Unit 1 Fundamentals of Computer Architecture

Structure:

1.1 Introduction

As you all know computers vary greatly in terms of physical size, speed of operation, storage capacity, application, cost, ease of maintenance and various other parameters. The hardware of a computer consists of physical parts that are connected in some way so that the overall structure achieves the pre-assigned functions. Each hardware unit can be viewed at different levels of abstraction. You will find that simplification can go on to still deeper levels. You will be surprised to know that many technologies exist for manufacturing microchips.

The complexity of integration is likely to go on increasing with time. As a consequence smaller and more powerful computers will go on appearing. Evidently, which components are used and how they are interconnected, dictates what the resulting computer will be good at doing. Thus, in a faster computer, you will find special components connected in a special way that enhances the speed of operation of the designed computer.

Different computer designs can have different components. Moreover, the same components can be interconnected in variety of ways. Each design will provide a different performance to the users. Exactly what components interconnected in what ways will produce what performance is the subject of Computer Architecture. In this unit, we will study about the basics of Computer Architecture.

Objectives:

After studying this unit, you should be able to:

- explain computational model and its types
- state the different levels of evolution of computer architecture
- differentiate between process and thread
- describe the concepts of concurrent and parallel execution
- identify the various classification of parallel processing
- list the types of parallelism
- list the levels of parallelism

1.2 Computational Model

Computer architecture may be defined as “The Structure and behavior of a Conceptual model of a Computer System to perform the required functionalities”.

Computer Architecture deals with the issue of selection of hardware components and interconnecting them to create computers that achieve specified functional, performance and cost goals.

Progressing in the earlier mentioned way, the hardware (at least the electronic part) breaks down to the following simple digital components.

- Registers
- Counters
- Adders

- Multiplexers
- De-multiplexers
- Coders
- Decoders
- I/O Controllers

A common foundation or paradigm that links the computer architecture and language groups is called a **Computational Model**. The concept or idea of computational model expresses a higher level of abstraction than can be achieved by either the computer architecture or the programming language alone, and includes both.

The computational model consists of the subsequent three abstractions:

1. The basic items of computations
2. The problem description model
3. The execution model

Unlike the ordinary delusions, the set of abstractions that must be selected to state computational models is not very clear. Some criteria will define fewer but relatively basic computational models, while a wide variety of criteria will result in a fairly a huge quantity of different models.

1.2.1 The basic items of computations

This concept recognises the basic items of computation. This is a requirement of the items to which the computation is referred and the sort of computations (operations) that are executed on them. For example, in the von Neumann computational model, the fundamental items of computation are data.

This data will normally be characterised by individual bodies so as to be capable of distinguishing among several different data items in the course of a computation. These identifiable bodies are commonly called variables in programming languages and are put into operation by register addresses or memory in architectures.

The acknowledged computational models, such as Turing model, the von Neumann model and the data flow model stand on the theory of data. These models are briefly explained as below:

The **Turing machine architecture** operates by manipulating symbols on a tape. In other words, a tape with innumerable slots exists, and at any one point in time, the Turing machine is in a specific slot. The machine can change the symbol and shift to a different slot based on the symbol read at that slot. All of this is inevitable.

The **von-Neumann architecture** explains the stored-program computer where data and instructions are stored in memory and the machine works by varying its internal state, In other words, an instruction operates on some data and changes the data. So naturally, there is a state maintained in the system.

Dataflow architecture expressively distinguishes the conventional von Neumann architecture or control flow architecture. There is a lack of a program counter in Dataflow architectures. The execution of instructions in dataflow systems is exclusively concluded depending on the accessibility of input arguments to the instructions. Even though dataflow architecture has not been used in any commercially successful computer hardware, it is extremely appropriate in many software architectures such as database engine designs and parallel computing frameworks.

On the other hand, there are various models independent of data. In these models, the basic items of computation are:

- Messages or objects sent to them needing an associated manipulation (as in the object-based model)
- Arguments and the functions applied on them (applicative model)
- Elements of sets and the predicates declared on them (predicate-logic-based model).

1.2.2 The problem description model

The problem description model implies in cooperation the style and method of problem description. The problem description style specifies the way troubles in a specific computational model are expressed. The style is either procedural or declarative. The algorithm to work out the problem is shown in a procedural style. A particular result is then stated in the form of an algorithm. In a declarative style, all the facts and dealings significant to the specified problem have to be stated.

There are two modes for conveying these relationships and facts. The first employs functions, as in the applicative model of computation, while the second declares the relationships and facts in the form of predicates, as in the predicate-logic-based computational model. Now, we will study the second component of the problem description model that is the problem description method. It is understood in a different way for the procedural and the declarative style. In the procedural style, the problem description model states the way in which the clarification of the known problem has to be explained. On the contrary, while using the declarative style, it states the method in which the difficulty itself has to be explained.

1.2.3 The execution model

This is the third and the final constituent of computational model. It can be divided into three stages.

- Interpretation of how to perform the computation
- Execution semantics
- Control of the execution sequences

The first stage pronounces the analysis of the computation, which is strongly linked to the problem description method. The selection of problem description method and the analysis of the computation are mutually dependent on one another.

The subsequent stage of the execution model states the execution semantics. This is taken as a rule that identifies the way a particular execution step is to be performed. This rule is, certainly, linked with the selected problem description method and the way the implementation of the computation is understood. The final stage of the model states the rule of the execution sequences. In the basic models, implementation is either control driven or data driven or demand driven.

- In Control driven execution, it is supposed that there is a program consisting of a succession of instructions. The execution sequence is then absolutely specified by the command of the directions. Nevertheless, explicit control instructions can also be used to identify an exit from the implied execution sequence.
- Data-driven execution is symbolised by the rule that an operation is made active instantly after all the needed input data is available. Data-

driven execution control is characteristic of the dataflow model of computation.

- In Demand-driven execution, the operations will be made active only when their implementation is required to attain the ultimate result. Demand-driven execution control is normally used in the applicative computational model.

Self Assessment Questions

1. The __________ model refers to both the style and method of problem description.
2. In a __________ , the algorithm for solving the problem is stated.
3. __________ execution is characterised by the rule that an operation is activated as soon as all the needed input data is available.

1.3 Evolution of Computer Architecture

With the advent of revolutionary development in area of semiconductor technology, the computer architecture has gradually evolved in stages over the years. The main target of such evolution is to enhance the performance of the processors. History of computers begins with the invention of the abacus in 3000 BC, followed by the invention of mechanical calculators in 1617. The years beyond 1642 till 1980 are marked by inventions of zeroth, first, second and third generation computers. The years beyond 1980 till today, are marked by fourth generation computers. Fifth generation computers are still under research and development.

Zeroth Generation Computers: The zeroth generation of computers (1642-1946) was distinctly made available by the invention of largely mechanical computers. In 1642, a French mathematician named Blaise Pascal invented the first mechanical device which was called Pascaline. In 1822, Charles Babbage, an English mathematician, invented a machine called Difference Engine to compute tables of numbers for naval navigation. Later on, in the year 1834, Babbage attempted to build a digital computer, called Analytical Engine. The analytical engine had all the parts of a modern computer i.e. the store (memory unit), the mill (computation unit), the punched card reader (input unit) and the punched/ printed output (output unit). As all the basic parts of modern computers were thought out by **Charles Babbage**, he is known as **Father of Computers**.

First Generation Computers: The first generation of computers (1946-1954) was marked by the use of vacuum tubes or valves as their basic electronic component. Although these computers were faster than earlier mechanical devices, they had many disadvantages. First of all, they were very large in size. They consumed too much power and generated too much heat, when used for even short duration of time. They were very unreliable and broke down frequently. They required regular maintenance and their components had also to be assembled manually.

Some examples of first generation computers are ENIAC (Electronic Numerical Integrator and Calculator), EDVAC (Electronic Discrete Variable Automatic Computer), EDSAC (Electronic Delay Storage Automatic Calculator), UNIVAC I (Universal Automatic Calculator) and IAS machine (Institute for Advanced Study machine built by Princeton's Institute for Advanced Study). The basic design of first generation computer is shown in figure 1.1.

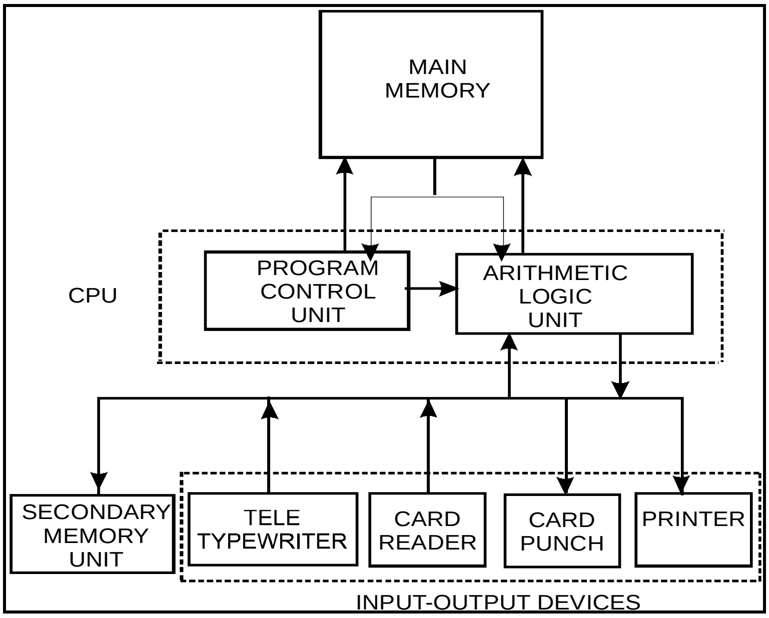

Figure 1.1: Basic Design of a First Generation Computer

IAS machine was a new version of the EDVAC, which was built by von Neumann. The basic design of IAS machine is now known as von Neumann machine, which had five basic parts - the memory, the arithmetic logic unit, the program control unit, the input and output unit as shown in figure 1.2.

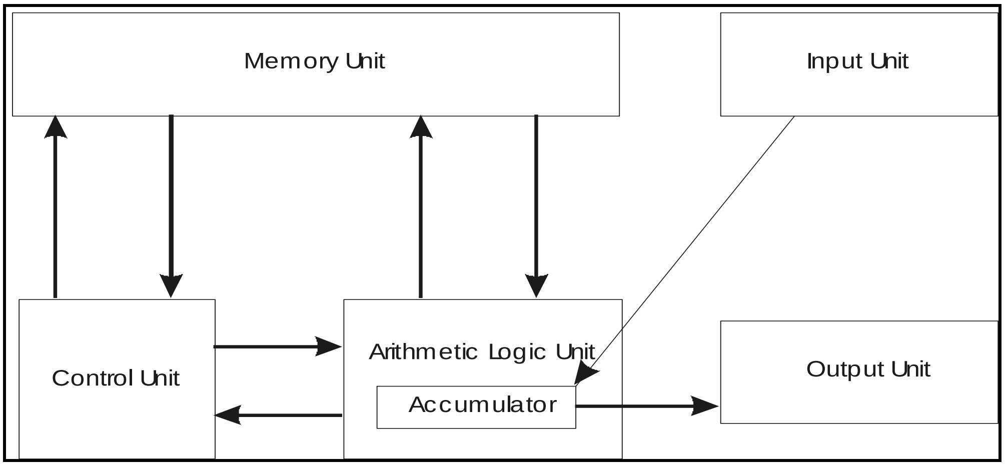

Figure 1.2: Basic Design of von Neumann Machine

Second Generation Computers: The first generation of computers became out-dated, when in 1954, the Philco Corporation developed transistors that can be used in place of vacuum tubes. The second generation of computers (1953-64) was marked by the use of transistors in place of vacuum tubes. Transistors had a number of advantages over the vacuum tubes. As transistors were made from pieces of silicon, so they were more compact than vacuum tubes.

The second-generation computers were smaller in size and generated less heat than first generation computers. Although they were slightly faster and more reliable than earlier computers, they also had many disadvantages.

They had limited storage capacity, consumed more power and were also relatively slow in performance. Some examples of second generation computers are IBM 701, PDP-1 and IBM 650.The basic design of a second generation computer is shown in figure 1.3.

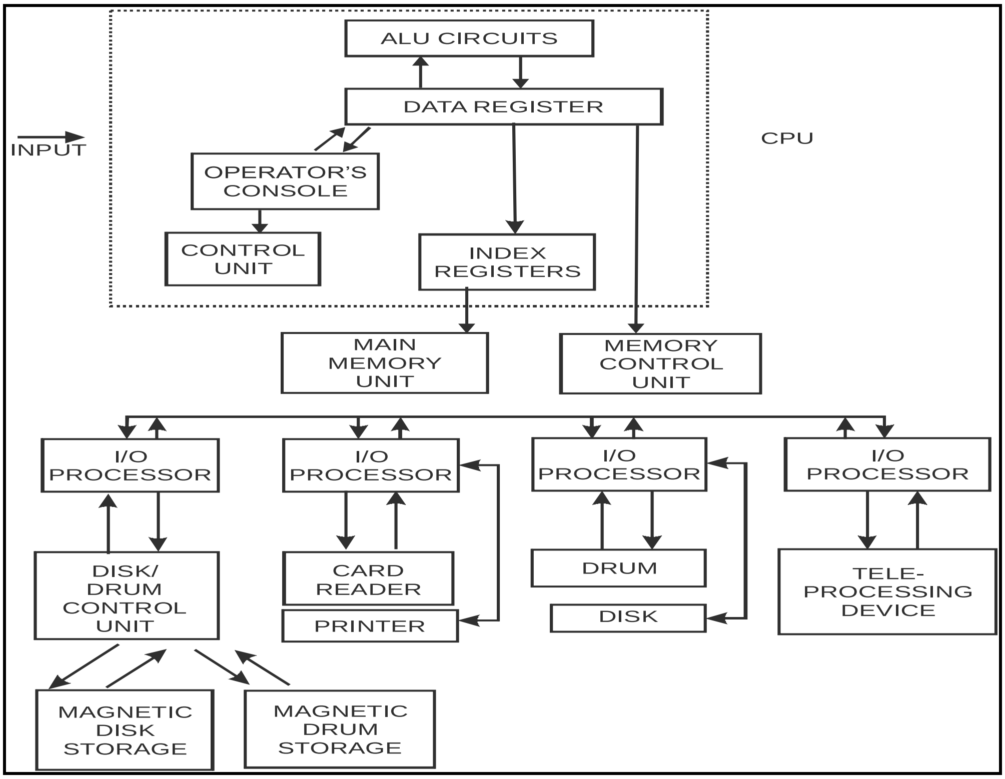

Figure 1.3: Basic Design of Second Generation Computer

Third Generation Computers: Second generation computers became out-dated after the invention of ICs. The third generation of computers (1964-1978) was marked by use of Integrated Circuits (ICs) in place of transistors. As hundreds of transistors could be put on a single small circuit, so ICs were more compact than transistors. The third generation computers removed many drawbacks of second generation computers. The third generation computers were even smaller in size, very less heat generated and required very less power as compared to earlier two generation of computers. These computers required less human labour at the assembly stage.

Some examples of third generation computers are IBM 360, PDP-8, Cray-1 and VAX. The basic design of a third generation computer is shown in figure 1.4.

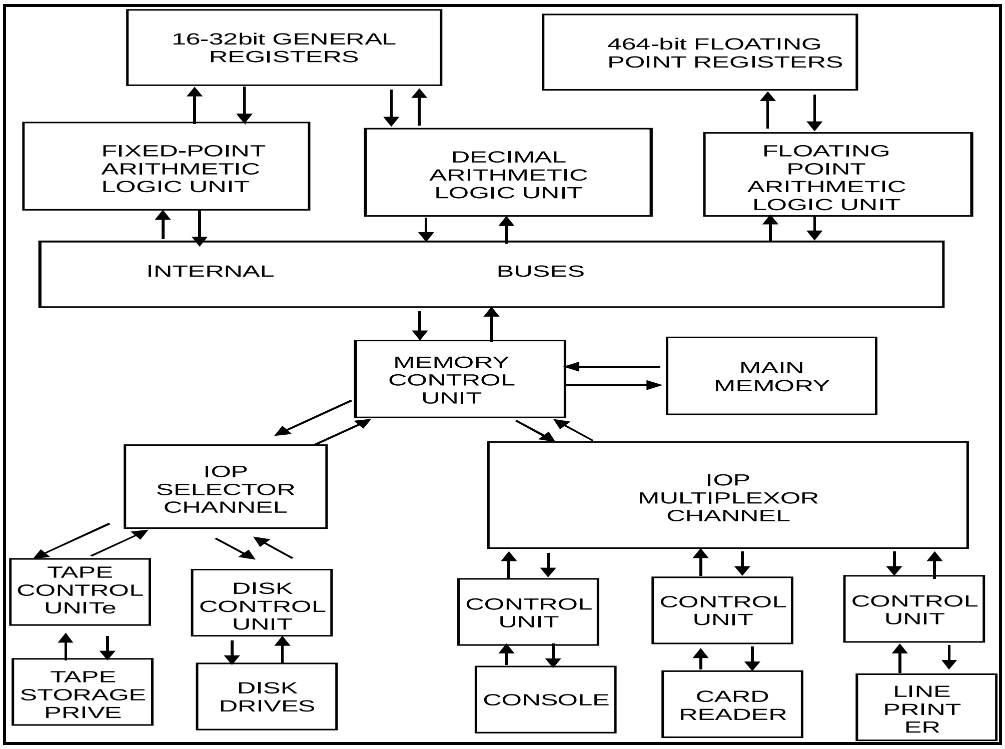

Figure 1.4: Basic Design of a Third Generation Computer

Fourth Generation Computers: The third generation computers became out-dated, when it was found in around 1978, that thousands of ICs could be integrated onto a single chip, called LSI (Large Scale Integration).

The fourth generation of computers (1978-till date) was marked by use of large-scale Integrated (LSI) circuits in place of ICs. As thousands of ICs could be put onto a single circuit, so LSI circuits are still more compact than ICs. In 1978, it was found that millions of components could be packed onto a single circuit, known as Very Large Scale Integration (VLSI). VLSI is the latest technology of computer that led to the development of the popular Personal Computers (PCs), also called as Microcomputers.

Some examples of fourth generation computers are IBM PC, IBM PC/AT, 386, 486, Pentium and CRAY-2. The basic design of a fourth generation computer is shown in figure 1.5.

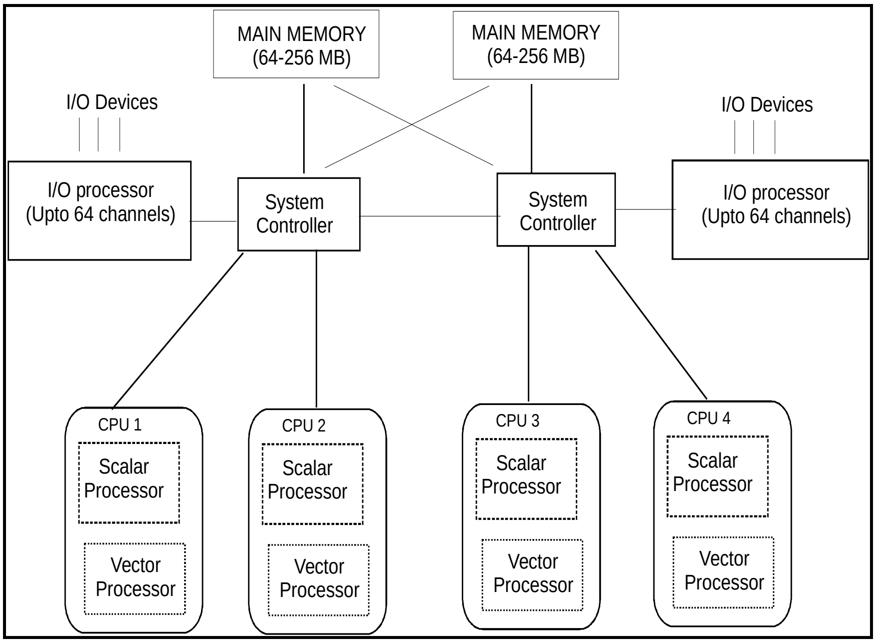

Figure 1.5: Basic Design of a Fourth Generation Computer

Fifth Generation Computers: Although fourth generation computers offer too many advantages to users, still they have one main disadvantage. The major drawback of these computers is that they have no intelligence on their own. Scientists are now trying to remove this drawback by making computers, which would have artificial intelligence. The fifth generation computers (Tomorrow's computers) are still under research and development stage. These computers would have artificial intelligence.

They will use USLI (Ultra Large-Scale Integration) chips in place of VLSI chips. One USLI chip contains millions of components on a single IC. Robots have some features of fifth generation computers.

Self Assessment Questions

4. ________ was the first mechanical device, invented by Blaise Pascal.
5. __________ was a new version of the EDVAC, which was built by von Neumann.
6. The fourth generation of computers was marked by use of Integrated Circuits (ICs) in place of transistors. (True/ False)
7. Personal Computers (PCs), also called as Microcomputers.

(True/ False)

Activity 1:
Using the Internet, find out about Fifth Generation Computer Systems project (**FGCS**), idea behind it, implementation, timeline and outcome

1.4 Process and Thread

Every process presents the resources required to execute a program. A process has an executable code, a virtual address space, open handles to system objects, a unique process identifier, a security context, minimum and maximum working set sizes, environment variables, a priority class, and at least one thread of execution. Each process is begun with a single thread, often called the primary thread, but can create additional threads from any of its threads.

A thread is the entity within a process that can be scheduled for execution. All threads of a process share its system resources and virtual address space. Additionally, each thread maintains exception handlers, thread local storage, a scheduling priority, a unique thread identifier, and a set of structures the system will utilise to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, a thread environment block, the kernel stack and a user stack in the address space of the thread's process. Threads can also have their own security context, which is valuable in impersonating clients.

The basic difference between process and thread is that every process has its own data memory location but all related threads can share same data memory and have their individual stacks. A process is a collection of virtual memory space, code, data and system resources whereas thread is a code which will be serially executed within a process.

Let’s study these concepts in detail.

1.4.1 Concept of process

In operating system terminology, instead of the term ‘program’, the notion of process is used in connection with execution. It designates a commission or job, or a quantum of work dealt with as an entity. Consequently, the resources required, such as address space, are typically allocated on a process basis. Each process has a life cycle, which consists of creation, an execution phase and termination.

Process creation involves the following four main actions:

- ***Setting up the process description:*** Usually, operating systems describe a process by means of a description table which is called the Process Control Block or PCB. A PCB contains all the information relevant to the whole life cycle of a process. It holds basic data such as process identification, owner, process status, description of the allocated address space and so on.
- ***Allocating address space:*** Allocation of address space to a process for execution is the second major component of process creation. This consists of two approaches: sharing the address space among the created processes (shared memory) or allocating distinct address spaces to each process (per-process address spaces).
- ***Loading the program into the allocated address space*:** Subsequently, the executable program file will usually be loaded into the allocated memory space.
- ***Passing the process description to the scheduler:*** Finally, the process thus created is passed to the process scheduler which allocates the processor to the competing processes. The process scheduler manages processes typically by setting up and manipulating queues of PCBs. Thus, after creating a process the scheduler puts the PCB into ready-to-run processes.

Process scheduling involves three key concepts: the declaration of distinct process states, the specification of the state transition diagram and the statement of a scheduling policy. As far as process states are concerned, there are three basic states connected with scheduling:

- The ready-to-run state
- The running state and
- The wait (or blocked) state.

In the wait state, they are suspended or blocked waiting for the occurrence of some event before getting ready to run again. When the scheduler selects a process for execution, its state is changed from ready-to-run to running. Finally, a process in the wait can go into the ready-to-run state, if the event it is waiting for has occurred. You can see various process states in figure 1.6.

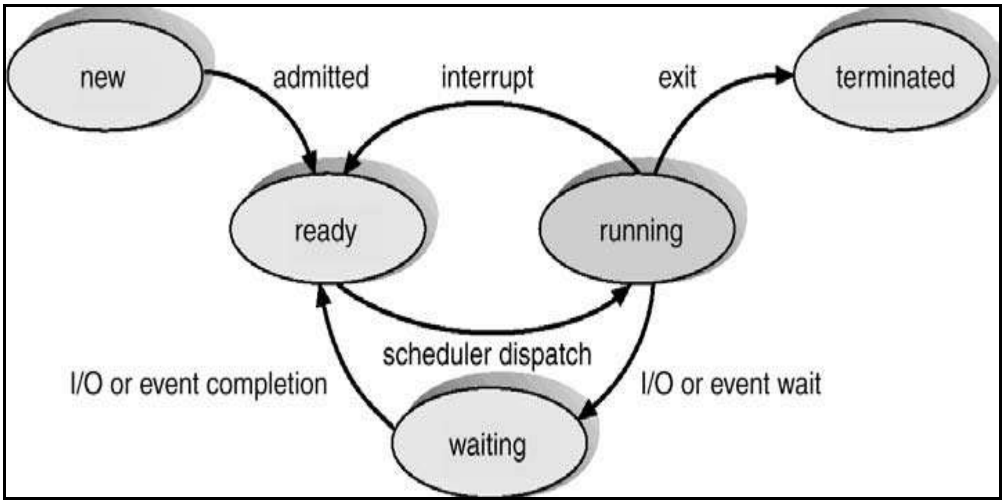

Figure 1.6: Process States

1.4.2 Concept of thread

A **thread** is a fundamental unit of CPU consumption, which consists of a program counter, a stack, and a set of registers and a thread ID. Conventional heavyweight processes consist of a single thread of control. In other words, there is one program counter, and one sequence of instructions that can be carried out at any specified time.

At present, multi-threaded applications have taken the place of single thread applications. These have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files. See figure 1.7 to find out the differences between the two processes.

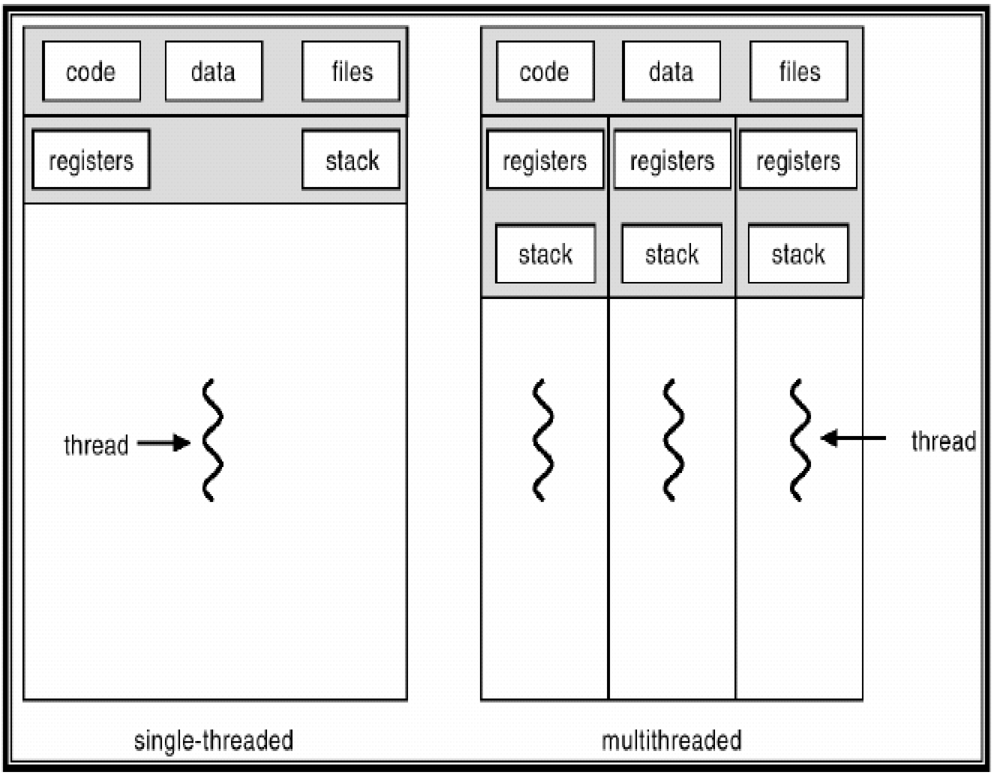

Figure 1.7: Single and Multithreaded Processes

Threads are of great use in recent programming particularly when a process has multiple tasks to perform in parallel of the others. This is mainly helpful when one of the tasks may block, and it is needed to permit the other tasks to continue without blocking. For example, in a word processor, a background thread may check spelling and grammar while a foreground thread processes user input (keystrokes), while yet a third thread loads images from the hard drive, and a fourth does periodic automatic backups of the file being edited.

Self Assessment Questions

8. All threads of a process share its virtual address space and system resources. (True/ False)
9. When the scheduler selects a process for execution, its state is changed from ready-to-run to the wait state. (True/ False)

1.5 Concepts of Concurrent and Parallel Execution

Concurrent execution is the temporal behaviour of the N-client 1-server model where one client is served at any given moment. This model has a dual nature; it is sequential in a small time scale, but simultaneous in a rather large time scale. In this situation, the key problem is how the competing clients, let us say processes or threads, should be scheduled for service (execution) by the single server (processor). The scheduling policy may be viewed as covering the following two aspects:

Pre-emption rule: It deals with whether servicing a client can be interrupted or not and, if so, on what occasions. The pre-emption rule may either specify time-sharing, which restricts continuous service for each client to the duration of a time slice, or can be priority based, interrupting the servicing of a client whenever a higher priority client requests service.

Selection rule: It states how one of the competing clients is selected for service. The selection rule is typically based on certain parameters, such as priority, time of arrival, and so on. This rule specifies an algorithm to determine a numeric value, which we will call the rank, from the given parameters. During selection, the ranks of all competing clients are computed and the client with the highest rank is scheduled for service.

Parallel execution: Parallel execution is associated with N-client N-server model. Having more than one server, allows the servicing of more than one client at the same time; this is called ***parallel execution***. Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. It may take the use of multiple CPUs. A problem is broken into discrete parts that can be solved concurrently. Each part is further broken down to a series of instructions and instructions from each part execute simultaneously on different CPUs as shown in figure 1.8.

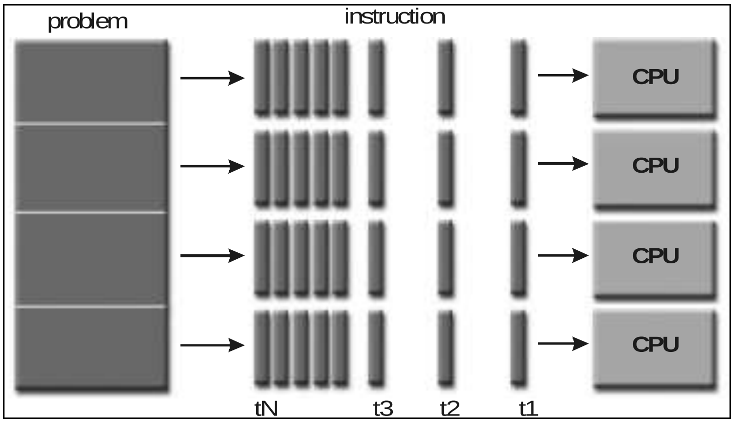

Figure 1.8: Parallel Computing Systems

Thus, we can say that a computer system is said to be Parallel Processing System or Parallel Computer if it provides facilities for simultaneous processing of various set of data or simultaneous execution of multiple instruction. On a computer with more than one processor each of several processes can be assigned to its own processor, to allow the processes to progress simultaneously. If only one processor is available the effect of parallel processing can be simulated by having the processor run each process in turn for a short time.

Self Assessment Questions

10. Concurrent execution is the temporal behaviour of the ______ Model.
11. During selection, the ranks of all competing clients are computed and the client with the highest rank is scheduled for service. (True/ False)

1.6 Classification of Parallel Processing

The core element of parallel processing is Central Processing Units (CPUs). The essential computing process is the execution of sequence of instruction on asset of data. The term stream is used here to denote a sequence of items as executed by single processor or multiprocessor. Based on a number of instruction and data streams can be processed simultaneously,

Flynn classifies the computer system into four categories. They are:

(a) Single Instruction Single Data (SISD)
(b) Single Instruction Multiple Data (SIMD)

(c) Multiple Instruction Single Data (MISD)
(d) Multiple Instruction Multiple Data (MIMD)

Let's learn more about them.

1.6.1 Single instruction single data (SISD)

Computers with a single processor that is capable of executing scalar arithmetic operations using a single instruction stream and a single data stream are called SISD (Single Instruction Single Data) computers. They are characterised by:

Single instruction: Only single instruction stream/linearised set is being acted on by the CPU during any one clock-cycle.

Single data: Merely a distinct data stream is being used as input during any one clock-cycle.

This is the oldest and of late, the most widespread structure of computer.

Examples: Most PCs, single CPU workstations and mainframes.
Figure 1.9 shows an example of SISD.

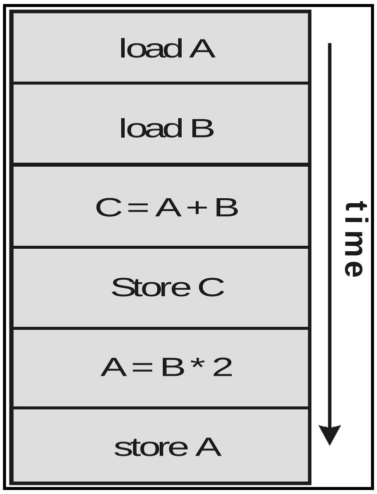

Figure 1.9: SISD Process

1.6.2 Single instruction multiple data (SIMD)

Computers with a single processor that is capable of executing vector arithmetic operations using a single instruction stream but multiple data streams are called SIMD (Single Instruction Multiple Data) computers. They are characterised by:

Single instruction: Every processing unit perform the identical instruction at every known clock-cycle.

Multiple data: Each processing unit can operate on a different data element.

This category of machine characteristically has an instruction dispatcher, a very high-bandwidth in-house arrangement, and a very large array of very small-capacity instruction units. It is best suitable for specialised problems characterised by a high level of consistency, such as image processing. Figure 1.10 shows an example of SIMD processing.

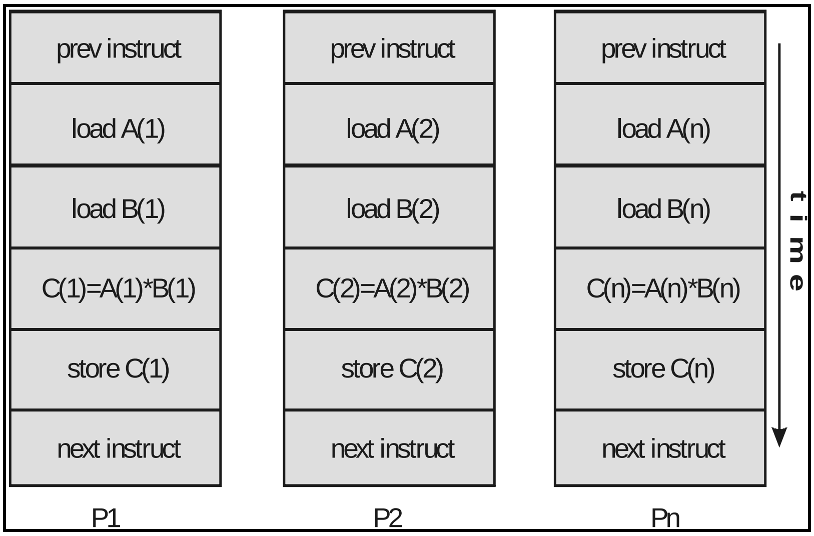

Figure 1.10: SIMD Process

1.6.3 Multiple instruction single data (MISD)

Computers with multiple processors that are capable of executing different operations using multiple instruction streams but single data stream are called MISD (Multiple instruction Single Data) computers. They are characterised by:

Multiple Instructions: Every processing unit functions on the data alone via independent instruction streams.

Single data: A single data stream is entered into multiple processing units.

Some conceivable uses of this architecture are in multiple frequency filters functional on a single signal stream and multiple cryptography algorithms trying to crack a single coded message. Figure 1.11 shows an example of MISD processing.

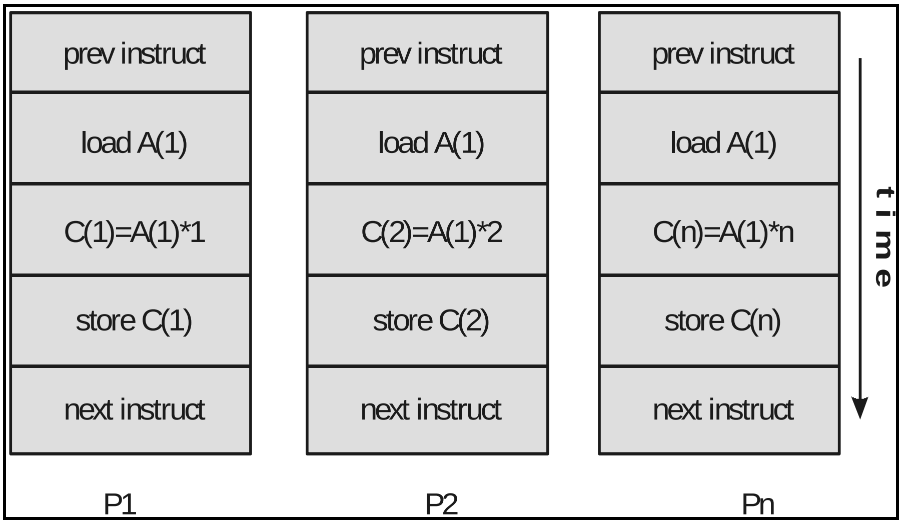

Figure 1.11: MISD Process

1.6.4 Multiple Instruction Multiple Data (MIMD)

Computers with multiple processors that are capable of executing vector arithmetic operations using multiple instruction streams and multiple data streams are called MIMD (Multiple Instruction Multiple Data) computers. They are characterised by:

Multiple Instructions: Each processor may be performing a dissimilar instruction stream.

Multiple Data: Each processor may be working with a dissimilar data stream.

It is the most common type of parallel computer. Most modern computers fall into this category. Execution can be synchronous or asynchronous, deterministic or non-deterministic.

Examples: most current supercomputers, networked parallel computer “grids” and multi-processor computers. Figure 1.12 shows a case of MISD processing.

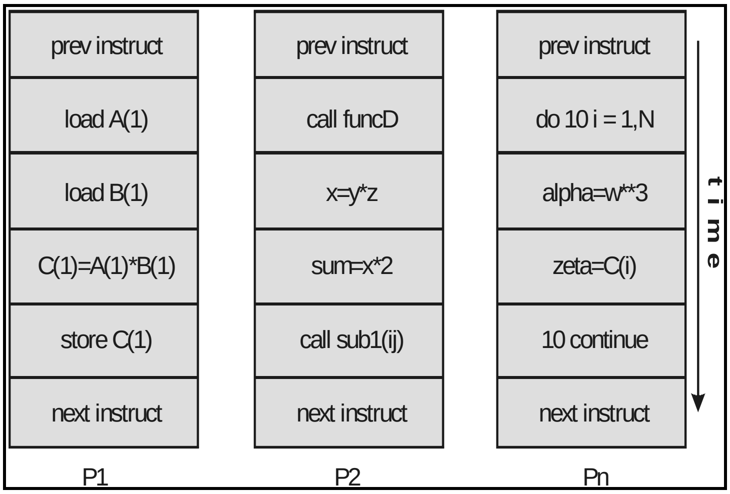

Figure 1.12: MIMD Process

Self Assessment Questions

12. In __________ all processing units execute the same instruction at any given clock cycle.
13. In which system a single data stream is fed into multiple processing units?
14. __________ is the most common type of parallel computer.

1.7 Parallelism and Types of Parallelism

A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. This definition broadly includes parallel supercomputers that have more than hundreds of processors, networks of workstations, embedded systems and multiple-processor workstations. Parallel computers have the potential to concentrate computational resources like processors, memory, or I/O bandwidth on important computational problems. The following are the various types of parallelism:

Bit-level parallelism: Bit-level parallelism is a form of parallel computing based on increasing processor word size. From the advent of very-large-scale integration (VLSI) computer chip fabrication technology in the 1970s until about 1986, advancements in computer architecture were conducted by increasing bit-level parallelism

Instruction-level parallelism: A computer program is a stream of linearised instructions carried out by a processor. These commands can be rearranged and united into groups which are then acted upon in parallel without altering the outcome of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mid-1980s until the mid-1990s.

Data parallelism: Data parallelism is parallelism inbuilt in program loops. It centres at allocating the data across various computing nodes to be processed in parallel. Parallelising loops recurrently leads to related (not necessarily identical) operation sequences or functions being performed on elements of a large data structure. Many scientific and engineering applications display data parallelism.

Self Assessment Questions

15. Parallel computers offer the potential to concentrate computational resources on important computational problems. (True/ False)
16. Advances in instruction-level parallelism dominated computer architecture from the mid-1990s until the mid-2000s. (True/False)

1.8 Levels of Parallelism

Parallelism is one of the most popular ideas in computing. Architectures, compilers and operating system have been striving for more than two decades to extract and utilise as much parallelism as possible in order to speed up computation. The notion of parallelism is used in two different contexts. Either it designates available parallelism in programs or it refers to parallelism occurring during execution, called utilised parallelism.

Types of available parallelism: Problem solutions may contain two different kinds of available parallelism, called functional parallelism and data parallelism.

Functional parallelism is that kind of parallelism which arises from the logic of a problem solution. On the contrary, data parallelism comes from using data structures that allow parallel operations on their elements, such as vectors or matrices, in problem solutions. From another point of view, parallelism can be considered as being either regular or irregular. Data parallelism is regular, whereas functional parallelism, with the execution of loop-level parallelism, is usually irregular.

Levels of available functional parallelism: Programs written in imperative languages may represent functional parallelism at different levels, that is, at different sizes of granularity. In this respect, we can identify the following four levels and corresponding granularity sizes:

- ***Parallelism at the instruction level (fine-grained parallelism):*** Available instruction-level parallelism means that particular instructions of a program may be executed in parallel. To this end, instructions can be either assembly (machine-level) or high-level language instructions. Usually, instruction-level parallelism is understood at the machine-language (assembly-language) level.
- ***Parallelism at the loop level (middle-grained parallelism):*** Parallelism may also be available at the loop level. Here, consecutive loop iterations are candidates for parallel execution. However, data dependencies between subsequent loop iterations, called recurrences, may restrict their parallel execution.
- ***Parallelism at the procedure level (middle-grained parallelism):*** Next, there is parallelism available at the procedure level in the form of parallel executable procedures. The extant of parallelism exposed at this level is subject mainly to the kind of the problem solution considered.
- ***Parallelism at the program level (coarse-grained parallelism):*** Lastly, different programs (users) are obviously independent of each other. Thus, parallelism is also available at the user level (which we consider to be coarse-grained parallelism). Multiple, independent users are a key source of parallelism occurring in computing scenarios.

Utilisation of functional parallelism: Available parallelism can be utilised by architectures, compilers and operating systems conjointly for speeding up computation. Let us first discuss the utilisation of functional parallelism.

In general, functional parallelism can be utilised at four different levels of granularity, that is,

- Instruction
- Thread
- Process
- User level

It is quite natural to utilise available functional parallelism, which is inherent in a conventional sequential program, at the instruction level by executing instructions in parallel. This can be achieved by means of architectures capable of parallel instruction execution. Such architectures are referred to as instruction-level function-parallel architectures or simply instruction-level parallel architectures, commonly abbreviated as ILP-architectures.

Available functional parallelism in a program can also be utilised at the thread and/or at the process level. Threads and processes are self-contained execution entities embodying an executable chunk of code. Threads and processes can be created either by the programmer using parallel languages or by operating systems that support multi-threading or multitasking. They can also be automatically generated by parallel compilers during compilation of high-level language programs. Available loop and procedure-level parallelism will often be exposed in the form of threads and processes.

Self Assessment Questions

17. Parallelism occurring during execution is called ——————.
18. Parallelism at the instruction level is also called middle-grained parallelism. (True/ False)
19. Data parallelism is regular, whereas functional parallelism, with the execution of loop-level parallelism, is usually irregular. (True/ False)

Activity 2:
Decide which architecture is most appropriate for a given application. First determine the form of parallelisation which would best suit the application, then decide both hardware and software for running your parallelised application

1.9 Summary

Let us recapitulate the important concepts discussed in this unit:

- Computer Architecture deals with the issue of selection of hardware components and interconnecting them to create computers that achieve specified functional, performance and cost goals.

- The concept of a computational model represents a higher level of abstraction than can be achieved by either the computer architecture or the programming language alone, and covers both.
- History of computers begins with the invention of the abacus in 3000 BC, followed by the invention of mechanical calculators in 1617. Fifth generation computers are still under research and development.
- Each process provides the resources needed to execute a program.
- A thread is the entity within a process that can be scheduled for execution.
- Concurrent execution is the temporal behaviour of the N-client 1-server model where one client is served at any given moment.
- Parallel execution is associated with N-client N-server model.
- Based on a number of instruction and data streams can be processed simultaneously, Flynn classifies the computer system into four categories.
- The notion of parallelism is used in two different contexts and three different types. Either it designates available parallelism in programs or it refers to parallelism occurring during execution, called utilised parallelism.

1.10 Glossary

- **EDSAC:** Electronic Delay Storage Automatic Calculator
- **EDVAC:** Electronic Discrete Variable Automatic Computer
- **ENIAC:** Electronic Numerical Integrator and Calculator
- **IC:** Integrated Circuit where hundreds of transistors could be put on a single small circuit.
- **LSI:** Large Scale Integration, it can pack more than a million transistors
- **MSI:** Medium Scale Integration, it packs as many as 100 transistors
- **PCB:** Process Control Block, it is a description table which contains all the information relevant to the whole life cycle of a process.
- **SSI:** Small Scale Integration, it can pack 10 to 20 transistors in a single chip.
- **UNIVAC I:** Universal Automatic Calculator
- **USLI:** Ultra Large-Scale Integration, it contains millions of components on a single IC
- **VLSI:** Very Large Scale Integration, it can have up to 1000 transistors

1.11 Terminal Questions

1. Explain the concept of Computational Model. Describe its various types.
2. What are the different stages of evolution of Computer Architecture? Explain in detail.
3. What is the difference between process and thread?
4. Explain the concepts of concurrent and parallel execution.
5. State Flynn's classification of Parallel Processing.
6. Explain the types of parallelism.
7. What are the various levels of parallelism?

1.12 Answers

Self Assessment Questions

1. Problem description
2. Procedural style
3. Data-driven
4. Pascaline
5. IAS machine
6. False
7. True
8. True
9. False
10. N-client 1-server
11. True
12. Single Instruction Multiple Data
13. Multiple Instruction Single Data
14. Multiple Instruction Multiple Data
15. True
16. False
17. Utilised parallelism
18. False
19. True

Terminal Questions

1. A common foundation or paradigm that links the computer architecture and language classes is called a Computational Model. Refer Section 1.2.

2. History of computers begins with the invention of the abacus in 3000 BC, followed by the invention of mechanical calculators in 1617. The years beyond 1642 till 1980 are marked by inventions of zeroth, first, second and third generation computers. Refer Section 1.3.
3. A thread is the entity within a process that can be scheduled for execution. Refer Section 1.4.
4. Concurrent execution is the temporal behaviour of the N-client 1-server model where one client is served at any given moment. Parallel execution is associated with N-client N-server model. Refer Section 1.5.
5. Flynn classifies the computer system into four categories. Refer Section 1.6.
6. There are three types of parallelism. Refer section 1.7.
7. The notion of parallelism is used in two different contexts. Either it designates available parallelism in programs or it refers to parallelism occurring during execution, called utilised parallelism. Refer Section 1.8.

References:

- Hwang, K. (1993) *Advanced Computer Architecture*. McGraw-Hill, 1993.
- D. A. Godse & A. P. Godse (2010). *Computer Organization*. Technical Publications. pp. 3–9.
- John L. Hennessy, David A. Patterson, David Goldberg (2002) "*Computer Architecture: A Quantitative Approach*", Morgan Kaufmann; 3rd edition.
- Dezsö Sima, Terry J. Fountain, Péter Kacsuk (1997) *Advanced computer architectures - a design space approach*. Addison-Wesley-Longman: I-XXIII, 1-766

E-references:

- www.cs.clemson.edu/~mark/hist.html
- www.people.bu.edu/bkia/
- www.ac.upc.edu/
- www.inf.ed.ac.uk/teaching/courses/car/