Welcome to the wonderful world of Computer Organization & Architecture

Basic Computer Organization & Design Unit 1

Introduction to Computer System

What is Computer

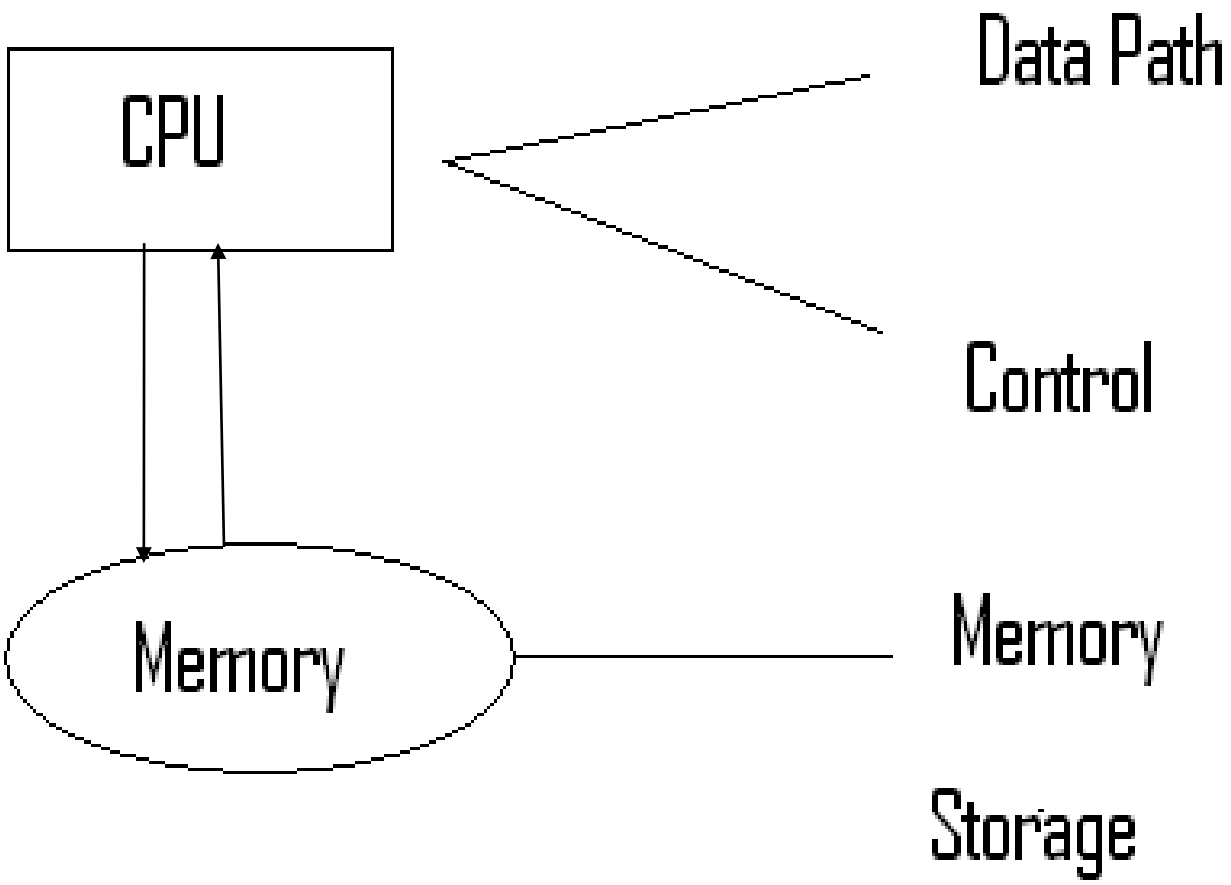
Computer means: compute

- *Points to be noted in mind*
  - The story begins from numbers
  - Computations on these numbers.
  - Basically Computers are essential used for processing
  - Computers are also widely used for communication purpose.

In any computer system we are considering processor and memory , essentially these two elements are there in any computer system.

- Processor for carrying out the processing
- Memory essentially to stores the numbers

- Along with that we are having I/O
  - I/O is an extended memory.
  - Another aspects of the I/O is the interaction of user with the system



Explanation

- We have the data stored in the memory and CPU is capable of processing.
- *What Cpu is to do ?*
  - It is keep addressing the memory and it will get the data from the memory then it will carry out some processing.
- Essentially the computer is having 2 main units
  - One for processing
  - One for storage

What the particular processor will consist of

- Processor is the one which is going to deal with the data.
- Processor must be able to setup some kind of a path to handle the data
- So Data path setting is one aspects of the processor.
- Apart from this setting a path for the data to be processed it should also able to carry out the processing through a sequence of control signal.

- The processor will essentially consist of Data Path circuits & control circuits.
- The one which directly interact with the cpu is also called memory.
- The one which indirectly interact with the cpu is called storage.
- Magnetic tape , disk system are all Storage
- RAM & ROM are coming under memory.

I/O

- CPU is either taking the data from the memory or from I/O
- I/O is for man machine interface
- CPU either takes data from the memory or from I/O system that is the real world data
- That's why we can say that I/O can be considered as a extended memory.

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

Points to be noted

- The basic functional units of computer are made of electronics circuit and it works with electrical signal. We provide input to the computer in form of electrical signal and get the output in form of electrical signal.
- There are two basic types of electrical signals namely, **analog** and **digital**. The analog signals are continuous in nature and digital signals are discrete in nature.
- The electronic device that works with continuous signals is known as **analog device** and the electronic device that works with discrete signals is known as **digital device**.

- Computer is a digital device, which works on two levels of signal. We say these two levels of signal as **High** and **Low**.
- The High-level signal basically corresponds to some high-level signal (say 5 Volt or 12 Volt) and Low-level signal basically corresponds to Low level signal (say 0 Volt).
- Since Computer is a digital electronic device, we have to deal with two kinds of electrical signals. But while designing a new computer system or understanding the working principle of computer, it is always difficult to write or work with 0V or 5V.

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

- To make it convenient for understanding, we use some logical value, say,
  LOW (L) - will represent 0V and
  HIGH (H) - will represent 5V
- Computer is used to solve mainly numerical problems. Again it is not convenient to work with symbolic representation. For that purpose we move to numeric representation. In this convention, we use 0 to represent **LOW** and 1 to represent **HIGH**.

0 means LOW
1 means HIGH

- With the symbol 0 and 1, we have a mathematical system, which is knows as **binary number system**. Basically binary number system is used to represent the information and manipulation of information in computer.
- This information is basically strings of 0s and 1s.
- The smallest unit of information that is represented in computer is known as Bit Binary Digit), which is either 0 or 1.
- Four bits together is known as **Nibble**, and Eight bits together is known as **Byte**.

- Computer technology has made incredible improvement in the past half century. In the early part of computer evolution, there were no stored-program computer, the computational power was less and on the top of it the size of the computer was a very huge one.
- Today, a personal computer has more computational power, more main memory, more disk storage, smaller in size and it is available in affordable cost.
- This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design. In this course we will mainly deal with the innovation in computer design.

- The task that the computer designer handles is a complex one: Determine what attributes are important for a new machine, then design a machine to maximize performance while staying within cost constraints.
- This task has many aspects, including instruction set design, functional organization, logic design, and implementation.
- While looking for the task for computer design both the terms computer organization and computer architecture come into picture.

Computer Architecture & Organization

S. No.	Computer Organization	Computer Architecture
1.	Study of computer system from the user's point of view	Study of computer system from the designer's point of view
2.	Eg : Multiplier : It is enough to know the system has a multiplier from the user's point of view(One did not know how it is designed or implemented)	How These things are implemented is in fact the emphasis in the case of architecture study

S. No.	Computer Organization	Computer Architecture
3.	Here user's are called as Programmer's	Here user's are called as designer's
4.	Study of system from software point of view	Study of the system from hardware point of view
5.	organization describes **how** it	Architecture describes **what** the

S. No.	Computer Organization	Computer Architecture
6.	The difference between architecture and organization is best illustrated by a non-computer example. Is the gear lever in a car part of its architecture or organization?	The architecture of a car is simple; it transports you from A to B.
	The gear lever belongs to the car's organization because it implements the function of a car but is not	

S. No.	Computer Architecture	Computer organization
7.	Computer architecture is the architectural attributes like physical address memory , CPU and how they should be made to coordinate with each other keeping the future demands and goals in mind.	Computer organization is how operational attributes are linked together and contribute to realize the architectural specifications

8.	Computer architecture comes before computer organization. It's like building the design and architecture of house takes maximum time and then organization is building house by bricks or by latest technology.	

S. No.	Computer Architecture	Computer organization
9.	**Computer organization** refers to the operational units and their interconnections that realize the architectural specifications. Computer organization is a study of a Computer Architecture.	E.g. Memory, Registers, RAM, ROM, CPU, ALU, 16 bit/ 32 bit/ 64 bit architecture, what different parts makes a computer, etc

Basic Operations Performed By The Computer

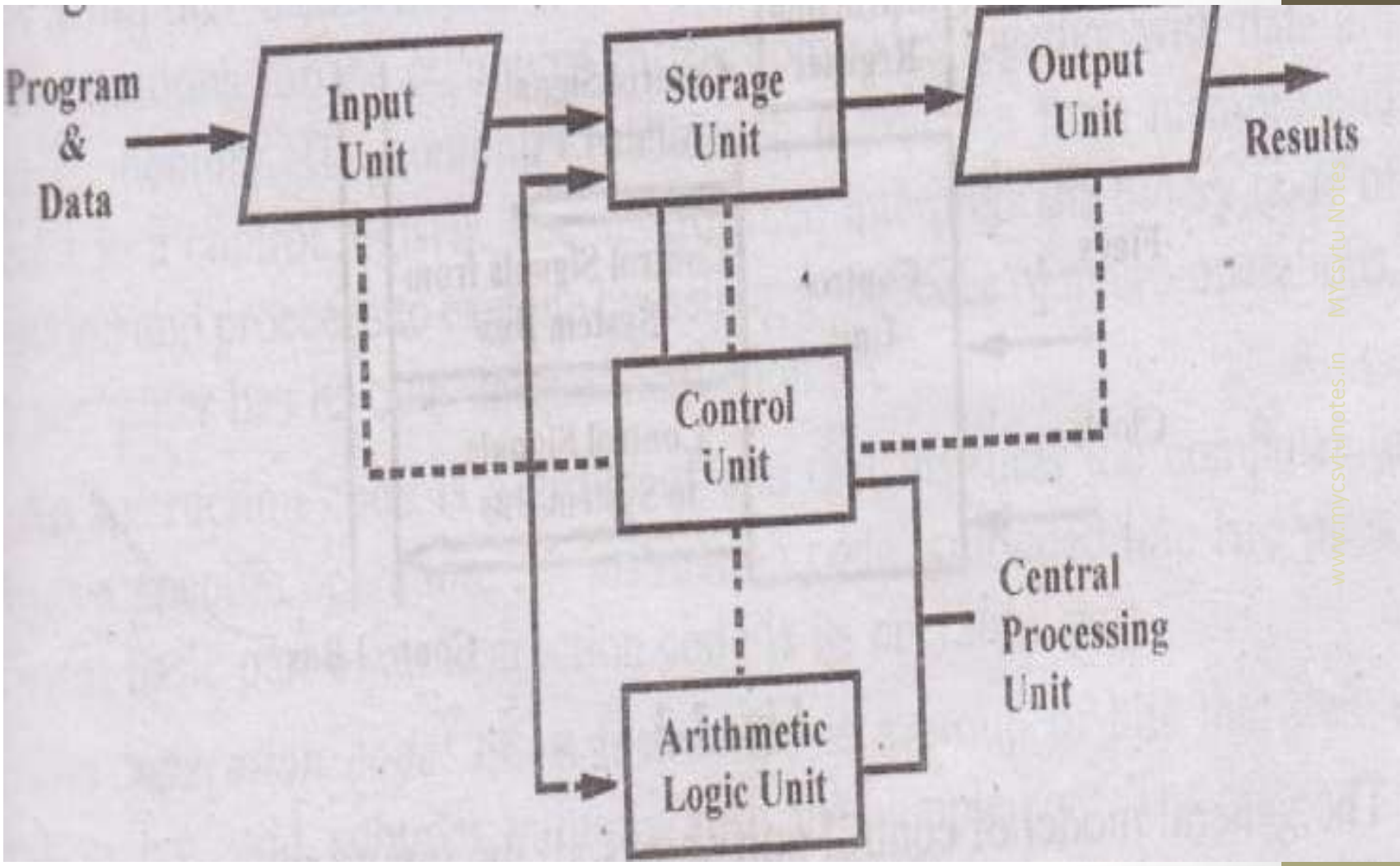
- Inputting : The process of entering data and instructions into the computer system.
- Storing : Saving data and instructions so that they are available for initial or for additional processing as and when required.
- Processing : Performing arithmetic or logical operation (comparison like equal to less then) on data in order to convert them into useful instructions.

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

- Outputting : The process of producing useful information or results for the user, such as a printed report or visual display.
- Controlling : Directing the manner and sequence in which all of the above operations are performed

However the basic operations remains the same for a computer system.

Basic Organization of Computer Syste



Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

- The basic organization of computer system consists of 5 major building blocks or functional unit.
- These 5 units correspond to the five basic operations performed by all computer systems.
- The function of each of these units is described below.

INPUT UNIT :-

- Data and instructions must enter the computer system before any computation can be performed on the supplied data.
- This task is done by the input unit that links the external environment with the computer system.
- For example, data is entered from a keyboard in a manner similar to typing, and this differs from the way in which data is entered through a card reader which is another type of input device.

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

In brief, the following functions are performed by an input unit.

- It accepts (or reads) the list of instructions and data from the outside world.
- It converts these instructions and data in computer acceptable form.
- It supplies the converted instructions and data to the computer system for further processing.

  Example of input devices: Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

Output Unit

- With the help of output unit computer results can be provided to the user or it can be stored in storage device permanently for future use.
- As computer works with binary code, the result provided are also in the binary form.
- Thus before supplying the results to the outside world, they must be converted to human acceptable form.
- This task is done by units called output interfaces.

In brief the following functions are performed by an output unit.

- It accepts the results produced by the computer which are in coded form and hence cannot be easily understood by us.
- It converts these coded results to human acceptable form.
- It supplies the converted results to the outside world

Storage Unit

- The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing start.
- Similarly, the results produced by the computer after processing must also be kept some where inside the computer system before being passed onto the output units.
- Moreover, the intermediate results produced by the computer must also be preserved for ongoing processing.
- The storage unit or the primary storage of a computer system is designed to cater all these needs.
- It provides space for storing data and instructions, space for intermediate results, and also space for the final results.

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

In short the specific functions of the storage unit are to be hold-

- All the data to be processed and the instructions required for processing (received from input devices).
- Intermediate results of processing.
- Final results of processing before these results are released to an output device.

Arithmetic Logic Unit

- The arithmetic logic unit (ALU) of a computer system is the place where the actual execution of the instructions takes place during the processing operation.
- To be more precise, all calculations are done and all comparisons are made in the ALU. The data and instructions, stored in the primary storage prior to processing, are transferred as and when needed to the ALU where processing takes place.
- No processing is done in the primary storage unit. Intermediate results generated in the ALU are temporarily transferred back to the primary storage until needed at a later time.

- Data may thus move from primary is over. After the completion of processing, the final results which are stored in the storage unit are releases to an output device.
- The type and number of arithmetic and logic operations that a computer can perform is determined by the engineering design of the ALU.
- However, almost all ALU's are designed to perform the four basic arithmetic operations- add, subtract, multiply, divide and logic operations or comparisons such as less than, equal to, or greater than.

Control Unit

- By selecting, interpreting and seeing to the execution of the program instructions , the control unit is able to maintain order and direct the operation of the entire system
- Although it does not perform any actual processing on the data, the control unit acts as a central nervous system for the other components of the computer.
- It manages and coordinates the entire computer system.

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

- It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals that cause other units of the system to execute them.

Machine Language

- A program is a list of instructions or statements for directing the computer to perform a required data processing task.
- There are various types of programming languages that one may write for a computer, but the computer can execute programs only when they are represented internally in binary form written in any language must be translated to the binary representation of instruction before they can be executed by the computer.
- Program written for a computer may be one of the following categories

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

- Binary Code : This is a sequence of instructions and operands in binary that list the exact representation of instructions as they appear in the computer memory.
- Octal or Hexadecimal Code : This is an equivalent translation of the binary code to octal or hexadecimal representation.
- Symbolic Code : The user employs symbols(letters, numerals , or special characters) for the operation part, the address part, and other part of the instruction code. Each symbolic instruction can be translated into one binary code instruction. This translation can be done by a special program called an assembler

Because an assembler translates the symbols, this type of symbolic program is referred to as an assembly language program.

- High Level Programming Language :

  These are the special languages developed to reflect the procedures used in the solution of a problem rather than be concerned with the computer hardware behavior.

  An example of high level language is Fortran, it employs problem oriented symbols and formats. The program is written in a sequence of statements in a form that people prefer to think in when solving a problem

However, each statement must be translated into a sequence of binary instructions before the program can be executed in a computer.

The program that translates a high level program to binary is called a compiler.

Diff Between Machine and Assembly

Machine Language	Assembly Language
Machine Language comes in the category of first generation language.	Assembly language comes in the category of second generation language.
Machine language is a language in which every instruction and data should be written by using 0's and 1's .Machine language is also known as computer's native language because this system of codes is directly understood by the computer.	An assembly language provides a mnemonic instruction, usually three letter long, corresponding to each machine instruction. The letter are usually abbreviated indicating what the instruction does.

Machine Language	Assembly Language
Example : To add the contents of register A and register B, the binary code is 10000000 for intel 8085	Example : ADD is used to perform an addition operation, MULT for multiplication etc.
Machine language is the only language that computers can directly execute with out the need for conversion.	A program written in assembly language is less efficient as compared to an equivalent machine language program because every instruction has to be converted into machine language

Machine Language	Assembly Language
Since no conversion is needed, the application developed using machine language are extremely fast.	The execution of assembly language program takes more time than its equivalent machine language program
Machine language is very difficult to read and write. Since all the data and instructions are in binary form, it is almost impossible to remember the instructions.	Assembly language uses mnemonics. Hence the program written in assembly language are much more easier to understand and use as compare to machine language.
It is very difficult to debug and to modify a machine language program	It is comparatively easy to debug and to modify a assembly language program

Basic Working Principle of a Computer

- Before going into the details of working principle of a computer, we will analyze how computers work with the help of a small hypothetical computer.
- In this small computer, we do not consider about Input and Output unit. We will consider only CPU and memory module. Assume that somehow we have stored the program and data into main memory. We will see how CPU can perform the job depending on the program stored in main memory.

Consider the Arithmetic and Logic Unit (ALU) of Central Processing Unit :

- Consider an ALU which can perform four arithmetic operations and four logical operations
- To distinguish between arithmetic and logical operation, we may use a signal line,

  0 - in that signal, represents an arithmetic operation and

  1 - in that signal, represents a logical operation.

The different operations and their binary code is as follows:

Arithmetic		Logical	
000	ADD	100	OR
001	SUB	101	AND
0I0	MULT	110	NAND
0II	DIV	111	NOR

Consider the part of control unit, its task is to generate the appropriate signal at right moment.

- There is an instruction decoder in CPU which decodes this information in such a way that computer can perform the desired task
- The simple model for the decoder may be considered that there is three input lines to the decoder and correspondingly it generates eight output lines. Depending on input combination only one of the output signals will be generated and it is used to indicate the corresponding operation of ALU.

- In our simple model, we use three storage units in CPU,
  Two -- for storing the operand and
  One -- for storing the results.
  These storage units are known as register.
- But in computer, we need more storage space for proper functioning of the Computer.
- Some of them are inside CPU, which are known as register. Other bigger chunk of storage space is known as primary memory or main memory. **The CPU can work with the information available in main memory only**.

Main Memory Organization

- Main memory unit is the storage unit, There are several location for storing information in the main memory module.
- The capacity of a memory module is specified by the number of memory location and the information stored in each location.
- A memory module of capacity 16 X 4 indicates that, there are 16 location in the memory module and in each location, we can store 4 bit of information.

- We have to know how to indicate or point to a specific memory location. This is done by address of the memory location.
- We need two operation to work with memory.
  - **READ** Operation: This operation is to retrieve the data from memory and bring it to CPU register
  - **WRITE** Operation: This operation is to store the data to a memory location from CPU register
- We need some mechanism to distinguish these two operations READ and WRITE.

We need some mechanism to distinguish these two operations READ and WRITE.

- With the help of one signal line, we can differentiate these two operations. If the content of this signal line is
  0, we say that we will do a READ
  operation; and if it is
  1, then it is a WRITE operation
- To transfer the data from CPU to memory module and vice versa, we need some connection. This is termed as **DATA BUS**.

- The size of the data bus indicates how many bit we can transfer at a time. Size of data bus is mainly specified by the data storage capacity of each location of memory module.
- We have to resolve the issues how to specify a particular memory location where we want to store our data or from where we want to retrieve the data.
- This can be done by the memory address. Each location can be specified with the help of a binary address.
- If we use 4 signal lines, we have 16 different combinations in these four lines, provided we use two signal values only (say 0 and 1).

- To distinguish 16 locations, we need four signal lines. These signal lines use to identify a memory location is termed as **ADDRESS BUS**. Size of address bus depends on the memory size.
- As for example, consider a memory module of 16 location and each location can store 4 bit of information
- The size of address bus is 4 bit and the size of the data bus is 4 bit
- The size of address decoder is 4 X 16.
- There is a control signal named R/W.
  If R/W = 0, we perform a READ operation and
- If R/W = 1, we perform a WRITE operation

- If the contents of address bus is 0101 and contents of data bus is 1100 and R/W = 1, then 1100 will be written in location 5.
- If the contents of address bus is 1011 and R/W=0, then the contents of location 1011 will be placed in data bus.

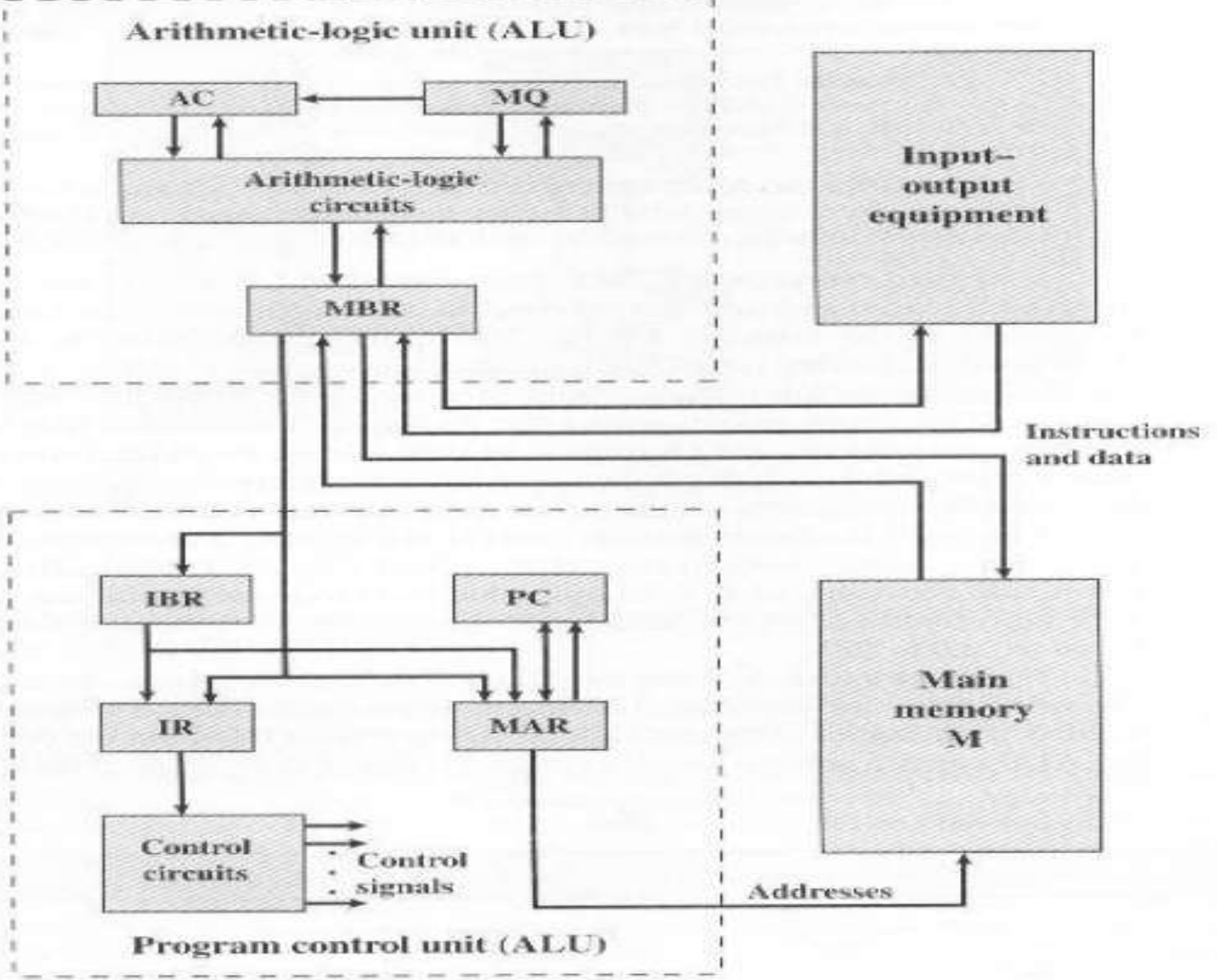


Figure 2.3 Expanded Structure of IAS Computer

To access the data from memory, we need two special registers :

- One is known as **Memory Data Register (MDR) or Memory Buffer Register (MBR).**
- Second one is **Memory Address Register (MAR)**.

Memory Data/Buffer Register

- It holds the instruction code or data received from or sent to the memory.
- It is connected to data bus.
- The data which are written into the memory are held in this register until the write operation is completed.
- It is also called data register (DR)
- Thus the flow of data from the CPU to the memory or from the memory to CPU is always through MBR.
- It is within the CPU.

Memory Address Register

- It hold the address of the instruction or data to be fetched from the memory.
- The CPU transfers the address of the next instruction from the program counter (PC) to the Memory Address Register.
- From MAR it is sent to the memory through the ADDRESS BUS.
- Sometimes it is simply called Address Register(AR).

Program Counter

- The program counter keeps track of the address of the instruction which is to be executed next.
- So it holds the address of the memory location which contains the next instruction to be fetched from the memory.
- Its content is automatically incremented after an instruction has been fetched assuming that instructions are normally executed sequentially.
- In case of jump instruction its contents are modified , and program jumps to the memory location which contains the desired instruction to be executed next.

Instruction Register

- It holds an instruction until it is decoded.
- Some computers have two instruction registers , and so they can fetch and save the next instruction while the execution of the previous instruction is going on.

IBR(Instruction Buffer Register)

- Employed to hold temporarily the right hand instruction from a data in memory

Accumulator (AC) & Multiplier Quotient (MQ)

- Employed to hold temporarily operands and results of the ALU operations.
- For example, the result of multiplying two 40 bit numbers is an 80 bit numbers;
- The most significant 40 bits are stored in the AC and the least significant in the MQ.

Memory Instruction

- We need some more instruction to work with the computer. Apart from the instruction needed to perform task inside CPU, we need some more instructions for data transfer from main memory to CPU and vice versa
- In our hypothetical machine, we use three signal lines to identify a particular instruction. If we want to include more instruction, we need additional signal lines.

Instruction	Code	Meaning
1000	LDAI imm	Load register A with data that is given in the program
1001	LDAA addr	Load register A with data from memory location addr
1010	LDBI imm	Load register B with data
1011	LDBA addr	Load register B with data from memory location addr
1100	STC addr	Store the value of register C in memory location addr
1101	HALT	Stop the execution
1110	NOP	No operation
1111	NOP	No operation

- With this additional signal line, we can go upto 16 instructions. When the signal of this new line is 0, it will indicate the ALU operation.
- For signal value equal to 1, it will indicate 8 new instructions. So, we can design 8 new memory access instructions.
- We have added 6 new instructions. Still two codes are unused, which can be used for other purposes. We show it as **NOP** means No Operation.
- We have seen that for ALU operation, instruction decoder generated the signal for appropriate ALU operation.

- Apart from that we need many more signals for proper functioning of the computer. Therefore, we need a module, which is known as control unit, and it is a part of CPU. The control unit is responsible to generate the appropriate signal.
- As for example, for LDAI instruction, control unit must generate a signal which enables the register A to store in data into register A.
- One major task is to design the control unit to generate the appropriate signal at appropriate time for the proper functioning of the computer.
- Consider a simple problem to add two numbers and store the result in memory, say we want to add 7 to 5.

- To solve this problem in computer, we have to write a computer program. The program is machine specific, and it is related to the instruction set of the machine.
- For our hypothetical machine, the program is as follows

Instruction	Binary	HEX	Memory Location
LDAI 5	1000 0101	8 5	(0, 1)
LDBI 7	1010 0111	A 7	(2, 3)
ADD	0000	0	(4)
STC 15	1100 1111	C F	(5, 6)
HALT	1101	D	(7)

- One question still remain unanswered: How to store the program or data to main memory. Once we put the program and data in main memory, then only CPU can execute the program. For that we need some more instructions.
- We need some instructions to perform the input tasks.
- These instructions are responsible to provide the input data from input devices and store them in main memory.
- For example instructions are needed to take input from keyboard.

- We need some other instructions to perform the output tasks. These instructions are responsible to provide the result to output devices. For example, instructions are needed to send the result to printer.
- We have seen that number of instructions that can be provided in a computer depends on the signal lines that are used to provide the instruction, which is basically the size of the storage devices of the computer.
- For uniformity, we use same size for all storage space, which are known as register. If we work with a 16-bit machine, total instructions that can be implemented is 2^{16}.
- The model that we have described here is known as *Von Neumann Stored Program Concept*.

- First we have to store all the instruction of a program in main memory, and CPU can work with the contents that are stored in main memory. Instructions are executed one after another.
- In 1946, Von Neumann and his colleagues began the design of a stored-program computer at the Institute for Advanced Studies in Princeton. This computer is referred as the IAS computer.

The structure of IAS computer is shown in Figure

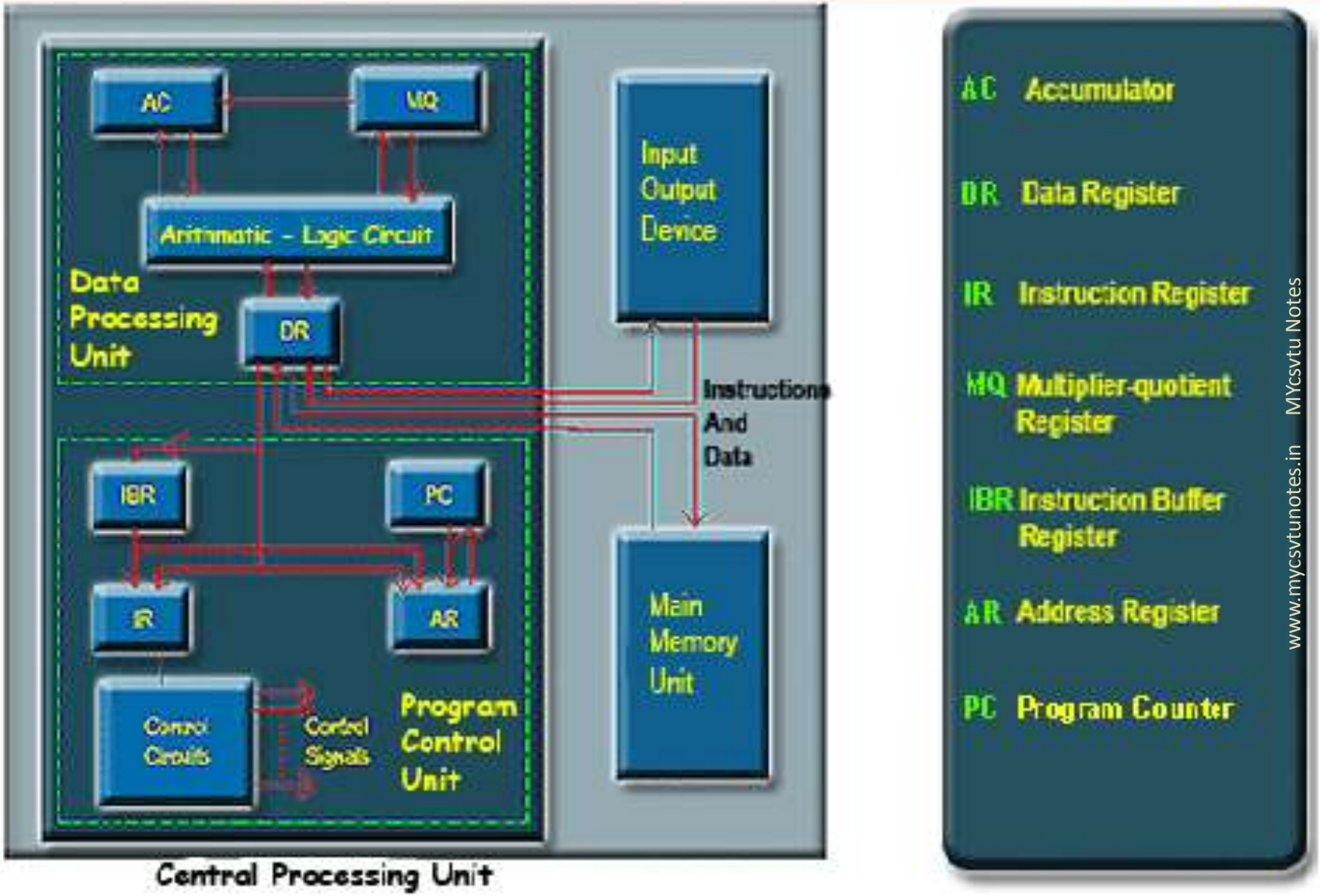


Figure 1.2: Structure of a first generation computer : IAS

Instruction Codes

- The organization of the computer is defined by its internal registers, the timing and control structure and the set of instructions that it uses.
- The internal organization of a digital system is defined by a sequence of micro operations, it performs on data stored in its registers.
- The user of a computer can control the process by means of a program.
- A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur.

A computer instruction is a binary code that specifies a sequence of micro-operations for the computer. Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of instruction and proceeds to execute by issuing a sequence of micro-operations. Each computer has its own unique instruction set.

An instruction code is a group of bits that instructs the computer to perform a specific operation. An instruction code is divided into two parts. The most basic part of an instruction code is its operation part.

The '*operation code*' of an instruction is a group of bits that define operation, e.g., add, subtract, multiply, shift and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

Example – Consider a computer with 64 distinct operations, one of them is an ADD operation. The operation code consists of six bits with a bit configuration 110010 assigned to the ADD operation. When this operation code is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register.

An *operation* is a part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation.

(a) Stored Program Organisation – The easiest way of computer organisation is to have one ***processor register*** and an ***instruction code*** format with two parts. The first part specifies the operation and the second part specifies an address. The memory address tells the control where to find an operand in the memory and then the operand is read from memory and used for further operations. The operation code is abbreviated as ***opcode.***

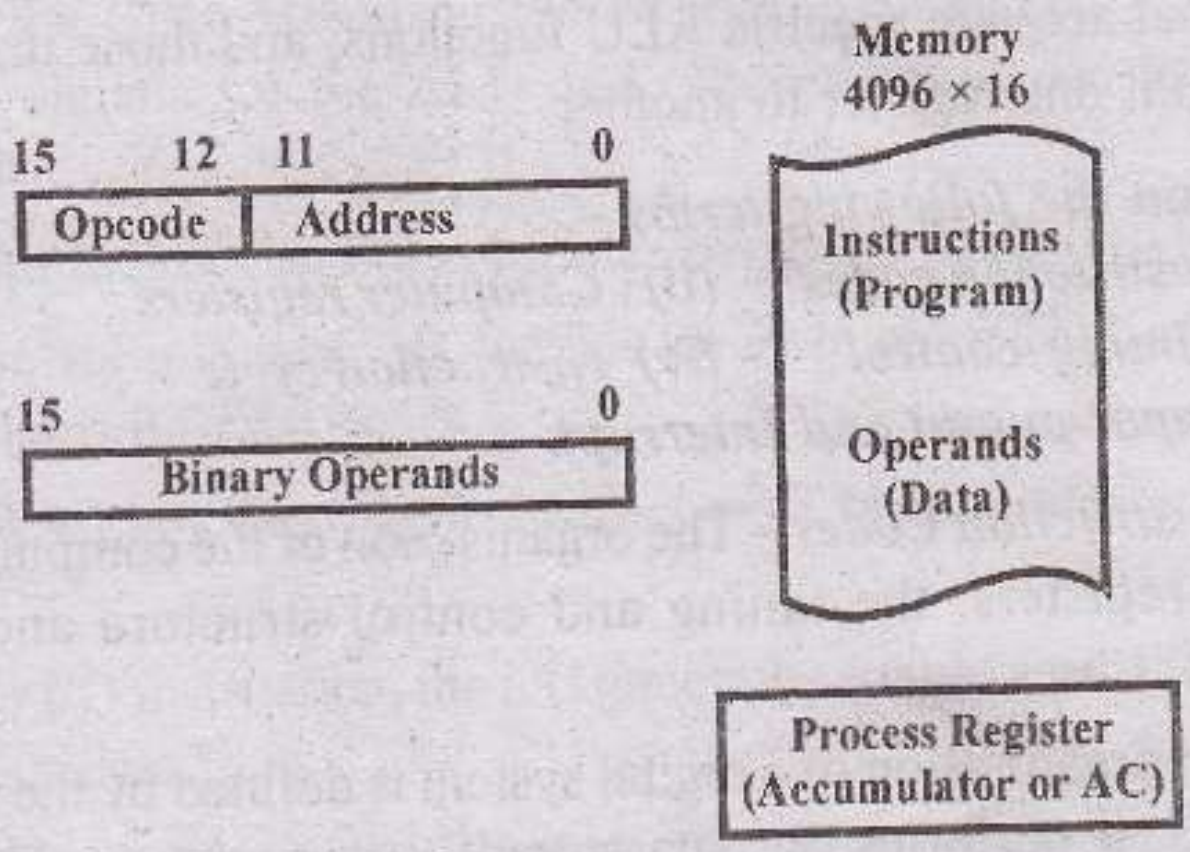


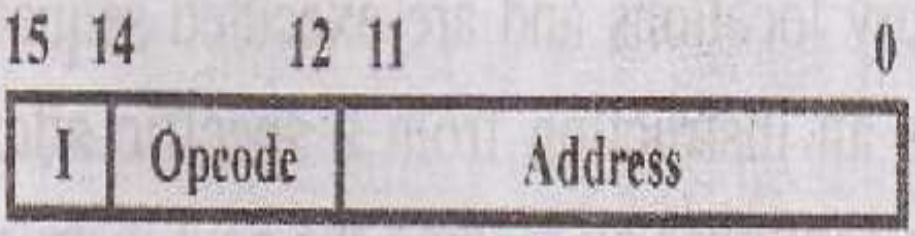
Fig. 3.3 Stored Program Organisation

Fig. 3.3 shows stored program organisation. Instructions are stored in one section of memory and data in another. We require 12 bits to specify an address for memory unit with 4096 words, since $2^{12} = 4096$. If we store each instruction code in one 16 bits memory word, we have available four bits for the operation code to specify one out of 16 possible operations, and 12 bits to specify the address of an operand. The control reads a 16-bits instruction from the program position of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data position of memory. It then executes the operation specified by the opcode.

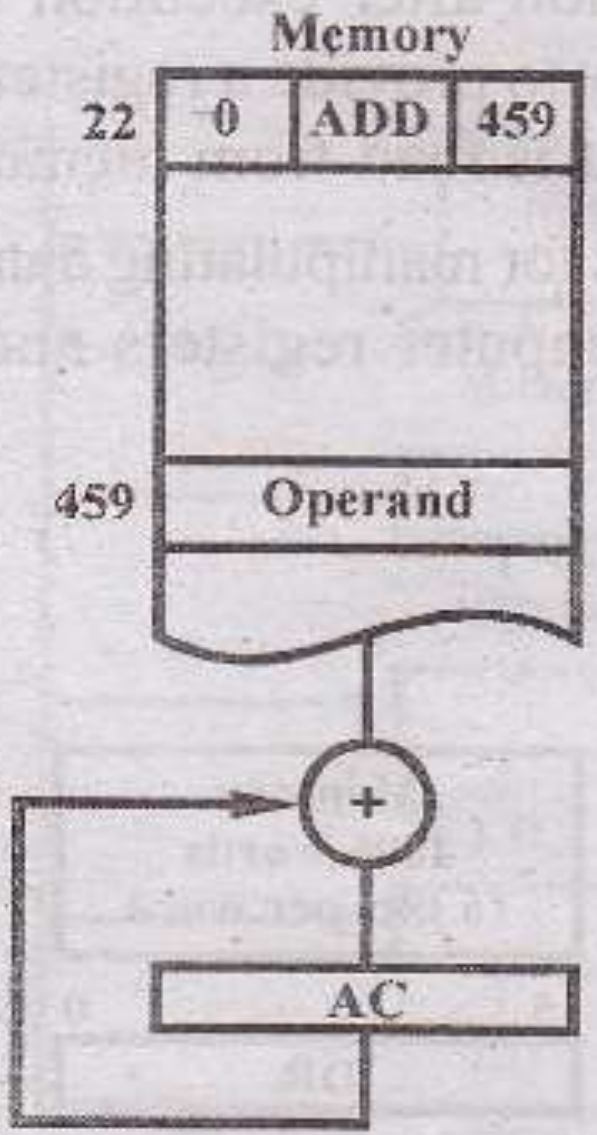
Computers that have a single processor register usually assign to it the name *accumulator* and label it AC.

(b) Direct and Indirect Address – When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand, when the second part specifies the address then it is called

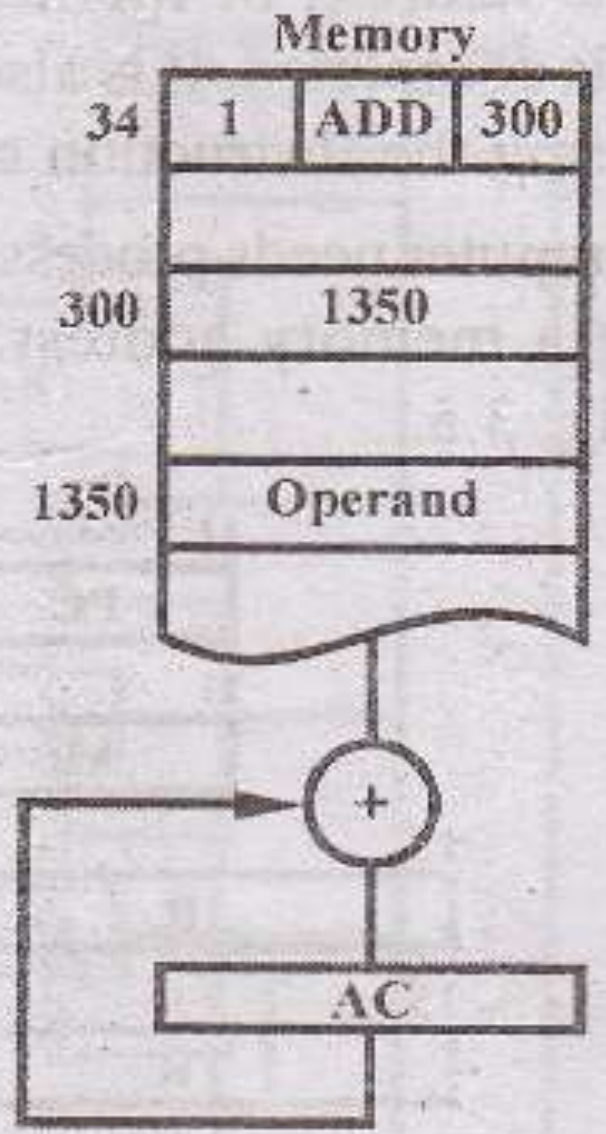
direct address. This is in contrast to a third possibility called indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.



(a) Instruction Format



(b) Direct Address



(c) Indirect Address

Fig. 3.4 Direct and Indirect Address

To understand, consider the instruction code format shown in fig. 3.4(a). It consists of 3-bit operation code, a 12-bit address and an indirect address mode bit designated by I. The mode bit is 0 for direct and 1 for an indirect address.

A ***direct address*** instruction is shown in fig. 3.4 (b). It is placed in address 22 in memory. The I bit is 0 so it is recognised as a direct address instruction. The opcode specifies an ADD instruction and the address part is the binary equivalent 459 and adds it to the content of AC.

An ***indirect address*** instruction is shown in fig. 3.4(c). It is placed in address 34 and has a mode bit I = 1. Therefore, it will be recognised as an indirect address instruction. The address part is the binary equivalent of 300. The control goes to address 300 to find the address of operand. The address of the operand in this case is 1350. The address found in address 1350 is then added to the content of AC.

The indirect address needed two references to memory to fetch an operand. The first reference is needed to read the address of the operand and

the second is for the operand itself.

Thus an effective address in the instruction of fig: 3.4 (b) is 459 and in the instruction of fig 3.4 (c) is 1350

Steps of Instruction Execution Cycle

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

The necessary steps that the processor has to carry out for fetching an instruction from the memory and executing it, constitute an instruction cycle. An instruction cycle consists of two parts – an execute cycle and a fetch cycle.

Steps of Instruction Execution Cycle : The main steps of instruction execution cycle are as follows :

In execute cycle an instruction is executed. The necessary steps which are carried to execute an instruction, constitute an execute cycle.

Step 1 : The opcode which is fetched from the memory is placed first of all in the data register, DR.

Step 2 : Thereafter it goes to the instruction register, IR.

Step 3 : From the instruction register it goes to the decoder circuitry which is within the CPU. The decoder circuitry decodes the opcode.

Step 4 : After the opcode is decoded the CPU comes to know what operation is to be performed, and then execution begins. If the operand is in a general purpose register, the execution is immediately performed. In such a situation the time required for decoding and executing the instruction is only one clock cycle.

Step 5 : If the required data or operand address is still in the memory, the CPU reads them from the memory. For reading data or the operand address from the memory the CPU performs read operation. In a read cycle the quantity received from the memory is data or address instead of opcode.

Step 6 : After receiving data from the memory, CPU performs execute operation.

Step 7 : Some instructions may require write operation. In write cycle data are transferred from the CPU to the memory or an output device.

Thus, an execute cycle may involve one or more read or write cycles or both.

Instruction Cycle (fetch cycle & execute cycle)

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

The necessary steps that the processor has to carry out for fetching an instruction from the memory and executing it, constitute an *instruction cycle*. An instruction cycle consists of two parts- a *fetch cycle* and an *execute cycle*. In fetch cycle the CPU fetches the machine code of the instruction (opcode) from the memory. The necessary steps that are carried out to fetch an opcode from the memory, constitute a *fetch cycle*. In execute cycle an instruction is executed. The necessary steps which are carried to execute an instruction, constitute an execute cycle.

Fetch Operation : To fetch an opcode from a memory location the following steps are performed:

(i) The program counter places the address of the memory location in which the opcode is stored, on the address bus.

(ii) The CPU sends the required memory control signals so as to enable the memory to send the opcode.

(iii) The opcode stored in the memory location is placed on the data bus and transferred to the CPU.

All the above steps require three clock cycles. If memory is slow the time taken may be more. In that case the CPU has to wait for some time till the memory transfers the opcode to the CPU. The extra clock cycles for which the CPU waits are known as *wait cycles*. Most of the microprocessors have circuitry to introduce wait cycles to cope with slow memories.

The opcode which is fetched from the

wait cycles to cope with slow memories.

Execute Operation : The opcode which is fetched from the memory is placed first of all in the data register, DR (data/address buffer in case of Intel 8085). Thereafter is goes to the instruction register, IR. From the instruction register it goes to the decoder circuitry which is with in the CPU. The decoder circuitry decodes the opcode. After the opcode is decoded the CPU comes to know what operation is to be performed, and then execution begins. If the operand is in a general purpose register, the execution is immediately performed. In such a situation the time required for decoding the executing the instruction is only one clock cycle. If the required data or operand address is still in the memory, the CPU reads them from the memory. For reading data or the operand address from the memory the CPU performs read operation. The read cycle is similar to an opcode fetch cycle. In a read cycle the quantity received from the memory, CPU performs execute operation. Some instructions may require write operation. In write cycle data are transferred from the CPU to the memory or an output device. Thus an execute cycle may involve one or more read write cycles or both.

steps which are carried out to

Stack Organization

A useful feature that is included in the CPU of most computers is a stack or last-in, first-out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.

The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (*SP*) because its value always points at the top item in the stack. Contrary to a stack of trays where the tray itself may be taken out or inserted, the physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

The two operations of a stack are the insertion and deletion of items. The operation of insertion is called *push* (or push-down) because it can be thought of as the result of pushing a new item on top. The operation of deletion is called *pop* (or pop-up) because it can be thought of as the result of removing one item so that the stack pops up. However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register Stack

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 8-3 shows the organization of a 64-word register stack. The stack pointer register *SP* contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: *A*, *B*, and *C*, in that order. Item *C* is on top of the stack so that the content of *SP* is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of *SP*. Item *B* is now on top of the stack since *SP* holds address 2. To insert a new item, the stack is pushed by incrementing *SP* and writing a word in the next-higher location in the stack. Note that item *C* has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place.

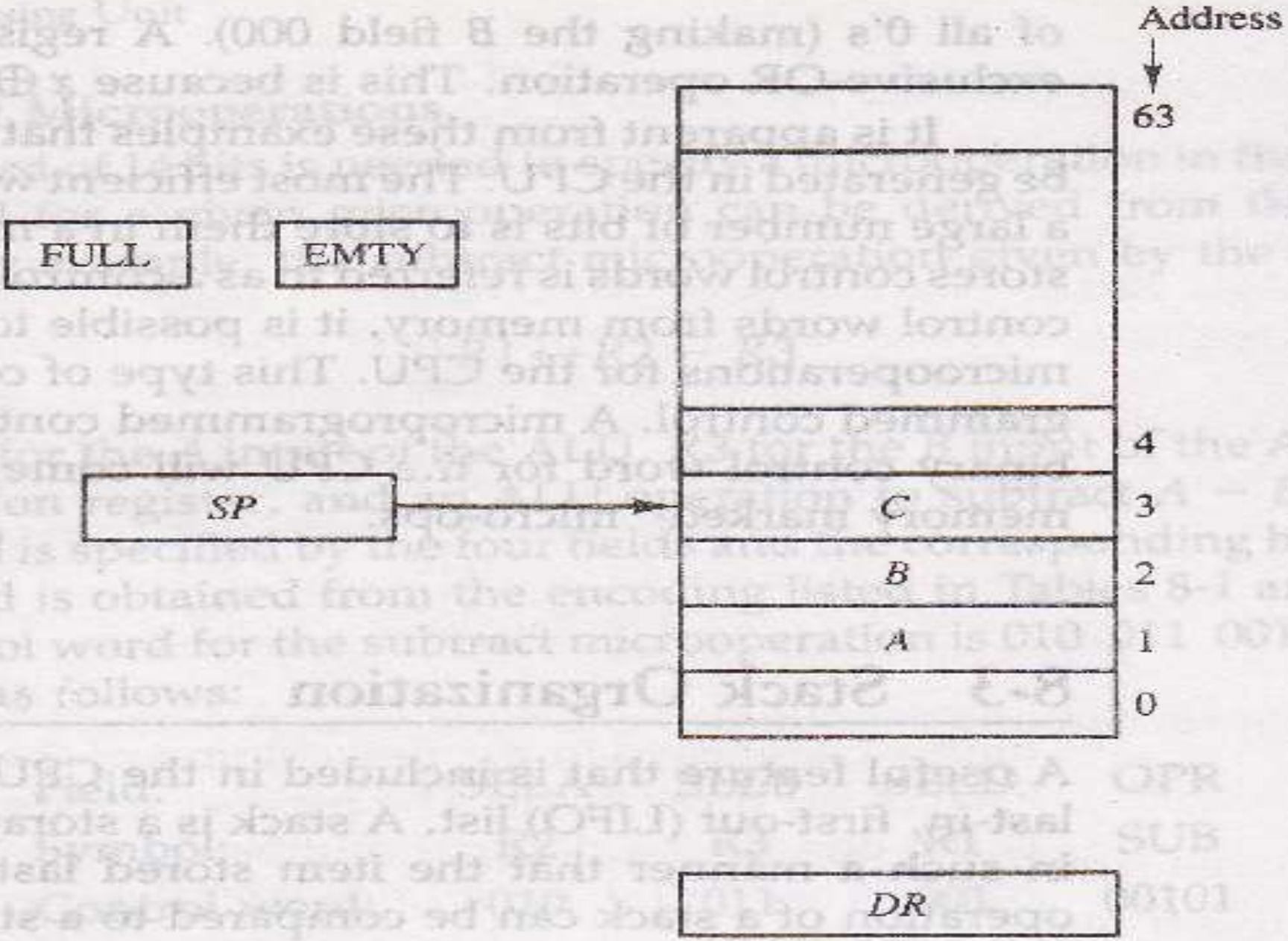


Figure 8-3 Block diagram of a 64-word stack.

In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$. Since *SP* has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but *SP* can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 111111. The one-bit register *FULL* is set to 1 when the stack is full, and the one-bit register *EMTY* is set to 1 when the stack is empty of items. *DR* is the data register that holds the binary data to be written into or read out of the stack.

Initially, *SP* is cleared to 0, *EMTY* is set to 1, and *FULL* is cleared to 0, so that *SP* points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if *FULL* = 0), a new item is inserted with a push operation. The push operation is implemented with the following sequence of microoperations:

$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack

If $(SP = 0)$ then $(FULL \leftarrow 1)$	Check if stack is full
$EMTY \leftarrow 0$	Mark the stack not empty

The stack pointer is incremented so that it points to the address of the next-higher word. A memory write operation inserts the word from *DR* into the top of the stack. Note that *SP* holds the address of the top of the stack and that *M*[*SP*] denotes the memory word specified by the address presently available in *SP*. The first item stored in the stack is at address 1. The last item is stored at address 0. If *SP* reaches 0, the stack is full of items, so *FULL* is set to 1. This condition is reached if the top item prior to the last push was in location 63 and, after incrementing *SP*, the last item is stored in location 0. Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so *EMTY* is cleared to 0.

A new item is deleted from the stack if the stack is not empty (if *EMTY* = 0). The pop operation consists of the following sequence of micro-operations:

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

The top item is read from the stack into *DR*. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so *EMTY* is set to 1. This condition is reached if the item read was in location 1. Once this item is read out, *SP* is decremented and reaches the value 0, which is the initial value of *SP*. Note that if a pop operation reads the item from location 0 and then *SP* is decremented, *SP* changes to 111111, which is equivalent to decimal 63. In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when *FULL* = 1 or popped when *EMTY* = 1.

Addressing Modes

- Each instruction needs data on which it has perform the specified operation.
- The operand may be Accumulator, general purpose Register or in some specified memory location
- Therefore, there are various ways to specify data.
- The techniques of specifying the address of the data are known as addressing Mode

Types Of Addressing Modes

- Direct or Absolute Addressing Mode
- Register Addressing Mode
- Register Indirect Addressing Mode
- Immediate Addressing Mode
- Indexed Mode or Based Indexed Mode
- Indexed Deferred Mode
- Auto Increment Addressing Mode
- Auto Decrement Addressing Mode
- Relative Addressing Mode

Deepak Bhalla (Asst. Professor (MCA & IT Dept.))

To explain the addressing modes, we use the following notation:

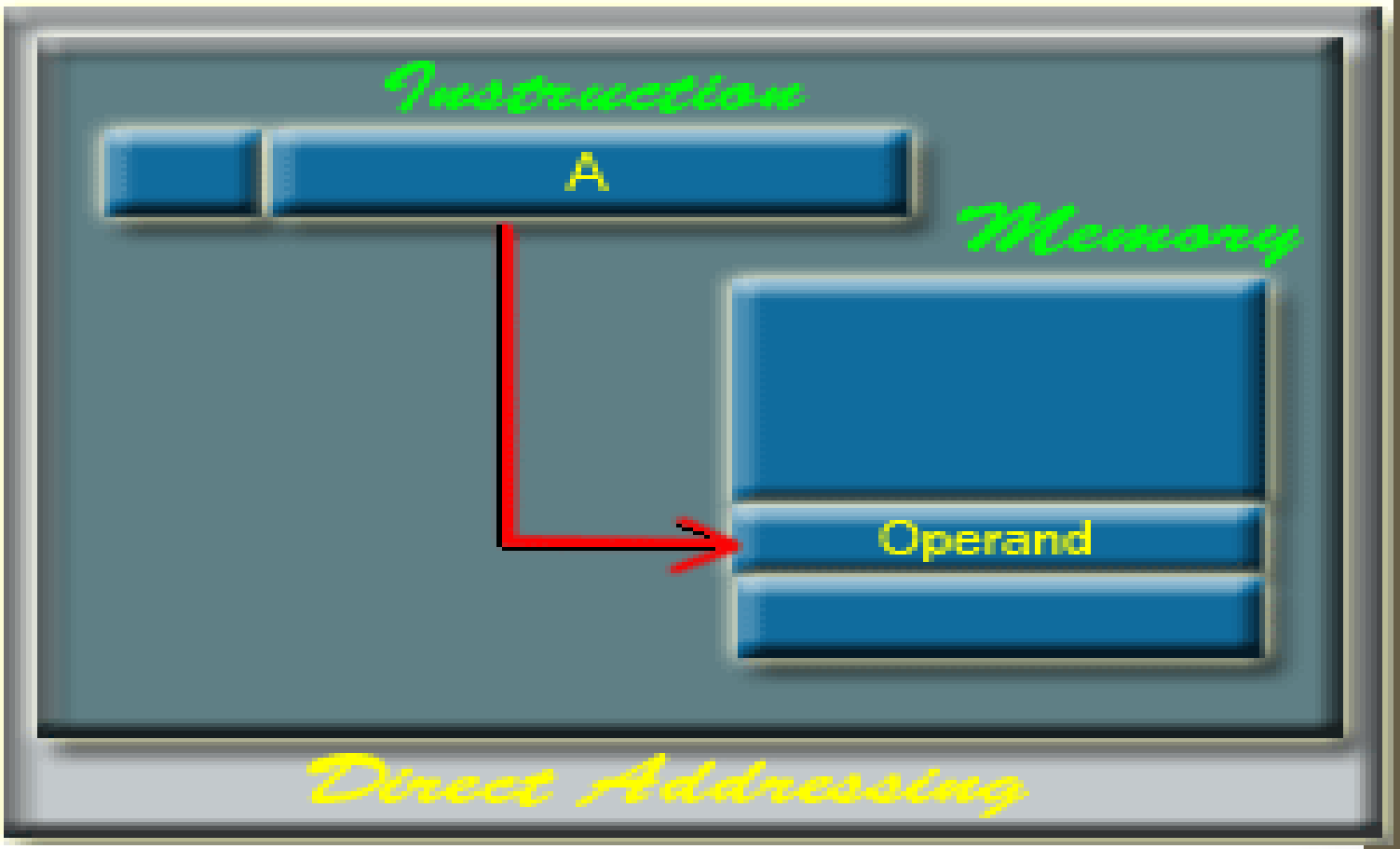
- A = contents of an address field in the instruction that refers to a memory
- R = contents of an address field in the instruction that refers to a register
- EA = actual (effective) address of the location containing the referenced operand
- (X) = contents of location X

Direct / Absolute Addressing Mode

(i) Direct or absolute Addressing : In direct addressing the address of the data (operand) is specified within the instruction itself. In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.

Example : STA 2500H, store the contents of the accumulator in the memory location 2500H.

In the above example 2500H is the memory address where data are to be stored. The memory address 2500H is given in the instruction itself. In this case it is understood that the source of data is accumulator.



Direct Addressing Mode

The fetching of data from the memory location in case of direct addressing mode is shown in the Figure above Here, 'A' indicates the memory address field for the operand.

Register Addressing Mode

(ii) Register Addressing : In register addressing the operands are located in general purpose registers. In other words the contents of a register is the operand. Therefore, only the names of the registers are to be specified in the instruction.

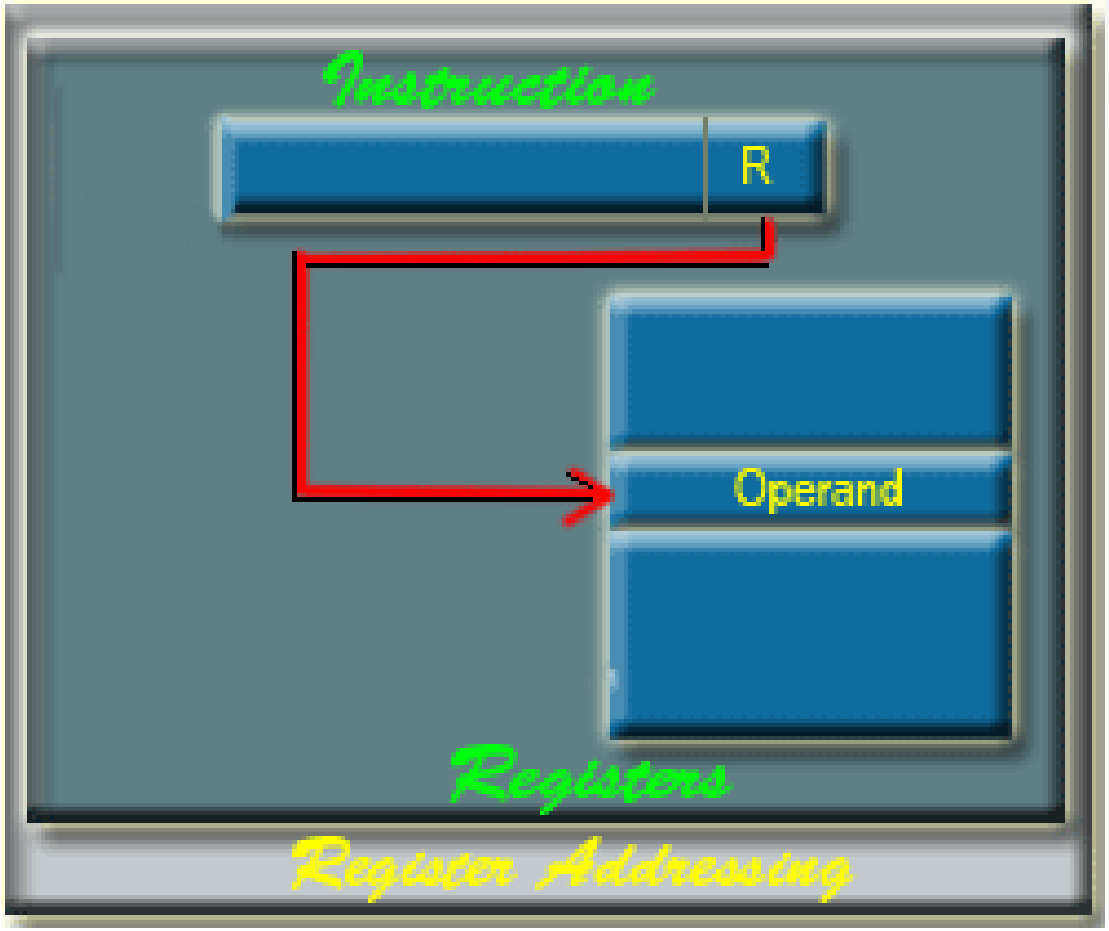
In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction A k-bit field can specify any one of 2^k registers.

Example : MOV A, B : Transfer the contents of register B to register A.

The opcode of this instruction is 78H. In addition to the operation to be performed, the opcode also specifies the addresses of the registers mentioned in the instruction. The opcode 78H in binary form is 01111000. The first two bits 01 denote MOV operation, the next three bits 111 are the binary code of register A and the last three bits 000 are the binary code of register B of intel 8085.

In register indirect

- Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address:
- EA = R
- The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.



Register Addressing Mode

The exact register location of the operand in case of Register Addressing Mode is shown in the above figure Here, 'R' indicates a register where the operand is present.

8085.

(iii) Register Indirect Addressing : In register indirect addressing the address of the operand is given indirectly. The contents of a register or a register-pair are the address of the operand.

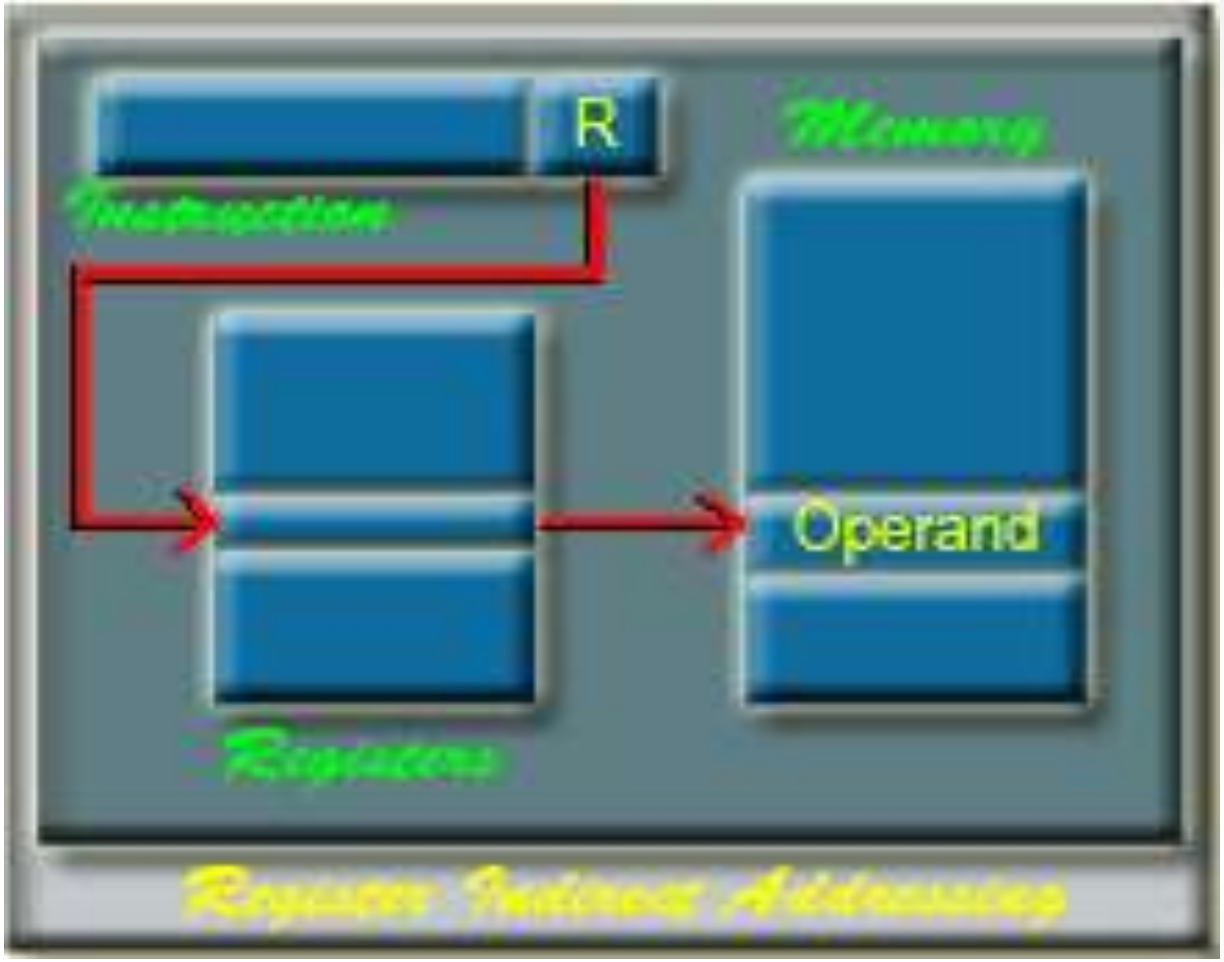
In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specify a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Example :

LXI H, 2400H	:	Load H-L pair with 2400H
MOV A, M	:	Move the content of the memory location, whose address is in H-L pair (i.e., 2400H), to the accumulator.

In this example MOV A, M is an example of register indirect addressing. For MOV A,M instruction the operand is in a memory location whose address is not directly given in this instruction. The address of the memory location is stored in H-L pair, which has been specified by the earlier instruction in the program, i.e., LXI H, 2400H.

In immediate addressing mode,



Immediate Addressing

H, Z400H.

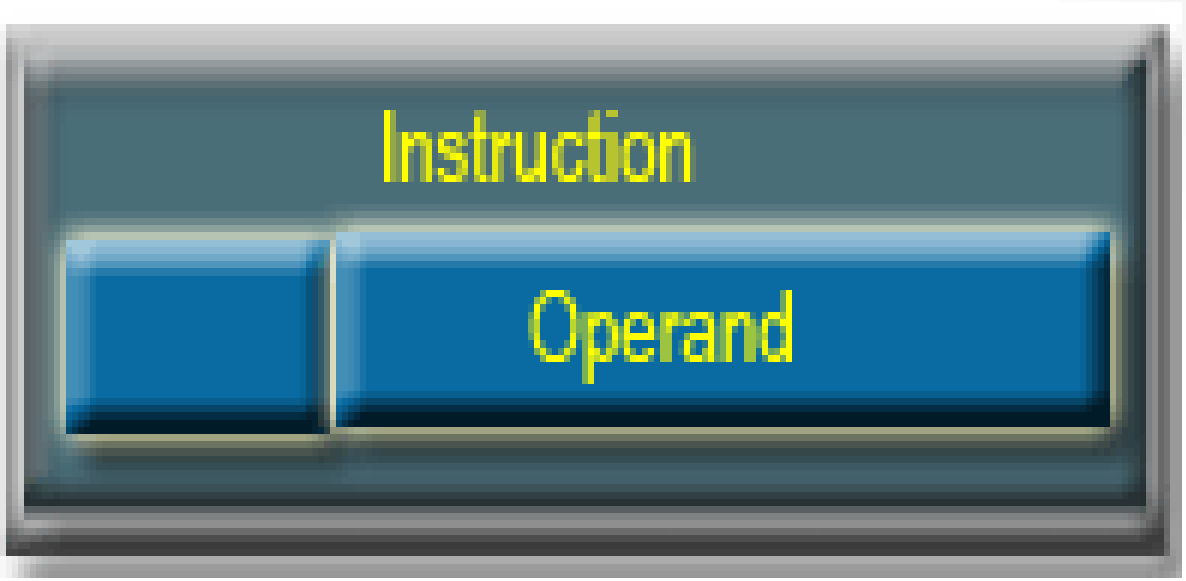
(iv) **Immediate Addressing :** In immediate addressing mode, the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate mode instructions are useful for initializing registers to a constant value.

Example : MVIA, 06, MOVE 06 to the accumulator.

- The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

  OPERAND = A

- This mode can be used to define and use constants or set initial values of variables. The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand. The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the world length.



Immediate Addressing Mode

The instruction format for Immediate Addressing Mode is shown in the above figure

Indexed Addressing Mode

- INDEX MODE OR Base Index Mode or Base Displacement Mode
- INDEX is actually come number X

  X : INDEX
- This X must be added to some register content

  (R) + X = EA
- So the effective address is calculated like this
- R holds the address
- R is called as BASE Register

Indexed Deferred Addressing Mode

- It contains address of address.
- Else all other part is sane as above,

Auto Increment Mode

- The register holds the address
- The address is Effective Address
- But After the address is used, the Register content will be incremented
- It is just like DIRECT ADDRESSING

Auto Decrement Mode

- The register holds the address
- The address is Effective Address
- But After the address is used, the Register content will be decremented

Relative Addressing Mode

a) The program counter itself be the register.

b) It is not much different from INDEX MODE.

(R) + X = EA

c) In the above statement

(R) – X = EA is also possible

Instruction Format

- An instruction format must include an opcode and, implicitly or explicitly, zero or more operands.
- Each explicit operand is referenced using one of the addressing mode that is available for that machine.
- Four common instruction format are shown in the figure :

- Page 149 DP

(a)

(b)

(c)

(d)

Four Common Instruction Formats

(a) Zero - address instruction

(b) One - address Instruction

(c) Two - address Instruction

(d) Three - address instruction

Pipelining

- Pipelining is a technique of decomposing a sequential process into sub-processes, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.
- Any operation that can be decomposed into a sequence of sub-operations of about the same complexity can be implemented by a pipeline processor.

How Pipelining Works

5.2. HOW PIPELINING WORKS

The basic principle behind pipelining is to utilize the time cycle of the instruction execution unit most efficiently. In one of the earliest instruction execution unit belonging to the 8085, a four step instruction cycle consisting of instruction fetch, decode, data fetch and execution was used (Figure 5.8). However it can be seen that in such a simple instruction cycle if we assume that each step takes one machine cycle than the execution of a instruction will take a minimum of four cycle. Thus it is a clear wastage of time as only one unit of the whole instruction cycle is used at a time. In order to use the machine efficiently we can try to utilize every unit of the instruction cycle thereby executing instructions every cycle.

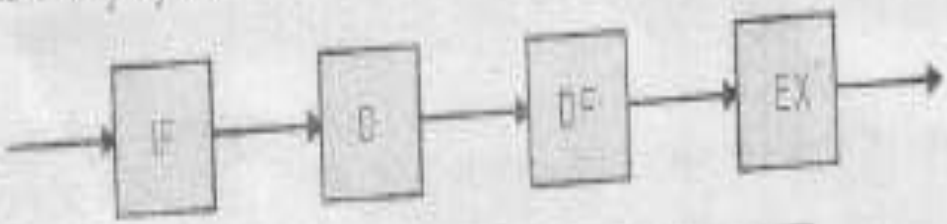


Fig. 5.8. Simple pipeline and its stages.

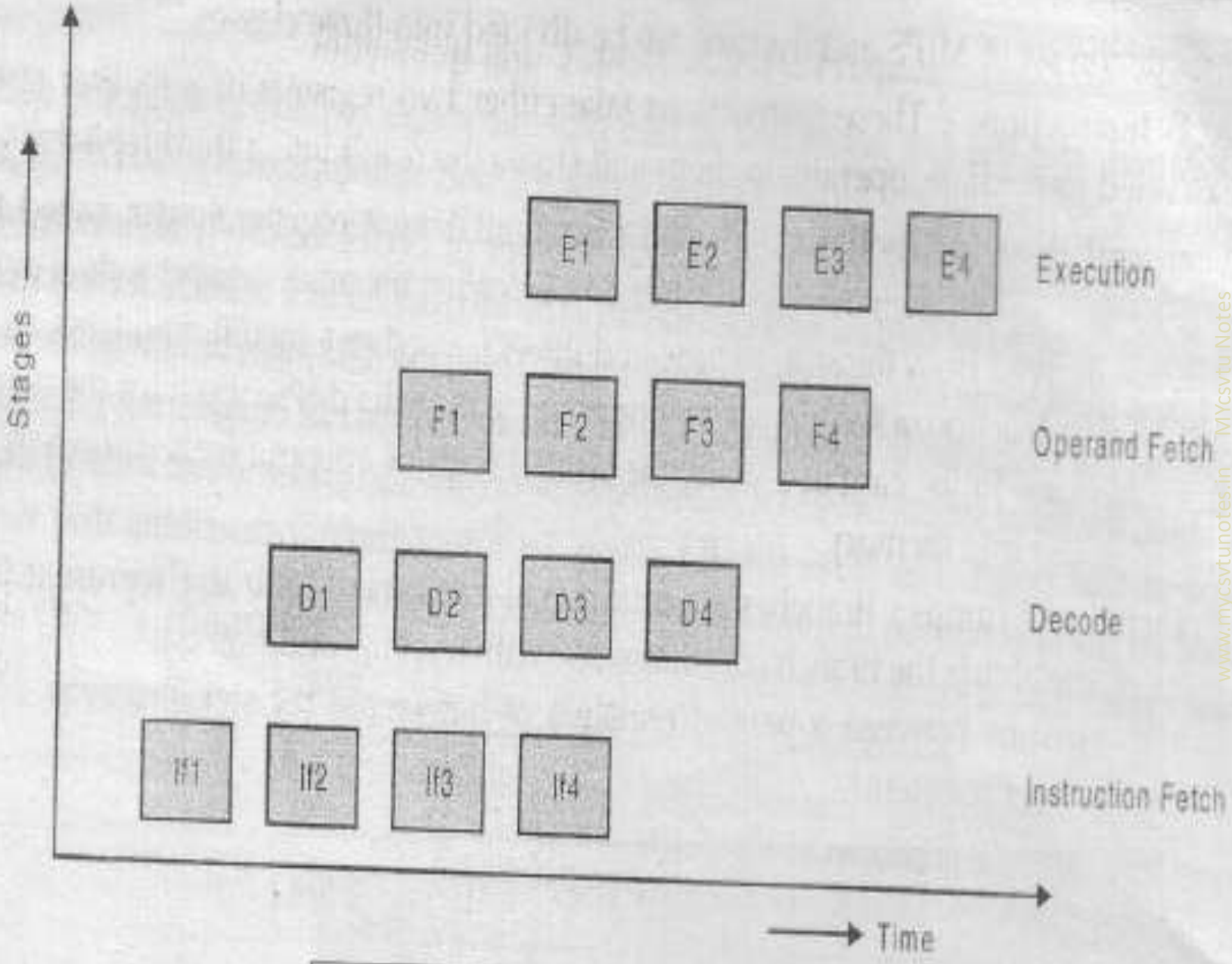


Fig. 5.9. Pipeline space time diagrams.

Pipelining, a standard feature in RISC processors, is much like an assembly line (Figure 5.9). Because the processor works on different steps of the instruction at the same time, more instructions can be executed in a shorter period of time.

A useful method of demonstrating this is the laundry analogy. Let's say that there are four loads of dirty laundry that need to be washed, dried, and folded. We could put the first load in the washer for 30 minutes, dry it for 40 minutes, and then take 20 minutes to fold the clothes. Then pick up the second load and wash, dry, and fold, and repeat for the third and fourth loads. Supposing we started at 6 PM and worked as efficiently as possible, we would still be doing laundry until midnight.

However, a smarter approach to the problem would be to put the second load of dirty laundry into the washer after the first was already clean and whirling happily in the dryer. Then, while the first load was being folded, the second load would dry, and a third load could be added to the pipeline of laundry. Using this method, the laundry would be finished by 9:30.

With a pipelined processor, we get the best of both: the low CPI of the single cycle CPU, and the low cycle time of the multiple cycles CPU, the cost, as usual, is increased complexity. Note that pipelining does not improve the performance of a single instruction. Rather, it improves the performance of groups of instructions. This is one of the classic issues in computer science: latency vs. throughput. A pipelined processor can't execute a single load instruction any faster than a multi-cycle CPU. However, a pipelined processor can execute five load instructions faster than a multi-cycle CPU.

Pipeline Implementation

In order to understand pipeline implementation, we need to take a simple processor which performs instruction execution in one cycle. Typically any RISC processor is able to execute one instruction in one cycle. The MIPS family of RISC processor is an ideal processor to demonstrate pipeline implementation. Here we shall deal with some basic issues of pipeline implementation. Readers can refer to the book of Patter son for more details.

The instruction set of MIPS architecture can be divided into three classes.

1. **ALU Instructions :** These instructions take either two registers or a register and a sign extended immediate, operate on them and stores the result into a third register.
2. **Load and store Instructions :** These instructions take a register source called the base register and an immediate field (16 bits in MIPS), called the offset, as operands. The effective address derived from the sum of base and offset is used as a memory address. In the case of a load instruction, a second register operand acts as the destination for the data loaded from memory. In the case of a store, the second register operand is the source of the data that is stored into memory.
3. **Branches and Jumps :** Branches are conditional transfers of control. There are usually two ways of specifying the branch condition: one with a set of condition bits or by a limited set of comparisons between a pair of registers or between a register and zero. Mips uses comparison of registers.

A five stage pipeline is shown in Fig. 5.10.

Pipeline Stages

- Instruction Fetch
- Instruction Decode
- Execute
- Memory access
- Write back

Time cycle	1	2	3	4	5	6	7	8	9
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

Fig. 5.10. Five stage pipeline.

Linear Pipeline Processors

- A *linear pipeline processor is a cascade of processing stages which* are linearly connected to perform a fixed function over a stream of data flowing from one end to the other.
- In modern computers, linear pipelines are applied for instruction execution, arithmetic computation and memory access operations.

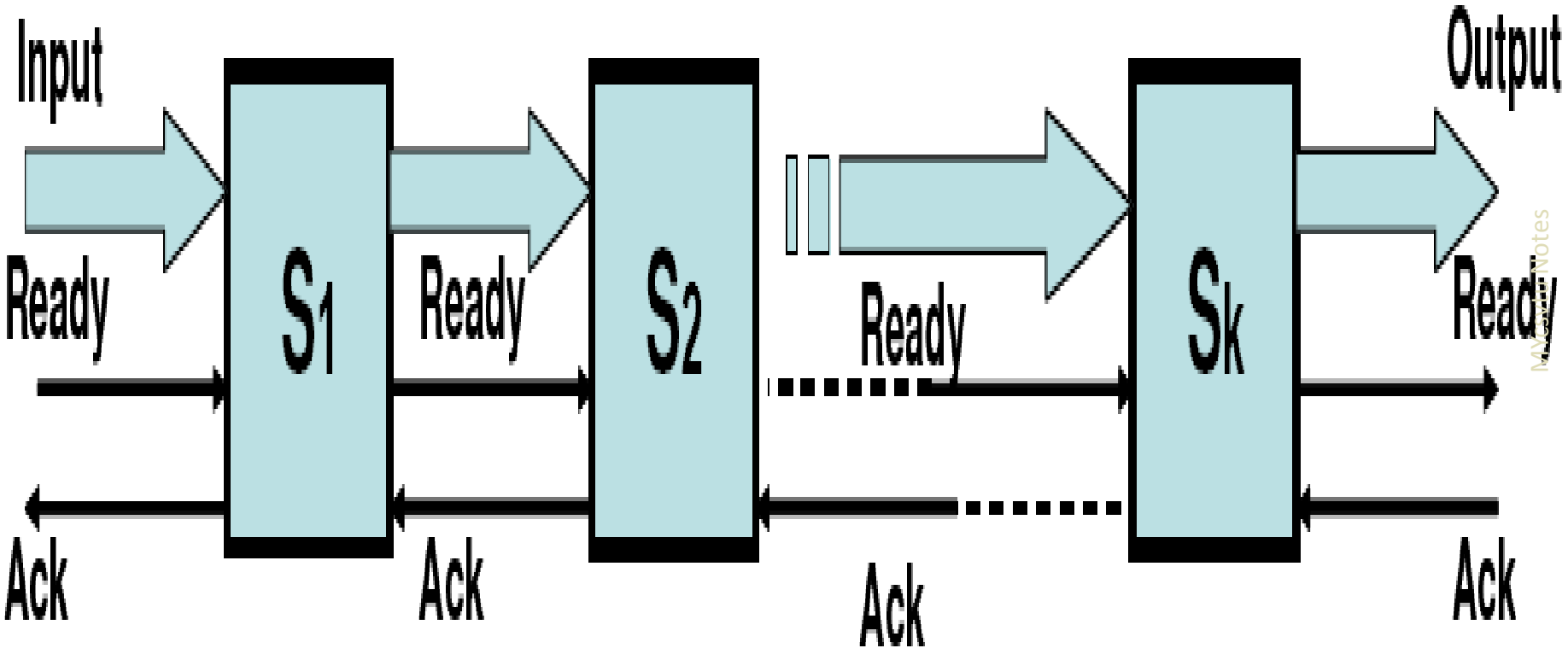
- A linear pipeline is constructed with *k processing stages* (segments). External inputs (operands) are fed into the pipeline at the first stage *S1. The processed results are passed from stage Si to*
- stage *Si+1 for all i = 1, 2, …., k-1. The final result emerges from*
- the pipeline at the last stage.Depending on the control of data flow
- along the pipeline, linear pipeline is modeled into two categories
- *asynchronous and synchronous.*

There are two types of linear pipeline processor

- Asynchronous Pipeline Model
- Synchronous Pipeline Model

Asynchronous Pipeline Model

- In this model, dataflow between adjacent stages is controlled by a handshaking protocol.
- When stage S_i is ready to transmit, it sends a ready signal to stage S_{i+1} .
- After stage S_{i+1} receives the incoming data, it returns an acknowledge signal to S_i

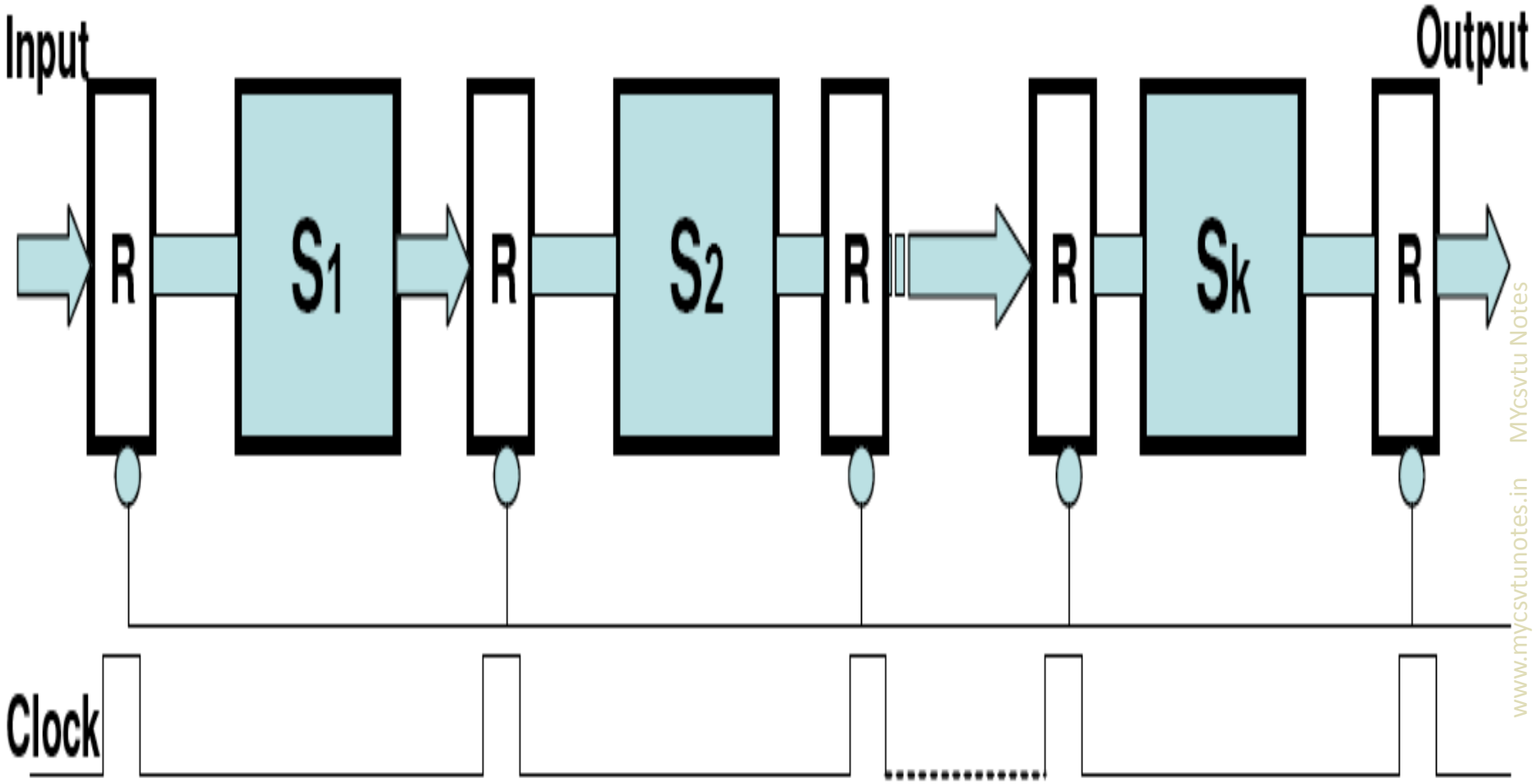


(a) An asynchronous pipeline model

- Asynchronous pipeline are useful to designing communication channels in message passing multicomputers .
- Asynchronous pipelines may have a variable throughput rate.
- Different amount of delay may be experienced in different stages.

Synchronous Pipeline Model

- In this model clocked latches are used to interface between stages.
- The latches are made with master slave flip flops, where can isolate inputs from outputs upon the arrival of clock pulse, all latches transfer data to the next stage simultaneously.



(b) A synchronous pipeline model

- The pipeline stages are combinational logic circuits.
- It is desired to have approximately equal delays in all stages.
- These delays determine the clock period and thus the speed of the pipeline

Unit 1 Test MM:50 Attempt Any 5
Q 5 & Q 7 is compulsory

Q 1) Write the difference between Computer Architecture & Organization.

Q 2) Explain Von Neumann Architecture.

Q 3) Explain the steps involved in instruction fetch decode execution cycle.

Q 4) Explain the organization of computer.

Q 5) Explain Addressing Modes with the help of diagram.

Q 6) Explain stack organization.

Q 7) What is pipelining. Explain the types of linear pipelining with the help of a diagram.