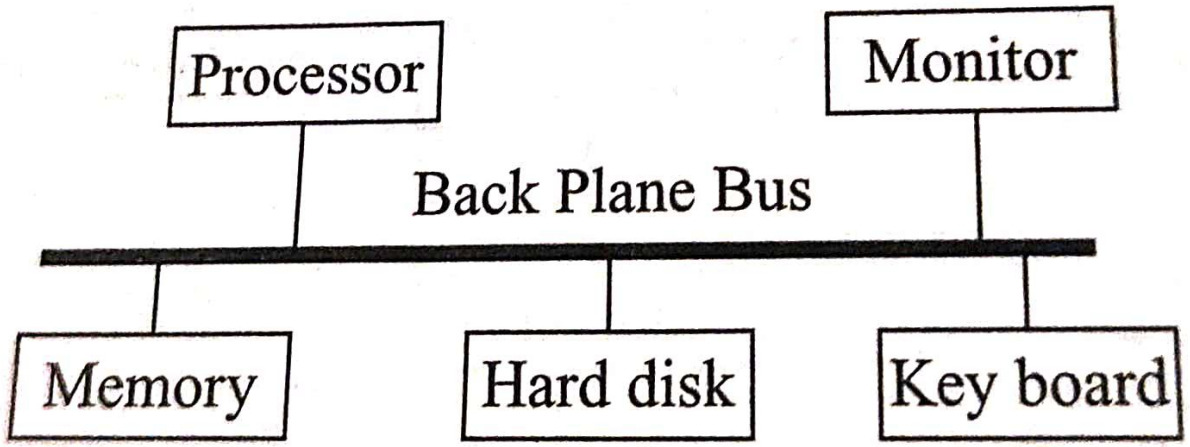
- In simplest organization of a computer, all these different parts of a computer can be connected trough a single bus called backplane bus.

Basic Organization of a computer

A single bus interconnecting all the components of a computer is usually called a backplane bus. This is so because the single bus can be considered to be a backbone communication medium, to which various components are attached.

Although a backplane was used in the early computers, it was later replaced by multiple specialized buses in modern computers for achieving higher performance

Basic organization of a digital computer



Basic organization of a digital computer.

Basic organization of a digital computer

Processor

The processor or the central processing unit (CPU) is responsible for fetching an instruction stored in the memory and executing it.

It contains an arithmetic an logic unit for manipulating data , a number of registers for storing data, and a control unit.

The control unit generates the necessary control signals for fetching and executing the instructions, and storing the results.

Basic organization of a digital computer

Main Memory

The main memory is also called random access memory (RAM) because the CPU can access any location in memory at random and either retrieve the binary information stored at that location, or can store some binary information at the location.

The main memory at present is usually made from dynamic RAM (DRAM) modules.

Basic organization of a digital computer

Other devices are

Monitor

Keyboard

Peripheral devices – Several peripheral devices can be attached to the backplane bus. These peripheral devices can be either output devices, such a printers and loud speakers, or can be input devices, such as scanners and cameras. The CPU can provide output to these output devices or collect input from the input devices over the backplane bus.

Basic organization of a digital computer

Backplane bus

The backplane bus is a group of wires. These wires are partitioned into control and address wires. The control wires carry control signals to different units. The address wires carry the address of the specific data in memory and the data wires are used to carry the data .

Historical Perspective

First generation computers (1941 - 1956)

Second generation computers (1956 - 1963)

Third generation computers (1964 - 1971)

Fourth generation computers (1971 - Present)

Fifth generation computers (Present and Beyond)

Performance Benchmarking

- Before we study the organization and architecture issues, we need to understand how one can determine whether one processor (or computer) works faster than another .

The reasons can be:

- Program execution is faster

  But this knowledge may be inadequate in many situations.

- Next, Comparison of clock rates

Performance Benchmarking

Execution time = CPI X clock cycle time X number of instructions.

CPI – Average clock cycles per instruction.

Clock cycle time is the inverse of clock rate.

This expression indicates that the performance of a computer depends on not only its clock rate but also its CPI.

Performance Benchmarking

Therefore, comparing the clock rates of two processors to determine their relative performance would be a meaningless exercise.

About two decades ago, MIPS (millions of instructions processed per second) and MFLOPS (millions of floating-point operations processed per second) became popular measures for performance evaluations and comparisons. However, there were shortcomings of this.

Performance Benchmarking

The first problem is that the MIPS rating is instruction set dependent.

Second, the MIPS rating of the same processor would vary depending which program is selected for running on the computer to determine the MIPS rating. This is because MIPS rating would depend on whether the program consists of very simple instruction or more complex instructions.

Performance Benchmarking

- Later synthetic benchmarks were proposed as a means to help more accurately determine the performance of computers.

Main concept of synthetic benchmarks :

From a study of a large number of programs, the different types of instructions (arithmetic, branch, memory access, etc.) that are used and their average frequencies of occurrence are determined.

Based on this , a benchmark is a program that is synthesized to comprised a sequence of instructions that has the same distribution of various types of instructions as a typical program.

Performance Benchmarking

That is, a synthetic benchmark program is just a collection of required types of instructions in required frequencies of occurrence , but need not compute any meaningful results.

Examples of synthetic benchmark programs are Dhrystone, Linpack, Khornerstone, etc.

Performance Benchmarking

The synthetic benchmark programs did get used extensively for some time to determine computer performance.

However, the synthetic benchmark programs had an important shortcoming that many times rendered the performance results inaccurate.

All synthetic benchmark programs were rather short programs and compiler writes could produce optimized code that artificially made the computer work well for those programs only.

Performance Benchmarking

- Later, benchmark suite (collection of benchmark programs) was proposed as a solution.
- A popular benchmark suite is available from a nonprofit organization called Standard Performance Evaluation Corporation (SPEC).
- The SPEC benchmark programs can be downloaded free of charge from www.specbench.org

Performance Benchmarking

- The available benchmark suite include : SPEC 2000 for computer system performance, SPEC JVM98 for the performance of the client-side Java virtual machine, SPEC JBB2000 for server-side Java application, SPEC WEB2005 for evaluating web severs, etc.

- So, to determine which computer would work best as a web server among the ones that meet the budget, would require comparing the SPEC WEB2005 benchmark performance of the computers.

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.

As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project.

Like a skeet shooter, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts.

Moore’s law

Abstractions

Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew by Moore's Law.

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. We'll use the abstract painting icon to represent this second great idea.

Abstraction

Abstract painting
icon

Clocking methodology

The approach used to determine when data is valid and stable relative to the clock is clocking methodology.

Edge-triggered clocking A clocking scheme in which all state changes occur on a clock edge.

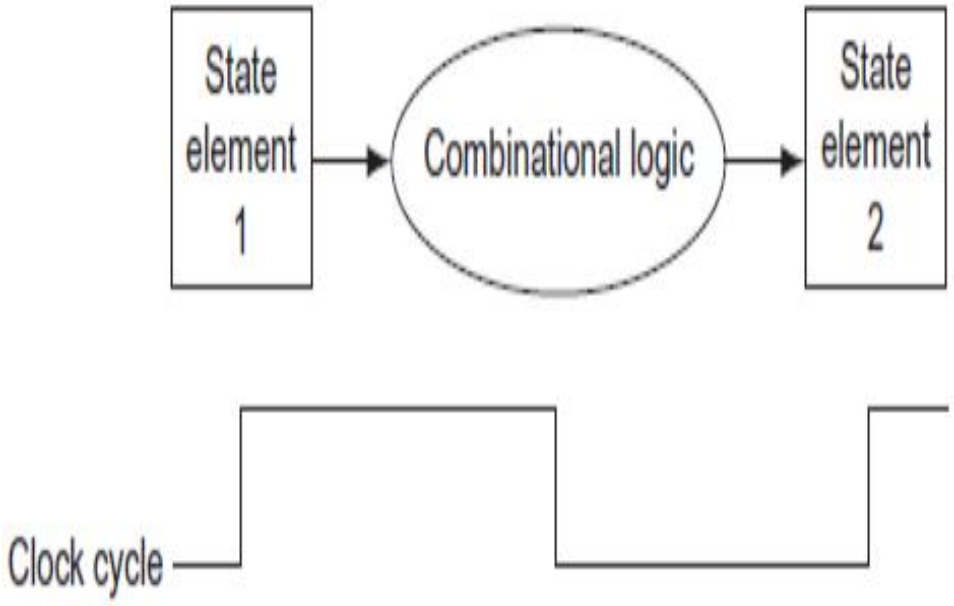
Clocking methodology

- A **clocking methodology** defines when signals can be read and when they can be written.
- It is important to specify the timing of reads and writes, because if a signal is written at the same time it is read, the value of the read could correspond to the old value, the newly written value, or even some mix of the two!
- Computer designs cannot tolerate such unpredictability. A clocking methodology is designed to make hardware predictable.

Clocking methodology

- For simplicity, we will assume an **edge-triggered clocking** methodology. An edge-triggered clocking methodology means that any values stored in a sequential logic element are updated only on a clock edge, which is a quick transition from low to high or *vice versa* (see Figure).
- Because only state elements can store a data value, any collection of combinational logic must have its inputs come from a set of state elements and its outputs written into a set of state elements. The inputs are values that were written in a previous clock cycle, while the outputs are values that can be used in a following clock cycle.

Clocking methodology



Combinational logic, state elements, and the clock are closely related.

In a synchronous digital system, the clock determines when elements with state will write values into internal storage. Any inputs to a state element must reach a stable value (that is, have reached a value from which they will not change until after the clock edge) before the active clock edge causes the state to be updated.

Clocking methodology

All state elements in this chapter, including memory, are assumed to be positive edge-triggered; that is, they change on the rising clock edge.

Clocking methodology

Figure shows the two state elements surrounding a block of combinational logic, which operates in a single clock cycle: all signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time of one clock cycle. The time necessary for the signals to reach state element 2 defines the length of the clock cycle.

Clocking methodology

For simplicity, we do not show a write **control signal** when a state element is written on every active clock edge. In contrast, if a state element is not updated on every clock, then an explicit write control signal is required. Both the clock signal and the write control signal are inputs, and the state element is changed only when the write control signal is asserted and a clock edge occurs.

Clocking methodology

We will use the word **asserted** to indicate a signal that is logically high and *assert* to specify that a signal should be driven logically high, and *deassert* or **deasserted** to represent logically low. We use the terms assert and deassert because when we implement hardware, at times 1 represents logically high and at times it can represent logically low.

Amdahl's law

A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. It is a quantitative version of the law of diminishing returns.

The execution time of the program after making the improvement is given by the following simple equation known as **Amdahl's Law**:

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

Amdahl's law

- We can use Amdahl's Law to estimate performance improvements when we know the time consumed for some function and its potential speedup. Amdahl's Law, together with the CPU performance equation, is a handy tool for evaluating potential enhancements.

Stored Program Architecture

Notion of IPC

Thank you