UNIT I

Syllabus: Basic Structure Of Computers: Functional unit, Basic Operational concepts, Bus structures, System Software, Performance, The history of computer development.

Functional Unit (Or) Structure of a Computer System:

EveryDigital computersystems consistof fivedistinct functional units. Theseunits areas follows:

1. **Input unit**
2. **Memory unit**
3. **Arithmetic and logic unit**

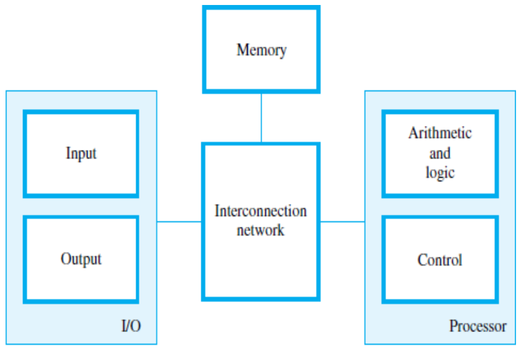


Figure 1.1 Basic functional units of a computer.

4. **Output unit**
5. **Control Unit**

These units are interconnected by electrical cables to permit communication between them. A computer must receive both data and program statements to function properly and be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device. **Computer input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU**. **Example for input devices are a keyboard, a mouse, or a scanner**.**Central Processing Unit The brain of a computer system is the central processing unit (CPU).The CPU processes data transferred to it from one of the various input devices. It then transfers either an intermediate or final result of the CPU to one or more output devices. A central control section and work areas are required to perform calculations or manipulate data.** The CPU is the computing center of the system**. It consists of a control section, an arithmetic-logic section, and an internal storage section(main memory).** Each section within the CPU serves a specific function and has a particular relationship with the other sections within the CPU.

Input Unit: An input device is usually a **keyboard or mouse**, the input device is the device through which data and instructions enter a computer.

1. **The most common input device is the *keyboard*, which accepts letters, numbers, and commandsfrom the user.**

2. **Another important type of input device is *the mouse*, which lets you select options from on-screen menus**. You use a mouse by moving it across a flat surface and pressing its buttons. A variety of other input devices work with personal computers, too:
3. **The trackball and touchpad are variations of the mouse and enable you to draw or point on the screen.**
   The joystick is a swiveling lever mounted on a stationary base that is well suited for playingvideo games

Memory unit:memory is used to store programs and data. **There are two classes of storage, calledprimaryand secondary.**

Primary storage: It is a fast memory that operates at electronic speeds. Programs must stay in memory while they are being executed.**The memory contains a large number of semiconductor storage cells, eachcapable of storing one bit of information.** To provide easy access to any word in the memory, a distinctaddress is associated with each word location. Addresses are numbers that identify successive locations. Agivenword is accessed by specifying its addressand issuing a control command.

The number of bits in each word is referred as the word length of the computer.

Typical wordlength range from 16 to 64 bits.

Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under thecontrol of theprocessor.

1. **Memory in which anylocation can be reached in a short and fixed amount of time after specifying its address is called *randomaccess memory* (RAM).**
2. **The time required to access oneword is called the *memoryaccesstime*.**
3. **The small, fast, Ram units are *called caches*. They are tightly coupled with the processor and are often contained on the same integrated circuitchip to achieve high performance**.
4. **The largest and slowest units are referred to as the *main memory*.**

Secondary storage: Secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed in frequently.
Examples for secondary storage devices are *MagneticDisks, TapeandOptical disks.*

Arithmetic-Logic Unit:-The arithmetic-logic section performs **arithmetic operations, such as addition, subtraction, multiplication, and division.**

Arithmetic-Logic Unit usually called the ALU is a digital circuit that performs two types ofoperations—arithmeticand logical.

Arithmetic operations are the fundamental mathematical operations consisting of addition, subtraction, multiplication and division.

Logical operations consist of comparisons. That is, two pieces of data are compared to see whetherone is equal to, less than, or greater than the other. The ALU is a fundamental building block of the centralprocessingunit of acomputer.

Out put Unit:-An **output device** is any piece of computer hardware equipment used to communicate theresults of data processing carried out by an information processing system (such as a computer) to the outsideworld.

In computing, input/output, or I/O, refers to the communication between an information processingsystem (such as a computer), and the outside world. Inputs are the signals or data sent to the system, and outputs are the signals or data sent by the system to the outside.

Examples of output devices:

- Speaker
- Headphones
- Screen
- Printer

\

Control Unit: All activities inside the machine are directed and controlled by the control unit. **Control Unit** is the part of the computer's central processing **unit** (CPU),which directs the **operation** of the processor. A **control unit** works by receiving input information to which it converts into **control** signals,whicharethen sent to the central processor

BASIC OPERATIONAL CONCEPTS OF COMPUTER

To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be stored are also stored in the memory.

Examples: - Add LOCA, R0

This instruction adds the operand at memory location LOCA, to operand in register R0 & places the sum into register. This instruction requires the performance of several steps,

1. First the instruction is fetched from the memory into the processor.
2. The operand at LOCA is fetched and added to the contents of R0.
3. Finally the resulting sum is stored in the register R0

The preceding add instruction combines a memory access operation with an ALU Operations. In some other type of computers, these two types of operations are performed by separate instructions for performance reasons.

Load LOCA, R1

Add R1, R0

Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory

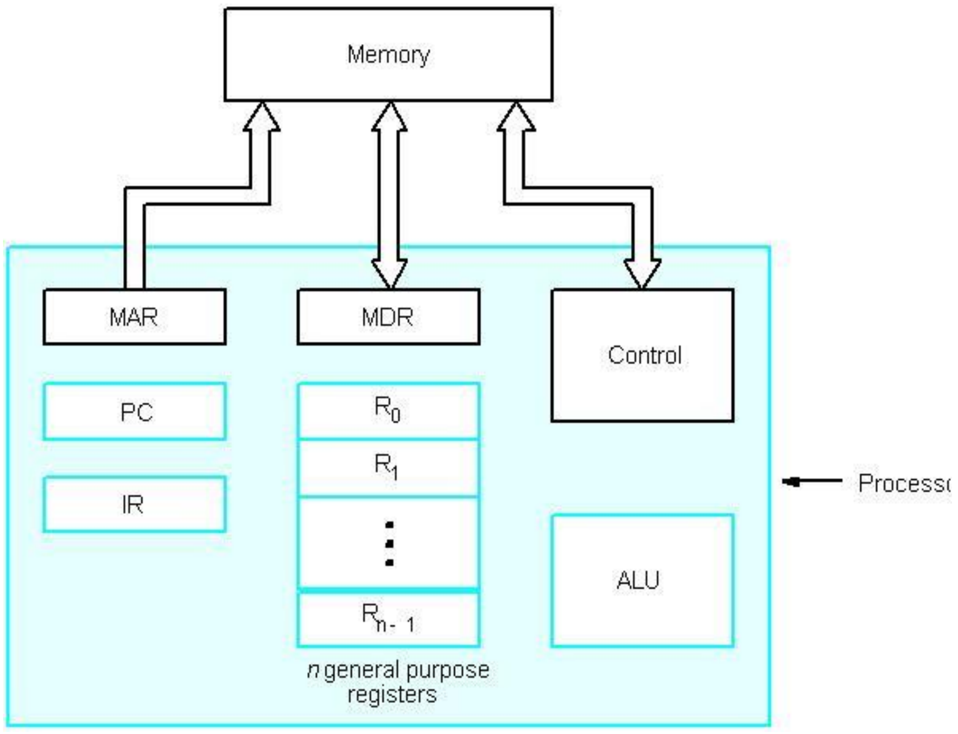


Fig 1.2 Connections between the processor and the memory

The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

The instruction register (IR):- Holds the instructions that is currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

The program counter (PC):- This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.

Besides IR and PC, there are n-general purpose registers R0 through Rn-1.

The other two registers which facilitate communication with memory are: -

1. MAR – (Memory Address Register):- It holds the address of the location to be accessed.
2. MDR – (Memory Data Register):- It contains the data to be written into or read out of the address location.

Operating steps are:-

1. Programs reside in the memory & usually get these through theInput unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desired operation.
10. If the result of this operation is to be stored in the memory, the result is sent to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated. 12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.

Normal execution of a program may be preempted (temporarily interrupted) if some devices require urgent servicing, to do this one device raises an Interrupt signal.

An interrupt is a request signal from an I/O device for service by the processor. The processor provides the requested service by executing an appropriate interrupt service routine.

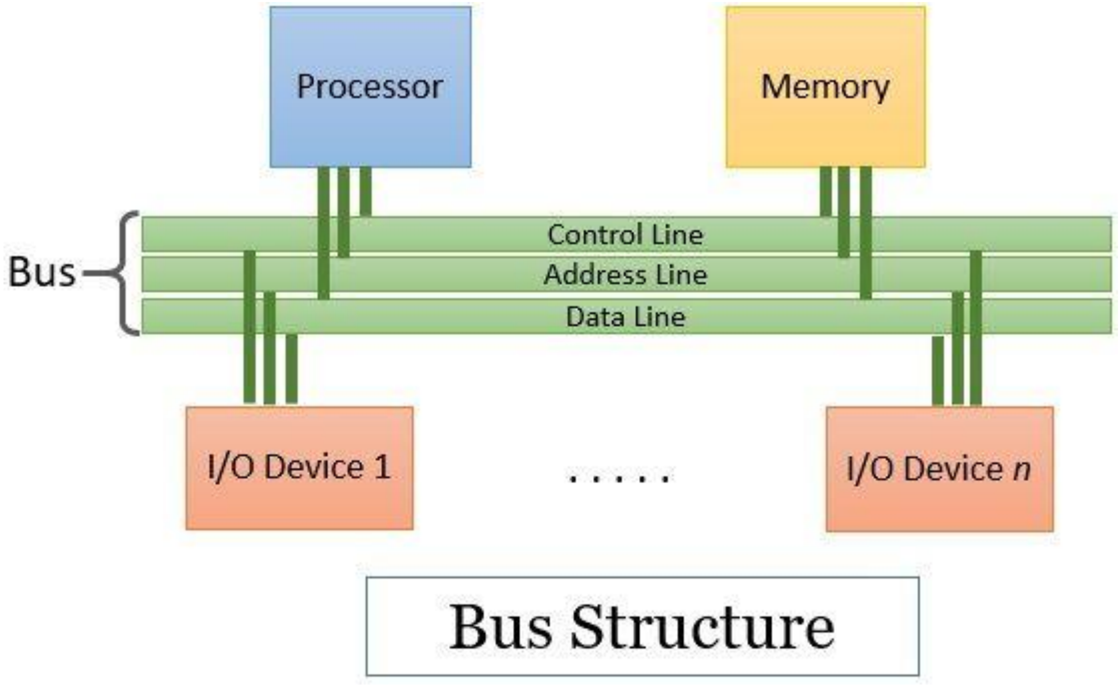
The Diversion may change the internal stage of the processor its state must be saved in the memory location before interruption. When the interrupt-routine service is completed the state of the processor is restored so that the interrupted program may continue.

Bus Structure

Bus structures in computer plays important role in connecting the internal components of the computer. The bus in the computer is the *shared transmission medium*. This means multiple components or devices use the same bus structure to transmit the information signals to each other.

At a time, only one pair of devices can use this bus to communicate with each other successfully. If multiple devices transmit the information signal over the bus at the same time, the signals overlap each other and get jumbled

A system bus has typically from fifty to hundreds of distinct lines where each line is meant for a certain function. These lines can be categorised into three functional groups i.e., data lines, address lines, and control lines. Let us discuss them one by one each.



1. Data Lines

Data lines coordinate in transferring the data among the system components. The data lines are collectively called data bus. A data bus may have 32 lines, 64 lines, 128 lines, or even more lines. The number of lines present in the data bus defines the *width* of the data bus.

Each data line is able to transfer only one bit at a time. So the number of data lines in a data bus determines how many bits it can transfer at a time. The performance of the system also depends on the width of the data bus.

2. Address Lines

The content of the address lines of the bus determines the source or destination of the data present on the data bus. The number of address lines together is referred to as the address bus. The number of address lines in the address bus determines its *width*.

The width of the address bus determines the memory capacity of the system. The content of address lines is also used for addressing I/O ports. The higher-order bits determine the bus module, and the lower-ordered bits determine the address of memory locations or I/O ports.

Whenever the processor has to read a word from memory, it simply places the address of the corresponding word on the address line.

3. Control Lines

The address lines and data lines are shared by all the components of the system, so there must be some means to control the use and access of data and address lines. The control signals placed on the control lines control the use and access to the address and data lines of the bus. The control signal consists of the *command* and *timing information.* Here the command in the control signal specifies the *operation* that has to be performed. And the timing information over the control signals specifies when the data and address information is valid.

The control lines include the lines for:

- **Memory Write:** This command causes the data on the data bus to be placed over the addressed memory location.
- **Memory Read:** This command causes the data on the addressed memory location to be placed on the data bus.
- **I/O Write:** The command over this control line causes the data on the data bus to be placed over the addressed I/O port.
- **I/O Read:** The command over this control line causes the data from the addressed I/O port to be placed over the data bus.
- **Transfer ACK:** This control line indicates the data has been received from the data bus or is placed over the data bus.
- **Bus Request:** This control line indicates that the component has requested control over the bus.
- **Bus Grant:** This control line indicates that the bus has been granted to the requesting component.
- **Interrupt Request:** This control line indicates that interrupts are pending.
- **Interrupt ACK:** This control line acknowledges when the pending interrupt is serviced.
- **Clock:** This control line is used to synchronize the operations.
- **Reset:** The bit information over this control line initializes all the modules.

Suppose a component connected to the bus wishes to send data to another connected component. In that case, it first has to acquire control over the bus, and then it can transfer the data to another component over the bus. The same happens when a component request data from another component.

During data transfer between two components, one component act as a master and the other act as a slave. The device initiating the data transfer is referred to as the *master,* and usually, it is a processor, or sometimes it may be some other device or component. The component addressed by the master component is referred to as a *slave*.

Timing in Bus

As we have seen, the bus's control lines also provide timing information along with the command. Well, the way of deriving the timing information over the control line can be categorized in two ways:

1. Synchronous Bus

With the synchronous bus scheme, all the devices or components connected to the bus derive timing information over the control line referred to as the *bus clock.* Over the bus clock line, the clock transmits an alternating sequence of 1s and 0s at regular intervals. A single 1-0 transmission is considered a clock or bus cycle.

All the devices or components connected to the bus can read this bus clock line, and all the events start at the starting the clock cycle. Here the transmitting component and the receiving component are synchronized using the clock. The data is sent or received constantly and, therefore used for high-speed transmission.

2. Asynchronous Bus

The clock does not synchronise the transmitter and receiver components in this asynchronous bus scheme. Instead, the data transfer is controlled using a handshake protocol between the master component and the slave component.

Here, the component initiating the data transfer i.e. master component then gets ready for data transfer, and indicates this by activating its master-ready line and placing the address and command information over the bus.

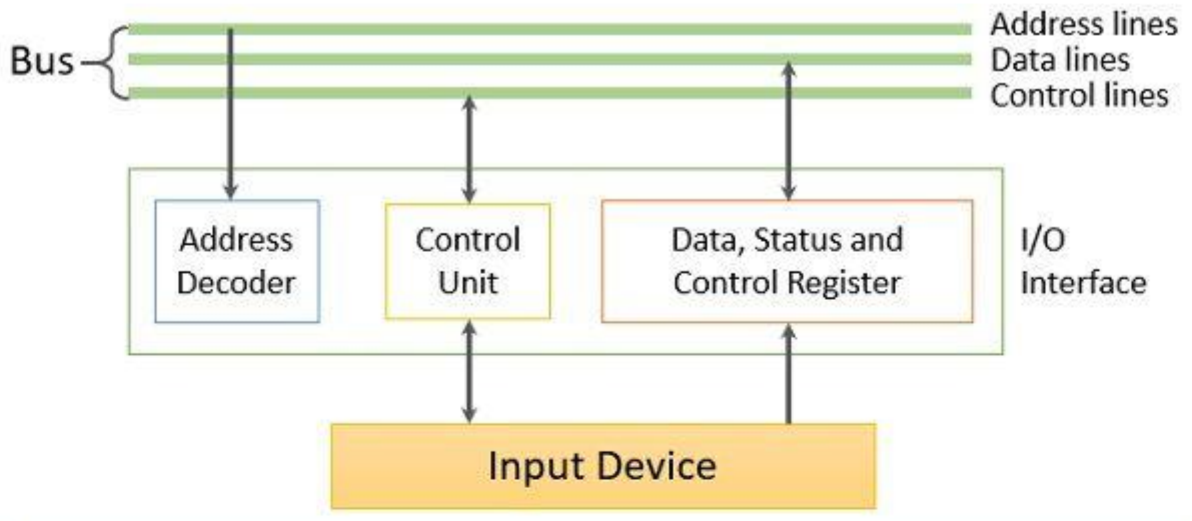
Then all the connected component decodes the address on the address line to recognize which component is being addressed by the master component.

Now the addressed component performs the required operation and notifies the processor by activating its slave ready-line. Once the master recognizes the activated slave ready-line, it removes its control over the bus.

In this way, the occurrence of one event on the bus is followed by and depends on the occurrence of a previous event.

I/O Interface of Input Device Connected to Bus

In the section ahead, we will discuss how these three things over the bus are decoded by the I/O interface.



Bus Structure for I/O Interface of an Input Device

Each I/O device interface has a set of registers with unique addresses. Whenever the processor puts an address over the address line of the bus, it is examined by all the devices connected to the bus. Whichever device recognizes this address responds to the control operations issued on the control lines of the bus.

The processor issues read/write operation over the control lines of the bus, and the data corresponding to the read and write operation is transmitted over the data lines of the bus. Consider that we have an input device keyboard, and some data from the keyboard has to be issued to the processor; the machine instruction for the same is:

LOAD R2, DATAIN

Here the DATAIN is the data register of the Keyboard. This instruction reads the content from the DATAIN register of the keyboard and transfers the content to the R2 register of the processor. Similarly, the instruction below:

STORE R2, DATAOUT

Here consider that the DATAOUT register is the data register of a display device. So, this instruction will transfer the content of register R2 of the processor to the data register of the display device.

The control register and the status register of the I/O device interface have information relevant to the operation performed on or by the I/O device. The address decoder, control circuitry and the registers of the I/O interface coordinate in transferring the content to or from the I/O device.

So, this is all about the bus structure in computer architecture. We have seen how the bus plays a vital role in data transfer.

Software:- A total computer system includes both software and Hardware.

1. **Hardware consists of physical components** and all associated equipment.
2. **Software refers to the collection programs that are written for the computer** and writing a program for a computer consists of specifying, directly or indirectly a sequence of machine instructions.
3. The computer software consists of the instructions and data that the computer manipulates to perform various data processing tasks.

Types:

1. Application software,
2. System software

System software: System software is used to run application software.

System software is a collection of programs that are executed as needed to perform functions such as

1. **Receiving and interpreting user commands.**
2. **Entering and editing application programs and sorting them as files in secondary storage devices.(Editor)**
3. **Managing the storage and retrieval of files in secondary storage devices.**
4. **Running standard application programs such as wordprocessors, spreadsheets, or games, with data supplied by the user.**
5. **Controlling I/O units to receive input information and produce output results.**
6. **Translating programs from high level language to low level language.(Assemblers)**
7. **Linking and running user-written application program with existing standard libraryroutines, such as numerical computation packages.(Linker)**

Application software: Application software allows end users to accomplish one or more specific (notdirectly computer development related) tasks. Its usually written in high level languages, such as c, c++, java. Typical applications include:

- **Wordprocessing**
- **spreadsheet**
- **computer games**
- **databases**
- **industrial automation**
- **business software**
- quantum chemistry and solid state physics software
- telecommunications(i.e.,theinternetand everythingthatflowsonit)
- educational software
- medical software
- military software
- molecular modelling software
- imageediting
- simulationsoftware
- Decisionmakingsoftware

Compiler:- A **compiler** is a computer program (or set of programs) that **transforms source code written ina computer language (the *source language*) into another computer language (the *target language*, oftenhaving a binary form known as *object code*).** The most common reason for wanting to transform source code is to create an executable program. The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language ormachine code). A program that translates from a low level language to a higher level one is a *decompiler*. A program that translates between high-level languages is usually called a *language translator, source to source translator*

Linker:- Linker is a program in a system which helps to link a object modules of program into a single object file. It performs the process of linking. Linker are also called link editors. Linking is process ofcollecting and maintaining piece of code and data into a single file. Linker also link a particular module intosystem library. It takes object modules from assembler as input and forms an executable file as output forloader. Linking is performed at both compile time, when the source code is translated into machine code andload time, when the program is loaded into memory by the loader. Linking is performed at the last step incompilingaprogram.

Assembler: - An assembler is a program that converts assembly language into machine code. **It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor**. Assemblers are similar to compilers in that they produceexecutable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code. Since each assembly language is designed for a specific processor, assembling a program is performed using a simple one-to-one mapping from assembly code to machine code.

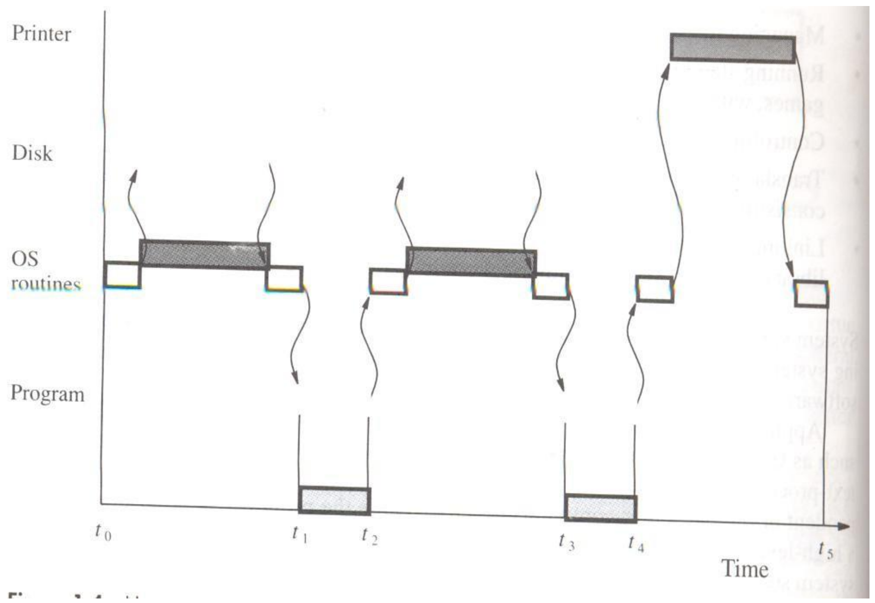
Loader:-A loader is a major component of an operating system that ensures all necessary programs andlibraries are loaded, which is essential during the startup phase of running a program. It places the librariesandprograms into themain memory in order to preparethem forexecution.

System software Component->OS (OPERATING SYSTEM)

Operating System: It is a large program or a collection of routines that is used to control the sharing of and interaction among various computer units.

Functionsof OS:

- Assign resources to individual application program.
- Assign memory and magnetic disk space to program and data files.
- ♦ Move the data between memory and disk units. Handles I/O operations.



User program and os routine sharing of the processor

Steps:-

1) The first step is to transfer the file into memory.

2) When the transfer is completed, the execution of the program starts.

3) During time period t0 to t1, an OS routine initiates loading the application program from disk to memory, wait until the transfer is complete and then passes theexecutioncontrol to the application program & print the results.
4) Similar action takes place during„t2‟to„t3‟ and „t4‟to„t5‟.
5) At „t5‟,Operating System may load and execute another application program.
6) Similarly during„ t0‟to„t1‟, the Operating System can arrange to print the previous program's results while the current program is beingexecuted.
7) The pattern of managing the concurrent execution of the several applicationprograms to make the best possible use of computer resources is called the multi-programming or multi-tasking.

Performance

Performance: -The most important measure of the performance of a computer is how quickly it cancompute programs. The speed with which a computer executes programs is affected by the design of its hardware and its machine language instructions. To represent the performance of a processor, we should consider only the periods during which the processor is active.

At the start of execution, all program instructions and the required **data are stored in the memory as shown below.As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache**. **When the execution of instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache**. Later, if the same instruction or data item is needed a second time, it is read directly from the cache.

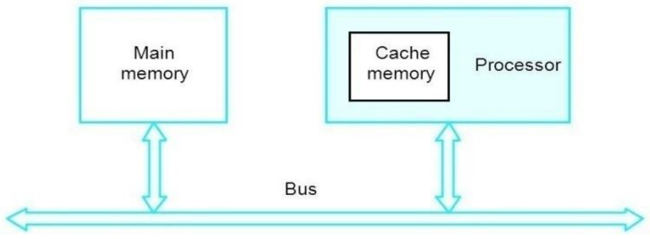


Figure 1.5. The processor cache.

Computer performance is often described in terms of clock speed (usually in MHz or GHz). This refers to the cycles per second of the main clock of the CPU. Performance of a computer depends on the following factors.

a) Processorclock:-

1. Processor circuits are controlled by a timing signal called a clock.A clock is a microchip thatregulatesspeed and timing ofall computer functions.
2. **Clock Cycle** is the speed of a computer processor, or CPU, which is the amount of time between twopulses of an oscillator. Generally speaking, the higher number of pulses per second, the faster the computer processor will be able to process information
3. CPU **clockspeed**, or **clockrate**, is measured in Hertz—generally in gigahertz, or GHz. A CPU's **clock speed rate** is a measure of how many **clock** cycles a CPU can perform per second
4. To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.
5. The length P of one clock cycle is an important parameter that affects processor performance.
6. Itsinverseis theclock rate, R=1/P, whichis measuredincycles persecond.
7. If the clock rate is 500(MHz) million cycles per second, then the corresponding clock period is 2nanoseconds.

b) Basic performance equation:-The **Performance Equation** is a term used in computer science. It refers to the calculation of the performance or speed of a centralprocessing unit (CPU).

Basically the *Basic Performance Equation[BPE]* is an equation with 3 parameters which are required for the calculation of "Basic Performance" of a given system. It is given by

T=(N*S)/R

Where **'T'** is the ***processor time*** [Program Execution Time] required to execute a given program written in some high level language. The compiler generates a machine language object program corresponding to the source program.

'N' is the **total number of steps required to complete program execution**. 'N' is the actual number of instruction executions, not necessarily equal to the total number of machine language instructions in theobject program. Some instructions are executed more than others (loops) and some are not executed at all(conditions).

'S' is the **average number of basic steps each instruction execution requires**, where each basic step is completed in one clock cycle. We say average as each instruction contains a variable number of stepsdependingon theinstruction.

'R'is the**clock rate**[In cyclespersecond]

c) <u>PipeliningandSuperscalaroperation:-</u>

1. A substantial improvement in performance can be achieved by overlapping the execution of successive instructions, using a technique called pipelining.
2. Considertheinstruction
3. Add R1, R2, R3
4. Which adds the contents of registers R1 and R2, and places the sum intoR3
5. The contents of R1 and R2 are first transferred to the inputs of the ALU.
6. After the add operation is performed, the sum is transferred toR3.
7. Processor can read the next instructionfromthememorywhiletheadditionoperationisbeingperformed.
8. Then, if that instruction also uses the ALU, its operands can be transferred to the ALU inputs at the same time that the result of add instruction is being transferred to R3.
9. Thus, pipelining increases the rate of executing instructions significantly.

d) Superscalaroperation:-

1. A higher degree of concurrency can be achieved if multiple instruction pipelines are implemented in the processor.
2. This means that multiple function units are used, creating parallel paths through which different instructions can be executed in parallel.
3. With such an arrangement, it becomes possible to start the execution of several instructions in every clock cycle.
4. This mode of execution is called super scalar operation.

e)Clockrate:-

1. There are two possibilities for increasing the clock rate, R.
2. First, improving the Integrated Circuittechnology makes logic circuit faster, which reduces theneeded to complete a basic step.This allows the clock period, P, to be reduced and the clock rate, R, to be increased.
3. Second, reducing the amount of processing done in one basic step also makes it possible to reduce theclockperiod, P.

f)Instructionset: CISCand RISC:-

1. The terms CISC and RISC refer to design principles and techniques.
2. RISC: Reduced instruction set computers.
3. Simple instructions require a small number of basic steps to execute.
4. For a processor that has only simple instructions, a large number of instructions may be need to perform a given programming task. This could lead to a large value of N and a small value for S.
5. It is much easier to implement efficient pipelining in processors with simple instruction sets.
6. CISC: Complex instruction set computers.
7. Complex instructions involve a large number of steps.
8. If individual instructions perform more complex operations, fewer instructions will be needed, leading to a lower value of N and a larger value of S.
9. Complex instructions combined with pipelining would achieve good performance.

g)OptimizingCompiler:-

1. A compiler translates a high-level language program into a sequence of machine instructions.
2. To reduce N, we need to have a suitable machine instruction set and a compiler that makes good use of it.
3. An optimizing compiler takes advantage of various features of the target processor to reduce the product N* S.
4. The compiler may rearrange program instructions to achieve better performance.

h)Performancemeasurement:-

1. SPECrating.
2. A non profit organization called ”System Performance Evaluation Corporation”(SPEC) selects and publishes representative application programs for different application domains.
3. The SPEC rating is computed as follows.
4. SPEC rating=$\frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test.}}$
5. Thus SPEC rating of 50 means that the computer under test is 50times faster than the reference computer for this particular benchmarks.
6. The test is repeated for all the programs in the SPEC suite, and the geometric mean of the results is computed.
7. Let SPEC, be the rating for program ‘i’ in the suite. The overall SPEC rating for the computer is given by

   $$\text{SPEC rating} = \prod_{i=1}^{n} (\text{SPEC}_i)$$

   Where n is the number of programs in the suite.

The history of computer development

Computer Types: Basing capacity, technology used and performance of computer, they are classified into two types

→According to computational ability
→According to generation

According to computational ability (Based on Size, cost and performance): There are mainly 4 types of computers. These include:

a) Micro computers
b) Mainframe computers
c) Mini computers
d) Super computer

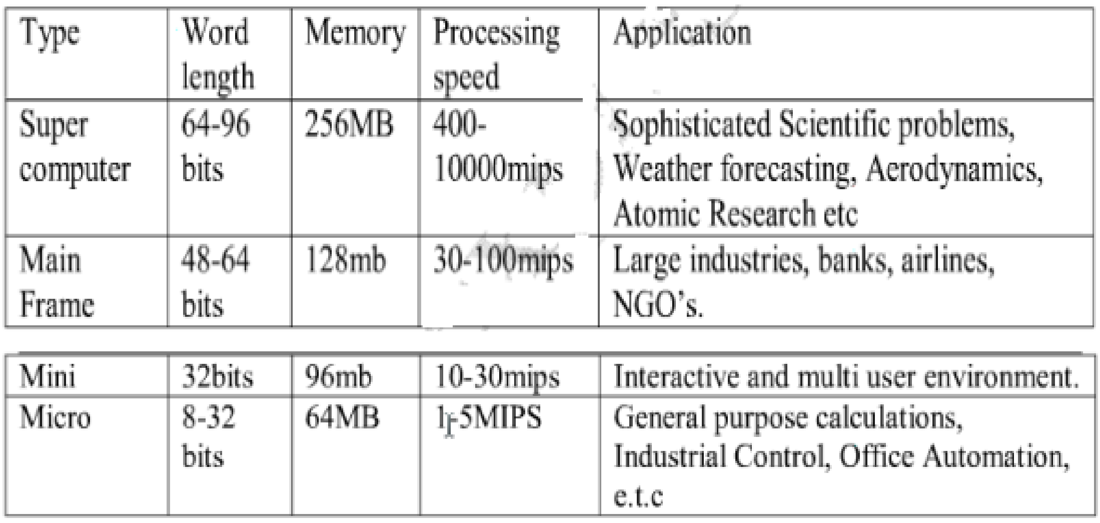
a) **Micro computers: -** Micro computers are the most common type of computers in existence today, whether at work in school or on the desk at home. These computers include:
    1. Desktop computer
    2. Personal digital assistants (more commonly known as PDA's)
    3. Palmtop computers
    4. Laptop and notebook computers

    Micro computers were the smallest, least powerful and least expensive of the computers of the time. The first Micro computers could only perform one task at a time, while bigger computers ran multi-tasking operating systems, and served multiple users. Referred to as a personal computer or "desktop computer", Micro computers are generally meant to service one user (person) at a time. By the late 1990s, all personal computers run a multi-tasking operating system, but are still intended for a single user.

b) **Mainframe Computers :-** The term Mainframe computer was created to distinguish the traditional, large, institutional computer intended to service multiple users from the smaller, single user machines. These computers are capable of handling and processing very large amounts of data easily and quickly. A mainframe speed is so fast that it is measured in millions of tasks per milliseconds (MTM). While other computers became smaller, Mainframe computers stayed large to maintain the ever growing memory capacity and speed.
   Mainframe computers are used in large institutions such as government, banks and large corporations. These institutions were early adopters of computer use, long before personal computers were available to individuals. "Mainframe" often refers to computers compatible with the computer architectures established in the 1960's. Thus, the origin of the architecture also affects the classification, not just processing power.

c) Mini Computers / Workstation :- Mini computers, or Workstations, were computers that are one step above the micro or personal computers and a step below mainframe computers. They are intended to serve one user, but contain special hardware enhancements not found on a personal computer. They run operating systems that are normally associated with mainframe computers, usually one of the variants of the UNIX operating system.

d) Super Computer:- A Super computer is a specialized variation of the mainframe. Where a mainframe is intended to perform many tasks, a Super computer tends to focus on performing a single program of intense numerical calculations. Weather forecasting systems, Automobile design systems, extreme graphic generator for example, are usually based on super computers.



Type	Word length	Memory	Processing speed	Application
Super computer	64-96 bits	256MB	400-10000mips	Sophisticated Scientific problems, Weather forecasting, Aerodynamics, Atomic Research etc
Main Frame	48-64 bits	128mb	30-100mips	Large industries, banks, airlines, NGO's.
Mini	32bits	96mb	10-30mips	Interactive and multi user environment.
Micro	8-32 bits	64MB	1-5MIPS	General purpose calculations, Industrial Control, Office Automation, e.t.c

According to Generations of Computers:
The history of computer development is often referred to in reference to the different generations of computing devices. Each generation of computer is characterized by a major technological development that fundamentally changed the way computers operate, resulting in increasingly smaller, cheaper, more powerful and more efficient and reliable devices.

a) **First Generation (1940-1956): Vacuum Tubes: The first computers used vacuum tubes for circuitry and magnetic drums for memory, and were often enormous, taking up entire rooms.** They were very expensive to operate and in addition to using a great deal of electricity, generated a lot of heat, which was often the cause of malfunctions. First generation computers relied on machine language, the lowest-level programming language understood by computers, to perform operations, and they could only solve one problem at a time.

   Input was based on punched cards and paper tape, and output was displayed on printouts. Example: The UNIVAC and ENIAC computers are examples of first-generation computing devices. The UNIVAC was the first commercial computer delivered to a business client, the U.S. Census Bureau in 1951.

b) **Second Generation (1956-1963): Transistors:- Transistors replaced vacuum tubes and ushered in the second generation of computers**. The transistor was invented in 1947 but did not see widespread use in computers until the late 1950s. The transistor was far superior to the vacuum tube, allowing computers to become smaller, faster, cheaper, more energy-efficient and more reliable than their first-generation predecessors. Though the transistor still generated a great deal of heat that subjected the computer to damage, it was a vast improvement over the vacuum tube. Second-generation computers still relied on punched cards for input and printouts for output.
   Second-generation computers moved from cryptic binary machine language to symbolic, or assembly, languages, which allowed programmers to specify instructions in words. High-level programming languages were also being developed at this time, such as early versions of COBOL and FORTRAN. These were also the first computers that stored their instructions in their memory, which moved from a magnetic drum to magnetic core technology. The first computers of this generation were developed for the atomic energy industry

c) **Third Generation(1964-1971):IntegratedCircuits: The development of the integrated circuit** was the hallmark of the third generation of computers.Transistors were miniaturized and placed on silicon chips, called semiconductors, which drastically increased the speed and efficiency of computers. Instead of punched cards and printouts, users interacted with third generation computers through keyboards and monitors and interfaced with an operating system, which allowed the device to run many different applications at one time with a central program that monitored the memory. Computers for the first time became accessible to a mass audience because they were smaller and cheaper than the earlier.

d) **Fourth Generation (1971-Present):Microprocessors:** The microprocessor brought the fourth generation of computers, as thousands **of integrated circuitswere built onto a single silicon chip**. What in the first generation filled an entire room could now fit in the palm of the hand. **The Intel 4004 chip**, developed in 1971, located all the components of the computer—fromthecentral processing unit and memory to input/output controls—onasinglechip.
   In 1981 IBM introduced its first computer for the home user, and in 1984 Apple introduced the Macintosh. Microprocessors also moved out of the realm of desktop computers and into many areas of life as more and more everyday products began to usemicroprocessors.
   As these small computers became more powerful, they could be linked together to form networks, which eventually led to the development of the Internet. Fourth generation computers also saw the development of GUIs, the mouse and handheld devices.

e) **FifthGeneration(PresentandBeyond):ArtificialIntelligence):**Fifth generation computing devices, based on artificial intelligence, are still in development, though there are some applications, such as voice recognition, that are being used today.

Theuse of parallelprocessing and superconductors is helping to make artificial intelligence a reality. Quantum computation andmolecular and nanotechnology will radically change the face of computers in years to come. The goal offifth-generation computing is to develop devices that respond to natural language input and are capable of learning and self-organization

.

Unit-2

Instruction:-The tasks carried out by a computer program consists of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen. Basic computer instructions are the elementary operations that a computer system can perform. These instructions are typically divided into three categories: data transfer instructions, arithmetic and logic instructions, and control instructions.

- Data transfer instructions are used to move data between different parts of the computer system. These instructions include load and store instructions, which move data between memory and the CPU, and input/output (I/O) instructions, which move data between the CPU and external devices.
- Arithmetic and logic instructions are used to perform mathematical operations and logical operations on data stored in the system. These instructions include add, subtract, multiply, and divide instructions, as well as logic instructions such as AND, OR, and NOT.
- Control instructions are used to control the flow of instructions within the computer system. These instructions include branch instructions, which transfer control to different parts of the program based on specified conditions, and jump instructions, which transfer control to a specified memory location.

Register Transfer Notation:- Register transfer notation used to describe how data is passed between CPU registers during the execution of instructions. It is written in human readable format as close to assembly language as possible.

- ➢ Transfer of information from one location in the computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem.
- ➢ Most of the time, we identify a location by a symbolic name standing for its hardware binary address. Example, names for the addresses of memory locations may be LOC, PLACE, A, VAR2;
- ➢ processor registers names may be R0, R5; and I/O register names may be DATAIN, OUTSTATUS, and so on.
- ➢ The contents of a location are denoted by placing square brackets around the name of the location. Thus, the expression R1<= [LOC] Means that the contents of memory location LOC are transferred into processor register R1.
- ➢ As another example, consider the operation that adds the contents of registers R1 and R2, and then places their sum into register R3. This action is indicated as R3<= [R1] [R2]. This type of notation is known as Register Transfer Notation (RTN). Note that the right-hand side of an RTN expression always denotes a value, and the left- hand side is the name of a location where the value is to be placed, overwriting the old contents of that location.

Assembly Language Notation:- Another type of notation to represent machine instructions and programs. For this, we use an assembly language format. For example, an instruction that causes the transfer described above, from memory location LOC to processor register R1, is specified by the statement Move LOC, R1

- The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1are overwritten.
- The second example of adding two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement

  Add R1, R2, R3

Basic Instruction Formats:-
The instruction formats are a sequence of bits (0 and 1). These bits, when grouped, are known as fields. Each field of the machine provides specific information to the CPU related to the operation and location of the data.
The instruction format also defines the layout of the bits for an instruction. It can be of variable lengths with multiple numbers of addresses. These address fields in the instruction format vary as per the organization of the registers in the CPU. The formats supported by the CPU depend upon the Instructions Set Architecture implemented by the processor.
Depending on the multiple address fields, the instruction is categorized as follows:

1. Three address instruction
2. Two address instruction
3. One address instruction
4. Zero address instruction

The operations specified by a computer instruction are executed on data stored in memory or processor registers. The operands residing in processor registers are specified with an address. The registered address is a binary number of k bits that defines one of the 2^k registers in the CPU. Thus, a CPU with 16 processors registers R0 through R15 and will have a four-bit register address field.
Example: The binary number 0011 will designate register R3.

A computer can have instructions of different lengths containing varying numbers of addresses. The number of address fields of a computer depends on the internal design of its registers. Most of the computers fall into one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

1. Single Accumulator Organization
All the operations on a system are performed with an implied accumulator register. The instruction format in this type of computer uses **one address field**.
For example, the instruction for arithmetic addition is defined by an assembly language instruction 'ADD.'
Where X is the operand's address, the ADD instruction results in the operation.
$AC \leftarrow AC + M[X]$.
AC is the accumulator register, M[X] symbolizes the memory word located at address X.

2. General Register Organization
The general register type computers employ **two or three address fields** in their instruction format. Each address field specifies a processor register or a memory. An instruction symbolized by ADD R1, X specifies the operation $R1 \leftarrow R + M[X]$.
This instruction has two address fields: register R1 and memory address X.

3. Stack Organization
A computer with a stack organization has PUSH and POP instructions that require an address field. Hence, the instruction PUSH X pushes the word at address X to the top of the stack. The stack pointer updates automatically. In stack-organized computers, the operation type instructions don't require an address field as the operation is performed on the two items on the top of the stack.

Types of Instruction Formats

1. Zero Address Instruction

This instruction does not have an operand field, and the location of operands is implicitly represented. The stack-organized computer system supports these instructions. To evaluate the arithmetic expression, it is required to convert it into reverse polish notation.

Example: Consider the below operations, which shows how X = (A + B) ∗ (C + D) expression will be written for a stack-organized computer.

TOS: Top of the Stack

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

2. One Address Instruction

This instruction uses an implied accumulator for data manipulation operations. An accumulator is a register used by the CPU to perform logical operations. In one address instruction, the accumulator is implied, and hence, it does not require an explicit reference. For multiplication and division, there is a need for a second register. However, here we will neglect the second register and assume that the accumulator contains the result of all the operations.

Example: The program to evaluate X = (A + B) ∗ (C + D) is as follows:

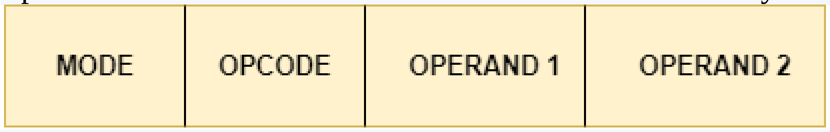
LOAD	A	AC ← M [A]
ADD	B	AC ← A [C] + M [B]
STORE	T	M [T] ← AC
LOAD	C	AC ← M [C]
ADD	D	AC ← AC + M [D]
MUL	T	AC ← AC ∗ M [T]
STORE	X	M [X] ← AC

All operations are done between the accumulator(AC) register and a memory operand.
M[ ] is any memory location. M[T] addresses a temporary memory location for storing the intermediate result.
This instruction format has only one operand field. This address field uses two special instructions to perform data transfer, namely:

- LOAD: This is used to transfer the data to the accumulator.
- STORE: This is used to move the data from the accumulator to the memory.

3. Two Address Instructions

This instruction is most commonly used in commercial computers. This address instruction format has three operand fields. The two address fields can either be memory addresses or registers.



MODE	OPCODE	OPERAND 1	OPERAND 2

Example: The program to evaluate X = (A + B) ∗ (C + D) is as follows:

MOV	R1, A	R1 ← M [A]
ADD	R1, B	R1 ← R1 + M [B]

```
MOV    R2, C      R2 ← M [C]
ADD    R2, D      R2 ← R2 + M [D]
MUL    R1, R2     R1 ← R1∗R2
MOV    X, R1      M [X] ← R1
```

The MOV instruction transfers the operands to the memory from the processor registers. R1, R2 registers.

4. Three Address Instruction

The format of a three address instruction requires three operand fields. These three fields can be either memory addresses or registers.

MODE	OPCODE	OPERAND 1	OPERAND 2	OPERAND 3

Example: The program in assembly language X = (A + B) ∗ (C + D) Consider the instructions given below that explain each instruction's register transfer operation.

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

Two processor registers, R1 and R2.

The symbol M [A] denotes the operand at memory address symbolized by A. The operand1 and operand2 contain the data or address that the CPU will operate. Operand 3 contains the result's address.

Stack Organisation:

Register Stack

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The stack in digital computers is essentially a memory unit with an address register that can count only. 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. 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.

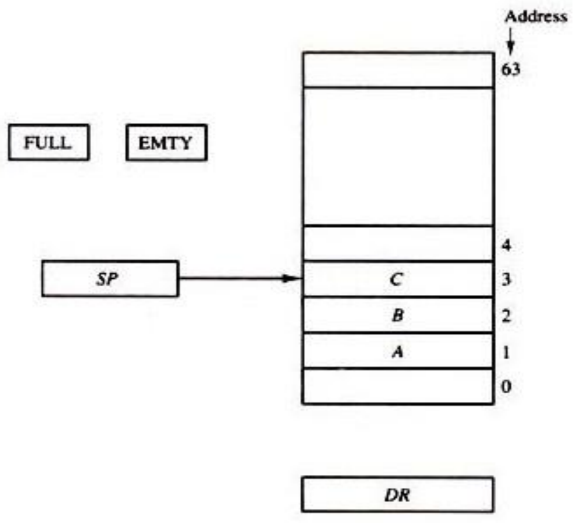


Fig 3.2 Register Stack Organisation

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 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 now3. 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 address2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the 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 are 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.

Push:

If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences of micro operations:

SP ← SP+ 1	Increment stack pointer
M [SP]←DR	WRITE ITEM ON TOP OF THESTACK
IF (SP = 0) then (FULL←1)	Check is stack is full
EMTY←0	Mark the stack not empty

The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of the stack. 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 location0. 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.

Pop:

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

DR ←M[SP]	Read item on top of the stack
SP ← SP - 1	Decrement stack pointer
IF (SP = 0) then (EMTY ← 1)	Check if stack is empty
FULL←0	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 to1.This condition is reached if the item read was in location1. Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is changes to 111111, which is equivalent to decimal63. 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.

Memory Stack:

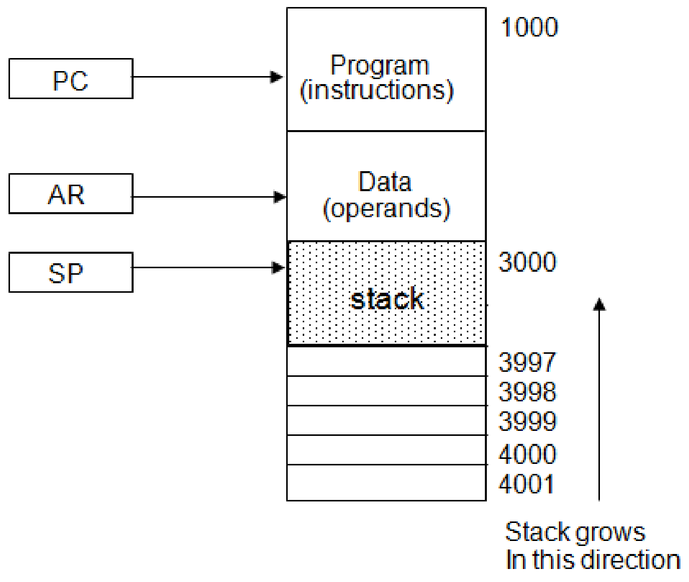


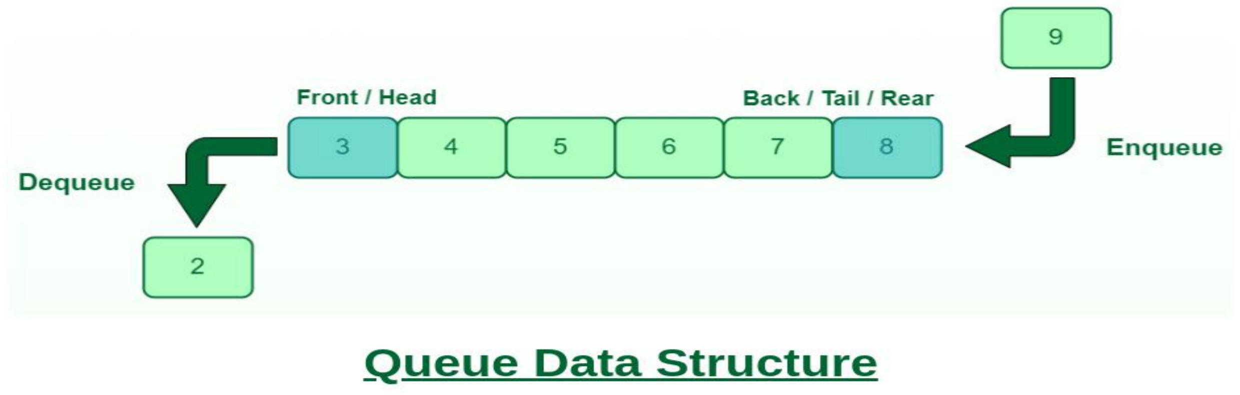
Fig 3.3 Memory Stack Organisation

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. Shows a portion of computer memory partitioned into three segments: program, data, and stack. The program counter PC points at the address of the next instruction in the program which is used during the fetch phase to read an instruction. The address registers AR points at an array of data which is used during the execute phase to read an operand. The stack pointer SP points at the top of the stack which is used to push or pop items into or from the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. 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 is3000. We assume that the items in the stack communicate with a data register DR.

Queue :-

- Another useful data structure that is similar to the stack is called a queue.
  - Data are stored in and retrieved from a queue on a first-in–first-out (FIFO) basis.
  - Thus, if we assume that the queue grows in the direction of increasing addresses in the memory, which is a common practice, new data are added at the back (high-address end) and retrieved from the front (low-address end) of the queue.
- There are two important differences between how a stack and a queue are implemented.
  - One end of the stack is fixed (the bottom), while the other end rises and falls as data are pushed and popped. A single pointer is needed to point to the top of the stack at any given time.
  - On the other hand, both ends of a queue move to higher addresses as data are added at the back and removed from the front. So two pointers are needed to keep track of the two ends of the queue.

- We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.



Fifo Property in Queue

Queue Representation:

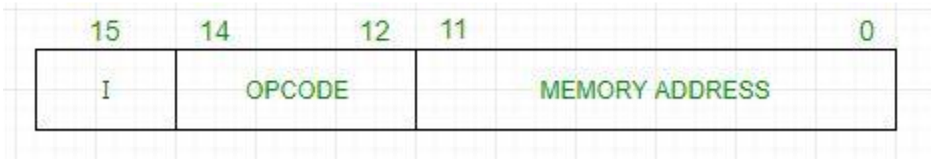
Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are

- **Queue:** the name of the array storing queue elements.
- **Front**: the index where the first element is stored in the array representing the queue.
- **Rear:** the index where the last element is stored in an array representing the queue.

Basic Instruction Types :-

The basic computer has 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

1. **Memory Reference –** These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct (0) and indirect (1) addressing.



Example – IR register contains = 0001XXXXXXXXXXXX, i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction for ADD operation.

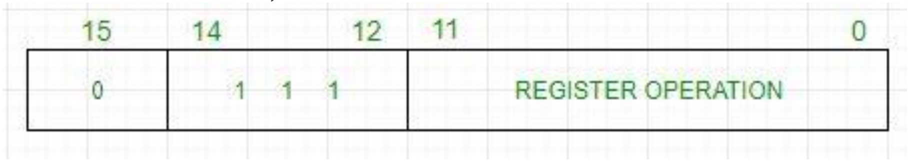
Hence, DR ← M [AR]

AC ← AC + DR, SC ← 0

Symbol	Hexadecimal Code		Description
AND	0xxx	8xxx	And memory word to AC
ADD	1xxx	9xxx	Add memory word to AC

Symbol	**Hexadecimal Code**		**Description**
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store AC content in memory
BUN	4xxx	Cxxx	Branch Unconditionally
BSA	5xxx	Dxxx	Branch and Save Return Address
ISZ	6xxx	Exxx	Increment and skip if 0

2. **Register Reference –** These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.

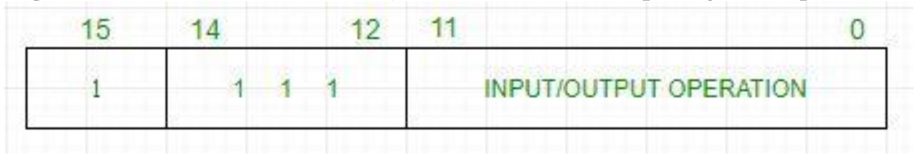


Example – IR register contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.
Hence, AC ← ~AC

Symbol	Hexadecimal Code	Description
CLA	7800	Clear AC
CLE	7400	Clear E(overflow bit)
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right AC and E
CIL	7040	Circulate left AC and E
INC	7020	Increment AC
SPA	7010	Skip next instruction if AC > 0

SNA	7008	Skip next instruction if AC < 0
SZA	7004	Skip next instruction if AC = 0
SZE	7002	Skip next instruction if E = 0
HLT	7001	Halt computer

3. **Input/ Output –** These instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.



Example – IR register contains = 1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.

Symbol	Hexadecimal Code	Description
INP	F800	Input character to AC
OUT	F400	Output character from AC
SKI	F200	Skip on input flag
SKO	F100	Skip on output flag
ION	F080	Interrupt On
IOF	F040	Interrupt Off

Arithmetic Instructions:- These are the instructions which perform basic arithmetic operations such as addition, subtraction and a few more. In 8085 microprocessor, the destination operand is generally the accumulator. Following is the table showing the list of arithmetic instructions:

Opcode	**Operand**	**Explanation**	**Example**
ADD	R	A = A + R	ADD B

ADD	M	A = A + Mc	ADD 2050
ADI	8-bit data	A = A + 8-bit data	ADI 50
ADC	R	A = A + R + prev. carry	ADC B
ADC	M	A = A + Mc + prev. carry	ADC 2050
ACI	8-bit data	A = A + 8-bit data + prev. carry	ACI 50
SUB	R	A = A – R	SUB B
SUB	M	A = A – Mc	SUB 2050
SUI	8-bit data	A = A – 8-bit data	SUI 50
SBB	R	A = A – R – prev. carry	SBB B
SBB	M	A = A – Mc -prev. carry	SBB 2050
SBI	8-bit data	A = A – 8-bit data – prev. carry	SBI 50
INR	R	R = R + 1	INR B
INR	M	M = Mc + 1	INR 2050
DCR	R	R = R – 1	DCR B
DCR	M	M = Mc – 1	DCR 2050

1 Add) – The content of operand are added to the content of the accumulator and the result is stored in accumulator .
with the addition instruction , the following 3 operations can be done.

1) any 8 bit number can be added to the contents of the accumulator and the result is stored in the accumulator.

2) The contents of a register can be added to the contents of the accumulator and result is stored in the accumulator .

3) The contents of a memory location can be added to the contents of the accumulator and result is stored in accumulator.

this 1 byte instruction.

example – add b c it adds the content of accumulator to the content of the register b

2) SUB Any 8 bit data or the contents of a register or contents of a memory location can be subtracted from the contents of the accumulator.
with the subtraction instruction the following 3 operator can be done .

1) any 8 bit number can be subtracted from the contents of the accumulator . The result is stored in the accumulator.

In the table, R stands for register M stands for memory Mc stands for memory contents r.p. stands for register pair

Logical instructions:-These are a set of instructions that perform logical operations on data in registers and memory. Logical operations are operations that manipulate the bits of data without affecting their numerical value. These operations include AND, OR, XOR, and NOT, the destination operand is always the accumulator. Here logical operation works on a bitwise level.

The logical instructions in the 8085 microprocessor include:

1. ANA – Logical AND: This instruction performs a logical AND operation between the accumulator and a specified register or memory location, and stores the result in the accumulator. For example, the instruction “ANA B” performs a logical AND operation between the contents of the accumulator and the contents of the B register.
2. ORA – Logical OR: This instruction performs a logical OR operation between the accumulator and a specified register or memory location, and stores the result in the accumulator. For example, the instruction “ORA C” performs a logical OR operation between the contents of the accumulator and the contents of the C register.
3. XRA – Logical XOR: This instruction performs a logical XOR operation between the accumulator and a specified register or memory location, and stores the result in the accumulator. For example, the instruction “XRA M” performs a logical XOR operation between the contents of the accumulator and the contents of the memory location pointed to by the HL register.
4. CPL – Logical Complement: This instruction performs a logical complement operation on the contents of the accumulator. This operation flips all the bits of the accumulator, effectively reversing its value.
5. CMA – Complement Accumulator: This instruction performs a bitwise complement operation on the contents of the accumulator. This operation flips all the bits of the accumulator, effectively reversing its value.

Following is the table showing the list of logical instructions:

OPCODE	**OPERAND**	**DESTINATION**	**EXAMPLE**
ANA	R	A = A AND R	ANA B
ANA	M	A = A AND Mc	ANA 2050
ANI	8-bit data	A = A AND 8-bit data	ANI 50
ORA	R	A = A OR R	ORA B
ORA	M	A = A OR Mc	ORA 2050
ORI	8-bit data	A = A OR 8-bit data	ORI 50

OPCODE	**OPERAND**	**DESTINATION**	**EXAMPLE**
XRA	R	A = A XOR R	XRA B
XRA	M	A = A XOR Mc	XRA 2050
XRI	8-bit data	A = A XOR 8-bit data	XRI 50
CMA	none	A = 1's complement of A	CMA
CMP	R	Compares R with A and triggers the flag register	CMP B
CMP	M	Compares Mc with A and triggers the flag register	CMP 2050
CPI	8-bit data	Compares 8-bit data with A and triggers the flag register	CPI 50
RRC	none	Rotate accumulator right without carry	RRC
RLC	none	Rotate accumulator left without carry	RLC
RAR	none	Rotate accumulator right with carry	RAR
RAL	none	Rotate accumulator left with carry	RAR
CMC	none	Compliments the carry flag	CMC
STC	none	Sets the carry flag	STC

In the table,
R stands for register
M stands for memory
Mc stands for memory contents

Shift micro-operations are those micro-operations that are used for the serial transfer of information. These are also used in conjunction with arithmetic micro-operation, logic micro-operation, and other data-processing operations. There are three types of shift micro-operations:

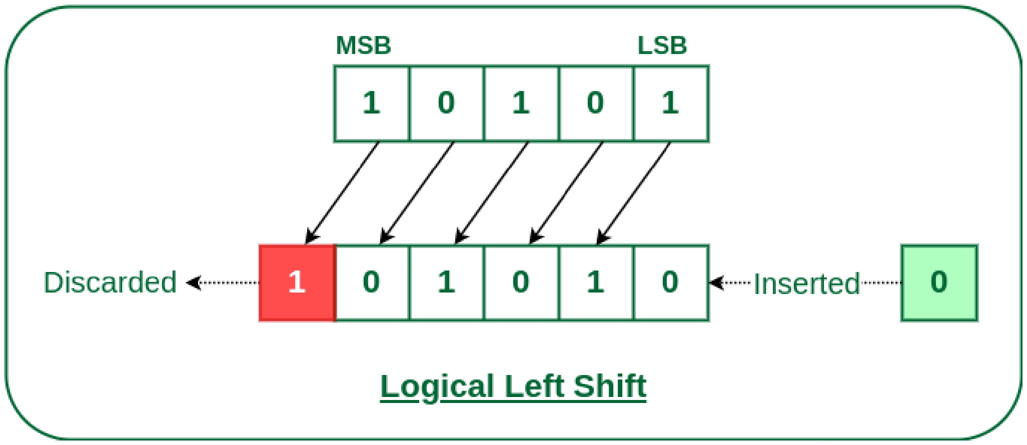
1. Logical Shift:

It transfers the 0 zero through the serial input. We use the symbols '<<' for the logical left shift and '>>' for the logical right shift.

- **Logical Left Shift:**

In this shift, one position moves each bit to the left one by one. The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected.

The left shift operator is denoted by the double left arrow key (<<). The general syntax for the left shift is shift-expression << k.



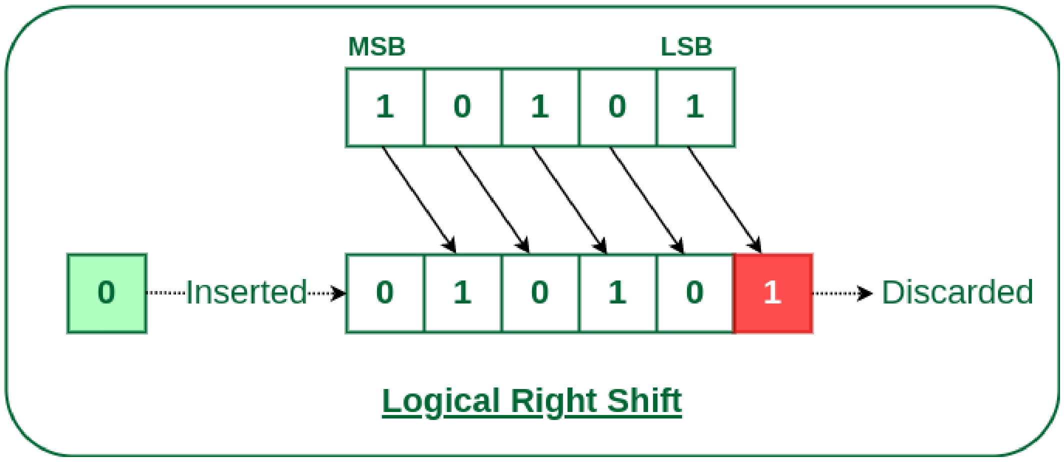
Logical Left Shift

Note: Every time we shift a number towards the left by 1 bit it multiplies that number by 2.

➢ **Logical Right Shift**

In this shift, each bit moves to the right one by one and the least significant bit(LSB) is rejected and the empty MSB is filled with zero.

The right shift operator is denoted by the double right arrow key (>>). The general syntax for the right shift is "shift-expression >> k".



Logical Right Shift

Note: Every time we shift a number towards the right by 1 bit it divides that number by 2.

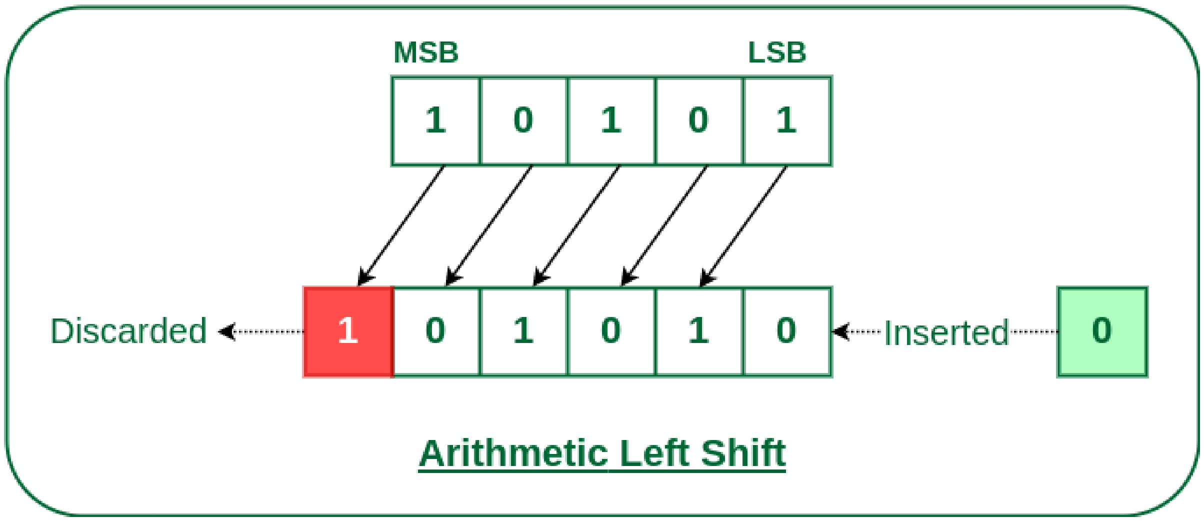
<u>2. Arithmetic Shift:</u>

The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position. Following are the two ways to perform the arithmetic shift.

1. Arithmetic Left Shift
2. Arithmetic Right Shift

➢ **Arithmetic Left Shift:**

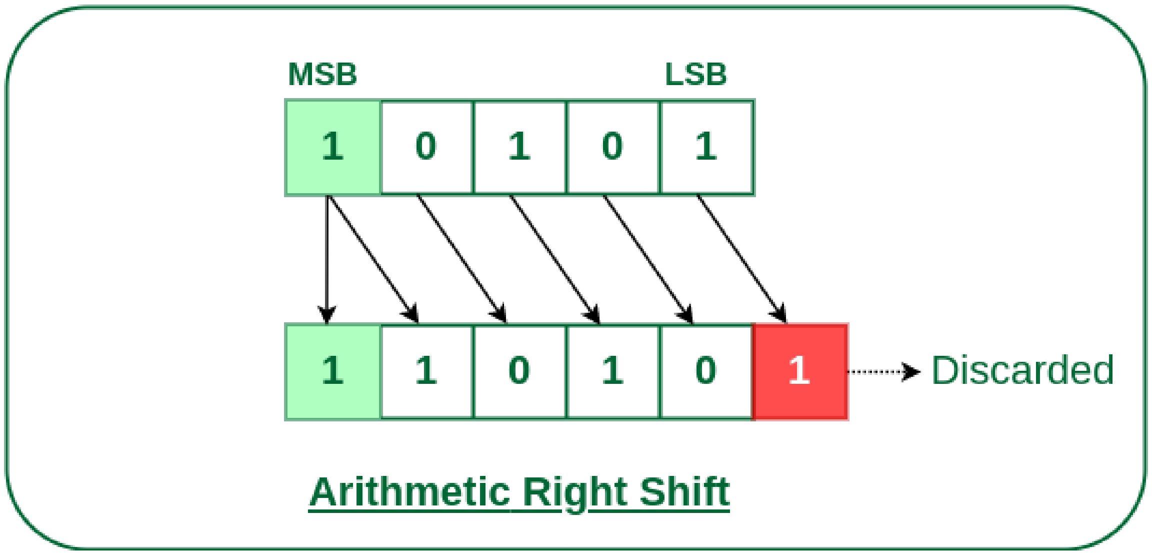
In this shift, each bit is moved to the left one by one. The empty least significant bit (LSB) is filled with zero and the most significant bit (MSB) is rejected. Same as the Left Logical Shift.



Arithmetic Left Shift

- **Arithmetic Right Shift:**

In this shift, each bit is moved to the right one by one and the least significant(LSB) bit is rejected and the empty most significant bit(MSB) is filled with the value of the previous MSB.



Arithmetic Right Shift

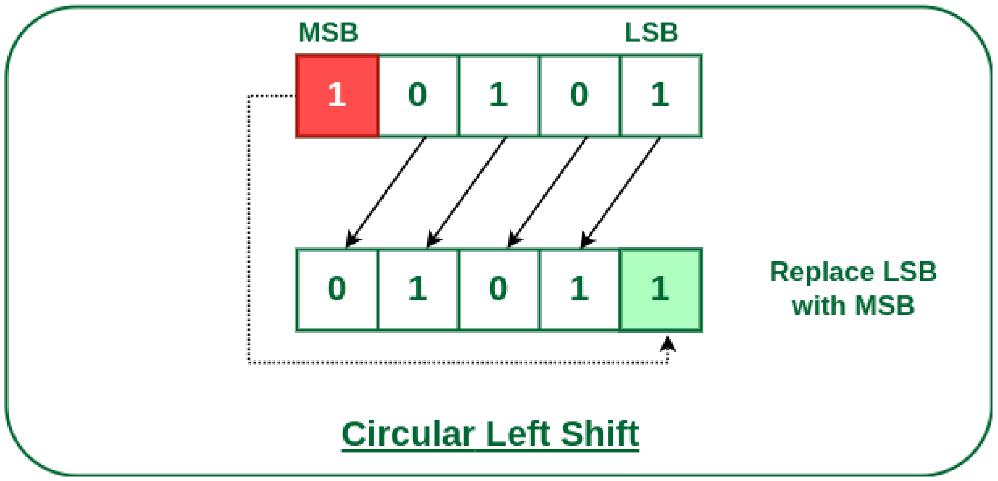
3. Circular Shift:

The circular shift circulates the bits in the sequence of the register around both ends without any loss of information.
Following are the two ways to perform the circular shift.

1. Circular Shift Left
2. Circular Shift Right

- **Circular Left Shift:**

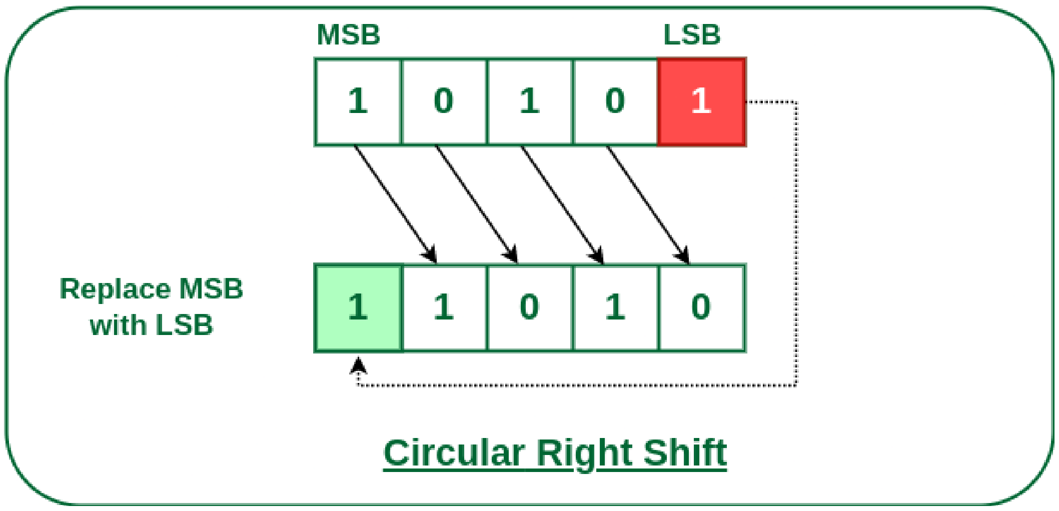
In this micro shift operation each bit in the register is shifted to the left one by one. After shifting, the LSB becomes empty, so the value of the MSB is filled in there.



Circular Left Shift

- **Circular Right Shift:**

In this micro shift operation each bit in the register is shifted to the right one by one. After shifting, the MSB becomes empty, so the value of the LSB is filled in there.



Circular Right Shift

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

The three types of branching instructions are:

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

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

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

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

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

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

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

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

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

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

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050
CZ	address	Calls if zero flag is 1	CZ 2050

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

3. Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Return instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.
(a) Unconditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program.

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

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

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

Basic Input/output operations:-

- The data on which the instructions operate are not necessarily already stored in memory. Data
- need to be transferred between processor and outside world (disk, keyboard, etc.) Input/ Output
- (I/O) operations are essential, and the way they are performed can have asignificant effect on the performance of the computer.
- Let's consider an example of read in a character input from a keyboard and produce character output on a display screen.
- A simple way of performing such I/O tasks is to use method known as program-controlled I/O
- Rate of data transfer of different units (keyboard, display, processor) participating in this may vary.
- Difference in speed between processor and I/O device creates the need for mechanisms to synchronize the transfer of data.

- A solution to this problem is as follows: On output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character, and so on.
- Input is sent from the keyboard in a similar way; the processor waits for a signal indicating that a character key has been struck and that its code is available in some buffer register associated with the keyboard. Then the processor proceeds to read that code.
- The keyboard and the display are separate devices as shown in Figure below.
- The action of striking a key on the keyboard does not automatically cause the corresponding character to be displayed on the screen.
- One block of instructions in the I/O program transfers the character into the processor, and another associated block of instructions causes the character to be displayed.

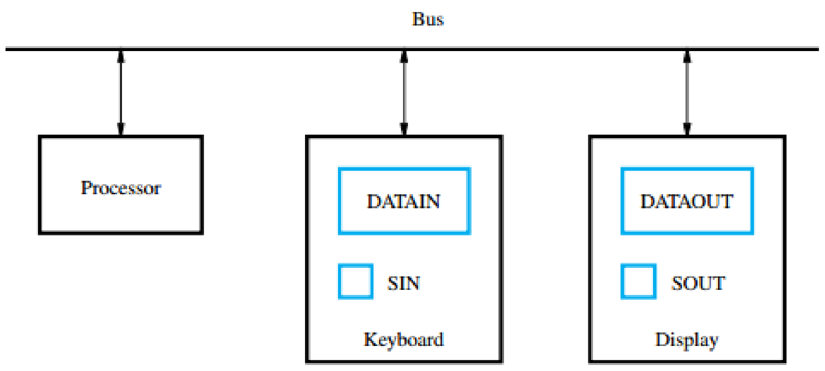


Figure Bus connection for processor, keyboard, and display.

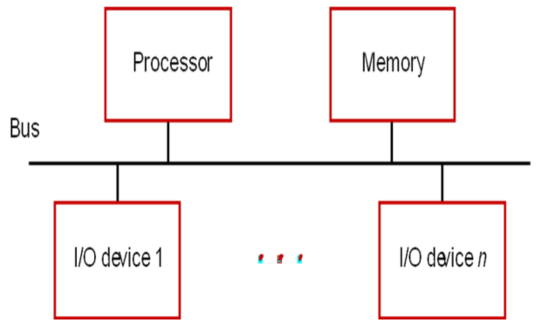
- Consider the problem of moving a character code from the keyboard to the processor. Striking a key store the corresponding character code in an 8-bit buffer register associated with the keyboard. Let us call this register DATAIN, as shown in the figure. To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1. A program monitors SIN, and when SIN is set to 1, the processor reads the contents of DATAIN. When the character is transferred to the processor, SIN is automatically cleared to 0. If a second character is entered at the keyboard, SIN is again set to 1 and the process repeats.
- An analogous process takes place when characters are transferred from the processor to the display. A buffer register, DATAOUT, and a status control flag, SOUT, are used for this transfer. When SOUT equals 1, the display is ready to receive a character. Under program control, the processor monitors SOUT, and when SOUT is set to 1, the processor transfers a character code to DATAOUT. The transfer of a character to DATAOUT clears SOUT to 0; when the display device is ready to receive a second character, SOUT is again set to 1.
- The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry commonly known as a device interface.

INPUT-OUTPUT ORGANIZATION

INPUT/OUTPUT ORGANIZATION: Accessing I/O Devices, Interrupts: Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Direct Memory Access, Buses: Synchronous Bus, Asynchronous Bus, Standard I/O Interface: Peripheral Component Interconnect (PCI) Bus, Universal Serial Bus (USB)

Accessing I/O devices

- A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement as shown in the figure below.



- The bus enables all the devices connected to it to exchange information.
- Multiple I/O devices may be connected to the processor and the memory via a bus.
- Bus consists of three sets of lines to carry address, data and control signals.
- Each I/O device is assigned a unique address.
- When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines.
- The processor requests either a read or a write operation and the requested data are transferred over the data lines.
- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O.
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- I/O devices and the memory may have different address spaces:
  - Special instructions to transfer data to and from I/O devices.
  - I/O devices may have to deal with fewer address lines.

- I/O address lines need not be physically separate from memory address lines.
- In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

- The following figure illustrates the hardware required to connect an I/O device to the bus.

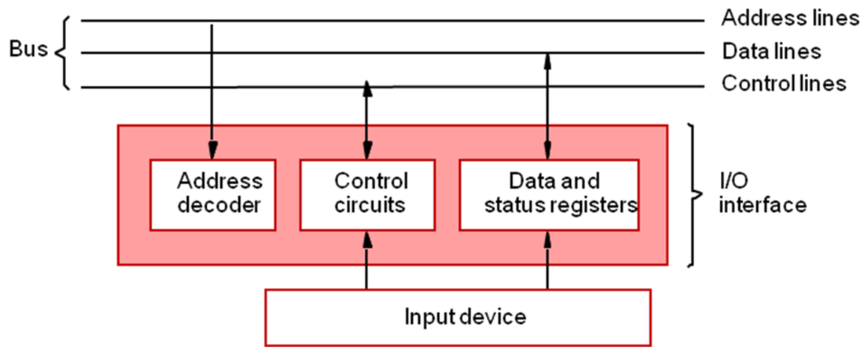
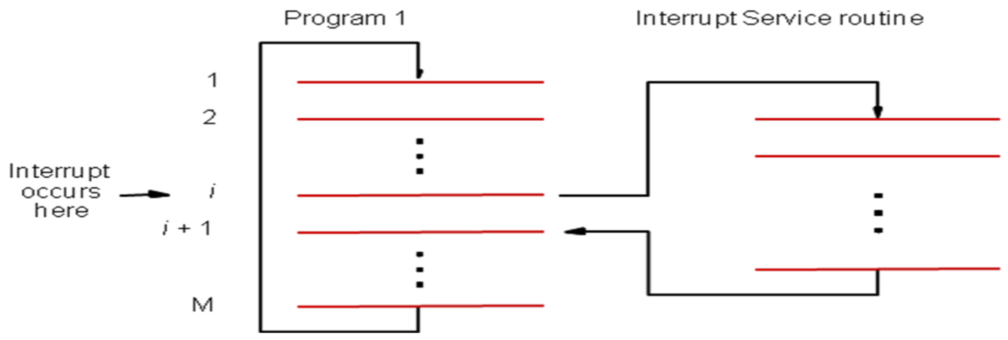


Figure: I/O interface for an input device.

- An I/O device is connected to the bus using an I/O interface circuit which has:
  - Address decoder, control circuit, data and status registers.
- The address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.
- The data register holds the data being transferred to or from the processor.
- The status register holds information relevant to the operation of the I/O device.
- Both data and status registers are connected to the data bus, and assigned unique addresses.
- I/O interface circuit coordinates I/O transfers.
- Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.
  - Program-controlled I/O:
    - In this method processor repeatedly monitors a status flag to achieve the required synchronization between the processor and an I/O device.
    - Processor polls the I/O device.
  - 2. There are two other mechanisms used for synchronizing the data transfers between the processor and memory:
    - Interrupts.
    - Direct Memory Access.

Interrupts

- In program-controlled I/O, the processor continuously monitors the status of the device; it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an ***interrupt-request*** line is dedicated for this purpose.
- Processor can perform other useful tasks in the mean time.



- Processor is executing the instruction located at address ***i*** when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine.
- When an interrupt occurs, control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC ***(i+1)***, must be saved in a known location.
- This will enable the return-from-interrupt instruction to resume execution at i+1.
- Return address, or the contents of the PC are usually stored on the processor stack.

Treatment of an interrupt-service routine is very similar to that of a subroutine. However there are significant differences:

- A subroutine performs a task that is required by the calling program.
- Interrupt-service routine may not have anything in common with the program it interrupts.
- Interrupt-service routine and the program that it interrupts may belong to different users.

- As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.

- ❑ This will enable the interrupted program to resume execution upon return from interrupt service routine.

- ❑ Saving and restoring information can be done automatically by the processor or explicitly by program instructions.

- ❑ Saving and restoring registers involves memory transfers:

  - ◆ Increases the total execution time.

  - ◆ Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called ***interrupt latency***.

- ❑ In order to reduce the interrupt latency, most processors save only the minimal amount of information:

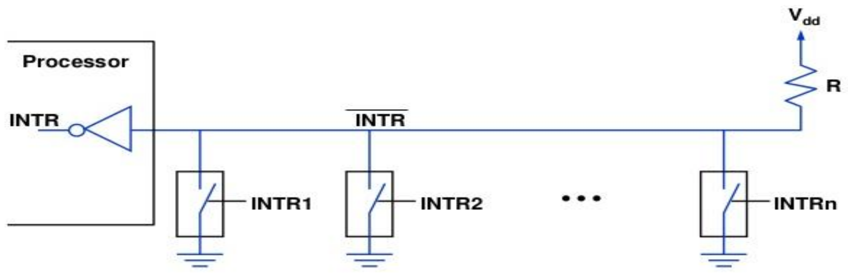
  - ◆ This minimal amount of information includes Program Counter and processor status registers.

- ❑ Any additional information that must be saved must be saved explicitly by the program instructions at the beginning of the interrupt service routine and restored at the end of the routine.

Interrupt Hardware

❑ An I/O device requests an interrupt by activating a bus line called ***interrupt-request***.

❑ Most computers are likely to have several I/O devices that can request an interrupt. A single interrupt –request line may be used to serve ***n*** devices as depicted in the following figure.



❑ All devices are connected to the line via switches to ground.

❑ To request an interrupt, a device closes its associated switch. Thus if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, that is if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}. This is the inactive state of the line.

❑ When a device requests an interrupt by closing its switch the voltage on the line drops to 0, causing the interrupt-request signal, INTR received by the processor to go to 1.

❑ Since the closing of one or more switches will cause the line voltage to drop to 0,the value of INTR is the logical OR of the requests from individual devices, that is $INTR=INTR_1+……+INTR_n$

- It is customary to use the complemented form, $\overline{\text{INTR}}$ because this signal is active when in the low-voltage state.
- In the electronic implementation of the above circuit special gates known as open-collector or open-drain are used.
- The output of an open-collector or an open-drain gate is equivalent to a switch to ground that is open when the gate's input is in the 0 state and closed when it is in the 1 state.
- Resistor R is called a pull-up resistor because it pulls the line voltage up to the high-voltage state when the switches are open.

Enabling and disabling interrupts

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as ***Interrupt-enable*** and ***Interrupt-disable*** for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be ***Interrupt-disable***.
  - Last instruction of an interrupt service routine can be ***Interrupt-enable***.

Handling Multiple Devices

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
  - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?

- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

- ❑ Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.

- ❑ When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?

1. This information is available in the status register of the device requesting an interrupt:

    - ◆ The status register of each device has an ***IRQ*** bit which it set to 1 when it requests an interrupt.

    - ◆ Interrupt service routine can poll the I/O devices connected to the bus. The first device encountered with its ***IRQ*** *bit set (*equal to 1) is the one that should be serviced.

    - ◆ Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

2. An alternative approach is to use ***vectored interrupts***. The device requesting an interrupt may identify itself directly to the processor.

    - ◆ The device can do so by sending a special code (4 to 8 bits) to the processor over the bus.

    - ◆ Code supplied by the device may represent a part of the starting address of the interrupt-service routine.

    - ◆ The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.

- ❑ Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

- ❑ I/O devices are organized in a priority structure:

    - ◆ An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.

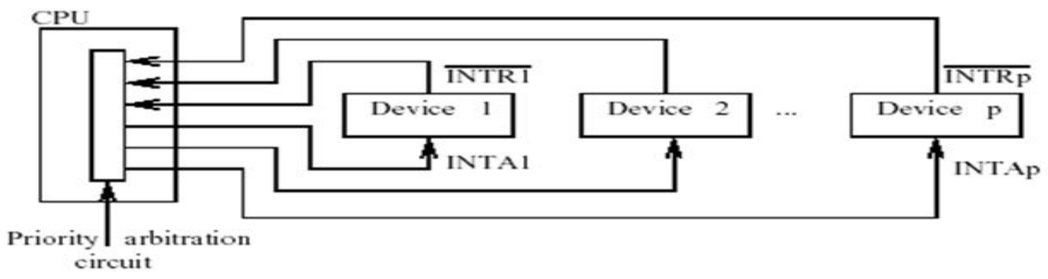
- ❑ A priority level is assigned to a processor that can be changed under program control.

    - ◆ Priority level of a processor is the priority of the program that is currently being executed.

    - ◆ When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.

    - ◆ If the device sending an interrupt request has a higher priority than that of the processor, the processor accepts the interrupt request.

- ❑ A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-acknowledge lines for each device, as in the figure given below.



- Each device has a separate ***interrupt-request*** and ***interrupt-acknowledge line***.
- Each interrupt-request line is assigned a different priority level.
- Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
- If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.

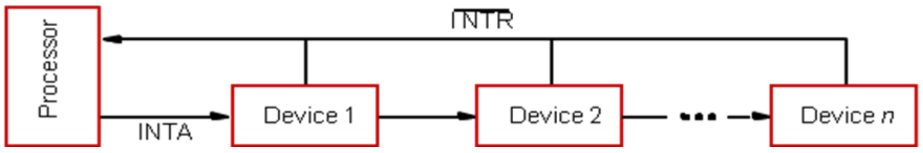
Simultaneous Requests

- ❑ Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
- ❑ However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

1) Polling scheme:

- Polling the status registers of the I/O devices is the simplest mechanism.
- In this case the priority is determined by the order in which the devices are polled.
- The first device with status bit set to 1 is the device whose interrupt request is accepted.

2) Daisy chaining scheme



- Devices are connected in a serial fashion.
- The interrupt-request line $\overline{\text{INTR}}$ is common to all devices.
- The interrupt-acknowledge line INTA is connected in a daisy-chain fashion such that the INTA signal propagates serially through the devices.
- When several devices raise an interrupt request and the $\overline{\text{INTR}}$ line is activated, the processor responds by setting the INTA line to 1.
- This signal is received by device 1.If device 1 does not need any service; it passes the signal to device 2.
- If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines.

- In daisy chaining arrangement, the device that is electrically closest to the processor has the highest priority.

Direct Memory Access

❑ To transfer large blocks of data at high speed, a special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called ***direct memory access*** **(DMA)**

❑ DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a ***DMA controller***.

❑ The DMA controller performs functions that would normally be carried out by the processor:

- For each word transferred, it provides the memory address and all the control signals.
- To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

DMA Controller

❑ DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.

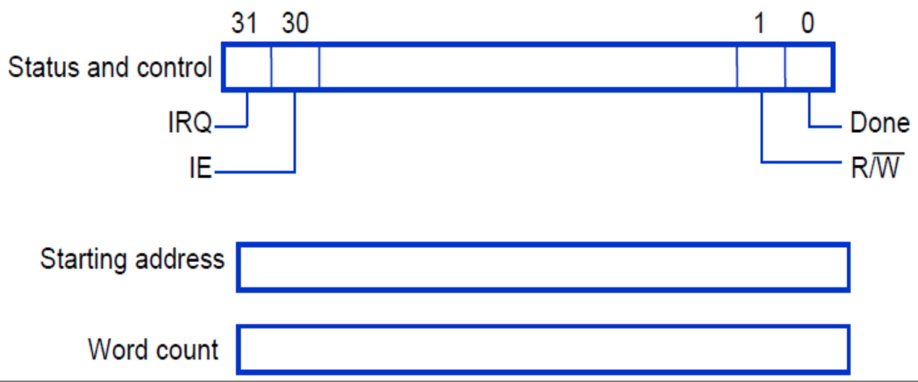
❑ However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.

To initiate the DMA transfer, the processor informs the DMA controller of:

- Starting address,
- Number of words in the block.
- Direction of transfer (I/O device to the memory, or memory to the I/O device).

❑ Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

DMA Controller registers

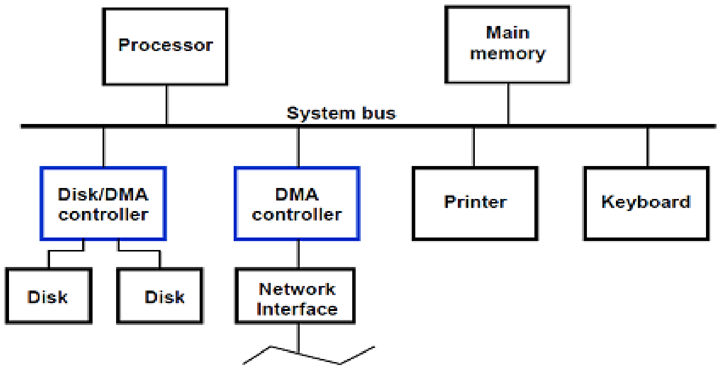


The above figure shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations.

- Two registers are used for storing the starting address and the word count.
- The third register contains status and control flags.
  - The R/W bit determines the direction of the data transfer.
  - When the bit is set to ***1*** by a program instruction, the controller performs a ***read*** operation, that is, it transfers data from the memory to the I/O device. Otherwise it performs a ***write*** operation.
  - When the controller has completed transferring a block of data and is ready to receive another command, it sets the ***Done*** flag to ***1***.
  - ***Bit 30*** is the ***Interrupt-enable flag, IE***. When this flag is set to ***1***, it causes the controller to raise an interrupt after it has completed transferring a block of data.
  - Finally the controller sets the ***IRQ*** bit to ***1*** when it has requested an interrupt.

DMA controller in a computer system

- An example of a computer system is given in the below figure showing how DMA controllers may be used.
  - A DMA controller connects a high speed network to the computer bus.
  - The disk controller, which controls two disks also has DMA capability and provides two DMA channels.
  - It can perform two independent DMA operations as if each disk had its own DMA controller.
  - The registers needed to store the memory address, the word count and so on are duplicated so that one set can be used with each device.



- Memory accesses by the processor and the DMA controllers are interwoven. Requests by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high speed network interface, etc.
- Since the processor originates most memory access cycles, the DMA controller can be said to "steal"

memory cycles from the processor. Hence, this interweaving technique is usually called ***cycle stealing.***

- ❑ The DMA controller may transfer a block of data to the main memory without interruption. This is called ***block/burst* mode**

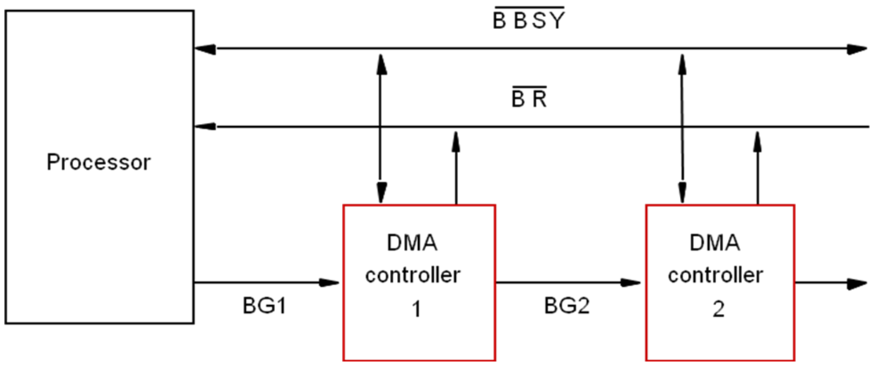
Bus Arbitration

- ❑ A conflict may arise if two DMA controllers try to use the bus at the same time to access the main memory.
- ❑ To resolve this problem, an arbitration procedure is implemented on the bus.
- ❑ The device that is allowed to initiate data transfer on the bus at any given time is called the ***bus master***.
- ❑ When the current master relinquishes control of the bus, another device can acquire this status.
  - ◆ The process by which the next device to become the bus master is selected and bus mastership is transferred to it is called ***bus arbitration***.

There are two approaches to bus arbitration:

1. Centralized and
2. distributed
   - ◆ In centralized arbitration, a single bus arbiter performs the required arbitration.
   - ◆ In distributed arbitration, all devices participate in the selection of the next bus master.

Centralized Arbitration



- ❑ Bus arbiter may be the processor or a separate unit connected to the bus.
- ❑ Normally, the processor is the bus master, unless it grants bus mastership to one of the DMA controllers.
- ❑ A DMA controller requests the control of the bus by asserting the Bus Request $\overline{BR}$ line.
- ❑ In response, the processor activates the Bus-Grant1 (BG1) line, indicating that the DMA controllers that they may use the bus when it becomes free.
- ❑ BG1 signal is connected to all DMA controllers in a daisy chain fashion.
  - ◆ If DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices.

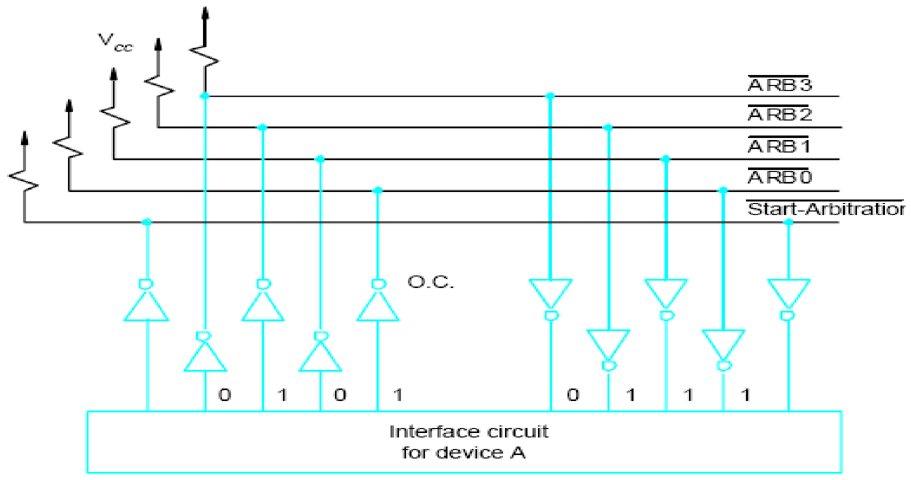
- Otherwise, it passes the grant downstream by asserting BG2.

The current bus master indicates to all the devices that it is using the bus by activating

$\overline{BBBSY}$ signal. If BBSY is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted $\overline{BR}$ can acquire control of the bus.

Distributed Arbitration

- In this method all devices waiting to use the bus share the responsibility in carrying out the arbitration process.
  - Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability.
- Each device on the bus is assigned a 4-bit unique ID (identification) number.
- All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal.
- To request the bus a device:
  - Asserts the $\overline{\text{Start} - \text{Arbitration}}$ signal.
  - Places its 4-bit ID number on the arbitration lines $\overline{ARB0}$ to $\overline{ARB3}$.



Example:

- Device A has the ID **'5'** and wants to request the bus:
  - It transmits the pattern 0101 on the arbitration lines.
- Device B has the ID **'6'** and wants to request the bus:

- It transmits the pattern 0110 on the arbitration lines.

- Pattern that appears on the arbitration lines is the logical OR of the patterns of devices A and B:
  - Pattern 0111 appears on the arbitration lines.

Arbitration process:

- Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.
- If it detects a difference at any bit position, it disables its drivers lines at that bit position and for all lower-order bits.
- It does so by placing a 0 at the input of these drivers.
- Device A compares its ID 5 (0101) with the pattern 0111.
- It detects a difference on the line ARB1.Hence it disables its drivers on the lines ARB1 AND ARB0.This causes the pattern on the arbitration lines to change to0100.
- The current pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110.
- This pattern is same as the ID of device B, and hence B has won the arbitration.

Buses

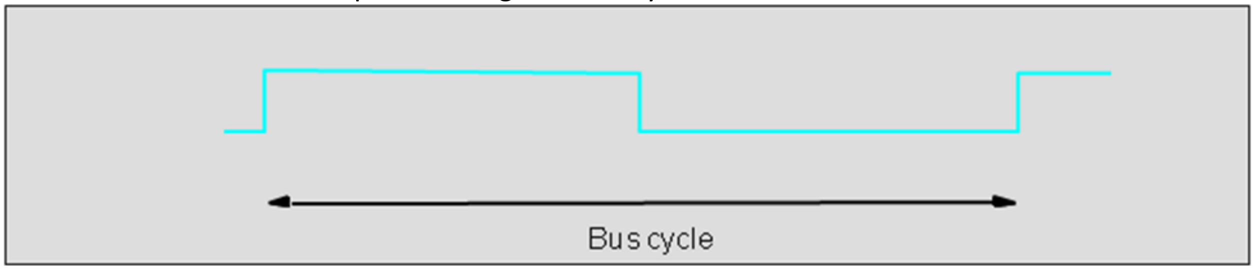
- Processor, main memory, and I/O devices are interconnected by means of a bus.
- Bus provides a communication path for the transfer of data.
  - Bus also includes lines to support interrupts and arbitration.
- A ***bus protocol*** is the set of rules that govern the behavior of various devices connected to the bus, as to when to place information on the bus, when to assert control signals, etc.
- Bus lines may be grouped into three types:
  - Data
  - Address
  - Control
- Control signals specify:
  - Whether it is a read or a write operation.
  - Required size of the data, when several operand sizes (byte, word, long word) are possible.

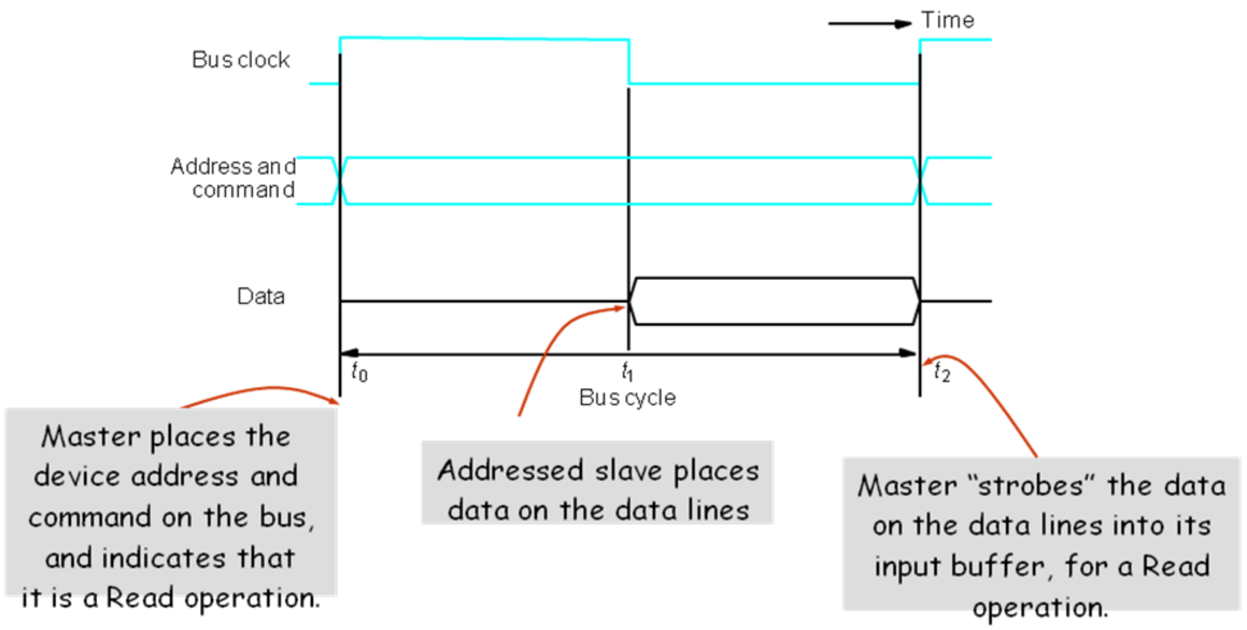
- Timing information to indicate when the processor and I/O devices may place data or receive data from the bus.

- Schemes for timing of data transfers over a bus can be classified into:
  - Synchronous,
  - Asynchronous.

Synchronous Bus

- All devices derive timing information from a common clock line.
- The clock line has equally spaced pulses which define equal time intervals.
  - In a simple synchronous bus, each of these pulses constitutes a bus cycle.
- One data transfer can take place during one bus cycle.

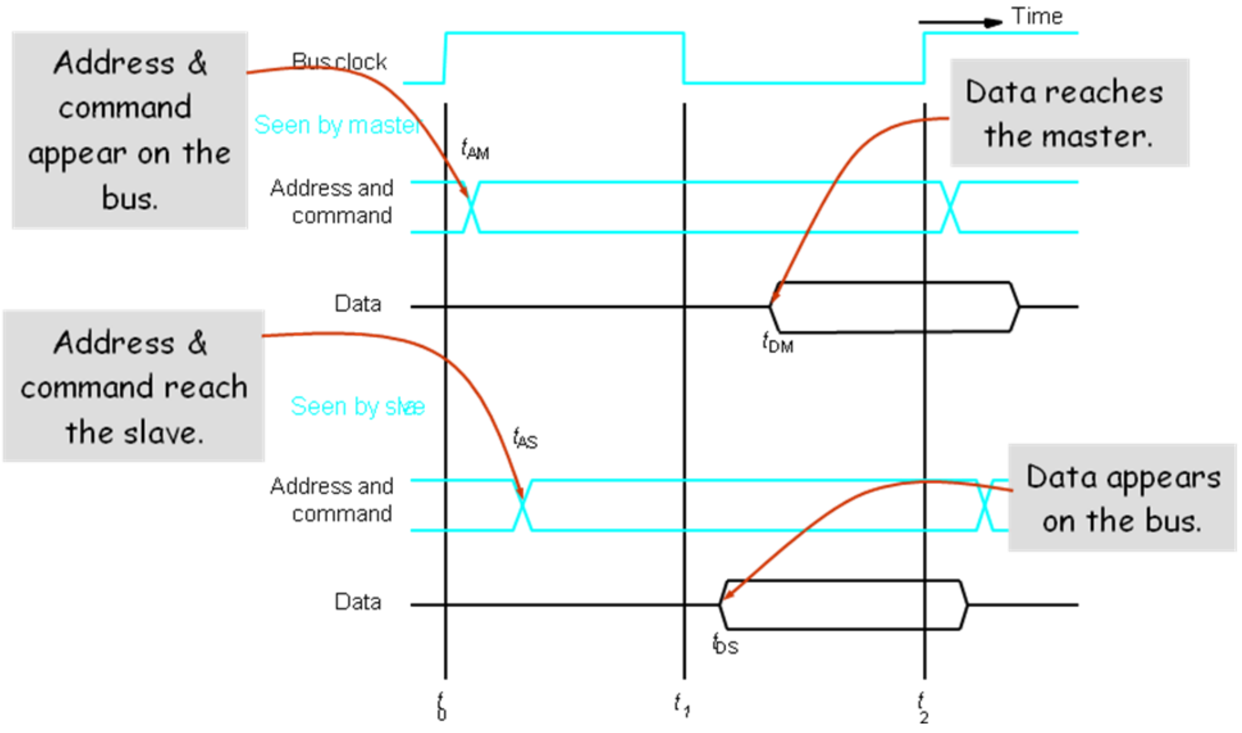



Time
Bus clock
Address and command
Data
t_0
t_1
t_2
Bus cycle
Master places the device address and command on the bus, and indicates that it is a Read operation.
Addressed slave places data on the data lines
Master "strobes" the data on the data lines into its input buffer, for a Read operation.

- In case of a Write operation, the master places the data on the bus along with the address and commands.
- The slave strobes the data into its input buffer at time t_2.
  - Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices. This time depends on the physical and electrical

characteristics of the bus. Also, all the devices have to be given enough time to decode the address and control signals, so that the addressed slave can place data on the bus.

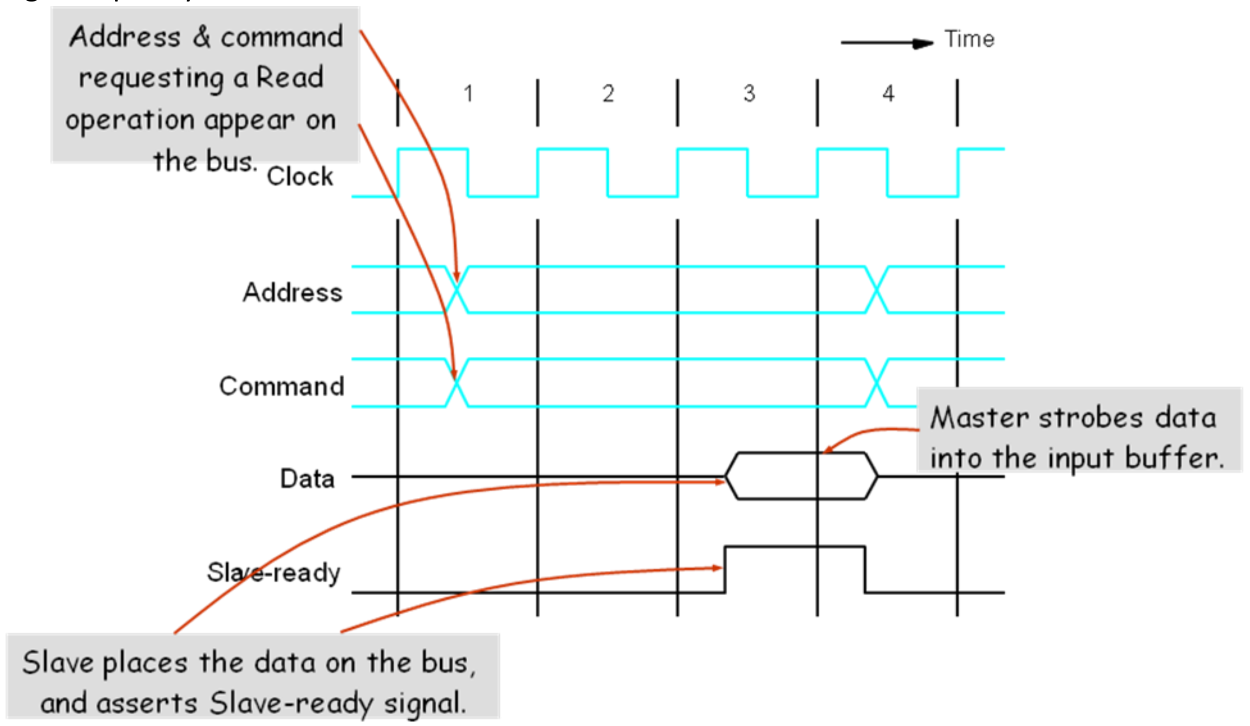
- ◆ Width of the pulse $t_1 - t_0$ depends on:
  - ➢ Maximum propagation delay between two devices connected to the bus.
  - ➢ Time taken by all the devices to decode the address and control signals, so that the addressed slave can respond at time t_1.
- ❑ At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer if it's a Read operation.
  - ◆ "Strobe" means to capture the values of the data and store them into a buffer.
- ❑ When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.
- ❑ Width of the pulse $t_2 - t_1$ should be longer than:
  - ◆ Maximum propagation time of the bus plus
  - ◆ Set up time of the input buffer register of the master.



- ❑ Data transfer has to be completed within one clock cycle.
  - ◆ Clock period ***t2 - t0*** must be such that the longest propagation delay on the bus and the slowest device interface must be accommodated.
  - ◆ Forces all the devices to operate at the speed of the slowest device.
- ❑ Processor just assumes that the data are available at t2 in case of a Read operation, or are read by the device in case of a Write operation.
  - ◆ What if the device is actually failed, and never really responded?
- ❑ Most buses have control signals to represent a response from the slave.
- ❑ Control signals serve two purposes:

- Inform the master that the slave has recognized the address, and is ready to participate in a data transfer operation.
- Enable to adjust the duration of the data transfer operation based on the speed of the participating slaves.

- High-frequency bus clock is used.

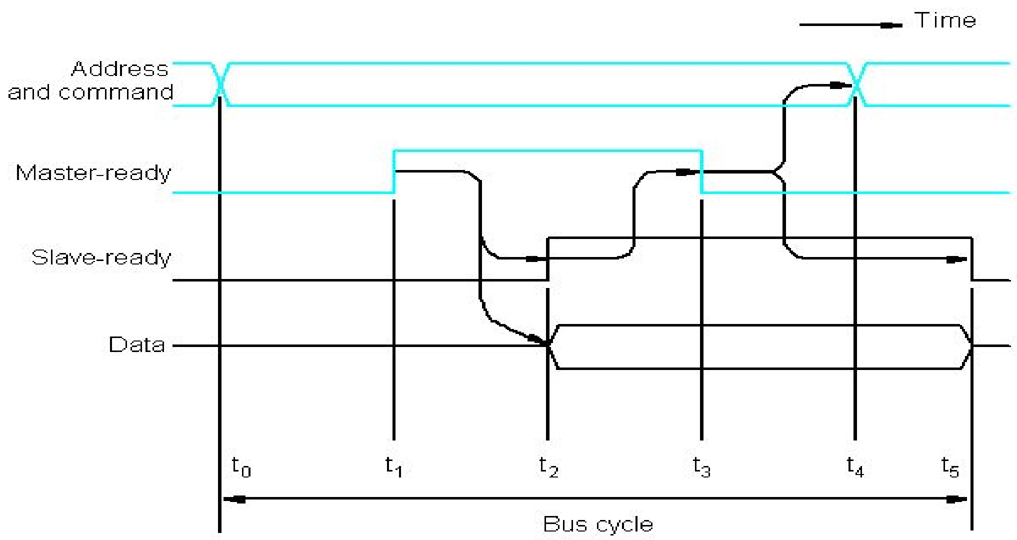

Address & command requesting a Read operation appear on the bus.
Time
1
2
3
4
Clock
Address
Command
Data
Slave-ready
Master strobes data into the input buffer.
Slave places the data on the bus, and asserts Slave-ready signal.

- Clock signal used on the bus is not necessarily the same as the processor clock.
  - Processor clock is much faster than the bus clock.
- Modern processor clocks are typically above 500 MHz.
- Clock frequencies used on the memory and I/O buses may be in the range of 50 to 150 MHz.

Asynchronous Bus

- A data transfer on the bus is controlled by a handshake between the master and the slave.
- Common clock in the synchronous bus case is replaced by two timing control lines:
  - Master-ready,
  - Slave-ready.
- Master-ready signal is asserted by the master to indicate that it is ready to participate in a data transfer.
- Slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to participate in the data transfer.
- Data transfer using the handshake protocol:
  - Master places the address and command information on the bus.
  - Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
  - All devices on the bus decode the address.

- Addressed slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
- Master removes all the signals from the bus, once Slave-ready is asserted.
- If the operation is a Read operation, Master also strobes the data into its input buffer.



- **t_0 -** Master places the address and command information on the bus. Command indicates that it is a read operation that is the data are transferred from the device to the memory.
- **t_1 -** Master asserts the Master-ready signal. Master-ready signal is asserted at t_1 instead of t_0 to allow for bus skew. Bus skew occurs when two signals transmitted simultaneously reach the destination at different times. This may occur because different bus lines may have different speeds. t_1-t_0 should be greater than maximum skew. t_1-t_0 should also include the time taken to decode the address information.
- **t_2 -** Addressed slave places the data on the bus and asserts the Slave-ready signal. The period t_2-t_1, depends on the propagation delay between the master and the slave, and the delay in the slave's interface circuit.
- **t_3 -** Slave-ready signal arrives at the master. Slave-ready signal was placed on the bus at the same time that data were placed on the bus. The master now strobes the data. It also deactivates the Master-ready signal to indicate that it has received the data.
- **t_4 -** Master removes the address and command information. Once Master-ready signal is set to 0, it should reach all the devices before the address and command information is removed from the bus.
- **t_5 -** Slave receives the transition of the Master-ready signal from 1 to 0. It removes the data and the Slave-ready signal from the bus.

Standard I/O Interface

- I/O device is connected to a computer using an interface circuit.
- Do we have to design a different interface for every combination of an I/O device and a computer?
- A practical approach is to develop standard interfaces and protocols.

- A personal computer has:

- ◆ A motherboard which houses the processor chip, main memory and some I/O interfaces.
- ◆ A few connectors into which additional interfaces can be plugged.

❑ Processor bus is defined by the signals on the processor chip.
- ◆ Devices which require high-speed connection to the processor are connected directly to this bus.

❑ Because of electrical reasons only a few devices can be connected directly to the processor bus.

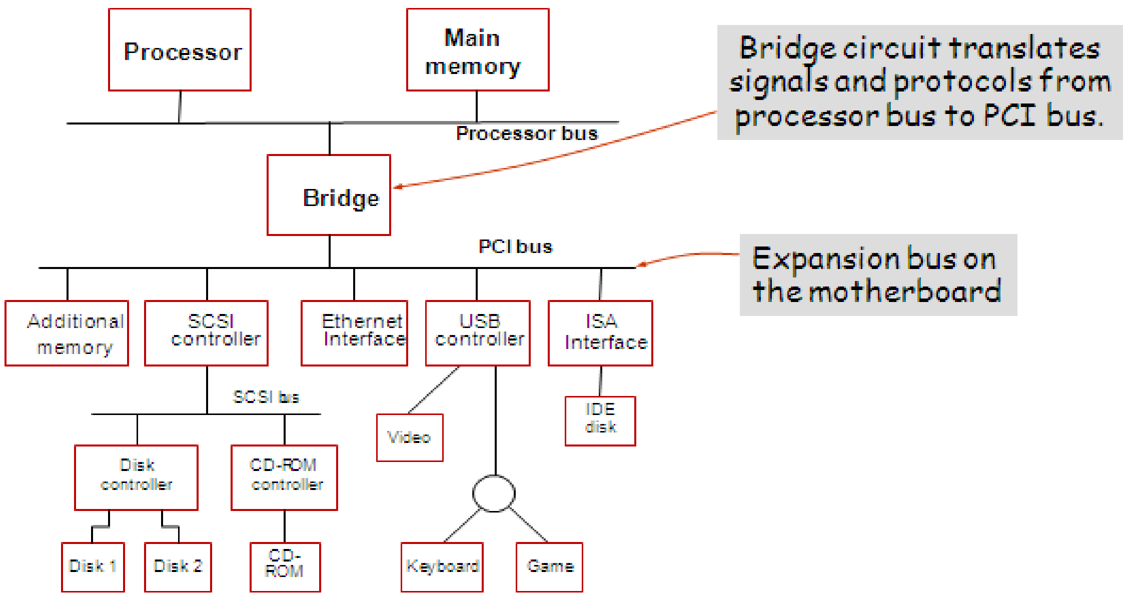
❑ Motherboard usually provides another bus that can support more devices.
- ◆ Processor bus and the other bus (called as expansion bus) are interconnected by a circuit called "bridge".
- ◆ Devices connected to the expansion bus experience a small delay in data transfers.

❑ Design of a processor bus is closely tied to the architecture of the processor.
- ◆ No uniform standard can be defined.

❑ Expansion bus however can have uniform standard defined.

❑ There are three widely used bus standards:
- ◆ PCI (Peripheral Component Interconnect)
- ◆ SCSI (Small Computer System Interface)
- ◆ USB (Universal Serial Bus)

The way these standards are used in a typical computer system is illustrated in the following figure.



- ➢ The PCI standard defines an expansion bus on the mother board.
- ➢ SCSI and USB are used for connecting additional devices, both inside and outside the computer box.
- ➢ A given computer may use more than one bus standard.

Peripheral Component Interconnect (PCI) Bus

- Introduced in 1992
- Low-cost bus
- Processor independent
- Plug-and-play capability
- In today's computers, most memory transfers involve a ***burst of data*** rather than just one word. The PCI is designed primarily to support this mode of operation.
- The bus supports three independent address spaces: memory, I/O, and configuration.

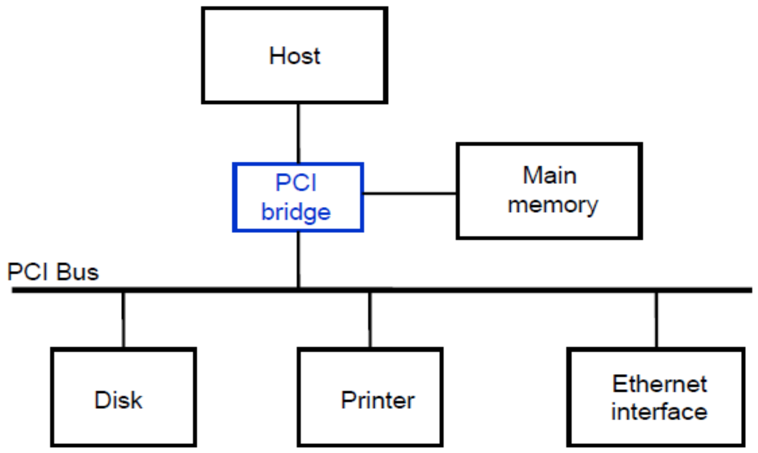


Fig- Use of a PCI bus in a computer system

Data transfer signals on the PCI bus.

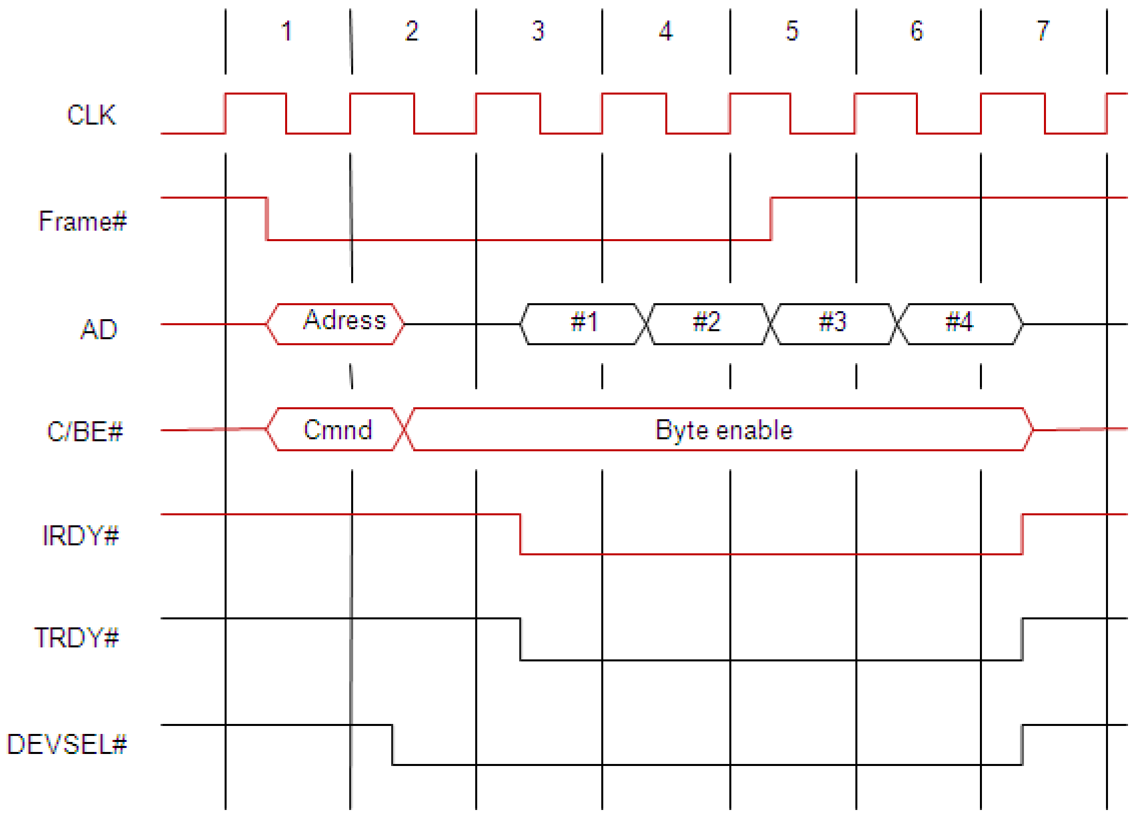
Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus).
IRDY#, TRDY#	Initiator-ready and Target-ready signals.
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction.
IDSEL#	Initialization Device Select.

Note: Signals whose name ends with the symbol # are asserted when in the low-voltage state.

- We assumed that the master maintains the address information on the bus until data transfer is completed. But, the address is needed only long enough for the slave to be selected. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles. The result is a significant cost reduction.

- A master is called an ***initiator*** in PCI terminology. The addressed device that responds to read and write commands is called a ***target***.

Read operation on the PCI bus



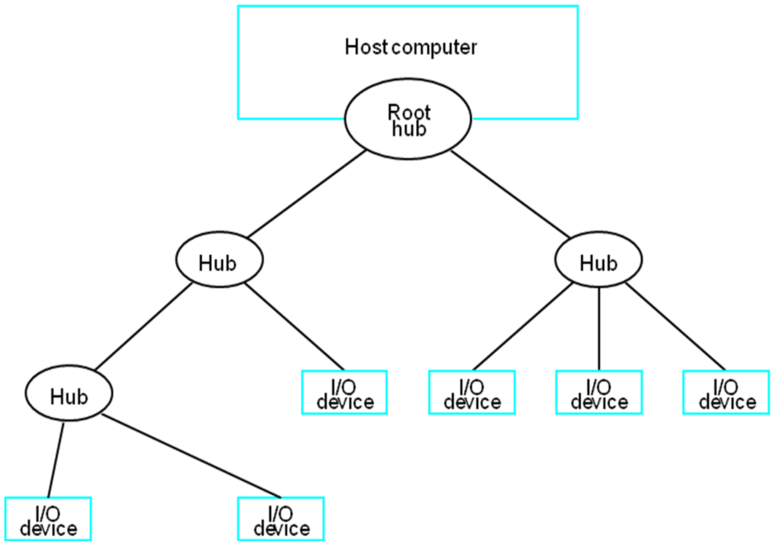
A read operation on the PCI bus

- ❑ In clock cycle 1, the processor asserts FRAME# to indicate the beginning of a transaction. At the same time it sends the address on the AD lines and a command on the C/BE# lines. In this case, the command will indicate that a read operation is requested and that the memory address space is being used.
- ❑ Clock cycle 2 is used to turn the AD bus lines around. The processor removes the address and disconnects from the AD lines. The selected target enables its drivers on the AD lines and fetches the requested data to be placed on the bus during clock cycle3.
- ❑ During clock cycle 3, the initiator asserts the initiator ready signal, IRDY# to indicate that it is ready to receive data. If the target has data ready to send at this time, it asserts target ready, TRDY# and sends a word of data. The initiator loads the data into its input buffer at the end of the clock cycle.
- ❑ The initiator uses the FRAME# signal to indicate the duration of the burst. It negates this signal during the second last word of the transfer. Since it wishes to read four words, the initiator negates FRAME# during clock cycle 5, the cycle in which it receives the third word. After sending the fourth word in clock cycle 6, the target disconnects its drivers and negates DEVSEL# at the beginning of clock cycle7.

Universal Serial Bus (USB)

- Universal Serial Bus (USB) is an industry standard developed through a collaborative effort of several computer and communication companies, including Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nortel Networks, and Philips.
- Speed
  - Low-speed(1.5 Mb/s)
  - Full-speed(12 Mb/s)
  - High-speed(480 Mb/s)
- Plug-and-play
- Simple, low cost bus

USB Tree structure



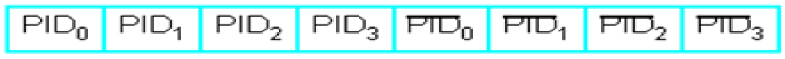
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.
- Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root hub connects the entire tree to the host computer. The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker, or digital TV)
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports.
  - As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message.
- However, a message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices.
  - Hence, the USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

USB Protocols

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information.

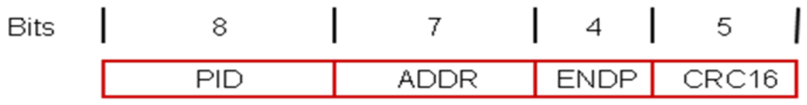
- ❑ There are many types of packets that perform a variety of control functions.
- ❑ The information transferred on the USB can be divided into two broad categories: *control and data.*
    - ◆ Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error.
    - ◆ Data packets carry information that is delivered to a device.

❑ A packet consists of one or more fields containing different kinds of information. The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.



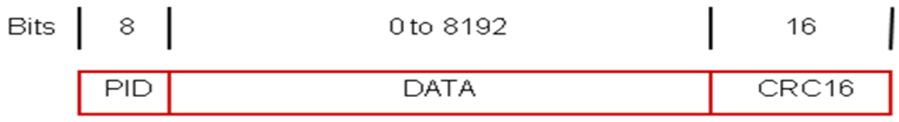
(a) Packet identifier field

❑ There are four bits of information in this field, but they are transmitted twice.
❑ The first time they are sent with their true values, and the second time with each bit complemented. This enables the receiving device to verify that the PID byte has been received correctly.
❑ The four PID bits identify one of 16 different packet types.



(b) Token packet, IN or OUT

- ❑ Control packets used for controlling data transfer operations are called ***token packets***.
- ❑ A token packet starts with a PID field, using one of two PID values to distinguish between an IN and OUT packet, which control input and output transfers respectively.
- ❑ The PID field is followed by the 7-bit address of a device and the 4-bit endpoint number within that device.
- ❑ The packet ends with 5 bits for error checking using a method called Cyclic Redundancy check(CRC).
- ❑ The CRC bits are computed based on the contents of the address and endpoints fields.

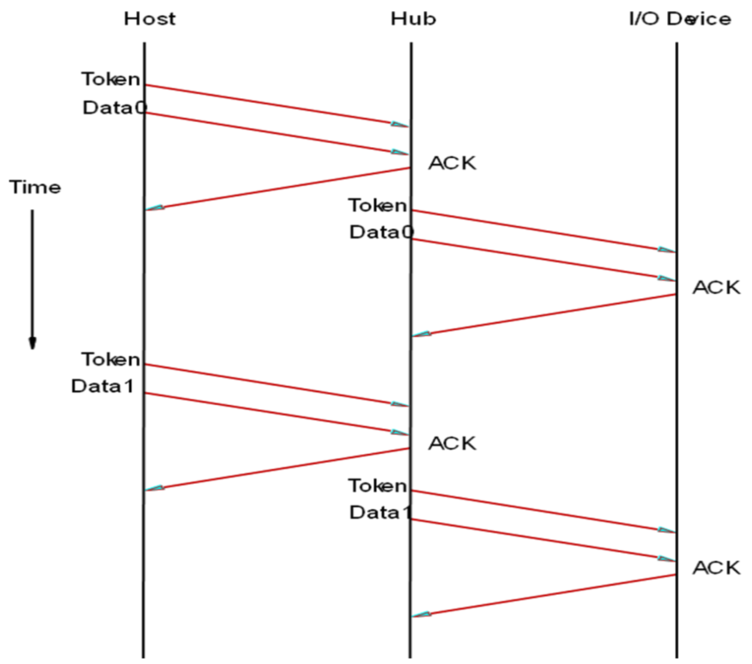


(c) Data packet

- ❑ Data packets carry input and output data.
- ❑ The PID field is followed by 8192 bits of data, then 16 error-checking bits.
- ❑ Three different PID patterns are used to identify data packets.

An Output Data Transfer

Consider an output device connected to a USB hub, which in turn is connected to a host computer. An example of an output operation is shown in the figure given below.



- ❑ The host computer sends a token packet of type OUT to the hub, followed by a data packet containing the output data.
- ❑ The PID field of the data packet identifies it as data packet number 0.
- ❑ The hub verifies that the transmission has been error free by checking the error control bits, and then sends an acknowledgement packet (ACK) back to the host.
- ❑ The hub forwards the token and data packets downstream.
- ❑ All the I/O devices receive this sequence of packets, but only the device that recognizes its address in the token packet accepts the data in the packet that follows.
- ❑ After verifying that transmission has been error free, it sends an ACK packet to the hub.
- ❑ If a token, data, or acknowledgement packet is lost as a result of a transmission error, the sender resends the entire sequence. By checking the data packet number in the PID field, the receiver can detect and discard duplicate packets.

Electrical Characteristics

- ❑ The cables used for USB connections consist of four wires.
- ❑ Two are used to carry power, +5V and Ground.
- ❑ The other two wires are used to carry data.
- ❑ Different signaling schemes are used for different speeds of transmission.
  - ◆ At low speed, 1s and 0s are transmitted by sending a high voltage state (5V) on one or the other of the two signal wires. For high-speed links, differential transmission is used.

UNIT-6: BASIC PROCESSING UNIT

Processing Unit: Some fundamental Concepts: Register transfers, Performing an arithmetic or logic Operation, Fetching a word from memory, Storing a word in memory, Execution of a complete instruction. Hardwired Control, Microprogrammed Control.

Introduction:

The processing unit executes machine instructions and coordinates the activities of other units. This unit is often called the ***Instruction Set Processor*** (ISP) or simply the ***processor.*** It is also called as the "central processing unit".

Some fundamental concepts:

To execute an instruction, the processor has to perform the following three steps:

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

Step 2: Assuming that the memory is byte addressable, increment the contents of the *PC* by *4*, that is, ***PC ←[PC]+4*** (Here 4 is added because each instruction comprises of 4 bytes)

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

Step 1 and Step 2→*fetch phase*

Step 3 → *decode phase & execution phase*

Processor Organization

To study these operations in detail there is a need to examine the internal organization of the processor. The following figure shows an organization in which the ALU and all the registers are interconnected via a single common bus. This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.

Internal organization of a processor

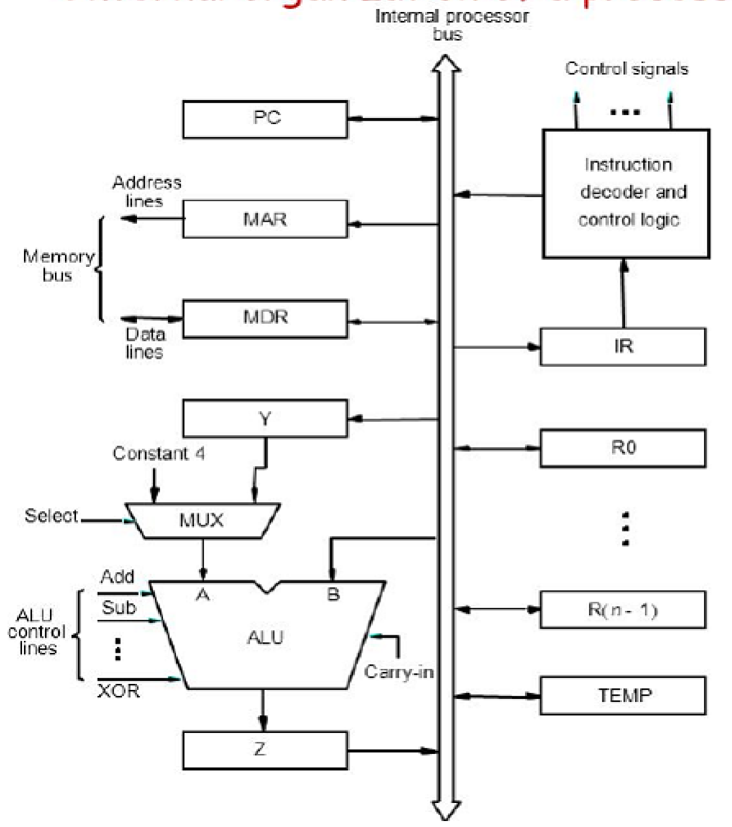


Fig-1: Single-bus organization of the datapath inside a processor

- The data and address lines of the external memory bus are connected to the internal processor bus via the memory data register, MDR, and the memory address register, MAR respectively.
- Register MDR has two inputs and two outputs.
- Data may be loaded into MDR either from the memory bus or from the internal processor bus.
- The data stored in MDR may be placed on either bus.
- The input of MAR is connected to the internal bus, and its output is connected to the external bus.
- The control lines of the memory bus are connected to the instruction decoder and control logic block.
- This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus.
- The number and use of the processor registers R0 through R(n-1) vary from one processor to another.
- Three registers X, Y &TEMP are used by the processor for temporary storage during the execution of some instructions.
- The MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU.
- The constant 4 is used to increment the contents of the program counter.
- The two possible values of the MUX control input Select are referred to as Select4 and SelectY for selecting the constant ***4*** or register ***Y*** respectively.
- The registers, the ALU and the interconnecting bus are collectively referred to as the ***datapath.***

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

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

1. Register Transfers

Instruction execution involves a sequence of steps in which data are transferred from one register to another.

•For each register two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register.

This is represented symbolically in the following figure.

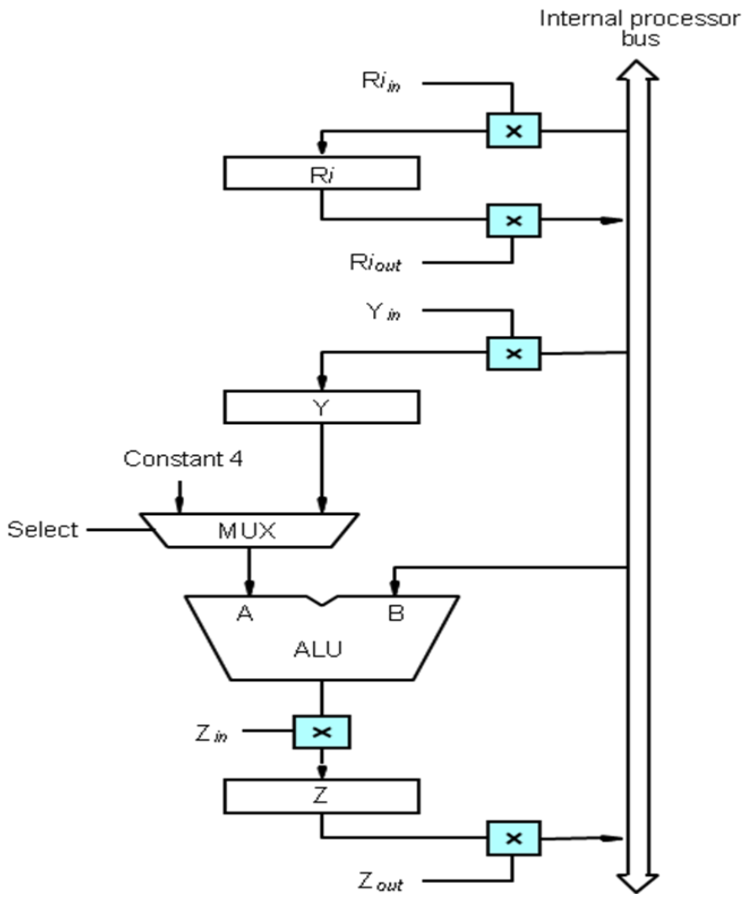


Fig-2: Input and output gating for the registers in Fig-1

•The input and output of register ***Ri*** are connected to the bus via switches controlled by the signals ***Ri***$_{in}$ and ***Ri***$_{out}$ respectively.

•When ***Ri***$_{in}$ is set to 1, the data on the bus are loaded into ***Ri.***

•Similarly, when ***Ri***$_{out}$ is set to 1, the contents of register ***Ri*** are placed on the bus.

Example

Suppose we wish to transfer the contents of register ***R1*** to register ***R4***.

This can be accomplished as follows.

•Enable the output of registers ***R1*** by setting ***R1***$_{out}$ to ***1***. This places the contents of ***R1*** on the processor bus.

• Enable the input of register ***R4*** by setting ***R4***$_{in}$ to ***1***. This loads data from the processor bus into register ***R4***.

2. Performing an arithmetic or logic operation

• The ALU is a combinational circuit that has no internal storage.

•It performs arithmetic and logic operations on the two operands applied to its A and B inputs.

•ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.

Example

What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

1. R1$_{out}$, Yin

2. R2$_{out}$, SelectY, Add, Z$_{in}$

3. Z$_{out}$, R3$_{in}$

3. Fetching a word from memory

• To fetch a word from memory the processor has to specify the address of the memory location and request a Read operation.

• The processor then transfers the required address to the MAR, whose output is connected to the address lines of the memory bus.

• At the same time the processor uses the control lines of the memory bus to indicate that a Read operation is needed.

• When the requested data are received from the memory they are stored in MDR from where they can be transferred to other registers in the processor.

The connections for the register MDR are given in the figure below.

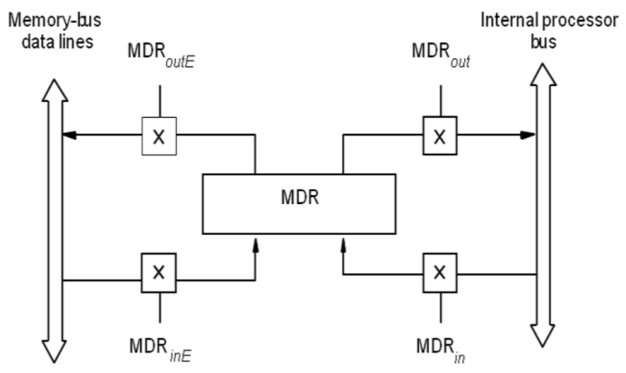


Fig-3: Connection and control signals for register MDR

It has 4 control signals:

1. MDR_{in} 2.MDR_{out} 3.MDR_{inE} 4.MDR_{outE}

MDR_{in} and MDR_{out} control the connection to the internal bus and MDR_{inE} and MDR_{outE} control the connection to the external bus.

Example

As an example of read operation, consider the instruction ***Move (R1), R2***.

The actions needed to execute this instruction are:

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

Memory read operations requires three steps, which can be described by the signals being activated as follows:

•$R1_{out}$, MAR_{in}, Read

•MDR_{inE}, WMFC

•MDR_{out}, $R2_{in}$

Note: WMFC (Wait for Memory Function Complete) is a control signal used to check whether the memory functions like Read and Write operations are completed or not.

4. Storing a word in memory

Writing a word into a memory location follows a similar procedure:

• The desired address is loaded into MAR.

• Then, the data to be written are loaded into MDR, and a write command is issued.

Example

Move R2, (R1) requires the following steps:

1. $R1_{out}$, MAR_{in}

2. $R2_{out}$, MDR_{in}, Write

3. MDR_{outE}, WMFC

Execution of a complete instruction

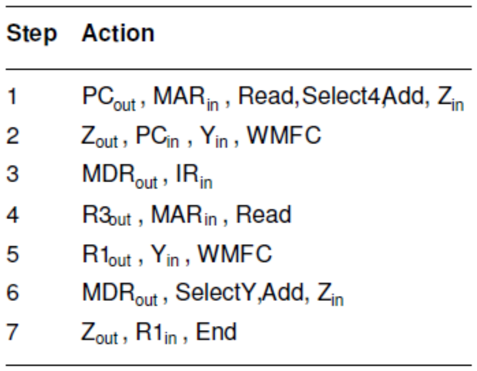
Let us now put together the sequence of elementary operations required to execute one instruction.

Example

Consider the instruction ***Add (R3), R1*** which adds the contents of a memory location pointed to by ***R3*** to register ***R1***.Executing this instruction requires the following actions:

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

The figure below gives the sequence of control steps required to perform these operations.



Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMFC
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Fig-4: Control sequence for the execution of the instruction Add (R3), R1

Instruction execution proceeds as follows:

- In step 1, the instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a Read request to the memory. The Select signal is set to Select4, which causes the multiplexer MUX to select the constant 4. This value is added to the operand at input B, which is the contents of the PC, and the result is stored in register Z.
- The updated value is moved from register Z back into the PC during step 2, while waiting for the memory to respond.
- In step 3, the word fetched from the memory is loaded into the IR.

  Steps 1 through 3 constitute the instruction *fetch phase*, which is the same for all instructions. The instruction decoding circuit interprets the contents of the IR at the beginning of step 4. This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the *execution phase*.
- The contents of register R3 are transferred to the MAR in step 4, and a memory read operation is initiated.
- Then the contents of R1 are transferred to register Y in step 5, to prepare for the addition operation.
- When the Read operation is completed, the memory operand is available in register MDR, and the addition operation is performed in step 6.
- The contents of MDR are gated to the bus, and thus also to the B input of the ALU, and register Y is selected as the second input to the ALU by choosing Select Y. The sum is stored in register Z, and then transferred to R1 in step 7. The End signal causes a new instruction fetch cycle to begin by returning to step 1.

Note: In the control sequence explained above all control signals except Y_{in} in step-2 are explained. There is no need to copy the updated contents of the PC into register Y when executing the Add instruction. But in Branch instructions the updated value of the PC is needed to compute the branch target address

Branch instructions

- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction to the updated value of PC.
- The following figure gives a control sequence that implements an unconditional branch instruction.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4,Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMF C
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Fig-5: Control sequence for an unconditional Branch instruction.

- Processing starts as usual with the fetch phase. This phase ends when the instruction is loaded into the IR in step 3.
- The offset value is extracted from the IR by the instruction decoding circuit.
- Since the value of the updated PC is already available in register Y, the offset is gated onto the bus in step 4 and an addition operation is performed.
- The result which is the branch target address is loaded into the PC in step 5.
- The offset X used on a branch instruction is usually the difference between the branch target address and the address immediately following the branch instruction.

Hardwired Control

To execute instructions the processor must have some means of generating the control signals in proper sequence. Control signals are generated by the control unit.

Control unit can be designed by using the following techniques:

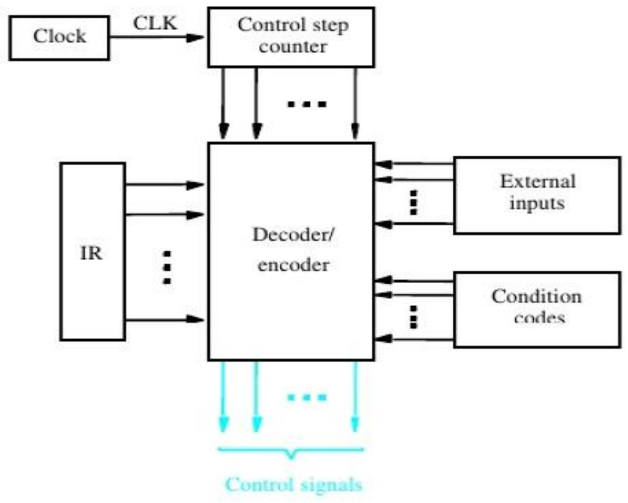
1. Hardwired control

2. Micro programmed control.

Hardwired Vs Microprogrammed

Hardwired Control	Microprogrammed control
1) It uses flipflops, decoders, logic gates and other digital circuits.	1) It uses sequence of micro-instructions in micro programming language.
2) Speed is high.	2) Speed is low.
3) More costlier.	3) Cheaper.
4) Occurrence of error is more	4) Occurrence of error is less
5) Control functions are implemented in hardware.	5) Control functions are implemented in software.
6) Not flexible to accommodate new system specification.	6) More flexible to accommodate new system specification.
7) Complicated design process.	7) Orderly, systematic and simple design process.

Simple Hardwired Control unit organization



The decoder/encoder block is a combinational circuit that generates the required control outputs, depending on the state of all its inputs.

The required control signals are determined by the following information:

•Contents of the control step counter

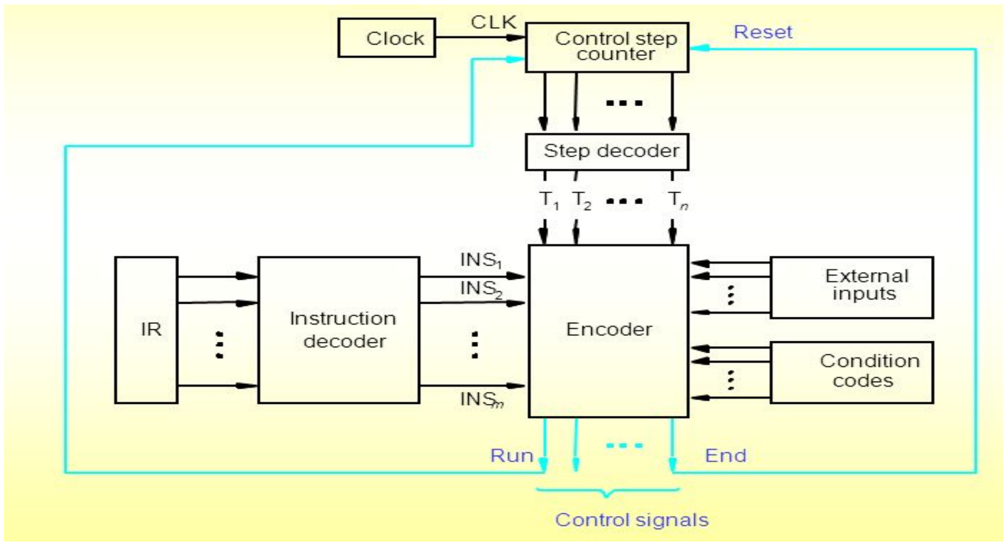
•Contents of the instruction register

•Contents of the condition code flags

•External input signals such as MFC and interrupt requests

Detailed Hardwired Control unit organization

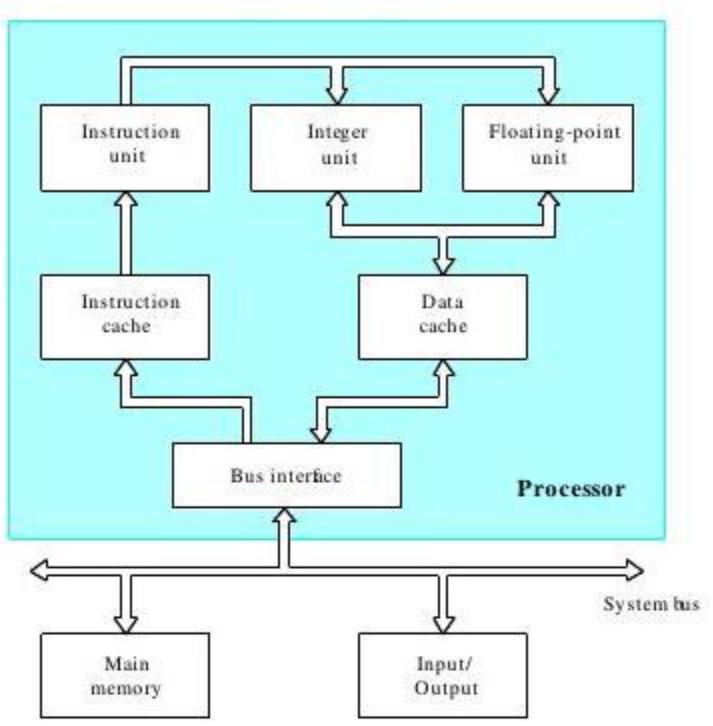
By separating the encoding and decoding functions we obtain the more detailed block diagram which is given below.



- The step decoder provides a separate signal line for each step in the control sequence.
- The output of the instruction decoder consists of a separate line for each machine instruction.
- For any instruction loaded into the IR, one of the output lines INS_1 to INS_m is set to *1* and all the others are set to *0*.
- The input signals to the encoder block are combined to generate individual control signals Y_{in}, PC_{out}, Add, End and so on.

A complete processor

- The instruction unit fetches instructions from the instruction cache or from the main memory when the desired instructions are not there in the cache.
- It has separate processing units to deal with integer data and floating-point data.
- A data cache is inserted between these units and the main memory.
- The processor is connected to the system bus and to the rest of the computer by means of a bus interface.



Microprogrammed Control

- In microprogrammed control unit, the logic of the control unit is specified by a microprogram.
- A microprogram consists of a sequence of instructions (micro-instructions) in a microprogramming language.
- Microinstructions specify micro-operations.
- Microprograms are stored in microprogram memory and the execution is controlled by microprogram counter (μPC).

Important terminologies:

Control Word (CW) :

- Control word is defined as a word whose individual bits represent the various control signals. Therefore each of the control steps in the control sequence of an instruction defines a unique combination of 0s and 1s in the CW.

Micro Program/Microroutine:

- A sequence of control words (CWs) corresponding to the control sequence of a machine instruction constitutes the microprogram for that instruction.

Micro Instruction:

- The individual control words in this microroutine are referred to as microinstructions.

Control store:

- The microroutines for all instructions in the instruction set of a computer are stored in a special memory called the control store.

An example of microinstructions for (Add (R3),R1)

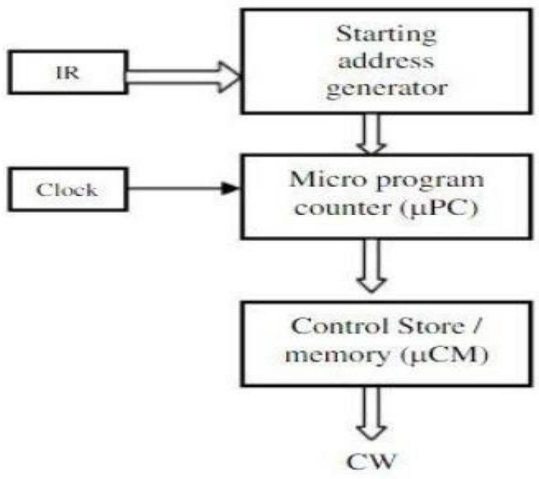
Step	Action
1.	PC $_{out}$, MAR $_{in}$, Read, Select4, Add , Z$_{in}$
2.	Z$_{out}$, PC$_{in}$, Y$_{in}$, MDR$_{inE}$ WMFC
3.	MDR$_{out}$, IR $_{in,}$
4.	R3 $_{out}$, MAR $_{in}$ Read
5.	R1 $_{out}$, Y$_{in}$, MDR$_{inE}$ WMFC
6.	MDR$_{out}$, Select Y, Add, Z$_{in}$
7.	Z$_{out}$, R1$_{in}$, End

Micro Instruction	...	PCin	PCout	MARin	Read	MDRout	IRin	Yin	Select	Add	Zin	Zout	R1out	R1in	R3out	WMFC	End	...
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0		0	0	0	1	0	1	0	0	1	

Basic organization of a microprogrammed control unit

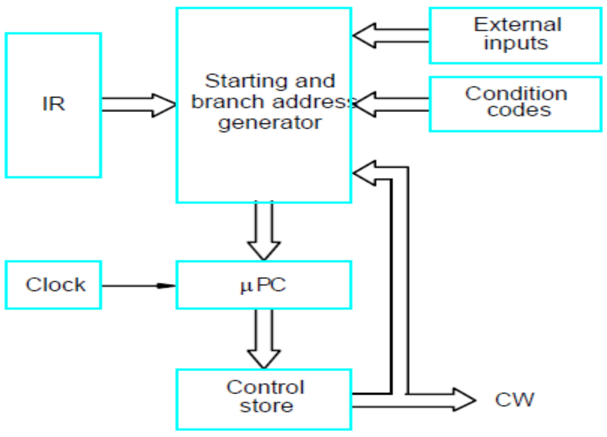
- The control unit can generate the control signals for any instruction by sequentially reading the CWs of the corresponding microroutine from the control store (microprogram memory).
- To read the control words sequentially from the control store, a microprogram counter (μPC) is used.

- Every time a new instruction is loaded into the IR, the output of the block labeled ***"starting address generator"*** is loaded into the µPC.
- The µPC is then automatically incremented by the clock, causing successive microinstructions to be read from the control store.


IR
Starting address generator
Clock
Micro program counter (µPC)
Control Store / memory (µCM)
CW

Organization of the control unit to allow conditional branching in the microprogram

- One important function of the control unit cannot be implemented by the basic organization of microprogrammed control unit.
- This is the situation that arises when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action.
- An alternative approach for microprogrammed control unit is to use conditional branch microinstructions.
- To support microprogram branching, the organization of control unit should be modified as shown in the figure below.



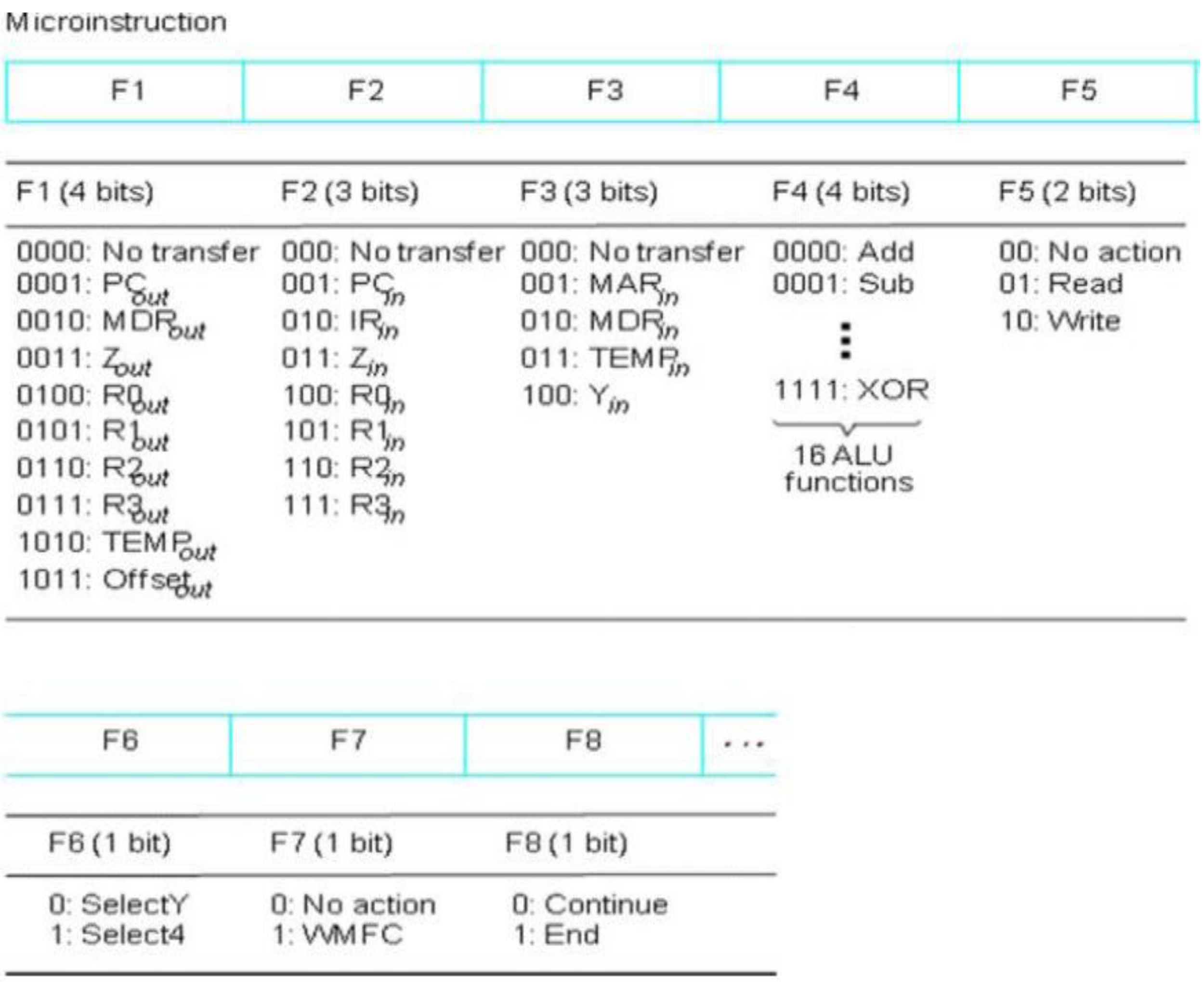
- The "Starting and branch address generator" loads a new address into the µPC when a microinstruction instructs it to do so.
- To allow implementation of a conditional branch, inputs to this block consists of the external inputs and condition codes as well as the contents of the IR.

- In this control unit, the µPC is always incremented everytime a new microinstruction is fetched from the microprogram memory, except in the following situations :

1. When a new instruction is loaded into the IR, the µPC is loaded with the starting address of the microroutine for that instruction.
2. When a branch microinstruction is encountered, the µPC is loaded with the branch address.
3. When an End microinstruction is encountered, the µPC is loaded with the address of the first CW in the microroutine for the instruction fetch cycle.

Microinstruction format

The figure given below specifies an example for a partial format for field-encoded microinstructions.


Microinstruction
F1
F2
F3
F4
F5
F6
F7
F8
. . .

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$	1111: XOR	
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	16 ALU functions	
0101: $R1_{out}$	101: $R1_{in}$			
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY	0: No action	0: Continue
1: Select4	1: WMFC	1: End