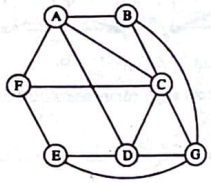
Ques 17 b) Let G be a connected graph and e an edge of G. Show that e is a cut-edge if and only if e belongs to every spanning tree. (05)

Or

Ques 18 a) State Kuratowski's theorem, and use it to show that the graph G below is not planar. Draw G on the plane without edges crossing. Your drawing should use the labelling of the vertices given. (09)



Ques 18 b) Let G be a connected graph and e an edge of G. Show that e belongs to a loop if and only if e belongs to no spanning tree. (05)

Ques 19 a) Define the circuit matrix B(G) of a connected graph G with n vertices and e edges with an example. Prove that the rank of B(G) is e – n+1. (07)

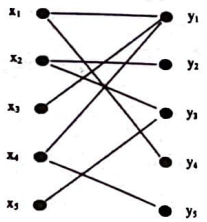
Ques 19 b) Give the definition of the chromatic polynomial $P_G(k)$. Directly from the definition, prove that the chromatic polynomials of W_n and C_n satisfy the identity $P_{W_n}(k) = k\, P_{C_{n-1}}(k-1)$. (07)

Or

Ques 20 a) Define the incidence matrix of a graph G with an example. Prove that the rank of an incidence matrix of a connected graph with n vertices is n–1. (04)

Ques 20 b) i) A graph G has chromatic polynomial $P_G(k) = k^4-4k^3+5k^2-2k$. How many vertices and edges does G have? Is G bipartite? Justify your answers. (10)

Ques 20 b) ii) Find a maximum matching in the graph below and use Hall's theorem to show that it is indeed maximum.



COMPUTER ORGANIZATION AND ARCHITECTURE

TP SOLVED SERIES

For

B.Tech Fourth Semester Students

(Computer Science and Engineering)

of

'APJ Abdul Kalam Technological University (KTU), Kerala'

Edition 2022

Books are Available for Online Purchase at: **tppl.org.in**

Download old Question papers from: **www.questionpaper.org.in**

THAKUR PUBLICATION PVT. LTD., ERNAKULAM

House No. 46/1309, Kattikaran House, Feroz Gandhi Lane, Vaduthala (Post), Ernakulam-682023. Ph. **9207296272, 9207296273, 09235318597**
For Supply: **9207296271, 09389557482**

Syllabus

CST-202: COMPUTER ORGANIZATION AND ARCHITECTURE

Module-1

Basic Structure of Computers – Functional units - basic operational concepts - bus structures. Memory locations and addresses - memory operations, Instructions and instruction sequencing, addressing modes.

Basic Processing Unit – Fundamental concepts – instruction cycle – execution of a complete instruction - single bus and multiple bus organization

Module-2

Register transfer Logic: Inter register transfer – arithmetic, logic and shift micro operations.

Processor Logic Design: Processor organization – Arithmetic logic unit - design of arithmetic circuit - design of logic circuit - Design of arithmetic logic unit - status register – design of shifter - processor unit – design of accumulator.

Module-3

Arithmetic Algorithms: Algorithms for multiplication and division (restoring method) of binary numbers. Array multiplier, Booth's multiplication algorithm.

Pipelining: Basic principles, classification of pipeline processors, instruction and arithmetic pipelines (Design examples not required), hazard detection and resolution.

Module-4

Control Logic Design: Control organization – Hard_wired control-microprogram control – control of processor unit - Microprogram sequencer,micro programmed CPU organization - horizontal and vertical micro instructions.

Module-5

I/O Organization: Accessing of I/O devices – interrupts, interrupt hardware -Direct memory access.

Memory System: Basic concepts – semiconductor RAMs. memory system considerations – ROMs, Content addressable memory, cache memories - mapping functions.

Printed at:
Savera Printing Press
Tirupatipuram, Jankipuram Extension Near AKTU, Lucknow-226031
E-mail: lkospp@gmail.com, Mobile No. 9235318506/07

Module 1

Basic Structure of Computers

BASIC STRUCTURE OF COMPUTERS

Ques 1) Define computer.

Ans: Computer
The term 'computer' is derived from the word 'compute' which means 'to calculate'.

Computer is an electronic device that receives data in a specific form, performs a series of operations according to the predetermined set of instructions and gives a result.

Computer operates upon received data. This data may arrive in various forms which in turn depend upon the computer application type. **For example,** receiving biodata of various candidates at the time of recruitment, marks gained by several students at the time of preparing results, getting details (like name, age, sex, etc.) of several passengers at the time of railway or airline reservations, etc. All these are various forms of data.

Ques 2) Draw the basic structure of the computer. Explain its various functional units.
Or
With a neat diagram, explain the internal architecture of the CPU. **(2018 [03])**

Ans: Basic Structure of Computers
A digital computer is a stored program electronic computer with internal memory that depends on software for system management and the automatic execution of stored programs. The simplified block diagram of computer is shown in **figure 1.1.**

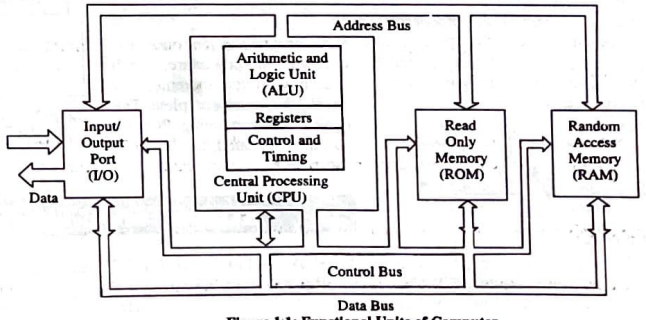


Figure 1.1: Functional Units of Computer

Functional Units of Computer
Main functional units of computers are:

1) **Central Processing Unit:** Central Processing Unit (CPU) is the brain of a computer. It's a part of the computer that performs the bulk of data processing. Its primary function is to execute programs. Program, which is to be executed, is stored in the main memory. **CPU also called Processor.**

   This consists of the following units:
   i) **Arithmetic and Logic Unit (ALU):** All arithmetic and logical operations are performed by this unit.
   ii) **Control Unit (CU):** Controlling and monitoring of all the components of computer system is carried out by this unit.
   iii) **Registers:** A CPU contains a number of registers to store data temporarily during the execution of a program. The number of registers differs from processor to processor.
2) **Memory Unit:** Memory unit is used to store the instructions and data received from the input unit. It stores the results of arithmetic operations conveyed to it by the arithmetic unit. Memory unit is used to store the instructions and data received from the input unit. It stores the results of arithmetic operations conveyed to it by the arithmetic unit.
3) **Input Unit:** This unit acts like an interface between the user and the computer system. The user enters the program and the data, interacts with the computer system during execution and issues commands to the

system through this unit. In general, this unit helps the user to communicate with the system. Common input units that are used in the market are like **keyboard, mouse, scanners, touch screen, light pen,** etc.

4) **Output Unit:** Output unit interlinks the computer system with the external environment. The output unit is used to deliver the result of processed data to the user or another device. It converts the processed data into human readable format. Some of the commonly used output devices are **monitor, printer,** etc.

5) **Buses:** Various input/output devices and memory devices are connected to a CPU by groups of lines called buses.

Ques 3) Give the basic operational concepts of the computer.

Ans: Basic Operational Concept of Computer

There are two duties of computer which are as follows:

1) Execute the software by fetching instructions from memory
2) Look for any external signal and react accordingly

Both these duties are entrusted upon and carried out by the microprocessor (or processor). The major duty of the processor is to run (or execute) the software and this program execution is done continuously until the computer is switched off.

The steps of the executions are as follows:

1) **Processor Clock:** Clock is the heart of any processor and it starts almost the moment a computer is
- powered. This clock is a simple digital signal producing *on* and *off* states alternately, at equal time intervals as shown in **figure 1.2**.

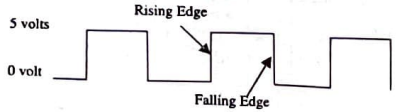


Figure 1.2: Processor Clock

2) **Program Counter:** Inside every processor, there is a binary counter, known as Program Counter(PC). The width (number of bits accommodated) of this counter varies from processor to processor. However its basic operation is identical in all cases. With every falling edge of the clock signal, this counter is incremented by one. The content of the PC, the group of bits, is immediately sent out to the memory area (main memory). This bunch of signals, which moves out from the PC to the memory is known as *address signal*. After reaching the memory section, depending upon the pattern of the signal generated by combinations of 1s and 0s of different bits, it targets one unique memory location. The content of that memory location is then brought within the processor. This process is known as *memory reading*.

3) **Instruction Fetch:** If we now assume that the original content of the PC was that address of memory, which contains the executable program, then the memory reading by the process must be to fetch an instruction of the program, and the procedure is known as *instruction fetch*. After completion of this instruction fetch, the binary content of the memory location (instruction byte) is available within the processor itself.

4) **Instruction Decodc:** Now the processor becomes busy with the instruction byte, freshly fetched from the memory, to understand what it is all about. After all, for every instruction, the processor must do something special as per that instruction. This concept of 'understanding the instruction' is also known as decoding the instruction or instruction decode. The more the total number of instructions offered by the processor, the more complex would be this instruction decoding unit.

5) **Instruction Execute:** From the instruction decoding module, after the processor knows what to do, it immediately starts to carry out the order. This execution may be of various types, depending upon the instruction and the processor, like:
   i) Fetch an operand from memory or
   ii) Add two registers

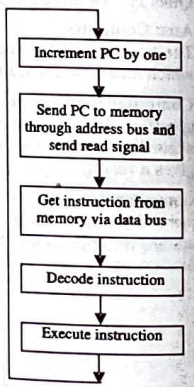


Figure 1.3: Major Functions of a Processor

Whatever be the job, once it is executed, the procedure, which started with the increment of the PC, becomes complete. The cycle of incrementing PC and so on, is repeated, as shown in **figure 1.3**.

BUS STRUCTURE

Ques 4) What is the bus? What is the function and configuration of bus?

Or

Discuss the address, data and control bus?

Or

Draw the diagram of a multi-bus organisation with 3 buses. Write the control sequence for the instruction Add R4, R5, R6 for the above mentioned multi-bus organisation. **(2019 [05])**

Ans: Bus

A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register the bus selects during each particular register transfer. A bus is a communication pathway connecting two or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus. Buses are the information highway for the CPU. Early personal computers had a single external bus or

system bus. It consisted of 50 to 100 parallel copper wires attached onto the motherboard, with connectors spaced at regular intervals for plugging in memory and I/O boards. Modem personal computers generally have a special-purpose bus between the CPU and memory and (atleast) one other bus for the I/O devices.

Functions of Buses

1) **Data Sharing:** The data bus must be able to transfer data between the computer components. The data is travel in a parallel stream or serial stream as the case may be.
2) **Addressing:** An address bus has address lines which carry addresses of memory locations.
3) **Power:** A power bus supplies power to various peripherals that are connected to it.
4) **Timing:** The control bus provides a system clock signal to synchronize the peripherals of various components.

Buses Configuration

There are three types of buses as shown in **figure 1.4**. As **figure 1.4** shows, many computer systems contain more than one bus. The configuration shows a computer system with two large buses, each composed of a control, address, and data bus. These buses are as follows:

1) **Address Bus:** It is controlled by CPUs is for specifying the physical addresses of computer memory elements that the requesting unit wants to access (read or write). The address bus is faster than the system or memory bus, enabling it to transfer an address in the same amount of time as an address bus of the same width as the address.

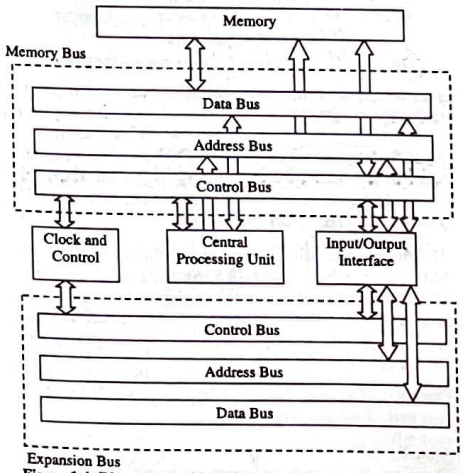


Figure 1.4: Diagram of a Simplified Computer System containing Two Buses, One for Memory and One for the Remaining Devices. Each Bus is Broken-Down into Three Sub-Sections, a Control Bus, an Address Bus, and a Data Bus

2) **Control Bus:** It is used by processor for communicating with other devices within the computer. The control bus carries commands from the CPU and returns status signals from the devices.

3) **Data Bus:** used by the CPU this carries the actual data being processed.

Table 1.1: Control Sequence for the Instruction Add R4, R5, R6, for the Three-Bus Organisation in Figure 1.5

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, Select A, Add, $R6_{in}$, End

Ques 5) What are the various types of bus structure? Explain in detail.

Or

Explain single and multiple bus strucutre in detail.

Or

In what categories, bus can be divided? Explain.

Ans: Bus Structure

Basically there are two types of bus structure:

1) **Single Bus Strucutre:** All units are connected to a single bus, so it provides the sole means of interconnection.

The **advantage** of single bus structure is its simplicity and low cost. Single bus structure has **disadvantages** of limited speed since usually only two units can participate in a data transfer at any one time. This means that an arbitration system is required and that units will be forced to wait. Only two units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for the use of the bus. **Figure 1.5(a)** shows single bus structure.

2) **Two Bus Structure: Figure 1.5** shows the two bus structure:

In the **first configuration (figure 1.5 (a)),** the processor is placed between the I/O unit and the memory unit. The processor is responsible for any data transfer between the I/O unit and the memory unit. The processor acts as a "messenger." In this structure, the processor performance and capability is not being maximised.

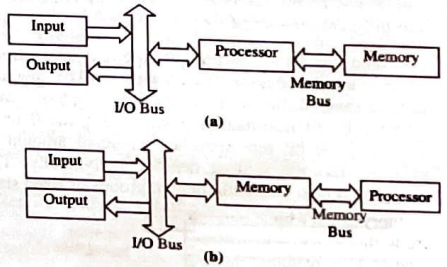


Figure 1.5: Two Bus Organization

Most of the time, the processor is doing data transfer between these units instead of performing more complex applications. Also, the processor is idle most of the time waiting for these slow devices.

In the **second configuration (figure 1.5 (b)),** I/O transfers are made directly to or from the memory. A special purpose processor called peripheral processor

or I/O channel is needed as part of the I/O equipment to control and facilitate such transfers. This special processor is the direct memory access (DMA) controller. It allows main memory to perform data transfer between I/O units.

Ques 6) Give the sequence of control steps required to perform the operation Add [R3], R1 in a single-bus organisation. **(2019 [04])**

Ans: The Control Sequence is shown in **table below:**

Table 1.2: Control Sequence for Execution of the Instruction Add (R3), R1

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

MEMORY LOCATIONS AND ADDRESSES

Ques 7) Briefly explain the concepts of memory locations and addresses with example.

Or

What is address space?

Ans: Memory Locations and Addresses

Memory address is a data concept used at various levels by software and hardware to access the computer's primary storage memory. Memory addresses are fixed-length sequences of digits conventionally displayed and manipulated as unsigned integers. Such numerical semantic bases itself upon features of CPU (such as the instruction pointer and incremental address registers), as well upon use of the memory like an array endorsed by various programming languages.

Number and character operands, as well as instructions, are stored in the memory of a computer. The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1. Because a single bit represents a very small amount of information, bits are seldom handled individually. The usual approach is to deal with them in groups of fixed size.

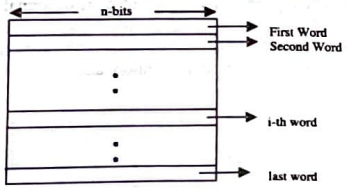


Figure 1.6: Memory Words

For this purpose, the memory is organised so that a group of n bits can be stored or retrieved in a single, basic operation. Each group of n bits is referred to as a word of information, and n is called the **word length.** The memory of a computer can be schematically represented as a collection of words as shown in **figure 1.6.** Modern computers have word lengths that typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit 2's-complement number or four ASCII characters, each occupying 8 bits, as shown in **figure 1.6.**

A unit of 8 bits is called a **byte.** Machine instructions may require one or more words for their representation. Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each item **location.** It is customary to use numbers from 0 through $2^k - 1$, for some suitable value of k, as the addresses of successive locations in the memory. The 2^k addresses constitute the **address space** of the computer, and the memory can have up to 2^k addressable locations.

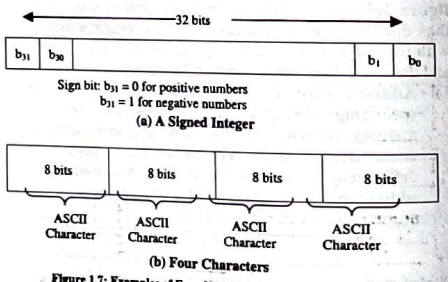


Figure 1.7: Examples of Encoded Information in a 32-bit Word

To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address. The total number of uniquely identifiable locations in memory is called the **address space. For example,** a 24-bit address generates an address space of 2^{24} (16,777,216) locations.

This number is usually written as 16M (16 mega), where 1M is the number 2^{20} (1,048,576). A 32-bit address creates an address space of 2^{32} or 4G (4 giga) locations, where 1G is 230. Other notational conventions that are commonly used are K (kilo) for the number 2^{10} (1,024), and T (tera) for the number 2^{40}.

Ques 8) Explain Byte addressability in big-Endian and little Endian assignment.

Ans: Byte Addressability

There are three basic information quantities to deal with: the bit, byte, and word. A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to successive byte locations in the memory. This is the assignment used in most modern computers.

The term byte-addressable memory is used for this assignment. Byte locations have addresses 0, 1, 2,.... Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8, ..., with each word consisting of four bytes. A byte of memory can be addressed during a read as shown in **figure 1.8:**

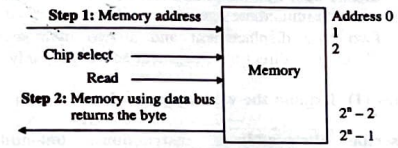


Figure 1.8: Fetching from a Memory Location at Address between $2^n - 1$

Assignment as Little-endian and Word Alignment at Memory

When a processor accesses a word (for instruction or data) from memory with word assignment in a **little-endian system,** the Least Significant (smallest value) Byte (LSB) of a word is written into the lowest-address, and the other bytes are written in increasing order of significance; order is LSB, LSB + 1, MSB – 1, and MSB. A word in the memory aligned as little endian can be addressed as shown in **figure 1.9.**

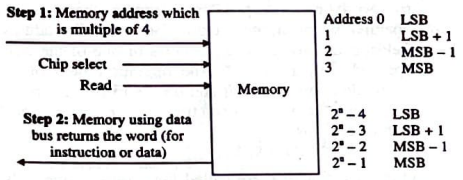


Figure 1.9: Fetching a 32-Bit Word in Little-Endian Example

Assignment as Big-endian and Word Alignment in Memory

A word in memory aligned as **big endian** can be addressed as shown in **figure 1.10:**

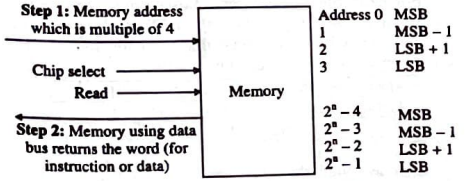


Figure 1.10: Big-Endian Word of 32-Bit at Memory Locations in Multiple of 4

For a processor accessing memory with word assignment (for instruction or data) in a big-endian system of word alignment at memory, the byte order is reversed with respect to little endian, with the Most Significant Byte (MSB) being written into the byte of memory with the lowest address.

[The other bytes are written in decreasing order of significance; order is MSB, MSB – 1, LSB + 1, and LSB.]

MEMORY OPERATIONS

Ques 9) What are the various memory operations, explain? Write the steps of read and typical write cycle.

Ans: Memory Operations

Memory operations are employed to transfer data between the processor and the memory system. Each of these operation functions on an amount of data equal to the word size of the machine.

Random access memories must have two basic operations:

1) **Write:** Writes a data into the specified location.
2) **Read:** Reads the data stored in the specified location.

In machine language program, the two basic operations usually are called:

1) **Store:** Write operation. The Store operation transfers an item of information from the processor to a specific memory location, destroying the former contents of that location.
2) **Load:** Read operation. The Load operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged.

Two metrics are used to characterise memory:

1) **Access Time:** Access time refers to the amount of time required by the memory to retrieve the data at the addressed location.
2) **Cycle Time:** This refers to the minimum time between successive memory operations.

Memory transfer rates can be measured by the bandwidth metric. It specifies the number of bytes transferred per second. **For example,** a Pentium system with the PC133 memory can transfer 8 bytes at a frequency of 133 times per second. This gives us a bandwidth of 8 × 133 = 1064 MB/s.

The read operation is non-destructive in the sense that one can read a location of the memory as many times as one wish without destroying the contents of that location. The write operation, on the other hand, is destructive, as writing a value into a location destroys the old contents of that memory location.

Steps in a Typical Read Cycle

Step 1: Place the address of the location to be read on the address bus.

Step 2: Activate the memory read control signal on the control bus.

Step 3: Wait for the memory to retrieve the data from the addressed memory location and place it on the data bus.

Step 4: Read the data from the data bus.

Step 5: Drop the memory read control signal to terminate the read cycle.

Steps in a Typical Write Cycle

Step 1: Place the address of the location to be written on the address bus.

Step 2: Place the data to be written on the data bus.

Step 3: Activate the memory write control signal on the control bus.

Step 4: Wait for the memory to store the data at the addressed location.

Step 5: Drop the memory write signal to terminate the write cycle.

INSTRUCTIONS & INSTRUCTION SEQUENCING

Ques 10) Define instruction. What are the elements of instruction format? Also give the length of instruction format.

Ans: Instruction

An instruction set is a collection of all the instructions a CPU can execute. Each instruction consists of several elements. An instruction element is a unit of information required by the CPU for execution. **An instruction format** defines the layout of bits of an instruction in terms of its constituent parts that the bits of an instruction can be grouped into parts called fields.

A computer can contain multiple instruction formats. It is the duty of the control unit of the CPU to determine the instruction format and send appropriate signals to various parts of the CPU.

Elements of an Instruction Format

1) **Operation Code:** An operation code also termed as opcode which specifies the operation to be performed.
2) **Reference to Opcode:** A reference to the operands on which data processing is to be performed. **For example**, an address of an operand.
3) **Reference to Operands which may Store Results of Data:** A reference to the operands which may store the results of data processing operation performed by the instruction.
4) **Reference for Next Instruction:** A reference for the next instruction, to be fetched and executed.

Length of Instruction Format

Instructions of 8086 vary from 1 to 6 bytes in length. **Figure 1.11** shows the instruction formats for 1 to 6 bytes instructions.

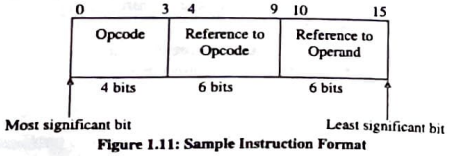


Figure 1.11: Sample Instruction Format

As shown in the **figure 1.11**, displacements and operands may be either 8-bits or 16-bits long depending on the instruction. The opcode and the addressing mode are specified using first two bytes of an instruction.

The opcode/addressing mode byte(s) may be followed by:

1) No additional byte.
2) Two byte EA (for direct addressing only).
3) One or two byte displacement.
4) One or two byte immediate operand.
5) One or two byte displacement followed by a one or two byte immediate operand.
6) Two byte displacement and a two byte segment address (for direct inter-segment addressing only).

Ques 11) Explain the various types of instruction.

Or

Describe three-address instructions, one-address instruction, two-address instructions and zero-address instruction.

Or

Explain one, two and three address instruction with an example for each. **(2019 [03])**

Ans: Types of Instruction

Instructions are of different types depending on the number of operands they contain or require, to operate on.

1) **Three-Address Instructions:** Machines with three-address instruction formats can use each address field to specify either a processor register or a memory operand. Such an instruction contains three address fields, each providing the address of one of the three operands- X, Y, and Z. The operation field of the instruction specifies that addition is to be performed. This three address instruction can be represented symbolically as:

ADD X, Y, Z

Operands X and Y are called the "source" operands, and C is called the "destination" operand. The general form of three-address instruction is as below:

Operation-Code	Source 1,	Source 2,	Destination

For example, the program in assembly language that evaluates X = (A + B) * (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

ADD	R1, A, B	R1 ← M [A] + M [B]
ADD	R2, C, D	R2 ← M [C] + M [D]
MUL	X, R1, R2	M[X] ← R1 * R2

It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolised by A.

2) **Two-Address Instructions:** Two-address instructions are the most common in commercial machines. Here again each address field can specify either a processor register or a memory word.

The general form of two-address instruction is as below:

Operation-Code	Source,	Destination

An ADD instruction of this type is:

ADD A, B

which performs the operation B← [A] + [B]. When the sum is calculated, the result is sent to the main memory and stored in location B, replacing the original contents of this location. This means that operand B is both a source and a destination. **For example,** the program to evaluate X = (A + B) * (C + D) is as follows:

MOV	R1, A	R1 ← M[A]
ADD	R1, N	R1 ← R1 + M[B]
MOV	R2, C	R2 ← M[C]
ADD	R2, D	R2 ← R2 + M[D]
MUL	R1, R2	R1 ← R1 * R2
MOV	X, R1	M[X] ← R1

MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

3) **One-Address Instructions:** One-address machines use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contain the result of all operations.

The general form of one-address instruction is as below:

Operation-Code	Source

For example, the program to evaluate X = (A + B) * (C + D) is:

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

All operations are done between the AC(accumulator) register and a memory operand. T is the address of temporary memory location for storing the intermediate result.

4) **Zero-Address Instructions:** Zero-address instructions are those in which locations of all operands are defined implicitly. The execution of these instructions is the fastest as the speed with which a given task is carried out depends on the time it takes to transfer instructions from memory into the processor, and to access the operand. These instructions are generally used for stacks. A stack-organised instruction does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

The general form of one-address instruction is as below:

Operation-Code

For example, the following program shows how X = (A + B) * (C + D) will be written for a stack organised computer. (TOS stands for top of stack).

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP	X	M[X] ← TOS

Ques 12) Represent zero, one, two and three address instructions formats for the evaluation of the expression X = A * B + C * C.

Ans: The expression can be evaluated by following address formats as **shown below:**

1) One-Address Format

X = A * B + C * C	
Instruction	**Comments**
LOAD A	Transfer A to accumulator AC
MULTIPLY B	AC ← AC * B
STORE T	Transfer AC to memory location T
LOAD C	Transfer C to accumulator AC
MULTIPLY C	AC ← AC * C
ADD T	AC ← AC + T
STORE X	Transfer result to memory location X

2) Two-Address Format

X = A * B + C * C	
Instruction	**Comments**
MOVE A, T	T ← A
MULTIPLY B,T	T ← T * B
MOVE C, X	X ← C
MULTIPLY C, X	X ← X * C
ADD T, X	X ← X + T

3) Three-Address Format

X = A * B + C * C	
Instruction	**Comments**
MULTIPLY A, B ,T	T ← A * B
MULTIPLY C, C, X	X ← C * C
ADD X, T, X	X ← X * T

4) Zero-Address Format

X = A * B + C * C	
Instruction	**Comments**
PUSH A	Transfer A to top of stack
PUSH B	Transfer B to top of stack
MULTIPLY	Remove A, B from stack and replace by A * B
PUSH C	Transfer C to top of stack
PUSH C	Transfer second copy of C to top of stack
MULTIPLY	Remove C, C from stack and replace by C * C
ADD	Remove C*C, A*B from stack, replace by their sum
POP X	Transfer result from top of stack to X

Ques 13) Describe how an instruction can be executed?

Or

What is straight line sequencing?

Ans: Instruction Execution and Straight Line Sequencing

The processor control circuits use information in PC (program counter) to fetch and execute instructions one at a time in order of increasing address. This is called **straight line sequencing.**

In this execution of an instruction done in two phase procedures:

1) **First Phase (Instruction Fetch):** Instruction is fetched from memory location whose address is in PC. This instruction is placed in instruction register in processor.
2) **Second Phase (Instruction Execute):** Instruction in IR is examined to determine which operation to be performed.

Let us take the operation c ← [A] + [B]. **Figure 1.12** shows the program segment as it appears in the main memory of a computer that has a two-address instruction format and a number of general purposes CPU registers. The program is:

```
Move   A, R0
Add    B, R0
Move   R0, C
```

These three instructions are placed in successive memory locations starting at location i as shown

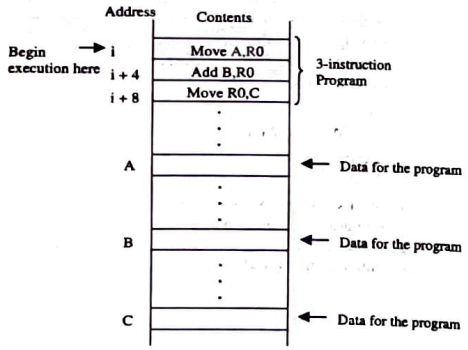


Figure 1.12: Program for C ← [A] + [B]

For executing this program, the following steps are to be performed:

1) CPU contains the register called PC which holds the address of the instruction to be executed next. To begin execution, the address of the first instruction 'i' must be placed in PC.
2) CPU control circuits use the information in the PC to fetch and execute the instructions one at a time in the increasing order of addresses. This is called straight line sequencing.
3) As each instruction is executed, the PC is incremented by 4 to point to the next instruction.

Ques 14) Discuss branching with example.

Ans: Branching

Branch-type of instruction loads a new value into program counter. So processor fetches and executes instruction at this new address called "branch target". Conditional branch – causes a branch if a specified condition is satisfied. **For example,** Branch > 0 LOOP-conditional branch instruction. It executes only if it satisfies condition.

Consider the task of adding 'n' numbers. Let the address of memory locations containing n numbers are NUM1, NUM2,...NUMn. Separate Add instruction is used to add each number to the contents of register R0. After all the numbers have been added, the result is placed in the memory location SUM.

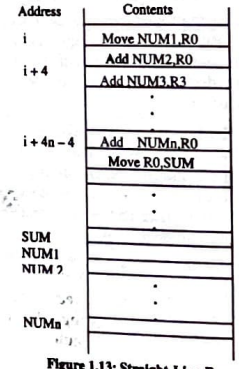


Figure 1.13: Straight-Line Program for Adding n Numbers

Instead of using long list of Add instruction, it is possible to place a single Add instruction in a loop as shown in **figure 1.14**:

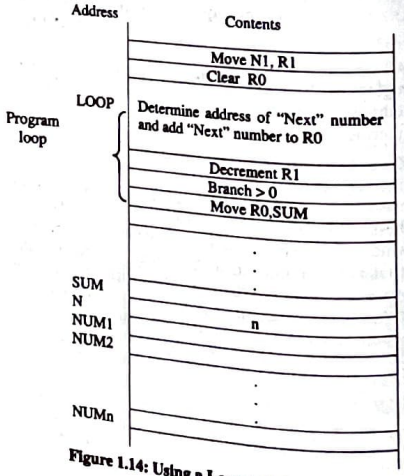


Figure 1.14: Using a Loop to Add n Numbers

Ques 15) Enumerate the sequence of actions involved in executing an unconditional branch instruction. **(2018 [03])**

Ans:

1) Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4.
2) Decode instruction.
3) PC ← [PC] + Branch offset.
4) No action.
5) No action.

Ques 16) Write down the sequence of actions needed to fetch and execute the instruction: Store R6, X (R8). **(2018 [03])**

Ans: Store R6, X (R8) stores the contents of register R6 into memory location X + (R8). It can be implemented as follows:

1) Fetch the instruction and increment the program counter.
2) Decode the instruction and read registers R6 and R8.
3) Compute the effective address X + (R8).
4) Store the contents of register R6 into memory location X + (R8).
5) No action.

ADDRESSING MODES

Ques 17) What is meant by addressing mode? What are the various types of addressing modes?

Or

Explain immediate addressing, direct addressing, indirect addressing, register addressing, displacement addressing, and auto increment/auto decrement addressing modes.

Or

Explain indirect addressing with an example. **(2017 [03])**

Or

With the help of examples, explain the different addressing modes. **(2018 [05])**

Or

List various addressing modes explain any four with an example for each. **(2019 [09])**

Ans: Addressing Mode

Addressing modes are an aspect of the instruction set architecture in most Central Processing Unit (CPU) designs. The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction. An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.

Addressing mode tells the computer where to get/place a number.

For executing an instruction, data is required. This data may be present in the accumulator (AC) or stored in some location in the memory. There are various ways to specify the address of data or more precisely operands. The techniques for specifying the address of the operands are known as addressing modes. The address of operand is known as effective address. **Effective Address** (EA) of an operand is the address of (or the pointer to) the MM or RF location in which the operand is contained: Operand = [EA].

Types of Addressing Modes

The different types of addressing modes are as follows:

1) **Immediate Addressing:** In this mode of addressing, data is present in the instruction (In this type of addressing mode the operand is specified within the instruction itself). Load the immediate data to the destination provided.

   This mode is identified by the letter 'I' in the instruction.

   Syntax:

   MVI R, data

   For example,

   MVI A, 32H

   Here 32H is the data. H represents Hexadecimal value and the immediate value is moved to the accumulator. In this case 32H is moved to the accumulator. In the above instruction, the operand is specified within instruction itself.

2) **Direct (or Absolute) Addressing:** In this mode of addressing, the address of the data (operand) is specified within the instruction. There is a very little difference between the direct addressing modes and immediate addressing modes. In immediate addressing mode, the data itself is specified within instruction, but in direct addressing mode the address of the data is specified in the instruction

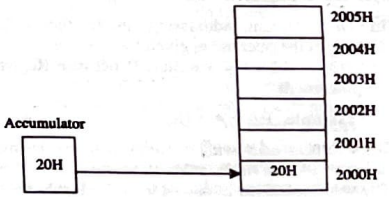
   Syntax:

   MOV R, Memory Location

   For example,

   STA 2000H

   When this instruction is executed, the contents of the accumulator are stored in the memory location specified. In the above **example** the contents of accumulator are stored in memory location 2000H.



3) **Indirect Addressing:** In this mode the instruction specifies the name of the register in which the address of the data is available.

   For example,

   MOV A, M

   This instruction will move the contents of memory location, whose address is in H-L register pair to the

Ques 13) Describe how an instruction can be executed?

Or

What is straight line sequencing?

Ans: Instruction Execution and Straight Line Sequencing

The processor control circuits use information in PC (program counter) to fetch and execute instructions one at a time in order of increasing address. This is called **straight line sequencing.**

In this execution of an instruction done in two phase procedures:

1) **First Phase (Instruction Fetch):** Instruction is fetched from memory location whose address is in PC. This instruction is placed in instruction register in processor.
2) **Second Phase (Instruction Execute):** Instruction in IR is examined to determine which operation to be performed.

Let us take the operation c ← [A] + [B]. **Figure 1.12** shows the program segment as it appears in the main memory of a computer that has a two-address instruction format and a number of general purposes CPU registers. The program is:

```
Move   A, R0
Add    B, R0
Move   R0, C
```

These three instructions are placed in successive memory locations starting at location i as shown

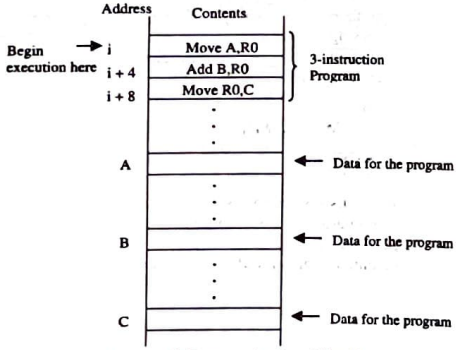


Figure 1.12: Program for C ← [A] + [B]

For executing this program, the following steps are to be performed:

1) CPU contains the register called PC which holds the address of the instruction to be executed next. To begin execution, the address of the first instruction 'i' must be placed in PC.
2) CPU control circuits use the information in the PC to fetch and execute the instructions one at a time in the increasing order of addresses. This is called straight line sequencing.
3) As each instruction is executed, the PC is incremented by 4 to point to the next instruction.

Ques 14) Discuss branching with example.

Ans: Branching

Branch-type of instruction loads a new value into program counter. So processor fetches and executes instruction at this new address called "branch target". Conditional branch – causes a branch if a specified condition is satisfied. **For example,** Branch > 0 LOOP-conditional branch instruction. It executes only if it satisfies condition.

Consider the task of adding 'n' numbers. Let the address of memory locations containing n numbers are NUM1, NUM2,...NUMn. Separate Add instruction is used to add each number to the contents of register R0. After all the numbers have been added, the result is placed in the memory location SUM.

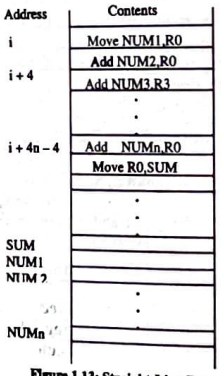


Figure 1.13: Straight-Line Program for Adding n Numbers

Instead of using long list of Add instruction, it is possible to place a single Add instruction in a loop as shown in **figure 1.14**:

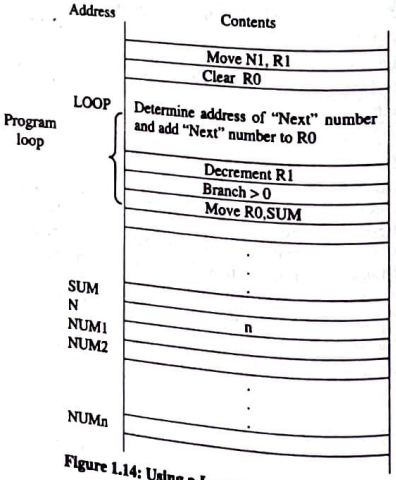


Figure 1.14: Using a Loop to Add n Numbers

Ques 15) Enumerate the sequence of actions involved in executing an unconditional branch instruction. **(2018 [03])**

Ans:

1) Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4.
2) Decode instruction.
3) PC ← [PC] + Branch offset.
4) No action.
5) No action.

Ques 16) Write down the sequence of actions needed to fetch and execute the instruction: Store R6, X (R8). **(2018 [03])**

Ans: Store R6, X (R8) stores the contents of register R6 into memory location X + (R8). It can be implemented as follows:

1) Fetch the instruction and increment the program counter.
2) Decode the instruction and read registers R6 and R8.
3) Compute the effective address X + (R8).
4) Store the contents of register R6 into memory location X + (R8).
5) No action.

ADDRESSING MODES

Ques 17) What is meant by addressing mode? What are the various types of addressing modes?

Or

Explain immediate addressing, direct addressing, indirect addressing, register addressing, displacement addressing, and auto increment/auto decrement addressing modes.

Or

Explain indirect addressing with an example. **(2017 [03])**

Or

With the help of examples, explain the different addressing modes. **(2018 [05])**

Or

List various addressing modes explain any four with an example for each. **(2019 [09])**

Ans: Addressing Mode

Addressing modes are an aspect of the instruction set architecture in most Central Processing Unit (CPU) designs. The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction. An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.

Addressing mode tells the computer where to get/place a number.

For executing an instruction, data is required. This data may be present in the accumulator (AC) or stored in some location in the memory. There are various ways to specify the address of data or more precisely operands. The techniques for specifying the address of the operands are known as addressing modes. The address of operand is known as effective address. **Effective Address** (EA) of an operand is the address of (or the pointer to) the MM or RF location in which the operand is contained: Operand = [EA].

Types of Addressing Modes

The different types of addressing modes are as follows:

1) **Immediate Addressing:** In this mode of addressing, data is present in the instruction (In this type of addressing mode the operand is specified within the instruction itself). Load the immediate data to the destination provided.

   This mode is identified by the letter 'I' in the instruction.

   Syntax:

   MVI R, data

   For example,

   MVI A, 32H

   Here 32H is the data. H represents Hexadecimal value and the immediate value is moved to the accumulator. In this case 32H is moved to the accumulator. In the above instruction, the operand is specified within instruction itself.

2) **Direct (or Absolute) Addressing:** In this mode of addressing, the address of the data (operand) is specified within the instruction. There is a very little difference between the direct addressing modes and immediate addressing modes. In immediate addressing mode, the data itself is specified within instruction, but in direct addressing mode the address of the data is specified in the instruction

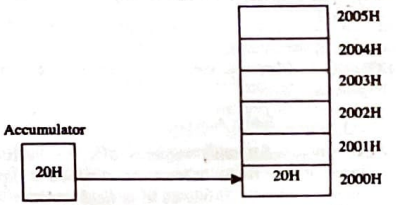
   Syntax:

   MOV R, Memory Location

   For example,

   STA 2000H

   When this instruction is executed, the contents of the accumulator are stored in the memory location specified. In the above **example** the contents of accumulator are stored in memory location 2000H.



3) **Indirect Addressing:** In this mode the instruction specifies the name of the register in which the address of the data is available.

   For example,

   MOV A, M

   This instruction will move the contents of memory location, whose address is in H-L register pair to the

accumulator. M represents the address present in the H-L register pair. So when MOV A, M is executed, the contents of the address specified in H-L register pair are moved to accumulator.

4) **Register Addressing:** In this type of addressing mode the instruction specifies the name of the register in which the data is available and Op-code specifies the name (or) address of the register on which the operation would be performed.

Syntax:

MOV Rd, Rs (Rd→destination register)
(Rs→source register)

For example, MOV A, B

Here the Opcode is MOV. If the above instruction is executed, the contents of register B are moved to the Register A, which is nothing but the accumulator as shown in **figure 1.15.**

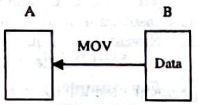


Figure 1.15

5) **Register Indirect Addressing:** Same as register addressing is analogous to direct addressing, register indirect addressing is analogous to indirect addressing. In both cases, the only difference is whether the address field refers to a memory location or a register.

For example, EA = (R)

The advantages and limitations of register indirect addressing are basically the same as for indirect addressing. In both cases, the address space limitation (limited range of addresses) of the address field is overcome by having that field refer to a word-length location containing an address. In addition, register indirect addressing uses one less memory reference than indirect addressing.

6) **Displacement Addressing:** A very powerful mode of addressing combines the capabilities of direct addressing and registers indirect addressing. It is known by a variety of names depending upon the context of its use, but the basic mechanism is the same. This is referred as displacement addressing.

In Displacement addressing mode the Effective address of the operand is given by

Effective Address = Implicit Processor Register + Displacement

For example, EA = A + (R)

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

7) **Auto Increment/Auto Decrement Addressing Mode:** It is generally used to increment or decrement the array pointer. For example, while executing a loop the processor may require to increment or decrement the pointer to the adjacent address at each iteration. So it can be used to increment or decrement file pointers, or it can be used to implement stack in which the top can be incremented (TOP++) or decremented (TOP– –). Also in Auto increment or Auto decrement addressing mode the contents of the register is automatically incremented or decremented before or after the execution of the instruction. **For example,** MOV (R2) +, R1

Ques 18) Register R6 is used in a program to point to the top of a stack containing 32-bit numbers. Write a sequence of instructions using the Index, Autoincrement, and Autodecrement addressing modes to perform each of the following tasks: (2018 [04])

a) Pop the top two items off the stack, add them, then push the result onto the stack.

b) Copy the fifth item from the top into register R_3.

For each case, assume that the stack contains ten or more elements.

Ans:

a)

i) Move-(R5, R1)
ii) Move-(R5), R2
iii) Add R3, (R1), (R2)
iv) Move R3, (R5) +

b)

i) Move-(R5), #10
ii) Store R3, (R5)

BASIC PROCESSING UNIT

Ques 19) What is processing unit?

Ans: Processing Unit

Processing unit executes machine instructions and coordinates the activities of other units. This unit is often called the Instruction Set Processor (ISP) or simply **processor.** This unit performs the tasks of fetching, decoding and executing instructions of a program. The processing unit used to be called Central Processing Unit (CPU).

A typical computing task consists of a series of steps specified by a sequence of machine instructions that constitute a program. An instruction is executed by carrying out a sequence of more rudimentary operations.

Ques 20) Discuss the fundamental concepts of operations of processing unit with suitable diagram.

Or

With the help of a diagram, describe the datapath inside the processor. (2018 [05])

Or

Illustrate the basic operational concepts in transferring data between main memory and processor with neat diagram. (2019 [05])

Ans: Fundamental Concepts of Operations of Processing Unit

To execute a program, the processor fetches one instruction at a time and performs the operations specified.

Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered. The processor keeps track of the address of the memory location containing the next instruction to be fetched using the Program Counter (PC). After fetching an instruction, the contents of the PC are updated to point to the next instruction in the sequence. A branch instruction may load a different value into the PC.

Another key register in the processor is the Instruction Register (IR). Suppose that each instruction comprises 4 bytes, and that it is stored in one memory word. To execute an instruction, the processor has to perform the following three steps:

Step 1) Fetch the contents of the memory location pointed to by the PC. The contents of this location are interpreted as an instruction to be executed. Hence, they are loaded into the IR. Symbolically, this can be written as:

IR ← [[PC]]

Step 2) Assuming that the memory is byte addressable, increment the contents of the PC by 4, i.e.,

PC ← [PC] + 4

Step 3) Carry out the actions specified by the instruction in the IR.

In cases where an instruction occupies more than one word, steps 1 and 2 must be repeated as many times as necessary to fetch the complete instruction. These two steps are usually referred to as the fetch phase; step 3 constitutes the execution phase.

To study these operations in detail, one first needs to examine the internal organisation of the processor. The main building blocks of a processor were introduced in **figure 1.16.**

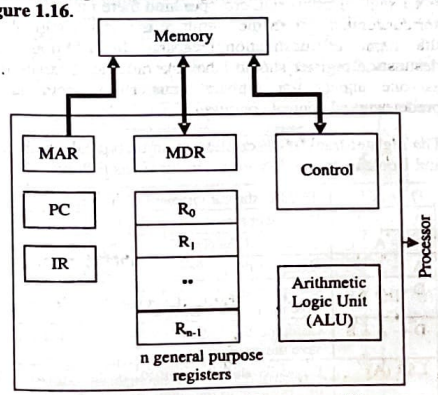


Figure 1.16: Connections between Processor and Memory

They can be organised and interconnected in a variety of ways. One will start with a very simple organisation. **Figure 1.17** shows an organisation in which the Arithmetic and Logic Unit (ALU) and all the registers are interconnected via a single common bus.

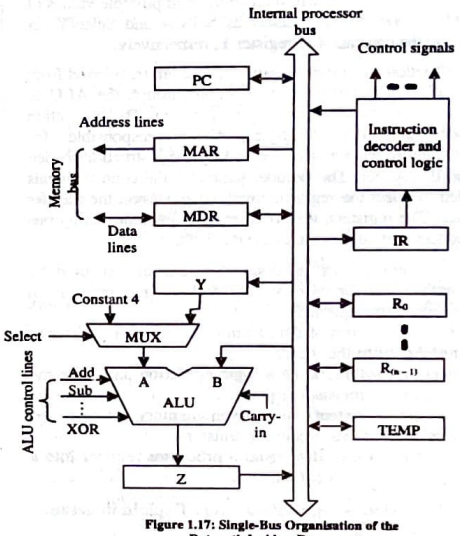


Figure 1.17: Single-Bus Organisation of the Datapath Inside a Processor

This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices. The data and address lines of the external memory bus are shown in **figure 1.17** connected to the internal processor bus via the memory data register, MDR, and the memory address register, MAR, respectively. Register MDR has two inputs and two outputs. Data may be loaded into MDR either from the memory bus or from the internal processor bus. The data stored in MDR may be placed on either bus. The input of MAR is connected to the internal bus, and its output is connected to the external bus. The control lines of the memory bus are connected to the instruction decoder and control logic block. This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus.

The number and use of the processor registers R_0 through $R_{(n-1)}$ vary considerably from one processor to another. Registers may be provided for general-purpose use by the programmer. Some may be dedicated as special-purpose registers, such as index registers or stack pointers. Three registers, Y, Z, and TEMP in **figure 1.17,** have not been mentioned before. These registers are transparent to the programmer, i.e., the programmer need not be concerned with them because they are never referenced explicitly by any instruction. They are used by the processor for temporary storage during execution of some instructions. These registers are never used for storing data generated by one instruction for later use by another instruction.

The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU. The constant 4 is used to increment the contents of the

program counter. One will refer to the two possible values of the MUX control input Select as Select4 and SelectY for selecting the constant 4 or register Y, respectively.

As instruction execution progresses, data are transferred from one register to another, often passing through the ALU to perform some arithmetic or logic operation. The instruction decoder and control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register. The decoder generates the control signals needed to select the registers involved and direct the transfer of data. The registers, the ALU, and the interconnecting bus are collectively referred to as the datapath.

With few exceptions, an instruction can be executed by performing one or more of the following operations in some specified sequence:

1) Transfer a word of data from one processor register to another or to the ALU.
2) Perform arithmetic or a logic operation and store the result in a processor register.
3) Fetch the contents of a given memory location and load them into a processor register.
4) Store a word of data from a processor register into a given memory location.

Ques 21) What is register transfer? Explain in detail.

Ans: Registers Transfers

The term "register transfer" implies the availability of hardware logic circuits that can perform a stated micro-operation and transfer the result of the operation to the same or another register.

For performing any arithmetical or logical operation, an ALU (Arithmetic Logic Unit) requires certain temporary memory locations where the data may be kept for a short while. Such memory locations built in the ALU, are called its registers. **For example,** Intel 8085 microprocessor has five types of registers, namely:

1) General purpose registers
2) Accumulator
3) Flag registers
4) Program counter
5) Stack pointer.

Computer registers are designated by capital letters to denote the function of the register. **For example**, the register that holds an address for the memory unit is usually called a Memory Address Register and is designated by the name MAR. Other designations for registers are PC (Program Counter), IR (Instruction Register), and R1 (Processor Register).

The individual flip-flops in an n-bit register are numbered in sequence from 0 through n – 1, starting from 0 in the rightmost position and increasing the numbers toward the left. The numbering of bits in a 16-bit register can be marked on top of the box as shown in **figure 1.18 (c)**.

A 16-bit register is partitioned into two parts as shown in **figure 1.18 (d)**. Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC.

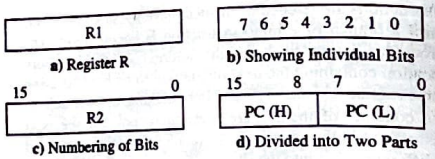


Figure 1.18: Block Diagram of Register

A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability. Normally, one wants the transfer to occur only under a predetermined control condition. This can be shown by means of if-then statement as given below:

If (P = 1) then (R2 ← R1)

Where, P is a control signal generated in the control section.

It is sometimes convenient to separate the control variables from the register transfer operation by specifying a control function. A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

P: R2 ← R1

These micro-operations as the name suggest transfer information from one register to another. The information does not change during this micro-operation.

A register transfer micro-operation may be designed as:

R1 ← R2

This implies that transfer the content of register R2 to register R1. R2 here is a source register while R1 is a destination register.

For a register transfer micro-operation there must be a path for data transfer from the output of the source register to the input of destination register. In addition, the destination register should have a parallel load capability, as one expect the register transfer to occur in a predetermined control condition.

The register transfer description of some typical arithmetic and Boolean operations are summarized as follows:

D ← A′	Transfer the complement of A to D.
A ← A+1	Increment the contents of A by 1.
A ← A – 1	Decrement the contents of A by 1.
A ← A′ + 1	Compute the 2's complement of A.
D ← A ∨ B	Compute the logical OR of A and B and save the result in D.
D ← A ∧ B	Compute the logical AND of A and B and save the result in D.
LSR (A)	Logically shift the contents of the register A one position to the right.
ASR (A)	Arithmetically shift the contents of the register A one position to the right.

The mnemonics LSL, ASL, ROR, and ROL are used to indicate logical left shift, arithmetic left shift, rotate right and rotate left operations respectively.

The $ symbol is used as the concatenation operator. For example, if A and Q are two 8-bit registers and ASR (A$Q) is written, the contents of AQ are arithmetically shifted one position to the right. This operation is a 16-bit operation with high and low-order bytes held in the A and Q registers, respectively.

The basic symbols of the register transfer notation are listed in table 1.3. Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register. The arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time.

For example, the statement: T: R2 ← R1, R1 ← R2

Denotes an operation that exchanges the contents of two registers during one common, clock pulse provided that T = 1. This simultaneous operation is possible with registers that have edge-triggered flip-flops.

Table 1.3: Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma,	Separates two micro-operations	R2 ← R1, R1 ← R2

Ques 22) Explain how to perform arithmetic or logic operations? Draw the diagram whenever necessary.

Ans: Performing of Arithmetic or Logic Operations

The Arithmetic Logic Unit (ALU) is a combinational circuit that has no internal storage. It performs arithmetic and logic operations on the two operands applied to its A and B inputs. In **figure 1.20** and **figure 1.21** one of the operands is the output of the multiplexer MUX and the other operand is obtained directly from the bus:

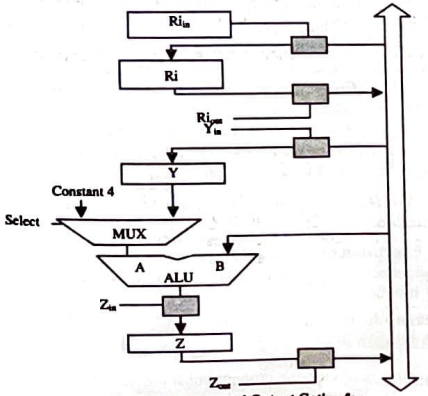


Figure 1.19: Input and Output Gating for the Registers in figure 1.19

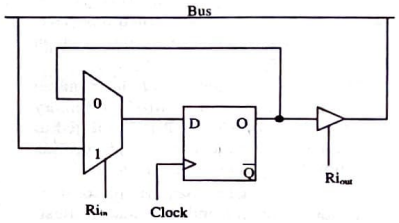


Figure 1.20: Input and Output Gating for One Register Bit

The result produced by the ALU is stored temporarily in register Z. Therefore, a sequence of operations to add the contents of register R1 to those of register R2 and store the result in register R3 is:

1) $R1_{out}$, Y_{in}
2) $R2_{out}$, SelectY, Add, Z_{in}
3) Z_{out}, $R3_{in}$

The signals whose names are given in any step are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive. Hence, in step 1, the output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y. In step 2, the multiplexer's Select signal is set to SelectY, causing the multiplexer to gate the contents of register Y to input A of the ALU. At the same time, the contents of register R2 are gated onto the bus and, hence, to input B. The function performed by the ALU depends on the signals applied to its control lines.

In this case, the Add line is set to 1, causing the output of the ALU to be the sum of the two numbers at inputs A and B. This sum is loaded into register Z because its input control signal is activated. In step 3, the contents of register Z are transferred to the destination register, R3. This last transfer cannot be carried-out during step 2, because only one register output can be connected to the bus during any clock cycle.

In this one assume that there is a dedicated signal for each function to be performed. **For example,** one assume that there are separate control signals to specify individual ALU operations, such as Add, Subtract, XOR, and so on. In reality, some degree of encoding is likely to be used. **For example,** if the ALU can perform eight different operations, three control signals would suffice to specify the required operation.

Ques 23) Write the steps to fetching a word from memory and also storing a word in memory.

Or

Explain the process of storing a word in memory using a single bus organisation. Specify which all control signals will be activated. (2017 [03])

Ans: Fetching a Word from Memory

To fetch a word of information from memory, the following steps to be performed:

Step 1) The CPU has to specify the address of the memory location where this information is stored

and request a Read operation. This applies whether the information to be fetched represents an instruction in a program or a word of data, an operand, specified by an instruction.

Step 2) The CPU transfers the address of the required word of information to the MAR (Memory Address Register), whose output is connected to the address lines of the memory bus. Hence, this address is transferred to the main memory.

Step 3) At the same time, the CPU uses the control lines of the memory bus to indicate that a Read operation is needed. After issuing this request, the CPU normally waits until it receives an answer from the memory informing it that the requested function, the Read operation, has been completed.

Step 4) Another control signal on the memory bus, the Memory-Function-Completed (MFC) signal, is used for this purpose. The memory sets this signal to 1 to indicate that the contents of the specified location have been read and are available on the data lines of the memory bus.

Step 5) As soon as the MFC signal is set to 1, the information on the data lines is loaded into MDR (Memory Data Register) and is thus available for use inside the CPU. Setting the MFC signal to 1 completes the memory fetch operation.

For example, let consider the Move R1, R2 instruction. This is done by the following sequence of operations:

1) MAR ← [R1]
2) Start Read operation on the memory bus
3) Wait for the MFC signal response from the memory
4) Load MDR from the memory bus
5) R2 ← [MDR]

These actions may be carried out as separate steps, but some can be combined into a single step. Each action can be completed in one clock cycle, except step 3which requires one or more clock cycles, depending on the speed of the addressed device.

The duration of step 3 depends on the speed of the memory used. Usually, the time required to read a word from the memory is longer than the time required to perform any single operation within the CPU. Hence, the overall execution time of an instruction can be decreased if the sequence of operations is organized so that a useful function is performed within the CPU while it is waiting for the memory to respond. Only functions that do not require the use of MDR or MAR can be carried out during this time. Such a situation arises during the fetch phase. The PC can be incremented while the CPU is waiting for the Read operation to be completed. The data transfer takes place between two devices, namely, the CPU and the main memory.

Storing a Word in Memory

The procedure for writing a word into a given memory location is as follows: After the address is loaded into the MAR, the data word to be written is loaded into MDR before, or at the same time as, the Write command is issued.

For example, let consider the Move R2, R1 instruction. If we assume that the data word to be stored in the memory is in R2 and that the memory address is in R1, the Write operation requires the following sequence:

1) MAR ← [R1] 2)MDR ← [R2], Write
3) Wait for MFC

Note that steps 1 and 2 can be carried out simultaneously if the architecture allows it, that is, if the two transfers do not use the same physical path. Of course, this is not possible in the single-bus organization. The wait period in step 3 may be overlapped with other operations, provided that such operations do not involve registers MDR or MAR.

Ques 24) Describe the instruction cycle.

Or

What are the various steps involved in the instruction cycle?

Ans: Instruction Cycle

The main function of a CPU is to execute programs. A program consists of a sequence of instructions to perform a particular task. Programs are stored in memory. The CPU fetches one instruction at a time from the memory and executes it. First of all the CPU fetches the first instruction of the program and executes it. Then it fetches the next instruction to execute it. The CPU repeats this process till it executes all the instructions of the program. Thereafter it may take another program if any, to execute.

Steps of Instruction Cycle

The three operations are as below:

1) **Instruction Fetch:** In this cycle, the instruction is fetches from the memory location whose address is in the PC. This instruction is placed in the instruction register (IR) in the processor.
2) **Instruction Decode:** In this cycle, the opcode of the instruction stored in the instruction register is decoded/ examined to determine which operation is to be performed.
3) **Instruction Execution:** In this cycle, the specified operation is performed by the processor. This often involves fetching operands from the memory or from processor registers, performing an arithmetic or logical operation, and storing the result in the destination location. During the instruction execution, PC contents are incremented to point to the next Instruction. After completion of execution of the current instruction, the PC contains the address of the next instruction, and a new instruction fetch cycle can begin.

Ques 25) Discuss the execution of a complete instruction.

Ans: Execution of a Complete Instruction

The sequence of elementary operations required to execute one instruction can be shown by an example.

For example, let consider the instruction:

Add (R3), R1

This instruction adds the contents of a memory location to register R1. The address of the memory operand is the contents of register R3 (Register Indirect Mode).

Executing this instruction requires the following actions:

1) Fetch the instruction.
2) Fetch the first operand (the contents of the memory location pointed to by R3).
3) Perform the addition.
4) Load the result into R1.

Instruction execution proceeds as follows:

Step 1) In **step 1**, loading the contents of the PC (Program Counter) into the MAR (Memory Address Register) and sending a Read request to the memory initiate the instruction fetch operation. While waiting for a response from the memory, the PC is incremented by 1 by setting one of the inputs to the ALU (register Y) to 0 and the other input (CPU bus) to the current value in the PC. At the same time, the carry-in to the ALU is set to 1, and an Add operation is specified.

Step 2) The updated value is moved from register Z back into the PC during **step 2.** Of course, this way of incrementing the PC by using the adder circuit in the ALU is not the fastest approach; instead, the PC can be implemented as a counter circuit with a parallel-load capability. Our approach, however, uses the simplest structure to explain the basic concepts. Note that step 2 begins immediately after the memory Read is requested, without waiting for the memory function to be completed.

Step 3) This step has to be delayed until the MFC signal is received. This is indicated by the WMFC (Wait for MFC) control signal in **step 2.** In **step 3,** the word fetched from the memory is loaded into the IR. The instruction decoding circuit interprets the contents of the IR at the beginning of **step 4.** This enables the control circuitry to choose the appropriate signals for the remainder of the control sequence, **steps 4** through 7, which constitute the execution phase. **Steps 1** through 3 constitute the instruction fetch phase of the control sequence. Of course, this portion is the same for all instructions.

Step 4) The contents of register R3 are transferred to the MAR, and a memory Read operation is initiated. Then the contents of R1 are transferred to register Y.

Step 5) In **step5**, when the Read operation is completed, the memory operand is available in register MDR.

Step 6) The addition operation is performed in **step 6,** and the result is transferred to R1 in **step 7.**

Step 7) The End signal in **step 7** indicates that this is the last step of the current instruction, and it causes a new fetch cycle to begin by returning to **step 1.**

Ques 26) Discuss about the single bus organisations.

Ans: Single Bus Organisations

In single bus organisation, a single bus is used for multiple purposes. A set of general purpose register, program counter, instruction register, memory address register (MAR), memory data register (MDR) are connected with the single bus. Memory read/write can be done with MAR and MDR. The program counter points to the memory location from where the next instruction is to be fetched. Instruction register is that very register will hold the copy of the current instruction. In case of single bus organisation, at a time only one operand can be read from the bus.

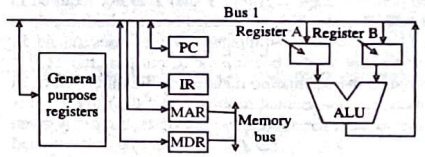


Figure 1.21: Single Bus Organisation

As a result of that, if the requirement is read two operand for the operation then read operation need to be carried twice. So that's why it is making the process little longer. One of the **advantages** of single bus organisation is that, it is one of the simplest and also this is very cheap to implement. At the same time a disadvantage lies that it has only one bus and this "single bus" is accessed by all general purpose registers, program counter, instruction register, MAR, MDR making each and every operation sequential.

Ques 27) Explain the multiple bus organisations with diagram.

Or

Write notes on multiple bus organisation. (2017 [05])

Ans: Multiple Bus Organization

The common bus organization is a very efficient method to interconnect all the units of a computer system. If a large number of registers are included in a processor unit, multiple bus system is needed. **Figure 1.21** depicts a three-bus structure used to connect the registers and the ALU of a CPU. All general-purpose registers are combined into a single block called the **register file.** In VLSI (Very-large-scale integration) technology, the most efficient way to implement these registers is in the form of an array of memory cells similar to those used to implement random-access memory (RAM).

The register file in **figure 1.21** has two outputs, allowing the contents of two registers to be placed on buses A and B simultaneously. **For example,** let consider a three-operand instruction of the form:

OP Rsrc1, Rsrc2, Rdst,

Where,

1) Rsrc1 denotes (Register source1),
2) Rsrc2 denotes (Register Source2) and
3) Rdst denotes (Register Destination)

This operation is performed on the contents of two source registers, and the result is placed into a destination register. Buses A and B are used to transfer the source operands, and bus C provides the path to the destination. The path from the source buses to the destination bus goes through the ALU, where the required operation is performed. Thus, assuming that the operation to be performed can be completed in one pass through the ALU, the structure of **figure 1.22** allows the execution phase of an instruction to be performed in one clock cycle. Note

that if it is merely necessary to copy the contents of one register into another, then the transfer is also done through the ALU, but no arithmetic or logic operation is performed.

The temporary storage registers Y and Z is not required in **figure 1.22.** Register Y is not needed because both inputs to the ALU are provided simultaneously via buses A and B. Register Z is not needed because the output from the ALU is transferred to the destination register via the third bus, C. In this structure, it is essential to ensure that the same register can serve as both the source and the destination in a given instruction. Instead, the register file must be implemented using either edge-trigger or master-slave circuits.

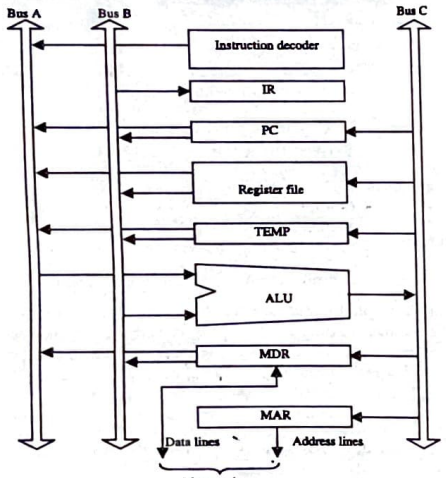


Figure 1.22: Three-bus Organization of CPU

The three-bus structure allows execution of register-to-register operations in a single clock cycle. This is particularly well suited to the requirements of RISC processors, in which most arithmetic and logic instructions have register operands.

Ques 28) How control signals are sequenced? Explain the various control design techniques.

Or

Describe the hardwired based design and micro Programmed based design.

Ans: Sequencing of Control Signals/Control Design Techniques

To execute instructions; the CPU must have some means of generating the control signals in the proper sequence. Computer designers have used a wide variety of techniques to solve this problem. Most of these techniques, however, fall into one of two categories:

1) **Hardwired Based Design of Control Unit:** It is a controller as a sequential logic circuit or a finite state machine that generates a sequence of control signals in response to the externally supplied instructions. In a hardwired implementation the following points must be noted.
   i) The control unit is essentially a combinational circuit.
   ii) The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
   iii) A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. Its input logic signals are transformed in to a set of output logic signals that are called the control signals.

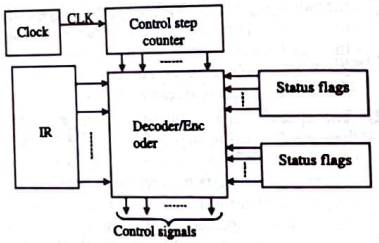


Figure 1.23: Block Diagram of Hardwired Control Unit

2) **Micro-Programmed Based Design of Control Unit:** The design of control unit must include logic for sequencing through micro-operations, for executing micro-operations, for interpreting opcodes, and for making decisions based on ALU flags. It is difficult to design and test such a piece of hardware. An alternative, which is quite common in contemporary CISC processors, is to implement a micro-programmed control unit. Due to lack of flexibility in hardwired control unit, it is quite difficult to design, test and implement as in many computers the number of control lines is in hundreds.

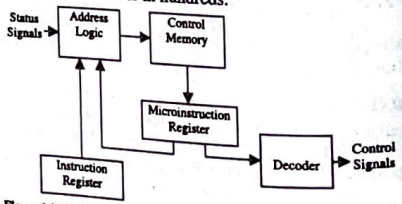


Figure 1.24: General Structure of a Micro-programmed Control Unit

The sequence of execution of micro-operations through a program will consist of instructions, with each of the instruction describing:

i) One or more micro-operations to be executed
ii) The information about the micro-instruction to be executed next.

Such an instruction is termed as **micro-instruction** and such a program is termed as a **micro-program** or **firmware**. In the micro-programmed organization, the control information is stored in a sequence of micro-operations.

Module 2

Register Transfer Logic

REGISTER TRANSFER LOGIC

Ques 1) What is register?

Ans: Register

Register is group of flip-flops with each flip-flop capable of storing one bit of information. Registers are temporary storage units, made-up of Integrated Circuits (IC's) of a computer that keep data as well as instructions in binary 0 and 1 form.

Registers are at the top of the memory hierarchy, and provide the fastest way for a CPU to access data. A register is normally used for temporary storage of a related set of bits for some operations. This is a common activity in digital design, especially when the system must process byte or word organised data.

A register can store as much bits as the number of flip-flop it contains. So, an 'n' bit register must be capable of storing 'n' number of bits, i.e., it has 'n' number of flip-flops. It can have logical too, so it has processing capability also.

Ques 2) Describe the various types of register.

Or

Explain general purpose and special purpose registers in detail.

Ans: Types of Registers

The register of CPU can be classified as below:

1) **General Purpose Registers:** These registers store data and intermediate results during the execution of a program. They are accessible to users through instructions if the users are working in assembly language. Some general-purpose registers are as follows:
   i) **AX (Accumulator Register):** It is used for arithmetic and other data operations. An accumulator is a register in which intermediate arithmetic and logic results are stored.
   ii) **BX (Base Register):** It is used to-store memory addresses of data stored in main memory during arithmetic and data movement operations.
   iii) **CX (Counter Register):** It is used for counting purpose. It acts as counter for looping.
   iv) **DX (Data Register):** It is used to hold data during division and multiplication operations.
2) **Special Purpose Registers:** A CPU contains a number of special purpose registers for different purposes. All CPUs do not contain all of these special registers. A powerful CPU contains most of them. These are:
   i) **Program Counter (PC):** This is a register, which deals with the order for execution of instructions. This acts like a pointer that indicates the subsequent memory location where instruction is stored. After one instruction is executed, the program counter gets incremented by one to indicate the location of the next instruction in serial order.
   ii) **Stack Pointer (SP):** Stack may be defined as a set of memory locations and the stack pointer may be defined as the indicator to these memory locations. Stack memory locations are used by a microprocessor for storing data temporarily for execution of a program.
   iii) **Status Register:** The status register also called **flag register**, holds 1-bit flags to indicate certain conditions that arise during arithmetic and logical operations.
   iv) **Instruction Register:** It holds an instruction until it is decoded. That is, it stores the instruction currently being executed.
   v) **Index Register:** Index registers are used for addressing.
   vi) **Memory Address Register (MAR):** It holds the address of the instruction or data to be fetched from the memory.
   vii) **Memory Buffer Register (MBR) or Data Register (DR):** It holds the instruction code or data received from or sent to the memory. It is connected to data bus.

Ques 3) Describe the register transfer logic. Also explain the various components of register transfer logic.

Ans: Register Transfer Logic

A digital system is a sequential logic system constructed with flip-flops and gates. It was shown in previous chapters that a sequential circuit can be specified by means of a state table. To specify a large digital system with a state table would be very difficult, if not impossible, because the number of states would be prohibitively large. To overcome this difficulty, digital systems are invariably

designed using a modular approach. The system is partitioned into modular subsystems, each of which performs some functional task.

The modules are constructed from such digital functions as registers, counters, decoders, multiplexers, arithmetic elements, and control logic. The various modules are interconnected with common data and control paths to form a digital computer system. A typical digital system module would be the processor unit of a digital computer. The interconnection of digital functions to form a digital system module cannot be described by means of combinational or sequential logic techniques. These techniques were developed to describe a digital system at the gate and flip-flop level and are not suitable for describing the system at the digital function level.

Components of Register Transfer Logic

The following four **components** form the basis of the register-transfer logic method:

1) **Set of Register:** A register, as defined in the register-transfer logic notation, not only implies a register, but also encompasses all other types of registers, such as shift registers, counters, and memory units. A counter is considered to be a register whose function is to increment by 1 the information stored within it.

2) **Binary-Coded Information:** The binary information stored in registers may be binary numbers, binary-coded decimal numbers, alphanumeric characters, control information, or any other binary-coded information. The operations that are performed on the data stored in registers depend on the type of data encountered.

3) **Operations:** The operations performed on the data stored in registers are called micro-operations. A micro-operation is an elementary operation that can be performed in parallel during one clock pulse period.

4) **Control Functions:** The control functions that initiate the sequence of operations consist of timing signals that sequence the operations one at a time. Certain conditions which depend on results of previous operations may also determine the state of control functions. A control function is a binary variable that, when in one binary state, initiates an operation and. when in the other binary state, inhibits the operation.

Ques 4) Write a short note on Register Transfer Language.

Or

What do you understand by Register Transfer Language?

Ans: Register Transfer Language (RTL)

Register Transfer Language (RTL) is a concise way of specifying microcode instructions. Register Transfer Language is a language for describing the behaviour of computer in terms of stepwise register contents.

A register transfer language is a system for expressing in symbolic form the micro operation sequences among the registers of a digital module.

A statement in a register-transfer language consists of:

1) **Control Function:** The control function (which may be omitted sometimes) specifies the control condition and timing sequence for executing the listed micro-operations.

2) **List of Micro-Operation:** The micro-operations specify the elementary operations to be performed on the information stored in registers.

The types of micro-operations most often encountered in digital systems can be classified into four categories:

i) **Inter-register transfer micro-operations** do not change the information content when the binary information moves from one register to another.

ii) **Arithmetic micro-operations** perform arithmetic on numbers stored in registers.

iii) **Logic micro-operations** perform operations such as AND and OR on individual pairs of bits stored in registers.

iv) **Shift micro-operations** specify operations for shift registers.

Ques 5) What is inter-register transfer? Explain with suitable example.

Ans: Inter-Register Transfer

The registers in a digital system are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

For example, the register that holds an address for the memory unit is usually called the memory address register and is designated MAR. Other designations for registers are A, B, R_1, R_2, and IR. The cells or flip-flops of an n-bit register are numbered in sequence from 1 to n (or from 0 to n – 1) starting either from the left or from the right.

Figure 2.1 shows four ways to represent a register in block diagram form. The most common way to represent a register is by a rectangular box with the name of the register inside, as shown in **figure 2.1 (a).** The individual cells can be distinguished as in (b), with each cell assigned a letter with a subscript number.

The numbering of cells from right to left can be marked on top of the box as in the 12-bit register MBR in (c). A 16-bit register is partitioned into two parts in (d). Bits 1 through 8 are assigned the symbol letter L (for low) and bits 9 through 16 are assigned the symbol letter H (for high). The name of the 16-bit register is PC. The symbol PC(H) refers to the eight high-ordered cells and PC(L) refers to the eight low-ordered cells of the register.

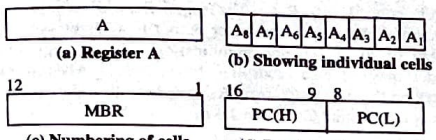


Figure 2.1: Block Diagram of Registers

Registers can be specified in a register-transfer language with a declaration statement. **For example,** the registers of **figure 2.1** can be defined with declaration statements such as:

DECLARE REGISTER A (8), MBR(12), PC(16)
DECLARE SUBREGISTER PC(L) = PC(1 – 8), PC(H) = PC(9 – 16)

The registers will be shown in block diagram form as in **figure 2.2.** Registers shown in a block diagram can be easily converted into declaration statements for simulation purposes. Information transfer from one register to another is designated in symbolic form by means of the replacement operator. The statement:

$A \leftarrow B$

Denotes the transfer of the contents of register B into register A. It designates a replacement of the contents of A by the contents of B. By definition, the contents of the source register B do not change after the transfer.

A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the cell inputs of the destination register. Normally, we do not want this transfer to occur with every clock pulse, but only under a predetermined condition. The condition that determines when the transfer is to occur is called a control function. A control function is a Boolean function that can be equal to 1 or 0. The control function is included with the statement as follows:

$x'T_1: A \leftarrow B$

The control function is terminated with a colon. It symbolises the requirement that the transfer operation be executed by the hardware only when the Boolean function $x'T_1 = 1$, i.e., when variable $x = 0$ and timing variable $T_1 = 1$.

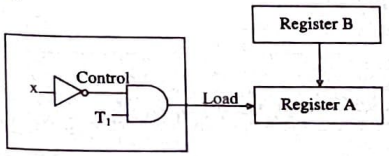


Figure 2.2: Hardware Implementation of the Statement $x'T_1: A \leftarrow B$

Every statement written in a register-transfer language implies a hardware construction for implementing the transfer. **Figure 2.2** shows the implementation of the statement written above.

Ques 6) Describe the micro-operations. What are the various types of micro-operations?

Ans: Micro-Operations

The primary function of a CPU is to execute sequence of instructions which is in accordance with the instruction cycle. The Instruction cycle has three major phases of fetch decode and execute. To perform these cycles the processor unit has to perform a set of operations called **micro-operations**. Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operation executed on data stored in registers are called **microoperations**.

A microoperation is an elementary operation performed on the information stored in one or more registers.

The result of the operation may replace the previous binary information of a register or may be transferred to another register. **For example,** some common microoperations are **shift, count, clear,** and **load.**

Types of Micro-Operations

The micro-operations most often encountred in digital computers are classfied into four categories:

1) **Arithmetic Micro-Operations:** Perform arithmetic operations on numeric data stored in registers. The arithmetic micro-operation is defined by the statement,

R3 ← R1 + R2

It states that the contents of register R1 are added to the contents of register R2 and the sum is transferred to register R3. Hence basically three registers are required for its implementation.

2) **Logic Micro-Operations:** Perform bit manipulation operations on non-numeric data stored in registers. Logic micro-operations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables. The logic micro-operation is defined by the statement,

R1 ← R1 ⊕ R2

It states that the contents of register R1 after the execution is equal to the bit by bit EX-OR operation on pair of bits in R2 and previous values of R1.

3) **Shift Micro-Operations:** Perform shift operations on data stored in registers. Shift micro-operation transfers binary bits between the register serially. This shift operation is also used to perform binary multiplication and binary division. Registers can be shifted to the left or to the right. There are no conventional symbols for the shift operations. The shift micro-operations are represented by **shl** and **shr** to shift the content to the left and to the right respectively. The logic micro-operation is defined by the statement,

R1←shl R1
R2←shr R2

It states that a 1 bit shift to the left of the content of register R1 and a 1 bit shift to the right of the content of register R2.

Ques 7) Explain the arithmetic micro-operations in detail.

Or

Design an adder/subtractor circuit with one selection variable s and two inputs A and B. When s = 0 the circuit performs A + B. When s = 1 the circuit performs A – B by taking 2's complement of B.

(2019 [05])

Ans: Arithmetic Micro-Operations

Arithmetic micro-operation of the form L: $R_A \leftarrow (R_A + R_B)$ refers to two data items contained in R_A and R_B input to a parallel adder with the result being stored back in R_A (**figure 2.3 a**). Loading of the result in R_A is subject to the availability of the control signal designated as L (that is load control of R_A). The parallel adder being a combinational logic, the adder output is available after a delay of the associated combinational circuit. Input to adder may be also controlled by providing AND word gates on the data paths from R_A and R_B to the adder under the control signal C (**figure 2.3 a**).

The subtract micro-operation L : $R_A \leftarrow (R_A + R_B)$ in 2's complement arithmetic has the form L: $R_A \leftarrow R_A + \overline{R}_B + 1$ where $\overline{R}_B$ is 1's complement of R_B. The hardware implementing both Add/Subtract operation is shown in (**figure 2.3 b**) where the control signals 'C' and 2's complement are at logic '1' for subtract operation and '0' for add operation. The EX-OR word gate inverts the content of R_B for C=1 which also makes input carry C_0=1 to make 2's complement of R_B. The circuit in (**figure 2.3 b**) is so designed that it can support both 1's and 2's complement of R_B as input to the adder.

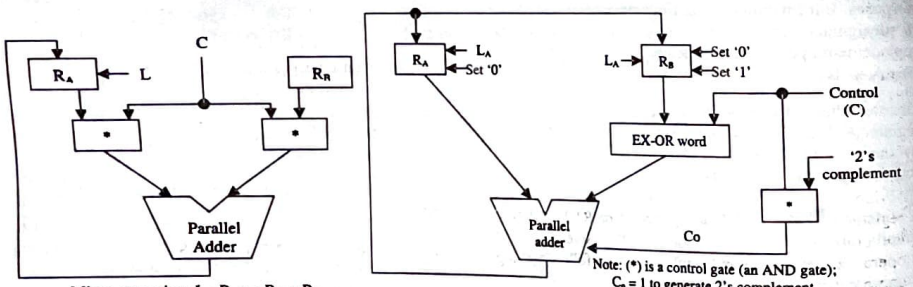


Figure 2.3: Arithmetic Micro-Operations

The other common arithmetic micro-operations are increment, decrement, 1's complement, 2's complement, etc. Each of these operations can be implemented with the help of multiple micro-operations on the structure of **figure 2.3** as given in **table 2.1**.

Table 2.1: Arithmetic Micro-Operation

Operation	Implemented by set of Micro Operations	Control Signals and their Application Point on Data Path as in figure Above
1) Increment	RB ⇐ 1;	i) Set RB to 1
RA ⇐ RA + 1	RA ⇐ RA + RB	ii) Set C to 0
		Set 2's complement to 0
		iii) LA at RA
2) Decrement	RB ⇐ 0;	i) Set RB to 0
RA ⇐ RA – 1	RA ⇐ RA + RB	ii) Set C to 1
		Set 2's complement to 0
		iii) LA at RA
3) 1's Complement	RA ⇐ 0;	i) Set RA to 0
RB ⇐ $\overline{R}$ B	RB ⇐ RA + RB	ii) Set C to 1
		Set 2's complement to 0
		iii) LB at RB
4) 2's Complement	RA ⇐ 0;	i) Set RA to 0
	RB ⇐ RA + RB	ii) Set C to 1
		Set 2's complement to 1
		iii) LB at Rb

The sequences of micro-operations are explicitly noted on the second column of the table. The control signals, as noted on the third column, are generated as per the desired sequence and applied on the data path.

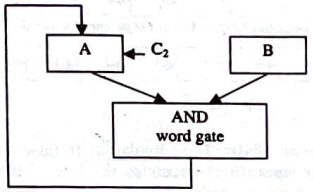


Figure 2.4: AND word Gate Implementing C_2: A ⇐ A · B

Ques 8) Discuss the logic micro-operations in detail.

Ans: Logic Micro-Operation

Logic micro-operations refer to logical operations on bits of a pair of registers or a single register. These micro-operations specify binary operations for strings of bits stored in registers. They consider each bit of the register separately and treat them as binary variables. **For example,** the exclusive-OR micro-operation with the contents of two registers R1 and R2 is symbolised by the statement,

P: R1 ← R1 ⊕ R2

It specifies a logic micro-operation to be executed on the individual bits of the registers provided that the control variable P = 1. As a numerical example, assume that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100. The exclusive-OR micro-operation stated below symbolises the following logic computation:

1010	Content of R1
1100	Content of R2
0110	Content of R1 after P = 1

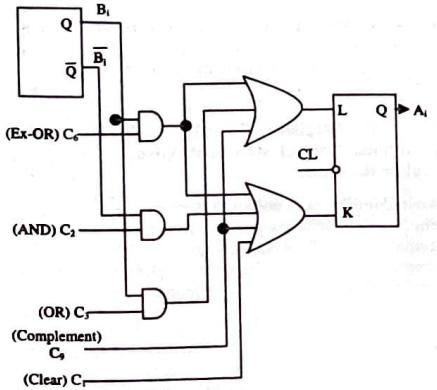


Figure 2.5: Hardware Realising the Behaviour of the Excitation Table

The content of R1, after the execution of the micro-operation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R1. The logic micro-operations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Special symbols will be adopted for the logic micro-operations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions. The symbol ∨ will be used to denote an OR micro-operation and the symbol ∧ to denote an AND micro-operation.

The complement micro-operation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name. By using different symbols, it will be possible to differentiate between a logic micro-operation and a control (or Boolean) function.

Typical logic micro-operations are noted in **table 5.2.** The specific control signals C_1 to C_{14} execute the corresponding micro-operations. To support such micro-operations on binary data stored on register(s), necessary hardware support needs to be embedded in the data path.

Simple word gates, for the AND word gate, can implement corresponding Boolean functions. However, more economical hardware can be realised by designing a sequential circuit involving the source and destination registers.

As indicated in **table 2.2,** one of the source registers acts as the destination register. The control signals generated by the controller for different operation codes are applied to the circuit as noted in the **figure 2.5.**

On similar considerations, the logic circuit of ALU can be designed to handle other logical micro-operations. Control signals are applied on the ALU block to execute the specific micro-operation.

Table 2.2: Logic Micro-Operation

Function	Micro-Operation
Clear	C_1 : A ⇐ 0
AND	C_2 : A ⇐ A · B
OR	C_3 : A ⇐ A + B
NOR	C_4 : A ⇐ $\overline{A + B}$
NAND	C_5 : A ⇐ $\overline{A \cdot B}$
Exclusive-OR	C_6 : A ⇐ A ⊕ B
Exclusive-NOR	C_7 : A ⇐ $\overline{A \oplus B}$
Complement	C_8 : A ⇐ $\overline{A}$
(Logical) shift left A	C_9 : A ⇐ SHLA
(Logical) shift right A	C_{10}: A ⇐ SHRA
Rotate right A	C_{11}: A ⇐ RRA
Rotate left A	C_{12}: A ⇐ RLA
Arithmetic shift left A	C_{13}: A ⇐ ASHLA
Arithmetic shift right A	C_{14}: A ⇐ ASHRA

designed using a modular approach. The system is partitioned into modular subsystems, each of which performs some functional task.

The modules are constructed from such digital functions as registers, counters, decoders, multiplexers, arithmetic elements, and control logic. The various modules are interconnected with common data and control paths to form a digital computer system. A typical digital system module would be the processor unit of a digital computer. The interconnection of digital functions to form a digital system module cannot be described by means of combinational or sequential logic techniques. These techniques were developed to describe a digital system at the gate and flip-flop level and are not suitable for describing the system at the digital function level.

Components of Register Transfer Logic

The following four **components** form the basis of the register-transfer logic method:

1) **Set of Register:** A register, as defined in the register-transfer logic notation, not only implies a register, but also encompasses all other types of registers, such as shift registers, counters, and memory units. A counter is considered to be a register whose function is to increment by 1 the information stored within it.

2) **Binary-Coded Information:** The binary information stored in registers may be binary numbers, binary-coded decimal numbers, alphanumeric characters, control information, or any other binary-coded information. The operations that are performed on the data stored in registers depend on the type of data encountered.

3) **Operations:** The operations performed on the data stored in registers are called micro-operations. A micro-operation is an elementary operation that can be performed in parallel during one clock pulse period.

4) **Control Functions:** The control functions that initiate the sequence of operations consist of timing signals that sequence the operations one at a time. Certain conditions which depend on results of previous operations may also determine the state of control functions. A control function is a binary variable that, when in one binary state, initiates an operation and. when in the other binary state, inhibits the operation.

Ques 4) Write a short note on Register Transfer Language.

Or

What do you understand by Register Transfer Language?

Ans: Register Transfer Language (RTL)

Register Transfer Language (RTL) is a concise way of specifying microcode instructions. Register Transfer Language is a language for describing the behaviour of computer in terms of stepwise register contents.

A register transfer language is a system for expressing in symbolic form the micro operation sequences among the registers of a digital module.

A statement in a register-transfer language consists of:

1) **Control Function:** The control function (which may be omitted sometimes) specifies the control condition and timing sequence for executing the listed micro-operations.

2) **List of Micro-Operation:** The micro-operations specify the elementary operations to be performed on the information stored in registers.

The types of micro-operations most often encountered in digital systems can be classified into four categories:

i) **Inter-register transfer micro-operations** do not change the information content when the binary information moves from one register to another.

ii) **Arithmetic micro-operations** perform arithmetic on numbers stored in registers.

iii) **Logic micro-operations** perform operations such as AND and OR on individual pairs of bits stored in registers.

iv) **Shift micro-operations** specify operations for shift registers.

Ques 5) What is inter-register transfer? Explain with suitable example.

Ans: Inter-Register Transfer

The registers in a digital system are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

For example, the register that holds an address for the memory unit is usually called the memory address register and is designated MAR. Other designations for registers are A, B, R_1, R_2, and IR. The cells or flip-flops of an n-bit register are numbered in sequence from 1 to n (or from 0 to n – 1) starting either from the left or from the right.

Figure 2.1 shows four ways to represent a register in block diagram form. The most common way to represent a register is by a rectangular box with the name of the register inside, as shown in **figure 2.1 (a)**. The individual cells can be distinguished as in (b), with each cell assigned a letter with a subscript number.

The numbering of cells from right to left can be marked on top of the box as in the 12-bit register MBR in (c). A 16-bit register is partitioned into two parts in (d). Bits 1 through 8 are assigned the symbol letter L (for low) and bits 9 through 16 are assigned the symbol letter H (for high). The name of the 16-bit register is PC. The symbol PC(H) refers to the eight high-ordered cells and PC(L) refers to the eight low-ordered cells of the register.

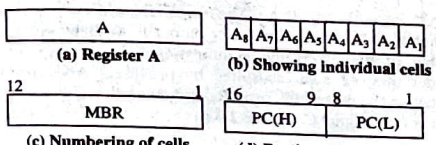


Figure 2.1: Block Diagram of Registers

Registers can be specified in a register-transfer language with a declaration statement. **For example,** the registers of **figure 2.1** can be defined with declaration statements such as:

DECLARE REGISTER A (8), MBR(12), PC(16)
DECLARE SUBREGISTER PC(L) = PC(1 – 8), PC(H) = PC(9 – 16)

The registers will be shown in block diagram form as in **figure 2.2.** Registers shown in a block diagram can be easily converted into declaration statements for simulation purposes. Information transfer from one register to another is designated in symbolic form by means of the replacement operator. The statement:

$A \leftarrow B$

Denotes the transfer of the contents of register B into register A. It designates a replacement of the contents of A by the contents of B. By definition, the contents of the source register B do not change after the transfer.

A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the cell inputs of the destination register. Normally, we do not want this transfer to occur with every clock pulse, but only under a predetermined condition. The condition that determines when the transfer is to occur is called a control function. A control function is a Boolean function that can be equal to 1 or 0. The control function is included with the statement as follows:

$x'T_1: A \leftarrow B$

The control function is terminated with a colon. It symbolises the requirement that the transfer operation be executed by the hardware only when the Boolean function $x'T_1 = 1$, i.e., when variable $x = 0$ and timing variable $T_1 = 1$.

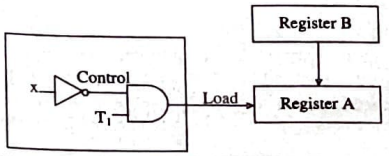


Figure 2.2: Hardware Implementation of the Statement $x'T_1: A \leftarrow B$

Every statement written in a register-transfer language implies a hardware construction for implementing the transfer. **Figure 2.2** shows the implementation of the statement written above.

Ques 6) Describe the micro-operations. What are the various types of micro-operations?

Ans: Micro-Operations

The primary function of a CPU is to execute sequence of instructions which is in accordance with the instruction cycle. The Instruction cycle has three major phases of fetch decode and execute. To perform these cycles the processor unit has to perform a set of operations called **micro-operations**. Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operation executed on data stored in registers are called **microoperations**.

A microoperation is an elementary operation performed on the information stored in one or more registers.

The result of the operation may replace the previous binary information of a register or may be transferred to another register. **For example,** some common microoperations are **shift, count, clear,** and **load.**

Types of Micro-Operations

The micro-operations most often encountred in digital computers are classfied into four categories:

1) **Arithmetic Micro-Operations:** Perform arithmetic operations on numeric data stored in registers. The arithmetic micro-operation is defined by the statement,

R3 ← R1 + R2

It states that the contents of register R1 are added to the contents of register R2 and the sum is transferred to register R3. Hence basically three registers are required for its implementation.

2) **Logic Micro-Operations:** Perform bit manipulation operations on non-numeric data stored in registers. Logic micro-operations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables. The logic micro-operation is defined by the statement,

R1 ← R1 ⊕ R2

It states that the contents of register R1 after the execution is equal to the bit by bit EX-OR operation on pair of bits in R2 and previous values of R1.

3) **Shift Micro-Operations:** Perform shift operations on data stored in registers. Shift micro-operation transfers binary bits between the register serially. This shift operation is also used to perform binary multiplication and binary division. Registers can be shifted to the left or to the right. There are no conventional symbols for the shift operations. The shift micro-operations are represented by **shl** and **shr** to shift the content to the left and to the right respectively. The logic micro-operation is defined by the statement,

R1←shl R1
R2←shr R2

It states that a 1 bit shift to the left of the content of register R1 and a 1 bit shift to the right of the content of register R2.

Ques 7) Explain the arithmetic micro-operations in detail.

Or

Design an adder/subtractor circuit with one selection variable s and two inputs A and B. When s = 0 the circuit performs A + B. When s = 1 the circuit performs A – B by taking 2's complement of B.

(2019 [05])

Ans: Arithmetic Micro-Operations

Arithmetic micro-operation of the form L: $R_A \leftarrow (R_A + R_B)$ refers to two data items contained in R_A and R_B input to a parallel adder with the result being stored back in R_A (**figure 2.3 a**). Loading of the result in R_A is subject to the availability of the control signal designated as L (that is load control of R_A). The parallel adder being a combinational logic, the adder output is available after a delay of the associated combinational circuit. Input to adder may be also controlled by providing AND word gates on the data paths from R_A and R_B to the adder under the control signal C (**figure 2.3 a**).

The subtract micro-operation L : $R_A \leftarrow (R_A + R_B)$ in 2's complement arithmetic has the form L: $R_A \leftarrow R_A + \overline{R}_B + 1$ where $\overline{R}_B$ is 1's complement of R_B. The hardware implementing both Add/Subtract operation is shown in (**figure 2.3 b**) where the control signals 'C' and 2's complement are at logic '1' for subtract operation and '0' for add operation. The EX-OR word gate inverts the content of R_B for C=1 which also makes input carry C_0=1 to make 2's complement of R_B. The circuit in (**figure 2.3 b**) is so designed that it can support both 1's and 2's complement of R_B as input to the adder.

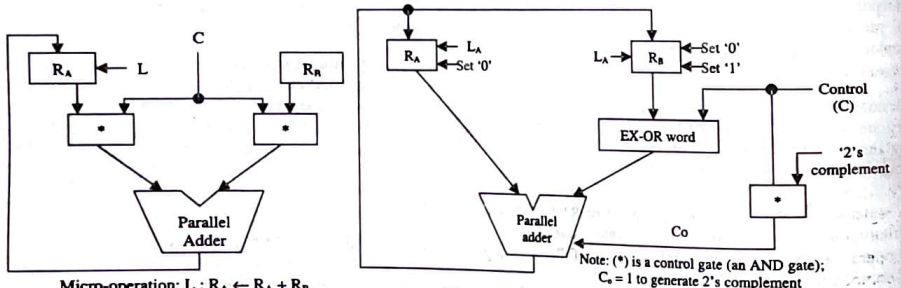


Figure 2.3: Arithmetic Micro-Operations

The other common arithmetic micro-operations are increment, decrement, 1's complement, 2's complement, etc. Each of these operations can be implemented with the help of multiple micro-operations on the structure of **figure 2.3** as given in **table 2.1**.

Table 2.1: Arithmetic Micro-Operation

Operation	Implemented by set of Micro Operations	Control Signals and their Application Point on Data Path as in figure Above
1) Increment	RB ⇐ 1;	i) Set RB to 1
RA ⇐ RA + 1	RA ⇐ RA + RB	ii) Set C to 0
		Set 2's complement to 0
		iii) LA at RA
2) Decrement	RB ⇐ 0;	i) Set RB to 0
RA ⇐ RA – 1	RA ⇐ RA + RB	ii) Set C to 1
		Set 2's complement to 0
		iii) LA at RA
3) 1's Complement	RA ⇐ 0;	i) Set RA to 0
RB ⇐ $\overline{R}$ B	RB ⇐ RA + RB	ii) Set C to 1
		Set 2's complement to 0
		iii) LB at RB
4) 2's Complement	RA ⇐ 0;	i) Set RA to 0
	RB ⇐ RA + RB	ii) Set C to 1
		Set 2's complement to 1
		iii) LB at Rb

The sequences of micro-operations are explicitly noted on the second column of the table. The control signals, as noted on the third column, are generated as per the desired sequence and applied on the data path.

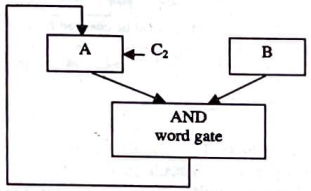


Figure 2.4: AND word Gate Implementing C_2: A ⇐ A · B

Ques 8) Discuss the logic micro-operations in detail.

Ans: Logic Micro-Operation

Logic micro-operations refer to logical operations on bits of a pair of registers or a single register. These micro-operations specify binary operations for strings of bits stored in registers. They consider each bit of the register separately and treat them as binary variables. **For example,** the exclusive-OR micro-operation with the contents of two registers R1 and R2 is symbolised by the statement,

P: R1 ← R1 ⊕ R2

It specifies a logic micro-operation to be executed on the individual bits of the registers provided that the control variable P = 1. As a numerical example, assume that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100. The exclusive-OR micro-operation stated below symbolises the following logic computation:

1010	Content of R1
1100	Content of R2
0110	Content of R1 after P = 1

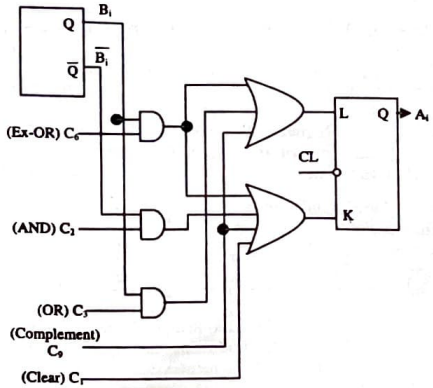


Figure 2.5: Hardware Realising the Behaviour of the Excitation Table

The content of R1, after the execution of the micro-operation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R1. The logic micro-operations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Special symbols will be adopted for the logic micro-operations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions. The symbol ∨ will be used to denote an OR micro-operation and the symbol ∧ to denote an AND micro-operation.

The complement micro-operation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name. By using different symbols, it will be possible to differentiate between a logic micro-operation and a control (or Boolean) function.

Typical logic micro-operations are noted in **table 5.2.** The specific control signals C_1 to C_{14} execute the corresponding micro-operations. To support such micro-operations on binary data stored on register(s), necessary hardware support needs to be embedded in the data path.

Simple word gates, for the AND word gate, can implement corresponding Boolean functions. However, more economical hardware can be realised by designing a sequential circuit involving the source and destination registers.

As indicated in **table 2.2,** one of the source registers acts as the destination register. The control signals generated by the controller for different operation codes are applied to the circuit as noted in the **figure 2.5.**

On similar considerations, the logic circuit of ALU can be designed to handle other logical micro-operations. Control signals are applied on the ALU block to execute the specific micro-operation.

Table 2.2: Logic Micro-Operation

Function	Micro-Operation
Clear	C_1 : A ⇐ 0
AND	C_2 : A ⇐ A · B
OR	C_3 : A ⇐ A + B
NOR	C_4 : A ⇐ $\overline{A + B}$
NAND	C_5 : A ⇐ $\overline{A \cdot B}$
Exclusive-OR	C_6 : A ⇐ A ⊕ B
Exclusive-NOR	C_7 : A ⇐ $\overline{A \oplus B}$
Complement	C_8 : A ⇐ $\overline{A}$
(Logical) shift left A	C_9 : A ⇐ SHLA
(Logical) shift right A	C_{10}: A ⇐ SHRA
Rotate right A	C_{11}: A ⇐ RRA
Rotate left A	C_{12}: A ⇐ RLA
Arithmetic shift left A	C_{13}: A ⇐ ASHLA
Arithmetic shift right A	C_{14}: A ⇐ ASHRA

Ques 9) Explain the shift micro-operation with example.

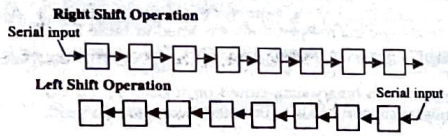
Or

Discuss shift micro operations. (2018 [3.5])

Ans: Micro Programmed Shift Operations along with Register Transfer

Shift is a useful operation that can be used for serial transfer of data. Shift operations can also be used along with other (arithmetic logic, etc.) operations. **For example,** for implementing a multiply operation arithmetic micro-operation (addition) can be used along with shift operation.

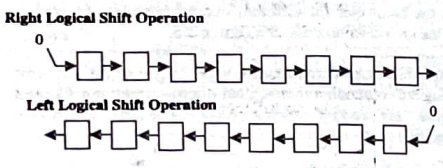
The shift operation may result in shifting the contents of a register to the left or right. In a shift left operation a bit of data is input at the rightmost flip-flop while in shift right a bit of a data is input at the leftmost flip-flop. In both the cases a bit of data enters the shift register.



Types of Shift Micro-Operations

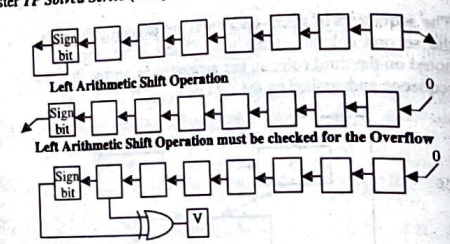
Depending on what bit enters the register and where does the shift out bit goes, the shifts are classified in three types. These are:

1) **Logical Shift:** In this the data entering by serial input to leftmost or rightmost flop-flop (depending on right or left shift operations respectively) is a 0.



2) **Arithmetic Shift:** In this a signed binary number is shifted to the left or to the right. Thus, an arithmetic shift-left causes a number to be multiplied by 2; on the other hand a shift-right causes a division by 2. But as in division or multiplication by 2 the sign of a number should not be changed, therefore, arithmetic shift must leave the sign but unchanged:
   i) An arithmetic shift is meant for signed binary numbers (integer).
   ii) An arithmetic left shift multiplies a signed number by two.
   iii) An arithmetic right shift divides a signed number by two.

   The main distinction of an arithmetic shift is that it must keep the sign of the number. The same as it performs the multiplication or division.



3) **Circular Shift:** The circular shift (also known as **rotate operation**) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input.

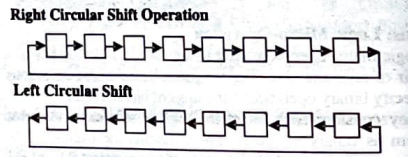


Table 2.3 shows the some common shift micro-operations.

Table 2.3: Shift Micro-Operation

Symbolic Designation	Description
R ← shl R	Shift-left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left register R
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic shift-left R
R ← ashr R	Arithmetic shift-right R

The shift micro-operations can be implemented with a bidirectional shift register.

Ques 10) What are conditional control statements?

Or

Discuss conditional control micro operations. (2018 [3 ½])

Or

Write the Register Transfer Logic format for a conditional control statement. Give an example and explain the same. (2019 [04])

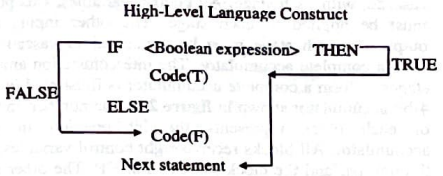
Ans: Conditional Control Statements

The control functions and conditional control statements formulate the Boolean functions for the gates in the control unit. High-level languages provide constructs for conditional code execution and loops.

These constructs translate into test and branch instructions in assembly language. Following are some commonly used conditional control statements:

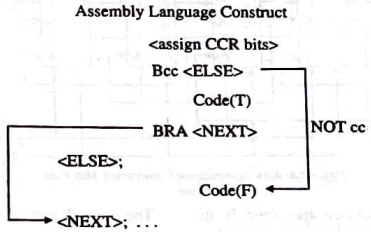
1) **IF ... THEN ... ELSE Statement:** Frequently, program logic requires an "either or" choice. If the condition is TRUE, take one line of action (Code (T)]: otherwise, select an alternative action (Code(F)]. Most

high-level languages have an IF .. THEN .. ELSE statement.



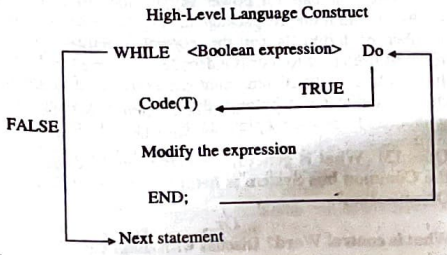
The corresponding assembly language construct requires two branches. A conditional branch directs program control to the Code(F) block.

In case the Code(T) block is executed, an unconditional BRA branch transfers control to the "next statement."

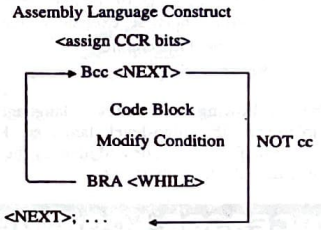


2) **WHILE ... DO Statement:** A program loop is an extension of simple conditionals. In this case, however, a branch returns program control to previously executed instructions. The WHILE... DO construct gives a general form of a loop. The WHILE clause tests a condition.

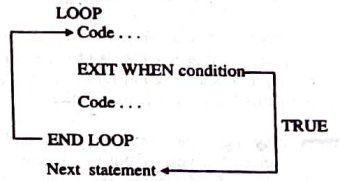
If it is TRUE, program control enters the loop and executes Code (T). Before returning to the WHILE test, some variable is modified so that the test eventually fails. The following diagram shows the structure of the WHILE loop:



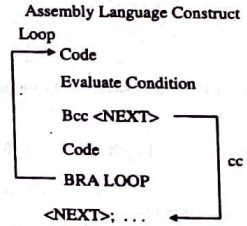
In an assembly language program, a conditional branch tests the Boolean expression. An unconditional BRA branch implements the return.



3) **LOOP ... EXIT Statement:** High-level languages generally provide a variety of loop constructs. The modem trend in languages is to unify these constructs in a single, all-purpose LOOP .. EXIT statement. The "exit" condition can be tested at any appropriate point in the code block. If it is TRUE, an exit from the loop is performed.



Assembly language code implements the "EXIT WHEN" test with a branch. The LOOP .. EXIT construct is the simplest way to implement an assembly language loop.



4) **FOR Loop Statement:** A special case of the WHILE ... DO construct is a loop with a fixed number of iterations. In many high-level languages, a FOR ... DO instruction, which carries out a series of steps to initialise and test a counter and execute a block, of code, is available. In Pascal, the FOR ... DO loop has the syntax:
FOR Counter: = Initial Value TO Upper Limit DO (Code)

The following sequence of steps implements the construct.

Step 1: Initialise the counter.

Step 2: Compare the counter with the upper limit.

Step 3: Exit if the value of the counter is greater than the upper limit; otherwise, execute the loop code.

Step 4: Increment the counter.

Step 5: Return to Step 2.

The following assembly language program implements the high-level language FOR .. DO statement. Trace the code and identify the five steps to implement the loop construct.

PROCESSOR LOGIC DESIGN

Ques 11) What is processor organisation? Also list the types of processor organisation.

Ans: Processor Organisation

The processor part of computer CPU is sometimes referred to as the data path of the CPU because the processor forms the paths for the data transfers between the registers in the unit.

The various paths are said to be controlled by means of gates that open the required path and close all others. A processor unit can be designed to fulfil the requirements of a set of data paths for a specific application.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organisation of its registers.

Most computers fall into one of the three types of CPU organisations:

1) Single Accumulator Organisation
2) General Register Organisation
3) Stack Organisation

Ques 12) Explain the single accumulator organisation with example.

Or

Draw and explain the block diagram for a 4-bit complete accumulator. **(2019 [06])**

Ans: Single Accumulator Organisation

An accumulator machine uses a single operational register, called an accumulator to hold the result of arithmetic, logical or shift operation. All operations are performed with an implied accumulator register.

The instructions format in this type of computer uses on address field. **For example,** the instruction that specifies an arithmetic addition is defined by an assembly language instruction as

ADD X

Where, X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC + M [X]. AC is the accumulator register and M [X] symbolises the memory word located at address X.

4-Bit Complete Accumulator

An accumulator with n bits requires n stages connected in cascade, with each stage. All control variables, except p_y, must be applied to each stage. The other inputs and outputs in each stage must be connected in cascade to form a complete accumulator. The interconnection among stages to form a complete accumulator is illustrated in the 4-bit accumulator shown in **figure 2.6.** The number on top of each block represents the bit position in the accumulator. All blocks receive eight control variables, pi through pg, and the clock pulses from CP. The other six inputs and four outputs in each block are identical to those of a typical stage, except that subscript i is now replaced by the particular number in each block.

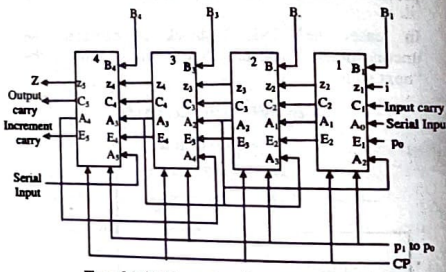


Figure 2.6: 4-bit Accumulator Constructed with Four Stages

The circuit has four B inputs. The zero-detect chain is obtained by connecting the z variables in cascade, with the first block receiving a binary constant 1. The last stage in this chain produces the zero-detect variable Z. The carries for the arithmetic addition are connected in cascade as in full-adder circuits. The serial input for the shift-left operation goes to input A_0, which corresponds to input A_{i-1} in the first stage. The serial input for the shift-right operation goes to input A_y, which corresponds to A_{i+1} in the fourth and last stage. The increment operation is enabled with control variables p_9 in the first stage. The other blocks receive the increment carry from the previous stage.

The total number of terminals in the 4-bit accumulator is 25, including terminals for the A outputs. Incorporating two more terminals for power supply, the circuit can be enclosed within one IC package having 27 or 28 pins. The number of terminals for the control variables can be reduced from nine to four if a decoder is inserted in the IC. In such a case, the IC pin count can be reduced to 22 and the accumulator can be extended to 16 micro-operations without adding external pins.

Ques 13) What is general register organisation? How the Common bus System is helpful in general Register Organisation?

Or

What is control Word? Discuss with diagram.

Ans: General Register Organisation

Figure 2.7 shows the general register organisation. General-purpose registers can be in generally 8-16.

Registers are temporary storage units, made-up of Integrated Circuits (IC's), of a computer that keep data as well as instructions in binary 0 and 1 form.

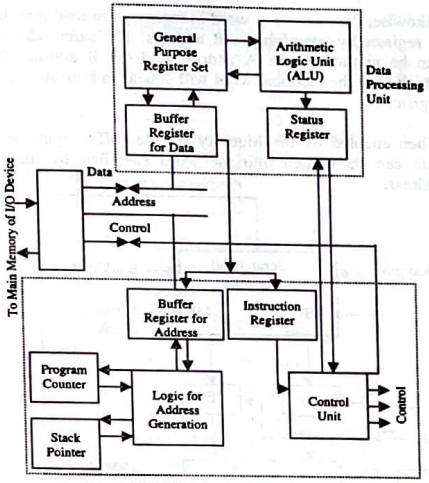


Figure 2.7: CPU with General Register Organisation

ALU performs operation on the data stored in these general-purpose registers and also stores results in these registers only. There are few special-purpose address registers. Two of these are Program Counter (PC) and Stack Pointer (SP).

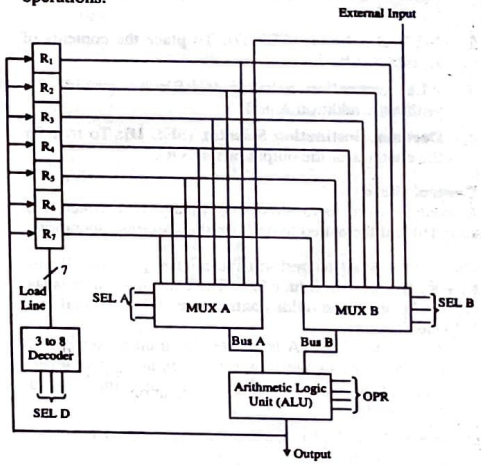
The status register stores the key characteristics or conditions of the result of the last ALU operation. A special simple arithmetic logic unit can be attached as an address generation logic performing the simple fixed-point computations.

The inputs to the control unit are from the instruction register that contains the operation code of the instruction to be executed and from status register which help in generating proper control signals on branch operation. The system bus plays the role of communication media. There are several intermediate buffer registers also which help in intermediate storage of information.

In this organisation of CPU, parallelism can be implemented within the ALU operation or through the overlapped operations of data processing and program control unit. Its registers serve the internal processor memory of a CPU. One of the key differences among various computers is the difference in their register sets. Some computers have very large while some has smaller sets.

Common Bus Organisation

Common bus organisation is a very efficient method of interconnecting all the units of computer system. If a large number of registers are included in processor unit, common bus system is needed. **Figure 2.8** shows the general register organisation. In this organisation, the registers communicate with each other not only for direct data transfers, but also while performing various micro operations.



a) Block Diagram

3	3	3	4
SEL A	SEL B	SEL D	OPR

b) Control Word

Figure 2.8: General Register Organisation

Seven registers are used for general purpose; the output of each register is connected to two Multiplexer (MUXs) inputs. Three select lines are used to select any one of the seven registers and the contents of selected registers are supplied to the inputs of ALU. The buses A and B are used to form the inputs to the common Arithmetic Logic Unit (ALU).

The operation to be performed is selected in the ALU and is used to determine the arithmetic or logic micro-operation by using function selects lines. The result of the micro-operation is available as output data and also goes into the inputs of all the registers.

Any one of the destination register receives the information from the output bus which is selected by a decoder.

To perform the following addition:

$R_3 \leftarrow R_1 + R_2$

The registers R_1 and R_2 are the source registers; the input data are received from these registers and directed to the inputs of ALU. The ALU performs the addition and the result is stored in the destination register, R_3. The operations are controlled by control unit that operates the

CPU bus system which directs the information flow through the registers and ALU by selecting the various components in the system. The control unit must provide binary selection variables to the following selector inputs:

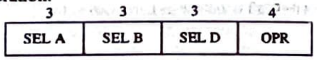
1) **MUX A Selector (SEL A):** To place the contents of R_1 into bus A.
2) **MUX B Selector (SEL B):** To place the contents of R_2 into bus B.
3) **ALU Operation Selector (OPR):** To provide the arithmetic addition A + B.
4) **Decoder Destination Selector (SEL D):** To transfer the contents of the output bus into R_3.

Control Word

A control word is a group of binary bits which are assigned and formatted to perform the specified operation.

The control word to perform the micro-operations $R_3 \leftarrow R_1 + R_2$ is shown in **figure 2.9.** The control word consists of four fields. Three fields contain three bits each and one field has four bits:

1) Three bits of SEL A is allotted for multiplexer A and it selects one of the inputs from eight inputs (seven inputs from seven registers and one input from external input).
2) The SEL B field is used for multiplexer B to select one of the inputs.
3) Three bits of SEL D selects a destination register using the decoder and its seven load outputs.
4) Four bits of OPR selects one of the operations in the ALU. The 13 bit control word when applied to the selection inputs of ALU specify a particular micro-operation.



General format of control word

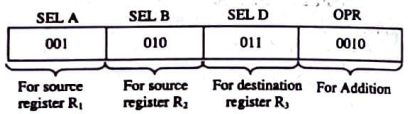


Figure 2.9

Ques 14) Describe processor organisation with diagram using: (2017 [10])

i) Scratchpad memory.

ii) Two port memory.

Or

Mention the advantages of using a scratch pad memory. Draw the diagram of a processor that employs a scratch pad memory and explain the same. (2019 [06])

Ans: Processor Organisation Using Scratchpad Memory and Two Port Memory

The organisation of a processor unit with a 2-port scratchpad memory is shown in **figure 2.10.** The memory has two sets of addresses, one for port A and the other for port B. Data from any word in memory are read into the A register by specifying an A address.

Likewise, data from any word in memory are read into the B register by specifying a B address. The same address can be applied to the A address and the B address, in which case the identical word will appear in both A and B registers.

When enabled by the Memory Enable (ME) input, new data can be written into the word specified by the B address.

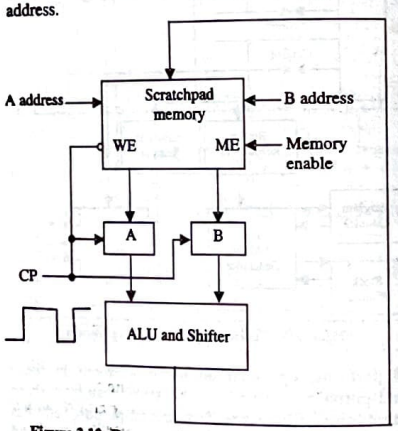


Figure 2.10: Processor Unit with a 2-Port Memory

Thus, the A and B addresses specify two source registers simultaneously, and the B address always specifies the destination register.

Figure 2.10 does not show a path for external input and output data, but they can be included as in previous organisations.

Advantages of Scratch Pad Memory

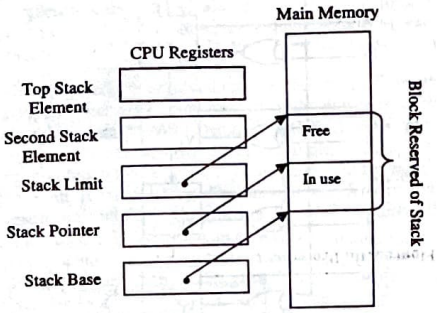
1) Low latency;
2) Smaller than HW cache;
3) Energy-efficient; and
4) Simpler WCET analysis

Ques 15) What is stack organisation? Discuss its implementation with diagram?

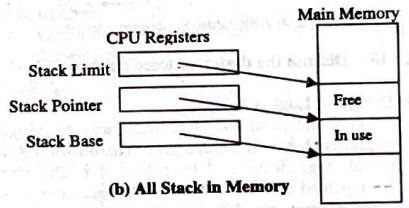
Ans: Stack Organisation

A stack is an ordered set of elements, only one of which can be accessed at a time. The point of access is called the top of the stack. Number of elements in the stack, or length of the stack, is variable. Items may only be added to or deleted from the top of the stack.

For this reason, a stack is also known as a **pushdown list** or a **Last-In-First-Out (LIFO) list.**



(a) Two Top Elements in Registers



(b) All Stack in Memory

Figure 2.11: Typical Stack Organisation

Stack in digital computer is essentially a memory unit with an address register that can count only after an initial value is loaded into it. Stack is a useful structure to provide as part of a CPU implementation. Stacks may also be useful to the programmer. **For example,** expression evaluation.

The implementation of a stack depends in part on its potential uses. If it is desired to make stack operations available to the programmer, then the instruction set will include stack-oriented operations, including PUSH, POP, and operations that use the top one or two stack elements as operands.

Because all of these operations refer to a unique location, namely the top of the stack, the address of the operand or operands is implicit and need not be included in the instruction. These are the zero-address instructions.

If the stack mechanism is to be used only by the CPU, for such purposes as procedure handling, then there will not be explicit stack-oriented instructions in the instruction set. In either case, the implementation of a stack requires that there be some set of locations used to store the stack elements.

The implementation of a stack requires that there be some set of locations used to store the stack elements. A typical approach is illustrated in **figure 2.11.** A contiguous block of locations is reserved in main memory (or virtual memory) for the stack. Most of the time, the block is partially filled with stack elements and the remainder is available for stack growth.

Three addresses are needed for proper operation and these are often stored in processor registers:

1) **Stack Pointer:** Contains the address of the top of the stack. If an item is appended to or deleted from the stack, the pointer is incremented or decremented to contain the address of the new top of the stack.
2) **Stack Base:** Contains the address of the bottom location in the reserved block. If an attempt is made to POP when the stack is empty, an error is reported.
3) **Stack Limit:** Contains the address of the other end of the reserved block. If an attempt is made to PUSH when the block is fully utilized for the stack, an error is reported.

Traditionally, and on most machines today, the base of the stack is at the high address end of the reserved stack block, and the limit is at the low-address end. Thus, the stack grows from higher addresses to lower addresses.

Ques 16) Explain the ALU with suitable diagram.

Ans: Inside a computer, there is an Arithmetic Logic Unit (ALU), which is capable of performing logical operations (e.g. AND, OR, Ex-OR, Invert etc.) in addition to the arithmetic operations (e.g. Addition, Subtraction etc.). The control unit supplies the data required by the ALU from memory, or from input devices, and directs the ALU to perform a specific operation based on the instruction fetched from the memory. ALU is the "calculator" portion of the computer.

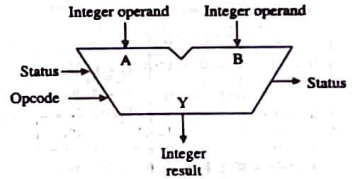


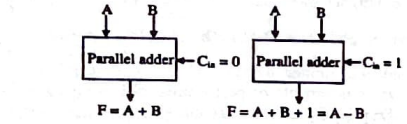
Figure 2.12: Arithmetic Logic Unit (ALU)

An arithmetic logic unit(ALU) is a major component of the central processing unit of the a computer system. It does all processes related to arithmetic and logic operations that need to be done on instruction words. In some microprocessor architectures, the ALU is divided into the arithmetic unit (AU) and the logic unit (LU). Different operation as carried out by ALU can be categorized as follows;

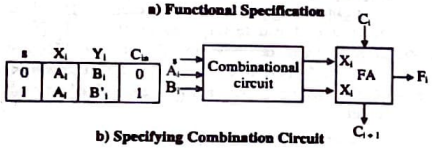
1) **Logical Operations:** These include operations like AND, OR, NOT, XOR, NOR, NAND, etc.
2) **Bit-Shifting Operations:** This pertains to shifting the positions of the bits by a certain number of places either towards the right or left, which is considered a multiplication or division operations.
3) **Arithmetic Operations:** This refers to bit addition and subtraction. Although multiplication and division are sometimes used, these operations are more expensive to make. Multiplication and subtraction can also be done by repetitive additions and subtractions respectively.

Ques 17) Explain the design of arithmetic circuit.
Or
Design an adder/subtractor circuit with one selection variable s and two inputs A and B. When s = 0 the circuit performs A+ B. When s = 1 the circuit performs A – B by taking the 2's complement of B.

Ans: Design of Arithmetic Circuit
The derivation of the arithmetic circuit is illustrated in **figure 2.13**. The function diagram is shown in **figure 2.13 (a)**. For the addition part, we need $C_{in} = 0$. For the subtraction part, we need the complement of B and $C_{in} = 1$. The function table is listed in **figure 2.13 (b)**. When s= 0, X_i and Y_i of each full adder must be equal to the external inputs A_i and B_i, respectively. When s = 1, we must have $X_i = A_i$ and $Y_i = B'_i$. The input carry must be equal to the value of s. The diagram in **figure 2.13 (b)** shows the position of the combinational circuit in one typical stage of the arithmetic circuit.



a) Functional Specification



b) Specifying Combination Circuit

s	A_i	B_i	X_i	Y_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

$X_i = A_i$
$Y_i = B_i \oplus s$
$C_{in} = s$

c) Truth Table and Simplified Equation

Figure 2.13: Derivation of an Adder/Subtractor Circuit

The truth table in **figure 2.13 (c)** is obtained by listing the eight values of the binary input variables. Output X_i is made to be equal to input A_i in all eight entries. Output Y_i is equal to B_i for the four entries when s = 0. It is equal to the complement of B_i for the last four entries where s = 1. The simplified output functions for the combinational circuit are:

$$X_i = A_i$$
$$Y_i = B \oplus s$$

The diagram of the 4-bit adder/subtractor circuit is shown in **figure 2.14**. Each input B_i requires an exclusive-OR gate. The selection variable s goes to one input of each gate and also to the input carry of the parallel adder. The 4-bit adder/subtractor can be constructed with two ICs. One IC is the 4-bit parallel adder and the other is a quadruple exclusive-OR gates.

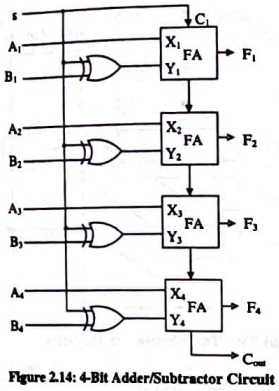
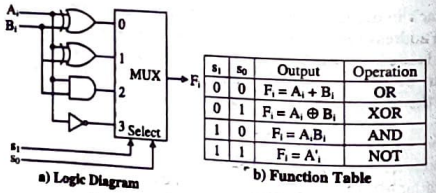


Figure 2.14: 4-Bit Adder/Subtractor Circuit

Ques 18) Discuss the design of logic circuit.

Ans: Design of Logic Circuit
The simplest and most straightforward way to design a logic circuit is shown in **Figure 2.15**. The diagram shows one typical stage designated by subscript i. The circuit must be repeated n times for an n-bit logic circuit. The Four gates generate the four logic operations OR, XOR, AND, and NOT.



a) Logic Diagram

s_1	s_0	Output	Operation
0	0	$F_i = A_i + B_i$	OR
0	1	$F_i = A_i \oplus B_i$	XOR
1	0	$F_i = A_i B_i$	AND
1	1	$F_i = A'_i$	NOT

b) Function Table

Figure 2.15: One Stage of Logic Circuit

The two selection variables in the multiplexer select one of the gates for the output. The function table lists the output logic generated as a function of the two selection variables. The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit. Selection variables s_1 and s_0 can be made common to both sections provided we use a third selection variable, s_2, to differentiate between the two. The outputs of the logic and arithmetic circuits in each stage go through a multiplexer with selection variable s_2. When $s_2 = 0$, the arithmetic output is selected, but when $s_2 = 1$, the logic output is selected. Although the two circuits can be combined in this manner, this is not the best way to design an ALU.

Ques 19) Design an arithmetic and logic unit.

Ans: Design of Arithmetic and Logic Unit
An ALU can be divided into two segments:
1) **Arithmetic Unit:** Arithmetic unit performs typical arithmetic operations. **For example,** addition, subtraction, and increment or decrement by 1.

2) **Logical Unit:** Logical unit perform logical operations. **For example,** less than, equals to and greater than.

Usually, the operands involved may be signed or unsigned integers. In some cases, however, an arithmetic unit must handle 4-bit Binary Coded Decimal (BCD) numbers and floating-point numbers. Design of a simple ALU using typical combinational element such as gates, multiplexers, and a 4-bit parallel adder. For this approach, first an arithmetic unit and a logic unit are designed separately, then they are combined to obtain an ALU:
Step 1: For the first step, a two-function arithmetic unit, as shown in **figure 2.16**, is designed. The key element of this system is a 4-bit parallel adder. The multiplexers select either Y or $\overline{Y}$ for the 3-input of the parallel adder. In particular, if $s_0 = 0$, then B = Y; otherwise B = $\overline{Y}$. Since the selection input (s_0) also controls the input carry (C_{in}), the following results:

If $s_0 = 0$ then F = X plus Y
else F = X plus $\overline{Y}$ plus = X minus Y

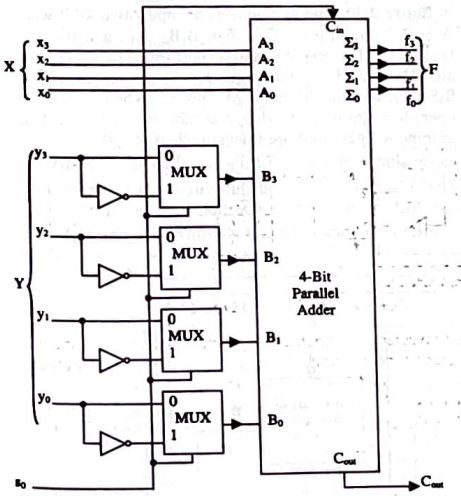


Figure 2.16: Organisation of an Arithmetic Unit

Step 2: For the second step, let us design a two-function logic unit; this is shown in **figure 2.17**:

From **figure 2.17**, it can be seen that when $s_0 = 0$, the output G = X AND Y; otherwise the output G = X ⊕ Y. Note that from these two Boolean operations, other operations such as NOT and OR can be derived by the following Boolean identities:

$1 \oplus x = \overline{x}$
$x \text{ OR } y = x \oplus y \oplus xy$

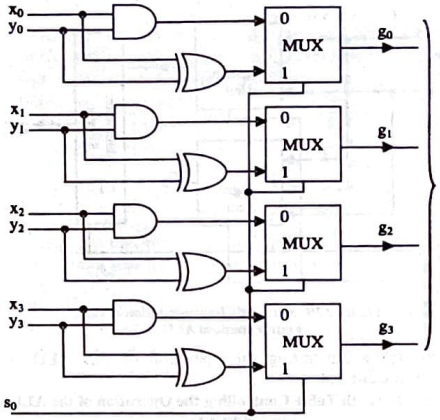


Figure 2.17: Organisation of a 4-Bit Two-Function Logic Unit

Therefore, NOT, and OR operations can be obtained by using additional hardware and the circuit of **figure 2.18**.

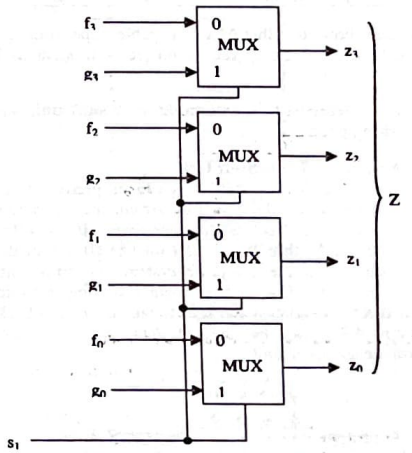


Figure 2.18: Combining the Outputs Generated by the Arithmetic and Logic Units

The outputs generated by the arithmetic and logic units can be combined by using a set of four multiplexers, as shown in **figure 2.18**. From this organisation, it can be seen that when the select line $s_1 = 1$, the multiplexers select outputs generated by the logic unit; otherwise, the outputs of arithmetic unit are selected. More commonly, the select line, s_1, is referred to as the mode input since it selects the desired mode of operation (arithmetic or logic). A complete block diagram schematic of this ALU is shown in above **figure 2.19**.

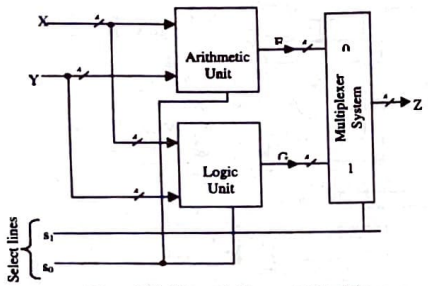


Figure 2.19: Schematic Representation of the Four-Functions ALU

Truth table illustrating the operation of this ALU is shown in **table 2.4**:

Table 2.4: Truth Table Controlling the Operation of the ALU of figure 5.18

Select Lines		Output Z	Comment
s_1	s_0		
0	0	X plus Y	Addition
0	1	X plus Y plus 1	Two Complement-Subtraction
1	0	X ∧ Y	Boolean AND
1	1	X ⊕Y	Exclusive-OR

This table shows that this ALU is capable of performing 2 arithmetic and 2 logic operations on the 4-bit operands X and Y.

Ques 20) Describe the arithmetic logic shift unit and its working process.

Ans: Arithmetic Logic Shift Unit

The Arithmetic Logical shift Unit can be perform three types of operations these are as Arithmetic operation, Logical operation and Shift operations. It is often abbreviated as **Arithmetic Logic Unit (ALU).** All of the other elements of the computer system – control unit, register, memory, I/O – are there mainly to bring data into the ALU for it to process and then to take the results back out. An ALU can be divided into two segments: **arithmetic** and **logic unit.**

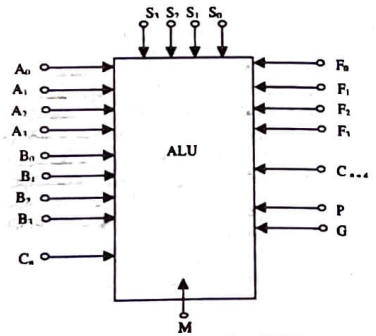


Figure 2.20: Representation of ALU

This circuit performs all basic operation. ALU consist of Arithmetic circuit, Logical circuit and Shift circuit. It is consider that every basic circuit can perform some function as Arithmetic circuit perform 8- operations, Logical circuit perform 16- operations and Shift circuit perform 2- operations. The ALU take the input from registers and perform then transfer to the destination register. All the operations done by ALU are performed in a single clock pulse. But Shift operation is not performed according to clock pulse.

To perform a micro-operation, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register. The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period. The shift micro-operations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.

Functionally, the operation of a typical ALU is represented in **figure 2.20**. The unit accepts as inputs two 4-bit words $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$ and, a carry input labeled C_n. The operations to be performed on these inputs are determined by the logic levels on the input $S = S_3S_2S_1S_0$ and on the input M (mode). When $M = 1$, the operations are logical, while a combination of logical and arithmetic operations are selected when $M = 0$. The output are available at $F = F_3F_2F_1F_0$ and the carry output C_{n+4}. The arithmetic, logic, and shift circuits are combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in **figure 2.21.**

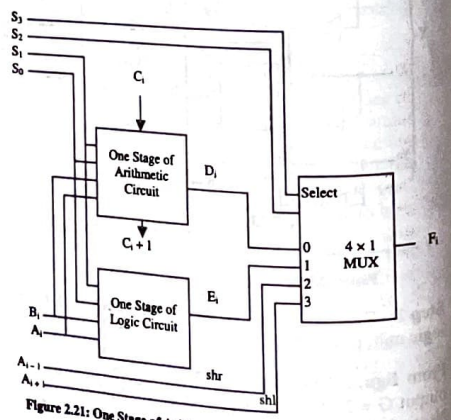


Figure 2.21: One Stage of Arithmetic Logic Shift Unit

The subscript i designates a typical stage. Inputs A_i and B_i are applied to both the arithmetic and logic units. A particular micro-operation is selected with inputs S_1 and S_0. A 4 × 1 multiplexer at the output chooses between an arithmetic output in E_i and a logic output in H_i.

The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.

Note: The diagram shows just one typical stage.

The circuit must be repeated n times for an n-bit ALU. The output carries C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence. The input carry to the first stage is the input carry C_{in}, which provides a selection variable for the arithmetic operations. The circuit whose one stage is specified in **figure 2.20** provides eight arithmetic operation, four logic operations, and two shift operations. Each operation is selected with the five variables S_3, S_2, S_1, S_0, and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only.

Table 2.5 lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with $S_3S_2 = 00$. The next four are logic operations and are selected with $S_3S_2 = 01$. The input carry has no effect during the logic operations and is marked with don't-care X's. The last two operations are shift operations and are selected with $S_3S_2 = 10$ and 11. The other three selection inputs have no effect on the shift.

Table 2.5: Function Table: Arithmetic Logic Shift Unit

S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	F= A	Transfer A
0	0	0	0	1	F= A + 1	Increment A
0	0	0	1	0	F= A + B	Addition
0	0	0	1	1	F= A + B + 1	Add with carry
0	0	1	0	0	F= A + $\overline{B}$	Subtract with borrow
0	0	1	0	1	F= A + $\overline{B}$ + 1	Subtraction
0	0	1	1	0	F= A – 1	Decrement A
0	0	1	1	1	F= A	Transfer A
0	1	0	0	x	F= A ∧ B	AND
0	1	0	1	x	F= A ∨ B	OR
0	1	1	0	x	F= A ⊕ B	XOR
0	1	1	1	x	F= $\overline{A}$	Complement A
1	0	x	x	x	F= shr A	Shift right A into F
1	1	x	x	x	F= shl A	Shift left A into F

An ALU performs the simple arithmetic-logic and shift operations. Complexity of an ALU depends on the type of instruction set which has been realised for it. The simple ALU can be constructed for fixed point numbers, on the other hand the floating point arithmetic implementation require more complex control logic and data processing capabilities, i.e., the hardware.

Several microprocessor families utilises only fixed point arithmetic capabilities in the ALU and for floating point arithmetic or other complex functions they may utilise an auxiliary special purpose unit. This unit is called **Arithmetic Coprocessor.**

Ques 21) Design a 4bit Arithmetic Module which performs the following operations on two inputs. A and B, controlled by selection variables s_1, and s_0, and input carry C_{in}: (2017 [10])

S_1	S_0	$C_{in} = 0$	$C_{in} = 1$
0	0	F = A	F = A + 1
0	1	F = A + B	F = A + B + 1
1	0	F = A + B'	F = A + B' + 1
1	1	F = A – 1	F = A

Ans: Arithmetic Circuit Design

The circuit can be designed by following the below given steps:

Step 1) Analyse the Circuit:

Use $G = A + Y + C_{in}$

For S_1 and $S_0 = 00$, then

$G = A + 0 + 0$

$G = A$

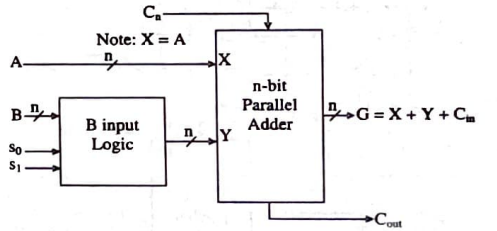


Figure 2.22: Arithmetic Circuit Design

For Example, We can verify for n = 4bit:

A = 1010, B = 0101

Function Table for Arithmetic Circuit

Select		Input	G = A + Y + C_{in}	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0's	G = A (Transfer)	G = A + 1 (increment)
0	1	B	G = A + B (add)	G = A + B + 1
1	0	$\overline{B}$	G = A + $\overline{B}$	G = A + $\overline{B}$ + 1 (subtract)
1	1	All 1's	G = A – 1 (decrement)	G = A (transfer)

Step 2) Building the B Input Logic

Select		Input
S_1	S_0	Y
0	0	all 0's
0	1	B
1	0	$\overline{B}$
1	1	All 1's

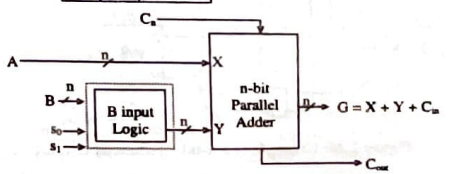


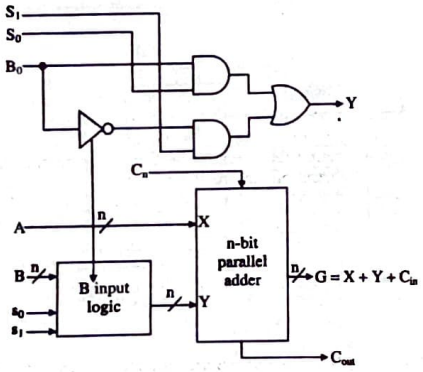
Figure 2.23: Building the B Input Logic

Input = S_1, S_0 and B Output = Y

Inputs			Output	
S_1	S_0	B_i		Y_i
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B}_i$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	

i) Obtain the K-Map.
ii) Get the Boolean Expression.

$Y = BS_0 + \overline{B}\, S_1$



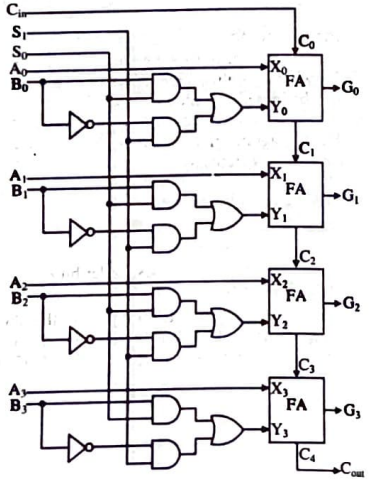


Figure 2.24: Example of a 4-bit Arithmetic Circuit

Step 3) Use Multiplexer

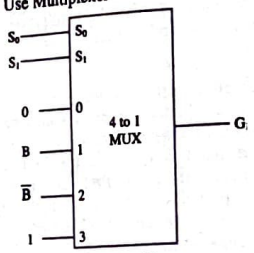


Figure 2.25

Step 4) Building the Logic Circuit-2
i) The logic circuit performs bitwise operation.
ii) Commonly–AND, OR, XOR and NOT.

S_1	S_2	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

Note: If 4 bit is wanted, then we have to arrange it in an array.

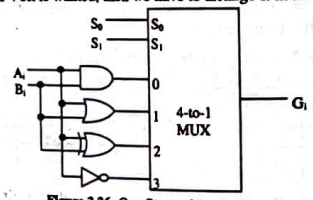


Figure 2.26: One Stage of Logic Circuit

Step 5) Building the Selector for Choosing Arithmetic or Logic Unit

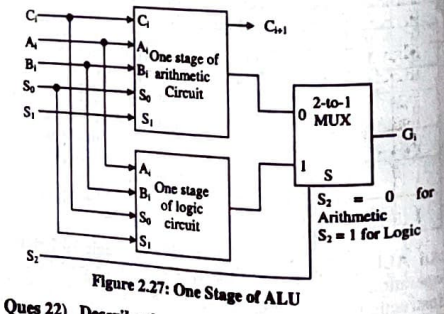


Figure 2.27: One Stage of ALU

Ques 22) Describe the status register. **(2017 [05], 2018 [10])**

Ans: Status Register
The status register also called **flag register**, holds 1-bit flags to indicate certain conditions that arise during arithmetic and logical operations. A flag is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU. The flags register is the status that contains the current state of the processor.

The flag register contains nine active flags as shown in the **figure 2.28.**

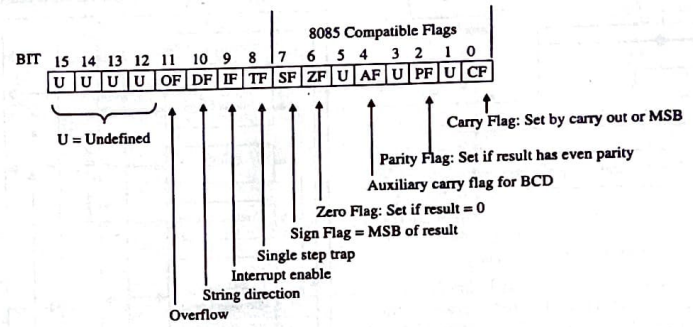


Figure 2.28: 8086 Flag Register Bit Pattern

Six of them are used to indicate some condition produced by instruction:

1) **Carry Flag (CF):** Indicates whether there is overflow or not.
2) **Parity Flag (PF):** It is set to 1 if result of byte operation or lower byte of the word operation contains an even number of ones; otherwise it is zero.
3) **Auxiliary Flag (AF):** This flag is set if there is an overflow out of bit, 3 i.e., carry from lower nibble to higher nibble (D_3 bit to D_4 bit). This flag is used for BCD operations and it is not available for the programmer.
4) **Zero Flag (ZF):** The zero flag sets if the result of operation in ALU is zero and flag resets if the result is non-zero. The zero flag is also set if a certain register content becomes zero following an increment or decrement operation of that register.
5) **Sign Flag (SF):** After the execution of arithmetic or logical operations, if the MSB of the result is 1, the sign bit is set. Sign bit 1 indicates the result is negative; otherwise it is positive.
6) **Overflow Flag (OF):** This flag is set if result is out of range. For addition this flag is set when there is a carry into the MSB and no carry out of the MSB or *vice versa*.

   For subtraction, it is set when the MSB needs a borrow and there is no borrow from the MSB, or *vice versa*.

Ques 23) What are condition codes? List the different condition codes. **(2017-18 [03])**

Ans: Condition Code
The condition code flags are used to store the results of certain condition when certain operations are performed during execution of the program. The condition code flags are stored in the status registers. The status register is also referred to as flag register.

ALU operations and certain register operations may set or reset one or more bits in the status register. Status bits lead to a new set of microprocessor instructions. These instructions permit the execution of a program to change flow on the basis of the condition of bits in the status register. So the condition bits in the status register can be used to take logical decision within the program. Some of the common condition code flags are:

1) Carry/Borrow
2) Zero
3) Negative/Sign
4) Auxiliary Carry
5) Overflow flag
6) Parity

Or

Ques 24) Discuss about condition code bits in a 4-bit status register. **(2019 [04])**

Ans: 4-Bit Status Register
It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions can be stored for further analysis. Status bits are also called condition-code bits or flag bits. **Figure 2.29** shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolised by C, S, Z and V. The bits are set or cleared as a result of an operation performed in the ALU:

1) Bit C (carry) is set to 1 if the end carry C_8 is 1. It is cleared to 0 if the carry is 0.
2) Bit S (sign) is set to 1 if the highest-order bit F_7 is 1. It is set to 0 if the bit is 0.
3) Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise. In other words, Z=1 if the output is zero and Z = 0 if the output is not zero.
4) Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement. For the 8-bit ALU,
   V= 1 if the output is greater than + 127 or less than– 128.

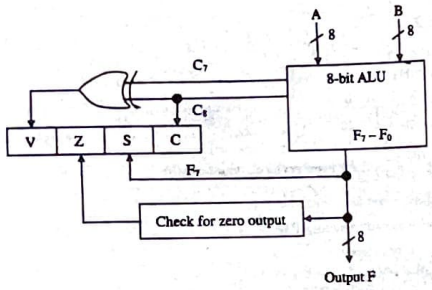


Figure 2.29 Status Register Bits

The status bits can be checked after an ALU operation to determine certain relationships that exist between the values of A and B. If bit V is set after the addition of two signed numbers, it indicates an overflow condition.

If Z is set after an exclusive-OR operation, it indicates that A = B. This is so because $x \oplus x = 0$, and the exclusive-OR of two equal operands gives an all-0's result which sets the Z bit.

A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z status bit.

For example, let A = 101x1100, where x is the bit to be checked. The AND operation of A with B = 00010000 produces a result 000x0000. If x = 0, the Z status bit is set, but if x = 1, the Z bit is cleared since the result is not zero. The AND operation can be generated with the TEST instruction.

Ques 25) Define processor unit. Draw internal architecture of processor? What are the different units of processor?

Ans: Processor Unit
Alternately referred to as a CPU, central processor, or microprocessor, is the Central Processing Unit of the computer. A computer's processor/CPU handles all instructions it receives from hardware and software running on the computer.

Central Processing Unit (CPU) is the brain of a computer. It is a part of the computer that performs the bulk of data processing. Its primary function is to execute programs. Program, which is to be executed, is stored in the main memory.

All major calculations and comparisons performed by a computer are carried-out inside its CPU. The CPU is also responsible for activating and controlling the operation of other units of the computer system.

Hence no other single component of a computer determines its overall performance as much as the CPU.

CPU Architecture
Figure 2.30 shows the overall architecture of CPU:

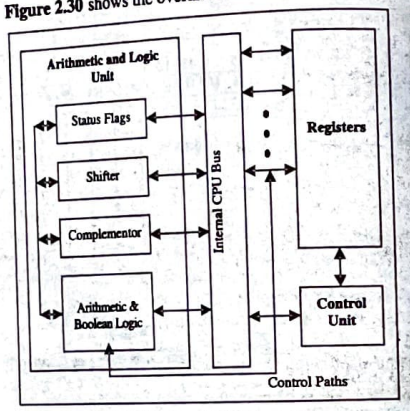


Figure 2.30: Internal Architecture of CPU

This architecture consists of the following **units**:

1) **Arithmetic and Logical Unit (ALU):** ALU is responsible for all the arithmetic (e.g., addition, subtraction, etc.) and logical operations (e.g., comparisons, decisions, etc.) inside the computer under the supervision of Control Unit (CU). ALU contains the necessary hardware circuitry for performing all kinds of arithmetic and logical operations.

2) **Control Unit:** Control unit controls the movement of data and instructions into and out of the CPU and controls the operation of the ALU. The main responsibilities of control units are as below:
   i) Data exchange of CPU with the memory or I/O modules
   ii) Internal operations in the CPU such as:
      a) Moving data between registers (register transfer operations),
      b) Making ALU to perform a particular operation on the data, and
      c) Regulating other internal operations.

3) **Registers:** A CPU contains a number of registers to store data temporarily during the execution of a program. The number of registers differs from processor to processor.

Arrangement of hardware components in a computer is specified by the number of registers it contains, as well as by various micro-operations it can perform on the data stored temporarily in the registers.

Ques 26) Write the short note on design of shifter.

Ans: Shifter
Shifters move bits and multiply or divide by powers of 2. As the name implies, a shifter shifts a binary number left or right by a specified number of positions.

There are several kinds of commonly used shifters:
1) **Logical shifter**- shifts the number to the left (LSL) or right (LSR) and fills empty spots with 0's.

   Example: 11001 LSR 2=00110; 11001 LSL 2=00100

2) **Arithmetic shifter-** is the same as a logical shifter, but on right shifts fills the most significant bits with a copy of the old most significant bit (msb). This is useful for multiplying and dividing signed numbers

   Example: 11001 ASR 2=11110; 11001 ASL 2=00100

Ques 27) What is accumulator? Discuss the design of the accumulator. Or
What is the control of AC Register? How adder and logic unit are associated with accumulator.

Ans: Accumulator Register
It is used for arithmetic and other data operations. An accumulator (AC) is a register in which intermediate arithmetic and logic results are stored. The accumulator is a register, which holds one of the operands prior to the execution of an instruction, and receives the result of most of the arithmetic and logical operations. So an accumulator is the most frequently used register. Some CPUs have a single accumulator; some have several accumulator.

Without an accumulator register, it would be necessary to write the result of each calculation (addition, multiplication, shift, etc.) to main memory, perhaps only to be read right back again for use in the next operation. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register. Early electronic computer systems were often split into two groups, those with accumulators and those without.

Modern computer systems often have multiple general purpose registers that operate as accumulators, and the term is no longer as common as it once was. However, a number of special-purpose processors still use a single accumulator for their work, in order to simplify their design.

Design of Accumulator
The circuits associated with the AC register are shown in **figure 2.31:**

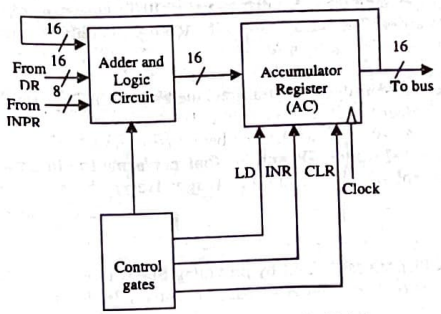


Figure 2.31: Circuits Associated with AC

The adder and logic circuit has three sets of inputs. One set of 16 inputs comes from the outputs of AC. Another set of 16 inputs comes from the data register DR.

A third set of eight inputs comes from the input register INPR.

The outputs of the adder and logic circuit provide the data inputs for the register. In addition, it is necessary to include logic gates for controlling the LD, INR, and CLR in the register and for controlling the operation of the adder and logic circuit.

In order to design the logic associated with AC, it is necessary to go over the register transfer statements in **table below** and extract all the statements that change the content of AC.

D_0T_5	$AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5	$AC \leftarrow AC + DR$	Add with DR
D_2T_5	$AC \leftarrow DR$	Transfer from DR
pB_{11}	$AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9	$AC \leftarrow \overline{AC}$	Complement
rB_7	$AC \leftarrow shr\ AC, AC(15) \leftarrow E$	Shift right
rB_6	$AC \leftarrow shl\ AC, AC(0) \leftarrow E$	Shift left
rB_{11}	$AC \leftarrow 0$	Clear
rB_5	$AC \leftarrow AC + 1$	Increment

Control of AC Register
The gate structure that controls the LD, INR, and CLR inputs of AC is shown in **figure 2.32.**

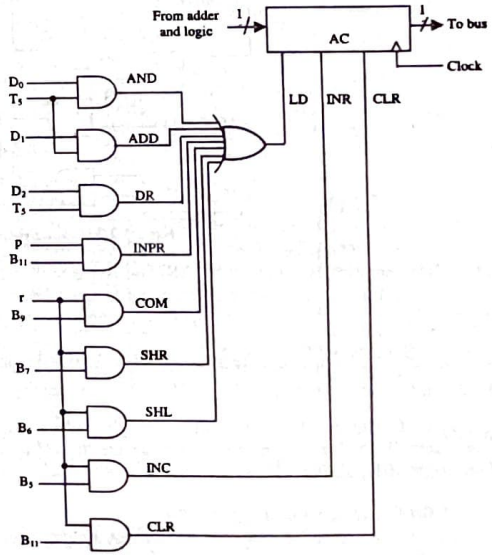


Figure 2.32: Gate Structure for Controlling the LD, INR, and CLR of AC

The gate configuration is derived from the control functions. The control function for the clear micro-operation is rB_{11}, where $r = D_7I'T_3$ and $B_{11} = IR(11)$. The output of the AND gate that generates this control function is connected to the CLR input of the register. Similarly, the output of the gate that implements the increment micro-operation is connected to the INR input of the register. The other seven micro-operations are generated in the adder and logic circuit and are loaded into AC at the proper time. The output of the gates for each control function is marked with a symbolic name. These outputs are used in the design of the adder and logic circuit.

Adder and Logic Unit

The adder and logic circuit can be subdivided into 16 stages, with each stage corresponding to one bit of AC. T The register has a JK flip-flop, two OR gates, and two AND gates in its each stage. The load (LD) input is connected to the inputs of the AND gates.

Figure 5.30 shows one such AC register stage (with the OR gates removed). The input is labeled I_i, and the output AC(i). When the LD input is enabled, the 16 inputs I_i for i = 0, 1, 2,..., 15 are transferred to AC (0-15). One stage of the adder and logic circuit consists of seven AND gates, one OR gate and a Full-Adder (FA), as shown in **figure 2.33**. The inputs of the gates come from the outputs of gates that controls the LD, INR, and CLR inputs of AC register.

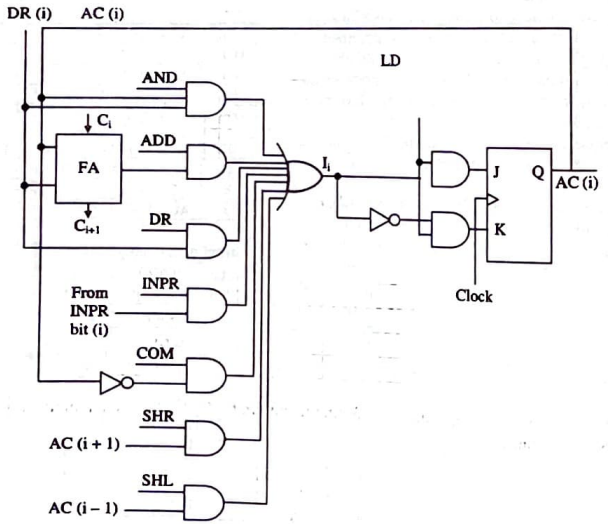


Figure 2.33: One Stage of Adder and Logic Circuit

The AND operation is achieved by ANDing AC(i) with the corresponding bit in the data register DR(i). One stage of the adder uses a full-adder with the corresponding input and output carries. The transfer from INPR to AC is only for bits 0 through 7. The complement micro-operation is obtained by inverting the bit value in AC

The shift-right operation transfers the bit from AC(i + 1), and the shift-left operation transfers the bit from AC(i – 1). The complete adder and logic circuit consists of 16 stages connected together.

Ques 28) Design a 4-bit combinational logic shifter with 2 control signals H_1 and H_0 that performs the following operations (bit values given in parenthesis are the values of control variables H_1 and H_0 respectively) – No shift (00), Shift-right (01), Shiftleft (10), Transfer 0's to S (11). **(2019 [05])**

Ans: 4-Bit Combinational Logic Shifter

The ALU designed in the previous section may be upgraded to CPU/Processor Unit by providing Status register, Shifter register. Set of general purpose register, register selection logic for first operand A of ALU, register selection logic for second operand B of ALU and selection logic for destination register.

Status register stores the value of flags after arithmetic/logic operation performed by ALU. When carry is generated carry flag C is set. When output of ALU becomes 0 then the 0 flag Z is set, When MSB of ALU output is one sign flag S is set.

When the result of computation by ALU exceeds then its range overflow flag V is set. Combinational circuit showing setting of these flags is shown in **figures 2.34**.

Sometimes output of ALU needs to be shifted. For this purpose a shifter is connected at the output of ALU that perform shifting or other operation like transfer or clear in accordance with its control signals. Let us design 4-bit shifter whose function table is given in **table 2.34.**

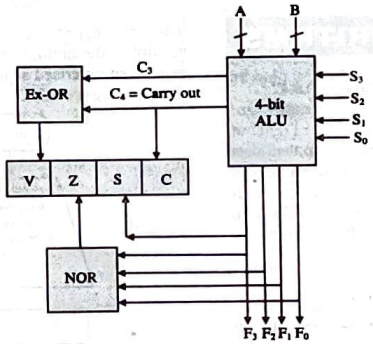


Figure 2.34: ALU with Status Register

Table 2.6: Register Function Table

H_1	H_0	Function
0	0	No shifting
0	1	Shift left
1	0	Shift right
1	1	Clear to 0

This shifter can be designed using multiplexer. Shifter inputs are F_0, F_1, F_2, F_3 as data input and H_1, H_0 as control inputs. S. o, S_1, S_2 and S_3 are outputs.

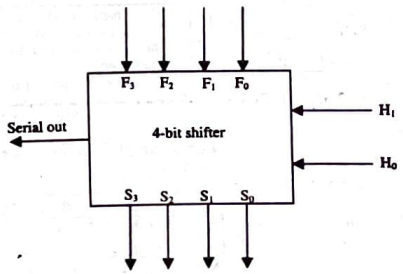


Figure 2.35: Shift Block Diagram.

Module 3

Arithmetic Algorithms

ARITHMETIC ALGORITHMS

Ques 1) What is meant by arithmetic algorithms?

Ans: Arithmetic Algorithms
Arithmetic instructions in digital computers manipulate data to produce results necessary for the solution of computational problems. These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in a computer. The four basic arithmetic operations performed by computers are:

1) Addition 2) Subtraction
3) Multiplication 4) Division

From these four basic operations, it is possible to formulate other arithmetic functions and solve scientific problems by means of numerical analysis methods. An arithmetic processor is the part of a processor unit that executes arithmetic operations. Performing the basic arithmetic operations in signed-magnitude representation is valuable when the operations are to be implemented by hardware. However, the designer must be thoroughly familiar with the sequence of steps to be followed in order to carry out the operation and achieve a correct result. The solution to any problem that is stated by a finite number of well-defined procedural steps is called an **algorithm**.

An algorithm is a sequence of finite instructions, often used for calculation and data processing. An algorithm is an effective method for solving a problem using a finite sequence of instructions. An algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state.

Arithmetic operations can be divided into two forms:
1) Fixed Point Numbers, and
2) Floating Point Numbers.

Ques 2) Define binary multiplication. Describe the multiplication of the positive numbers.

Ans: Binary Multiplication
Multiplication of binary numbers can be performed in the same way as multiplication of decimal numbers. However, in this case, 2 (radix of binary number system) plays the role of 10 (radix in decimal number system).

Multiplication of Positive Numbers
Multiplying a decimal number by 10 means, in effect, shifting the multiplicand to left by one place and appending a zero at its end. Analogous to this, in binary multiplication, shifting the multiplicand to left by one place and appending a zero at its right end, effectively multiplies the multiplicand by 2. The binary multiplication may be summarised as follows in the form of flow chart as shown in **figure 3.1**:

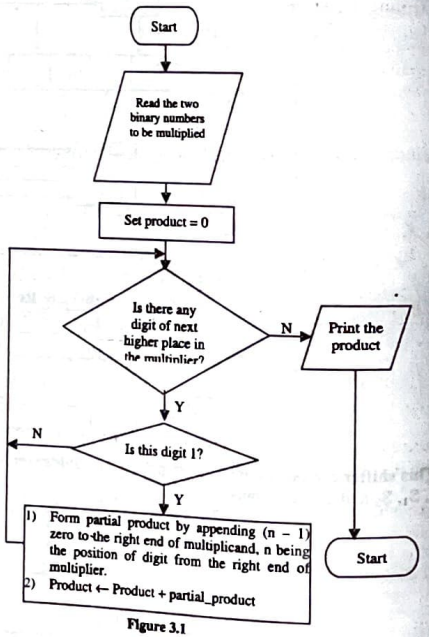


Figure 3.1

For example, let consider multiplying $(1010111)_2$ with $(1001)_2$.

Binary Multiplication	
	1010111 × 1001
Multiply 1010111 with 1 of 1001 (digit at unit's place).	1010111
Multiply 1010111 with 0 of 1001 (digit at ten's place); left shift padding 0 at the end.	00000000
Multiply 1010111 with 0 of 1001 (digit at next higher place); left shift padding 0 at the end.	000000000
Multiply 1010111 with 1 of 1001(digit at next higher place); left shift padding 0 at the end.	1010111000
Add the partial products.	1100001111

Ques 3) Multiply $(1010111)_2$ with $(1111)_2$.

Ans: Binary Multiplication

	1010111 × 1111
Multiply 1010111 with 1 of 1111 (digit at unit's place).	1010111
Multiply 1010111 with 1 of 1111 (digit at ten's place); left shift padding 0 at the end.	10101110
Multiply 1010111 with 1 of 1111 (digit at next higher place); left shift padding 0 at the end.	101011100
Multiply 1010111 with 1 of 1111 (digit at next higher place); left shift padding 0 at the end.	1010111000
Add the partial products.	10100011001

Ques 4) Multiply $(11)_2$ by $(11)_2$.

Ans:

	2^3	2^2	2^1	2^0	
			1	1	3
	×		1	1	× 3
	1	1	1	1	9
Carry			1	1	
	1	0	0	1	

Ques 5) Multiply $(1001)_2$ by $(10)_2$.

Ans:

	2^4	2^3	2^2	2^1	2^0	
		1	0	0	1	9
Carry		×		1	0	× 2
		0	0	0	0	18
	1	0	0	1		
	1	0	0	⌟	0	

Ques 6) Multiply the following binary numbers: 111 and 101.

Ans: Multiplication of 111 and 101 is as follows:

			1	1	1	(= 7)
			1	0	1	(= 5)
			1	1	1	
		0	0	0	×	
	1	1	1	×	×	
1	0	0	0	1	1	(= 35)

1 + 1 = 10 1 + 1 = 10 1 + 1 = 10 1 + 0 = 1
carry 1 carry 1

Ques 7) Multiply the following binary numbers: 101.11_2 and 110.01_2

Ans:

```
      101 11
    ×110 01
    -------
      101 11
   101110 00
+1011100 00.
------------
100011.1111₂
```

Ques 8) Multiply the following binary numbers: 111 and 101.

Ans:

			1	1	1	(= 7)
			1	0	1	(= 5)
			1	1	1	
		0	0	0	×	
	1	1	1	×	×	
1	0	0	0	1	1	(= 35)

1 + 1 = 10 1 + 1 = 10 1 + 1 = 10 1 + 0 = 1
carry 1 carry 1

Ques 9) Explain the concept of binary division with the help of suitable example.

Ans: Binary Division
Division in the binary number system is very simple. As is prevalent in the decimal or any other number system, division by zero is meaningless. Following are the conditions which are followed by the binary division:

0 + 1= 0, 1 + 1= 1

The rules for the division process in the binary system are the same as that in decimal division:

Step 1) Start from left of the dividend.
Step 2) A series of subtractions are performed where the divisor is subtracted from the dividend.
Step 3) In this step put 1 in the quotient if subtraction is possible. Then from the matching digits of the dividend, divisor is subtracted.
Step 4) In the quotient, 0 is saved when divisor is greater than the remainder (or subtraction is not possible).
Step 5) In this step, to add in the remainder digits, the next digit is brought. Then in the same manner it is continued as in long division.

For example, let consider the division of 101100 by 100

```
100 ) 101100 ( 1011   (Quotient)
      - 100
      -----
        110
      - 100
      -----
         100
      - 100
      -----
         000          (Remainder)
```

Step 1) Since 100 contain three digits, so 100 are multiplied by 1 and result 100 is written below the first three digits 101. The difference so obtained is 1.
Step 2) Now is 1 brought-down giving 11. Since, it is not divisible by 100 so a '0' is written in the quotient. Again 0 is brought-down making the number 110. Multiplication of 100 by 1 gives 100 which is written below 110. The subtraction gives 10. 1 is written in the third place in the quotient.
Step 3) The last 0 is brought-down to give 100. Since, 100 × 1 = 100. So, 1 is written in the fourth place of quotient and 100 below 100 giving the difference as 000.

Thus, the result of division = 1011_2.
(This is equivalent to 44/4 = 11 in the decimal system.)

Ques 10) Divide the 100001_2 by 110_2.

Ans:

```
      0101      (Quotient)
110 ) 100001    (Dividend)
      110       1← If divisor greater than 100, then put 0 in quotient
      1000      2← Add digit from dividend to group used above
                3← Subtraction possible, hence, put 1 in quotient
       100      4← Remainder from subtraction plus digit from dividend
       110      5← If divisor greater then put 0 in quotient
       1001     6← Add digit from dividend to group used above
       110      7← Subtraction possible, hence, put 1 in quotient
        11      (Remainder)
```

To verify the result divide the 33_{10} (100001_2) by 6_{10} (110_2), which gives a quotient of $5_{10}(101_2)$ and a remainder of 3_{10} (11_2).

Ques 11) Divide 11001 by 101

Ans:

Decimal	Binary
5	101
5) 25	101) 11001
	101
	101
	101
	—

Ques 12) Divide 11101.00 by 1100

Ans:

Decimal	Binary
2.416...	10.011010101...
12) 29.0000	1100) 11101.00
24	1100
50	10100
48	1100
20	10000
12	1100
80	10000
72	1100
8	—
	...

Ques 13) Divide 101101_2 (dividend) by 110_2 (divisor).

Ans:

```
      111.1
110 ) 101101
      110
      1010
      110
      1001
      110
       110
       110
       ---
```

Ques 14) Divide, 10000111 by 101.

Ans:

```
          11011 = 27 (base 10)
101 ) 10000111 = 135 (base 10)
      - 111    = 5 (base 10)
        ---
        110
      - 101
        ---
          11
        -  0
          ---
          111
        - 101
          ---
           101
         - 101
           ---
             0
```

Ques 15) Describe the restoring and non-restoring division method. Write its algorithm and give example.

Or

Non-restoring division is faster than restoring division. Justify the statement. (2019 [03])

Or

Divide $(1000)_2$ by $(11)_2$ using restoring division method. (2019 [04])

Ans: Restoring Division Method

Figure 3.2 shows a logic circuit arrangement that implements restoring division. An n-bit positive divisor is loaded into register M and an n-bit positive dividend is loaded into register Q at the start of the operation. Register A is set to 0. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The required subtractions are facilitated by using 2's-complement arithmetic. The extra bit position at the left end of both A and M accommodates the sign bit during subtractions.

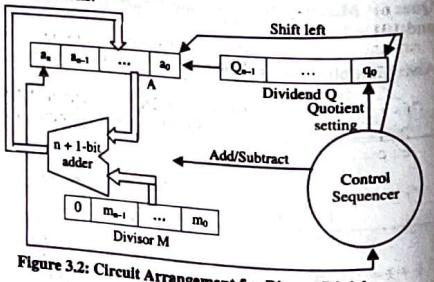


Figure 3.2: Circuit Arrangement for Binary Division

The following **algorithm** performs restoring division (Do the following n times):

Step 1) Shift A and Q left one binary position.

Step 2) Subtract M from A, and place the answer back in A.

Step 3) If the sign of A is 1, set q_0 to 0 and add M back to A (i.e., restore A); otherwise, set q_0 to 1.

Figure 3.3 shows a 4-bit example as it would be processed by the circuit:

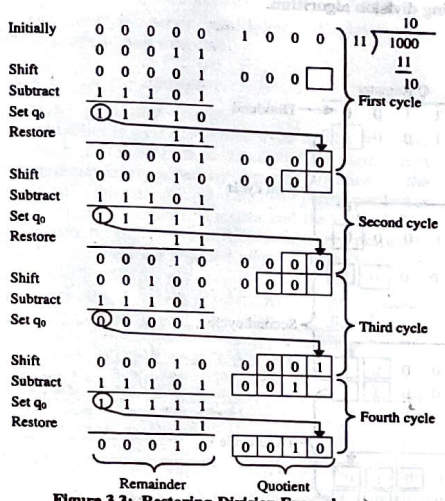


Figure 3.3: Restoring-Division Example

Figure 3.3 shows how the division example in **figure 3.4** is executed by the non-restoring division algorithm.

There are no simple algorithms for directly performing division on signed operands that are comparable to the algorithms for signed multiplication.

In division, the operands can be pre-processed to transform them into positive values. After using one of the algorithms, the results are transformed to the correct signed values, as necessary.

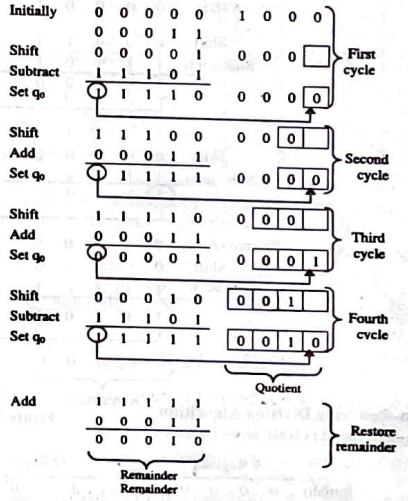


Figure 3.4: Non-Restoring-Division Example

Non-Restoring Division

The restoring-division algorithm can be improved by avoiding the need for restoring A after an unsuccessful subtraction. Subtraction is said to be unsuccessful if the result is negative. Therefore, **Non-Restoring Algorithm is faster than restoring Division.** Consider the sequence of operations that takes place after the subtraction operation in the preceding algorithm.

If A is positive, one shift left and subtract M, i.e., he/she perform 2A – M. If A is negative, he/she restore it by performing A + M, and then shift it left and subtract M.

This is equivalent to performing 2A + M. The q_0 bit is appropriately set to 0 or 1 after the correct operation has been performed.

One can summarise this in the following **algorithm** for non-restoring division:

Step 1) Do the following n times:

i) If the sign of A is 0, shift A and Q left one bit, position and subtract M from A; otherwise, shift A and Q left and add M to A.

ii) Now, if the sign of A is 0, set q_0 to 1; otherwise, q_0 to 0.

Step 2) If the sign of A is 1, add M to A.

Step 2 is needed to leave the proper positive remainder in A at the end of the n cycles of **Step 1.** The logic circuitry in **Figure 3.4** can also be used to perform this algorithm. Note that the Restore operations are no longer needed, and that exactly one Add or Subtract operation is performed per cycle.

Ques 16) What is difference between restoring and non-restoring method?

Ans: Difference between Restoring and Non-Restoring Method

Table 3.1 shows the difference between restoring and non-restoring method:

Table 3.1: Difference between Restoring Division Method and Non-Restoring Algorithm

Restoring	Non-Restoring
Needs restoring of register A if the result of subtraction is negative.	Does not need restoring.
In each cycle content of register A is first shifted left and then divisor is subtracted from it.	In each cycle content of register A is first shifted left and then divisor is added or subtracted with the content of register A depending on the sign of A.
Does not need restoring of remainder.	Needs restoring of remainder if remainder is negative.
Slower algorithm.	Faster algorithm.

Ques 17) Perform 1100 / 11 using restoring and non-restoring division algorithm.

Ans: Restoring Division Algorithm

Q ← Dividend (1100), B ← Divisor (11).

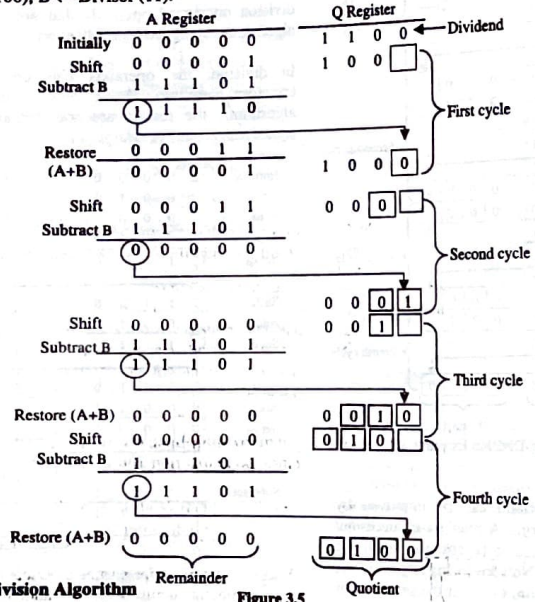


Figure 3.5

Non-Restoring Division Algorithm

Q ← Dividend (1100), B ← Divisor (11).

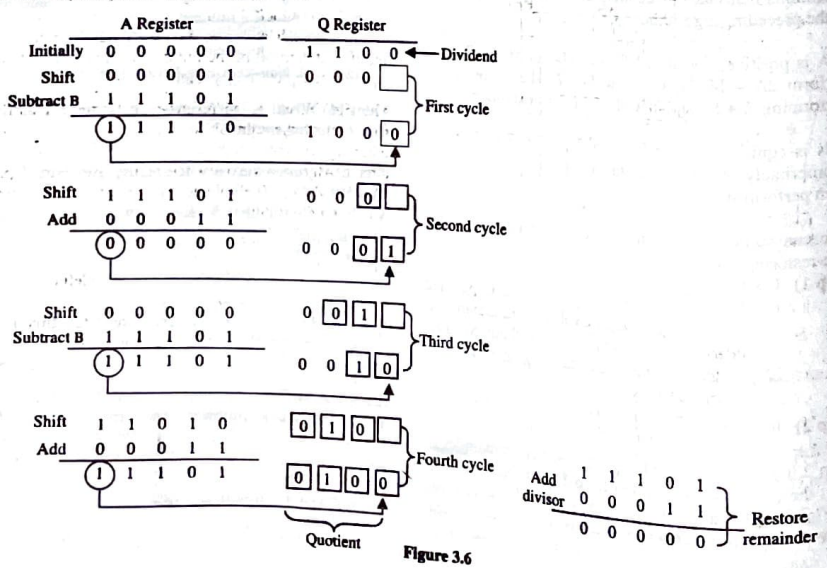


Figure 3.6

Ques 18) What is array multiplier? Explain with suitable example and diagram.

Or

Draw a 3 × 2 array multiplier. **(2019 [03])**

Ans: Array Multiplier

Array multiplier is well known due to its regular structure. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product is shifted according to their bit orders and then added. The addition can be performed with normal carry propagate adder. N – 1 adders are required where N is the multiplier length.

		A3	A2	A1	A0	Inputs
	x	B3	B2	B1	B0	
	C	B0×A3	B0×A2	B0×A1	B0×A0	
+	B1×A3	B1×A2	B1×A1	B1×A0		
C	Sum	Sum	Sum	Sum		
+	B2×A3	B2×A2	B2×A1	B2×A0		Internal Signals
C	Sum	Sum	Sum	Sum		
+	B3×A3	B3×A2	B3×A1	B3×A0		
C	Sum	Sum	Sum	Sum		
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Outputs

An **example** of 4-bit multiplication method is shown below:

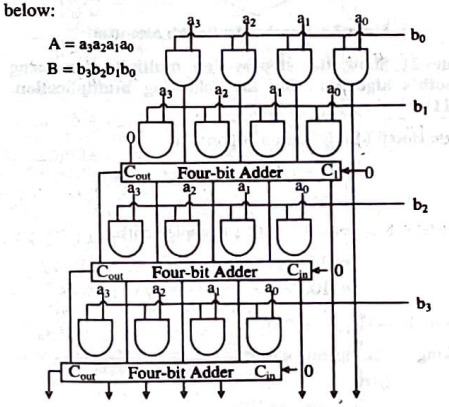


Figure 3.7: Product (A × B)

Although the method is simple as it can be seen from this example, the addition is done serially as well as in parallel. To improve on the delay and area the CRAs are replaced with Carry Save Adders, in which every carry and sum signal is passed to the adders of the next stage. Final product is obtained in a final adder by any fast adder (usually carry ripple adder). In array multiplication we need to add, as many partial products as there are multiplier bits.

Total Area = $(N-1) \times M \times \text{Area}_{FA}$

Delay = $2(M-1)\tau_{FA}$

Now as both multiplicand and multiplier may be positive or negative, 2's complement number system is used to represent them. If the multiplier operand is positive then essentially the same technique can be used but care must be taken for sign bit extension.

This arrangement is shown in the **figure 3.8**:

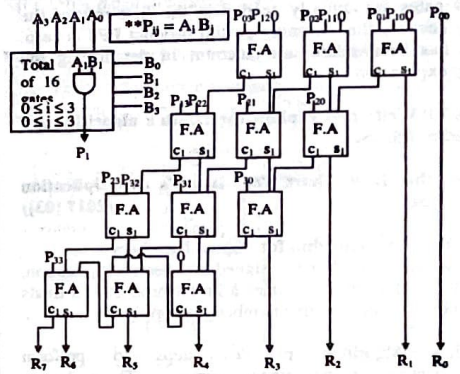
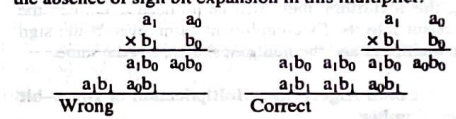


Figure 3.8

The reason for dealing with signed number incorrectly is the absence of sign bit expansion in this multiplier.



There is a way to correct this fault, which do not need to expand all of the bits in the partial product addition.

When 2's complement partial products are added in carry save arithmetic all numbers to be added in one adder stage have to be of equal bit length. Therefore, the sign bits of the partial product(s) in the first row and the sum and carry signals of each adder row are extended up to the most significant sign bit of the number with the largest absolute value to be added in this stage.

The sign bit extension results in a higher capacitive load (fan out) of the sign bit signals compared to the load of other signals and accordingly slows down the speed of the circuit. Algorithms exist when adding two partial products (A + B) which will eliminate the need of sign bit extension:

1) Extend sign bit of A by one bit and invert this extended bit.
2) Invert the sign bit of B.
3) Add A and B. Add '1' to one position left of MSB of B.

Ques 19) Prove that the worst case delay through an n × n array multiplier is 6(n –1) –1 gate delays. **(2018 [03])**

Ans: The worst case signal propagation delay path is from the upper right corner of the array to the high-order product bit output at the bottom left corner of the array. The path consists of the staircase pattern that includes the two cells at the right end of each row, followed by all the cells in the bottom row.

Assuming that there are two gate delays from the inputs to the outputs of a full adder block, the path has a total

of 6(n – 1) – 1 gate delays, including the initial AND gate delay in all cells, for the n × n array. Only the AND gates are actually needed in the first row of the array because the incoming partial product PP0 is zero. This has been taken into account in developing the delay expression.

Ques 20) Write and explain the booth's algorithm for signed numbers.

Or

Draw the flow chart for Booth's Multiplication algorithm. (2017 [03])

Ans: Booth's Algorithm for Signed Numbers

A powerful algorithm for signed-number multiplication, the Booth algorithm generates a 2n-bit product and treats both positive and negative numbers uniformly.

Booth's Algorithm provides steps to perform multiplication among binary numbers. These binary numbers are in 2's compliment representation.

Using this algorithm, there will be no need to convert the final result into its 2's compliment form, even if the sign of the multiplier and the multiplicand are not the same.

Steps of Booth Algorithm: Multiplication of two 4–bit Binary Number

Step 1) The multiplier and the multiplicand are placed in registers Q and M, respectively. Q and M registers are of 4 bits each.

Step 2) A 1 bit register Q_1 is placed logically to the right of least significant bit, Q_0, of register Q. Remember least significant bit is the rightmost bit.

Step 3) An 8 bit register A is also used. Both A and Q_1 are initialized to zero (The result of multiplication will be placed in A and Q_1). A counter C is set to a number equal to the number of bits in the multiplier i.e. 4.

Step 4) Bits in the multiplier (Q) are scanned.

Step 5) The rightmost bit in Q i.e., Q_0 is compared with the bit in register Q_1. If these bits are same i.e., 1-1 or 0-0, then the bits in the registers A, Q and Q_1 are shifted to right once. The shift performed is arithmetic i.e., the value of sign bit is taken into consideration.

Step 6) If the bits are different then the multiplicand (bits in M) is added to or subtracted from the bits in the register A. If the bits are 0-1 then the multiplicand is added to A and if the bits are 1-0 then the multiplicand is subtracted from A. After doing addition or subtraction, the bits in A, Q and Q_1 are arithmetically shifted right.

Step 7) After this, the counter C, is decremented by one. (In our case, Counter C was initially initialized to 4.

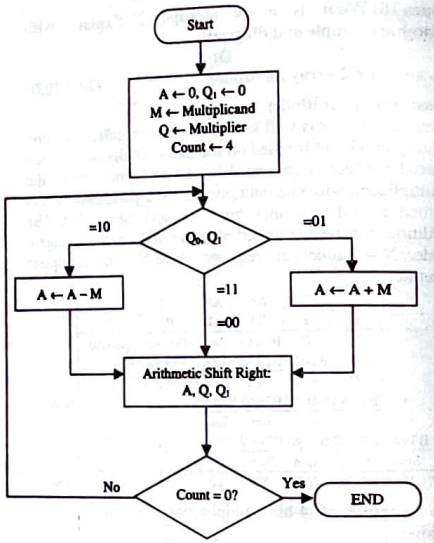


Figure 3.9: Flowchart for Booth's Algorithm

Ques 21) Show the step-by-step multiplication using Booth's algorithm for the following multiplication: (–11) × (–8).

Ans: Booth Multiplication Algorithm

$(+8)_{10} = (1000)_2$

0111

Hence, $(-8)_{10}$ = +1 (2's Complement)

$= (1000)_2$

$= (1000)_2$

Similarly $(+11)_{10} = (1011)_2$

Taking 2's Complement then

0100

$(-11)_{10}$ = +1

$= 0101$

$(-11)_{10} = (0101)_2$

From the two numbers, pick the number with smallest difference between a series of consecutive numbers, and make it a multiplier, i.e., $(-8)_{10} = 1000 \rightarrow$ from 1 to 0 – one change, 0 to 0 – no change, and 0 to 0 – no other change.

Hence, total one change for this one.

For $0101 \rightarrow$ from 0 to 1 – one change, 1 to 0 – another change, 0 to 1 – another change.

Hence, total three changes which are greater than last one.

Hence, multiplication is $(-11)_{10} \times (-8)_{10}$ and $(-8)_{10}$, i.e., 1000 is the multiplier.

Multiplicand (B) = (0101) (–11) and Multiplier (Q) ← (1000) (–8)

Steps	A	Q	Q_{-1}	Operation
	0000	1000	0	Initial
Step1	0000	0100	0	Shift-right
Step2	0000	0010	0	Shift-right
Step3	0000	0001	0	Shift-right
Step4	1011	0001	0	A ← A–B
	0101	1000	1	Shift-right
Result: 0101 1000 = +88				

Ques 22) Multiply 85 × 59 using Booth's algorithm.

Ans: Multiplicand = 85 = 01010101;
2's complement = 10101011

Multiplier = 59 = 00111011

Iteration	Step/Action	Accumulator	Multiplier	Q_{0-1}
0	Initial values	00000000	00111011	0
1	(a) Subtract MD	10101011		
	(Add 2's Comp)	10101011	00111011	0
	(b) Shift right	11010101	10011101	1
2	(a) No action	11010101	10011101	1
	(b) Shift right	11101010	11001110	1
3	(a) Add MD	01010101		
		00111111	11001110	1
	(b) Shift right	00011111	11100111	0
4	(a) Subtract MD	10101011		
		11001010	11100111	0
	(b) Shift right	11100101	01110011	1
5	(a) No action	11100101	01110011	1
	(b) Shift right	11110010	10111001	1
6	(a) No action	11110010	10111001	1
	(b) Shift right	11111001	01011100	1
7	(a) Add MD	01010101		
		01001110	01011100	1
	(b) Shift right	00100111	00101110	0
8	(a) No action	00100111	00101110	0
	(b) Shift right	00010011	10010111	0

Product — Ignore

Product = 0001001110010111 = 5015

Ques 23) Multiply –52 × 59 using Booth's algorithm.

Ans: Multiplicand = (–52) = 11001100;
2's complement = 00110100
Multiplier = 59 = 00111011

Iteration	Step/Action	Accumulator	Multiplier	Q_{0-1}
0	Initial values	00000000	00111011	0
1	(a) Subtract MD	00110100		
		00110100	00111011	0
	(b) Shift right	00011010	00011101	1
2	(a) No action	00011010	00011101	1
	(b) Shift right	00001101	00001110	1
3	(a) Add MD	11001100		
		11011001	00001110	1
	(b) Shift right	11101100	10000111	0
4	(a) Subtract MD	00110100		
		00100000	10000111	0
	(b) Shift right	00010000	01000011	1
5	(a) No Action	00010000	01000011	1
	(b) Shift right	00001000	00100001	1
6	(a) No Action	00001000	00100001	1
	(b) Shift right	00000100	00010000	1
7	(a) Add MD	11001100		
		11010000	00010000	1
	(b) Shift right	11101000	00001000	0
8	(a) No Action	11101000	00001000	0
	(b) Shift right	11110100	00000100	0

Product — Ignore

Product = 1111010000000100= –3068

Note: Sign bit of product is 1. Hence the product is in 2's complement.

Ques 24) Multiply 7 by 3 using Booth's algorithm.

Ans: Table 3.2 shows the sequence of events in Booth's algorithm for the multiplication of 7 by 3:

Table 3.2: Example of Booth's Algorithm (7×3)

A	Q	Q_{-1}	M	Initial Values	
0000	0011	0	0111		
1001	0011	0	0111	A ← A – M	First cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth cycle

Ques 25) Multiply each of the following pairs of signed 2's-complement numbers using the Booth algorithm. In each case, assume that A is the multiplicand and B is the multiplier. (2018 [06])

i) A = 001011 and B = 011011

ii) A = 000111 and B = 000111

Ans:

i)

```
A          0 0 1  0 1 1                    11
X        x 0 1 1  0 1 1                    27
Y          1 0 –1 1 0 –1                   Recoded multiplier


Add-A    + 1 1 0  1 0 1
Shift      1 1 1  0 1 0  1
Shift Only 1 1 1  1 0 1  0 1
Add A    + 0 0 1  0 1 1


           0 0 1  0 0 0  0 1
Shift      0 0 0  1 0 0  0 0 1
Add-A    + 1 1 0  1 0 1
           1 1 1  0 0 1  0 0 1
Shift      1 1 1  1 0 0  1 0 0 1
Shift only 1 1 1  1 1 0  0 1 0 0 1
Add A    + 0 0 1  0 1 1


           0 0 1  0 0 1  0 1 0 0 1
Shift      0 0 0  1 0 0  1 0 1 0 0 1 2 9 7
```

ii)

```
A          0 0 0  1 1 1                    7
X        x 0 0 0  1 1 1                    7
Y          0 0 1  0 1 –1                   Recoded multiplier

Add-A    + 1 1 1  0 0 1
Shift      1 1 1  1 0 0  1
Shift Only 1 1 1  1 1 0  0 1
Shift only 1 1 1  1 1 1  0 0 1
Add A    + 0 0 0  1 1 1


           0 0 0  1 1 0  0 0 1
Shift      0 0 0  0 1 1  0 0 0 1
Shift only 0 0 0  0 0 1  1 0 0 0 1
Shift only 0 0 0  0 0 0  1 1 0 0 0 1   49
```

PIPELINING

Ques 26) What is pipelining? Explain briefly.

Ans: Pipelining
Pipelining is a technique of decomposing a sequential process into smaller fragments or sub-operations. Multiple fragments or sub-operations are executed in a given segment in parallel with all other segments. A pipeline is a collection of processing segments through which data is processed. The result of one segment is passed to other segment until a desired result is obtained. The characteristics of a pipeline are that a number of computations are in progress in different segments, at the same time.

A pipeline is the continuous and somewhat overlapped movement of instruction to the processor or in the arithmetic steps taken by the processor to perform an instruction. Pipelining is the use of a pipeline.The pipeline consists of a cascade of processing stages. The stages are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high-speed interface latches. The latches are fast registers for holding the intermediate results between the stages. Information flows between adjacent stages are under the control of a common clock applied to all the latches at the same time.

Without a pipeline, a computer processor gets the first instruction from memory, performs the operation it calls for, and then goes to get the next instruction from memory, and so forth. While fetching (getting) the instruction, the arithmetic part of the processor is idle. It must wait until it gets the next instruction.

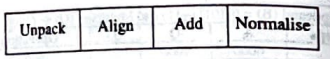
With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor until each instruction operation can be performed. The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed during a given time period.

Ques 27) Explain the basic principle of the pipelining. Also explain the classification of pipeline processors.

Ans: Pipelining
According to the levels of processing, **Handler** has proposed the following classification scheme for pipeline processors, as illustrated in **figure 3.11**.

1) **Arithmetic Pipelining:** The arithmetic logic units of a computer can be segmentized for pipeline operations in various data formats (**figure 3.11-a**). Well-known arithmetic pipeline examples are the four-stage pipe used is Star-100, the eight-stage pipes used in the TI-ASC, the up to 14 pipeline stages used in the Cray-1.

   Pipelining can also be applied to arithmetic operations. **For example,** we show a floating point add pipeline in **figure 3.10 b**.



b) Floating-point add pipeline stages

Figure 3.10: Floating Point Add Pipeline

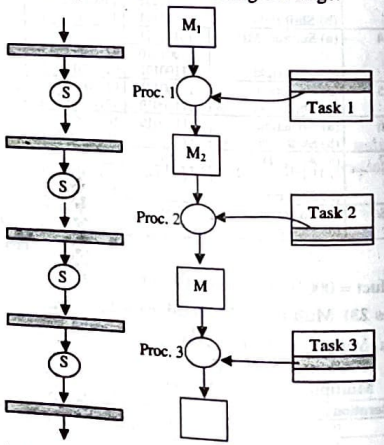
The floating-point add unit has several stages:

i) **Unpack:** The unpack stage partitions the floating-point numbers into the three fields – the sign field, exponent field, and mantissa field. Any special cases such as not-a-number (NaN), zero, and infinities are detected during this stage.

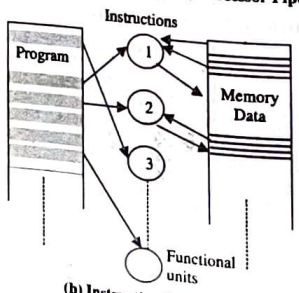
ii) **Align:** This stage aligns the binary points of the two mantissas by right-shifting the mantissa with the smaller exponent.

iii) **Add:** This stage adds the two aligned mantissas.

iv) **Normalise:** This stage packs the three fields of the result after normalisation and rounding into the IEEE-754 floating-point format. Any output exceptions are detected during this stage.



(a) Arithmetic Pipelining (c) Processor Pipelining



(b) Instruction Pipelining

Figure 3.11: Handler Classification of Pipelined Processors

2) **Instruction Pipelining:** The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode, and operand fetch of subsequent instructions (**figure 3.11-b**). This technique is also known as **instruction look ahead**. Almost all high-performance computers are now equipped with instruction-execution pipelines.

3) **Processor Pipelining:** This refers to pipeline processing of the same data stream by a cascade of processors (**figure 3.11-c**), each of which processes a specific task. The data stream passes the first processor with results stored in a memory block, which is also accessible, by the second processor. The second processor then passes the refine results to the third, and so on. The pipelining of multiple processors is not yet well accepted as a common practice

Ques 28) Write the performance issues of Pipelining?

Ans: Performance Issues of Pipelining
In the following analysis, we provide three performance measures for the goodness of a pipeline. These are the Speed-up S(n), Throughput U(n), and Efficiency E(n). It should be noted that in this analysis we assume that the unit time T = t units:

1) **Speed-Up S (n):** Consider the execution of m tasks (instructions) using n-stages (units) pipeline. As can be seen, n + m – 1 time units are required to complete m tasks.

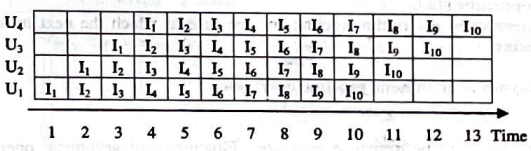


Figure 3.12: Space-Time Chart (Gantt Chart)

$$\text{Speed-up } S(n) = \frac{\text{Time Using Sequential Processing}}{\text{Time Using Pipeline Processing}} = \frac{m \times n \times t}{(n+m-1) \times t} = \frac{m \times n}{n+m-1}$$

2) **Throughput U(n):** Throughput U(n) = No. of Tasks Executed Per Unit Time

$$= \frac{m}{(n+m-1) \times t}$$

3) **Efficiency E(n):** Efficiency E(n) = Ratio of the Actual Speed-up to the Maximum Speed-up

$$= \frac{\text{Speed-up}}{n} = \frac{m}{n+m-1}$$

Ques 29) What is pipeline delay? Explain in detail.

Ans: Pipeline Delay
The simple analysis ignores an important aspect that can affect the performance of a pipeline, i.e., **pipeline stall**. A pipeline operation is said to have been stalled if one unit (stage) requires more time to perform its function, thus forcing other stages to become idle.

For example, consider the case of an instruction fetch that incurs a cache miss. Assume also that a **cache miss** requires **three extra time units**. **Figure 3.13** illustrates the effect of having instruction I_2 incurring a cache miss (assuming the execution of ten instructions I_1 to I_{10}):

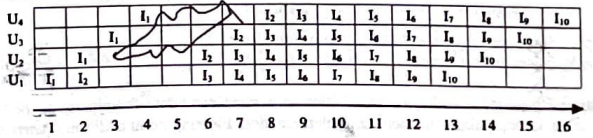


Figure 3.13: Effect of a Cache Miss on the Pipeline

Figure shows that due to the extra time units needed for instruction I_2 to be fetched, the pipeline stalls, i.e., fetching of instruction I_3 and subsequent instructions are delayed. Such situations create what is known as pipeline **bubble** (or pipeline **hazards**). The creation of a pipeline bubble leads to wasted unit times, thus leading to an overall increase in the number of time units needed to finish executing a given number of instructions. The number of time units needed to execute the 10 instructions shown in **figure 3.13** is now 16 time units, compared to 13 time units if there were no cache misses.

Pipeline hazards can take place for a number of other reasons. Among these are instruction dependency and data dependency.

To illustrate the impact of the variable stage time on pipelined execution, let us assume that the operand fetch of the I_2 instruction takes more time – three clock cycles rather than one. **Figure 3.14** shows how the increased execution time for I_2's OF stage causes the pipeline to **stall** for two clock cycles:

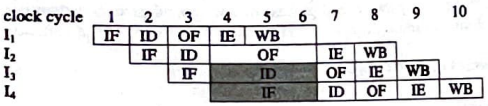


Figure 3.14: Delay in Any Stage can Cause Pipeline Stalls

As one can see from this **figure**, the stalled pipeline reduces the overall throughput. One of the goals in designing a pipeline is to minimise pipeline stalls.

Bubble is an alternative name for a pipeline stall.
The delay between when a branch instruction enters the pipeline and the time at which the next instruction enters the pipeline is often called **processor's delay**.

Ques 30) Discuss about the various types of arithmetic pipeline operation.

Ans: Arithmetic Pipeline
Arithmetic pipelines form the heart of a high performance computer. Floating-point arithmetic operations (like add, subtract, multiply and divide) take considerable amount of time compared to the (simpler) fixed-point arithmetic and logical operations.

1) **Fixed Point Addition Pipeline:** Ripple carry addition involves carry propagation through all the bit positions of the adder i.e. for a N bit adder, the adder spends N units of time (one unit = time taken by a full adder to generate the carry). The use of carry look ahead addition technique reduces this time to the order of log N. Pipelining of ripple carry addition makes it possible to achieve one addition every clock cycle.

   This idea is presented in **figure 3.15**, which shows a four-bit pipeline adder. It uses only four 1-bit full adders and a number of D flip-flops. This adder, although based on ripple carry principle, produces one addition every clock cycle. The minimum period of the clock is given by the sum of delays of the full adder and that of the D flip-flop.

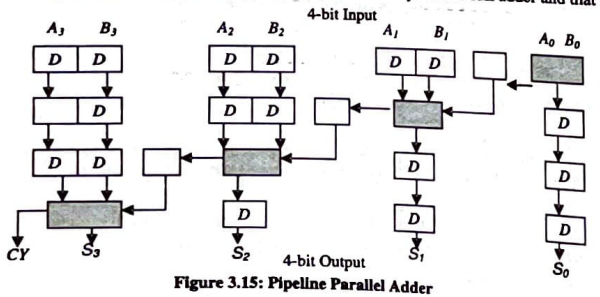


Figure 3.15: Pipeline Parallel Adder

Where D= D flip-flops, which are all connected by a common clock
FA= Full Adder (gray boxes in the figure)

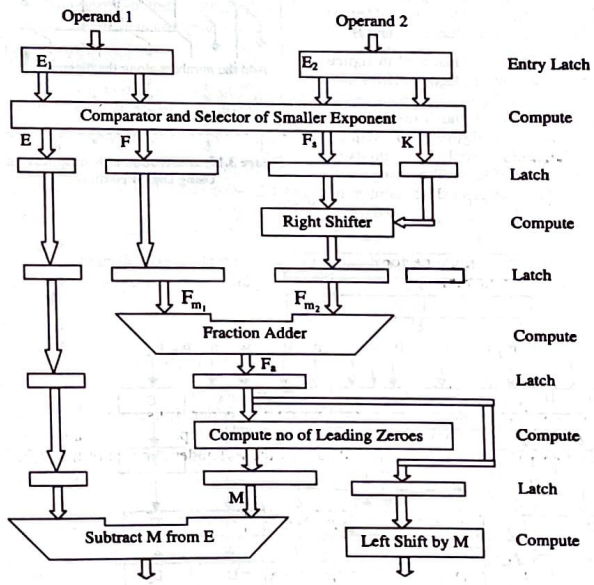
2) **Floating Point Addition Pipeline:** Floating point addition is a good candidate for pipelining because several steps are required to perform it using a nonpipelined sequential implementation. Floating point addition is carried out as follows:
   Let, $n_1 = (f_1, e_1)$ and
   $n_2 = (f_2, e_2)$ be two floating-point numbers to be added, where f_1, f_2 and e_1, e_2 are the mantissa and the exponents of the numbers n_1 and n_2 respectively.

The steps are summarized in the following algorithm for addition of $n_1 + n_2$:
i) Compare e_1 and e_2 and pick up the fraction of a number with a smaller exponent. Pick up difference $k = |e_1 - e_2|$ and also the larger exponent.

ii) Shift right the mantissa of the number with smaller exponent by k positions.
iii) Add the fractions.
iv) Compute M = the number of leading zeroes in the mantissa of the result.
v) Subtract the number M from the exponent and shift lefty the mantissa of the result by M positions.

A pipeline implementation of the floating-point addition based on the above algorithm is shown in **figure 3.16.** There are five stages corresponding to the five steps of the algorithm. Each stage has latches followed by the combinational logic to implement the stage's function. Some of the stages in the pipeline operate only on a part of the input bits and pass the remaining bits to the respective next stages without any transformation.

For example, the exponent bits (shown as E), i.e., output from the first stage moves without any change till the last stage.



Notes:
i) M = No. of leading zeroes.
ii) K = Exponent Difference.
iii) F_s = Fraction of number with a smaller exponent.
iv) E = Larger Exponent.
v) F = Fraction of number with larger exponent.
vi) F_s = Fraction resulting due to addition of F_1 and F_2.
vii) Fm_1, Fm_2 = Fractions to be added.

Figure 3.16: Floating Point and Pipeline

Where, M= number of leading zeros
K= Exponent difference.
F_2= Fraction of number with a smaller exponent.
E= larger exponent.
F= Fraction of number with larger exponent.
F_s= Fraction resulting due to addition of F_1 and F_2.
Fm_1, Fm_2= Fraction to be added.

3) **Fixed Point Multiplication Pipeline:** The fixed-point multiplication operations involve a sequence of a number of basic steps composed of shift and add operations. The number of these steps required is of the order of n for multiplying two n-bit numbers.

Multiplication Using Digit Products

A pipelined multiplier based on the digit products can be designed using digit product generation logic and the digit adders. The idea is illustrated through the following decimal multiplication example.

Consider the multiplication of numbers 25 and 35. The partial products of the digits of the numbers 25 and 35 are filled in a 2 × 2 matrix as shown in **figure 3.17.** For n digit numbers matrix size will be n × n. Each element of the matrix holds a pair of numbers as shown in **figure 3.17.**

The element in the i^{th} row and the j^{th} column holds the product of two digits a_i and b_j of numbers A and B respectively. The numbers formed out of these digit products are added as illustrated in the **figure 3.17**to form the product of A and B.

Based on the principles of multiplication illustrated in **figure 3.17,** a pipelined multiplier is presented in **figure 3.18.** Since the numbers are binary, digit products P_{ij} are given by a_i and b_j , where a_i and b_j are i^{th} and j^{th} digits of A and B respectively. The multiplier architecture shown in **figure 3.18** is quite general and binary bits a_i and b_j could be replaced by digits of any radix (represented by a number of bits). The change will also require the D blocks to have corresponding number of bits.

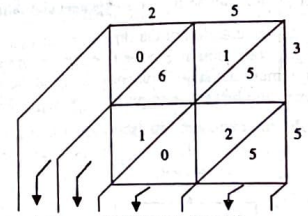


Figure 3.17: Illustration of Multiplication Using Digit Products

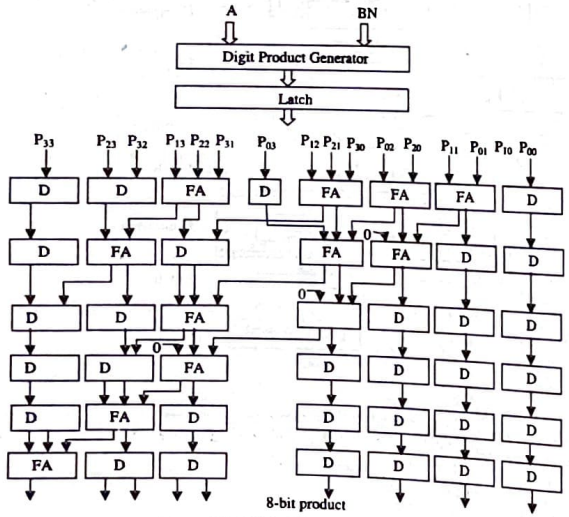


Figure 3.18: Multiplier Based on Digit Products

4) **Floating Point Multiplication Pipeline:** Floating point multiplication involves the following three major steps.
   i) Multiplication of fractions
   ii) Addition of Exponents
   iii) Normalization of the result.

   Since fractions and exponents are fixed point numbers, the steps 1 and 2 can be implemented using the principles discussed earlier. Normalization step can be implemented as given in the floating-point addition.

5) **Floating Point Division:** Division operation appears less frequently in computer programs compared to addition, subtraction and multiplication and hence separate pipeline unit for the division is rarely implemented. It is common to

schedule the division using adder and multiplier pipelines. There are two basic methods employed in high performance computers to implement the division. The first method employs **reciprocal approximation** using Newton's method for finding the roots of a polynomial, while the second uses an **iterative convergence** of a series. Both these methods have been commercially used in high performance computers. **Cray** systems use the reciprocal approximation method while the **IBM** systems employ the series convergence method.

i) **Reciprocal Approximation:** The quotient Q = M/D of numbers M and D can be expressed as the inverse of D multiplied by the number M. Let the required inverse be X. Thus we have:

$D = 1/X$

We define f(X) as: $f(X) = (D - 1/X)$

A real root of f(X) gives us the inverse of D. The root may be found using **Newton's iterative method.** In each iteration, a new value of X is computed using the previous value of X by the formula:

$X_n = X_0 - f(X)/f'(X)$,

Where X_n, X_0 are new and old values of X respectively.
Substituting the values of $f'(X)$, we obtain:
$X_n = X_0 - (D - 1/X_0) \cdot X_0 \cdot X_0$
$X_n = X_0 (2 - D \cdot X_0)$

The successive values of X converge to the required reciprocal. For 32-bit numbers, at most 10 steps may be enough. A pipeline could be arranged to iterate the above computations. The initial value of X_0 must be chosen such that, the value of X converges.

ii) **Iterative Convergence Method:** The quotient Q = M/D can also be computed as follows. Let d = 1 – D, then required quotient is given by

$$Q = \frac{M}{(1-d)}$$

Now multiply both the numerator and the denominator of the above equation by the following series of numbers:
$(1+d), (1+d^2), (1+d^4), (1+d^8) \cdots$
To obtain

$$Q = \frac{M(1+d)(1+d^2)(1+d^4)(1+d^8)\cdots}{(1-d)(1+d)(1+d^2)(1+d^4)(1+d^8)\cdots}$$

If D < 1 then d is also less than 1, and therefore the denominator converges to 1 after sufficient number of terms. This assumption is valid for the floating-point unsigned numbers, since fractions are normalized and are less than one in these numbers. The quotient Q is thus given by

$M \cdot (1+d)(1+d^2) + (1+d^4) \cdots$

The above computation can be done using multiplier and adder pipelines. Division is among the least occurring operation (percentage wise) in the programs and hence the overall cost performance benefit must be looked at if a separate divide pipe is to be implemented. On the other hand when performance is a prime objective, the divide pipe may also be implemented as a separate unit.

Ques 31) Explain about the Instruction Pipeline with suitable diagram.

Ans: Instruction Pipeline

The execution cycle of a typical instruction includes four phases:
1) Fetch
2) Decode
3) Execute
4) write-back

These instruction phases are often executed by an instruction pipeline as demonstrated in **figure 3.19 (a).** The pipeline, like an industrial assembly line, receives successive instructions from its input end and executes them in a streamlined, overlapped fashion as they flow through.

A pipeline cycle is intuitively defined as the time required for each phase to complete its operation, assuming equal delay in all phases (pipelines stages). The basic definitions associated with instruction pipeline operations are:
1) **Instruction Pipeline Cycle:** The clock period of the instruction pipeline.

2) **Instruction Issue Latency:** The time (in cycles) required between the issuing of two adjacent instructions.
3) **Instruction Issue Rate:** The number of instructions issued per cycle, also called the degree of a superscalar processor.
4) **Simple Operation Latency:** Simple operations make up the vast majority of instructions executed by the machine, such as integer adds, loads, stores, branches, moves, etc. On the contrary, complex operations are those requiring an order-of-magnitude longer latency, such as divides, cache misses, etc. These latencies are measured in number of cycles.
5) **Resource Conflicts:** It is the situation where two or more instructions demand use of the same functional unit at the same time.

A base scalar processor is defined as a machine with one instruction issued per cycle, a one-cycle latency for a simple operation, and a one-cycle latency between instruction issues. The instruction pipeline can be fully utilized if successive instructions can enter it continuously at the rate of one per cycle, as shown in **figure 3.19 (a)**.

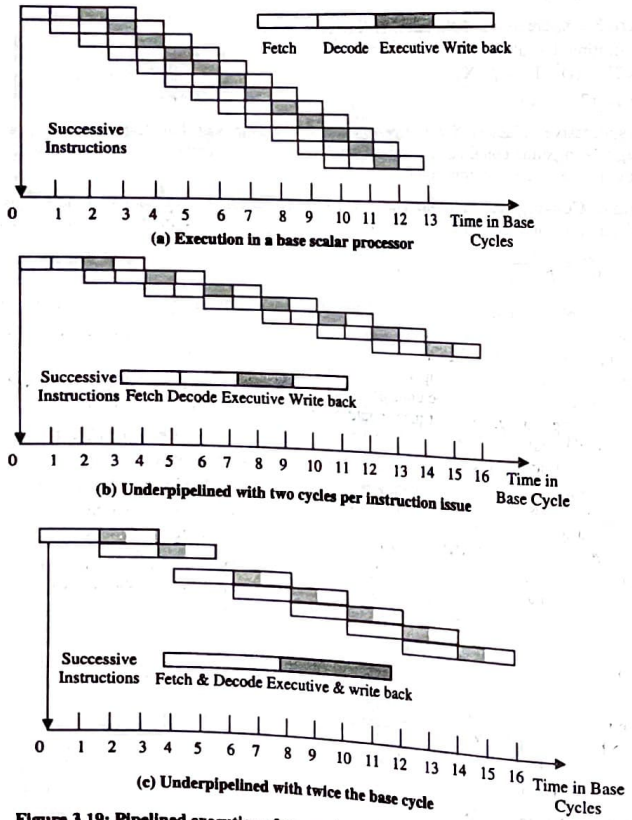


Figure 3.19: Pipelined execution of successive instruction in a base scalar processor and in two under pipelined cases

The instruction issue latency can be more than one cycle for various reasons. For example, if the instruction issue latency is two cycles per instruction, the pipeline can be underutilized, as demonstrated in Another underpipelined situation is shown in **figure 3.19 (c)**, in which the pipeline cycle time is doubled by combining pipeline stages. In this case, the fetch and decode phases are combined into one pipeline stage, and execute and write-back are combined into another stage. This will also result in poor pipeline utilization.

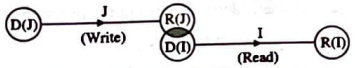
Ques 32) What are hazards?
Or
What is the data hazards and control hazards?
Or
Write the short notes on Hazard Resolution

Ans: Hazards
Pipeline hazards are caused by "resource-usage conflicts" among various instructions in the pipeline. Such hazards are triggered by inter-instruction dependencies.

When successive instructions overlap their fetch, decode and execution through a pipeline-processor, inter instruction dependencies may arise to prevent the sequential data flow in the pipeline.

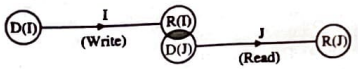
For example, an instruction may depend on the results of a previous instruction. Until the completion of the previous instruction, the present instruction cannot be initiated into the pipeline. In other instances, two stages of a pipeline may need to update the same memory location. Hazards of this sort, if not properly detected and resolved, could result in an "interlock-situation", in the pipeline or produce unreliable results by overwriting.

Types of Hazards
1) **Data Hazards:** It occurs when there is a conflict in the access of an operand location. In general terms, one can state the hazard in this form – Two instructions in a program are to be executed in sequence and both access a particular memory or register operand. If the two instructions are executed in strict sequence, no problem occurs. However, if the instructions are executed in a pipeline, then it is possible for the operand value to be updated in such a way as to produce a different result than would occur with strict sequential execution. In other words, the program produces an incorrect result because of the use of pipelining.
There are 3-classes of data dependent hazards, according to various data update patterns:
   i) **Write After Read (WAR) Hazards:** It may occur when J-attempts to modify some data object that is read by I, i.e.,



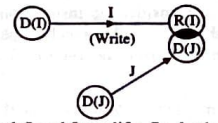
[∵ It is modified by J. So, J writes]
∴ The necessary condition for this hazard are:
$D(I) \cap R(J) \neq \phi$ for WAR .....(1a)
↑
(f_j) or Null

   ii) **Read-After-Write (RAW) Hazards:** A RAW hazard between the two (2) instructions I and J may occur when J attempts to read some data object that has been modified by I, i.e.,



[∵ It is modified by I. So, I writes]
∴ The necessary condition for this hazard is:
$R(I) \cap D(J) \neq \phi$ for RAW (Null) .....(1b)

   iii) **Write-After-Write (WAW) Hazards:** A WAW hazard may occur if both I and J attempt to modify the same data object. i.e.,



[∵ Both I and J modify. So, both write]

Note: RAR is not there because when we do read-after-read nothing is changed, so RAR is no problem.

$R(I) \cap R(J) \neq \phi$ for WAR .....(1c)
↑
(f_j) or null

∴ Improper timing and data dependencies may create some hazardous situations.

2) **Control Hazard:** A control hazard, also known as a branch hazard, occurs when the pipeline makes the wrong decision on a branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded.

"Resource-objects" are used to refer to working-registers, memory-locations and special flags.

∴ (Resource-object) = data-object
↑
Content of instruction

∴ Each instruction can be considered as a mapping from set of data objects to a set of other data objects.

Domain, D(I), of an instruction I is defined as the set of 'resource-objects' whose contents may affect the execution of I (i.e., operands only).

Range, R(I), of an instruction, I is the set of resource objects whose data objects may be modified by execution of instruction I (i.e., results of execution of instruction).

Obviously, the operands to be used in an instruction execution are retrieved (read) from its domain and the results will be stored (written) in its range. Consider execution of 2 instruction I and J in a program. Instruction J appears after instruction I in the program. There may be none or other instructions between instruction I and J. Instruction J may enter the execution pipe before or after the completion of the execution of instruction I. The improper timing and data dependencies may create some hazards situations.

3) **Resource Hazards:** These hazards result when two or more instructions in the pipeline want to use the same resource. Such resource conflicts can result in serialised execution, reducing the scope for overlapped execution. Resource hazards, sometimes referred to as structural hazards.

Hazard Resolution

Once a hazard is detected, the system should resolve the interlock situation. Consider the instruction sequence {... I, I + 1, ... J, J + 1, ...} in which a hazard has been detected between the current instruction. J and a previous instruction I.

A straight forward approach is to stop the pipe and to suspend the execution of instruction J, J + 1, J + 2 ..., until the instruction. I has passed the point of resource conflict.

In order to avoid RAW-type of hazards, IBM engineers developed a short-circuiting approach, also called as data-forwarding, which forwards multiple copies of the data to as many waiting instruction as may wish to read it.

Note: Another type of hazards is due to the job-scheduling problem.

Once a task is initiated in a static pipeline, its, flow pattern is fixed. An 'initiation' refers to the start of a single function evaluation when 2 or more initiations attempt to use the same stage at the same time, a collision results. Thus, the job sequencing problem is "To properly schedule queued tasks awaiting initiation in order to avoid collisions and to achieve high throughput."

Ques 33) Consider the following instructions ($I_1 - I_5$) and identify various hazards:
I_1:$R_1 \leftarrow (M(R_4))$; Load
I_2:$R_2 \leftarrow R_1 + R_2$; Add
I_3:M[R] $\leftarrow R_2$; Store
I_4:$R_2 \leftarrow R_6$ and R_3; AND (logical)
I_5:$R_6 \leftarrow R_6 + 1$; Increment?

Ans: The instruction < I_1, I_2 > pose a RAW hazard.

< I_2, I_4 > pose WAW hazard and < I_4, I_5 > pose a WAR hazard.

Ques 34) Identify various hazards in the following instruction stream:
I_1:$R_2 \leftarrow R_2 + R_3$
I_2: if zero (R_2) then RAW Hazard.

Ans: < I_1, I_2 > show RAW Hazard.

Data dependency hazards are problematic. They may prevent simultaneous processing of two instructions in the pipeline. As a result, they disrupt and reduce throughput.

Ques 35) Identify the hazards in the following instruction steam:
I_1:$R_2 \leftarrow R_2 + R_3$
I_2: if R_2 = 0 then go to L3
I_3:M[500] $\leftarrow R_2$
I_4:L3:M[600] $\leftarrow$ M[500] + R_5?

Ans: < I_1, I_2, I_3 > show RAW hazard, < I_2, I_3 > show control hazard

Here, I_3 is the constituent of control dependency hazard.

Ques 36) Identify all of the RAW, WAR, WAW and control hazards in the following instruction sequence:
DIV r2, r5, r8
SUB r9, r2, r7
ASH r5, r14, r6

MUL r11, r9, r5
BEQ r10, 0, r12
OR r8, r15, r2

Ans: Let us denote all these instructions by statement numbers – S1 to S6 as follows:
S1:r2 $\leftarrow$ r5 /r8
S2:r9 $\leftarrow$ r2/r7
S3:r5 $\leftarrow$ r14 << r6
S4:r11 $\leftarrow$ r9 × r5
S5:r10 $\leftarrow$ r12
S6:r8 $\leftarrow$ r15 OR r2.

1) Now, flow dependencies in the program are:
   i) S1 → S2
   ii) S3 → S4
   iii) S1 → S6
   iv) S2 → S4

   ∴ The program has 4 RAW hazards.

2) No anti-dependencies in the program.

   ∴ No WAR hazards.

3) No output dependencies in the program.

   ∴ Now WAW hazards.

Module 4

Control Logic Design

CONTROL LOGIC DESIGN

Ques 1) Describe the control unit. What are the functions of control unit?
Or
What are the input to and output from control unit?

Ans: Control Unit
Control unit is a digital circuitry that controls the flow of information through the processor and coordinates the activities of other units within it. The control unit of a CPU controls the entire operation of the computer. This part of the CPU really acts as the brain of the computer.

A control unit contains a clock portion, whose job is to provide clock pulses. This clock signal of control unit is used for measuring the timing of the micro-operations.

Functions of Control Unit
The functions of the control unit are given below:
1) It controls all other devices such as memory, input and output devices connected to the CPU.
2) It fetches instruction from the memory, decodes the instruction, interprets the instruction to know what tasks are to be performed and sends suitable control signals to other components to perform further necessary steps to execute the instruction.
3) It maintains order and directs the operation of the entire system. It gives order to ALU what operations are to be performed.
4) It generates **timing and control** signals, and provides them for all operations. It controls the data flow between CPU and peripherals (including memory).
5) It provides status, control and timing signals that the memory and I/O devices require.

Inputs to the Control Unit
1) **Master Clock Signal:** This signal causes micro-operations to be performed. In a single clock cycle either a single or a set of simultaneous micro-operations can be performed.
2) **Input from Instruction Register:** The operation code which normally include addressing mode bits of the instruction, help in determining the various cycles to be performed and hence determines the related micro-operations which are needed to be performed.
3) **Flags:** The control unit for determining the status of the CPU uses Flags. The outcomes of a previous operation on ALU can also be detected using flags.
4) **Control Signals from Control Bus:** Some of the control signals are provided to the control unit through the control bus. These signals are issued from outside the CPU. Some of these signals are interrupt signals and acknowledgment signals.

Output from Control Unit
1) **Control Signals which are required within the CPU:** These control signals cause two types of micro-operations, *viz.*, for data transfer from one register to another and for performing an ALU operation using input and output registers.
2) **Control Signals to Control Bus:** The basic purposes of these control signals are to bring or to transfer data from CPU register to memory or I/O modules. These control signals are issued on the control bus to activate a data path.

Ques 2) Write short note on control organisation.
Or
Discuss the hardwired control organisation and its design. What are the different methods of Hardwired Control Design? **Or**
With a diagram, explain how control signals are generated using hardwired control. **(2018 [10])**

Ans: Control Organisation
The control unit of a computer controls all the operations within it. It generates control signals using logic circuits. Timing control means providing clock pulses to the flip-flops and registers in the system to control specific operations. There can be two types of control organisation:
1) Hardwired Control Organisation
2) Micro-Programmed Control Organisation

Hardwired Control Organisation
The hardwired control unit is considered as a sequential logic circuit that implements micro-operations using a combinational circuit.

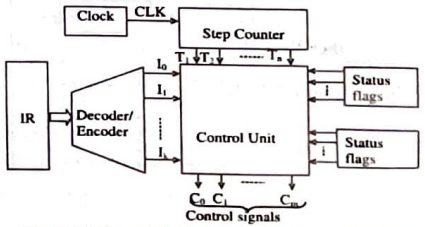


Figure 4.1: General Block Diagram of Hardwired Control

The inputs are from the micro-operation Counter, Instruction Register (IR) and Status Register, which are transformed into a set of outputs that are the control signals (**figure 4.1**). It is a controller as a sequential logic circuit or a finite state machine that generates a sequence of control signals in response to the externally supplied instructions.

For the hardwired control unit, the hardware involved is decoder/encoder circuits, which are combinational circuits that generate the required control signals as output. The input to instruction decoder is from the instruction register IR. When one instruction is selected from n available instructions, that line is set to 1 while all other lines are set to 0. Thus, a separate signal line is provided for each step in the control sequence by the instruction decoder. The output of the instruction decoder consists of a separate line for each machine instruction. The input signals to the encoder are combined to generate the individual control signals.

Design of Hardwired Control Unit

The controller is designed as a sequential logic circuit which generates the specific sequence of control signals as its primary output. **For example,** let consider the micro-operation sequence of fetch cycle as shown in **figure 4.2**.

C_{01}: MAR ⇐ PC
C_{02}, C_{03}: MDR ⇐ M(MAR); PC ⇐ PC + 1
C_{04}: IR ⇐ MDR
C_{05}: F ⇐ 0; E ⇐ 1

Figure 4.2: Sequence of Micro-Operations of Fetch Cycle

The hardwired control circuit implementing the sequence of micro-operations noted in **figure 4.2** is developed in **figure 4.3**. A sequence of 4 control signals C_0, C_1, C_2, C_3 can be developed by using a 2-bit sequence counter. Its output is decoded to derive the desired control signals in the given sequence. Application of control signals at the appropriate points on the data path (shown within the dotted rectangle) is also indicated. Thus, whenever the C_0 signal is available the current content of PC gets loaded in the MAR. Subsequently, the memory Read cycle is initiated with the control signal C_1.

For the sake of simplicity it is assumed that duration of clock is long enough to complete the memory read operation in one cycle while MDR gets loaded with the word addressed by MAR. Also, parallel PC can be incremented by the same control signal. The subsequent two control signals C_2 and C_3 execute the indicated micro-operations. The control path (i.e., the controller circuit along with the application of control signals on the datapath) is explicitly indicated in **figure 4.3**.

Hardwired Control Design Methods

The major goal of the hardwired control scheme is to minimise cost of the circuit while achieving higher efficiency in terms of operation speed. Following methods have evolved for systematic design of hardwired control logic:

1) **Sequence Counter Method:** This is the most popular method conveniently employed for design of controller of moderate complexity. The illustration of **figure 6.3** is a simple design using this method which employs a sequence counter.
2) **Delay Element Method:** This method depends on the use of clocked delay elements for generating the sequence of control signals.
3) **State Table Method:** This scheme employs the traditional algorithmic approach to sequential circuit design using classical state table method.

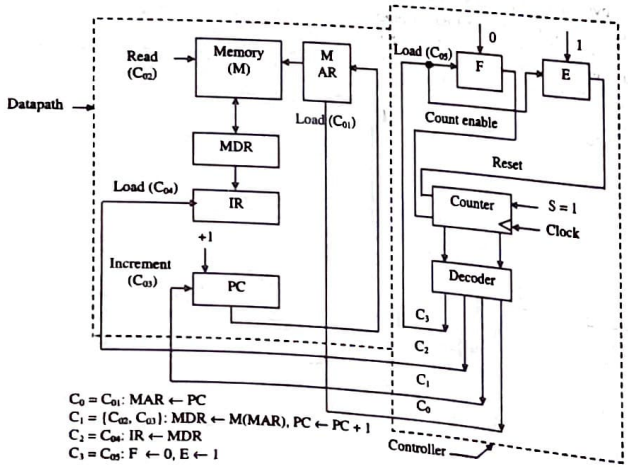


Figure 4.3: Hardwired Control Circuit Executing the Sequence of Micro-Operations

However, whatever design methodology may be employed, the hardwired control even for a moderately complex function usually leads to random control logic circuit. It may fail to exploit the regularity implied in the control function behaviour. With higher complexity of control function, it becomes extremely difficult to design and debug a hardwired controller. An alternative to hardwired control is the micro-programmed controller.

Ques 3) Describe the control of processor unit.

Ans: Control of Processor Unit

The control unit of the CPU consists of a small, high speed memory used to store temporary results and certain control information. The control unit directs the operation of input devices, output devices, ALU, main memory and secondary memory of a computer. It sends control signals to various parts of the computer.

Main functions of the control unit can be summarised as follows:

1) In general, it performs the data processing operations with the aid of programs prepared by the users and sends control signals to various parts of the computer systems.
2) It gives commands to transfer data from the input device to the memory to arithmetic logic unit.
3) It also transfers the results from ALU to the memory and then to the output devices.
4) It stores the program in the memory.
5) It fetches the required instructions from the main storage and analyses each instruction and hence deduces what operation is to be performed (that is decoding of instructions).

The block diagram of the control unit is shown in **figure 4.4**. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the Instruction Register (IR). The instruction register is shown again in **figure 4.4**, where it is divided into three parts:

1) The I bit,
2) The operation code, and
3) Bits 0 through 11.

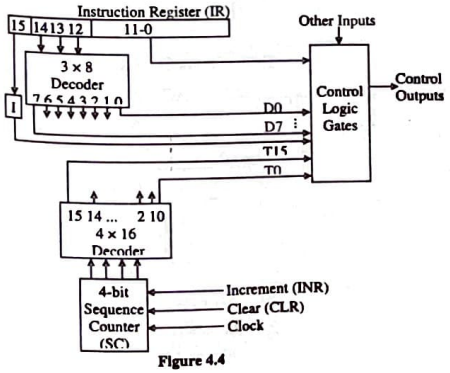


Figure 4.4

The operation code in bits 12 through 14 are decoded with a 3 × 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}. The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 × 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.

Ques 4) What is Programmable Logic Arrays (PLA) control? What are its advantages and disadvantages?

Ans: Programmable Logic Array (PLA) Control

A programmable logic array (PLA) is a kind of programmable logic device used to implement combinational logic circuits. The PLA has a set of programmable AND gate planes, which link to a set of programmable OR gate planes, which can then be conditionally complemented to produce an output. The first developed was the Programmable Logic Array (PLA). The general structure of a PLA is shown in **figure 4.5**.

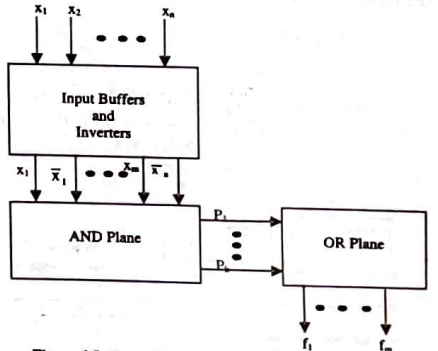


Figure 4.5: General Structure of a PLA

Based on the idea that logic functions can be realised in sum-of-products form, a PLA comprises a collection of AND gates that feeds a set of OR gates. As shown in the **figure 6.5**, the PLA's inputs $x_1, \ldots x_n$ pass through a set of buffers (which provide both the true value and complement of each input) into a circuit block called an AND plane, or AND array. The AND plane produces a set of product terms $P_1 \ldots P_k$. Each of these terms can be configured to implement any AND function of $x_1, \ldots x_n$. The product terms serve as the inputs to an OR plane, which produces the outputs $f_1 \ldots f_m$. Each output can be configured to realise any sum of $P_1, \ldots P_k$ and hence any sum-of-products function of the PLA inputs. A more detailed diagram of a small PLA is given in **figure 6.6**, which shows a PLA with three inputs, four product terms, and two outputs:

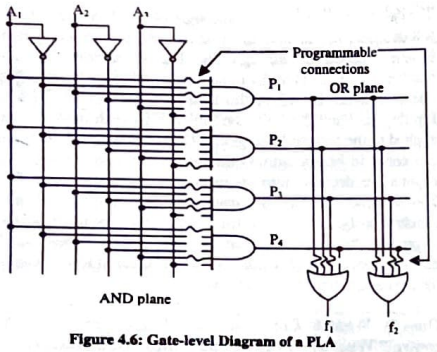


Figure 4.6: Gate-level Diagram of a PLA

Each AND gate in the AND plane has six inputs, corresponding to the true and complemented versions of the three input signals. Each connection to an AND gate is programmable; a signal that is connected to an AND gate is indicated with a wavy line, and a signal that is not connected to the gate is shown with a broken line. The circuitry is designed such that any unconnected AND-gate inputs do not affect the output of the AND gate. In commercially available PLAs, several methods of realising the programmable connections exist.

In **figure 4.7,** the AND gate that produces P_1 is shown connected to the inputs x_1 and x_2. Hence $P_1 = x_1x_2$. Similarly, $P_2 = x_1\bar{x}_3$, $P_3 = \bar{x}_1\bar{x}_2x_3$ and $P_4 = x_1x_3$. Programmable connections also exist for the OR plane. Output f_1 is connected to product terms P_1, P_2, and P_3. It therefore realises the function $f_1 = x_1x_2 + x_1\bar{x}_3 + \bar{x}_1\bar{x}_2x_3$. Similarly, output $f_2 = x_1x_2 + \bar{x}_1\bar{x}_2x_3 + x_1x_3$.

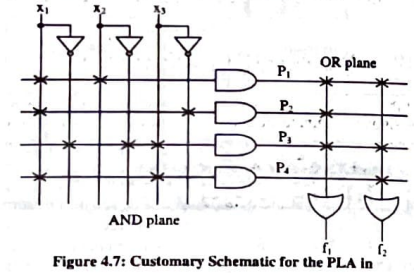


Figure 4.7: Customary Schematic for the PLA in

Although **figure 4.7** shows the PLA programmed to implement the functions described above, by programming the AND and OR planes differently, each of the outputs f_1 and f_2 could implement various functions of x_1, x_2, and x_3. The only constraint on the functions that can be implemented is the size of the AND plane because it produces only four product terms.

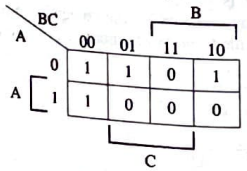
Note: Commercially available PLAs come in larger sizes thanwe have shown here. Typical parameters are 16 inputs, 32 product terms, and eight outputs.

Although **figure 4.6** illustrates clearly the functional structure of a PLA, this style of drawing is awkward for larger chips. Instead, it has become customary in technical literature to use the style shown in **figure 4.7.** Each AND gate is depicted as a single horizontal line attached to an AND-gate symbol. The possible inputs to the AND gate are drawn as vertical lines that cross the horizontal line. At any crossing of a vertical and horizontal line, a programmable connection, indicated by an ×, can be made. **Figure 4.7** shows the programmable connections needed to implement the product terms in **figure 4.6.**

Each OR gate is drawn in a similar manner, with a vertical line attached to an OR-gate symbol. The AND-gate outputs cross these lines, and corresponding programmable connections can be formed. The figure illustrates the programmable connections that produce the functions f_1 and f_2 from **figure 4.6.** The PLA is efficient in terms of the area needed for its implementation on an integrated circuit chip. For this reason, PLAs are often included as part of larger chips, such as microprocessors. In this case a PLA is created so that the connections to the AND and OR gates are fixed, rather than programmable. **For example,** implement the combinational circuit having the shown truth table, using PLA.

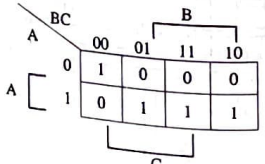
A	B	C	F_1	F_2
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Each product term in the expression requires an AND gate. To minimise the cost, it is necessary to simplify the function to a minimum number of product terms.



$$F_1 = \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{B}\bar{C}$$

$$\bar{F}_1 = AB + AC + BC$$



$$F_2 = AB + AC + \bar{A}\bar{B}\bar{C}$$

$$\bar{F}_2 = \bar{A}\bar{C} + \bar{A}B + A\bar{B}\bar{C}$$

Figure 4.8

Designing using a PLA, a careful investigation must be taken in order to reduce the distinct product terms. Both the true and complement forms of each function should be simplified to see which one can be expressed with fewer product terms and which one provides product terms that are common to other functions. The combination that gives a minimum number of product terms is:

$F_1 = AB + AC + BC$ or $F_1 = (AB + AC + BC)'$

$F_2 = AB + AC + A'B'C'$

This gives only 4 distinct product terms – AB, AC, BC, and A'B'C'. So the PLA table will be as follows:

Table 4.1: PLA Programming Table

	Product Term	Inputs			Outputs (C)	Outputs (T)
		A	B	C	F_1	F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$\bar{A}\bar{B}\bar{C}$	4	0	0	0	–	1

For each product term, the inputs are marked with 1, 0, or – (dash). If a variable in the product term appears in its normal form (unprimed), the corresponding input variable is marked with a 1.

A=1 in the Inputs column specifies a path from the corresponding input to the input of the AND gate that forms the product term.

A=0 in the Inputs column specifies a path from the corresponding complemented input to the input of the AND gate. A dash specifies no connection. The appropriate fuses are blown and the ones left intact form the desired paths. It is assumed that the open terminals in the AND gate behave like a 1 input.

In the Outputs column, a T (true) specifies that the other input of the corresponding XOR gate can be connected to 0, and a C (complement) specifies a connection to 1.

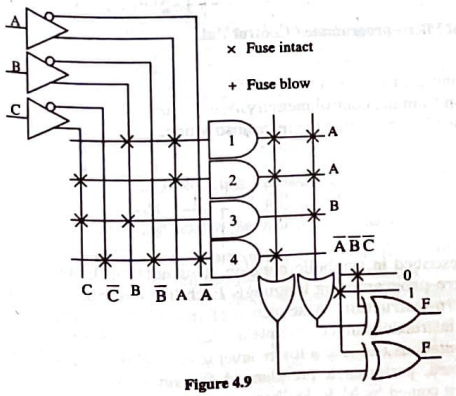


Figure 4.9

The output F_1 is the normal (or true) output even though a C (for complement) is marked over it. This is because F_1' in generated with AND-OR circuit prior to the output XOR. The output XOR complements the function F_1' to produce the true F_1 output as its second input is connected to logic 1.

Advantages of PLA

1) A PLA can have large N and M permitting implementation of equations that are impractical for a ROM because of the number of inputs, N, required.
2) A PLA has all of its product terms connectable to all outputs, overcoming the problem of the limited inputs to the PAL Ors.
3) Some PLAs have outputs that can be complemented, adding POS functions.

Disadvantages of PLA

1) Often, the product term count limits the application of a PLA.
2) Two-level multiple-output optimisation is required to reduce the number of product terms in an implementation, helping to fit it into a PLA.
3) Multi-level circuit capability available in PAL not available in PLA. PLA requires external connections to do multi-level circuits.

MICRO-PROGRAMMED CONTROL

Ques 5) Describe the micro-programmed control organisation? Or
Explain the design of micro-programmed control unit. What is the function of Micro-programmed Control Unit?

Ans: Micro-Programmed Control Unit Organisation

The control unit is responsible for initiating the sequence of micro-operations that comprise instructions. When these control signals originate in data stored in a special unit and constitute a program on the small scale, the control unit is **Micro-programmed.** The control function specifying a micro-operation is a binary variable whose active state could be either 1 or 0. In the variable's active state, the micro-operation is executed. The string of control variables which control the sequence of micro-operations is called a **control word.** Each control word contains signals to activate one of more micro-operations. When these words are retrieved in a sequence, a set of micro-operations are activated that will complete the desired task. The micro-operations specified in a control word are called a microinstruction.

Design of Micro Programmed Control Unit

A micro programmed control unit is an interconnection of following components (**figure 4.10**):

1) **Instruction Register:** Fetched instruction is stored in the Instruction Register. In certain machines, IR only holds the op-code of the instruction to be executed.

2) **Control Memory:** The control memory is normally implemented in the ROM and not in the main memory. This memory is utilised for keeping the micro-programs for all op-codes. In addition, control memory may contain routines for instruction fetch; interrupt initiation and other machine startup routines.

3) **Address-Computation Circuit:** The address-computation circuit computes the address of the next microinstruction.

4) **Micro Program Counter Register:** The role played by micro program counter register is similar to that of a program counter in CPU. The micro program counter holds the address of the next microinstruction to be executed.

5) **Microinstruction Buffer:** The microinstruction buffer holds the microinstruction, which is currently being executed.

6) **Sequencing Logic:** The sequencing logic plays key role in synchronising all these component of micro programmed control unit. The sequencing logic also plays an important role in the startup of machine. In normal circumstances, sequencing logic controls the control unit.

7) **Decoder of Microinstruction:** The decoder of microinstruction generates micro orders on the basis of microinstruction and op-code of the current instruction.

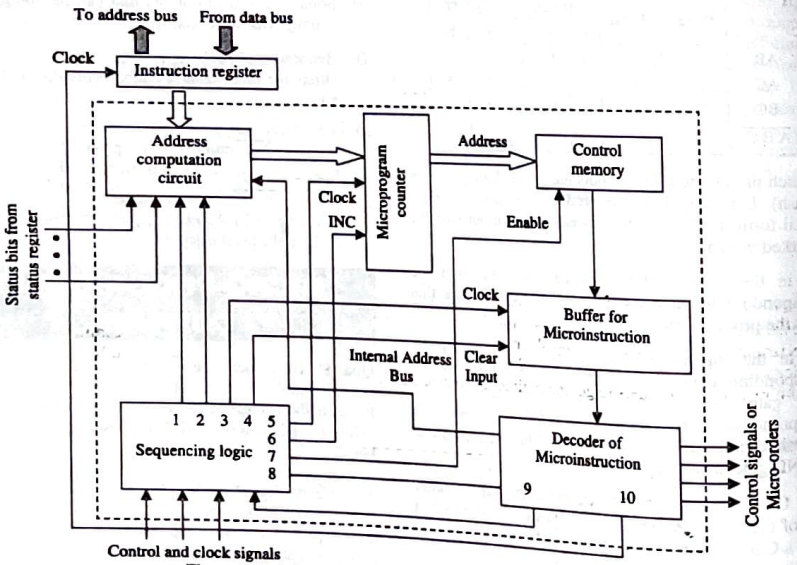


Figure 4.10: A Block Diagram of Micro-programmed Control Unit

Functions of Micro-programmed Control Unit

The two basic tasks performed by a Micro-programmed control unit are as follows:

1) **Microinstruction Sequencing:** Get the next microinstruction from the control memory.
2) **Microinstruction Execution:** Generate the control signals needed to execute the microinstruction.

Ques 6) What are the micro instructions? Describe.

Or

Discuss about the horizontal and vertical micro instructions.

Ans: Micro Instructions

In addition to use of control signals, each micro-operation is described in symbolic notation. This notation looks like a programming language. In fact it is a language, known as a micro-programming language. Each line describes a set of micro-operations occurring at one time and is known as a micro-instruction. A sequence of instructions is known as micro-program or firmware. In other words, the commands or instructions that constitute a micro-program are known as microinstructions. Microinstruction is an instruction for micro-operations and is at a lower level than machine instructions. Micro-program is a group of micro-instructions that if executed, performs a pre-planned function; mainly used to implement machine instructions. The term micro-program was first coined by M.V. Wilkes.

A microinstruction consists of four functional parts:

1) Micro-operation fields (F1, F2, etc.)
2) Condition for branching (CD)
3) Branch field (BR)
4) Address field (AD)

Figure 4.11 shows the format for a microinstruction of 20 bits. The fields F1, F2 and F3 specify the micro-operations, which are to be performed. If a control word needs to specify only one micro-operation then F2 and F3 field contains value 000. The CD field specifies the conditions for branching like Unconditional branch, indirect address bit, overflow and zero value in accumulator. 2 bits in CD can form 4 combinations, i.e., can specify 1 of 4 possible conditions.

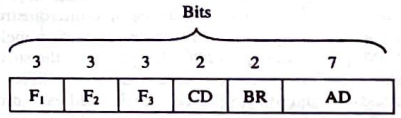


Figure 4.11: Format of Microinstruction

The two widely used formats used for microinstructions are described below:

1) **Horizontal Microinstruction:** In the horizontal microinstruction, each bit of the microinstruction represents a micro-order or a control signal, which directly controls a single bus line or sometimes a gate in the machine. However, the length of such a microinstruction may be hundreds of bits. A typical horizontal microinstruction with its related fields is shown in the **figure 4.12**.

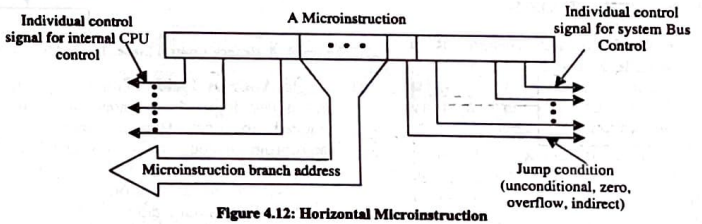


Figure 4.12: Horizontal Microinstruction

2) **Vertical Microinstructions:** In vertical microinstructions many similar control signals can be encoded into few microinstruction bits. **For example**, for 16 ALU operations, which may require 16 individual micro-orders in horizontal microinstruction, only 4 encoded bits are needed in vertical microinstruction. Similarly, in a vertical microinstruction only 3 bits are needed to select one of the 8 registers. However, these encoded bits need to be passed from respective decoders to get the individual control signals. This is shown in **figure 4.13**.

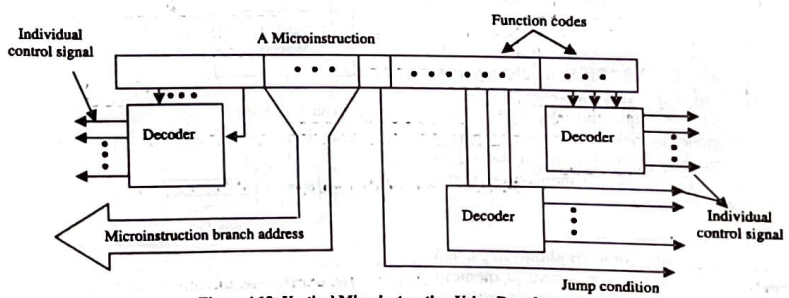


Figure 4.13: Vertical Microinstruction Using Decoders

Ques 7) Distinguish between horizontal and vertical microinstructions. **(2017 [05])**

Or

Compare vertical and horizontal microinstructions formats, giving examples. **(2018 [10])**

Ans: Difference between Horizontal and Vertical Microinstruction

Table 4.2 shows the difference between Horizontal and Vertical Microinstruction:

Table 4.2: Difference between Horizontal and Vertical Microinstruction

Horizontal	Vertical
Long formats	Short formats
Ability to express a high degree of parallelism	Limited Ability to express parallelism micro-operations
Little encoding of the control information	Considerable encoding of the control information
Useful when higher operating speed is desired	Slower operating speed

Ques 8) What is microprogram sequencing? Give various sequencing techniques.

Ans: Micro-program Sequencing

A micro-program sequencer for generating in a proper sequence the addresses of the successive microinstructions used in executing a given machine instruction includes a PROM next address generator that produces the successive addresses. The successive addresses are utilized as the successive microinstructions. Each address produced includes a normal next address, but this normal next address may be alterable by address alteration signals that are generated in response to a number of sensed conditions within the computer and in response to predetermined machine instruction register bits. The address alteration of a normal next address, if required, is accomplished within the same clock period in which the normal next address is initially formed, permitting jump or branch instructions to be performed as rapidly as normal instructions.

Two concerns are involved in the design of a microinstruction sequencing technique:

1) **Size of Microinstruction:** This concern is obvious that is minimising the size of the control memory reduces the cost of that component.
2) **Address Generation Time:** This concern is simply a desire to execute microinstructions as fast as possible. This implies that the address of next microinstruction should be calculated at a fast rate.

In executing a microprogram, the address of the next microinstruction to be executed is in one of these categories:

1) Determined by Instruction Register,
2) Next sequential address,
3) Branch.

The first category occurs only once per instruction cycle, just after an instruction is fetched. The second category is the most common in most designs. However, the design cannot be optimised just for sequential access. Branches, both conditional and unconditional, are a necessary part of a microprogram. The purpose of microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.

Sequencing Techniques

Based on the current microinstruction, condition flags, and the contents of the instruction register, a control memory address must be generated for the next microinstruction. There are different techniques used for this.

These are grouped into three categories as below:

1) **Two Address Fields:** The simplest approach is to provide two address fields in each microinstruction. **Figure 4.14** suggests how this information to be used. A multiplexer is provided that serves as a destination for both address fields plus the instruction register. Based on an address-selection input, the multiplexer transmits either the opcode or one of the two addresses to the Control Address Register (CAR). The CAR is subsequently decoded to produce the next microinstruction address. The address-selection signals are provided by a branch logic module whose input consists of control unit flags plus bits from the control portion of the microinstruction.

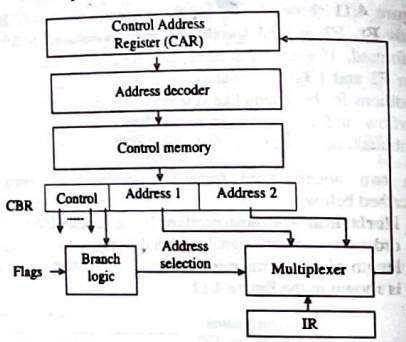


Figure 4.14: Branch Control Logic, Two Address Fields

2) **Single Address Field:** With some modified circuit and added logic the number of addresses can be reduced to one. Here, a new register called microprogram counter, as introduced in **figure 4.15** can be used. In, this case, the next microinstruction address can be the address of:
   i) Next Sequential Address
   ii) Instruction Register Code
   iii) Address Field

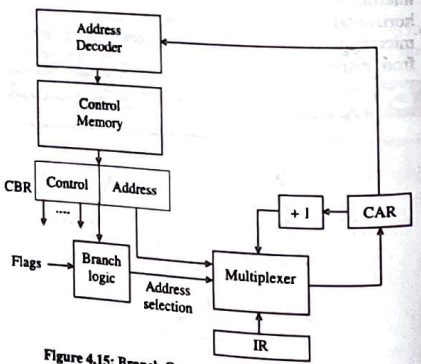


Figure 4.15: Branch Control Logic, Single Address Field

The address selection signal, which will be based on branch logic, can indicate which of the above address is to be selected.

Although this scheme saves some space, yet the space provided for even one address is not used very often. Thus, there remains some inefficiency in the coding scheme of the microinstruction.

3) **Variable Format:** Another approach, which can be used, is to provide variable format. In such a case two formats are used (**figure 4.16**):

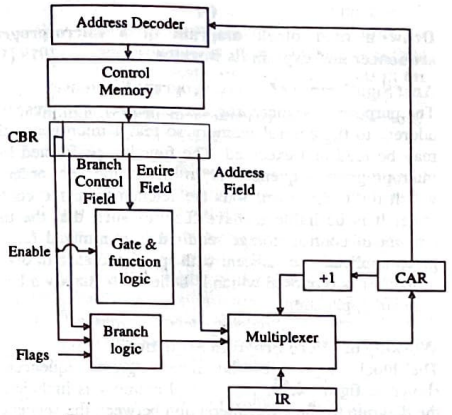


Figure 4.16: Branch Control Logic, Variable Format

i) In one format, the remaining bits are used to activate control signals. With the first format, the next address is either the next sequential address or an address derived from the instruction register.

ii) In second format, some bits drive the branch logic modules and the remaining bits provide the address. Either a conditional or unconditional branch is being specified.

The limitation of this approach is that one entire cycle is consumed with each branch microinstruction.

Ques 9) What is micro-sequencer?

Or

What are the different micro sequencer design steps?

Ans: Micro-Sequencer

A sequencer or micro-sequencer generates the addresses used to step through the microprogram of a control store. It is used as a part of the control unit of a CPU or as a stand-alone generator for address ranges, A sequencer or microsequencer is a part of the control unit of a CPU. It generates the addresses used to step through the microprogram of a control store. Usually the addresses are generated by some combination of a counter, a field from a microinstruction and some subset of the instruction register. A counter is used for the typical case, that the next microinstruction is the one to execute.

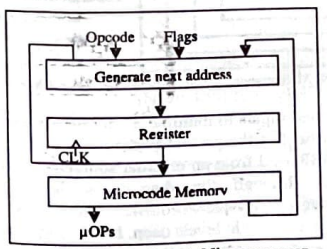


Figure 4.17: Generic Microsequencer

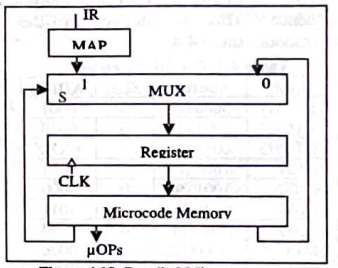


Figure 4.18: Detailed Microsequencer

A microsequencer is also designed as a finite state machine. **Figure 4.17** and **4.18** shows the generic and detailed microsequencer. **Figure 4.18** shows the basic layout of the very simple microsequencer.

Microsequencer Design Steps

The steps of microsequencer design steps are as below:

1) Assign each state of the FSM (Finite State Machine) to an address in microcode:
   i) Start by assigning the address of the first state of each of the execute routines. This will decide the mapping logic. One needs to be able to find a mapping from the instruction to the microcode address that ideally:
      a) Is easy to implement for the initial state of each execute routine.
      b) Allows for consecutive addresses for the states in fetch and execute routines, if possible, for readability and ease of debugging.
   ii) This gives the set of addresses shown in the first of the **tables 4.3**:

Table 4.3: State Addresses for the VS_CPU

State	Address
FETCH1	0000 (0)
FETCH2	0001 (1)
FETCH3	0010 (2)
ADD1	1000 (8)
ADD2	1001 (9)
AND1	1010 (10)
AND2	1011 (11)
JMP1	1100 (12)
INC1	1110 (14)

2) Next one has to design the microcode so that it will sequence through the states correctly and generate the correct micro-operations (µOps) at each step. Each microcode entry in the sequence will have three fields and ultimately be of the format:

SEL	µOperations	Address

   i) First one decides on the value of the select (SEL) field for each state. '0' will use the address field as the next address to access in the microcode, '1' will select the mapping from the Instruction. In this case SEL only needs to be '1' when user goes from the end of the FETCH states

(FETCH3) to the start of the current execute sequence. This is shown in the 'Partial Microcode' **table 4.4**:

Table 4.4: Partial Microcode

State	Address	SEL	ADDR
FETCH1	0000 (0)	0	0001
FETCH2	0001 (1)	0	0010
FETCH3	0010 (2)	1	XXXX
ADD1	1000 (8)	0	1001
ADD2	1001 (9)	0	0000
AND1	1010 (10)	0	1011
AND2	1011 (11)	0	0000
JMP1	1100 (12)	0	0000
INC1	1110 (14)	0	0000

ii) One now needs to deal with the μops. These are taken from the RTL for CPU, listing each unique micro-operation such as AR ← PC.

iii) Provide each μop with a mnemonic, e.g., ARPC, as it is easier than writing AR ← PC all the time and the mnemonic will become the bit-field name in the microcode **table 4.5**.

Table 4.5: Micro-Ops and Their Mnemonics

Mnemonic	Micro-Operation
ARPC	AR←PC
ARDR	AR←DR[5..0]
PCIN	PC←PC + 1
PCDR	PC←DR[5..0]
DRM	DR←M
IRDR	IR←DR[7..6]
PLUS	AC←AC + DR
AND	AC←AC ∧ DR
ACIN	AC←AC + 1

iv) One can now complete the microcode for the CPU by entering the appropriate value in each bit-field for the μops. For each state, when an RTL transfer must take place, we enter a '1' in the bit-field for that micro-operation and that state. This gives us the following completed table:

Horizontal Microcode

State	Address	SEL	ARPC	ARDR	PCIN	PCDR	DRM	IRDR	PLUS	AND	ACIN	ADDR
FETCH1	0000	0	1	0	0	0	0	0	0	0	0	0001
FETCH2	0001	0	0	0	1	0	1	0	0	0	0	0010
FETCH3	0010	1	0	1	0	0	0	1	0	0	0	XXXX
ADD1	1000	0	0	0	0	0	1	0	0	0	0	1001
ADD2	1001	0	0	0	0	0	0	0	1	0	0	0000
AND1	1010	0	0	0	0	0	1	0	0	0	0	1011
AND2	1011	0	0	0	0	0	0	0	0	1	0	0000
JMP1	1100	0	0	0	0	1	0	0	0	0	0	0000
INC1	1110	0	0	0	0	0	0	0	0	0	1	0000

Ques 10) What is the significance of a micro program sequencer? Explain its working with the help of a diagram. **(2017 [10])**

Or

Describe the purpose of microprogram sequencing. How is it carries out? **(2018 [10])**

Or

Draw a neat block diagram of a micro-program sequencer and explain its working. **(2019 [10])**

Ans: Significance of Micro Program Sequencer

The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed. The function performed by a microprogram sequencer is to determine the order in which the microinstructions are fetched from the control store. It is desirable to have features such that the total amount of control storage required is minimized for any given application consistent with speed and ease of use. A sequencer is proposed which is believed to satisfy a broad range of applications.

Working of Micro Program Sequencer

The block diagram of the microprogram sequencer is shown in **figure 4.19**. The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it. There are two multiplexers in the circuit. The first multiplexer selects an address from one of four sources and routes it into a control address register CAR. The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit. The output from CAR provides the address for the control memory. The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine registers SBR.

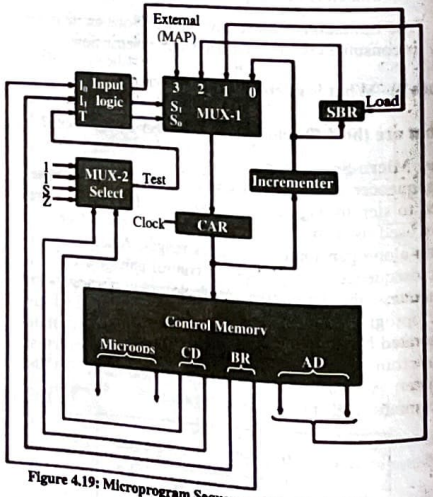


Figure 4.19: Microprogram Sequencer for a Control Memory

The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction. Although the diagram shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the same time.

A push and pop operation, in conjunction with a stack pointer, stores and retrieves the return address during the call and return microinstructions. The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer. If the bit selected is equal to 1, the T(test) variable is equal to 1; otherwise, it is equal to 0. The T value together with the two bits from the

BR (branch) field go to an input logic circuit. The input logic in a particular sequencer will determine the type of operations that are available in the unit. Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations. With three inputs, the sequencer can provide up to eight address sequencing operation. Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations.

The input logic circuit in has three inputs, I_0, I_1, and T, and three outputs, S_0, S_1, and L. Variables S_0 and S_1 select one of the source addresses for CAR. Variable L enables the load input in SBR. The binary values of the two selection variables determine the path in the multiplexer.

For example, with S_1, S_0 = 10, multiplexer input number 2 is selected and establishes a transfer path from SBR to CAR. Note that each of the four inputs as well as the output of MUX 1 contains a 7-bit address.

The truth table for the input logic circuit is shown in **table 4.6**. Inputs I and I_0 are identical to the bit values in the BR field. the bit values for S_1 and S_0 are determined from the stated function and the path in the multiplexer that establishes the required transfer. The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bi-condition is satified (T = 1). The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$

$$S_0 = I_1 I_0 + I_1' T$$

$$L = I_1' I_0 T$$

Table 4.6: Input Logic Truth Table for Microprogram Sequencer

BR Field		Input I_1	Input I_0	Input T	MUX 1 S_1	MUX 1 S_0	Load SBR L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	1	1	1
1	0	1	0	×	1	0	0
1	1	1	1	×	1	1	0

The circuit can be constructed with three AND gates, an OR gate, and an inverter.

Ques 11) Explain how a microprogram is executed?

Or

What is Microinstruction Encoding? What are the main characteristics of un-encoded and encoded microinstructions?

Ans: Microprogram Execution

The microinstruction cycle is the basic event on a Micro-programmed processor. Each cycle is made up of two parts:

1) Fetch
2) Execute

Each computer instruction fetched from the main memory leads to initiation of a series of microinstructions from the control memory. These microinstructions issued micro-orders to the CPU for instruction fetch, calculation of effective address of operands, and execution of the instruction and then prepare it again for fetching next instruction from the main memory.

Thus, for each instruction, one must determine the series of microinstructions, which will be needed to get those instructions executed. This series of microinstruction will be different for different instructions. The execution of microinstruction simply means generation of control signals. These control signals may drive the CPU (internal control signals) or the system bus. The format of microinstruction and its contents determines the complexity of a logic module, which executes a microinstruction.

Microinstruction Encoding

One of the key features, which are incorporated in microinstruction, is the encoding of microinstructions. The micro programmed control unit designs are neither completely unencoded nor highly encoded. They are slightly coded, in general, to reduce the width of control memory and microprogramming efforts. For encoding, a microinstruction is divided into a set of fields such that:

1) Only one field, thus, making the fields independent of each other, can activate a specific control signal.
2) Each field represents a pattern of control signals, which in turn depicts an action. As the fields are independent, therefore, the actions depicted by different fields can be performed simultaneously, that is, in parallel.
3) A specific field specifies actions, which are mutually exclusive, that is, only one of these actions can occur at a time.

In Wilkes control unit, each bit of information either generates a control signal or form a bit of next instruction address. Now, assume that a machine needs N total number of control signals. If the Wilkes scheme is followed N bits are required, one for each control signal in the control unit. While dealing with binary control signals, an N bit microinstruction can represent 2^N combinations of control signal. If only 2^k combinations are needed then only k bit are needed to encode microinstruction. The k bit microinstruction is an extreme encoded microinstruction.

Characteristics of Un-Encoded Microinstructions

1) One bit is needed for each control signal; therefore, number of bits required in a microinstruction is high.
2) It presents a detailed hardware view, as control signal needs can be determined.

3) Since each of the control signal can be controlled individually, therefore, these microinstructions are difficult to program. However, concurrency can be exploited easily.
4) Almost no control logic is needed to decode the instruction as there is one to one mapping of control signal to a bit of microinstruction. Thus, execution of microinstruction and hence the microprogram is faster.

Characteristics of Encoded Microinstructions

1) It provided an aggregated view that is a higher view of the CPU as only an encoded sequence can be used for microprogramming.
2) The encoding helps in reduction in programming burden; however, the concurrency may not be exploited to the fullest.
3) Complex control logic is needed, as decoding is a must. Thus, the execution of microinstruction has propagation delay through gates. Therefore, execution of microprogram takes longer time than that of un encoded microinstruction.
4) The highly encoded microinstructions are aimed at optimising programming effort.

Ques 12) Discuss the micro-programmed CPU organisation with diagram.

Or

Explain micro programmed CPU organisation with the help of a diagram. (2017 [10])

Or

Explain the organisation of a micro-programmed computer with a block diagram. (2019 [10])

Ans: Micro-Programmed CPU Organisation

A digital computer consists of a Central Processor Unit (CPU), a memory unit, and input-output devices. The CPU can be classified into two distinct but interactive functional sections. One section is the processing section and the other is the control section. A processor unit is a useful device for constructing the processor section of a CPU. The microprogram sequencer is a convenient element for constructing a microprogram control for a CPU. A block diagram of a micro-programmed computer is shown in **figure 4.20**. It consists of a memory unit, two processor units, a microprogram sequencer, a control memory, and few other digital functions.

The memory unit stores the instructions and data supplied by the user via an input device. The data processor manipulates the data, and the address processor manipulates the address information received from memory. The two processors can be combined into one unit, but sometimes it is convenient to separate them in order to provide a distinct bus for the memory address. An instruction extracted from memory during the fetch cycle goes into the instruction register. The instruction-code bits in the instruction register specify a macro-operation for control memory.

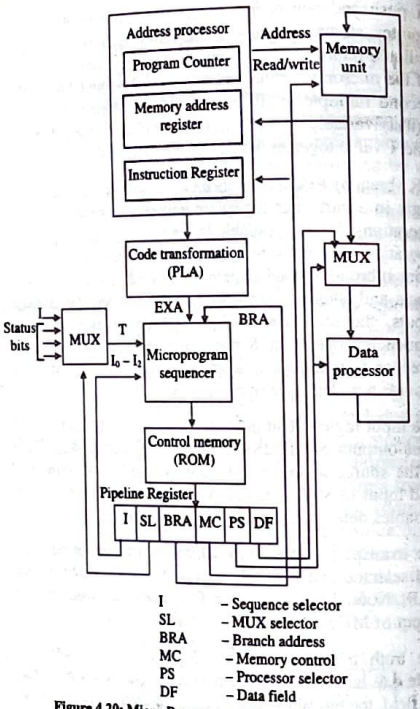


Figure 4.20: Micro-Programmed Computer Organisation

A code transformation is sometimes required to convert the operation-code bits of an instruction into a starting address for the control memory.

This code transformation constitutes a mapping function and can be implemented with a ROM or a PLA (Programmable Logic Array – It is an LSI (Large Scale Integration) device that can implement any complex combinational circuit).

The mapping concept provides a flexibility for adding instructions or macro-operations for control memory as the need arises. The address generated in the code transformation mapping function is applied to the External Address (EXA) input of the sequencer.

The microprogram control unit consists of the sequencer, a control memory for storing the microinstructions, a multiplexer, and a pipeline register. The multiplexer selects one of many status bits and applies it to the T (test) input of the sequencer.

One of the inputs to the multiplexer is always I to provide an unconditional branch operation. The pipeline register is not always necessary, because the outputs from control memory can go directly to the control inputs of the various units in the CPU.

However, a pipeline register speeds up the control operation. It allows the next address to be generated and the output of control memory to change while the current control word in the pipeline register initiates the micro-operations given by the present microinstruction.

A possible microinstruction format for the control memory is illustrated within the pipeline register. The I field consists of three bits and supplies the input information for the sequencer. The SL field selects a status bit for the multiplexer.

The BRA field is the address field of the microinstruction and supplies a Branch Address (BRA) to the sequencer. These three fields of a microinstruction provide information to the sequencer to determine the next address for control memory.

The sequencer generates the next address and the control memory reads the next microinstruction while the present micro-operations are being executed in the other units of the CPU.

The other three fields in the microinstruction are for controlling the micro-operations in the processor and memory units. The Memory Control (MC) field controls the address processor and the read and write operations in the memory unit.

The Processor Select (PS) field controls the operations in the data processor unit. The last field is a Data Field (DF) used to introduce constants into the processor.

The procedure of introducing data into the system from the control memory is a frequently used technique in many micro-programmed systems. Outputs from the data field may be used to set up control registers and introduce data in processor registers.

For example, a constant in the data field may be added to a processor register to increment its contents by a specified value. Another use of the data field is in setting a sequence counter to a constant value.

The sequence counter is then used to count the number of times a microprogram loop is traversed, as is usually required in a multiply or divide routine.

Once the hardware configuration of a micro-programmed CPU is established, the designer can use it to construct any one of many possible computer configurations. First, the instruction set for the computer is formulated and then a microprogram is written for control memory.

One can change the microprogram in control memory if a different computer with a different set of instructions is desired. No hardware changes are required if the computer's specifications change; the change is only in the control memory ROM.

This involves removing the present ROM from its socket and replacing it with another unit with a different microprogram.

Ques 13) What is difference between hardwired and micro-programmed control.

Ans: Comparison between Hardwired and Micro-programmed Control

Table 4.7 shows the difference between Hardwired and Micro-programmed Control:

Table 4.7: Difference between Hardwired and Micro-Programmed Control

Basis	Hardwired Control	Micro-programmed Control
Speed	Fast	Slow
Implementation	Hardware	Software
Flexibility	No flexibility	More flexibility
Ability to handle large/complex instruction sets	Somewhat difficult	Easier
Ability to support operating system and diagnostic features	Very difficult	Easy
Design process	Difficult for more operation	Easy
Memory	No memory used	Control memory used (RAM or ROM)
Chip area efficiency	Uses least area	Uses more area

Ques 14) Describe the control of processor unit in detail.

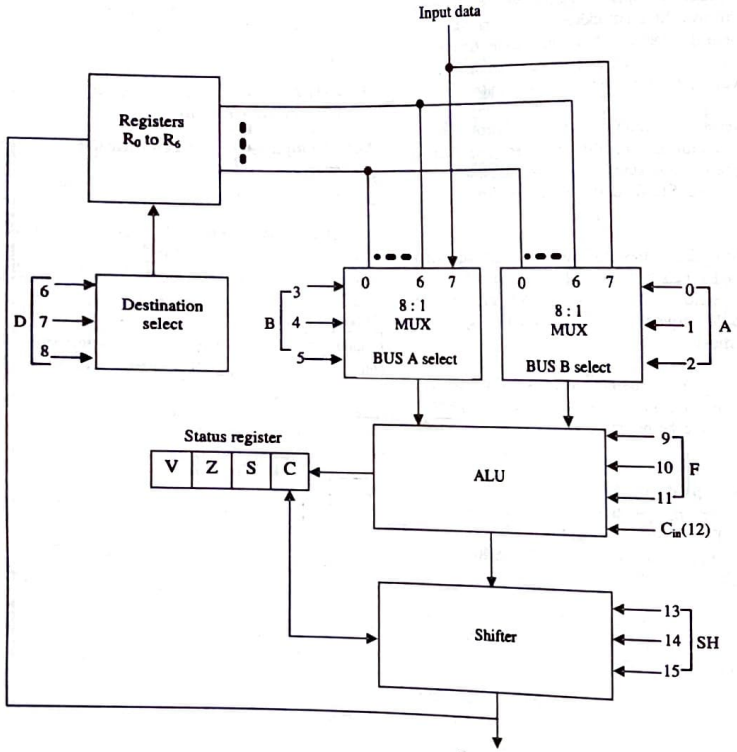
Or

With the help of a block diagram, describe a complete processor unit with all components and appropriate control variables. Show with an example, how a control word for the processor can be defined. (2017 [10])

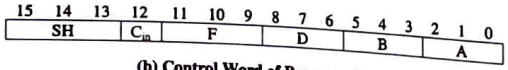
Ans: Control of Processor Unit/Complete Processor Unit

The **figure 4.21** shows the block diagram of processor unit with selection variables. These selection variables control the micro-operations executed within the processor. As shown in the **figure 4.21,** the selection variables control the source for A bus, the source for B bus, ALU function, shifter operation and destination register. A and B inputs of ALU have eight possible input sources: seven from R_0 to R_6 and one external input. Two 8: 1 multiplexers are used to select the desired source for ALU.

The 3-bit field A and 3-bit field B of control word is used to select a source for A and B inputs of ALU, respectively. The output of the ALU goes either to output data or to the destination register through the shifter. The D field selects the destination register. The F field, together with the bit C_{in}, selects a function for the ALU and the SH field selects the type of shift operation to be performed by the shifter.



(a) Block Diagram of Processor Unit



(b) Control Word of Processor Unit

Figure 4.21

The Table lists the functions of all selection variables. The table also gives the 3-bit binary code for each of the five fields A, B, D, F and SH.

Table 4.8: Function of control variables for the processor

Binary Code	Function of selection variables					
	A	B	D	F with $C_{in} = 0$	F with $C_{in} = 1$	SH
000	R_0	R_0	R_0	A, C ← 0	A + 1	No Shift
001	R_1	R_1	R_1	A + B	A + B + 1	Shift right, $I_R = 0$
010	R_2	R_2	R_2	A − B − 1	A − B	Shift left, $I_L = 0$
011	R_3	R_3	R_3	A − 1	A, C ← 1	0's to output bus
100	R_4	R_4	R_4	A ∨ B	–	–
101	R_5	R_5	R_5	A ⊕ B	–	Rotate left with C
110	R_6	R_6	R_6	A ∧ B	–	Rotate right with C
111	Input Data	Input Data	None	$\overline{A}$	–	–

All selection variables collectively form the 16-bit control word shown in **figure 6.21 (b).** This control word is used to specify a micro-operation for the processor unit. The **table 4.9** list some examples of micro-operations and corresponding control word for processor.

Table 4.9: Examples of micro-operations for processor

Micro-operation	Control word						Code in Hex	Function
	BH	C_{in}	F	D	B	A		
$R_0 \leftarrow R_1 + R_2$	000	0	001	000	010	001	0211	Add R_1 and and R_2 and store result in R_0
$R_2 - R_1$	000	1	010	111	001	010	15CA	Compare R_2 and R_1
$R_5 \leftarrow R_2$	000	0	000	101	111	101	017D	Transfer R_2 to R_5
$R_2 \leftarrow 0$	011	0	000	011	000	000	60C0	Clear R_3
$R_4 \leftarrow$ Shl R_4	010	0	000	100	111	100	413C	Shift left R_4 with $I_L = 0$
$R_6 \leftarrow$ RR R_6	110	0	100	110	110	110	C9B6	Rotate right R_6 with carry

Ques 15) Draw the block diagram for the hardware that implements the following statement x + yz: AR ← AR + BR where AR and BR are two n-bit registers and x, y, and z are control variables. Include the logic gates for the control function. (The symbol + designates an OR operation in a control or Boolean function and an arithmetic plus in a micro operation. **(2018 [10])**

Ans:

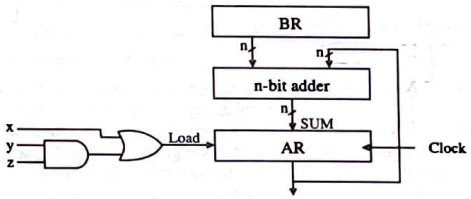


Figure 4.22

Module 5

Input/Output Organisation

INPUT/OUTPUT ORGANISATION

Ques 1) Discuss about the input/output organisation.
Or
Describe the Input/output interface with the help of an example.

Ans: Input/Output Organisation
I/O operations are accomplished through external devices that provide a means of exchanging data between external environment and computer. An external device attaches to the computer by a link to an I/O module. Computer is an electronic machine which communicates with the users through I/O device, such as printer and keyboard. These external devices are called **peripherals**. The computer will be of no use unless it is able to communicate with the outside world. Input/output devices are required for users to communicate with the computer.

In simple terms, input devices bring information into the computer and output devices bring out information to the user. These input/output devices are also known as **peripherals** since they surround the CPU and memory of computer system.

Input/Output Interface
An interface is the connecting circuitry that forms a shared boundary to link the computer to other peripheral devices in order to allow the synchronization of the transmission of digital information between computer and peripheral devices. An I/O interface is required whenever the I/O device is driven by the processor. The interface must have necessary logic to interpret the device address generated by the processor.

I/O interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to the system need special communication links for interfacing with CPU.

An **example** of an I/O interface unit is shown in **figure 5.1**. It consists of two data registers called ports, a control register, a status register, bus buffers and timing and control circuits. The four registers communicate directly with the I/O device attached to the interface. The I/O data to and from the device can be transferred into either port A or port B. Port A may be defined as an input port and port B may be defined as an output port.

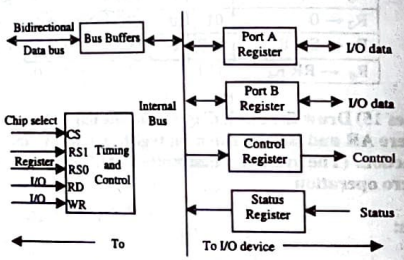


Figure 5.1: Example of I/O Interface

The output device such as magnetic disk transfers data in both directions. So bidirectional data bus is used. CPU gives control information to control register. The bits in the status register are used for status conditions. It is also used for recording errors that may occur during the data transfer. The bus buffers use the bidirectional data bus to communicate with the CPU. A timing and control circuit is used to detect the address assigned to the bus buffers.

CS	RS1	RS0	Register selected
0	X	X	None: data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Ques 2) What are the functions of I/O Module?

Ans: Functions of I/O Module
The major functions of an I/O module are:

1) **Processor Communication:** This involves the following tasks:
   i) **Exchange of Data:** This module transfer data between the processor and I/O module.
   ii) **Command Decoding:** I/O module accepts commands sent from the processor. **For example,** the I/O module for a disk drive may accept the following commands from the processor: READ SECTOR, WRITES SECTOR, SEEK track etc.
   iii) **Status Reporting:** The device must be able to report its status to the processor, **e.g.,** disk drive busy, ready etc. Status reporting may also involve reporting various errors.
   iv) **Address Recognition:** Each I/O device has a unique address and the I/O module must recognize this address.

2) **Device Communication:** The I/O module must be able to perform device communication such as status reporting.
3) **Control and Timing:** The I/O module must be able to coordinate the flow of data between the internal resources (such as processor, memory) and external devices.
4) **Data Buffering:** This is necessary as there is a speed mismatch between the speed of data transfer between processor and memory and external devices. Data coming from the main memory are sent to an I/O module in a rapid burst. The data is buffered in the I/O module and then sent to the peripheral device at its rate.
5) **Error Detection:** The I/O module must also be able to detect errors and report them to the processor. These errors may be mechanical errors (such as paper jam in a printer) or changes in the bit pattern of transmitted data. A common way of detecting such errors is by using parity bits.

Ques 3) Describe the structure of the I/O module with diagram.

Ans: Structure of I/O Module
Although, there is no standard structure of I/O module, but certain general characteristics and the structure of I/O module are given below:

1) **I/O Logic Mechanism:** There is a need of I/O logic, which should interpret and execute the dialogue between the CPU and I/O module. Therefore, there need to be control lines between CPU and this I/O module. In addition, the address lines are needed to recognize the address of the I/O module and its specific device.
2) **Data Lines:** Connecting I/O module to system bus. These lines serve the purpose of data transfer.
3) **Data Registers:** These may act as buffer between CPU and I/O module. The data is stored or buffered in data registers.
4) **Interface of I/O Module:** Device has interface logic to control the device, to get the status information and transfer of data.
5) **Status Registers:** The status registers are used to indicate the current state of a device, e.g., the device is busy, the system BUS is busy, the device is out of order, etc.

Figure 5.2 shows the diagram of a typical I/O module which in addition to all the above have status/control registers which might be used to pass on the status information or can store Control information.

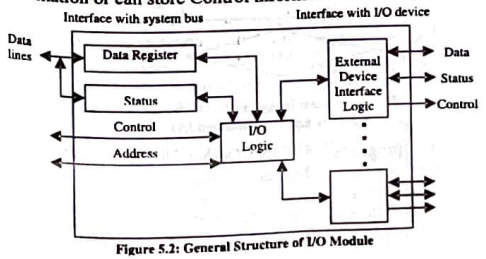


Figure 5.2: General Structure of I/O Module

Ques 4) Explain how I/O devices can be accessed?
Or
Give the difference between memory-mapped I/O and isolated mapped I/O devices.
Or
Describe about memory-mapped input/output and I/O mapped input/output.
Or
Discuss the different methods of Interfacing I/O.

Ans: Accessing I/O Devices/Methods of Interfacing I/O
When there is a shared memory between the processor and the I/O then the instructions generated are decoded by the processor and a control signal is activated. I/O devices can be interfaced into computer system in two ways as shown below:

1) **Memory-Mapped Input/Output:** With memory-mapped I/O, there is a single address space for memory locations and I/O devices. The processor treats the status and data registers of I/O modules as memory locations and uses the same machine instructions to access both memory and I/O devices. **For example,** with 10 addresses lines, a combined total of 2^{10} = 1024 memory locations and I/O addresses can be supported, in any combination.

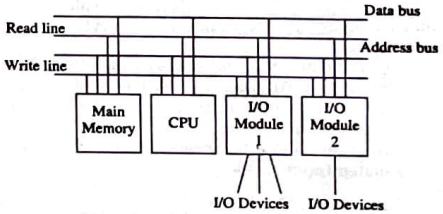


Figure 5.3: Structure of Memory Mapped I/O

If there is a need of single address space for memory locations and I/O devices, i.e., the CPU treats the status and data registers of I/O module as memory locations, then memory and I/O devices can be accessed using the same instructions. This is referred to as memory mapped I/O. For a memory mapped I/O only a single READ and a single WRITE line is needed for memory or I/O module read or writes operations.

Figure 5.4 illustrates how the command registers of a hard disk that uses 32-bit command registers might be handled by memory-mapped wants it to do. The platter, track, and sector registers encode the location of the data that the processor wants to read or write.

Command Registers

Base Address	Command
Base Address + 4	Platter
Base Address + 8	Track
Base Address + 12	Sector
Base Address + 16	Start Address
Base Address + 20	Size

Figure 5.4: Memory-Mapped I/O

The start address field tells the disk where in the main memory the data for the request should be read from (in the case of a write to the disk) or written to (in the case of a read from the disk), and the size field tells the disk how much data to transfer. These registers are used for DMA memory transfers.

2) **Isolated or I/O Mapped Input/Output:** With memory-mapped Input/Output, a single read line and a single write lines are needed on the bus. Alternatively, the bus may be equipped with memory read and write plus input and output command lines. Now, the command line specifies whether the address refers to a memory location or an I/O device.

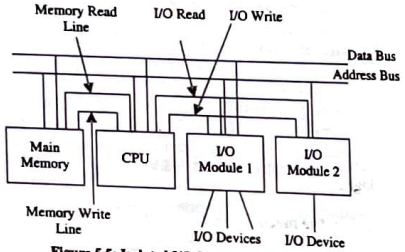


Figure 5.5: Isolated I/O Scheme or I/O Mapped I/O

The full range of addresses may be available for both. Again, with 10 address lines, the system may now support both 1024 memory locations and 1024 I/O addresses. Since the address space for Input/ Output is isolated from that for memory, this is referred to as **isolated Input/Output**.

Hence, in I/O Mapped I/O the I/O devices and memory are addressed separately. There are separate control lines for memory and I/O device read or write operations, thus, a memory reference instructions do not affect an I/O device. Here separate I/O instructions are needed which cause data transfer between addressed I/O module and CPU.

Steps in Read and Write Operation during Interfaces

In either case, the I/O module must engage in a dialogue with the peripheral. In general terms, the dialogue for a write operation is as follows:

i) The I/O module sends a control signal requesting permission to send data.
ii) The peripheral acknowledges the request.
iii) The I/O module transfers data (one word or a block depending on the peripheral).
iv) The peripheral acknowledges receipt of the data.

Key to the operation of an I/O module is an internal buffer that can store data being passed between the peripheral and the rest of the system. This buffer allows the I/O module to compensate for the differences in speed between the system bus and its external lines.

MODES OF TRANSFER

Ques 5) What do you mean by modes of transfer? List the various modes of transfer of data.

Ans: Modes of Transfer

In a computer, data transfer takes place between two devices such as CPU and memory, CPU and I/O devices, and memory and I/O devices. Usually, memory is compatible with microprocessor while input and output devices are not. A computer is interfaced with a number of input/output devices of different speed. In such a situation a slow I/O device may not be ready to transfer data when microprocessor issues instruction for this purpose. To solve the problem of speed mismatch a number of data transfer schemes have been developed.

There are three **modes of data transfer** as shown below:

1) Programmed I/O
2) Interrupt-Driven I/O
3) Direct Memory Access

Ques 6) Describe the programmed I/O using suitable diagram.

Ans: Programmed I/O

In a programmed data transfer scheme, the data transfer takes place between the CPU and an I/O device under the control of a program, which resides in the memory. The CPU executes the program. To execute an I/O-related instruction, the CPU issues an address, specifying the particular I/O module and external device, and an I/O command.

There are four types of I/O commands that an I/O module may receive when a CPU addresses it. They are classified as,

1) Control 2) Test
3) Read 4) Write

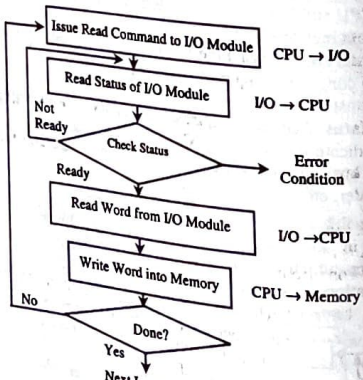


Figure 5.6: Programmed I/O

With programmed Input/Output, there is a closed correspondence between the Input/Output-related instructions that the processor fetches from memory and the Input/Output commands that the processor issues to an Input/output module to execute the inst

That is, the instructions are easily mapped into Input/Output commands, and there is often a simple one-to-one relationship. The form of the instruction depends on the way in which external devices are addressed.

In programmed I/O mode, when an input output instruction is encountered in a program, the CPU checks the input output flag. It keeps on checking the flag till the flag is set to 1. When the flag is set to 1, the CPU transfers data from the I/O device to the memory. It is also known as **polling**.

Ques 7) A processor executes 4,00,00,000 cycles in one second. A printer device is sent 16 bytes in programmed I/O mode. The printer can print 250 characters per second and does not have a print-buffer.

1) How much time will be taken to acknowledge the character status?

2) How many processor cycles are used in transferring just 16-bytes?

Ans:

1) Time taken to acknowledge the character status = 1s + 250 = 4ms ≡ 4 × 40,000 cycles = 1,60,000 processor cycle time.
2) Processor cycles used in transferring 16 bytes = 16 × 1,60,000 = 25,60,000 Cycles.

Ques 8) Consider a processor with programmed I/O that executes instructions at a rate of 0.1MIPS. An I/O subroutine for writing to a floppy disk requires 10,000 instructions and the disk has a total latency of 100ms. A disk request is initiated every second of task time. What is the efficiency of this system?

Ans: The time that the main program is suspended is 100ms + I/O sub-routine time:

$$\text{Suspend time} = 100\times10^{-3} + \frac{10{,}000}{1{,}00{,}000}\text{s} = 100 \times 10^{-3} + 100 \times 10^{-3}\text{s} = 200 \times 10^{-3}\text{s}$$

$$\text{Efficiency} = 100\left(\frac{\text{task time}}{\text{total time}}\right) = 100\left(\frac{1\text{s}}{1\text{s}+200\text{ms}}\right) = 83\%$$

Ques 9) Define Interrupts. What are its various types?
Or
What are interrupts? **(2018 [02])**

Ans: Interrupts

An interrupt is a signal sent by an I/O interface to the CPU when it is ready to send information to the memory or receive information from the memory. An alternative to the programmed controlled procedure is to let the external device inform the CPU when it is ready for the transfer. In the meantime CPU concentrate on some other program and use an interrupt signal to inform the CPU that the input output flag is set to 1. Interrupt causes transfer of control to an interrupt service routine (ISR). ISR is also called a 'handler'. When the ISR is completed, the original program resumes execution.

Types of Interrupts

There are three major types of interrupts that cause a break in the normal execution of a program:

1) **External Interrupts:** External interrupts arise from a circuit monitoring the power supply, or from any other external source.
2) **Internal Interrupts:** Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution. The service program that processes the internal interrupt determines the corrective measure to be taken.
3) **Software Interrupt:** A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction.

Ques 10) Discuss the interrupt driven I/O with diagram.
Or
Explain the mechanism of Interrupt Driven I/O.

Ans: Interrupt Driven I/O

A more common and efficient control method is interrupt-driven I/O. Interrupt-driven I/O can be thought of as the converse of programmed I/O. instead of the CPU continually asking its attached devices whether they have any input, the devices tell the CPU when they have data to send. The CPU proceeds with other tasks until a device requesting service sends an interrupt to the CPU. These interrupts are typically generated for every word of information that is transferred. In most interrupt-driven I/O implementations, this communication takes place through an intermediary interrupt controller.

Mechanism of Interrupt Driven I/O

First from the point of view of the Input/Output module, for input, the Input/Output module receives a READ command from the processor. The Input/Output module then proceeds to read data in from an associated peripheral. Once the data are in the module's data register, the module signals an interrupt to the processor over a control line. The module then waits until the processor requests its data. When the request is made, the module places its data on the data bus and is then ready for another Input/Output operation.

For CPU point of view, the CPU issues a READ command. At the end of each instruction cycle, the CPU checks for interrupts, when the interrupt from the I/O module occurs, the CPU saves the context (i.e., program counter and CPU registers) of the current program and processes the interrupt.

In this case, CP

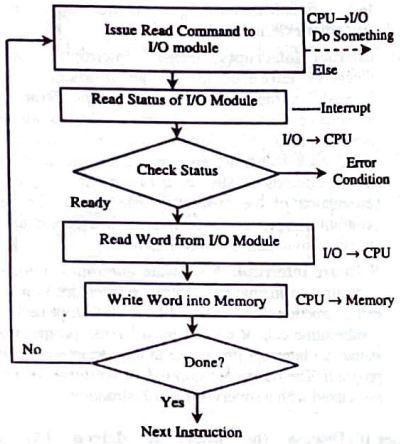


Figure 5.7: Interrupt-Driven I/O

Ques 11) Differentiate between programmed I/O and interrupt driven I/O. (2017 [03])

Ans: Difference between Programmed I/O and Interrupt-Driven I/O

Table below shows the difference between programmed I/O and Interrupt-driven I/O:

Programmed I/O	Interrupt-Driven I/O
Comparatively less efficient.	More efficient.
Processor is dedicated to the task of I/O and can move data at a rather high rate.	Requires active intervention of processor to transfer data between memory and I/O module.
Speeder.	Comparatively slow.
Interrupt hardware support is not required for implementation.	Interrupt hardware support is required for implementation.
Not depends on interrupt status.	Depends on interrupt status.
Number of I/O devices is inversely proportional to system throughput.	Number of I/O devices has no effect on system throughput.
Stack is not used for its initialisation.	Stack is used for its initialisation.

Ques 12) List the sequence of steps following an interrupt request. (2018 [03])

Ans: Interrupt Processing

An interrupt triggers a number of events, both in the processor hardware and in software. The given sequence occurs when an interrupt come across:

Step 1) The device issues an interrupt signal to the processor.

Step 2) The processor finishes execution of the current instruction before responding to the interrupt.

The processor tests for an interrupt determines that there is one and sends an acknowledgement signal to the device that issued the interrupt.

Step 3) The processor now needs to prepare to transfer control to the interrupt routine.

The processor now loads the program counter with the entry location of the interrupt-handling program that will respond to this interrupt.

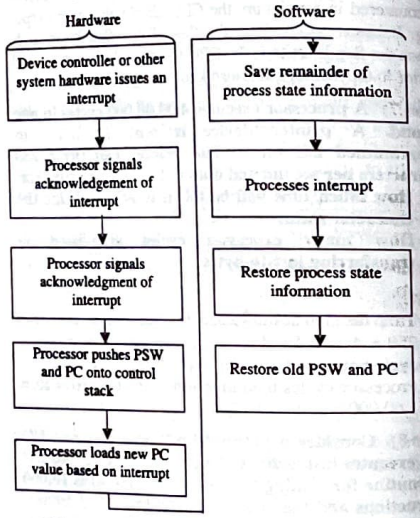


Figure 5.8: Simple Interrupt Processing

Ques 13) What are vectored interrupts? 2019 [03])

Ans: Vector Interrupts

A vectored interrupt is an I/O interrupt that tells the part of the computer that handles I/O interrupts at the hardware level that a request for attention from an I/O device has been received and also identifies the device that sent the request.

A vectored interrupt is an alternative to a polled interrupt, which requires that the interrupt handler poll or send a signal to each device in turn in order to find out which one sent the interrupt request.

Devices that use vectored interrupts are assigned an interrupt vector. This is a number that identifies a particular interrupt handler. This vector may be fixed, configurable (using jumpers or switches), or programmable.

In the case of programmable devices, an interrupt device cookie is used to program the device interrupt vector. When the interrupt handler is registered, the kernel saves the vector in a table.

When the device interrupts, the system enters the interrupt acknowledge cycle, asking the interrupting device to identify itself. The device responds with its interrupt vector. The kernel then uses this vector to find the responsible interrupt handler.

Ques 14) What is Interrupt hardware? Explain with suitable diagram.

Ans: Interrupt Hardware

A single interrupt-request line may be used to serve n devices as depicted in **figure 5.9**. All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all interrupt-request signals INTR1 to $INTR_n$ are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}.

This is the inactive state of the line. When a device requests an interrupt by closing its switch, the voltage on the line drops to 0, causing the interrupt-request signal, INTR, received by the processor to go to 1. Since the closing of one or more switches will cause the line voltage to drop to 0, the value of INTR is the logical OR of the requests from individual devices, that is, INTR = $INTR_1$ ... $INTR_n$.

It is customary to use the complemented form, INTR, to name the interrupt-request signal on the common line, because this signal is active when in the low-voltage state. In the electronic implementation of the circuit in **figure 5.9**, special gates known as opencollector (for bipolar circuits) or open-drain (for MOS circuits) are used to drive the INTR line.

The output of an open-collector or an open-drain gate is equivalent to a switch to ground that is open when the gate's input is in the 0 state and closed when it is in the 1 state. The voltage level, hence the logic state, at the output of the gate is determined by the data applied to all the gates connected to the bus, according to the equation given above. Resistor R is called a pull-up resistor because it pulls the line voltage up to the high-voltage state when the switches are open.

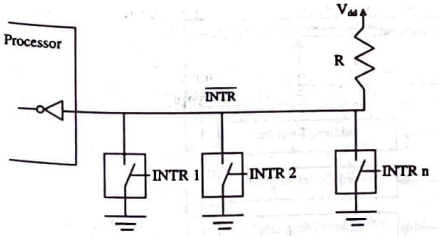


Figure 5.9: An Equivalent Circuit for an Open-drain bus used to implement a common interruptrequest line

Ques 15) What is Direct Memory Access (DMA)? Describe its various modes of operations.

Or

Explain Direct Memory Access. What is burst mode DMA? (2017 [03])

Ans: Direct Memory Access (DMA)

Direct Memory Access (DMA) is a technique that transfers data between a microcomputer's memory and an I/O device without involving the microprocessor.

DMA is wildly used in transferring large blocks of data between a peripheral device such as a hard disk and the microcomputer's memory.

DMA technique uses a DMA controller chip for the data transfer operations. DMA controller chip implements various components, such as a computer containing the length of data to be transferred in hardware in order to speed up data transfer.

Modes of Operation

Different modes of operations are as follows:

1) **Burst Mode:** An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode". It is also used to stop unnecessary data.

2) **Cycle Stealing Mode:** The cycle stealing mode is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes.

   In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR (Bus Request) and BG (Bus Grant) signals, which are the two signals controlling the interface between the CPU and the DMA controller.

3) **Transparent Mode:** The transparent mode takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance.

   The DMA controller only transfers data when the CPU is performing operations that do not use the system buses. It is the primary advantage of the transparent mode that the CPU never stops executing its programs and the DMA transfer is free in terms of time.

Ques 16) Describe the mechanism of data transfer through DMA.

Ans: Mechanism of Data Transfer through DMA

A block diagram of the bus control transfer between CPU and DMA is shown in **figure 5.10**. When an input device needs to transfer data to or from memory, it requests the DMA controller by setting the DMA Request input to 1.

DMA will then set the Bus Request (BR) input to 1 to request control over memory bus. The CPU will relinquish the bus control to DMA and will set to 1 the Bus Grant (BG) output. The DMA controller acknowledges the peripheral device by setting to 1 the DMA acknowledgement output.

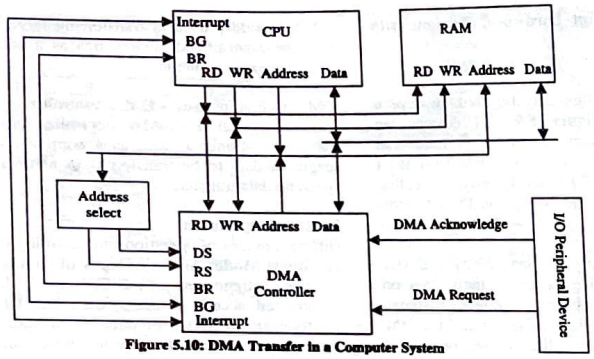


Figure 5.10: DMA Transfer in a Computer System

Ques 17) Explain DMA controller using the block diagram.

Or

List and describe the registers in a DMA interface. **(2018 [03])**

Ans: DMA Controller

DMA Controller is the master of the bus and the peripheral device which can communicate with the memory without CPU's intervention. The input device will then place a word in the data bus, which is then transferred to the memory under the direct control of DMA Controller. When the input device finishes its task, the DMA Controller clears the Bus Request line and the CPU clears the Bus Grant line and control of bus is again with the CPU. A similar method is adopted for transfer of data from memory to output device such as printer.

It contains three registers namely, **Address Register**, **Word Count Register** and **Control Register**. When the bus is under the control of CPU, the CPU can write or read data from these registers through the Read and Write inputs:

1) **Address Register**: Address Register contains the address of memory where the data from the input device needs to be stored or from where the data for the output device needs to be fetched.

   CPU gives this address to the data bus. The address is transferred to the Address Register from the data bus through the internal bus. Address in this register is incremented after transferring each word to or from memory. This is done so that the address register now points to the address of next word in the sequence.

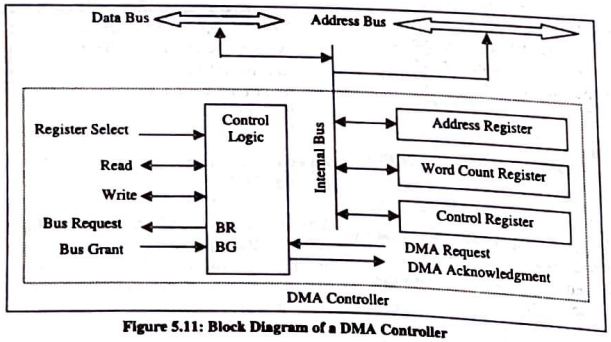


Figure 5.11: Block Diagram of a DMA Controller

2) **Word Count Register:** Word Count Register stores the number of words that need to be read from or written into memory. It is decremented after each word transfer.

   When the value in this register becomes zero, it is assumed that data transfer has completed.

3) **Control Register:** Control Register specifies the mode of data transfer, i.e., whether it is a read operation or it is a write operation.

MEMORY SYSTEM

Ques 18) What is memory? What are its characteristics?

Or

Discuss the basic concept of memory along with its characteristics.

Or

Define the terms: **(2017 [03])**

i) **Latency**
ii) **Bandwidth**
iii) **Memory cycle time**

Ans: Memory

Memory refers to the physical devices used to store instructions to execute programs or data on a temporary or permanent basis for use in a computer or other digital electronic device. Memory in a computer system is required for storage and subsequent retrieval of the instructions and data. A computer system uses variety of devices for storing these instructions and data, which is required for its operations. Basic objective of a memory system is to provide fast-uninterrupted access by the processor to the memory such that the processor can operate at the speed it is expected to work.

Characteristics of Memory

The main characteristics of memory are as below:

1) **Access Time:** For random access memories, this is the time it takes to perform a read or write operation that is the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For **non-random access** memory access time is the time it takes to position the read write mechanism at the desired location. Generally access time has two main components.

2) **Seek time:** This is the time to position the head on specific track. On a fixed head disk it is the time taken by electronic circuit to select the required head while in movable disk it is the time to required move the head to a particular track.

3) **Latency time:** Time required by a sector to reach below read write head. On an average it is half of the time taken for a rotation of the disk.

4) **Cycle Time:** Memory Cycle time is the access time (to perform read/write operation) plus any additional time required before a second access occurs. But for most of the commonly used semiconductor memories cycle time is equal to the access time.

5) **Data Transfer Rate:** Amount of information that can be transferred in or out of the memory in a second is termed as **data transfer rate** or **bandwidth**. It is measured in bits per second. Maximum number of bits that can be transferred in a second depends on how many bits can be transferred in or out of the memory simultaneously and thus the data bus width

   For **random-access** memory, it is equal to the 1/(cycle time).

   For **non- random-access** memory, the following relationship holds:

   $T_N = T_A + N/R$

   Where, T_N = Average time to read or write N bits
   T_A = Average access time
   N = Number of bits
   R = Transfer rate, in bits per second (bps)

Ques 19) What are the different types of memory?

Ans: Types of Memory

There are two types of computer memories) that are commonly used:

1) **Primary Memory:** Primary memory is also known as **internal memory**. All computers must have primary memory. Primary memory is volatile in nature. Its design looks like a honeycomb that has many small compartments. These compartments are individually known as **'cell'** or **'memory cell**. These cells can be recognised by a unique built-in address that cannot be changed. A fixed number of bits can be stored in each cell. These bits are termed as **word length**.

   Primary memory is in the form of memory chip that is housed on rectangular circuit boards. A special slot is provided over the motherboard for plugging up this rectangle circuit board. **Motherboard** is the principle circuit board which consists of several electronic components.

   Types of Primary Memory

   On the basis of some features, primary memory can be categorised into two types:
   i) RAM
   ii) ROM

2) **Secondary/Auxiliary Memory:** Secondary memory is also called as **Auxiliary memory**. Secondary memory is the component of computer system that can be added additionally according to the user's requirement. They are mainly used to store huge amount of data for long time period. The storage capacity of this memory is in terabytes or more. This memory cannot be addressable by the CPU directly. It is a high speed memory. A high speed data channel is used to connect it with main memory.

   Types of Secondary Memory

   i) **Magnetic Memory:** In this memory, magnetic material is used for storing data. Several types of magnetic media are as follows:
      a) Magnetic Tape
      b) Hard Disk (HD)
      c) Floppy Disk (FD)

   ii) **Optical Memory:** These memory devices use optical or laser beams in place of magnetic storage. Tiny holes presented on the reflective platter disk are burnt with the use of pin point laser beam. Another laser beam installed on the disk, shines on

the disk surface at the time of reading data. Reflection shows one binary state. No reflection means a reading of next binary state. As compared to magnetic media, this storage is slower.

Following are the some basic types of optical media:

a) Compact Disk (CD)
b) Digital Video Disk (DVD)

iii) **Flash Memory:** It is a solid chip developed from EEPROM by Toshiba in 1984. It does not require any external power to maintain stored data. They are mainly used in portable devices.

Ques 20) What are the main characteristics of memory?

Or

Give the hierarchical arrangement of memory.

Or

Draw the memory hierarchy according to sizes and speed of memory.

Ans: Memory Hierarchy

The hierarchical arrangement of storage in computer architectures is called **memory hierarchy** as shown in **figure 5.12** designed to take advantage of memory locality in computer programs. Each level of the hierarchy has the properties of bandwidth, size, and latency.

1) **Registers:** Registers are normally at the top of the memory hierarchy, and provide the fastest way to access data. Registers are very high-speed storage areas located inside the CPU. After CPU gets the data and instructions from the cache or RAM, the data and instructions are moved to the registers for processing. Registers are manipulated directly by the control unit of CPU during instruction execution.

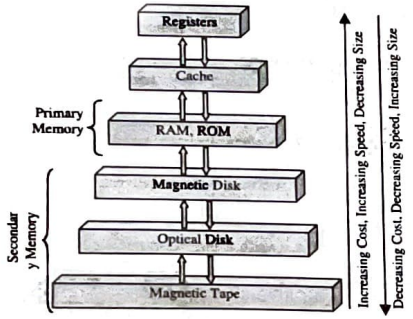


Figure 5.12: Memory Hierarchy

2) **Cache Memory:** Cache Memory provides very fast accessing of data. This is positioned between CPU and main memory to reduce the internal accessing gap or time between CPU (processor) and main memory. Cache memory is intended to speed up the access time and at the same time provide larger capacity at least price. There is relatively large and slower main memory together with the faster cache memory.

3) **Main Memory/Primary Memory:** These memories occupy third position in memory hierarchy if cache memory is not inserted between CPU registers and main memory. These memories are not as fast as registers. The data and instruction requires to be processed earlier reside in main memory. It is divided into two sub categories RAM (Random Access Memory) and ROM (Read Only Memory).

4) **Secondary Memory:** This type of Memory is also known as **external memory** and is slower than main memory. These are used for storing Data/Information permanently. CPU directly does not access these memories instead they are accessed via input-output routines. Contents of secondary memories are first transferred to main memory, and then the CPU can access it. **For example,** magnetic disc, magnetic tape, optical disc, etc.

Ques 21) Explain the different technologies used for constructing memory.

Or

What do you mean by semiconductor memory?

Or

What are the disadvantages of ferrite-core memory?

Ans: Memory Technologies

Memory technology falls into following two categories:

1) **Ferrite-Core Memory:** Ferrite core memories were used as main-memory in computers before the semiconductor memories were invented. The ferrite core memories are based on the magnetic properties of the ferrite material. Core is a ring shaped ferrite material, which is used for storing binary information.

The core can be magnetised in clockwise or anti-clockwise direction, thus representing logical 0 and 1. Electronic current is used to magnetise the core and even after the current is removed the ferrite material stays in the specific magnetic state. This implies that the ferrite core memories were non-volatile. Large RAM can be made from this ferrite core by arranging these in multi-dimensional arrays.

Disadvantages of Ferrite-Core Memories

i) Slow, bulky and consume a lot of power.

ii) They were incompatible to the processor technology, which are semiconductor-based logic circuits.

iii) Difficult to construct because of difficult wiring patterns. These wires are needed for magnetization.

2) **Semiconductor Memory:** The semiconductor memories falls under two main technologies: **Bipolar Semiconductor memories** and **Metal Oxide Semiconductor (MOS)** transistor semiconductor memories. For larger RAM chips normally MOS is used at present. There are two main categories of semiconductor memories known as **static** and **dynamic**. In contrast to ferrite core memory semiconductor RAM are volatile in nature.

The basic element of a semiconductor memory is the **memory cell**. A memory cell may be defined as a device which can store a symbol selected from a set of symbols.

Ques 22) Describe semiconductor RAM memories. **(2018[04])**

Ans: Semiconductor RAM

Semiconductor memory is a type of semiconductor device tasked with storing data. The basic element of a semiconductor memory is the **memory cell**. In earlier computers, the most common form of random-access storage for computer main memory employed an array of doughnut-shaped ferromagnetic loops referred to as cores. Hence, main memory was often referred to as core, a term that persists to this day. The advent of, and advantages of, microelectronics has long since vanquished the magnetic core memory. Today, the use of semiconductor chips for main memory is almost universal.

Numerous developments in semiconductor technology has emerged into large numbers of LSI and MSI memory devices, called memory chips. Besides being faster and compatible with CPU they are economical also. A semiconductor memory is organised as a rectangular array (preferably square array) of storage cells that are integrated on a silicon wafer and are available in DIP packages. Due to this organisation any memory cell can be accessed randomly, thus all the semiconductor memories are called Random Access Memory (RAM).

The basic memory cell may be a flip-flop, or a charge storing element like capacitor that is said to hold logic 1 when charged and logic 0 when discharged. The type of transistors, e.g., bipolar, or MOS, or CMOS, used to form a memory cell dictates the storage capacity and speed of a memory chip. Developing the concept of memory begins with the basic building block for memory called the basic memory cell or simply basic cell. A basic cell plays a major role in designing a memory device (element) that will remember logic 1 or logic 0 indefinitely or until it is directed to change to the other value. Two flavors of the basic cell will be heuristically developed and used later in the design of important memory elements called flip-flops.

Ques 23) What is memory cell in semiconductor memory?

Or

What operations are performed on memory cell?

Ans: Basic Memory Cell

A basic memory cell may be defined as a device which can store a symbol selected from a set of symbols **(figure 5.13).**

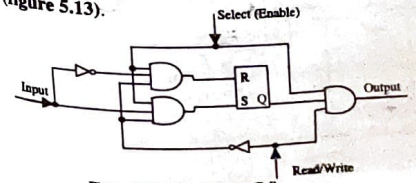


Figure 5.13: Basic Memory Cell

Although a variety of electronic technologies are used, all semiconductor memory cells share certain properties:

1) They exhibit two stable (or semi-stable) states, which can be used to represent binary 1 and 0.
2) They are capable of being written into (atleast once), to set the state.
3) They are capable of being read to sense the state.

Figure 5.14 depicts the operation of a memory cell. Most commonly, the cell has three functional terminals capable of carrying an electrical signal.

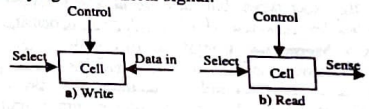


Figure 5.14: Memory Cell Operation

1) The select terminal, as the name suggests, selects a memory cell for a read or write operation.
2) The control terminal indicates read or write.
3) For writing, the other terminal provides an electrical signal that sets the state of the cell to 1 or 0. For reading, that terminal is used for output of the cell's state.

The details of the internal organisation, functioning, and timing of the memory cell depend on the specific integrated circuit technology used.

Table 5.1: Semiconductor Memory Types

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random Access Memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-Only Memory (ROM)	Read-only memory	Not possible	Masks	Non-volatile
Programmable ROM (PROM)				
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level	Electrically	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Ques 24) Discuss the different categories of IC memories.

Ans: Categories of IC Memories

The ability to fabricate large arrays of electronic components using straightforward processing techniques and to make these arrays in small containers at reasonable prices has made semiconductor memories the most popular at this time. Although a number of different schemes and devices are available, at present there are six main categories of IC memories:

1) **Bipolar Memories:** These are essentially flip-flop memories with the flip-flops fabricated using standard pn-junction transistors. These memories are fast but tend to be expensive.
2) **Static MOS Memories:** These are fabricated by using MOS field-effect devices to make flip-flop circuits. These memories are lower in speed than the bipolar memories, but cost less, consume less power, and have high packing densities.

3) **Dynamic MOS Memories:** These are fabricated using MOS devices. But instead of a flip-flop being used for the basic memory cell, a charge is deposited on a capacitor fabricated on the IC chip, and the presence or absence of this charge determines the state of the cell. The MOS devices are used to sense and deposit the charge on the capacitors used. Since the charge used will slowly dissipate in time, it is necessary to periodically refresh this charge, and so the memories are called dynamic memories. These memories tend to be slower than the other types, but they are also less expensive, consume less power, and have a high packing density.
4) **CMOS Memories:** CMOS utilises both p- and n-channel devices on the same substrate. As a result, it involves a more complex manufacturing process. CMOS has improved speed and power output figures over n- and p-channel MOS, but it costs more.
5) **Silicon-on-Sapphire (SOS) Memories:** SOS is similar to CMOS. Devices are formed on an insulating substrate of sapphire. This reduces the device capacitance and improves speed. However, SOS is costly.
6) **Integrated Injection Logic (IIL) Memories:** The IIL circuits eliminate the load resistors and current sources of TTL circuits. This reduces power consumption over bipolar memories, giving greater packing density than bipolar devices. As a result, IIL mixes the speed of bipolar memories with the packing density of MOS. It is medium-cost.

Ques 25) Explain the static and dynamic random access memories.

Or

Why do dynamic RAMs need constant refreshing? How is this done? (2017[03])

Ans: Types of Random Access Memories

1) **Static Random Access Memories:** A static memory cells is essentially a flip-flop. The cell is illustrated in **figure 5.15** within a circle:

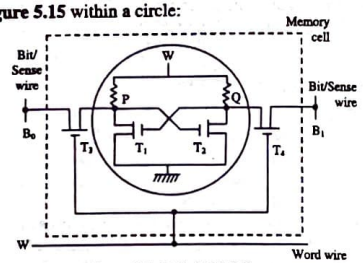


Figure 5.15: Static MOS Cell

In this **figure**, one have simplified the circuit to its essentials in order to explain the working of the cell. The cell works as follows – the flip-flop consisting of transistors T_1 and T_2 can be in one of the two stable states. One stable state is with T_1 conducting and T_2 non-conducting and other is with T_2 conducting and T_1 non-conducting. When T_1 is conducting the voltage at the point P (**figure 5.15**) is nearly 0. This voltage is applied to the gate of transistor T_2 which keeps it switched off. The voltage at Q is thus at V.

In this state the memory cell is said to be storing a 0. The opposite state is with T_2 conducting and T_1 non-conducting. In this case the voltage at P would be V (logic 1) and that at Q will be 0 (logic 0). In this state the memory cell is said to store a 1. To summarise one observe that the two transistor memory cells can be in one of the two stable states. One state with the voltage P = V is (arbitrarily) called the 1 state and the other state with P = 0 is called the 0 state.

2) **Dynamic Random Access Memories (DRAM):** **Figure 5.16** shows a simple dynamic storage cell. It incorporates a transistor T (called a pass transistor) which controls the charging of a capacitor C. The capacitor is of the order of 0.1 picofarad (pico = 10^{-12}) and can hold a very small charge when it is charged. The pass transistor is connected to an address line and a bit/sense line. A cell is selected for writing or reading by applying a voltage V to the address line. To write a 1 in the cell a voltage V is applied to the bit/sense line. This switches on T and C is charged to voltage V. If 0 voltages is applied to the bit/sense line, then if C is charged it will discharge and a 0 is stored.

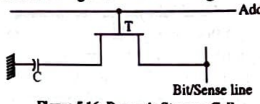


Figure 5.16: Dynamic Storage Cell

Capacitance C is not an ideal capacitance. It has a very large but finite leakage resistance. Thus the charge stored in C when a 1 is written will slowly leak away (in a few milliseconds) and the data will be lost. It is therefore necessary to rewrite the data periodically. This is called **refreshing**. As the charge stored in C is very small, reading data from the cell is a little tricky. To read a cell, the address line is selected and a voltage V is applied to it. This switches on the pass transistor T. If a 1 is stored in the cell, the voltage of the bit/sense line will tend to go upto V and if a 0 is stored in the cell, it will tend to go down to 0. The direction of change of voltage in the bit/sense line is sensed by the sense amplifier. A positive change is taken as a 1 and a negative change as a 0. Observe that the read operation is destructive and a write should follow a read.

Ques 26) Explain the read write operation in RAM.

Ans: Read Write Operation in RAM

The access time and cycle time in RAM are constant and independent of the location accessed. **Figure 5.17** shows the logic diagram of a binary cell.

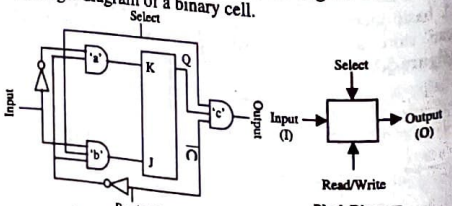


Figure 5.17: Binary Cell

Input is fed in complemented form to AND gate 'a'. The Read/Write signal has a value 1 if it is a read operation. Therefore, during the read operation only the AND gate 'c' becomes active. Since AND gate 'a' & 'b' have 0 inputs and if the select is 1, i.e., this cell is currently being selected, and then the output will become equal to the state of flip-flop. In other words the data value stored in flip-flop has been read. In write operation only 'a' & 'b' gates become active and they set or clear the JK flip flop depending on the input value. Please note that in case input is 0, the flip flop will go to clear state and if input is 1, the flip flop will go to set state. In effect, the input data is reflected in the state of the flip-flop. Thus, it can be stated that the input data has been stored in flip-flop or binary cell.

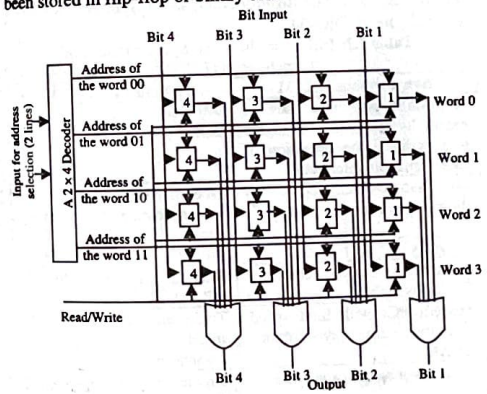


Figure 5.18: Logic diagram of 4 × 4 RAM

Figure 5.18 is the extension of binary cell to an Integrated Circuits (IC) RAM circuit, where a 2 × 4 bit decoder is used. Each decoder output is connected to a 4-bit word and the read/write signal is supplied to each binary cell. The output is derived using OR gates, since all the non-selected cells will produce a zero output the word which is selected will determine the overall output.

Ques 27) Give the structure of a typical static RAM cell and explain its read and write operations. (2019 [05])

Ans: Read Operation

Considering the case of reading Q = 0; before reading a value from the storage nodes, the bitline BL is pre-charged to V_{DD}. The read wordline RL is then asserted to V_{DD}. The storage node QB that stores a 1 is statically connected to the gate of MRA (Read Access Transistor) and will drain the charges on the bitline through MRD to GND as the RL is 1, which means that the bitline has just read a 0. On the contrary, when Q = 1, QB will be 0 and MRA will be in cutoff and the bitline BL will not be able to discharge through MRD to GND, and it would read a 1.

Figure 5.19 shows the 6T SRAM equivalent schematic diagram during read operation. Bitlines are pre-charged to supply voltage before read operation. The read operation is initiated by enabling the Word-Line (WL) and thereby connecting the internal nodes of the SRAM bitcell to bit-lines. The bitline voltage is pulled down by the transistor at the '0' storage node and the difference between two bitline voltages will be detected by sense amplifier. When the Word-Line (WL) is high, one of the bitline voltages is pulled down through transistors M_2 and M_6 or M_1 and M_4. The transistors M_2 and M_6 forms a voltage divider, because of current flowing through M_2, the potential at node QB is no longer at '0'V. Also, it should not go beyond switching threshold of inverter (INV1) to avoid destructive read. The rising of potential depends on sizing of access transistor and pull down transistor, which is defined as a bitcell ratio.

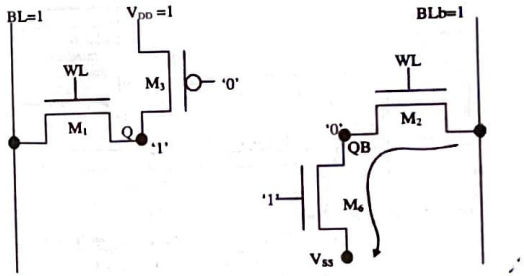


Figure 5.19: Memory Read and Write Equivalent Circuit

Write Operation

The word-line WL is charged to V_{DD} as in 6T Standard SRAM. Since NMOS is a stronger driver than PMOS, no problem is incurred while writing a 0 into the cell. The absence of the pull down NMOS for memory node, Q allows writing a 1 into the cell easily. Writing a 1 is done by pre-charging bit-line BL to V_{DD}.While writing 0, the bit-line BL is discharged, and then word-line WL is charged toV_{DD}. as in 6T Standard SRAM. The write operation begins by forcing a differential voltage (V_{DD}. and 0) at the bitline pairs (BLB and BL).

This differential voltage corresponds to the data to be written at the storage nodes (Q and QB) and it is controlled by the write drivers. The WL is then activated to store the information from the bit-line pairs to corresponding storage nodes. Assume the nodes Q and QB initially store values '1' and '0' respectively. When the WL is asserted the access transistor (M_1) connected to BL (at '0') is turned ON, a current flows from V_{DD}. to BL through M_3 and M_1. This current flow lowers the potential at Q. The potential at the node Q has to go below the trip point of the inverter (INV2) for a successful write operation and this depends on the ratio of pull-up transistor (M_3) and the access transistor (M_1). This ratio is referred to as the pull-up ratio.

Ques 28) Explain what is to be considered in memory system?

Ans: Memory System Considerations

Two considerations are logic-related to access control of DRAMs and refresh control of DRAMs.

1) DRAM access controller
2) Refresh control logic

The number of pins connecting the DRAM is one of the considerations. The address issued by the processor is multiplexed at a DRAM controller to generate the row address bits and column address bits. The DRAM controller issues RAS and CAS signals, chip select, and DRAM clock. **Figure 5.20** shows a major overhead. It has access controller circuit and refresh-control logic for DRAM and sequencing and refreshing control logic. The following example shows that refresh operations within the DRAM are not important, but the accessing controller circuitry is the main consideration.

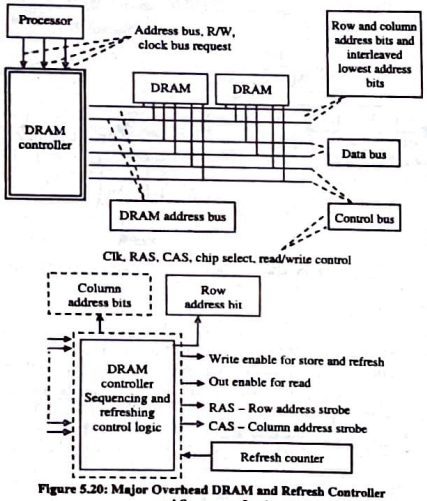


Figure 5.20: Major Overhead DRAM and Refresh Controller and Sequencer Logic

For example, let us assume a DRAM has 4096 row in its array of bit cells and a refresh time of 2ms. Each row must be refreshed once every refresh time, so 4096 row-refresh operations must be done every 2ms, or 1 row refreshes every 480 ns or average. Assume that each row-refresh process at the DRAM takes 4.8ns. So the fraction of the DRAM'S time taken by row refreshes is 4.8ns/480ns = 1.0 per cent. Therefore, refresh overhead is negligible compared to the overhead in accessing the cells of DRAM.

Ques 29) Describe about the synchronous and asynchronous DRAM in detail.

Ans: Asynchronous DRAM

This was the first type of DRAM in use but was gradually replaced by synchronous DRAM. This was called asynchronous because the memory access was not synchronized with the system clock. In asynchronous DRAM, the system clock does not coordinate or synchronizes the memory accessing. When accessing the memory, the value appears on the input, output bus after a certain period. Therefore, it has some latency that minimizes the speed.

Synchronous DRAM

This DRAM replaced the asynchronous RAM and is used in most computer systems today. In synchronous DRAM, the clock is synchronised with the memory interface. All the signals are processed on the rising edge of the clock.

Ques 30) Distinguish between synchronous and asynchronous DRAMs. (2017 [04])

Or

Compare Synchronous and Asynchronous DRAM. (2019 [03])

Ans: Difference between Synchronous and Asynchronous DRAM

Table 5.2 shows the difference between Synchronous and Asynchronous DRAM:

Table 5.2: Difference between Synchronous and Asynchronous DRAM

Synchronous DRAM	Asynchronous DRAM
A DRAM type that uses an externally supplied clock signal to coordinate the operation of its external pin interfaces.	An older type DRAM used in earlier personal computers.
The system clock coordinates the memory access.	Does not use a system clock to coordinate the memory access.
Provides high performance.	Provides low performance.
Manufacture of synchronous DRAM is high.	Manufacture of asynchronous DRAM is relatively rare.
Modern PCs with high speed memory use synchronous DRAM.	Traditional PCs with low speed memory use asynchronous DRAM.
Higher bandwidth capabilities	Simpler interface Legacy design that is common.
Lower bandwidth capabilities	More complicated interface.

Ques 31) Write a note on semiconductor ROM.

Or

What is ROM? Explain ROM Organisation in detail.

Ans: Semiconductor ROM

ROM stands for "Read Only Memory" which is a part of main memory. Read Only Memory (ROM) cannot be written into. That is, write operation cannot be performed in it. It is non-volatile memory, i.e., the information stored in it is not lost even if the power supply goes off. It is used for permanent storage of information and also possesses random access property. A ROM is basically a combinational circuit and can be constructed as shown in **figure 5.21**:

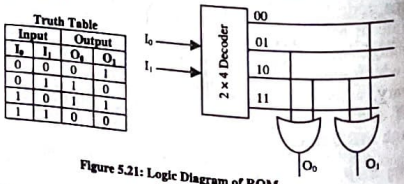


Figure 5.21: Logic Diagram of ROM

Thus, using this hardwired combinational circuit ROM can be created. On applying an Input $I_0 = 0$, $I_1 = 0$ the output is $O_0 = 0$ and $O_1 = 1$; on applying $I_0 = 0$ and $I_1 = 1$, the output is $O_0 = 1$ and $O_1 = 0$ as 01 line of decoder will be selected.

This same logic can be used for constructing larger ROMs. ROMs are much cheaper compared to RAM when produced in large volumes.

The stored information can only be read from ROMs at the time of operation. Information cannot be written into a ROM by the users/programmers. The term ROM by itself generally refers to a memory whose contents are permanently stored by the manufacturer at the time the circuit is fabricated, so it is called **factory-programmed ROM**. Storing the contents of a ROM is called **programming ROM**.

ROM Organisation

A typical block diagram for a ROM is shown in **figure 5.21.** It has three sets of signals:

1)Address Inputs 2) Control Input 3) Data Outputs

ROM shown in the **figure 5.22** can store 32 words since it has 5 address lines and can have $2^5 = 32$ possible words, and each word contains eight bits since there are eight data outputs. Thus, this is a 32 × 8 ROM. Another way to describe the ROM capacity is to say that it stores 32 bytes of data.

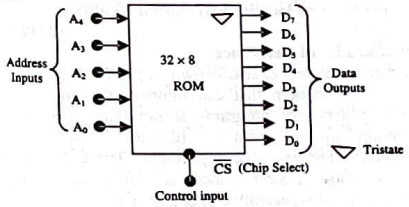


Figure 5.22: ROM Block Diagram

The data outputs of most ROM Integrated Circuits (ICs) are tri-state outputs to permit the connection of many ROM chips to the same data bus for memory expansion the control input $\overline{CS}$ stands for chip select. This is essentially an enable input that enables or disables the ROM outputs. Many ROMs have two or more control inputs that have to be active in order enable the data outputs. In some ROM ICs, one of the control inputs is used to place the ROM in a low-power standby mode when it is not being used. This reduces the current drive from the system power supply.

Ques 32) Describe the different type of ROMs. (2017 [04])

Ans: Types of ROM

1) **PROM:** PROM stands for **Programmable Read Only Memory**. Even though the users do not have the authority to manipulate the firmware yet they can customize the system. For customization, they have to convert the programs into micro programs and store them into the ROM with the help of PROM-programs. Like a ROM, once a PROM is programmed, its contents cannot be altered.

2) **EPROM:** EPROM is the abbreviation of **Erasable Programmable Read Only Memory**. EPROM has the facility to erase and re-program the stored information. All this can be done by exposing the chip to ultraviolet light for certain period. It is non-volatile memory. One cannot modify or re-program it until all current information has been completely erased.

3) **EEPROM:** EEPROM stands for **Electrically Erasable Programmable Read-Only Memory**. It is also non-volatile memory developed in mid-1970s. An electronic circuitry is used in EEPROM for erasing and reprogramming. It is like EPROM except that a high voltage electrical pulse is used in place of ultraviolet light for erasing EEPROM's content.

4) **EAROM:** EAROM is an acronym for "Electrically Alterable Read-Only Memory". It is a type of memory that combines the characteristics of RAM and read-only memory.

It is non-volatile, like read-only memory, but can be written into by the processor. It can only be re-programmed a limited number of times. It is a specialised read-only memory with a special slow-write cycle and a much faster read cycle, used with microprocessors and microcomputers.

Ques 33) What is flash memory? What are the main features of flash memory?

Or

What is flash memory? (2018 [03])

Ans: Flash Memory

It is a non-volatile memory used mainly to store user programs. This type of memory can be programmed electrically while embedded on the board. Some microcontrollers have only 1KB flash memory, while some others can have 32KB or more. In addition to computers, flash memory is also used in mobile phones and digital cameras. It is a solid chip developed from EEPROM by Toshiba in 1984. It does not require any external power to maintain stored data. They are mainly used in portable devices.

Feafures of Flash Memory

Flash memory has many **features**:

1) It is a lot less expensive than EEPROM and does not require batteries for solid-state storage such as static RAM (SRAM).
2) It is non-volatile, has a very fast access time and has a higher resistance to kinetic shock compared to a hard disc drive.
3) Flash memory is extremely durable and can withstand intense pressure or extreme temperatures.
4) It can be used for a wide array of applications such as digital cameras, mobile phones, laptop computers, PDAs (personal digital assistants), digital audio players and solid-state drives (SSDs).

Ques 34) Explain the 'content addressable memory'?

Ans: Content-Addressable Memory (CAM)

It is silicon chip architecture that is purpose-built for extremely fast but very specific type of memory lookups.

Lookups using a CAM is conceptually similar to associative array logic in data structures but the outputs are highly simplified. When key is passed to a CAM sub-system it returns the associated value to that key. As a result a "key -> value" pair is created that can be referenced further as an object. The most important feature is that a lookup of an entry in a CAM can be performed in a single clock cycle in the silicon.

A function calls the CAM by passing a key that consists of data word structure and the CAM lookup returns memory addresses. A CAM cell in the chip actually consists of two SRAM cells. SRAM requires extensive silicon gates to implement that require a lot of power per gate for fast switching. In a chip, power consumption generates heat and leads to limits on thermal dissipation by the limited footprint of a chip.

CACHE MEMORY

Ques 35) What is cache memory? Also draw the structure of cache memory.

Ans: Cache Memory

Cache Memory provides very fast accessing of data. This is positioned between CPU and main memory to reduce the internal accessing gap or access time between CPU and main memory. Cache memory is intended to speed up the access time and at the same time provide larger capacity at least price.

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories. The concept is illustrated in figure 5.23.

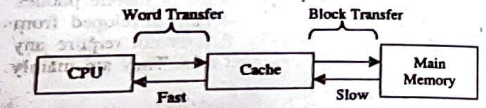


Figure 5.23: Relationship between Cache and Main Memory

Figure 5.24 depicts the structure of a cache/main-memory system. Main memory consists of up to 2^n addressable words, with each word having a unique n-bit address.

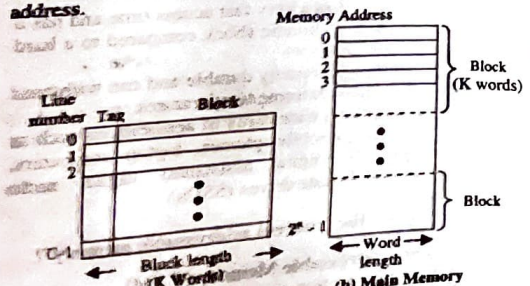


Figure 5.24: Representation of Cache and Main Memory Structure

For mapping purposes, this memory is considered to consist of a number of fixed-length blocks of K words each. That is, there are $M = 2^n/K$ blocks.

Cache consists of C lines of K words each, and the number of lines is considerably less than the number of main memory block (C << M). At any time, some subset of the blocks of memory resides in lines in the cache. If a word in a block of memory is read, that block is transferred to one of the lines of the cache. Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular block.

Thus, each line includes a tag that identifies which particular block is currently being stored. The tag is usually a portion of the main memory address. The term block is used to refer to a set of contiguous addresses of some size. Another term that is often used to refer to a cache block is **cache line**.

Ques 36) Explain the concept of locality of reference?

Or

Define temporal locality and spatial locality.

(2019 [03])

Ans: Locality of Reference

Locality of reference, also known as **principle of locality**, is an access pattern that can improve the efficiency of cache. Analysis of programs shows that most of their execution time is spent on routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other.

The actual detailed pattern of instruction sequencing is not important – the point is that many instructions in localized areas of the program are executed repeatedly during some time period, and the remainder of the program is accessed relatively infrequently. This is referred to as **locality of reference**.

Properties of Locality

1) **Temporal Locality:** Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.

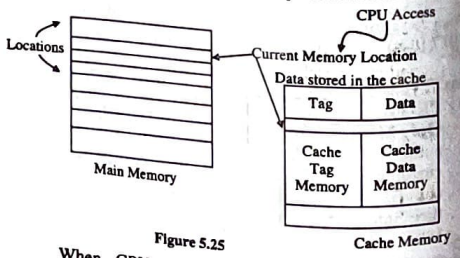


Figure 5.25

When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based

on the fact that same data or instruction may be needed in near future. This is known as temporal locality. If some data is referenced, then there is a high probability that it will be referenced again in the near future.

2) **Spatial Locality:** Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.

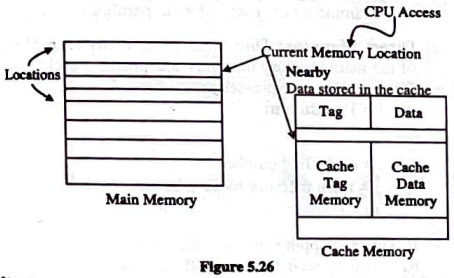


Figure 5.26

3) **Sequential Locality:** Execution of instruction follows a sequential order (or program order) unless branch instructions create out-of-order execution in ordinary programs. Besides, the access of a large data array also follows a sequential order.

Hence, it can be said that a program spends 90% of its time in 10% of memory. Now if one place this 10% of memory in a small fast memory, the memory access time would be greatly reduced and therefore it would reduce the total execution time of the program. Such programs can be kept in a small and fast memory, which is called a cache memory.

Ques 37) How performance of the cache memory can be measured? Explain.

Or

What is cache hit and miss?

Ans: Performance of Cache Memory

The performance of cache memory is measured in terms of what is called the hit ratio.

1) **Cache Hit:** If a data item is requested by CPU is found in the cache, it is called a hit.

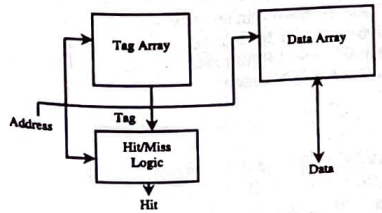


Figure 5.27: Block Diagram of Cache

2) **Cache Miss:** If the requested data item is not found, it is called a miss.

3) **Hit Ratio:** Hit ratio is the ratio of number of hits divided by the total requests.

This is,

$$\text{Hit Ratio} = \frac{\text{Number of hits}}{\text{Number of request by CPU}}$$

Average Memory Access Time = Hit Ratio × Cache Access Time + (1-Hit Ratio) × Main Memory Access Time

For example, suppose a computer has main memory access time of 500 ns or 500×10^{-9} seconds and cache memory access time of 100 ns, and the hit ratio of the cache memory is 0.8. Then an average access time of the memory will be 180 ns:

Average memory access time = hit ratio × cache access time + (1–hit ratio) × main memory access time

$= (0.8) \times 100 + 0.2 \times 500 = 180$ ns

Hit ratio is measured by running many sample programs and then calculating the number of hits and misses in a given time interval. Hit ratios of 0.9 and higher have been reported in many programming situations. Such high hit ratios have proved the theory of locality of reference.

Ques 38) What is mapping function? Explain the different types of the mapping function.

Or

Write notes on set associative cache mapping.

(2017 [05])

Or

With the help of an example, explain the different cache mapping function. **(2018[06])**

Or

Elaborate the various cache mapping techniques with an example for each. **(2019 [09])**

Ans: Mapping Functions

The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Mapping is process to discuss possible methods for specifying where memory blocks are placed in the cache. Mapping function dictates how the cache is organised.

Types of Cache Mapping Function/Cache Mapping Techniques

There are different mapping methods to search for a word in the cache memory. These mappings functions are as explained below:

1) **Associative Mapping:** In associative mapping, the contents of cache memory are not associated with the address. Data stored in the cache memory is not accessed by specifying any address. Instead, data or part of data is searched by matching with the actual contents. The data are then accessed by its contents. In the Associative mapping method, both the words and the address of the word is stored in the cache.

The address bits, sent by CPU to search are matched with the addresses stored in the cache memory. If any address is matched, the corresponding word is fetched from the cache and sent to CPU. If no match is found in cache memory, the word is searched in the main memory. The word along with its address is then copied from main memory into cache. This is done because the word is most likely to be referenced again in future. If the cache is full, then the existing word along with its address must be removed to make room for the new word.

The associative mapping is illustrated in **figure 5.28**. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory. The diagram shows three words presently stored in the cache. Address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four digit octal number.

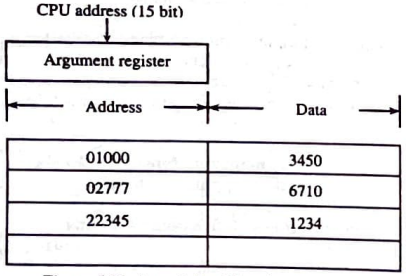


Figure 5.28: Associative Mapping Cache (All numbers in octal)

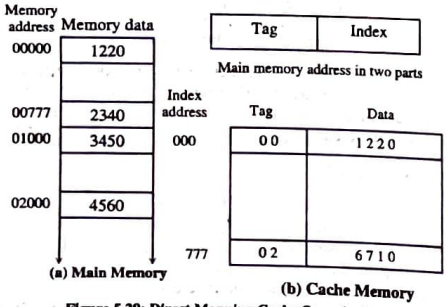


Figure 5.29: Direct Mapping Cache Organisation

A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory. If the cache is full, and address-data pair must be displaced to make room for a pair that is needed and not presently in the cache.

Decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache. A simple procedure is to replace cells of the cache in round-robin order whenever a new word is requested from main memory. This constitutes a **first-in first-out (FIFO)** replacement policy.

Disadvantages of Associative Mapping

i) In this method, it is difficult to determine whether the block is in the cache or not .This process of determining is carried out simultaneously.

ii) This mapping is required complex circuitry to examine all the cache slots in parallel.

2) **Direct Mapping:** Direct mapping, maps each block of the main memory into only one possible cache line. The mapping is expresses as,

i = j modulo m

Where,

i = cache line number

j = main memory block number'

m = number of line in the cache

In Direct mapping method, the cache memory stores the word as well as the tag field. The words will be stored at that location in cache, which is represented by the index fields of their addresses.

The shaded part in **figure 5.29** shows the address of the word in memory and its storage method in cache. When the CPU sends an address, the index part of the address is used to get a location in the cache memory.

If the tag stored at that address matches the tag field of the requested address, the word is fetched. Else, if tag does not match, the word is searched in the main memory.

When a word needs to be moved into cache memory from the main memory, its address in main memory is divided into index and tag fields. At this location in the cache, the word which is already present is removed and this word with its tag is stored. If a computer has main memory of capacity 2^m and cache memory of 2^n, then the address bits will be divided into n bits index field and (m-n) bits as tag field.

For example, suppose a computer has 4K main memory i.e., 4 × 1024 bytes and 1 K cache memory. To address a word in the main memory, a 12 bit (2^{12} = 4096 = 4 × 1024) address is required. Similarly, to address a word in the cache memory, a 10 bit (2^{10} = 1024 = 1 × 1024) address is required.

In such a case, the cache memory needs 10 bits to address a word and main memory requires 12 bits to address a word. In direct mapping method, the 12 bit address sent by CPU is divided into two parts called **tag field** and **index field**. The tag field will contain 2 bits and index field will have 10 bits.

Disadvantage of direct mapping is that two words with the same index in their address but with different tag value cannot reside in cache memory at the same time.

3) **Set Associative Mapping:** A third type of cache organisations, called **set-associative mapping**, is an improvement over the direct mapping organisation in that each word of cache can store two nor more words of memory under the same index address.

Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.

Each index address refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is 2(6 + 12) = 36 bits.

For example, an index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512 × 36. It can accommodate 1024 words of main memory since each word of cache contains two data words. In general, a set-associative cache of set size k will accommodate *k* words of main memory is each word of cache.

When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value. The most common replacement algorithms used are: **Random Replacement, First-In-First-Out (FIFO),** and **Least Recently Used (LRU).**

Ques 39) In a direct-mapped cache with a capacity of 16 KB and a line length of 32 bytes, how many bits are used to determine the byte that a memory operation references within a cache line, and how many bits are used to select the line in the cache that may contain the data?

Ans: Log_2 of 32 is 5, so 5 bits are required to determine which byte within a cache line is being referenced. With 32-byte lines, there are 512 lines in the 16-KB cache, so 9 bits are required to select the line that may contain the address ($\log_2$ of 512 = 9).

Ques 40) How many sets are there in a two-way set-associative cache with 32-KB capacity and 64-byte lines, and how many bits of the address are used to select a set in this cache? What about an eight-way set-associative cache with the same capacity and line length?

Ans: A 32-KB cache with 64-byte lines contains 512 lines. In a two-way set-associative cache, each set contains 2 lines, so there are 256 sets in the cache. $\log_2(256) = 8$, so 8 bits of an address are used to select a set that the address maps to. The eight-way set-associative cache has 64 lines and uses 6 bits of the address to select a set.

Ques 41) A computer system has a main memory consisting of 1M 16-bit words. It also has a 4K-word cache organized in the block-set-associative manner, with 4 blocks per set and 64 words per block. Calculate the number of bits in each of the Tag, Set, and Word fields. (2018 [03])

Ans: Word offset = 6-bit
Byte offset = 1-bit
virtual address = 21-bits
Number of cache block = 4k words/64words = 64
Number of set = 64 block/4 block = 16

So, set offset = 4-bits
Tag bit = 21 – 6 – 1 – 4 = 10-bits
∵ 1word = 16-bit = 2-byte
mm size = $2^{20} \times 2 = 2^{21}$
Number of bit for physical add = 21

$$\text{No of block} = \frac{2^2 \times 2^{10}}{2^6} \text{ words}$$

Number of block = 2^6

$$\text{Number of set} = \frac{2^6}{2^2} = 2^4$$

Hence the set bit = 4
Let assume byte addressable. So,
Byte addressable = 64 word/block
$2^6 \times$ (1word = 2-byte)
2^7 = 7-bit
Hence the bits in each of fields

Tag	Set	Word
10	4	6

Ques 42) The cache block size in many computer is in the range of 32 to 128 bytes. What would be the main advantages and disadvantages of making the size of cache blocks larger? [2018(03)]

Ans: Advantages of making the size of cache blocks larger:

1) Take advantage of spatial locality.
2) Decreases compulsory misses.

However, larger blocks have disadvantages:

1) May increase the miss penalty (need to get more data).
2) May increase hit time (need to read more data from cache and larger mux).
3) May increase miss rate, since conflict misses.

KTU

B.TECH - FOURTH SEMESTER EXAMINATION, MAY-2017

COMPUTER ORGANISATION AND ARCHITECTURE

Time: 3 Hours **Max. Marks: 100**

PART – A

Note: Answer all questions. Each carries 3 marks.

Ques 1) Write notes on condition codes. **(03)**

Ans: Condition Code
Refer Module-2, Page No. B-35, Question No. 23

Ques 2) Explain indirect addressing with an example. **(03)**

Ans: Indirect Addressing
Refer Module-1, Page No. B-11, Question No. 17

Ques 3) Draw the flow chart for Booth's Multiplication algorithm. **(03)**

Ans: Flowchart for Booth's Multiplication Algorithm
Refer Module-3, Page No. B-46, Question No. 20

Ques 4) Explain the process of storing a word in memory using a single bus organisation, Specify which all control signals will be activated. **(03)**

Ans: Process of Storing a Word in Memory
Refer Module-1, Page No. B-15, Question No. 23

PART – B

Note: Answer any two question. Each carries 9 marks.

Ques 5 a) Briefly explain the memory access instruction and addressing modes of ARM processor. **(04)**

Ans: Out of syllabus

Ques 5 b) Write notes on multiple bus organisation. **(05)**

Ans: Multiple Bus Organisations
Refer Module-1, Page No. B-17, Question No. 27

Ques 6) Explain the terms processor stack, stack frame and frame pointer with relation to subroutine processing. Use a relevant example. **(09)**

Ans: Out of syllabus

Ques 7) Draw and explain the flow charts for floating point multiplication and division. **(09)**

Ans: Floating Point Multiplication and Division
Refer Module-3, Page No. B-52, Question No. 30

PART – C

Note: Answer all questions. Each carries 3 marks.

Ques 8) Differentiate between programmed I/O and interrupt driven I/O. **(03)**

Ans: Difference between Programmed I/O and Interrupt Driven I/O
Refer Module-5, Page No. B-76, Question No. 11

Ques 9) Define the terms: **(03)**
i) Latency
ii) Bandwidth
iii) Memory cycle time

Ans: Latency and Bandwidth
Refer Module-5, Page No. B-79, Question No. 18

Memory Cycle Time
Refer Module-5, Page No. B-79, Question No. 18

Ques 10) Why do dynamic RAMs need constant refreshing? How is this done? **(03)**

Ans: Need of Refreshing to Dynamic Memory
Refer Module-5, Page No. B-82, Question No. 25

Ques 11) Explain Direct Memory Access. What is burst mode DMA? **(03)**

Ans: DMA –
Refer Module-5 Page No. B-77, Question No. 15

Burst Mode DMA
Refer Module-5, Page No. B-77, Question No. 15

PART – D

Note: Answer any Two questions. Each carries 9 marks.

Ques 12 a) Distinguish between centralised and distributed bus arbitration? **(04)**

Ans: Out of syllabus

Ques 12 b) Write notes on set associative cache mapping. **(05)**

Ans: Set Associative Cache Mapping
Refer Module-5, Page No. B-87, Question No. 38

Ques 13 a) Distinguish between synchronous and asynchronous DRAMs. **(04)**

Ans: Difference between Synchronous and Asynchronous DRAM
Refer Module-5, Page No. B-84, Question No. 30

Ques 13 b) Explain the important Data transfer signals on the PCI bus. **(05)**

Ans: Out of syllabus

Ques 14 a) Describe the different type of ROMs. **(04)**

Ans: Types of ROM
Refer Module-5, Page No. B-85, Question No. 32

Ques 14 b) Explain the procedure and the packets used for an output transfer in USB interface. **(05)**

Ans: Out of syllabus

PART – E

Note: Answer any Four questions. Each carries 10 marks.

Ques 15) Describe processor organisation with diagram using: **(10)**
i) Scratchpad memory.
ii) Two port memory.

Ans: Processor Organisation Using Scratchpad Memory and Two Port Memory
Refer Module-2, Page No. B-28, Question No. 14

Ques 16) Design a 4bit Arithmetic Module which performs the following operations on two inputs. A and B, controlled by selection variables s_1, and s_0, and input carry C_{in}: **(10)**

S_1	S_0	$C_{in} = 0$	$C_{in} = 1$
0	0	F = A	F = A + 1
0	1	F = A + B	F = A + B + 1
1	0	F = A + B'	F = A + B' + 1
1	1	F = A – 1	F = A

Ans: Refer Module-2, Page No. B-33, Question No. 21

Ques 17 a) Write notes on status register. **(05)**

Ans: Status Register
Refer Module-2, Page No. B-34, Question No. 22

Ques 17 b) Distinguish between horizontal and vertical microinstructions. **(05)**

Ans: Difference between Horizontal and Vertical Microinstruction
Refer Module-4, Page No. B-63, Question No. 7

Ques 18) What is the significance of a micro program sequencer? Explain its working with the help of a diagram. **(10)**

Ans: Significance of Micro Program Sequencer
Refer Module-4, Page No. B-66, Question No. 10

Ques 19) Explain micro programmed CPU organisation with the help of a diagram. **(10)**

Ans: Micro Programmed CPU Organisation
Refer Module-4, Page No. B-68, Question No. 12

Ques 20) With the help of a block diagram, describe a complete processor Module with all components and appropriate control variables. Show with an example, how a control word for the processor can be defined. **(10)**

Ans: Complete Processor Module
Refer Module-4, Page No. B-69, Question No. 14

KTU

B.TECH - FOURTH SEMESTER EXAMINATION, MAY-2018

COMPUTER ORGANISATION AND ARCHITECTURE

Time: 3 Hours **Max. Marks: 100**

PART-A

Answer all questions, each carries 3 Marks.

Ques 1) With a neat diagram, explain the internal architecture of the CPU. (03)

Ans: Refer Module-1, Page No. B-3, Question No. 2

Ques 2) What are condition codes? List the different condition codes. (03)

Ans: Refer Module-2, Page No. B-35, Question No. 23

Ques 3) Prove that the worst case delay through an n × n array multiplier is 6(n –1) –1 gate delays. (03)

Ans: Refer Module-3, Page No. B-45, Question No. 19

Ques 4) Enumerate the sequence of actions involved in executing an unconditional branch instruction. (03)

Ans: Refer Module-1, Page No. B-11, Question No. 15

PART-B

Answer any Two questions, each carries 9 marks.

Ques 5 a) With the help of examples, explain the different addressing modes. (05)

Ans: Refer Module-1, Page No. B-11, Question No. 17

Ques 5 b) Register R6 is used in a program to point to the top of a stack containing 32-bit numbers. Write a sequence of instructions using the Index, Autoincrement, and Autodecrement addressing modes to perform each of the following tasks: (04)
a) Pop the top two items off the stack, add them, then push the result onto the stack.

Ans: Refer Module-1, Page No. B-12 Question No.18

Ques 5 b) b) Copy the fifth item from the top into register R3. For each case, assume that the stack contains ten or more elements.

Ans: Refer Module-1, Page No. B-12, Question No. 18

Ques 6 a) With the help of a diagram, describe the datapath inside the processor. (05)

Ans: Refer Module-1, Page No. B-12, Question No. 20

PART-D

Answer any Two questions, each carries 9 marks.

Ques 12 a) With diagram, explain the PCI bus. (05)

Ans: Out of syllabus

Ques 6 b) Discuss the different ways in which the return address can be saved during a subroutine call. Which of these methods support subroutine nesting? Justify your answer. (04)

Ans: Out of syllabus

Ques 7 a) Write down the sequence of actions needed to fetch and execute the instruction: Store R6, X (R8). (03)

Ans: Refer Module-1, Page No. B-11, Question No. 16

Ques 7 b) Multiply each of the following pairs of signed 2's-complement numbers using the Booth algorithm. In each case, assume that A is the multiplicand and B is the multiplier. (06)

i) A = 001011 and B = 011011

Ans: Refer Module-3, Page No. B-47, Question No.25

Ques 7 b) ii) A = 000111 and B = 000111

Ans: Refer Module-3, Page No. B-47, Question No.25

PART-C

Answer all questions, each carries 3 marks.

Ques 8) Explain the functions of interface circuits. (03)

Ans: Out of syllabus

Ques 9) List and describe the registers in a DMA interface. (03)

Ans: Refer Module-5, Page No. B-78, Question No. 17

Ques 10) The cache block size in many computer is in the range of 32 to 128 bytes. What would be the main advantages and disadvantages of making the size of cache blocks larger? (03)

Ans: Refer Module-5, Page No. B-89, Question No. 42

Ques 11) what is flash memory? (03)

Ans: Refer Module-5, Page No. B-85, Question No. 33

Ques 12 b) Write a note on the packet type formats of USB. (04)

Ans: Out of syllabus

Ques 13 a) What are interrupts? List the sequence of steps following an interrupt request. (05)

Ans: Interrupts
Refer Module-5, Page No. B-75, Question No. 9

Sequence of Steps Following an Interrupt Request
Refer Module-5, Page No. B-76, Question No. 12

Ques 13 b) Describe semiconductor RAM memories. (04)

Ans: Refer Module-5, Page No. B-81, Question No. 22

Ques 14 a) With the help of an example, explain the different cache mapping function. (06)

Ans: Refer Module-5, Page No B-87, Question No. 38

Ques 14 b) A computer system has a main memory consisting of 1M 16-bit words. It also has a 4K-word cache organized in the block-set-associative manner, with 4 blocks per set and 64 words per block. Calculate the number of bits in each of the Tag, Set, and Word fields. (03)

Ans: Refer Module-5, Page No. B-89, Question No. 41

PART-E

Answer any Four questions, each carries 10 marks.

Ques 15 a) Discuss shift and conditional control micro operations (07)

Ans: Shift Micro Operation
Refer Module-2, Page No. B-24, Question No. 9

Conditional Control Micro Operation
Refer Module- 2, Page No. B-24, Question No. 10

Ques 15 b) An 8-bit register A has one input x. The register operation is represented symbolically as P: $A_7 \leftarrow x$, $A_i \leftarrow A_{i+1}$ i = 0, 1, 2, 3... 6. What is the function of the register? (03)

Ans: Out of syllabus

Ques 16) Compare vertical and horizontal microinstructions formats, giving examples. (10)

Ans: Refer Module-4, Page No. B-63, Question No. 7

Ques 17) With a diagram, explain how control signals are generated using hardwired control. (10)

Ans: Refer Module-4, Page No. B-57, Question No. 2

Ques 18) Describe the purpose of microprogram sequencing. How is it carries out? (10)

Ans: Refer Module-4, Page No. B-66, Question No. 10

Ques 19) Draw the block diagram for the hardware that implements the following statement x + yz: AR ← AR + BR where AR and BR are two n-bit registers and x, y, and z are control variables. Include the logic gates for the control function. (The symbol + designates an OR operation in a control or Boolean function and an arithmetic plus in a micro operation. (10)

Ans: Refer Module 4, Page No. B-71, Question No. 15

Ques 20) Explain the design of status register. (10)

Ans: Refer Module-2, Page No. B-34, Question No. 22

KTU

B.TECH - FOURTH SEMESTER EXAMINATION, MAY-2019

COMPUTER ORGANISATION AND ARCHITECTURE

Time: 3 Hours **Max. Marks: 100**

PART-A

Note: Answer all questions, each carries 3 marks.

Ques 1) Explain one, two and three address instruction with an example for each.

Ans: Address Instruction
Refer Module 1, Page No., B-8, Question No. 11

Ques 2) List the steps involved in invoking a subroutine through the use of a link register. (03)

Ans: Out of Syllabus

Ques 3) Draw a 3 × 2 array multiplier.

Ans: 3×2 Array Multiplier
Refer Module 3, Page No. B-45, Question No.18

Ques 4) Non-restoring division is faster than restoring division. Justify the statement. (03)

Ans: Non-Restoring Division
Refer Module 3, Page No. B-42, Question No. 15

PART-B

Note: Answer any Two questions, each carries 9 marks.

Ques 5) List various addressing modes explain any four with an example for each. (09)

Ans: Addressing Modes
Refer Module 1, Page No. B-11, Question No. 17

Ques 6 a) Draw the diagram of a multi-bus organisation with 3 buses. Write the control sequence for the instruction Add R4, R5, R6 for the above mentioned multi-bus organisation. (05)

Ans: Refer Module 1, Page No. B-4, Question No. 4

Ques 6 b) Give the sequence of control steps required to perform the operation Add [R3], R1 in a single-bus organisation.

Ans: Refer Module 1, Page No. B-6, Question No. 6

Ques 7 a) Divide $(1000)_2$ by $(11)_2$ using restoring division method. (04)

Ans: Refer Module 3, Page No. B-42, Question No. 15

Ques 7 b) b) Illustrate the basic operational concept involved in transferring data between main memory and processor with neat diagram. (05)

Ans: Refer Module 1, Page No. B-12, Question No. 20

PART-C

Note: Answer all question, each carries 3 marks.

Ques 8) What are vectored interrupts? (03)

Ans: Vector Interrupts
Refer Module 5, Page No.B-76, Question No. 12

Ques 9) Give the functions of initiator and target controllers in SCSI bus. (03)

Ans: Out of Syllabus

Ques 10) Compare Synchronous and Asynchronous DRAM. (03)

Ans: Refer Module 5, Page No. B-84, Question No. 30

Ques 11) Define temporal locality and spatial locality.

Ans: Locality of Reference
Refer Module 5, Page No. B-86, Question No. 36

PART-D

Note: Answer any Two questions, each carries 9 marks.

Ques 12 a) Differentiate Centralised and Distributed Bus Arbitration mechanism used in DMA. (04)

Ans: Out of Syllabus

Ques 12 b) Give the structure of a typical static RAM cell and explain its read and write operations. (05)

Ans: Static RAM Cell
Refer Module 5, Page No. B-83, Question No. 27

Ques 13) Differentiate Serial Port and Parallel Port. Draw the diagram of a bidirectional 8-bit parallel interface and explain its working. (04)

Ans: Out of Syllabus

Ques 14) Elaborate the various cache mapping techniques with an example for each. (09)

Ans: Mapping Functions
Refer Module 5, Page No.B-87, Question No. 38

PART-E

Note: Answer any four questions, each carries 10 marks.

Ques 15 a) Write the Register Transfer Logic format for a conditional control statement. Give an example and explain the same.

Ans: Refer Module 2, Page No. B-24, Question No. 10

Ques 15 b) Mention the advantages of using a scratch pad memory. Draw the diagram of a processor that employs a scratch pad memory and explain the same.

Ans: Refer Module 2, Page No. B-28, Question No. 14

Ques 16 a) Design an adder/subtractor circuit with one selection variable s and two inputs A and B. When s = 0 the circuit performs A + B. When s = 1 the circuit performs A – B by taking 2's complement of B. (05)

Ans: Refer Module 2, Page No. B-22, Question No. 07

Ques 16 b) Design a 4-bit combinational logic shifter with 2 control signals H_1 and H_0 that performs the following operations (bit values given in parenthesis are the values of control variables H_1 and H_0 respectively) – No shift (00), Shift-right (01), Shift-left (10), Transfer 0's to S (11). (09)

Ans: 4-Bit Combinational Logic Shifter
Refer Module 2, Page No. B-38, Question No. 28

Ques 17 a) Draw and explain the block diagram for a 4-bit complete accumulator. (06)

Ans: 4-Bit Complete Accumulator
Refer Module 2, Page No. B-26, Question No. 12

Ques 17 b) Discuss about condition code bits in a 4-bit status register. (04)

Ans: 4-Bit Status Register
Refer Module 2, Page No. B-35, Question No. 24 **(04)**

Ques 18) Design a hard-wired control unit based on the one flip-flop per state method to add/subtract 2 signed numbers represented in the sign-and-magnitude form.

Ans: Out of Syllabus **(06)**

Ques 19) Explain the organisation of a micro-programmed computer with a block diagram. (10)

Ans: Micro-Programmed CPU Organisation
Refer Module 4, Page No. B-68, Question No. 12

Ques 20) Draw a neat block diagram of a micro-program sequencer and explain its working. (10)

Ans: Working of Micro Program Sequencer
Refer Module 4, Page No. B-66, Question No. 10

MODEL PAPER

B.TECH - FOURTH SEMESTER EXAMINATION

COMPUTER ORGANISATION AND ARCHITECTURE

Time: 3 Hours **Max. Marks: 100**

Part A

Note: Answer All questions. Each question carries 3 marks.

Ques 1) Give the significance of instruction cycle.

Ques 2) Distinguish between big endian and little endian notations. Also give the significance of these notations.

Ques 3) Compare I/O mapped I/O and memory mapped I/O.

Ques 4) Give the importance of interrupts in I/O interconnection.

Ques 5) Justify the significance of status register.

Ques 6) How does the arithmetic circuitry perform logical operations in an ALU.

Ques 7) Illustrate divide overflow with an example.

Ques 8) Write notes on arithmetic pipeline.

Ques 9) Briefly explain the role of micro program sequence.

Ques 10) Differentiate between horizontal and vertical micro instructions.

Part-B

Note: Answer Any One Question from each module. Each question carries 14 Marks.

Ques 11 a) What is the significance of addressing modes in computer architecture. (04)

Ques 11 b) Write the control sequence for the instruction DIV R1, [R2] in a three bus structure. (10)

Or

Ques 12) Explain the concept of a single bus organisation with the help of a diagram. Write the control sequence for the instruction ADD [R1], [R2]. (14)

Ques 13) Explain various register transfer logics. (14)

Or

Ques 14 a) Design a 4 bit combinational logic shifted with 2 control signals H1 and H2 that perform the following operations (a bit values given in parenthesis are the values of control variable H1 and H2 respectively.): Transfer of 0's to S (00), shift right (01), shift left (10), no shift (11). (05)

Ques 14 b) Design an ALU unit which will perform arithmetic and logic operation with a given binary adder. (09)

Ques 15 a) Give the logic used behind Booth's multiplication algorithm. (04)

Ques 15 b) Identify the appropriate algorithm available inside the system to perform the multiplication between –14 and –9. Also trace the algorithm for the above input. (10)

Or

Ques 16 a) List and explain the different pipeline hazards and their possible solutions. (10)

Ques 16 b) Design a combinational circuit for 3×2 multiplication. (04)

Ques 17) Design a hardwared control unit used to perform addition/subtraction of 2 numbers represented in sign magnitude form. (14)

Or

Ques 18) Give the structure of the micro program sequencer and its role in sequencing the micro instructions. (14)

Ques 19 a) Explain the different ways in which interrupt priority schemes can be implemented. (10)

Ques 19 b) Give the structure of SRAM cell. (04)

Or

Ques 20 a) Explain the various mapping functions available in cache memory. (09)

Ques 20 b) Briefly explain content addressable memory. (05)

DATABASE MANAGEMENT SYSTEMS

TP SOLVED SERIES

For

B.Tech Fourth Semester Students

(Computer Science and Engineering)

of

'APJ Abdul Kalam Technological University (KTU), Kerala'

Edition 2022

THAKUR PUBLICATION PVT. LTD., ERNAKULAM

House No. 46/1309, Kattikaran House, Feroz Gandhi Lane, Vaduthala (Post),
Ernakulam-682023. Ph. 9207296272, 9207296273, 09235318597
For Supply: 9207296271, 09389557482