

BASIC ORGANIZATION OF COMPUTER ARCHITECTURE

Guided by-Dr. SARVESH PANDEY

Presented by-

- ▶ GITIKA KISHOR (20229MAT009)
- ▶ IPSITA CHATTERJEE (20229MAT010)
- ▶ PRIYANKA PAUL (20229CMP005)
- ▶ SHREYA SRIVASTAVA (20229CMP006)

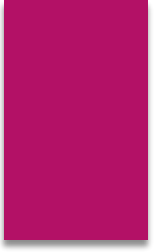
Contents



1. Stored program concepts
2. Components of a computer system
3. Machine instruction
4. Opcodes and Operands
5. Instruction cycle
6. Organization of Central Processing Unit: ALU, Hardwired & Microprogrammed Control Unit
7. General purpose and Special purpose Registers

Stored Program Concept

- ▶ It is the fundamental idea upon which all modern computers are based.
- ▶ First introduced by John von Neumann in 1945.
- ▶ This architecture offers storage of your programs into Read-Only Memory (ROM) chips.
- ▶ Instead of having to hardwire them into your computer's circuitry as was done with previous computers.
- ▶ This concept allows much easier programming and reprogramming of a machine.
- ▶ Also makes hardware upgrades much simpler than before because only software needs to be modified if changes need to be made.

- 
- ▶ The Stored Program Control Concept is an innovation that has allowed for the storage of instructions in computer memory.
 - ▶ Enable computers not only execute tasks sequentially but also intermittently.
 - ▶ The ENIAC (Electronic Numerical Integrator and Computer) is often called the “first computer.”
 - ▶ But it was actually a programming system rather than just an Instruction Machine.
 - ▶ It is used stored program concepts in which machine use memory for processing data.

Stored program concept

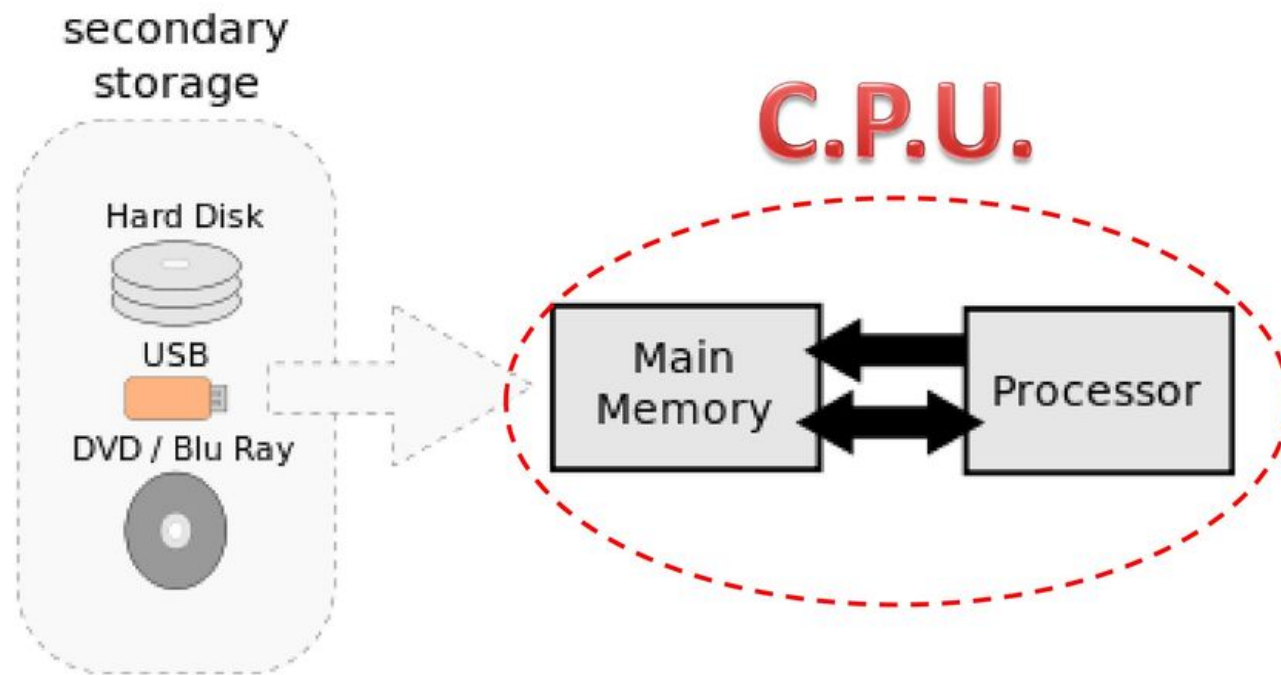


Fig.1

3 Ways Of Stored Program Concept

1. VON-NEUMANN MODEL

- ▶ It consisted of three main parts: The Control Unit (CU), Arithmetic & Logic Memory Unit(ALU) Registers with input/outputs.
- ▶ There is only one processor and it uses memory for both instructions as well as data.
- ▶ The Central Processing Unit (CPU) is the most important part in von-Neumann model.
- ▶ It processes information and makes decisions concerning what to do with it all.

Von-Neumann Basic Structure:

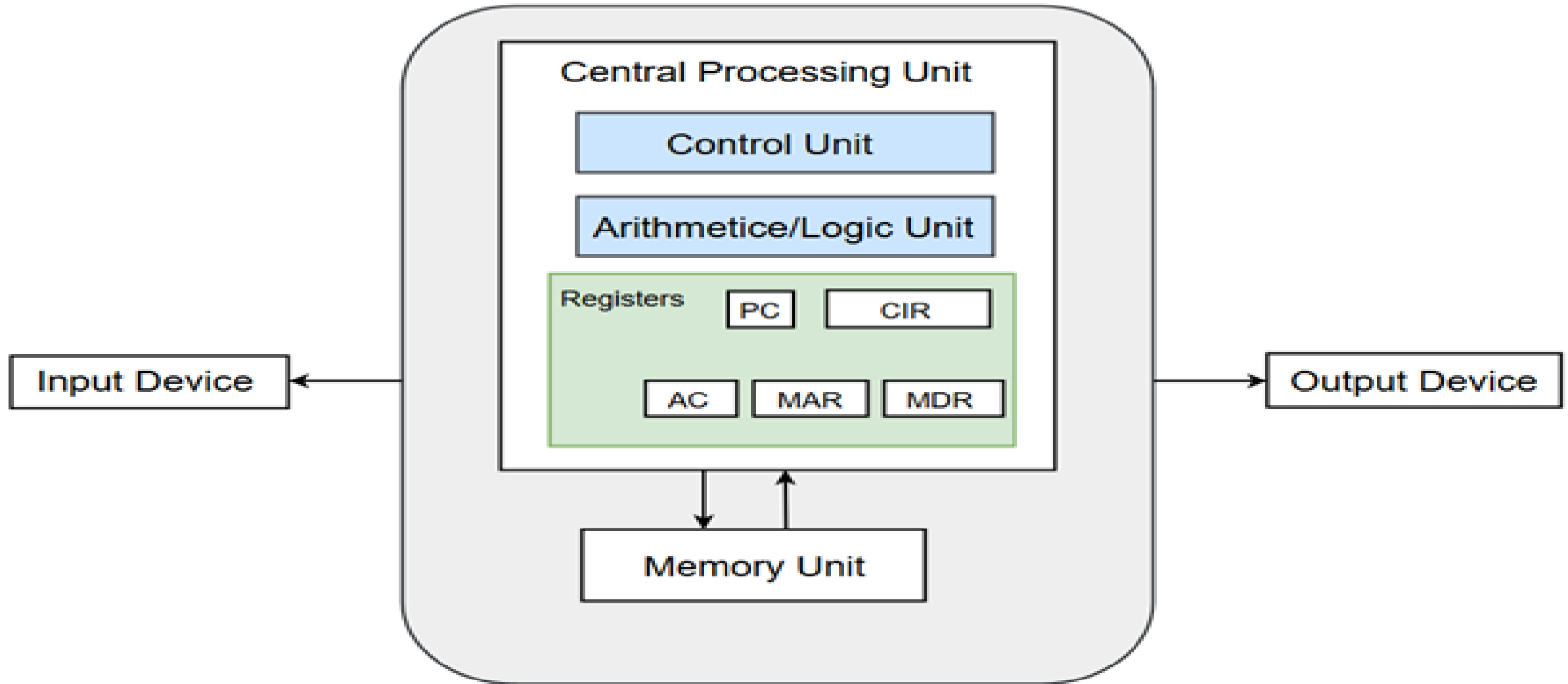


Fig.2

2. GENERAL PURPOSE SYSTEM

In more modern terms we have general purpose computers which use Central Processing Units (CPUs) that contain an ALU along with several registers all interconnected by System Bus lines including Address Data & Control Status Signals.

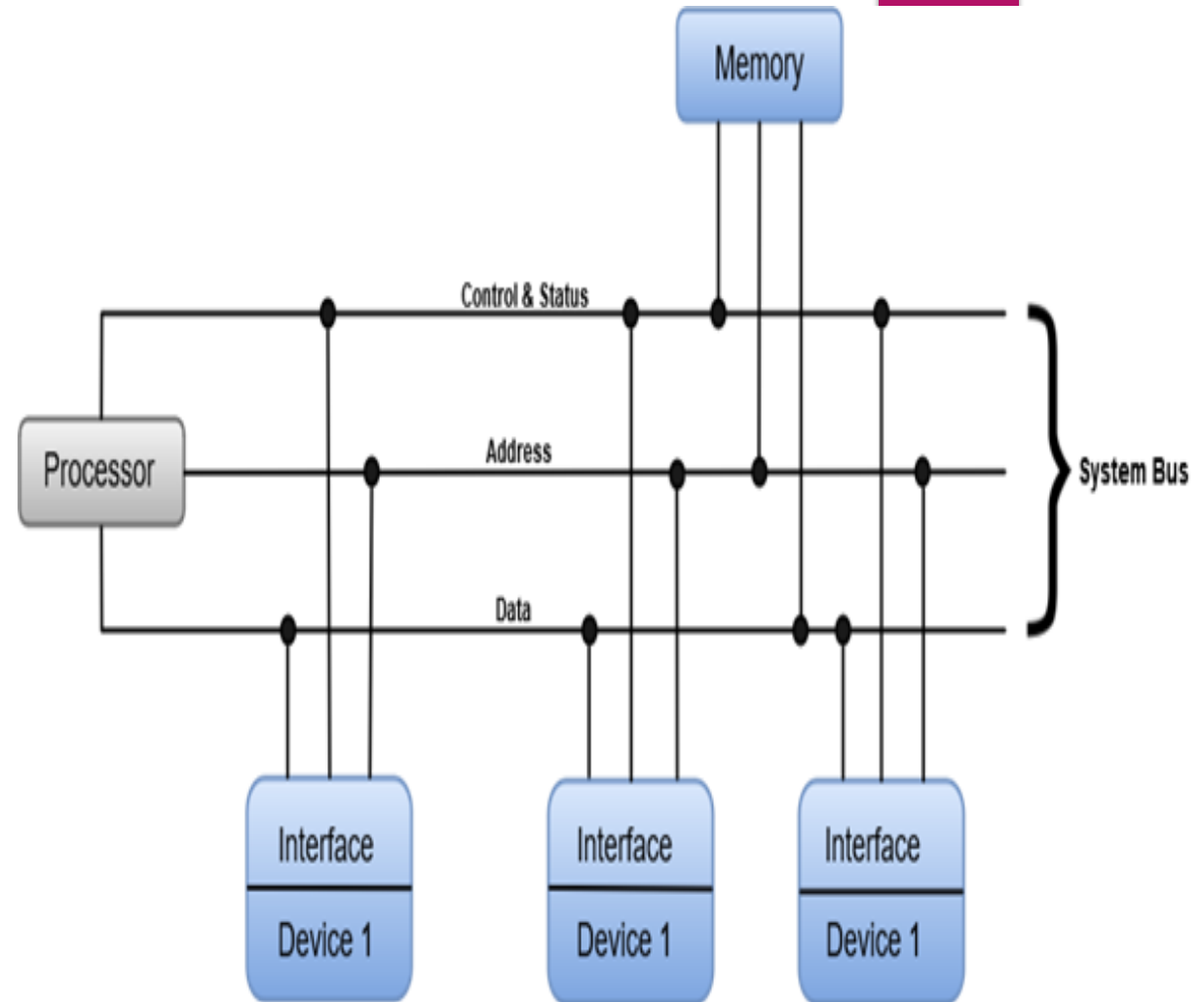


Fig.3

3. PARALLEL PROCESSING

- ▶ Described as a class of techniques which enables the system to achieve simultaneous data-processing tasks.
- ▶ These systems are typically faster than single-core or serial computing because they have more cores, speed up an instruction while it's being processed in the ALU component for CPU (central processing units).

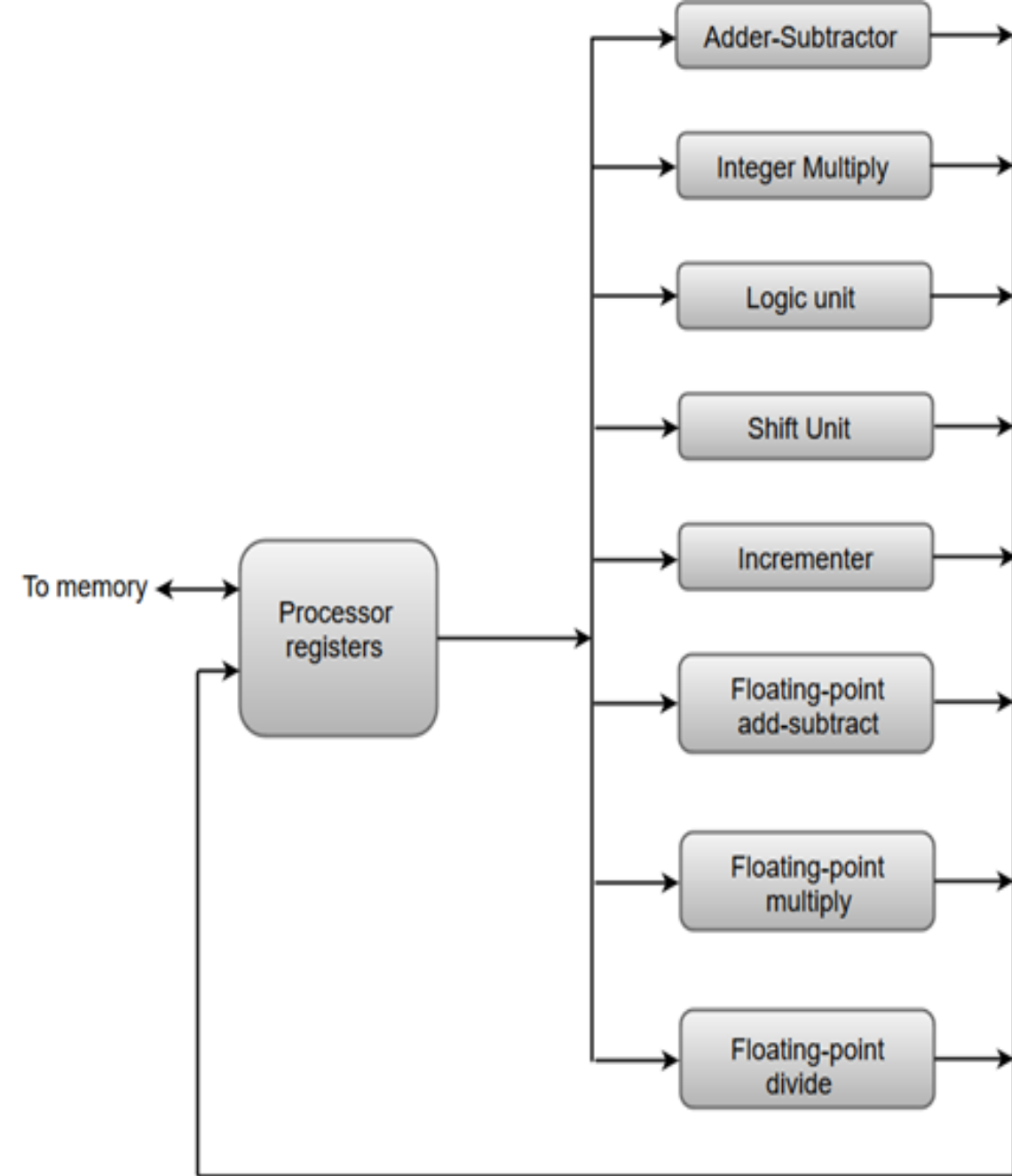
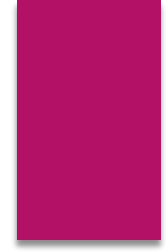


Fig.4

COMPONENTS OF COMPUTER SYSTEM



- Saying that computers have revolutionized our lives would be an understatement. These machines have completely changed the way we perform all daily tasks.
- Some basic elements of computer include hardware, software, programmes, data and connectivity. No computer can function in the absence of these elements.
- Every computer system has the following three basic components:
 - ▶ 1) Input unit
 - ▶ 2) Central processing unit
 - ▶ 3) Output unit
- Apart from other components these three are primarily responsible for making a computer function. They must work in complete synergy because that will ensure smooth overall functioning.

Elements of a computer system

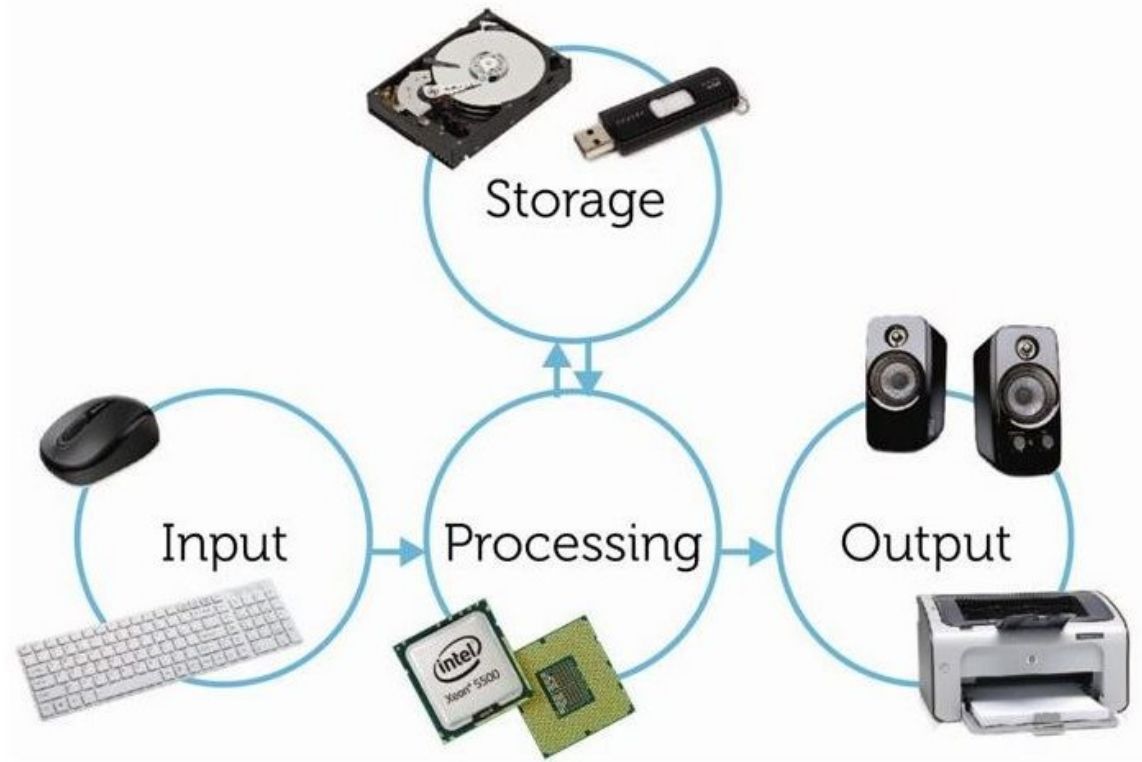


Fig. 5

Input Unit

- The main function of input devices is to direct commands and data into computers.
- Data can be in the form of numbers, words, actions, commands, etc.
- Computers then use their CPU to process this data and produce output.
- **Example :**
 1. A laptop's keyboard is an input unit that enters numbers and characters.
 2. A mouse can be an input unit for entering directions and commands.
 3. Other examples include barcode readers, Magnetic Ink Character Readers (MICR), Optical Character Readers (OCR), etc.

Central Processing Unit (CPU)

- After receiving data and commands from users, a computer system now has to process it according to the instructions provided.
- Here, it has to rely on a component called the central processing unit.
- The CPU further uses these three elements.

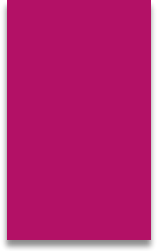
Output Unit

- The third and final component of a computer system is the output unit.
- After processing of data, it is converted into a format which humans can understand.
- After conversion, the output units displays this data to users. Examples of output devices include monitors, screens, printers and speakers.
- Thus, output units basically reproduce the data formatted by the computer for users' benefit.

Machine Instructions

- These are commands or programs written in machine code of a machine (computer) that it can recognize and execute.
- Consists of several bytes in memory that tells the processor to perform one machine operation.
- The processor looks at machine instructions in main memory one after another, and performs one machine operation for each machine instruction.
- The collection of machine instructions in main memory is called a machine language program.
- Machine language is a set of instructions executed directly by a CPU.
- Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory.

Opcodes & Operands



- Each **assembly language statement** is split into an **opcode** and an **operand**.
- The **opcode** is the **instruction** that is executed by the **CPU** and the **operand** is the **data** or **memory location** used to execute that instruction.
- Within the computer, each instruction is represented by a sequence of bits.
- The instruction is divided into fields, corresponding to the constituent elements of the instruction.
- During instruction execution, an instruction is read into an **instruction register (IR)** in the processor.
- The processor must be able to extract the data from the various instruction fields to perform the required operation.
- It has become common practice to use a symbolic representation of machine instructions.
- Opcodes are represented by abbreviations, called ***mnemonics***, that indicate the operation.

Assembly language opcode mnemonics and instructions	Meaning/use
INP (Input)	Inputs a value, then stores the value in the accumulator
OUT (Output)	Outputs the accumulator contents
STA (Store)	Transfers a number from the accumulator to RAM
LDA (Load)	Transfers a number from RAM to the accumulator
ADD (Add)	Adds accumulator contents to the contents at a RAM address
SUB (Subtract)	Subtracts accumulator contents from the contents at a RAM address
BRA (Branch)	When looping, jumps to the RAM memory address
HLT (Halt/Stop/End)	Stops the processor
DAT (Data definition)	Variable definition

Operands

- These are also represented symbolically.
- For example, the instruction **ADD R, Y** may mean add the value contained in data location Y to the contents of register R. Here, Y refers to the address of a location in memory, and R refers to a particular register. Note that the operation is performed on the contents of a location, not on its address.
- Thus, it is possible to write a machine- language program in symbolic form. Each symbolic opcode has a fixed binary representation, and the programmer specifies the location of each symbolic operand. For example, the programmer might begin with a list of definitions:
$$X = 513$$
$$Y = 514$$
- A simple program would accept this symbolic input, convert opcodes and operand references to binary form, and construct binary machine instructions.
- An operand (written using **hexadecimal** notation) provides the data itself, or the location where the data to be processed is stored.

The table below is a small assembly language program with a description of the opcode and operand components during execution.

Original assembly language	Opcode	Operand	Description
INP	INP		Input value and store in the accumulator
STA 1C	STA	1C	Store the number at memory address 1C
INP	INP		Input value and store in the accumulator
ADD 1C	ADD	1C	Add this number to the number stored at memory address 1C
OUT	OUT		Output the result
HLT	HLT		Stop the program

Instruction Cycle

- ▶ A program residing in the memory unit of the computer consists of a sequence of instructions and is executed in the computer by going through a cycle for each instruction.
- ▶ The instruction cycle (also known as the fetch–decode–execute cycle, or simply the fetch-execute cycle) is the cycle that the CPU follows from **boot-up** until the computer has shut down in order to process instructions.
- ▶ In simpler CPUs, the instruction cycle is executed sequentially, each instruction being processed before the next one is started.
- ▶ In the basic computer each instruction cycle consists of the following phases:
 1. **Fetch an instruction from memory.**
 2. **Decode the instruction.**
 3. **Read the effective address from memory if the instruction has an indirect address.**
 4. **Execute the instruction.**
- ▶ Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Role of components

- The program counter (PC) is a special register that holds the memory address of the next instruction to be executed.
- During the fetch stage, the address stored in the PC is copied into the memory address register (MAR) and then the PC is incremented in order to "point" to the memory address of the next instruction to be executed.
- The CPU then takes the instruction at the memory address described by the MAR and copies it into the memory data register (MDR).
- The MDR also acts as a two-way register that holds data fetched from memory or data waiting to be stored in memory (it is also known as the memory buffer register (MBR) because of this).

- Eventually, the instruction in the MDR is copied into the current instruction register (CIR) which acts as a temporary holding ground for the instruction that has just been fetched from memory.
- During the decode stage, the control unit (CU) will decode the instruction in the CIR.
- The CU then sends signals to other components within the CPU, such as the arithmetic logic unit (ALU) and the floating point unit (FPU).
- The ALU performs arithmetic operations such as addition and subtraction and also multiplication via repeated addition and division via repeated subtraction. It also performs logic operations such as AND, OR, NOT, and binary shifts as well.
- The FPU is reserved for performing floating-point operations.

Summary of stages

Each computer's CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

- **Fetch Stage:** The next instruction is fetched from the memory address that is currently stored in the PC and stored into the instruction register. At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.
- **Decode Stage:** During this stage, the encoded instruction presented in the instruction register is interpreted by the decoder.
- **Read the effective address:** In the case of a memory instruction (direct or indirect), the execution phase will be during the next clock pulse.
 - ✓ If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers (clock pulse: T_3).
 - ✓ If the instruction is direct, nothing is done during this clock pulse.
 - ✓ If this is an I/O instruction or a register instruction, the operation is performed during the clock pulse.

- **Execute Stage:**

The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant functional units of the CPU to perform the actions required by the instruction, such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register.

- **Repeat Cycle:**

In addition, on most processors interrupts can occur. This will cause the CPU to jump to an interrupt service routine, execute that and then return. In some cases an instruction can be interrupted in the middle, the instruction will have no effect, but will be re-executed after return from the interrupt.

- **Initiation:**

The cycle begins as soon as power is applied to the system, with an initial PC value that is predefined by the system's architecture (for instance, in Intel IA-32 CPUs, the predefined PC value is 0xffffffff).

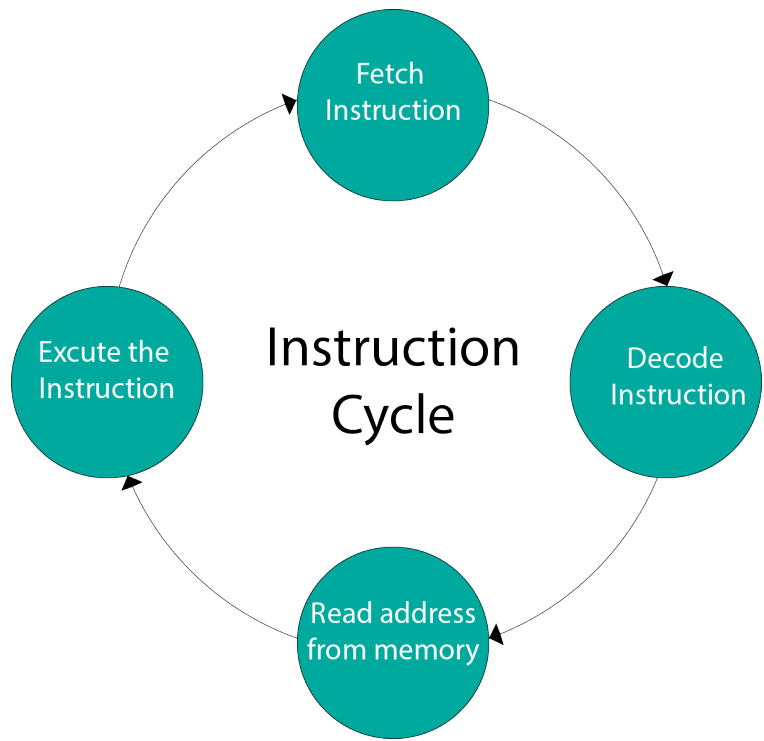
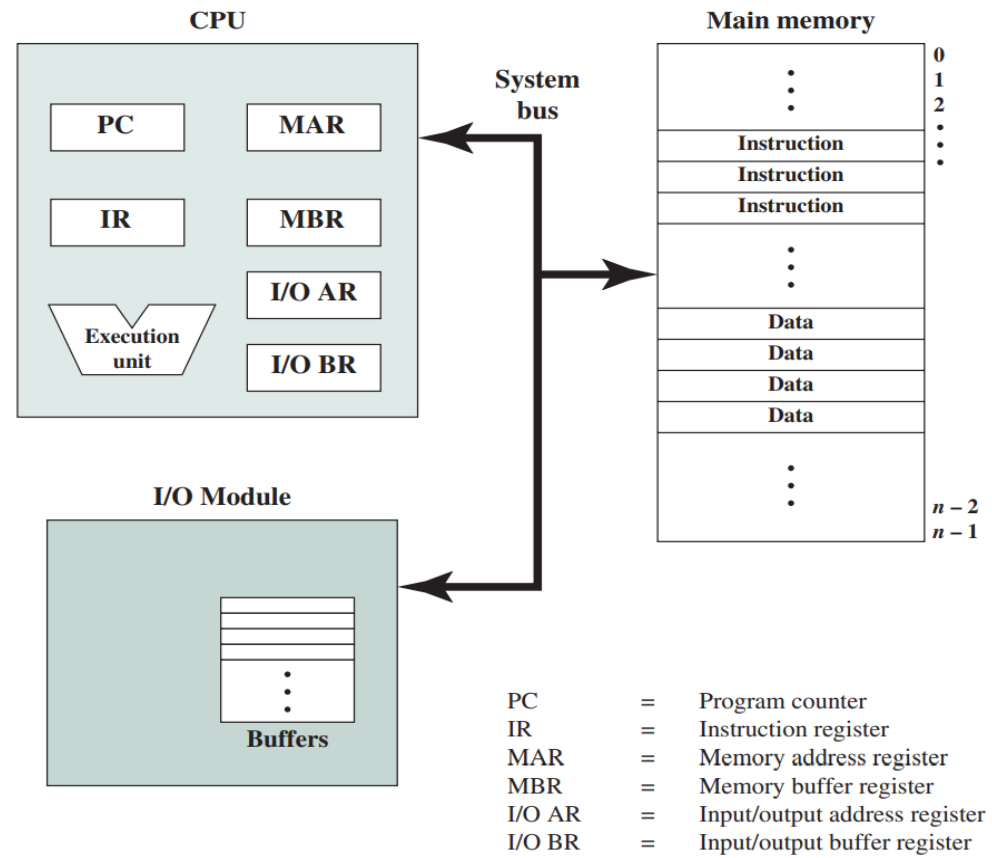
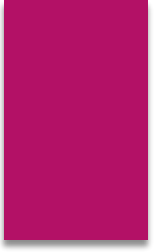
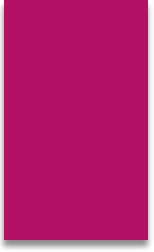


Fig 6

Fig 7



- 
- Instruction processing consists of two steps: The processor reads (fetches) instructions from memory one at a time and executes each instruction.
 - Program execution consists of repeating the process of instruction fetch and instruction execution.
 - The instruction execution may involve several operations and depends on the nature of the instruction.
 - The processing required for a single instruction is called an **instruction cycle**.
 - Using the simplified two-step description given previously, the instruction cycle is depicted in Fig 7. The two steps are referred to as the **fetch cycle** and the **execute cycle**. Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.



Prerequisite – Execution, Stages and Throughput

Registers involved in each Instruction Cycle:

1. **Memory address registers(MAR)** : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
2. **Memory Buffer Register(MBR)** : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
3. **Program Counter(PC)** : Holds the address of the next instruction to be fetched.
4. **Instruction Register(IR)** : Holds the last instruction fetched.

The Instruction Cycle –

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations. In the above examples, there is one sequence each for the *Fetch*, *Indirect*, *Execute* and *Interrupt Cycles*.

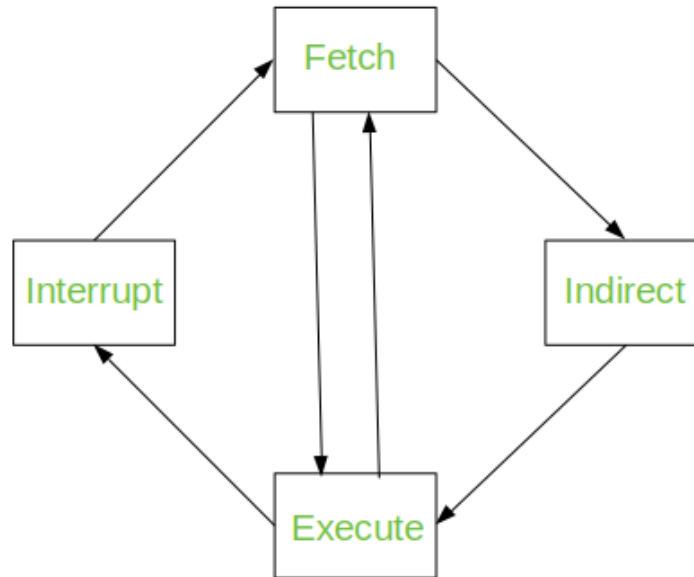


Fig 8 The Instruction Cycle

The *Indirect Cycle* is always followed by the *Execute Cycle*. The *Interrupt Cycle* is always followed by the *Fetch Cycle*. For both fetch and execute cycles, the next cycle depends on the state of the system.

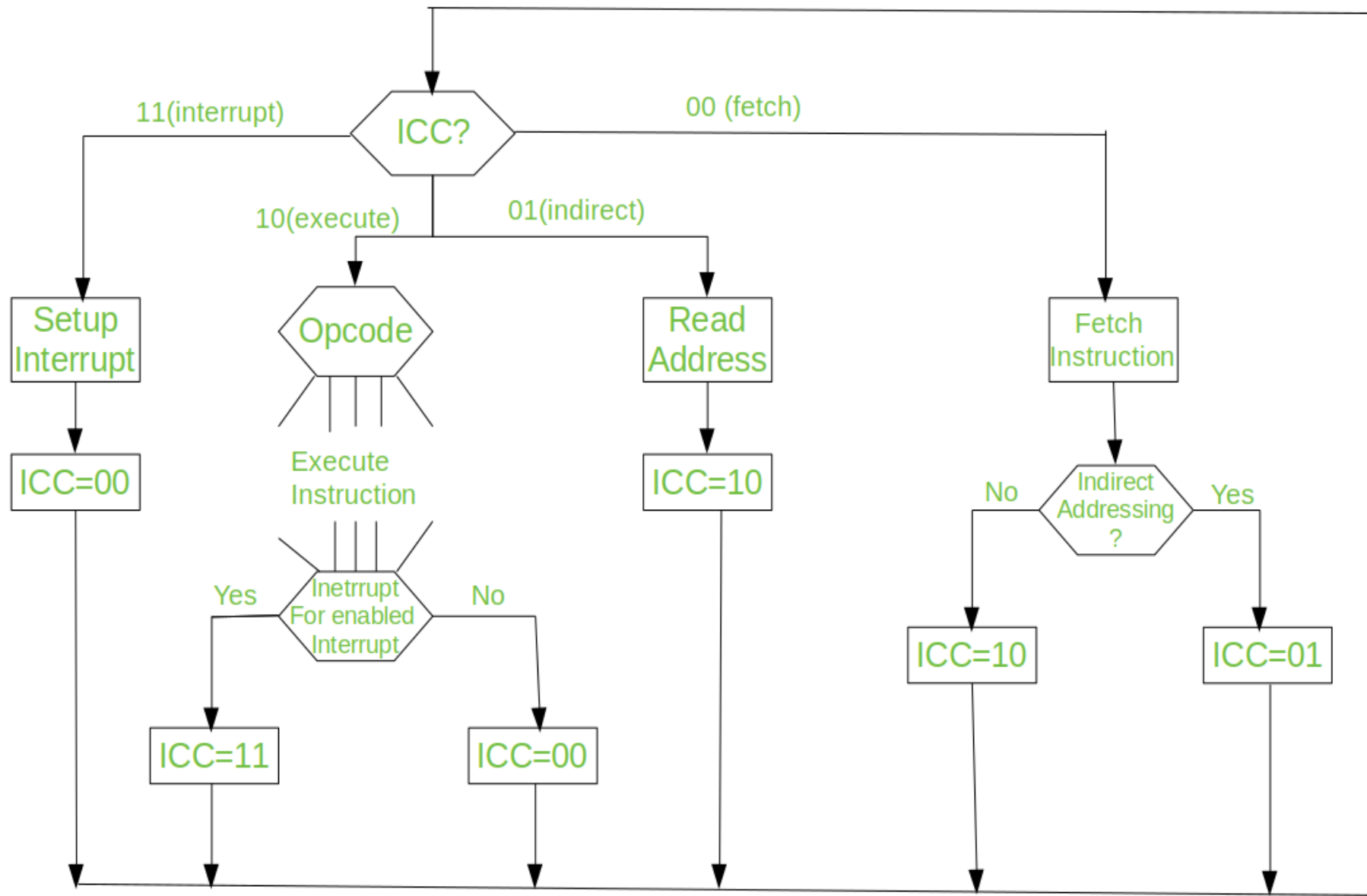
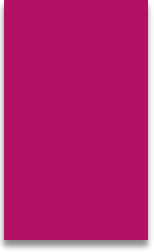


Fig.9 Flowchart for Instruction Cycle



We assumed a new 2-bit register called *Instruction Cycle Code (ICC)*. The ICC designates the state of processor in terms of which portion of the cycle it is in:-

00 : Fetch Cycle

01 : Indirect Cycle

10 : Execute Cycle

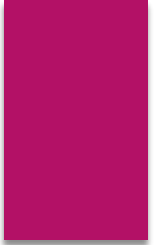
11 : Interrupt Cycle

At the end of the each cycles, the ICC is set appropriately. The above flowchart of *Instruction Cycle* describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern(this is a simplified example).

Different Instruction Cycles:

➤ **The Fetch Cycle –**

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter(PC)*.



MAR	
MBR	
PC	0000000001100100
IR	
AC	

BEGINNING

- **Step 1:** The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.

MAR	0000000001100100
MBR	
PC	0000000001100100
IR	
AC	

FIRST STEP

- **Step 2:** The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction. (These two action can be performed simultaneously to save time).

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	
AC	

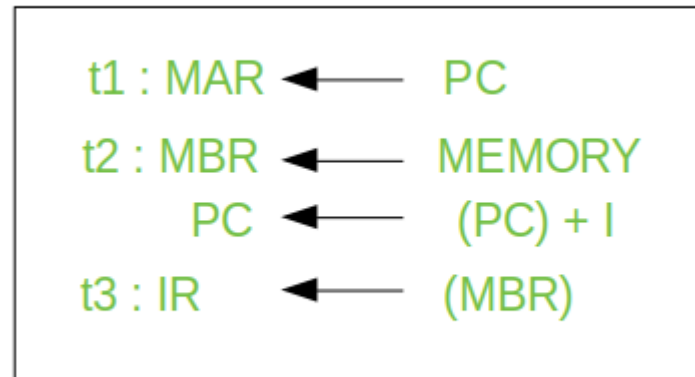
SECOND STEP

- **Step 3:** The content of the MBR is moved to the instruction register(IR).

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100100
IR	0001000000100000
AC	

| THIRD STEP

- Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



- Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses.

Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.

- ✓ First time unit: Move the contents of the PC to MAR.
- ✓ Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by 1.
- ✓ Third time unit: Move contents of MBR to IR.

Note: Second and third micro-operations both take place during the second time unit.

➤ **The Indirect Cycles** – Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing. Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-

t1 : MAR	← (IR(ADDRESS))
t2 : MBR	← MEMORY
t3 : IR(ADDRESS)	← (MBR(ADDRESS))

Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.

Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing).

Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

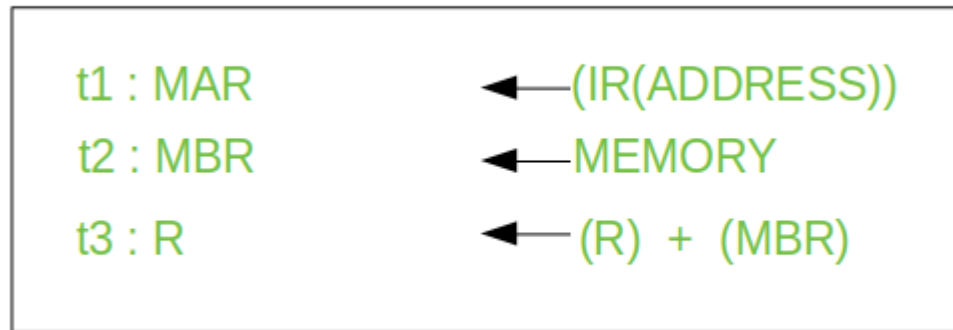
Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the *Interrupt Cycle* .

➤ **The Execute Cycle** - Execute Cycle is different from the other cycles. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.

➤ Lets take an hypothetical example :-
consider an add instruction:

ADD R , X

- Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-



- We begin with the IR containing the ADD instruction.

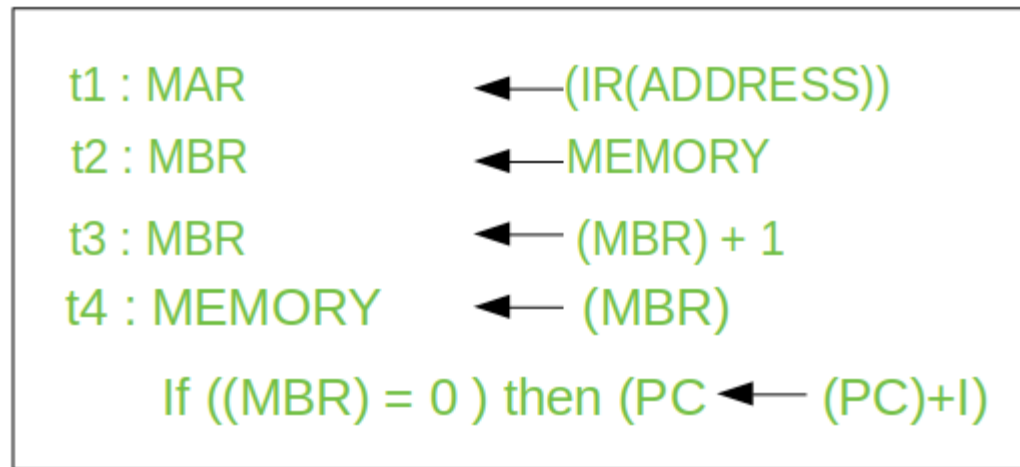
Step 1: The address portion of IR is loaded into the MAR.

Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.

Step 3: Now, the contents of R and MBR are added by the ALU. Lets take a complex example :-



- Here, the content of location X is incremented by 1. If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-

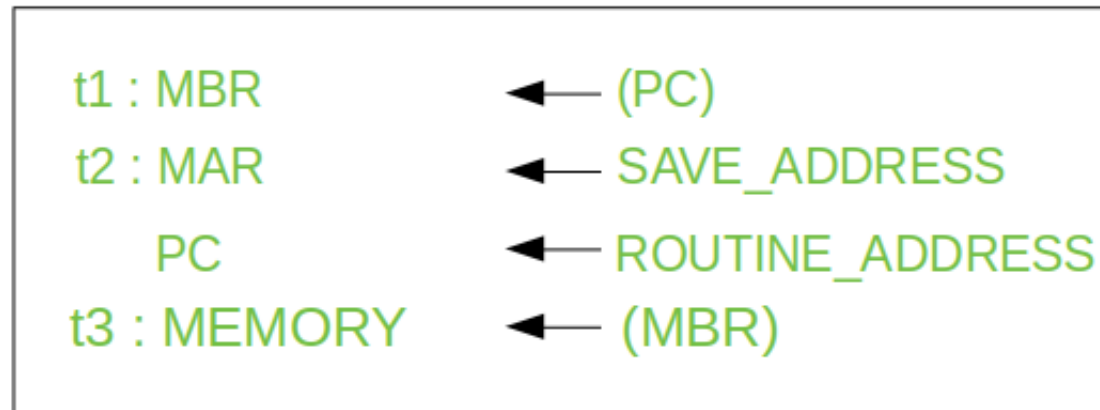


- Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.

➤ The Interrupt Cycle:

At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another.

Lets take a sequence of micro-operation:-



Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.

Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.

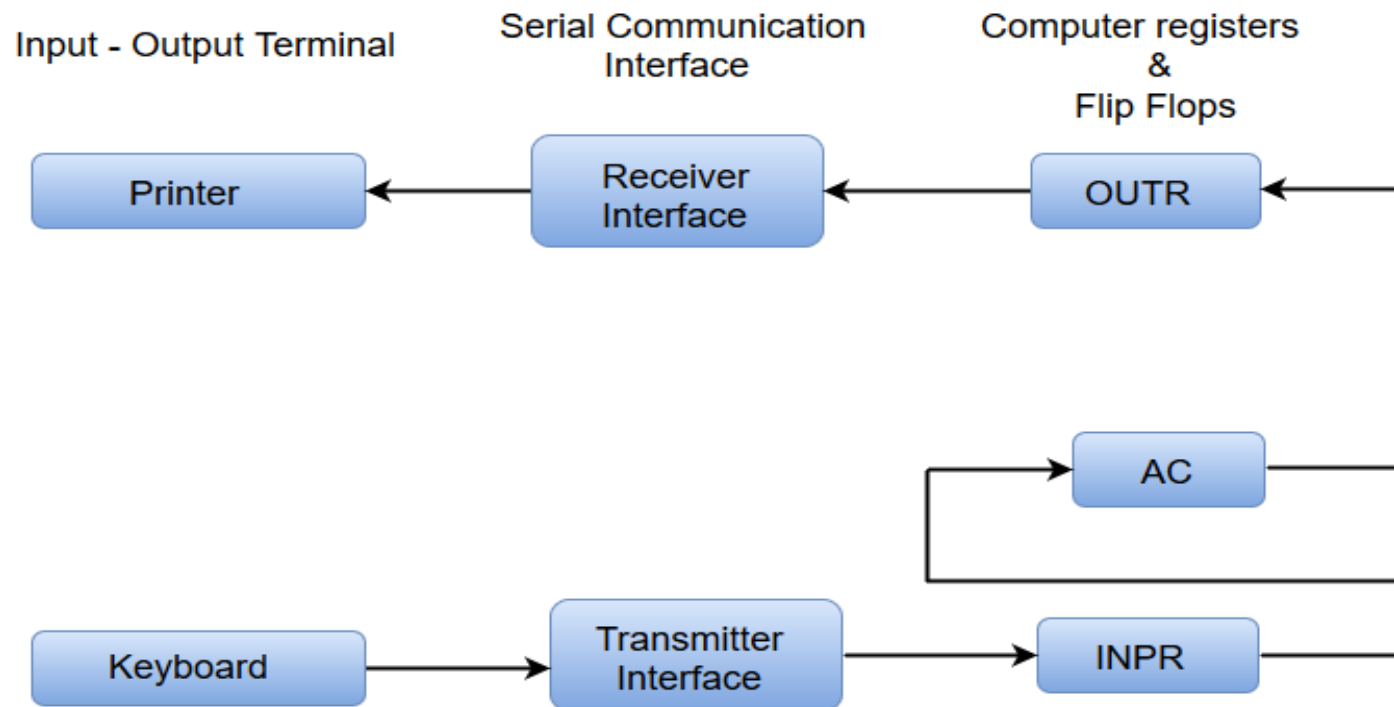
Step 3: MBR, containing the old value of PC, is stored in memory.

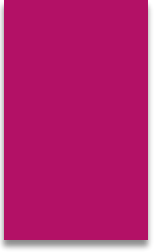
Input-Output Configuration

In computer architecture, **input-output devices act as an interface between the machine and the user.**

Instructions and data stored in the memory must come from some input device. The results are displayed to the user through some output device.

Fig.10 Input - Output Configuration:



- 
- The input-output terminals send and receive information.
 - The amount of information transferred will always have eight bits of an alphanumeric code.
 - The information generated through the keyboard is shifted into an input register 'INPR'.
 - The information for the printer is stored in the output register 'OUTR'.
 - Registers INPR and OUTR communicate with a communication interface serially and with the AC in parallel.
 - The transmitter interface receives information from the keyboard and transmits it to INPR.
 - The receiver interface receives information from OUTR and sends it to the printer serially.

CENTRAL PROCESSING UNIT (CPU)

CPU - The part of the computer that performs the bulk of data-processing operations.

Consists of the following features –

- Considered as the brain of the computer
- Performs all types of data processing operations
- Stores data, intermediate results, and instructions(program)
- Controls the operation of all parts of the computer

Three components of CPU -

- Memory or Storage Unit
- Control Unit
- ALU(Arithmetic Logic Unit)

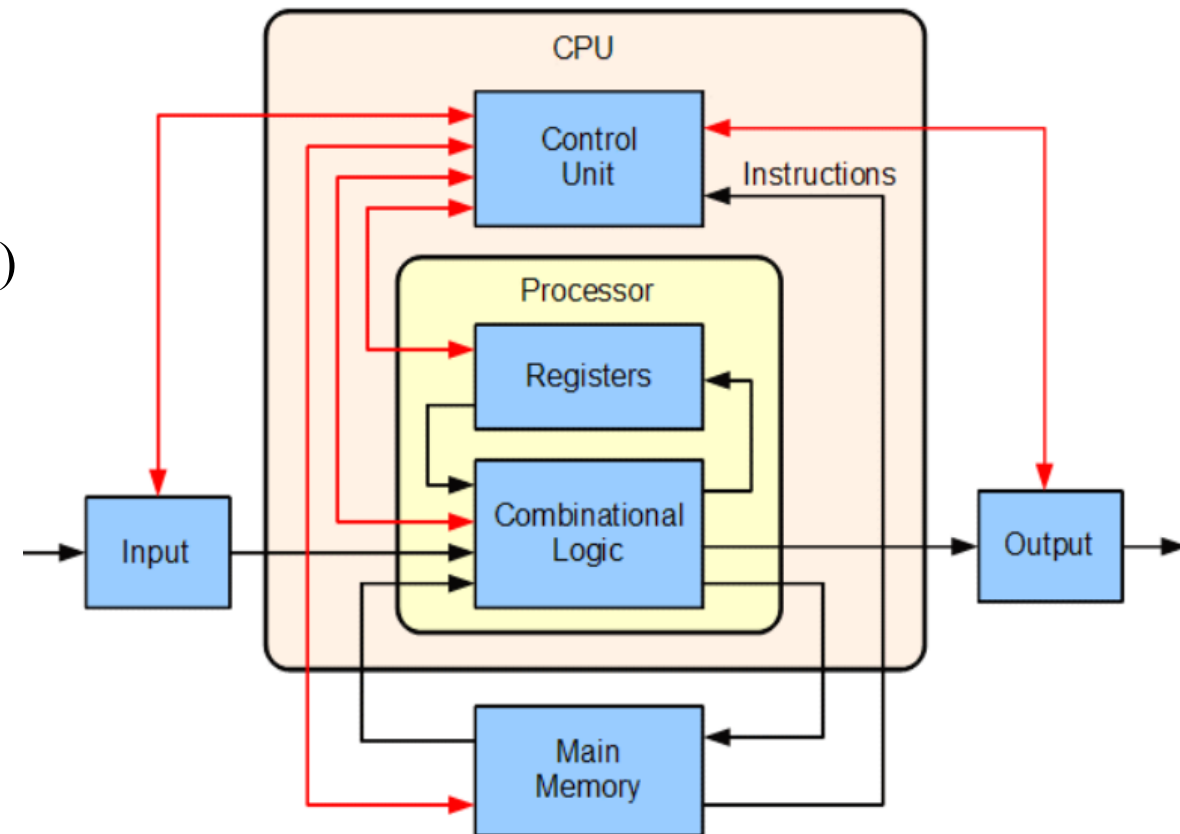


Fig. 11

MEMORY OR STORAGE UNIT

- Supplies information to other units of the computer when needed
- Known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM)
- Its size affects speed, power, and capability
- Types of memories - Primary memory and secondary memory

Functions of the memory unit are –

- Stores all the data and the instructions required for processing
- Stores intermediate results of processing
- Stores the final results of processing before these results are released to an output device
- All inputs and outputs are transmitted through the main memory

CONTROL UNIT

This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations.

Functions of this unit are –

- Responsible for controlling the transfer of data/instructions among other units of a computer.
- Manages and coordinates all the units of the computer.
- Obtains the instructions from the memory, interprets them, and directs the operation of the computer.
- Communicates with Input/Output devices for transfer of data or results from storage.
- Does not process or store data.

ARITHMETIC LOGIC UNIT(ALU)

- ▶ This unit consists of two subsections namely,
 - Arithmetic Section
 - Logic Section
- ▶ **Arithmetic Section-** Function of arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication, and division. All complex operations are done by making repetitive use of the above operations.
- ▶ **Logic Section -**Function of logic section is to perform logic operations such as comparing, selecting, matching, and merging of data.

PROCESSOR ORGANIZATION

To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:

- **Fetch instruction**: The processor reads an instruction from memory (register, cache, main memory).
- **Interpret instruction**: The instruction is decoded to determine what action is required.
- **Fetch data**: The execution of an instruction may require reading data from memory or an I/O module.
- **Process data**: The execution of an instruction may require performing some arithmetic or logical operation on data.
- **Write data**: The results of an execution may require writing data to memory or an I/O module.

- ▶ To do these things, it should be clear that the processor needs to store some data temporarily.
- ▶ It must remember the location of the last instruction so that it can know where to get the next instruction.
- ▶ It needs to store instructions and data temporarily while an instruction is being executed. In other words, the processor needs a small internal memory.

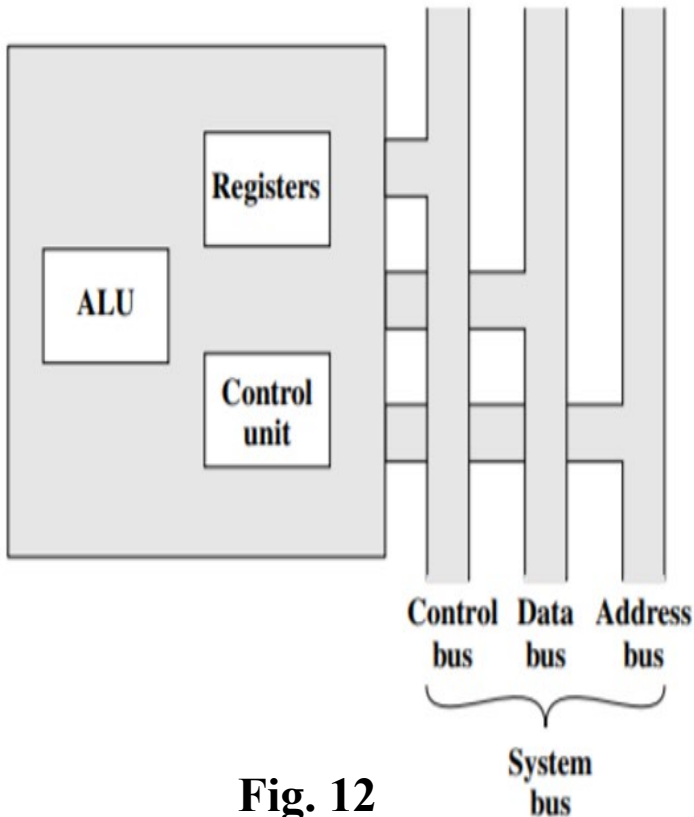


Fig.12 is a simplified view of a processor, indicating its connection to the rest of the system via the system bus.

- The ALU does the actual computation or processing of data.
- The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU.
- In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers.

DESIGN OF CONTROL UNIT

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and the Micro-programmed control unit.

Hardwired Control-

Involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

- A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.
- An instruction fetched from the memory unit is placed in the instruction register (IR).
- The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.
- The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.
- The operation codes from Bits 0 through 11 are applied to the control logic gates.
- The Sequence counter (SC) can count in binary from 0 through 15.

The following image shows the block diagram of a Hardwired Control organization.

Control Unit of a Basic Computer:

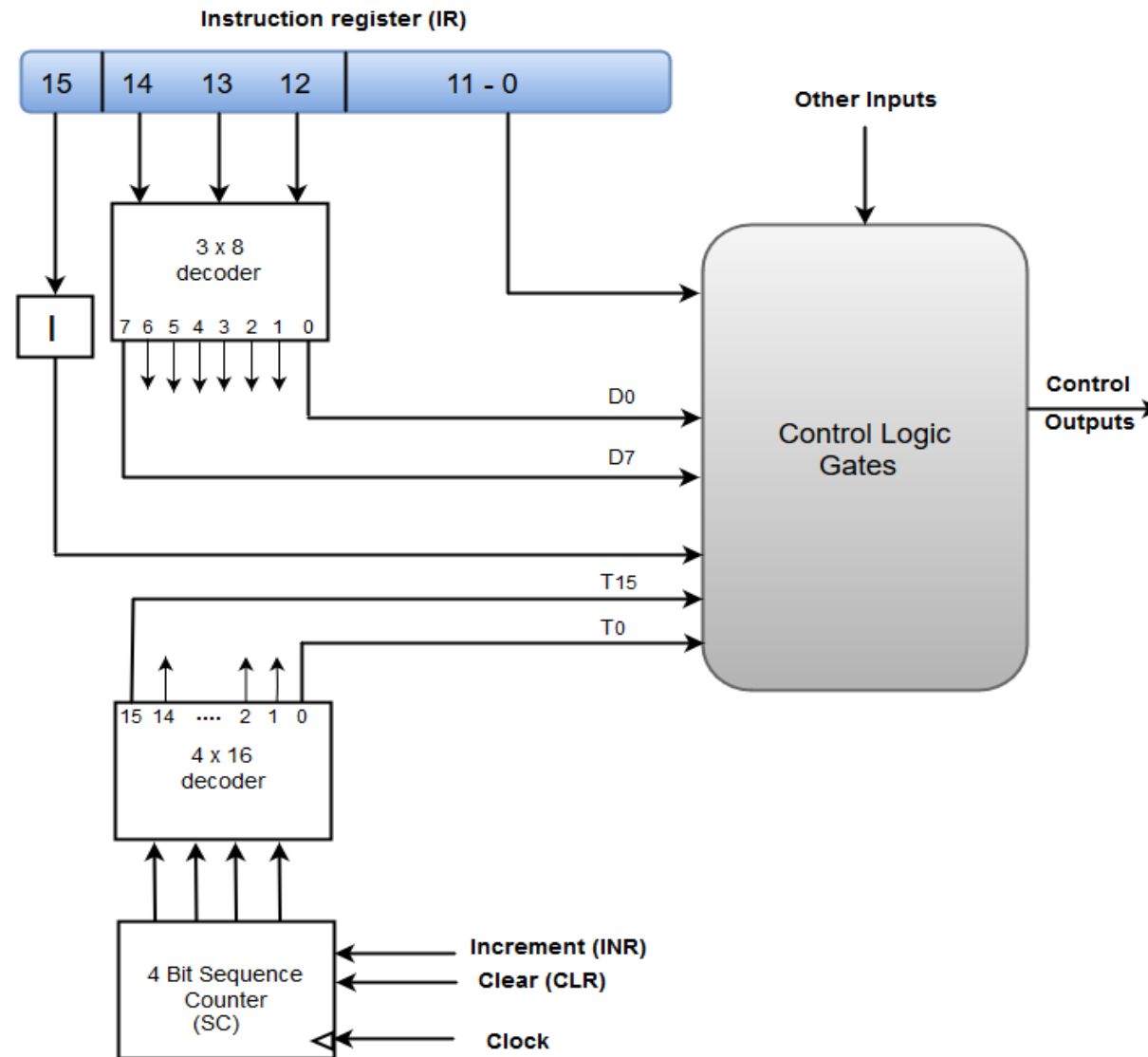


Fig. 13

Micro-programmed Control-

In Microprogrammed Control, the micro-operations are performed by executing a program consisting of micro-instructions.

The following image shows the block diagram of a Microprogrammed Control organization.

Microprogrammed Control Unit of a Basic Computer:

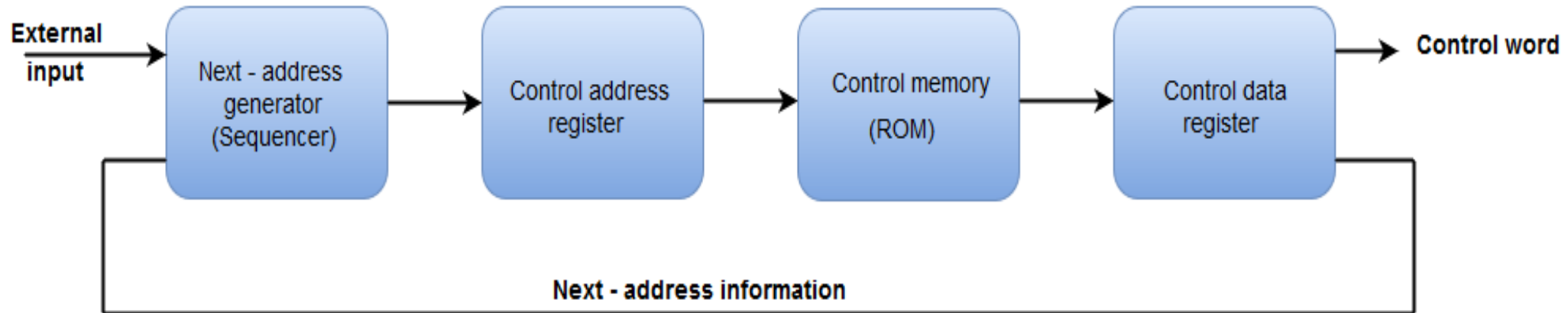
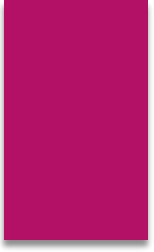


Fig.14

- 
- ❑ The Control memory address register specifies the address of the micro-instruction.
 - ❑ The Control memory is assumed to be a ROM, within which all control information is permanently stored.
 - ❑ The control register holds the microinstruction fetched from the memory.
 - ❑ The micro-instruction contains a control word that specifies one or more micro-operations for the data processor.
 - ❑ While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
 - ❑ The next address generator is often referred to as a micro-program sequencer, as it determines the address sequence that is read from control memory.

COMPUTER REGISTERS

- ▶ Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.
- ▶ A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

Special Purpose Registers

- Special purpose registers hold the status of a program.
- These registers are designated for a special purpose.
- Some of these registers are stack pointer, program counter etc.

Let us see some of the special purpose registers used in CPUs.

Instruction registers

- The Instruction Register (IR) stores the instruction currently being executed.
- In simple processors each instruction to be executed is loaded into the instruction register which holds it while it is decoded, prepared and ultimately executed.

Status Registers

- A Status register or flag register or condition code register is a collection of status flags based on a processor.
- Status register is also a hardware register that contains the information about the state of the processor.
- This register has a size of 16 bits with each bit having a flag.
- Status register is used in different conditions like if the result is negative, result is zero etc.

Shift Registers

A Shift Register is a kind of sequential logic circuit that have ability of both storing and transferring data, made up of flip-flops and connected in such manner that the output of one flip-flop will work as the input of other flip-flop (depending upon which type of shift register is used).

There are six types of shift register which are as follows –

- **Serial in - Serial out Shift Register** – It streams-in data serially one after the other and streams-out in similar manner.
- **Serial in - Parallel Out Shift Register** – It converts the data in serial manner and out the data in parallel manner.
- **Parallel in - Serial Out Shift Register** – It takes data in parallel and streamsout in serial manner (one after other).
- **Parallel in - Parallel Out Shift Register** – Input data fed-in parallel and output data is streams-out simultaneously parallel.

- **Bidirectional Shift Register** – This shift register can perform either right or left data shift or could perform in both directions.
- **Counters** – It feedbacks their output into the device as input in such a manner that creates a particular pattern or sequence.

Accumulator Registers

- It is used for storing the results that are produced by the system.
- When the CPU gives the results after the executing then all the results are stored into the AC Register.

Memory Address Registers

- It is used to hold the memory addresses of data and instructions.
- It accesses data and instructions from memory during the execution phase of an instruction.

Program Counter

It contains the address of the next instruction to be executed. In other words, it holds the address of the memory location of the next instruction when the current instruction is executed by the microprocessor.

General Purpose Registers

These are used to store temporary data within the microprocessor.

- There are 8 general purpose registers in 8086 microprocessor -
 1. **AX** – This is the accumulator of 16 bits and is divided into two 8-bit registers AH and AL to perform 8-bit instructions.
 - It is generally used for arithmetical and logical instructions but in 8086 microprocessor it is not mandatory to have accumulator as the destination operand.

2. BX – This is the base register of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions.

➤ It is used to store the value of the offset.

3. CX – This is the counter register of 16 bits and is divided into two 8-bit registers CH and CL to also perform 8-bit instructions.

➤ It is used in looping and rotation.

4. DX – This is the data register of 16 bits and is divided into two 8-bit registers DH and DL to also perform 8-bit instructions.

➤ It is used in multiplication and input/output port addressing.

5. SP – This is the stack pointer of 16 bits which points to the topmost item of the stack.

If the stack is empty the stack pointer will be (FFFE)H ,i.e. offset address relative to stack segment.



6. **BP** – This is the base pointer of 16 bits.

- It is primarily used in accessing parameters passed by the stack. It's offset address relative to stack segment.

7. **SI** – This is the source index register of 16 bits.

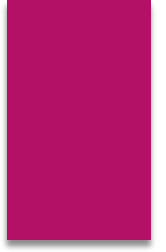
- It is used in the pointer addressing of data and as a source in some string related operations. It's offset is relative to data segment.

8. **DI** – This is the destination index register. It is of 16 bits.

- It is used in the pointer addressing of data and as a destination in some string related operations. It's offset is relative to extra segment.

References

- ❖ M.M.Mano, Computer System Architecture, PHI
- ❖ W.Stallings, Computer Organization and Architecture: Designing for Performance, Prentice Hall
- ❖ <https://www.geeksforgeeks.org>
- ❖ <https://www.tutorialspoint.com>
- ❖ <https://www.Wikipedia.org>
- ❖ <https://www.javatpoint.com>
- ❖ <https://www.britannica.com>
- ❖ <https://www.testbook.com>
- ❖ V. Rajaraman, T. Radhakrishnan, An Introduction to Digital Computer Design, PHI



- Thank you for Patience