Digital Systems: Functional Units & their Interconnections

COMPUTER: Common Operating Machine Particularly/Purposely Used for Technological/Trade, Educational and Research.

Types of Computers:

- Embedded computers
- Personal computers
- Servers and Enterprise systems
- Supercomputers and Grid computers

Embedded computers: They are integrated into a larger device or system in order to automatically monitor and control a physical process or environment. They are used for a specific purpose rather than for general processing tasks. Typical applications include industrial and home automation, appliances, telecommunication products, and vehicles. Users may not even be aware of the role that computers play in such systems.

Personal computers: They have achieved widespread use in homes, educational institutions, and business and engineering office settings, primarily for dedicated individual use. They support a variety of applications such as general computation, document preparation, computer-aided design, audiovisual entertainment, interpersonal communication, and Internet browsing. A number of classifications are used for personal computers. Desktop computers serve general needs and fit within a typical personal workspace. Workstation computers offer higher computational capacity and more powerful graphical display capabilities for engineering and scientific work. Finally, Portable and Notebook computers provide the basic features of a personal computer in a smaller lightweight package. They can operate on batteries to provide mobility.

Servers and Enterprise systems: They are large computers that are meant to be shared by a potentially large number of users who access them from some form of personal computer over a public or private network. Such computers may host large databases and provide information processing for a government agency or a commercial organization.

Supercomputers and Grid computers: normally offer the highest performance. They are the most expensive and physically the largest category of computers. Supercomputers are used for the highly demanding computations needed in weather forecasting, engineering design and simulation, and scientific work. They have a high cost. Grid computers provide a more cost-effective alternative. They combine a large number of personal computers and disk storage units in a physically distributed high-speed network, called a grid, which is managed as a coordinated computing resource. By evenly distributing the computational workload across the grid, it is possible to achieve high performance on large applications ranging from numerical computation to information searching.

Functional Units & Interconnections: A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output, and control units, as shown in Figure below. The input unit accepts coded information from human operators using devices such as keyboards, or from other computers over digital communication lines. The information received is stored in the computer's memory, either for later use or to be processed immediately by the arithmetic and logic unit. The processing steps are specified by a program that is also stored in the memory. Finally, the results are sent back to the outside world through the output unit. All of these actions are coordinated by the control unit. An interconnection network provides the means for the functional units to exchange information and coordinate their actions.

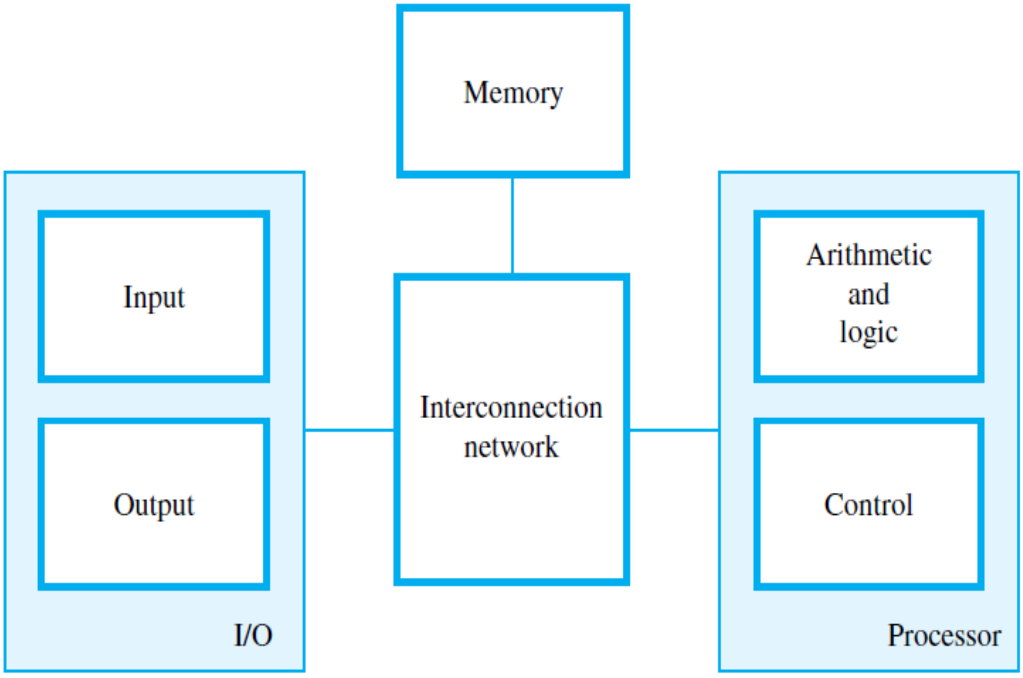


Fig: Basic Functional Units of a Computer

Input Unit: Computers accept coded information through input units. The most common input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor. Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball. These are often used as graphic input devices in conjunction with displays. Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing. Similarly, cameras can be used to capture video input.

Output Unit:
The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world. A familiar example of such a device is a printer. Most printers employ either photocopying techniques, as in laser printers, or ink jet streams. Such printers may generate output at speeds of 20 or more pages per minute. However, printers are mechanical devices, and as such are quite slow compared to the electronic speed of a processor. Some units, such as graphic displays, provide both an output function, showing text and graphics, and an input function, through touchscreen capability. The dual role of such units is the reason for using the single name input/output (I/O) unit in many cases.

Memory Unit: The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

Primary Memory
Primary memory, also called main memory, is a fast memory that operates at electronic speeds. Programs must be stored in this memory while they are being executed. The memory consists of a large number of semiconductor storage cells, each capable of storing one bit of information. These cells are rarely read or written individually. Instead, they are handled in groups of fixed size called words. The memory is organized so that one word can be stored or retrieved in one basic operation. The number of bits in each word is referred to as the word length of the computer, typically 16, 32, or 64 bits. To provide easy access to any word in the memory, a distinct address is associated with each word location. Addresses are consecutive numbers, starting from 0, that identify successive locations. A particular word is accessed by specifying its address and issuing a control command to the memory that starts the storage or retrieval process. Instructions and data can be written into or read from the memory under the control of the processor. It is essential to be able to access any word location in the memory as quickly as possible. A memory in which any location can be accessed in a short and fixed amount of time after specifying its address is called a random-access memory (RAM). The time required to access one word is called the memory access time. This time is independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for current RAM units.

Cache Memory
As an adjunct to the main memory, a smaller, faster RAM unit, called a cache, is used to hold sections of a program that are currently being executed, along with any associated data. The cache is tightly coupled with the processor and is usually contained on the same integrated-circuit chip. The purpose of the cache is to facilitate high instruction execution rates. At the start of program execution, the cache is empty. All program instructions and any required data are stored in the main memory. As execution proceeds, instructions are fetched into the processor chip, and a copy of each is placed in the cache. When the execution of an instruction requires data located in the main memory, the data are fetched and copies are also placed in the cache. Now, suppose a number of instructions are executed repeatedly as happens in a program loop. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use. Similarly, if the same data locations are accessed repeatedly while copies of their contents are available in the cache, they can be fetched quickly.

Secondary Storage: Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off. Thus additional, less expensive, permanent secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently. Access times for secondary storage are longer than for primary memory. A wide selection of secondary storage devices is available, including magnetic disks, optical disks (DVD and CD), and flash memory devices.

Arithmetic and Logic Unit:

Most computer operations are executed in the arithmetic and logic unit (ALU) of the processor. Any arithmetic or logic operation, such as addition, subtraction, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU. For example, if two numbers located in the memory are to be added, they are brought into the processor, and the addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.

Control Unit:

The memory, arithmetic and logic, and I/O units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the responsibility of the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states. I/O transfers, consisting of input and output operations, are controlled by program instructions that identify the devices involved and the information to be transferred. Control circuits are responsible for generating the timing signals that govern the transfers and determine when a given action is to take place. Data transfers between the processor and the memory are also managed by the control unit through timing signals.

----------XXX----------

6/23/2020

BUS, ITS ARCHITECTURE, and TYPES

COA: Unit-1

In this lecture notes, students will learn about the following:

- Basics of Bus
- Its Architecture
- Typical Bus Transactions
- Types of Buses
- Classification of Buses on the basis of Personal Computer

So, at the end of this lecture, students would be able to understand different concepts of Bus and its related topics.

Kalpna Sagar
ASST PROF AND ASST DEAN ACADEMICS
KIET GROUP OF INSTITUTIONS
GHAZIABAD

KIET GROUP OF INSTITUTIONS GHAZIABAD

(An ISO – 9001: 2008 Certified & 'A' Grade accredited Institution by NAAC)

A. BUS

What is bus?

- A bus is a set of wires that act a shared but common data path to connect multiple subsystems within the system.
- It consists of multiple lines, allowing the parallel movement of bits.
- The speed of the bus is affected by its length as well as by the number of devices sharing it.
- The size of a bus, known as its **width**, is important because it determines how much data can be transmitted at one time. For example, a 32-bit bus can transmit 32 bits of data, whereas a 64-bit bus can transmit 64 bits of data.
- Every bus has a **clock speed** measured in MHz or GHz. A fast bus allows data to be transferred faster, which makes applications run faster.
- An address bus is measured by the amount of memory a system can retrieve. A system with a 32-bit address bus can address 4 gibibytes of memory space. Newer computers using a 64-bit address bus with a supporting operating system can address 16 exbibytes or approx. 18446744073 GB of memory locations, which is virtually unlimited.

[**Note:1-GB** is defined as 1000^3 bytes, whereas **1-GiB** is defined as 1024^3 bytes.]

Why the bus is required?

- For communication and connections.
- The CPU communicates with the other component via a bus.
- Buses make it easy to connect new devices to each other and to the system.

Advantages of Bus

- Buses are low cost & very versatile.

Disadvantage of Bus

- At any one time, only one device (maybe it a register, the ALU, memory, or some other component) may use the bus. However, this sharing often results in a communication bottleneck.

Examples of Buses: Industry standards Buses e.g. ISA, EISA, SCSI, and PCI

B. BUS ARCHITECTURE

Devices share the bus, because of this sharing, the bus protocol (set of usage rules) is very important. Typical bus architecture consists of

1. Data lines,

2. Control lies,

3. Address lines,

4. Power lines.

a. Data lines

- The lines of a bus dedicated to moving data are called the data bus.
- These data lines contain the actual information that must be moved from one location to another.

b. Control lines

- They indicate which device has permission to use the bus and for what purpose (reading or writing from memory or from an input/output (I/O device, for example).
- These lines also transfer acknowledgements for bus requests, Interrupts, and clock synchronization signal.

c. Address lines

- These lines indicate the location (e.g. in memory) that the data should be either read from or written to.

d. Power lines

- These lines provide the electrical power necessary.

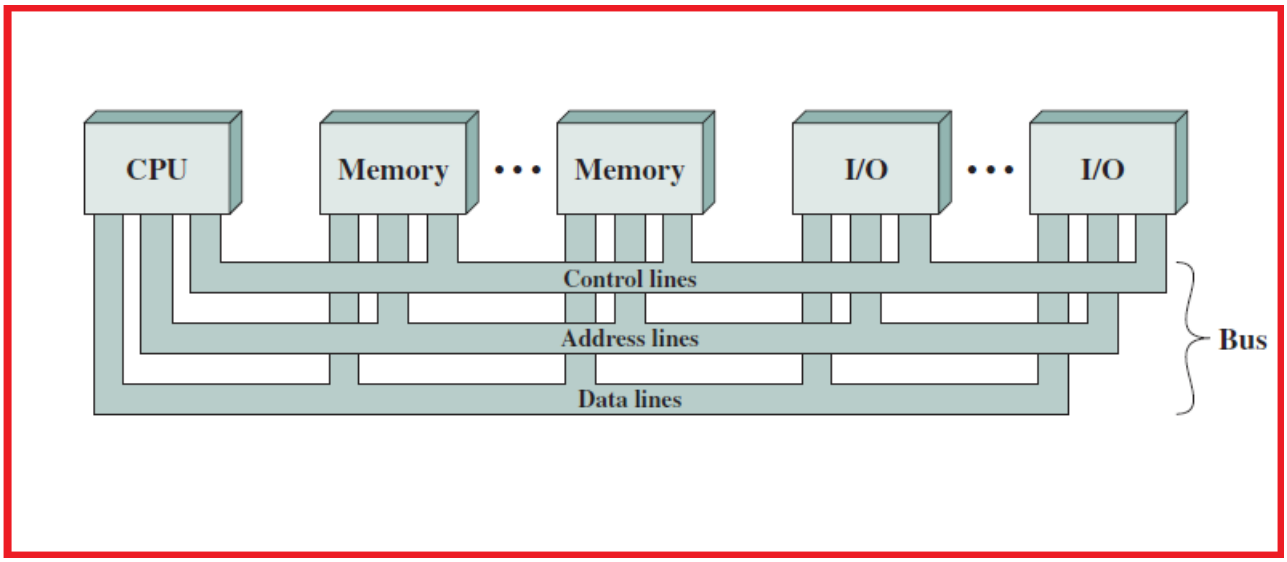


Figure 1: Bus Architecture/Bus Interconnection Scheme

C. TYPICAL BUS TRANSACTIONS

It includes

a) sending an address (for a read or write).

b) transferring data from memory to a register (a memory read).

c) transferring data to the memory from a register (memory write).

- Buses are used for I/O reads and writes from peripheral devices. Each type of transfer occurs within a bus-cycle, the time between two ticks of the bus clock.

D. TYPES OF BUSES

a. Point-to-Point Bus

- A bus can be point-to-point, connecting two specific components.

b. Multipoint Bus

- It can be a common pathway that connects a number of devices, requiring these devices to share the bus.

c. Processor -Memory Bus

- These buses are short, high-speed buses that are closely matched to the memory system on the machine to maximize the bandwidth (transfer of data) and are usually design specific.

d. I/O Bus

- These buses are typically longer than processor-memory buses and allow for many types of devices with varying bandwidth.
- These buses are compatible with many different architectures.

e. Back Plane Bus

It is actually built into the chasis of the machine and connects the processor, the I/O devices, and the memory (So all device are one bus).

f. Dedicated and Multiplexed Bus

Types of Bus	Dedicated Bus	Multiplexed Bus
Explanation	A dedicated bus is permanently assigned either to a particular function or to a physical subset of computer components. **Functional Dedication** means a separate bus for each different function. **Physical Dedication** means use of separate buses connecting different system components (same functionality).	Common bus used for both data and address transmission. Presence of a Control Line: Address valid signal. • Activate address valid control line. • Place address on multiplexed bus. • Modules check if the address refer them. • After sometime, address removed. • Data is now sent on this bus.
Advantages	High Throughput and Less Waiting Time	Use of fewer lines -> Less Cost and Space
Disadvantages	More Cost (Larger Bus Size)	Reduced Performance-> More Complex Circuit

g. Single Bus and Multiple Bus

All computing devices, from smartphones to supercomputers, pass data back and forth along electronic channels called "buses." The number and type of buses used strongly affect the machine's overall speed. Simple computer designs move data on a single bus; multiple buses, however, vastly improve performance. In a multiple-bus architecture, each pathway is suited to handle a particular kind of information.

Distinguishing features of Single Bus and Multiple Bus

1. Speed and Efficiency

In a single-bus architecture, all components including the central processing unit, memory and peripherals share a common bus. When many devices need the bus at the same time, this creates a state of conflict called bus contention; some wait for the bus while another has control of it. The waiting wastes time, slowing the computer down. Multiple buses permit several devices to work simultaneously, reducing time spent waiting and improving the computer's speed. Performance improvements are the main reason for having multiple buses in a computer design.

2. Expansion

Having multiple different buses available gives you more choices for connecting devices to your computer, as hardware makers may offer the same component for more than one bus type. For example, most desktop PCs use the Serial Advanced Technology Attachment interface for internal hard drives, but many external hard drives and flash drives connect via USB. If your computer's SATA connections are all used, the USB interface lets you connect additional storage devices.

3. Compatibility

As with all of a computer's components, bus designs evolve, with new types being introduced every few years. For example, the PCI bus that supports video, network and other expansion cards predates the newer PCIe interface, and USB has undergone several major revisions. Having multiple buses that support equipment from different eras lets you keep legacy equipment such as printers and older hard drives and add newer devices as well.

4. Multi-core

A single central processing unit places heavy demands on the bus that carries memory data and peripheral traffic for hard drives, networks and printers; since the mid-2000s, however, most computers have adopted a multi-core model that require additional buses. To keep each core busy and productive, the new bus designs ferry increased amounts of information in and out of the microprocessor, keeping wait times to a minimum.

E. CLASSIFICATION OF BUSES ON THE BASIS OF PERSONAL COMPUTER

a. Internal/System Bus

- These buses connect the CPU, memory, and all other internal components.

b. External/Expansion Bus

- The buses connect external devices, peripherals, expansion slots, and I/O ports to the rest of the computer.
- These buses are slower but allow for more generic connectivity.

c. Local Buses/Data Bus

- These buses connect a peripheral device directly to the CPU.
- These high - special buses can be used to connect only a limited number of similar devices.

6/23/2020

BUS ARBITRATION

COA: Unit-1

In this lecture notes, students will learn about the following:

- What is the Bus Arbitration?
- What are different Bus Arbitration Approaches?
- What are three Bus Arbitration Schemes or Methods along with their advantages, disadvantages, and applications?
- Comparison of various Bus Arbitration Methods.
- Hybrid of Bus Arbitration Methods.

So, at the end of this lecture, students would be able to understand which Bus Arbitration Method is best under which circumstances.

Kalpna Sagar
ASST PROF AND ASST DEAN ACADEMICS
KIET GROUP OF INSTITUTIONS
GHAZIABAD

BUS ARBITRATION OR CONTENTION/ PRIORITY RESOLVING SCHEMES

In Loosely coupled systems, all processors can use their local buses simultaneously. But the system bus can be used by only one module at a time. Hence there is a contest for the system bus. This is called bus arbitration. There are two approaches to Bus Arbitration

A. Centralized Bus Arbitration: Single bus arbiter performs the required arbitration.

B. Distributed Bus Arbitration: Devices participate in the selection of the next master.

To resolve the conflicts, we need to understand various bus arbitration schemes having different priority methods. These are explained below:

A) Daisy Chaining Method

- All bus masters use the same line for Bus Request.
- If the Bus Busy line is inactive, the Bus Controller gives the Bus Grant signal.
- Bus Grant signal is propagated serially through all masters starting from the nearest one.
- The bus master, which requires the system bus, stops this signal, activates the Bus Busy line, and takes control of the system bus.

Let's try to understand Daisy Chaining Method with the help of figure drawn below:

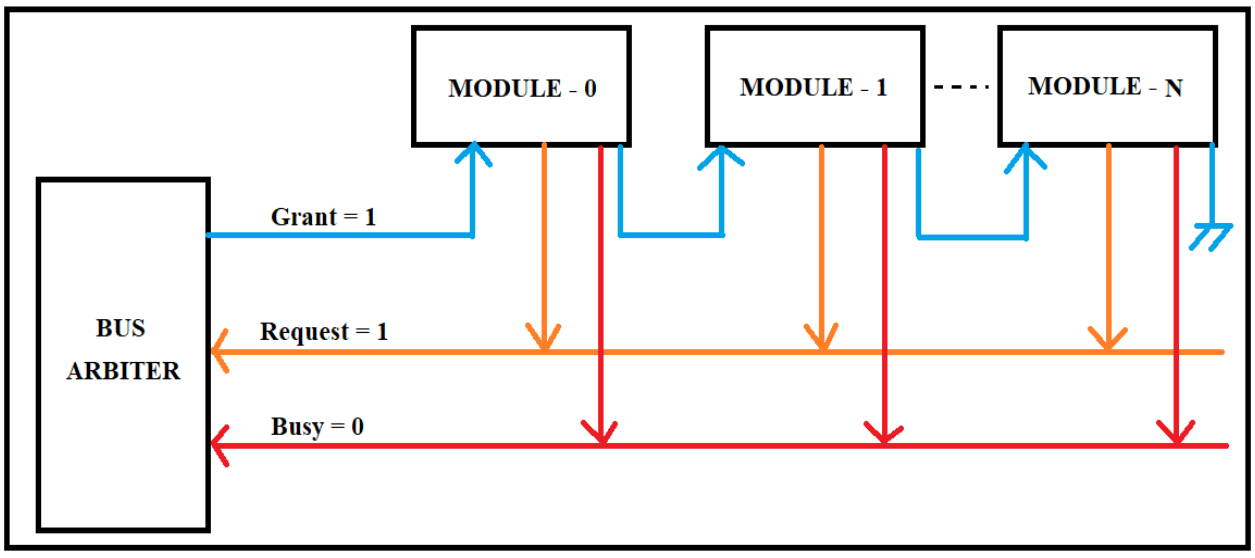


Figure: Daisy Chaining

Table: Advantages of Daisy Chaining Method

✓ The design is simple.
✓ The number of control lines is less.
✓ Also adding new bus masters is easy.

Table: Disadvantages of Daisy Chaining Method

✓ The priority of bus masters is rigid and depends on the physical proximity of the bus masters with the bus arbiter i.e. The one nearest to the Bus Arbiter gets the highest priority. Therefore, it has **poor performance.**
✓ The bus is granted serially and hence a propagation delay is induced in the circuit. Therefore, it has poor **priority mechanism.**
✓ Failure of one of the devices may fail the entire system. Therefore, it has **poor reliability.**

Application of Daisy Chaining Method

- **Daisy Chaining Method is very well suited for the smallest simplest network with very few computers e.g. 2 to 4 computers.**

B) Polling Method

- Here also all bus masters use the same line for Bus Requests.
- Here the controller generates a binary address for the master. E.g: To connect 8 bus masters we need 3 address lines (2^3= 8).
- In response to a Bus Request, the controller "polls" the bus masters by sending a sequence of bus master addresses on the address lines. Eg: 000, 010, 100, 011 etc.
- The selected master activates the Bus Busy line and takes control of the bus.

Let's try to understand Polling Method with the help of figure drawn below:

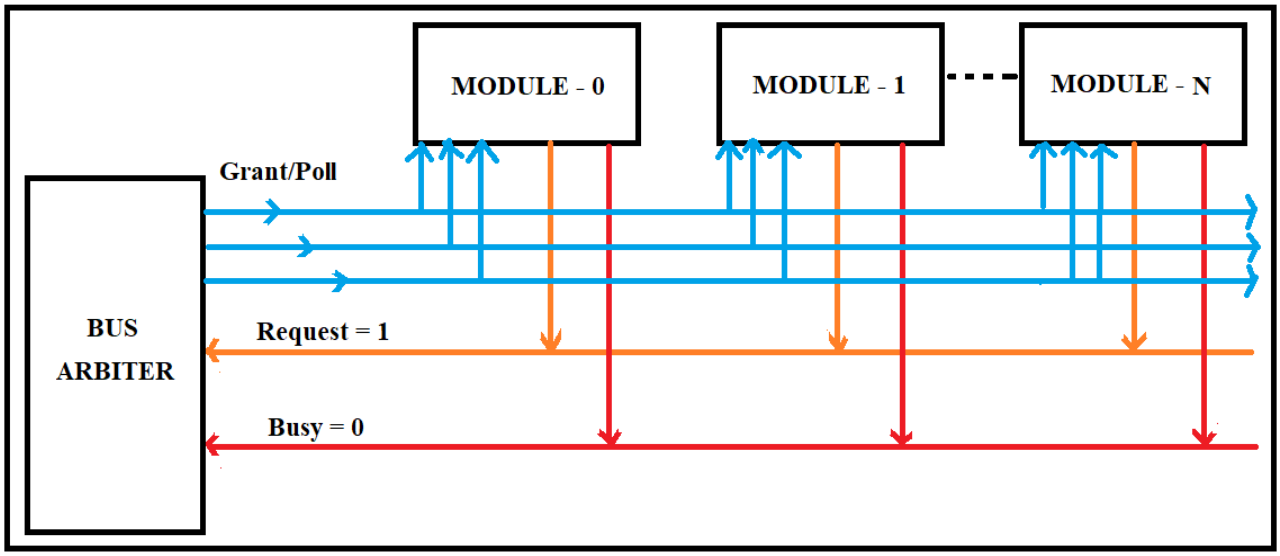


Figure: Polling

Table: Advantages of Polling Method

✓ This method is also quite simple.
✓ The priority is flexible and can easily be changed by altering the polling sequence.
✓ If one module fails, the entire system does not fail.

Table: Disadvantages of Polling Method

✓ Adding more bus masters is difficult as it increases the number of address lines of the circuit. E.g: In the above circuit to add the 9th Bus Master we need 4 address lines.

Application of Polling Method

- It is well suited for big networks which demand good reliability, decent priority scheme with controlled cost, though performance will not be the most important parameter for such networks.

C) Independent Request Method

- Here, all bus masters have their individual Bus Request and Bus Grant lines.
- The controller thus knows which master has requested, so the bus is granted to that master.
- Priorities of the masters are predefined so on simultaneous Bus Requests, the bus is granted based on the priority, provided the Bus Busy line is not active.
- The Controller consists of encoder and decoder logic for the priorities.

Let's try to understand Independent Request Method with the help of figure drawn below:

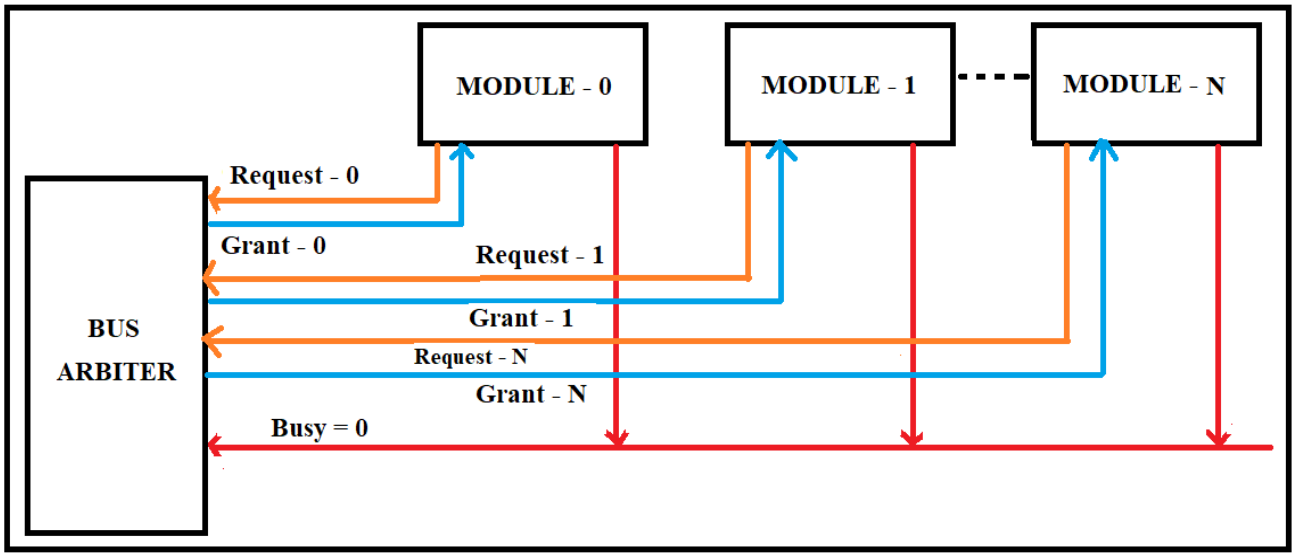


Figure: Independent Request

Table: Advantages of Independent Request

✓ Bus Arbitration is a fast and dynamic priority is also possible.
✓ The speed of Bus Arbitration is independent of the number of devices connected.

Table: Disadvantages of Independent Request

✓ The number of control lines required is more (2n line required for n devices).
✓ Hardware cost is high as large nos. of control lines are required.

Application of Independent Request

- It is well suited for big networks like in huge organizations where cost is not an issue but the performance is the key requirement.

Comparison of Bus Arbitration Methods

Differentiating Parameters	Daisy Chaining Method	Polling Method	Independent Request Method
Performance	Poor	Better performance than Daisy Chaining Method.	Better performance than Polling Method.
Reliability	Poor	Good	Good
Priority Mechanism	Poor	Good	Good
Addition of New Device	Adding a new device to the network is easy.	Adding a new device in the network means disturbing the entire existing system.	Adding new devices in the network makes the circuit more complex.
Number of Total Lines Required to Build a Network e.g. 1024 computers	Small (Total= 3)	Moderate (Total= 12)	Large (Total= 2049)
Applications	Used in the smallest simplest network having two to four computers.	Used in a large network where performance is not much required as well as where cost needs to be under control.	Used in those organizations, where cost is not the issue but good performance is the key requirement. E.g. In banks

- **In the modern technological era, we need the Hybrid of Bus Arbitration Methods. For example, polling and independent requests when putting together, give superb best features of both methods but yes, it makes the system structure more complex.**

Register

- Register is very fast memory in computer system, It is used to store data/instruction in-execution.
- It is also considered to be groups of flip-flop.
  - ↳ flip flops are capable of storing one (1) bit of information.
- N-bit register have n-flip flops.
- Computer needs processor registers for manipulating data and a register for holding a memory address.

There are two categories in which type of register can be categorise.
These are:

General purpose Register	Special purpose Register
This is used to data intermediate results during program execution. It can be accessed via assembly language.	User do not access these registers. These register are for computer System
Eg: Accumulator Data Register	Eg: MAR, MDR, PC, IR, Flag Regis

Brief Description of Important Registers

① PC - Program Counter (12 bits) : It holds the address of the memory location of the next instruction when the current instruction is executed by the microprocessor.

② Memory Address Register or MAR : This register (12 bit) stores the address location of the memory where the CPU wants to read or write some data.

③ MDR (Memory Data Register) : It contains (16 bit) the data to be stored in the computer storage (i.e. RAM), or the data after a fetch from the computer storage.

④ IR (Instruction Register) : It holds the 16 bit instruction which is currently been executed.

⑤ flag register : Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0).

⑥ MBR (Memory Buffer Register) : This register holds the content of data or instruction read from, or written in memory.

⑦ Data Register: A register used in micro computer to temporarily store data being transmitted to or from a peripheral device.

⑧ Index Register: A index register receives, stores, and outputs instruction - changing codes in a computer.

⑨ Accumulator (ACC) (16 bit): Employed to hold temporarily operands and results of ALU operations.

—X—

Register transfer

Microoperations – The operations executed on data stored in registers.

Eg: counter - increment and load.
Bidirectional shift [left or right]

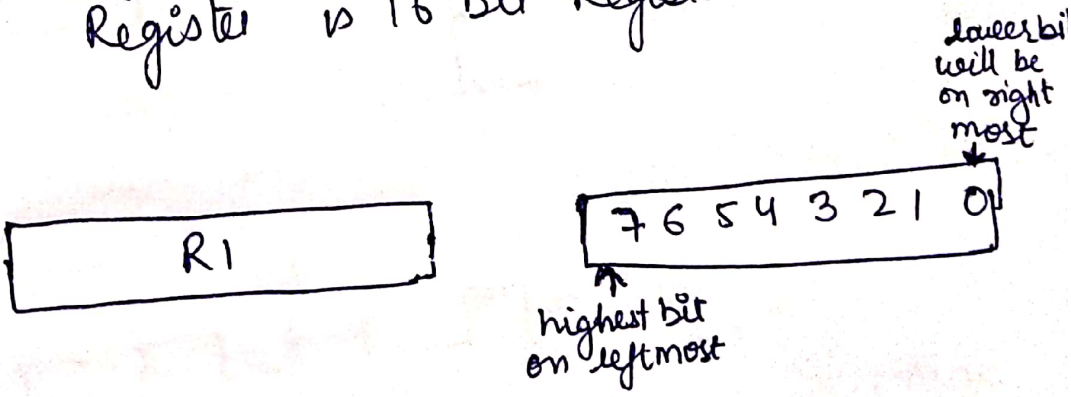
Register transfer language – is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.

Registers are **denoted** by **capital letters.**
Eg: MAR for Memory Address Register
PC for Program counter.

[if n-bit Register – n flip flops are present]

* **Block Representation of Register**

if Register is 8 bit Register named R1
and if Register is 16 bit Register named R2
then



In R2, there are 16 bits, which is further divided in two parts Bits (0-7) [for low byte] symbol L
Bits (8-15) [for high byte] symbol H

PC is Program counter.

Information transfer

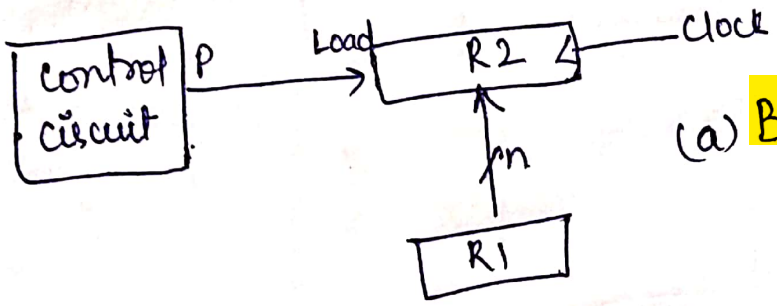
- $R2 \leftarrow R1$: This denotes a transfer of the content of register R1 into register R2.
[NOTE - content of R1 does not change].

-

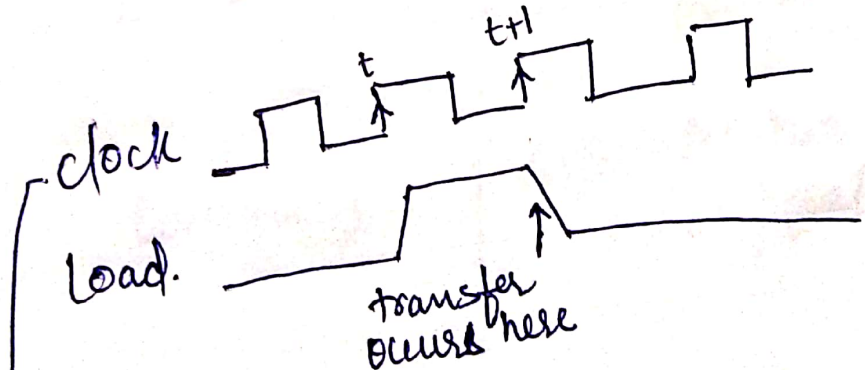
$P : R2 \leftarrow R1$: This denotes a transfer of the content of register R1 into R2 only when P (control signal) is true(1). Here, P is control function. is Boolean variable that may be 1 or 0.

- Control condition P is terminated by :(colon) the transfer microoperation.

Block Diagram - transfer from R1 to R2 when P = 1



(a) Block Diagram



(b) Timing Diagram.

↓ is a representation of a set of signal in the time domain

↳ A clock pulse is a time varying voltage signal applied to control operation of a flip flop.

As shown in timings diagram, P is is activated in the control section by the rising edge of a clock pulse at time t. The next positive transition of the clock at time t+1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel.

P may go back to 0 at time t+1; otherwise, the transfer will occur with every clock pulse transition while P remains active.

Basic Symbols for Register transfer.

Symbols	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parenthes ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma ,	Separates two Microoperations	R2 ← R1, R1 ← R2

Memory transfer

Memory can be read or written. It is denoted by M. Address has to specified from where to read or write, that address is written in [ ] square brackets.

Memory read: The transfer of information from memory to the outside environment

Read: DR ← M[AR]

here DR is Data Register
AR is address Register.

Memory write: the transfer of new info to be stored into the memory.

Write: M[AR] ← DR

Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Addition
$R3 \leftarrow R1 - R2$	Subtraction.
$R2 \leftarrow \overline{R2}$	complement. (1's)
$R2 \leftarrow \overline{R2} + 1$	2's complement.
$R1 \leftarrow R1 + 1$	Increment by 1
$R1 \leftarrow R1 - 1$	Decrement by 1
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2

6/23/2020

BUS SYSTEM USING MULTIPLEXER

COA: Unit-1

In this lecture notes, students will learn about the following:

- MULTIPLEXER
- BUS SYSTEM USING MULTIPLEXER

So, at the end of this lecture, students would be able to understand Bus System Using Multiplexer.

Kalpna Sagar
ASST PROF AND ASST DEAN ACADEMICS
KIET GROUP OF INSTITUTIONS
GHAZIABAD

MULTIPLEXER(MUX)

A multiplexer is a combinational circuit that receives binary information from one of 2^n input data lines and directs it to a single output line. The selection of a particular input data line for the output is determined by a set of selection inputs. A 2^n -to-1 multiplexer has 2^n input data lines and n-input selection lines whose bit combinations determine which input data are selected for the output.

A 4-to-1-line multiplexer is shown in figure 1. Each of the four data inputs I_0 through I_3 is applied to one input of an AND gate. The two selection inputs S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate to provide the single output. To demonstrate the circuit operation, consider the case when $S_1S_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1. The third input of the gate is connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of I_2, thus providing a path from the selected input to the output. The 4-to-1 line multiplexer of figure has six inputs and one output. A truth table describing the circuit needs 64 rows since 6-input variables can have 2^6 binary combinations. This is an excessively long table and will not be shown here. A more convenient way to describe the operation of multiplexers is by means of a function table. The function table for the multiplexer is shown in Table 1. The table demonstrates the relationship between the four data inputs and the single output as a function of the selection inputs S_1 and S_0.

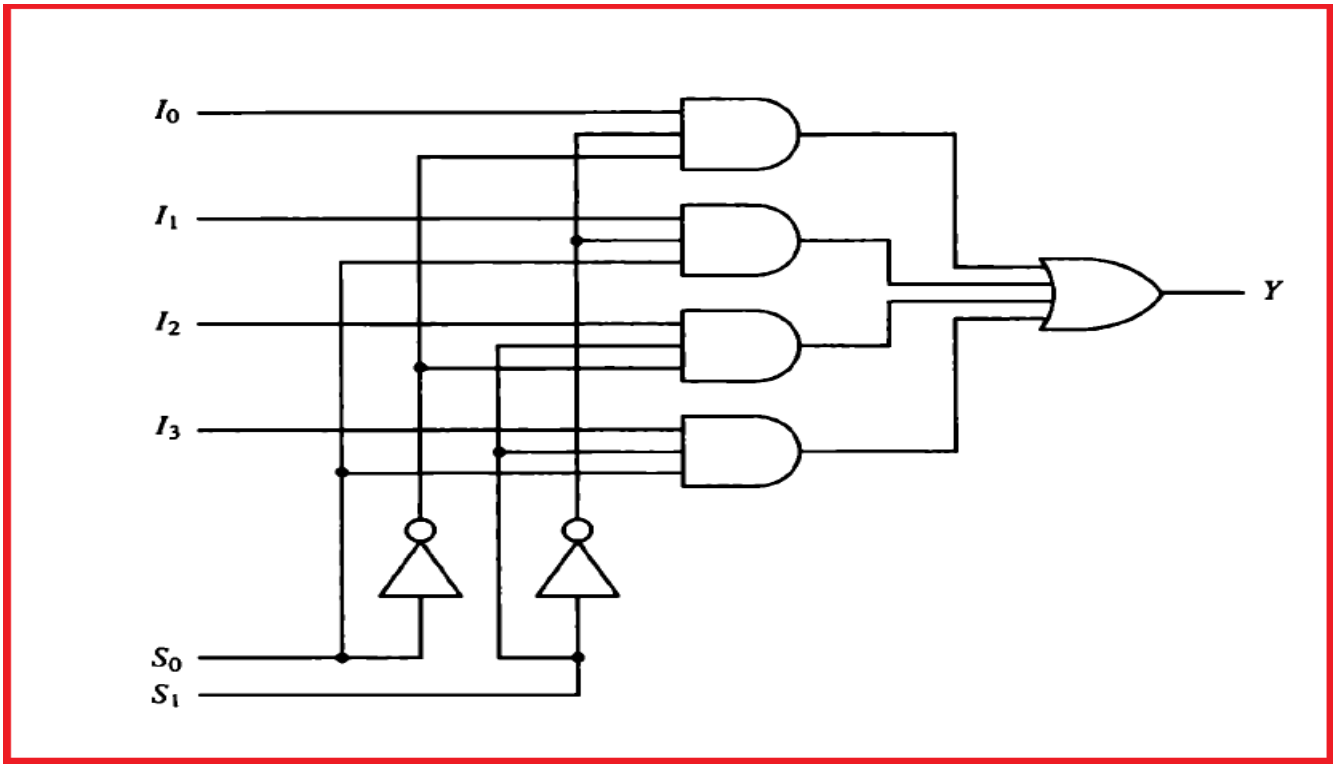


Figure 1: 4-to-1-line multiplexer

When the selection inputs are equal to 00, output Y is equal to input I_0. When the selection inputs are equal to 01, input I_1 has a path to output Y, and similarly for the other two combinations. The multiplexer is also called a data selector, since it selects one of many data inputs and steers the binary information to the output.

Table 1: Function Table for 4-to-1 Line Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

The AND gates and inverters in the multiplexer resemble a decoder circuit, and indeed they decode the input selection lines. In general, a 2^n-*to-1*-line multiplexer is constructed from an *n*-*to*-2^n decoder by adding to it 2^n input lines, one from each data input. The size of the multiplexer is specified by the number 2^n of its data inputs and the single output. It is then implied that it also contains *n*-input selection lines.

As in decoders, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer. The enable input is useful for expanding two or more multiplexers to a multiplexer with a larger number of inputs.

BUS SYSTEM USING MULTIPLEXER

A typical digital computer has many registers, and paths must be provided to transfer information from one register to another. The number of wires will be excessive if separate lines are used between each register and all other registers in the system. A more efficient scheme for transferring information between registers in a multiple-register configuration is a **common bus system**.

A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

One way of constructing **a common bus system is with multiplexers**. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers in shown in figure 2.

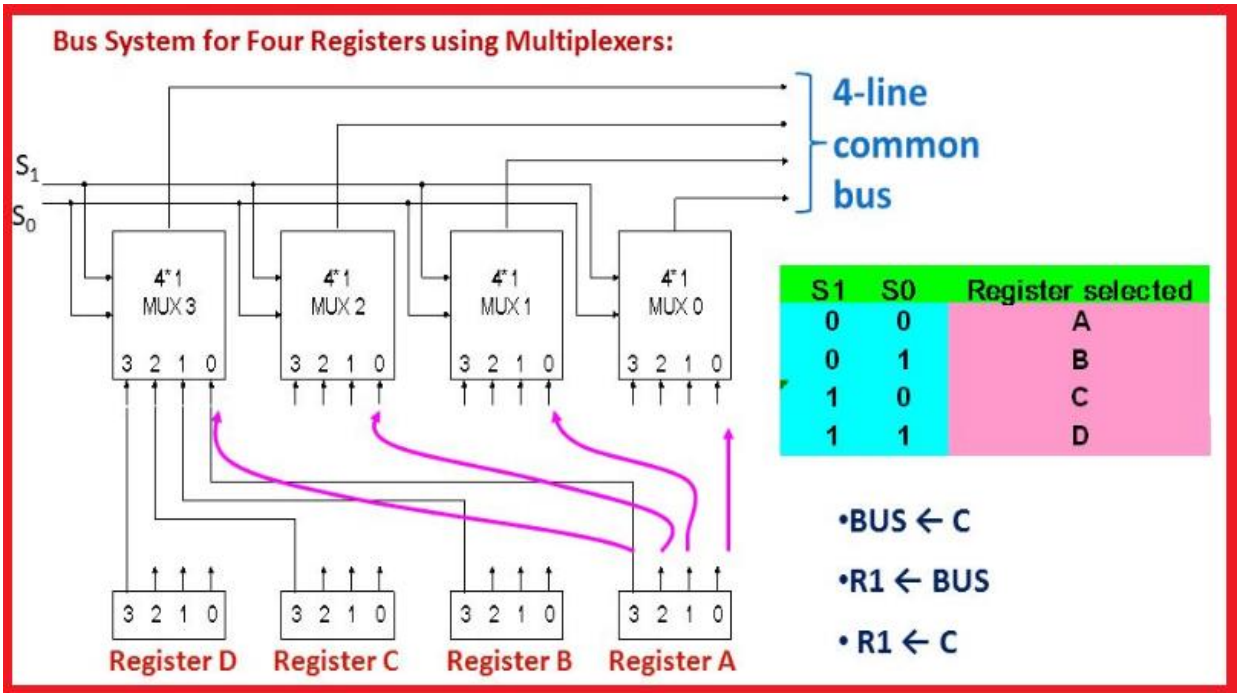


Figure 2: Bus System for Four Registers

Each register has four bits, numbered 0 through 3. The bus consists of four 4X1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers. For example, output 1 of register A is connected to input 0 of Mux 1 because this input is labelled A_1. The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. The MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.

The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus. When S_1S_0=00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers. Similarly, register B is selected if S_1S_0=01, and so on. Table shown in above figure shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

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

For example, a common bus for 8- registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected. The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. The statements are mentioned in above figure.

6/23/2020

BUS SYSTEM USING TRI-STATE BUFFER

COA: Unit-1

In this lecture notes, students will learn about the following:

- TRI-STATE BUS BUFFERS
- BUS SYSTEM USING TRI-STATE BUFFER

So, at the end of this lecture, students would be able to understand Bus System Using Tri-State Buffer.

Kalpna Sagar
ASST PROF AND ASST DEAN ACADEMICS
KIET GROUP OF INSTITUTIONS
GHAZIABAD

Tri-State Buffer

A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. The third state is a *high-impedance state*. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance. Three-state gates may perform any conventional logic, such as AND or NAND. However, the one most commonly used in the design of a bus system is the buffer gate.

Application of Tri-State Buffer: A bus system can be constructed with three-states gates instead of multiplexers too.

The graphic symbol of a tri-state buffer gate is shown in Figure 1.

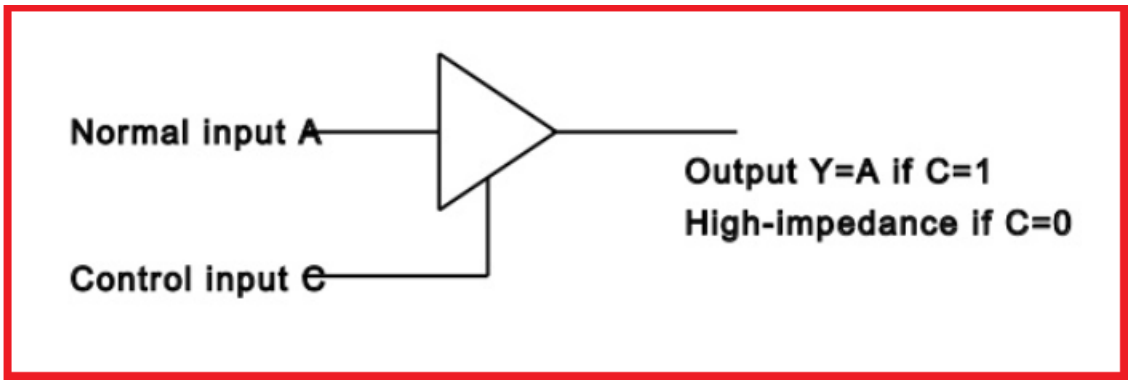


Figure 1: Graphic Symbols for tri state-buffer.

It is distinguished from a normal buffer by having both a normal input and a control input. The control input determines the output state. When the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input.

When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.

Special Feature of tri state-buffer: The high-impedance state of a three-state gate provides a special feature not available in other gates. Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.

Bus System with Tri-State Buffers

The construction of a bus system with tri-state buffers is demonstrated in Figure 2.

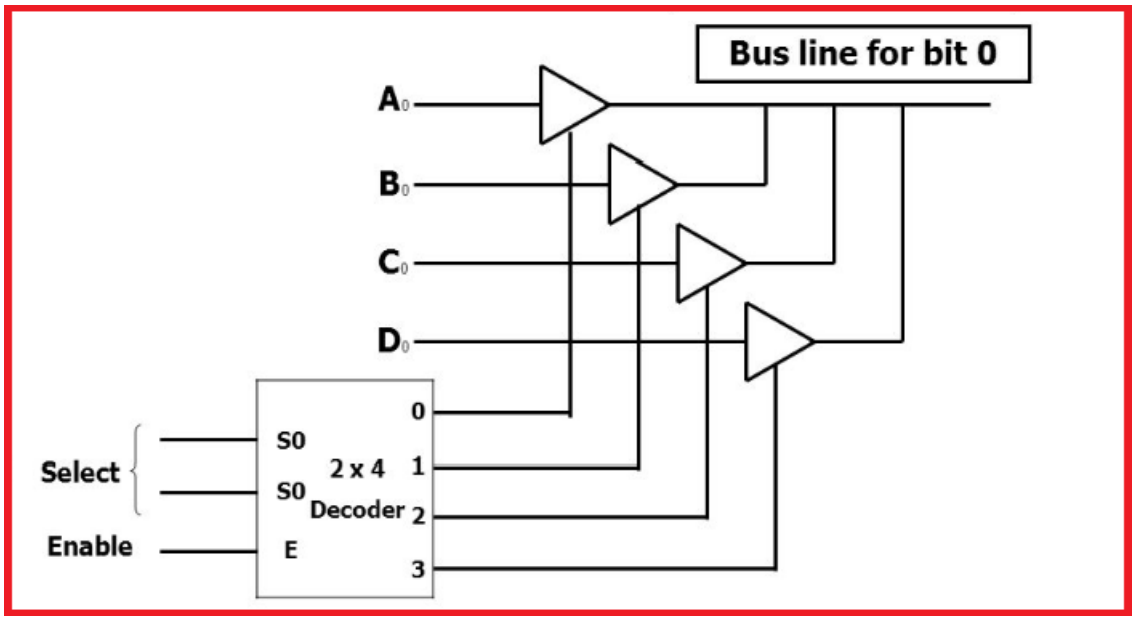


Figure 2: Bus Line with tri state-buffers.

The outputs of four buffers are connected together to form a single bus line. (It must be realized that this type of connection cannot be done with gates that donot have three-state outputs.) The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line. No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high-impedance state.

One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the Figure 2. When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled. When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

To construct a common bus for four registers of n bits each using three-state buffers, we need n circuits with four buffers in each as shown in Figure 2. Each group of four buffers receives one significant bit from the four registers. Each common output produces one of the lines for the common bus for a total of n lines. Only one decoder is necessary to select between the four registers.

KIET Group of Institutions, Delhi-NCR, Ghaziabad

(An ISO – 9001: 2008 Certified & 'A' Grade accredited Institution by NAAC)

Department of Information Technology

Processor Organization

-Compiled by Prof. Minakshi

Topics Covered:

- **Introduction of Computer components**
- **Introduction to Processor**
- **Processor organization**
- **Register Organization**

After this lecture, a student will be able to

1. **List the components of a computer.**
2. **Describe the working of a processor.**
3. **List the components of a processor.**
4. **Determine the purposes of different registers in the processor.**

What is a computer?

A computer is a programmable electronic digital device designed to accept data, perform prescribed mathematical and logical operations and display the results of these operations.

Or

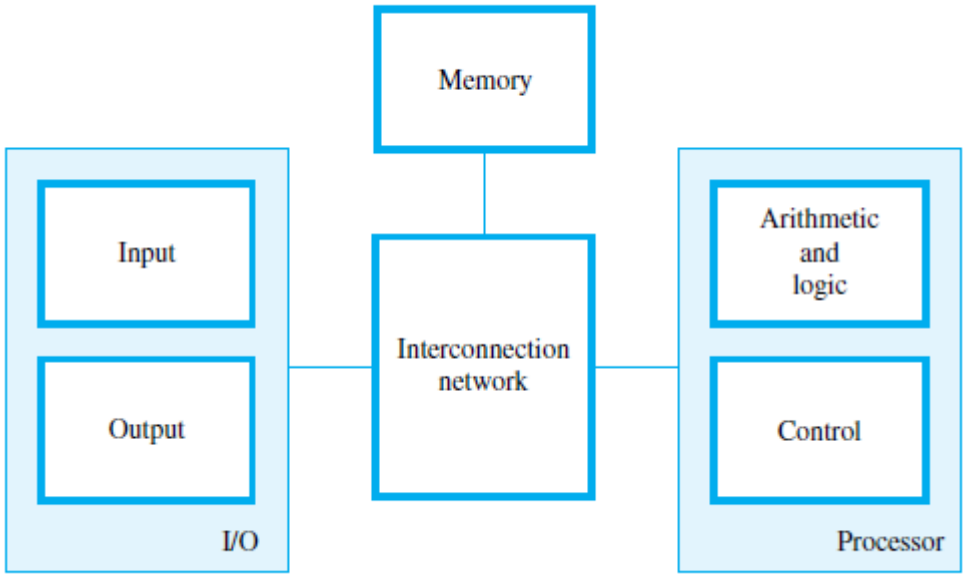
A computer is an electronic digital device that manipulates binary data. It has the ability to store, retrieve, and process the data.

What are the components of a computer?

A computer consists of five functionally independent main parts:

- Input
- Memory
- Processor (Arithmetic and Logic Unit, and Control Units)
- Output

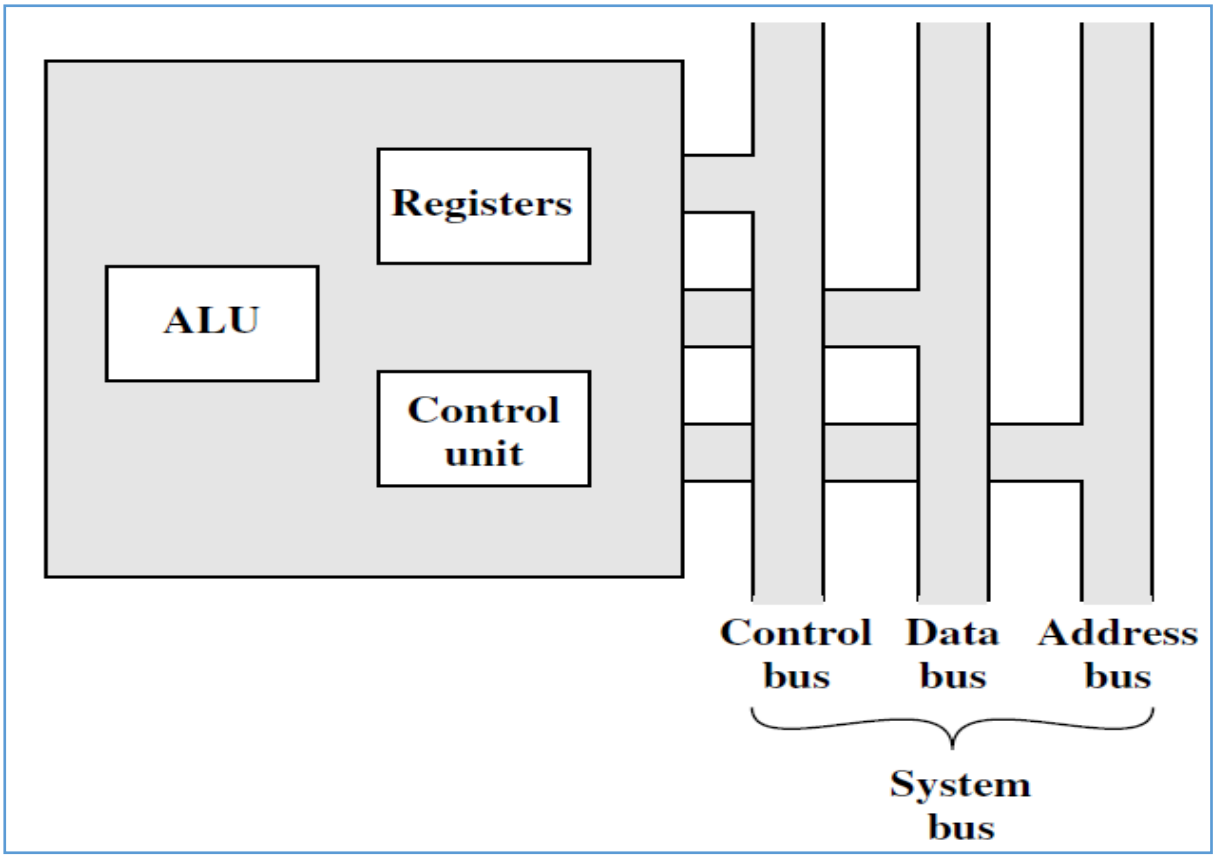
As shown in the given figure:



- The input unit accepts coded information from human operators using devices such as keyboards.
  Input is a device that transfers the information from outside world to the computer.

- The information received is stored in the computer's memory, either for later use or to be processed immediately by the arithmetic and logic unit.
  Memory is a medium that stores the binary information (instructions and data)
- The processing steps are specified by a program that is also stored in the memory.
  ALU (Arithmetic and Logic Unit) is a group of circuits that performs arithmetic (addition, increment/decrement or complement) and logic (AND, OR, NOT, X-OR etc) operations.
- Finally, the results are sent back to the outside world through the output unit.
- All of these actions are coordinated by the control unit. A control unit is a group of circuits that provides timing and signals to all the operations in the computer and controls the data flow.
- An interconnection network provides the means for the functional units to exchange information and coordinate their actions.

The processor with the system bus has been shown in the following figure:



What are the functions of a processor?

- Fetch instruction
- Interpret instruction (or decode the instruction)
- Process data
- Write data

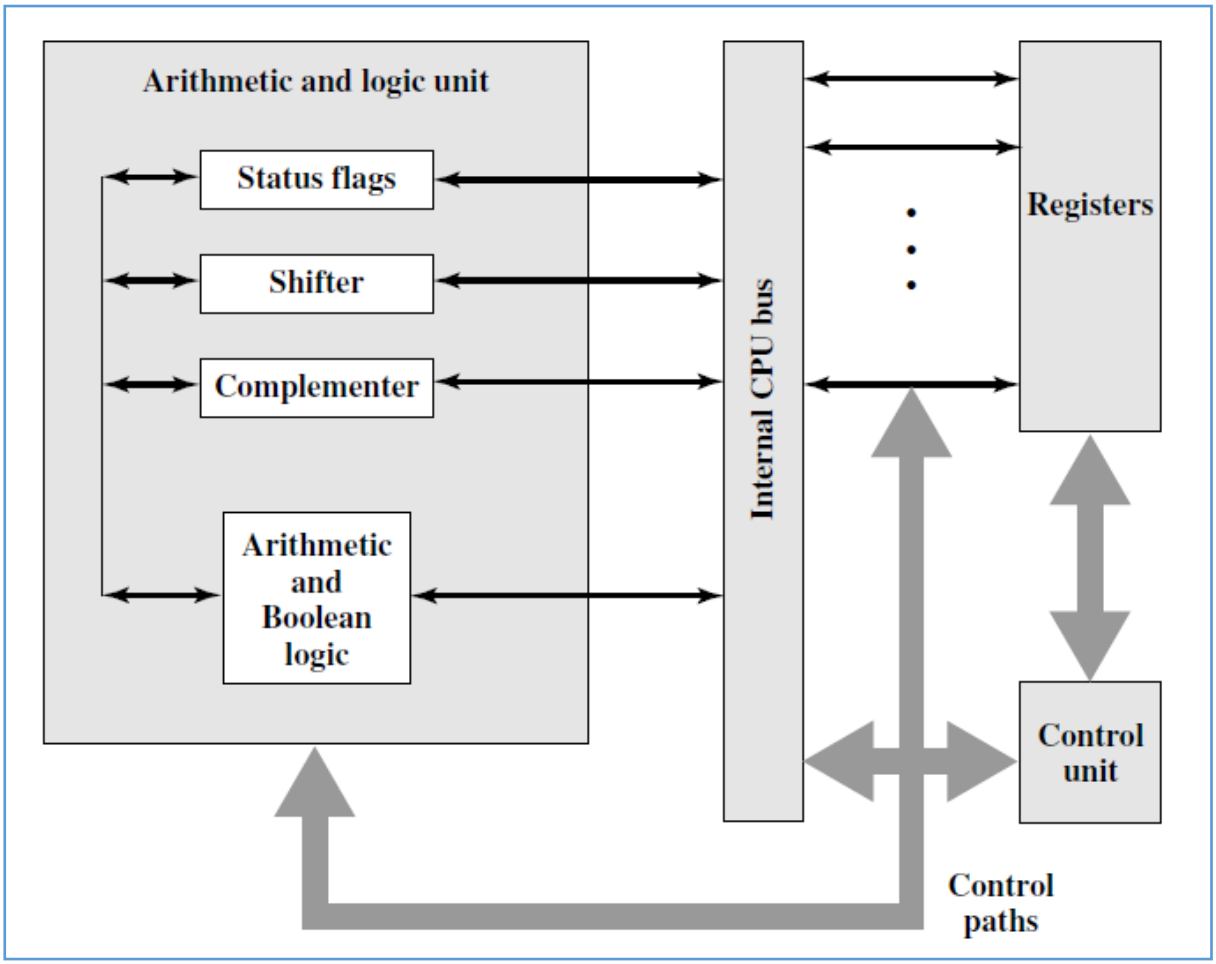
Fetch instruction: The processor reads an instruction from memory.

Interpret instruction (or decode the instruction): The instruction is decoded to determine what action is required.

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.

What is the internal structure of the processor?



Components of the processor:

- ALU (Arithmetic and Logic Unit)
- CU (Control Unit)
- Registers
- Internal Bus system

The ALU does the actual computation or processing of data. ALU (Arithmetic and Logic Unit) is a group of circuits that performs arithmetic (addition, increment/decrement or complement) and logic (AND, OR, NOT, X-OR etc) operations.

The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. A control unit is a group of circuits that provides timing and signals to all the operations in the computer and controls the data flow.

A minimal internal memory, consisting of a set of storage locations, called registers.

An internal processor bus is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory. Some mechanism that provides for communication among the control unit, ALU, and registers

Note: The basic operations are addition, shifting, and complement.
The arithmetic operations such as subtractions, multiplication and division are performed with the help of these basic operations.

Register organization:

- User-visible registers
- Control and status registers

User-visible registers: used by the machine language or assembly language programmer.

User-visible registers:

- General purpose
- Data
- Address
- Condition codes

General-purpose registers can be assigned to a variety of functions by the programmer.

Data registers may be used only to hold data and cannot be employed in the calculation of an operand address.

Address registers may themselves be somewhat general purpose, or they may be devoted to a particular addressing mode.

A final category of registers, which is at least partially visible to the user, holds condition codes (also referred to as flags). Condition codes are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result

Examples include the following:

- Segment pointers
- Index registers
- Stack pointer

• Segment pointers: In a machine with segmented addressing, a segment register holds the address of the base of the segment. There may be multiple registers: for example, one for the operating system and one for the current process.

• Index registers: These are used for indexed addressing and may be auto-indexed.

• Stack pointer: If there is user-visible stack addressing, then typically there is dedicated register that points to the top of the stack. This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.

Control and Status Registers

Control and status registers: Used by the control unit to control the operation of the processor

Four registers are essential to instruction execution:

• Program counter (PC): Contains the address of an instruction to be fetched

• Instruction register (IR): Contains the instruction most recently fetched

• Memory Address Register (MAR): Contains the address of a location in memory where a word of data to be written to or read from.

• Memory buffer register (MBR): Contains a word of data to be written to memory or the word most recently read.

Many processor designs include a register or set of registers, often known as the program status word (PSW), that contain status information.

The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

- Sign Flag
- Zero Flag
- Equal Flag
- Overflow Flag
- Interrupt Enable/Disable Flag
- Supervisor Flag

Sign: Contains the sign bit of the result of the last arithmetic operation.

Zero: Set when the result is 0.

Carry: Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations.

Equal: Set if a logical compare result is equality.

Overflow: Used to indicate arithmetic overflow.

Interrupt Enable/Disable: Used to enable or disable interrupts.

Supervisor: Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

Note: The number of registers and types of register in a processor depend on the design of a processor

KIET Group Of Institutions

General Registers Organization(COA Unit 1)

Lecture number: 9, students will learn about the following:

- ❖ A bus organization for seven CPU registers
- ❖ Control Word for any microoperation
- ❖ Encoding for Registers and Operations
- ❖ Examples of microoperation as Control Word

Notes Compiled By:

Himanshi Chaudhary
Assistant Professor
CSE

1.0 Prerequisites

- Registers
- Multiplexer
- Decoder

1.1 General Register Organization:

A bus organization for seven CPU registers is discussed here. How any Microoperation is executed on Register Organization. Block Diagram is shown in Fig. 1.

1. The output of each register is connected to two multiplexers (MUX A and MUX B) to form the two buses A and B. [Each Mux have 8 inputs: 7 Registers and 1 Input]
2. The selection lines in each multiplexer select one register or the input data for the particular bus.
3. The A and B buses form the inputs to a common arithmetic logic unit (ALU). The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed.
4. The result of the microoperation is available for output data and also goes into the inputs of all the registers.
5. The register that receives the information from the output bus is selected by a 3*8 decoder.
6. The decoder activates one of the register loads inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.
(NOTE: Numbers written on block diagram to match the steps in red)

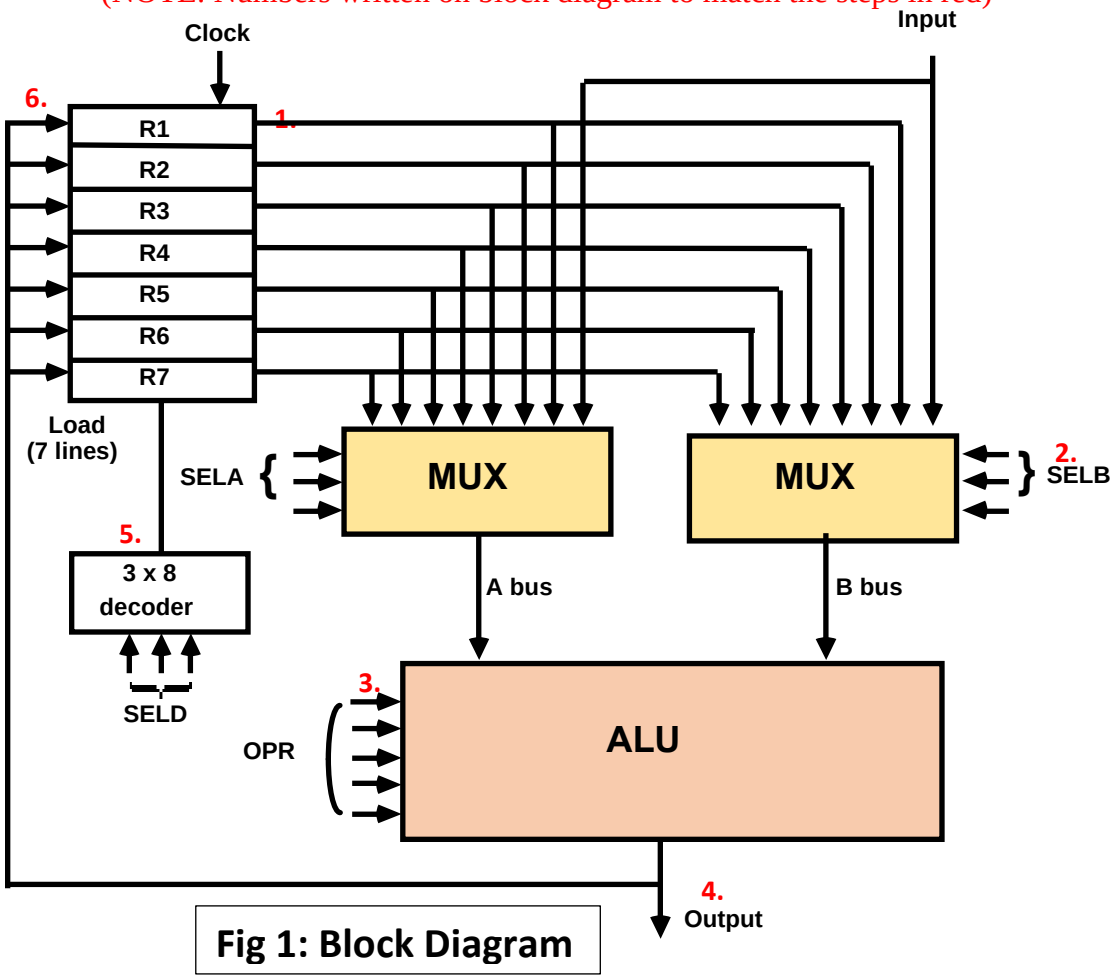


Fig 1: Block Diagram

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.

For example, to perform the operation **R1 <-- R2 + R3.**

The control must provide binary selection variables to the following selector inputs:
1. MUX A selector (SELA): to place the content of R2 into bus A. (1,2)
2. MUX B selector (SELB): to place the content o f R 3 into bus B. (1,2)
3. ALU operation selector (OPR): to provide the arithmetic addition A + B. (3,4)
4. Decoder destination selector (SELD): to transfer the content of the output bus into R1. (5,6)

(NOTE: Numbers in red are points to match Fig. 1)

1.2 Control Word

There are 14 binary selection inputs in the unit, and their combined value specifies a **control word**.

It consists of four fields. Three fields contain three bits each, and one field has five bits. The three bits of SELA select a source register for the A input of the ALU. The three bits of SELB select a register for the B input of the ALU. The three bits of SELD select a destination register using the decoder and its seven load outputs. The five bits of OPR select one of the operations in the ALU. The 14-bit control word when applied to the selection inputs specify a particular microoperation.

1.3 Encoding of Register Selection Fields

Binary Codes	SEL A	SEL B	SEL D
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

NOTE: When SELA or SELB is 000, the corresponding multiplexer selects the external input data.
When SELD = 000, no destination register is selected but the contents of the output bus are available in the external output.

1.3 Encoding of ALU OPERATION

In table below, the encoding for common operations as 5 bit binary word and Symbol is given:

Opr Select	Operation	Symbol
00000	**Transfer A**	**TSFA**
00001	**Increment A**	**INCA**
00010	**ADD A + B**	**ADD**
00101	**Subtract A - B**	**SUB**
00110	**Decrement A**	**DECA**
01000	**AND A and B**	**AND**
01010	**OR A and B**	**OR**
01100	**XOR A and B**	**XOR**
01110	**Complement A**	**COMA**
10000	**Shift right A**	**SHRA**
11000	**Shift left A**	**SHLA**

1.4 Examples of Microoperations

In table below, the encoding for some example are shown:

SEL A : Symbol of first Input register/input

SEL B : Symbol of second Input register/input

SEL D : Symbol of Output register

OPR : Symbol of Operation to be performed

Control word : Binary codes for each symbol

Symbolic Designation					
Microoperation	**SEL A**	**SEL B**	**SEL D**	**OPR**	**Control Word**
R1 ¬ R2 - R3	**R2**	**R3**	**R1**	**SUB**	**010 011 001 00101**
R4 ¬ R4 Ú R5	**R4**	**R5**	**R5**	**OR**	**100 101 100 01010**
R6 ¬ R6 + 1	**R6**	**-**	**R6**	**INCA**	**110 000 110 00001**
R7 ¬ R1	**R1**	**-**	**R7**	**TSFA**	**001 000 111 00000**
Output ¬ R2	**R2**	**-**	**NONE**	**TSFA**	**010 000 000 00000**
Output ¬ Input	**INPUT**	**-**	**NONE**	**TSFA**	**000 000 000 00000**
R4 ¬ shl R4	**R4**	**-**	**R4**	**SHLA**	**100 000 100 11000**
R5 ¬ 0	**R5**	**R5**	**R5**	**XOR**	**101 101 101 01100**

KIET Group of Institutions, Delhi-NCR, Ghaziabad

(An ISO – 9001: 2008 Certified & 'A' Grade accredited Institution by NAAC)

Department of Information Technology

Stack Organization

-Compiled by Prof. Minakshi

Topics Covered:

- **Introduction of Stack**
- **Operations to be performed onto the stack**
- **Stack pointer movement through the Stack**
- **Types of the Stack**
- **List of some applications of stack**

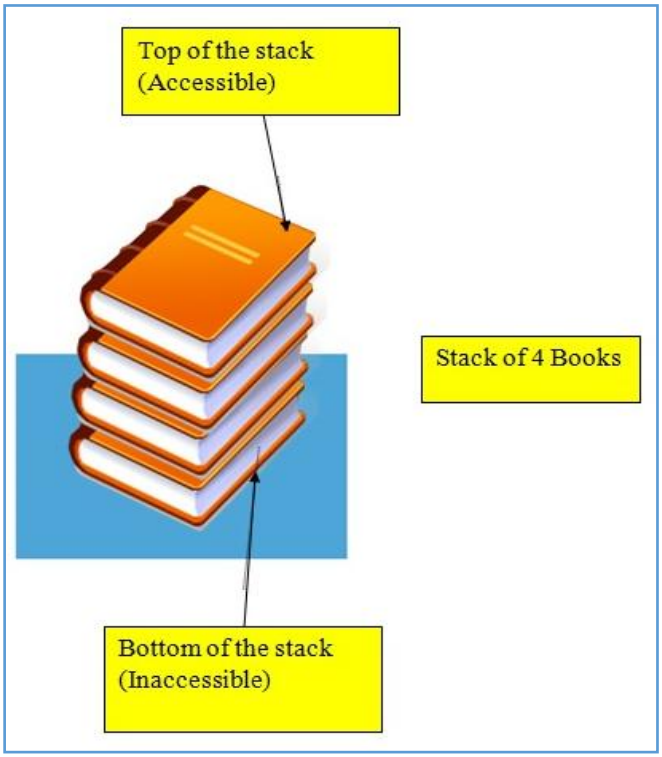
After this lecture, a student will be able to

1. **Define the stack.**
2. **Describe the working of a stack.**
3. **Insert an element into and remove an element from the stack.**
4. **Analyse the pointer movement across the ends of a stack.**
5. **Differentiate between register and memory stacks.**
6. **Design register and memory stacks**

What is Stack?

A stack is a list of data elements, usually words, with the accessing restriction that elements can be added or removed at one end of the list only. This end is called the top of the stack, and the other end is called the bottom.

The following figure shows the example stack of 4 books.



A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.

Last-in–First-out (LIFO) Principle: The last data item placed on the stack is the first one removed when retrieval begins. In other way, we may call it as First-in-Last out (FILO) principle; the first data item placed on the stack will be removed in the last.

The structure is sometimes referred to as a pushdown stack.

How to work with Stack? Or

What are the operations to be performed onto the stack?

The terms push and pop are used to describe placing a new item on the stack and removing the top item from the stack, respectively.

Push: To insert an element into Stack

Pop: To remove the topmost element from the stack

A register is used to store the address of the topmost element of the stack which is known as Stack pointer (SP). The stack pointer register SP contains a binary number whose value is equal to the address of the element that is currently on top of the stack.

What are types of Stack?

- Register Stack
- Memory Stack

Register Stack:

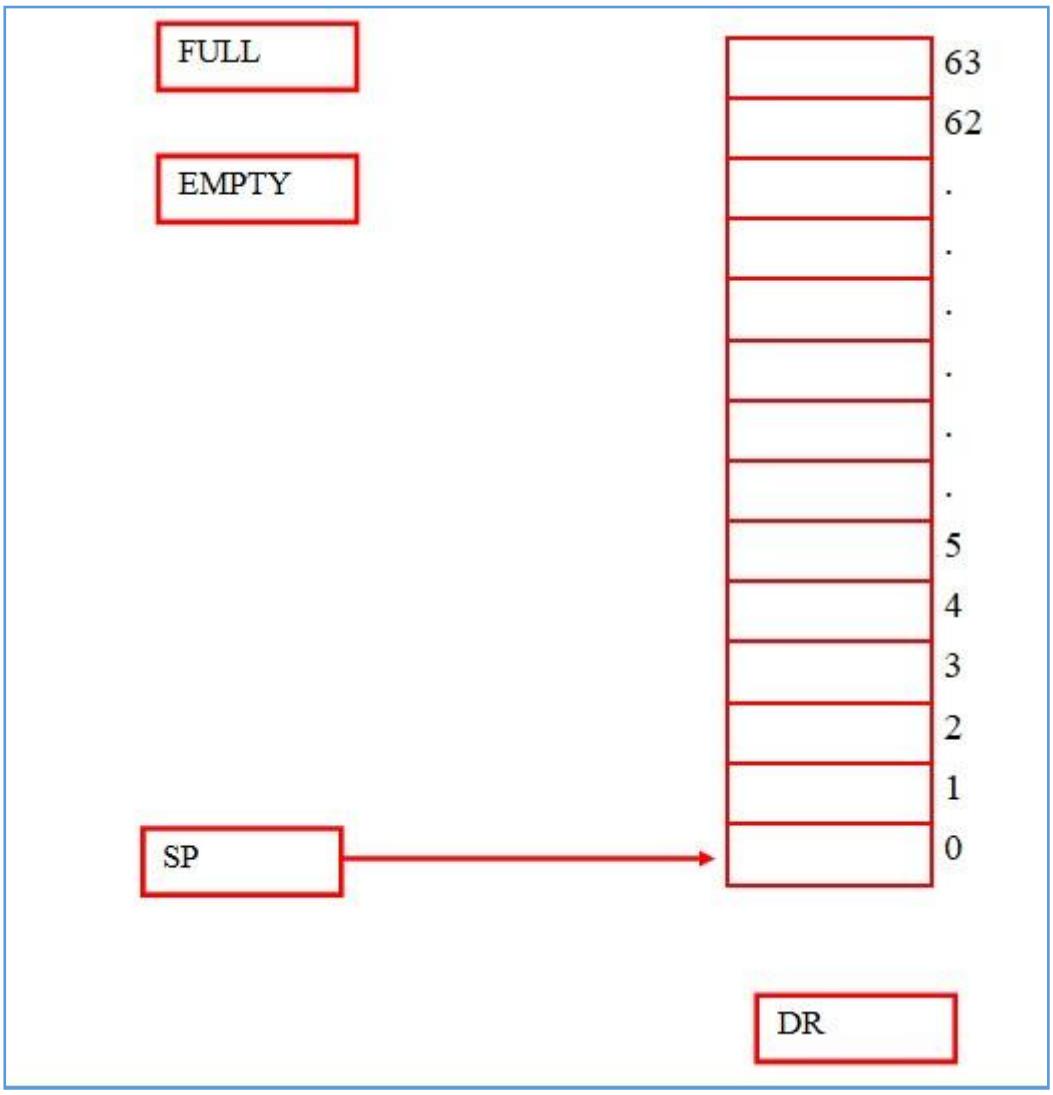
- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.
- Two one-bit registers namely FULL and EMPTY will be used to get the status of stack such as whether the stack if full (containing all the elements according to the capacity) or empty (containing no element).
- Numbering for N registers in a register stack: 0 to (N-1).

  Example: the registers will be number from 0 to 7 in 8-register stack.

Overflow: An error condition occurs during insertion a new element when stack is full. This condition is called overflow)

Underflow: An error condition occurs during removal of an element when stack is empty. This condition is also called underflow)

Example: A register stack of 64 words or registers



FULL is a one-bit flag register for showing the status of stack whether it is Full (containing all 64 elements) or not.

- FULL=0 (Stack is not full)
- FULL=1 (Stack is full)

EMPTY is also a one-bit flag register for showing the status of stack whether it is Empty (containing no element) or not

- EMPTY =0 (Stack is Not empty)
- EMPTY = 1 (Stack is Empty)

DR is the data register that holds the binary data to be written in to or read out of the stack.

Initially:

- FULL=0
- EMPTY=1
- SP=0

The above figure 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 element that is currently on top of the stack.

Suppose number of bits in register can be stored = k
Size of the register = k bits
Number of the registers in Stack = N
Then, value of k,

$$2^k = N$$
$$K = \lfloor \log_2 (N) \rfloor + 1$$

Example: In a 64-word stack, the stack pointer contains 6 bits because 2^6

Initially, SP is cleared to 0, Empty 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, a new item is inserted with a push operation.

The operation which is performed on the registers is called a micro-operation.

PUSH Operation: The push operation is implemented with the following sequence of micro-operations.

SP ←SP + 1	(Increment stack pointer)
M [SP] ← DR	(Write item on top of the stack)
if (SP=0) then (Full ← 1)	(Check if stack is full)
Empty ← 0	(Marked 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.

Note:
The first item stored in the stack is at address 1.
The second last (or prior to last) item stored in the stack is at address 63.
The last item is stored at address 0

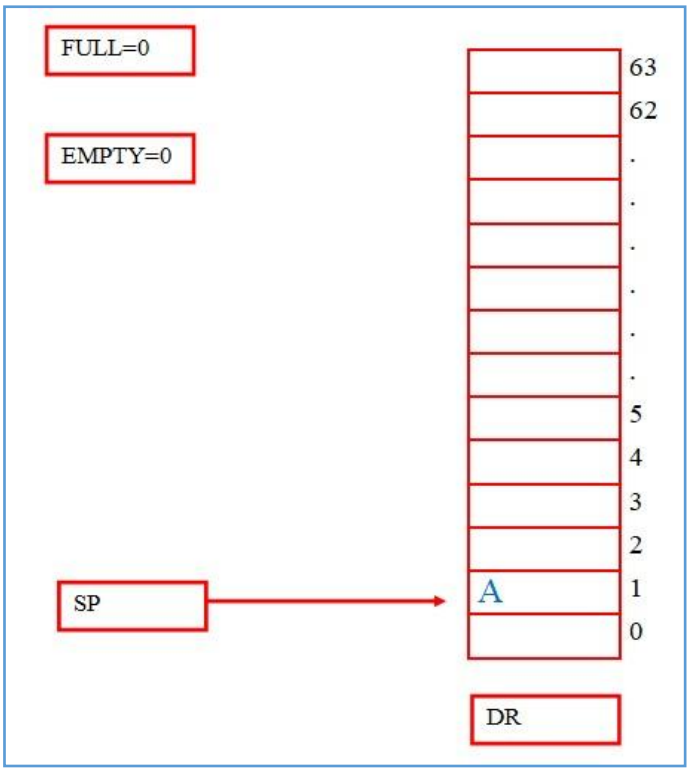
Initially, SP points to 0. The first item stored in the stack is at address 1. The last item is stored at address 0, if SP reaches 0; the stack is full of item, so FULL is set to 1. This condition is reached if the top item prior to the last push was in location 63 and after increment SP, the last item stored in location 0. Once an item is stored in location 0, there are no more empty register in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

Example: Insertion of items into the stack:

Add element 'A' to the Stack

• Initially:	• Push an element:
FULL=0 EMPTY=1 SP=0	SP= SP+1 = 0+1 =1 M[SP]← DR ➔(M[1]=A) If (SP==0) ➔No EMPTY=0

The following figure shows the Stack after inserting item 'A'



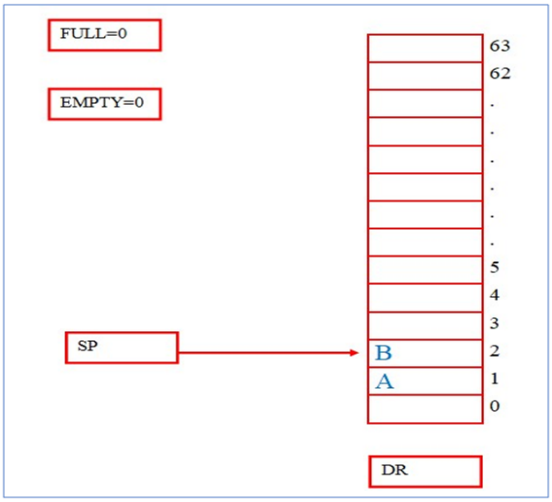
The following figure shows the Stack after inserting item 'B'

Add an element B to the Stack

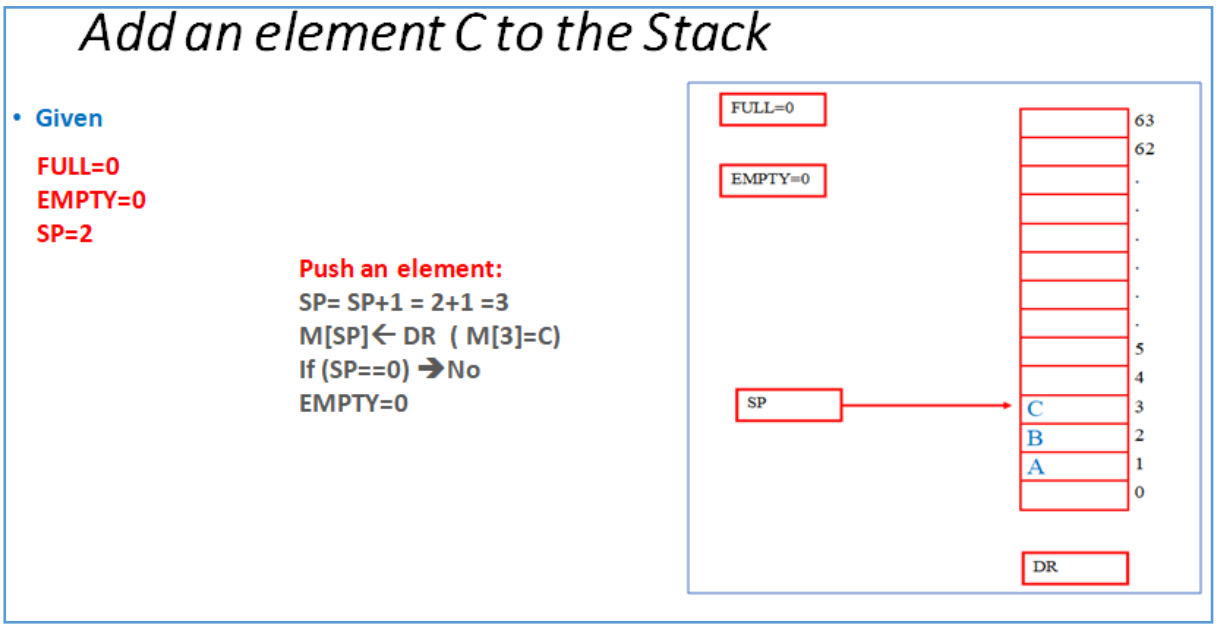
- **Given**

FULL=0
EMPTY=0
SP=1

Push an element:
SP= SP+1 = 1+1 =2
M[SP]← DR (M[2]=B)
If (SP==0) ➔No
EMPTY=0



The following figure shows the Stack after inserting item ‘C’



Three items are placed in the stack: A, B, and C in the order. Item C is on the top of the stack so the SP points to address 3.

How does a stack pointer move across the ends of the stack?

How does SP points to address 0 from address 63 after incrementing SP by 1?

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 after 63, SP is incremented by 1, the result is 0 since 111111 + 1 =1000000 in binary according to the binary addition, but SP can accommodate only the six least significant bits i.e. SP=000000.

How does SP points to address 63 from address 0 after decrementing by 1?

SP =0= 000000 (in binary)
SP=SP-1 =000000-1

The subtraction in digital computer system will be done using 2's complement method. So, the subtraction will be converted into binary addition like
SP = SP + (-1)
SP = SP + 2's complement (+1)
= 000000 + 111111
= 111111
=63

How 2's complement of 1 is calculated?

6-bit binary representation of 1 in binary is 000001
2's complement (1) = 1's complement (1) +1
= 1's complement (000001) +1
= 111110 + 1
= 111111

How does SP points to address 0 from address 63 after incrementing SP by 1?

Ans:

SP=	63		SP holds decimal value 63
SP=		111111	6-bit representation of 63
	+	000001	6-bit representation of 1
SP=SP+1		1000000	7-bit answer of (SP+1)
SP=		000000	Least significant bits will be stored in SP, 1 will be discarded

How does SP points to address 63 from address 0 after decrementing by 1?

SP=	0		SP holds decimal value 0
SP=		000000	6-bit representation of 0
Decrement by 1	-	000001	6-bit representation of 1
SP=SP-1			SP=SP+(-1)
(-1)		111111	2's Complement(1) 2's Complement(1)= 1's complement(1)+1 =111110 +1 =111111
SP=		000000	
(-1)	+	111111	
SP=SP+(-1)=	63	111111	6-bit representation of 63

POP Operation: The pop operation is implemented with the following sequence of micro-operations.

DR← M [SP]	(Read item from the top of stack)
SP ← SP-1	(Decrement stack Pointer)
If (SP=0) then (Empty ← 1)	(Check if stack is empty)
FULL ← 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 Empty 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 equal 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.

Example of removal of an element or pop operation:

• Given Full = 0 Empty= 0 SP=3 M[SP]=C	• Pop an item DR← M [SP] ➔ DR= M[3]= C SP= SP-1 ➔ SP=3-1= 2 If (SP==0) ➔ NO FULL=0

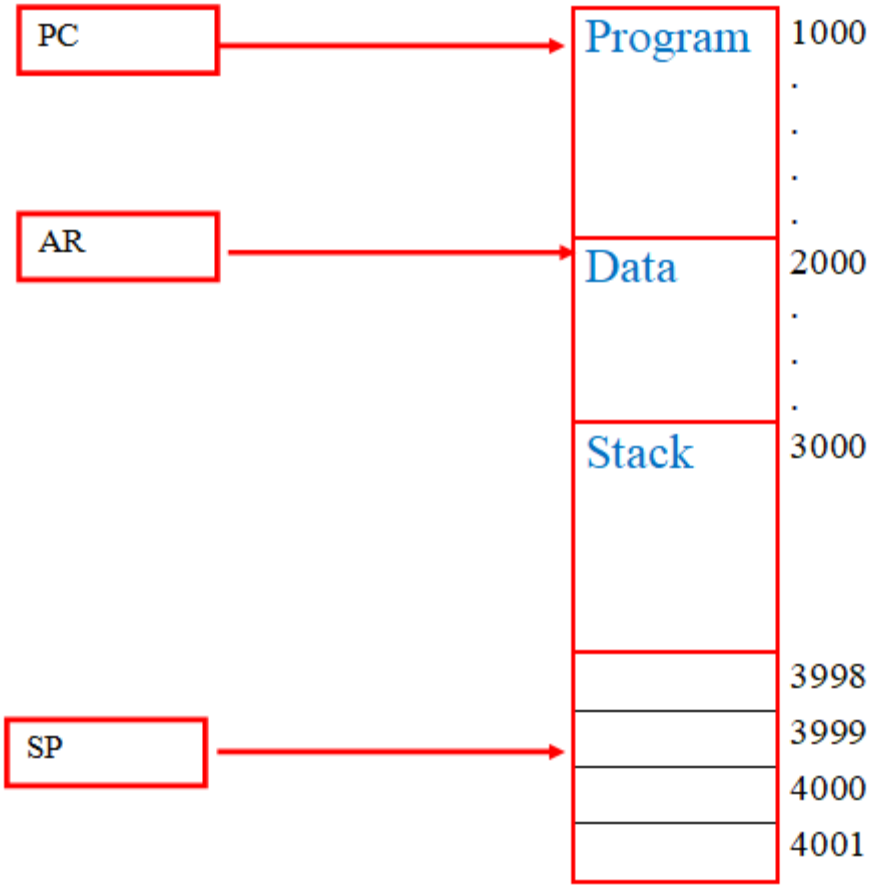
1. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address

2. Note that item C has read out but not physically removed. But it is logically out of the stack. This does not matter because when the stack is pushed, a new item is written in its place.

Memory Stack:

A stack can exist as a stand-alone unit or can be implemented in a random access memory attached to CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a Stack Pointer.

Figure given below shows a portion of computer memory partitioned in to three segments program, data and stack.



The Program Counter PC points at the address of the next instruction in the program.

The Address Register AR points at an array of data.

The Stack Pointer SP points at the top of the stack.

The three register are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or POP items into or from the stack.

The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks. We assume that the items in the stack communicate with a data register DR.

Push Operation:

A new item is inserted with the push operation as follows.

SP← SP-1 (Decrement stack Pointer)

M [SP] ← DR (Write item on top of the stack)

The stack pointer is decremented so that it points at the address of the next word. A Memory write operation inserts the word from DR into the top of the stack.

Pop Operation:

A new item is deleted with a pop operation as follows.

DR← M [SP] (Read item from the top of stack)

SP←SP + 1 (increment stack Pointer)

The top item is read from the stack in to DR. The stack pointer is then incremented to point at the next item in the stack.

Most computers do not provide hardware to check for stack overflow (FULL) or underflow (Empty). The stack limit can be checked by using two processor register:

One to hold upper limit and other hold the lower limit. After the pop or push operation SP is compared with lower or upper limit register.

Note:
In case of memory stack, stack may grow in direction of increasing addresses or decreasing addresses of memory. It depends on the design of the system.

What are the applications of a Stack in a computer system?

There are the following applications in computer architecture.

- *Subroutines Call*
- *Zero-address Instruction Implementation*
- *Infix notation to Post-fix notation conversion*
- *Evaluation of Arithmetic Expressions*

KIET Group of Institutions
Department of Information Technology
COURSE B.Tech., 3rd SEM,
Computer Organization and Architecture (KCS-302)
Session 2020-21

Address Modes: The addressing mode specifies a rule for interpreting or modifying the address field of instⁿ before the operand is actually referenced. Computer use addressing mode techniques for the purpose of accomodating one or both of the following:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counter for loop control, indexing of data and program relocation.
2. To reduce the number of bits in the addressing field of the instⁿ.

Operation Cycle of Compute :—

1. Fetch the instⁿ from memory
2. Decode the instⁿ
3. Execute the instⁿ.

Program Counter: Program counter register hold the address of the instⁿ to be executed next and is incremented each time an instⁿ is fetched from memory.

. Instruction Format

① Implied Addressing mode → In this mode, the operand are specified implicitly in the definition of the instruction. For e.g the inst^n "compliment accumulator". Zero address inst^n in a stack-organized computer are implied mode inst^n.

② Immediate Mode → In this mode, the operand is specified in the inst^n itself. The inst^n contains the actual operand rather than its address.

③ Register Mode → In this mode the operands are in registers that resides with in the CPU. The particular register is selected from a register field in the inst^n.

④ Register Indirect Mode → In this mode, the inst^n specifies a register in the CPU, whose contents give the address of operand rather than the operand itself. The advantage of register indirect mode is that the address field uses fewer bits to select a register as compare to bits needed to select a memory location.

⑤ Autoincrement or Autodecrement Mode → This is similar to register indirect mode except that the register is incremented or decremented after (or before)

Its value is used to access memory.

effective Address : is defined to be the memory address obtained from computation dictated by the given addressing mode. Effective address is the address of the operand in a computational type instⁿ.

⑥ Direct Address Mode → In this effective address is equal to the address part of the instⁿ the operand resides in memory and its address is given directly in instⁿ.

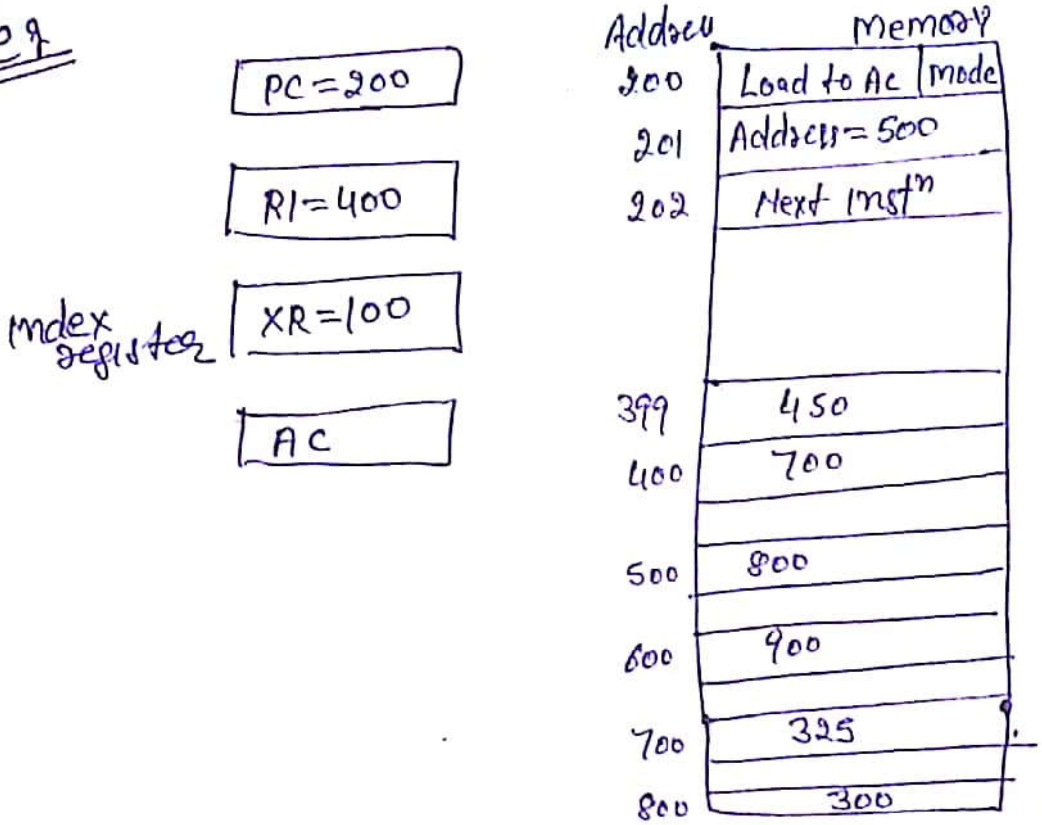
⑦ Indirect Address Mode → In this mode the address field of the instⁿ gives the address where the effective address is stored in memory. Control fetches the instⁿ, and uses its address part to access memory again to read the effective address.

⑧ Relative Address Mode → In this mode, the contents of PC are added to the address part of instⁿ in order to obtain the effective address. The address part of instⁿ is usually a signed number

⑨ Indexed Addressing Mode → In the mode, the contents of index register is added to the address part of instⁿ to obtain effective address. The index register is a special CPU register that contains the index value

(6) Base Register Addressing → In this mode, the contents of a base register is added to the address part of the instn to obtain the effective address.

eg



Numerical example for addressing mode

a) Direct Addressing

effective address is address part of instn 500 and 800 is loaded to AC.

b) Immediate: 500 is loaded to AC

c) Indirect

effective address is stored at 500 i.e 800
So operand at address 800 i.e 300 is loaded to AC

d) Relative Mode:

effective address = 500 + 202 = 702

So 325 is loade to AC

e) Index Mode

effective address = 500 + XR

= 500 + 100 = 600

So 900 is loaded to AC

F) Register Mode

operand is int R1 and 400 is loaded to AC

g) Register Indirect

effective address = 400

So 700 is loaded to AC

h) The autoincrement is same as register indirect except that R1 is incremented to 401 after execution of inst^{n}

Autodecrement

effective address = 399

So 450 is loaded to AC