Microprocessor and Computer Architecture

EG2204CT

Year: II
Part: II

Total: 7 hours /week
Lecture: 3 hours/week
Tutorial: 1 hour/week
Practical: … hours/week
Lab: 3 hours/week

Course Contents:

Theory

Unit 1. Introduction of Microprocessor **[8 Hrs.]**

1.1. Evolution of microprocessor and it's types
1.2. Microprocessor Bus organization: Data Bus, Address Bus and Control Bus
1.3. Operations of microprocessor: internal data manipulation, microprocessor initiated and peripheral or external initiated
1.4. Pin diagram and internal Architecture of 8085
1.5. Internal registers organization of 8085
1.6. Limitations of 8085

Unit 2. Instruction Cycle and Timing Diagram **[3 Hrs.]**

2.1. 8085 machine cycles
2.2. Bus timings to fetch, decode, execute instruction from memory
2.3. Memory read and write
2.4. Input/output read and write cycle with timing diagram

Unit 3. 8085 Instruction set **[12 Hrs.]**

3.1. Machine language instruction format:
  3.1.1. Single byte
  3.1.2. Two bytes
  3.1.3. Three-byte instructions
3.2. Various addressing modes
3.3. Data transfer operation and instruction
3.4. Arithmetic operation and instruction
3.5. Logical operation and instruction
3.6. Branch operation and instruction
3.7. Stack operation and instruction
3.8. Input/output and machine control operation and instruction
3.9. Simple programs with 8085 instructions

Unit 4. Basic Computer Architecture **[4 Hrs.]**

4.1. Introduction
  4.1.1. History of Computer Architecture
  4.1.2. Overview of Computer Organization
  4.1.3. Memory Hierarchy and cache
4.2. Instruction Codes
4.3. Stored Program Organization
4.4. Indirect address, computer registers
4.5. Common Bus system
4.6. Instruction sets
4.7. Instruction types

Unit 5. Design of Microprogrammed Control Unit **[10 Hrs.]**

5.1. Control Word, Microprogram, Control Memory
5.2. Control Address Register, Sequencer
5.3. Address Sequencing
5.4. Conditional Branch
5.5. Mapping of Instructions
5.6. Subroutines, Microinstruction Format, Symbolic Microinstructions
5.7. Central Processing Unit
  5.7.1. Introduction
  5.7.2. General Register Organization
  5.7.3. Stack Organization
5.8. Instruction Formats
5.9. Addressing Modes
5.10. RISC vs CISC
5.11. Pipeline and Vector Processing
  5.11.1. Arithmetic and Instruction pipeline
  5.11.2. Vector operations
  5.11.3. Matrix Multiplication, memory interleaving

Unit 6. Computer Arithmetic **[3 Hrs.]**

6.1. Data Representation
  6.1.1. Fixed point Representation
  6.1.2. Floating point Representation
6.2. Addition and Subtraction with Signed Magnitude Data
6.3. Addition and Subtraction with Signed 2's Complement Data
6.4. Multiplication of Signed Magnitude Data
6.5. Booth Multiplication

Unit 7. Input Output Organization **[5 Hrs.]**

7.1. Input-Output Interface
  7.1.1. I/O Bus and Interface Modules
  7.1.2. I/O vs. Memory Bus
  7.1.3. Isolated vs. Memory-Mapped I/O
7.2. Asynchronous Data Transfer: Strobe, Handshaking
7.3. Modes of Transfer:
  7.3.1. Programmed I/O
  7.3.2. Interrupt-Initiated I/O
  7.3.3. Direct memory Access
7.4. Direct Memory Access, Input-Output Processor, DMA vs. IOP

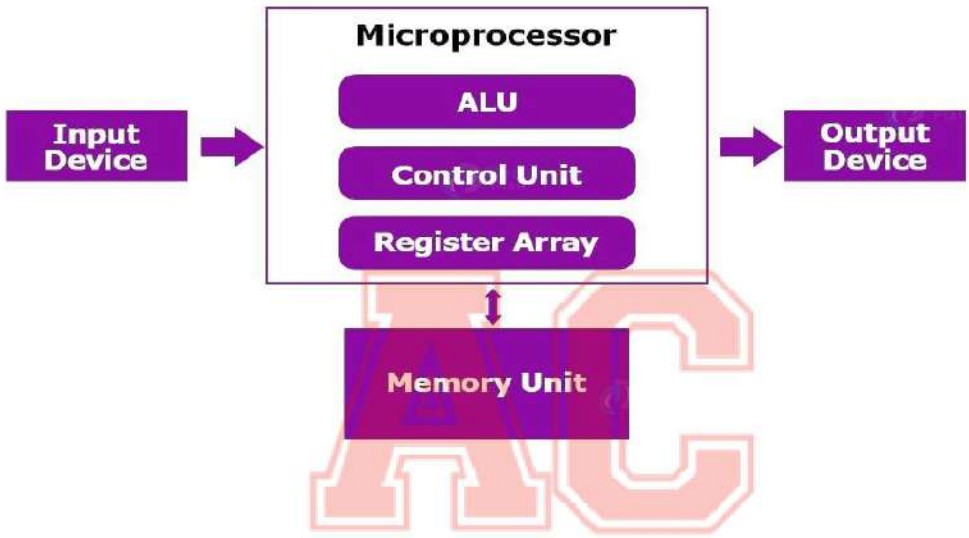
Final written exam evaluation scheme			
Unit	**Title**	**Hours**	**Marks Distribution***
1	Introduction of Microprocessor	8	14
2	Instruction Cycle and Timing Diagram	3	5
3	8085 Instruction set	12	22
4	Basic Computer Architecture	4	7
5	Design of Microprogrammed Control Unit	10	18
6	Computer Arithmetic	3	5
7	Input Output Organization	5	9
	Total	**45**	**80**

Unit 1. Introduction of Microprocessor [Marks – 14]

1.1.Evolution of microprocessor and it's types

What is microprocessor: A microprocessor is an integrated circuit (IC) or chip that serves as the central processing unit (CPU) of a computer or electronic device. It is responsible for executing instructions and performing calculations, making it the "brain" of the system. Microprocessors are a critical component in a wide range of electronic devices, from personal computers and smartphones to appliances, industrial machinery, and embedded systems.

Block Diagram of Microprocessor



Arithmetic and Logical Unit, Control Unit, and Register array make up the microprocessor. The ALU deals with input devices or memory for receiving data. The control unit takes care of instructions and structure. Register array identifies and saves the registers like B, C, and accumulator.

Working of a Microprocessor

There are three steps that a microprocessor follows –

1. Fetch – The instructions are in storage from where the processor fetches them.

2. Decode – It then decodes the instruction to assign the task further. During this, the arithmetic and logic unit also performs to register the data temporarily.

3. Execute – The assigned tasks undergo execution and reach the output port in binary form.

Evolution of Microprocessors

1. First Generation – 4bit Microprocessors

The Intel corporation came out with the first generation of microprocessors in 1971. They were 4-bit processors namely Intel 4004. The speed of the processor was 740 kHz taking 60k instructions per second. It had 2300 transistors and 16 pins inside.

Built on a single chip, it was useful for simple arithmetic and logical operations. A control unit was there to understand the instructions from memory and execute the tasks.

2. Second Generation – 8bit Microprocessor

The second generation began in 1973 by Intel as the first 8 – bit microprocessor. It was useful for arithmetic and logic operations on 8-bit words. The first processor was 8008 with a clock speed of 500kHz and 50k instructions per second.
Followed by an 8080 microprocessor in 1974 with a speed of 2 MHz and 60k instruction per second. Lastly came the 8085 microprocessors in 1976 having an ability of 769230 instruction per second with 3 MHz speed.

3. Third Generation – 16bit Microprocessor
The third generation began with 8086-88 microprocessors in 1978 with 4.77, 8 & 10 MHz speed and 2.5 million instructions per second. Other important inventions were Zilog Z800, and 80286, which came out in 1982 and could read 4 million instructions per second with 68 pins inside.

4. Fourth Generation – 32bit Microprocessors
Intel was still the leader as many companies came out with 32-bit microprocessors around 1986. Their clock speed was between 16 MHz to 33 MHz with 275k transistors inside.
One of the first ones was the Intel 80486 microprocessor of 1986 with 16-100MHz clock speed and 1.2 million transistors with 8 KB of cache memory. Followed by the PENTIUM microprocessor in 1993 which had 66 MHz clock speed and 8-bit of cache memory.

5. Fifth Generation – 64bit Microprocessors
Began in 1995, the Pentium processor was one of the first 64-bit processors with 1.2 GHz to 3 GHz clock speed. There were 291 million transistors and 64kb instruction per second. Followed by i3, i5, i7 microprocessors in 2007, 2009, 2010 respectively.

Advantages of Microprocessors
- High-speed processing
- Brings intelligence to the system
- Is flexible in nature
- Has a compact size
- Is easy to maintain

Disadvantages of Microprocessors
- Leads to overheating due to continuous use.
- The data size decides the performance
- Larger than microcontrollers
- Doesn't support floating-point operations

Classification of Microprocessor:
Based on size of data bus (Specification):

1. **4 – bit Microprocessor**: A 4-bit microprocessor is a type of central processing unit (CPU) that processes data and instructions in 4-bit chunks. It is a relatively simple and outdated form of microprocessor compared to the more common 8-bit, 16-bit, or 32-bit microprocessors.
2. **8 – bit Microprocessor:** An 8-bit microprocessor is a central processing unit (CPU) that processes data and instructions in 8-bit chunks. This means that it can handle and

manipulate data in 8-bit binary representations, making it more capable than its 4-bit counterparts but less powerful than more modern processors with larger bit widths

3. **16 – bit Microprocessor:** 16-bit microprocessors were widely used in personal computers during the 1980s and early 1990s, including systems like the IBM PC and the Apple Macintosh. They offered a significant boost in processing power and memory addressing capabilities, making them well-suited for a wide range of applications, from word processing and spreadsheets to early video games. However, as technology continued to advance, processors with larger bit widths, such as 32-bit and 64-bit, became the standard for handling increasingly complex computing tasks and larger memory spaces.
4. **32 – Microprocessor:** 32-bit microprocessors are widely used in various computing devices, including desktop and laptop computers, embedded systems, and some mobile devices. They can efficiently run a wide range of software and applications, making them versatile and capable of handling tasks such as web browsing, office productivity, multimedia playback, and more.
5. **64 – bit Microprocessor:** 64-bit microprocessors are the standard in modern computing systems, including desktop and laptop computers, servers, and high-performance computing clusters. They are capable of handling complex calculations, running memory-intensive applications, and managing vast amounts of data, making them well-suited for a wide range of tasks, from gaming and multimedia editing to scientific research and enterprise-level data processing.

Based on Application:

1. **General purpose Microprocessor:** General-purpose microprocessors are the workhorses of modern computing, serving as the primary processing units in a wide range of devices and systems. Their versatility and programmability make them well-suited for handling the ever-evolving and diverse needs of today's technology landscape.
2. **Microcontroller:** Microcontrollers play a crucial role in various industries and applications, including consumer electronics, automotive, healthcare, industrial automation, and IoT. They offer a cost-effective and efficient solution for tasks that require a dedicated computing device with real-time capabilities.
3. **Special purpose Microprocessor:** A special-purpose microprocessor, also known as an application-specific microprocessor or a dedicated microprocessor, is a type of microprocessor that is designed and optimized to perform a specific, narrowly defined set of tasks or functions. Unlike general-purpose microprocessors, which are versatile and capable of handling a wide range of computing tasks, special-purpose microprocessors are tailored to excel in one particular application or use case.

Based on architecture:

1. **Reduced Instruction Set Computer (RISC):** A Reduced Instruction Set Computer (RISC) is a type of microprocessor architecture that focuses on simplicity and efficiency in instruction execution. RISC architectures are characterized by a small and well-defined set of instructions, with each instruction typically performing a single, basic operation. The main principles behind RISC architecture are simplicity, reduced instruction set, and efficient pipelining.

2. **Complex Instruction Set Computer (CISC):** A Complex Instruction Set Computer (CISC) is a type of microprocessor architecture that emphasizes the execution of complex and multi-step instructions. In contrast to Reduced Instruction Set Computers (RISC), CISC processors have a more extensive and diverse instruction set, with each instruction capable of performing a more comprehensive set of operations.

Different Between RISC and CISC :-

RISC	CISC
It is a Reduced InstructionSet Computer.	It is a Complex Instruction SetComputer.
It emphasizes on software tooptimize the instruction set.	It emphasizes on hardware tooptimize the instruction set.
It is a hard-wired unit of programming in the RISCProcessor.	Microprogramming unit in CISCProcessor.
It requires multiple registersets to store the instruction.	It requires a single register set tostore the instruction.
RISC has simple decodingof instruction.	CISC has complex decoding ofinstruction.
Uses of the pipeline aresimple in RISC.	Uses of the pipeline are difficultin CISC.
It uses a limited number ofinstructions that requires less time to execute the instructions.	It uses a large number of instructions that requires more time to execute the instructions.

1.2. Microprocessor Bus organization: Data Bus, Address Bus and Control Bus

A microprocessor's bus organization refers to how the data and control signals are arranged and transmitted within the microprocessor and its interaction with the rest of the computer system. The bus organization plays a crucial role in determining how data is moved between the various components of a computer, such as the CPU, memory, and input/output devices.

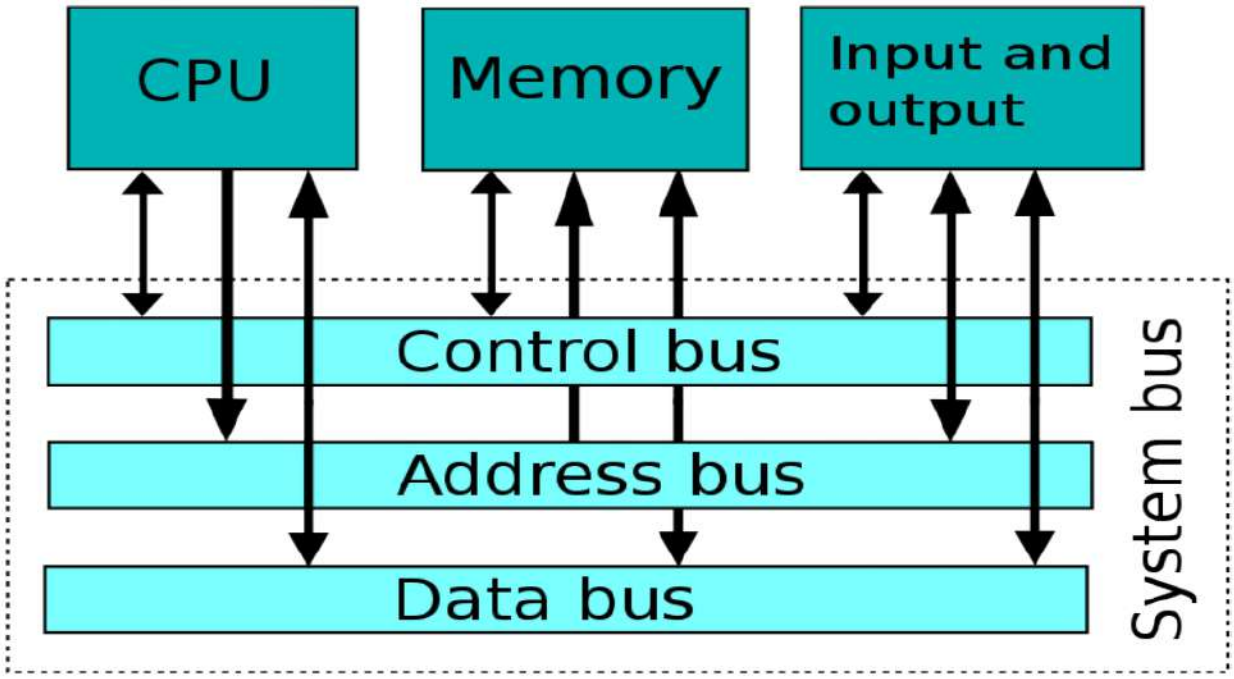


Fig: Block Diagram of System Bus

The bus consists of three main parts:

1. Control Bus: The control bus carries that control signal. The control signal is used for controlling and coordinating the various activities across the computer. It is generated from the control unit within the CPU. The control unit generates a specific **Control Signal** for every operation, such as memory rod. This signal is also used to identify a device type, with which the microprocessor communicates.

2. Address Bus: Bus carries the memory the address within the device allows the CPU to reference memory locations within the device. It connects the CPU and another peripheral and carries only the memory address. The address bus contains the connections between the processor and memory that carry the signals relating to the addresses which the **CPU** is processing at that time, such as the locations that the CPU is reading from or writing to. It addresses but could carry 8 bit at a time, the CPU could address only.

3. Data Bus: Transfer data from one location to another across the computer. The meaningful data which is to be sent from a device is placed on these lines. The CPU uses a data bus to transfer data. It May be 16 or 32-bit data bus. It is an **electrical** connects the CPU, memory, and other hardware devices on the motherboard. These lines are bidirectional data flow in both directions between the **processor** and memory and peripheral devices.

1.3. Operations of microprocessor: internal data manipulation, microprocessor initiated and peripheral or external initiated

Microprocessors execute instructions at a very high speed, often measured in gigahertz (GHz). They have various registers, data paths, and control units to ensure efficient execution of instructions. Additionally, they may have multiple cores, pipelining, and other features to further enhance performance.

All the operation of microprocessor can be classified into three types:

- Internal Data Manipulation
- Microprocessor initiated
- Peripheral or external initiated

Internal Data Manipulation:

Following operations are performed by microprocessor internally:

- Arithmetic and logic operation:
- Instruction Decode: The decoding process allows the processor to determine what instruction is to be performed so that the CPU can tell how many operands it needs to fetch in order to perform the instruction.
- Timing and control: It is also used to control the operations performed by the microprocessor and the devices connected to it. There are certain timing and control signals like Control signals, DMA Signals, RESET signals and Status signals.
- Sequence of execution: once we execute operation after fetch then program counter will be incremented by one automatically and ready to fetch next instruction. The microprocessor follows a sequence: Fetch, Decode, and then Execute.
- Testing of condition: There are many conditions that microprocessor deal with test like interrupt, compare operation, data manipulation with respect to their status all those task that deal with test inside microprocessor by using this internal operation.

Microprocessor initiated:

Microprocessor initiates following four operations

- Memory Read: The process of transferring the data stored in memory to processor or secondary storage is called read operation.
- Memory Write: The process of storing data in the memory is called write operation.
- IO read [Get data from input]
- IO write [Send data to output]

Peripheral or external initiated:

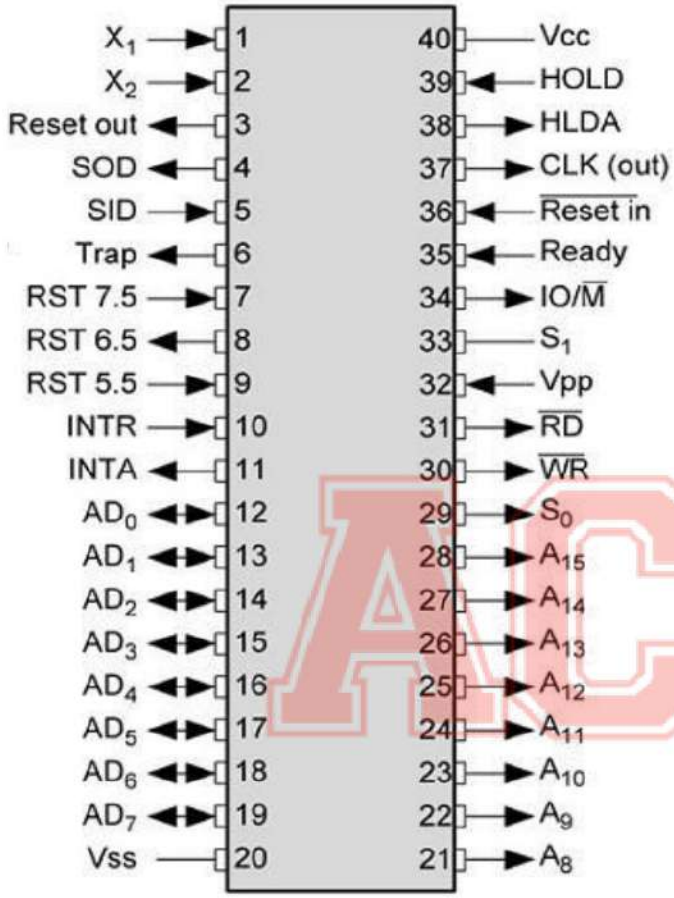
Following operations are initiated by peripheral to microprocessor

- RESET: RESET IN − This signal is used to reset the microprocessor by setting the program counter to zero. RESET OUT − This signal is used to reset all the connected devices when the microprocessor is reset.
- Ready: This signal indicates that the device is ready to send or receive data.
- Hold: The HOLD pin in a microprocessor is used to temporarily halt or pause the normal operation of the processor

- Interrupt: An interrupt is a condition that halts the microprocessor temporarily to work on a different task and then returns to its previous task.

1.4. Pin diagram of 8085 microprocessor:

40 pins are arranged in seven categories on the 8085 microprocessor's pin diagram: the address bus, the data bus, the control signals and status signals, the power supply and frequency, the reset signals, the DMA signals, and the serial input/output ports.



The pins of a 8085 microprocessor can be classified into seven groups –

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signaland 3 status signals.

Three control signals are RD, WR & ALE:-

- **RD** – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.

- **ALE** – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

  Three status signals are IO/M, S0 & S1.

  IO/M

  This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

  S1 & S0

  These signals are used to identify the type of current operation.

  Power supply

  There are 2 power supply signals – VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

  Clock signals

  There are 3 clock signals, i.e. X1, X2, CLK OUT.
- **X1, X2** – A crystal is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

  Interrupts & externally initiated signals

  Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.
- **INTA** – It is an interrupt acknowledgment signal.
- **RESET IN** – This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** – This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY** – This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** – This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** – It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) – The output SOD is set/reset as specified by the SIM (Set Interrupt Mask) instruction.
- **SID** (Serial input data line) – The data on this line is loaded into accumulator whenever a RIM (Read Interrupt Mask) instruction is executed.

1.5. Internal Architecture of 8085:

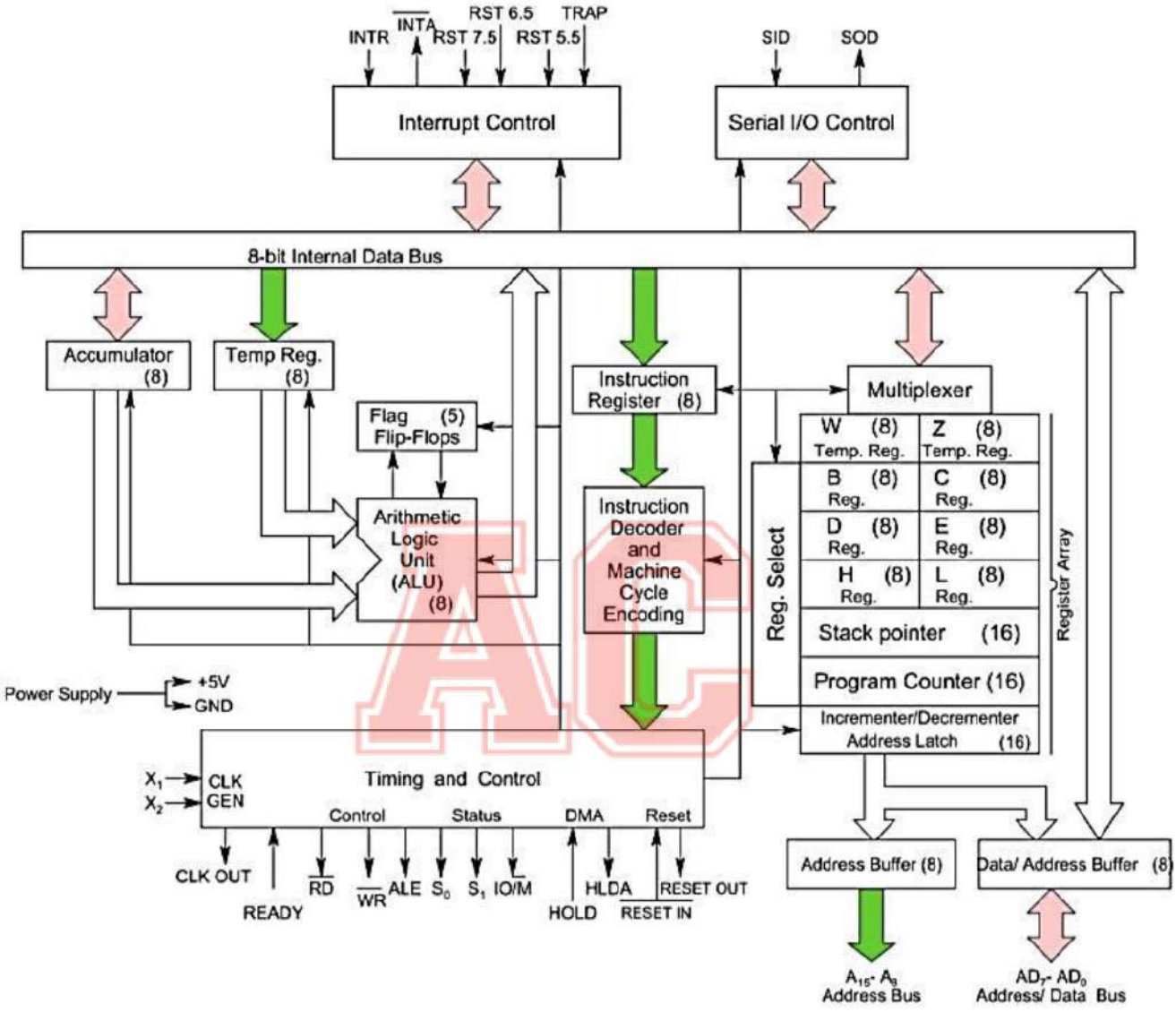


Fig: The 8085A microprocessor Functional Block Diagram

Source: Intel Corporation. *Embedded Microprocessors (Santa* Clara. Calif: Author.1994) pp 1-11

1. ALU (Arithmetic and Logic Unit):

ALU performs the computation functions and it includes accumulator, temporary registers, arithmetic and logic circuits, and five flags.

The temporary register holds the data temporarily during the arithmetic or logical operation of the processor.

Accumulator stores the final result of the arithmetic or logical operations.

The flags are the set of flip-flops. When the processor performs some operation the flags are set or reset according to the result of the operation.

2. Accumulator (register A):

The accumulator is an 8-bit register and also it is a part of ALU. This register can store 8 bits of data and is used during the arithmetic and logic operation of the processor.

As an accumulator register is used in arithmetic and logical operation of the process, the 8085 microprocessor is also termed an accumulator-based microprocessor. The data to be read from the input port is first moved in the accumulator or register 'A' and also the data to be sent to the output port is first stored in the accumulator.

3. Temporary Registers (W and Z):

W and Z are the temporary registers. These are 8-bit processors. Temporary registers are not accessible to the programmer. The processor stores data for a brief moment in these registers during the program execution.

4. Instruction Register (IR):

IR is also an 8-bit register. Like temporary registers, the IR register is also not accessible to the programmer. The operation codes of the instructions are first received by the IR. Then IR passes these operation codes or the opcodes to the instruction decoder which then decodes the information and finally, the microprocessor can know the type of operation which it has to perform.

5. Register Array (Register B, C, D, and E):

B, C, D, and E are the register array which is of 8 bits. These registers are available for the programmers during the programming of the 8085 microprocessor. The programmer can store data in these registers during program executions.

We can use these register arrays as 8-bit registers or in pairs such as BC, DE as 16-bit registers.

During program execution, we can add or transfer the data to and from the register. We can combine the contents of these registers with the accumulator's content.

Register H and L:

H and L are 8 bits registers. We can use these registers in the same way as that of the registers B, C, D, and E.

Stack Pointer (SP):

SP or stack pointer is a 16-bit register that works as a memory pointer. This register points to the stack which is the location in the R/W memory.

Program Counter (PC):

To sequence the execution of the instructions, the microprocessor uses a PC register. PC points to the memory address from which the next byte of information is to be fetched.

6. Flags:

Flags registers consist of a combination of five flip-flops. Each of these flip-flops will hold the status of different states of the arithmetic and logical operations performed by the microprocessor.

The figure below shows the flags of the 8085 microprocessor.

To make further decisions in the program we can use the status of the flag register.

The flags are described in detail below:

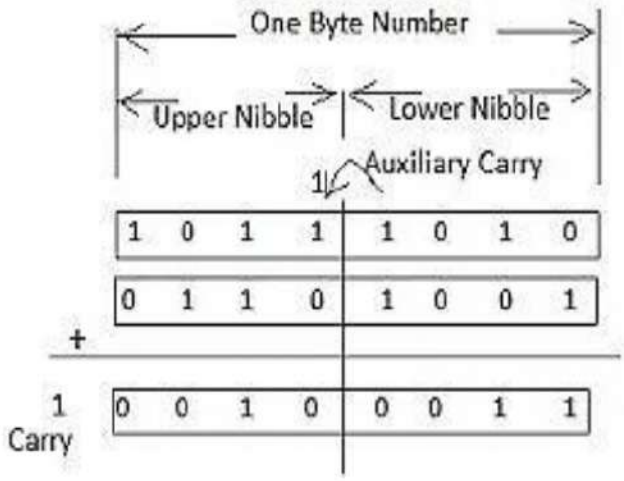
Carry Set (1) if the last operation generates carry otherwise reset (0).

Zero: If the result of the last operation of the processor is zero then this flag is set (1). Otherwise, the flag is reset (0).

Sign: Set (1) if the MSB of the result of the last operation is 1 (as MSB=1 signifies negative), otherwise reset (0).

Parity: Set (1) if the result of the last operation has even numbers of 1's (i.e. even parity) otherwise reset (0).

Auxiliary Carry: Set (1) if the last operation yield carries from the lower half-word otherwise reset (0).



7. Timing and Control Unit:

The timing unit synchronizes the operation of the microprocessor with the clock. The control unit is responsible for generating the control signals necessary for the microprocessor for communication between the microprocessor and the peripherals.
RD and WR are the control signals which indicate the availability of the data on the data bus.

8. Interrupt Control:

To handle the interrupt in the microprocessor there are various interrupt control signals such as INTR, RST 5.5, RST 6.5, RST 7.5, and TRAP.

9. Serial I/O Controls:

For serial data transmission, SID (Serial Input Data) and SSIO (Storage Systems and Input Output) are two serial I/O control signals available in the 8085 microprocessors.

1.6. Limitation of 8085 Microprocessor:

1. **Limited Instruction Set:** The 8085 had a relatively limited instruction set with only 74 instructions. This limited the flexibility and functionality of the microprocessor, making it less versatile compared to later processors with larger instruction sets.
2. **Low Processing Speed:** The 8085 operated at relatively low clock speeds, typically around 3 MHz. This limited its processing power and made it less suitable for high-performance applications.
3. **Limited Addressable Memory:** The 8085 could address a maximum of 64 KB of memory, which was a significant limitation for more complex applications. Modern microprocessors can address much larger memory spaces.
4. **Lack of Built-in Peripheral Interfaces:** The 8085 lacked built-in support for many peripheral devices and interfaces, which meant that external circuitry was often required for interfacing with external devices like keyboards, displays, and storage devices.
5. **Lack of Multiply and Divide Instructions:** The 8085 did not have dedicated instructions for multiplication and division, which meant that these operations had to be implemented using a sequence of other instructions, making them slow and resource-intensive.
6. **Limited Stack Size:** The 8085 had a small hardware stack, which limited the depth of subroutine calls and made it less suitable for complex and deeply nested programs.
7. **Single Accumulator Architecture:** The 8085 had only one general-purpose accumulator register, which made some operations less efficient since data often had to be moved between the accumulator and other registers.

8. **Limited Interrupt Handling:** The interrupt handling mechanism in the 8085 was relatively simple and less flexible compared to later microprocessors.
9. **Limited Number of Registers:** The 8085 had only a few general-purpose registers, which could limit the flexibility of programming and data manipulation.
10. **Limited Direct Memory Access (DMA) Support:** The 8085 had limited support for DMA operations, which are essential for efficient data transfer between memory and peripherals in many applications.

Unit 2. Instruction Cycle and Timing Diagram [Marks – 5]

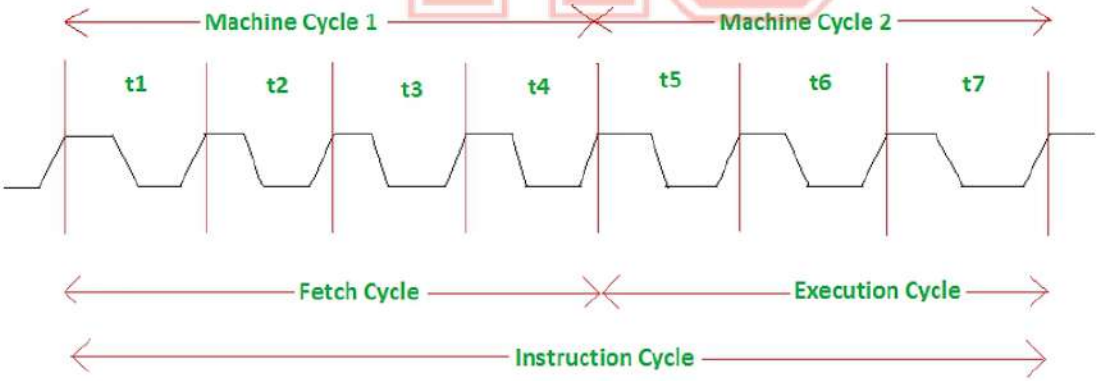
Machine Cycle:

The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called **machine cycle**. Following are the different types of machine cycle:

- **Opcode fetch**, which takes 4 t-states
- **Memory Read**, which takes 3 t-states
- **Memory Write**, which takes 3 t-states
- **I/O Read**, which takes 3 t-states
- **I/O Write**, which takes 3 t-states

One-time period of frequency of microprocessor is called **t-state**. A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

Fetch cycle takes four t-states and execution cycle takes three t-states. It is shown below:



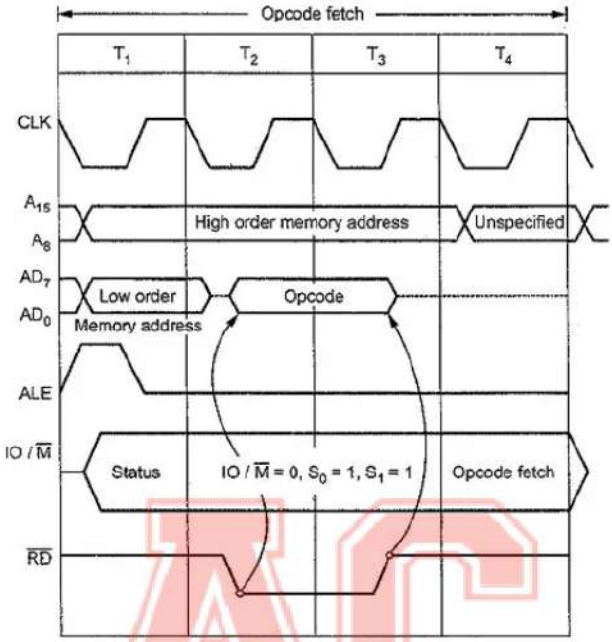
Instruction cycle in 8085 microprocessor

Machine Cycle	Status			Control		
	IO/$\overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	INTA
Opcode Fetch	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/O Read	1	1	0	0	1	1
I/O Write	1	0	1	1	0	1
INTR Acknowledge	1	1	1	1	1	0

Opcode fetch cycle:

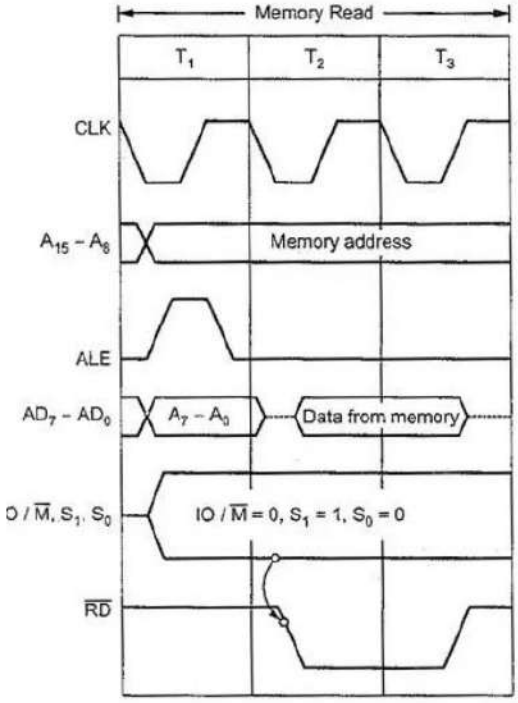
The first machine cycle of every instruction is opcode fetch cycle in which the 8085 finds the nature of the instruction to be executed. In this machine cycle, processor places the contents of the Program Counter on the address lines, and through the read process, reads the opcode of the instruction. The length of this cycle is not fixed. It varies from 4T states to 6T states as per the instruction.

Below figure shows timing diagram of opcode fetch:



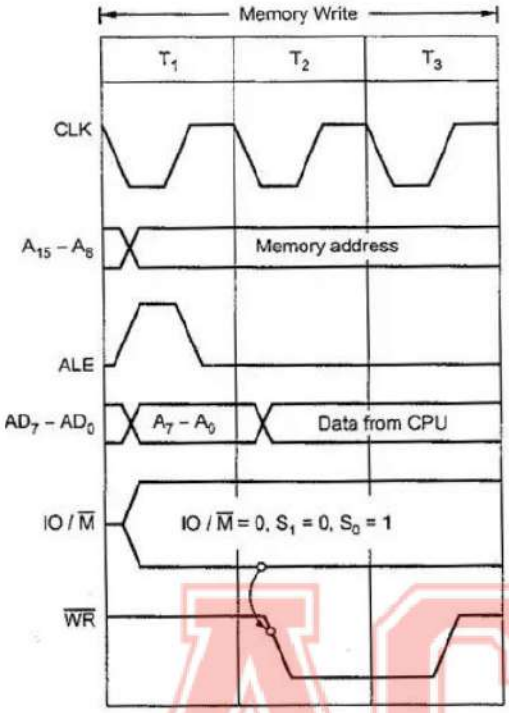
2.3. Memory Read Cycle

The 8085 executes the memory read cycle to read the contents of R/W memory or ROM. The length of this machine cycle is 3-T states (T1 – T3). In this machine cycle, processor places the address on the address lines from the stack pointer, general purpose register pair or program counter, and through the read process, reads the data from the addressed memory location.Timing diagram is shown below:



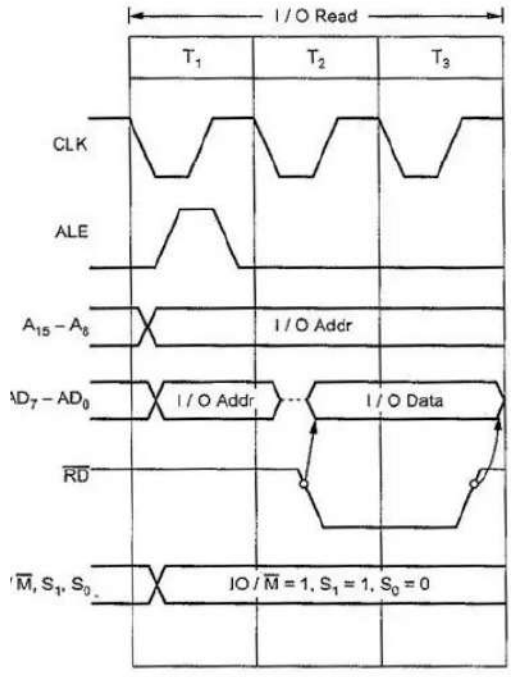
2.3. Memory Write Cycle

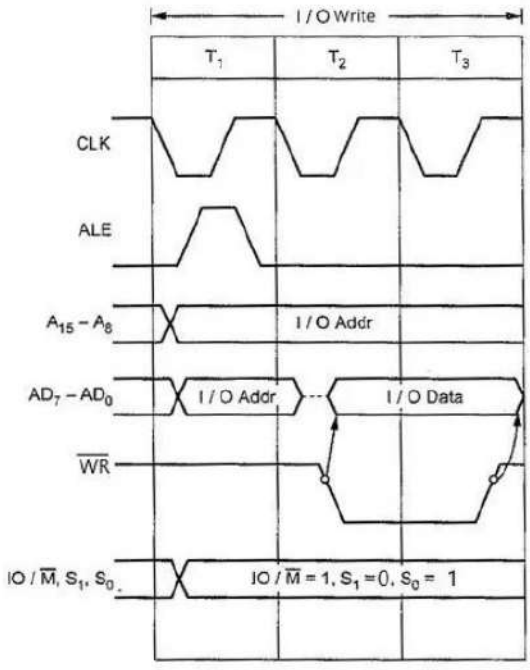
The 8085 executes the memory write cycle to store the data into data memory or stack memory. The length of this machine cycle is 3T states. (T1 — T3). In this machine cycle, processor places the address on the address lines from the stack pointer or general-purpose register pair and through the write process, stores the data into the addressed memory location.



2.4. I/O Read and I/O Write cycles

The I/O read and I/O write machine cycles are similar to the memory read and memory write machine cycles, respectively, except that the I0/M signal is high for I/O read and I/O write machine cycles. High IO/M signal indicates that it is an I/O operation.





Bus timings to fetch, decode, execute instruction from memory

The process of fetching, decoding, and executing instructions from memory is a fundamental concept in computer architecture, often associated with the Von Neumann architecture and the fetch-decode-execute cycle. This cycle is a simplified description of how a CPU (Central Processing Unit) processes instructions.

1. **Fetch**: During the fetch stage, the CPU retrieves the next instruction from memory. The program counter (PC) keeps track of the address of the next instruction to be executed. The CPU sends a request to the memory unit with the address stored in the PC, and the memory unit responds by providing the instruction.
2. **Decode**: Once the instruction is fetched, it needs to be decoded to understand what operation it represents. The CPU has a control unit that interprets the instruction and determines which operation to perform. This involves identifying the opcode (operation code) and any operands or addressing modes specified by the instruction.
3. **Execute**: After the instruction is decoded, the CPU executes the operation it represents. This may involve performing arithmetic or logical operations on data, transferring data between registers or memory, or making decisions based on conditional instructions.

The exact timing of the fetch, decode, and execute stages will vary depending on the CPU's architecture, clock speed, and specific microarchitecture features. In modern CPUs, these stages are designed to be highly efficient and may not correspond to a fixed number of clock cycles. Instead, the timing is dependent on various factors such as instruction complexity, cache hits/misses, and pipelining.

Here are some key points related to timing:

- **Clock Speed**: The speed at which the CPU operates is measured in Hertz (Hz) and is often referred to as the clock speed. Each clock cycle corresponds to a unit of time, and the speed at which the CPU executes instructions is determined by how many clock cycles are required for each stage.
- **Pipelining**: Many modern processors use a technique called pipelining to overlap the fetch, decode, and execute stages. In a pipelined architecture, while one instruction is being executed, the next instruction can be fetched and decoded in subsequent stages of the pipeline.
- **Caches**: Processors often have multiple levels of cache memory to store frequently used instructions and data. Accessing instructions and data from cache is faster than accessing them from main memory, which can affect the timing.
- **Branch Prediction**: In cases where conditional branches in the code can affect the flow of instructions, branch prediction is used to speculatively fetch and decode instructions. This can impact the timing if the prediction is incorrect.

Unit 3. 8085 Instruction set [Marks – 22]

3.1. Machine language instruction format:

An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the operation code (opcode), and the second is the data to be operated on, called the operand. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

The 8085-instruction set is classified into the following three groups according to word size:

a. One-word or 1-byte instructions

b. Two-word or 2-byte instructions

c. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

3.1.1. One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

Table 2.1 Example for 1 byte Instruction

Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (compliment) each bit in the accumulator.	CMA		0010 1111	2FH

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8- bit binary format in memory; each requires one memory location.

Example:

ADD B

(Add the contents of register B to the contents of the accumulator).

3.1.2. Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

Example for 2-byte Instruction:

Table 2.2 Example for 2 byte Instruction

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110 DATA	3E Data	First Byte Second Byte

The instruction would require two memory locations to store in memory.

Example:

MVI A, 32H

(Load an 8-bit data byte in the accumulator)

3.1.3. Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

opcode + data byte + data byte

Example for 3-byte Instruction

Table 3.3 Example for 3 byte Instruction

Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011 1000 0101 0010 0000	C3 85 20	First byte Second Byte Third Byte

This instruction would require three memory locations to store in memory.

Three-byte instructions - opcode + data byte + data byte.

Example:

LDA 2050 H

(Load contents of memory 2050H into A).

3.2. Various addressing modes:

The way of specifying data to be operated by an instruction is called addressing mode. Addressing mode are used in microprocessor for flexibility, memory optimization, performance optimization, reduced code size.

In 8085 microprocessor there are 5 types of addressing modes:

1. **Immediate Addressing Mode:** In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Examples:

MVI B 45 (move the data 45H immediately to register B)

LXI H 3050 (load the H-L pair with the operand 3050H immediately)

JMP address (jump to the operand address immediately)

MVI A 00H: 2byte

MVI B, 00H: 2byte

MVI C, 00H: 2byte

2. **Register Addressing Mode –**

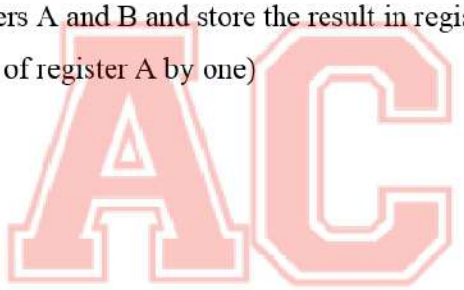
   In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore, the operation is performed within various registers of the microprocessor.

Examples:

MOV A, B (move the contents of register B to register A)

ADD B (add contents of registers A and B and store the result in register A)

INR A (increment the contents of register A by one)



MVI A 00H: 2byte

MOV B, A: 1byte

MOV C, A: 1byte

3. **Direct Addressing Mode –**

In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.

Examples:

LDA 2050 (load the contents of memory location into accumulator A)

LHLD address (load contents of 16-bit memory location into H-L register pair)

IN 35 (read the data from port whose address is 35)

4. **Register Indirect Addressing Mode –**

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

Examples:

MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)

LDAX B (move contents of B-C register to the accumulator)

STAX B (store accumulator contents in memory pointed by register pair B-C)

INX B (increment the address)

5. Implied/Implicit Addressing Mode –

In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.

Examples:

CMA (finds and stores the 1's complement of the contents of accumulator A in A)

RRC (rotate accumulator A right by one bit)

RLC (rotate accumulator A left by one bit)

6.Relative Addressing Mode:

In this mode, the operand is a memory location specified by the contents of the program counter plus a constant value.

example:

MOV R0,#05H

AGAIN:

MVI A,#55H

ADD A,R0

JMP AGAIN

In this example, the instruction JMP AGAIN uses the Relative Addressing Mode. The instruction jumps to the label AGAIN by adding the contents of the program counter with the specified constant value. The constant value is calculated based on the distance between the current instruction and the label AGAIN.

In this case, the instruction JMP AGAIN is executing a loop that loads the accumulator with the value 55H, adds the contents of the register R0 to the accumulator, and then jumps back to the label AGAIN to repeat the loop.

3.3. Data Transfer Operation and Instruction:

Data transfer is the process of copying data from one location to another. The transferred data may be transformed in transit, or arrive at its destination as-is. Data transfer instructions in a microprocessor or microcontroller are used to move data between registers, memory locations, and other data storage elements. These instructions play a crucial role in manipulating data within the system.

Load (LDA, LDAX, MOV): These instructions transfer data from memory or registers to a register.

- Example (8085 Assembly Language):

MOV A, M ; Move data from memory to accumulator

MOV B, C ; Move data from register C to register B

Store (STA, STAX): These instructions transfer data from a register to a memory location.

- Example (8085 Assembly Language):

STA 3000H ; Store accumulator content to memory address 3000H

STAX D ; Store accumulator content to memory location specified by register pair DE

Exchange (XCHG): This instruction swaps the content of two register pairs.

- Example (8085 Assembly Language):

XCHG ; Swap content of register pairs HL and DE

Load Immediate (MVI): This instruction loads an immediate value into a register or memory location.

- Example (8085 Assembly Language):

MVI A, 42H ; Load immediate value 42H into the accumulator

Push and Pop Operations (PUSH, POP): These instructions are commonly used in stack-based architectures to push data onto the stack or pop data from the stack.

- Example (8085 Assembly Language):

PUSH B ; Push the content of register pair BC onto the stack

POP D ; Pop the top of the stack into register pair DE

3.4. Arithmetic operation and instruction

Arithmetic Instructions in 8085 Microprocessor includes the instructions, which performs the addition, subtraction, increment or decrement operations. The flag conditions are altered after execution of an instruction in this group.

The descriptions of the instructions (including INR and DCR) are as follows:

Opcode	Operand	Description
ADD	R	ADD ▪ This is 1-byte instructions. ▪ Adds the contents of register R to the contents of the accumulator.
ADI	8-bit	Add Immediately ▪ This is 2-byte instructions. ▪ Adds the second byte to the contents of the accumulator
SUB	R+	Subtract ▪ This is 1-byte instructions. ▪ Subtract the contents of register R from the contents of theaccumulator.
SUI	8-bit	Subtract Immediately ▪ This is 2-byte instructions. ▪ Subtracts the byte from the contents of the accumulator.

INR	R*	Increment ▪ This is 1-byte instructions. ▪ Increases the contents of register R by 1 ***Caution:*** All flags except the CY are affected
DCR	R*	Decrement ▪ This is 1-byte instructions. ▪ Decreases the contents of register R by 1 ***Caution:*** All flags except the CY are affected.

3.5. Logical operation and instruction:

Logical instructions in the 8085 microprocessors are a set of instructions that perform logical operations on data in registers and memory. Logical operations are operations that manipulate the bits of data without affecting their numerical value. These operations include AND, OR, XOR, and NOT.

Opcode	Operand	Meaning	Explanation
CMP	R M	Compare the register or memory with the accumulator	The contents of the operand (register or memory) are M compared with the contents of the accumulator.
CPI	8-bit data	Compare immediate with the accumulator	The second byte data is compared with thecontents of the accumulator.
ANA	R M	Logical AND register or memory with the accumulator	The contents of the accumulator are logicallyAND with M the contents of the register or memory, and the result is placed in the accumulator. (*)

ANI	8-bit data	Logical AND immediate with the accumulator	The contents of the accumulator are logically AND with the 8-bit data and the result is placedin the accumulator.
XRA	R M	Exclusive OR register or memory with the accumulator	The contents of the accumulator are ExclusiveOR with M the contents of the register or memory, and the result is placed in the accumulator. (same = 0 else =1)
ORI	8-bit data	Logical OR immediate with the accumulator	The contents of the accumulator are logically OR with the 8-bit data and the result is placed in the accumulator.
RLC	None	Rotate the accumulator left	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.
RRC	None	Rotate the accumulator right	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.
RAL	None	Rotate the accumulator left through carry	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.

RAR	None	Rotate the accumulator right through carry	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
CMA	None	Complement accumulator	The contents of the accumulator are complemented. No flags are affected.
CMC	None	Complement carry	The Carry flag is complemented. No other flags are affected.
STC	None	Set Carry	Set Carry

3.6. Branch operation and instruction:

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction.

The three types of branching instructions are:

1. Jump (unconditional and conditional)
2. Call (unconditional and conditional)
3. Return (unconditional and conditional)

1. Jump Instructions – The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JMP	address	Jumps to the address	JMP 2050

(b) Conditional Jump Instructions: Transfers the program sequence to the described memory address only if the condition in satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JC	address	Jumps to the address if carry flag is 1	JC 2050
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

2. Call Instructions – The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

(a) Unconditional Call Instructions: It transfers the program sequence to the memory address given in the operand.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

(b) **Conditional Call Instructions:** Only if the condition is satisfied, the instructions execute.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050
CZ	address	Calls if zero flag is 1	CZ 2050
CNZ	address	Calls if zero flag is 0	CNZ 2050
CPE	address	Calls if parity flag is 1	CPE 2050
CPO	address	Calls if parity flag is 0	CPO 2050
CM	address	Calls if sign flag is 1	CM 2050
CP	address	Calls if sign flag is 0	CP 2050

3. Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Return instructions are 2 types: Unconditional return Instructions and Conditional return Instructions.

(a) **Unconditional Return Instruction:** The program sequence is transferred unconditionally from the subroutine to the calling program.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RET	none	Return from the subroutine unconditionally	RET

(b) **Conditional Return Instruction:** The program sequence is transferred unconditionally from the subroutine to the calling program only is the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RC	none	Return from the subroutine if carry flag is 1	RC
RNC	none	Return from the subroutine if carry flag is 0	RNC
RZ	none	Return from the subroutine if zero flag is 1	RZ
RNZ	none	Return from the subroutine if zero flag is 0	RNZ
RPE	none	Return from the subroutine if parity flag is 1	RPE
RPO	none	Return from the subroutine if parity flag is 0	RPO
RM	none	Returns from the subroutine if sign flag is 1	RM
RP	none	Returns from the subroutine if sign flag is 0	RP

3.7. Stack operation and instruction:

The 8085 microprocessor provides a set of instructions that can be used to manipulate the stack. These instructions include: PUSH: This instruction is used to push a register or memory location onto the stack.

The stack operates on the **Last In, First Out** (LIFO) principle. The location of the most recent data on the stack is known as the **TOP** of the stack. The stack pointer always points to the top of the stack. Contents can be stored in the stack using the PUSH instruction and can restore the contents by using the instruction POP

MNEMONIC	DESCRIPTION
LXI SP, 16-bit	Load the stack pointer register with a 16-bit address.
PUSH Rp	Copies the contents of the specified register pair on the stack

POP Rp	Copies the contents of the top two memory locations of the stack into the specified register pair.

PUSH and POP Operation in 8085

PUSH R_p

This is a 1-byte instruction. This instruction copies the contents of the specified register pair on the stack as described below:

- The stack pointer is decremented and the contents of the higher-order register are copied to the location shown by the stack pointer register.
- The stack pointer is again decremented and the contents of the low-order register are copied to that location.

POP R_p

This is a 1-byte instruction. This instruction copies the contents of the top two memory locations of the stack into the specified register pair.

- First, the contents of the memory location indicated by the stack pointer register are copied into the low-order register and then the stack pointer register is incremented by 1.
- The contents of the next memory location are copied into the high-order register and the stack pointer register is again incremented by 1.

Example

LXI SP,2099H

LXI H, 42F2H

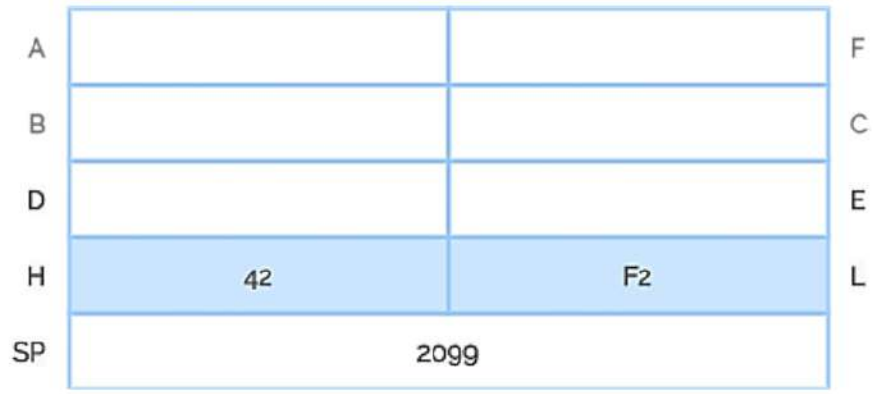
PUSH H

Delay Counter

POP H

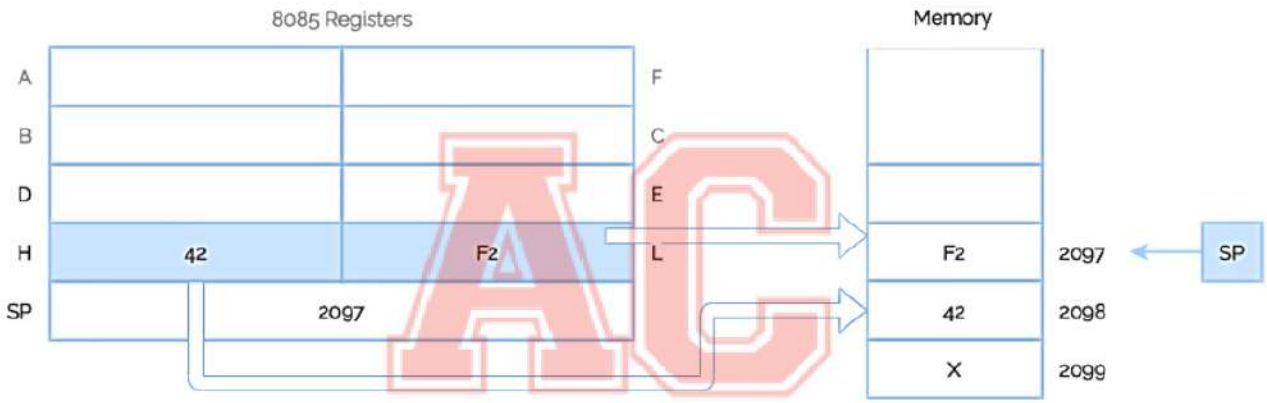
- The instruction **LXI** SP, *2099H* will initialize the stack pointer with the address of 2099H.
- **LXI** H, *42F2H* will initialize or load HL register pair with *42F2H* data so **H** = 42 and **L** = F2

.



- After the execution of **PUSH** H instruction, the stack pointer is decreased by one to *2098H* and the contents of the H register are copied to memory location *2098H*.
- The stack pointer is again decreased by one to *2097H* and the contents of the L register are copied to memory location *2097H*

.



3.8. Input/output and machine control operation and instruction:

Input/output operation and instruction:

In 8085 Instruction set, there are two instructions in 8085 for communication with I/O ports. They are the IN and OUT instructions. The IN or OUT instruction mnemonics should be followed by an 8-bit port address. Thus, we can have $2^8 = 256$ input ports and 256 output ports are possible in 8085-based microcomputers. IN and OUT both are 2-Bytes instructions.

In: initiates input operations. IN 80H

Out: initiates output operation. OUT C0H

Machine Control Instructions:

These instructions control machine functions such as Halt, Interrupt, or do nothing.

Types of Machine Control Instructions:

1. NOP (No operation)
2. HLT (Halt)
3. DI (Disable interrupts)
4. EI (Enable interrupts)
5. SIM (Set interrupt mask)
6. RIM (Reset interrupt mask)

1. NOP (No operation):

Opcode- 00
Operand- None
Length- 1 byte
M-Cycles- 1
T-states- 4
Hex code- 00

NOP is used when no operation is performed. It is commonly used to fill in time delay or to delete and insert instructions while troubleshooting. During the execution of NOP, no flags are affected.

2. HLT (Halt)

Opcode- 76
Operand- None
Length- 1 byte
M-Cycles- 2 or more
T-states- 5 or more
Hex code- 76

HLT is used to stop the execution of the program temporarily. The microprocessor finishes executing the current instruction and halts any further execution. The contents of the registers are unaffected during the HLT state. HLT can be used to enter a wait state in which the microprocessor waits for a specific event to occur before resuming execution.

3. DI (Disable interrupts)

Opcode- F3
Operand- None
Length- 1 byte
M-Cycles- 1
T-states- 4
Hex code- F3

DI is used when the execution of a code sequence cannot be interrupted. For example, in critical time delays, this instruction is used at the beginning of the code and the interrupts are enabled at the end of the code. The TRAP interrupt cannot be disabled.

4. EI (Enable interrupts)

Opcode- FB
Operand- None
Length- 1 byte
M-Cycles- 1
T-states- 4
Hex code- FB

EI is used to enable interrupts after a system reset or the acknowledgement of an interrupt. The Interrupt Enable flip-flop is reset, thus disabling the interrupts.

5. SIM (Set interrupt mask)

Opcode- 30
Operand- None
Length- 1 byte
M-Cycles- 1
T-states- 4
Hex code- 30

- Masking/unmasking of RST7.5, RST6.5, and RST5.5
- Reset to 0 RST7.5 flip-flop.
- Perform serial output of data.

SIM is used for the implementation of different interrupts of 8085 microprocessor like RST 7.5, 6.5, and 5.5 and also serial data output. It does not affect the TRAP interrupt.

6. RIM (Reset interrupt mask)

Opcode- 20
Operand- None
Length- 1 byte
M-Cycles- 1
T-states- 4
Hex code- 20

- To check whether RST7.5, RST6.5, and RST5.5 are masked or not.
- To check whether interrupts are enabled or not.
- To check whether RST7.5, RST6.5, or RST5.5 interrupts are pending or not.
- To perform serial input of data.

RIM is a multipurpose instruction used to read the status of 8085 interrupts 7.5, 6.5, 5.5, and to read serial data input bit.

3.9. Simple programs with 8085 instructions

1. Program to add two 8-bit numbers.

Statement: Add numbers 05H & 13H and display result in output port 03H.MVI A,

05H //Move data 05H to accumulator

MVI B, 13H //Move data 13H to B register

ADD B //Add contents of accumulator and B registerOUT 03H

//Transfer result to output port 03H

HLT //Terminate the program.

Input: A=05H B=13H Output: (port 03H) = 18H

2. Program to add two 8-bit numbers.

Statement: *Add numbers from memory location 2050H & 2051H and store result in memory location 2055H.*

```
LDA 2051H   //Load contents of memory location 2051 to accumulator
MOV B, A    //Move contents of accumulator to B register
LDA 2050H   //Load contents of memory location 2050 to accumulator
ADD B       // Add contents of accumulator and B register
STA 2055H   //Store contents of accumulator in memory location 2055H
HLT         //Terminate the program.
```

Input:

Memory LocationData

2050H 45H

2051H 53H

Output:

Memory LocationData

2055H 98H

3. Program to subtract two 8-bit numbers.

Statement: *Subtract numbers 25H & 12H and display result in output port 01H.*

MVI A, 25H //Move data to accumulator

MVI B, 12H //Move data to B register

SUB B //Subtract contents of accumulator and B register

OUT 01H //Transfer result to output port 01H

HLT //Terminate the program.

Input: A=25H B=12H **Output**: (port 01H) = 13H

4. Program to subtract two 8-bit numbers.

Statement: *Subtract numbers from memory location 2050H & 2051H and store result in memory location 2055H.*

LDA 2051H//Load contents of memory location 2051 to accumulator

MOV B, A //Move contents of accumulator to B register

LDA 2050H //Load contents of memory location 2050 to accumulator

SUB B // Subtract contents of accumulator and B register

STA 2055H //Store contents of accumulator in memory location 2055H

HLT //Terminate the program.

Input:

Memory LocationData

2050H 65H

2051H 53H

Output:

Memory Location Data

2055H 12H

5. Program to multiply two 8-bit numbers.

Ex. 04 * 03

```
LXI H, C050H
MOV B, M        // (B= 04H)
MVI A, 00H
MVI D, 00H
INX H
MOV C, M      //(C=03)
(loop2) ADD B
JNC  (loop)
INR D
(loop) DCR C
JNZ (loop2)
STA C052
MOV A, D
STA C053
HLT
```

Statement: Multiply 06 and 03 and store result in memory location 2055H.

```
MVI A, 00H
 MVI B, 06H
MIV C, 03H
X: ADD B
DCR C
JNZ X
STA 2055H
HLT
```

6. Program to divide to 8-bit numbers.

Statement: Divide 08H and 03H and store quotient in memory location 2055H and remainder in memory location 2056H.

```
MVI A, 08H
MVI B, 03H
```

```
MVI C, 00H
X: CMP B
JC Y
SUB B
INR  C
JMP X
Y: STA 2056H
MOV A, C
STA 2055H
HLT
```

Division of two numbers example: 10/ 2

```
LXI H, C050H
MOV B, M (B <- 0AH)
INX H
MOV C, M (C<- 02H)
MVI D, 00H
MOV A, B (A<-0A)
SUB C (A<- 08H)
INR D (D=1,2,3…)
CMP C (compare A with C)
```

[A>C; CY-0, Z-0 A<C; CY-1, Z-0 A=C; CY-0, Z=1]

```
JNC loop
STA C052H
MOV A, D
STA C053H
HLT
```

Unit 4. Basic Computer Architecture [Marks: 7]

Computer architecture refers to the design and organization of the components that make up a computer system. It encompasses both the hardware and software aspects of a computer.

The main components of a computer architecture are the CPU, memory, and peripherals. All these elements are linked by the system bus, which comprises an address bus, a data bus, and a control bus.

4.1.1. History of Computer Architecture

First Generation (1940s-1950s):

- The earliest computers, such as the ENIAC (Electronic Numerical Integrator and Computer) and UNIVAC (Universal Automatic Computer), were built during this era.
- These machines used vacuum tubes for electronic components and were programmed using plugboards and wires.
- The von Neumann architecture, which separates program and data storage, was proposed in the mid-1940s and became the foundation for most subsequent computer designs.

Second Generation (1950s-1960s):

- Transistors replaced vacuum tubes, leading to smaller, more reliable, and faster computers.
- Batch processing systems were developed, allowing users to submit jobs in groups, and the computer would process them in sequence.

Third Generation (1960s-1970s):

- Integrated circuits (ICs) were introduced, combining multiple transistors on a single chip.
- Multiprogramming and time-sharing systems were introduced, enabling multiple users to interact with the computer simultaneously.
- Operating systems and high-level programming languages became more prevalent.

Fourth Generation (1970s-1980s): Microprocessors and Personal Computers

- The invention of the microprocessor marked a significant shift, as entire CPUs could now be integrated onto a single chip.
- The advent of personal computers, such as the Apple II and IBM PC, brought computing to individuals and small businesses.

Fifth Generation (1980s-Present): VLSI, RISC, and Parallel Processing

- Very Large Scale Integration (VLSI) technology allowed for even greater integration of components on a chip.
- Reduced Instruction Set Computing (RISC) architectures emerged, streamlining instruction sets for improved performance.
- Parallel processing became more prominent, with the development of multiprocessor and multicore systems.

Sixth Generation (1990s-Present): Networked and Distributed Computing

- The rise of the internet and advancements in networking led to widespread connectivity and the development of distributed computing.
- Mobile computing, cloud computing, and virtualization technologies became prominent.

Seventh Generation (2000s-Present): Power Efficiency and Specialized Processors

- Energy efficiency became a focus, leading to the development of power-efficient processors and mobile devices.
- Specialized processors, such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs), gained importance in various applications.

Eighth Generation (2010s-Present): Machine Learning and Quantum Computing

- Advancements in machine learning and artificial intelligence have influenced the development of specialized hardware for these tasks.
- Quantum computing research has progressed, aiming to leverage the principles of quantum mechanics for unprecedented computing capabilities.

The history of computer architecture reflects a constant quest for improved performance, increased functionality, and greater efficiency. Each generation has brought about innovations that have shaped the landscape of modern computing. Advances in materials, manufacturing, and design principles continue to drive the evolution of computer architecture into the future.

4.1.2. Overview of Computer Organization

Computer Organization is realization of what is specified by the computer architecture. It dealswith how operational attributes are linked together to meet the requirements specified by computer architecture. Some organizational attributes are hardware details, control signals, peripherals.

Difference Between Computer Organization and Architecture

Computer Architecture	Computer Organization
The computer system's structure and behavior as seen by the user, is concerned here.	The connection of the computer's hardware system is concerned with forming a computersystem.
We decide on computer architecturewhile designing computer systems.	We decide organization after architectureWhile designing a computer system.

How the computer system's designing is done is described here.	How the computer system works is describedhere.
The computer system's logical aspectsare described here.	The computer system' physical aspects are described here.

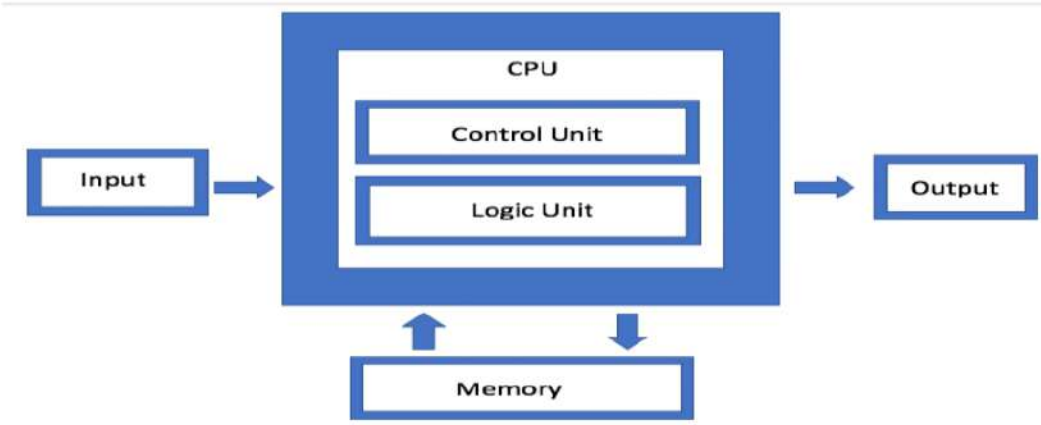
It is also called an instruction set architecture.	It is also known as microarchitecture.
The system's hardware and software are coordinated by architecture.	A computer organization handles thenetwork's segment in a system.
Between hardware and software, it actsas an interface.	It is concerned with the components of a computer and its connections.
High-level design issues are addressedby computer architecture.	Computer organizations address low-leveldesign issues.
It consists of logical components such as Instruction Set, Addressing Modes, and so on.	It involves physical units like circuit design, adders, signals, and peripherals.
The software programmer can see it.	It is apparent to the software programmer.

Computer System Functional Units

A computer is made of five important functional units. These units are part of the CPU (Central Processing Unit). These are as follows:

- Input Unit
- Memory Unit
- Arithmetic Logic Unit (ALU)
- Output Unit
- Control Unit

We can illustrate these with the diagram below:



Input Unit

It is responsible for reading the data. It takes the data from the user directly or through commands. This sends the data or the field upon which the task will be done and what task will be done. It interacts with the I/O devices and the computer. Thus, it uses peripheral devices for it. Some examples include keyboard, mouse, touchpad, etc.

ALU (Arithmetic and Logical Unit)

It is responsible for performing the arithmetic and logical operations. Arithmetic operations include addition, subtraction, multiplication, division, and comparing numbers. And, logical operations include AND, OR, comparison, etc. It presents the instructions to the system and stores the results in the memory. It also stores the results in the registers. It has various operands placed into registers. Operands store one word at a time. They help in the quick retrieval of data.

Output Unit

It is responsible for displaying the result and giving outputs in a language understandable by humans (converting it from the binary result). Some examples include printers, monitors, etc.

Control Unit

It is responsible for controlling the data flow sent and received by the CPU. It acts as a coordinator. It is responsible for handling units for taking inputs, placing them in the memory, processing them, and generating the final output. It also acts as an interpreter as it decodes information, converts it into executable format, and commands other components to take action.

Memory Unit

It is responsible for storing the data and instructions. It stores two types of data: the operations to be formed and the data related to the program. The memory is divided into three types: primary, secondary, and cache memory.

Primary memory or RAM (Random Access Memory) is the main storage part. Programs for execution are placed into this memory. It is fast, expensive, and has a volatile memory. Secondary Memory is the hard disk of the computer. Also, pen drives, SSD cards, etc., are modern types of secondary memory. It is slow, less expensive, and has greater storage capacity. Cache memory is the supplementary memory. The process of loading programs from secondary to primary memory is slow, so the computer loads the frequently loaded programs

into the cache memory for fast retrieval. It stores the data temporarily and is small in size.

4.1.3. Memory Hierarchy and cache

The computer memory hierarchy looks like a pyramid structure which is used to describe the differences among **memory types**. It separates the computer storage based on hierarchy.

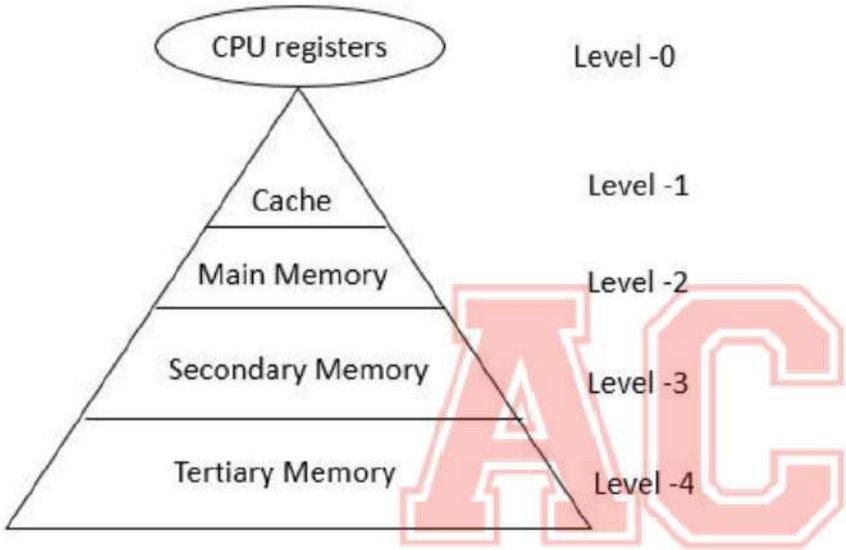
Level 0: CPU registers

Level 1: Cache memory

Level 2: Main memory or primary memory

Level 3: Magnetic disks or secondary memory

Level 4: Optical disks or magnetic types or tertiary Memory



In Memory Hierarchy the cost of memory, capacity is inversely proportional to speed. Here the devices are arranged in a manner Fast to slow, that is form register to Tertiary memory.

Level-0 – Registers

The registers are present inside the **CPU**. As they are present inside the CPU, they have least access time. Registers are most expensive and smallest in size generally in kilobytes. They are implemented by using Flip-Flops.

Level-1 – Cache

Cache memory is used to store the segments of a program that are frequently accessed by the processor. It is expensive and smaller in size generally in Megabytes and is implemented by using static **RAM**.

Level-2 – Primary or Main Memory

It directly communicates with the CPU and with auxiliary memory devices through an I/O processor. **Main memory** is less expensive than cache memory and larger in size generally in Gigabytes. This memory is implemented by using **dynamic RAM**.

Level-3 – Secondary storage

Secondary storage devices like **Magnetic Disk** are present at level 3. They are used as backup storage. They are cheaper than main memory and larger in size generally in a few TB.

Level-4 – Tertiary storage

Tertiary storage devices like magnetic tape are present at level 4. They are used to store removable files and are the cheapest and largest in size (1-20 TB).

4.2. Instruction Codes

An **instruction code** is a group of bits that tells the computer to perform a specific operation part.

Instruction Code: Operation Code

The operation code of an instruction is a group of bits that define operations such as add, subtract, multiply, shift and compliment. The number of bits required for the operation code depends upon the total number of operations available on the computer. The operation code must consist of at least **n bits** for a given **2^n** operations. The operation part of an instruction code specifies the operation to be performed.

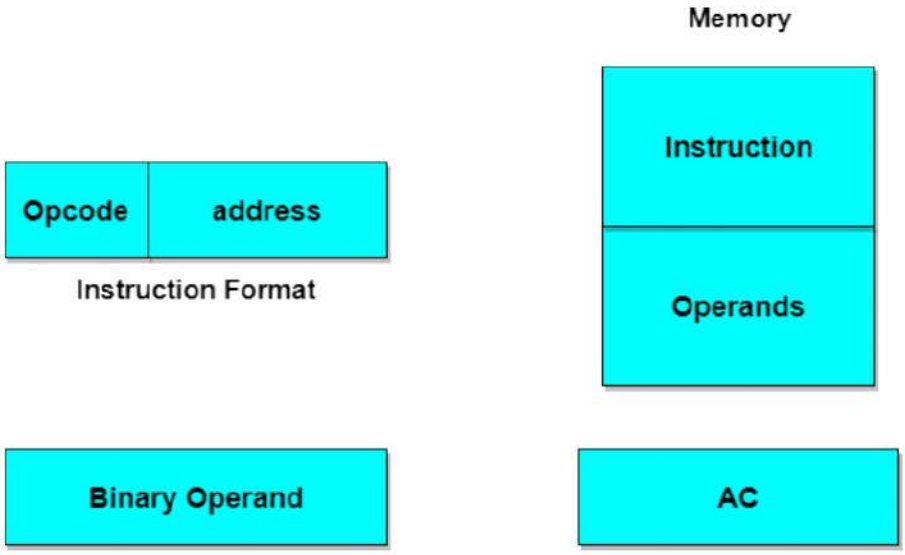
Instruction Code: Register Part

The operation must be performed on the data stored in registers. An instruction code therefore specifies not only operations to be performed but also the registers where the operands(data) will be found as well as the registers where the result has to be stored.

4.3. Stored Program Organization

The simplest way to organize a computer is to have **Processor Register** and instruction code with two parts. The first part specifies the operation to be performed and second specifies an address. The memory address tells where the operand in memory will be found.

Instructions are stored in one section of memory and data in another.



Computers with a single processor register is known as **Accumulator (AC)**. The operation is performed with the memory operand and the content of AC.

The operand address parts.

When the 2nd part of an instruction code specifies the operand, the instruction is said to have **immediate operand**. And when the 2nd part of the instruction code specifies the address of an operand, the instruction is said to have a **direct address**. And in **indirect address**, the 2nd part of instruction code, specifies the address of a memory word in which the address of the operand is found.

Computer Instructions

The basic computer has three instruction code formats.

There are three types of formats:

1. Memory Reference Instruction

It uses 12 bits to specify the address and 1 bit to specify the addressing mode (**I**). **I** is equal to 0 for *direct address* and 1 for *indirect address*.

2. Register Reference Instruction

These instructions are recognized by the opcode 111 with a 0 in the left most bit of instruction. The other 12 bits specify the operation to be executed.

Register Reference Instruction

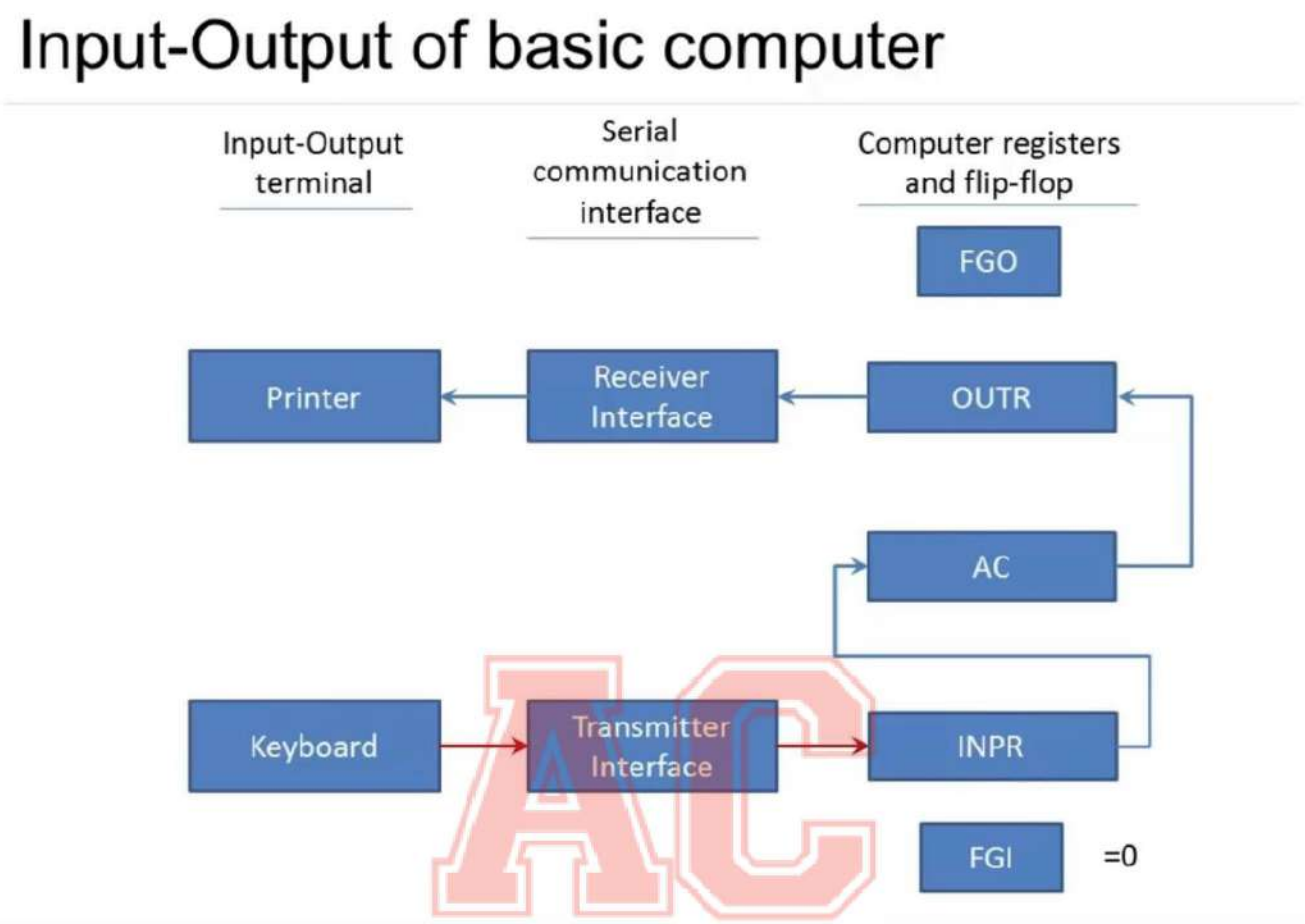
$D_7 I' T_3 = r$ (common to all register reference instructions)

$IR(i) = B_i$ [bit in IR(0-11) that specifies the operation]

CLA	rB_{11}	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}	$E \leftarrow 0$	Clear E
CMA	rB_9	$AC \leftarrow AC'$	Complement AC
CME	rB_8	$E \leftarrow E'$	Complement E
CIR	rB_7	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	rB_5	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if AC is positive
SNA	rB_3	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if AC is negative
SZA	rB_2	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	Skip if AC is zero
SZE	rB_1	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E is zero
HLT	rB_0	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt Computer

3. Input-Output Instruction

These instructions are recognized by the operation code 111 with a 1 in the left most bit of instruction. The remaining 12 bits are used to specify the input-output operation.



Format of Instruction

The format of an instruction is depicted in a rectangular box symbolizing the bits of an instruction. Basic fields of an instruction format are given below:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates the memory address or register.
3. A mode field that specifies the way the operand of effective address is determined.

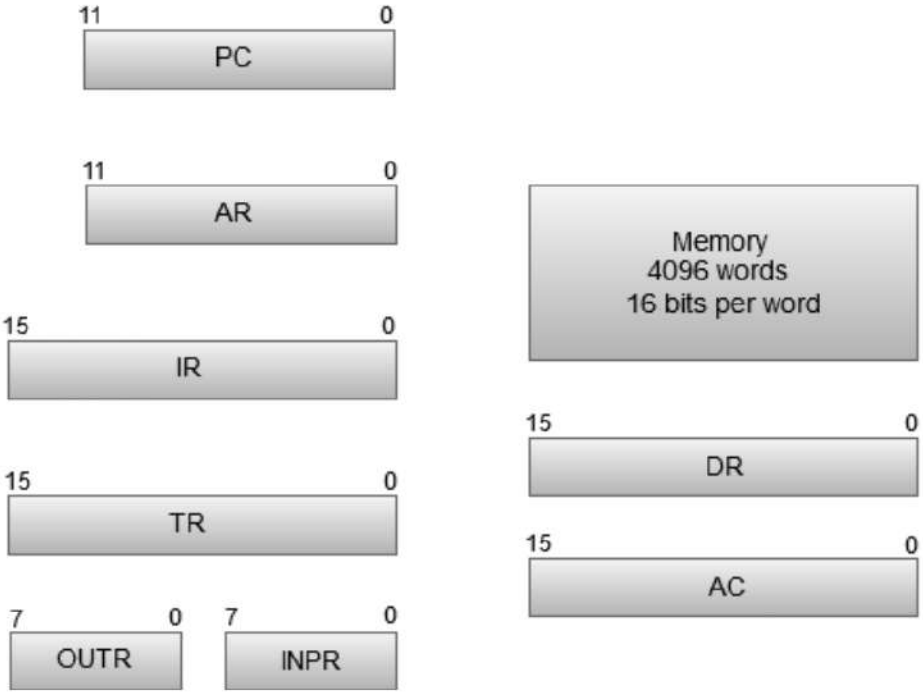
4.4. Computer registers

Registers are a type of computer memory built directly into the processor or CPU (Central Processing Unit) that is used to store and manipulate data during the execution of instructions. A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters).

Following is the list of some of the most common registers used in a basic computer:

Register	Symbol	Number of bits	Function
Data register	DR	16	Holds memory operand
Address register	AR	12	Holds address for the memory
Accumulator	AC	16	Processor register
Instruction registers	IR	16	Holds instruction code
Program counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input register	INPR	8	Carries input character
Output register	OUTR	8	Carries output character

Register and Memory Configuration of a basic computer:

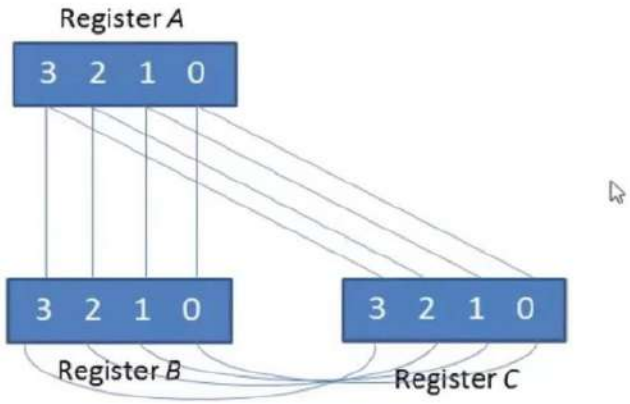


- The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- The Accumulator (AC) register is a general-purpose processing register.
- The instruction read from memory is placed in the Instruction register (IR).
- The Temporary Register (TR) is used for holding the temporary data during the processing.
- The Input Registers (IR) holds the input characters given by the user.
- The Output Registers (OR) holds the output after processing the input data.

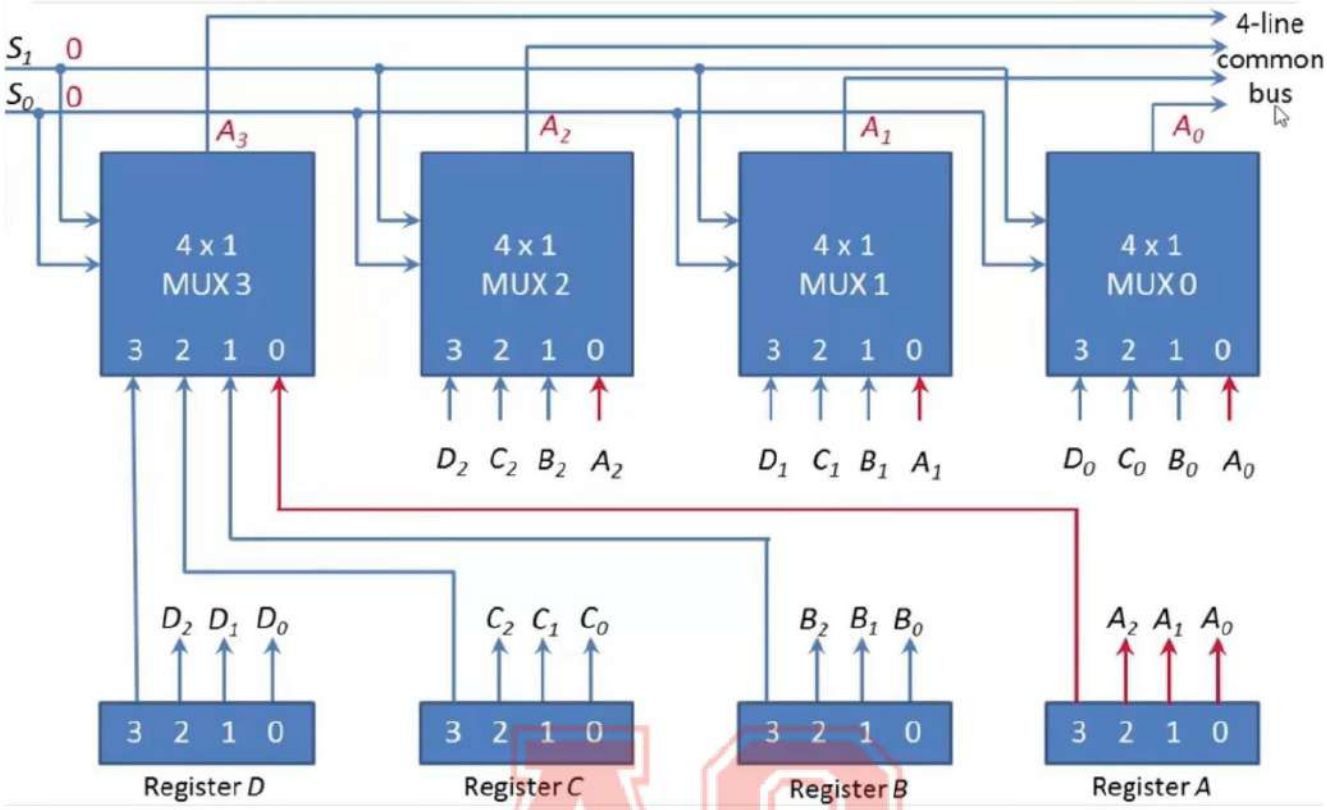
4.5. Common Bus system

Common bus system for 4 registers

- A typical digital computer has many registers, and paths must be provided to transfer information from one register to another.
- The number of wires will be excessive if separate lines are used between each register and all other registers in the system.



Common bus system for 4 registers



Common bus system for 4 registers

- Table shows the register that is selected by the bus for each of the four possible binary values of the selection lines.

S_1	S_0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

Common bus system for 4 registers

- In general, a bus system will multiplex *k* registers of *n* bits each to produce an *n*-line common bus.
- The number of multiplexers needed to construct the bus is equal to *n*, the number of bits in each register.
- The size of each multiplexer must be *k x 1* since it multiplexes *k* data lines.
- For example, a common bus for eight registers of 16 bits requires

  Multiplexers – 16 of (8 x 1)

  Select Lines - 3

Connections:
The outputs of all the registers except the OUTR (output register) are connected to the common bus. The output selected depends upon the binary value of variables S2, S1 and S0. The lines from common bus are connected to the inputs of the registers and memory. A register receives the information from the bus when its LD (load) input is activated while in case of memory the Write input must be enabled to receive the information. The contents of memory are placed onto the bus when its Read input is activated.

Various Registers:
4 registers DR, AC, IR and TR have 16 bits and 2 registers AR and PC have 12 bits. The INPR and OUTR have 8 bits each. The INPR receives character from input device and delivers it to the AC while the OUTR receives character from AC and transfers it to the output device. 5 registers have 3 control inputs LD (load), INR (increment) and CLR (clear). These types of registers are similar to a binary counter.

Abbreviation	Register name
OUTR	Output register
TR	Temporary register
IR	Instruction register
INPR	Input register

AC	Accumulator
DR	Data register
PC	Program counter
AR	Address register

4.6. Instruction sets:

The instruction set is part of a computer that pertains to programming, which is more or less machine language. The instruction set provides commands to the processor, to tell it what it needs to do. The instruction set consists of addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O.

Examples of instruction set :-

- ADD - Add two numbers together.
- COMPARE - Compare numbers.
- IN - Input information from a device, e.g., keyboard.
- JUMP - Jump to designated RAM address.
- JUMP IF - Conditional statement that jumps to a designated RAM address.
- LOAD - Load information from RAM to the CPU.
- OUT - Output information to device, e.g., monitor.
- STORE - Store information to RAM

4.7. Instruction types:

- Data transfer: registers, main memory, stack or I/O
- Data processing: arithmetic, logical
- Control: systems control, transfer of control

Data Transfer Instructions

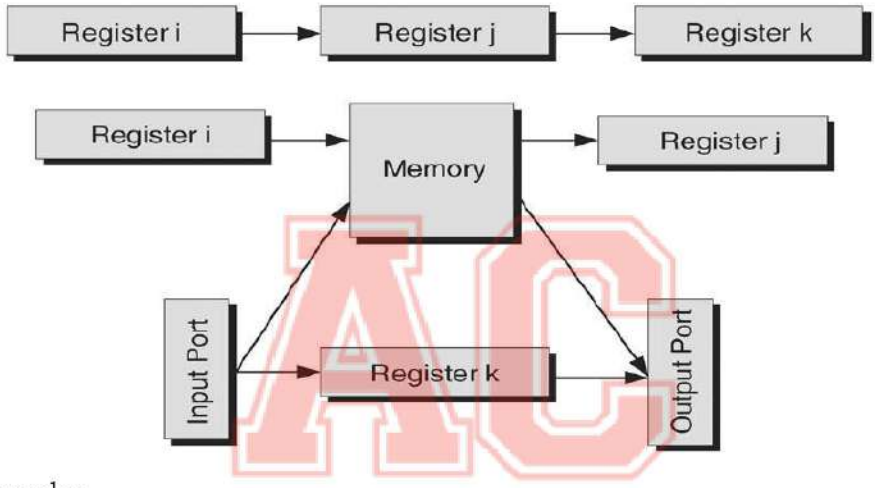
• Are responsible for moving data around inside the processor as well as brining in data or sending data out

• Examples: Store, load, exchange, move, set, push, pop

• Each Instruction should have:

• source and destination (memory, register, input/output port)

• amount of data



Instruction example:

LD destination, source	Load—source operand transferred to destination operand can be either register or memory location.
ST source, destination	Store—source operand transferred to destination operand source must be a register and the destination must be memory.
MOVE destination, source	Transfer from register to register or memory to memory.
XCH destination, source	Interchange the source and destination operands.
PUSH/POP	Operand pushed onto or popped off of the stack.
IN/OUT destination, source	Transfer data from or to an input/output port.

Arithmetic

- Add, Subtract, Multiply, Divide for signedinteger (+ floating point and packed decimal) – may involve data movement
- May include

- Absolute (i.e |a|)
- Increment (i.e a++)
- Decrement (i.e a--)
- Negate (i.e -a)

Logical

- Bitwise operations: AND, OR, NOT, XOR,CMP, SET
- Shifting and rotating functions, e.g.
  - logical right shift for unpacking: send 8-bitcharacterfrom 16-bit word
  - arithmetic right shift: division and truncationfor oddnumbers
  - arithmetic left shift: multiplication withoutoverflow
  -

Systems Control and Execution Flow

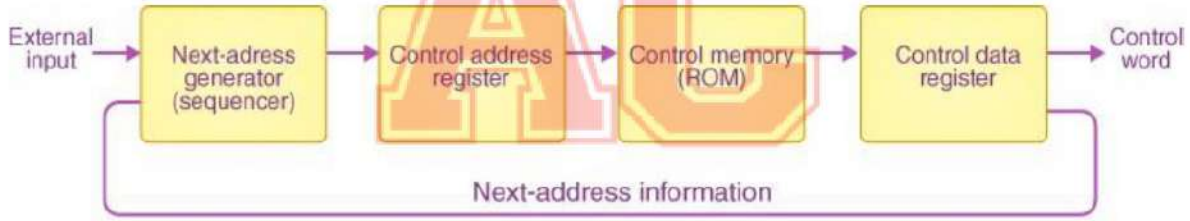
- The execution flow captures the order ofevaluation/execution ofeach instruction
  - Sequential
  - Branch
  - Loop
  - Procedure or Function call

Unit 5. Design of Microprogrammed Control Unit [Marks – 18]

5.1. Control Word, Microprogram, Control Memory

Micro-programmed Control Unit

- It is implemented using programming approach
- A sequence of microoperation is carried out by executing a program consisting of micro instruction.
- Micro program consisting of micro instruction is stored in the control memory (ROM) of the control unit.
- Execution of a micro instruction is responsible for generation for a set of control signal.
- The address of micro instruction that is to be executed is stored in CAR (Control Address Register).
- Micro instruction corresponding to the address stored in CAR is fetched from control memory and stored in control data register.
- This micro instruction contains control word to be execute one or more micro-operation.
- After the execution of all micro-operations of micro instruction, the address of next micro instruction is located.



Microprogrammed control unit of a Basic Computer

Control Word:

The string of control variables which control the sequence of microoperations is called a control word. The microoperations specified in a control word is called a microinstruction. Each microinstruction specifies one or more microoperations that is performed.

Control word is defined as a word whose individual bits represent the various control signal. The control words related to an instruction that is stored in microprogram memory. It is a set of micro-instructions in a micro-routine.

- In control words, microoperations are specified, they are known as microinstructions.
- Each microinstruction specifies one or more microoperations that are performed.
- The Control Word consists of bits, and each bit corresponds to a function or commands such as Pause, Stop, Enable, Start, Stop, Move, etc.

Microprogram:

Microprogramming, process of writing microcode for a microprocessor. Microcode is low-level code that defines how a microprocessor should function when it executes machine-language instructions. Typically, one machine-language instruction translates into several microcode instructions. On some computers, the microcode is stored in ROM (read-only memory) and cannot be modified; on some larger computers, it is stored in EPROM (erasable programmable read-only memory) and therefore can be replaced with newer versions.

Control Memory:

The control memory consists of microprograms that are fixed and cannot be modified frequently. They contain microinstructions that specify the internal control signals required to execute register micro-operations.

The machine instructions generate a chain of microinstructions in the control memory. Their function is to generate micro-operations that can fetch instructions from the main memory, compute the effective address, execute the operation, and return control to fetch phase and continue the cycle.

5.2. Control Address Register, Sequencer

the Control Address Register (CAR) and Sequencer are integral components of the control unit in a CPU. The CAR stores the address of the next microinstruction to be fetched, while the sequencer generates the sequence of addresses, ensuring the correct flow of control signals for the execution of machine instructions. Together, they play a crucial role in microprogramming and control unit operation.

Control Address Register (CAR):

- The Control Address Register is a special-purpose register within the control unit of the CPU.
- Its primary function is to store the address of the next microinstruction to be fetched from the control memory.
- During the execution of a machine instruction, the control unit fetches microinstructions sequentially from control memory, and the CAR holds the address of the next microinstruction.

Sequencer:

- The sequencer is a component of the control unit responsible for generating the sequence of addresses that the Control Address Register (CAR) uses to fetch microinstructions.
- It controls the flow of microinstructions during the execution of machine instructions, ensuring that the correct sequence of control signals is applied to the CPU components.

- The sequencer may be implemented using counter circuits, state machines, or other logic circuits that determine the next address based on the current state and the microinstructions being executed.

5.3. Address Sequencing

Address sequencing is a fundamental aspect of microprogramming and control unit operation. The sequencer generates a sequence of addresses, and these addresses are used to fetch microinstructions from control memory, allowing for the orderly execution of machine instructions and the coordination of CPU operations.

Microinstruction Execution Cycle:

- Microinstructions are executed in a cycle, and each cycle involves fetching a microinstruction from control memory, decoding it, and executing the specified control operations.
- Address sequencing ensures the continuous flow of microinstructions through this cycle.

Advantages:

- Address sequencing provides flexibility in controlling the flow of microinstructions, allowing for the implementation of complex instruction sets and the coordination of various CPU components.
- It allows for the orderly execution of machine instructions, ensuring that each step in the execution process is properly coordinated.

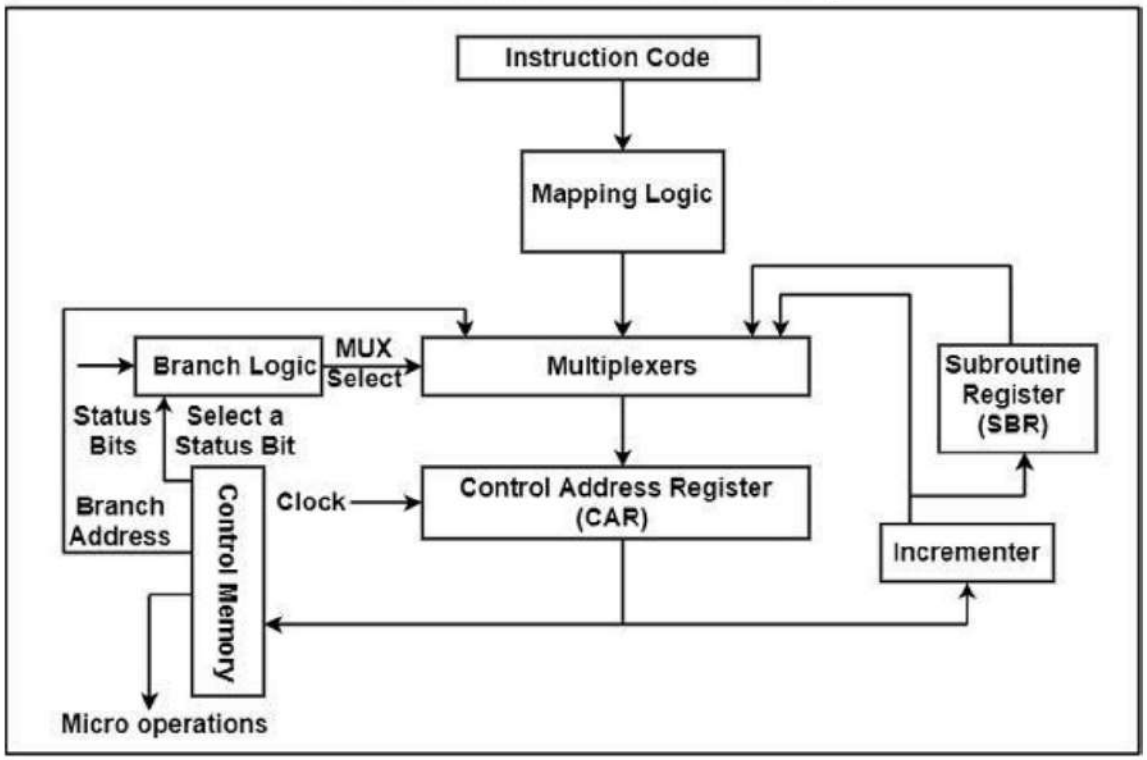
Implementation Techniques:

- Address sequencing can be implemented using various techniques, including counters, state machines, or combinational logic circuits.
- Counters can be used to increment the address in a predefined sequence, while state machines can transition between states based on the current state and external inputs.

Branching and Control Flow:

- Address sequencing also handles branching and control flow instructions. When a branch or jump instruction is encountered, the sequencer adjusts the next address accordingly.

Selection of Address for Control Memory



The diagram shows the block diagram of a control memory and its associated hardware to support in choosing the next microinstruction. The microinstruction present in the control memory has a set of bits that facilitate to start off the micro-operations in registers.

There are four different directions are showed in the figure from where the control address register recovers its address. The CAR is incremented by the incrementor and selects the next instruction. In multiple fields of microinstruction, the branching address can be determined to result in branching.

It can specify the condition of the status bits of microinstruction, conditional branching can be applied. A mapping logic circuit can share an external address. A special register can save the return address so that when the microprogram needs to return from the subroutine, it can need the value from the unique register.

5.4. Conditional Branch

A conditional branch is a type of branch instruction that alters the flow of control based on a certain condition. When a conditional branch instruction is encountered during the execution of a program or microprogram, the decision to branch or not is determined by the evaluation of a specified condition.

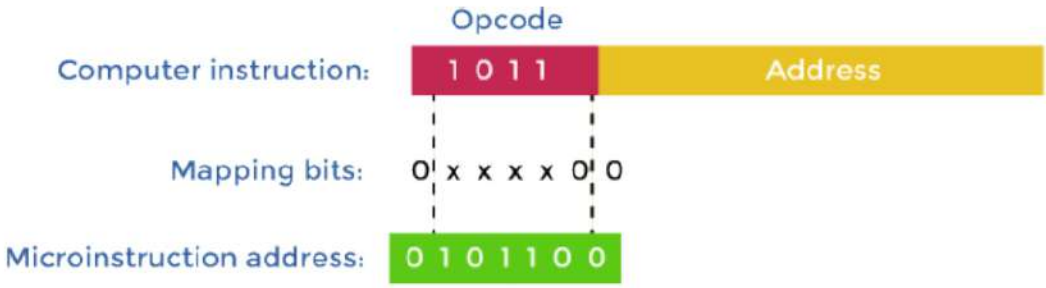
1. Condition Evaluation:

- The conditional branch instruction includes a condition that is evaluated. This condition can be based on the status of certain flags, the contents of registers, or other relevant data.

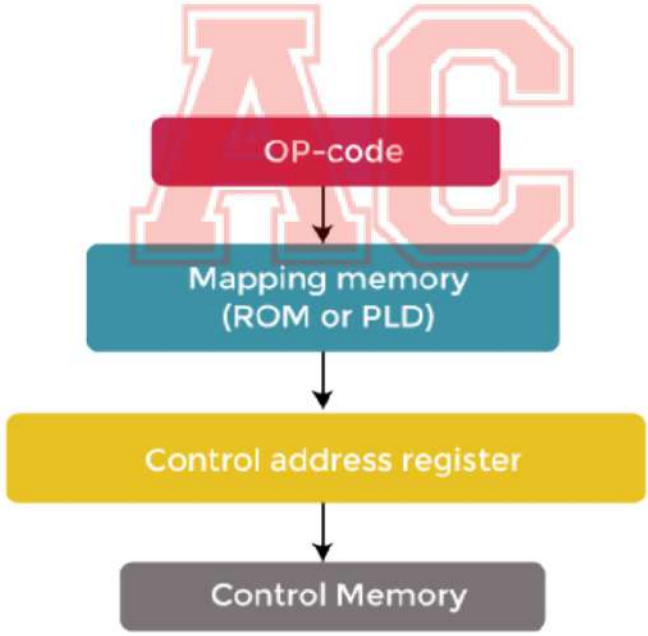
2. **Branch Decision:**
    - If the condition specified in the instruction is true or satisfied, the program or microprogram branches to a target address or executes a specified set of instructions.
    - If the condition is false or not satisfied, the program or microprogram continues with the next sequential instruction.
3. **Types of Conditions:**
    - Conditions can take various forms, such as checking for equality, inequality, greater than, less than, or a combination of these conditions.
    - Commonly used conditions include zero/non-zero, positive/negative, overflow, carry, and specific bit states.
4. **Branch Target:**
    - The branch target is the address or location in memory to which control is transferred when the condition is true.
    - In microprogramming, the branch target is often represented by the next address to be fetched from control memory.
5. **Effect on Control Flow:**
    - Conditional branches play a crucial role in altering the control flow of a program or microprogram. They enable the execution of different sets of instructions based on specific conditions.

5.5. Mapping of Instructions

Each instruction has its own microprogram routine stored in a given location of the control memory. The transformation from the instruction code bits to an address in the control memory where the routine is located is called as Mapping. After the execution of the instruction control must return to the fetch routine.



Mapping from instruction code to microinstruction address

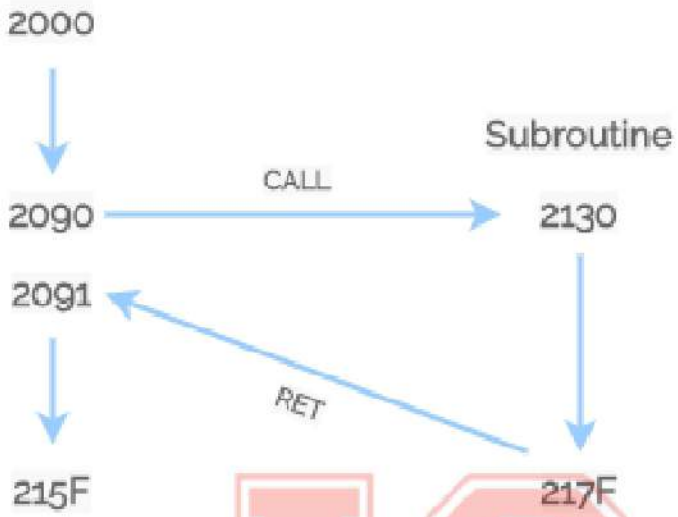


Mapping Function Implemented by ROM and PLD

5.6. Subroutines, Microinstruction Format, Symbolic Microinstructions

Subroutine:

A subroutine is a set of instructions that will be used repeatedly in different locations of the program. Instead of repeating the same instructions several times, they can be combined into a subroutine that is called from various locations.



The microprocessor has two instructions to implement subroutines.

The CALL

Instruction is used to redirect program execution to a subroutine.

The RET

Instruction is used to return the execution to the calling routine.

Stack: The stack can be described as a reserved area of the memory in the R/W memory where we can store temporary information. It is a shared resource as it can be shared by the microprocessor and the programmer. Programmers use the stack to store data and the microprocessors use the stack to execute subroutines.

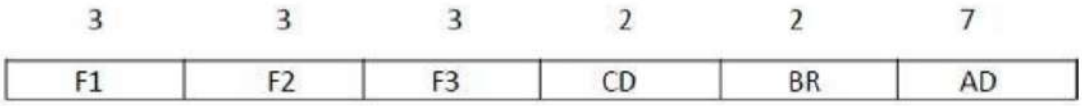
A 16-bit register known as the **Stack Pointer**. The function of the stack pointer is to hold the starting address of the stack. This address can be decided by the programmer.

The stack operates on the **Last In, First Out** (LIFO) principle. The location of the most recent data on the stack is known as the **TOP** of the stack. The stack pointer always points to the top of the stack. Contents can be stored in the stack using the PUSH Instruction and can restore the contents by using the instruction POP.

Microinstruction Format:

A microinstruction format includes 20 bits in total. They are divided into four elements as displayed in the figure.

Microinstruction Code Format



F1, F2, F3 are the micro-operation fields. They determine micro-operations for the computer.

CD is the condition for branching. They choose the status bit conditions.

BR is the branch field. It determines the type of branch.

AD is the address field. It includes the address field whose length is 7 bits.

The micro-operations are divided into three fields of three bits each. These three bits can define seven different micro-operations. In total there are 21 operations as displayed in the table.

Symbols with their Binary Code for Microinstruction Fields

Microinstruction Fields

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

Pritee Parwekar

As shown in the table, each microinstruction can have only three micro-operations, one from each field. If it uses less than three, it will result in more than one operation using the no operation binary code.

Condition Field

A condition field includes 2 bits. They are encoded to define four status bit conditions. As stated in the table, the first condition is always a 1, with CD = 0. The symbol that can indicate this condition is 'U'. The table displays the multiple condition fields and their summary in an easy manner.

Condition Field Symbols and Descriptions

	Condition	Symbol	Comments
00	Always = 1	U	Unconditional Branch
01	DR (15)	I	Indirect address bit
10	AC (15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

As shown in the table, when condition 00 is connected with **BR** (branch) field, it results in an unconditional branch operation. Then the execution is read from memory the indirect bit I is accessible from bit 15 of **DR**. The status of the next bit is supported by the AC sign bit. If all the bits in **AC** are 1, then it is indicated as Z (its binary variable whose value is 1). The symbols **U, I, S**, and **Z** can indicate status bits while writing microprograms.

Branch Field

The BR (branch) field includes 2 bits. It can be used by connecting with the AD (address) field. The reason for connecting with the AD field is to select the address for the next microinstruction. The table illustrates the various branch fields and their functions.

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR +1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR +1 if condition = 1 CAR ← CAR +1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0, 1, 6) ← 0

Table 3-1 : Symbols and Binary code for Microinstruction Fields

As shown in the table, when **BR = 00**, a **JMP** operation is implemented and when **BR = 01**, a subroutine is called. The only difference between the two instructions is that when the microinstruction is saved, the return address is saved in the **Subroutine Register (SBR)**.

These two operations are dependent on the **CD** field values. When the status bit condition of the CD field is defined as **1**, the address that is next in order is transferred to **CAR.** Else, it gets incremented. If the instruction needs to return from the subroutine, its **BR** field is determined as **10**.
This results in the transfer of the return address from **SBR** to **CAR**. The opcode bits of instruction can be mapped with an address for **CAR** if the **BR** field is **11**. They are present in **DR (11 - 14)** after instruction is read from memory. The last two conditions in the BR fields are not dependent on the **CD** and **AD** field values.

Symbolic Microinstructions:
The microinstructions can be determined by symbols. It is interpreted to its binary format with an assembler. The symbols should be represented for each field in the microinstruction. The users should be enabled to represent their symbolic addresses. Each line in an assembly language represents symbolic instruction. These instructions are divided into five fields such as label, micro-operations, CD, BR, and AD.
The fields that specify the following information are as follows –

- The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
- The micro-operations field consists of one, two, or three symbols, separated by commas. But each F field includes only a single symbol.
- The CD field has one of the letters U, I, S, or Z. [U: Unconditional, I: Indirect address bit, S: sign of AC, Z: Zero value in AC]
- The BR field contains one of the four symbols defined.
- The AD field specifies a value for the address field of the microinstruction in one of three possible ways –
    - With a symbolic address, which must also appear as a label.
    - With the symbol NEXT to designate the next address in sequence.
    - When the BR field includes a RET or MAP symbol, the AD field is left null and is transformed to seven zeros by the assembler.

Fetch Routine

The control unit includes 128 words, each including 20 bits. The value of each bit should be specified to microprogram the control memory. Between the 128 words, the first 64 are composed for the routines of 16 instructions. The remaining 64 can be used for different goals. The best opening location for the fetch routine to start is the 64th address.

- Read instruction from memory
- Decode instruction and update PC

The microinstructions necessary for fetch routine are –

$AR \leftarrow PC$

$DR \leftarrow M[AR], PC \leftarrow PC + 1$

$AR \leftarrow DR(0-10), CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

The address of the instruction is transferred from PC to AR and the instruction is then read from memory into DR. Since no instruction register is available, the instruction code remains in DR. The address part is transferred to AR and then control is transferred to one of 16 routines by mapping the operation code part of the instruction from DR into CAR.

Microinstructions that are situated in addresses 64, 65, and 66 are important for the fetch routine. There are various symbolic language is as follows –

	ORG 64
FETCH:	PCTAR U JMP NEXT
	READ, INCPC U JMP NEXT
	DRTAR U MAP

The table shows the results of binary translation for the assembly language.

Binary Translation for Assembly Language

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	101	000	00	11	0000000

Each microinstruction executes the internal register transfer operation displayed by the register transfer representation. The representation in symbols is important while writing microprograms in an assembly language format. The actual internal content which is saved in the control memory is in binary representation.

5.7. Central Processing Unit

5.7.1 Introduction:

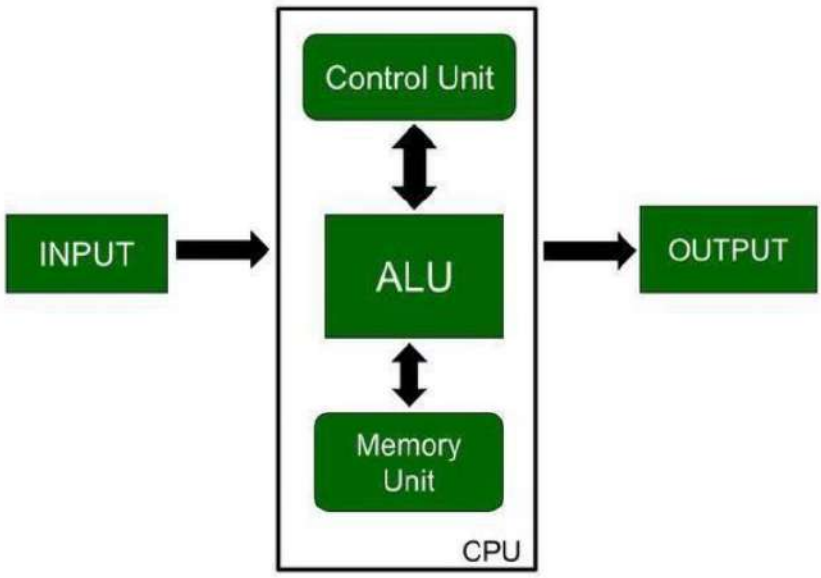
The full form of CPU is Central Processing Unit. It is a brain of the computer. All types of data processing operations and all the important functions of a computer are performed by the CPU. It helps input and output devices to communicate with each other and perform their respective operations. It also stores data which is input, intermediate results in between processing, and instructions.

Different Parts of CPU

Now, the CPU consists of 3 major units, which are:

1. Memory or Storage Unit
2. Control Unit

3. ALU(Arithmetic Logic Unit)



Memory or Storage Unit

- Data and instructions are stored in memory units which are required for processing.
- It also stores the intermediate results of any calculation or task when they are in process.
- The final results of processing are stored in the memory units before these results are released to an output device for giving the output to the user.
- All sorts of inputs and outputs are transmitted through the memory unit.

Control Unit

- Controlling of data and transfer of data and instructions is done by the control unit among other parts of the computer.
- The control unit is responsible for managing all the units of the computer.
- The main task of the control unit is to obtain the instructions or data which is input from the memory unit, interprets them, and then directs the operation of the computer according to that.
- The control unit is responsible for communication with Input and output devices for the transfer of data or results from memory.
- The control unit is not responsible for the processing of data or storing data.

ALU (Arithmetic Logic Unit)

ALU (Arithmetic Logic Unit) is responsible for performing arithmetic and logical functions or operations. It consists of two subsections, which are:

- Arithmetic Section
- Logic Section

What Does a CPU Do?

The main function of a computer processor is to execute instruction and produce an output. CPU work are Fetch, Decode and Execute are the fundamental functions of the computer.

- **Fetch:** the first CPU gets the instruction. That means binary numbers that are passed from RAM to CPU.
- **Decode:** When the instruction is entered into the CPU, it needs to decode the instructions. with the help of ALU (Arithmetic Logic Unit) the process of decode begins.
- **Execute:** After decode step the instructions are ready to execute
- **Store:** After execute step the instructions are ready to store in the memory.

5.7.2. General Register Organization

When we are using multiple general-purpose registers, instead of a single accumulator register, in the CPU Organization then this type of organization is known as General register-based CPU Organization. In this type of organization, the computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word. If many CPU registers are available for heavily used variables and intermediate results, we can avoid memory references much of the time, thus vastly increasing program execution speed, and reducing program size.

For example:

MULT R1, R2, R3

This is an instruction of an arithmetic multiplication written in assembly language. It uses three address fields R1, R2, and R3. The meaning of this instruction is:

R1 <-- R2 * R3

This instruction also can be written using only two address fields as:

MULT R1, R2

In this instruction, the destination register is the same as one of the source registers. This means the operation

R1 <-- R1 * R2

Features of a General Register based CPU organization:

Registers: In this organization, the CPU contains a set of registers, which are small, high-speed memory locations used to store data that is being processed by the CPU. The general-

purpose registers can be used to store any type of data, including integers, floating-point numbers, addresses, and control information.

Operand access: The CPU accesses operands directly from the registers, rather than having to load them from memory each time they are needed. This can significantly improve performance, as register access is much faster than memory access.

Data processing: The CPU can perform arithmetic and logical operations directly on the data stored in the registers. This eliminates the need to transfer data between the registers and memory, which can further improve performance.

Instruction format: The instruction format used in a General Register based CPU typically includes fields for specifying the operands and operation to be performed. The operands are identified by register numbers, rather than memory addresses.

Context switching: Context switching in a General Register based CPU involves saving the contents of the registers to memory, and then restoring them when the process resumes. This is necessary to allow multiple processes to share the CPU.

The advantages of General register-based CPU organization –

- The efficiency of the CPU increases as large number of registers are used in this organization.
- Less memory space is used to store the program since the instructions are written in a compact way.

The disadvantages of General register-based CPU organization –

- Care should be taken to avoid unnecessary usage of registers. Thus, compilers need to be more intelligent in this aspect.
- Since a large number of registers are used, thus extra cost is required in this organization.

General register CPU organization of two types:

1. Register-memory reference architecture (CPU with less register) – In this organization Source 1 is always required in the register, source 2 can be present either in the register or in memory. Here two address instruction formats are compatible instruction formats.
2. Register-register reference architecture (CPU with more register) – In this organization, ALU operations are performed only on registered data. So operands are required in the register. After manipulation, the result is also placed in a register. Here three address instruction formats are the compatible instruction format.

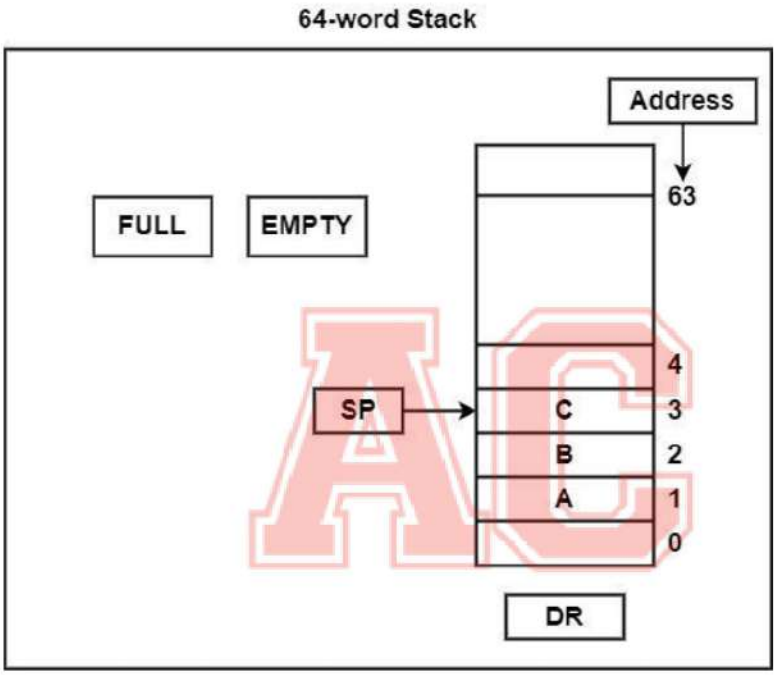
5.7.3. Stack Organization

The computers which use Stack-based CPU Organization are based on a data structure called a **stack**. The stack is a list of data words. It uses the **Last In First Out (LIFO)** access method which is the most popular access method in most of the CPU. A register is used to store the address of the topmost element of the stack which is known as **Stack pointer (SP)**.

In this organization, ALU operations are performed on stack data. It means both the operands are always required on the stack. After manipulation, the result is placed in the stack.

The stack can be arranged as a set of memory words or registers. Consider a 64-word register stack arranged as displayed in the figure. The stack pointer register includes a binary number, which is the address of the element present at the top of the stack. The three-element A, B, and C are located in the stack.

The element C is at the top of the stack and the stack pointer holds the address of C that is 3. The top element is popped from the stack through reading memory word at address 3 and decrementing the stack pointer by 1. Then, B is at the top of the stack and the SP holds the address of B that is 2. It can insert a new word, the stack is pushed by incrementing the stack pointer by 1 and inserting a word in that incremented location.



The stack pointer includes 6 bits, because $2^6 = 64$, and the SP cannot exceed 63 (111111 in binary). After all, if 63 is incremented by 1, therefore the result is 0(111111 + 1 = 1000000). SP holds only the six least significant bits. If 000000 is decremented by 1 thus the result is 111111.

Therefore, when the stack is full, the one-bit register 'FULL' is set to 1. If the stack is null, then the one-bit register 'EMTY' is set to 1. The data register DR holds the binary information which is composed into or readout of the stack.

First, the SP is set to 0, EMTY is set to 1, and FULL is set to 0. Now, as the stack is not full (FULL = 0), a new element is inserted using the push operation.

The push operation is executed as follows –

SP←SP + 1	It can increment stack pointer
M[SP] ← DR	It can write element on top of the stack
If (SP = 0) then (FULL ← 1)	Check if stack is full
EMTY ← 0	Mark the stack not empty

The stack pointer is incremented by 1 and the address of the next higher word is saved in the SP. The word from DR is inserted into the stack using the memory write operation. The first element is saved at address 1 and the final element is saved at address 0. If the stack pointer is at 0, then the stack is full and 'FULL' is set to 1. This is the condition when the SP was in location 63 and after incrementing SP, the final element is saved at address 0. During an element is saved at address 0, there are no more empty registers in the stack. The stack is full and the 'EMTY' is set to 0.

A new element is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation includes the following sequence of micro-operations –

DR←M[SP]	It can read an element from the top of the stack
SP ← SP – 1	It can decrement the stack pointer
If (SP = 0) then (EMTY ← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full

The top element from the stack is read and transfer to DR and thus the stack pointer is decremented. If the stack pointer reaches 0, then the stack is empty and 'EMTY' is set to 1. This is the condition when the element in location 1 is read out and the SP is decremented by 1.

5.8. Instruction Formats

In computer organization, instruction formats refer to the way instructions are encoded and represented in machine language. There are several types of instruction formats, including zero, one, two, and three-address instructions.

What are the Different Types of Field in Instruction?

A computer performs a task based on the instruction provided. Instruction in computers comprises groups called fields. These fields contain different information for computers everything is in 0 and 1 so each field has different significance based on which a CPU decides what to perform. The most common fields are:

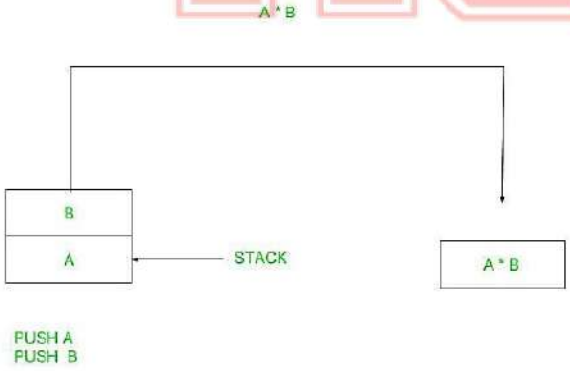
- The operation field specifies the operation to be performed like addition.
- Address field which contains the location of the operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

Instruction is of variable length depending upon the number of addresses it contains. Generally, CPU organization is of three types based on the number of address fields:

In the first organization, the operation is done involving a special register called the accumulator. In the second multiple registers are used for the computation purpose. In the third organization the work on stack basis operation due to which it does not contain any address field. Only a single organization doesn't need to be applied, a blend of various organizations is mostly what we see generally.

Zero Address Instructions

These instructions do not specify any operands or addresses. Instead, they operate on data stored in registers or memory locations implicitly defined by the instruction. For example, a zero-address instruction might simply add the contents of two registers together without specifying the register names.



A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

PUSH	A	TOP = A
PUSH	B	TOP = B

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

One Address Instructions

These instructions specify one operand or address, which typically refers to a memory location or register. The instruction operates on the contents of that operand, and the result may be stored in the same or a different location. For example, a one-address instruction might load the contents of a memory location into a register.

This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

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

Two Address Instructions

These instructions specify two operands or addresses, which may be memory locations or registers. The instruction operates on the contents of both operands, and the result may be stored in the same or a different location. For example, a two-address instruction might add the contents of two registers together and store the result in one of the registers.

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent the address.

MOV	R1, A	R1 = M[A]
ADD	R1, B	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

Three Address Instructions

These instructions specify three operands or addresses, which may be memory locations or registers. The instruction operates on the contents of all three operands, and the result may be stored in the same or a different location. For example, a three-address instruction might multiply the contents of two registers together and add the contents of a third register, storing the result in a fourth register.

This has three address fields to specify a register or a memory location. Programs created are much short in size but number of bits per instruction increases. These instructions make the creation of the program much easier but it does not mean that program will run much faster because now instructions only contain more information but each micro-operation (changing the content of the register, loading address in the address bus etc.) will be performed in one cycle only.

opcode	Destination address	Source address	Source address	mode

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

5.9. Addressing Modes

To perform any operation, we have to give the corresponding instructions to the microprocessor.

In each instruction, we have to specify the following three things:

- Operation to be performed.
- Address of source of data.
- Address of destination of result.

The method by which the address of source of data or the address of destination of result is given in the instruction is called *Addressing Mode*. The term addressing mode refers to the way in which the operand of the instruction is specified.

Types of Addressing Modes

Intel 8085 uses the following addressing modes:-

1. Direct Addressing Mode
2. Register Direct Addressing Mode
3. Register Indirect Addressing Mode
4. Immediate Addressing Mode
5. Implied Addressing Mode

1. Direct Addressing Mode

Instructions using this mode specify the effective address as part of instruction. Instructions using this mode may contain 2 or 3 bytes, with first byte as the Op-code followed by 1 or 2 bytes or address of data. In this mode, the address of the operand is given in the instruction itself.

LDA 2500 H ⟶ Load the contents of memory location 2500 H in accumulator.

LDA is the operation.2500 H is the address of source. Accumulator is the destination.

2. Register Direct Addressing Mode

In this mode, the operand is in general purpose register i.e. data is provided through registers.

MOV A, B ⟶Move the contents of register B to A.

MOV is the operation. B is the source of data. A is the destination.

3. Register Indirect Addressing Mode:

In this mode, the address part of instruction specifies the memory location whose content is the address of the operand. In 8085 µp, wherever the instruction uses the HL pointer the address is called indirect addressing.

MOV A, M ⟶Move data from memory location specified by H-L pair to accumulator.

MOV is the operation. M is the memory location specified by H-L register pair. A is the destination.

4. Immediate Addressing Mode

In this mode, the operand is specified within the instruction itself. This mode of instructions uses first byte as the Op-code and following 1 or 2 byte data itself.

MVI A, 05 H ⟶ Move 05 H in accumulator.

MVI is the operation.05 H is the immediate data (source).A is the destination.

LXI H, 7A21 H ⟶Loads register H with 7A H and register L with 21 H

LXI is the operation. 7A21 H is the immediate data (Source). H & L registers are the destination.

So in both cases, the actual data is part of the instruction, and hence called immediate addressing.

5. Implied Addressing Mode

If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction. So, instructions of such mode don"t have operands.

CMA ⟶ Complement accumulator.

CMA is the operation. A is the source. A is the destination.

5.10. RISC vs CISC.

RISC	CISC
Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and microprogrammed control unit
Transistors are used for more registers	Transistors are used for storing complex Instructions

Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small
An instruction executed in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word.	Instructions are larger than the size of one word
Simple and limited addressing modes.	Complex and more addressing modes.
RISC is Reduced Instruction Cycle.	CISC is Complex Instruction Cycle.
The number of instructions are less as compared to CISC.	The number of instructions are more as compared to RISC.
It consumes the low power.	It consumes more/high power.
RISC is highly pipelined.	CISC is less pipelined.
RISC required more RAM.	CISC required less RAM.
Here, Addressing modes are less.	Here, Addressing modes are more.

5.11. Pipeline and Vector Processing

Pipeline processing is an implementation technique where arithmetic sub operations or the phases of a computer instruction cycle overlap in execution.

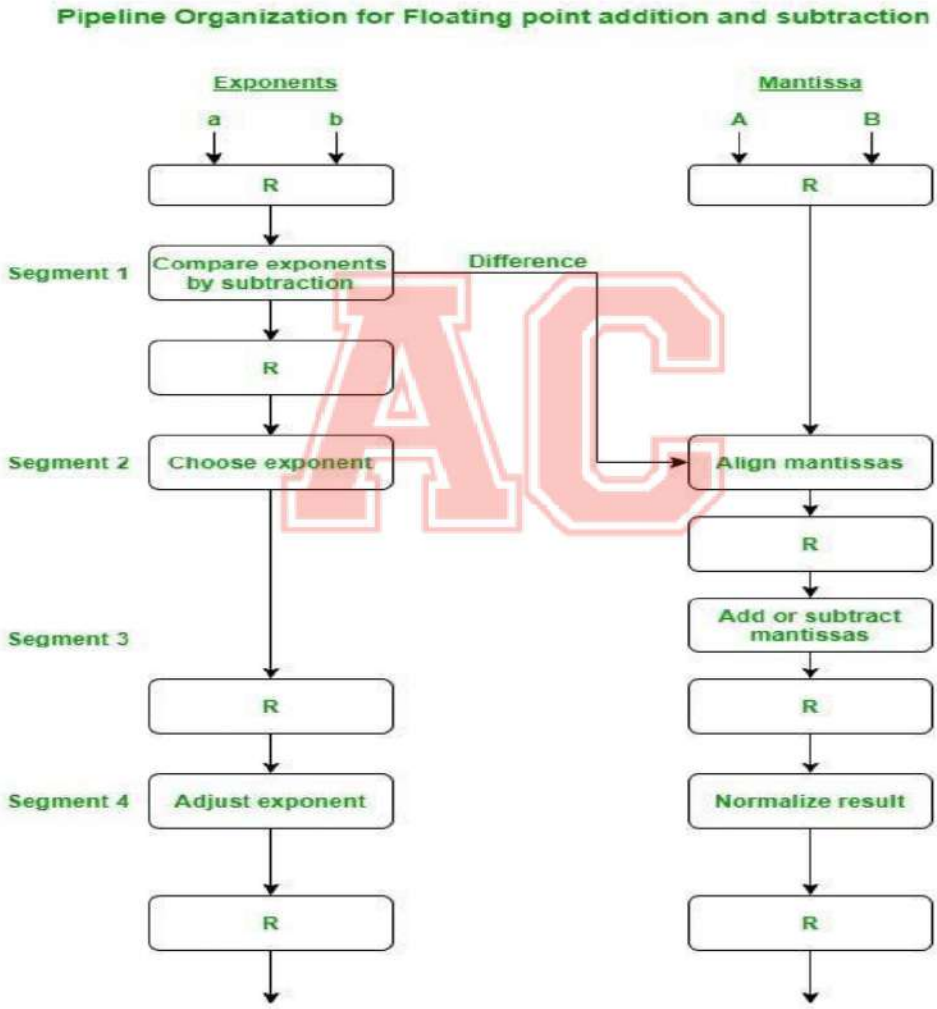
Vector processing ▪ Deals with computations involving large vectors and matrices.

Array processing ▪ Perform computations on large arrays of data.

5.11.1. Arithmetic and Instruction pipeline

1. Arithmetic Pipeline:

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.



Floating point addition using arithmetic pipeline:

The following sub operations are performed in this case:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalise the result

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:
Let us consider two numbers,

X=0.3214*10^3 and Y=0.4500*10^2

Explanation:
First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 times to the right to give

Y=0.0450*10^3

Finally, the two numbers are added to produce

Z=0.3664*10^3

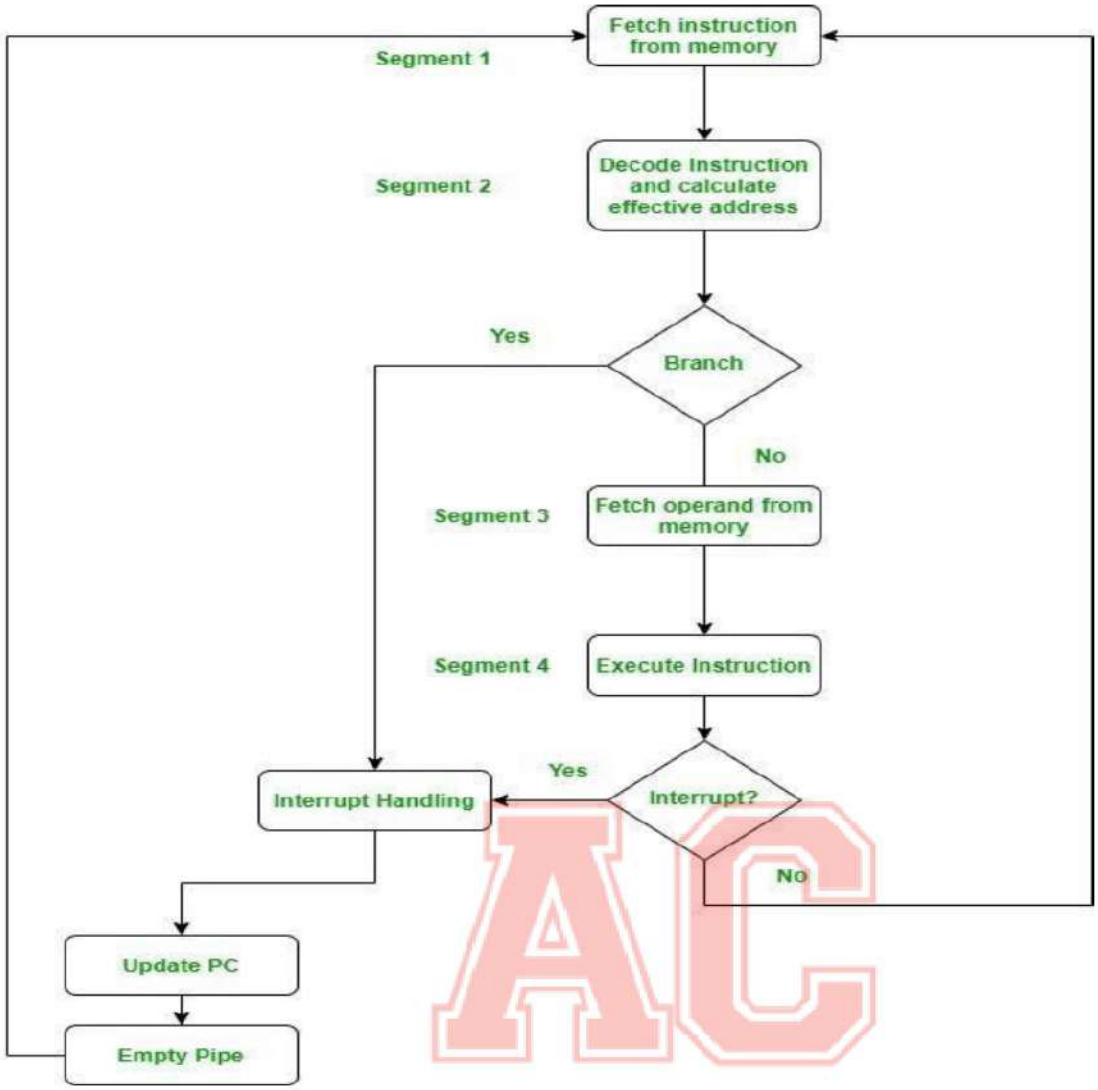
As the result is already normalized the result remains the same.

2. Instruction Pipeline :
In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

In the most general case computer needs to process each instruction in following sequence of steps:

1. Fetch the instruction from memory (FI)
2. Decode the instruction (DA)
3. Calculate the effective address
4. Fetch the operands from memory (FO)
5. Execute the instruction (EX)
6. Store the result in the proper place

The flowchart for instruction pipeline is shown below.



Let us see an example of instruction pipeline.

Example:

	Stage	1	2	3	4	5	6	7	8	9	10	11	12	13
	1	FI	DA	FO	EX									
Instruction	2		FI	DA	FO	EX								
Branch	3			FI	DA	FO	EX							
	4				FI	---	---	FI	DA	FO	EX			
	5								FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Here the instruction is fetched on first clock cycle in segment 1.
Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline.

In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may

point to some other instruction when it is decoded. Thus fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

Vector Processing

- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.
- Computers with vector processing capabilities are in demand in specialized applications. e.g.
  - Long-range weather forecasting
  - Petroleum explorations
  - Seismic data analysis
  - Medical diagnosis
  - Artificial intelligence and expert systems
  - Image processing
  - Mapping the human genome
- To achieve the required level of high performance it is necessary to utilize the *fastest and most reliable hardware* and apply innovative procedures from *vector and parallel processing techniques*.

5.11.2. Vector operations

- Many scientific problems require arithmetic operations on large arrays of numbers.
- A vector is an ordered set of a one-dimensional array of data items.
- A vector V of length n is represented as a row vector by V=[v1,v2,…,Vn].
- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

- This is implemented in machine language by the following sequence of operations.

```
      Initialize I=0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I)+B(I)
      Increment I = I + 1
      If I <= 100 go to 20
      Continue
```

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.

  C(1:100) = A(1:100) + B(1:100)
- A possible instruction format for a vector instruction is shown in Fig. 4-11.
  - This assumes that the vector operands reside in *memory*.
- It is also possible to design the processor with a large number of *registers* and store all operands in registers prior to the addition operation.

- The base address and length in the vector instruction specify a group of CPU registers.

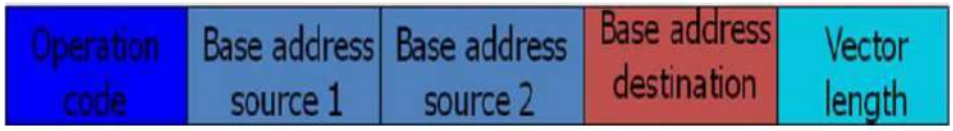


Fig 4-11: Instruction format for vector processor

5.11.3. Matrix Multiplication

- The multiplication of two n x n matrices consists of n^2 inner products or n^3 multiply-add operations.
  - Consider, for example, the multiplication of two 3 x 3 matrices A and B.
  - $c11 = a11b11 + a12b21 + a13b31$
  - This requires three multiplication and (after initializing c_{11} to 0) three additions.
- In general, the inner product consists of the sum of *k* product terms of the form $C = A_1B_1 + A_2B_2 + A_3B_3 + \ldots + A_kB_k$.
  - In a typical application k may be equal to 100 or even 1000.
- The inner product calculation on a pipeline vector processor is shown in Fig. 4- 12.

$$C + A1\,B1 + A5\,B5 + A9\,B9 + A13B13 \ldots.$$

$$+ A2\,B2 + A6\,B6 + A10\,B10 + A14\,B14 \ldots..$$

$$+ A3\,B3 + A7\,B7 + A11B11 + A15\,B15 \ldots...$$

$$+ A4\,B4 + A8\,B8 + A12\,B12 + A16\,B16 \ldots.$$

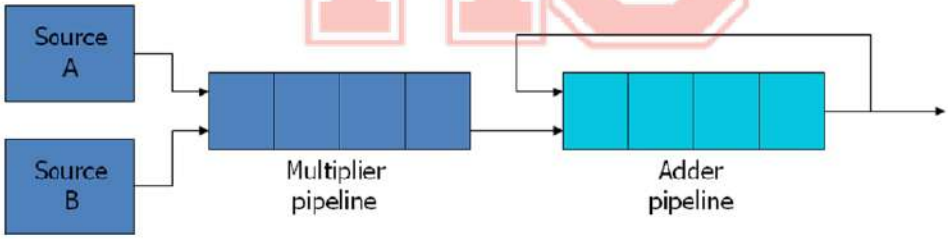


Fig 4-12: Pipeline for calculating an inner product

Memory Interleaving

- *Pipeline* and *vector processors* often require simultaneous access to memory from two or more sources.
  - An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments.
  - An arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time.
- Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses.
  - A memory module is a memory array together with its own address and data registers.

- Fig. 4-13 shows a memory unit with four modules.

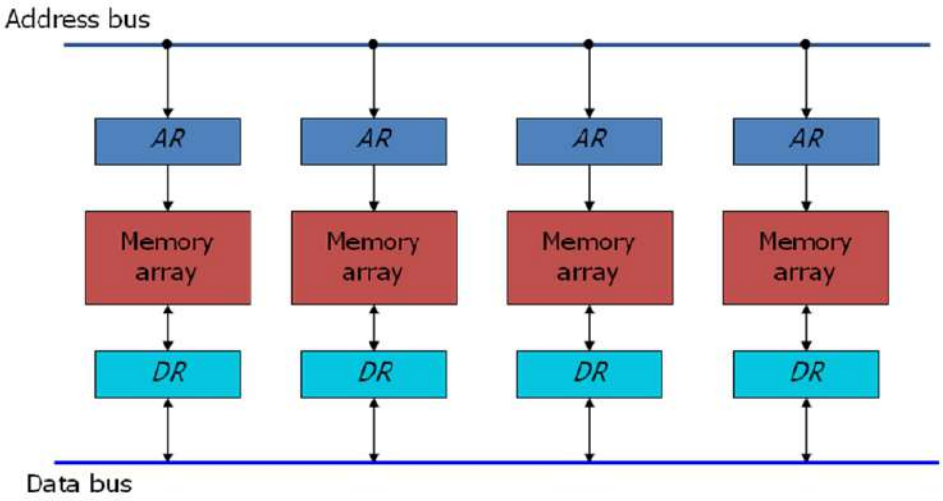


Fig 4-13: Multiple module memory organization

- The advantage of a modular memory is that it allows the use of a technique called *interleaving*.
- In an interleaved memory, different sets of addresses are assigned to different memory modules.
- By staggering the memory access, the effective memory cycle time can be *reduced by a factor close to the number of modules*.

Unit 6. Computer Arithmetic [Marks – 5]

6.1. Data Representation

Data Representation is about how computers interpret and function with different information types, including binary systems, bits and bytes, number systems (decimal, hexadecimal) and character encoding (ASCII, Unicode).

There are two major approaches to store real numbers (i.e., numbers with fractional component) in modern computing. These are (i) Fixed Point Notation and (ii) Floating Point Notation. In fixed point notation, there are a fixed number of digits after the decimal point, whereas floating point number allows for a varying number of digits after the decimal point.

6.1.1 Fixed-Point Representation –

This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field.

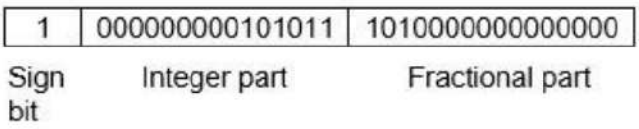
We can represent these numbers using:

- Signed representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 1's complement representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 2's complementation representation: range from $-(2^{(k-1)})$ to $(2^{(k-1)}-1)$, for k bits.

2's complementation representation is preferred in computer system because of unambiguous property and easier for arithmetic operations.

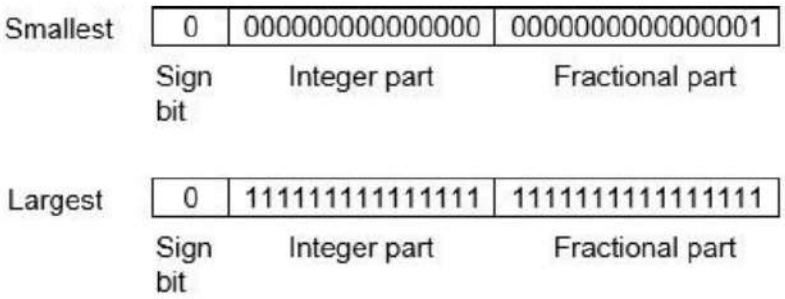
Example −Assume number is using 32-bit format which reserve 1 bit for the sign, 15 bits for the integer part and 16 bits for the fractional part.

Then, -43.625 is represented as following:



Where, 0 is used to represent + and 1 is used to represent -. 000000000101011 is 15 bit binary value for decimal 43 and 1010000000000000 is 16 bit binary value for fractional 0.625.

The advantage of using a **fixed-point representation** is performance and disadvantage is relatively limited range of values that they can represent. So, it is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy. A number whose representation exceeds 32 bits would have to be stored inexactly.



These are above smallest positive number and largest positive number which can be store in 32-bit representation as given above format. Therefore, the smallest positive number is $2^{-16} \approx 0.000015$ approximate and the largest positive number is $(2^{15}-1)+(1-2^{-16})=2^{15}(1-2^{-16})$ $=32768$, and gap between these numbers is 2^{-16}.

We can move the radix point either left or right with the help of only integer field is 1.

[There is fixed number of digits after decimal part]

- **Unsigned fixed-point number**
- **Signed fixed point number: 2's complement and sign & magnitude**

Unsigned fixed-point number:

It represents the fixed-point representation of unsigned binary number:

Example:

0110110 using four integer bits and three fractional bits.

0110.110

$= 0\times2^3+1\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2}+0\times2^{-3}$

$= 0 + 4 + 2 + 0 + 0.5 + 0.25 + 0$

$= (6.75)_{10}$

Signed fixed point number:

Represent (- 7.5) using 8-bit representation with 4-bit integer and 4 fractional.

$(7.5)_{10}$= (111.1)

= (0111.1000)

2's complement = (1000.1000)

6.1.2. Floating-Point Representation –

This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead, it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two parts: the first part represents a signed fixed-point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed-point mantissa may be fraction or an integer.

Binary point floats to the right of most significant 1 and an exponent is used.

$+- M \times B^E$

It has three parts:

- Mantissa
- Base
- Exponent

Number	Mantissa	Base	Exponent
9×10^8	9	10	8
4364.784	4364784	10	-3

IEEE Floating point Number Representation –

IEEE (Institute of Electrical and Electronics Engineers) has standardized Floating-Point Representation as following diagram.

N 0

Sign bit	Exponent	Mantissa

So, actual number is $(-1)^s(1+m)x2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number. The sign bit is 0 for positive number and 1 for negative number. Exponents are represented by or two's complement representation.

According to IEEE 754 standard, the floating-point number is represented in following ways:

- Half Precision (16 bit): 1 sign bit, 5 bit exponent, and 10-bit mantissa
- Single Precision (32 bit): 1 sign bit, 8 bit exponent, and 23-bit mantissa
- Double Precision (64 bit): 1 sign bit, 11 bit exponent, and 52-bit mantissa
- Quadruple Precision (128 bit): 1 sign bit, 15 bit exponent, and 112-bit mantissa

Special Value Representation –

There are some special values depended upon different values of the exponent and mantissa in the IEEE 754 standard.

- All the exponent bits 0 with all mantissa bits 0 represents 0. If sign bit is 0, then +0, else -0.
- All the exponent bits 1 with all mantissa bits 0 represents infinity. If sign bit is 0, then $+\infty$, else $-\infty$.
- All the exponent bits 0 and mantissa bits non-zero represents denormalized number.
- All the exponent bits 1 and mantissa bits non-zero represents error.

Example:

Represent (1259.125) in a single and double precision format.

Sol^{n}:

Step 1: Convert decimal to binary

$1259 = (10011100011)_2$

$0.125 = (001)$

$(1259.125)_{10} = (10011100011.001)_2$

Step 2: Normalize the number

For Single Precision: $(1.N)2^{e-127}$

For Double Precision: $(1.N)2^{e-1023}$

$(10011100011.001)_2$

$1.0011100011001 \times 2^{10}$

Step 3: Single Precision Format

$(1.N)2^{e-127} = 1.0011100011001 \times 2^{10}$

$E - 127 = 10$

$E = 137$

$E = (10001001)_2$

31	30 ... 23	22 ... 0
0	10001001	0011100011001
1 bit	8 bits	23 bits

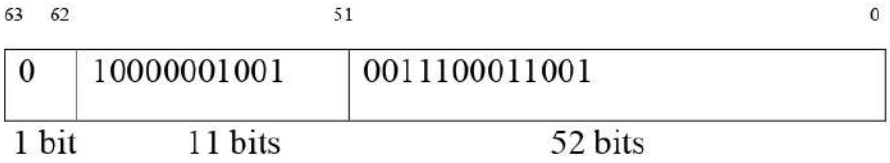
Step 4: Double Precision Format

$(1.N)2^{e-1023}=1.0011100011001\times2^{10}$

$E - 1023 = 10$

$E = 1033$

$E = (10000001001)_2$



6.2. Addition and Subtraction with Signed Magnitude Data

A signed-magnitude method is used by computers to implement floating-point operations. Signed-2's complement method is used by most computers for arithmetic operations executed on integers. In this approach, the leftmost bit in the number is used for signifying the sign; 0 indicates a positive integer, and 1 indicates a negative integer. The remaining bits in the number supported the magnitude of the number.

Example: -2410 is defined as − 10011000

In this example, the leftmost bit 1 defines negative, and the magnitude is 24.

The magnitude for both positive and negative values is the same, but they change only with their signs.

The range of values for the sign and magnitude representation is from **-127** to **127**.

There are eight conditions to consider while adding or subtracting signed numbers. These conditions are based on the operations implemented and the sign of the numbers.

The table displays the algorithm for addition and subtraction. The first column in the table displays these conditions. The other columns of the table define the actual operations to be implemented with the magnitude of numbers. The last column of the table is needed to avoid a negative zero. This defines that when two same numbers are subtracted, the output must not be - 0. It should consistently be +0.

In the table, the magnitude of the two numbers is defined by P and Q.

Addition and Subtraction of Signed Magnitude Numbers

Operations	Addition of magnitudes	Subtraction of magnitudes		
(+P) + (+Q)	+(P+Q)	**P>Q**	**P<Q**	**P=Q**
(+P) + (-Q)		+(P-Q)	-(Q-P)	+(P-Q)
(-P) + (+Q)		-(P-Q)	+(Q-P)	+(P-Q)
(-P) + (-Q)	-(P+Q)			
(+P) - (+Q)		+(P-Q)	-(Q-P)	+(P-Q)
(+P) - (-Q)	+(P+Q)			
(-P) - (+Q)	-(P+Q)			
(-P) - (-Q)		-(P-Q)	+(Q-P)	+(P-Q)

As display in the table, the addition algorithm states that –

- When the signs of P and Q are equal, add the two magnitudes and connect the sign of P to the output.
- When the signs of P and Q are different, compare the magnitudes and subtract the smaller number from the greater number.
- The signs of the output have to be equal as P in case P > Q or the complement of the sign of P in case P < Q.
- When the two magnitudes are equal, subtract Q from P and modify the sign of the output to positive.

The subtraction algorithm states that –

- When the signs of P and Q are different, add the two magnitudes and connect the signs of P to the output.
- When the signs of P and Q are the same, compare the magnitudes and subtract the smaller number from the greater number.
- The signs of the output have to be equal as P in case P > Q or the complement of the sign of P in case P < Q.
- When the two magnitudes are equal, subtract Q from P and modify the sign of the output to positive.

6.3. Addition and Subtraction with Signed 2's Complement Data

Signed 2's complement is a binary representation of signed integers. In this system, positive numbers are represented as regular binary, while negative numbers are represented using 2's complement notation. Here's how addition and subtraction work with signed 2's complement data.

Addition:

1. **Addition of positive numbers:**
    - Perform binary addition as usual.
    - If there's a carry out of the most significant bit (MSB), discard it (overflow is ignored in 2's complement).

Example:

0101 (5 in decimal)

0011 (3 in decimal)

1000 (8 in decimal, discard the carry)

2. **Addition of negative numbers:**

- Add the numbers as if they were positive.

- If there's a carry out of the MSB, ignore it.

- If the result is positive, it means there was an overflow, and the result is incorrect for negative numbers.

Example:

1101 (-3 in decimal)

1010 (-6 in decimal)

10111 (Incorrect result, ignore the carry)

Subtraction:

1. **Subtraction of positive numbers:**

- Perform binary subtraction as usual.

Example:

1101 (13 in decimal)

0101 (5 in decimal)

1000 (8 in decimal)

2. **Subtraction of negative numbers: **

- To subtract a negative number, add the 2's complement of that number.

- Perform binary addition as explained in the addition section.

Example:

1101 (-3 in decimal)

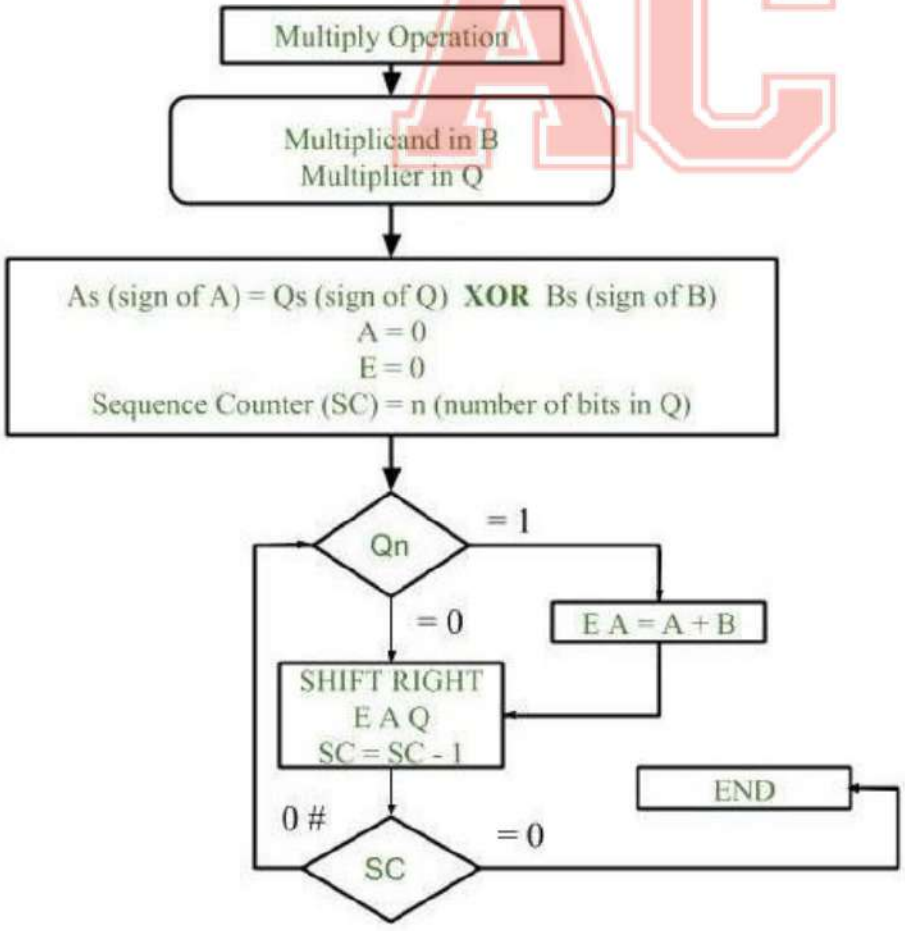
1010 (10 in decimal, 2's complement of -6)

011 (Discard the carry)

Remember that overflow may occur in signed 2's complement arithmetic, and the result might not be accurate if the numbers are too large or too small to be represented with the given number of bits. In practice, overflow checks are important to ensure the correctness of calculations.

6.4. Multiplication of Signed Magnitude Data

Multiplication of signed magnitude data involves multiplying two numbers, one of which or both are in signed magnitude representation. The process is similar to the multiplication of unsigned numbers, but you need to consider the sign bit and perform additional steps.



Example:

Multiplicand = 10111

Multiplier = 10011

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Qn = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Qn = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Qn = 0; shift right EAQ	0	01000	10110	010
Qn = 0; shift right EAQ	0	00100	01011	001
Qn = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000

Final product in AQ
0110110101

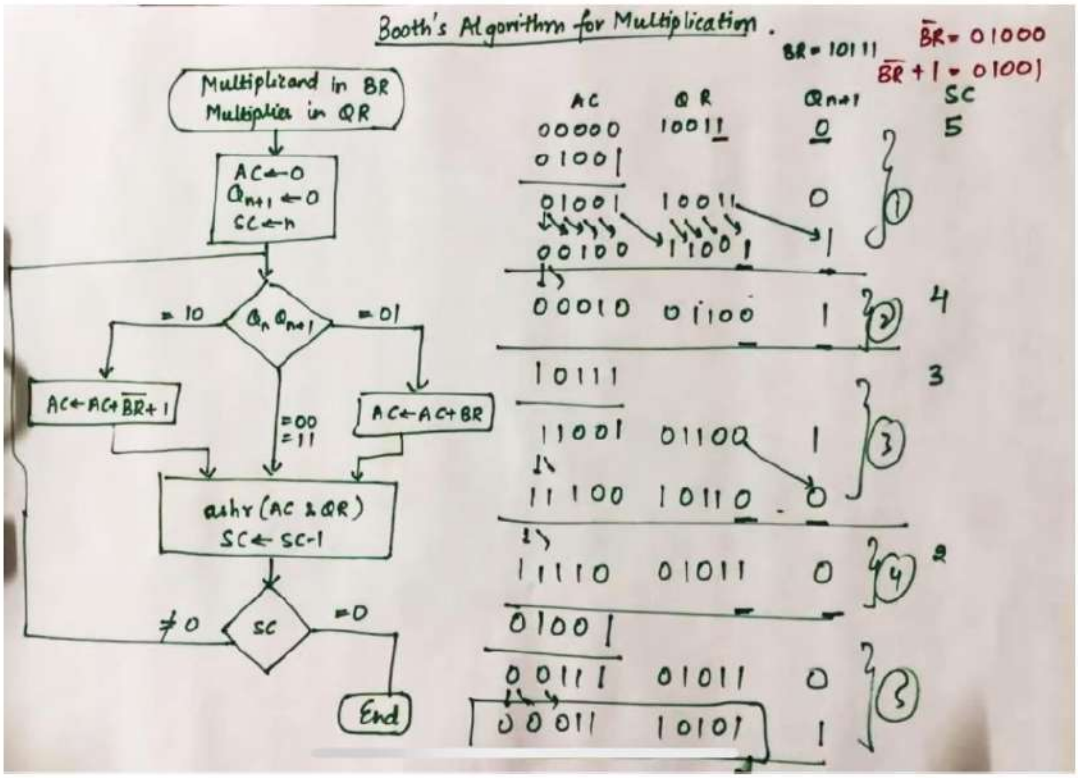
Notes:

1. **Sign Bit:**
   - The sign bit is determined by the signs of the original numbers. In this case, both numbers are negative, so the sign of the result is negative.
2. **Overflow:**
   - Check for overflow, especially in systems with limited bit representation. Overflow occurs if the result is too large or too small to be represented with the given number of bits.
3. **Sign Extension:**
   - If you need to store the result in a larger register, you may need to sign-extend the result.

6.5. Booth Multiplication

Booth's multiplication algorithm is a technique used for the multiplication of signed binary numbers, particularly in binary systems where signed numbers are represented using 2's complement notation. It's an algorithm that reduces the number of addition and subtraction operations required for multiplication.

Example:



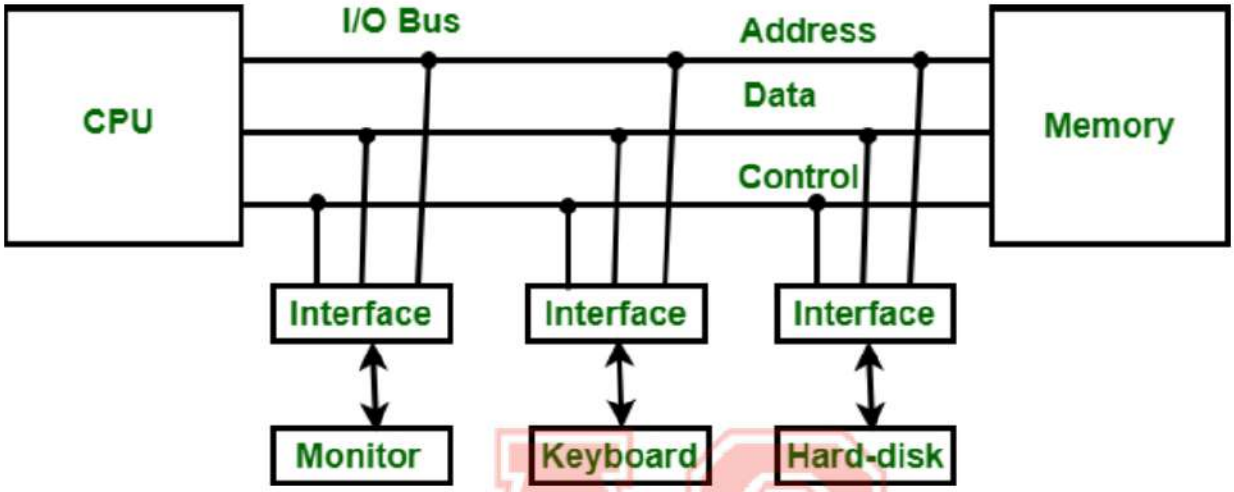
Notes:

- Booth's algorithm can be more efficient than traditional multiplication algorithms in terms of the number of additions and subtractions required.
- The algorithm is not limited to 4-bit numbers; it can be applied to numbers of any bit length.
- Care should be taken to handle overflow conditions, especially in systems with limited bit representation.

Unit 7. Input Output Organization [Marks: 9]

7.1. Input-Output Interface

Input-Output Interface is used as a method which helps in transferring of information between the internal storage devices i.e., memory and the external peripheral device. A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices. For Example: A keyboard and mouse provide Input to the computer are called input devices while a monitor and printer that provide output to the computer are called output devices. Just like the external hard-drives, there is also availability of some peripheral devices whichare able to provide both input and output.

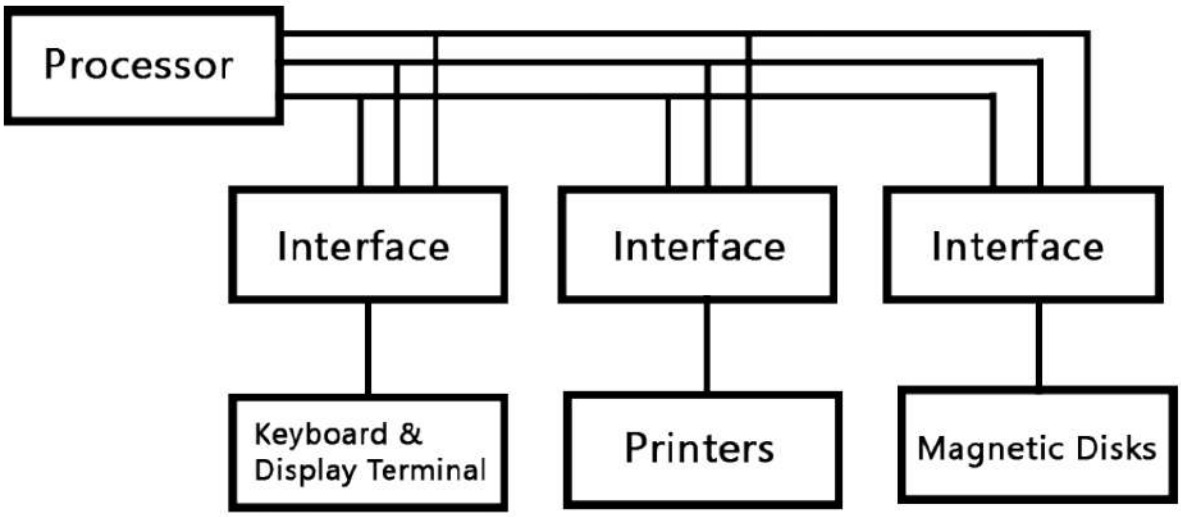


In micro-computer base system, the only purpose of peripheral devices is just to provide **special communication links** for the interfacing them with the CPU.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output signal.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also converts digital data into analog signal and vice-versa.

7.1.1. I/O Bus and Interface Modules



Communication Link between Processor and Peripherals

The data bus, address bus and control bus that arise out of the processor and are intended to communicate with I/O devices are called I/O bus. The communication link between the processor and several peripherals is shown in the given figure. The I/O bus is connected to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address bus. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects an address to be its own, it activates the path between the bus and the device that it controls. All other peripherals are disabled. At the same time, a function code is provided to the control bus which is called I/O command. The types of I/O commands that are given out by the processor are:

1. **Control Commands:** This is the function code that activates the corresponding peripherals and informs them about what to do.
2. **Status Commands:** A status command is used to test various status conditions in the interface and the peripheral devices like BUSY, ERROR, data available or not in the buffer etc .
3. **Data Output Command:** A data output command causes the interface to respond by transferring the data from the processor to the peripheral. The data is sent from the CPU to the buffer of interface after this command is provided.
4. **Data Input Command:** This command is sent by the CPU if the data is to be read from the peripheral. After this command is issued, the data of peripheral are extracted into the buffer of the interface and are read by the CPU.

7.1.2. I/O vs. Memory Bus

I/O (Input/Output) Bus and Memory Bus are both critical components of a computer's architecture, but they serve different purposes in facilitating data transfer within a computer system.

Memory Bus:

- The Memory Bus is responsible for the transfer of data between the CPU and the main memory (RAM).
- It is a high-speed communication pathway used to fetch instructions and data from RAM for processing by the CPU.
- The width of the memory bus (measured in bits) determines the amount of data that can be transferred in a single bus cycle. Wider buses allow for larger data transfers.
- Access to RAM is essential for the CPU's operations, and a faster memory bus can result in better overall system performance.

I/O Bus:

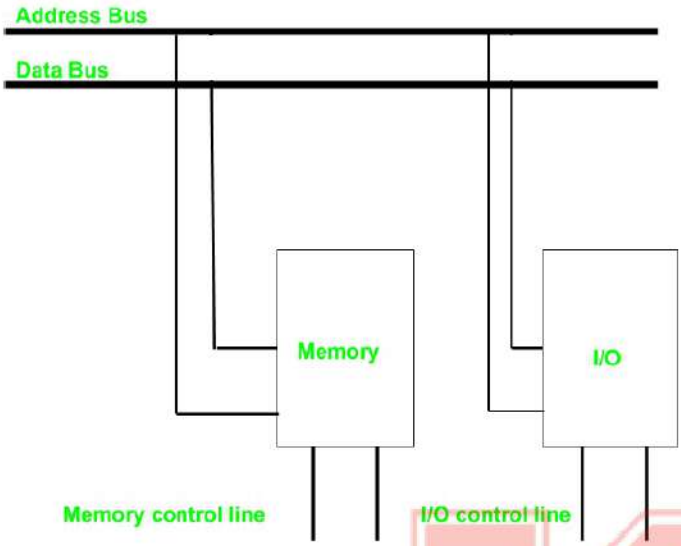
- The I/O Bus, also known as the Peripheral Component Interconnect (PCI) bus or simply I/O channel, is responsible for connecting the CPU to peripheral devices and external components.
- I/O operations involve the transfer of data between the CPU and devices such as hard drives, graphics cards, network cards, USB peripherals, etc.
- Unlike the memory bus, which primarily deals with RAM, the I/O bus deals with a diverse range of devices, each with its own data transfer requirements.
- The PCI Express (PCIe) is a common modern I/O bus standard that provides high-speed data transfer between the CPU and various peripherals.

In summary, while the Memory Bus focuses on the fast exchange of data between the CPU and RAM for the purpose of program execution, the I/O Bus facilitates communication between the CPU and a variety of peripheral devices for input and output operations. Both buses are crucial for the overall functionality and performance of a computer system.

7.1.3. Isolated vs. Memory-Mapped I/O

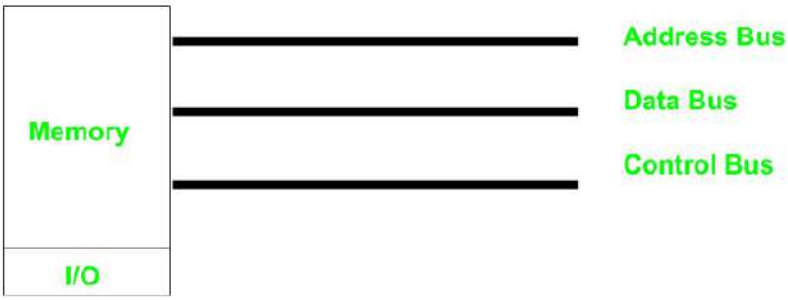
Isolated I/O and Memory-Mapped I/O are two different approaches to managing communication between a computer's Central Processing Unit (CPU) and Input/Output (I/O) devices.

Isolated I/O:



- In Isolated I/O, dedicated instructions and separate address spaces are used for input/output operations.
- Special I/O instructions, distinct from memory instructions, are employed to communicate with I/O devices. These instructions typically include IN (Input) and OUT (Output) instructions.
- The CPU and the I/O devices have separate address spaces, meaning that the addresses used for I/O operations are different from the memory addresses. The CPU sends commands and data to specific I/O ports using dedicated instructions.
- This approach provides isolation between memory and I/O operations, making it clear when the CPU is performing I/O operations and allowing for better control over the flow of data between the CPU and peripherals.

Memory-Mapped I/O:



- In Memory-Mapped I/O, I/O devices are mapped to the same address space as the computer's memory.
- Both memory and I/O devices share the same address space, and specific addresses are reserved for I/O operations. The CPU uses load and store instructions to read from or write to these addresses, treating I/O devices as if they were memory locations.
- Memory-Mapped I/O simplifies the programming model as the same set of instructions can be used for both memory and I/O operations. However, it can make it more challenging to distinguish between memory access and I/O operations.
- This approach may lead to potential conflicts if there is a need to access the same address for both memory and I/O, and careful management is required to avoid issues.

Isolated I/O	Memory Mapped I/O
Memory and I/O have separate address space	Both have same address space
All address can be used by the memory	Due to addition of I/O addressable memory become less for memory
Separate instruction control read and write operation in I/O and Memory	Same instructions can control both I/O and Memory
In this I/O address are called ports.	Normal memory address is for both
More efficient due to separate buses	Lesser efficient

Larger in size due to more buses	Smaller in size
It is complex due to separate logic is used to control both.	Simpler logic is used as I/O is also treated as memory only.

7.2. Asynchronous Data Transfer: Strobe, Handshaking

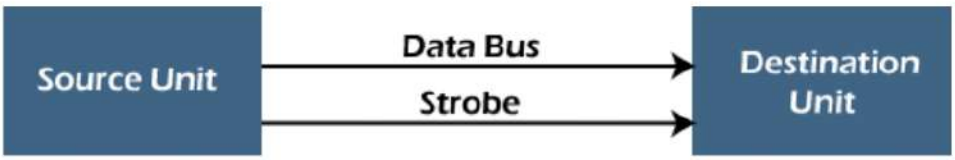
Asynchronous data transfer involves the communication between two devices without a synchronized clock signal. Strobe and handshaking are two methods used in asynchronous data transfer to manage the timing and coordination of data transmission between sender and receiver.

Strobe (or Ready/Busy) Method:

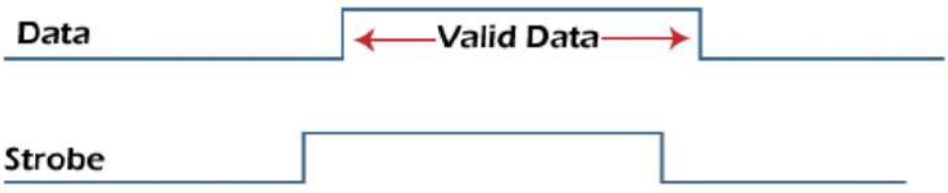
- In the strobe method, the sender initiates data transfer when it is ready, and the receiver acknowledges its readiness.
- The sender uses a signal called "strobe" or "ready" to indicate that it has data available for transmission.
- The receiver, upon receiving the strobe signal, acknowledges its readiness to receive the data by asserting a "busy" signal or a similar acknowledgment signal.
- The sender then releases the data, and the receiver captures it.

Source initiated strobe:

In the below block diagram, you can see that strobe is initiated by source, and as shown in the timing diagram, the source unit first places the data on the data bus.



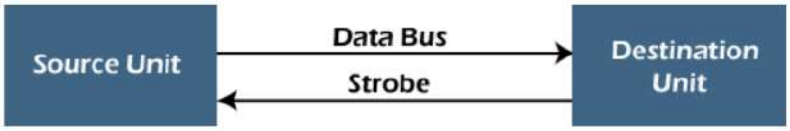
(a) Block Diagram



(b) Timing Diagram

Destination initiated strobe:

In the below block diagram, you see that the strobe initiated by destination, and in the timing diagram, the destination unit first activates the strobe pulse, informing the source to provide the data.



(a) Block Diagram

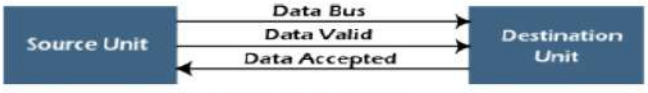
(b) Timing Diagram

Handshaking Method:

- Handshaking is a more elaborate method that involves a series of signals exchanged between the sender and receiver to coordinate the transfer.
- The process typically begins with the sender asserting a "request to send" (RTS) signal, indicating its intention to transmit data.
- The receiver responds with a "clear to send" (CTS) signal, indicating its readiness to receive data.
- The sender, upon receiving the CTS signal, initiates the actual data transfer.
- After the data transfer is complete, an acknowledgment signal or an end-of-transmission signal may be exchanged to confirm the successful transfer.

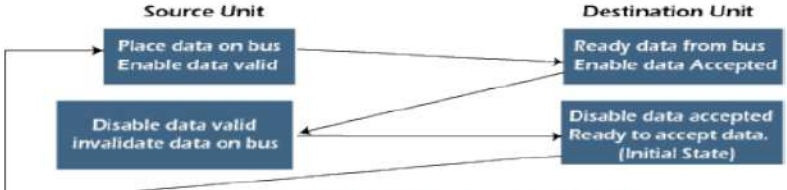
Source initiated handshaking:

In the below block diagram, you can see that two handshaking lines are "**data valid**", which is generated by the source unit, and "**data accepted**", generated by the destination unit.



(a) Block Diagram

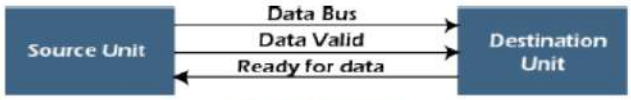
(b) Timing Diagram



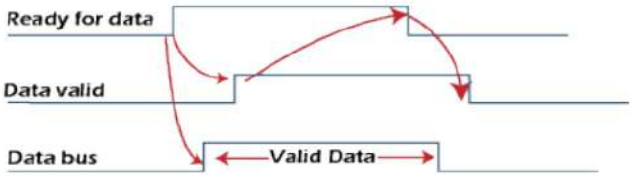
(c) Sequence Diagram (Sequence of events)

Destination initiated handshaking:

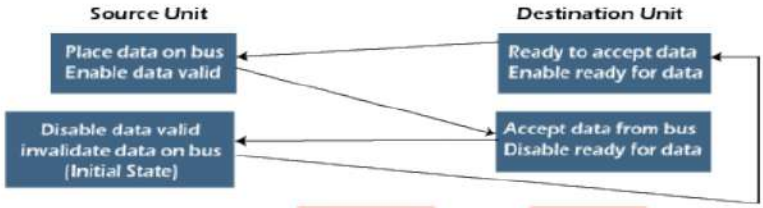
In the below block diagram, you see that the two handshaking lines are "**data valid**", generated by the source unit, and "**ready for data**" generated by the destination unit. Note that the name of signal data accepted generated by the destination unit has been changed to ready for data to reflect its new meaning.



(a) Block Diagram



(b) Timing Diagram



(c) Sequence Diagram (Sequence of events)

7.3. Modes of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target are always the memory unit.

We can transfer this information using three different modes of transfer.

- Programmed I/O.
- Interrupt- initiated I/O.
- Direct memory access (DMA)

7.3.1. Programmed I/O

Programmed I/O (Input/Output) refers to a method of data transfer between a computer's central processing unit (CPU) and an external device. In Programmed I/O, the CPU takes an active role in managing the data transfer process, issuing commands to the I/O device and waiting for the completion of each operation.

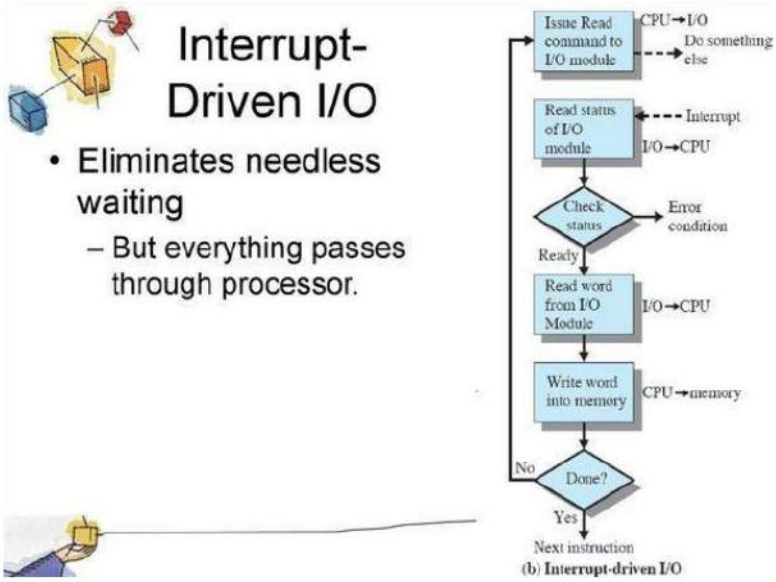
Here's a basic overview of how Programmed I/O works:

1. **CPU Issues Command:** The CPU sends a command to the I/O device, instructing it to perform a specific operation, such as reading from or writing to a particular memory location or port.
2. **Device Performs Operation:** The I/O device carries out the requested operation, whether it's reading data from a peripheral or writing data to it.
3. **CPU Polls for Completion:** The CPU regularly checks (polls) the status of the I/O operation to determine if it has been completed. During this time, the CPU may be idle or performing other tasks.
4. **Data Transfer:** Once the I/O operation is completed, the CPU can then transfer the data between the device and the system's memory.

Programmed I/O has some advantages, such as simplicity and ease of implementation, but it also has significant drawbacks, especially in terms of performance. The CPU has to wait for the completion of each I/O operation, leading to inefficient use of processing time. This method is suitable for low-speed devices or situations where the CPU can afford to wait for I/O operations to complete without impacting overall system performance.

7.3.2. Interrupt-Initiated I/O

Interrupt-Initiated I/O, also known as Interrupt-Driven I/O, is a technique used in computer systems to improve the efficiency of input/output operations. In this approach, the CPU is not actively involved in constantly polling the I/O device for the completion of operations, as in Programmed I/O. Instead, the CPU is interrupted by the I/O device when an operation is completed or when the device requires attention.



Here's how Interrupt-Initiated I/O typically works:

1. **CPU Issues Command:** Similar to Programmed I/O, the CPU issues a command to the I/O device, instructing it to perform a specific operation.
2. **I/O Device Performs Operation:** The I/O device starts executing the requested operation (e.g., reading data from a peripheral or writing data to it).
3. **Interrupt Request (IRQ):** When the I/O operation is completed or when the device needs attention (e.g., data is ready for transfer), the I/O device sends an interrupt signal to the CPU.
4. **CPU Interrupted:** Upon receiving the interrupt signal, the CPU temporarily stops its current task and jumps to an interrupt service routine (ISR) specific to the interrupting device. The ISR handles the completion of the I/O operation or takes appropriate actions based on the device's needs.
5. **Data Transfer:** The CPU and the I/O device can then transfer data or perform other necessary actions.

Interrupt-Initiated I/O has several advantages over Programmed I/O:

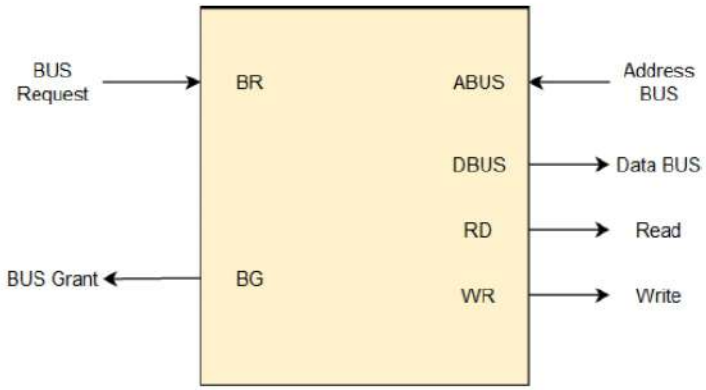
- **Efficiency:** The CPU is not continuously polling the I/O device, which improves overall system efficiency. The CPU can perform other tasks while waiting for interrupts.
- **Reduced CPU Overhead:** The CPU is only involved when necessary, reducing the overhead associated with managing I/O operations.
- **Responsiveness:** The system can respond quickly to events and data availability, as the CPU can be interrupted as soon as an I/O operation is completed.

This method is more suitable for high-speed devices and scenarios where the CPU's time is better utilized for other tasks rather than waiting for I/O operations to complete. However, it does require the presence of interrupt hardware and proper interrupt handling mechanisms in the operating system.

7.3.3. Direct memory Access

The Direct Memory Access (DMA) mode of transfer involves the use of a DMA controller to facilitate data transfer between peripheral devices and system memory without requiring constant intervention from the central processing unit (CPU). DMA is beneficial for enhancing data transfer rates and reducing the CPU's involvement in managing each step of the transfer process.

Here is an overview of the DMA mode of transfer:



1. **Initialization:**
   - The CPU initializes the DMA controller by providing it with necessary information about the data transfer, including the source and destination addresses in memory, the number of data bytes to transfer, and the direction of the transfer (read or write).
   - The DMA controller may be configured to operate in different modes, such as burst mode (transferring multiple data items in a single cycle) or cycle stealing mode (transferring one data item per cycle).
2. **Request for DMA:**
   - When a peripheral device, such as a disk controller or a network interface, is ready to transfer data, it sends a request to the DMA controller for DMA access.
3. **Arbitration:**
   - If multiple devices request DMA simultaneously, the DMA controller performs arbitration to determine which device gets access to the system bus for data transfer. This ensures fair and efficient usage of the DMA controller.
4. **Bus Control:**
   - Once the DMA controller gains control of the system bus, it coordinates the data transfer between the peripheral device and memory without direct involvement from the CPU.

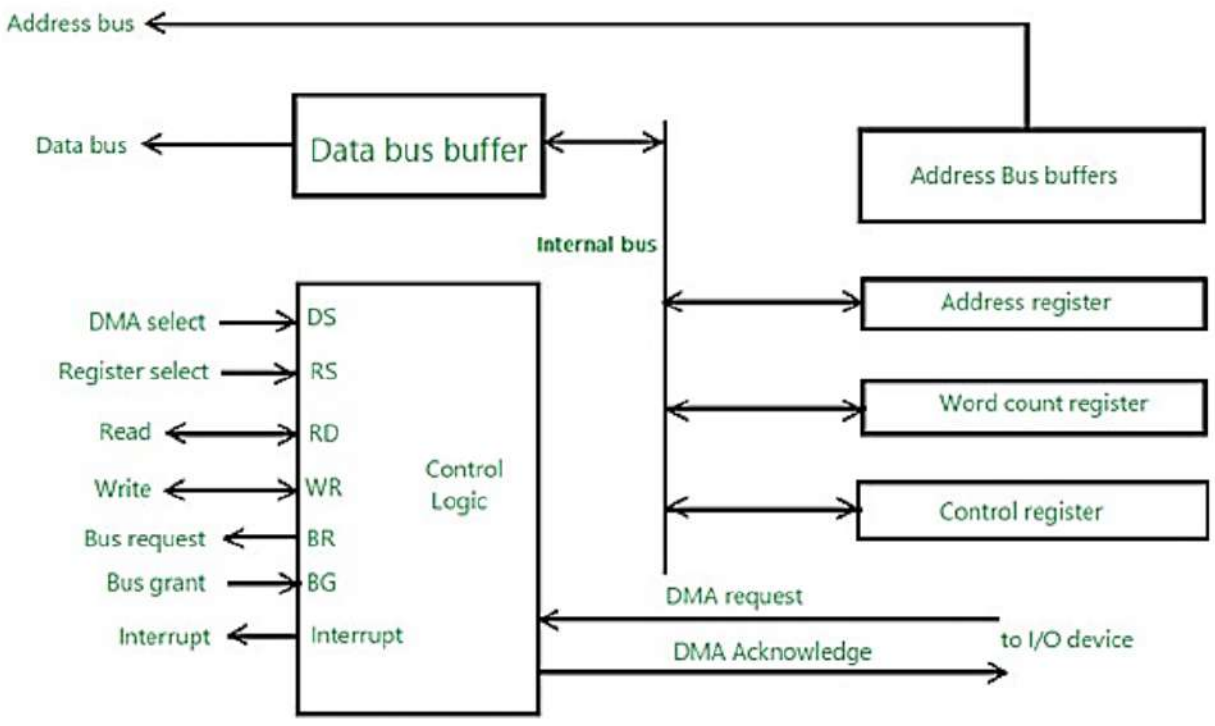
5. **Data Transfer:**
    - The DMA controller reads or writes data directly between the peripheral device and memory, utilizing the data addresses provided during initialization. This occurs independently of the CPU, allowing the CPU to perform other tasks simultaneously.
6. **Interrupt or Notification:**
    - Upon completion of the data transfer, the DMA controller may generate an interrupt signal to notify the CPU. This interrupt allows the CPU to perform any necessary tasks related to the completed data transfer.
7. **Repeat:**
    - If there are more data to transfer or if the peripheral device has additional data ready, the DMA controller can be configured to repeat the process.

The DMA mode of transfer is particularly advantageous for scenarios involving large-scale data movement, such as disk I/O operations, graphics rendering, and network communication. It reduces the CPU overhead associated with managing data transfers and enables more efficient utilization of the CPU for other computational tasks.

7.4 Direct Memory Access, Input-Output Processor, DMA vs. IOP

Direct Memory Access (DMA)

Direct Memory Access (DMA) is a feature in computer architecture that allows peripherals (devices such as disk drives, graphics cards, or network interfaces) to access the system's memory without involving the central processing unit (CPU). DMA is used to enhance the overall performance of the system by offloading data transfer tasks from the CPU.



Modes of Data Transfer in DMA

- **Burst Mode:** In Burst Mode, buses are handed over to the CPU by the DMA if the whole data is completely transferred, not before that.
- **Cycle Stealing Mode:** In Cycle Stealing Mode, buses are handed over to the CPU by the DMA after the transfer of each byte. Continuous request for bus control is generated by this Data Transfer Mode. It works more easily for higher-priority tasks.
- **Transparent Mode:** Transparent Mode in DMA does not require any bus in the transfer of the data as it works when the CPU is executing the transaction.

Here's how DMA typically works:

1. **Data Transfer Initiation:** When a peripheral device needs to transfer data to or from the system memory, it sends a request to the DMA controller.
2. **DMA Controller:** The DMA controller is a specialized hardware component that manages the data transfer between the peripheral device and the system memory. It operates independently of the CPU.
3. **Memory Access:** The DMA controller gains control of the system bus and directly accesses the memory to read or write data. This bypasses the CPU, which can continue with other tasks while the data transfer is in progress.
4. **Interrupts:** Once the data transfer is complete, the DMA controller can interrupt the CPU to notify it that the operation has finished. The CPU can then process the data or perform other tasks as needed.

DMA is particularly beneficial for high-speed data transfer operations, such as large file transfers or streaming multimedia. It reduces the CPU's involvement in these data transfers, freeing up the CPU to focus on other computations and improving the overall efficiency of the system.

Types of Direct Memory Access (DMA)

There are four popular types of DMA.

Single-Ended DMA: Single-Ended DMA Controllers operate by reading and writing from a single memory address. They are the simplest DMA.

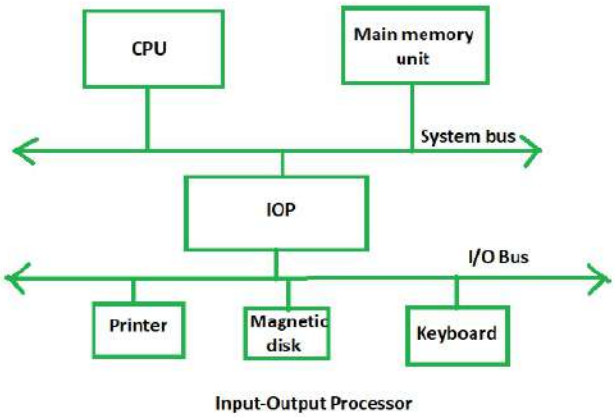
Dual-Ended DMA: Dual-Ended DMA controllers can read and write from two memory addresses. Dual-ended DMA is more advanced than single-ended DMA.

Arbitrated-Ended DMA: Arbitrated-Ended DMA works by reading and writing to several memory addresses. It is more advanced than Dual-Ended DMA.

Interleaved DMA: Interleaved DMA are those DMA that read from one memory address and write from another memory address.

Input-Output Processor

An Input-Output Processor (IOP) is a specialized processor or controller that manages communication between the central processing unit (CPU) and peripheral devices in a computer system. The primary purpose of an IOP is to handle input and output operations efficiently, freeing up the main CPU to focus on computational tasks.



Input-Output Processor

Here are key functions and characteristics of an Input-Output Processor:

1. **Peripheral Communication:** The IOP manages communication with various peripheral devices such as storage devices (hard drives, SSDs), input devices (keyboard, mouse), output devices (display, printer), and communication devices (network interface cards).
2. **Data Transfer Control:** IOPs control the data transfer between the CPU and peripheral devices. They handle the protocols and data formats required for communication with different types of peripherals.
3. **Buffering:** IOPs often use buffers to temporarily store data during input or output operations. This buffering helps in managing the speed differences between the CPU and peripheral devices.
4. **Interrupt Handling:** IOPs can generate interrupts to notify the CPU when an input or output operation is complete or when attention is required. This allows the CPU to efficiently multitask and respond to events as they occur.
5. **I/O Address Decoding:** The IOP decodes the addresses associated with input and output operations, directing data to or from the appropriate peripheral device.
6. **DMA Control:** In some systems, IOPs may incorporate Direct Memory Access (DMA) controllers to facilitate high-speed data transfers between peripheral devices and system memory without involving the CPU extensively.
7. **Device Management:** IOPs often include functionality for managing the configuration and status of connected devices. This can include error handling, device initialization, and power management.

IOPs play a crucial role in enhancing the overall efficiency of a computer system by offloading I/O-related tasks from the main CPU. This separation allows the CPU to focus on

computation while IOPs handle the complexities of interacting with diverse peripheral devices.

DMA vs. IOP

Characteristics	DMA	IOP
Definition	DMA (Direct MemoryAccess) is a data transfer method in which a peripheral device can directly access system memorywithout involving the CPU.	IOP is a technique that allows the CPUto handle data transfers between a peripheral device and system memory.
Data transfer speed	The data transfer speedof DMA is faster compared to IOP because it doesn't involve the CPU.	The data transfer speed of IOP is slower compared toDMA because it involves the CPU.
Involvement of CPU	It doesn't involve theCPU.	It involves the CPU.
Complexity	DMA is more complexcompared to IOP because it requires specialized hardware	IOP is less complexcompared to DMA because it depends on software to control the data transfer process.
CPU Utilization	The usage of CPU isless in DMA.	The usage of CPUis more in PIO.
Suitability	It is suitable for transferring large amounts of data between peripheraldevices and main memory.	It is suitable for transferring a smallamount of data between peripheraldevices and main memory.

THE END