



Computer Architecture



Course Design Committee

Prof. Ashutosh Gupta **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Allahabad

Prof. Suneeta **Agarwal**
Member
Department of CSE
MNNIT Allahabad, Prayagraj

Dr. Upendra **Nath** **Tripathi**
Member
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish **Khare**
Member
Associate Professor, Department of Computer Science
University of Allahabad, Prayagraj

Ms. Marisha **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Allahabad

Mr. Manoj Kumar Balwant **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Allahabad

Course Preparation Committee

Dr. Brajesh Kumar **Author**
Associate Professor
Department of CS & IT
Mahatma Jyotiba Phule Rohilkhand University- Bareilly

Dr. Manu Pratap Singh **Editor**
Professor, Department of Computer Science Engineering
Dr. Bhimrao Ambedkar University, Agra

Mr. Manoj Kumar **Balwant**
Coordinator
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad



Uttar Pradesh Rajarshi Tandon

Bachelor of Computer Application

BCA-EC
Open University

Block

1

PROCESSOR BASICS

Unit 1

CPU Organization

Unit 2

Data Representation

Unit 3

Instruction Sets

BLOCK INTRODUCTION

This block deals with the basic concepts of computer architecture. It provides the insight into fundamentals of computer organization, data representation, and instructions. The block is divided into three units: Unit 1, Unit 2, and Unit 3. The Unit 1 discusses the evolution of computer systems with emphasis on technological development, processing, storage, etc. The development of computers from the first generation to the fifth generation is presented. The central processing unit and its major components are explained. Unit 2 introduces the number systems and different number representations. The methods of conversion from one number system to other number systems are explained. Unit 3 explains how the computer system is given commands with the help of computer instructions. The computer instruction formats and their memory representation are discussed.

UNIT-1 CPU Organization

Structure

1.0 Introduction

1.1 Objectives

1.2 First Generation Computers

1.3 Second Generation Computers

1.4 Third Generation Computers

1.5 Fourth Generation Computers

1.6 Fifth Generation Computers

1.7 Central Processing Unit

1.8 Bus Organization

1.9 Summary

Review Questions

Unit 1: CPU Organization

1.0 Introduction

A **computer** is an electronic device that can perform a large range of computational tasks by manipulating data or information. Modern computers are able to **store**, **retrieve**, and **process** data according to the given set of instructions or **program**. The first electronic computers were developed during late 1940s. The early computers were able to perform mainly numerical computations. The modern computer systems are based on binary number system. The numbers in binary system are represented by just two types of digits: 0 and 1. The digits in binary system are called as bit and computers that use binary number system are known as digital computers. Digital computers represent the information in the form of a group of bits. The computer systems evolved over the years witnessing a significant progress in technology.

The digital computer systems have three main modules: **Central Processing Unit (CPU)**, **Memory**, and **Input/Output (I/O) systems**. Each module contains their internal components. Both memory and I/O systems are connected to CPU through some communication interfaces. All components work collectively to provide the desired results. The CPU acts like a brain of the computer, which carries out the most of operations and data manipulation. The memory provides storage for the data and instructions. The CPU can access the data and instruction from the memory. The I/O system consists of various input and output devices including keyboards, printers, mouse, and camera, etc. These devices are used provide data to and from the users. The **Computer Organization** is concerned with how the different hardware components or operational units in a computer system work and with the way they are linked together. The **Computer Architecture** is concerned with the instruction sets, representation of data types, techniques for memory addressing, methods for I/O operations, etc. The programmers are concerned with the computer architecture, while computer organization deals with low level designing.

The computer system executes a set of instructions to perform a particular task. The instructions are grouped in the form of a program. The program performs a variety of operations depending on the algorithm. It needs to process some data during the course of execution. The data could take many different forms from simple to complex structures. It is necessary to store the data in computer system for the processing. The data storage is a highly important requirement for the computer system so that data could be stored and subsequently retrieved when needed. The movement of data is also equally important. Data may originate outside the computer system. It goes into and moves out of the computer system through a set of I/O devices. The movement is done through some communication lines. The various events in the system should be controlled to manage and efficiently utilize the various resources.

1.1 Objectives

The major objectives of this unit are as follows.

1. To provide the knowledge on the evolution of computer systems.
2. To provide overview on the technological progress of the computers.
3. To understand the major modules of the computer system.
4. To discuss the basic organization and architecture of computer system.

1.2 First Generation Computers

World's first electronic computer named as **Electronic Numerical Integrator And Computer (ENIAC)** was developed at University of Pennsylvania by Professor John Mauchly and his student John Eckert in 1946. It was a huge machine occupying 1500 square feet space, weighing around 30 tons, and consuming 140 kilowatts power to operate. It was a decimal machine implemented using more than 18000 **vacuum tubes**. It used 10-digits decimal number and each digit was represented by a group of 10 vacuum tubes. It was capable of performing 5000 additions in a second. ENIAC had to be programmed manually by adjusting cables and switches, which was its major drawback.

The programs of ENIAC were not stored memory. Therefore, it was a tedious job to alter a program as it required to set switches on/off and plugging/unplugging cables.

Later, an idea of *stored program concept* was given by John von Neumann, who was also involved with ENIAC project as a consultant. He proposed an idea of a machine **Electronic Discrete Variable Computer (EDVAC)** based on stored program concept. John von Neumann developed a new stored program computer named **Institute for Advanced Studies (IAS) computer** in 1952, which can be considered as a prototype for the modern general purpose computers. The architecture of IAS computer also known as Von Neumann architecture is shown in Figure 1.1 It consists of a main memory that stores both data and instructions, an **Arithmetic and Logic Unit (ALU)** capable of performing arithmetic and logical operations, a **Control Unit (CC)** responsible for the interpretation of instruction and causing them to execute, and **Input Output (I/O)** equipment operating under the control unit. The contents of the memory are addressable by location. The instructions are executed in sequential way. IAS computer uses a binary system representing both data and instructions in binary form. Its memory consists of 1000 storage locations called words. Each word contains 40 binary digits. The control unit fetches the instruction from memory causing them to execute sequentially.

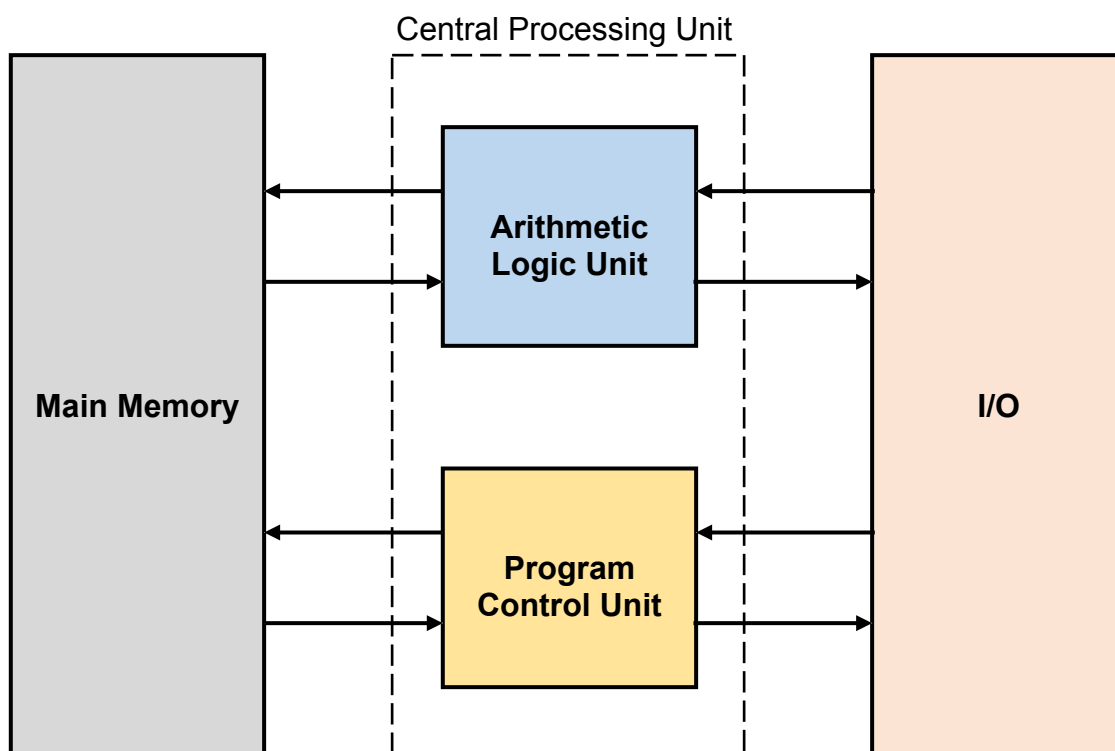


Figure 1.1: von Neumann structure of computer system

Eckert-Mauchly Computer Corporation developed the first commercially successful computer named **Universal Automatic Computer I (UNIVAC I)**. It was capable of performing matrix algebraic computations and solving statistical and logistical problems. Later **UNIVAC II**, came into the market with larger memory capacity and higher performance in 1950. The major success of the UNIVAC division its 1100 series of computers. However it was intended for scientific applications involving complex and long calculations. IBM launched its first electronic stored-program computer named 701 in 1953, which was also intended primarily for scientific applications. Later IBM introduced 702 and other 700/7000 series computers. Major highlights of the first generation computers are vacuum tubes, magnetic drum memories, and machine language programs. These computers were highly sequential able to solve only one problem at a time. The punch cards and paper tapes were used for input and output was obtained through printouts. These computers were in trend till 1956, when transistor based computers evolved in late 1950s.

1.3 Second Generation Computers

Development of **transistor** in 1947 at Bell Labs was great invention that revolutionized the computer world during late 1950s. The vacuum tubes were replaced by the transistors in electronic computers leading to the rise of second generation of computers. The transistor is a solid-state device, which is smaller in size, lower in cost, and generate lesser amount of heat as compared to vacuum tubes. The use of transistors was the major feature of the second generation computers, although the input and output operations were still performed with the help of punch cards/tapes and

print outs respectively. The use of multiplexor is another important highlight of the second generation. The multiplexor allows CPU and data channels to act independently. The computers became much smaller called as mini-computers with better processing speed, larger memory capacity, and more complex ALU and controls units than previous generation computers. The computers moved from binary/machine language to assembly language. Early versions of some high level languages also came into existence in that era. The memory also moved from magnetic drum to magnetic core technology. Digital Electronic Corporation (DEC) product **PDP-1** and **IBM 7094** are the examples of second generation computers.

1.4 Third Generation Computers

The era of third generation computers witnessed the emerging of microelectronics that significantly reduced the size of the computers and drastically improved the processing and efficiency of the computers. The development of **Integrated Circuit (IC)** was the revolutionary achievement of that era that changed the electronic industry broadly. IC allows to fabricate many components such as registers, transistors, logic gates, and memory cells onto a thin silicon wafer. The keyboards and monitors interfaced with an operating system replaced the punch cards and printouts. The cost of the computers reduced so much that these machine became accessible to mass audience first time. **IBM System/360** and DEC **PDP-8** are the examples of third generation computers.

1.5 Fourth Generation Computers

In year 1971, Intel developed a chip Intel 4004 that placed CPU, memory, and I/O controls all onto a single chip leading to the birth of microprocessor. The next major breakthrough was the development of **Intel 8008** in 1972. It was the first 8-bit microprocessor. Later in 1974 Intel introduced its first general purpose microprocessor Intel 8080. Like Intel 8008, Intel 8080 was also a 8-bit processor but it was faster having more rich instruction set, and larger addressing capabilities. By the end of 1970s, the 16-bit microprocessor also began to appear. One such processor was **Intel 8086**. The journey continued and a number of 16-bit to 64-bit processors were developed in subsequent years including **Pentium, Pentium Pro, Pentium II, Pentium III, Pentium**

IV, Core2Duo, and Core2Quad, etc. During that period the microprocessors came out of the realm of computers and microprocessors began to use in home appliances and other everyday products. Apple introduced the **Macintosh** operating system in 1984 that gained popularity with Apple devices. The increasing processing power of small computers lead to the computer networks that ultimately gave birth to **Internet**. The use of **Graphical User Interface (GUI)** based software, use of mouse, and development of laptops and other hand-held devices were the major advances of this era.

1.6 Fifth Generation Computers

The research and development in the field of microprocessors, memory, I/O controls and other related components of computing devices continues. In recent times Intel produced another revolutionary microprocessors such **i3, i5, i7, i9**, and **Xeon** series processors, etc. with more processing power. This era observed the production of laptops and hand-held devices at economical cost. The machines with high processing power, larger memories, and high storage capacity are now available at affordable prices. The parallel processing with **multi-core** processors and **Graphics Processing Units (GPU)** is thrust of modern computers. Computers have become household commodities. Now computers with artificial intelligence are under development. Quantum computing, Nano technology, and massive parallel processing are the essence of future generation computers.

Check your progress 1

1. Define a computer system.
2. Write the names of three major modules of a digital computer system.
3. Differentiate computer organization and computer architecture.
4. Write the name of the first electronic computer.
5. Write the name of the first commercially successful electronic computer.
6. Differentiate UNIVAC I and UNIVAC II.

7. What are the major features of the second generation computers?
8. What is transistor?
9. How is a multiplexor works?
10. What is the revolutionary achievement of the era of third generation computers?
11. Write two examples of third generation computers.
12. Which microprocessor is the first 8-bit microprocessor?
13. When did GUI based software appear?
14. Write the current generation of the computers.
15. What is the thrust of the fifth generation computers?

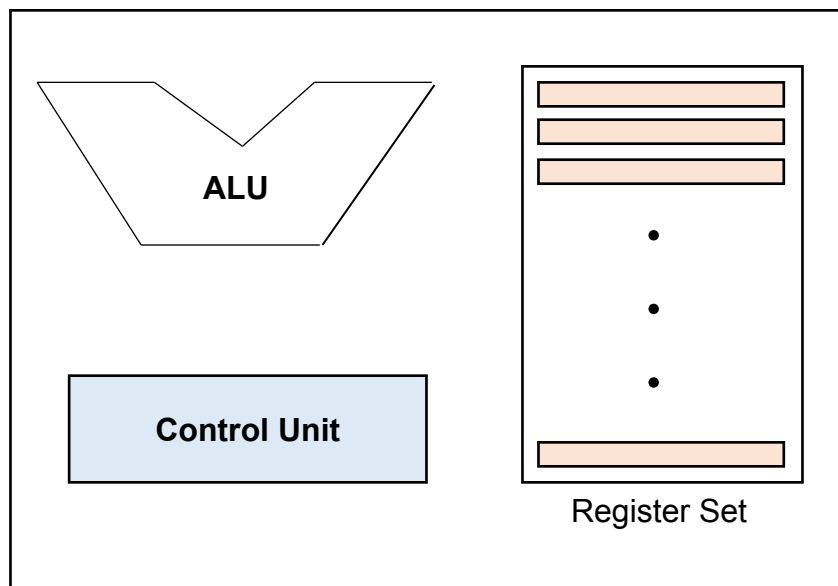


Figure 1.2: Major components of CPU

1.7 Central Processing Unit

Virtually all contemporary computer design are based on von Neumann architecture as shown in Figure 1.1. A computer consists of three main modules: CPU, memory, and I/O components. Both data and instructions/programs are stored in a single memory. It is a read-write memory whose data/instructions are addressable by their location. These

three modules are interconnected in some way to achieve the program execution. The instructions are executed sequentially unless explicitly specified otherwise. The behaviour of a computer system can be described by explaining the behaviour of each module. In this section, the functions of CPU are discussed. The program execution is actually done by CPU, which reads instructions specified in a program from the memory and executes them. A typical CPU consists of three major components: a set of registers, ALU, and **Control Unit (CU)** as shown in Figure 1.2.

The main functions of CPU are to fetch instructions from memory, decode instructions and execute, read-write data from/to memory, and data transfer from/to I/O devices. Registers are used to hold instructions, memory addresses, and other kind of data within the CPU. ALU is mainly concerned with the execution of arithmetic and logical operations. CU is responsible for providing directions to ALU, memory, and I/O systems to respond to the instructions. The CPU can be divided into two sections: data section and control section. Data section contains registers and ALU. Control section basically contain control unit only. Data section performs operations on data elements and control section issues signals to control the flow of data.

1.7.1 Register Set

The registers are extremely fast memories within CPU that are used to temporarily hold the data. The number and type of registers vary from computer to computer. During the execution of an instruction, the CPU needs to read and store intermediate results into memory. But the data transfer between CPU and memory is time consuming as memory is too slow compared to processor. Therefore, it is convenient to store intermediate results in processor registers during the operation. There are different types of registers that are used for different purposes.

General Purpose Registers are multipurpose registers that can store both data elements and memory addresses. Programmers can use general purpose registers in a variety of ways. There are four general purpose registers in Intel 8086 microprocessor: **AX**, **BX**, **CX**, and **DX**. The description of four general purpose registers is as follows.

AX: Known as accumulator register, it is preferred for the most of the operations.

BX: Known as base register, it is used to store memory address.

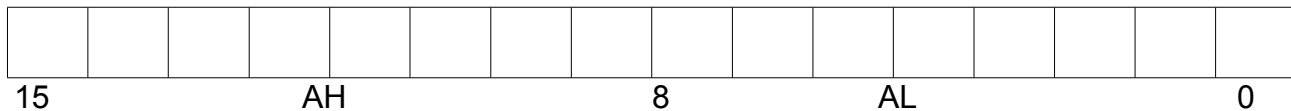
CX: Known as count register, it is used in looping.

DX: Known as data register, it is used for multiplication and division operations.

Each of these registers can be used as two 8-bit registers or a single 16-bit register. Eight high-end bits (high-end byte) are referred as AH and eight low-end bits (low-end byte) are referred as AL for register AX as shown in Figure 3. Other three registers also work in the same manner. Modern 32-bit Intel processors have eight general purpose registers while there are 16 such registers in 64-bit Intel processors.

Memory Access Registers are needed for memory read/write operations. These registers are used by CPU exclusively and are not directly available for the programmers. For each memory read/write operation, two registers are required: **Memory Data Register (MDR)** and **Memory Address Register (MAR)**. MDR is used to hold the data to store/fetch into/from memory and MAR keeps the address of the memory location.

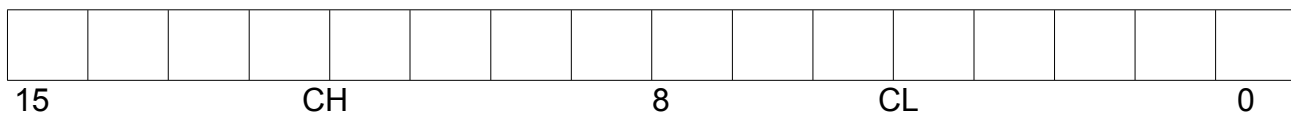
AX:



BX:



CX:



DX:

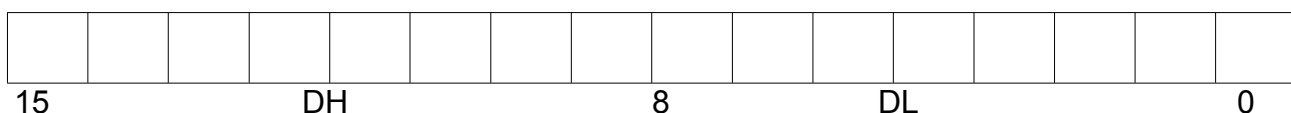


Figure 1.3: Use of single 16-bit register as two 8-bit registers

Instruction Fetching Registers are used in the instruction fetch operation. There are two main registers namely **Instruction Register (IR)** and **Program Counter (PC)** used to fetch the instructions from the memory. IR is used to store the fetched instruction and PC contains the address of memory location of the next instruction to fetch. PC is updated after each successful fetch operation. A special register known as **Program Status Word (PSW)** is used to maintain status information of the executing program. The bits of PSW are set by the CPU to indicate the current status. In addition some **Condition Registers** and **Control Bits/Flags** are also used to maintain the status of the program.

The flags are as follows.

Aux Carry: Set if there is a carry from bit 3 to bit 4

Carry: Set if the last unsigned arithmetic operation had a carry

Direction: Used for block transfer of data

Interrupt: Set to enable/disable interrupt

Overflow: Set if the last signed arithmetic operation overflowed

Parity: Used for a parity check

Sign: Set if the result of last arithmetic operation is negative

Trap: Set after each instruction if CPU should halt.

Zero: Set if the result of last arithmetic operation is zero

There are set of **Special Purpose Registers** that are used to load only specific information. **Index Register (IR)** contains offset from a segment register for the operand that need to fetch. The contents of IR are added to the segment register to obtain the memory address of the operand. There are four **Segment Pointers** known as **Code Segment (CS)**, **Data Segment (DS)**, **Stack Segment (SS)**, and **Extra Segment (ES)** that contain base locations for program instructions, data elements, or stack. CS is used to hold base location of program code, DS contains base location for variables, base location of the stack is loaded in SS, and ES is an additional register for the base location of the variables. **Stack** is a data structure in which last element stored is

retrieved first. A register known as **Stack Pointer** is used to indicate the top of the stack. When a data element is stored in the stack, the stack pointer is incremented/decremented depending on whether stack grows up/low in memory.

1.7.2 Arithmetic Logic Unit

Arithmetic Logic Unit (ALU) is an electronic component that is implemented using logic devices. The logic devices are capable of storing binary digits and perform Boolean operations. It performs the arithmetic, logical, and related operations. The major arithmetic operations are addition, subtraction, multiplication, and division. The major logical operations are AND, OR, and NOT, etc. ALU is connected to other components of the CPU that are supposed to provide the data to it, where desired operations are performed and results are sent back. The data elements are sent to ALU by writing values to its registers. All information is stored in binary form. The operations are performed using transistor switches and by connecting multiple transistors. The results are also provided by storing the data in registers. ALU may also provide the results by setting the flags.

1.7.3 Control Unit

The actions of CPU are controlled by the Control Unit (CU), which issues micro-orders to CPU and I/O systems. The micro-orders are sent in the form of control signals over dedicated control lines. The flow of data between CPU and other units is controlled by the CU. There are two different types of CU: hardwired and micro-programmed. In a hardwired CU, the control signals are generated by fixed logic circuits. On the other hand in micro-programmed CU, the controls signals are stored in a special memory called **control memory** in the form of microinstructions. The control memory is not accessible to the users or programmer. A sequence of microinstructions is called micro-program that is stored in control memory. The hardwired CU is faster and economical than micro-programmed CU. But micro-programmed CU can easily adapt to the changes in the system.

1.8 Memory

Memory is another important component of the digital computer system. It is required to store program and related data. The data and program instructions are transferred between memory and processor during the operation. The memory is an expensive component and often it is not large enough to accommodate the entire program and data. But not all the information is required by the processor at the same time. Therefore, a small memory can support a larger program with the help of a low-cost backup storage. The programs and data are kept in backup storage. Only the currently required programs or data are brought in memory from the backup storage as per need. The low-cost storage are larger in capacity but slower in speed. It is important to make a balance among speed, size, and cost of the memory. Therefore, a combination of different types of memory devices are used in the computer system. Different memories vary in their characteristics and several attributes. A relatively fast storage unit is known as the primary or main memory of the system. The main memory is volatile in nature that loses the entire data when there is no power. It cannot store the data permanently. While the low-cost backup storage is known as the secondary memory. The secondary memory is slow in speed but it can store the data permanently even in the absence of the power. The technology used to access the data from the storage is also different in different memories that will be discussed later in subsequent units.

1.9 I/O System

I/O devices in a computer system are required to perform fundamental input output operations. The I/O devices are a medium of communication between computer system and outside world. The I/O devices are also known as peripheral devices. The most common devices are keyboard, mouse, printer, and display units, etc. The peripheral devices communicate with CPU through communication lines. The peripheral devices are too slow as compared to the CPU. Therefore, for the proper functioning of the system a synchronous mechanism is required.

An interface unit is associated with each peripheral device that receives the command and control signals from I/O bus. The I/O bus connects the processor to interfaces of all the peripheral devices. The processor issues the commands to the peripheral devices that are decoded and interpreted by the interface. There are different

types of commands such as control, status, data input, and data output commands that can be issued to a device. A control command is used to activate the peripheral device and inform it to perform the specific operation. The status command is issued to test the device status condition. A device could be in various status conditions such as ready, busy, and error, etc. In case of data input command, the data is transferred from the device to the I/O bus. While, data output command causes the data transfer from I/O bus to the device. The peripherals not only communicate with processor but they also communicate with the main memory. A separate bus is used between memory and peripherals.

Data transfer between peripherals and CPU could be performed in different modes including Programmed I/O, Interrupt-initiated I/O, and Direct Memory Access (DMA). In programmed I/O, the data transfer is done with the help of a program and CPU takes active part in the operation. It results in loss of useful CPU cycles. In interrupt based approach, the CPU does not continuously involved in the data transfer, instead it initiates the process and goes back to its normal work. In DMA, the data transfer takes place between I/O devices and memory through memory bus.

1.10 Bus Organization

The data transfer and issuance of control signals between different components of the computer system are performed with the help of system buses. A system bus is a group of wires or printed circuits that permits bidirectional communication among the different components of computer. A system bus may have 50 to 100 communication lines. The system buses can broadly be divided into three categories: data buses, address buses, and control buses. Data buses carry data elements, address buses provides memory address to place the data, and control buses carry control signals between CU and other components of CPU. There are three different types of bus organizations used in computer systems: one-bus, two-bus, and three-bus organizations. The one-bus organization makes use of one bus for the incoming and outgoing data between registers. Single bus can carry only one operand or data element during a clock cycle. Therefore, for the operations that need two operands, two clock cycles are required to fetch the data. The one-bus organization is the most simple and the least expensive but

it slows down the performance of the system as it can transfer only one data element in a cycle. In the two-bus organization, the general purpose registers are connected to two buses. Whenever, there is a need of two operands, both operands can be transferred at a time. The two-bus organization is faster but expensive also than one-bus organization. The three-bus organization uses two buses to move data to the registers and one bus is used for transferring the output from the registers. By using more buses, more data can be transferred at a time, but it also increases the cost of the system.

Check your progress 2

1. List the main modules of a computer system.
2. What are the main functions of CPU?
3. Write the major components of CPU.
4. What is the role of ALU in CPU?
5. What is a register?
6. List the general purpose registers in Intel 8086.
7. What kind of information is stored in MDR and MAR?
8. What is program status word?
9. How many types of control unit are there? Write the names of different control units.
10. Differentiate primary memory and secondary memory.
11. What is the role of interface unit in I/O system?
12. List the different data transfer modes for I/O devices.

1.11 Summary

Modern computers evolved after the continuous technological advancements through different generations. Each generation of computers has a hallmark technology from vacuum tubes to multi-core processors that revolutionized the contemporary era. However, the most of computer systems are based on von Neumann architecture proposed by John von Neumann several decades ago. According to von Neumann architecture, the computer systems have three main modules: CPU, memory, and I/O

systems. The CPU further has three main components: ALU, CU, and a set of registers. The most of the arithmetic and logical operations are carried out by ALU and CU issues instructions in the form of control signal to different components that act accordingly. The registers are extremely fast memories that temporarily hold the small piece of data during the operations. All components work collectively to complete the operations. The communication with CPU or between CPU and other modules takes place through systems buses.

Memory is another important component of the digital computer system. It is required to store program and related data. The data and program instructions are transferred between memory and processor during the operation. The programs and data are kept in a backup storage. Only the currently required programs or data are brought in memory from the backup storage as per need. A relatively fast storage unit is known as the primary or main memory of the system. The main memory is volatile in nature that loses the entire data when there is no power. While the low-cost backup storage is known as the secondary memory. The secondary memory is slow in speed but it can store the data permanently even in the absence of the power.

I/O devices in a computer system are required to perform fundamental input output operations. The I/O devices are a medium of communication between computer system and outside world. The I/O devices are also known as peripheral devices. The peripheral devices are too slow as compared to the CPU. Therefore, for the proper functioning of the system a synchronous mechanism is required. An interface unit is associated with each peripheral device that receives the command and control signals from I/O bus. The I/O bus connects the processor to interfaces of all the peripheral devices. The processor issues the commands to the peripheral devices that are decoded and interpreted by the interface. The peripherals not only communicate with processor but they also communicate with the main memory. A separate bus is used between memory and peripherals. Data transfer between peripherals and CPU could be performed in different modes including Programmed I/O, Interrupt-initiated I/O, and DMA. The involvement of CPU in I/O operations is minimal in case of DMA.

The data transfer and issuance of control signals between different components of the computer system are performed with the help of system buses. A system bus is a

group of wires or printed circuits that permits bidirectional communication among the different components of computer. The system buses can broadly be divided into three categories: data buses, address buses, and control buses. Data buses carry data elements, address buses provides memory address to place the data, and control buses carry control signals between CU and other components of CPU. By using more buses, more data can be transferred at a time, but it also increases the cost of the system.

Review Questions

- Q1. Describe the development of computer systems through different generations with emphasis on hallmark technology for each generation.
- Q2. With the help of a suitable figure, explain the von Neumann architecture.
- Q3. Write main components of CPU with their major functions and describe CPU organization with the help of a figure.
- Q4. How many types of registers are there in register set of CPU? Explain utility of different registers.
- Q5. What is system bus? Write different bus organization with their important features.
- Q6. A computer system consists of different kind of storages. Explain the reason.
- Q7. Why is the I/O system required in a computers system? Explain how data transfer takes place between I/O devices and CPU.

Unit 2: Data Representation

Structure

2.0 Introduction

2.1 Objectives

2.2 Number System

2.3 Negative Number Representation

2.4 Range Extension

2.5 Fixed-Point Numbers

2.6 Floating-Point Numbers

2.7 IEEE Floating-Point Standard

2.8 Summary

Review Questions

Unit 2: Data Representation

2.0 Introduction

The arithmetic logic unit (ALU) is an important component of the central processing unit (CPU) that performs arithmetic and logic operations on data. The data elements are transferred from memory or I/O devices to ALU for the processing. Data representation refers to the form in which data can be stored, processed, and transmitted. The data presented to ALU are stored in registers. The registers are made of flip-flops, which are two-stage devices. Therefore, binary-coded representation of data is the most suitable for digital computers. There are three major types of data elements used in computer systems: numbers, alphabets, and special characters. All these data elements can be represented in binary-coded form.

A number system with base r is a system that uses distinct symbols to represent r distinct digits. The base is also known as radix. Other numbers are represented in terms of those r digits. For example, the decimal number system having base 10, uses 10 distinct symbols for digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. All other numbers are the combinations of these 10 digits. Similarly binary number system uses a base 2, octal number system has a base 8, and base or radix for hexadecimal number system is 16. An unsigned integer number N can be represented in a base r number system using n digits as $(a_{n-1}a_{n-2}\dots a_1a_0)_r$, $0 \leq a_i \leq r - 1$. The binary number system is highly useful for

digital computers. Any data numeric or alphanumeric is internally represented in binary form in a computer system. The software or programmers need to interpret the binary pattern appropriately. The interpretation of binary pattern is called data representation. Other number systems such as decimal number system are also important as users are more convenient with this to perform different types of operations. Sometimes it is necessary to convert a number from one representation to other representation. The numbers in one system can be converted to other number systems with help of some algorithms or methods.

2.1 Objectives

The major objectives of this unit are:

1. To acquire basic knowledge of number systems.
2. To learn conversion of numbers in one system to the numbers in another system.
3. To learn different representations of data.

2.2 Number Systems

The binary number system is the most suitable system for digital computers. There are only two symbols 0 and 1 in binary number system. Each number is represented as string of 0 and 1. There are other number systems such as octal and hexadecimal. In octal number system, eight different digits 0, 1, 2, 3, 4, 5, 6, 7 are used to represent the numbers. On the other hand, in hexadecimal representation sixteen different digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The numbers in different number systems can be converted to other number systems using **Radix Conversion** method. To convert a number representation in base (or radix) b_1 to another number system with base b_2 , the number is successively divided by b_2 and remainders are noted down until quotient becomes zero. Similarly, a fractional part of the number can be converted to other representation by successively multiplying the fractional part by b_2 and noting down the resulting integers. However, in case of fractional part conversion, the conversion may not terminate after the finite number of multiplications.

2.2.1 Decimal to Binary Conversion and Vice Versa

Let us consider the conversion of decimal numbers to binary numbers for example $(67)_{10}$ is converted to binary representation as follows.

Division by 2	Quotient	Remainder
$67 \div 2$	33	1
$33 \div 2$	16	1
$16 \div 2$	8	0
$8 \div 2$	4	0
$4 \div 2$	2	0
$2 \div 2$	1	0
$1 \div 2$	0	1

The resultant binary string is obtained by writing remainders bottom to up (in reverse order). The equivalent binary string for 67 is therefore would be $(1000011)_2$. The binary string $(1000011)_2$ can be converted to decimal number by writing binary numbers as power of 2 as follows

$$\begin{aligned} & 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ = & 64 + 0 + 0 + 0 + 0 + 2 + 1 \\ = & 67 \end{aligned}$$

The decimal fraction is converted to binary in a different way. The decimal fraction is multiplied by the base (or radix) of the intended number system and digits so obtained are accumulated. The process is repeated until fraction becomes zero. For example $(0.8125)_{10}$ is converted to binary number $(0.1101)_2$ by multiplying the decimal fraction by 2 as follows.

Multiplication	Result	Integer	Fraction
0.8125×2	1.625	1	0.625
0.625×2	1.25	1	0.25
0.25×2	0.50	0	0.50

0.50×2	1.00	1	0
-----------------	------	---	---

Write all the integer parts from top to bottom and obtain the binary number $(0.1101)_2$.

The fractional binary string $(0.111)_2$ is converted to decimal fraction as follows

$$\begin{aligned}
 & 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\
 = & 0.5 + 0.25 + 0.125 \\
 = & 0.875
 \end{aligned}$$

The power terms of 2 in fractional part always have negative powers.

2.2.2 Decimal to Octal Conversion and Vice Versa

The octal number system is a base 8 system that uses eight digits 0 to 7 to represent any number. Let us consider the example of decimal representation to octal representation conversion. The conversion is performed by successive division operation. Here, the divisor would be 8 as it is base or radix for octal number system. If $(348)_{10}$ is a decimal number, it is converted to octal representation as follows.

Division by 8	Quotient	Remainder
$348 \div 8$	43	4
$43 \div 8$	5	3
$5 \div 8$	0	5

The octal number is obtained by writing all remainders bottom to up leading to $(534)_8$.

The octal representation to decimal number conversion is done with power terms of 8 as follows

$$\begin{aligned}
 & 5 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 \\
 = & 5 \times 64 + 24 + 4 \\
 = & 320 + 24 + 4 \\
 = & (348)_{10}
 \end{aligned}$$

Similarly, we can work with fractional parts as discussed earlier for binary numbers. Let us convert $(0.45)_{10}$ to octal representation by multiplying it by 8 and noting the integer part of the resultant number.

Multiplication	Result	Integer	Fraction
0.45×8	3.6	3	0.6
0.6×8	4.8	4	0.8
0.8×8	6.4	6	0.4
0.4×8	3.2	3	0.2
0.2×8	1.6	1	0.6

It can be observed from the above table that it is not possible to terminate the conversion process for $(0.45)_{10}$ to octal representation in finite number of iteration. Therefore, it is stopped after desired precision. From above table, the resultant octal number for $(0.45)_{10}$ is obtained as $(0.34631)_8$ by noting down all integer parts from top to bottom. The conversion of fractional octal number to decimal number is done using negative power terms. For example, let us convert $(534.28)_8$ to decimal representation.

$$\begin{aligned}
 & 5 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 + 2 \times 8^{-1} + 8 \times 8^{-2} \\
 & = 5 \times 64 + 24 + 4 + 0.25 + 0.125 \\
 & = (348.375)_{10}
 \end{aligned}$$

2.2.3 Decimal to Hexadecimal Conversion and Vice Versa

In hexadecimal number system is a base 16 system. There are sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The hexadecimal digit 'A' is equivalent to decimal 10 and digit 'F' is equivalent to decimal 15. The base 10 numbers or decimal numbers can be converted to equivalent hexadecimal number by successively dividing it by 16. Let us consider the example by converting $(56356)_{10}$ to hexadecimal representation. The calculations are given in the following table. After successive division operations, the remainders are noted down from bottom to up. The resultant hexadecimal number is obtained as $(DC24)_{16}$.

Division by 16	Quotient	Remainder
56356÷16	3522	4
3522÷16	220	2
220÷16	13	12=C
13÷16	0	13=D

The reverse conversion i.e. hexadecimal number to decimal can be performed by using power terms of 16 in a similar manner as it is done in other number system. For example, $(DC24)_{16}$ can be converted to decimal number as follows.

$$\begin{aligned}
& D \times 16^3 + C \times 16^2 + 2 \times 16^1 + 4 \times 16^0 \\
&= 13 \times 16^3 + 12 \times 16^2 + 2 \times 16^1 + 4 \times 16^0 \\
&= 13 \times 4096 + 12 \times 256 + 2 \times 16 + 4 \times 1 \\
&= 53248 + 3072 + 32 + 4 \\
&= 56356
\end{aligned}$$

We can work with fractional numbers in a similar way as we do in other number systems. The fractional decimal number can be converted to hexadecimal by successively multiplying by 16 and noting the integers in top-down manner. Let us convert $(0.06640625)_{10}$ to hexadecimal. The resultant hexadecimal number is $(0.011)_{16}$.

Multiplication	Result	Integer	Fraction
0.06640625×16	1.0625	1	0.0625
0.0625×16	1.0	1	0
0×16	0	0	0

2.2.4 BCD Representations

Computer users are more convenient with decimal number system as it is commonly used in real life. The solution of this problem is that users are allowed to work with decimal number system and the decimal numbers are converted to binary numbers while processing them with computers. After completing the processing, the results are

converted back to decimal form. All the data is processed or stored with computers in binary coded form. The decimal numbers are stored in registers using binary codes. The conversion of a decimal number to binary coded form is different from converting it to binary number. In binary coded form, each digit of the decimal number is independently converted to binary representation. Each decimal digit is represented with four binary digits. Such a representation is called binary coded decimal (BCD). For example, the number $(542)_{10}$ is represented in BCD form as follows

Decimal number	BCD
542	0101 0100 0010

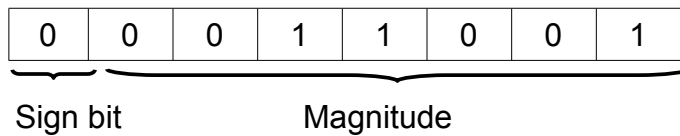
The alphanumeric character set used in computers contains alphabet, digits, and some special characters. All the elements of alphanumeric character set are represented by binary codes as registers can hold only binary information. American Standard Code for Information Interchange (ASCII) provides a standard seven bits binary codes for alphanumeric character set.

Check your progress 1

1. What is radix or base in number systems?
2. The value of radix in octal number system is:
3. Convert $(10101)_2$ to equivalent decimal number.
4. Convert $(A2)_{16}$ to binary.
5. Convert $(A2)_{16}$ to decimal.
6. Convert $(128)_8$ to decimal.
7. Convert $(321)_{10}$ to octal.
8. Convert $(45862)_{10}$ to hexadecimal.
9. What is BCD representation?
10. How the alphanumeric character set is represented in computer system?
11. Convert $(0.00390625)_{10}$ to hexadecimal.
12. Convert $(0.75)_{10}$ to binary.

2.3 Negative Number Representation

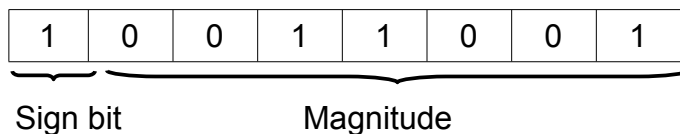
In conventional mathematics, the signed numbers are represented with the help of plus (+) and minus (-) symbols. However, in digital computers everything including sign of a number must be represented in binary form. All positive numbers and zero can be considered as unsigned numbers but some mechanisms are required for negative number representation. In a simple approach, the sign of a number is represented by a single bit known as **sign bit**. If the value of the sign bit is 0, the number is a positive number or a negative number otherwise. The positive numbers are represented in signed-magnitude form i.e. the most significant bit of the binary string is the sign bit and remaining bits provide the magnitude of the number. For example, +25 is stored in a 8-bit register as follows



However, a negative integer can be represented in three different ways:

1. Signed-magnitude representation
2. Signed-1's complement representation
3. Signed-2's complement representation

The signed-magnitude represented is same as discussed earlier with sign bit set to 1. For example the number -25 is stored as follows



The 1's complement of a negative number is obtained from the signed-magnitude representation of the positive number by complementing its every bit including the sign bit. Therefore, 1's complement of -25 is obtained from +25 as follows

Signed-magnitude representation of +25

0	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---

Sign bit

Magnitude

Complement each bit to get 1's complement of -25

1	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---

Sign bit

Magnitude

The 2's complement of a number is obtained by adding 1 to the 1's complement representation of the number. It is a two-step procedure given as follows.

1. Invert each bit of the binary string including the sign bit.
2. Treat the result of step 1 as an unsigned integer and add 1.

Usually 2's complement is preferred for arithmetic operations. The 2's complement of -25 is obtained from its 1's complement as follows.

1. First, obtain 1's complement of -25.

1	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---

Sign bit

Magnitude

2. Add 1 to 1's complement of -25 to get its 2's complement

1	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

Sign bit

Magnitude

2.4 Range Extension

Usually, the same amount of memory is used to store a particular type of data elements irrespective of their values. Often n bits are used to store the number even though it may need on m bits ($n > m$). In such case, the bit length is increased. The expansion of bit length is referred to as **range extension**. For example, signed magnitude notation of

+25 is 11001 that actually involves only 5 bits. But if we have to store it in an 8-bit register then we have to increase the length to 8 bits. In signed magnitude notation, the range extension is simple. The additional bit positions are filled with 0's and the sign bit is shifted to the new leftmost position. Let us consider the following example.

Number	Signed Magnitude Representation	Range Extension (8-bit)	Resultant Number
+25	011001	00011001	+25
-25	111001	10011001	-25

Number	Signed Magnitude Representation	Range Extension (16-bit)	Resultant Number
+25	011001	000000000011001	+25
-25	111001	100000000011001	-25

The signed magnitude representation of +25 or -25 requires 6 bits including sign bit. In order to store it in an 8-bit register, two additional 0's are inserted and sign bit is shifted to the new leftmost position. Similarly for 16-bit memory, 10 additional 0's are inserted and sign bit is shifted accordingly. However, this approach does not work with 2's complement numbers. For such numbers, sign extension is used. In sign extension, the additional bits are filled with sign bit. In other words, additional 0's are inserted for positive numbers, while additional 1's are inserted for negative numbers as follows.

Number	2's Complement Representation	Sign Extension (16-bit)	Resultant Number
+25	011001	000000000011001	+25
-25	11100111	1111111111100111	-25

2.5 Fixed-Point Numbers

A number may be integer, fraction, or mixed integer-fraction. Therefore, in addition to the sign of the number, some mechanism is required to separate the integer and fraction parts of a number. A **base point** or **radix point** is the mechanism used to identify integer and fraction parts of the number. The position of the radix point is used to decide whether a number is an integer, fraction, or mixed integer-fraction. For binary numbers, the **binary point** is specified by a particular location in the register. If the location of the binary point is fixed i.e. always a specific location to keep the binary point, the representation is known as **fixed-point** numbers. The binary point is a kind of divider between integer and fraction parts. The fixed-point numbers can be given by a tuple $\langle w, b \rangle$, where w is the width of the number i.e. the total number bits used for the number and b is the position of binary point from the least significant bit. The notation for a n -bit number is illustrated in Figure 2.1. As shown in the figure, the entire number length is divided into two parts: Signed integer and Fraction. Each part has a fixed length. In fact, a number in fixed-point data type is stored as an integer that is scaled by a specific factor. The scaling factor for a number is an implicit value determined by its type. For example, a given number 2.34 can be stored as 2340 with scaling factor $1/1000$ or a number 2340000 can be also be represented as 2340 with scaling factor 1000.

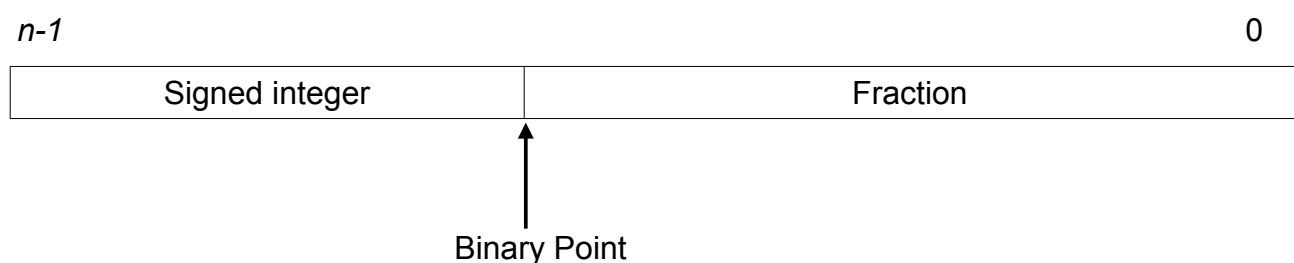


Figure 2.1: Fixed-point number representation

The fixed-point numbers are closely related to integer numbers. In fact, integer numbers can be consider a special case of fixed-point numbers, where binary point is at position 0. Therefore, all mathematical operations that a computer can perform with

integers can be applied on fixed-point numbers without any additional hardware. The fixed-point representation of number is therefore simple and efficient. But the system has to compromise with the range of numbers as fixed-point representation can represent only a relatively limited range of numbers.

Check your progress 2

1. What is the value of sign bit for negative numbers?
2. What is the value of sign bit for positive numbers?
3. Determine 1's complement of -45.
4. Determine 1's complement of +45.
5. Obtain 2's complement of -45.
6. What is radix point?
7. How scale factor is determined for a number?
8. What is fixed-point number?
9. What would be radix point for integers?
10. What is range extension?
11. What is sign extension?
12. Perform range extension for +18 and -18.
13. Perform sign extension for +18 and -18.
14. What is the major limitation of fixed-point numbers?
15. How integer numbers are special case of fixed-point numbers?

2.6 Floating-Point Numbers

The fixed-point numbers can be processed at a faster rate with the available capabilities of the system. But at the same time, only a small range of numbers can be provided with this representation. To overcome the limitations of the fixed-point numbers, there is another approach of representing signed numbers known as floating-point. It represents a number in two parts: mantissa (m) and exponent (e). The mantissa is a fixed-point number and exponent specifies the location of base/radix point. It is similar to scientific

notation, where a number with base b is written as $m \times b^e$. For example, a decimal number 589.621 is written as 0.589621×10^3 in scientific notation. It can be written in floating-point representation as follows

Mantissa	Exponent
+0.589621	+3

Both mantissa and exponent are stored in registers separately. A binary number is written in the same way with base 2. Both mantissa and exponent use 2's complement format. For example, a binary number $(+1011.01)_2$ is written as

Mantissa	Exponent
00000101	000011

where mantissa and exponent are 8-bit and 6-bit components respectively. If the most significant bit of the mantissa is nonzero then the number is called to be normalized.

2.7 IEEE Floating-Point Standard

A working committee of IEEE has developed three standards of floating-point representations known as: **single precision**, **double precision**, and **extended precision**. The single precision numbers require a 32-bit word whose representation is shown in Figure 2.2.



Figure 2.2: Single precision floating-point representation

In the Figure 2.2, S represents the sign bit, exp is the exponent, and $frac$ represents the mantissa of the number. The mantissa part has 23 bits, exponent takes 8 bits, and remaining 1 bit is occupied by sign part. The sign bit is 0 for a positive number and 1 if negative. The number is normalized so that the binary point is at right of the

leading 1 i.e. the mantissa is always 1.xxxxxx...xxx in the normalized form. Because leading bit is always 1, there is no need to store it. It is an implied bit. Therefore, effectively the mantissa part has 24 bits. For zero, a special representation is used with all 0's both for mantissa and exponent fields as well as sign bit. In IEEE standard floating-point numbers, the exponent part does not use sign-magnitude, 1's complement, or 2's complement format. It uses a **biased representation**, which is simply the binary representation of $E+127$, where E is the actual exponent and the number 127 is called the exponent bias. The actual exponent is obtained by subtracting the exponent bias from the stored exponent. Mathematically, the above representation gives

$$- 1^S . 2^{\text{exp}-127} . 1.\text{frac}$$

The dynamic range of single precision floating-point numbers goes from 1.175×10^{-38} to 3.4×10^{38} .

Although single precision floating-point numbers are adequate for the most of the applications, there is another commonly used alternative is double precision floating-point numbers. The double precision numbers use 64-bit words. However, arrangement is similar to the single precision floating point numbers. Only the width of the fields and exponent bias are different. The arrangement of different fields in the double precision floating-point representation is shown in Figure 2.3. It uses two 32-bit registers designated as odd and even registers to make it 64-bit numbers.

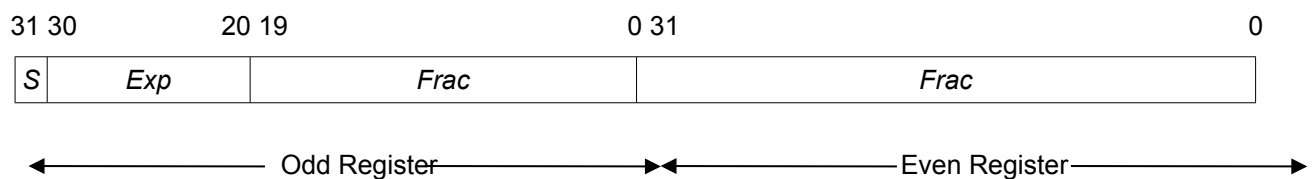


Figure 2.3: Double precision floating-point representation

Mathematically, the above representation gives

$$- 1^S . 2^{\text{exp}-1023} . 1.\text{frac}$$

where S is the sign bit, exp is 11-bit exponent, and $frac$ is 52-bit mantissa. The mantissa takes 32 bits from even register and 20 bits from odd register. The dynamic range of double precision numbers varies from 2.2×10^{-308} to 1.7×10^{308} .

The third floating-point representation known as extended precision floating-point numbers is not very different from the previous two representation but requires 80-bit word. Like other two notations, it uses 1 bit as sign bit, 15 bits as exponent, and 64 bits for mantissa. Unlike single and double precision, the leading bit in extended precision is not generally hidden. As compared to fixed-point numbers, the arithmetic operations with the floating-point numbers are complex to perform and require complex hardware and longer execution time. However, the most of computing devices use floating-point arithmetic as flexible base point is required in scientific operations.

Check your progress 3

1. Is mantissa a fixed-point number?
2. What is exponent in floating-point numbers?
3. Which number representation is used for mantissa and exponent?
4. How do you identify a normalized number?
5. List the standards defined by IEEE for floating-point numbers.
6. How many bits are required for single precision floating-point numbers?
7. What is biased representation?
8. How many bits are required for extended floating-point numbers?

2.8 Summary

Data representation refers to the form in which data can be stored, processed, and transmitted. There are different number systems that are used to represent numbers in different forms. For digital computers, the binary number system is the most useful. But users are more convenient with decimal number system. Therefore, numbers are

needed to convert from one system to another. The numbers in different number systems can be converted to other number systems using Radix Conversion method. All the data is processed or stored with computers in binary coded form. The decimal numbers are stored in registers using binary codes. The conversion of a decimal number to binary coded form is different from converting it to binary number. In binary coded form, each digit of the decimal number is independently converted to binary representation. Each decimal digit is represented with four binary digits. Such a representation is called binary coded decimal (BCD). American Standard Code for Information Interchange (ASCII) provides a standard seven bits binary codes for alphanumeric character set.

In conventional mathematics, the signed numbers are represented with the help of plus (+) and minus (-) symbols. However, in digital computers everything including sign of a number must be represented in binary form. All positive numbers and zero can be considered as unsigned numbers but some mechanisms are required for negative number representation. In a simple approach, the sign of a number is represented by a single bit known as sign bit. If the value of the sign bit is 0, the number is a positive number or a negative number otherwise. However, a negative integer can be represented in three different ways: Signed-magnitude representation, Signed-1's complement representation, Signed-2's complement representation.

The computers store the numbers in registers. All the information related to numbers such as sign, base point, and magnitude, etc. is stored in binary form. Often n bits are used to store the number even though it may need on m bits ($n > m$). In such case, the bit length is increased. The expansion of bit length is referred to as range extension. There are two prominent notations for numbers: fixed-point and floating-point numbers. The floating-point arithmetic is preferred in digital computers despite the higher complexity due to flexible base point. A number may be integer, fraction, or mixed integer-fraction. Therefore, in addition to the sign of the number, some mechanism is required to separate the integer and fraction parts of a number. A base point or radix point is the mechanism used to identify integer and fraction parts of the number. The position of the radix point is used to decide whether a number is an integer, fraction, or mixed integer-fraction. If the location of the base point is fixed, the

representation is known as fixed-point numbers. The fixed-point numbers can be processed at a faster rate with the available capabilities of the system. But at the same time, only a small range of numbers can be provided with this representation. To overcome the limitations of the fixed-point numbers, there is another approach of representing signed numbers known as floating-point. A working committee of IEEE has developed three standards of floating-point representations known as: single precision, double precision, and extended precision.

Review Questions

- Q.1 Convert following numbers to binary: $(295)_{10}$, $(5482)_{10}$, $(763)_{10}$, and $(4169)_{10}$.
- Q.2 Convert following numbers to decimal: $(1100)_2$, $(101101)_2$, $(10010)_2$, and $(101110)_2$.
- Q.3 Convert following numbers to hexadecimal: $(78956)_{10}$, $(89645879)_{10}$, $(84845684)_{10}$, $(24242424)_{10}$, and $(1816181615)_{10}$.
- Q.4 Convert following numbers to octal: $(1100)_2$, $(101101)_2$, $(10010)_2$, and $(101110)_2$.
- Q.5 Write the following numbers in BCD form: $(745)_{10}$, $(8632)_{10}$, and $(9100)_{10}$.
- Q.6 Write following numbers in 1's complement: +47, -95, +63, -81.
- Q.7 Write following number in 2's complement: +47, -95, +63, -81.
- Q.8 Convert following numbers to binary: $(5462)_8$, $(2534)_8$, $(163)_8$, and $(15)_8$.
- Q.9 With the help of suitable diagrams explain the single-precision and double-precision floating-point number representation. Differentiate both representations.
- Q.10 Why range extension is required and how it is performed?
- Q.11 Explain different representations for negative numbers with suitable examples.

Unit 3: Instruction Sets

Structure

3.0 Introduction

3.1 Objectives

3.2 Instruction Formats

3.3 Instruction Types

3.4 Addressing Modes

3.5 Programming Considerations

3.6 Summary

Review Questions

Unit 3: Instruction Sets

3.0 Introduction

The digital computers perform operations as directed with the set of instructions. The instructions or machine instructions or computer instructions are the commands that control the operations of a computer. A computer instruction is a binary code that instructs the computer to execute specific operations. The instructions and data are stored together in the computer memory. The CPU reads the instruction from the memory, interprets it, and performs the desired operation on the relevant data fetched from the memory. A set of instructions is used to control the sequence of operations. Each processor has its own collection of instructions that it can recognize and execute. Such a set is called processor's instruction set.

An instruction is a group of bits that is divided into multiple fields. Each field provides specific information such as operation type, operands, and references to the next instructions, etc. Different fields of the instructions are loaded in registers. Each instruction must contain at least two types of information, the operation to perform and data elements required during the operation. The operation to be performed is provided in a field known as **op-code** and addresses of operands are given in **address** field. The op-code represents operations such as add, subtract, multiply, divide, read, and write, etc. There may be two types of addresses, **source** and **destination**. The source address field operands, while destination address field is used to write the result of the operation. Sometimes, the field is used as source and destination. The instructions can be divide into different categories depending on the number of operands involved. As machine instruction are in binary format, it is difficult for programmers to work with machine instructions. Therefore, it is a general practice to use symbolic representation for the machine instructions. Although it is possible to write computer programs directly in machine language, but the modern programmers rarely use the machine instructions to write the programs. The most of the programmers work with high level languages.

3.1 Objectives

After the completion of this unit, the students will be able

1. To understand the instructions of computer systems.
2. To understand instruction formats.
3. To understand use of addresses and registers in instructions.
4. To learn different types of instructions.

3.2 Instruction Formats

In order to perform a given task, the digital computers are provided a sequence of instructions. Each instruction specifies a particular operation to perform on one or more data elements. A computer instruction is a binary string with codes for both operation and data. For each microprocessor, there is a set of instructions. The most of the instruction sets contain instructions in multiple formats. The simple form of instruction has two fields, op-code and address as shown in Figure 3.1. The figure shows a 16-bit instruction formats. The four most significant bits represent op-code and other bits provide address for operands. The instruction set for such a processor can have at most 16 instructions as 4 bits are used for op-code field. When instructions are loaded from the memory, the op-code and operands are stored in registers. The **accumulator (AC)** is the most rigorously used register. There are some operations that do not need any operands from memory. Clear the AC, complement the AC, and increment the AC, etc. are some examples of such operations. The instructions for such operations do not need address field.

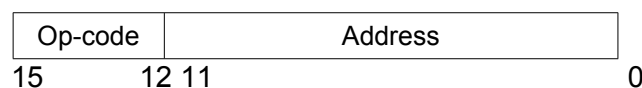


Figure 3.1: Instruction format having two fields

It is a general practice to use a symbolic representation for machine instructions. The op-codes are given by abbreviations known as **mnemonics**. Some examples of mnemonics are given here.

LOAD	Load data from memory
ADD	Add
SUB	Subtract
MUL	Multiply
DIV	Division
STOR	Store data to memory
MPY	Multiply

Symbols are also used to represent the operands in an instruction. The symbols such as R, A, X, and Y, etc. may represent the contents of registers, memory location, or data elements. There are different types of arithmetic operations involving different number of operands. The **unary** operations require only one operand, while **binary** operations need two operands. These operands are called as **source operands**. In addition to operands, one location (register or memory) is required to place the result of the operation. It is called as **destination operand**. There are some other operations mostly non-mathematical that can be performed without any operand. Therefore, depending on the number of source operands and destination operands, the number of addresses used in an instruction may vary from one to three. Based on the number of addresses, the instructions can be classified as **zero-address**, **one-address**, **two-address**, and **three-address** instructions. Some examples of instructions written with symbolic representation are given in the Table 3.1.

The stack data structure loads and retrieves data elements in last-in-first-out fashion. The location of the stack is usually known or at least top two elements are stored in predefined registers. The *pop* operation retrieves top element from the stack.

Therefore, the pop operation do not need any operand as location of the top of stack is available. As discussed earlier, clear AC and increment AC are other operations for which no operands is required. Thus, the instructions for such operations are zero-address instructions.

Table 3.1: Some Examples of Instructions

Instruction	Type	Comment
LOAD D	One-address	Load AC from location D
STOR Y	One-address	Load content of AC to location Y
SUB B	One-address	Subtract B from AC and store result to AC
ADD B	One-address	Add B and AC and store result to AC
MOVE B, A	Two-address	Load content of A to location B
SUB Y, B	Two-address	Subtract B from Y and store result to Y
ADD C, B	Two-address	Add B and C and store result to C
MPY Y, B	Two-address	Multiply B with Y and store result to Y
SUB Y, A, B	Three-address	Subtract B from A and store result to Y
ADD Y, C, B	Three-address	Add B and C and store result to Y
MPY Y, C, B	Three-address	Multiply B with C and store result to Y
DIV Y, C, B	Three-address	Divide C by B and store result to Y

There are different types of operands such as number, character, logical data, and address. The second field in the instruction may not contain addresses but actual operands. If second field provides the operands, it is an instruction with **immediate operands** and it is called an **immediate instruction**. If the second field contains address, there are further two possibilities. The address either specifies the location of the operand itself or it provides the location that contains the address of the operand. If address field provides the memory location of the operand itself, it is known as **direct address** otherwise in case it specifies another address then it is called **indirect address**. To indicate whether it is a direct or indirect address, the most significant bit of the instruction designated as I is reserved as shown in Figure 3.2. The value of bit I is 0 for direct address and 1 for indirect address.

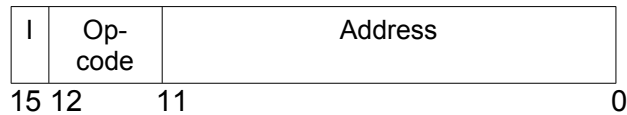


Figure 3.2: Instruction format having three fields

Check your progress 1

1. What are different fields in instruction format?
2. What is the use of opcode field of instruction format?
3. Classify the types of instructions based on number of addresses.
4. What is immediate instruction?
5. What is the difference between direct address instruction and indirect address instruction?

3.3 Instruction Types

The basic instruction set of a computer contains three types of instructions: **register-reference** instruction, **input-output** instruction and **memory-reference** instruction. Register-reference instructions are recognized by I=0 and op-code 111. For input-output instructions I=1 and op-code 111 are used. For memory-reference instructions I may be 0 or 1 and op-code varies 000 through 110. There are a number of instructions in the instruction set that can be further divided into different categories depending on the kind of operation they perform. There are some basic operations for which there should be instructions in instruction set. If all such instructions are there in the instruction set, the programmers may write programs to evaluate any function. Such an instruction set is said to be complete. On this basis, the instructions can be divided into following categories.

3.3.1 Data Transfer Instructions

The data transfer are the most fundamental instructions of any instruction set in a computer system. The majority of operations need data elements to transfer from

memory to register, register to memory, or register to register. A data transfer instruction must have a source location, a destination location, length of the data to transfer, and addressing mode. There could be different variants of the data transfer instructions depending on whether operand is a register or memory. The data length is usually 8, 16, 32, or 64 bits. The length of data to be transferred also lead to different variants of instructions. If source and destination locations both registers, then data transfer operations are simple as registers are internal to the processor. However, if operands are memory locations then data transfer operations are comparatively complex. Some examples of data transfer instructions are given in Table 3.2.

Table 3.2: Examples of logical instructions

Instruction	Operation	Description
LD	Load	Data transfer from memory to floating point register
LDR	Load	Data transfer from floating point register to floating point register
ST	Store	Data transfer from register to memory
STD	Store	Data transfer from floating point register to memory

3.3.2 Arithmetic Instructions

The instructions for basic arithmetic operations such as addition, subtraction, multiplication, and division are included in the instruction set of the most of the computers. These instructions are available for both fixed point and floating point numbers. All the basic arithmetic operations need two operands. Therefore, usually two-address or three-address instructions are needed for such operations. Sometimes one-address instructions may also be used for such operations, where one operand is inherently available and second operand is explicitly given. In addition, there are other arithmetic instructions that need single operand. Some of the single operand arithmetic operations are as follows.

Increment: Add 1 to the operand

Decrement: Subtract 1 from the operand

Absolute: Take absolute value of the operand

Negate: Negate the operand

Some examples of arithmetic instructions are given in Table 3.3.

3.3.3 Logical Instructions

Logical instructions manipulate bits or binary numbers. A number of logical instructions are part of the instruction set of the most of the computers. The operations that work on individual bits of a binary number are called the bitwise logical operations. The widely used bitwise logical instructions are **AND**, **OR**, **NOT**, and **XOR**, etc. The NOT operation works with one operand, while AND, OR, and XOR operations require two operands. Shift and rotation are other important logical operations. There are two types of logical shift operations: **logical left shift** and **logical right shift**. The logical left shift operation shifts the bits of a binary number from the least significant location towards the most significant location. The vacated location is filled with the 0s and bit coming out of the most significant location is just lost or discarded. The logical right shift operation shifts the bits from the most significant location towards the least significant location. The new 0s are inserted at the most significant position and bit coming out of the least significant location is discarded.

Table 3.3: Examples of logical instructions

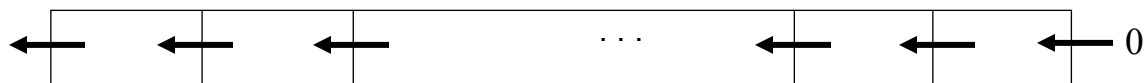
Instruction	Operation	Description
ADD	Addition	One-address or Two-address instruction for addition operation
SUB	Subtraction	One-address or Two-address instruction for subtraction operation
INC	Increment	Increase AC by one
CMA	Complement	Complement AC
CLA	Clear	Clear AC

Like logical shift, there are two types of logical rotate operations: **logical left rotate** and **logical right rotate**. The logical rotate is a kind of cyclic shift operation, where no new bit is inserted and no bit is discarded. While shifting the bit from location to another location, the bit that is coming out of one end is placed at the location vacated at other

end. In logical left rotate operation, a bit comes out of the most significant location and it is placed at the least significant location. Similarly, in logical right rotation, the bit that is coming out of the least significant location is placed at the most significant location. The logical shift and logical rotate operations are shown with the help of pictorial representation in Figure 3.3.

3.3.4 Transfer of Control Instructions

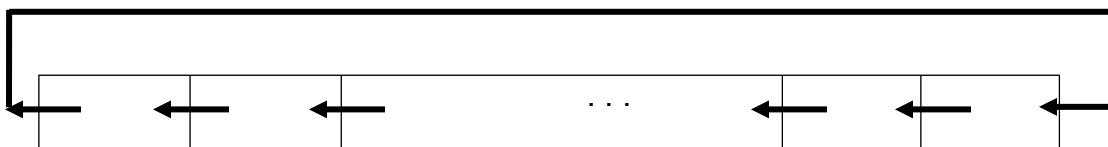
The instructions discussed so far manipulate data by performing some kind of operations. There are some other type of instruction that do not do data manipulation but change the sequence of instruction execution. These instructions are known as **transfer of control** instructions that update the program counter of the processor. Transfer of control instructions are further divided into different categories depending on their purpose and usage. **Branch instructions** are those transfer of control instructions are jump instructions that transfer flow of execution to a specified location. The jump could be forward or backward in direction. Moving to higher location address is forward jump and moving to a lower location address is backward jump as shown in Figure 3.4. If jump in sequence is made only if certain conditions met, then it is called conditional branch instruction. Otherwise, if there are no such conditions, then is an unconditional branch instruction. Some branch instructions are given in Table 3.4.



(a) Logical left shift

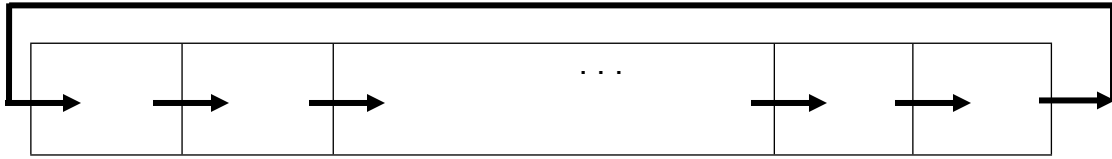


(b) Logical right shift



(c) Logical left rotate

Figure 3.3: Logical shift and logical rotate operations (continued)



(d) Logical right rotation

Figure 3.3: Logical shift and logical rotate operations



Figure 3.4: Forward and backward branch instructions

Table 3.4: Examples of branch instructions

Instruction	Operation	Description
BRP X	Jump	Branch to location X if result is positive
BRN X	Jump	Branch to location X if result is negative
BRZ X	Jump	Branch to location X if result is zero
BRO X	Jump	Branch to location X if overflow occurs
BRE R1, R2	Jump	Branch to location X if contents of R1 and R2 are equal

Skip instructions are another form of transfer of control instructions. Depending on a condition, these instructions skip one memory location. These instructions do not require destination location. The destination location is imperative in a skip instruction as it always skips next one location if conditions met. **Procedure call** is an important transfer of control instruction, which incorporate another piece of program into a larger program. The procedure call transfers the control of execution to the procedure. The control returns back at the location from which it was called upon completion of the procedure. Some other instructions related to I/O system, memory management, and system control are also the part of the instruction set of a computer.

Check your progress 2

1. What are different types of instructions?
2. Explain the difference between LOAD and STORE data transfer instructions?
3. Explain arithmetic instructions?
4. Explain how shift right and shift left logical instruction works?
5. What are branch instructions?

3.4 Addressing Modes

An instruction's operation field indicates the operation to be done. This operation must be performed on data stored in registers or memory words in a computer. The addressing mode of the instruction determines how the operands are chosen during programme execution. Before the operand is actually referenced, the addressing mode sets a guideline for interpreting or altering the address field of the instruction. The addressing modes are classified as follows:

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode
5. Autoincrement or Autodecrement Mode

6. Direct Address Mode
7. Indirect Address Mode
8. Relative Address Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

3.4.1 Implied Mode

The operands are implicitly stated in the specification of the instruction in this mode. Because the operand in the accumulator register is implied in the definition of the instruction, the instruction "complement accumulator" is an implied-mode instruction. In reality, all accumulator-based register reference instructions are implied-mode instructions. In a stack-organized computer, zero-address instructions are implied-mode instructions since the operands are assumed to be on top of the stack.

3.4.2 Immediate Mode

The operand is mentioned in the instruction itself in this mode. In other words, instead of an address field, an immediate-mode instruction has an operand field. The actual operand to be utilised in combination with the operation specified in the instruction is stored in the operand field. Immediate-mode instructions are useful for setting constant values in registers.

3.4.3 Register Mode

The address field of an instruction can specify either a memory word or a processor register, as previously stated. The instruction is considered to be in register mode when the address field specifies a processor register. The operands are stored in registers on the CPU in this mode. A register field in the instruction is used to select the specific register.

3.4.4 Register Indirect Mode

In this mode, the instruction provides a CPU register whose contents give the operand's memory address. In other words, the operand's address is stored in the specified register rather than the operand itself. The programmer must guarantee that the

operand's memory location is stored in the processor register with a preceding instruction before executing a register indirect mode instruction. In this case, referring to the register is the same as specifying a memory address. A register indirect mode instruction has the advantage of using fewer bits in the address field to choose a register than would be required if a memory address were specified directly.

3.4.5 Autoincrement or Autodecrement Mode

The register is incremented or decremented after (or before) its value is used to access memory, similar to the register indirect mode. When the address contained in the register refers to a memory table containing data, the register must be incremented or decremented after each access to the table. The increment or decrement instruction can be used to do this. Because this is such a common requirement, some computers include a special mode that automatically increments or decrements the register content on data access.

Effective Address: The control unit in the CPU uses the address field of an instruction to retrieve the operand from memory. The value in the address field is sometimes the operand's address, and other times it is only an address from which the operand's address is calculated. To distinguish between the various addressing modes, the address part of the instruction must be distinguished from the effective address used by the control when executing the instruction. The memory address produced from the computation specified by the given addressing mode is defined as the effective address. In a computational type instruction, the effective address is the address of the operand. Control branches at this address in response to a branch-type instruction.

3.4.6 Direct and Indirect Address Modes

The effective address is equal to the address part of the instruction in this mode. The operand is in memory, and the instruction's address field directly specifies its location. The address parameter in a branch-type instruction indicates the actual branch address. In indirect address mode, the instruction's address field specifies the location in memory where the effective address is kept. Control reads the effective address by retrieving the instruction from memory and using its address component to visit memory again.

3.4.7 Relative Address Mode

To acquire the effective address, the content of the programme counter is added to the address section of the instruction in this mode. The address portion of the instruction is typically a signed number (in 2's complement representation) that might be positive or negative. When this number is added to the contents of the programme counter, the result is an effective address that is relative to the next instruction's address in memory.

3.4.8 Indexed Addressing Mode

To acquire the effective address, the contents of an index register are added to the address component of the instruction in this mode. The index register is a specific CPU register that stores the value of an index. The instruction's address field specifies the start address of a data array in memory. Each array operand is stored in memory in relation to the start address. The index value stored in the index register is the distance between the beginning address and the address of the operand. If the index register contains the correct index value, any operand in the array can be accessed with the same instruction.

3.4.9 Base Register Addressing Mode

To acquire the effective address, the contents of a base register are added to the address component of the instruction in this mode. This is identical to indexed addressing, only the register is now referred to as a base register rather than an index register. The distinction between the two modes is in how they are used rather than how they are computed. An index register is supposed to contain an index number that is relevant to the instruction's address component. The address field of the instruction gives a displacement relative to the base address, which is expected to be held in a base register.

The effect of the addressing modes on the instruction defined in Figure 3.5 are shown to demonstrate the distinctions between the various modes. The two-word instruction at address 100 and 101 is a "load to AC" instruction with an address field equal to 400. The operation code and mode are specified in the first word of the instruction, while the address component is specified in the second word. For fetching

this instruction, PC has the value 100. The contents of processor register R1 and an index register XR are 251 and 100, respectively. After the instruction is executed, AC receives the operand. The figure lists a few key addresses and displays the memory content at each of them. The mode field of the instruction can specify any one of a number of modes. For each possible mode we calculate the effective address and the operand that must be loaded into AC.

In the direct address mode, the effective address is the address part of the instruction 400 and the operand to be loaded into AC is 600. **In the immediate mode** the second word of the instruction is taken as the operand rather than an address, so 400 is loaded into AC (The effective address in this case is 101). **In the indirect mode**, the effective address is stored in memory at address 400. Therefore, the effective address is 600 and the operand is 200. **In the relative mode**, the effective address is $400 + 102 = 502$ and the operand is 150. (Note that the value in PC after the fetch phase and during the execute phase is incremented to 102). **In the index mode**, the effective address is $XR + 400 = 100 + 400 = 500$ and the operand is 1200. **In the register mode**, the operand is in R1 and 251 is loaded into AC. (There is no effective address in this case). **In the register indirect mode**, the effective address is 251, equal to the content of R1 and the operand loaded into AC is 510. **The autoincrement mode**, is the same as the register indirect mode except that R1 is incremented to 252 after the execution of the instruction. **The autodecrement mode**, decrements R1 to 250 prior to the execution of the instruction. The operand loaded into AC is now 300.

3.5 Programming Considerations

The data are stored in the memory and can be directly accessed from thereon. But memory is too slow compare to the speed of microprocessor leading to slow execution. Therefore, processors have a set of registers which are small internal memory locations. The data elements are transferred from the memory to the registers to speed up the execution. An assembly language programmer should be aware of the registers available with the microprocessor and their purpose, capabilities, and limitations. A program is a sequence of instructions. A programmer needs good understanding of the instruction set, instruction types and their formats. However, for writing programs in a

high level language such as C, C++, or Java, these details are transparent to the programmer.

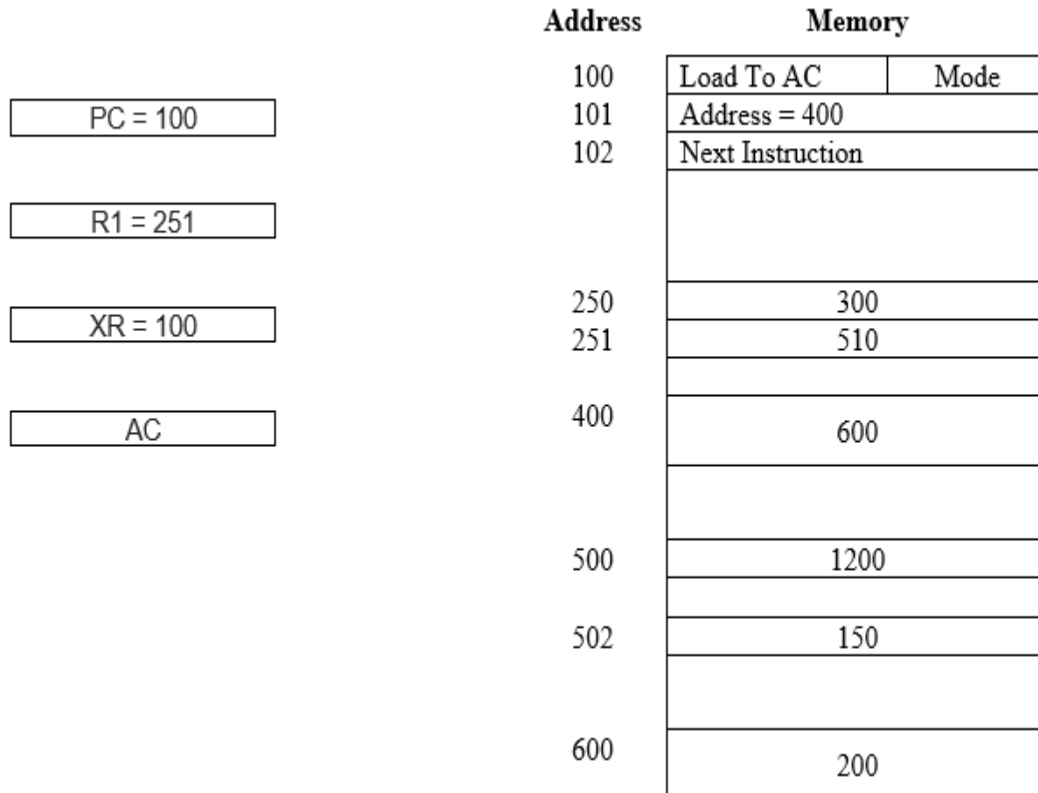


Figure 3.5: Example for addressing modes

3.6 Summary

The computers operate under the commands issued by the user. The commands are issued in the form of instructions. The instructions are binary code that specify a particular action that computer is supposed to carry out upon receiving it. Each instruction consists of a code and some operands. The code simply represents the operation to carry. An operand could be a data element or address of a memory location. Depending on the number and type of operands, the instructions have different formats. Some instructions do not require any explicit operand as they use implicit operands. There are unary and binary instructions. The unary instructions involve only

one operand, while binary instructions have two operands. An operand could be a source providing data or address required for the operation or it could be a destination operand where the result of the operation is placed.

There are different types of instructions in the instruction set of the computer. The arithmetic instructions are involved in the most of the operations. The logical instructions are used on binary numbers. There are transfer of control instructions that are very useful in writing the programs. A program is a set of instructions that specifies the sequence of operations to perform a particular task. The programmers need to have a good understanding of the instruction set of the machine. However, for high level languages, these details are almost transparent to the programmer.

Review Questions

- Q.1 List the typical elements of a computer instruction.
- Q.2 List different types of operands for an instruction.
- Q.3 Differentiate between logical shift and logical rotate operation.
- Q.4 What is use of transfer of control instructions? Give some examples of transfer of control instructions.
- Q.5 Differentiate between branch and skip instructions.
- Q.6 Explain different types of instruction formats used in computer system.
- Q.7 Why do data transfer instruction used? Write five examples of data transfer instructions.
- Q.8 What are addressing modes?
- Q.9 Explain immediate addressing mode?
- Q.10 What is effective address?
- Q.11 Explain the difference between register addressing mode and register indirect addressing mode.
- Q.12 What is the difference between direct addressing mode and indirect addressing mode?



**Uttar Pradesh Rajarshi Tandon
Open University**

Bachelor of Computer Application

**BCA-EC
Computer Architecture**

Block

2

DATA PATH DESIGN

Unit 4

Fixed Point Arithmetic

Unit 5

Arithmetic Logic Unit

Unit 6

Advanced Topics

BLOCK INTRODUCTION

The block 2 deals with arithmetic operations in a computer system. Original representation of the numbers may not be useful for performing operations in a computer system. Therefore, a number is converted to appropriate form to carry out arithmetic operations. In addition, a basic hardware support is also required for such operations. This block is divided into Unit 4, Unit 5, and Unit 6. The major basic mathematical operations such as addition, subtraction, multiplication, and division are discussed in Unit 4. Arithmetic Logic Unit (ALU) is primarily responsible for such operations. It consists of different types of circuits. Different types of digital circuits and digital electronic components are presented in Unit 5. On the other hand, several advanced topics related to arithmetic operations are covered in Unit 6.

UNIT- 4 Fixed-Point Arithmetic

Structure

4.0 Introduction

4.1 Objectives

4.2 Representation of Fixed-Point Numbers

4.3 Fundamental Rules of Fixed-Point Arithmetic

4.4 Addition and Subtraction

4.5 Multiplication and Division

4.6 Division

4.7 Summary

Review Questions

Unit 4: Fixed-Point Arithmetic

4.0 Introduction

This unit introduces a number of techniques to perform arithmetic operations on fixed-point representation of numbers. Data elements are stored in registers for the purpose of processing. The registers are made up of flip-flops. A flip-flop can store only one bit (1 or 0) of information. Therefore, all kind of data except binary numbers are store in binary-coded form in the registers. Binary number system is the most appropriate number system for the digital computers. In order to perform arithmetic or other kind of mathematical operations, the numbers are converted to binary numbers. All kind of information associated with a number is represented in binary form including the sign. A number is an integer, fraction, or mixed integer-fraction number is identified with the help of basis or radix point. The basis point is kept somewhere in the register itself along with the number. If the location of the basis point in the register is fixed, it is called fixed-point representation.

Sometimes, the original representation of the numbers may not be very useful for performing the operation. Therefore, the numbers are converted to 1's complement or 2's complement form before carrying out the operations. The major operations discussed in this unit are addition, subtraction, multiplication, and division. The complements are mostly useful in the simplification of the subtraction operation. To perform these operations, a basic hardware support is required. A basic circuit takes three inputs including one bit from each number and one carry. It generates two output bits representing sum and carry. This circuit is known as *full-adder* and it is used for addition/subtraction operations. Multiple full adders are connected in an arrangement to perform operations. The multiplication is carried out as a series of addition and shift operations. The division is performed as series of subtraction and shift operations. A number of other techniques and algorithms are also there for different operations. In comparison to addition and subtraction, the multiplication and division are complex operations. In particular, the division is the most complex operation among four basic arithmetic operations.

4.1 Objectives

The major objectives of this unit are outlined here.

1. To discuss methods for addition of fixed-point numbers.
2. To discuss methods for subtraction of fixed-point numbers.
3. To discuss methods for multiplication of fixed-point numbers.
4. To discuss methods for division of fixed-point numbers.

4.2 Representation of Fixed-Point Numbers

As discussed earlier the computer users are comfortable working with decimal number system as it is the system that we use in our daily routine works. On the other hand, the binary number system suits to digital computers due to technological reasons. Therefore, users are allowed to work on decimal numbers that are converted to binary or binary coded numbers while processing with computer systems. The signed numbers could be positive or negative. Some arrangement is required to include the sign information with the binary representation. For this purpose a bit in the register is designated as sign bit. Usually the left most bit of the binary string is designated as sign bit. For a positive number, the sign bit is 0 and for negative number it is 1. This arrangement is known as signed-magnitude form. Two other forms known as 1's complement and 2's complement are also used to represent negative numbers. In 1's complement all the bits of a binary string are complemented individually. If 1 is added to 1's complement of a number then it becomes 2's complement of the number.

A number could be an integer, a fraction, or a mixed integer-fraction number. In order to separate integer and fraction parts of a number, base point is used. The binary point is identified at a particular location in the register. If a specific location is used always for the base point, the numbers are known as fixed-point numbers. The fixed-point numbers are closely related to integer numbers. In fact, integer numbers can be considered as a special case of fixed-point numbers, where binary point is at position 0. Therefore, all mathematical operations that a computer can perform with integers can be applied on fixed-point numbers without any additional hardware. Some major arithmetic operations on fixed-numbers are discussed here.

4.3 Fundamental Rules of Fixed-Point Arithmetic

It is important to know the word length i.e. the number of bits required to represent the number. An unsigned number $U(a,b)$ with a integer bits and b fractional bits needs $a+b$ bits for the representation. A signed number $S(a,b)$ with same number of bits for integer and fractional parts requires $a+b+1$ bits. There are some points of consideration for the arithmetic operations as given here.

1. **Addition:** Two numbers are always added as two positive numbers.
2. **Unsigned word length:** The number of bits required to represent unsigned number $U(a,b)$ is $a+b$
3. **Signed word length:** The number of bits required to represent signed number $S(a,b)$ is $a+b+1$
4. **Unsigned range:** The range of unsigned numbers $U(a,b)$ is $0 \leq U(a,b) \leq 2^a - 2^{-b}$
5. **Signed range:** The range of signed numbers $S(a,b)$ is $-2^a \leq S(a,b) \leq 2^a - 2^{-b}$
6. **Addition of numbers:** Only those two numbers can be added that have same format. If one operand is unsigned number and other one is signed number then both operands need to be scaled to the same format before addition.
7. **Subtraction of numbers:** The same rule exists for the subtraction operation also. To subtract one number from another, the 2's complement or 1's complement of subtrahend is added to minuend.
8. **Addition result:** The result of the addition of two n -bit numbers requires $n+1$ bits.
9. **Overflow:** The result of addition operation may be larger than that can be held in the intended register. This situation is called overflow. To handle this situation a special bit known as **overflow bit** is used. If overflow occurs, the ALU must indicate it by setting the overflow bit.
10. **Carry:** If there is a carry in extreme left then it is discarded in case of 1's complement or it is added to the result in case of 2's complement.
11. **Unsigned multiplication:** The result of the multiplication of two unsigned numbers $U(a_1,b_1)$ and $U(a_2,b_2)$ would be an unsigned number $U(a_1 + a_2, b_1 + b_2)$.
12. **Signed multiplication:** The result of the multiplication of two signed numbers $S(a_1,b_1)$ and $S(a_2,b_2)$ would be a signed number $S(a_1 + a_2 + 1, b_1 + b_2)$.

13. **Unsigned division:** The result of the division of an unsigned number $U(a_1, b_1)$ by an unsigned number $U(a_2, b_2)$ would be an unsigned number $U(a_1 + b_2, \lceil \log(2^{a_2+b_1} + 2^{b_1-b_2}) \rceil)$.

14. **Signed division:** The result of the division of a signed number $S(a_1, b_1)$ by an unsigned number $S(a_2, b_2)$ would be a signed number $S(a_1 + b_2 + 1, a_2 + b_1)$.

Check your progress 1

1. How many ways are there to represent negative numbers?
2. What is sign bit?
3. Write range of the unsigned integers.
4. What is overflow bit?
5. What is base point?

4.4 Addition and Subtraction

In signed-magnitude form, the addition follows the conventional mathematical rules. The signs of two numbers are compared. If both signs are same, the two magnitudes are added and common sign is given to the result. In case two signs are different, the smaller magnitude is subtracted from the larger one and sign of the larger magnitude is given to the result. The 2's complement provides a simple approach for performing addition operation, where no comparison of signs and subtraction are required. The two numbers including sign bits are added and any carry out of the sign bit is discarded. The negative numbers must be in 2's complement form before addition. Some examples are given here. Addition of two positive numbers:

$$\begin{array}{r}
 +13 \quad 00001101 \\
 +19 \quad 00010011 \\
 \hline
 +32 \quad 00100000
 \end{array}$$

Addition of a positive and a negative numbers:

$$\begin{array}{r}
-13 \quad 11110011 \quad (2's \text{ complement}) \\
+19 \quad 00010011 \\
\hline
+6 \quad 00000110
\end{array}$$

Addition of two negative numbers:

$$\begin{array}{r}
-13 \quad 11110011 \quad (2's \text{ complement}) \\
-19 \quad 11101101 \quad (2's \text{ complement}) \\
\hline
-32 \quad 11100000 \quad (2's \text{ complement})
\end{array}$$

Sometimes, the result of addition of two positive numbers may result in a number that is larger than size of the registers. For example, if 8-bit registers are being used, the addition of two 8-bit positive integers may produce a 9-bit number. In such a case, it is not possible to place the result in a 8-bit register. This situation is known as **overflow**. If there is an overflow, it is indicated to the user. When two positive or two negative numbers are added and the result has opposite sign i.e. for positive numbers, the result is negative or for negative numbers, the result is positive then this situation indicates the overflow. In the following example, the addition of two positive numbers gives a negative result, therefore its overflow.

$$\begin{array}{r}
+71 \quad 01000111 \\
+78 \quad 01001110 \\
\hline
+149 \quad \mathbf{1}0010101 \quad \text{Overflow}
\end{array}$$

Similarly, in the following example, the addition of two negative numbers gives a positive result discarding the extra bit.

$$\begin{array}{r}
-95 \quad 10100001 \quad (2's \text{ complement}) \\
-67 \quad 10111101 \quad (2's \text{ complement}) \\
\hline
-162 \quad \mathbf{10}1011110 \quad \text{Overflow}
\end{array}$$

The subtraction can be easily carried out with 2's complement. The 2's complement of subtrahend is taken including the sign bit and it is added to the minuend. The carry out of sign bit position is discarded. For example

$$\begin{array}{r}
 +19 \quad 00010011 \\
 -13 \quad 11110011 \quad (2's \text{ complement}) \\
 \hline
 +6 \quad 00000110
 \end{array}$$

Both addition and subtraction operations can be performed with the help of the same circuit. The main element of the circuit as shown in Figure 4.1 is a binary adder. In case of an overflow, the adder indicates the situation with the help of a designated bit. There are two registers in the circuit, which are used to provide input. The result is also placed in a register, which could be any one of these two registers. There is a complementer to take 2's complement of negative numbers. A switch is used to select addition or subtraction operation whatever is required.

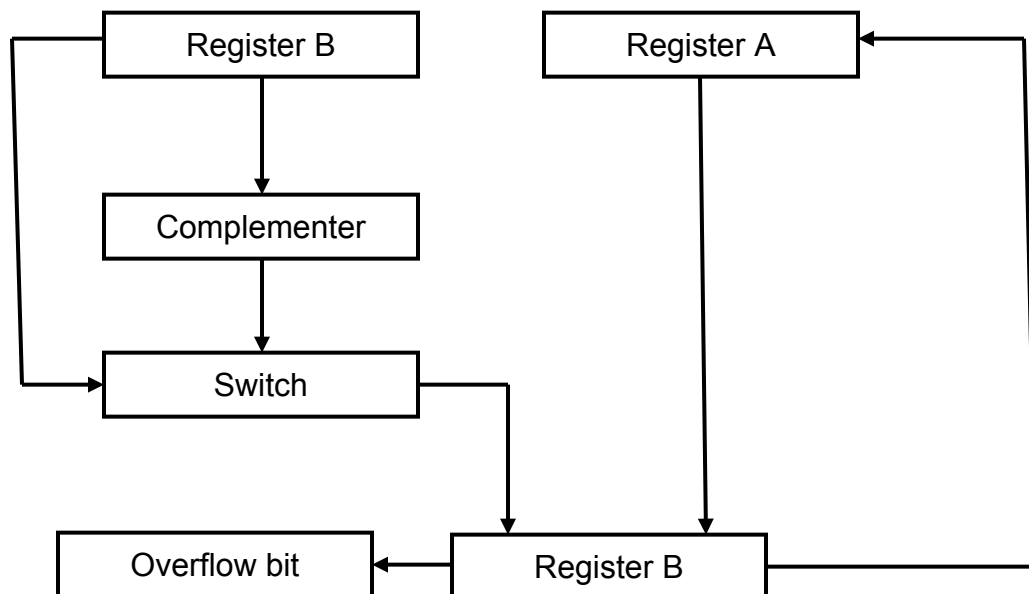


Figure 4.1: Circuit for addition and subtraction operations

4.5 Multiplication and Division

The multiplication operation is more complex than addition and subtraction operations. There are various ways and different algorithms for carrying out multiplication of two numbers. For different types of numbers.

4.5.1 Multiplication of unsigned numbers

The multiplication of two unsigned numbers can be performed by using a simple procedure similar to the paper-pencil method. In this method every bit of the multiplier is multiplied with the multiplicand in a shift and add manner. On the multiplication of two n -bit numbers, a $2n$ -bit result is produced. The procedure is explained with the help of an example as follows. In this example two numbers 14 $\{=(1110)_2\}$ and 13 $\{=(1101)_2\}$ are multiplied.

1 1 1 0	Multiplicand
× 1 1 0 1	Multiplier

1 1 1 0	Partial product 1
0 0 0 0	Partial product 2
1 1 1 0	Partial product 3
1 1 1 0	Partial product 4

1 0 1 1 0 1 1 0	Result

This is a simple method that produces partial products. The partial products are padded out to the left or right as needed with binary 0's. All the partial products are added to get the final result. The result of the multiplication is $(10110110)_2 = 182$. It can be observed that when a bit of the multiplier is 1, the partial product is multiplicand itself and it is simply copied. While if the multiplier bit is 0, the partial product is all zero. The partial products are shifted to one position to left from the previous partial product. The sum of the partial products gives the result. Therefore, this method can be implemented in circuit simply with the help of adders and shift registers.

4.5.2 Multiplication of signed numbers

The simple method that works well for the unsigned number does not go well with signed numbers. Like other operations, the negative numbers are represented in 2's complement form. If any of the multiplier and multiplicand to both are negative than the simple multiplication method does not work. If multiplicand is a negative number then problem is that all the partial products should also be negative. It could be accomplished by padding out the partial products to the left with binary 1's. If the multiplier is a negative number and it is represented in 2's complement form, there are issues related to the actual bit position during multiplication and shift operations. Thus, for signed numbers some different ways of multiplications are adopted.

There are many different methods for performing multiplication of signed numbers. One of the most popular methods is known as Booth's algorithm. It works on the numbers in 2's complement form. The algorithm uses following rules:

1. For a binary bit 0 in the multiplier, no addition but only shifting is required.
2. Based on the bit positions, a string of 1's can be considered as $2^{n+1} - 2^m$, where m and n are the lowest and highest bit positions respectively in the string or substring such that $n > m$. For such a multiplier, the multiplication with a multiplicand Q is equivalent to $Q \times 2^{n+1} - Q \times 2^m$. Let us consider 5×30 . The binary representation of multiplicand 30 is $(00011110)_2$. Here $m=1$ and $n=4$, thus $30 = 2^{4+1} - 2^1$,
Therefore, $5 \times 30 = 5 \times (2^{4+1} - 2^1) = 5 \times (32 - 2) = (160 - 10) = 150$.
3. Like simple multiplication method, the Booth's algorithm also uses the shifted partial products. It examines the multiplier's two least significant bits to decide whether the partial product should be added or subtracted to the previous partial product or left unchanged as per following criteria.
 - (i) If 00: no operation
 - (ii) If 01: add the multiplicand to partial product
 - (iii) If 10: subtract the multiplicand from partial product
 - (iv) If 11: no operation

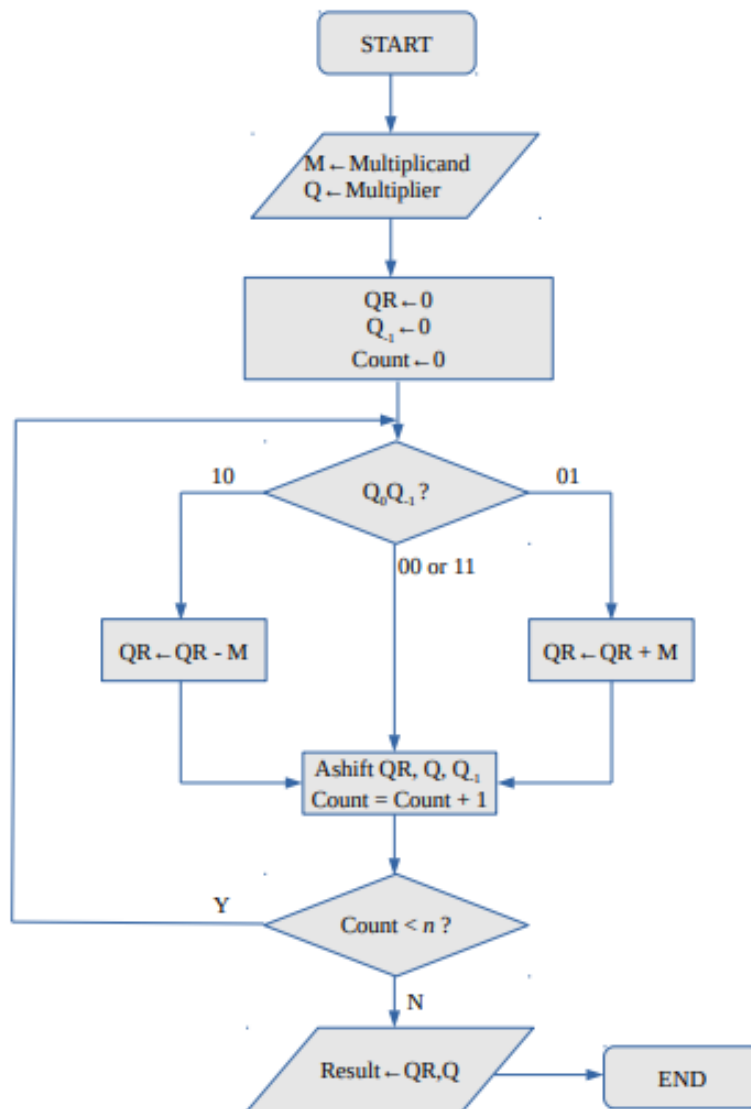


Figure 4.2: Booth's algorithm for multiplication

The method is depicted with the help of a flow chart in Figure 4.2. The multiplier and multiplicand are placed in the registers Q and M respectively. A flip-flop Q_{-1} is appended to register Q to facilitate double bit examination of the multiplier. A register QR is used along with Q to hold the result. The algorithm works as follows.

Step 1: Place the multiplicand and multiplier in registers M and Q respectively.

- Step 2: Initialize both QR and Q_{-1} to 0.
- Step 3: Examine two bits Q_0, Q_{-1}
If (Q_0Q_{-1} is 00 or 11) *Then* Shift QR, Q, and Q_{-1} to right one bit
Else If (Q_0Q_{-1} is 01) *Then* ($QR \leftarrow QR + M$) & (Shift QR, Q, and Q_{-1} to right one bit)
Else If (Q_0Q_{-1} is 10) *Then* ($QR \leftarrow QR - M$) & (Shift QR, Q, and Q_{-1} to right one bit)
- Step 4: *If* (All the bits of Q are examined) *Then* GOTO Step 3 *Otherwise* GOTO Step 5
- Step 5: STOP

Table 4.1: Multiplication of 3 and -7 using Booth's Algorithm

Multiply: $3 \times (-7)$					
Iteration	Operation	Multiplicand (M)	QR	Multiplier (Q)	Q_{-1}
	Initialize	0011	0000	1001	0
1	$Q_0Q_{-1} = 10, QR = QR - M$	-	1101	1001	0
	AShift QR, Q, Q_{-1}	-	1110	1100	1
2	$Q_0Q_{-1} = 01, QR = QR + M$	-	0001	1100	1
	AShift QR, Q, Q_{-1}	-	0000	1110	0
3	$Q_0Q_{-1} = 00, \text{No operation}$	-	0000	1110	0
	AShift QR, Q, Q_{-1}	-	0000	0111	0
4	$Q_0Q_{-1} = 10, QR = QR - M$	-	1101	0111	0
	AShift QR, Q, Q_{-1}	-	1110	1011	1
Result (QR,Q) = 11101011 (2's complement) = -21					

The right operation used in Booth's algorithm is an arithmetic shift (Ashift), which is performed in such a way that sign bit is preserved during the shift. Some examples of

multiplication of signed numbers using Booth's algorithm are given in Table 4.1 and Table 4.2. In these examples 4-bit numbers (including sign bit) are used, therefore, algorithm runs for four iterations. It should be noted that in these tables, if QR, Q, and Q_{-1} are 1xxx, yyyy, and z respectively then after Ashift operation QR, Q, and Q_{-1} become 11xx, xyyy, and y respectively. On the other hand, if QR, Q, and Q_{-1} are 0xxx, yyyy, and z respectively then after Ashift operation QR, Q, and Q_{-1} become 00xx, xyyy, and y respectively.

Table 4.2: Multiplication of -3 and 5 using Booth's Algorithm

Multiply: $-3 \times (5)$					
Iteration	Operation	Multiplicand (M)	QR	Multiplier (Q)	Q_{-1}
	Initialize	1101	0000	0101	0
1	$Q_0Q_{-1} = 10, QR = QR - M$	-	0011	0101	0
	AShift QR, Q, Q_{-1}	-	0001	1010	1
2	$Q_0Q_{-1} = 01, QR = QR + M$	-	1110	1010	1
	AShift QR, Q, Q_{-1}	-	1111	0101	0
3	$Q_0Q_{-1} = 10, QR = QR - M$	-	0010	0101	0
	AShift QR, Q, Q_{-1}	-	0001	0010	1
4	$Q_0Q_{-1} = 01, QR = QR + M$	-	1110	0010	1
	AShift QR, Q, Q_{-1}	-	1111	0001	0
Result (QR, Q): 11110001 (2's complement) = -15					

Table 4.3: Multiplication of -4 and -6 using Booth's Algorithm

Multiply: $-4 \times (-6)$					
Iteration	Operation	Multiplicand (M)	QR	Multiplier (Q)	Q_{-1}
	Initialize	1100	0000	1010	0
1	$Q_0Q_{-1} = 00$, No operation	-	0000	1010	0
	AShift QR, Q, Q_{-1}	-	0000	0101	0
2	$Q_0Q_{-1} = 10$, $QR = QR - M$	-	0100	0101	0
	AShift QR, Q, Q_{-1}	-	0010	0010	1
3	$Q_0Q_{-1} = 01$, $QR = QR + M$	-	1110	0010	1
	AShift QR, Q, Q_{-1}	-	1111	0001	0
4	$Q_0Q_{-1} = 10$, $QR = QR - M$	-	0011	0001	0
	AShift QR, Q, Q_{-1}	-	0001	1000	1
Result (QR, Q) = 00011000 = 24					

Check your progress 2

1. How addition of negative numbers is performed?
2. How do you identify overflow during addition?
3. How the same circuit is used for both addition and subtraction operations?
4. Which operations are performed with Booth's algorithm?
5. Which form of numbers is used in Booth's algorithm?

4.6 Division

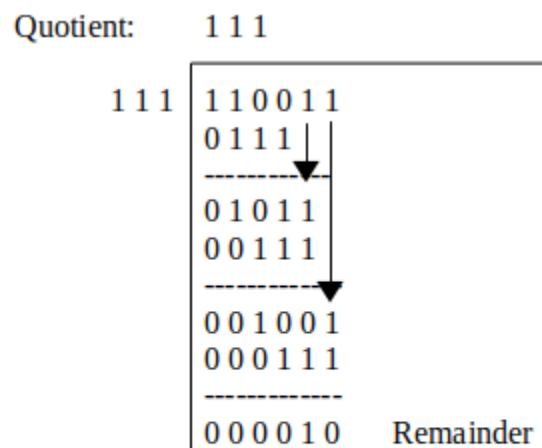
The division operation is little complex than multiplication. The computer method for division is also inspired by the paper-pencil method, where digits of the dividend are

repeatedly shifted and divisor is subtracted to produce intermediate remainders. The digits of the dividend are inspected from left to right until the inspected digits form a number greater than or equal to divisor. This number is divided by divisor and remainder (if any) is clubbed with next digits to form another number that is again divided by divisor. The process goes on until all the digits of the dividend are exhausted. The process is illustrated for binary number as follows.

Example of unsigned binary division (paper-pencil method): $51/7$

Dividend: $51 = (110011)_2$

Divisor: $7 = (111)_2$



Quotient: $(0\ 0\ 0\ 1\ 1\ 1)_2 = 7$

Remainder: $(0\ 0\ 0\ 0\ 1\ 0)_2 = 2$

Based on this simple method, a machine algorithm can be designed for unsigned numbers using shift, addition, and subtraction operations. The divisor is placed in register M and dividend is stored in register Q. Another register QR is also used in the process, which is set to zero initially. At each step the register QR and Q are left shifted by one bit and M is subtracted from QR. If QR divides the partial remainder, QR contains a non-negative number and Q_0 is set to 1. Otherwise Q_0 is reset to 0 and M is added back to QR. After processing all the bits of dividend, the quotient and remainder are available in registers Q and QR respectively. The algorithm is illustrated with the help of a flowchart in Figure 4.3.

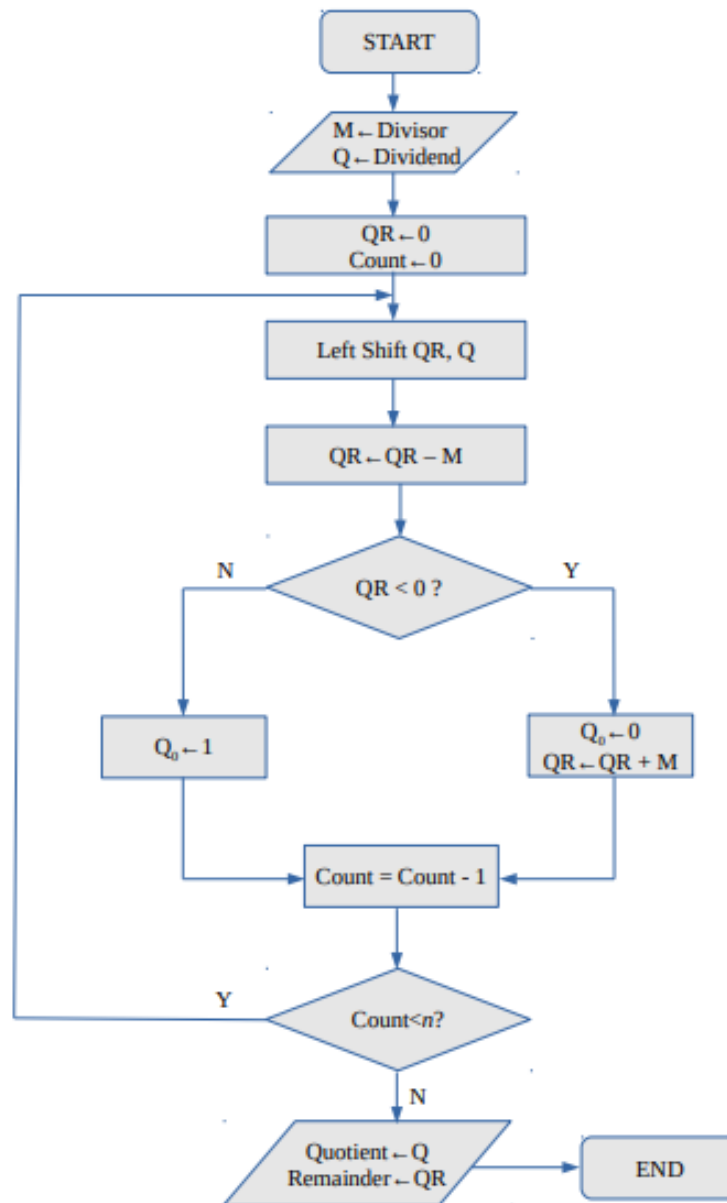


Figure 4.3: Unsigned division operation

The step by step algorithm for n -bit numbers is given as follows.

- Step 1: Place the dividend and divisor in registers M and Q respectively.
- Step 2: Initialize both QR and a counter C to 0.
- Step 3: Shift left QR and Q

Step 4: $QR \leftarrow QR - M$
 Step 5: *If* $(QR < 0)$ *then* $Q_0 \leftarrow 0, QR \leftarrow QR + M$ *Else* $Q_0 \leftarrow 1$
 Step 6: $C = C + 1$
 Step 7: *If* $(C < n)$ *then* GOTO Step 3 *Else* STOP

Table 4.3: Unsigned division

Divide: $29 \div 3$				
Iteration (C) $n=6$	Operation	Divisor (M)	QR	Dividend (Q)
	Initialize	000011	000000	011101
0	Left shift QR, Q		000000	111010
	$QR = QR - M$		111101	111010
	$QR < 0$		000000	111010
1	Left shift QR, Q		000001	110100
	$QR = QR - M$		111110	110100
	$QR < 0$		000001	110100
2	Left shift QR, Q		000011	101000
	$QR = QR - M$		000000	101000
	$QR > 0$		000000	101001
3	Left shift QR, Q		000001	010010
	$QR = QR - M$		111110	010010
	$QR < 0$		000001	010010
4	Left shift QR, Q		000010	100100
	$QR = QR - M$		111111	100100
	$QR < 0$		000010	100100

5	Left shift QR, Q		000101	001000
	QR = QR – M		000010	001000
	QR > 0		000010	001001
Result: Quotient (Q) = $(001001)_2 = 7$, Remainder (QR) = $(000010)_2 = 2$				

4.7 Summary

The mathematical operations by the machine are performed on binary numbers. All the operands are therefore converted to binary numbers during the operations. The addition and subtraction operations are comparatively simple mathematical operations. In case of addition, the signs of two operands are compared. If two signs are same, both numbers are simply added and common sign is assigned to the result. However, if two numbers have different signs, the 2's complement of the negative number is added to the other number. Any carry out of the sign bit is discarded. No comparison of magnitudes is required with this approach. Both addition and subtraction operations can be performed using the same hardware circuit that needs some adders and shift registers.

Multiplication and division operations are complex operations as compared to addition and subtraction operations. The multiplication operation can be performed in many different ways. Booth's algorithm is one of the most widely used algorithm for multiplication. It is executed using some shift, addition, and subtraction operations. The division operation is also carried out with the help of shift, addition, and subtraction.

Review Questions

Q.1 Perform following operations with binary numbers. For negative numbers use 2's complement form.

- (i) $(+25) + (-39)$
- (ii) $(-19) + (+37)$
- (iii) $(-17) + (-23)$

(iv) $(-34) - (+19)$

(v) $(+26) - (+15)$

Q.2 Apply Booth's algorithm to multiply following numbers. Show step-by-step process.

(i) $14 \times (-11)$

(ii) $-14 \times (-11)$

(iii) $-13 \times (14)$

Use 5-bit binary numbers including sign bit.

Q.3 Divide 145 by 13 using the machine algorithm on binary numbers. Show step by step process.

Q.4 Divide (-45) by 7 using the machine algorithm on binary numbers. Show step by step process.

Q.5 Divide 45 by (-7) using the machine algorithm on binary numbers. Show step by step process.

Q.6 Find the difference:

(i) $11010 - 01110$

(ii) $01111 - 00111$

(iii) $11100 - 10111$

(iv) $11110 - 11011$

UNIT- 5 Arithmetic Logic Unit

Structure

5.0 Introduction

5.1 Objectives

5.2 Logic Gates

5.3 Combinational Circuits

5.4 Sequential Circuits

5.5 Summary

Review Questions

Unit 5: Arithmetic Logic Unit

5.0 Introduction

The computer system consists of many interconnected parts including **control unit**, **arithmetic logic unit (ALU)**, I/O devices, and memory, etc. ALU is that component of the system that performs the arithmetic and logical operations on data. It is a digital circuit that is the fundamental building block of the **central processing unit (CPU)**. In some processors, the ALU is divided into two parts: **arithmetic unit (AU)** and **logic unit (LU)**. The AU performs arithmetic operations such as addition, subtraction, multiplication, and division. The LU performs different logic operations. There may be more than one AUs in some processors for fixed-point operations and floating-point operations separately. Some processors have multiple and complex ALUs.

A typical ALU has direct access to memory, controller, and I/O system. The data transfer between these components takes place through a bus. ALU receives an instruction word that contains an **op-code**, operands, and **format code**. The op-code is used to tell the ALU about the operation that it is supposed to carry out. The operands are the data elements on which operation is performed. The format code tells whether it is a fixed-point or floating-point operation. It is not necessary to include format code separately, instead it may be combined with the op-code. The result of the operation is stored in a register called **accumulator** and completion of the result is indicated with the help of machine **status word**. The ALU performs its operations with the help of a circuit of **logic gates**. The logic gates are tiny digital devices that can perform simple Boolean operations. A typical ALU can symbolically be represented as shown in Figure 5.1.

The control lines A and B in Figure 5.1, provide the operands to perform operation on them. The operation is specified with the help of op-code that is provided by control line F. The completion of the operation is indicated through control line D and results are provided by R. The status D provides information on carry-in, carry-out, overflow, and division-by-zero, etc. Overflow indicates integer overflow of add and subtract operations on unsigned integers, carry-out line indicates unsigned integer overflow. ALU is the most important component of CPU. It can be considered as the

heart of CPU since all other components in CPU are there to provide support to ALU. Basically ALUs are digital circuits that can be divided into two broad categories: **combinational circuits** and **sequential circuits**. The output in combinational circuits depends on present input only. With the same input, each time, exactly the same output is generated. On the other hand, the sequential circuits consists of some memory elements and their output is influenced by the previous results. The output depends on input values as well as the current state of the circuit. Therefore, with same set of inputs, the output may differ during different executions.

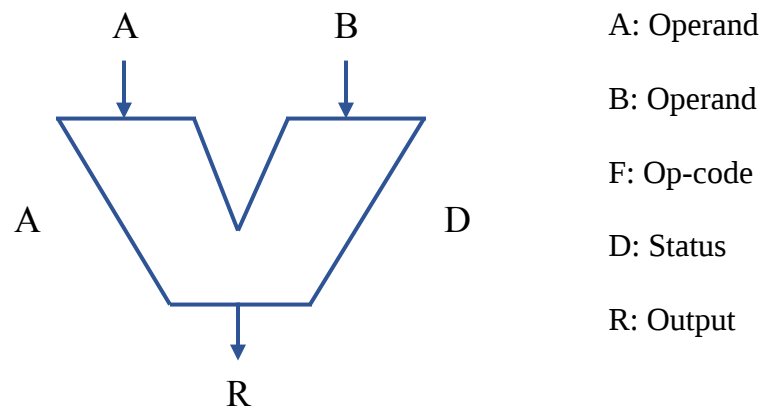








Figure 5.1: Schematic symbol of an ALU

5.1 Objectives

The major objectives of this unit are as follows:

1. To have knowledge of different types of logic gates.
2. To learn to construct and simplify the digital circuits using logic gates.
3. To have knowledge of different types of latches and flip flops.
4. To study the combinational and sequential circuits used in ALUs.
5. To study the implementation of digital circuits to perform addition, subtraction, multiplication, and division operations.
6. To study the implementation of registers and counters.

Table 5.1: Graphical symbols, algebraic functions, and truth tables for logic gates

Name	Symbol	Truth Table															
AND		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	O	0	0	0	0	1	0	1	0	0	1	1	1
A	B	O															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	O	0	0	0	0	1	1	1	0	1	1	1	1
A	B	O															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
XOR		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	O	0	0	0	0	1	1	1	0	1	1	1	0
A	B	O															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
NAND		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	O	0	0	1	0	1	1	1	0	1	1	1	0
A	B	O															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
NOR		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	O	0	0	1	0	1	0	1	0	0	1	1	0
A	B	O															
0	0	1															
0	1	0															
1	0	0															
1	1	0															
NOT		<table border="1"> <thead> <tr> <th>A</th> <th>O</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	O	0	1	1	0									
A	O																
0	1																
1	0																

5.2 Logic Gates

The binary information in digital computers is presented in the form of electric signals. The two different levels of voltage represent binary value 0 or 1. For example 5 volt and 0.5 volt may represent 1 and 0 respectively. A gate is a tiny device that performs some basic operation on electric signals. The gates can be combined into circuits to perform more complex tasks. Logic gates are the basic building blocks in the microprocessor. The circuits can be implemented with the help of logic gates to perform logical **AND**, **NOT**, **OR**, **XOR** and other Boolean operations. The logic gates can be implemented using transistors. A transistor is a semiconductor device that acts like a switch that either conducts electricity or blocks the flow of electricity depending on the level of

voltage. The behaviour of logic gates and circuits can be described with the help of a **logic diagram**. The logic diagram is one of the most popular methods to represent the logic gates and circuits with the help of graphical symbols. Each gate type is represented by a specific graphical symbol. Another important method to describe the function of a circuit is **truth table**, which defines the circuit behaviour by listing all possible input combinations and corresponding output in the form of a table. Another method to describe the function of a logic gate or circuit is **Boolean expression**. The Boolean expressions use algebraic notations to demonstrate the activity of electric circuits. There are four fundamental logic gates: NOT, OR, AND, and XOR that can be used to construct other logic gates and logic circuits.

The NOT gate is also known as the inverter. It produces the logic complement of the binary number. It accepts one input value and produces one output. If input to NOT gate is 0 then output is 1 and if input is 1 then output is 0. AND, OR, and XOR all accept two input values and produce one output value. If two input values to AND gate are both 1, the output is 1 otherwise the output is 0. The output of OR gate is 0 only if both input values are 0, otherwise it is 1. The XOR is also known as exclusive OR gate. It produces an output 0 when both input values are same, otherwise, the output is 1. Some other gates can be designed with the combinations of fundamental gates. A **NAND** gate is developed if a NOT gate is placed in front of the AND gate. It is essentially the opposite of the AND gate. Similarly, a **NOR** gate is obtained by placing NOT gate in front of OR gate, which is opposite of the OR gate. The graphical symbols or logic diagrams, and truth tables for the six gates are given in Table 5.1.

Apart from the above discussed basic gates, some more gates can be developed with multiple input lines (three or more). For example a three input AND gate produces an output 1 only when all the three input lines receive a binary value 1. Different gates can be combined to form digital circuits. It could be a combinational or sequential circuit. A digital circuit can also be described with the help of a logic diagram, Boolean expression, or truth table. Multiple NAND or NOR gates can be interconnected in such a way that any of the basic gates can be constructed. Therefore, both NAND and NOR gates are called universal gates as other gates can be developed using solely NAND or solely NOR gates.

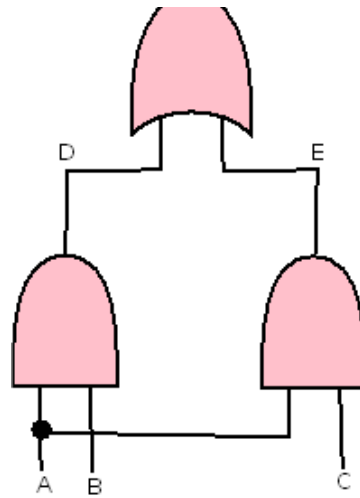


Figure 5.2: An example of combinational circuit

Table 5.2: Truth table for the circuit given in Figure 5.2

A	B	C	D	E	X
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

5.3 Combinational Circuits

The combinational circuits are made of logic gates with no memory elements. Therefore, the combinational circuits cannot store any information. The present output solely depends on the present input. The output of one logic gate is connected to the

input of other logic gate to form a combinational circuit. An example of combinational circuit is given in Figure 5.2. In this circuit output ports of two AND gates are connected to an OR gate. There are three input lines A, B, and C and one output line X.

The truth table of the circuit is given in Table 5.2. Since there are three input lines to the circuit, eight input combinations are possible. Therefore, there are eight rows in the given truth table to describe all possible input combinations. There could be several other alternative realizations of any circuit. The given circuit can be described with the help of a Boolean expression given as follows

$$X = (AB + AC)$$

However, it can be observed from the truth table that there are three combinations of inputs A, B, and C that lead to output 1. If any of these combinations takes place, the output is 1. Therefore, considering the truth table the same circuit can also be represented by following Boolean expression

$$X = \overline{A}BC + A\overline{B}C + ABC$$

The above form of expressions is known as **Sum of Products (SOP)**. For a given truth table, the corresponding circuit can be implemented with the help of SOP using the AND, OR, and NOT gates only. The SOP implementation of truth table given by Table 5.2 is provided in Figure 5.3. In the given circuit, corresponding to each algebraic term of SOP expression, there is a AND gate. The output lines of all the AND gates are connected to the same OR gate. If output of any AND gate is true, the output of the circuit is 1.

There is another alternative approach for the implementation of digital circuits corresponding to given truth table. If none of the input combinations that produce 0 is true then the output of the circuit would be 1. This approach is known as **Product of Sums (POS)**. It can be expressed in Boolean algebraic form as follows

$$X = (\overline{A}BC).(\overline{A}B\overline{C}).(\overline{A}\overline{B}C).(\overline{A}\overline{B}\overline{C}).(\overline{A}BC)$$

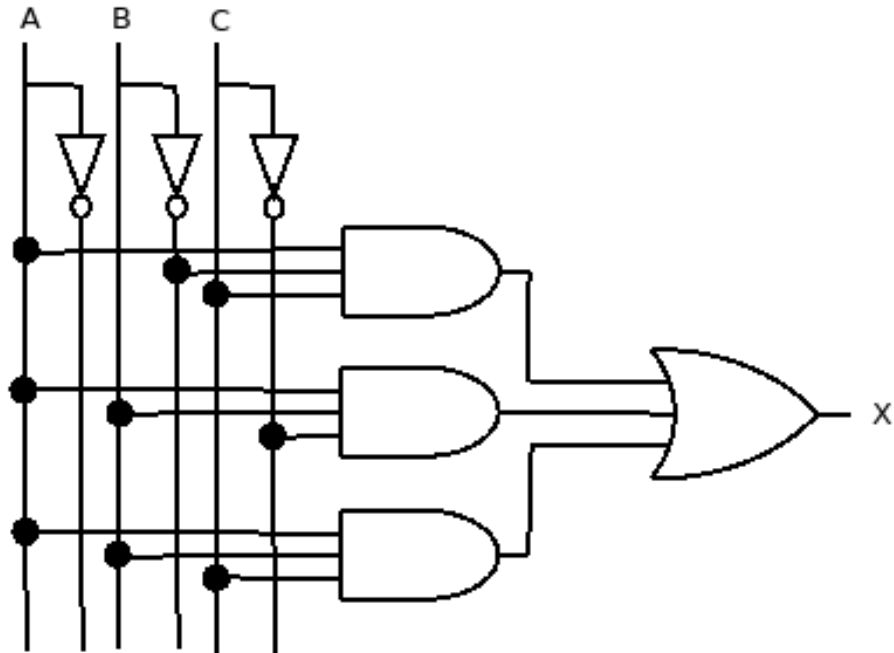


Figure 5.3: SOP implementation of the truth table given in Table 5.2

The POS implementation of the truth table given in Table 5.2 is given in Figure 5.4. Any truth table can be implemented in SOP or POS form. The choice depends on whether the truth table has more number of 0s or 1s as output. SOP has an AND gate for each term 1 and POS has an OR gate for each term 0. So if there are more number of 1s than 0s as output in the truth table then POS form is preferred otherwise SOP form is preferred leading to less number of gates in the circuit resulting in lower cost. However, there may be other expression that lead to even lesser number of gates. Therefore, a particular choice of the implementation depends on various factors. For example the Boolean expression $X=(AB + AC)$ can be simplified by algebraic method as follows

$$X=A(B+C)$$

The corresponding implementation is given in Figure 5.5, which has just two gates for the same truth table given in Table 5.2.

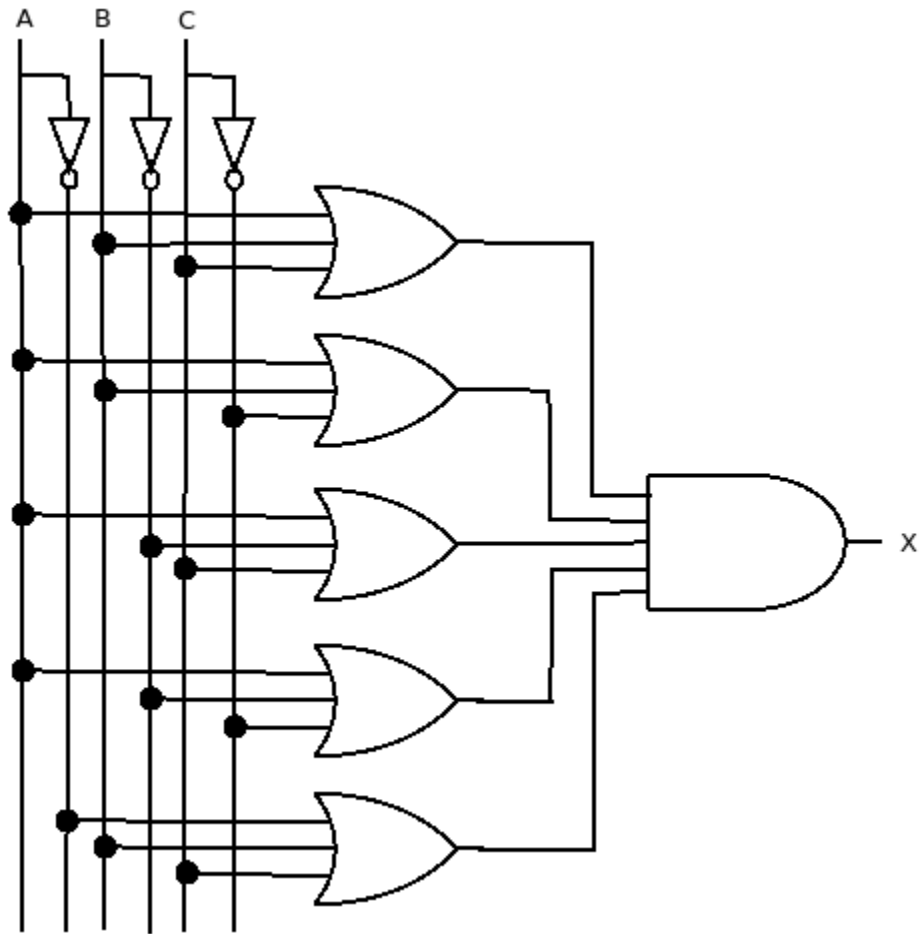


Figure 5.4: POS implementation of the truth table given in Table 5.2

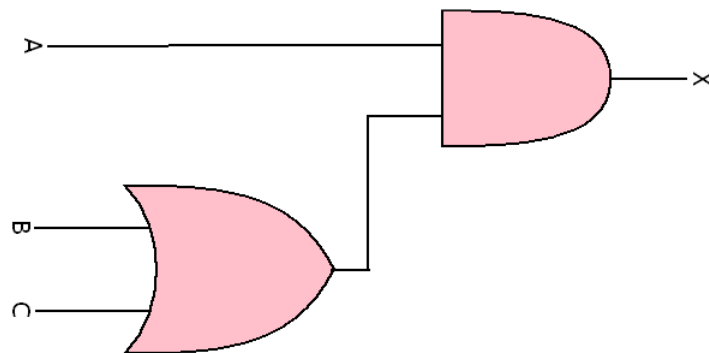


Figure 5.5: Implementation of the simplified Boolean expression

5.3.1 Karnaugh Maps

The Boolean expression for a truth table should be simplified before the implementation to reduce the number of logic gates and subsequently the cost of the circuit. There are various other methods for Boolean expression simplification. The **Karnaugh Maps** or **K-maps** is one of the most popular method for Boolean expression simplification. It represents the truth table in pictorial way that contains an array of 2^n squares for n variable binary data. Each square represents a possible combination of variables in SOP form called minterm. Each square in the array or map is representing a minterm. The squares corresponding to minterms that produce 1 are marked by a 1, while others are marked by a 0 or left blank. For easy reference in the map, each minterm is assigned a minterm reference binary number. The adjacent squares are assigned numbers that differ only by one digit. Thus, the combinations are listed in the following order 00, 01, 11, and 10.

Once K-map for a function is created, the corresponding Boolean expression can be simplified by considering position of 1s in the map. The process of simplification involves identification of two adjacent squares such that both have entry of 1s. The minterms for these squares can be merged. The adjacent squares differ in only one variable and that variable can be eliminated while merging the two terms.

5.3.2 Half Adder

Addition is the most basic arithmetic operation. A combinational circuit can be designed using logic gates to perform the addition of binary digits. Half adder is a combinational circuit that can perform addition of two binary digits. Half adder takes two digits called as augend (A) and addend (B) as inputs and generates two output variables known as sum (S) and carry (C). The circuit of half adder is shown in Figure 5.7 and its truth table is given in Table 5.3. It uses an XOR gate and an AND gate. There are two output ports because addition of 1 and 1 gives 10, which has two digits. S provides the least significant bit of the result. The S and C can be provided by the following expressions

$$S = A \oplus B$$

$$C = AB$$

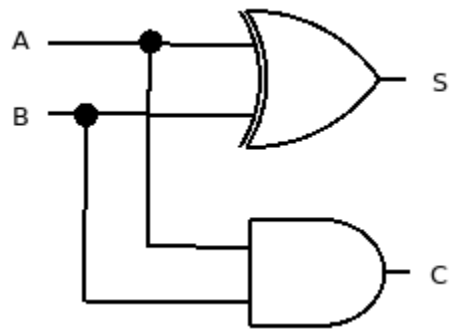


Figure 5.7: Combinational circuit for half adder

Table 5.3: Truth table for half adder

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

5.3.3 Full Adder

A full adder performs addition of three binary digits. It is also implemented as combinational circuit by using two half adders as shown in Figure 5.8 and truth table is given by Table 5.4. It takes three binary digits A, B, and D as input and provides two binary output variables S and C. Two input variables A and B are two significant bits, while third input variable D is the carry (also called input carry) from previous least bit position. The output variable S gives the least bit of the sum and C is the output carry. The Boolean expression for sum and carry can be written as follows

$$S = A \oplus B \oplus D$$

$$C = AB + (A \oplus B)D$$

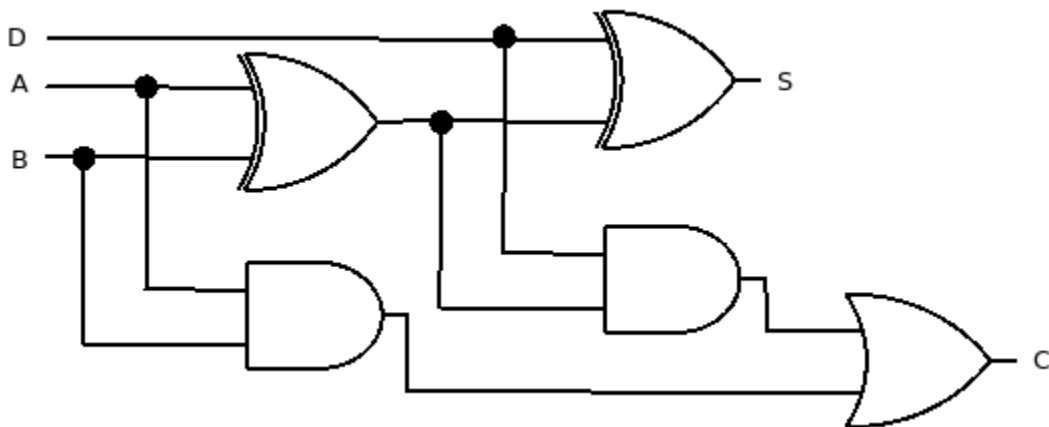


Figure 5.8: Combinational circuit for full adder

Table 5.4: Truth table for full adder

A	B	D	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

5.3.4 Multiplexers

A multiplexer is a combinational circuit that connects multiple input lines to a single output. It selects a particular input line to connect to output based on the signals from control lines. Figure 5.9 shows a block diagram of an eight-to-one multiplexer with eight input lines marked as D0 through D7 and three control lines S0, S1, and S2. A 3-bit signal is used to select a particular input line to provide the output F. The truth table for the multiplexer is given in Table 5.5.

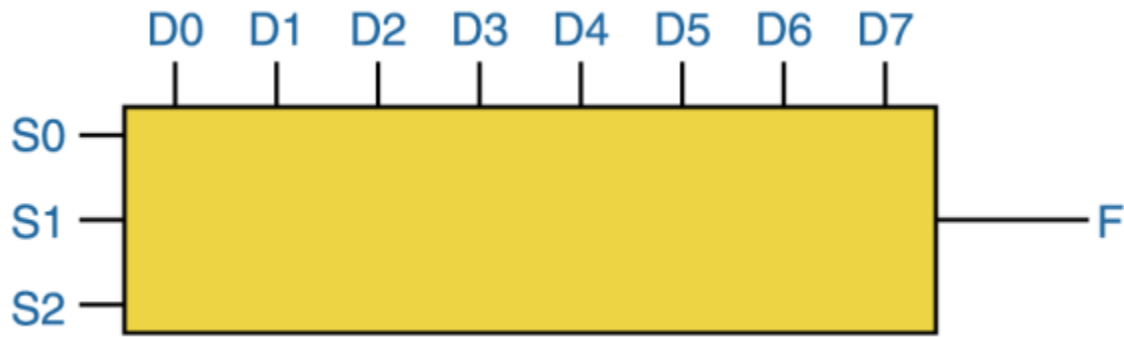


Figure 5.9: Block diagram of 8-to-1 multiplexer

(Source: University of Loyola Chicago, anh.cs.luc.edu)

Table 5.5: Truth table for 8-to-1 multiplexer

S0	S1	S2	F
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

The implementation of 8-to-1 multiplexer is shown in Figure 5.10. It is implemented with the help of eight AND gates and an OR gate. The signal lines S0, S1, and S2 are connected to AND gates in such a way that at a time seven AND generate 0, while the remaining one AND produces the value of the selected line, which could be 0 or 1.

5.3.5 Decoders

Another example of combinational circuits is a decoder, which generally has n inputs and 2^n output lines. At any time only a particular output line is asserted depending on the pattern of input. Decoders are many applications in digital computers such address

decoding. With some minor changes, the decoder can be converted to demultiplexer. Demultiplexer connects single input line to multiple output lines. It performs inverse function of a multiplexer. It can also be implemented with the combination of AND and OR gates. A circuit for a decoder with two inputs and four outputs is shown in Figure 5.11 and its truth table is given in Table 5.6.

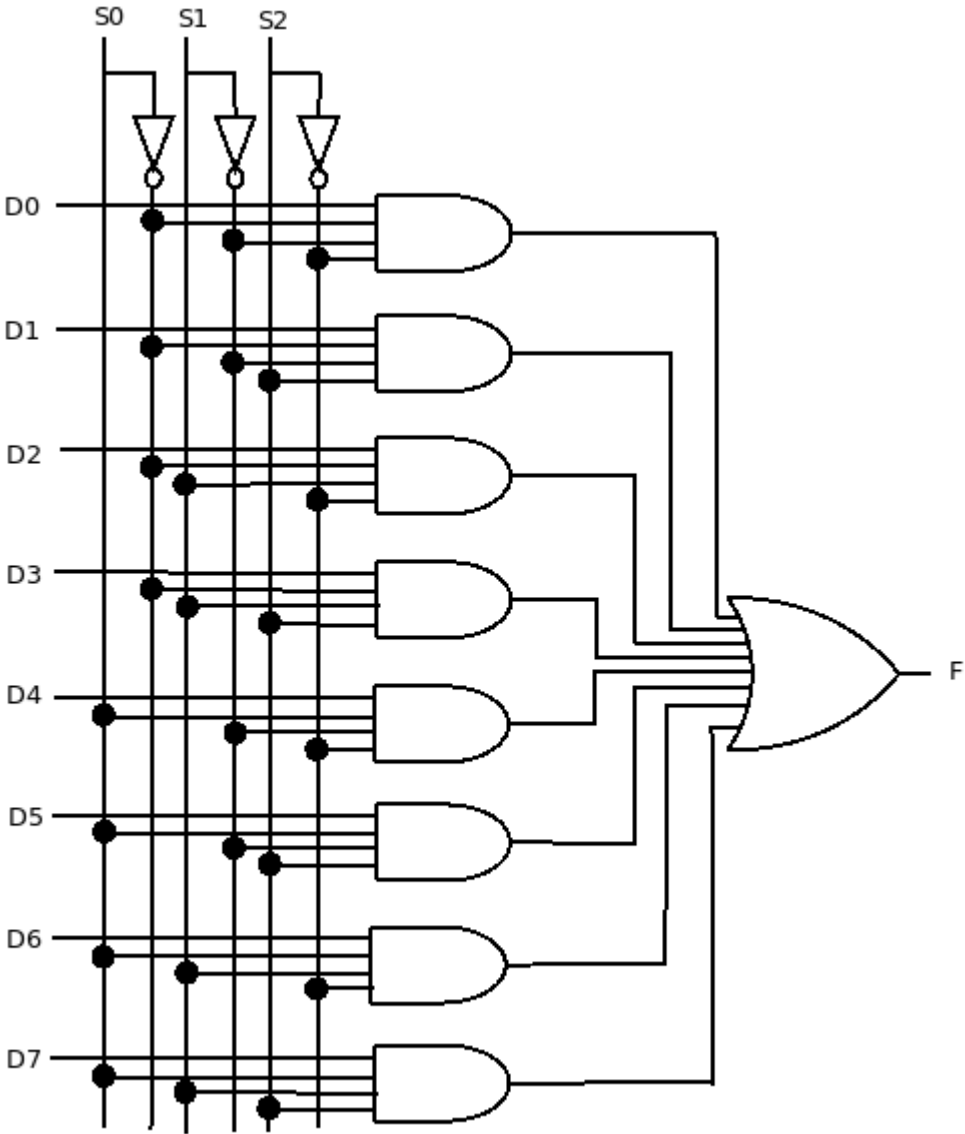


Figure 5.10: Implementation of 8-to-1 multiplexer

5.3.6 Read Only Memory

The read only memory (ROM) is a kind of memory whose contents cannot be changed. The output of ROM depends on the selected memory cell i.e. input address only. Therefore, it can be implemented as a combinational circuit. Although, combinational circuits themselves do not have any memory. This is special case of combinational circuit otherwise, they do not provide memory as a combinational circuit ROM is implemented using decoder and OR gates. However, not all kind of ROMs are implemented as combinational circuit.

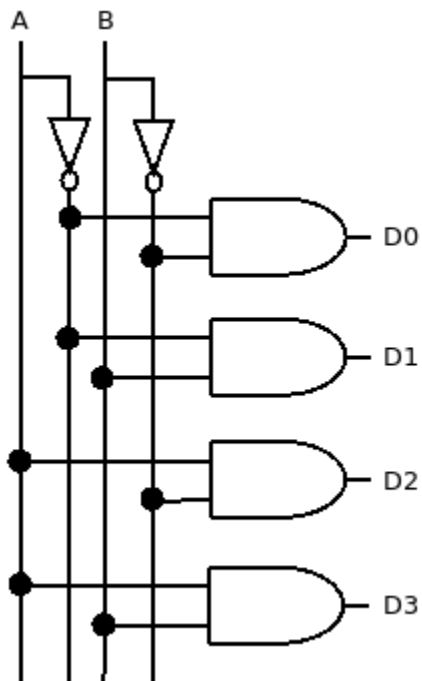


Figure 5.11: Implementation of decoder with 2 inputs and 4 outputs

Table 5.6: Truth table for 2-to-4 decoder

A	B	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Check your progress 1

1. What is truth table?
2. Which logic gates are known as universal gates?
3. What is combinational circuits?
4. What do you understand by SOP and POS?
5. How Boolean expressions are simplified?
6. How many outputs are generated by a half adder?
7. How many half adders are required to make a full adder?
8. How many inputs are taken by a full adder?
9. What is multiplexer?
10. Can you convert a decoder into demultiplexer?

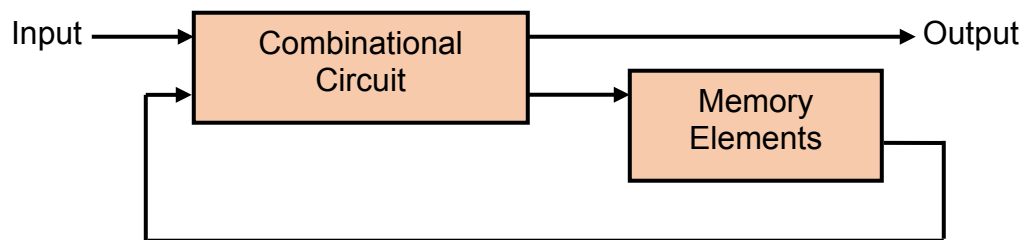


Figure 5.12: Block diagram of sequential circuit

5.4 Sequential Circuits

The sequential circuit is capable of storing the information. The output of sequential circuits not only depends on the current input but on the present state also. A combinational circuit when included with a memory element, the overall circuit is called sequential circuit. It is illustrated with the help of a block diagram in Figure 5.12. The memory element defines the present state of the circuit at any time. The memory element in sequential circuit is connected to combinational circuit through a feedback connection. As compared to the combinational circuits, the sequential circuits are complex to implement. The present state and current input determines the output of the circuit and its next state. The behaviour of a sequential circuit is described by inputs,

outputs, and its states. It is specified with the help of a table known as **state table**, which provides output and next state of the circuit as a function of corresponding input and present state.

5.4.1 Latches and Flip Flops

Latches and flip flops are the basic sequential circuits, which can store a bit of information. The primary function of these circuits is storage and they are capable of storing 1-bit of binary data (0 or 1). The latches or flip flops are controlled by an enabling signal. The major difference between latches and flip flops is that the output of a latch changes according to input immediately as control signal is enabled. On the other hand output of a flip flop changes only at the rise or fall of the enabling signal. After the rise or fall of the enabling signal, the output of the flip flop remains stable even if the input changes. A flip flop has two stable states, which can be treated as 0 or 1 depending on the absence/presence of signals. The flip flop can change its state only during the clock transitions. There are many different types of flip flops. The behaviour of a latch or flip flop can be described with the help of a **characteristics table**, which is also known as **function table**. The characteristics table is similar to truth table.

S-R Latch also known as set-reset latch is a sequential logic circuit that can be constructed with the help of NAND or NOR gates. It has two inputs called set (S) and reset (R). There are two output ports Q and Q'. In fact, Q' is the complement of Q. Figure 5.13 shows implementation of S-R latch using two NOR gates and its state table is given in Table 5.7. The two gates are connected using feedback connections. There are four states of S-R latch as given in its characteristics table. If S=1 and R=0, the output Q=1 and the latch is in SET state. When S=0 and R=0 the output Q is still 1 as circuit remembers the previous state and it works as a memory element. If S=0 and R=1, the output Q=0 and the latch is in RESET state. When S=1 and R=1, the circuit enters to an invalid state as Q=0 and Q'= 0 but both Q and Q' must complement each other. Therefore, both S and R are not allowed to have input value as 1 at the same time.

S-R latch can also be implemented by connecting two NAND gates in a feedback arrangement as shown in Figure 5.14. The arrangement is similar to the circuit shown in

Figure 5.13 but NOR gates are replaced by NAND gates. The characteristics table is given in Table 5.8. The NAND gate implementation works in the similar way with slightly different states. As it can be observed from the characteristics table, the inputs $S=1$ and $R=1$ representing a memory state, when $S=0$ and $R=0$ it is an invalid state.

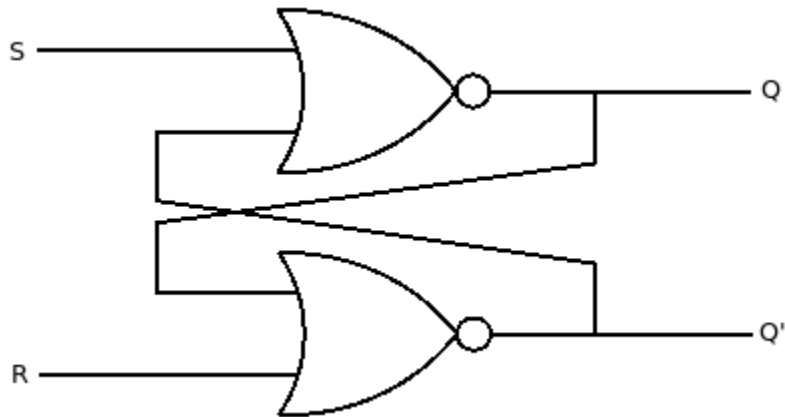


Figure 5.13: S-R latch implementation using two NOR gates

Table 5.7: Characteristics table for S-R latch using NOR

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

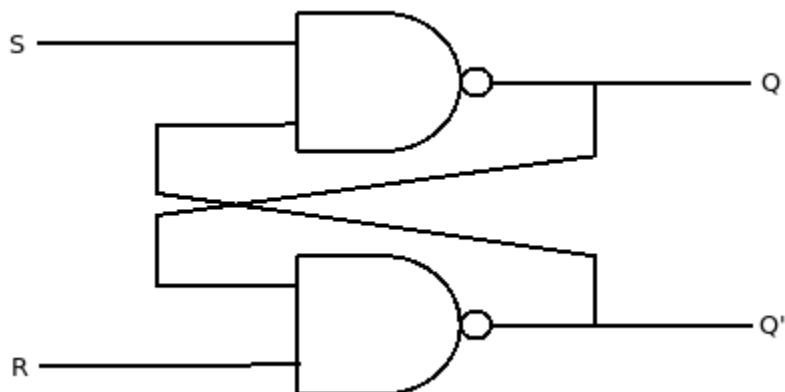


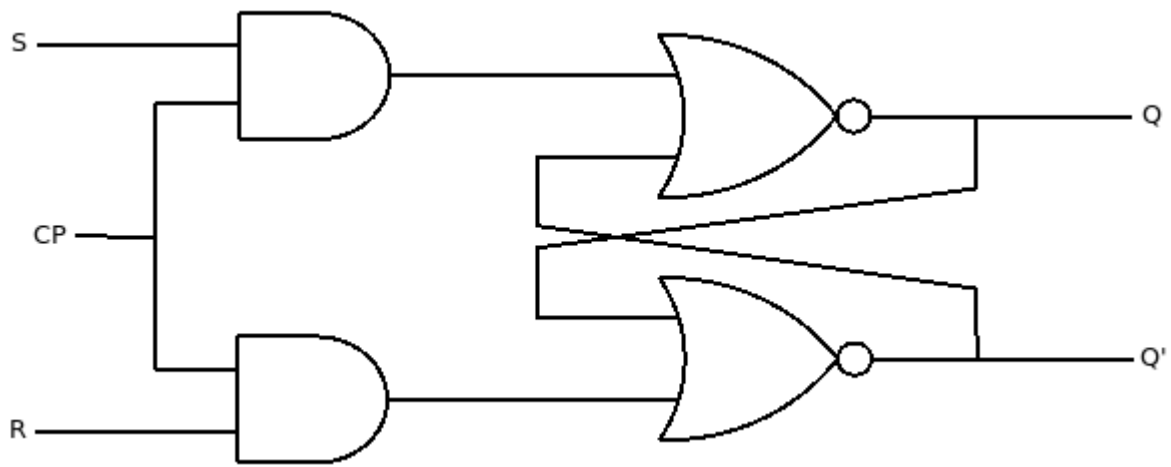
Figure 5.14: S-R latch implementation using two NAND gates

Table 5.8: Characteristics table for S-R latch using NAND

S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

The S-R latch with NAND or NOR gates has a problem of invalid states. Apart from that output changes in response to input after some delay. This type of operation is called as asynchronous operation. The circuit can be made more stable and synchronous by adding a clock as another input to the circuit. This type of circuit is known as flip flop. Flip flop allows to change state in response to the change in S and R values only during clock transitions.

Clocked S-R Flip Flop: A clocked S-R flip flop is designed to overcome limitations of S-R latch. It is also known as gated S-R flip flop. It has the ability to change the output when certain invalid state is reached irrespective of the inputs. The clocked S-R flip flop can be developed by adding two AND gates and a clock to the basic NOR latch circuit as shown in Figure 5.15. The outputs of the two AND gates remain 0 regardless the values of S and R as long as clock pulse is 0. When clock pulse becomes 1, the input from S and R passes through to the NOR gates in the circuit.



CP: Clock pulse

Figure 5.15: Clocked S-R flip flop

Table 5.9: Characteristics table for clocked S-R flip flop

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Intermediate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Intermediate

The characteristics table for clocked S-R flip flop is given in Table 5.9. If $S=1$ and $R=1$, both outputs go to 0 for a short time as clock pulse occurs. When clock pulse is removed, the flip flop state is intermediate. The intermediate state could result in either 0 or 1 depending on the transition.

D Flip Flop: A minor modification is made to the circuit of clocked S-R flip flop to avoid intermediate state problem. The modified circuit is shown in Figure 5.16. It is known as D flip flop. The D input directly takes the S input and its complement goes to R input. The D flip flop is considered a data flip flop as it provides storage for 1 bit. It always produces the last input. The characteristics table of D flip flop is given in Table 5.10.

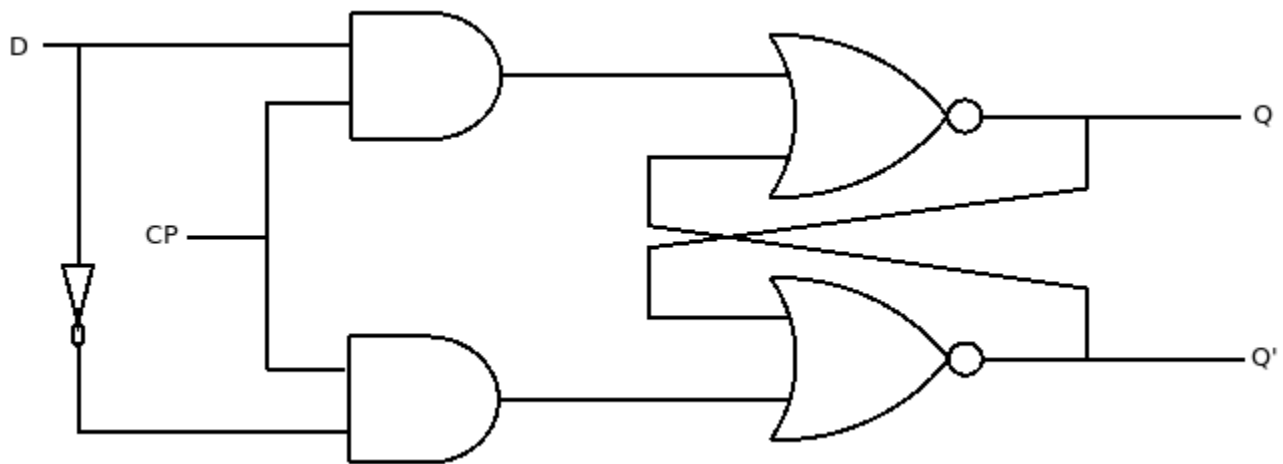


Figure 5.14: D flip flop

Table 5.10: Characteristics table for D flip flop

Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

J-K Flip Flop: The major problem with S-R flip flop is its intermediate states. J-K flip flop is an important modification of S-R flip flop whose all states are valid. It is the most

commonly used flip flop. Similar to S-R flip flop, J-K flip flop also consists of two input ports designated as J (SET) and K (RESET) and a clock pulse input. The fundamental difference between two flip flops is the feedback connections to AND gates of the inputs. The circuit diagram is shown in Figure 5.17 and the characteristics table is given by Table 5.11.

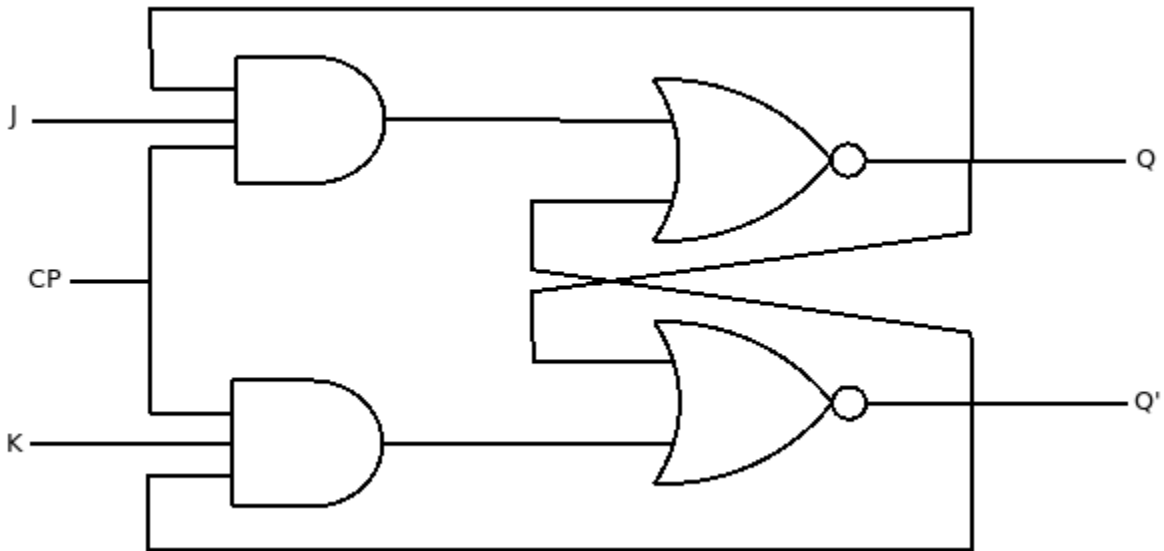


Figure 5.17: J-K Flip flop

Table 5.11: Characteristics table for clocked J-K flip flop

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

T Flip Flop The T flip flop is a simplified version of the J-K flip flop, It is constructed by connecting inputs of J-K flip flop together as shown in Figure 5.18. It is

also known as single input J-K flip flop. The output of T flip flop simply toggles after each pulse. The characteristics table of the T flip flop is given by Table 5.12.

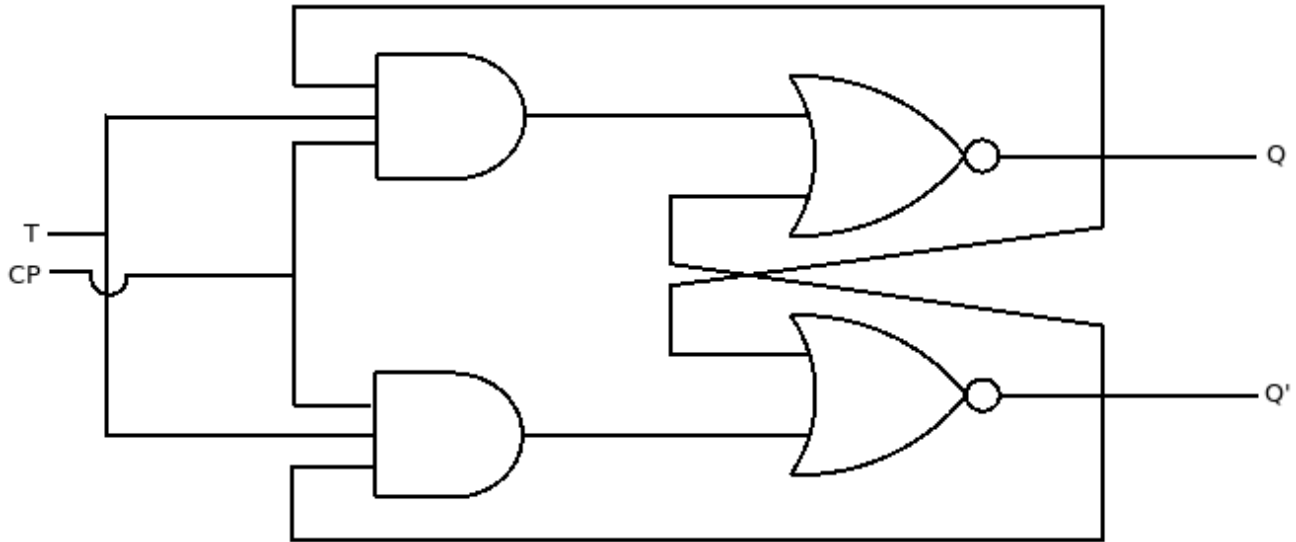


Figure 5.18: T flip flop

Table 5.12: Characteristics table for T flip flop

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

5.4.2 Registers

Registers are the essential elements of a CPU. The primary function of registers is data storage. A register can be implemented as a chain of n flip flops with common control signals. Each flip flop stores 1 bit information, therefore a register can store n bits of binary data. In addition to the flip flops, some combinational gates may also be used in a register to perform some kind of data processing. The logic gates in the register control transfer of data. The data transfer to a register is called **loading** the register.

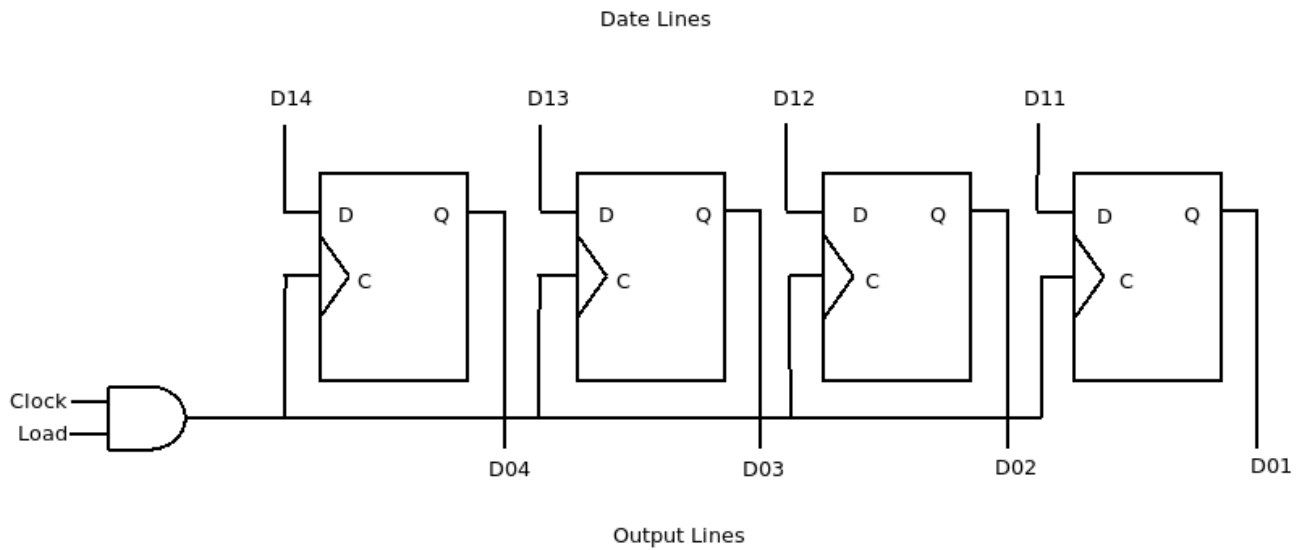


Figure 5.19: 4-bit parallel register using D flip flops

There are two basic types of registers namely **parallel registers** and **shift registers**. In parallel register, all 1-bit memories can be loaded simultaneously. The circuit diagram for an 8-bit parallel register is shown in Figure 5.19. It uses a clock to supply clock pulses continuously to the system. The clock pulse determines the next of the output. A control signal is also used in the register to decide the impact of clock pulse on a particular register. With a single clock pulse, all the bits of a parallel register can be loaded simultaneously.

The shift register has the ability to transfer its data bits to left or right or both directions. The flip flops in a shift register are connected in a cascade. The output of one flip flop is connected to the input of the next flip flop in the chain. The common clock pulse is used to initiate the shift operation in the register. The circuit diagram of a shift register implemented using D flip flops is shown in Figure 5.20. In this register, the input goes into the leftmost flip flop and the output is taken from the rightmost flip flop. The shift registers have a range of applications in ALUs, I/O devices, and interfaces, etc.

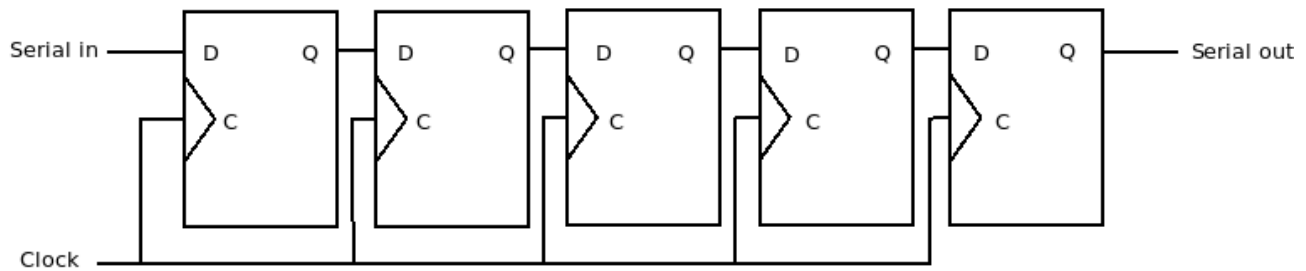


Figure 5.20: Implementation of a 5-bit shift register using D flip flops

Check your progress 2

1. Differentiate combinational and sequential circuits.
2. Write some examples of sequential circuits.
3. How many stable states a flip flop can have?
4. Differentiate S-R flip flop and Clocked S-R flip flop.
5. Which logic gates can be used to implement S-R flip flop?
6. Which flip flop is known as data flip flop?

5.4.3 Counters

Counters are another example of sequential circuit, which are used in many digital circuits to count the occurrence of an event in the system. An n -bit counter is a kind of register constructed with the help of n flip flops and associated logic gates. An n -bit counter can be used to count from 0 to $2^n - 1$. The program counter in CPU is a good example of counter. A counter is usually implemented with complementing flip flops such as T flip flop or J-K flip flop. There could be two types of counters: **synchronous** or **asynchronous**. In a synchronous counter, all the flip flops can change their states simultaneously. On the other hand, in an asynchronous counter, the output of one flip flop triggers the change in the state of next flip flop. Therefore, asynchronous counters are relatively slower than the synchronous counters.

The asynchronous counters are also known as **ripple counter** as change at one end ripples through to the other end when there is an increment in the counter. An implementation 4-bit asynchronous counter is shown in Figure 5.21 using 4 J-K flip

flops. The output of the leftmost flip flop provides the least significant bit. The counter is incremented with each clock pulse. The concept can be generalized to n -bit counter by using n J-K flip flops.

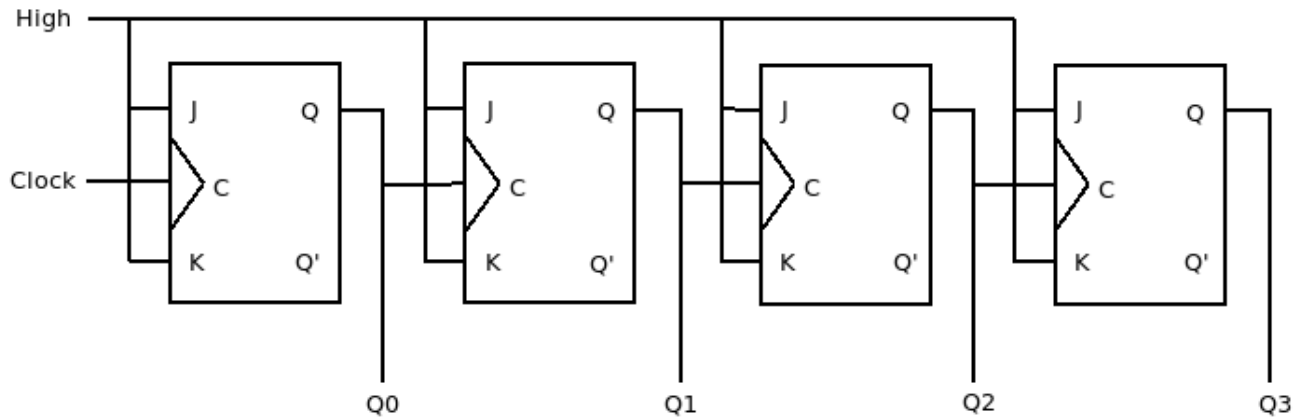


Figure 5.21: 4-bit Asynchronous counter

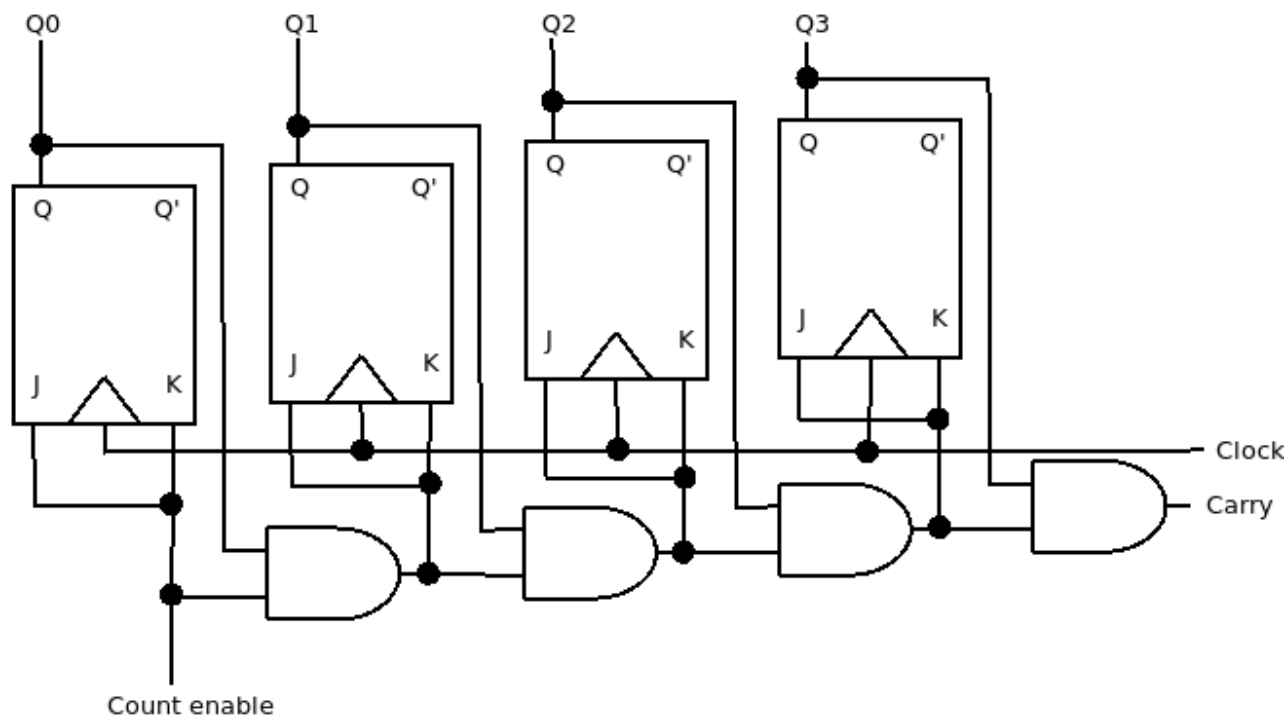


Figure 5.22: 4-bit synchronous counter

The asynchronous counter introduces a delay in changing the value, which is proportional to the counter length. To overcome the problem of delay, usually synchronous counters are preferred over the asynchronous counters. The C inputs of all flip-flops in a synchronous counter receive a common clock pulse as shown in Figure 5.22 and all flip-flops change at the same time.

5.5 Summary

ALU is one of the most important components of the microprocessor, which performs arithmetic and logic operations. The logic gates are the basic building blocks of the ALU. The logic gates can carry out Boolean operations. Different computing elements can be constructed with the help of a circuit of logic gates to perform different arithmetic operations. There are two different types of digital circuits: combinational and sequential. The output of a combinational circuit is the function of its inputs only at a particular time. Any change in the inputs, immediately reflected to its output. Half adder, full adder, multiplexer, decoder, and ROM are the examples of combinational circuits.

Q.5 Highlight the major differences between synchronous and asynchronous counters.

Q.6 Draw a logic diagram for an 8-bit parallel shift register using D flip flops.

Q.7 What are the major differences between a latch and a flip flop?

Q.8 Compare the characteristics of S-R flip flop and J-K flip flop.

UNIT- 6 Advanced Topics

Structure

6.0 Introduction

6.1 Objectives

6.2 Pipeline Processing

6.3 Floating-Point Arithmetic

6.4 Summary

Review Questions

Unit 6: Advanced Topics

6.0 Introduction

The fixed-point numbers can be processed efficiently but only a small range of numbers can be provided with fixed-point representation. The arithmetic operations on fixed-point numbers can be performed in a faster and simpler way. However, it is not possible to represent very large numbers and very small fractions with this notation. The floating-point notation can represent a large range of negative and positive numbers. A number is represented in three parts: sign, mantissa (or significand), and exponent. All parts are stored in two different registers.

IEEE has developed three standards of floating-point representations known as: single precision, double precision, and extended precision. In single precision notation, mantissa has 23 bits, exponent takes 8 bits, and remaining 1 bit is used for the sign of the number. The number is normalized so that the mantissa is always 1.xxxxxx...xxx in the normalized form. Because leading bit is always 1, it is not stored but it is an implied bit. Effectively the mantissa part has 24 bits. For zero, a special representation is used with all 0's both for mantissa and exponent fields as well as sign bit. IEEE standard for floating-point numbers uses a biased representation, which is simply the binary representation of $E+127$, where E is the actual exponent and 127 is the exponent bias. The actual exponent is obtained by subtracting the exponent bias from the stored exponent. Mathematically, single precision representation gives $(-1^S .2^{exp-127} .1. frac)$, where S is sign, exp is exponent, and $frac$ is mantissa. The dynamic range of single precision floating-point numbers goes from 1.175×10^{-38} to 3.4×10^{38} . The double precision numbers use 64-bit words but arrangement is similar to the single precision floating point numbers. The mantissa takes 52 bits, 11 bits are used for exponent, and remaining 1 bit is used as sign bit. The dynamic range of double precision numbers varies from 2.2×10^{-308} to 1.7×10^{308} . The extended precision floating point notation uses 80-bit word out of which, 15 bits are used as exponent, 64 bits are reserved for mantissa, and 1-bit is used as sign bit.

For large numbers floating-point representation is used. However, it is more complex to perform arithmetic operations on floating-point numbers. These operations require more complex hardware and take longer execution time. The two operands may have different exponents that causes problem especially for addition and subtraction operations. On the other hand, there may arise some kind of overflow and underflow conditions that need to handle in such operations.

The complex operations can be divided into several sub-operations that can be executed in separate hardware segments. The pipelining is a technique that executes sub-operations belonging to the same operation in concurrently operating segments simultaneously. Each segment performs partial processing and output is transferred to the next segment for further processing. The final result is obtained through the last segment in the cascade. The overlapping of computation leads to the faster execution.

6.1 Objectives

The objectives of this unit are as follows:

1. To introduce the concept of computational pipelining.
2. To study the techniques for floating-point addition/subtraction.
3. To study the techniques for floating-point multiplication/division.

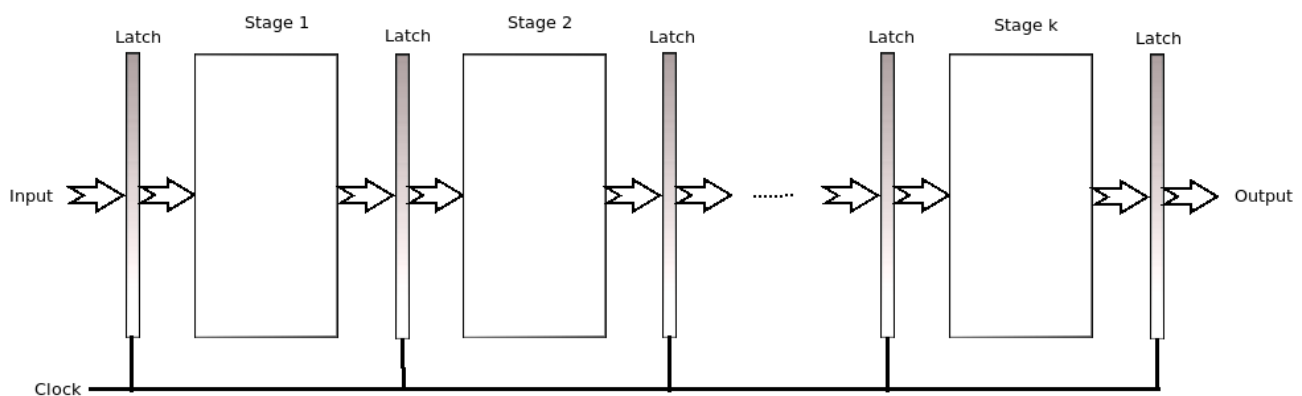


Figure 6.1: A k-stage linear pipeline structure

6.2 Pipeline Processing

Pipeline processing is similar to the assembly line in manufacturing plant, where computational processing goes through different phases and each phase is carried out by different computing segments. In computational pipeline, the operands are processed by one segment and the intermediate result so obtained is transferred to the next segment in the pipeline. In this way, the data elements are processed by each segment before the final result is obtained through the last segment of such pipeline. The computations belonging to different data elements can be overlapped in a pipeline processing leading to a faster execution. The structure of a basic linear pipeline processor is shown in Figure 6.1. Each segment in the pipeline is separated by a latch that works as an interface between the two stages. Different segments may introduce different amount of delay as each of them performs a different type of sub-operation. The latch helps to manage the delay to avoid any kind of data conflicts at a segment.

The overlapping of computations at different segments can be illustrated with the help of a space-time diagram for a four-segment pipeline as shown in the Figure 6.2. Each marked cell in the diagram represents the utilization of corresponding segment in a particular clock cycle. During the first clock cycle, only the first segment S_1 is busy with task T_1 . After the first clock cycle, the output of segment S_1 for task T_1 is transferred to second segment S_2 and a new task T_2 is assigned to segment S_1 in second clock cycle. In this way the computation makes progress and the first result is obtained after four clock cycles. But after that once the pipeline is filled up with tasks, it produces the results per clock cycle. In general, a k -segment pipeline takes k clock cycles for the first result. If a total number of n tasks are processed with the pipeline processor, it takes $T_p = k+n-1$ clock cycles to complete the tasks. On the other hand a non-pipeline processor takes $T_{np} = k.n$ clock cycles to complete the same number of tasks. Therefore the speed up provided by a k -segment is

$$S = \frac{T_{np}}{T_p} = \frac{k.n}{k + n - 1}$$

If $n \gg k$, the speed up for a k -segment pipeline is approximately k times. The pipeline organization can be applied to perform arithmetic operations.

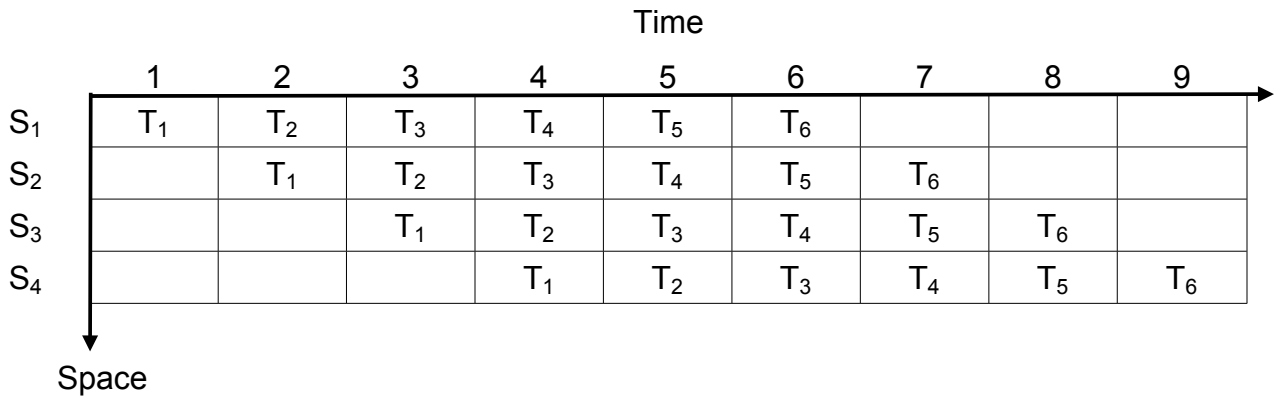


Figure 6.2: Space-Time diagram for a four-segment pipeline

The other parameters that can be used to analyse the performance of a pipeline are throughput and efficiency. The efficiency (E) of k -stage pipeline can be defined as follows.

$$E = \frac{n}{k + n - 1}$$

On the other hand, the throughput (T) is defined as follows.

$$T = \frac{E}{\tau}$$

where τ is the clock period.

Check your progress 1

1. Which three standards are developed by IEEE for floating-point numbers?
2. Write the dynamic range of single precision floating-point numbers.
3. How many bits are used for mantissa in double precision floating-point numbers?
4. What is the role of latch in pipelining?
5. How much speed up can be achieved by a k -stage pipeline?

6. Write equation to determine efficiency of a pipeline.
7. How throughput is related to the efficiency of a pipeline?

6.3 Floating-Point Arithmetic

The floating-point arithmetic operations are more complex to perform than fixed-point arithmetic operations. There are certain issues that arise during floating-point arithmetic operations as discussed here.

Overflow conditions: As discussed earlier, exponent and mantissa are two major parts in the floating-point number representation. As a result of an operation, the exponent of the resultant number may exceed the maximum possible positive value. Similarly, when the two mantissas of the same sign are added, it may exceed the maximum allowed value. The overflow conditions need to be fixed in such operations.

Underflow conditions: If two negative exponents are added, it may result in a negative exponent that is lower than the minimum allowed exponent. Such numbers are not possible to represent. In the same way, the mantissa may also result in the underflow condition. Usually the underflow conditions are handled by some kind of rounding.

6.3.1 Floating-point Addition and Subtraction

The floating-point addition or subtraction operation takes two normalized floating-point numbers P and Q as input. The numbers P and Q can be written as follows

$$P = A \times 2^p \text{ and } Q = B \times 2^q$$

In the above notation, A and B are the mantissas and p and q are exponents. The addition/subtraction operations are performed as four sub-operations. The four sub-operations are as follows

1. Compare the exponents
2. Align the mantissa of the operand with smaller exponent value. If two exponent values are equal then no alignment required
3. Add or subtract the mantissas

4. Normalize the result by shifting the resulting mantissa and adjusting resulting exponent

The main difficulty with the addition/subtraction operation is that the exponents of the two operands may be different i.e. $p \neq q$. The larger exponent is the exponent of the result. Therefore, both exponents are compared and if p and q are found different then the mantissa of the operand with lower exponent value is adjusted to make both the exponents equal. In order to adjust the mantissa, it is shifted right by $(p - q)$ places. After the addition of the two mantissas, the overflow/underflow conditions are handled and result is normalized.

The sequence of four sub-operations in an addition operation can be illustrated with the help of an example. To keep it simple, the base 10 scientific notation is used in the example. Let us consider the addition of 997.6 and 4.3. The two operands can be written as normalized numbers as follows

$$P = 0.9976 \times 10^3 \text{ and } Q = 0.4300 \times 10^1$$

As a first sub-operation, the two exponents are compared and it is found that the difference in two exponents is 2. Both exponents are made equal by shifting the base point of the mantissa of the operand with smaller exponent. It is shifted by 2 places to right to align the two mantissas under the same exponents

$$P = 0.9976 \times 10^3 \text{ and } Q = 0.0043 \times 10^3$$

Now the two aligned mantissas are added to produce the sum

$$S = 1.0019 \times 10^3$$

The sum is normalized to obtain the final result

$$S = 0.10019 \times 10^4$$

Suppose that the mantissa can have maximum four digits. In that case, the result in the above example leads to the overflow condition as there are five digits in the resultant mantissa. To handle the overflow condition, the least significant digit is dropped by rounding off as follows

$$S = 0.1002 \times 10^4$$

Another example is given here for subtraction operation. The procedure for the subtraction operation is exactly same. Let us carry out $(857.64 - 697.58)$. The two operands can be written as

$$P = 0.85764 \times 10^3 \text{ and } Q = 0.69758 \times 10^3$$

On comparison, both exponents are found equal. Therefore no alignment is required. Subtract the smaller mantissa from the larger one to obtain the following result

$$S = 0.16006 \times 10^3$$

Here no issue related to overflow or underflow occurs. Therefore, it is the final result.

6.3.2 Hardware Implementation for Floating-Point Addition and Subtraction

The floating-point operations can be implemented both in software and hardware. For software implementation the compiler is designed with subroutines for such operations. The hardware implementation is expensive on the other hand, but it is more efficient than software implementation. The most of the computer systems have hardware for floating-point arithmetic operations in their ALUs. A set of registers and adders are required in ALU for floating-point operations. The algorithm for addition/subtraction operation can be written as follows

1. Check for zeros
2. Compare and align mantissas
3. Add or subtract the mantissas
4. Normalize the result

The zero value of the operands may cause problem during the computation. Therefore, it is a good practice to check for zeros in the beginning. The zero cannot be normalized. If only one operand is zero then the other one is provided as the result. In case of subtraction operation, the sign of the subtrahend is changed. In some situations the result is zero and it is better to terminate the process in the beginning at appropriate stage. The two operands are aligned to make exponents equal. It is achieved by repeatedly shifting the mantissa to right and incrementing the exponent until both exponents become equal. If mantissa becomes zero during this process, the other operand is reported as a result. For larger numbers, the shifting could lead to the loss of

bits. Since it is the least significant bit that is discarded, the loss is small. After the addition/subtraction the result is normalized by shifting the bits before transferring the result to the memory. The result could be un-normalized before providing it to the user. The shift operation may lead to overflow or underflow conditions that should be handled effectively. The complete procedure is shown with the help of a flow chart in Figure 6.3.

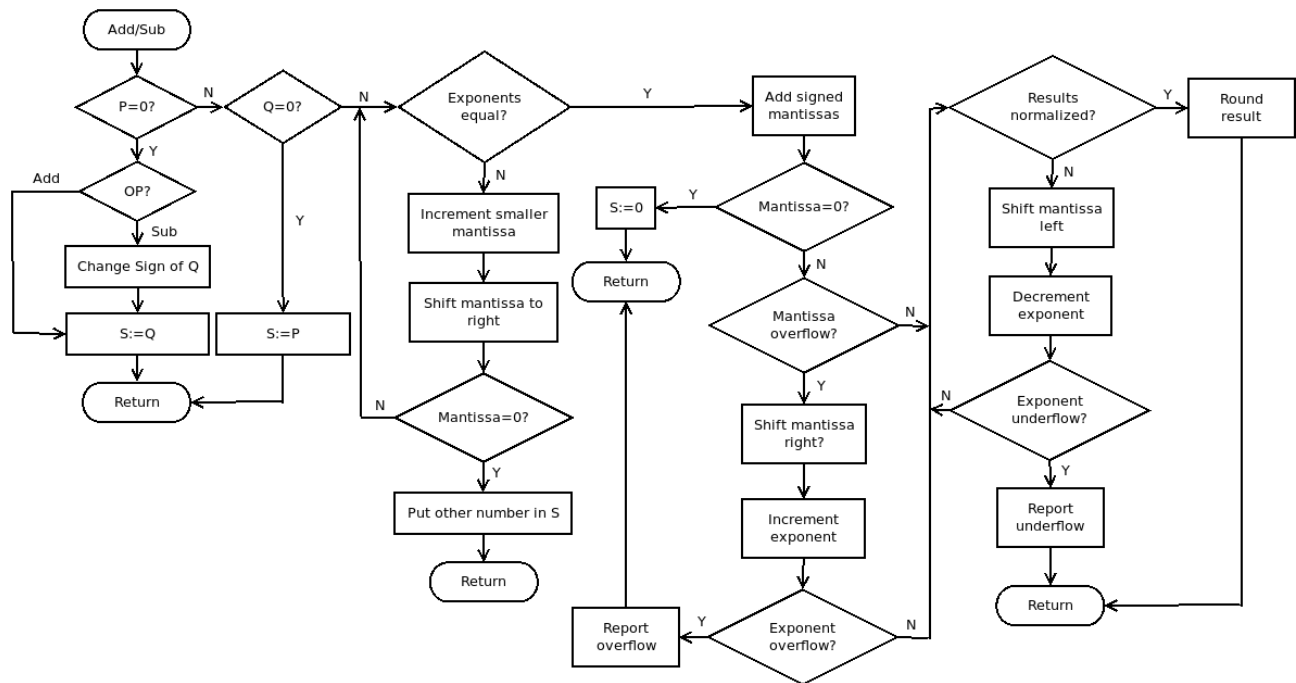


Figure 6.3: Floating-point addition and subtraction

The procedure of floating-point addition/subtraction operations can be divided into four sub-operations. Each sub-operation can be carried out by a separate segment. Therefore, it is possible to implement a pipeline to perform these operations. A pipelined implementation for floating-point addition and subtraction operations is shown in Figure 6.4. In the figure, letters *A* and *B* denote the mantissas of two operands and *a* and *b* are corresponding exponents respectively. There are four segments in the pipeline. The first segment named Segment1 compares the two exponents. The second segment (Segment2) chooses the larger exponent, addition/subtraction of mantissas is performed by third segment (Segment3), and finally result is normalized in fourth segment (Segment4). The addition/subtraction of two mantissas is equivalent to the

addition/subtraction of fixed-point numbers. The overflow and underflow conditions are handled by right and left shift operations respectively. All the segments are separated by a register R , which is used to hold any intermediate results.

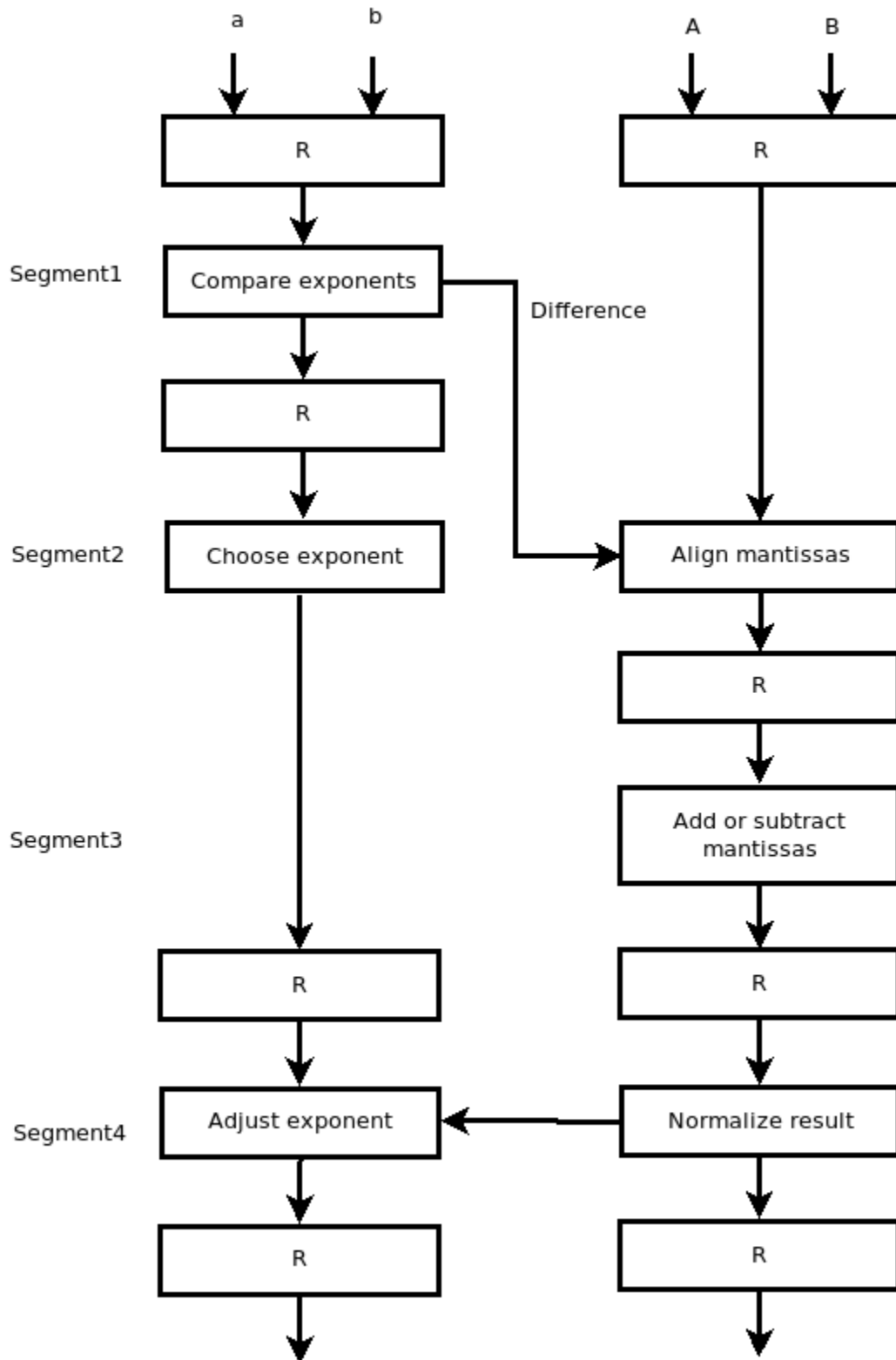


Figure 6.4: Floating-point addition and subtraction by pipeline

6.3.3 Floating-point multiplication

The floating-point multiplication and division operations are simpler than floating-point addition and subtraction operations as no comparison of exponents and alignment of mantissas are necessary. To perform the multiplication of two floating-point numbers, the mantissas are simply multiplied and exponents are added. The multiplication of mantissas is carried out in the same way as the multiplication of two fixed-point numbers.

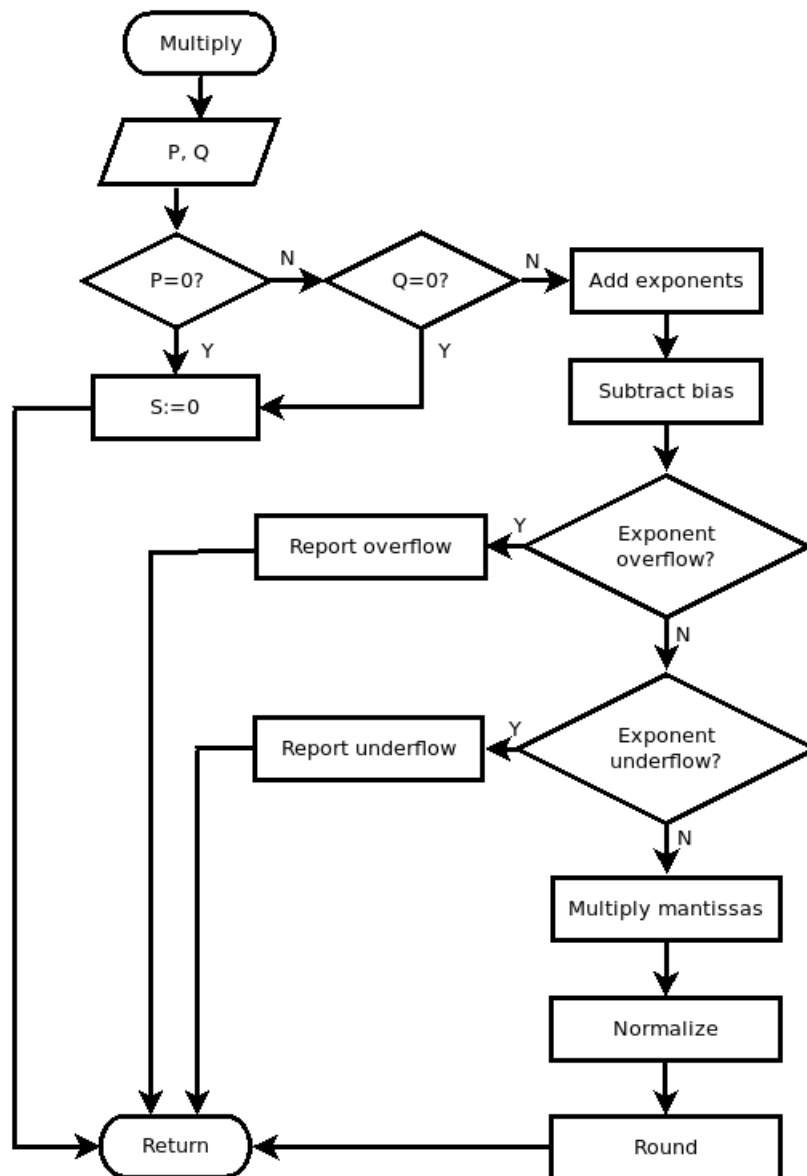


Figure 6.5: Flow chart for floating-point multiplication

The multiplication operation can be carried out as sequence of four sub-operations as follows:

1. Check for zeros
2. Add the exponents
3. Check for overflow/underflow
4. Multiply the mantissas
5. Normalize the result

First of all, the two numbers suppose P and Q are checked for zeros. If any one of the two numbers is found zero then result is zero and no further processing is required. Otherwise the remaining three steps of the algorithm are also executed. The step 2 and step 3 can be performed simultaneously if required hardware support is available. The representation of the number is an important concern as the length of the result would be double of the word length of the multiplier/multiplicand. If exponents are in biased form then addition of exponents would result in double bias. Therefore, bias value is subtracted from the sum to maintain the proper bias (Refer Unit 2, Section 2.4). Any overflow or underflow conditions would be reported. Any extra bits would be lost during the rounding process. The floating-point multiplication process is illustrated with the help of a flow chart in Figure 6.5.

6.3.4 Floating-point division

The floating-point division operation has many similarities with floating-point multiplication. Here, the two exponents are subtracted instead of addition and mantissas are divided. The division is carried out as a fixed-point operation. The step-by-step procedure is given as follows:

1. Check for zeros
2. Subtract the exponents

3. Check for overflow/underflow
4. Divide the mantissas
5. Normalize the result

The process is illustrated in Figure 6.6. As in case of multiplication operation, the two operands are checked for zero. If dividend is zero, the result is also zero and procedure terminates here itself. If divisor is zero, it leads to undetermined condition, which is reported as infinity or as an error condition.

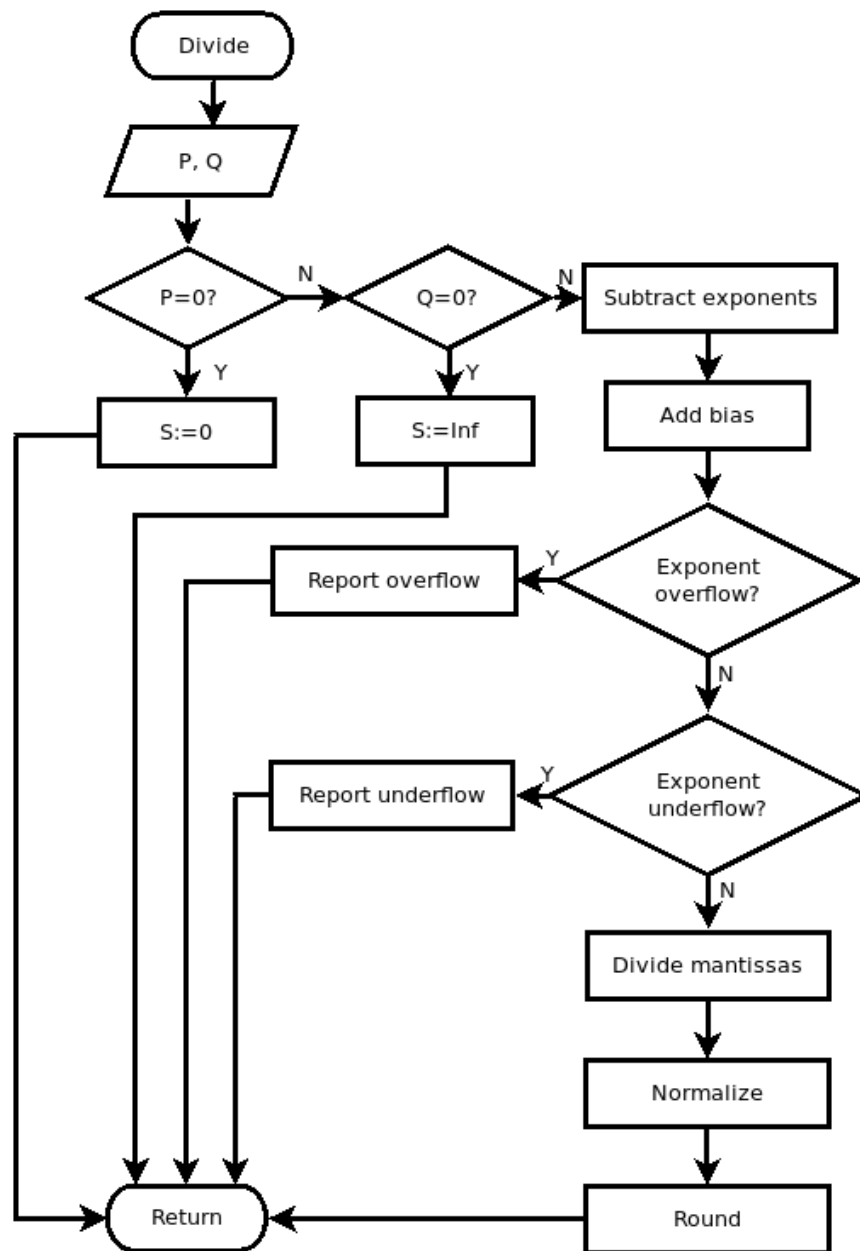


Figure 6.6: Flow chart for floating-point division

After testing the operands for zero value, exponents are subtracted if operands found non-zero. In case of bias representation, subtraction operation removes the bias. Therefore, it is added back in. After that overflow/underflow conditions are tested. The next step is to divide the mantissas followed by the normalization and rounding of the result.

Check your progress 2

1. What is overflow in floating-point arithmetic?
2. What is underflow in floating-point arithmetic?
3. What types of problems can be caused by zero operand value?
4. Which shift type is used to handle overflow condition?
5. Which shift type is used to handle underflow condition?
6. How undetermined condition is reported?

6.4 Summary

Floating-point representation is used for the large numbers. There are two major parts of the number in this representation: mantissa and exponent. The arithmetic operations on floating-point numbers are more complex to perform. These operations can be divided into multiple sub-operations, which could be implemented either in software or in hardware. The hardware implementation takes more cost but it is more efficient than software implementation. Each sub-operation can be performed by a separate segment in a pipeline approach. The output of one segment is provided as input to the next segment in the pipeline and the result is obtained through the last segment. Different segments can work in an overlapped fashion that speeds up the overall execution if there are large number of same operations.

The operands with different exponents have to be handled carefully in the floating-point addition/subtraction operations. This issue is handled by normalizing the operands. The normalization is done by making both the exponents equal to the larger

exponent and shifting the mantissa of the smaller operand to right by the number of places equal to the difference of the exponents. Thus larger exponent is the exponent of the result. Later, the mantissas are added and result is normalized. The floating-point multiplication/division operations are little simpler as normalization is not required for these operations. The exponent of the result is obtained adding both exponents in multiplication operation. In division operation, the exponents are subtracted to get exponent of the result. The mantissa for floating-point multiplication/division is obtained by multiplying/dividing the mantissas of operands. The multiplication/division of the mantissas is simply a fixed-point operation. During the normalization and rounding process, some least significant bits may be lost. Losing a few least significant bits does not make large impact on the result.

Review Questions

- Q.1 What do you understand by pipeline processing? With the help of diagram, explain the pipeline processing technique and its impact on the speed up of the execution.
- Q.2 Write an algorithm to perform floating-point addition/subtraction algorithm and illustrate it with the help of a flow chart. Discuss the major issues that arise during these operations.
- Q.3 With the help of a diagram, discuss the pipeline implementation for performing the floating-point addition/subtraction operation.
- Q.4 Write an algorithm to perform floating-point multiplication operation and explain it with the help of a flow chart. Discuss the major issues with the operation.
- Q.5 Write an algorithm to perform floating-point division operation and explain it with the help of a flow chart. Discuss the major issues with the operation.

Q.6 Perform the following floating-point addition operations. Use 7-bit word including sign bit for to accommodate the operands and the result. Find out if there is any overflow/underflow condition and explain how to handle it.

(a) $(+49) + (+25)$

(b) $(+26) + (+37)$

(c) $(+49) + (-25)$

(d) $(-49) + (-25)$

Q.7 Perform the following floating-point subtraction operations. Use 7-bit word including sign bit for to accommodate the operands and the result. Find out if there is any overflow/underflow condition and explain how to handle it.

(a) $(+49) - (+25)$

(b) $(+26) - (+37)$

(c) $(-49) - (+25)$

(d) $(-49) - (-25)$

Q.8 Show the step-by-step procedure with flow-chart to perform following operations:

(a) 485×254

(b) $485 \div 26$

Q.9 Perform floating-point addition and show the results in normalized form.

(a) $4.5257 \times 10^2 + 7.8148 \times 10^3$

(b) $2.6842 \times 10^2 + 9.5247 \times 10^2$

(c) $3.7246 \times 10^2 + 5.2354 \times 10^4$

(d) $3.4568 \times 10^2 + 2.5246 \times 10^{-1}$

Q.10 Perform floating-point subtraction and show the results in normalized form.

(a) $4.5257 \times 10^3 - 7.8148 \times 10^2$

(b) $9.5247 \times 10^2 - 2.6842 \times 10^2$

(c) $5.2354 \times 10^{-3} - 3.7246 \times 10^{-2}$

(d) $5.2354 \times 10^3 - 3.7246 \times 10^{-2}$

Q.11 Show step-by-step procedure to perform floating-point multiplication:

(a) $(1.5000 \times 10^2) \times (2.5000 \times 10^2)$

(b) $(2.5577 \times 10^2) \times (4.2222 \times 10^0)$

(c) $(3.9999 \times 10^2) \times (1.5555 \times 10^3)$

Q.12 Show step-by-step procedure to perform floating-point division

(a) $(8.5684 \times 10^3) \div (2.5000 \times 10^2)$

(b) $(9.8645 \times 10^2) \div (3.4224 \times 10^3)$



Uttar Pradesh Rajarshi Tandon
Open University

Bachelor of Computer Application

BCA-EC
Computer Architecture

Block

3

CONTROL DESIGN

Unit 7

Basic Concepts

Unit 8

Micro-programmed Control

Unit 9

Pipeline Control

BLOCK INTRODUCTION

Arithmetic logic unit (ALU) and a control unit (CU) are two major components of the central processing. ALU deals with the arithmetic operations, while the primary function of the CU is to generate relevant timing and control signals. Block 3 deals with the organization of the CU and some advanced hardware design issues in a computer system. This block consists of Unit 7, Unit 8, and Unit 9. The Unit 7 provides insight into the major functions and organization of CU. Different approaches for CU implementation are discussed in Unit 8. It also discusses sequencing and execution of micro-operations and micro-instructions. The conventional computing systems perform operations in a sequential way. Parallel processing is the modern approach to simultaneously perform data processing to speed up the computational speed of the computer system. It is an efficient form of data processing that takes advantage of concurrent exploitation of resources. An economical realizing of the parallelism is the pipelining technique, which is discussed in Unit 9.

UNIT- 7 Control Design Basic Concepts

Structure

7.0 Introduction

7.1 Objectives

7.2 Micro-operations

7.3 Functional Requirements of Control Unit

7.4 Hardwired Control Unit

7.5 Control Unit Logic

7.6 Summary

Review Questions

Unit 7: Control Design Basic Concepts

7.0 Introduction

As it was discussed in Unit 1, the central processing unit (CPU) of the computer system has two major components: an arithmetic logic unit (ALU) and a control unit. ALU deals with the arithmetic operations, while the primary function of the control unit is to generate relevant timing and control signals for all kind of operations performed by the system. The computer performs operations by executing a program that is a sequence of instructions. However, the sequence of the instruction cycles executed by the computer is not exactly the same as written in the program due to branch and loop instructions. The instruction cycle is made up of a number of smaller steps involving the CPU registers. These steps are known as the **micro-operations**. A micro-operation performs a simple operation amounting to a little contribution. The responsibility of the control unit is ensure the execution of the proper sequence of the micro-operations based on the program under execution. The control unit also controls the flow of data between CPU and memory and I/O system. It issues the control signals to ALU, memory, storage devices, and peripherals. The control unit is responsible for data transfer to and from registers, ALU operations, and opening/closing of logic gates by issuing relevant control signals. Overall, the control unit coordinates the activities of different components in the computer system.

There are two major organizations for control unit to generate control signals: **hardwired control unit** and **micro-programmed control unit**. A hardwired control unit generates control signals with hardware implementation using logic circuits. The micro-programmed control unit uses a sequence of micro-operations to control the operations in a digital computer. The desired sequence of micro-operations is provided in the form of a **micro-program**. A control function of a micro-operation is specified by a binary variable. The execution of the micro-operation depends on the state of the associated binary variable. The control binary variables are stored in a control memory. Each word of the control memory contains micro-instructions. A micro-instructions specifies one or

more micro-operations. The sequence of micro-instructions is in fact makes a micro-program for the system. Usually the micro-programs are not required to change for an operation control unit. Therefore, control memory could be a read-only-memory (ROM). All micro-instructions and control variables are placed in the ROM. Each type of implementation of the control unit has some merits and demerits. The hardwired control unit is complex and difficult to implement but at the same time it is faster as control signals are generated by logic circuits. The micro-programmed control unit is less complex, flexible, easy to implement but slower.

7.1 Objectives

The major objectives of this unit are listed as follows:

1. To have an insight of the major functions and responsibilities of the control unit.
2. To have an overview of the different approaches for control unit implementations.
3. To understand the organization of the hardwired control unit.
4. To learn the major features of hardwired control unit.

7.2 Micro-operations

The major tasks in a digital computer system are carried out with the help of computer programs. A computer program is a sequence of instructions. The primary job of the CPU is to execute instructions or a sequence of instructions. Each instruction is made up of a number of sub-operations such as instruction fetch, operands fetch, decode, execute, and interrupt, etc. It is possible to further decompose the sub-operations as a sequence of smaller steps, which are known as micro-operations. A micro-operation performs an elementary job that makes a little contribution towards the accomplishment of a larger task. The relation among program, instructions, and micro-operations can be depicted with the help of a hierarchical structure as shown in Figure 7.1.

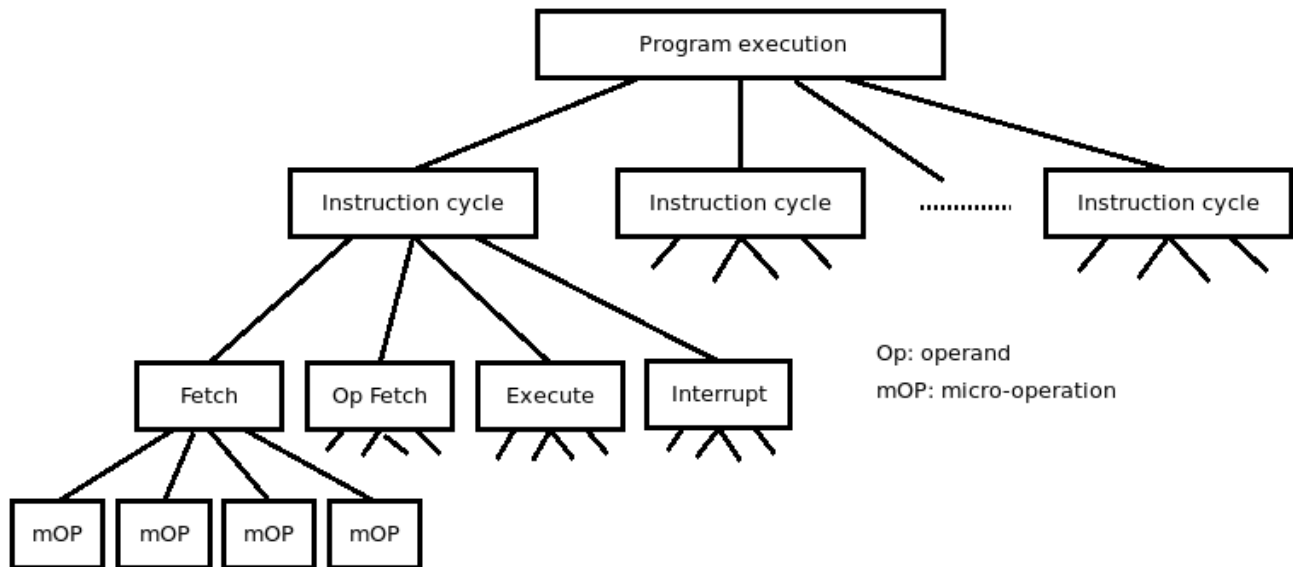


Figure 7.1: Program execution and its elements

An instruction is executed during one instruction cycle. Each instruction cycle is divided into a number of subcycles. During a subcycle usually only one micro-operation (sometimes more) is executed. Micro-operations involve some registers whose contents may be modified during the execution. The contents of one register may be transferred to another register replacing its previous contents. The CPU uses a sequence of micro-operations on the data stored in registers to carry out a specific function. The micro-operations are the functional operations of a processor. There are different types of micro-operations that perform different types of elementary jobs. The micro-operations can be classified into four broad categories: register transfer micro-operations, arithmetic micro-operations, logic micro-operations, and shift micro-operations.

7.2.1 Register Transfer Micro-Operations

The CPU of the computer system consists of a set of registers in addition to ALU and control unit. The registers are the fast memory elements that can store a small amount of data. Some elementary operations can be carried out on data contents of the registers with the help of available logic circuits. The most prominent register operations include data transfer operations that do transfer from one register to another. These operations are carried out through micro-operations that are usually performed through

hardware logic circuits. The information is not modified by these operations but it is transferred from one location to another. The symbolic notation used to express micro-operations for register transfer is called **register transfer language**. Symbolically, information transfer from one register to another is represented as follows

$$R1 \leftarrow R2$$

The above statement denotes the data transfer from register $R2$ to register $R1$. The symbol \leftarrow represents the data transfer, $R2$ is the source register, and $R1$ is the destination register. The existing contents of $R1$ are overwritten during this operation. The contents of the source register remain unchanged in transfer operation. All the statements written in the same line are executed simultaneously provided there is no conflict. The statements written in the same line are separated by a comma. For example, let us consider the following statements.

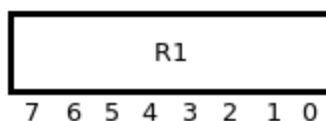
$$R1 \leftarrow R2, R1 \leftarrow R0$$

Here, both $R2$ and $R0$ are attempting to transfer data to register $R1$ leading to conflict. Therefore, these two micro-operations cannot be carried out simultaneously. On the other hand the following statements have no conflict.

$$R1 \leftarrow R2, R3 \leftarrow R0$$

Such micro-operations can be carried out at the same time as destination registers are different in both statements.

It is a standard notation to represent the processor registers by uppercase letters. The program counter register is designated as PC , address register is AR , instruction register is written as IR , and $R1$, $R2$, etc. are the general purpose registers. The individual bits of an n -bit register are designated 0 (rightmost) to $n-1$ (leftmost). The register IR is a 16-bit register that can be divided into two 8-bit sub fields $IR(L)$ and $IR(H)$. The symbol $IR(L)$ refers to bits 0 through 7 and bits 8 through 15 are assigned to $IR(H)$. The register formats are shown in Figure 7.2. It is a general practice to denote a register with uppercase letters and to represent it with the help of a rectangular box having register name written inside.



(a)



(b)

Figure 7.2: Register formats (a) 8-bit register (b) 16-bit register

In the statements discussed so far there is control information included but normally the register transfer operations take place under some predetermined control. The control function is specified with the help of a Boolean variable P placed in front of the actual statement as follows

$$P: R1 \leftarrow R2$$

The Boolean variable P can have a value 0 or 1. The associated statement is executed only if $P=1$. The clock is not mentioned explicitly in any forms of the statements but the register transfers are normally assumed to occur only at the transition of the clock edge as shown in Figure 7.3.

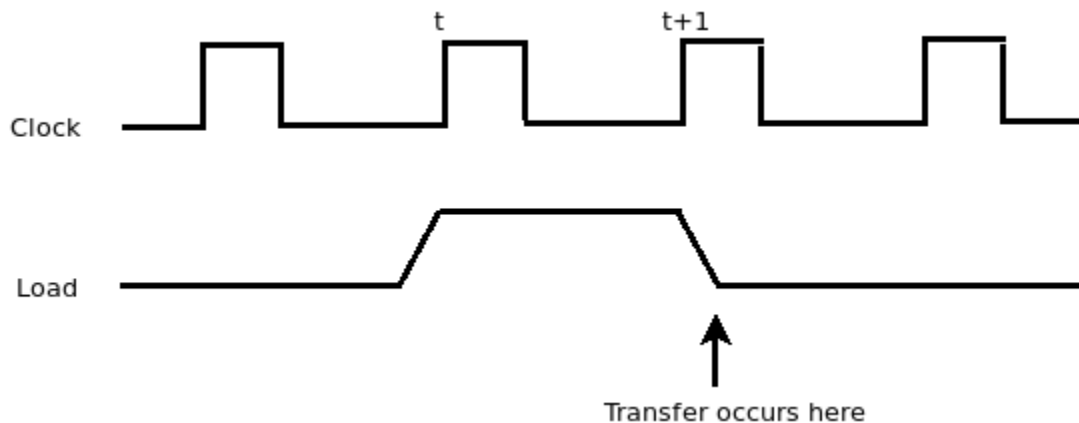


Figure 7.3: Timing diagram for register transfer

(Source: Computer System Architecture, 3/e, M. Morris Mano, 1993)

The hardware implementation of logic circuit for the register transfer micro-operation is depicted in the Figure 7.4. $R1$ is the n -bit source register and $R2$ is the n -bit destination register. Both registers have n inputs and n outputs. The outputs of $R1$ are connected to the inputs of $R2$. The register $R2$ has a load function P . Both P and the register are synchronized by the same clock. As shown in Figure 7.3, P is activated ($P=1$) at rising edge of the clock at time t . The next transition of the clock at time $t+1$ finds the load active and all the bits of $R2$ are loaded in parallel. The data transfer takes place at every clock transition until P becomes 0.

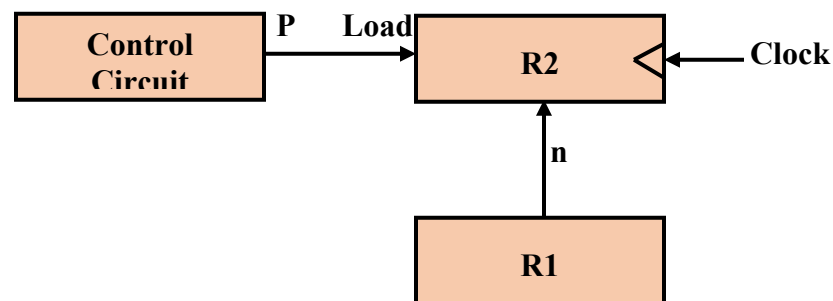


Figure 7.4: Block diagram for data transfer from $R1$ to $R2$ when $P=1$

(Source: Computer System Architecture, 3/e, M. Morris Mano, 1993)

7.2.2 Bus Transfer Micro-Operations

There are many registers in a digital computer system where frequent data transfer takes place during the operations. It is not an economical and efficient solution to have dedicated wired connections among the registers. Instead a common bus is used for connecting the registers for data transfer. A bus is a group of wires commonly used for data/signal transfer between different components. One separate wire is needed for each bit of the register. The control signal or multiplexers are used to select registers for data transfer through the bus. The bus lines are connected to the inputs of the destination register to transfer the information. Symbolically the data transfer from register $R1$ to register $R2$ through a bus can be expressed as follows

$$\text{BUS} \leftarrow R1, R2 \leftarrow \text{BUS}$$

Firstly, contents of $R1$ are placed onto the bus and later contents of the bus are written to the register $R2$ by activating its load control input.

7.2.3 Memory Transfer Micro-Operations

The read and write operations are the major memory operations in a computer system. In a memory read operation the information is transferred from the memory to the other devices or components. While, memory write operation transfers the data from some outside source to the memory locations. The destination memory location is specified by the memory address. The memory address is provided for each type of memory operations. The read operation can be expressed as follows

$$R1 \leftarrow M$$

where M is the address of the memory location that is the source of data in this operation and $R1$ is the destination register. Similarly, the memory write operation is specified as follows

$$M \leftarrow R1$$

7.2.4 Arithmetic Micro-Operations

The arithmetic micro-operations are related to basic arithmetic operations: addition, subtraction, increment, decrement, and shift, etc. The micro-operations perform these operations on numeric data loaded into registers. The operations are not carried out directly on the individual registers. Instead, data contents of the processor registers are transferred to the registers directly to the ALU. The desired operation is performed by ALU and the result is transferred to the destination register. A micro-operation for the addition operation can be expressed as follows

$$R2 \leftarrow R1 + R0$$

The contents of registers $R0$ and $R1$ are added and the result is loaded to register $R2$. This operation involves three registers. In ALU, the arithmetic operations are carried out with the help of accumulator register. The micro-operation for the subtraction operation is expressed as follows

$$R2 \leftarrow R1 - R0$$

The subtraction is usually implemented with the help of 2's complement representation. The increment simply adds 1 to the contents of the source register

$$R2 \leftarrow R1 + 1$$

Similarly the decrement micro-operation subtracts 1 from the contents of the source register

$$R2 \leftarrow R1 - 1$$

The micro-operations for increment and decrement operations are performed with hardware implementation using logic gates. There could be micro-operations for the multiplication and division operations if these are carried out with the help of combinational circuits.

7.2.5 Logic Micro-Operations

Logic operations are binary operation that consider each individual bit of a register as an individual binary number. If logic operation is carried out between two registers then individual bits of one register are operated with the corresponding bits of the other register involved in the micro-operation. For example the following micro-operation performs logic OR operation between the bits of the register $R1$ and $R2$ and result is stored in register $R3$.

$$R3 \leftarrow R2 + R1$$

The symbol + represents the logic OR operation in the logic micro-operation. If the contents of $R1$ are 10110110 and the contents of $R2$ are 11011001, then result is obtained as follows

1 1 0 1 1 0 0 1	$R2$
1 0 1 1 0 1 1 0	$R1$

1 1 1 1 1 1 1 1	$R3$

The logic AND micro-operation is stated and performed as follows

$$R3 \leftarrow R2 \cdot R1$$

1 1 0 1 1 0 0 1	$R2$
1 0 1 1 0 1 1 0	$R1$

1 0 0 1 0 0 0 0 *R3*

The logic XOR micro-operation is stated and performed as follows

$$R3 \leftarrow R2 \oplus R1$$

1 1 0 1 1 0 0 1 *R2*

1 0 1 1 0 1 1 0 *R1*

0 1 1 0 1 1 1 1 *R3*

Similarly other logic micro-operations such as NOT, NOR, and NAND can be carried out on the register contents. However the symbol used in the expressions so far are also used in Boolean expressions. Therefore some different symbols are used for logic micro-operations to distinguish them from Boolean operations. The logic micro-operations with their special symbols are listed in Table 7.1. The registers *R1* and *R2* are used to store the data and the result is stored in register *F* in all the micro-operations given in the table. The logic micro-operations can be easily implemented in the hardware with logic gates. There are fourteen logic micro-operations listed in the Table 7.1 but only four operations AND, NOT, OR, and XOR are implemented and rest other operations are derived from these four basic logic micro-operations. The logic micro-operations have range of applications involving manipulation of individual bits in the registers. In some operations only a subset of the register bits is need to manipulate. Logic micro-operations are highly useful for such operations.

Table 7.1: List of logic micro-operations

Micro-operation	Name
$F \leftarrow 0$	Clear

$F \leftarrow R2 \wedge R1$	AND
$F \leftarrow R2 \vee R1$	OR
$F \leftarrow \overline{R2 \wedge R1}$	NAND
$F \leftarrow \overline{R2 \vee R1}$	NOR
$F \leftarrow \overline{R1}$	NOT
$F \leftarrow R1 \oplus R2$	XOR
$F \leftarrow \overline{R1 \oplus R2}$	XNOR
$F \leftarrow \text{all } 1\text{'s}$	Set to all 1's
$F \leftarrow A$	Transfer A
$F \leftarrow R2 \wedge \overline{R1}$	
$F \leftarrow \overline{R2} \wedge R1$	
$F \leftarrow R2 \vee \overline{R1}$	
$F \leftarrow \overline{R2} \vee R1$	

7.2.6 Shift Micro-Operations

Shift micro-operations are very useful for serial transfer of data as well as they are used to implement various arithmetic and logic operations such as multiplication and division. The shift micro-operation could be used to shift contents of a register either towards left or right. Therefore, there are two kind of shift micro-operations: left shift and right shift. In case of a left shift micro-operation, all the existing bits are shifted one place to the left. As a result new bit enters at the vacant leftmost location and a bit comes out of the rightmost location. Similarly in case of a right shift micro-operation, a new enters at the vacant rightmost position and one bit comes out of the leftmost bit position. Shift micro-operations can be divided into three categories depending on what bit enters at one end and where the bit shift out of the other end goes. The three categories of shift micro-operations are:

- a) Logical shift micro-operation
- b) Arithmetic shift micro-operation
- c) Circular or rotate shift micro-operation

In a logical shift operation, the bit that enters through one end (left or right depending on the type of shift) is always assumed to be 0. The logical shift micro-operation is

represented by a symbol *shl* (left shift) or *shr* (right shift). The statement for a shift micro-operation is written as follows

$$R1 \leftarrow shl R1 \quad \text{or} \quad R1 \leftarrow shr R1$$

In case of a circular shift, the bit shift out of the register enters back to the register through other end. The circular shift micro-operation is represented by a symbol *cil* (left shift) or *cir* (right shift). The statement for a circular shift micro-operation is written as follows

$$R1 \leftarrow cil R1 \quad \text{or} \quad R1 \leftarrow cir R1$$

An arithmetic shift multiplies or divides the binary number by 2 depending whether it is left shift or right shift. Description of the major shift micro-operations is given in Table 7.2.

Table 7.2: List shift micro-operations

Shift Micro-operation	Description
$R1 \leftarrow shl R1$	Shift left register <i>R1</i>
$R1 \leftarrow shr R1$	Shift right register <i>R1</i>
$R1 \leftarrow cil R1$	Circular shift left register <i>R1</i>
$R1 \leftarrow cir R1$	Circular shift right register <i>R1</i>
$R1 \leftarrow ashl R1$	Arithmetic shift left register <i>R1</i>
$R1 \leftarrow ashr R1$	Arithmetic shift right register <i>R1</i>

Check your progress 1

1. What is micro-operation?
2. Differentiate hardwired control unit and micro-programmed control unit.
3. Write different categories of micro-operations.
4. Perform single circular shift operation on string 101010.
5. How control function is specified in a statement?

7.3 Functional Requirements of Control Unit

The basic functional elements of a processor are ALU, registers, data paths, and control unit. As already discussed, the ALU carries out the most of fundamental operations in CPU, registers are used to store the different types of data, the information stored at different memory location (registers or main memory) is transferred through data paths, and control unit is responsible to cause the operations to happen. Two major functions of the control unit are **sequencing** and **execution**. A task can be accomplished through a sequence of instructions. An instruction is executed as series of micro-operations. It is the responsibility of the control unit to ensure the proper sequence of micro-operations. Although control unit does not directly perform any micro-operation but it causes the micro-operations to execute by the processor through control signals. The generation of the appropriate control signal is key feature of the control unit. The sequencing and execution is performed with help of required logic. It controls the data flow in the processor. A general model of the control unit is shown in Figure 7.5. The major components in the model of control unit are: clock, a set of flags, instruction register, and control bus.

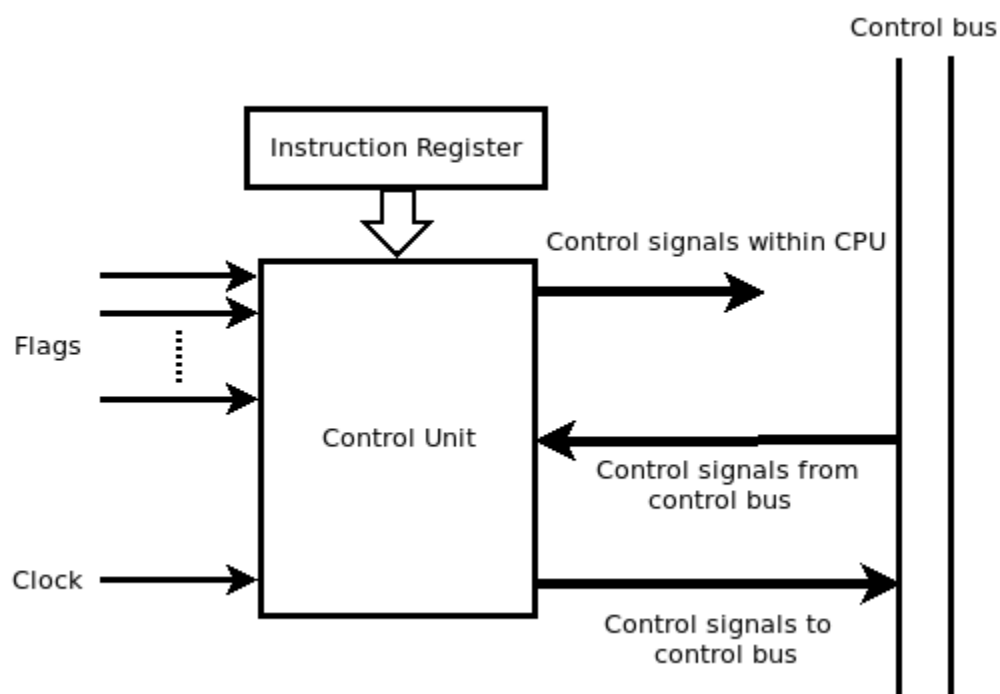


Figure 7.5: General model of control unit

The control unit receives input from the following sources: clock, flags, instruction register, and control bus. The output of the control unit i.e. control signals are issued through control bus only. The sources of input to control unit are described here.

a) Clock: Control unit uses a clock to cause micro-operations to be performed. One micro-operation is performed per clock pulse. The clock pulse is also referred to as clock cycle or clock time. The micro-operations are allowed at the edge of clock pulse transition.

b) Flags: Flags are used to determine current state of the processor and operations of ALU in the computer system. Flags are designated bits of a flag register. Before performing any micro-operation, the control unit checks the associated status of flags.

c) Instruction Register: In order to determine which micro-operation to perform, the control unit needs opcode and addressing mode. The opcode and addressing mode of the current instruction are available in the instruction register.

d) Control Bus: The control unit receives signals through control bus.

The control unit determines the micro-operation to perform and generates signals for processor and other components of the processor. There two types of the outputs:

a) Control Signals to Processor: The control signals that are sent to the processor itself can cause register transfer operation or activate ALU operations.

b) Control Signals to Bus: The control signals that are sent through the control bus cause operations related to memory or I/O devices.

7.4 Hardwired Control Unit

A number of techniques are there to implement the control unit. However, two techniques are more prominent for control unit implementation: hardwired and micro-programmed. The hardwired control unit is a state machine implemented with logic circuits hardware that cannot be modified without physically changing the circuit structure. A model of hardwired control unit is shown in Figure 7.6. The control unit receives input from an instruction register, flags, a clock, and control bus. The Instruction Register contains two information fields: opcode and address field. The opcode of the instruction provides the basic information for control signal generation. The control unit does not access the instruction directly. Instead it is accessed through a

decoder, which decodes the opcode of the instruction. The encoded input is provided to control unit, which takes different actions for different instructions depending on opcode and issues a combination of control signals. The timing unit generated clock pulses, which are useful to organize the execution of the instruction. The duration of the clock pulse should be long enough to execute a micro-operation. The timing unit may also receive some kind of feedback that helps control unit to generate the control signal. When a new instruction is received, it is important to have information about the present state of the previous instruction and accordingly the new instruction is allowed to enter a different state.

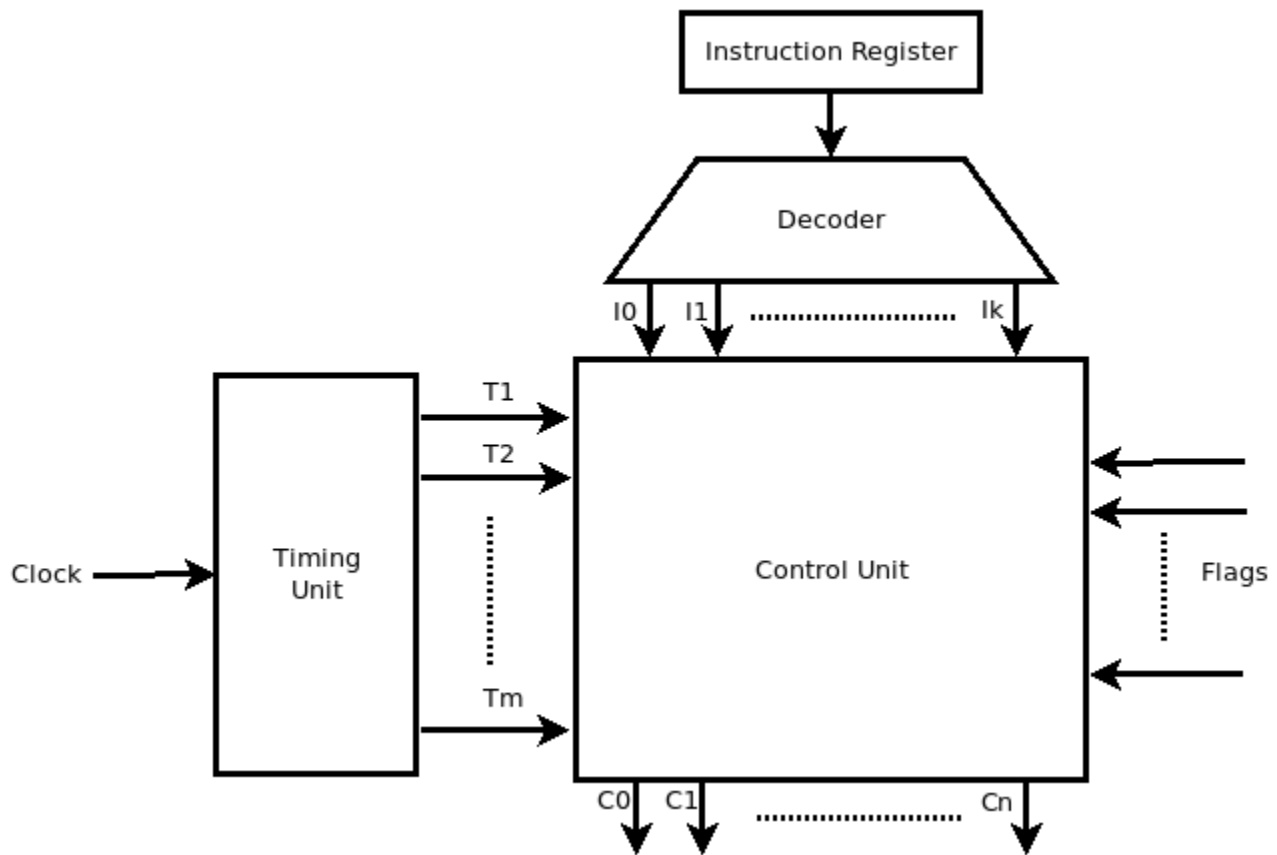


Figure 7.6: Hardwired control unit with decoded inputs

The design of the control unit becomes complex if more control signals are required by the CPU. In a hardwired control unit, the modifications are very difficult as it requires changes in the logic circuits. Therefore, incorporation of the new features in a hardwired

unit is difficult. Another architecture of a hardwired control unit is shown in Figure 7.7. It uses two decoders and a 4-bit sequence counter. The instruction is obtained from a 16-bit Instruction Register. The first twelve bits (0-11) provide the address, the next three bits (12,13, and 14) specify the opcode, and the remaining last bit (the most significant bit) specifies whether it is a direct or indirect addressing mode. The address field directly goes to the logic circuits of the control unit, the opcode is processed by a 3x8 decoder. The output of the decoder (D0 to D7) then provided to the logic circuit unit, the most significant bit is processed by a flip-flop represented by I before going to the logic circuits. A 4-bit counter generates 16 timing pulses T0 through T15. It can be cleared or incremented based on the inputs from CLR or INR respectively. For example timing signals T0, T1, T2, and T3 are generated and counter is cleared at T3 if decoder output D2 is found active. It can be written symbolically as follows

$$D2 T3: SC \leftarrow 0$$

where SC is the sequence counter. The relationship of control signals is shown in Figure 7.8.

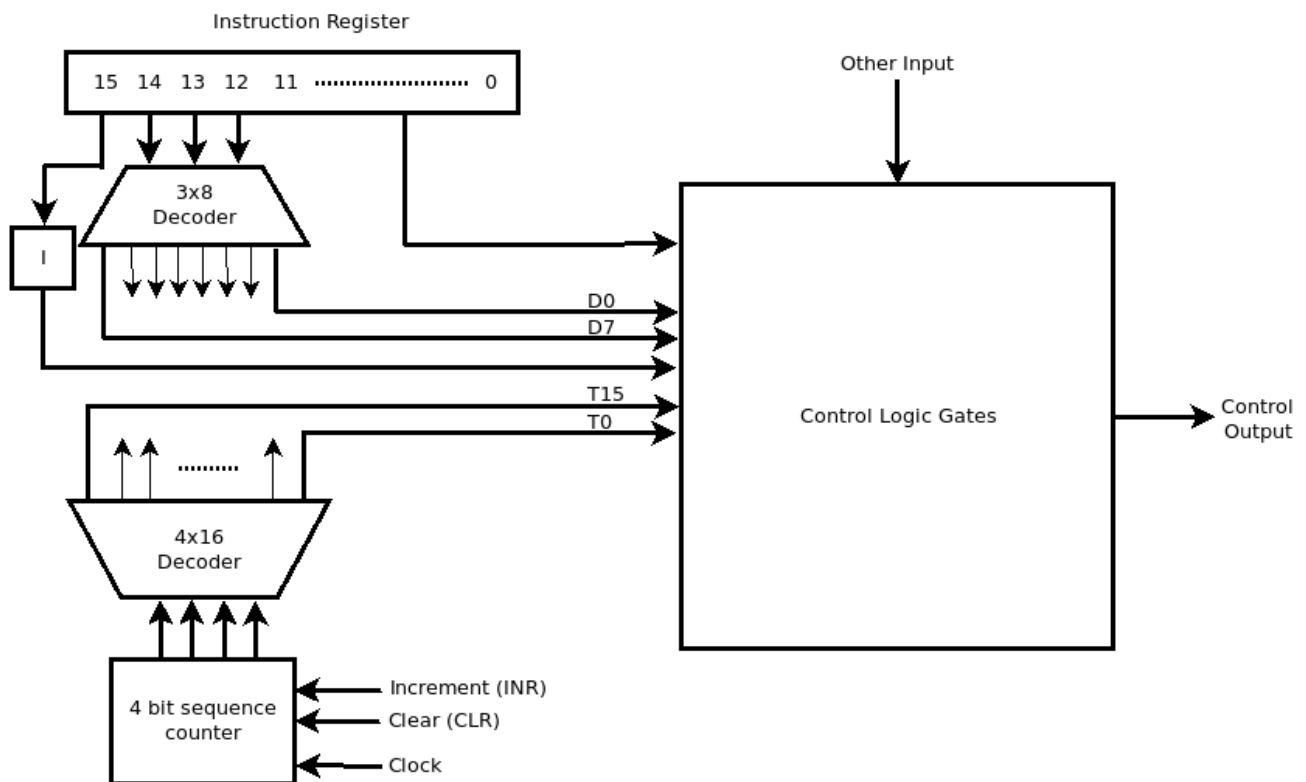


Figure 7.8: A hardwired control unit with two decoders and a sequence counter

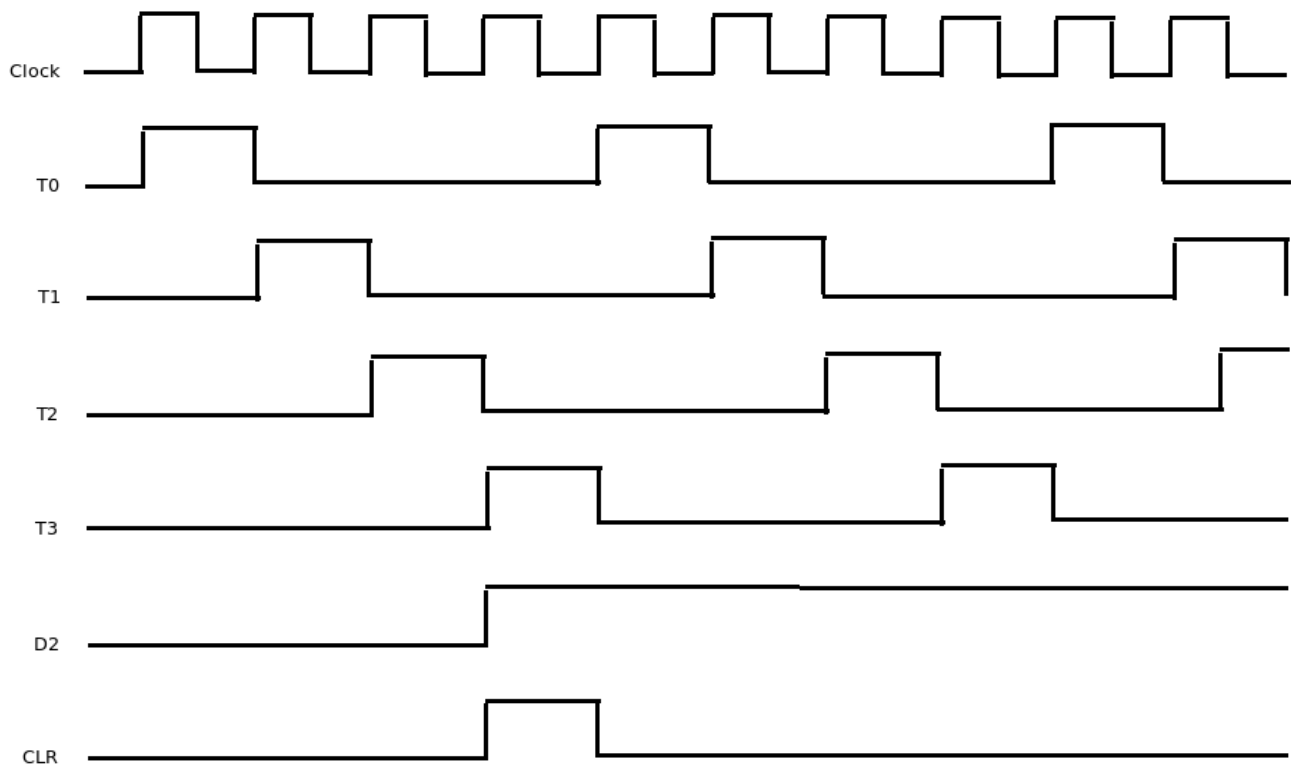


Figure 7.9: Timing Diagram

In Figure 7.9, the top line represents the clock pulses. The next four lines represent the states T0, T1, T2, and T3 generated by the sequence counter. Each state is asserted for one clock period and then remain at logic 0 for the next three clock periods. When decoder output D2 is detected, the change in state is triggered.

Check your progress 2

1. What are the major operations of Control Unit?
2. What are major components of Control Unit?
3. What is the use of Instruction register?
4. What are the major techniques to implement Control Unit?
5. Why decode is used in Control Unit?

7.5 Control Unit Logic

The control unit logic works on the input signals to generate output control signals. Let us assume that there are four micro-operations named as mOP1, mOP2, mOP3, and mOP4 performed in the four phases of the instruction cycle. Let us define two control signals as P and Q with the following interpretation:

PQ = 00 mOP1

PQ = 01 mOP2

PQ = 10 mOP3

PQ = 11 mOP4

If micro-operations mOP1 and mOP2 are needed to perform during the same time unit T1 then control signal C1 can be defined by the following Boolean expression

$$C1 = \bar{P}.\bar{Q}.T1 + \bar{P}.Q.T1$$

Any control signal can be defined in this way using Boolean expressions. A set of Boolean expressions defines the behaviour of the control unit. The modern computer systems use a large number of Boolean expressions to define the control unit. All input signals to the Control Logic Gates section are combined to generate control signal. Suppose a control signal ADD is needed to add the contents of memory location to a register. Let it needs to perform micro-operation AM in time cycle T₅ and a branch micro-operation BR in time cycle T₆. The Boolean expression for generating control signal can be given by

$$ADD = T1 + T5.AM + T6.BR$$

The logic function for ADD can be implemented as a combinational circuit using AND and OR logic gates. The circuit diagram for ADD control signal is shown in Figure 7.10. AM and BR are provided by Instruction Register through decoder. Consider another example, where a control signal is described by the Boolean expression

$$END = T4.mOP1 + T5.mOP2 + T6.mOP3.BR + T7.mOP4.BR$$

The circuit diagram is given in Figure 7.11 that is implemented using AND and OR logic gates.

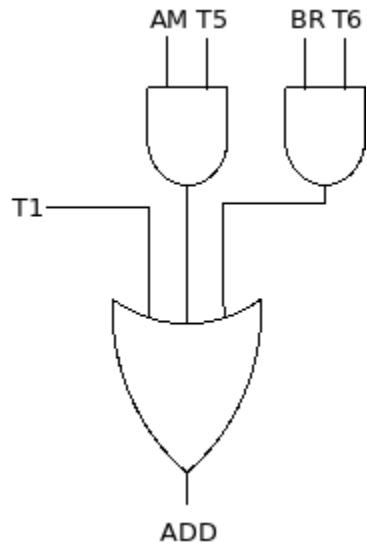


Figure 7.10: Combinational circuit for generating ADD control signal

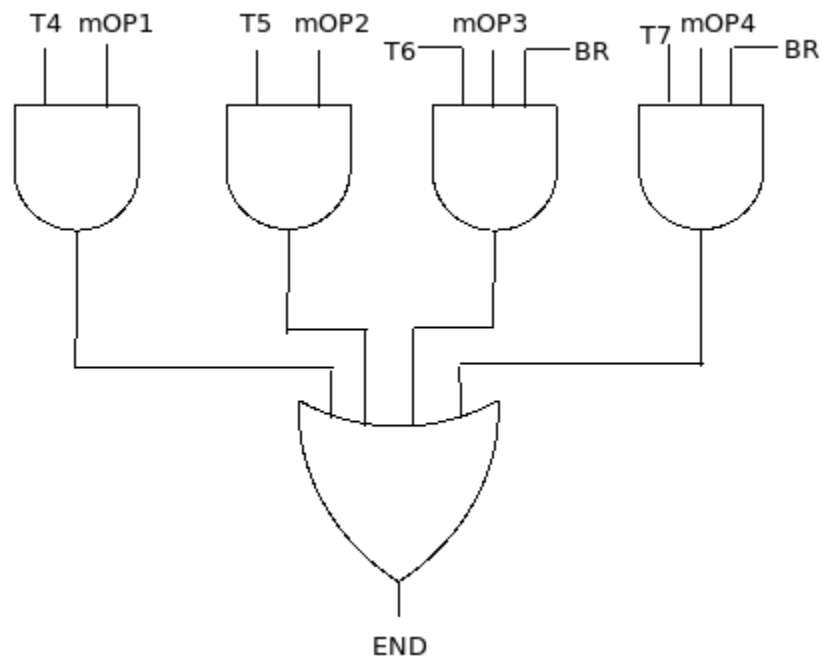


Figure 7.11: Combinational circuit for generating END control signal

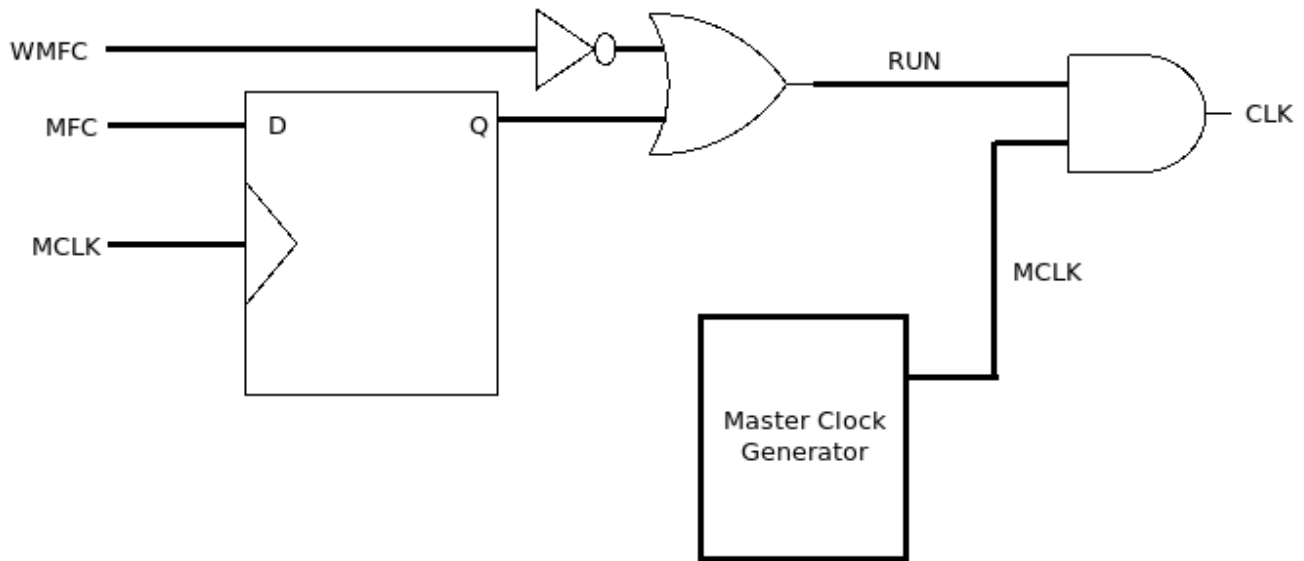


Figure 7.12: Circuit diagram for synchronizing WMFC with RUN signal.

Let us consider that CPU does not do any works when it receives wait for MFC (WMFC) signal and a delay is introduced while it is waiting for an MFC signal from memory unit. Memory operations are independent of CPU cycles. MFC is an asynchronous signal that is generated any time relative to CPU cycle. Therefore, it is required to synchronize MFC with CPU clock. It could be achieved with the help of a D flip flop as shown in Figure 7.12. The RUN signal with a master clock pulse through an AND logic gate helps to synchronize the signal. When RUN signal is low, no progress is allowed.

7.6 Summary

The control unit is an important component of the CPU, which plays a crucial role in the sequencing and execution of the operations. Any task in the computer system is performed by executing a sequence of instructions. The execution of an instruction is done in different phases. Each phase is further divided into multiple micro-operations. A micro-operation accomplishes a little fraction of the overall larger task. The responsibility of the control unit is ensure the proper sequencing of the micro-operations and to control the flow of data between CPU and memory and peripherals. It issues the control signals to ALU, memory, storage devices, and peripherals. The control unit is

responsible for data transfer to and from registers, controlling ALU operations, and opening/closing of logic gates by issuing relevant control signals.

There are two major organizations for control unit: hardwired and micro-programmed. A hardwired control unit generates control signals using logic circuits. The micro-programmed control unit uses a micro-program to control the operations in a digital computer. A control function of a micro-operation is specified by a binary variable. The execution of the micro-operation depends on the state of the associated binary variable. Usually the micro-programs are not required to change for an operation control unit. Therefore, control memory could be a read-only-memory (ROM).

The micro-operations are the functional operations of a processor. There are different types of micro-operations that perform different types of elementary jobs. The micro-operations can be classified into four broad categories: register transfer micro-operations, arithmetic micro-operations, logic micro-operations, and shift micro-operations. Some elementary operations can be carried out on data contents of the registers. The most prominent register operations include data transfer operations that do transfer from one register to another. These micro-operations are known as register transfer micro-operations. The symbolic notation used to express micro-operations for register transfer is called register transfer language. The logic circuit for register transfer micro-operations can be implemented with the help of logic gates and a clock that controls the execution of the operations. The actual data transfer between registers and other components of the computer takes place through data bus. A data bus is a group of wires commonly used by different components for information transfer. The micro-operations related to bus operations are known as bus transfer micro-operations, which are implemented with the help of a multiplexer. The micro-operations related to the memory read/write operations are known as memory transfer micro-operations. The read operation means data transfer out of the memory, while write operation simply means the data transfer into the memory. The arithmetic micro-operations are related to basic arithmetic operations: addition, subtraction, increment, decrement, and shift, etc. The micro-operations perform these operations on numeric data loaded into registers. The operations are not carried out directly on the individual registers. Instead, data contents of the processor registers are transferred to the registers directly to the ALU.

Logic operations are binary operation that consider each individual bit of a register as an individual binary number. If logic operation is carried out between two registers then individual bits of one register are operated with the corresponding bits of the other register involved in the micro-operation. Shift micro-operations are very useful for serial transfer of data as well as they are used to implement various arithmetic and logic operations such as multiplication and division. There are two kind of shift micro-operations: left shift and right shift depending on the direction of data movement. The shift micro-operations are further divided into three categories: logical shift micro-operation, arithmetic shift micro-operation, and circular or rotate shift micro-operation

Two major functions of the control unit are sequencing and execution. A task can be accomplished through a sequence of instructions. An instruction is executed as series of micro-operations. It is the responsibility of the control unit to ensure the proper sequence of micro-operations. The major components in the model of control unit are: clock, a set of flags, instruction register, and control bus. A clock is required to cause micro-operations to be performed. Flags are used to determine current state of the processor and operations of ALU in the computer system. The opcode and addressing mode of the current instruction are available in the instruction register. The control unit receives signals through control bus.

The hardwired control unit is a state machine implemented with logic circuits hardware that cannot be modified without physically changing the circuit structure. The control unit receives input from an instruction register, flags, a clock, and control bus. The control unit does not access the instruction directly. Instead it is accessed through a decoder, which decodes the opcode of the instruction. The action is taken according to the encoded input. The design of the control unit becomes complex if more control signals are required by the CPU. In a hardwired control unit, the modifications are very difficult as it requires changes in the logic circuits. Therefore, incorporation of the new features in a hardwired unit is difficult.

Review Questions

Q.1 What are the major responsibilities of a control unit in a digital computer system?

- Q.2 What are the major organizations of a control unit? Make a comparison between different organizations.
- Q.3 What is micro-operation? Explain different types of microoperations with suitable examples.
- Q.4 What is the role of a clock in control unit? Give a simple organization involving a clock and explain the utility of a clock with the help of a timing diagram.
- Q.5 Perform AND, OR, and XOR micro-operations on the following data:
- (a) 11010011 and 10011001
 - (b) 10101010 and 11000011
 - (c) 01010110 and 10010111
 - (d) 11110111 and 11001101
- Q.6 Draw a block diagram for the general model of a control unit and explain its functional requirements.
- Q.7 Draw a block diagram for a control unit with two decoders and explain the role of various components.
- Q.8 Implement logic circuit to generate control signal expressed by the following Boolean expressions.
- (a) $C1 = \bar{P}.Q.T1 + \bar{P}.\bar{Q}.T2$
 - (b) $C2 = P.Q.T1 + \bar{P}.Q.T2 + \bar{P}.\bar{Q}.T3$
 - (c) $C3 = P.Q.T1 + \bar{R}.Q.T3 + \bar{Q}.T4 + P.R.T5$
 - (d) $C4 = T1 + Q.R.T5 + \bar{R}.\bar{Q}.R.T6$

UNIT- 8 Micro-programmed Control

Structure

8.0 Introduction

8.1 Objectives

8.2 Basic Organization of Micro-programmed Unit

8.3 Micro-instruction Sequencing

8.4 Micro-instruction Execution

8.5 CPU Control Unit Organization

8.6 Summary

Review Questions

Unit 8: Micro-programmed Control

8.0 Introduction

A considerable discussion has already been made on control unit in Unit 7. The primary responsibility of the control unit is to do sequencing of **micro-operations** and to generate control signals that cause opening and closing of the logic gates leading to the data transfer between registers and ALU or peripherals in a digital computer system. There are two major approaches for implementing control unit: **hardwired** and **micro-programming**. When control unit is implemented in hardware with the help of logic circuits, it is called hardwired control unit. There are many different types of micro-operations that are needed to carry out various computational tasks. As the number of micro-operations increases, the complexity of the implementation of the hardwired control unit also increases and so the cost of implementation. The micro-programmed control unit is another alternative implementation which uses a **micro-program** to specify the logic of the control unit. A micro-program consists of a sequence of instructions that specify the micro-operations. The instructions of micro-program are written in a **micro-programming language**. A micro-program is a sequence of binary strings. The concept of micro-program is similar to the computer program. As a computer program is formed by a set of statements written in some programming language, a micro-program is also a set of statements written in micro-programming language. The computer program is stored in main memory and its instructions are fetched in a sequence with the help of program counter to execute. In the same way, the micro-program is stored into a micro-program memory and the **micro-instructions** are fetched from the micro-program memory during the execution. The sequencing of the instructions during the execution is maintained with the help of a micro-program counter. Since the micro-program is usually not changed often, the micro-program memory is a read only memory (ROM). The memory that is the part of control unit is known as **control memory**. The micro-programmed control unit is slower than the hardwired control unit. The advantage of micro-program control unit is that its design is simple and it is easy to implement.

The micro-programmed control unit uses a relatively simple logic circuit that is capable of sequencing and executing micro-instructions to generate the control signals. The contents of micro-memory specify the micro-instructions that are strings of 0's and 1's. A particular memory cell provides the state of a micro-instruction. If it is a binary state 0, it simply means that no control signal to generate. If it is a binary 1, the corresponding micro-instruction is executed and the output corresponds to a control signal. Some advanced micro-programmed control units use **dynamic micro-programming** that allows to initially load the micro-program from the auxiliary memory. The control units with dynamic micro-programming use the writable control memory.

8.1 Objectives

After the completion of this unit the readers should be able to:

1. Understand different implementations of control unit with their merits and demerits.
2. Understand the sequencing of micro-operations.
3. Learn addressing techniques.
4. Understand the execution of the micro-instructions.

8.2 Basic Organization of Micro-programmed Unit

The basic organization of a micro-programmed control unit is shown in Figure 8.1. The computer program and related data are kept in main memory of the computer. The program instructions are read from the main memory through instruction register and corresponding to each instruction a series of micro-instructions is initiated. Each micro-instruction is defined by a unique sequence of 0's and 1's in control word (CW). The individual bits of a control word represent various control signals. The sequence of CWs makes a micro-program and individual CW in a micro-program is a micro-instruction.

The control memory in the control unit is used to store fixed micro-program. The contents of the control memory cannot be usually altered by the users. The micro-program is a sequence of micro-instructions that initiate the micro-operations to fetch instruction, to evaluate effective address, or to perform various arithmetic, logic, and shift operations, etc. Once the execution of an instruction is completed, the address of

next address is determined. The fetch phase of all the instruction is same, therefore the same micro-instruction is used to fetch the instruction from the memory. After the completion of the fetch micro-instruction, the address of the instruction currently available in instruction register is determined. The execution of micro-instructions is controlled by micro-program counter in the same way as execution of instructions is controlled by program counter. The micro-instructions are fetched from micro-program memory and sequenced by micro-program counter to generate appropriate control signals.

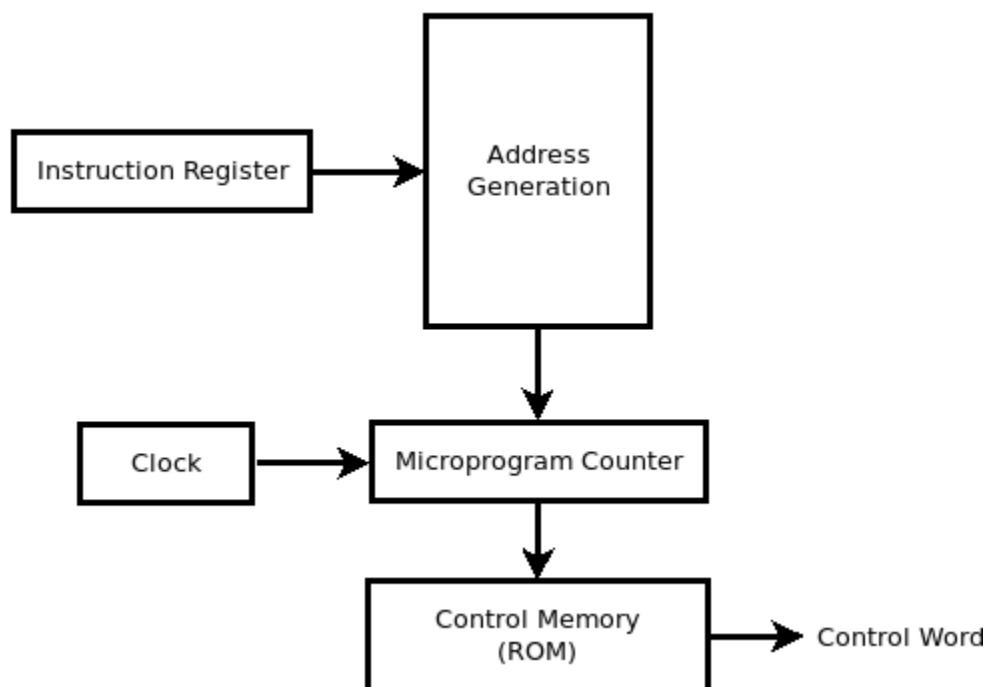


Figure 8.1: Basic organization of micro-programmed unit

When an instruction is fetched from the main memory, a micro-program associated with that instruction is executed to execute that instruction. Overall, the following elements are required in order to design a micro-programmed control unit:

- a) Micro-programs: For each CPU instruction, a micro-program is written and stored in control memory.
- b) Micro-instructions: A micro-program is sequence of micro-instructions.

- c) c) Control word: The micro-instructions are kept as a string of 0's and 1's known as control word. A 0 in control word means low signal, while a 1 represents high signal value.
- d) Micro-program counter: A micro-program counter is required to maintain the sequence and execution of micro-instructions of a micro-program.
- e) Instruction register: An instruction register contains the starting address of a micro-program.
- f) In addition to above mentioned components, a branch address generation unit is also required for branching micro-instructions in the micro-programs.

Micro-instruction sequencing and micro-instruction execution are the two important basic tasks of a micro-programmed control unit. Both affect the type and format of the micro-instructions. The size of micro-instructions, address generation, faster execution, and cost are the major design concerns of the micro-programmed control unit.

Check Your Progress: 1

1. Compare the major features of hardwired and micro-programmed control units.
2. Identify the merit/demerits of a micro-programmed control unit.
3. Define the following terms: micro-instruction, micro-program, control word, and micro-program counter.

8.3 Micro-instruction Sequencing

The design considerations of micro-instruction sequencing involve the micro-instruction size and timing of address generation. The smaller size of control memory is desirable that leads to lower cost. The faster execution of the micro-instructions is another important concern. During the execution, address of the next micro-instruction is needed. The address of the next micro-instruction could be: branch address, next sequential address, or determined by instruction register. The sequential addresses are

more common but branch addresses are also frequently required. Therefore, it is important to keep the instructions small and time efficient.

The address sequencing of the control memory must be capable of sequencing the micro-instructions in a micro-program. The initial address is loaded into control address register (CAR), which is usually the address of micro-instruction that initiates the instruction fetch routine. The CAR is incremented to sequentially access the other micro-instructions in the micro-program. At the end of the fetch micro-program routine, the instruction register contains the next instruction. The next step is to determine the address of the operand. There are different types of addressing techniques:

1. Immediate
2. Direct
3. Indirect
4. Register
5. Register Indirect
6. Displacement

These addressing techniques are also known as addressing modes. Different types of operands use different addressing modes. The instruction format may have some designated bits to represent the type of the addressing mode. These designated bits are known as **mode field**. The control unit can use mode field to determine the addressing mode. Immediate address is the simplest form of addressing in which actual operand is available in the instruction itself. It is preferred to initialize the variables or to define constants. It requires no additional memory reference but large numbers can be used for operands due to the small size of address field as compared to word length. Figure 8.2 shows the instruction format for indirect address mode.

Direct addressing is another simple form of addressing as shown in Figure 8.3 in which the address field contains the effective address of the operand that requires no additional calculations. There is only one memory reference required.



Figure 8.2: Instruction format for immediate addressing

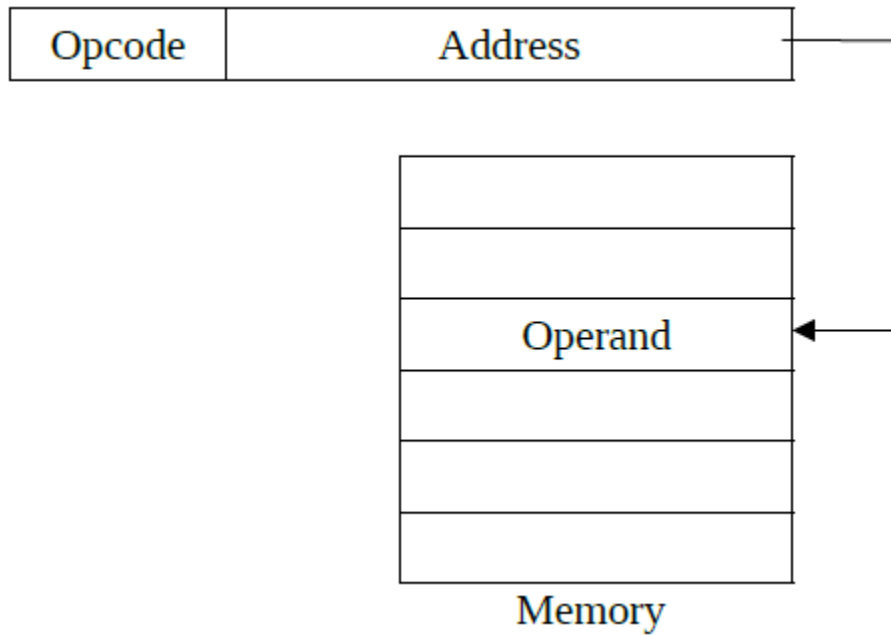


Figure 8.3: General format for direct addressing

The direct addressing limits the addressing range because the address field is smaller than word length. To overcome this limitation, the indirect addressing is used. The address field refers to a memory word, which in turn provides the address of the actual operand. The general format of indirect addressing is shown in Figure 8.4.

The register addressing is similar to direct addressing, where address field refers to a register instead of memory location. The advantage of register addressing is that it requires only a small address field but range of the addresses is very limited. The general format of addressing is shown in Figure 8.5. The register indirect addressing as shown in Figure 8.6 is similar to indirect addressing where address field refers to a register instead of memory location. The register provides the address of the operand in memory. Compared to indirect addressing, the register indirect approach needs only one memory reference. The address field refers a register, which provides address of a memory location that contains the operand. The major advantage of using registers is that memory access is faster.

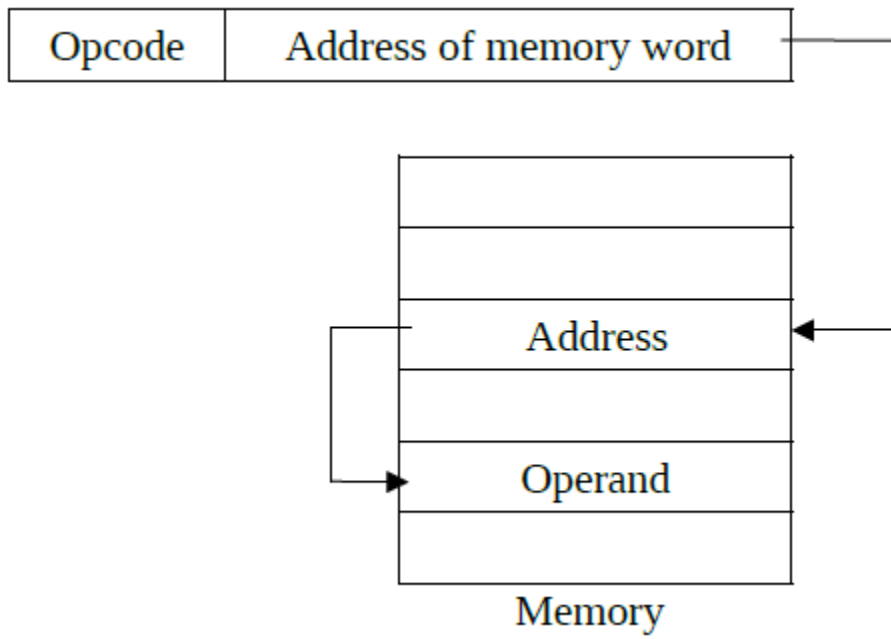


Figure 8.4: General format for direct addressing

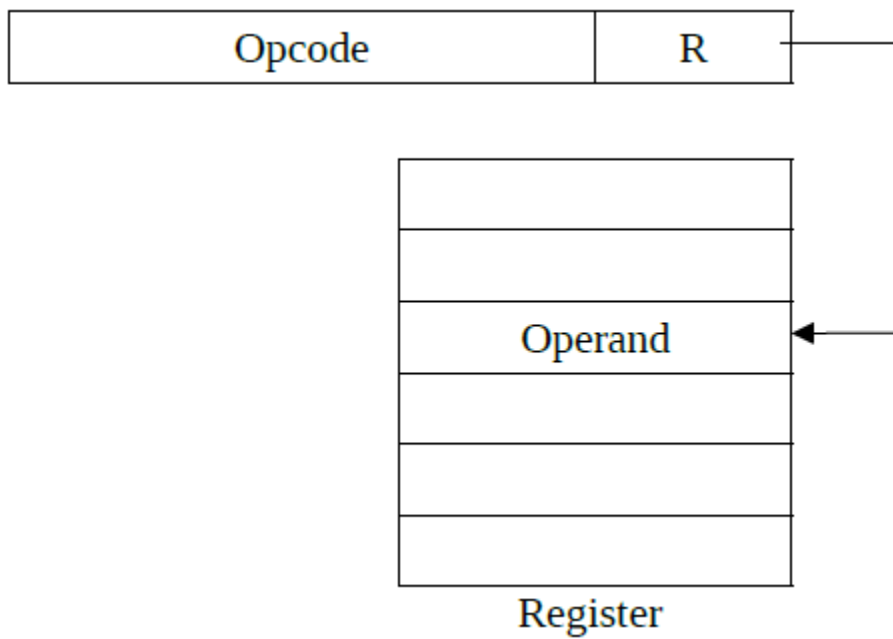


Figure 8.5: General format for register addressing

The memory provide is a large capacity while the registers provide a faster access. The displacement addressing combines the advantages of direct memory addressing and indirect register addressing. The instructions with displacement addressing use two address fields, at least one of which should be explicit. One address field refers to a register whose contents are added with the value of the other address field to determine the effective address as shown in Figure 8.7. Once effective address of the operand is available in memory address register, it can be fetched from the memory. The next step would be to determine the microoperation to execute the instruction fetched from the memory. Once the execution of the instruction is over, the control is returned to fetch micro-program routine.

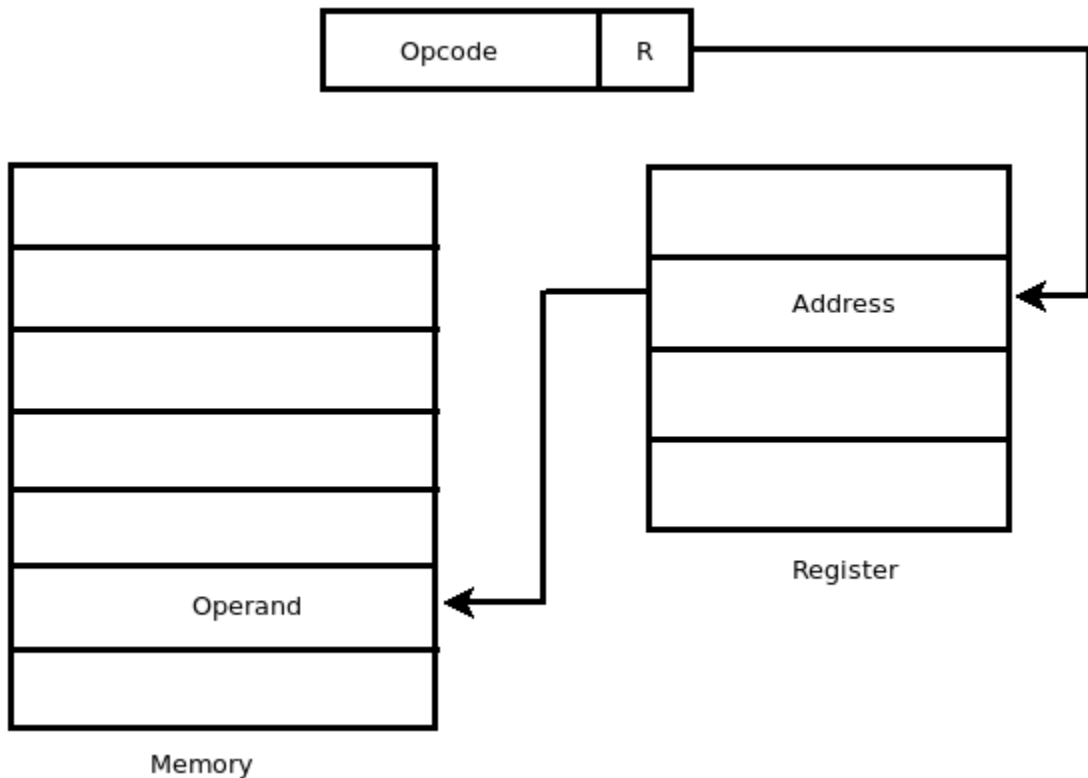


Figure 8.6: General format for register indirect addressing

For sequencing of micro-instructions, the address of next micro-instruction is required. The address of next micro-instruction can be generated based on the current micro-instruction, contents of control memory, and contents of the instruction register. A

variety of sequencing techniques are available based on the format of address. Considering the addressing techniques discussed so far, the address formats in micro-instruction can be divided into three categories:

1. Two address fields
2. Single address field
3. Variable format.

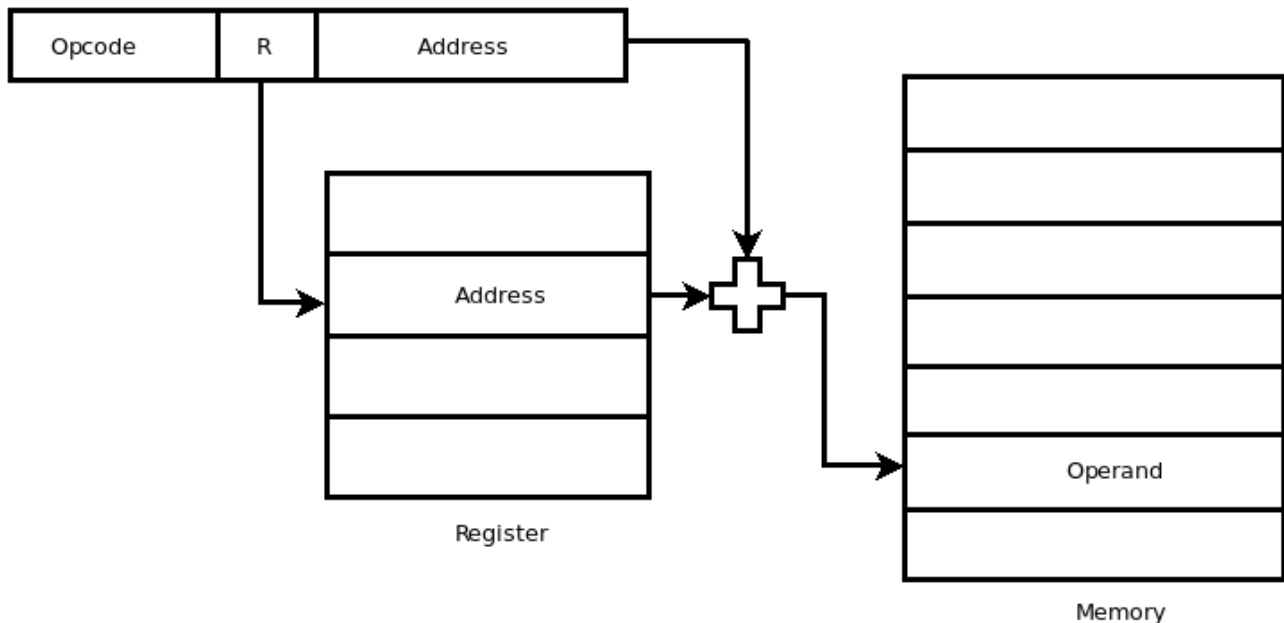


Figure 8.7: General format for register displacement addressing

Figure 8.8 shows the logic for generating the address of the next micro-instruction using the two address fields. The contents of instruction register and two address fields are provided to a multiplexer. Based on the selection inputs, the multiplexer outputs the opcode or one of the two addresses to CAR. The next micro-instruction address can be obtained by decoding the CAR. The two address fields approach is simple but it requires more bits in micro-instruction than other approaches. The number bits can be reduced by using single address field as shown in Figure 8.9. Although, single address filed micro-instruction approach involves some inefficiency. A different kind of approach is shown in Figure 8.10 that provides two different kind of micro-instruction formats. One bit in control buffer register designates the format type. In one format all the remaining

bits are used to activate control signal. In the second format, some of the remaining bits are for branch logic and other bits give the address. The branch logic could be conditional or unconditional.

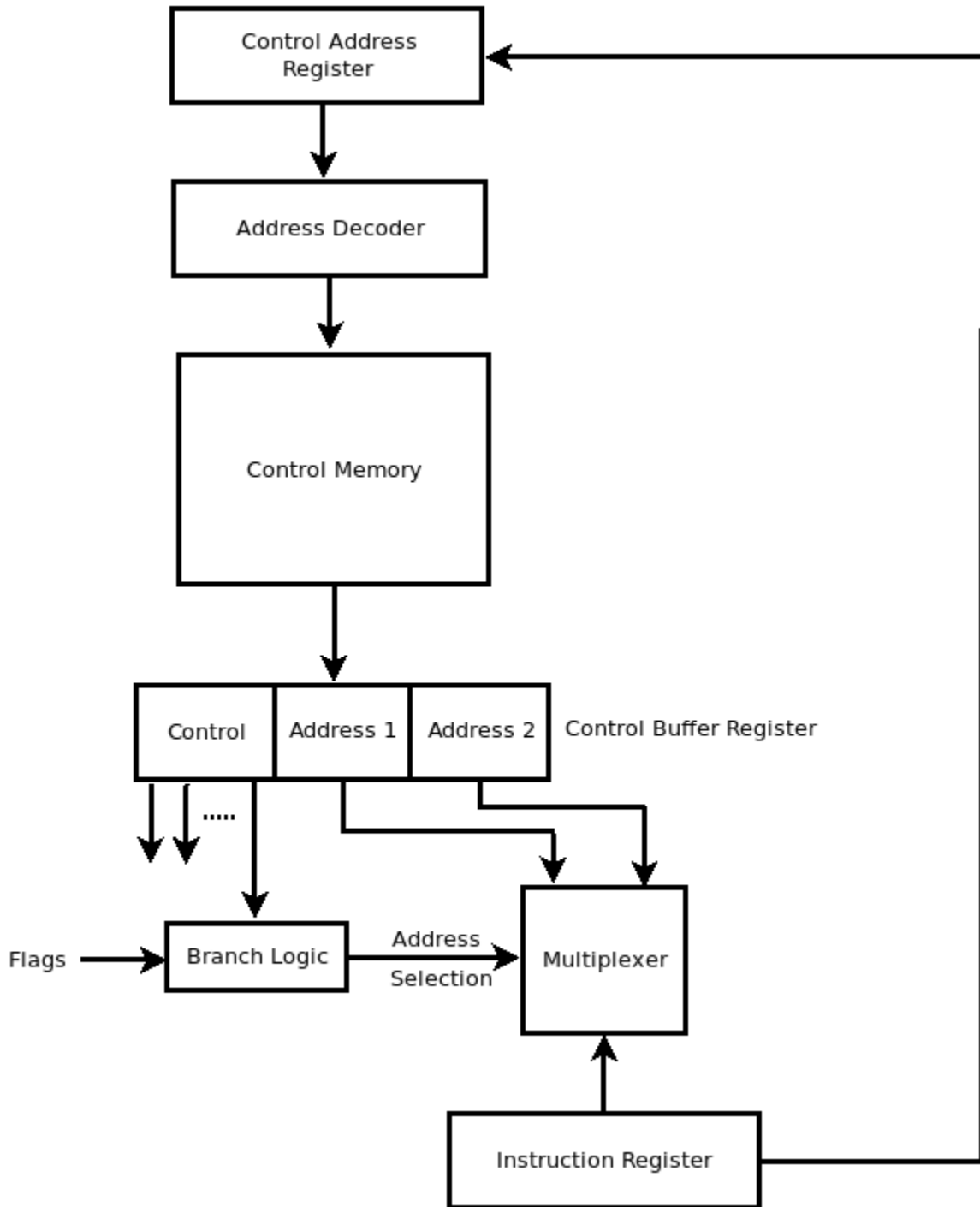


Figure 8.8: Micro-instruction format with two address fields

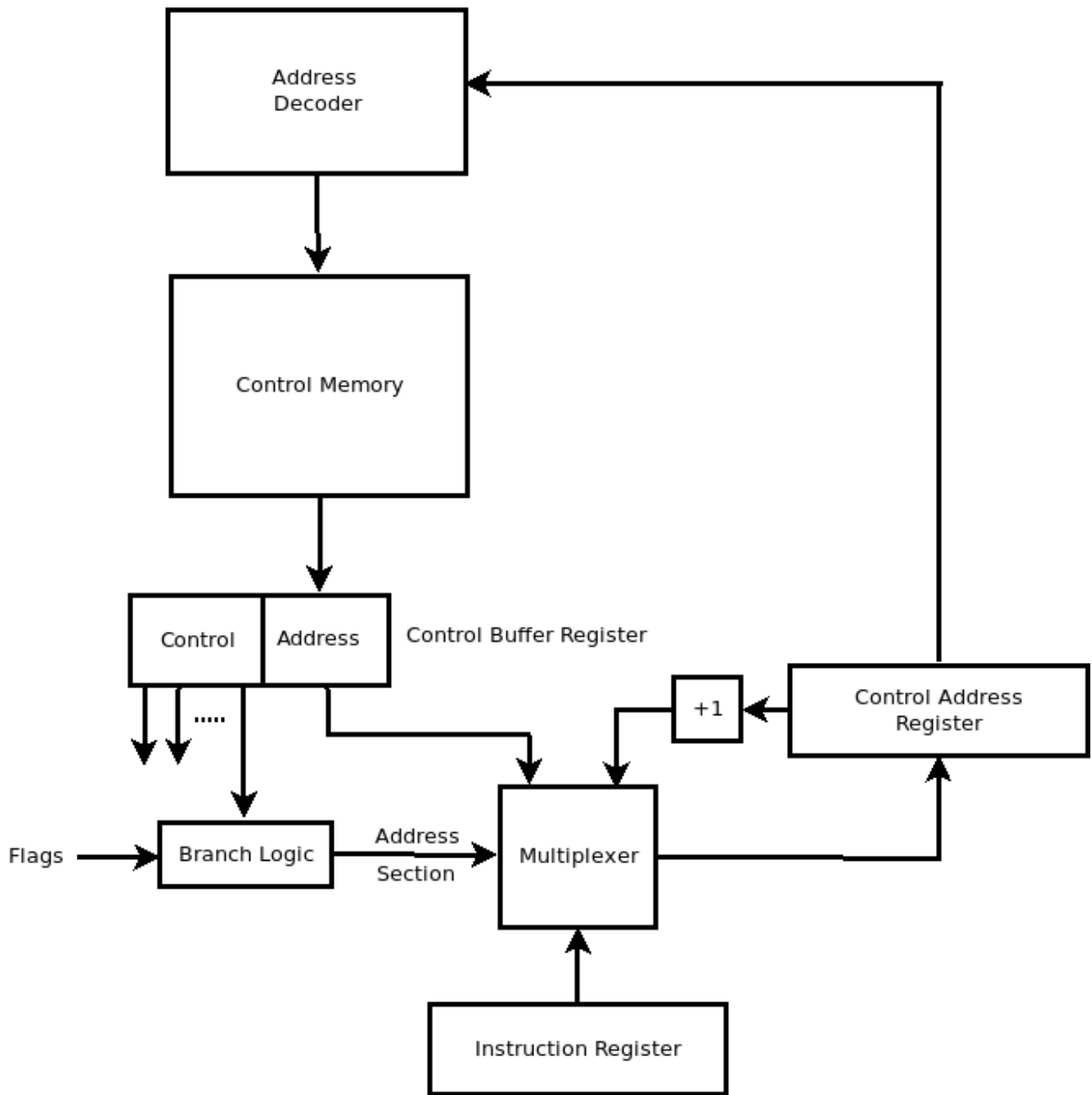


Figure 8.9: Micro-instruction format with single address fields

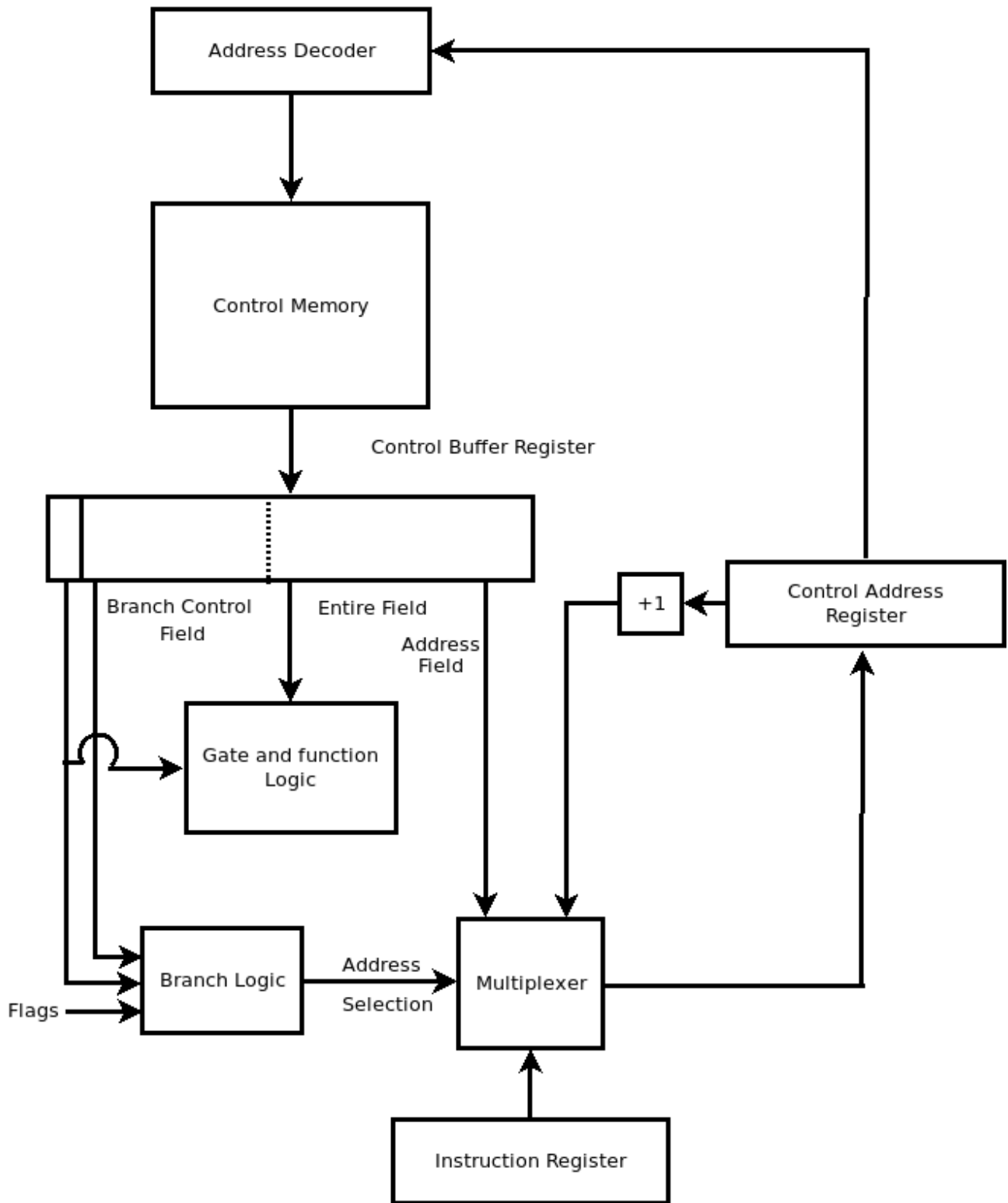


Figure 8.10: Micro-instruction: variant format

There are various techniques to determine the address of the next micro-instruction. These techniques could be implicit or explicit. The explicit techniques provide the address explicitly with micro-instructions. On the other hand the implicit techniques require additional logic to compute the address. The implementation of branch micro-instructions uses information available from ALU flags, opcode, some registers, and some status bits from the control unit.

Check Your Progress: 2

1. What is utility of control address register?
2. Compare direct and indirect addressing.
3. Identify the advantage of register addressing mode.
4. What is micro-instruction sequencing?

8.4 Micro-instruction Execution

The execution of a micro-instruction is the basic event in a processor with micro-programmed control unit. The execution of micro-instruction generates the control signal for processor as well as for the external units such as memory and peripherals. Fetch and execute are two parts of the micro-instruction cycle. The fetch operation is related to micro-instruction address generation. The micro-instruction sequencing is related to next address generation. It generates next address using the information from ALU flags, control status bits, opcode, CAR, and current micro-instruction. The bits in a micro-instruction directly or indirectly produce control signals or next address. It is important to keep the size of micro-instruction smaller. By using some encoding/decoding schemes, the number of bits in the micro-instruction can be reduced.

Let N number of control signals are driven by the control unit including internal and external signals. If a separate bit is used to designate each different control signal then a total number of N dedicated bits are required for a micro-instruction. By using encoding techniques and later decoding it, the required number of bits can be reduced.

For an N bit micro-instruction, a total number of 2^N combinations of control signals are possible but not all of these combinations are used. Only a subset of possible combinations are used. Suppose only $M < 2^N$ combinations of control signals are used by a processor. These M combinations can be encoded with $\log_2 M$ bits, which is smaller than N . But this approach also has some issues that need to be considered. Firstly, a pure encoding scheme is difficult to program and on the other hand it requires a complex control logic module. The complexity makes it a slow control logic module. Therefore, some simplifications are made to a pure encoding scheme to alleviate these issues. The simplification includes using more than $\log_2 M$ bits and considering that physically allowable control signals are not possible encode. There are several encoding techniques. The most of the micro-programmed control units use some degree of encoding to simplify the micro-programming and to reduce the width of control memory. Pure un-encoded micro-programmed control units are rarely designed. The micro-instructions could be termed as vertical or horizontal depending on their width. The micro-instructions with a width in the range between 6 to 40 bits are known as vertical micro-instructions. While the micro-instructions with a width in the range 40 to 100 bits are called horizontal micro-instructions. The micro-programs are also divided into two categories: hard and soft. The hard micro-programs are generally fixed and cannot be changed. Hard micro-programs are stored in ROM. The soft micro-programs provide some degree of flexibility and are changeable to some extent.

A basic encoding technique is illustrated in Figure 8.11 in which a micro-instruction has multiple fields. Each field contains encoded information that activates control signals after decoding. Every field is decoded during the execution of the micro-instruction and control signals are generated. If micro-instruction has Q fields then Q simultaneous actions can be specified. It is desirable that a control signal should not be activated by more than one fields to minimize the width of the micro-instruction. If length of a field is K bits then 2^K codes are possible, which can be decoded into at least 2^K different control signals. But a particular field can contain only one code. Therefore, codes and actions of all the fields belonging to a micro-instruction are mutually exclusive.

Another encoding approach known as indirect encoding is illustrated in Figure 8.12. In indirect encoding, one field of the micro-instruction is used to interpret the other field. It is kind of two level decoding that is slower than the direct decoding. In this approach, different bits of field are used for different purposes. Some bits are used to indicate operation type, a set of bits is decoded to identify the operations, and the remaining bits are used to select the registers. The implementation of encoding techniques require a set of registers including program counter and other temporary registers.

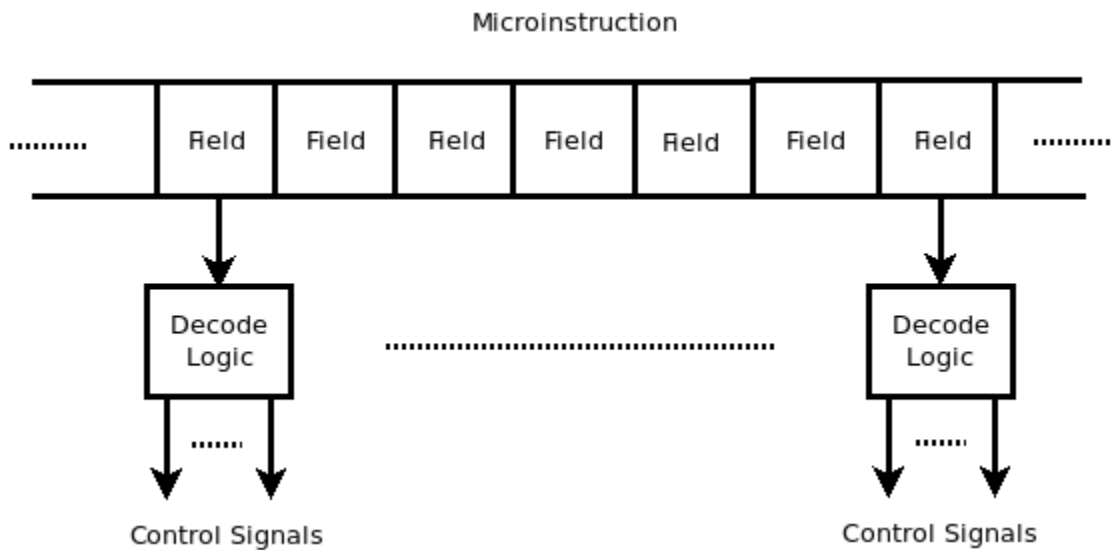


Figure 8.11: Direct encoding

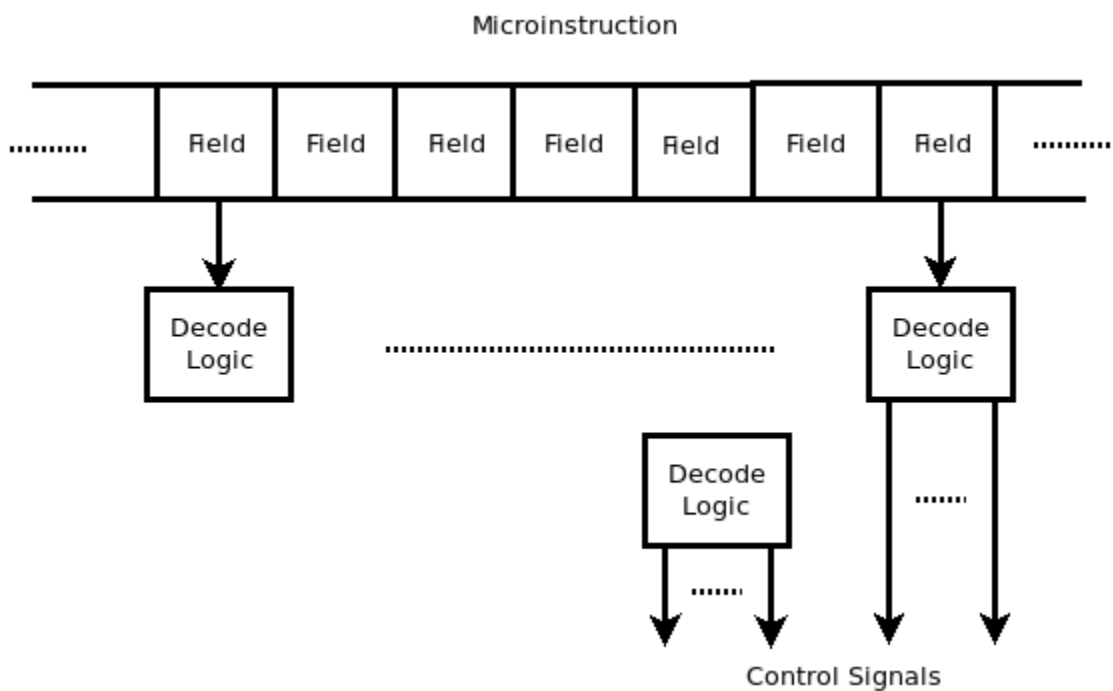


Figure 8.12: Indirect encoding

Check Your Progress: 3

1. How are the control signals generated?
2. Why are the encoding techniques used for micro-instructions?
3. Compare the direct and indirect encoding.

8.5 CPU Control Unit Organization

The organization of a micro-programmed control unit is shown in Figure 8.13. The major components of the organization include micro-instruction sequencing, CAR, control buffer register, control logic, and control memory, etc. The micro-instruction sequencing module generates the next address with the help of ALU, control flags, instruction register, CAR, and control buffer register. The micro-instruction sequencing module can be implemented as digital functions for a particular application. But a general purpose micro-programmed control can be implemented with the help of a ROM. It generates next address and sends it to control memory for the execution of the micro-instruction. The next address logic determines the source of address that is loaded to the control address register. If there are multiple sources, a particular source could be selected with the help of a multiplexer. Another multiplexer could be used for selecting the status bit. CAR provides the address for the control memory. There are different types of operations that a micro-instruction sequencing module can perform including branch, function call, function return, increment, address load, and push or pop the stack, etc. There is a clock that determines the timing of the micro-instruction cycle. Various combinational circuits can be designed to support these operations. The control signals are generated by the control logic module. The complexity of the control module highly depends on the contents and format of the micro-instructions.

Figure

8.13:

Micro-

progr

amme

d

Contr

ol Unit

Organ

izatio

n

8.6

Sum

mary

The

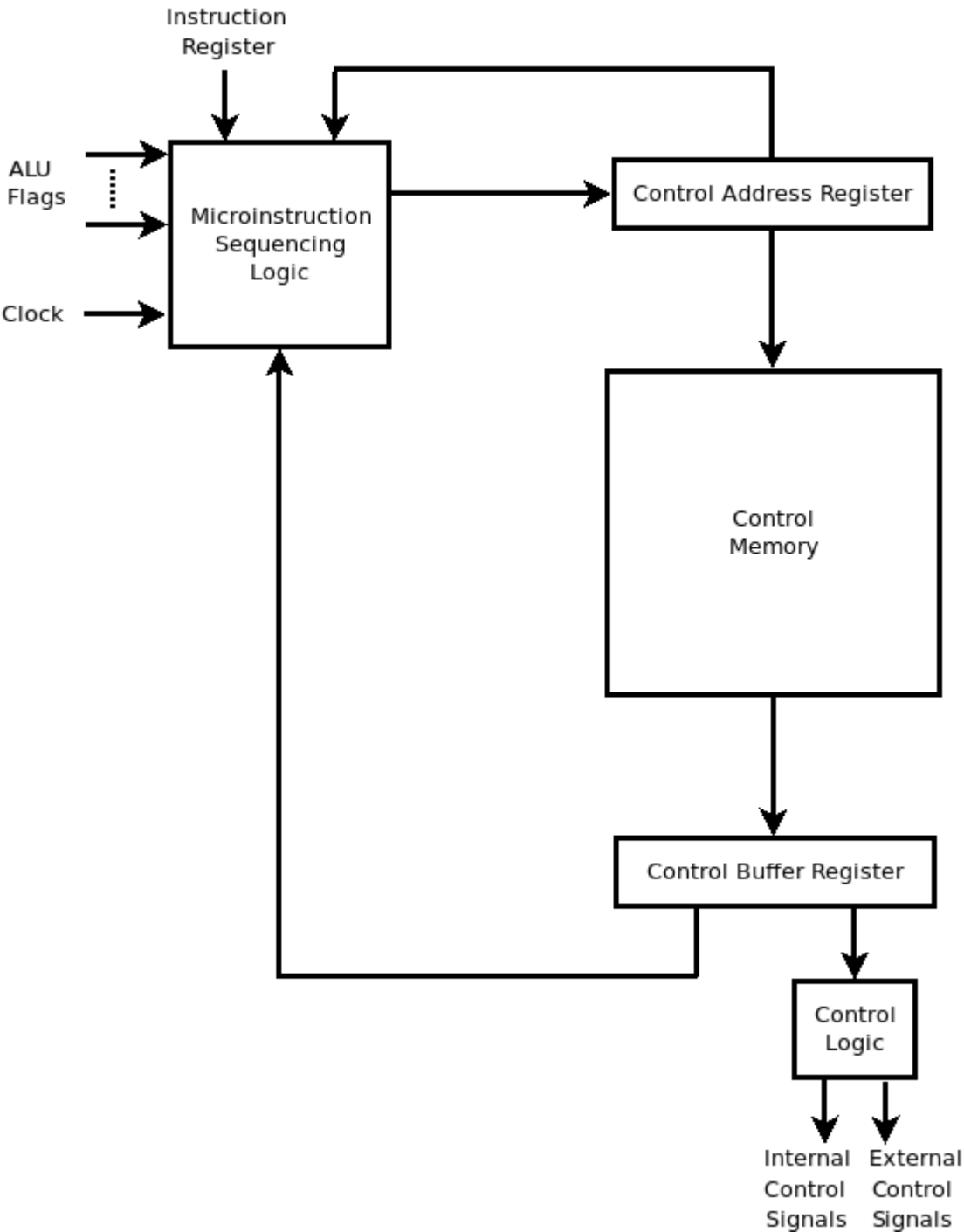
micro-

progr

amme

d

contro



l unit is an alternative approach for implementing the control unit of the computer system. It uses micro-programs to specify the control logic. A micro-program consists of a sequence of micro-instructions that specify the micro-operations. The micro-program is stored into a micro-program memory and the micro-instructions are fetched from the micro-program memory during the execution. The sequencing of the instructions during

the execution is maintained with the help of a micro-program counter. Since the micro-program is usually not changed often, the micro-program memory is a ROM also known as control memory. The micro-programmed control unit is slower than the hardwired control unit. The advantage of micro-program control unit is that its design is simple and it is easy to implement. The micro-programmed control unit uses a relatively simple logic circuit that is capable of sequencing and executing micro-instructions to generate the control signals. The contents of micro-memory specify the micro-instructions that are strings of 0's and 1's. A particular memory cell provides the state of a micro-instruction. The micro-programmed control unit is capable of sequencing and executing micro-instructions to generate the control signals. The contents of micro-memory are strings of 0's and 1's and a particular memory cell provides the state of a micro-instruction.

The smaller size of control memory and faster execution of the micro-instructions are highly desirable. The smaller control memory leads to lower cost. During the execution, address of the next micro-instruction is needed, which could be a branch address, next sequential address, or determined by instruction register. The micro-instruction sequencing generates the address of next instruction. The initial address is loaded into control address register. The initial micro-instruction initiates the instruction fetch routine. At the end of the fetch micro-program routine, the instruction register contains the next instruction. The next step is to determine the address of the operand.

There are different types of addressing techniques also known as addressing modes. Different types of operands use different addressing modes. The instruction format may have some designated bits to represent the type of the addressing mode. These designated bits are known as mode field. The control unit can use mode field to determine the addressing mode. The direct addressing limits the addressing range because the address field is smaller than word length. To overcome this limitation, the indirect addressing is used. The registers provide faster access to the information. A variety of sequencing techniques are available based on the format of address. The execution of a micro-instruction is the basic event in a processor with micro-programmed control unit. The execution of micro-instruction generates the control signal for processor as well as for the external units such as memory and peripherals. Fetch

and execute are two parts of the micro-instruction cycle. The fetch operation is related to micro-instruction address generation. The micro-instruction sequencing is related to next address generation. It generates next address using the information from ALU flags, control status bits, opcode, CAR, and current micro-instruction.

The size of micro-instructions can be reduced by using the encoding techniques. With smaller micro-instructions, size of control memory can be reduced. But encoding also introduces some complexity that leads to the slow execution of the micro-instruction. However, the most of the micro-programmed control units employ encoding techniques with some simplification strategies.

A micro-programmed control unit consists of micro-instruction sequencing, CAR, control buffer register, control logic, and control memory, etc. The micro-instruction sequencing module generates the next address with the help of ALU, control flags, instruction register, CAR, and control buffer register. The CAR provides the address for the control memory and clock determines the timing of the micro-instruction cycle. The control signals are generated by the control logic module. The complexity of the control module highly depends on the contents and format of the micro-instructions.

Review Questions

- Q.1 What are the major tasks performed by a micro-programmed control unit?
- Q.2 With the help of suitable diagram, describe the basic organization of a micro-programmed control unit. Explain all the important related terms.
- Q.3 What is the utility of control memory?
- Q.4 What is micro-instruction sequencing? Why is it required by the control unit?
- Q.5 What is addressing mode? Explain different types of addressing modes with their merits and demerits.
- Q.6 How is the next address generated for micro-instruction sequencing? Explain different types of address formats.
- Q.7 What kind of measures can be taken to reduce the control memory size?
- Q.8 Draw a logic diagram for the micro-programmed control unit and explain the role of different components.

UNIT- 9 Pipelined Control

Structure

9.0 Introduction

9.1 Objectives

- 9.2 Principles of Pipelining
- 9.3 Pipeline Classification
- 9.4 Instruction Pipeline
- 9.5 Pipeline Performance
- 9.6 Handling Data Dependency
- 9.7 Super Scalar Processing
- 9.8 Summary
- Review Questions

Unit 9: Pipelined Control

9.0 Introduction

The conventional computing systems perform operations in a sequential way. A task is assigned to the processor at a time and it remain engaged until the task is over. After the completion of the one task another task is allotted to processor. The approach is simple but it makes inefficient use of resources and takes longer time to complete the

tasks. Parallel processing is the modern approach to simultaneously perform data processing to speed up the computational speed of the computer system. It is an efficient form of data processing that takes advantage of concurrent exploitation of resources. Multiple resources can be accessed simultaneously in the same time interval. In parallel processing environment, different events may occur simultaneously or in an overlapped fashion at the same instant of time. In order to reduce the execution time, the data elements or information is processed concurrently or simultaneously. Parallel processing can take place at different levels. The highest level of parallel processing takes place at program level. It allows multiple programs to execute simultaneously by efficiently sharing the limited hardware resources. The next level parallel processing occurs at program segment level within the same program. Multiple independent segments belonging to the same program may be processed concurrently. Parallel processing can further be exploited at instruction level, where multiple instructions in a program segment can be considered for simultaneous execution. Finally, the parallel processing can be applied to the different segments of an instruction. Both hardware and software have to play an important role for achieving the parallelism in computation. The role of hardware increases from highest to lowest level of parallelism. On the other hand, algorithm or software implementation is more prominent at highest level. With the increasing hardware and software requirements, the cost of the system also increases. However, parallel processing systems are becoming affordable with the advancement in the hardware technology. Uniprocessor systems have certain limitations to promote concurrency due to single processor, single control unit, and limited number of other processing elements.

An economical realizing of the parallelism is the pipelining technique, which is inspired from the assembly line in manufacturing plants. The basic concept behind pipeline techniques is to divide a larger task into several smaller sub-tasks. Each sub-task can be assigned to a different stage operating concurrently with other stages in the pipeline. The sub-tasks belonging to different tasks can be executed in an overlapped fashion to stream out the result through the last stage in the pipeline. This approach can provide significant improvements in system throughput and speed up the execution.

There could be different types of pipelines in modern computer systems designed for specific tasks.

9.1 Objectives

After the completion of this unit, the readers will be to:

1. Understand the concept of parallel processing.
2. Understand the concept of pipeline in computing.
3. Learn the basic properties of pipeline processors.
4. Understand different segments of instruction pipeline.
5. Understand performance issues in the pipeline.
6. Understand concept of super scalar processing.

9.2 Principles of Pipelining

Pipelining is a technique that introduces some kind of parallelism by decomposing a sequential process into sub-operations and executing them in overlapped manner similar to industrial assembly lines. The individual sub-operations are assigned to separate segments, where each segment is capable of operating concurrently with other segments in the pipeline. A segment is a specialized hardware stage that partially processes the task and the output goes to the next segment in the pipeline. The final result is produced after processing the data through all the segments. Similar to industrial assembly lines, the different segments are capable of performing several computations at the same time. The segments of a pipeline can be implemented with the help of latches and combinational circuits. In a uniform pipeline all the segments have same delay. But in practice, all the segments do not operate at the same speed as they perform different types of operations. Different operations need different amount of computational work and thus different amount of time. Therefore, usually the most of the pipelines are not uniform. The slowest segment could become a bottleneck for the pipe causing other segments waiting for data. The latches are fast register that are used for holding intermediate data. The output of the combinational circuit of one segment is placed into the input register of the next segment. The combinational circuit takes data from the input register of the segment and performs the specific operation. The latches

help to provide isolation between segments so that each can operate on distinct data simultaneously. Figure 9.1 shows a linear pipeline structure (same as Figure 6.1). The data flow through the pipeline according to the slowest stage to avoid the bottleneck. The data flow is controlled by a common clock applied to all the latches as shown in Figure 9.1. The overlapping among the sub-operations in a pipeline can be illustrated with the help of a space-time diagram. A space-time diagram for a 4-stage pipeline is given in Figure 9.2. The symbol 'x' represents the idle time-space span. The busy time-space spans are marked by T_{ij} , where i represents i^{th} operation and j represents its j^{th} sub-operation. The first output is generated after 4 time units but once the pipe is filled up, it produces an output per time unit as shown in the figure.

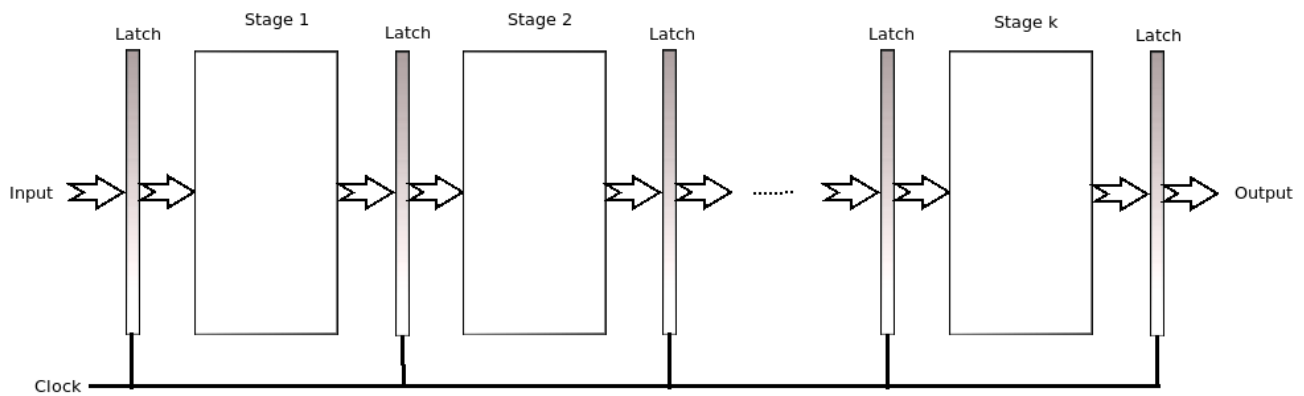


Figure 9.1: A k -stage pipeline controlled by common clock

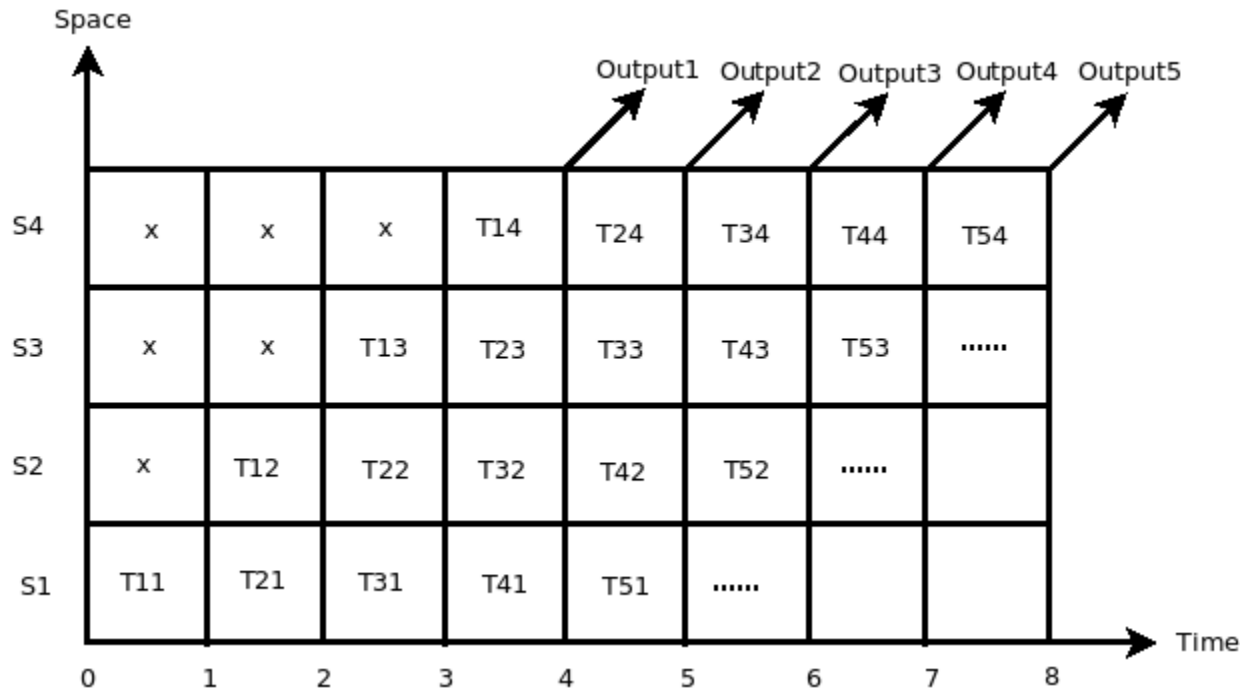


Figure 9.2: Space-Time diagram for 4-stage pipeline

There are some important parameters related to pipeline processor as discussed here.

🏢 **Clock Period:** The clock period can be defined as follows

$$t = \max\{t_i\}_{i=1}^k + t_l$$

where t_l is the delay of latch and t_i is the delay of i^{th} storage. The clock period is an important parameter that indicates the frequency with which the input can be provided to the pipeline.

🏢 **Frequency:** The frequency f of the pipeline processor is the reciprocal of its clock period given as

$$f = 1/t$$

🏢 **Speed-up:** If an operation can be divided into k sub-operations and each sub-operation takes one clock period to complete, a non-pipelined processor would take k clock periods to complete the operation. It takes $n.k$ clock periods to perform n operations. A k -stage pipeline processor takes k clock periods for the first operation but after that only one clock period is required for performing one operation. Therefore, a pipeline processor takes $n+k-1$ clock periods to complete n operations.

The speed-up of a pipeline processor over a non-pipelined processor is given as follows

$$S = \frac{n.k}{n + k - 1}$$

If there are a large number of operations of the same type then $n \gg k$ and $S \approx k$. Therefore, a k -stage pipeline can provide a speed up of at most k times.

▀ **Efficiency:** The efficiency of a pipeline is defined by the ratio of total time-space spans over busy time-space spans. The total time-space spans include all busy and idle spans. Thus, the efficiency of a pipeline is defined as follows

$$\eta = \frac{n.k}{k(n + k - 1)} = \frac{n}{n + k - 1}$$

It can be observed from the above definition that if $n \gg k$ then $\eta \approx 1$. It simply indicates that if there are large numbers of operations, the efficiency of the pipeline approaches to 1.

▀ **Throughput:** The number of results produced by the pipeline per unit time is called its throughput defined as follows

$$T = \frac{\eta}{t}$$

Since maximum value of the efficiency can be approximately 1, the maximum throughput can be given by

$$T = \frac{1}{t}$$

Therefore, the maximum throughput of the pipeline is equal to its frequency.

Check Your Progress: 1

1. What is pipeline technique?
2. Why are the latches used in a pipeline?
3. How is the clock period defined for a pipeline?
4. How much speed up a pipeline can achieve over a non-pipelined processor?
5. Define efficiency and throughput of a pipeline.

9.3 Pipeline Classification

There are different types of pipeline processors, which can be classified according to levels of processing and control strategies.

9.3.1 Arithmetic pipeline

The floating-point operations and multiplication operation on any kind of numbers are well suited to the pipeline processing as these operations can be easily divided into sub-operations. The arithmetic pipelines are designed to perform repeated arithmetic operations. Several computers have separate pipelines for floating-point and fixed-point arithmetic operations. The arithmetic pipelines work at data level.

9.3.2 Instruction Pipeline

The instruction execution in a computer system goes through several phases and the entire process can be easily divided into sub-operations. The instruction pipeline helps faster execution of a stream of instructions. The most of the high performance computers are equipped with instruction pipelines. This instruction pipeline is also known as instruction look ahead technique.

9.3.3 Processor Pipeline

In a processor pipeline, the multiple processors are in a cascade to process a stream of data. A processor carries out a specific task on the data that is stored into a memory as an intermediate result. The memory is also accessible to the next processor in the cascade. The next processor takes the intermediate result to refine it. The process of refinement goes on from one processor to another and final result is obtained through the last processor. The processor pipelines are not so popular.

9.3.4 Multifunction Pipeline

In its simple form, a pipeline is designed to perform a specific task such as floating point addition. Such a pipeline is known as unifunction pipeline dedicated to a single operation. However, there are some other pipelines that are capable of performing

different operations at different times or even at the same time. These pipelines are usually reconfigurable for a variety of operations. A particular subset of pipeline stages can be used for a specific operation, while other subset may carry out some other operation.

9.3.5 Dynamic pipeline

A unification or multifunction pipeline that assumes only one functional configuration at time is called static pipeline. The static pipelines are suitable for the tasks that require the same type of operations continuously. The function of a static multifunction pipeline should not change frequently otherwise it may perform poorly. The dynamic pipelines are essentially multifunctional and can be reconfigured frequently as there exists several configurations simultaneously. These pipelines have complex sequencing and control mechanism as compared to the static pipelines.

9.3.6 Vector Pipeline

The vector pipelines are specially designed for executing vector instructions over vector operands. The computers having vector pipelines are referred to as vector processors. Conventionally, the instructions operate on single data elements or scalars, whereas for vector processors, an instruction set is implemented that operate on data vectors. Vector processing can perform arithmetic operations on a large number of integers or floating-point numbers. Vector pipelines were widely used in the early generation supercomputers.

9.4 Instruction Pipeline

The processing of an instruction involves four major phases: instruction fetch, instruction decode, operand fetch, and execute. The complex instructions may also have more than four phases of instruction execution. In general, the instruction processing in a computer system involves the following steps:

- Instruction Fetch (IF): to read the instruction from the memory
- Instruction Decode (ID): to interpret the instruction and identify the operation to perform

- 🎬 Address Generation (AG): to calculate the effective addresses
- 🎬 Operand Fetch (OF): to read the operands from the memory
- 🎬 Execute (EX) : to actually carry out the operation on the operands
- 🎬 Store (ST): to write the result at appropriate location in the memory

It allows us to develop a pipelined instruction execution unit having multiple segments. It can read consecutive instructions from the memory while previous instructions are still being processed by other segments. However, an instruction causing a branch out of sequence may create problems. In case of a branch instruction, it needs to flush the subsequent instructions that have already been fetched. There are certain other difficulties with the implementation of an instruction pipeline. Different phases listed above take different amount of time. So that it could not be a uniform pipeline and different amount of delay between segments need to be handled to avoid the bottleneck in the pipe. Another issue is related to memory access. A single memory module is a kind of resource that can be accessed by only one entity (process or segment) at a time. Therefore, it causes memory conflict in case more than one segments of a pipeline attempt to access the memory simultaneously. Multiple memory modules and multiple data bus can be used to alleviate the memory conflicts some extent. With this arrangement, different segments can simultaneously access the data in different memory modules through different data bus.

Let us consider an example instruction pipeline with four segments. Here instruction decode and address generation are combined into one segment. Similarly, the execution and storing of the result are also combined into one segment. In this way, the six phases of an instruction pipeline can be reduced to four segments. Figure 9.3 shows the instruction cycle for a four-segment pipeline. While Segment 4 is executing one instruction, the Segment 3 is busy fetching the operands for the next instruction. At the same time, Segment 2 decodes and generates address for the third instruction and Segment 1 is fetching the fourth instruction. It can accommodate up to four instructions simultaneously in the pipeline performing different sub-operations for each instruction leading to the simultaneous progress of up to four instructions. If there is an instruction causing a branch out of the normal sequence, the pending sub-operations in Segment 3 and Segment 4 are completed and information in the remaining pipe is deleted. The

pipeline is then restarted from a new address available in the program counter register. Similar kind of action is taken if there occurs an interrupt. The information from the pipe is removed and it restarts from a new address made available through the program counter.

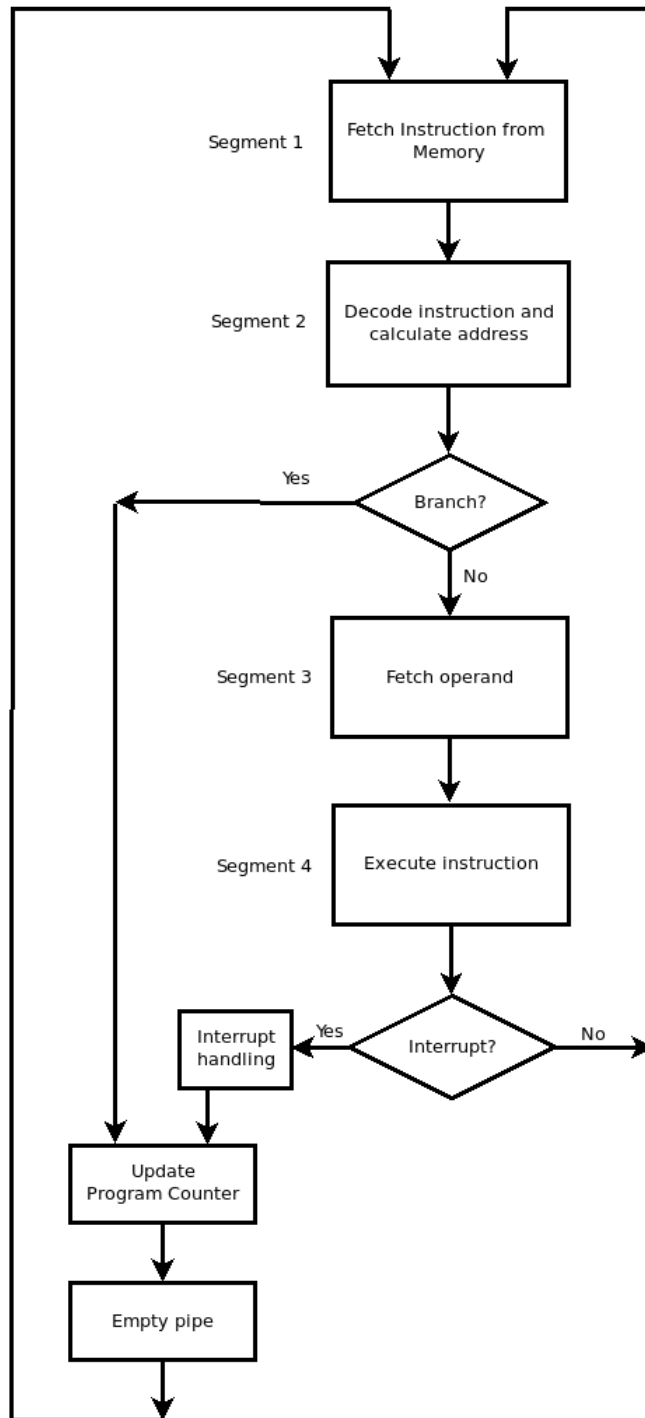


Figure 9.3: Four-Segment Instruction Pipeline

The instruction pipeline shown in Figure 9.3 is a uniform pipeline that is assumed to have equal delay for each segment. There are different memory modules for holding instructions and operands separately. It provides simultaneous access to Segment 1 and Segment 3. The operation of the pipeline is illustrated in Figure 9.4. It can be observed from the space-time diagram given in the figure that segment 'IF' for the first is performed in the first clock period. During the second clock period, segment 'ID' of the first instruction and segment 'IF' of the second instruction are undergoing simultaneously. The first output is generated after the fourth clock period. The first three outputs are produced in three consecutive clock periods. But the fourth instruction is a branch instruction, therefore the subsequent instructions are halted until the execution of the previous instructions is over. After that control is transferred to the specified instruction and subsequently other instructions are executed segment by segment and corresponding outputs are produced by the pipeline in the consecutive clock periods until another branch instruction is encountered.

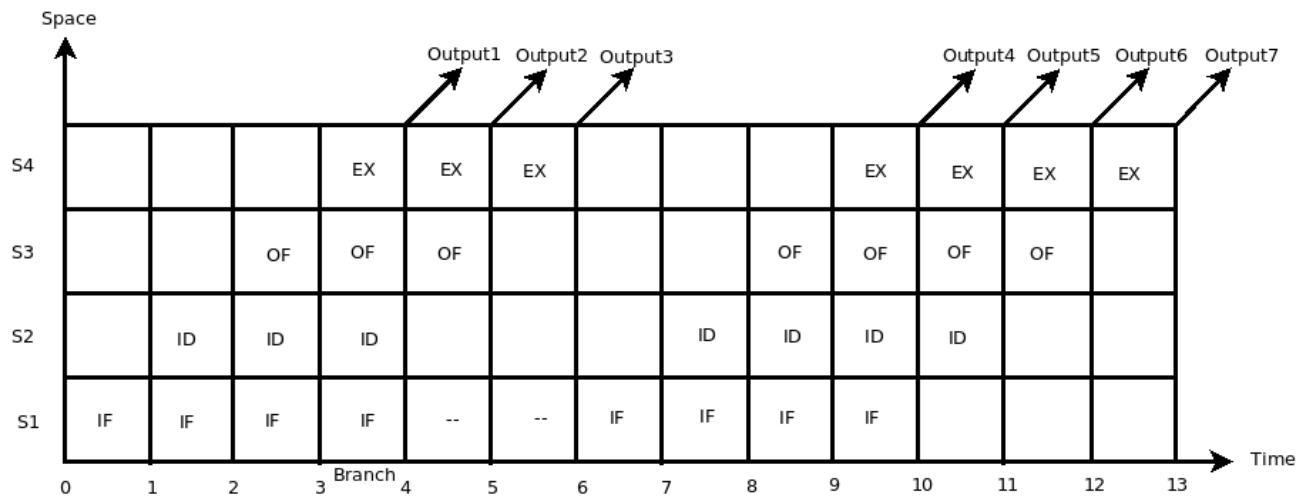


Figure 9.4: Working of Instruction Pipeline

Check Your Progress 2

1. Find out difference between static and dynamic pipelines.
2. Find out difference between unification and multifunction pipelines.
3. What are the possible phases of an instruction pipeline?

9.5 Pipeline Performance

Ideally all the instructions in the instruction sequence for a pipeline should be independent. But in a real scenario, some instructions may have certain dependency on other instructions or they may have some kind of conflicts to access the resources. An instruction pipeline may observe delay due to various reasons. The major factors that cause delay and affect the pipeline performance are as follows:

- Resource conflicts
- Branch instructions
- Data dependency

The resource conflicts occur when two different stages in a pipeline attempt to access the same resource during the same clock period. In such a situation, one of stage has to wait until the resource is release by the other stage to avoid the collision. The waiting period is reason of delay in the pipeline. The memory conflicts are major resource conflicts as memory is accessed multiple times during the execution of an instruction. For example both 'IF' and 'OF' segments read the data from the memory. The memory conflicts can be avoided by having multiple memory modules and using separate modules for instruction and data.

Occurrence of the branch instructions is a major issue that causes the delay. Use of branch instructions in programs is common. There are two types of branch instructions: conditional and unconditional. An unconditional branch instruction always modifies the sequence of the instructions, while in case of conditional branch it depends on the given condition. If condition is satisfied in the branch instruction only then the sequence is altered. The change in the sequence is done by changing the contents of the program counter with new address leading to the problems in the pipeline operations. An approach to handle the branch is to fetch the target instruction in

advance as well as instruction following the branch. In case of successful condition, the execution continues from the target instruction. Another approach generates two stream of instructions from both places until the branch decision is reached. The flow of program depends on the success or failure of the condition. Out of the two branches, one is chosen if condition is successful and other one is taken in case of unsuccessful condition.

Some hardware techniques can be employed to handle the branch instruction related issues. One such approach uses a **branch target buffer** (BTB), which is a kind of associative memory. BTB stores the previously executed branch and its target instruction along with some instructions that follow the target instruction. Whenever a branch instruction is encountered, the pipeline searches the BTB. If related information is available there, it continues from the new path. If branch information is not available in BTB, the pipeline finds the new stream and stores the target branch instruction and subsequent instructions in BTB. A variation of BTB is **loop buffer** that uses a fast register for the instruction fetch segment of the pipeline. When a loop in the program is found, it is stored in the loop buffer including all the branches. The program loop is directly executed from the loop buffer without accessing the memory.

Some computers use a concept known as **branch prediction** that uses a logic to predict the outcome of branch condition before instruction is executed. Depending the prediction, the instructions are pre-fetched that are useful in case a branch condition is met. In this way a wait time is avoided. In another approach called **delayed branch**, the compiler detects the branch instructions and rearranges the machine code by inserting some useful instructions to ensure the uninterrupted flow of execution.

Data dependency is another important factor that highly affects the pipeline performance. Data dependency occurs when the availability of source operands of an instruction depends on the results of the previous instructions. Although it is possible to fetch the instruction but its operands cannot be fetched until the result of the previous instructions are stored. In this case, an instruction cannot proceed if certain segments of the previous instruction are still incomplete. Consider the following piece of code:

I1: R3←R1 + R2

I2: $R4 \leftarrow R4 - R3$
I3: $R7 \leftarrow R5 + R6$

There three instructions I1, I2, and I3 in the above piece of code. The first instruction I1 adds the contents of register R1 and register R2 and stores the result to register R3. The second instruction I2 subtracts contents of R3 from the register R4 and stores the result back to register R4. Instruction I3 performs same operation as I1 performs on different registers. It can be observed that I2 has a data dependency on I1. The space-time diagram for the above piece of code is shown in Figure 9.5. It can be observed from the figure that all the three instructions are in the pipe during the second clock period. Ideally in the next clock period, all the instructions should have moved to their next segments. But I2 is unable to move to next segment OF because its operands are not available yet to access. The operands of I2 are R3 and R4 out of which R3 would be updated by the end of the execution of I1. The execution of I1 is over only after the fourth clock period. Therefore, I2 moved to segment OF during fifth clock period instead of fourth thus causing a delay of one clock period. It is necessary to allow I2 to be in wait state to avoid wrong fetching of operands. The instruction I3 also suffers a delay of one clock period due to data dependency between I1 and I2. The major design issues in pipeline include to avoid fetching of wrong operands and reducing the delay due to data dependency. The kind of dependency observed in the given example is called read-after-write dependency. Other possible cases are write-after-write, write-after-read, and read-after-read. An example of write-after-write dependency is given as follows

I1: $R3 \leftarrow R1 + R2$
I2: $R3 \leftarrow SR(R3)$
I3: $R4 \leftarrow R1 + 2$

Instruction I1 modifies the contents of register R3 (write operation) and subsequently I2 modifies R3 by shift right operation (write operation). The space-time diagram for the example of write-after-write dependency would be same as given in Figure 9.5.

execution of another instruction, the circuit causes delay by enough clock cycles to handle the conflicts. The sequence of instructions in a program is maintained by introducing the sufficient delay.

Another approach to resolve conflicts due to data dependencies is hardware operand forwarding. It uses a special hardware to detect the instructions whose source operands depend on the action of other instruction. If such dependency is found, the hardware directly transfers the result to the ALU input bypassing the register. The destination instruction gets the operand before the register write operation. Let us consider the example

I1: $R3 \leftarrow R1 + R2$

I2: $R4 \leftarrow SR(R3)$

The above sequence of instructions cause a delay to instruction I2 after the decode operation as the operand R3 is modified by I1. The correct value of R3 is available after I1 writes the result to R3 in the last stage of pipeline. However, it is possible to forward the result directly to the ALU before writing it to the register. In this way the delay or stall time can be reduced. This kind of data forwarding requires special hardware and it is expensive.

The data forwarding can also be achieved with software implementation. A compiler can be developed, which is smart enough to detect the subsequent instructions and their operand dependency to previous instructions. The compiler quickly makes available the results as operands to subsequent instructions to reduce the delay. The dependency can take three different forms: STORE-FETCH, FETCH-FETCH, and STORE-STORE. Consider the following example for STORE-FETCH form:

I1: $M[R2] \leftarrow R1$

I2: $R3 \leftarrow M[R2]$

In the example, the first instruction stores the contents of register R1 to a memory location whose address is stored in register R2. The same memory location is

subsequently fetched by the second instruction. However, the data needed by I2 is already available in the register R1. Therefore, there is no need to access the same memory location again and a smart compiler can replace the given sequence of instructions with the following sequence,

```
I1:  M[R2]←R1
I2:  R3←R1
```

The FETCH-FETCH form of dependency analysis is shown by the following example, where the same memory location is accessed by two consecutive instructions.

```
I1:  R2←M[R1]
I2:  R3←M[R1]
```

In this example operand needed by I1 is also needed by I2. After the execution of I1, the operand needed by I2 is now available in register R2. Therefore, a smart compiler can replace the sequence with the following code to reduce the delay.

```
I1:  R2←M[R1]
I2:  R3←R2
```

In case of STORE-STORE analysis, the memory location updated by one instruction is overwritten subsequently by the next instruction. For example,

```
I1:  M[R2]←R1
I2:  M[R2]←R3
```

The memory location whose address is contained in register R2 is updated by instruction I1, which is overwritten by the instruction I2. A smart compiler can replace the above code with the following single statement.

I: $M[R2] \leftarrow R3$

However, the above replacement is possible only if I1 does not influence the other instructions in the program. Otherwise a number of modifications would be required in the program or sometimes it is not possible at all to make such replacements.

Check Your Progress: 3

1. What kind of resource conflicts affect the performance of the pipeline?
2. Consider Figure 9.5 and determine the speed up of the pipeline.
3. Explain the utility of BTB.
4. What is hardware interlock?
5. Name the different forms of data dependency analysis.

9.7 Super Scalar Processing

A basic scalar processor issues one instruction per clock cycle, while a pipeline processor can allow multiple instructions simultaneously at different stages of execution during the same clock period. The super scalar processors on the other hand can have multiple instructions at the same stage of execution in a clock cycle. A super scalar processor implements a kind of parallel processing within the single processor. It requires multiple instruction pipelines to issue multiple instructions per cycle and to generate multiple results per cycle. They are designed to achieve more instruction level of parallelism in programs. Although, only those instructions can be executed in parallel which are independent of each other. In practice, the super scalar processor are allowed to consist of 2-5 instruction pipelines. Figures 9.6 shows the functioning of a super scalar processor with two instruction pipelines. It allows two instructions at the same stage during each clock period. It generates two outputs per clock period. Ideally in a super scalar processor with n-instruction pipelines, n instructions should be

processed per clock period. But in practice there are certain issues that cause break in the flow of instructions. These issues include data dependency, address dependency, branch control, and structural hazards, etc.

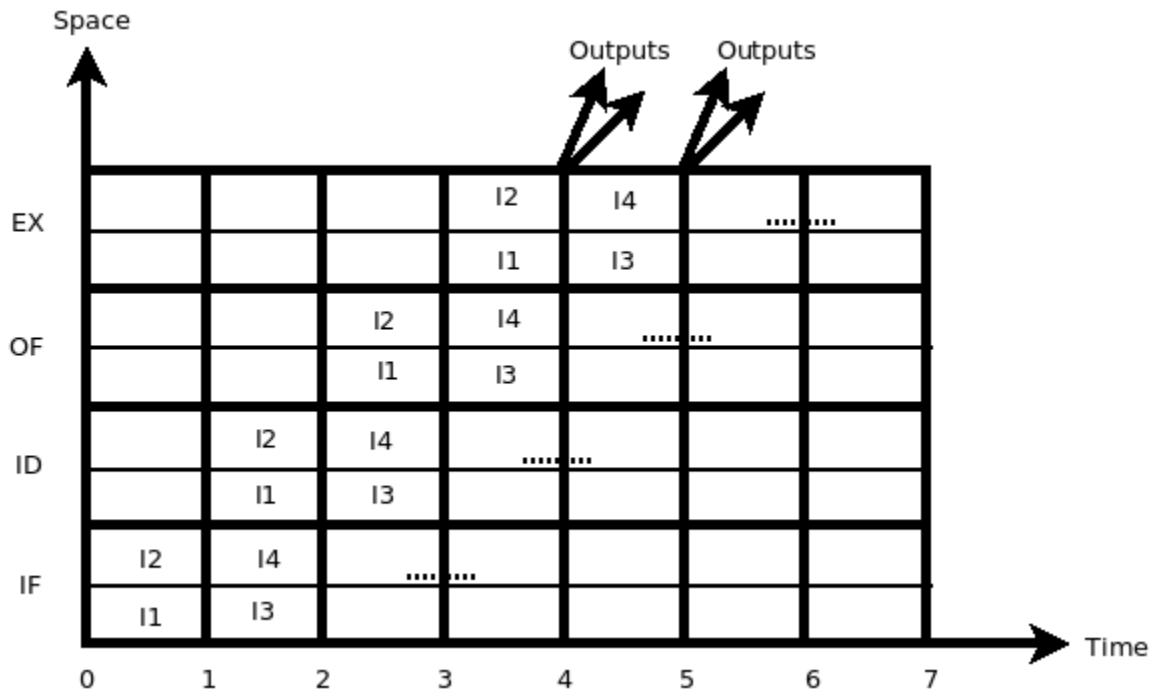


Figure 9.6: Space-time diagram for super scalar processor with two instruction pipeline

The implementation of the super scalar processing involves some challenges. The fetch operation requires advanced techniques to perform multiple cache line lookups. Since it is not possible to provide parallel access to a single memory module, therefore memory interleaving is used to provide parallel access. The memory interleaving is a technique that uses multiple smaller memory modules instead of a large memory block. With memory interleaving, consecutive instructions or programs can be stored in different memory modules that can be accessed simultaneously. The branch operations become more complicated that require fast prediction so that a proper flow of the program can be decided to keep consecutive instructions together. For improving the performance of the system, a specialized instruction cache is used that is known as **trace cache** or **execution trace cache**. The trace cache stores the instructions that are already fetched

and decoded to increase the instruction fetch bandwidth. Traces are dynamically created as instructions are fetched and branches are resolved. Traces can be start with any static instruction and a trace is identified by the starting instruction. When a trace is encountered again, the instructions can be obtained directly from the trace cache. The additional hardware support and logic are required to fetch those branch instructions that are not available in contiguous memory locations.

As discussed earlier that a branch can deviate the flow of the instructions, which affects the performance of the pipeline. If information about the branch is available in advance then performance of the pipeline can be improved. A digital circuit known as **branch predictor** is used to predict the direction of the branch before it is definitely known. The branch predictors are very useful in super scalar processors that are highly crucial to improve the flow in instruction pipeline. But sometimes, the prediction of the branch predictor is wrong leading to more delays in the pipeline. Therefore, advanced branch predictors are required for such processors to avoid the mispredictions.

Other issues in super scalar processing are related to instruction decoding, instruction issue rate, and instruction execution. Instruction decoding in scalar processor is a complex task as multiple instructions are decoded in parallel. The higher instruction issue rate makes it more complex as decoding cycle becomes lengthy. A higher instruction issue rate is good for the better performance of the system but it gives rise to other issues related to control and data dependencies. Prerecording can alleviate the problem to some extent. The instructions can be partially decoded while loading the instructions into instruction cache. Parallel execution of instructions also creates some issues such as some instructions may get completed out of order because of unequal execution times. So special measures are needed to preserve the logical sequential consistency.

9.8 Summary

The efficient use of resources is highly important for the desirable performance of a computer system. Conventionally, the instructions of a program are executed in a given sequence. An instruction starts executing only after the execution of the previous instruction is completed. The conventional approach is simple with easy implementation

but it makes inefficient use of resources as a number of resources remain idle till the instruction under execution is not finished. It is highly desirable to fully exploit the available resources for a better throughput. Parallel processing techniques allow multiple instructions or programs to run simultaneously/concurrently to reduce the idle time of the computing resources. Parallel processing can be achieved at different levels of the programming. Pipeline is an economical approach of the parallel processing that allows multiple instructions to execute simultaneously in an overlapped fashion.

The operation of an instruction can be divided into multiple sub-operations. The sub-operations of different instructions can be performed in an overlapped way similar to the industrial assembly lines. The pipeline processor consists of different segments capable of performing specific sub-operations. The output of one segment is input to the next segment in the pipeline. During a particular clock period, different sub-operations belonging to different instructions are in the pipeline. The pipeline segments are implemented as a combinational circuits. All the segments are interconnected with help of latches. A latch works as an interface between the segments that holds the intermediate results before transferring it to the next segment. The latches are controlled by the same clock. Each segment has its own delay. The delay of all the segments is not uniform. Therefore, clock period is decided according the slowest segment in the pipeline to avoid any kind of data collision and bottlenecks. A pipeline having k -stages can provide a speed up of at most k times. The maximum speed up can be achieved if there are a large of operations of the same type to perform. There are different types of pipelines such as arithmetic pipeline, instruction pipeline, unifunction/multifunction pipeline, static/dynamic pipeline, and scalar/vector pipeline, etc.

The processing of an instruction involves six major steps: instruction fetch, instruction decode, address generation, operand fetch, execute, and store. Some of these steps such as instruction decode and address generation can be combined and accordingly an instruction pipeline can be developed with four or more segments. The working of the pipeline processor can be illustrated with the help of space-time diagram. For the best performance of the pipeline, all the instructions should be independent but in reality some kind of dependences exist among the instructions. The data dependency

and branch instructions degrade the pipeline performance. Therefore, special measures are required to handle the dependencies for a better performance.

A pipeline processor can have only one instruction at a stage/segment in a clock period. The super scalar processors have multiple pipelines and allow multiple instructions at the same stage in the same clock period. They are designed to achieve more instruction level of parallelism in programs. Although, only those instructions can be executed in parallel which are independent of each other. In practice, the super scalar processor are allowed to consist of 2-5 instruction pipelines. Ideally in a super scalar processor with n-instruction pipelines, n instructions should be processed per clock period. But in practice there are certain issues that cause break in the flow of instructions. These issues include data dependency, address dependency, branch control, and structural hazards, etc. Special measures are required to handle these issues.

Review Questions

- Q.1 What could be reasons of bottleneck on pipeline? Define and derive various performance parameters for a pipeline processor.
- Q.2 Briefly describe different types of pipeline processors.
- Q.3 With the help of a flow chart, explain the working of an instruction pipeline.
- Q.4. How do branch instructions affect the pipeline performance? Explain with the help of a suitable example.
- Q.5 What are the important measures to handle the branch instruction related issues? Explain.
- Q.6 What do understand by data dependency between instructions? How are these issues resolved in the pipeline?

Q.7 What is super scalar processing? What are major challenges with super scalar processing?



**Uttar Pradesh Rajarshi Tandon
Open University**

Bachelor of Computer Application

**BCA-EC
Computer Architecture**

Block

4

MEMORY ORGANIZATION

Unit 10

Memory Technology

Unit 11

Memory Systems

Unit 12

Caches

BLOCK INTRODUCTION

The memory is essential component of any computer system. It provides the storage for instructions and data. Both instructions and data are transferred between memory and processor as and when required in the system. There are different types of memories based on different technologies. Each memory possesses different characteristics related to data accessing speed, storage, and cost, etc. The role of memory in a computer system and various memory technologies are discussed in Block 4. It is divided into Unit 10, Unit 11, and Unit 12. The concepts of memory, memory devices, and technologies are discussed in Unit 10. The memory management is the main focus of the Unit 11. The Unit 12 primarily deals with the cache memory and related issues like placement, replacement, and updating policies, etc.

UNIT- 10 Memory Technology

Structure

10.0 Introduction

10.1 Objectives

10.2 Memory Device Characteristics

10.3 Random Access Memory

10.4 Read Only Memory

10.5 Magnetic Disk

10.6 Optical Disk

10.7 Magnetic Tape

10.8 Flash Memory

10.9 Associative Memory

10.10 Cache Memory

10.11 Memory Hierarchy

10.12 Summary

Review Questions

Unit 10: Memory Technology

10.0 Introduction

The memory provides the storage for instructions and data for the processor. Both instructions and data are transferred between memory and processor as and when required. It is essential component of any computer system. The instructions that are required for processing are stored in the memory. The memory is divided into small cells, each cell is identified by unique address varying from zero to some maximum number depending on the size of the memory. The memory should be large enough to hold all the programs used in a computer. The programs like operating system, data processing software, and user programs, etc. all reside in the memory. It is desirable that memory size should be larger than the space required for accommodating all such programs for the better performance of the system. There are different types of memories based on different technologies. Each memory possesses different characteristics related to data accessing speed, storage, and cost, etc. Some memories are volatile as they can store data or instructions as long as power supply is available and contents are lost if power is off. Such memories usually provide faster data access. Other memories are slower but can accommodate data permanently irrespective of power is available or not. Since, the processor receives data or instructions through the memory, the data processing speed of the two should be compatible. A slower memory is a kind of bottleneck in the system that greatly reduces the processor efficiency. But the faster memories are higher in cost and it is not possible to afford larger memories for an economical system. On the other hand, the users need to store their programs and data permanently in system. Therefore, a computer system does not use one type of memory, but different types of multiple memories are used due to their different characteristics and cost factor. There is a complete sub-system of memories in the computer system having different types of memories of different sizes to make a

balance in the cost and accessing speed to fulfil the different needs of storage in the system. A detailed description of different types of memories and their technologies is provided in this unit. Depending on the characteristics and usage, there are four types of computer memories:

- Primary memory
- Secondary memory
- Tertiary memory
- Off-line memory

The **primary memory** is also known as main memory of the computer. It is directly or indirectly connected to the processor of the computer through memory bus. The processor continuously reads and writes data and instructions to the primary memory as required during the execution. The primary memory is faster, volatile, and usually smaller in size. It cannot provide permanent storage to the data. The **secondary memory** is not directly connected to the processor and it is accessed through some input/output channels to access or transfer the data. It is usually non-volatile memory that provides a permanent storage. The secondary memory is larger in size but much slower than primary memory. The secondary memory is not accessed directly by the processor during the execution, instead data or programs are transferred to primary memory. The **tertiary memory** is very slow and large size removable mass storage devices that are robotically mounted/unmounted to the computer to provide the storage. It is used to archive the data that is not frequently required. It is useful to store extraordinarily large data. The **off-line memory** is also known as **removable storage** is the memory that is not under the control of the processing unit. It requires human intervention to be connected before a computer can access it. It can be connected/disconnected as and when desired by the user.

10.1 Objectives

After the completion of this unit, the students would:

1. Have a better understanding on the role of memory in program execution.
2. Understand different types of computer memory devices and their characteristics.

3. Know the technologies used in different types of memories.

10.2 Memory Device Characteristics

Computer memory is the storage space of a computer system, where data and instructions required for processing are stored. It is an essential part of the computer, which plays an important role in the processing, saving, and retrieving data or instructions. The performance of the system highly depends on the memory. There are many types of memory devices available for modern computer systems. These devices took several decades to be developed and they significantly differ in characteristics and technology. Some memory devices are low cost and can be used for data backup. The devices that provide backup storage are called **auxiliary memory**, which are usually magnetic disks. The backup devices are used to store large data files, programs, and other information. Other memory devices provide faster access rate and therefore such devices are more suitable for directly communicating with the CPU. The memory that directly communicates with the CPU is known as **main memory** of the system. Only those programs, data, or instructions that are currently needed by the processor reside in the main memory. The data or programs are transferred between auxiliary memory and main memory as and when required as main memory is smaller in size and cannot store all the programs and data. The memory devices are characterized by the following major features:

- 🎬 Access method
- 🎬 Volatility
- 🎬 Portability
- 🎬 Cost
- 🎬 Speed

Any type of memory is characterized by the smallest addressable unit, which could be a byte or a word. The byte addressable memory uses an 8-bit unit as the smallest addressable unit. In word addressable memory, the smallest word could be usually 16 bits to 64 bits in length. The memory can be viewed as a collection of a large number memory locations. Each location is identified by a unique address. In order to access data from the memory, first its location should be identified and only then it could be

read from that location. There are four major methods to access the memory: sequential access, direct access, random access, and associative access.

10.2.1 Sequential Access Method

The sequential access method assumes that memory locations can be accessed sequentially only. The storage device must be searched from the beginning until it finds the required data. The method starts from the beginning and reads through in order to the desired location. The access time depends on the data location and previous location. The data are read or written in a sequential manner. If a new data element is to be stored, it is written after the previously read/written location. Some early memory devices such as tapes could have been accessed only in a linearly sequential.

10.2.2 Random Access Method

In the memory devices that uses random access method, each memory location is identified by a unique address. Any memory location can be accessed by using its address. All the memory locations can be reached in any order in the same amount of time. The address of the desired location is provided by a decoder. It provided the faster access to the data compared to sequential as well as direct access method.

10.2.3 Direct Access Method

This approach is a compromise between sequential access and random access methods. Here, the memory is divided into blocks and each block has a unique identity. The access is provided by directly jumping to the vicinity and then performing sequential search. The access time depends on the location within the block and previous accessed location. The memory device is implemented as a rotating disk that is divided into information tracks and each track having its own read/write head. To read/write data elements, the heads are positioned at appropriate track and then read/write operation is performed at appropriate place.

10.2.4 Associative Access Method

It is also known as content addressable access method. In this method, the memory location is identified by its contents instead of its physical location. The memory is implemented as a chip in which each bit can be compared. The contents are compared with each bit allowing a very fast memory access. If a new data element is needed to store in the memory, it is randomly stored as the entire memory can be compared at once.

10.2.5 Volatility

The volatile memory requires power to maintain the stored data but when power is off or interrupted the entire data or information is lost. Non-volatile memory can maintain the stored data even in the absence of power. The volatile memories are usually faster than non-volatile memories and good for the security of the data is quickly lost as soon as power is off. But it cannot be used as storage or backup device. The mass devices are non-volatile, which are slower but good for data storage.

10.2.6 Portability

The portable memory devices provide an option for additional storage backup. These devices are not integral part of the computer and serve as external memory devices. They are like plug and play devices that can be used to carry data from one place to another or to transfer data from one computer to another. The portable memory devices are mostly non-volatile. Nowadays a number of portable storage devices are available such as flash drives, external hard drives, USB drives, and CD/DVD, etc.

10.2.7 Cost and Speed

The cost and speed of the memory are important factors for any computer system. Keeping the cost affordable for the masses is important for making the system available for a large section of the society as well as from business point of view. On the other hand, memory speed is important for the desirable performance of the computers. However, both cost and speed are proportional to each other. Higher speed leads to the higher cost. The size is another factor that is highly influenced by the cost. A sizable

memory is required for accommodating the data and instructions for the processing. But increasing the size increases the cost. So it is a challenging task to make a balance among cost, size, and speed of the memory.

Check Your Progress 1

1. What is computer memory?
2. Describe auxiliary memory and main memory.
3. Differentiate primary memory and secondary memory.
4. What is offline memory?
5. What is the use of tertiary memory?
6. Arrange different accessing methods in increasing order of access time.
7. How are cost, size, and speed of the memory related?
8. What is volatile memory?
9. How does the associative memory work?

10.3 Random Access Memory

Random access memory or popularly known as **RAM** has a distinguish quality that data can be read or written rapidly irrespective of the location in the memory. It is also called read-write memory. The access time does not depend on the location. The data is read or written directly from a given address regardless of location. The same amount of time is required to access any location. Since access time is location independent, it is easy to reach each location inside the memory. It is a volatile memory that needs a constant power supply. If power is interrupted, the entire data is lost. Therefore, RAM only provides a temporary storage. Therefore, an uninterrupted power backup is always required for smooth functioning of the computer. RAM provides a fast and random access but it bears a higher cost. The semiconductor integrated circuit based technology is principally used for RAM. Usually RAM is used as a primary memory of the computer due to its read/write capabilities and faster and easy access. There are

two conventional variations of RAM available: **dynamic RAM** and **static RAM**. The primary difference between the two RAMs is the lifetime of stored data.

10.3.1 Dynamic Random Access Memory

The dynamic random access memory (DRAM) is implemented as memory cells and each memory cell composed of one transistor and one capacitor. The memory cells store data as electric charge on capacitors. Since capacitors have a characteristic to discharge, DRAM must be continually refreshed to maintain the data. DRAM consists of a refresh circuit known as controller that periodically rewrites data thousands of times per second. By refreshing the memory before data expire, the memory contents can be kept alive as long as they are needed. The term dynamic refers this tendency of losing charge even with continuous power supply. DRAMs are high density as only one transistor is required for one memory cell and therefore, DRAMs are less expensive. There are several types of DRAMs such as synchronous DRAM (SDRAM), which uses a clock signal to synchronize all read/write operations. Double data rate (DDR) memory is a kind of DRAM that performs two memory operations per clock cycle, one operation on rising and another on the falling off the clock signal. Another type of DRAM is quad data rate (QDR) memory that is twice as fast as DDR. It performs four operations per clock cycle.

10.3.2 Static Random Access Memory

The static random access memory (SRAM) retains the data as long as electrical power remains applied or not interrupted. When power gets turned off, the data is lost due to volatile nature. In SRAM chip, each bit is implemented with the help of six transistors. Since transistors do not require power to prevent leakage and there are no capacitors in SRAM chip, it is not required to refresh the memory periodically. The memory density of SRAM is lower than DRAM. The cell size is larger and it causes more power consumption. Due to lower density, SRAM needs more chips leading to higher manufacturing cost. However, SRAM is much faster than DRAM. Therefore, DRAM is preferred for larger memory requirements and SRAM is used for faster memory.

10.4 Read Only Memory

The read only memory (ROM) is a non-volatile memory that stores the data permanently. The memory cells in ROM can be accessed randomly irrespective of their locations. In that sense, it is also random access but it is not RAM. Only the access method is similar for ROM and RAM. Once data is stored, no power supply is required to maintain it. Data stored in ROM can be read but it cannot be modified or new data cannot be written. ROM is used to permanently store those programs or information that do not need any change once the manufacturing of the computer is completed. Initial programs such as **bootstrap loader** are kept in ROM. The bootstrap loader is a program that is required to imitate the operating system when the computer is powered on. Some other small programs known as **firmware** are also stored in ROM, which are used devices like BIOS. These programs are used by the computer system to perform some basic operations. ROM is also implemented like an integrated chip with programs or data fabricated into it. The data/programs are actually burned into circuitry. A very careful process of manufacturing is required as an error even in a single bit results in a useless device. There are various types of ROMs including PROM, EPROM, and EEPROM.

10.4.1 Programmable Read Only Memory

Programmable read only memory (PROM) is also a non-volatile memory and the contents are fixed once they are written just as in case of ROM. The stored data can be read electrically any number of times. But unlike a ROM, the contents in PROM are not written at the time of manufacturing. Instead, the user can store the programs or data later when it is needed to do so. The information is stored electrically through a writing or programming process with the help of a special device. However, it can be written only once and contents cannot be modified later. PROM provides the users a kind of flexibility and convenience to write the memory.

10.4.2 Erasable Programmable Read Only Memory

Erasable programmable read only memory (EPROM) is a kind of PROM that can be written multiple times. It is also a non-volatile memory whose contents are read or

written electrically just like a PROM. But it can be erased if required. Therefore, its contents can be modified unlike a ROM or PROM. However, each time an EPROM is erased to restore to its initial state before writing or programming. A special device can erase the EPROM with the help of ultra-violet light passed through a quartz crystal window. The process of erasing takes a lot of time up to several tens of minutes. It is more expensive than PROM but it can be erased multiple times and new data can be stored.

10.4.3 Electrically Erasable Programmable Read Only Memory

Electrically erasable programmable read only memory (EEPROM) provides the flexibility of writing the memory again without erasing the previous contents. It is a non-volatile memory that combines the characteristics of non-volatility with the option to reuse with updated contents. It is erased or programmed electrically. Only one byte can be erased at a time instead of the entire chip. Therefore, process of erasing and re-programming is slow. It is more expensive than EPROM and stores less amount of data per chip.

10.5 Magnetic Disk

A magnetic disk is a circular plate of metal coated with magnetized material just like a gramophone record. The magnetic disk used in computer consists of a collection of platters stacked on a spindle as shown in Figure 10.1, which rotates at 5400 to 15000 revolutions per minute. Each platter is covered with a magnetized material similar to a material used in cassettes or video tapes on both sides. Each platter is divided into concentric circles known as tracks. Each track is further divided into small sections called sectors as shown in Figure 10.2. There are spots on the magnetized surface along the tracks and sectors. The data bits are stored in these spots. A read/write head is available on each surface. A read/write head is an electromagnetic coil mounted on a movable arm. The tracks that are exactly above or below each other form a cylinder. All the tracks in a cylinder can be read without moving the read/write arm. The entire arrangement is permanently sealed for smooth functioning of the drive.

The tracks near the circumference of the disk are longer than the tracks closer to the centre. So, some tracks may have more recorded bits than other tracks if bit are

recorded with equal density. The sector is a minimum addressable unit in a magnetic disk. Whenever a particular byte is needed for an operation, the operating system locates the platter, track, and sector containing that byte and the read/write head is positioned in the identified sector. When the read/write head reaches the specified sector, the entire sector is read into main memory. Therefore, instead of storing a file in several tracks on the same platter, it should be stored in a cylinder i.e. in multiple tracks belonging to different platters above/below each other. The disk may use multiple heads to simultaneously transfer data from multiple tracks for providing a faster access.

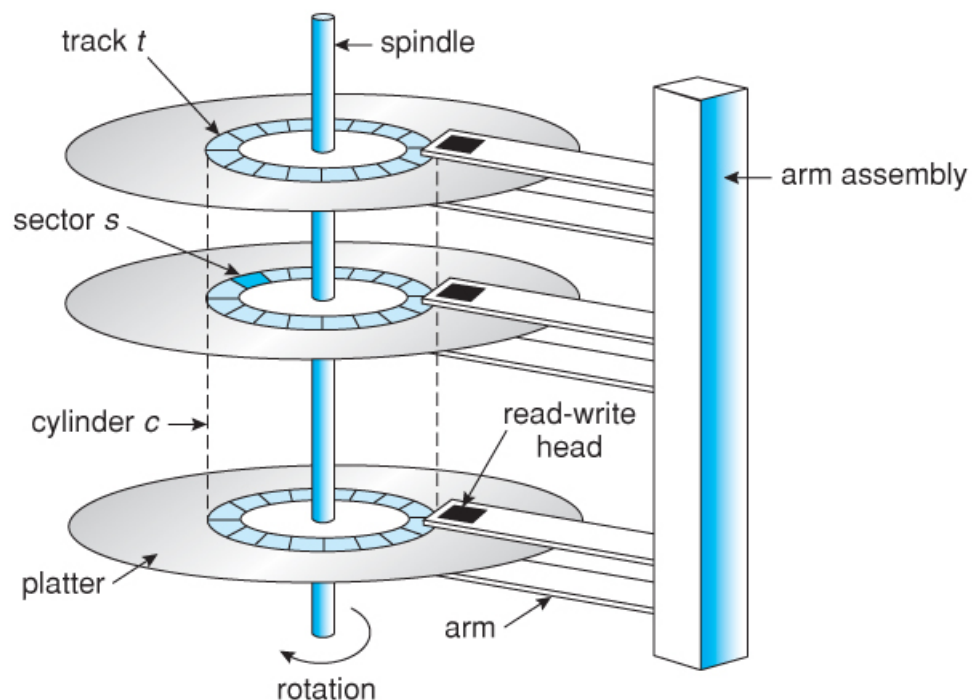


Figure 10.1: Mechanism of a magnetic disk

(Source: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/10_MassStorage.html, accessed on 16-07-2020)

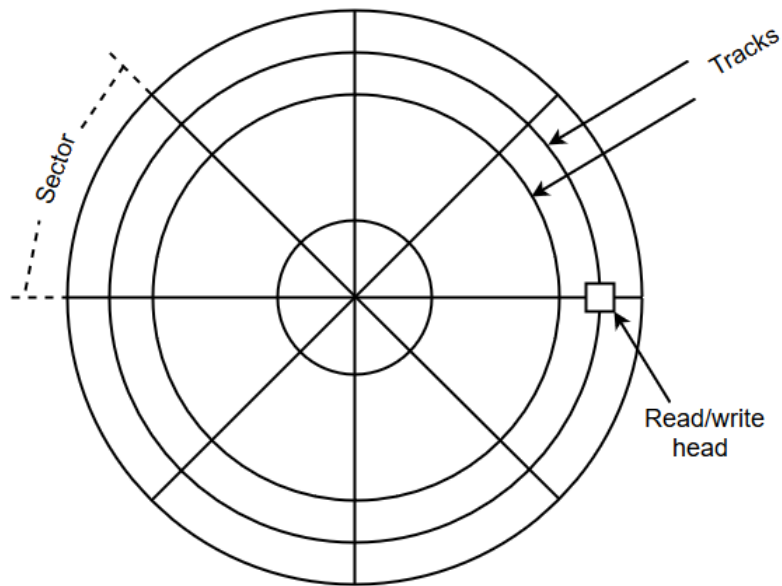


Figure 10.2: Division of platter surface in tracks and sectors

(Source: Computer System Architecture, 3/e, M. Morris Mano, 1993)

Some important parameters related to magnetic disk can be defined as follows:

- Number of cylinders: the number of cylinders is equal to number of tracks in a platter as a cylinder contains vertically aligned tracks from all the platters.
- Track capacity: it is defined as the number of sectors in a track multiplied by the number of bytes per sector.
- Cylinder capacity: the cylinder capacity is obtained by multiplying number of platters by the track capacity.
- Drive capacity: the drive capacity is given by the number of cylinders multiplied by the cylinder capacity.
- Seek time: the time it takes to move the read/write arm to the correct cylinder.
- Rotation time: the time it takes to move the desired sector under the read/write head.
- Access time: it is obtained as an addition of seek time and rotation time.
- Transfer time: the time it takes to transfer the data once the read/write head is placed at desired data.

The sectors are typically 512 to 4096 bytes in size. Some overhead information is stored at the beginning of the sector such as sector address, track address, gap between sectors, and status (defective or not). The disk tracks may also be divided into blocks instead of sectors. The number of blocks is defined by the user. The blocks are further divided into sub-blocks. The amount of data transferred in an I/O operation depends on the software design. Such a disk is called a block organized disk and disks with sectors is known as sector addressable disk.

Check Your Progress 2

1. Which of SRAM and DRAM has higher density?
2. Which of SRAM and DRAM is faster?
3. What are different types of DRAM?
4. Differentiate RAM and ROM.
5. What is utility of ROM in a computer system?
6. What are different types of ROM?
7. Define track, sector, and cylinder in a magnetic disk.
8. What is role of read/write arm and read/write head in a magnetic disk?
9. Define seek time, rotation time, and access time for a magnetic disk.
10. What is the minimum addressable unit in a magnetic disk?

10.6 Optical Disk

Optical disk storage devices store data bits as variations in light reflections. The data bits are accessed using a laser assembly. The storage medium is in the form of a rotating disk. The disks are usually pre-formatted for positioning the optical head. The information is recorded on the media as a change in the material characteristics using a thermally induced effect. Optical radiation is used as a thermal source. Optical disks have higher density than magnetic storage and provide a reliable and removable storage medium. Unlike a magnetic storage, the optical recording and reading can be done by positioning the optical head far away from the storage medium. It helps the

medium to be removable without head crash making it a reliable removable storage medium. However, longer distance between optical head and storage medium leads to higher access time. There are various types of optical disks such as compact disk, digital video disk, and blue-ray disk.

Compact disk (CD) is a read only storage made of clear polycarbonate plastic covered with a very thin layer of aluminium. For the protection of the disk another thin layer of acrylic is sprayed over the aluminium layer. CD has single track starting from the inside of disk near centre to the outside. The data bits are recorded in digital form as a series of pits and lands. A pit is a little depression forming a lower level in the track and the land is a flat part or upper level between pits. The data recording and reading is done by shining a laser beam at the disk and detecting changing reflection patterns. The speed of reading is relatively higher such as 24X, while writing the disk is much slower often 2X, where X represents a speed of 150 KBps (also known as CD audio speed). The data on the CD cannot be modified or erased once it is burned. The early version CDs had to burn in a single session, wasting the unused space of the CD if any. These drives are known as single-session CDs. Later, multi-session CDs were evolved that allow the burning of the disk in multiple sessions until it is full. A special burner software is provided with the CD for burning the disk. The data storage capacity of a CD is around 0.7 GB. A rewritable version of the CD called **CD-RW** was developed using amorphous material. A laser beam is used to heat up the amorphous substance, which becomes crystalline when cooled slowly. The crystal areas are reflective of light but amorphous area are not. Using the reflective properties of crystalline substance data are recorded and read using a universal data format.

Digital video disk (DVD) or digital versatile disk is based on the CD technology but uses smaller pits than CD media. DVDs are packed more densely and therefore, they can store larger amount of data than CDs. Both single sided and double sided formats are available. It can store around two hours of video or 4.7 GB data on one side of the disk. The total data storage capacity of the DVD could be up to 20GB. It uses a universal disc format (UDF) for writing data. A sector is the minimum addressable unit in UDF and a sector address space forms the volume.

Blue-ray disk (BD) was developed by Blue-Ray Association, which uses blue-violet laser to read or write data. The blue wavelength allows very compact tracks on the disk and therefore, a huge amount of data can be stored. A BD can store five times data than a single sided DVD close to 25 GB. Like a DVD, the BD can also be recorded in dual-layer format allowing a storage capacity of around 50GB. BD is available both in read only and rewritable formats. With doubling the rotation rate of CD/DVD, the BD provides a 5X higher data rate.

10.7 Magnetic Tape

A tape storage is a plastic strip coated with magnetic recording material. For storage purpose, the tap is wound on a reel. To access or write the data, the tape is unwound and passes through a read/write head. As tape crosses the read/write head, the data are read or written onto the tape as magnetic spots along several tracks. Typically seven or nine bits are used along with a parity bit to form a character. The parity bit is used for error checking. There are read/write heads one for each track. The information is recorded in blocks, each data block is a sequence of contiguous records. A record is a data unit that a user program deals with. The tape can start, stop, and rewind. There are gaps between the blocks where tape can be stopped. The read/write head can read the entire block of records at once. The major disadvantage of the tapes is that they are accessed sequentially and cannot directly access a given location. The access time depends on the data location and previous location. But tapes provide an inexpensive storage option, which can store a large amount of data. Tapes are a good choice for archival storage where we want to store data for a long time and we do not need to access it very often.

10.8 Flash Memory

Flash memory is a mix of EPROM and EEPROM technologies. Flash memory devices are high density, low cost, non-volatile, fast, and electrically reprogrammable. Like EEPROM, it is electrically erasable and it uses one register per cell like an EPROM. Instead of a byte-by-byte operation, a large chunk of the memory could be erased at once that gives it a name “flash memory”. Each memory cell consists of a transistor and

a floating gate similar to EPROM but oxide between floating gate and silicon is thinner that allows it to be electrically programmed and erased. The information in an array of memory cells made from floating-gate transistors. Traditionally each cell stores only one bit of information but some modern flash memory devices can store multiple bits in a single cell. The entire memory or some blocks can be erased in a few seconds. Flash memory storage is commonly used in electronic devices such as digital mobile phones, digital cameras, music players, and other portable devices, etc.

10.9 Associative Memory

The conventional memories in computers are address based memories. A number of applications require the search of data elements stored in the memory. There are various search algorithms that access a set of memory locations to find out the data. The number of access to depends on the location of the data element. Various efforts have been made to reduce the search time. The search time can be significantly reduced if the memory location could be identified by its contents instead of location address. A memory that is accessed by its contents is known as **content addressable memory (CAM)** or **associative memory**. In this type of memory, locations are accessed simultaneously in parallel based on their contents. Whenever some data are required to store in the memory, no memory address is specified. The memory finds a vacant location and writes the data. In order to read a word from the memory, contents of the word or part of it is specified. The memory locates all the matching words and marks them to read. All the memory cells have storage capabilities as well as logic circuit for matching that makes CAM more expensive than random access memory. CAM is highly suitable for the parallel processing due to its searching abilities and faster access.

Check Your Progress 3

1. How does optical disk store data?
2. What are the reading and writing speeds of CD?
3. What is maximum storage capacity of a DVD?

4. How is the BD different from a CD or DVD?
5. Describe the reading/writing approach of a magnetic tape.
6. What is the main usage of a magnetic tape?
7. How is the data stored in CAM?
8. What are merits and demerits of a CAM?
9. What is flash memory?

10.10 Cache Memory

Cache is a high speed memory, which is 10 to 50 times faster than RAM. It is divided into equal sized blocks and with each block a tag is associated, which is used as an identifier to search the data. The searching ability of cache based on contents makes it a kind of CAM. Generally cache is implemented to consist of two parts: cache directory and storage memory. The cache directory is usually implemented with CAM and it stores the block address tags and some control bits. The control bits are used for cache management. The memory part is implemented using static RAM. Cache is very fast but expensive also, therefore a large sized cache is not affordable for the most computers. Generally a small cache is used that keeps the recently used data and instructions. It has been observed that usually some instructions and data are referenced repeatedly over a short interval of time in a computing environment. This phenomenon is known as **locality of reference**. Therefore, due to locality of reference, the recently accessed information is kept both in cache and main memory so that access time can be reduced as cache is faster than main memory. Whenever, some data or instructions are referenced by the processor, it is searched in the cache memory. If required data/instructions are available in cache, the entire block containing the data is made available to the processor immediately otherwise data/instructions are transferred from the main memory to the cache and to the processor. If the same data/instructions are referenced by the processor again, it is available in cache, thus providing a faster access. When the required information is available in cache, it is called a **hit**. Since, the size of cache is small, the required information is not available always. In case the required information is not available in cache but in memory, a **miss** occurs. In case of

miss, the data items are transferred from other memory devices to the cache. The performance of the cache is measured in terms of hit ratio, which is defined as ratio of hits to the total memory references.

10.11 Memory Hierarchy

Many different types of memory devices have been discussed so far. Different memory devices are based on different technologies and possess different characteristics such as storage capacity, access speed, volatility, and portability, etc. A single memory is sufficient to fulfil all the requirements of a computer system. Therefore, multiple memory devices of different types are used in a computer system due to various factors including application, cost, and speed, etc. **Main memory** is the central storage that directly communicates with the processor. It is responsible to keep the data and programs required by the processor. The main memory is therefore should be fast and large enough to keep the required data. Generally, RAM is used as a main memory of the computer as it is fast and could be sufficiently large is size and affordable. Although, RAM is fast but still there is a huge gap between the speeds of a processor and RAM. So, it is desirable to have faster memory devices to match the processor speed. The cache is a very fast but expensive memory and a large cache memory is not affordable for a general purpose computer. Usually, a small cache memory is used between main memory of the computer and its processor that helps to reduce access time as discussed earlier. There are some data or small programs required in a computer that do not change. Such data and programs are stored in ROM. All these memory devices including RAM, cache, and ROM all together form the primary memory the system.

The memory devices used as main memory are volatile and there cannot store data permanently. For a permanent storage, a non-volatile memory with large size is required. The magnetic disks are a good choice for this purpose. The magnetic disks used for storage in the computer are popularly known as **hard disks**. Hard disks are the major storage devices of the computer for storing user data and programs permanently. The storage memory devices are known as secondary memory of the system. Although, the hard disks can store a large amount of data usually of the order of 1TB or more, it may get exhausted after some time. Therefore, user data that are not required very

often are achieved to another device to free up the space in secondary storage or for security reasons. The storage used for data archiving are known as tertiary memory. Magnetic tapes are primarily use as tertiary memory but optical disks can also be used for this purpose.

There are three desirable key characteristics of memory large size, high speed, and low cost. As discussed, no single memory can fulfil all the storage requirements of a computer system. Therefore, a combination of multiple memory devices is used in a system to achieve all memory design goals. This arrangement is known as memory hierarchy that combines small sized fast, expensive memory and large sized slow, inexpensive memory. The purpose of memory hierarchy is to balance the three desirable characteristics of the memory. It takes the advantage of locality of reference, which is of two types: temporal locality and spatial locality. According to temporal locality, if some information is referenced, it is supposed to be referenced again soon. All the instructions of program and related data items are usually kept at contiguous memory locations. It leads to the spatial locality. The spatial locality suggests that if some data items are referenced then data items whose address are close to the reference data will be referenced sooner or later. Memory hierarchy is a kind of abstraction that gives an illusion that a large and fast memory is available to the processor.

A structure of memory hierarchy having seven levels is shown in Figure 10.3. The registers are the smallest and fastest memory closest to the processor. The processor searches the registers for the required information. If information is not available in the registers, the next level memory i.e. the cache is verified for the information. There are multiple levels of cache, L1 cache is on chip memory also known as primary cache that is built-in with processor chip itself. There can be one or more off-chip cache memory levels i.e. L2, and L3, etc. The L1 cache is faster than other cache levels. Each level is searched one after another until information is found. If information is not available in cache at any level then main memory is searched for the data. Main memory is slower and much larger than cache. If information is found in main memory, it is transferred to the cache and then to the processor. If not, the secondary memory i.e. hard disk is access and information is transferred to the upper level memories. At the bottom of the

hierarchy is tertiary memory, which is not often required. It can be observed from Figure 10.3 that memory closer to the processor or CPU in hierarchy are more expensive, faster, and smaller. The memory size and access time increase from top to the bottom. While, cost and speed grow from bottom to the top of the hierarchy.

Check Your Progress 4

1. What is cache memory?
2. Define locality of reference.
3. Which type of RAM is used in cache?
4. Define hit ratio of cache memory.
5. What is memory hierarchy?
6. What is the purpose of memory hierarchy?
7. Arrange memory devices in increasing order of access time.
8. Arrange memory devices in increases order of storage size.
9. Define temporal and spatial locality.

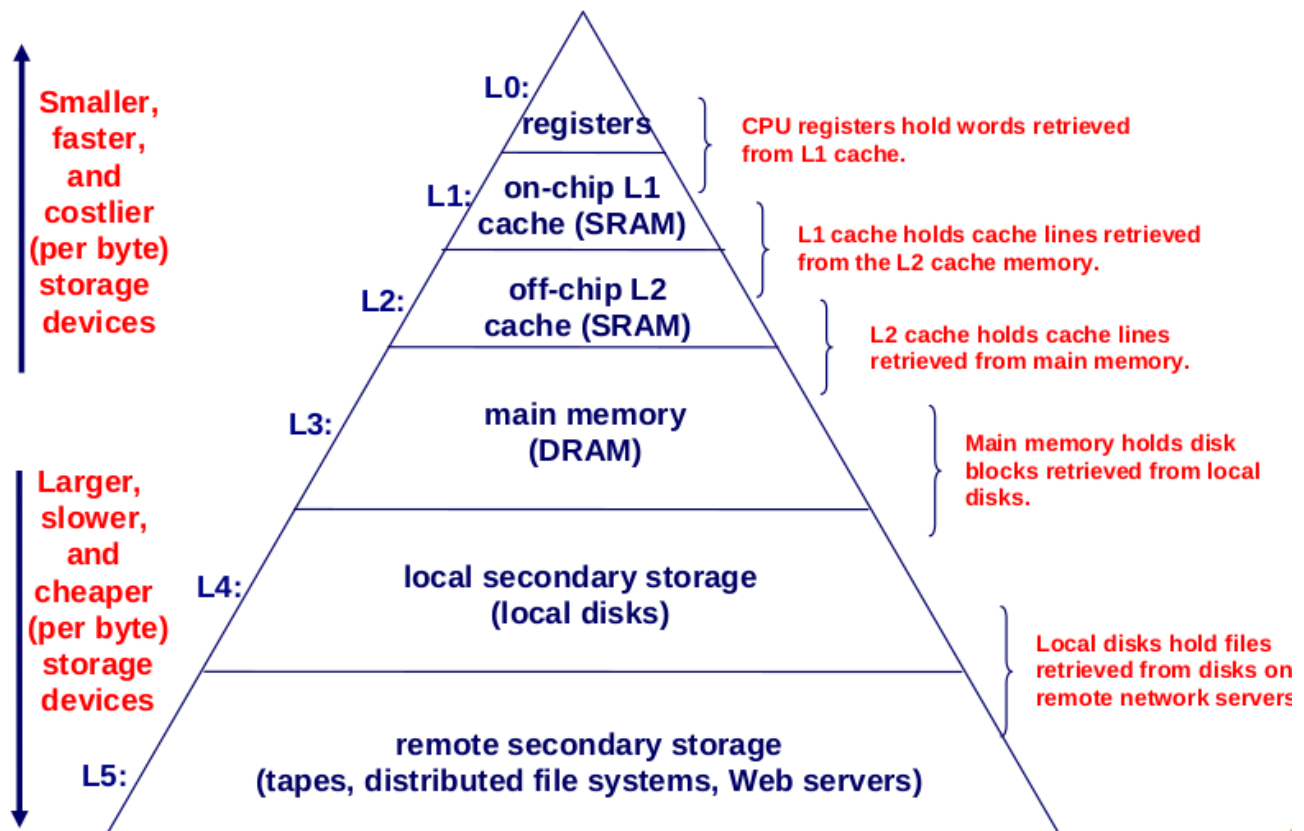


Figure 10.3: Memory hierarchy

(Source: <http://www.cs.princeton.edu/~jrex>)

10.12 Summary

Memory is an essential and important component of a computer system, which provides the space for data storage. A number of different types of memory technologies are available having different types of characteristics such as speed, access method, volatility, portability, and cost. The smallest unit in a memory is a bit. A bit can be implemented with the help of transistors, and capacitors, etc. The number of transistors/capacitors required to implement a bit decides the cost of the memory, volatility, and other characteristics. RAM is a read-write volatile memory whose access time does not depend on the location. Any location can be accessed randomly with same amount of time. There are two types of RAM namely DRAM and SRAM. The DRAM is implemented using transistors and capacitors both and it is needs to be

refreshed periodically to maintain the data. SRAM is implemented using transistors only that allows it to retain data as long as power supply is maintained without any refreshments. SRAM is faster as well as expensive than DRAM.

ROM is a non-volatile memory that maintains the data permanently once it is stored irrespective of the power supply continued or not. The access method of ROM is similar to RAM i.e. the same amount of time is required to access any location. There are different variations of ROM such as PROM, EPROM, and EEPROM. Data in a ROM can be written at the time of manufacturing only, while PROM can be programmed when user needs to do so. EPROM and EEPROM both can be written multiple times. Flash memory is a mix of EPROM and EEPROM technologies. Flash memory devices are high density, low cost, non-volatile, fast, and electrically reprogrammable. The entire memory or some blocks can be erased in a few seconds. Flash memory storage is commonly used in electronic devices.

A magnetic disk provides the permanent storage that maintains data even in the absence of the continuous power supply. The data are recorded on circular plastic platters coated with magnetic material. Each platter is divided into tracks and sectors. A sectors is the minimum addressable unit in the disk. Multiple platters are stacked on a spindle. The data can be accessed or stored with the help of a read/write arm and read/write heads. In another arrangement, the tracks are divided into blocks instead of sectors, where the number of blocks is defined by the user. Optical disks are another medium that stores the data permanently. An optical disk records the information as a change in the material characteristics using a thermally induced effect with the help of a laser assembly. Optical disks have higher density than magnetic storage and provide a reliable and removable storage medium. Unlike a magnetic storage, the optical recording and reading can be done by positioning the optical head far away from the storage medium. It helps to make it a reliable removable storage medium. Various products based on optical disks with different characteristics are available such as CD, DVD, and BD, etc.

A tape storage is a plastic strip coated with magnetic recording material, which is accessed sequentially. Unlike random access memory, the access time in a tape depends on the location of data and previously accessed location. The information is

recorded in blocks, each data block is a sequence of contiguous records. A record is a data unit that a user program deals with. The tape can start, stop, and rewind. Tapes are a good choice for archival storage where we want to store data for a long time and we do not need to access it very often.

Associative memory is a different kind of memory that is accessed by its contents. It is also known as CAM. In this type of memory, locations are accessed simultaneously in parallel based on their contents. In order to read a word from the memory, contents of the word or part of it is specified. The memory locates all the matching words and marks them to read. All the memory cells have storage capabilities as well as logic circuit for matching that makes CAM more expensive than random access memory. Cache is a high speed memory that is implemented to consist of two parts: cache directory and storage memory. The cache directory is usually implemented with CAM and it stores the block address tags and some control bits. The memory part is implemented using static RAM. Cache is very fast but expensive. Generally a small cache is used that keeps the recently used data and instructions according to locality of reference.

It is desirable to have a large and fast memory for the computer. However, the faster memory devices are more expensive and huge amount of faster memory is not affordable for a general computer system. On the other hand, some memory devices are volatile that cannot permanently store the data. The most of the faster devices are volatile and therefore, only having faster memory cannot fulfil all the desirable goals. In order to achieve all the goals related to storage, a combination of different memory devices used in a hierarchical way. At the top of the hierarchy are registers, which are built-in on the same chip with the processors. The registers are the fastest tiny memory devices that can hold a very small piece of information. The registers are followed by the multiple levels of cache. Then there is main memory of the computer in the hierarchy. Main memory is larger than cache in size. After the main memory, there are various secondary memory devices in the system. The size of the memory increases from top to the bottom, while speed and cost increase from bottom to the top of the hierarchy. Whenever, the processor needs some information it is searched in the registers. If information is not found in registers, the cache is verified for it. The memory

hierarchy gives an illusion that there is a high speed memory of the large size in the system.

Review Questions

- Q.1 What are the major general characteristics of the memory devices? Briefly describe.
- Q.2 What are major access methods for memory? Describe each of them.
- Q.3 What are the differences in SRAM and DRAM in terms of characteristics and application?
- Q.4 What are the major applications of ROM?
- Q.5 Find out the key difference among EPROM, EEPROM, and flash memory.
- Q.6 Define primary, secondary, and tertiary memory. Which memory devices can be used as primary, secondary, and tertiary memory?
- Q.7 How is a magnetic disk implemented? Explain the procedure to read/write the magnetic disk.
- Q.8 Compare magnetic disk and magnetic tape.
- Q.9 How optical disks are different than a magnetic disk?
- Q.10 Describe different types of optical disks available in market and make comparison among them.
- Q.11 What is cache memory? Why is it required?
- Q.12 What is the role of main memory in the system?
- Q.13 What is memory hierarchy? What are different memory levels in memory hierarchy?
- Q.14 What is purpose of memory hierarchy?
- Q.15 Suppose there are two levels in memory hierarchy L1 and L2. Size of L1 and L2 is 1000 words and 10,000 words respectively. Hit ratio of L1 is 0.95. If access time of L1 and L2 is 0.01ms and 0.1ms respectively, find out the average access time of the memory.
- Q.16 Consider a magnetic disk with following parameters:
- No. of cylinders: 1331
 - No. of tracks per cylinder: 11

No. of sectors per track: 40

No. of bytes per sector: 256

Suppose a file with 2048 records needs to be stored. Find out following:

- a) Total capacity of the disk.
- b) No. of cylinders required for the file if each data record takes 128 bytes.

UNIT- 11 Memory Systems

Structure

11.0 Introduction

11.1 Objectives

11.2 Evolution of Memory Technology

11.3 Advanced DRAMs

11.4 Static Random Access Memory

11.5 Multilevel Memories

11.6 Virtual Memory

11.7 Address Translation

11.8 Memory Allocation

11.9 Summary

Review Questions

Unit 11: Memory Systems

11.0 Introduction

As it was discussed in previous unit that there is a big gap between the processor cycle time and memory access time. Due to the speed gap, the processor cannot work to its full potential. A variety of memory technologies have been used over the generations. With the advancing technology, the memories have become faster. But interestingly, the processors have become even faster and the speed gap between main memory and processor has become wider. Mostly, the dynamic random access memory (DRAM) is used as the main memory in computers. Figure 11.1 shows the progress of memory and processor performance starting in 1980. It can be observed from the figure that speeds of memory and processor were compatible in 1980. Although both technologies progressed significantly thereafter but processor technology observed more progress leading to a huge gap between the speeds of two components. The primary reason behind this gap is that the memory and processor industries headed in two different directions. The emphasis of research was to increase the speed of processor while the focus of the research in case of memory was to increase the capacity. As a result the speed gap grew exponentially as shown in the figure. This performance gap is the main obstacle in the overall performance of the computer system. This disparity is expected to increase further in coming years.

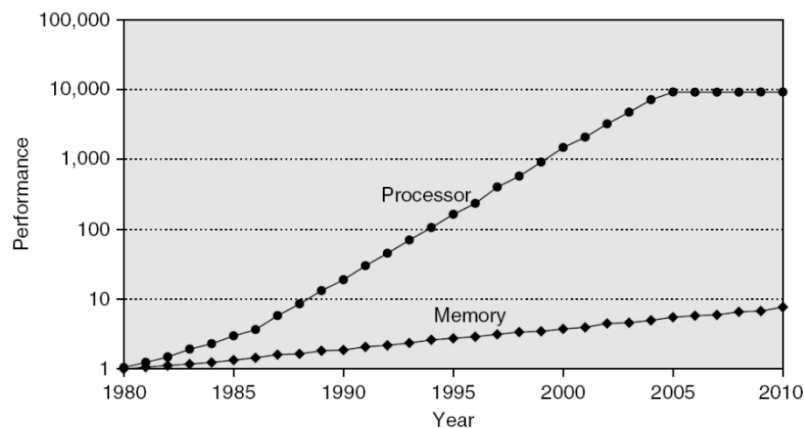


Figure 11.1: Processor-memory performance gap starting in 1980.

(Source: Mengjia Yan, MIT, 2020)

It is highly desirable to reduce the speed gap between processor and memory. The performance of the computers is limited by the memory bandwidth and latency. The bandwidth is the rate at which information can be transferred between memory and processor or other devices. The time duration from the memory request initiated by the processor to its completion is known as the latency. Ideally the zero latency and infinite bandwidth are desirable which is practically impossible. A cache memory was introduced between main memory and processor. The cache memory is formed by static random access memory (SRAM), which is faster and more expensive than DRAM. Cache memory helps to reduce the latency that improves the system performance. In recent time, the researchers are working to reduce the latency and increase the memory bandwidth to bridge the performance gap between processor and memory. In this unit, the modern memory technologies will be discussed.

11.1 Objectives

The objectives of this unit are outlined as follows.

1. To learn different types of computer memory and their characteristics.
2. To understand the memory management in a computer system.
3. To learn how memory is allocated to the user programs and process.

11.2 Evolution of Memory Technology

A number of things have been already discussed about memory and storage technologies in unit 10. This section specifically presents evolution of main memory technologies. In early days, von Neumann suggested that a memory would be required for multiple reasons including:

- 🎬 To hold the intermediate results during the complex operations
- 🎬 To store the instructions for complex operations
- 🎬 To store initial as well as boundary conditions for partial differential equations
- 🎬 To avoid repeated complex calculations

The commercial systems in late 1950s had a drum memory. In drum memory, a metal cylinder coated with magnetic material is used for recording the data. The access time of drum memories was around 2.5ms. The drum memories have a high rotational delay, therefore, drums were soon became obsolete as a main memory. Although, drum technology was in use for a longer time as a secondary memory. During 1960s, the core magnetic memories evolved, which had no moving parts and any memory word could have been accessed randomly. A core is a doughnut shaped ferromagnetic loop that can hold one bit. Core magnetic memories were around 1000 times faster than drum memories with cycle time around 6 μ s, which was improved to 1 μ s. The core magnetic memories were far superior to drum memories both in terms of latency and reliability.

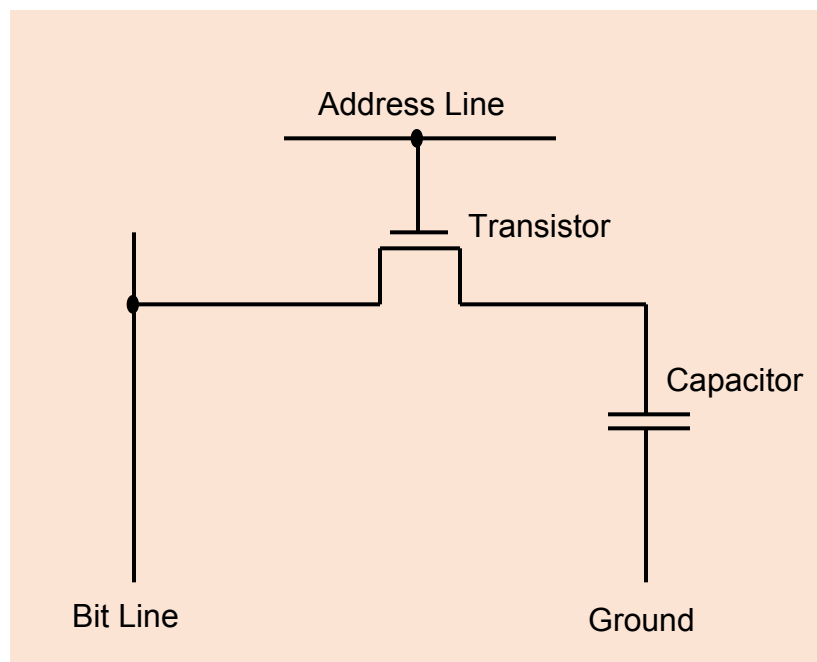


Figure 11.2: Structure of a DRAM cell

(Source: Computer Organization and Architecture, 9th Edition, William Stalings, 2012, Pearson Education)

In 1970s, the semiconductor memory was evolved and replaced core magnetic memory. Robert Dennard invented the DRAM that revolutionized the computer industry. It stores each bit of data on a small capacitor within the memory cell. The structure of

DRAM is shown in Figure 11.2. It contains one capacitor and one transistor. The capacitor may be either in a charged or discharged state. These states can represent "1" or "0" for the cell. The transistor just acts as a switch, which is open if address line is having voltage and closed if no voltage is present at address line. A voltage signal is applied to the bit line for the write operation. The bit value '1' is represented by a high voltage and a low voltage represents the bit value 0. When address line receives a write signal, the charge is transferred to the capacitor. When address line receives a read signal, the transistor turns on and the charge stored on the capacitor is released onto a bit line and to a sense amplifier. The sense amplifier determines whether the cell contains a 1 or 0 by comparing the capacitor voltage to a reference value. The read operation discharges the capacitor, therefore it must be restored to complete the operation.

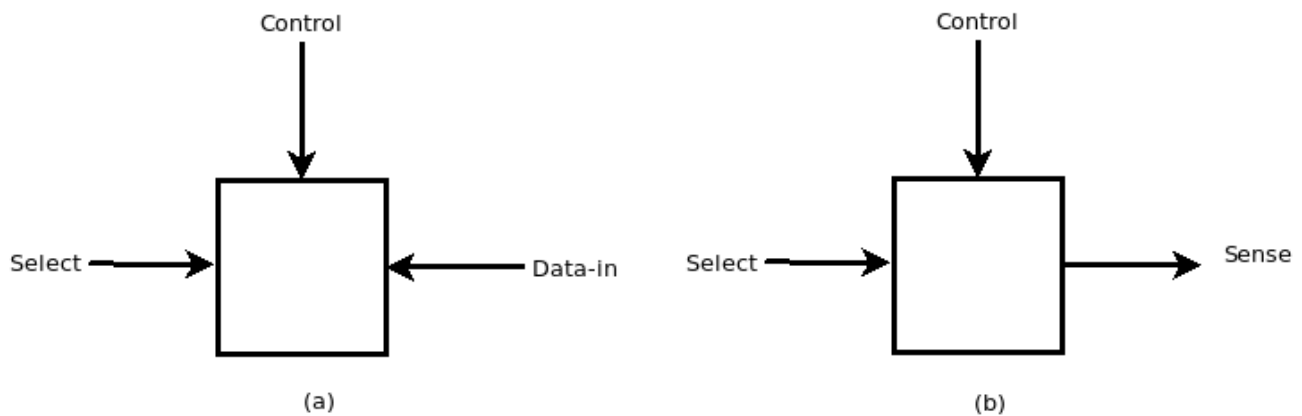


Figure 11.3: Memory cell operations (a) Write (b) Read

The operation of a memory cell is shown in Figure 11.3. The cell mostly has three functional terminals: select, control, and data-in/sense. The terminals are capable of carrying electrical signals. The select terminal is used to select a memory cell for a read or write operation. The control terminal indicates read or write operation. The third terminal provides an electrical signal to set the state as 1 or 0 in case of write operation. In case of read operation, the same terminal is used to provide the cell's state as output. Since the characteristics of the capacitor is to leak charge, it is necessary to refresh

each memory cell periodically. This refresh requirement gives rise to the term dynamic. DRAM provides better density than core magnetic memory but with some disadvantages such as volatility, more power consumption, need of power refreshment periodically, etc. But despite several disadvantages, DRAM replaced core magnetic memory as the main memory of the system due to its higher density and lesser cost. The capacity of an early single DRAM chip was 1 kb and its access time was 400 ns. Later, the capacity of a single DRAM chip was increased to 64 kb with access time 100 ns. In early 1980s, the capacity of DRAM chip was increased to 1 MB. In 1981, Bill Gates claimed that 640 KB would be good enough for any future application. Therefore, DOS does not support memory more than 640 KB.

The capacity of DRAM chip has grown up to 8 Gb today from 1kb in its first edition. The modern memories are composed of dual inline memory modules (DIMMs). The DIMM is a module in which DRAM chips are typically packaged. The main memory consists of several DIMMs that allows system memory to be as large as 16 TB. DRAM chips are large rectangular arrays of memory cells. Memory arrays are arranged in rows and columns of memory cells. These rows and columns are known as wordlines and bitlines, respectively. Each memory cell has a unique location identified by an address, which is defined by the intersection of a row and a column. To read or write a memory cell, it must be selected by its row and column coordinates. At the selected memory cell, the charge is sensed and amplified. As long as the capacitors are refreshed to keep charge decay low enough, the data is maintained in the memory cells. The access time is same irrespective of the cell location. Semiconductor memory is manufactured as a packaged chips and each chip contains an array of memory cells. Figure 11.4 shows a printed circuit board containing DRAM chips. It is designed only to store the data. The allowed operations are read, write, and refresh.



Figure 11.4: Printed circuit board containing DRAM chips

11.3 Advanced DRAMs

DRAMs are the basic building blocks of main memory for several decades. Over the years some enhancements have been done to the DRAM architecture. Based on these enhancements, there are several different products are available in the market.

11.3.1 Synchronous DRAM

The traditional DRAM architecture is asynchronous, which is not synchronized with the system clock signal. A comparatively newer form of DRAM is synchronous DRAM (SDRAM) that uses system clock signal to coordinate the memory access. The read/write operations in SDRAM are performed in synchronous to the system clock signal. The read/write instructions are issued by the processor or some other master along with address information. The SDRAM then responds after a set number of clock cycles. This tends to increase the number of instructions that the processor can perform in a given time. In the meantime, the processor can safely perform other tasks. In addition, it also uses a mode register and a kind of on-chip parallelism to provide improved performance. SDRAM technology observed a huge amount of development resulting in several generations with improved performance.

11.3.2 Double Data Rate SDRAM

The basic version of SDRAM is known as single data rate (SDR) SDRAM, which was later replaced by an enhanced version double data rate (DDR) SDRAM family. As the name suggests, the DDR SDRAM provides data transfer at twice the speed of the traditional type of SDRAM memory. The double data rate is achieved by transferring data twice per cycle, once on the rising edge of the clock pulse and once on the falling edge. DDR SDRAM is also known as DDR1. The transfer rate of DDR1 is between 266-400 MT/s. The DDR2 further improved the data transfer rate by increasing the

operational frequency. DDR2 memory uses the same internal clock speed as DDR1 but provides better performance with improved bus. The DDR2 can provide data transfer rate between 533-800 MT/s. The DDR2 was superseded by DDR3 with increased bus speed and other new features such as Automatic Self-Refresh (ASR) and Self-Refresh Temperature (SRT), which helps to control refresh rate according to temperature. DDR3 reduces power consumption by 40% to the DDR2. The transfer rate offered by DDR3 is 800-1600 MT/s. DDR3 is superseded by DDR4 having significant architectural changes. DDR4 operates at lower voltage and provide higher data rate of 2133-3200 MT/s. DDR4 can process 4 data within a clock cycle. DDR4 also provides some new features such as Data Bus Inversion (DBI), Cyclic Redundancy Check (CRC), and CA parity to offer better integrity and stability in data transmission. The next generation of DDR SDRAM is DDR5 that provides data transfer rate of 3200-6400 MT/s at reduced power consumption along with some new features. The different DDR generations are not backward or forward compatible and therefore, the same motherboard cannot be used with different generation DDR SDRAMs.

11.3.3 Rambus DRAM

Rambus DRAM popularly known as RDRAM is a kind of synchronous dynamic random access memory that uses a special bus to deliver address and control information with the help of an asynchronous block-oriented protocol. The bus defines impedances, clocking, and signals very precisely to achieve data rate upto 1.6 Gbps. The RDRAM chips were used in Intel Pentium and Itanium processors. RDRAM was an expensive alternative of SDRAM. Therefore, subsequently the most of the manufacturers did not preferred RDRAM.

Check Your Progress 1

1. Highlight the major differences between magnetic core memory and semiconductor memory.
2. How is a bit stored in DRAM?
3. What is the role of transistor in the DRAM cell?

4. Why does the DRAM cell need to refresh periodically?
5. How does DDR SDRAM achieve double data rate?
6. How is the synchronous DRAM different than traditional DRAM?
7. Write the transfer rates of different generations of DDR SDRAMs?
8. What is RDRAM?

11.4 Static Random Access Memory

The DRAM cell is an analog device although it is used to store logic 0 and 1 values. The static random access memory (SRAM) on the other hand is a digital device that uses flip-flops to store binary values. The data remain maintained in the memory as long as power supply is available and unlike DRAM, the memory cell is not required to refresh. The structure of a SRAM cell is shown in Figure 11.5, which uses an arrangement of six transistors to provide two stable operating points. Depending on the stable state of the circuit the data is interpreted either as 0 or 1. Therefore, SRAM cell is more expensive and more area on the chip than a DRAM cell. But SRAM does not require any voltage refresh circuitry. SRAM are faster and expensive, therefore they are preferred for implementing cache memories. While DRAM is preferred for implementing larger main memory.

11.5 Multilevel Memories

It is highly desirable that a computer memory should be large enough to contain large programs and that could work at a speed that is comparable to the processor. However, there is no such memory technology available that could fulfil all these requirements related to size and speed. Therefore, multiple memories having different characteristics are used in a computer system to meet all the requirements. Usually a large size DRAM works as the main memory of the system that is capable of containing large programs and data. But DRAM is not a good match to speed of the processor. Therefore, a cache memory is used between the main memory and processor to hide slow speed of the memory from the processor. This two-level memory arrangement of a fast, small memory and a slow, large memory builds a composite memory system that behaves

like a large fast memory overall. The two-level memory arrangement is further extended to memory hierarchy that includes several levels including main memory, cache memory, and secondary memory, etc.

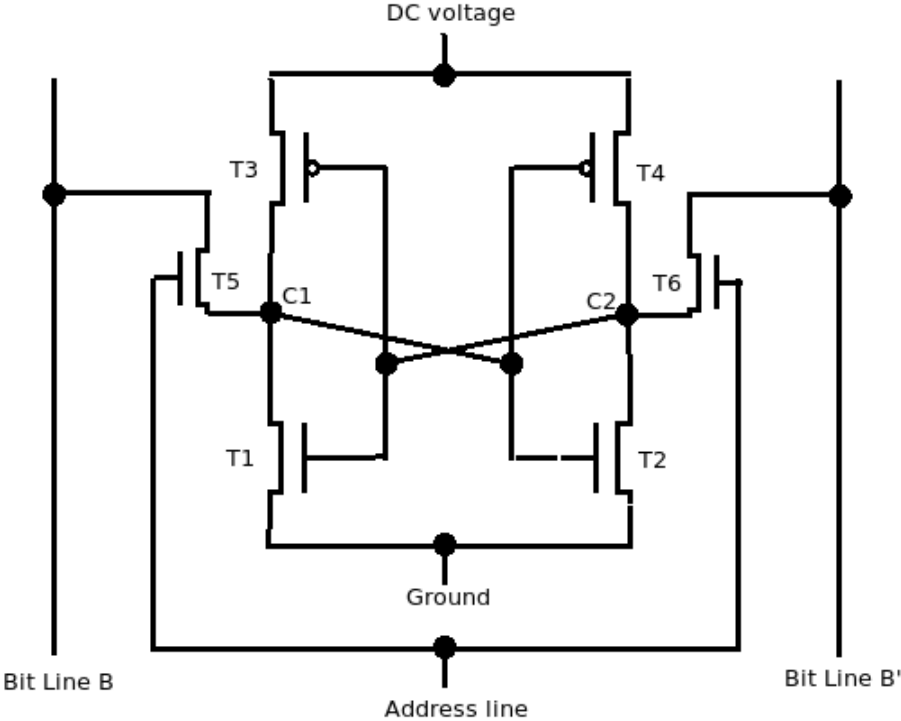


Figure 11.5: Structure of a SRAM cell

As mentioned earlier, a cache memory is inserted between main memory and processor to reduce the access time. The cache memory is manufactured with the help of SRAM. It consists of SRAM array of memory cells. The speed (or access time) of a cache is a function of its size, organization technology. As the size of SRAM array increases, the length of access wires also increases accordingly. The larger length of access wires leads to higher access time of cache. Therefore, if we wish to keep the access time under certain threshold, the size of the cache is also limited accordingly. Therefore, we use multiple cache memories at different levels. The size, organization, and technology of the cache at different levels may be different. Although, the arrangement of multiple cache memories also has its own overhead.

There are several design considerations of multilevel memories. The cache at level 2 should be same or larger than level 1 cache. Otherwise, a miss at level 1 would also cause a miss at level 2. The size of the memory increases from lower level to higher level. Usually level 1 cache (L1 cache) is less than 64KB, level 2 cache (L2 cache) is less than 512KB, and level 3 (L3 cache) uses multiple arrays of 256KB to create large size. A multilevel system is called inclusive if all the contents of lower level cache are also available at the next higher level. It is possible only if the higher level cache is significantly higher than the lower level caches. The performance of a memory system can be evaluated in terms of miss ratio and access time. Several miss rates can be defined in a multilevel memory system as follows:

- A solo miss rate for a memory at any level is defined as the miss rate if it would have been the only memory in the system.
- A local miss rate is the number of misses experienced by the memory divided by the number of references to it.
- A global miss rate is the number of misses experienced by the highest level memory divided by the number of references made by the processor.

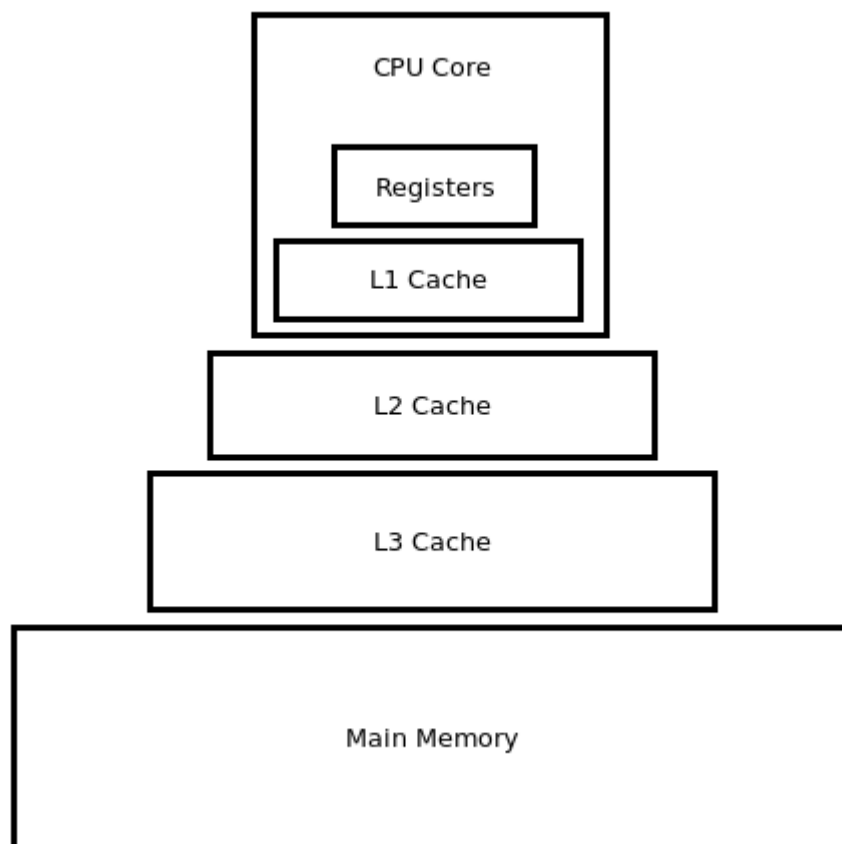


Figure 11.6: Multilevel cache organization

If the L2 cache is significantly larger than the L1 cache, its miss rate is compatible with its solo miss rate. A larger L2 cache can retain all the data available in L1 cache. But a primary concern with this kind of inclusive cache organization is that the copies of the same data at different levels should be exactly same. Otherwise, it may lead to wrong results. Figure 11.6 shows a multilevel cache organization up to levels. L1 cache is usually on chip cache memory. L2 cache is off-chip cache, which is larger and slower than L1. The L3 cache is built on the motherboard and it is slower and much larger than L2 cache. The L1 cache needs to be faster to provide the fastest possible access time. Therefore, it is manufactured with larger transistors and wider metal tracks. The L2 and L3 caches should be larger in capacity but they can afford to be little slower. Therefore, L2 and L3 are built with smaller transistors.

11.6 Virtual Memory

A virtual memory system intends to optimize the use of the main memory using the lower speed secondary memory. It is a technique that extends the physical memory beyond its physical size. The virtual memory allows to accommodate the programs larger than the available physical memory with the help of secondary memory. It is based on the fact that usually not all the parts of a given program are active at a time. Therefore, only the active parts of a program are brought to the main memory keeping the remaining non-active parts in the secondary storage. During the execution, if the segment of a program requested by the processor is not available in the primary memory, it is brought from the disk to the primary memory. If adequate space is not available in the primary memory to accommodate the requested segment, some existing segments/data are replaced. It employs the same principles as those used in the case of cache memory. In other words, the most relevant segments are kept in the high-speed main memory and transferring less relevant or inactive segments in the secondary memory.

The virtual memory is divided into pages and corresponding unit on the main memory is known as page-frame. The size of pages and page-frames is same. The data transfer between disk and main memory is performed in the form of pages. In fact, a page is a group of memory words. The size of a page typically ranges from 2K to 16 bytes. When a particular word is requested by a processor, the corresponding page is searched. If it is not available in the memory, a page fault occurs. In this case, the entire page containing the requested word is transferred from secondary memory to the main memory. All this arrangement is completely transparent to the application developer. The addresses issued by a processor are therefore not actual physical addresses. Instead, such addresses are called virtual addresses. The virtual addresses issued by the processor are mapped to some physical addresses in order to access the data. The mapping or translation of virtual to physical address is performed by the memory management unit.

Check your progress 2

1. Differentiate DRAM and SRAM.
2. Which of DRAM and SRAM is a faster memory?
3. Why multilevel memories are used in a computer system?
4. Which memory technology is used for the main memory of the computer system?
5. Which memory technology is to manufacture cache memory?
6. How is the performance of the memory system evaluated?
7. Differentiate local cache miss rate and global cache miss rate?
8. Cache at which level is usually the on chip memory?
9. Which cache is built on motherboard?
10. Which cache memories are built with small registers?
11. What is virtual memory?
12. Define page and page frames?
13. What is the typical size of a page?
14. What is page fault?
15. What is virtual address?

11.7 Address Translation

As discussed in previous section, the virtual address issued by the processor is mapped to a particular physical address to perform actual data transfer. There are various address mapping techniques with their own merits and demerits. All these mapping schemes make use of a translation table known as page table, which is stored in main memory. The page table contains the pages and corresponding memory location along with some other information. The additional information is maintained with the help of some special bits such as valid bit and dirty bit. The valid bit associated with a page is set if the page is available in the memory. The dirty bit is set if the page is modified after loading into memory.

11.7.1 Direct Mapping

In direct mapping scheme, the virtual address is divided into two fields: the virtual page number and the offset. The page number field directly identifies an entry in the page table. If the valid bit associated with the page table entry identified by the virtual page number is valid, the contents of the page table are related to the physical address. The page table contents and page offset are concatenated to form the actual physical address. On the other hand, if the valid bit of the page table entry associated with the virtual page number is not set, a page fault occurs. In case of a page fault, the required data are transferred from the disk to the main memory. The data transfer may also require the page replacement to make space in the memory. For that purpose, the replacement policies are needed. The simplicity of direct mapping is its main advantage. But the size of page table is large in case of direct mapping.

11.7.2 Associative Mapping

In associative mapping also, the virtual address is divided into two fields, a page number and an offset similar to direct mapping technique. The page table entries are

also divided into two parts: the virtual page number and the physical page number. When a virtual address is generated by the processor, the page table is searched for a match of the virtual page number field. If a match is found, the corresponding physical page number is extracted from the page table. The physical page number is concatenated with the offset field to generate the physical address. In case a match is not found for the virtual page number, it leads to a page fault. If a page fault occurs, the memory management unit loads the corresponding page from the disk into the main memory. The main advantage of the associative mapping is that it needs a shorter page table. On the other hand, searching overhead is its main disadvantage.

11.7.3 Set Associative Mapping

The set associative mapping is a compromise between direct mapping and associative mapping. It is a hybrid technique that combines the features of direct mapping and associative mapping. In set associative mapping, the virtual address is divided into three fields: a tag, an index, and an offset. The page table is divided into sets. Each set consists of a number of fields. Like direct mapping, the tag field of a virtual address is used to directly find the page table set to search the physical address. After identifying the set, a search is conducted to match the tag field with all the entries in the set. If a match is found, the corresponding physical page address is extracted and concatenated with the offset field to generate the physical address. In case of a no match, a page fault occurs that is handled as discussed in previous two mapping techniques. The set associative mapping reduces the searching overhead to a specific set. While matching operation is as simple as in direct mapping. Thus it establishes a compromise between direct mapping and associative mapping.

In modern computer system, a portion of the page table is kept in translational lookaside buffer (TLB), which is typically designed as set associative memory. When a virtual address is generated by the processor, the TLB is searched first to find the corresponding physical address. With the help of TLB memory access overhead can be reduced. If a match is not found in TLB then the page table is searched as usual.

11.8 Memory Allocation

Memory allocation is the process of reserving computer memory for the execution of a computer program or process. It could be a partial or complete allocation of the memory needed by a program depending on the availability. The memory management unit takes the responsibility of memory allocation. Memory allocation is managed through operating system but it is primarily a computer hardware operation. When a program is executed, the required memory is allocated to the program. Once the program execution is over or it becomes idle, the allocated memory is released so that the released memory could be allocated to other programs or services if needed. This process is known as memory de-allocation. There are two types of memory allocation approaches: static memory allocation and dynamic memory allocation. The static memory allocation is done at compile time, while dynamic memory allocation is performed at program runtime.

Operating system handles the primary memory and moves the programs/processes between disk and main memory. It keeps track of allotted and available/free the memory locations. It is the job of operating system to decide when and how much memory to allocate to the processes. The main memory is usually divided into two parts: low memory and high memory. The operating system itself resides in low memory, while the user processes are kept in high memory partition. The physical allocation of memory can be performed in two different ways: single partition allocation and multiple partition allocation. In single partition allocation, all the user processes are held in a single large partition. Relocation register is used to prevent the user programs to modify each other and operating system related code and data. Another register called limit register is used to specify the range of addresses belonging to a user process. On the other hand, in multiple partition allocation scheme, the memory is divided into multiple fixed-sized parts. A partition is allowed to contain only one process at a time. A user process is picked from the queue and a free partition is allocated to it. When a process terminates, its partition is released that becomes available for other processes in the queue.

As memory is allocated and released during the process execution and termination, it creates some free space at scattered locations in the memory. Sometimes memory could not be allocated to a process because no particular available

piece is sufficient to hold the process. This problem is known as fragmentation. There are two types of fragmentation: internal fragmentation and external fragmentation. When memory block or partition assigned to a process is larger than its requirement, some space remain unused but it cannot allocated to other process until it is released. This type of fragmentation is known as internal fragmentation. The internal fragmentation can be reduced by careful memory allocation such that a memory block good enough to hold a process should be assigned to that process. If the required memory space cannot be allocated to a process as sufficient space is not available at contiguous locations even though collectively the free space is larger than the required memory. This type of fragmentation is known as external fragmentation. The external fragmentation can be reduced by compaction. The memory contents are shuffled to bring the unused space together to form a larger free block.

Check your progress 3

1. Why virtual address is translated to a physical address?
2. What is the role valid bit?
3. Explain the utility of dirty bit?
4. What is page table?
5. What is the advantage and disadvantage of direct mapping?
6. Write the advantages and disadvantages of associative mapping?
7. What is in translational lookaside buffer?
8. Differentiate static and dynamic memory allocation.
9. Explain multi-partition memory allocation.
10. Differentiate single partition and multi-partition allocation.
11. What is memory fragmentation?
12. How can you reduce the external memory fragmentation?

11.9 Summary

There is a big gap between the processor cycle time and memory access time. Due to the speed gap, the processor cannot work to its full potential. A variety of memory technologies have been used over the generations. It is highly desirable to reduce the speed gap between processor and memory. The performance of the computers is limited by the memory bandwidth and latency. The bandwidth is the rate at which information can be transferred between memory and processor or other devices. In early computers magnetic material was used to store the data. In 1970s, the semiconductor memory was evolved and replaced core magnetic memory. Robert Dennard invented the DRAM that revolutionized the computer industry. DRAM provides better density than core magnetic memory but with some disadvantages such as volatility, more power consumption, need of power refreshment periodically, etc. But despite several disadvantages, DRAM replaced core magnetic memory as the main memory of the system due to its higher density and lesser cost. The traditional DRAM architecture is asynchronous, which is not synchronized with the system clock signal. A comparatively newer form of DRAM is synchronous DRAM (SDRAM) that uses system clock signal to coordinate the memory access. The basic version of SDRAM is known as single data rate (SDR) SDRAM, which was later replaced by an enhanced version double data rate (DDR) SDRAM family. Later several generations of SDRAM evolved over the years.

It is highly desirable that a computer memory should be large enough to contain large programs and that could work at a speed that is comparable to the processor. However, there is no such memory technology available that could fulfil all these requirements related to size and speed. Therefore, multiple memories having different characteristics are used in a computer system to meet all the requirements. Usually a large size DRAM works as the main memory of the system. But DRAM is not a good match to speed of the processor. Therefore, a cache memory is used between the main memory and processor to hide slow speed of the memory from the processor. Therefore, if we wish to keep the access time under certain threshold, the size of the cache is also limited accordingly. Therefore, we use multiple cache memories at different levels. The size, organization, and technology of the cache at different levels may be different. A virtual memory system intends to optimize the use of the main

memory using the lower speed secondary memory. It is a technique that extends the physical memory beyond its physical size. It is a technique that extends the physical memory beyond its physical size. The virtual memory allows to accommodate the programs larger than the available physical memory with the help of secondary memory. It is based on the fact that usually not all the parts of a given program are active at a time. Therefore, only the active parts of a program are brought to the main memory keeping the remaining non-active parts in the secondary storage. The virtual memory is divided into pages and corresponding unit on the main memory is known as page-frame. The size of pages and page-frames is same. The data transfer between disk and main memory is performed in the form of pages. The addresses issued by a processor are therefore not actual physical addresses. Instead, such addresses are called virtual addresses. The virtual addresses issued by the processor are mapped to some physical addresses in order to access the data. The mapping or translation of virtual to physical address is performed by the memory management unit. There are various address mapping techniques with their own merits and demerits. The major mapping techniques include direct mapping, associative mapping, and set-associative mapping. The direct mapping is simple but uses a large page table. On the other hand associative mapping although uses a smaller page table involves searching overhead. The set-associative mapping is a compromise between direct mapping and associative mapping. Memory allocation is the process of reserving computer memory for the execution of a computer program or process. It could be a partial or complete allocation of the memory needed by a program depending on the availability. The memory management unit takes the responsibility of memory allocation. As memory is allocated and released during the process execution and termination, it creates some free space at scattered locations in the memory. Sometimes memory could not be allocated to a process because no particular available piece is sufficient to hold the process. This problem is known as fragmentation. There are two types of fragmentation: internal fragmentation and external fragmentation. It is desirable to reduce any kind of fragmentation.

Review Questions

- Q.1 Write a short note on the evolution of the computer memory technology.
- Q.2 Briefly explain different generations of DRAMs with their major characteristics.
- Q.3 Why do you need multilevel memories? What is the criteria to place different memories at different levels?
- Q.4 How is the capacity of the physical memory extended with the help of virtual memory? Explain.
- Q.5 Explain different address mapping techniques with their merits and demerits.
- Q.6 What approaches are used to allocate memory to the programs or processes in a computer system?
- Q.7 What is fragmentation? How many types of fragmentation are there? How can you reduce the fragmentation?

UNIT- 12 Cache Memory

Structure

- 12.0 Introduction
 - 12.1 Objectives
 - 12.2 Cache Memory System
 - 12.3 Cache Performance
 - 12.4 Program Locality
 - 12.5 Mapping Techniques
 - 12.6 Replacement Algorithms
 - 12.7 Cache Updating
 - 12.8 Summary
- Review Questions

Unit 12: Cache Memory

12.0 Introduction

In comparison to the speed of processors, the main memory is extremely slow. The processor cannot spend much of its time waiting for instructions and data in main memory if it is to perform well. As a result, it's essential to design a strategy that cuts down on the time it takes to get the needed information. As technological and packaging constraints limit the speed of the main memory unit, the solution must be found in a different architectural configuration. A good approach is to employ a fast cache memory, which makes the main memory appear faster to the processor than it actually is.

Cache memories are small, high-speed buffer memories that are used in computer systems to temporarily store chunks of main memory that are (believed to be) in use. Information in cache memory can be accessible in a fraction of the time it takes to access data in main memory. As a result, a central processing unit (CPU) with cache memory requires considerably less time to fetch and/or store instructions and operands. Cache memory are found in almost all modern computer systems.

12.1 Objectives

After the completion of this unit, the students would:

1. Have a better understanding of cache memory used in computer system.
2. Know the concept of locality of reference.
3. Understand the different mapping techniques used in cache memory.
4. Know various replacement algorithms and updating policies used in designing cache memory system.

12.2 Cache Memory System

As we all know, a memory unit is an important part of any digital computer since it stores programmes and data. The Dynamic RAM (DRAM) is used as main memory. The processor then reads the program's code and data from main memory and executes it. The DRAMs that make up the primary memory are slower. As a result, wait states must be inserted into memory read/write cycles. This slows down the execution process. High-speed memory, such as Static RAM (SRAM) must be employed to speed up the process. The average memory access time can be decreased by placing the active portions of the programme and data in fast small memory, reducing the total execution time of the programme. Such a fast small memory is referred to as a cache memory.

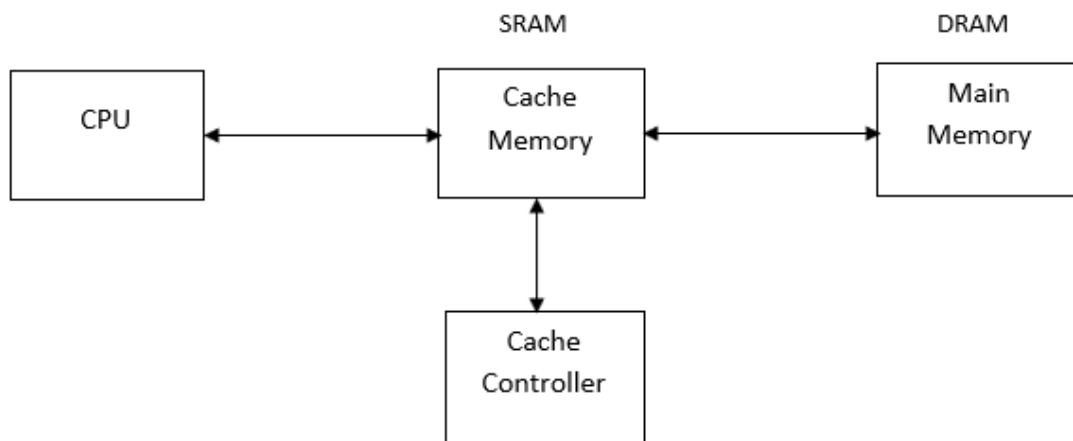


Figure 12.1: Cache memory system

A cache memory system consists of a large amount of slow low-cost memory, DRAM, and a small rapid memory, SRAM. This system has been set up to simulate a large amount of rapid memory. Figure 12.1 shows a cache memory system. When compared to main memory, the access time of cache memory is extremely fast. When the CPU sends a read request, the contents of a block of memory words comprising the given location are sent one word at a time into the cache. The desired items are then read directly from the cache whenever the programme refers to any of the locations in this block. The cache memory can usually store a significant number of blocks at a time, although this number is little when compared to the total amount of blocks in the main

memory. If the CPU discovers that the requested data is not in cache, the processor retrieves it from main memory (DRAM).

12.3 Cache Performance

When the processor needs to read or write data from main memory, it first looks in the cache for a matching item.

■ A cache hit said to occurs when the CPU discovers that the requested memory location is in the cache, and data is read from the cache.

■ A cache miss occurs when the CPU cannot locate the memory location in the cache. When a cache miss occurs, the cache creates a new entry and copies data from main memory, after which the request is fulfilled using the cache's contents.

Cache memory performance is commonly quantified in terms of a metric known as Hit ratio. Cache hit ratio is calculated by comparing the number of cache hits to the total number of content requests received. It can be determined as follows.

$$\begin{aligned}\text{Cache Hit Ratio} &= \text{no. of hits} / \text{total accesses} \\ &= \text{Cache hit} / (\text{Cache hit} + \text{Cache miss})\end{aligned}$$

Cache miss ratio is the inverse of this, in which the number of cache misses is calculated and compared to the total number of requests as follows.

$$\begin{aligned}\text{Cache Miss Ratio} &= \text{no. of miss} / \text{total accesses} \\ &= \text{Cache miss} / (\text{Cache hit} + \text{Cache miss}) \\ &= 1 - \text{Hit Ratio}\end{aligned}$$

Average memory access time (AMAT) is a common metric to analyse computer memory system performance. It can be determined as follows.

$$\text{Average Memory Access Time} = h_r \times (t_c) + (1 - h_r) \times (t_c + t_m)$$

Where, h_r = hit ratio

t_c = cache memory access time

t_m = main memory access time

Check your progress 1

6. What is cache memory?
7. What is the need of cache memory in modern computer systems?
8. Define cache hit?
9. What is a cache miss?
10. Explain cache hit ratio?
11. What is cache miss ratio?
12. Find the cache hit ratio, if total number of requests generated by the CPU is 90 out of which 85 requests were fulfilled by cache memory.
13. Find the average memory access time, if a hit takes 0.5ns and happens 90% of the time, and a miss takes 10ns and happens 10% of the time.

12.4 Program Locality

Prediction of memory location for the future access is critical in every memory system. This is achievable because most computer systems access data from the same area. Program locality is the prediction of the next memory location from the current memory address. Cache controller is enabled by programme locality.

12.4.1 Locality of Reference

A programme can have a simple loop, nested loops, or a few routines that call each other repeatedly. The precise structure of instruction sequencing is unimportant; what matters is that many instructions in specific parts of the programme are executed repeatedly over a period of time. The rest of the programme is accessed on a fairly infrequent basis. The term for this is "locality of reference." It appears in two forms: temporal and spatial. A recently executed command is likely to be executed again very soon, according to the temporal. Because of the spatial relationship, instructions that are placed close to recently performed instructions are more likely to be executed shortly.

The temporal element of the locality reference recommends that whenever instruction and data information is needed for the first time, it should be brought into cache and stored there until it is needed again. The spatial element recommends that instead of moving only one item from main memory to the cache, several items at

nearby addresses should be brought as well. A set of continuous addresses of size is referred to as a block.

12.4.2 Elements of Cache Design

The major issues in cache design include cache size, mapping function, replacement algorithm, write policy, block size, and cache number, etc. All these are briefly explained here. The cache should be small enough that the overall average cost per bit is comparable to that of main memory alone, while also being large enough that the overall average access time is near to that of the cache alone.

At any given time, the cache memory can store a fair number of blocks, although this amount is minimal in comparison to the total number of blocks in the main memory. To link the main memory blocks and cache blocks, we must utilise mapping functions. Cache mapping is a mechanism for transferring the contents of main memory to cache memory. In the situation of a cache miss, cache mapping specifies how a block from main memory is mapped to the cache memory. There are three commonly used mapping functions: direct mapping, associative mapping, set associative mapping. All these techniques are introduced in Unit 11 and explained in more detail later in sections.

When the cache is full and a memory word is addressed that isn't in the cache. To make room for the new block containing the referred word, the cache controller must select which block should be eliminated. The replacement algorithm is a collection of rules for making this decision. There are four most common replacement algorithms: Least-Recently Used (LRU), First-In First-Out (FIFO), Least-Frequently-Used (LFU), and Random.

Cache updating policy is another name for it. Two copies of the same data can exist in a cache system at the same time, one in cache and one in main memory. Two different sets of data are associated with the same address if one copy is updated while the other is not. To prevent this from happening, the cache system provides following updating policies: Write through system, Buffered write through system, Write-back system.

Check your progress 2

1. Explain Locality of reference?
2. What are different elements of cache design?
3. What is the use of mapping function?
4. Write the names of different cache mapping functions?
5. Why replacement policy is required in cache design?
6. List some of the replacement algorithms used in cache design?
7. What is write policy in cache design?
8. Name the various write policies used in cache design?

12.5 Mapping Techniques

The speed with which cache memory may be accessed is one of its primary qualities. As a result, while looking for terms in the cache, very little or no time should be lost. A mapping procedure refers to the process of moving data from main memory to cache memory. There are three types of mapping techniques used in cache memory organization:

1. Direct mapping technique
2. Associative mapping technique
3. Set-associative mapping technique

To discuss various mapping techniques for specifying where memory blocks are placed in the cache, we use a specific small example. Consider a cache consisting of 256 blocks of 16 words each, for a total of 4096 (4K) words, and assume that the main memory is addressable by a 17-bit address. The main memory has 128K words, which we will view as 8K blocks of 16 words each. For simplicity, we will assume that consecutive addresses refer to consecutive words. To help in the discussion of these three mapping procedures we will use a specific example of a memory organization as shown in Figure 12.2.

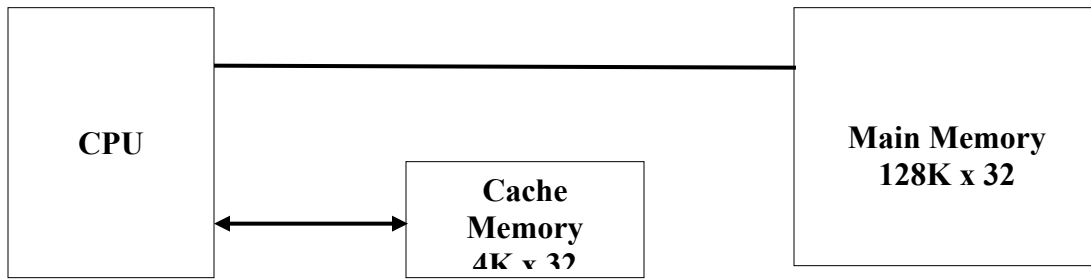


Figure 12.2: Cache Memory Example

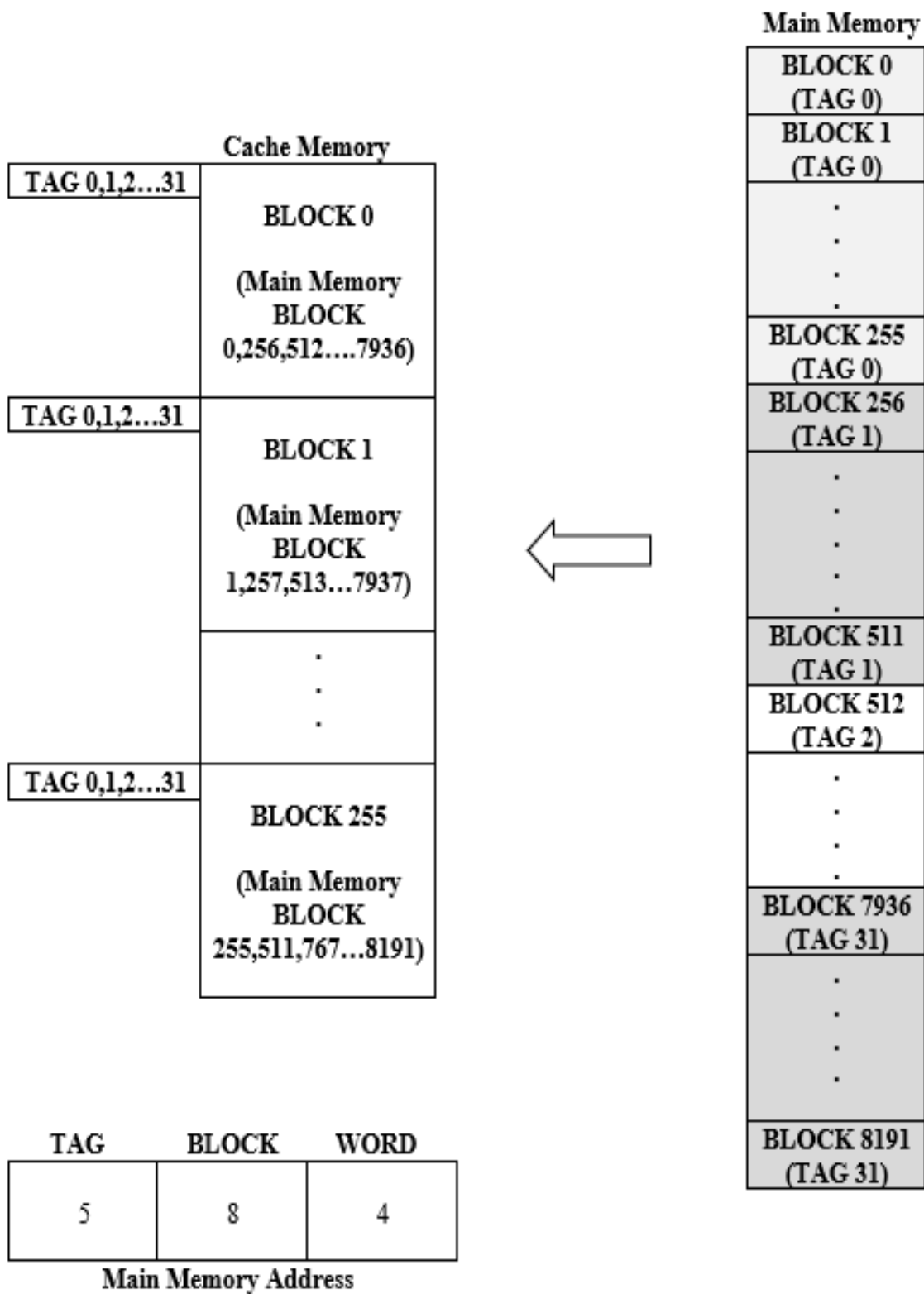


Figure 12.3: Direct Mapped Cache

The main memory can store 128K words of 32 bit each. The cache is capable of storing 4K words at any given time. For every word stored in cache, there is a duplicate copy in main memory. The CPU communicates with both memories. It first sends a 17-bit address to cache. If there is a hit, the CPU accepts the data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

12.5.1 Direct mapping

The direct-mapping technique is the easiest way to determine cache locations in which to store memory blocks. In this technique, block i of the main memory maps onto block i modulo 256 of the cache, as shown in Figure 12.3. Thus, whenever the main memory blocks 0, 256, 512, etc. is loaded in the cache, it is stored in cache block 0. Blocks 1, 257, 513, etc. are stored in cache block 1, and so on. Because more than one memory block is mapped to a single cache block position, even if the cache isn't full, there may be contention for that location. For example, instructions of a program may start in block 2 and continue in block 258, potentially after a branch. As this program is executed, both of these blocks must be transferred to the block-2 position in the cache. Contention is resolved by allowing the new block to overwrite the currently resident block. The placement algorithm is simple in this situation.

The memory address is used to identify where a block should go in the cache. The memory address can be divided into three fields, as shown in Figure 12.3. The lower-order 4 bits of memory address selects one of 16 words in a block. When a new block enters the cache, the next 8-bits of memory address determines the cache block number in which this block must be stored. The high-order 5 bits of the memory address of the block are stored in 5 *tag* bits associated with its location in the cache. They determine which of the 32 blocks assigned to this cache slot is currently stored in the cache.

As execution progresses, the 8-bit cache block field of each address generated by the processor points to a specific block location in the cache. The high-order 5 bits of the address are compared with the tag bits associated with that cache location. *If they match, then the desired word is in that block of the cache. If no matches are found, then*

the block containing the required word must first be read from the main memory and loaded into the cache. The direct-mapping technique is simple to use, although it is limited in its flexibility.

12.5.2 Associative mapping technique

Associative memory is the fastest and most versatile cache structure method. This organization is illustrated in Figure 12.4. It allows any block from main memory to be placed anywhere in the cache. A specific block is uniquely identifiable by its memory block number once it is placed in the cache, this is known as tag. As there is no fixed block, the memory address has only two fields: Word and tag. This technique is also referred to as fully associative cache. In this case, 13 tag bits are required to identify a memory block when it is resident in the cache. To see if the desired block is present, the tag bits of an address received from the processor are compared to the tag bits of each block of the cache. It gives complete freedom over where the memory block is to store in the cache. As a result, the cache's space can be better utilised. If the cache is full, a new block must eject (replace) an existing block. In this situation, we'll need an algorithm to choose the replacement block. Many replacement algorithms are possible, that we will discuss in Section 12.6. Because it is necessary to scan all 256 tag patterns to determine whether a given block is in the cache, the cost of an associative cache is higher than the cost of a direct-mapped cache. This type of search is known as an associative search. The tags must be searched in parallel for speed reasons.

12.5.3 Set Associated Mapped Cache

The set associative mapping is a hybrid of direct and associative mapping techniques. It is made up of numerous groups of direct mapped blocks that work in parallel. A block of data from any page in main memory can be placed in any direct-mapped cache block. As a result, the direct mapping technique's contention problem is reduced by providing a limited number of block placement options. The number of required address comparisons is determined by the number of direct mapped cache in the cache system. These comparisons are always less than the fully associative mapping's required comparisons.

The set associative mapped cache with two blocks per set is shown in Figure 12.5. Each page in the main memory is structured so that the size of each page is the same as that of a single directly mapped cache. Because each block from main memory has two options for placement, it's called a two-way set associative cache.

Block 0,128, 256... 8064 of main memory can be placed in any of the two blocks (block 0 or block 1) of set 0 and occupy one of two block locations within this set using this technique. With 128 sets, the address's 7-bit set field indicates which set of the cache may store the specified block. The address's tag field must then be compared associatively to the tags of the set's two blocks to see if the requested block is present. It's easy to set up this two-way associative search. In order to see if a match exists, the tag field of the CPU address is compared to both tags in the cache. An associative search of the tags in the set, similar to an associative memory search, is used to perform the comparison logic. Because the set size increases, more words with the same index but different tags can be cached, the hit ratio improves. However, as the set size grows, the number of bits in cache words grows, necessitating more complicated comparison logic. When a miss occurs in a set associative cache and the set is full, any replacement mechanism must be used to replace one of the tag- data items with new data.

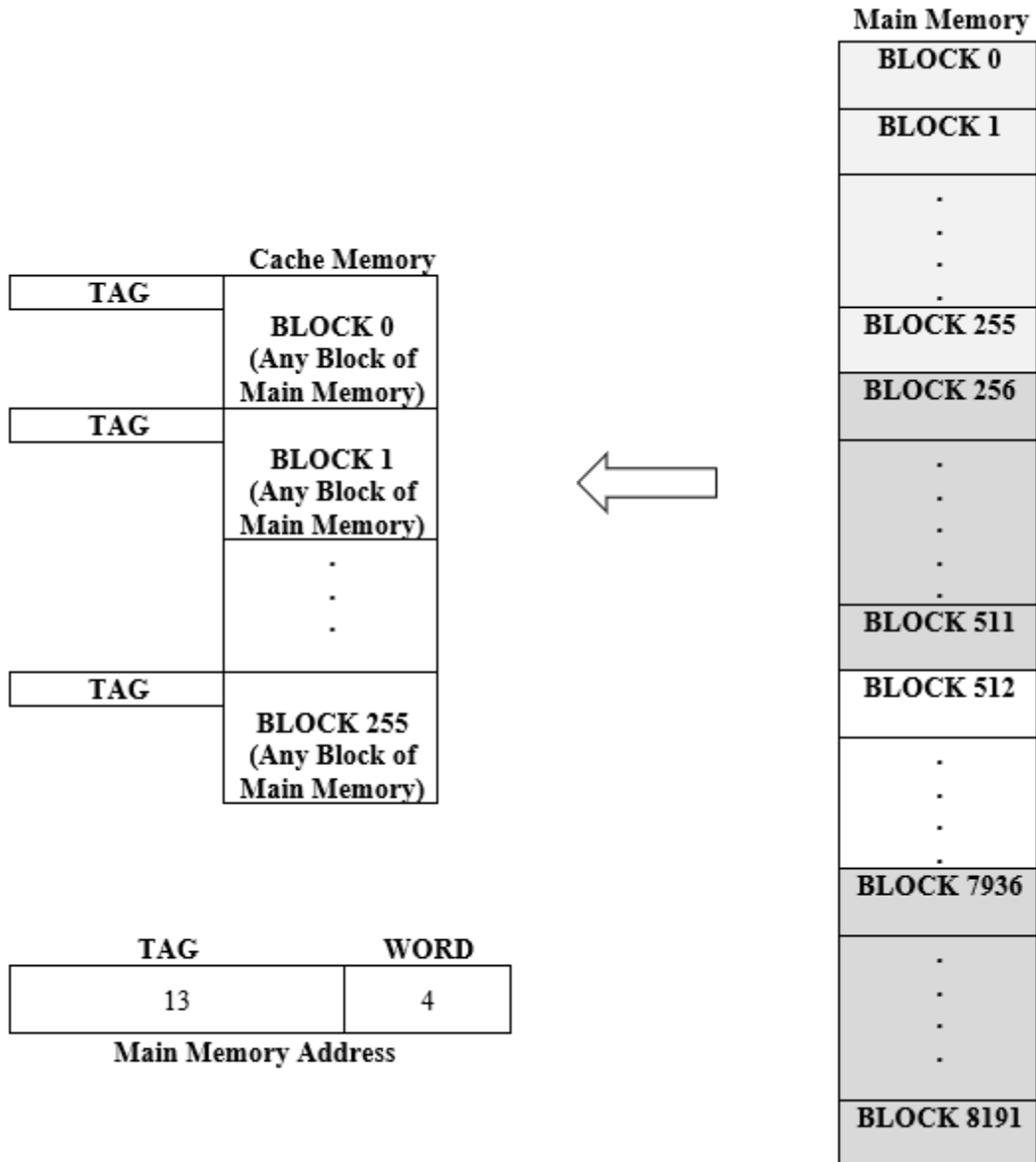


Figure 12.4: Associated Mapped Cache

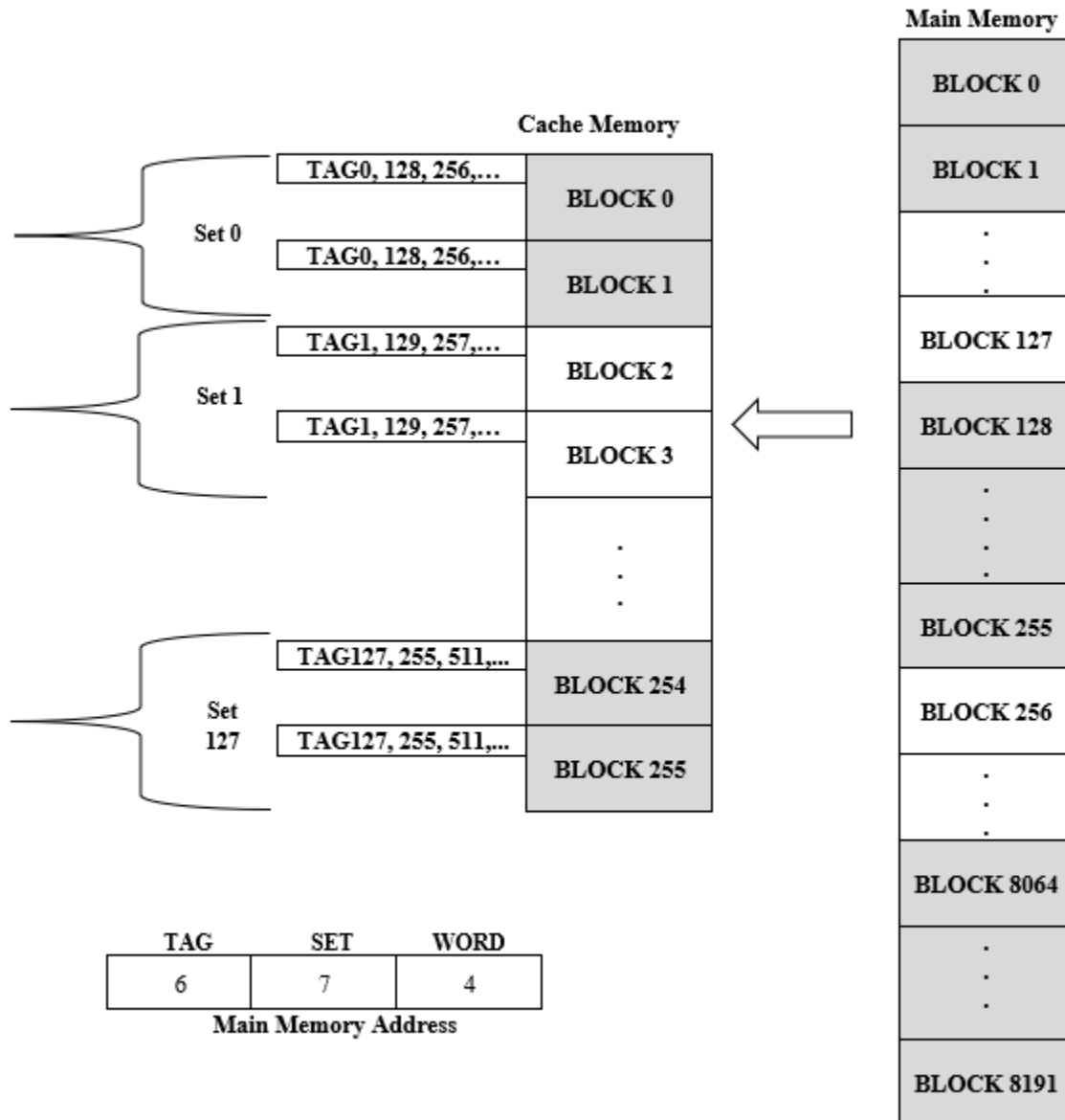


Figure 12.5: Set Associated Mapped Cache

12.6 Replacement Algorithms

Only if the cache is full should a new block be inserted into the cache to replace an old block. The replacement algorithm is used to replace the block. The four most prevalent replacement algorithms are as follows:

- LRU
- FIFO

- LFU
- Random

In LRU technique, the block that has been in the cache for the longest time without a reference is chosen for replacement. The idea behind this strategy is that the memory locations referenced recently are more likely to be referenced again. Therefore, it is better to replace the idle blocks. This strategy is more suitable for the two-way set associative cache organisation.

The FIFO is the most simple replacement technique. It selects the block that is residing in memory for longest time without any bias. It is simple and easy to implement but may replace those blocks that would be referenced again soon leading to more replacements. The LFU strategy selects the block having fewest references is chosen for replacement. There are no explicit criteria for replacing any block in the Random strategy. Any of the existing blocks are randomly chosen to replace.

12.7 Cache Updating

In a memory hierarchy, two copies of the same data can exist at different memory levels at the same time. The copies of the same data may be different in different memory levels. Therefore, it is important to keep all the copies updated. There are different techniques to update the cache memory.

- Write through system
- Buffered write through system and
- Write back system

12.7.1 Write through system

In write through updating system, data is copied to memory by the cache controller as soon as it is written to the cache. As a result, main memory always includes correct data, and any cache block can be replaced without losing data.

12.7.2 Buffered Write through system

In buffered write-through system, before the write cycle to the main memory is completed, the CPU can start a new cycle. The write accesses to the main memory are buffered as a result of this. When main memory is updated in such systems, read access, often known as a "cache hit," can be conducted simultaneously. However, the processor must wait for consecutive write operations to the main memory or read operations with "cache miss."

12.7.3 Write back system

In this method, data is written into the cache every time a change occurs, but only at specified intervals or under particular situations it is stored into the corresponding address in main memory. When a data location is modified in write back system, the data in cache is referred to as fresh, whereas the matching data in main memory is referred to as stale. If the cache controller receives a request for stale data in main memory, it updates the data in main memory before accessing it. Because writing data into cache alone takes less time than writing the same data into both cache and main memory, write back improves system performance. However, in the case of a crash or other adverse event, this speed comes with the danger of data loss.

12.8 Summary

Cache memories are small, high-speed buffer memories that are used in computer systems to temporarily store chunks of main memory that are (believed to be) in use. Information in cache memory can be accessible in a fraction of the time it takes to access data in main memory. A cache memory system consists of a large amount of slow low-cost memory, DRAM, and a small amount of rapid memory, SRAM. This system has been set up to simulate a large amount of rapid memory. When compared to main memory, the access time of cache memory is extremely fast. When the processor needs to read or write data from main memory, it first looks in the cache for a matching item. A cache hit said to occurs when the CPU discovers that the requested memory location is in the cache. A cache miss occurs when the CPU cannot locate the memory location in the cache.

Program locality is the prediction of the next memory location from the current memory address. Cache controller is enabled by programme locality. For a given time period, a computer programme tends to access the same set of memory locations, which is known as locality of reference. It appears in two forms: temporal and spatial. A recently executed command is likely to be executed again very soon, according to the temporal. Because of the spatial relationship, instructions that are placed close to recently performed instructions are more likely to be executed shortly.

The cache should be small enough that the overall average cost per bit is comparable to that of main memory alone, while also being large enough that the overall average access time is near to that of the cache alone. Cache mapping is a mechanism for transferring the contents of main memory to cache memory. In the situation of a cache miss, cache mapping specifies how a block from main memory is mapped to the cache memory. Three types of mapping techniques used in cache memory organization, direct mapping, associative mapping and set associative mapping. In direct mapping, the main memory blocks can be placed in only one place in the cache. In associative mapping, any block from main memory can be placed anywhere in the cache. Associative memory is the fastest and most versatile cache structure method. The set associative mapping is a hybrid of direct and associative mapping techniques. Blocks of the cache are grouped in to sets, and the mapping allows a block of the main memory to reside in any block of a specific set. Hence, the contention problem of the direct mapping is eased by having a few choices for block placement in set associative mapping.

Only if the cache is full should a new block be inserted into the cache to replace an old block. The replacement algorithm is used to replace the block. Two copies of the same data can exist in a cache system at the same time, one in cache and the other in main memory. Two different sets of data are associated with the same address if one copy is updated while the other is not. To avoid this, the caching system has been updated with features such as write through policy, buffered write through system and write back policy. In write through system, data is copied to memory by the cache controller as soon as it is written to the cache. In buffered write through system, cache uses a "write buffer" to hold data being written back to main memory. This frees

the cache to service read requests while the write is taking place. There is usually only one stage of buffering so subsequent writes must wait until the first is complete. Most accesses are reads so buffered write-through is only useful for very slow main memory. In write back system, only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.

Review Questions

1. How does a cache memory improve computer performance?
2. A cache memory needs an access time of 30ns and main memory 150ns, what is the average access time of CPU (assume hit ratio =80%) ?
3. A digital computer has a memory unit of 64k x 16 and a cache memory of 2^{10} words. The cache uses direct mapping with a block size of four words. How many bits are there in the Tag, Block and Word fields of address format?
4. Explain set associative mapping with an example?
5. Explain why associative mapping is fastest compare to other two techniques?
6. Consider a cache consisting of 128 blocks of 16 words each for a total of 2048 (2K) words, and assume that the main memory is addressable by a 16 bit address and it consists of 4 blocks. How many bits are there in each of the Tag, Block\Set and word fields for different mapping techniques?
7. Explain various replacement policies used in cache design?
8. Explain write through and write back policies used in cache memory system?



Bachelor of Computer Application

Block

5

SYSTEM ORGANIZATION

Unit 13

IO and System Control

Unit 14

Parallel Processing

BLOCK INTRODUCTION

Peripheral devices are also important for a computer system. A set of I/O modules is used to access and control these devices. Each I/O module controls one or more peripheral devices. I/O modules are responsible for performing communication between the peripheral devices and computer system. Block 4 presents a discussion on mechanism and functioning of I/O system. In addition, it also deals with the concept of parallelism. Block 4 is divided into Unit 13 and Unit 14. The Unit 13 is primarily concerned with the mechanisms of I/O devices and I/O systems. While, Unit 14 gives overview of parallel processing mechanisms and multiprocessor systems.

UNIT- 13 IO and System Control

Structure

13.0 Introduction

13.1 Objectives

13.2 I/O Module Functions

13.3 Programmed I/O

13.4 Interrupt Driven I/O

13.5 Interrupt Hardware

13.6 Direct Memory Access

13.7 I/O Channels and I/O Processors

13.8 Summary

Review Questions

Unit 13: IO and System Control

13.0 Introduction

The processor and memory are the most important modules of a computer system. In addition, a set of I/O modules is another important components of a computer system. Each I/O module controls one or more peripheral devices. All such modules interface with the system bus. I/O modules are responsible for performing communication between the peripheral devices and computer system with the help of the bus. The general model of an I/O module is shown in Figure 13.1.

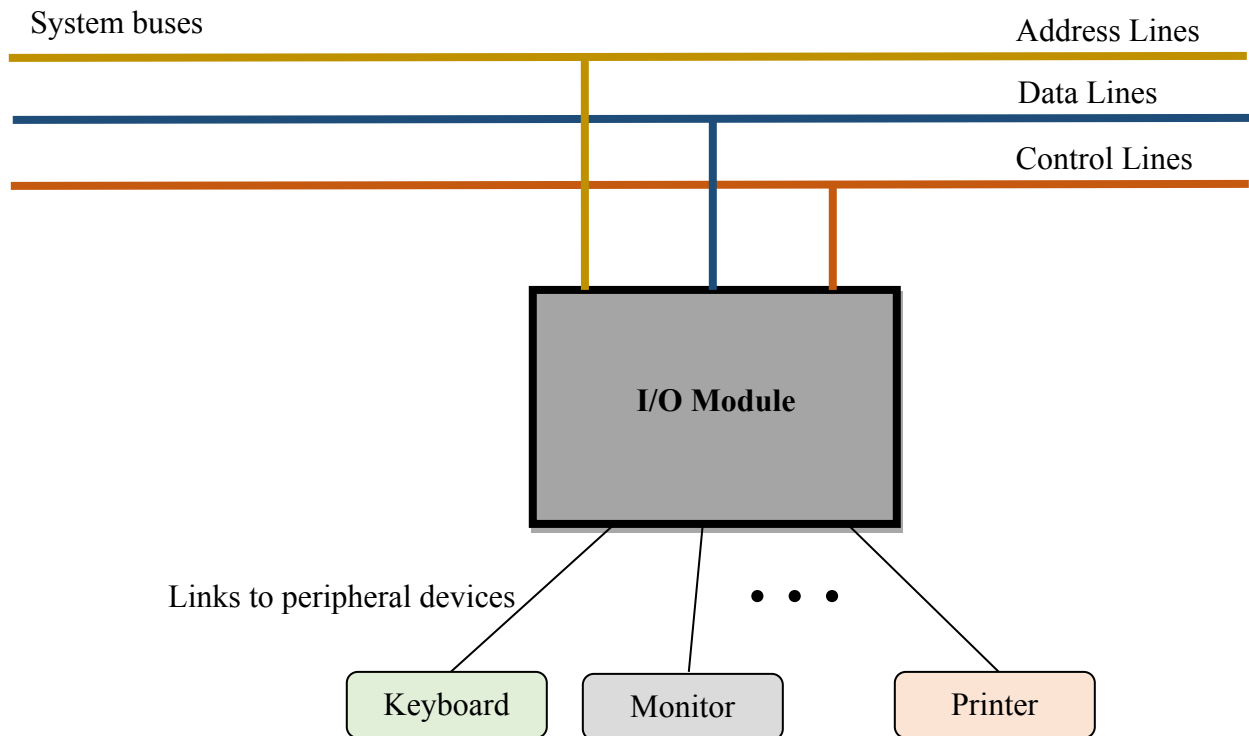


Figure 13.1: General model of an I/O module

As shown in Figure 13.1, a peripheral device is attached to the computer system through an I/O module. The status of the device, control information, and data are transferred between I/O module and the device through a link. The operation of a device is controlled by a control logic. Each device contains a transducer that converts electrical signals received by the device to other forms and vice versa. In addition, there is buffer with the transducer that temporarily holds the data that is transferred between the device and external environment. It was discussed in previous units that there is big

speed gap between processor and memory. This gap is even larger between processor and I/O devices. The buffer devices play important role to manage this speed gap. A peripheral device keeps the data received by the user in the input buffer and indicates the availability of data to the processor. When processor takes the data, it indicates the device to proceed further. Similarly, when processor has data to transfer to the device, it holds the data in an output buffer and indicates to the device. The device takes the data and indicates it to the processor to proceed further. This communication between processor and devices is carried out with the help of an I/O protocol.

13.1 Objectives

The learning objectives of this unit are as follows.

1. To explain the interaction mechanism between peripheral devices and computer system.
2. To discuss different types of I/O systems with their merits and demerits.
3. To understand operation of programmed I/O, interrupt driven I/O, and direct memory access.
4. To understand the function of I/O channels and processors.

13.2 I/O Module Functions

An I/O module is important for controlling the peripheral devices and related operations. The I/O modules perform a variety of operations including control and timing, communication, data buffering, and error detection, etc. During the execution of a program or process, the processor may need to communicate with one or more peripheral devices depending on the need for input or output. In order to perform data transfer, resources such as main memory and the system bus are shared input/output activities. Therefore, a control and timing function is required to coordinate the flow of traffic between internal resources and external devices. The communication process involves the following components or steps.

1. **Device Address:** A single I/O module can handle multiple peripheral devices. Each device associated with an I/O module is identified with a unique device address. The I/O module identifies the devices with their addresses.

2. **Control Signal:** The processor sends a command to the device through control bus that is intercepted by I/O module. The command is received in the form of control signal. The command parameters are transferred through data bus.
3. **Device Status:** The peripheral devices are very slow as compared to the processor. The device may be busy with previous operation or device may not be functional due to some error when I/O module received the command signal. Therefore, I/O module reports the device status to the processor before any data transfer takes place.
4. **Data Buffering:** When I/O module informs that device is ready, the related data exchange is performed over the data bus. The data are transferred either directly between processor and I/O module or between memory and I/O module. Usually signals are exchanged directly between I/O module and processor, while memory sends larger chunks of data. Both processor and memory are much faster than peripheral devices. The peripheral devices cannot receive data at such a high transfer rate. Although I/O module can operate at a speed compatible to the memory. To manage large speed gap between two components, the I/O module uses data buffering as also discussed earlier. The memory sends the data rapidly to the I/O module of a device. The I/O module holds the data in its buffer, which is sent to the destination device at its rate. Similarly, when a device sends the data to memory, the memory can get tied up due to slow data rate. Therefore, the I/O module uses data buffering during data transfer in reverse direction also. The I/O module receives the data from peripheral device and collects it in the buffer. Later, data are transferred at the desired rate.
5. **Error Detection:** I/O module is also responsible for error detection and reporting. A device could be non-working or malfunctioning due to some electrical and/or mechanical faults. Some form of error codes are used to identify and report the errors in the system. Besides electrical and mechanical malfunctions, unintentional changes in bit pattern could lead to another type of errors. I/O modules use parity bit checking to verify the correctness of the received bit pattern.

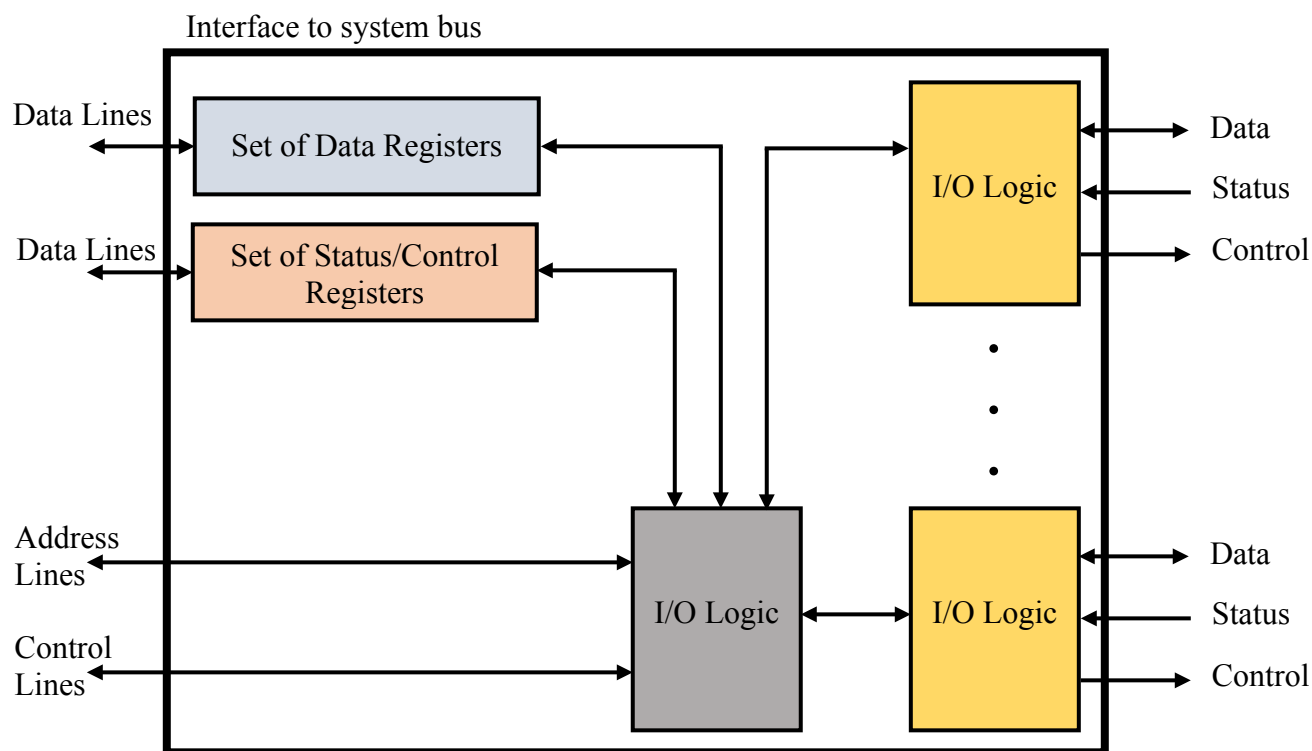


Figure 13.2: General structure of I/O module.

There are different types of I/O modules that vary in their complexity and the number of devices controlled by them. However, a general structure is followed by all the I/O module. Figure 13.2 shows the general structure of I/O modules. In this structure, there are some data registers that are used to buffer the data during send and receive operations between the devices and computer. It also includes some status registers, which maintain current status information. The status registers also serve as control registers to receive control information from the processor. Each I/O module consists of a logic unit that is responsible to interact with the processor. The logic unit communicates with the processor through control lines. The processor issues commands to the device through the set of control lines. An I/O module might be connected to multiple devices. It uses device specific logic interface to interact with the devices. Each external device is identifies by a unique device address. Therefore, I/O module maintains a set of device addresses and it is able to associate a particular

address to a specific device. The I/O modules are able to hide complicated details of the device to the processor and allow the processor to perform operation using simple read/write and open/close commands. Thus an I/O module can act as a high-level interface between the processor and the devices. Such I/O modules are referred as I/O controller or I/O channel. There is an important difference between I/O controller and I/O channel. The I/O channel takes the most of the processing burden, while an I/O controller requires the detailed control. I/O controller are used with minicomputers and I/O channels on the other hand are used with mainframe computers. Any kind of I/O module connects to the processor through a set of system buses as in the figure.

The I/O subsystems can be classified according to the extent of processor involvement in data transfer between processor and peripheral devices. The data can be transferred either between processor and I/O device or between memory and I/O device. In its simplest form, the entire processing is carried out between processor and the I/O devices. However, it is highly desirable to restrict the involvement of processor in I/O operations to save useful CPU cycles.

Check your progress 1

1. What are responsibilities of an I/O module in a computer system?
2. Why is a transducer used in a device?
3. How is the speed gap between device and processor or memory managed?
4. Write major components involved in communication between processor and devices?
5. What is I/O protocol?
6. What is the role of status registers?
7. Differentiate I/O controller and I/O channel.
8. How are the devices connected to the same I/O module are identified during data transfer?
9. Why is I/O logic used in an I/O module?
10. How does an I/O module verify the correctness of the bit pattern?

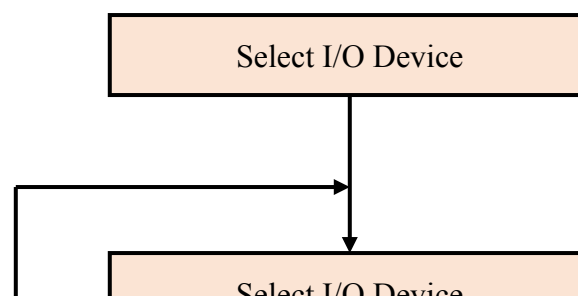


Figure 13.3: Work flow of programmed I/O.

13.3 Programmed I/O

In a programmed I/O subsystem, the processor executes a program that initiates the operation and checks the device status. If device is ready for operation, it sends a read or write command for transferring the data. In this arrangement, the processors needs to wait after issuing a command until I/O module completes its operation. Since the processor is much faster than the I/O module, it leads to the wastage of useful processor time. The working of the programmed I/O module is illustrated with the help of flow chart in Figure 13.3. When the processor encounters an I/O related instruction while executing a program, it selects the appropriate I/O device and issues a command to its I/O module. The I/O module performs the requested action and then sets device status with the help of status register. The I/O module does not inform or interrupt the processor about the status of device. Instead, the processor periodically checks the

status register. When processor finds the device status ready, it performs the read/write operation. There are four types of I/O command that can be issued by the processor to I/O module.

1. **Control:** The control command is used to activate the peripheral devices. A peripheral device is informed about the desired operation using the command.
2. **Test:** The test command is used to check the status of the peripheral device. The processor can test the availability of the device before data transfer. It checks if the device is powered on and the previous operation is over or not.
3. **Read:** With the read command the processor causes the I/O module to obtain the data from the device. The I/O module receives the data and places it in the internal buffer. The processor can then receive the data through the data bus.
4. **Write:** By issuing the write command the processor causes the I/O module to take the data from the data bus. The I/O module receives the data over data bus and subsequently sends it to the peripheral device.

Programmed I/O works under the processor control. Operations under programmed I/O involve a complete instruction fetch, decode, and execute cycle. It is useful where data transfer is carried out character by character. But programmed I/O is highly slow leading to significant loss of CPU time in busy waiting. The busy-wait feature of programmed I/O is its main disadvantage. Even a moderate I/O operation significantly degrades the CPU performance. Therefore, some mechanism is highly desirable to manage the speed gap between processor and peripherals. The status registers play important role to avoid processor overwriting the device contents before it is used by the device.

During program execution, the processor fetches the I/O-related instructions from memory and issues the I/O commands to an I/O module to execute the instructions. The instructions often have one-to-one mapping into I/O commands. Typically, multiple I/O devices are connected through the same I/O modules to the system. As discussed earlier, each device is given a unique address. The I/O command issued to an I/O module contains the address of the desired device. The I/O module interprets the address lines to check if the issued command is for itself. There are two possible addressing modes: memory mapped and isolated. With memory-mapped addressing, the I/O devices and memory locations share the same addressing space. The full range

of addresses may be used for both memory and I/O devices. The same bus can be used for memory related and I/O operations with a single read line and a single write line on the bus. In another arrangement, the bus may consist of memory read/write plus I/O command lines. With this arrangement, the command line needs to specify whether the address refers to a memory location or an I/O device. In isolated I/O, the I/O address space is separated from memory address space. In this case, the I/O ports are accessible only by I/O commands. With isolated I/O a smaller instruction set is used. While, a range of instructions can be made available with memory mapped I/O. It is good for programming but needs more memory space. Both type of addressing is used in commercial computers.

13.4 Interrupt Driven I/O

The busy-wait feature of the programmed I/O is the main problem with programmed I/O as the processor has to spend a long time idle waiting for the I/O module of the device to become ready. The processor frequently checks the status of the I/O module before transmitting the data. It significantly degrades the overall performance of the system. It is highly desirable to reduce the involvement of processor in the procedure of I/O operations to save the useful CPU cycles. The system throughput can be improved by saving the CPU cycles in such operations. Interrupt driven I/O is an alternative to programmed I/O that does not involve processor in busy-wait. It does not require the processor to repeatedly check the device status. Instead, the processor goes to its usual work after issuing the command to I/O module. When I/O module is ready to serve the command, it interrupts the processor to exchange data. The processor halts the current execution and performs the data transfer and then resumes its work.

In input operation, the processor issues a READ command and goes off to work on something else. It checks for interrupt at the end of each instruction cycle. When the processor receives the interrupt from the I/O module, it saves the context of the current program and goes to process the interrupt. The processor reads the data from I/O module through data bus and stores it in memory. It then resumes execution of the program whose context was saved earlier.

When I/O module receives the READ request, it proceeds to read data from the associated peripheral device. The data from the device is received in data register of the I/O module. After receiving the data in its data register, the I/O module signals an interrupt to the processor and waits until its data are requested by the processor. When the processor request is received, the module transfers its data through data bus.

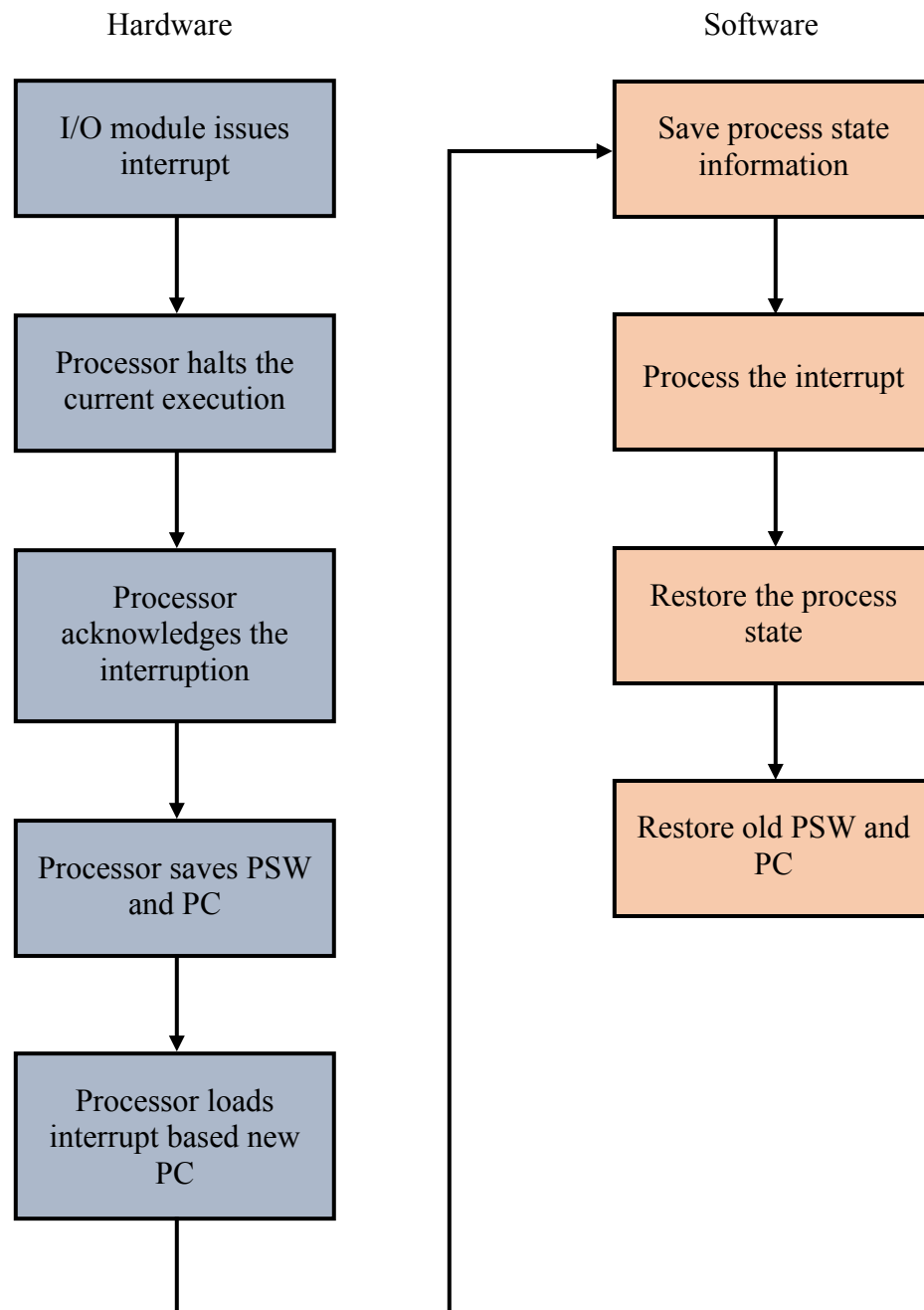


Figure 13.4: Involvement of processor in interrupt driven I/O.

The involvement of processor in interrupt driven I/O is shown in Figure 13.4. The sequence of events takes place as follows.

1. An interrupt signal issued by I/O module to the processor.
2. The processor responds to the interrupt only after finishing with the execution of the current instruction.
3. The processor finds the interrupt and sends an acknowledgment to the device. After receiving the acknowledgement, the device removes the interrupt.
4. The processor saves its status in program status word (PSW) register. It also saves the location of next instruction to be executed in program counter (PC) register. After saving the necessary information, the processor transfers the control to the interrupt routine. There are different implementations of the interrupt routine. It could be a single program for all kind of interrupts or a separate program could be written for each different interrupt. The interrupt routines could also be developed according to device type i.e. one routine for each device type. Therefore, the processor needs to determine the interrupt type and accordingly the interrupt handling routine is invoked.
5. Now the processor loads the entry location of the interrupt-handling program into PC and processor proceeds to the next instruction cycle.
6. Some other information such as state of the current program, processor registers, and stack pointer, etc. is also needed to save before processing the interrupt.
7. After saving all the necessary information, it's now turn of the interrupt handler to process the interrupt.
8. Once the interrupt processing is over, the saved information is retrieved from the stack and registers are restored.
9. Finally, the PSW and PC values are restored from the stack and the next instruction from the previously interrupted program is executed.

The interrupt can occur asynchronously at any point of time during the execution of user programs. Therefore, it is necessary to save important state information about the interrupted program to resume it later. The interrupt relieves the CPU from periodically checking the device status, thus saving the useful CPU time. Most processors complete

the current instruction cycle before serving the interrupt. When processor receives the interrupt signal, it may not be interested to serve the interrupt. It can indicate its willingness to accept the interrupt by setting a flag. This flag indicates the willingness of the processor to the device.

When processor is serving an interrupt and yet another interrupt occurs, the response to the new interrupt depends upon the priority of the newly arrived interrupt. If the newly arrived interrupt has a lower or equal priority to currently served interrupt, then it has to wait until the current interrupt is served. On the other hand, if the newly arrived interrupt has a higher priority over the currently served interrupt, then the processor saves the status of currently served interrupt and serves the newly arrived interrupt.

13.5 Interrupt Hardware

The computers need hardware support for the processor to recognize and serve the interrupts. The specialized interrupt lines are provided to the processor, which are used to send interrupt signals to the processor. If there are multiple peripheral devices, multiple simultaneous interrupt requests could be generated by different devices. The processor must be able to recognize and handle the multiple interrupt requests. It should be able to recognize the device generating the interrupt request. There are four general techniques to identify the device.

13.5.1 Multiple Interrupt Lines

A straightforward approach is to use dedicated multiple interrupt lines between I/O modules and processor. However, it is difficult to dedicate too many bus lines or processor pins to interrupt lines. On the other hand, even if multiple interrupt lines are used, multiple I/O modules are likely to be connected with an interrupt line. Therefore, this approach is highly impractical and rarely used in the commercial computers.

13.5.2 Software Poll

Software poll is an alternative to multiple interrupt lines approach. This approach uses an interrupt-service routine, which responsible to poll each I/O module and identify the

device that caused the interrupt. The poll could be conducted through conducted with help of a separate command line. When the processor detects an interrupt, it calls the interrupt-service routine to determine the interrupting device. Alternatively, the processor can read the status register of I/O modules to identify the interrupting module.

13.5.3 Daisy Chain

The software poll is a time consuming technique. The hardware poll is an efficient alternative to the software poll. Daisy chain is an effective hardware poll technique that uses a common interrupt request line for all the modules. The interrupt acknowledge line is daisy chained through the modules. If the processor finds an interrupt, an acknowledgement is sent that propagates through a series of I/O modules before it reaches the requesting module. Upon receiving the acknowledgement, the requesting module responds by placing a word on the data lines. This word contains the address or unique identifier of the I/O module. In this approach no general interrupt-service routine is required.

13.5.4 Bus Arbitration

There is a technique known as independent source bus arbitration that allows each device to have its own interrupt request line and grant line. A device can send its interrupt request over the interrupt request line independent of other devices. The grant line is used by the device to receive the grant signal for its request. In this arbitration scheme, the priority of a device is independent of the device location.

Check your progress 2

1. What is the main disadvantage of the programmed I/O?
2. Write the name of commands issued by I/O module in programmed I/O.
3. How many addressing modes are there for I/O modules? Write the addressing modes.
4. How is the speed gap between processor and I/O module managed?
5. How does a processor respond when an interrupt is issued to it?
6. What is usage of PSW and PC in interrupt driven I/O?
7. When does processor respond to the interrupt?

8. Why is it necessary to save the information of the interrupted program?
9. What is the main advantage of the interrupt driven I/O?
10. How does the processor indicate its willingness to accept the interrupt?
11. How are the multiple interrupts handled?
12. What type of hardware support is required for interrupt driven I/O?
13. Write different mechanisms for identifying interrupt generating device.
14. What is the major limitation of having multiple interrupt lines?
15. What is responsibility of interrupt service routine in software poll?
16. How is the daisy chain is efficient than software poll?

13.6 Direct Memory Access

The interrupt driven I/O reduces the involvement in the data transfer between peripherals and computer system. But still, the processor play an important role in the interrupt based mechanism for data transfer. It would be good if the processor is involved in peripheral related operations. Direct memory access more popularly known as DMA is one such technique that further reduces the intervention of the processor in data transfer between peripheral devices and memory. It allows the CPU to perform other important work by providing direct memory access to the devices. It is highly effective in case of large data transfers.

DMA uses an additional module known as DMA controller on the system bus. DMA controller is a hardware component that controls the peripheral devices. DMA controller takes place of processor during the data transfer from or to the memory. For this purpose, it uses the same system bus that is used by the processor. Therefore, it must wait for the processor to free the system bus or it should request the processor to temporarily suspend its operation. When processor receives the request from DMA, it returns the acknowledgement granting the access to system bus. After taking the permission, DMA can conduct the data transfer through the bus. The processor that supports DMA uses some input output signals for receiving DMA request and sending acknowledgement. Because DMA controller does a kind of bus cycle stealing, this technique is called cycle stealing. When processor needs to send or receive data, it

issues a command to DMA controller. The system bus sharing mechanism used by DMA is shown in Figure 13.5.

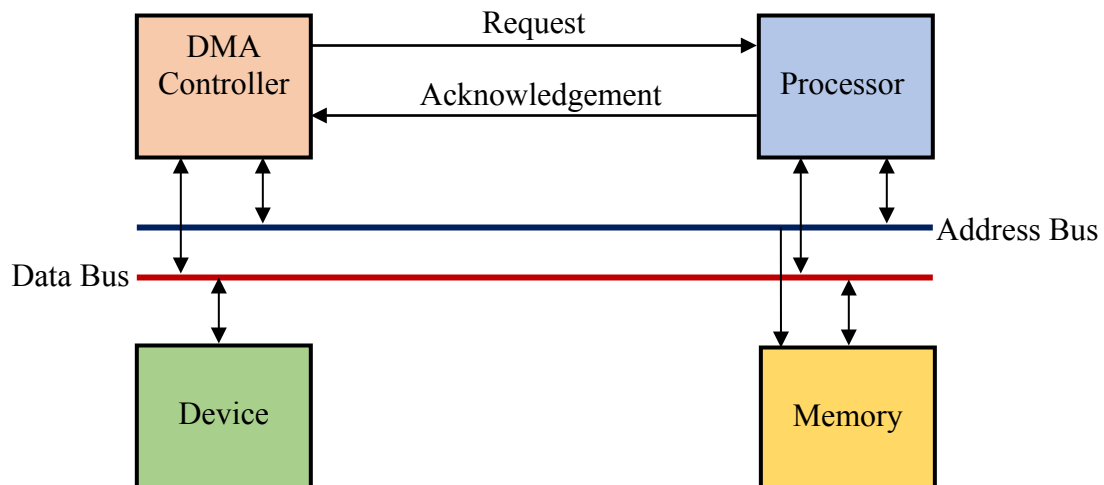


Figure 13.5: Bus sharing by DMA and processor

A DMA controller consists of an *address* register, a *data count* register, and a *control* register. The address register specifies the address of memory location that contains the data to be transferred. The data count register is used to specify the number of words to be transferred. It is decremented by one after each word transfer. The transfer mode is specified by the control register. DMA is able to transfer the data in single cycle mode or burst mode. In burst mode, DMA keeps control of the bus until entire data is transferred from or to memory. In single-cycle mode, DMA relinquishes the bus after transferring each data word. The single-cycle mode results in higher overhead as request/acknowledge sequence is performed for each data word transfer. It could degrade the system performance especially in case of large data transfer. It is preferred if the system cannot tolerate interrupt latency of more than a few cycles.

13.7 I/O Channels and I/O Processors

DMA technology does not take complete control of I/O operations from the processor. I/O channel is an extension of the DMA that has the ability to take complete control of I/O instructions. The I/O channel is often also called I/O processor. The processor does not execute I/O instructions if I/O channels are used in a computer system. The processor only initiates an I/O transfer by instructing the I/O channel. The I/O channel executes program in memory that specifies the devices, memory locations, priority, and instructions for handling some error conditions. There are two types of I/O channels: selector channel and multiplexor channel. A selector channel can control multiple high-speed devices. But at any one time, only one of those devices performs data transfer. A particular device is selected by the I/O channel for the data transfer. With each device an I/O module is associated that is used to handle that device. A multiplexor channel on the other hand is able to handle multiple devices at the same time.

Check your progress 3

1. What is DMA controller?
2. Which system bus is used by DMA for data transfer?
3. How does DMA take access to system bus from the processor?
4. What is cycle stealing?
5. Explain the usage of address register, data count register, and control register in DMA controller?
6. Why is there higher overhead in single-cycle mode in DMA?
7. Differentiate DMA and I/O channel.
8. What is the main advantage of I/O channel or I/O processor?
9. Write different types of I/O channels.
10. Differentiate selector and multiplexor channels.

13.8 Summary

A set of I/O modules is an important components of a computer system. Each I/O module controls one or more peripheral devices. All such modules interface with the system bus. I/O modules are responsible for performing communication between the

peripheral devices and computer system with the help of the bus. A peripheral device is attached to the computer system through an I/O module. The status of the device, control information, and data are transferred between I/O module and the device through a link. The operation of a device is controlled by a control logic. A buffer is used to temporarily hold the data that is transferred between the device and external environment. The buffer devices play important role to manage this speed gap. There are different types of I/O modules that vary in their complexity and the number of devices controlled by them.

In a programmed I/O subsystem, the processor executes a program that initiates the operation and checks the device status. If device is ready for operation, it sends a read or write command for transferring the data. In this arrangement, the processors needs to wait after issuing a command until I/O module completes its operation. Since the processor is much faster than the I/O module, it leads to the wastage of useful processor time. The busy-wait feature of programmed I/O is its main disadvantage. Even a moderate I/O operation significantly degrades the CPU performance. It is highly desirable to reduce the involvement of processor in the procedure of I/O operations to save the useful CPU cycles. The system throughput can be improved by saving the CPU cycles in such operations. Interrupt driven I/O is an alternative to programmed I/O that does not involve processor in busy-wait. It does not require the processor to repeatedly check the device status. Instead, the processor goes to its usual work after issuing the command to I/O module. When I/O module is ready to serve the command, it interrupts the processor to exchange data. The processor halts the current execution and performs the data transfer and then resumes its work.

The computers need hardware support for the processor to recognize and serve the interrupts. The specialized interrupt lines are provided to the processor, which are used to send interrupt signals to the processor. If there are multiple peripheral devices, multiple simultaneous interrupt requests could be generated by different devices. The processor must be able to recognize and handle the multiple interrupt requests. It should be able to recognize the device generating the interrupt request. There are four general techniques to identify the device: multiple interrupt lines, software poll, daisy chain, and bus arbitration. Multiple interrupt lines are not used in the modern computers.

The interrupt driven I/O reduces the involvement in the data transfer between peripherals and computer system. But still, the processor play an important role in the interrupt based mechanism for data transfer. It would be good if the processor is involved in peripheral related operations. Direct memory access more popularly known as DMA is one such technique that does not require the intervention of the processor in data transfer between peripheral devices and memory. It allows the CPU to perform other important work by providing direct memory access to the devices. It is highly effective in case of large data transfers. DMA uses an additional module known as DMA controller on the system bus. DMA controller is a hardware component that controls the peripheral devices. DMA controller takes place of processor during the data transfer from or to the memory. For this purpose, it uses the same system bus that is used by the processor. Therefore, it must wait for the processor to free the system bus or it should request the processor to temporarily suspend its operation. When processor receives the request from DMA, it returns the acknowledgement granting the access to system bus. After taking the permission, DMA can conduct the data transfer through the bus.

DMA technology does not take complete control of I/O operations from the processor. I/O channel is an extension of the DMA that has the ability to take complete control of I/O instructions. The I/O channel is often also called I/O processor. The processor only initiates an I/O transfer by instructing the I/O channel. The I/O channel executes program that specifies the devices, memory locations, priority, and instructions for handling some error conditions. Selector channel and multiplexor channel are two types of I/O channels. The selector channel can control multiple high-speed devices but only one device can perform data transfer at a time. The multiplexor channel can handle multiple devices at the same time.

Review Questions

Q.1 What is I/O module in a computer system? With the help of a diagram explain the general model of an I/O module.

Q.2 What is the role of buffer devices in I/O module?

Q.3 Write and explain major steps or components involved in the communication between a peripheral device and processor.

Q.4 With the help of a diagram explain the general structure of an I/O module and explain the job of each component.

Q.5 How does programmed I/O work? Explain with the help of a flow chart.

Q.6 Write and explain all kind of commands that are used in a programmed I/O.

Q.7 How many types of addressing modes are used in programmed I/O? Explain each and differentiate them.

Q.8 What is the major advantage of interrupt-driven I/O over programmed I/O? Explain the working process of interrupt-drive I/O.

Q.9 Explain the involvement of the processor in interrupt-driven I/O with help of a diagram.

Q.10 What are the major techniques to identify a device? Explain each.

Q.11 What are the major advantages of DMA over other I/O techniques? Explain its working process.

Q.12 With the help of a diagram, explain the bus sharing mechanism between DMA and processor.

UNIT- 14 Parallel Processing

Structure

14.0 Introduction

14.1 Objectives

14.2 Parallel Processing Mechanisms

14.3 Multiprocessor Systems

14.4 Loosely Coupled Multiprocessors

14.5 Tightly Coupled Multiprocessors

14.6 Multiprocessor Operating System

14.7 Summary

Review Questions

Unit 14: Parallel Processing

14.0 Introduction

Earlier computer systems were considered to operate in a sequential manner that execute one instruction at a time. Although, it is not completely true. Even the early computers could have generated multiple control signals simultaneously. Modern computers consists of several components such as instruction pipelining that are able to function in overlapping manner. Now computers have multiple execution units to execute multiple instructions simultaneously. The parallel execution of instructions or programs can significantly enhance the performance of the computer system. With the advancement in hardware technology and its availability at affordable cost, the computer designers have more opportunities to consider parallelism for better performance. Formally the parallel processing as defined by Hwang and Briggs (1984)

Parallel processing is an efficient form of information processing, which emphasizes the exploitation of concurrent events in the computing process. Concurrency implies parallelism, simultaneity, and pipelining. Parallel events may occur in multiple resources during the same time interval; simultaneous events may occur at the same time instant; and pipelined events may occur in overlapped time spans. These concurrent events are attainable in a computer system at various processing levels. Parallel processing demands concurrent execution of many programs in the computer. It is in contrast to sequential processing. It is a cost effective means to improve system performance through

is given in the following box.

Parallel processing needs collective efforts in the field of hardware, software, algorithms, and languages. The parallelism can be achieved from program level to intra-instruction level. The role of hardware in parallelism increases from program level to intra-instruction level. It is important to have a trade-off between hardware and software. The uniprocessor systems have their limitations limitation to achieve high degree of parallelism. The computing power can be increased by uniprocessor architecture to multiprocessor architecture. A computer system with multiple processors and shared memory space and peripherals is known as multiprocessor system. Like a conventional uniprocessor system, the multiprocessor systems also run under the control of a single operating system. However, multiprocessor operating systems have some different management and design issues.

14.1 Objectives

The learning objectives of this unit are as follows.

1. To provide an overview of the parallel processing mechanisms.
2. To introduce the processor level parallelism.
3. To understand design issues of multiprocessor system.
4. To understand software requirements for multiprocessors.

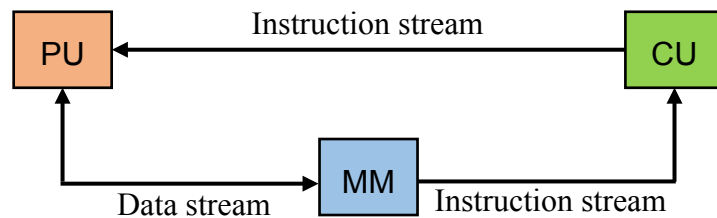
14.2 Classification of Computers

There exist many different types of computers that can be differentiated on the basis of their architectural features. The computers having common features can be put together into one category. There are various categorization or classification schemes including Flynn's classification, Feng's classification, Hwang and Briggs classification, and Bell's classification, etc. Flynn's classification is one of the most popular classification schemes that divides the computers on the basis of the multiplicity of instruction/data stream units. According to Flynn's classification, the computers are classified into following categories.

- Single Instruction Stream Single Data Stream (SISD)

- 🎬 Single Instruction Stream Multiple Data Stream (SIMD)
- 🎬 Multiple Instruction Stream Single Data Stream (MISD)
- 🎬 Multiple Instruction Stream Multiple Data Stream (MIMD)

SISD computers are mostly the sequential processing computers that may also have some degree of overlapping in their execution stages. The modern serial computers and pipelined computers fall under this category. These computers consist of a single control unit. However, there could be multiple functional units in the system. The conceptual illustration of SISD is given in Figure 14.1.



PU: Processing Unit, MM: Memory Module, CU: Control Unit

Figure 14.1: SISD architecture

The SIMD computers are able to process multiple data streams simultaneously, but all the data streams are operated by the same instruction. There are multiple arithmetic logic units (ALUs) known as processing elements. All the processing elements are controlled by the same control unit and they share the same main memory as shown in Figure 14.2. The memory may have multiple modules. The control unit broadcasts the instruction to the processing elements. The processing elements perform the same operation on different data sets.

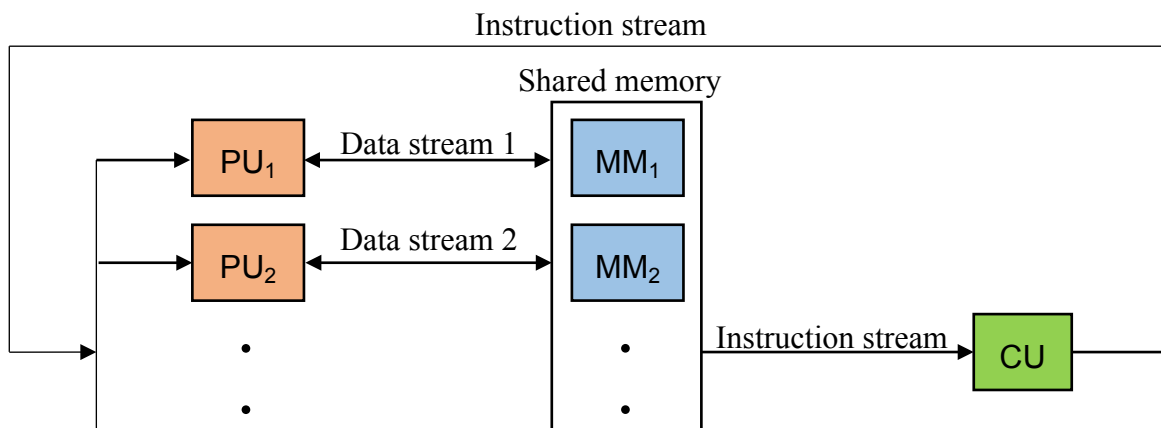


Figure 14.2: SIMD architecture

The MISD computers consist of multiple processing units that conceptually operate on the same data stream. The multiple processing units work in a macro-pipe such that output of one processing unit is the input for the next processing unit. However, there is no known commercial computer available that is based on MISD architecture. Although, some pipelined computers and systolic arrays are considered by some architects as examples of MISD architecture. The conceptual design of SIMD computers is shown in Figure 14.3.

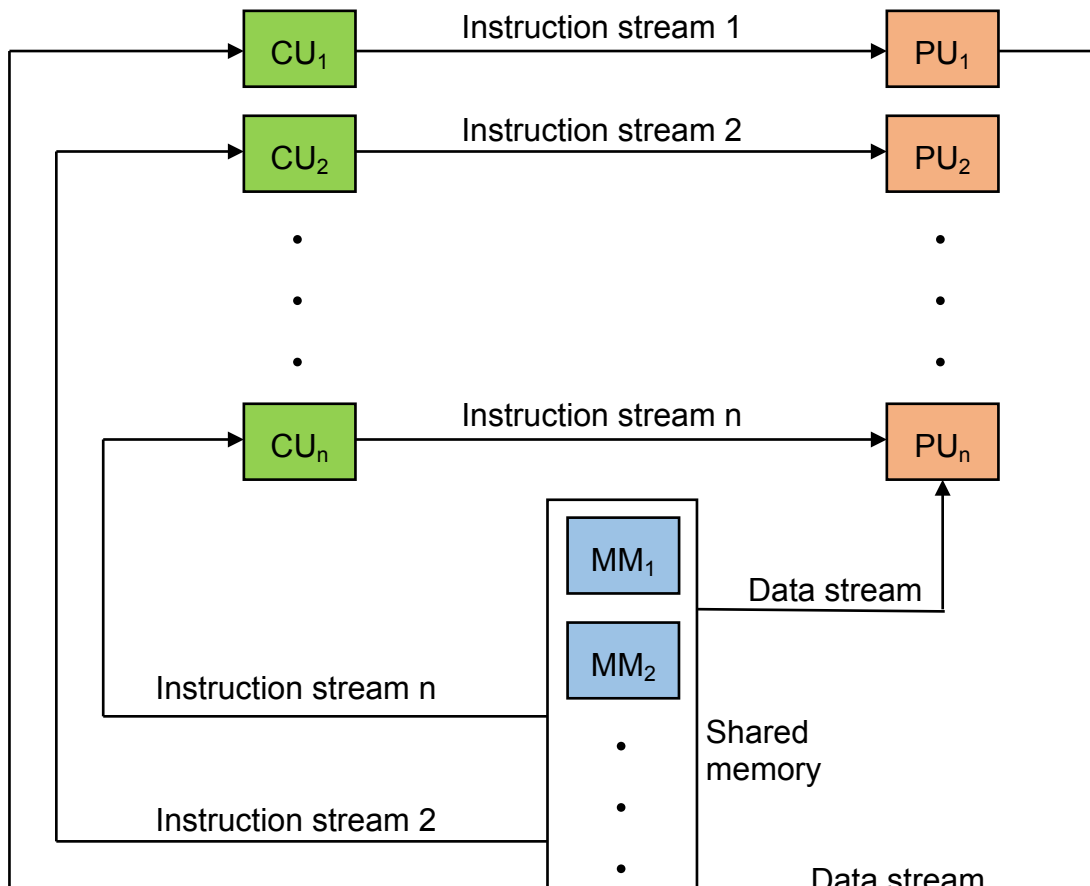


Figure 14.3: MISD architecture

The multiprocessor systems and multi-computer systems fall under MIMD category, which are able to process multiple instructions on the different data streams simultaneously. The processing units in MIMD computers interact with the help of a shared memory or through an interconnection network. Based on the type of method used for communication, the MIMD computers can be further classified into different subcategories. The conceptual illustration of SISD is given in Figure 14.4.

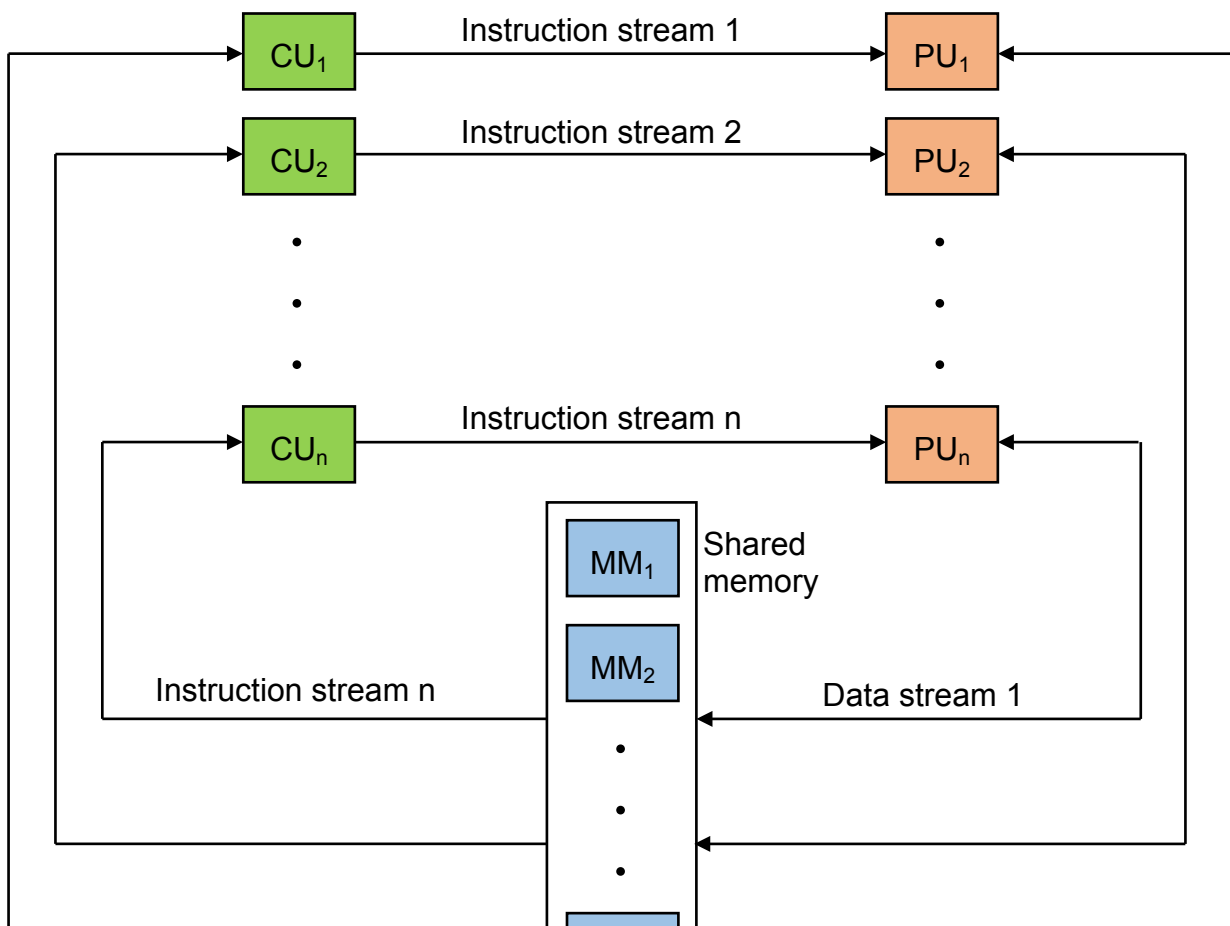




Figure 14.4: MIMD architecture

14.3 Parallel Processing Mechanisms

The parallel computer architecture can be divided into three main categories depending on their features. Three main categories of paralleling processing approaches are as follows:

1. Pipelined systems
2. Array processors
3. Multiprocessor systems

Although, these three categories are not mutually exclusive and a parallel processing system the most of these features to some extent.

14.3.1 Pipelined Systems

The concept of pipelining is discussed already introduced in Unit 9. As discussed earlier, it is a technique that executes program instructions/operations in an overlapped manner to achieve parallelism. An instruction can be divided into some steps. Usually, there are four major steps of an instruction including instruction fetch, instruction decode, operand fetch, and execution. A sequential system completes all these parts of an instruction before taking the next instruction into execution. A pipelined computer arranges the different stages of an instruction as a linear cascade to execute successive instructions in an overlapped fashion. When one instruction moves from one stage to next step, the other instruction comes in to occupy the vacated stage. The delay at different stages is different usually. Therefore, instructions move from one

stage to another stage at the pace of the slowest stage in the pipeline to avoid any kind of conflict. The operation of all the stages is synchronized with the help of a common clock. The common clock triggers the flow of data from stage to stage. Interface latches are used between two stages to temporarily hold the intermediate results. If there are n stages in a pipeline, the first output is generated after n clock periods. But once the pipeline is filled, it gives output every clock period. As compared to a sequential processor, a pipelined processor is at most n times faster if there are n stages in the pipeline. However, this kind of speed-up can be achieved only in an ideal case. Actually, that kind of speed-up could not be achieved due to several reasons such as data dependency, interrupts, and memory dependencies. There could be different types of pipelines such as instruction pipeline, arithmetic pipeline, scalar pipeline, and vector pipeline, etc. Different pipelines can be dedicated to different operations.

14.3.2 Array Processors

Array processors consist of multiple ALUs that can operate in parallel. These ALUs are known as processing elements. The pipelines processors achieve temporal parallelism. With the help of multiple processing elements, the array processor achieve spatial parallelism. The processing elements are able to perform the same function at the same time. Therefore, array processors are suitable to carry out vector operations in parallel. Each processing element consists of ALU, a local memory, and some registers. All the processing elements in an array processor are interconnected through a data routing network and work under the control of a single control unit. The processing elements do not have any separate control units. For any vector operation, the control unit takes the responsibility of instruction fetching from the control memory and to decode it. The decoded instruction is broadcast to all the processing elements involved in the operation. The processing elements fetch operands from their local memories. The interconnection network plays an important role in array processors. There are many different types of interconnection networks that can be used in array processors for internal communication. Parallel algorithms are also required for carrying out vector and matrix operations such as multiplication, sorting, and fast Fourier transformation.

14.3.3 Multiprocessor Systems

A multiprocessor system consists of multiple processors that are usually comparable in their capabilities. All the processors have access to a shared memory and peripheral devices. Each processor also has its own local memory and private I/O devices. The entire system works under the control of a single operating system. The operating system is responsible for interaction between processors besides other resource management tasks. The processors communicate and cooperate at job, task, or data levels to solve a problem. The communication could be done with the help of a shared memory or through an interrupt network. Multiprocessor systems are often differentiated by the type of interconnection network used.

At architectural level, the multiprocessors can be divided into two major categories: loosely-coupled and tightly-coupled multiprocessors. In a tightly-coupled multiprocessor system, the processors communicate with the help of a shared main memory. The processors and memory are interconnected either through multi-ported memory or through an interconnection network. In tightly-coupled multiprocessor system, managing the memory conflicts among different processors is a major issue.

In loosely coupled multiprocessor systems, each processor has a large local memory and I/O devices. The processors do not communicate through memory instead a message transfer system is used to exchange messages for information sharing. The advantage of message transferring system is that no memory conflicts are observed unlike tightly coupled multiprocessors. But there are larger communication delays in case of message transferring system. Therefore, tightly coupled multiprocessors are preferred if a high degree of interaction is required among the processors. While, loosely coupled multiprocessors are good if communication requirement is minimal.

Check your progress 1

1. Formally define parallel processing.
2. Write the categories of computers according to Flynn's classification.
3. Pipelined computers are sequential processing computers or parallel processing computers?

4. Differentiate SIMD and MIMD computers.
5. Write an example of MISD computers.
6. How do processors in MIMD computers communicate?
7. How is the delay between different stages managed in a pipelined computer?
8. What kind of speed-up is achieved by pipelined computers?
9. Differentiate the parallelism achieved by pipelining and parallelism achieved by array processors.
10. Differentiate loosely coupled and tightly coupled multiprocessor systems.

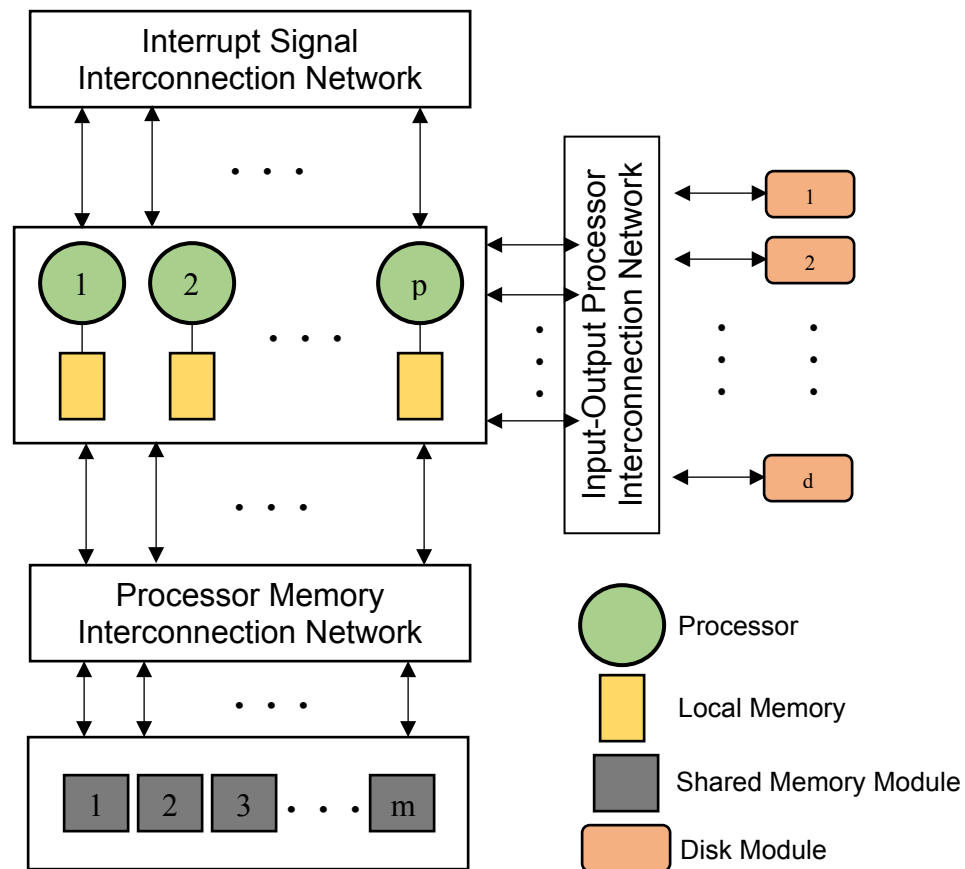


Figure 14.5: Tightly coupled multiprocessor system without private cache memory.

14.4 Tightly Coupled Multiprocessor

As discussed earlier, the processors in tightly coupled multiprocessor systems have a local memory and a shared memory that is used for communication and exchange of information. The shared memory is a fast memory such as cache to deal with the conflicts. If high degree of interaction is needed in an application, the tightly coupled multiprocessor systems are a good choice. The tightly coupled multiprocessor systems are also good for high speed processing. There are two major models for tightly coupled multiprocessor systems. One of these models is shown in Figure 14.5. In this model, each processor has its local memory and share a global memory. The local memory stores operating system related code and data for the processor. There are three different types of interconnection networks: Interrupt Signal Interconnection, Input-Output Interconnection Network, and Processor Memory Interconnection Network. The global memory is divided into several memory modules. The processors are connected to memory modules through Processor Memory Interconnection Network. One memory module can be connected to only one processor during a memory cycle. If there are requests from multiple processors to access the same memory module during the same memory cycle, it results in memory conflicts. Such memory conflicts are handled by Processor Memory Interconnection Network. The number of memory modules is kept compatible to the number of processors in the system to reduce the memory conflicts. To further reduce the memory conflicts, a local cache can be associated with each processor. The cache reduces the traffic to Processor Memory Interconnection Network that helps to reduce the requests to access memory modules. The Interrupt Signal Interconnection Network is responsible to transfer interrupt from one processor to other processors. Similarly, Input-Output Interconnection Network connects processors to peripherals. Cyber-170, UNIVAC 1100/94, Cray X-MP, Honeywell 60/66, IBM 3081, and PDP-10 are the example of tightly coupled multiprocessor system.

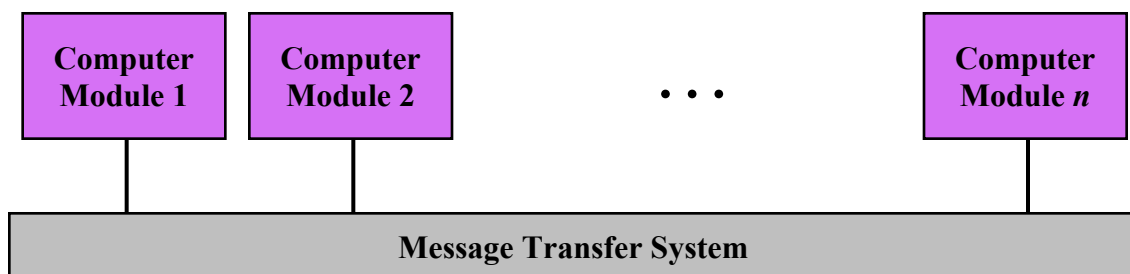


Figure 14.6: Loosely coupled multiprocessor system.

14.5 Loosely Coupled Multiprocessor

In loosely coupled multiprocessor systems each processor has a large local memory and a local set of peripheral devices. Therefore, processors access most of the instructions and data in their own local memories. Thus loosely coupled multiprocessor systems do not experience that kind of memory conflicts as observed in tightly coupled multiprocessor systems. In case a process executes on multiple processors, the communication among the processors takes place through a message transferring system. Loosely coupled multiprocessors are preferred when communication among the processors is not required frequently. A processor with its local memory and I/O devices is referred as *computer module*. The computer modules are connected to message transfer system as shown in Figure 14.6. In this system, all the computer modules are connected in a non-hierarchical manner. It is also possible to arrange computer modules in a hierarchical manner for large multiprocessor system. In such a system a cluster is formed by connecting a set of computer modules in non-hierarchical manner. Then, multiple clusters are interconnected through inter-cluster bus.

The internal structure of a computer module is shown in Figure 14.7. In addition to local memory and local I/O devices, it contains a channel and arbiter switch (CAS). The CAS provides an interface with other computer modules in the system. CAS can buffer the messages and resolves the conflicts accessing the message transfer system. When there are requests to access message transfer system from multiple computer modules, the CAS decides which particular computer module gets the service. Example of loosely coupled multiprocessor system is Cm*.

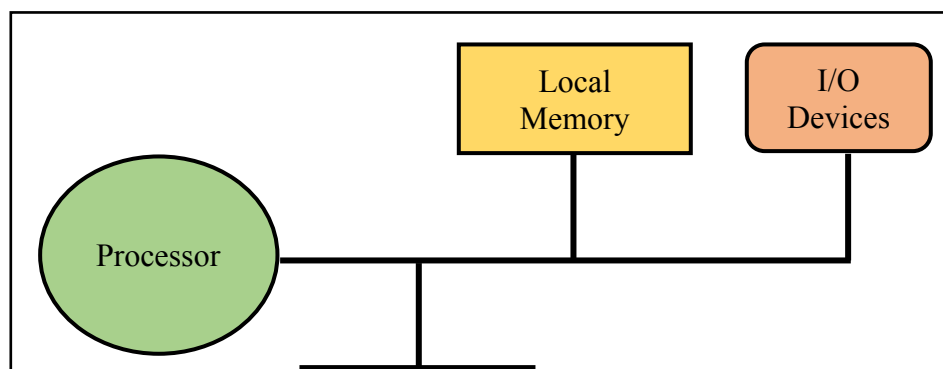


Figure 14.7: Computer Module

14.6 Multiprocessor Operating System

All those functionalities required for uniprocessor operating systems such as memory management, resource allocation, deadlock handling, and process management, etc. are also required for the multiprocessor operating systems. The underlying hardware in multiprocessors is different from a uniprocessor system. Therefore, some additional capabilities are needed in a multiprocessor operating system for efficient resource management, load balancing, scheduling, support for parallelism, synchronization, and reliability, etc. When a user program is executed, the operating system should be able to exploit parallel processing capabilities of the system. Without support for parallel processing and other additional capabilities, it would not be possible to take advantage of the hardware capabilities of the multiprocessor systems.

The multiple computer modules make the job of a multiprocessor operating system challenging. The performance of such operating system depends on its ability to communicate, load balancing, memory management, and resource management, etc. There are three main operating system designs for multiprocessors namely *master-slave*, *separate supervisor*, and *floating supervisor control*. The master-slave configuration is the simplest approach and it is a kind of simple extension of uniprocessor operating system. It is easy to implement but it is inefficient in resource utilization. Also, it does not have desirable control over the resources and events in the system. All the three operating system organizations are described here.

In *master-slave configuration* based approach, one of the processors is designated as master and rest other processors are considered as slaves. The operating system runs on the master processor only and slave processors are controlled by the operating system as other resources. The master is responsible to maintain the status of all the slave processors and work distribution. The slave processors send their request for any executive service to the master via a special instruction call. When master receives the request, it is acknowledged and appropriate action is taken. As the entire operating system runs on a single processor, the conflict problems are simplified. However, master-slave configuration is highly inflexible and susceptible to the failures. If master processor goes down or an unrecoverable error occurs, the entire system comes to a halt. The performance of the entire system highly depends on the master processor. If master processor is unable to quickly dispatch the process, the slave processors cannot be used to their potential. The utilization of the slave processors depends on the efficiency of the master processor. Therefore, it is desirable that master processor should have higher capabilities than slave processors. Master processor with higher capabilities could better control the slave processors. The master-slave configuration is preferred for those applications where work load is well defined and well distributed.

In *separate supervisor system*, each processor contains its own copy of the basic kernel, a set of private tables, and a set of private I/O devices. All processors are able to service their most of the needs with the help of their kernels and file system. For resource sharing and communication purpose, a set of global tables and shared file structure are used. The global tables and files are shared and accessed by all the processors. Therefore, a mechanism is required to control the access to the shared resources. The separate supervisor system is not as vulnerable to the system failure as master-slave configuration. However, to keep a copy of kernel at each processor takes a lot of memory. Also, reconfiguration of I/O devices is difficult in this type of system.

The third configuration of the multiprocessor operating system is the floating supervisor control. It is the most flexible approach that considers the processors as a set of resources. The kernel floats from one processor to other. This approach is better for load balancing and makes efficient use of resources. Like other multiprocessor

operating system approaches, the floating supervisor has to deal with the conflicts due to simultaneous resource access requests from multiple processors. The conflicts are supposed to be handled protecting the system integrity. None of the available multiprocessor operating systems is exclusively based on any of the three configuration. All such operating systems are make use some or more features of all the three configurations.

Check your progress 2

1. When do you prefer the tightly coupled multiprocessor systems?
2. What are the advantages of having a local memory?
3. How are the processors connected to share memory modules in tightly coupled multiprocessors?
4. How can the memory conflicts be reduced?
5. Give some examples of tightly coupled multiprocessors.
6. What is the advantage of having a larger local memory in loosely coupled multiprocessor?
7. What is advantage of loosely coupled multiprocessor?
8. How do processors communicate in loosely coupled multiprocessors?
9. What is the role of CAS in a computer module?
10. Give an example of loosely coupled multiprocessor.
11. What are the additional functionalities required for a multiprocessor operating system as compared to the uniprocessor operating system?
12. What are challenges for a multiprocessor operating system?
13. Write the major operating designs for multiprocessors?
14. What are the responsibilities of master in master-slave operating system?
15. What are the limitations of master slave, separate supervisor, and floating supervisor designs?
16. Which of the three operating system designs is better for load balancing?
17. Master slave approach is preferred for which type of applications?

14.7 Summary

Parallel processing is an efficient form of information processing, which emphasizes the exploitation of concurrent events in the computing process. Concurrency implies parallelism, simultaneity, and pipelining. Parallel processing needs collective efforts in the field of hardware, software, algorithms, and languages. The role of hardware in parallelism increases from program level to intra-instruction level. It is important to have a trade-off between hardware and software. The uniprocessor systems have their limitations limitation to achieve high degree of parallelism. The parallel computer architecture can be divided into three main categories depending on their features. Three main categories of paralleling processing approaches are as follows: Pipelined systems, Array processors, and Multiprocessor systems. The pipelining is a technique that executes program instructions/operations in an overlapped manner to achieve parallelism. Array processors consist of multiple ALUs that can operate in parallel. These ALUs are known as processing elements. The pipelines processors achieve temporal parallelism. With the help of multiple processing elements, the array processor achieve spatial parallelism. The processing elements are able to perform the same function at the same time. Therefore, array processors are suitable to carry out vector operations in parallel. The computing power can be increased by uniprocessor architecture to multiprocessor architecture. A computer system with multiple processors and shared memory space and peripherals is known as multiprocessor system. The entire system works under the control of a single operating system. The operating system is responsible for interaction between processors besides other resource management tasks. The processors communicate and cooperate at job, task, or data levels to solve a problem.

At architectural level, the multiprocessors can be divided into two major categories: loosely-coupled and tightly-coupled multiprocessors. In a tightly-coupled multiprocessor system, the processors communicate with the help of a shared main memory. The processors in tightly coupled multiprocessor systems have a local memory and a shared memory that is used for communication and exchange of information. The shared memory is a fast memory such as cache to deal with the

conflicts. If high degree of interaction is needed in an application, the tightly coupled multiprocessor systems are a good choice.

In loosely coupled multiprocessor systems, the processors do not communicate through memory instead a message transfer system is used to exchange messages for information sharing. Each processor has a large local memory, a local set of peripheral devices, and CAS. The processors access most of the instructions and data in their own local memories. Thus loosely coupled multiprocessor systems do not experience that kind of memory conflicts as observed in tightly coupled multiprocessor systems. The CAS provides an interface with other computer modules in the system. CAS can buffer the messages and resolves the conflicts accessing the message transfer system.

The underlying hardware in multiprocessors is different from a uniprocessor system. Therefore, some additional capabilities are needed in a multiprocessor operating system for efficient resource management, load balancing, scheduling, support for parallelism, synchronization, and reliability, etc. There are three main operating system designs for multiprocessors namely *master-slave*, *separate supervisor*, and *floating supervisor control*. The master-slave configuration is the simplest approach and it is a kind of simple extension of uniprocessor operating system. It is easy to implement but it is inefficient in resource utilization. Also, it does not have desirable control over the resources and events in the system. In *separate supervisor system*, each processor contains its own copy of the basic kernel, a set of private tables, and a set of private I/O devices. All processors are able to service their most of the needs with the help of their kernels and file system. For resource sharing and communication purpose, a set of global tables and shared file structure are used. The floating supervisor control is the most flexible approach that considers the processors as a set of resources. The kernel floats from one processor to other. This approach is better for load balancing and makes efficient use of resources.

Review Questions

Q.1 What is parallel processing? How can the parallel processing be achieved in a computer system?

- Q.2 Describe the different categories of computers according to Flynn's classification. Draw the figures to illustrate the conceptual architecture of all the categories and highlight the important features.
- Q.3 Compare the pipelined computers, array processors, and multiprocessor systems from architectural and parallel processing point of view. Discuss their applications.
- Q.4 What is multiprocessor system? How is it different from a multicomputer system? Describe and contrast the different architectural configuration for multiprocessor systems.
- Q.5 With the help of a sketch, describe the architecture of a tightly coupled multiprocessor system. Explain the process of communication and information sharing in the system.
- Q.6 With the help of a sketch, describe the architecture of a loosely coupled multiprocessor system. Explain the process of communication and information sharing in the system.
- Q.7 What are similarities and differences between a uniprocessor and a multiprocessor operating systems?
- Q.8 Compare the features of different configurations for a multiprocessor operating system.