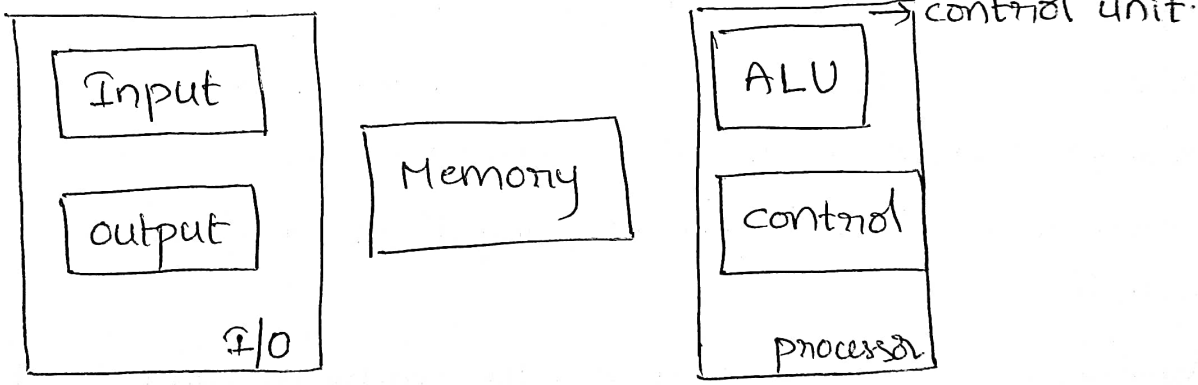
Unit-1

Basic structure of Computers

Computer organization describes the function and design of the various units of digital computers that store and process information. It also deals with the units of the computer that receive information from external sources and send computed results to external destinations.

Functional units :- A computer consists of five functionally independent main parts → input → memory → ALU → output → control unit. as shown in figure. Fig does not show connections among the functional units.



These connections, which can be made in several ways, will discuss later. We refer to the ALU along with the main control, as the processor & input and output is often collectively referred to as the input - output (I/O) unit.

Instructions or machine instructions, are explicit commands that

- govern the transfer of information within a computer as well as between the computer and its I/O devices
- specify the arithmetic and logic operations to be performed.

A list of instructions that performs a task is called a program.

Data are numbers and encoded characters that are used as operands by the instructions.

Input: The i/p unit accepts coded information from human operators, from electromechanical devices such as keyboards or from other computers over digital comm. lines. Ex: keyboard, joysticks, trackballs & mouses.

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

Primary storage is a fast memory that operates at electronic speeds. Programs must be stored in the memory while they are being executed. The memory contains a large no. of semiconductor storage cells, each capable of storing one bit of information. These cells

are processed in groups of fixed size called words, containing n bits. The no of bits in each word is often referred to as the word length of the computer. Data are usually processed within a machine in units of words, multiples of words or parts of words, when the memory is accessed, usually only one word of data is read or written.

Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random access memory (RAM). the time required to access one word is called the memory access time. This time is fixed, independent of the location of the word being accessed. It ranges from a few nanoseconds to about 100ns. The memory of a computer is normally implemented as a memory hierarchy of 3 or 4 levels of semi-conductor RAM units with different speeds and sizes.

Secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently. EX: Magnetic disks and tapes, optical disks (CD-ROMs).

Arithmetic and logic unit : Most computer operations are executed in the arithmetic and logic unit of the processor. Consider one example, suppose two numbers located in the memory are to be added. They are brough into the processor, and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use. When operands are brought into the processor, they are stored in high speed storage elements called registers. Each register can store one word of data. Access times to registers are some what faster than access times to the fastest cache unit in the memory hierarchy.

The control & ALU units are many times faster than other devices connected to a computer system. This enables a single processor to control a no. of external devices such as keyboards, displays, magnetic and optical disks, sensors and mechanical controllers.

Output : Its function is to send processed results to the outside world. Ex: printer. Some units, such as graphic displays provide both an o/p & i/p function. The dual role of such units is the reason for using the single name I/O unit in many cases.

Control unit :- The memory, ALU, I/P, O/P units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the task of the control unit. The control unit is effectively the nerve center that sends control signals to other units & senses their states.

I/O transfers, consisting of input and output operations are controlled by the instructions of I/O programs that identify the devices involved and the information to be transferred. However, the actual timing signals that govern the transfers are generated by the control circuits. Timing signals are signals that determine when a given action is to take place. Data transfers between the processor and the memory are also controlled by the control unit through timing signals. A large set of control lines carries the signals used for timing and synchronization of events in all units.

Basic operational concepts :- 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 used as operands are also stored in the memory

Add LOCA, R0 → Adds the operand at memory location A to the operand in a register in the processor, R0, and places the sum into R0. LOCA contents are preserved, those of R0 are overwritten.

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

```
Load  LOCA, R1
Add   R1, R0
```

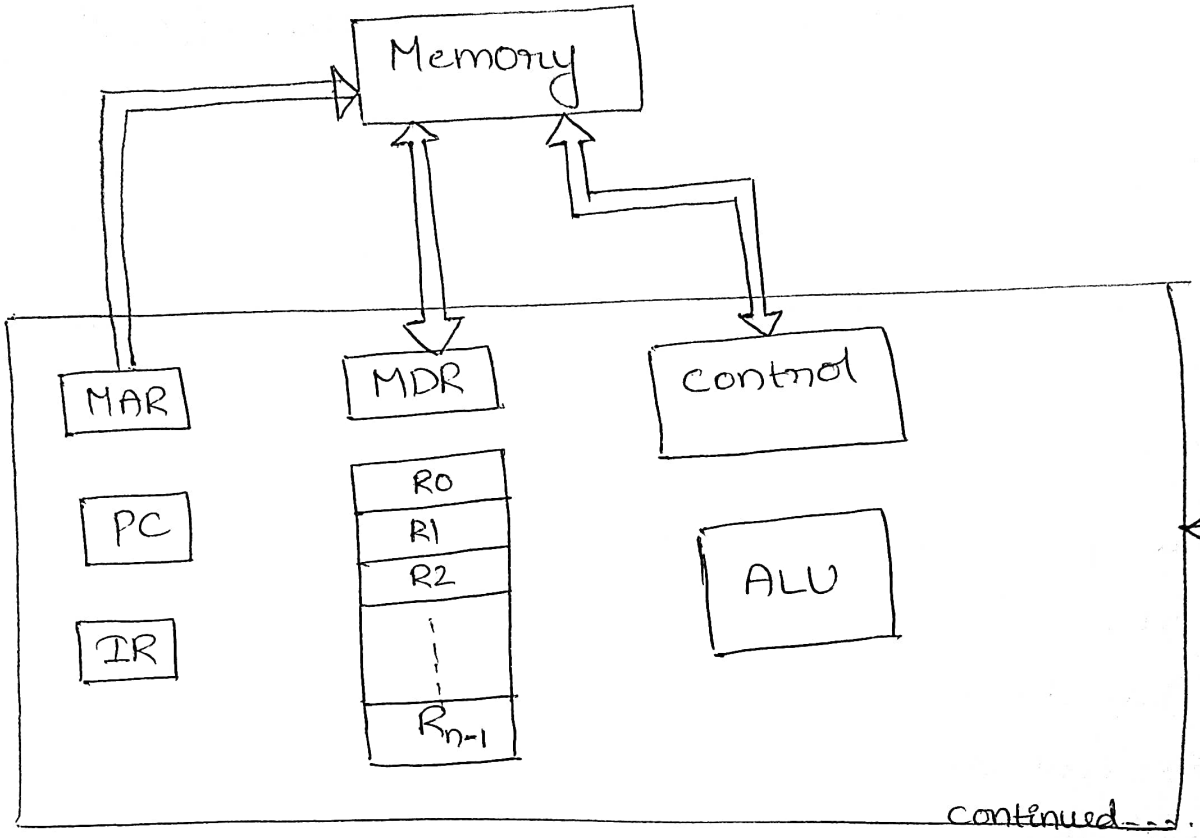
This register destroys the former contents of register R1 & as well as those of R0, where as the original contents of memory location LOCA are preserved.

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

In addition to the ALU & control unit, the processor contains a no. of registers used for several different purposes. The instruction register (IR) holds the instruction that is currently being executed. Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction. The program counter (PC) contains the memory address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the pc are updated to correspond to the address of the next instruction to be executed. Besides the IR & PC, fig shows n general purpose registers R_0 through R_{n-1}.

The Memory Address Register (MAR) holds the address of the location to be accessed. The Memory Data Register (MDR) contains the data to be written into or read out of the addressed location. These two registers faciliate communication with the memory.

Normal execution of programs may be preempted if some device requires urgent servicing. Inorder to deal with the situation immediately, the normal execution of the current program



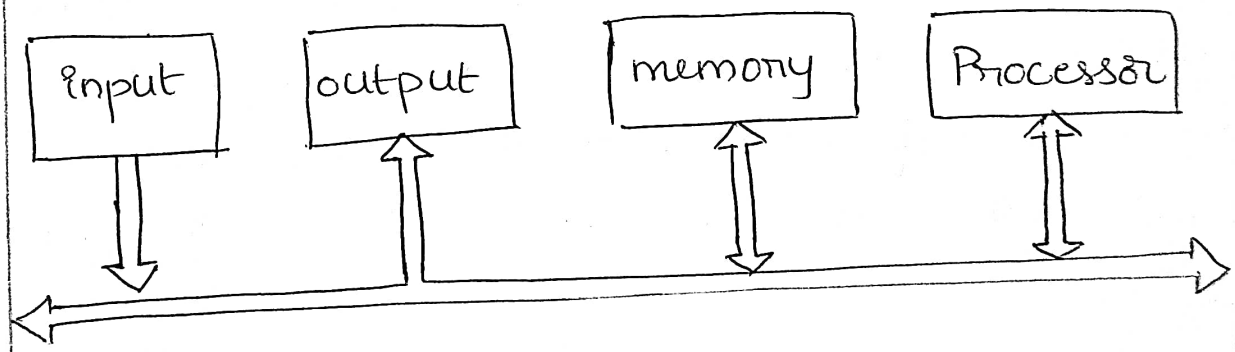
Additional :

Digital computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions and produces the resulting output information.

Types : Personal computers
desktop computers
notebook computers
work stations
enterprise systems (Servers)
Super Computers

must be interrupted. An interrupt is a request from an I/o device for service by the processor. the processor provides the requested service by executing an appropriate interrupt service routine.

Bus structures :- To form an operational system, the individual parts of a computer must be connected in some organized way. The simplest way to interconnect functional units is to use a single bus as shown below. All units are connected to bus.



Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. The main virtue of the single bus is its low cost & its flexibility for attaching peripheral devices. Systems that contain multiple buses achieve more concurrency in operations by allowing two or more transfers to be carried

out at the same time. This leads to better performance but at an increased cost.

The devices connected to a bus vary widely in their speed of operation. Some electromechanical devices, such as keyboards and printers are relatively slow. Others like magnetic or optical disks are faster. Memory and processor units operate at electronic speeds, making them the fastest parts of a computer. Because all these device must communicate with each other over a bus, an efficient transfer mechanism is necessary. A common approach is to include buffer registers with the devices to hold the information during transfers. Buffer registers smooth out timing differences among processors, memories and I/O devices.

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

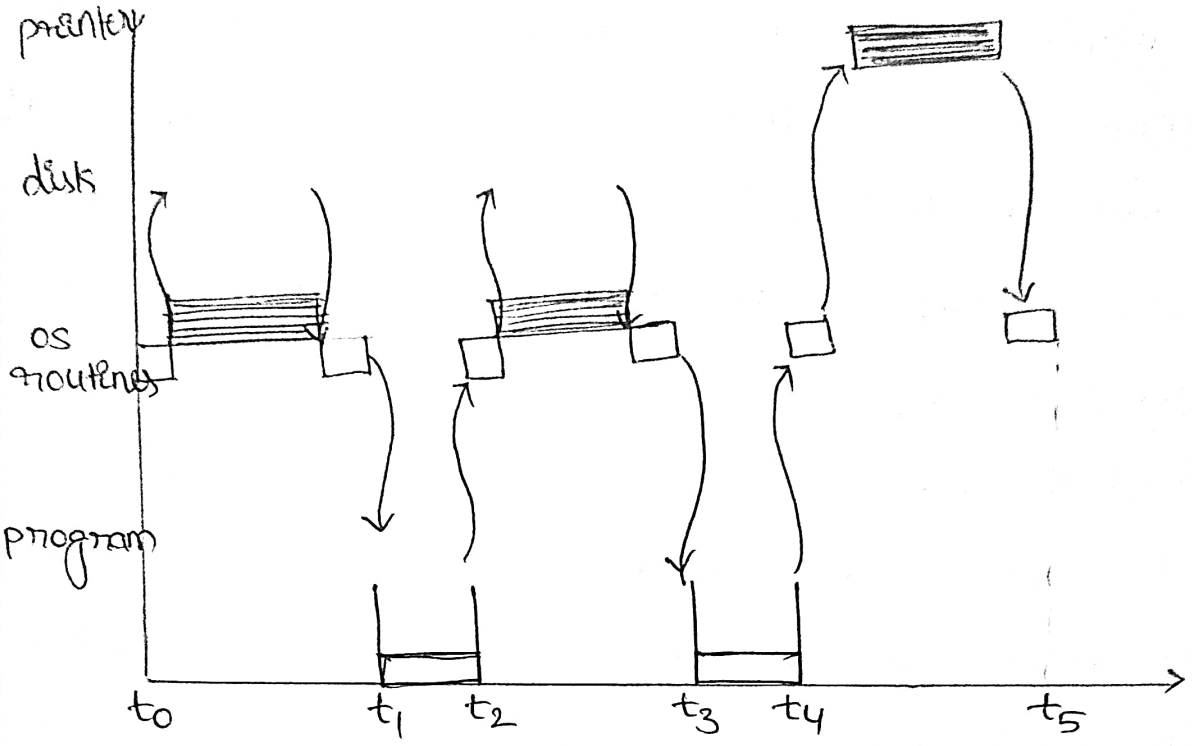
- receiving and interpreting user commands
- entering and editing application programs and storing them as files in secondary storage devices.

- Managing the storage and retrieval of files in secondary storage devices.
- running standard application programs such as word processors, spread sheets or games with data supplied by the user.
- controlling I/O units to receive input information and produce output results.
- translating programs from source form prepared by the user into object form consisting of machine instructions.
- linking and running user written application programs with existing standard library routines, such as numerical computation packages.

System software is thus responsible for the coordination of all activities in a computing system. Application programs are usually written in a high level programming language such as c, c++, Java, Fortran. A system software program called a compiler translates the high level language into a suitable machine language program.

Another important system program that all programmers use is a text editor. It is used for entering and editing application programs. Operating system is a large program or actually a collection of routines, that is used to control the sharing of and interaction among various computer units as they execute application programs. The OS routines perform the tasks required to assign computer resources to individual application programs.

Ist we discuss the steps involved in running one application program. Let us consider a system with one processor, one disk and one printer. Assume that the application program has been compiled from a high level language form into a machine language form and stored on the disk. The first step is to transfer this file into the memory. When the transfer is complete, execution of the program is started. Assume that part of the program's task involves reading a data file from the disk into the memory, performing some computation on the data and printing the results. When execusion of the program reaches the point where the data file is needed, the program requests the OS to transfer the data file from the disk to the memory.



A convienient way to illustrate this sharing of the processor execution time is by a time line diagram. During t_0 to t_1 an OS routine initiates loading the application program from disk to memory waits until the transfer is completed and then passes execution control to the application program. A similar pattern of activity occurs during t_2 to t_3 & t_4 to t_5 when the OS transfers the data file from the disk and prints the results. At t_5, the OS may load and execute another application program.

Performance: The most important measure of the performance of a computer is how quickly it can execute programs. For best performance, it is necessary to design the compiler, the machine instruction set and the hardware in a coordinated way. The total time required to execute the program in fig $t_5 - t_0$ is called elapsed time is a measure of the performance of the entire computer system. To discuss the performance of the processor, we should consider only the periods during which the processor is active. These are the periods labeled program and OS routines in fig. We will refer to the sum of these periods as the processor time needed to execute the program.

At the start of execution, all program instructions and the required data are stored in the main memory. As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache. Later, if the same instruction or data item is needed a and time, it is read directly from the cache. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use.

System topics :-

- Processor clock
- Basic performance equation
- Pipelining and superscalar operation
- clock rate
- Instruction set : CISC and RISC
- compiler
- performance measurement.

Processor clock :- Processor circuits are controlled by a timing signal called a clock. The clock defines regular time intervals called clock cycles. The length P of one clock cycle is an important parameter that affects processor performance. Its inverse is the clock rate $R = \frac{1}{P}$ which is measured in cycles per second. The term cycles per second is called hertz (Hz)

Basic performance equation :- Let T be the processor time required to execute a program that has been prepared in some high level language. Assume that complete execution of the program requires the execution of N machine language instructions. The number N

is the actual no. of instruction executions, and is not necessarily equal to the no. of machine instructions in the object program. Some instructions may be executed more than once, others may not be executed at all. Suppose that the average no. of basic steps needed to execute one machine instruction is S. If the clock rate is R cycles per second, the program execution time is given by $T = \frac{N \times S}{R}$. This is often referred to as the basic performance equation.

To achieve high performance, we must reduce the value of T, which means reducing N and S, and increasing R. The value of N is reduced if the source program is compiled into fewer machine instructions. The value of S is reduced if instructions have a smaller no. of basic steps to perform or if the execution of instructions is overlapped. We must emphasize that N, S and R are not independent parameters, changing one may affect another. A processor advertised as having a 900MHz clock doesnot neecessarily provide better performance than a 700MHz processor because it may have a different S value.

Pipelining :- A substantial improvement in performance can be achieved by overlapping the execution of successive instructions, using a technique called pipelining.

Consider the instruction Add R1, R2, R3 which adds the contents of registers R1 and R2 & places the sum into R3. The contents of R1 and R2 are first transferred to the inputs of the ALU. After the add operation is performed, the sum is transferred to R3. The processor can read the next instruction from the memory while the addition operation is being performed. 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 the Add instruction is being transferred to R3. The ideal value of $S = 1$ cannot be attained in practice. However, pipelining increases the rate of executing instructions significantly and causes the effective value of S to approach 1

Execution of several instructions in every clock cycle is called superscalar execution.

clock rate :- there are two possibilities for increasing the clock rate R. First improving the IC technology makes logic circuits faster, which reduces the time needed to complete a basic step. There by P↓, R↑.

Second, reducing the amount of processing done in one basic step also makes it possible to reduce the clock period P.

Instruction set :- Simple instructions require a small no. of basic steps to execute. Complex instructions involve a large no. of steps. A key consideration in comparing the two choices is the use of pipelining. The effective value of S in a pipelined processor is close to 1, eventhough the no. of basic steps per instruction may be considerably larger. This seems to imply that complex instructions combined with pipelining would achieve the best performance. However, it is much easier to implement efficient pipelining in processors with simple instruction sets. The suitability of the instruction set for pipelined execution is an important & often deciding consideration.

Compiler :- To reduce N, we need to have a suitable machine instruction set and a compiler that makes good use of it. An optimizing compiler takes advantage of various features of the target processor to reduce the product NXS, which is the total no. of clock cycles needed to execute program. The no. of cycles is dependent not only on the choice of instructions, but also on the order in which they appear in the program. The compiler may rearrange program instructions to achieve better performance. The compiler & the processor are often designed at the same time, with much interaction between the designers to achieve best results.

Performance measurement :- Computing the value of T is not simple. Moreover, parameters such as the clock speed and various architectural features are not reliable indicators of the expected performance. For these reasons, the computer community adopted the idea of measuring computer performance using benchmark programs. A nonprofit organization called system performance evaluation coorporation (SPEC) selects & publishes representative application programs for

different application domains, together with test results for many commercially available computers.

$$\text{SPEC rating} = \frac{\text{running time on the reference computer}}{\text{running time on the computer under test}}$$

For SPEC 95, ref is SUN SPARC station 10/40
For SPEC 2000, ref is Ultra SPARC 10

The test is repeated for all the programs in the SPEC suite, and the geometric mean of the results is computed. Let $SPEC_i$ be the rating for program i in the suite. The overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} SPEC_i\right)^{1/n}$$ where n = no. of programs in the suite.

The SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, the operating system, the processor and the memory of the computer being tested.

Historical developement of computers

Computers have been developed over the past 60 years. In the mid 1900s, gear wheels, levers, pulleys were used to perform basic operations of addition, subtraction, multiplication, division. Electro-mechanical relay devices such as those used in early telephone switching systems, provided the means for performing logic functions in computers built during worldwar II. At the same time, the 1st electronic computer was designed & built at the University of Pennsylvania, based on vaccum tube technology that was used in radios and military radars.

1st generation → 1945-1955
2nd " → 1955-1965
3rd " → 1965-1975
4th " → 1975 to present.

First generation :- The key concept of a stored program was introduced by John von Neumann. Assembly language was used to prepare programs and was translated into machine language for execution. Vaccum tube technology was implemented. Mercury delay line memory was used at first,

and I/O functions were performed by devices like typewriters. Magnetic core memories and magnetic tape storage devices were also developed.

Second generation :- The transistor was invented at AT&T Bell Laboratories in the late 1940s & quickly replaced the vacuum tube. Magnetic core memories and magnetic drum storage devices were more widely used. High level languages, such as Fortran were developed. System programs ie called compilers were developed. Separate I/O processors were developed. IBM became a major computer manufacturer during this time.

Third generation :- Integrated circuit memories began to replace magnetic core memories. Other developments included the introduction of microprogramming, parallelism and pipelining. Cache and virtual memories were developed. System 360 mainframe computers from IBM & the line of PDP (programmed data processor) minicomputers from Digital Equipment Corporation were dominant commercial products of the 3rd generation.

Fourth generation: In the early 1970s VLSI was coined, VLSI technology allowed a complete processor to be fabricated on a single chip, this became known as a microprocessor. Companies such as Intel, National Semiconductor, Motorola, Texas Instruments, Advanced Micro Devices (AMD) were the driving forces of this technology. Portable notebook computers, desktop personal computers and workstations, interconnected by LAN, WAN and the Internet, have become the dominant mode of computing.

Beyond the fourth generation: In recent years, there has been a tendency to use such features rather than a generation number to describe these evolving systems. Computers featuring artificial intelligence, massively parallel mechines and extensively distributed systems are examples of current trends. The growth of the computer industry is fueled by increasingly powerful and affordable desktop computers and widespread use of the vast information resources on the Internet.

Evolution of performance: The shift from mechanical and electromechanical devices to the 1st electronic devices based on vacuum tubes caused a 100 to 1000 fold speed increase, from seconds to milli seconds. The replacement of tubes by transistors led to another 1000 fold increase in speed, when basic operations could be performed in microseconds. Increased density in the fabrication of IC has led to current microprocessor chips that perform basic operations in nano seconds or less, achieving a further 1000 fold increase in speed. In addition to developments in technology, there have been many innovations in the architecture of computers, such as the use of caches and pipelining, which have had a significant impact on computer performance.

UNIT-2

Machine Instruction & Programs

This chapter considers the way programs are executed in a computer from the machine instruction set viewpoint. & study the ways in which sequences of instructions are brought from the memory into the processor and executed to perform a given task.

Instructions and instruction sequencing :-

The tasks carried out by a computer program consist of a sequence of small steps. A computer must have instructions capable of performing four types of operations :

- data transfers between the memory and the processor registers.
- arithmetic and logic operations on data.
- program sequencing and control.
- I/O transfers

Register transfer notation :-

We need to describe the transfer of information from one location in the computer to another. Names for the addresses of memory locations may be LOC, PLACE, A, VAR2, processor register names may be R0, R5. The contents of a location are denoted by

placing square brackets around the name of the location. Thus the expression $R1 \leftarrow [LOC]$ means the contents of memory location LOC are transferred into processor register R1.

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

$$R3 \leftarrow [R1] + [R_2]$$

This type of notation is known as Register Transfer Notation (RTN). Note that the RHS of an RTN expression always denotes a value & the LHS is the name of a location where the value is to be placed, overwriting the old contents of that location.

Assembly language notation:-

Assembly language format is the type of notation to represent machine instructions and programs. 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 R1 are overwritten.

The second example of adding two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be written as

Add R1, R2, R3

Basic instruction types :- The operation of adding two numbers is a fundamental capability in any computer. The statement C = A+B in high level language program is a command to the computer to add the current values of the two variables called A and B and to assign the sum to third variable C.

$C \leftarrow [A] + [B]$

Let us 1st assume that this action is to accomplished by a single machine Instruction. Assume that this instruction contains the memory addresses of the three operands A, B and C. This three address instruction can be represented symbolically as

Add A, B, C

Operands A and B are called the source operands, C is called the destination operand.

A general instruction of this type has the format

Operation source1, source2, destination.

If K bits are needed to specify the memory address of each operand, the encoded form of the above instruction must contain 3K bits for addressing purposes. For a modern processor with a 32 bit address space a 3 address instruction is too large to fit in one word for a reasonable word length.

An alternative approach is to use a sequence of simpler instructions to perform the same task. Suppose that two address instructions of the form

Operation source, destination

An add instruction of this type is

Add A, B

which performs the operation $B \leftarrow [A] + [B]$

When the sum is calculated, the result is sent to the memory and stored in location B, replacing B original contents. This means operand B is both a source & destination.

A single two address instruction cannot be used to solve our original problem $C \leftarrow A+B$ without

destroying either of them & to place the sum in location C. The problem can be solved by

Move B, C which performs C ← [B], leaving the contents of location B unchanged.

Move B, C

Add A, C

In above source operands are specified first, followed by the destination. This order is used in the assembly language expressions for machine instructions in many computers.

We have defined 3 & 2 address instructions. But, even two address instructions will not normally fit into one word for usual word lengths and address sizes. Another possibility is to have machine instructions that specify only one memory operand. When a second operand is needed, a processor register, usually called the accumulator, may be used for this purpose. Thus one address instruction

Add A → add the contents of memory location A to the contents of the accumulator & place the sum back into the accumulator.

Let us also introduce the one address instructions

Load A → copies the contents from A to accumulator

& Store A → copies the contents of accumulator into memory location A.

Using only one address instructions, the operation C←[A]+[B] can be performed by

```
Load A
Add B
Store C
```

The operand specified in the instruction may be a source or a destination, depending on the instruction.

Some early computers were designed around a single accumulator structure. Most modern computers have a no of general purpose processor registers. Access to data in these registers is much faster than to data stored in memory locations because the registers are inside the processor. The no of registers are relatively small, a few bits are needed to specify. For ex: 32 registers, 5 bits are needed. Because the use of registers allows faster processing & results in shorter instructions.

Let R_i represent a general purpose register. The instructions Load A, R_i

Store R_i, A

Add A, R_i are generalizations of the load, store, add instructions for the single accumulator case, in which R_i performs the function of the accumulator.

We will use move instead of load or store because it is often necessary to transfer data between different locations.

The speed with which a given task is carried out depends on the time it takes to transfer instructions from memory into the processor and to access the operands referenced by these instructions.

We have discussed 3, 2, 1 address instructions. It is also possible to use instructions in which the locations of all operands are defined implicitly. Such instructions are found in machines that store operands in a structure called a pushdown stack. In this case, the instructions are called zero address instructions.

Addressing modes: In general, a program operates on data that reside in the computer's memory. These data can be organized in a variety of ways. Programmers use organizations called data structures to represent the data used in computations. The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

Implementation of variables and constants: Variables & constants are the simplest data types and are. In assembly language, a variable is represented by allocating a register or a memory location to hold its value. We accessed an operand by specifying the name of the register or the address of the memory location where the operand is located. The definition of these modes are

register mode – the operand is the contents of a processor register, the name of the register is given in the instruction.

Absolute mode – the operand is in a memory location, the address of the location is given explicitly in the instruction.

The instruction Move LOC, R2 uses these two modes. Processor registers are used as temporary storage locations where the data in a register are accessed using the register mode. The absolute mode can represent global variables in a program. A declaration such as

Integer A, B in a high level language program will cause the compiler to allocate a memory location to each of the variables A and B. Address and data constants can be represented in assembly language using Immediate mode

Immediate mode — the operand is given explicitly in the instruction.

For ex: Move #200, R0

The sharp sign (#) infront of the value to indicate that this value is to be used as an immediate operand

ex: A = B+6

```
Move B, R1
Add #6, R1
Move R1, A
```

Indirection and pointers: In the addressing modes that follow, the instruction does not give the operand or its address explicitly. Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of the operand.

Indirect mode :- The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

The register or memory location that contains the address of an operand is called a pointer.

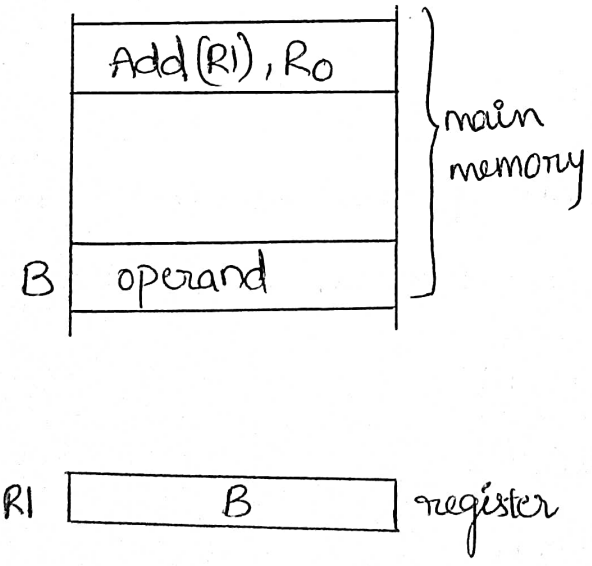


fig: through a general purpose register

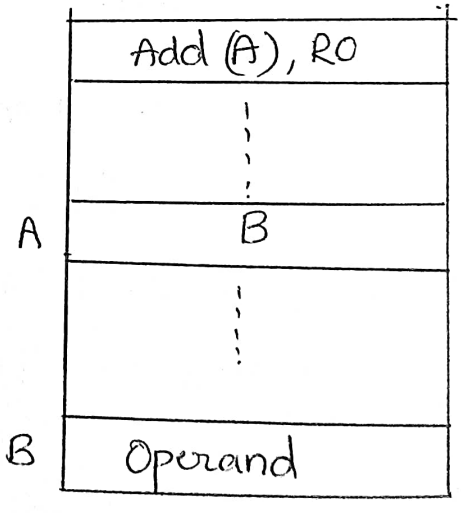


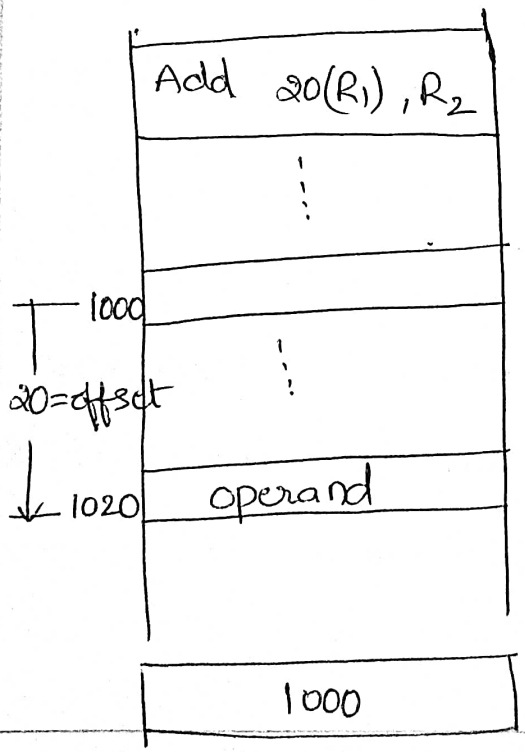
fig: through a memory location

Indexing and arrays :

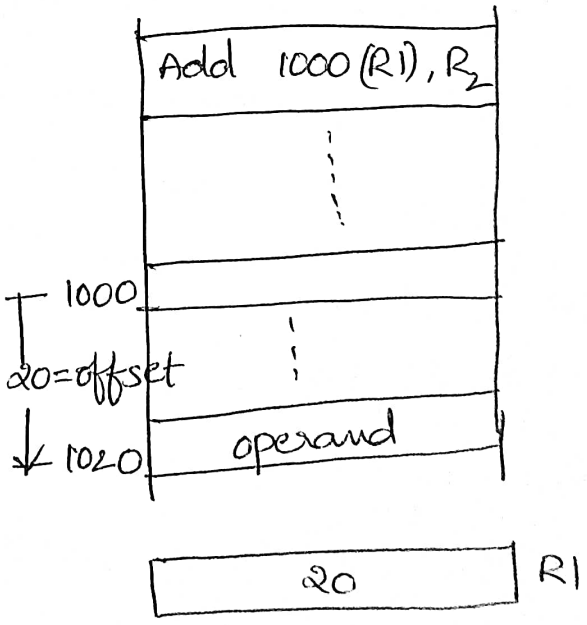
Index mode – The effective address of the operand is generated by adding a constant value to the contents of a register.

The register used may be either a special register or general purpose register. In either case, it is referred to as an index register. We indicate the index mode symbolically as $X(R_i)$ where X denotes the constant value contained in the instruction and R_i is the name of the register. The effective address of the operand is given by

$$EA = X + [R_i]$$



offset is given as a constant



offset is in the index register

Figure illustrates two ways of using the index mode. In fig1 the index register R_i contains the address of a memory location, and the value X defines an offset from this address to the location where the operand is found. In fig2, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand. In either case, the effective address is the sum of two values

A second register may be used to contain the offset X, in which we can write the index mode as (R_i, R_j). The effective address is the sum of the contents of registers R_i and R_j. The second register is usually called the base register. This form of addressing provides more flexibility in accessing operands.

Yet another version of the index mode uses two registers plus a constant, which can be denoted as $X(R_i, R_j)$. The EA is the sum of the constant X and the contents of registers R_i and R_j

Relative addressing :- This mode is obtained if the program counter (PC) is used instead of a general purpose register in index addressing mode.

Relative mode – The EA is determined by the index mode using the program counter in place of the general purpose register R_i.

This mode can be used to access data operands. But, its most common use is to specify the target address in branch instructions. An instruction such as | Branch >0 LOOP | causes program execution to go to the branch target location identified by the name LOOP if branch condition is satisfied.

Additional modes : Auto increment mode
" decrement "

Autoincrement mode – The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list

We denote the autoincrement mode by putting the specified register in parentheses to show that the contents of the register are used as the EA, followed by a + sign to indicate that these contents are to be incremented after the operand is accessed. Thus, the autoincrement mode is written as $(R_i)+$

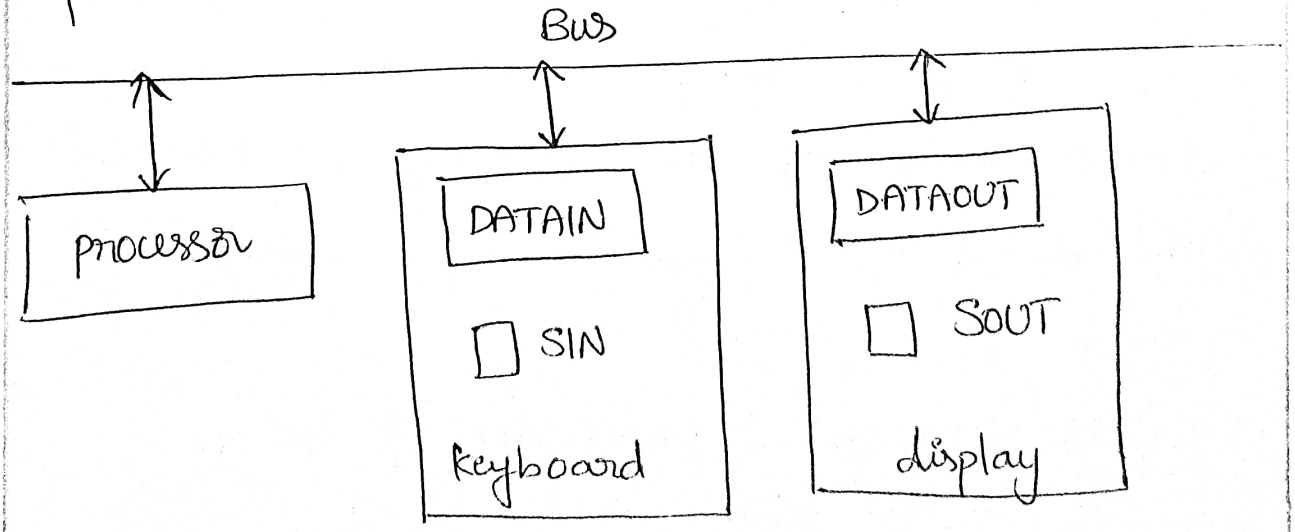
Autodecrement mode – The contents of a register specified in the instruction are first automatically decremented and are then used as effective address of the operand. This mode can be write as

$-(R_i)$

Basic input output operations :– Consider a task that reads in character input from a keyboard and produces character output on a display screen. A simple way of performing such I/o tasks is to use a method known as program controlled I/O. The rate of data transfer from the keyboard to a computer is limited by the typing speed of user. The rate of o/p transfers from the computer to the display is much higher. The difference in speed between the

processor and I/o devices creates the need for mechanisms to synchronize the transfer of data between them.

Consider the problem of moving a character code from the keyboard to the processor. Striking a key stores the corresponding character code in an 8-bit buffer register associated with the keyboard. Let us call this register DATAIN, as shown in fig. To inform the processor 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 at the output also. 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.

Inorder to perform I/O transfers, we need machine instructions that can check the state of the status flags and transfer data between the processor and the I/O device. The processor can monitor the keyboard status flag SIN & transfer a character from DATAIN to register R1 by the following sequence of operations

```
READWAIT  branch to READWAIT if SIN=0
          input from DATAIN to R1
```

The first instruction tests the status flag and second performs the branch.
An analogous sequence of operations is used for transferring output to the display

```
WRITEWAIT   branch to WRITEWAIT if SOUT = 0
            output from R1 to DATAOUT.
```

The 1st wait loop is executed repeatedly until the status flag SOUT is set to 1 by the display when it is free to receive a character. The output operation transfers a character from R1 to DATAOUT to be displayed, and it clears SOUT to 0.

Until now, we have assumed that the addresses issued by the processor to access instructions and operands always refer to memory locations. Many computers use an arrangement called memory mapped I/O in which some memory address values are used to refer to peripheral device buffer registers, such as DATAIN and DATAOUT. Thus, no special instructions are needed to access the contents of these registers, data can be transferred between these registers and the processor using instructions such as move, load, store.

The contents of the keyboard character buffer DATAIN can be transferred to register R1 in the processor by the instruction

```
Move byte DATAIN, R1
```

Similarly, the contents of register R1 can be transferred to DATAOUT by the instruction

```
Move byte R1, DATAOUT
```

The status flags SIN and SOUT are automatically cleared when the buffer registers DATAIN and DATAOUT are referenced

move → used for word (16 bits)

move byte → used for byte (8 bits)

It is more common to include SIN and SOUT in device status registers, one for each of the two devices. Let us assume that bit b_3 in registers ~~INSATU~~ INSTATUS and OUTSTATUS corresponds to SIN and SOUT respectively. The read operation can be implemented by the machine instruction sequence

```
READWAIT   testbit #3, INSTATUS
           branch =0 READWAIT
           movebyte DATAIN, R1
```

The write operation may be implemented as

```
WRITEWAIT   testbit #3, OUTSTATUS
            branch=0 WRITEWAIT
            move byte R1, DATAOUT
```

The testbit instruction tests the state of one bit in the destination location, where the bit position to be tested is indicated by the first operand. If the bit tested is equal to 0, then the condition of the branch instruction is true, and a branch is made to the begining of the wait loop. When the device is ready i.e when the bit tested becomes equal to 1, the data are read from the input buffer or written into the output buffer.

Program to read a line of characters typed at a keyboard and send them out to a display device.

```
move #LOC, R0       initialize pointer register R0 to
                    point to the address of the first
                    location in memory where the
                    characters are to be stored.
READ test bit #3, INSTATUS   wait for a character to be
     branch=0 READ           entered, in the keyboard
                             buffer DATAIN
     movebyte DATAIN, (R0)   transfer the character
                             from DATAIN into the
                             memory (this clears SIN to 0)
```

```
ECHO  testbit  #3, OUTSTATUS     wait for the display
      branch=0 ECHO              to become ready
      movebyte (R0), @DATAOUT    move the character
                                 just read to the
                                 display buffer register
                                 (this clears SOUT to 0)
      compare  #CR.(R0)+         check if the character
      branch#0                   just read is CR(carriage
      branch≠0 READ              return). If it is not CR,
                                 then branch back and
                                 read another character.
                                 Also, increment the
                                 pointer to store the
                                 next character
```

STACKS and QUEUES :-

A stack is a list of data elements, usually words or bytes, with the accessing restriction that elements can be added or removed at one end of the list only. This end is called the top of the stack, and the other end is called the bottom. The structure is sometimes referred to as a push down stack. Last in first out (LIFO) stack is also used to describe this type of storage mechanism. The terms push and pop are used to describe placing a new item on the stack and removing the

top item from the stack, respectively.

Data stored in the memory of a computer can be organized as a stack with successive elements occupying successive memory locations. Assume that the 1st element is placed in location BOTTOM, and when new elements are pushed onto the stack, they are placed in successively lower address locations.

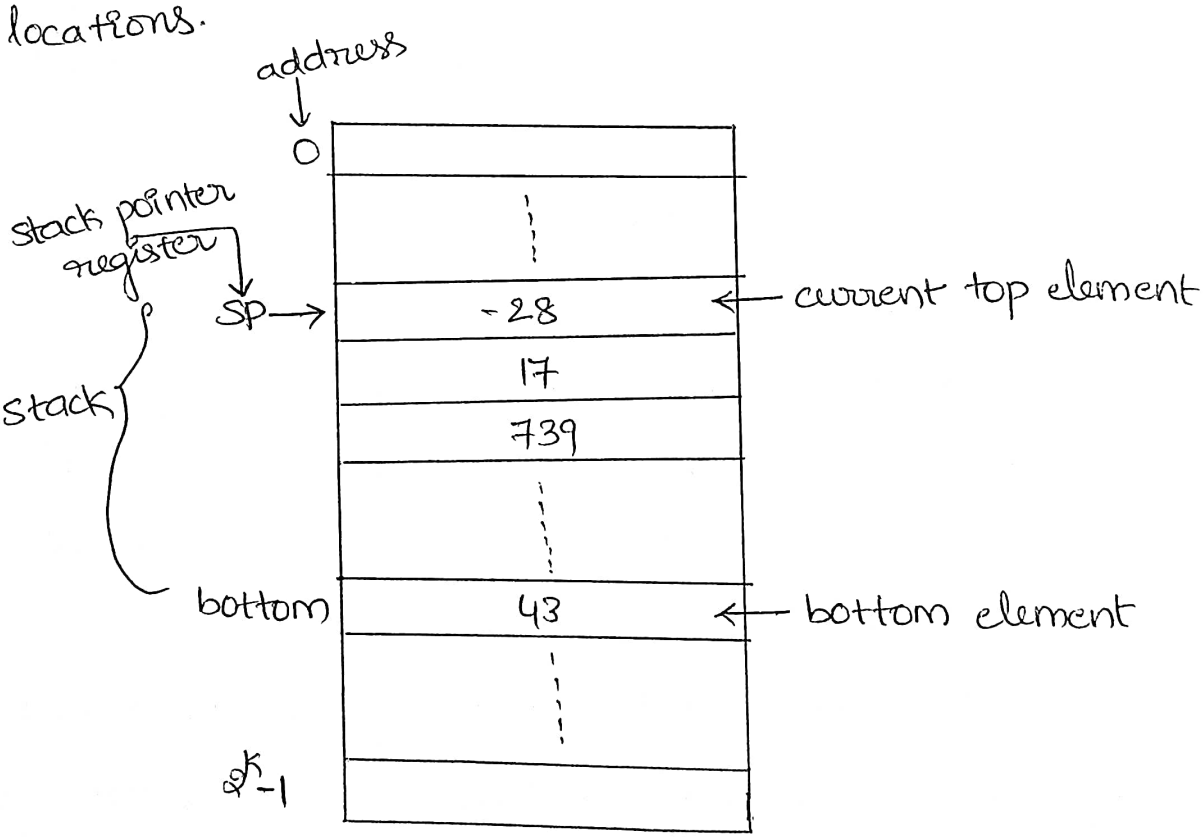


Figure shows a stack of word data items in the memory of a computer. A processor register is used to keep track of the address of the element of the stack that is at the top at any given time. This register is called the stack pointer. It is one of the

general purpose registers or a register dedicated to this function. If we assume a byte addressable memory with a 32bit word length, the push operation can be implemented as

```
subtract  #4, SP
move      newitem, (SP)
```

These two instructions move the word from location newitem onto the top of the stack, decrementing the stack pointer by 4 before the move. The pop operation can be implemented as

```
move  (SP), item
Add   #4, SP.
```

These two instructions move the top value from the stack into location item and then increment the stack pointer by 4 so that it points to the new top element.

If the processor has the autoincrement and autodecrement addressing modes, then the push operation can be performed by the single instruction

```
move  newitem, -(SP)
```

and the pop operation can be performed by

```
move  (SP)+, item
```

When a stack is used in a program, it is usually allocated a fixed amount of space in the memory. In this case, we must avoid pushing an item onto the stack when the stack has reached its maximum size. Also, we must avoid attempting to pop an item off an empty stack, which could result from a programming error. To prevent either pushing an item on a full stack or popping an item off an empty stack, the single instruction push and pop operations can be replaced by the instruction compare.

The compare instruction

compare src, dst

performs the operation [dst] - [src] and sets the condition code flags according to the result.

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, 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, 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.

Another difference between a stack and a queue is that, without further control, a queue would continuously move through the memory of a computer in the direction of higher addresses. One way to limit the queue to a fixed region in memory is to use a circular buffer. Let us assume that memory addresses from BEGINNING to END are assigned to the queue. The first entry in the queue is entered into location BEGINNING and successive entries are appended to the queue by entering them at successively higher addresses.

Logic instructions :- Logic operations such as AND, OR, NOT are done using instructions.

Not dst

complements all bits contained in the destination operand. Adding 1 to the 1's complement of a signed positive number forms the negative version in 2's complement representation.

```
Not R0
Add #1,R0
```
} 2's complement of R0

Many computers have a single instruction to do this

Negate R0 → changes the sign of R0

The AND instruction is often used in practical programming tasks where all bits of an operand except for some specified field are to be cleared to zero.

Shift and rotate instructions :- There are many applications that require the bits of an operand to be shifted right or left some specified no. of bit positions. For general operands, we use a logical shift. For a number we use an arithmetic shift, which preserves the sign of the number.

Logical shifts :-

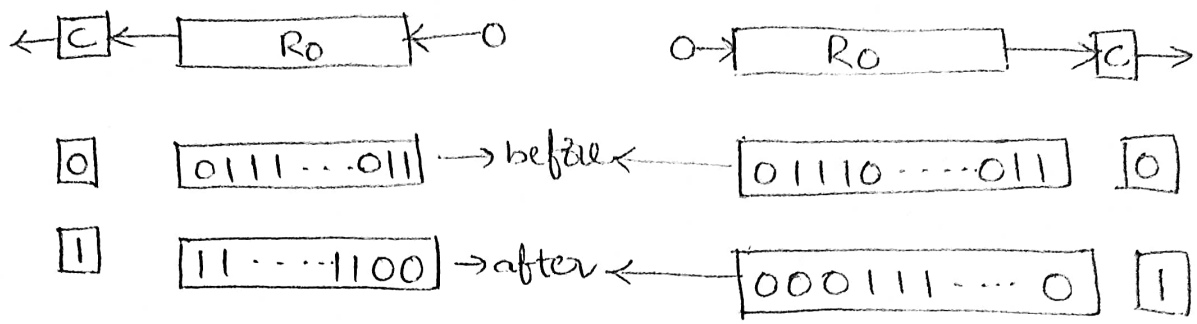
LShift L → logical shift left

LShift R → " " right

These instructions shift an operand over a no. of bit positions specified in a count operand contained in the instruction.

LshiftL count, dst

The count operand may be given as an immediate operand or it may be contained in a processor register. Vacated positions are filled with zeros, and the bits shifted out are passed through the carry flag C and then dropped.



LshiftL #2, R0

fig: Logical shift left

LshiftR #2, R0

fig: logical shift right

Digit packing example :- Suppose that two decimal digits represented in ASCII are located in memory at byte locations LOC and LOC+1. We wish to represent each of them digits in the 4-bit BCD code and store both of them in a single byte location PACKED. The result is said to be in packed BCD format. Hence, the required task is to extract the low order four bits in LOC and LOC+1 and concatenate them into the single byte at packed.

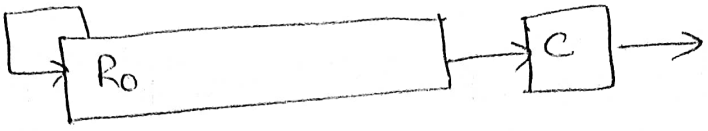
Arithmetic shifts :

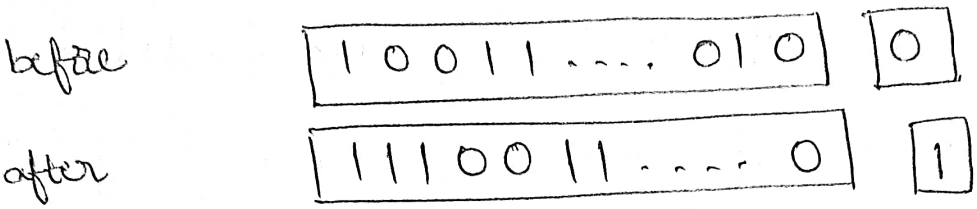
shift right = ÷2 → remainder will be lost.

" left = X2 → overflow might occur

On a right shift the sign bit must be repeated as the fill in bit for the vacated position.

AshiftR

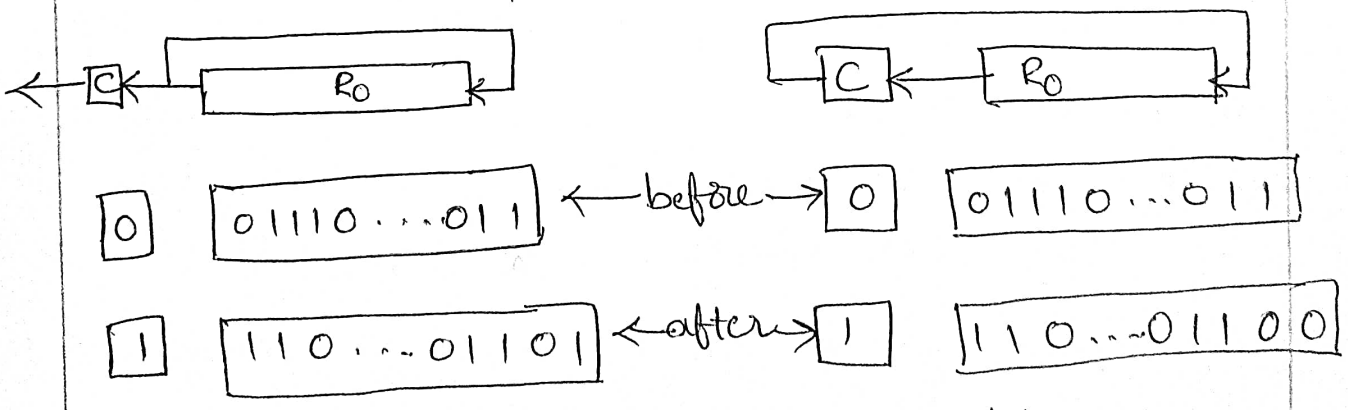




Arithmetic shift right Ashift R #2, R0

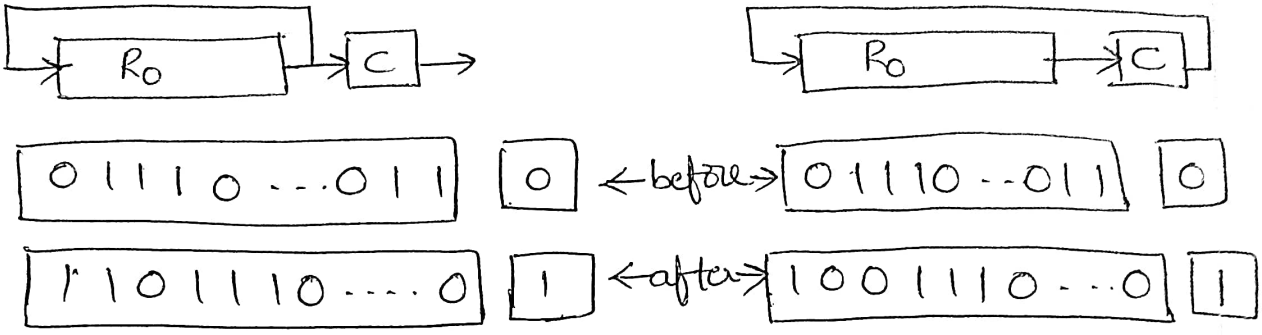
Rotate operations: In the shift operations, the bits shifted out of the operand are lost, except for the last bit shifted out which is retained in the carry flag C. To preserve all bits, a set of rotate instructions can be used. ~~Two~~ Two versions of both the left and right rotate instructions are usually provided. In one version the bits of the operand are simply rotated. In the other version, the rotation includes the C flag.

RotateL, RotateLC, rotateR and rotateRC.



rotate left without carry
RotateL #2, R0

Rotate left with carry
RotateLC #2, R0



rotate right without carry

Rotate R #2, R0

rotate right with carry

Rotate RC #2, R0

UNIT-III

Type of Instructions

Advanced RISC Machines (ARM) Limited has designed a family of microprocessors in early 1980's. The main use for ARM microprocessors is in low power and low cost embedded applications such as mobile telephones, communication modems, automotive engine management systems and hand held digital assistants.

Arithmetic & logic instructions :- The ARM instruction set has a no. of instructions for arithmetic and logic operations on operands that are either contained in general purpose registers or given as immediate operands in the instruction itself. Memory operands are not allowed for these instructions.

Arithmetic instructions :

The basic assembly language expression for arithmetic instructions is

Opcode R_d, R_n, R_m

R_n, R_m → general purpose registers

R_d → destination register

ADD R0, R_2, R_4 performs the operation

$R_0 \leftarrow [R_2] + [R_4]$

SUB R0, R_6, R_5 performs

$R_0 \leftarrow [R_6] - [R_5]$

Instead of being contained in register R_m, the second operand can be given in the immediate mode.

ADD R_0, R_3, #12

$R_0 \leftarrow [R_3] + 12$

The immediate value contained in the 8 bit field in bits b_{7-0} of the instruction.

The second operand can be shifted & rotated before being used in the operation. The instruction

ADD R_0, R_1, R_5, LSL#4

The 2nd operand contained in R_5, is shifted left 4 bit positions ($R_5 \times 16$) and then added to R_1. & sum is placed in R_0.

Two versions of a multiply instruction are provided The 1st version multiplies the contents of two registers and places the loco order 32 bits of the product in a 3rd register. The high order bits of the product, if there are any, are discarded.

MUL R_0, R_1, R_2 $R_0 \leftarrow [R_1] \times [R_2]$

The 2nd version specified a 4th register whose contents are added to the product before storing the result in the destination register.

MLA R_0, R_1, R_2, R_3 $R_0 \leftarrow [R_1] \times [R_2] + [R_3]$

This is called multiply accumulate operation. It is often used in numerical algorithms for digital signal processing. There are no provisions made for shifting or rotating any of the operands before they are used in the two multiply instructions.

Operand shift operations: One of the feature is, the shifting & rotation operations that are incorporated into most instructions. By incorporating these instructions, the ARM architecture saves code

space and can potentially improve execution time performance relative to more conventional processor designs. This feature is implemented using a barrel shifter circuit in the data path between the registers and ALU unit in the processor.

Logic instructions:

The logic operations AND, OR, XOR and bit clear are implemented by instructions with the opcodes AND, ORR, EOR and BIC. They have the same format as the arithmetic instructions.

opcode R_d, R_n, R_m

ex: AND R_d, R_n, R_m performs $R_d \leftarrow [R_n] \wedge [R_m]$ which is a bitwise logical AND between R_n and R_m.

for ex. $R_0 = (02FA\ 62CA)_h$

$R_1 = (0000\ FFFF)_h$

AND R_0, R_0, R_1 will result in pattern

0000 62CA being placed in register R_0.

The bit clear instruction (BIC) is closely related to the AND instruction. It complements each bit in operand R_m before ANDing them with R_n.

ex: BIC R_0, R_0, R_1

for the same values before we have assumed

R_0 = 02FA0000 being placed in R_0.

The move negative instruction, with the opcode mnemonic MVN, complements the bits of the source operand & places the result in R_d. If the contents of R_3 are OFOFOFOF then the instruction

MVN R_0, R_3 places the result in R_0, R_0 = FOFOFOFO

Branch instructions :- The format for the branch instructions is shown below.

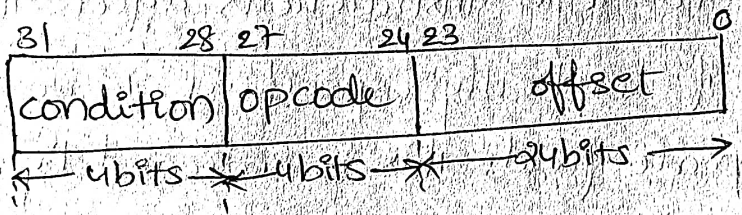


fig: instruction format

The condition to be tested to determine whether or not branching should take place is specified in the high order 4 bits b_{31-28} of the instruction word. A branch instruction is executed in the same way as any other ARM instruction i.e it is executed only if the current state of the condition

code flags corresponds to the condition specified in the condition field of the instruction.

At the time that the branch target address is computed, the contents of the PC have been updated to contain the address of the instruction that is two words beyond the branch instruction itself.

Setting condition codes:

Some instructions such as compare, given by

CMP R_n, R_m

which performs $[R_n] - [R_m]$ have the sole purpose of setting the condition code flags based on the result of the subtraction operation. On the otherhand the arithmetic and logic instructions affect the condition code flags only if explicitly specified to do so by a bit in the opcode field. This is indicated by appending the suffix S to the assembly language opcode mnemonic.

ADDS R_0, R_1, R_2 sets the condition code flags but ADD R_0, R_1, R_2 does not.

A loop program for adding numbers

```
        LDR  R1, N       Load count into R1
        LDR  R2, pointer  load address num1 into R2
        MOV  R0, #0       clear accumulator R0
loop    LDR  R3, [R2], #4 load next number into R3
        ADD  R0, R0, R3   Add number into R0
        SUBS R1, R1, #1   decrement loop counter R1
        BGT  loop         branch back if not done
        STR  R0, SUM      store sum
```

The load and store operations performed by the 1st, and and last instructions use the relative addressing mode. This assumes that the memory locations N, pointer and sum are within the range reachable by the offset relative to the PC. Memory location pointer contains the address num1 of the 1st of the numbers to be added, N contains the no of entries in the list and sum is used to store the sum.

I/O operations :- The ARM architecture uses memory mapped I/O. Suppose that bit 3 in each of the device status registers INSTATUS and OUTSTATUS contains the respective control flags SIN and SOUT. Also

assume that the keyboard DATAIN and display DATAOUT registers are located at addresses INSTATUS+4 and OUTSTATUS+4, immediately following the status register locations. The read and write wait loops can then be implemented as follows. Assume that address INSTATUS has been loaded into register R1. The instruction sequence

```
READWAIT  LDR   R3, [R1]
          TST   R3, #8
          BEQ   READWAIT
          LDRB  R3, [R1, #4]
```

reads a character into register R3 when a key has been pressed on the keyboard. The test instruction performs the bitwise logical AND operation on its two operands and sets the condition code flags based on the result. The immediate operand 8 has a single one in the bit 3 position. ∴ the result of test operation will be zero if bit 3 of INSTATUS is zero and will be nonzero if bit 3 is one, signifying that a character is available in DATAIN. The BEQ instruction branches back to READWAIT if the result is zero, looping until a key is pressed, which sets bit 3 of INSTATUS to one.

Assuming that address OUTSTATUS has been loaded into register R2, the instruction sequence

```
WRITEWAIT  LDR  R4, [R2]
           TST  R4, #8
           BEQ  WRITEWAIT
           STRB R3, [R2, #4]
```

Sends the character in register R3 to the DATAOUT register when the display is ready to receive it. These two routines can be used to read a line of characters from a keyboard, store them in the memory and echo them back to a display. Register R0 is assumed to contain the address of the 1st byte in the memory area where the line is to be stored. Registers R1 through R4 have the same usage as in the READWAIT and WRITEWAIT loops described above. The first store instruction (STRB) stores the character read from the keyboard into the memory. The Test if Equal (TEQ) instruction tests whether or not the two operands are equal, and sets the Z condition code flag accordingly.

Addressing modes :- LDR R_d, [R_n, #offset]

$R_d \leftarrow [[R_n] + offset]$,

LDR R_d, [R_n, R_m]

$R_d \leftarrow [[R_n] + [R_m]]$,

LDR R_d, [R_n]

$R_d \leftarrow [[R_n]]$

The opcode mnemonic LDR specifies that a 32 bit word is loaded from the memory into a register. A byte operand can be loaded into the low order byte position of a register by using the mnemonic LDRB. The higher order bits are filled with zeros.

Preindexed mode — The EA of the operand is the sum of the contents of base register R_n and an offset value.

$$EA = [R_n] + \text{offset}$$

Post indexed with writeback mode — The EA of operand is generated in the same way as in the preindexed mode, and then the EA is written back into R_n.

$$EA = [R_n] + \text{offset}$$
$$R_n \leftarrow [R_n] + \text{offset}$$

Post indexed mode — The EA of the operand is the contents of R_n. The offset is then added to this address and the result is written back into R_n

$$EA = [R_n]$$
$$R_n \leftarrow [R_n] + \text{offset}$$

In all three addressing modes, the offset may be given as an immediate value. Alternatively, the magnitude of the offset may be specified as the contents of the third register R_m with the sign of the offset specified by a ± prefix on the reg name

UNIT - 4

Input Output Organization

One of the basic feature of a computer is its ability to exchange data with other devices. We make extensive use of computers to communicate with other computers over the internet and access information around the globe.

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 fig

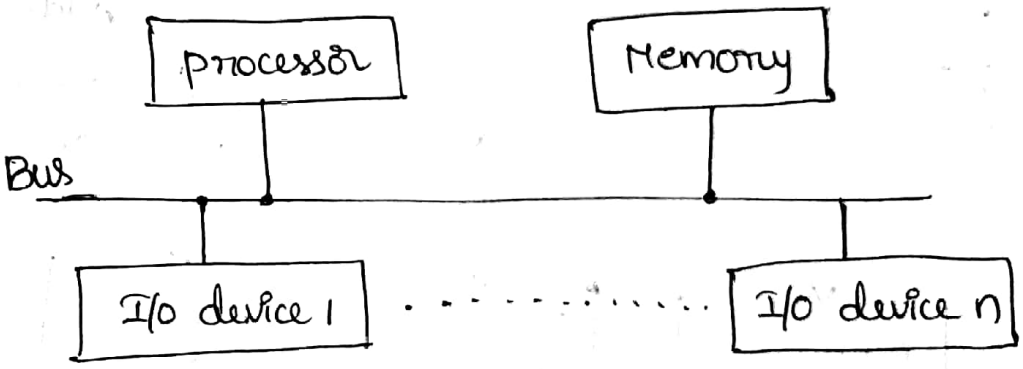


fig : single bus structure

The bus enables all the devices connected to it to exchange information. Typically it consists of three sets of lines used to carry address, data and control signals. Each I/O device is assigned a unique set of addresses. 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 write operation, the requested

data are transferred over the data lines. When I/O devices and the memory share the same address space, the arrangement is called memory mapped I/O. Most computer systems use memory mapped I/O. Some processors have special In and out instructions to perform I/O transfers. (ex Intel family).

Figure illustrates the hardware required to connect an I/O device to the bus. The address decoder

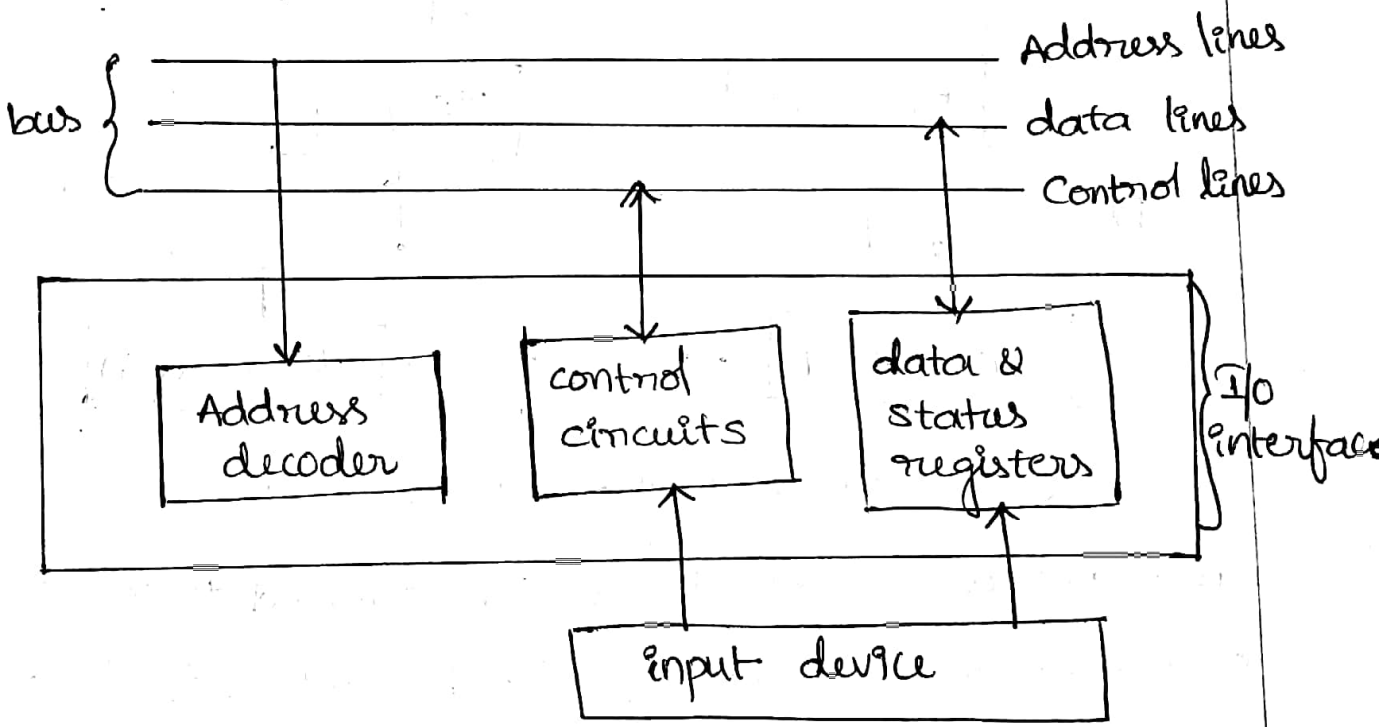


fig: I/O interface for an input device

enables the device to recognize its address when this address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register contains information relevant to the operation from the I/O device.

Both the data and status registers are connected to the data bus and assigned unique addresses. The address decoder, status register, control circuitry required to coordinate I/O transfers constitute the devices interface circuit.

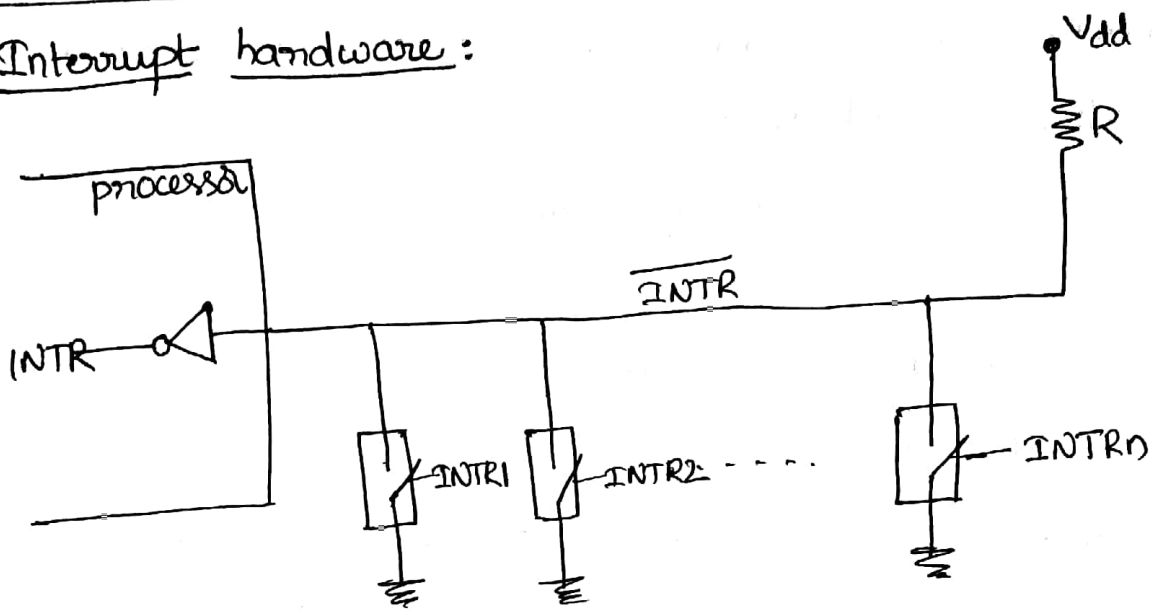
Program controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor polls the device. There are two other commonly used mechanisms for implementing I/O operations: interrupts and direct memory access. In the case of interrupts synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer. DMA is a technique used for high speed I/O devices.

Interrupts: There are many situations where other tasks can be performed while waiting for an I/O device to become ready. To allow this to happen, we can arrange for the I/O device to ~~alter~~ alter alert the processor when it becomes ready. It can do so by sending a hardware signal called an interrupt to the processor. One of the bus control lines, called an interrupt request line is usually dedicated for this.

The routine executed in response to an interrupt request is called the interrupt service routine. The processor must inform the device that its request has been recognized so that it may remove its interrupt request signal. An interrupt acknowledge signal used for this purpose.

Before starting execution of the interrupt Service routine (ISR) any information that may be altered during the execution of that routine must be saved. This information must be restored before execution of that interrupted program is resumed. The information needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the ISR. Saving registers also increases the delay b/w the time an interrupt request is received and the start of execution of the ISR. This delay is called interrupt latency. Amount of information saved should be kept minimum. The processor saves only the contents of the PC and the processor status register.

Interrupt hardware:



An I/O device requests an interrupt by activating a bus line called interrupt request. A single interrupt request line used to serve n devices. All devices are connected to the line via switches to ground. If all INTR signals are inactive i.e if all switches are open, the voltage on the $\overline{INTR}$ 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 INTR signal, received by the processor to go to 1.

The above fig is known as open collector or open drain are used to drive the $\overline{INTR}$ line. Resistor R is called pull up resistor because it pulls the line voltage upto the high voltage state when the switches are open.

Enabling and disabling interrupts: There are many situations in which the processor should ignore interrupt requests. A fundamental facility found in all computers is the ability to enable and disable such interruptions as desired.

The 1st possibility is to have the processor hardware ignore the interrupt request line until the execution of the first instruction of the interrupt service routine has been completed. Then, by using an interrupt disable instruction as the first instruction in the ISR, the programmer can ensure that no further interruptions will occur until an interrupt enable instruction is executed. Typically, the interrupt enable instruction will be the last instruction in the ISR before the return from interrupt instruction.

The 2nd option is, the processor automatically disable interrupts before starting the execution of the interrupt service routine. After saving the contents of the PC and the processor status register (PS) on the stack, the processor performs the equivalent of executing an interrupt disable instruction. It is often the case that one bit in the PS register, called interrupt enable, indicates

whether interrupts are enabled. An interrupt req, received while this bit is equal to 1 will be accepted. After saving the contents of the PS on the stack, with the interrupt enable bit equal to 1, the processor clears the interrupt enable bit in its PS register, thus disabling further interrupts. When a return from interrupt instruction is executed, the contents of the PS are restored from the stack, setting the interrupt enable bit back to 1.

In the 3rd option, the processor has a special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line is said to be edge triggered. In this case, the processor will receive only one request, regardless of how long the line is activated. Hence, there is no danger of multiple interruptions and no need to explicitly disable interrupt requests from this line.

The sequence of events involved in handling an interrupt request from a single device will be summarized as:

1. The device raises an interrupt request
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS
4. The device is informed that its request has been recognized and in response, it deactivates the interrupt request signal.
5. The action requested by the interrupt is performed by the interrupt service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

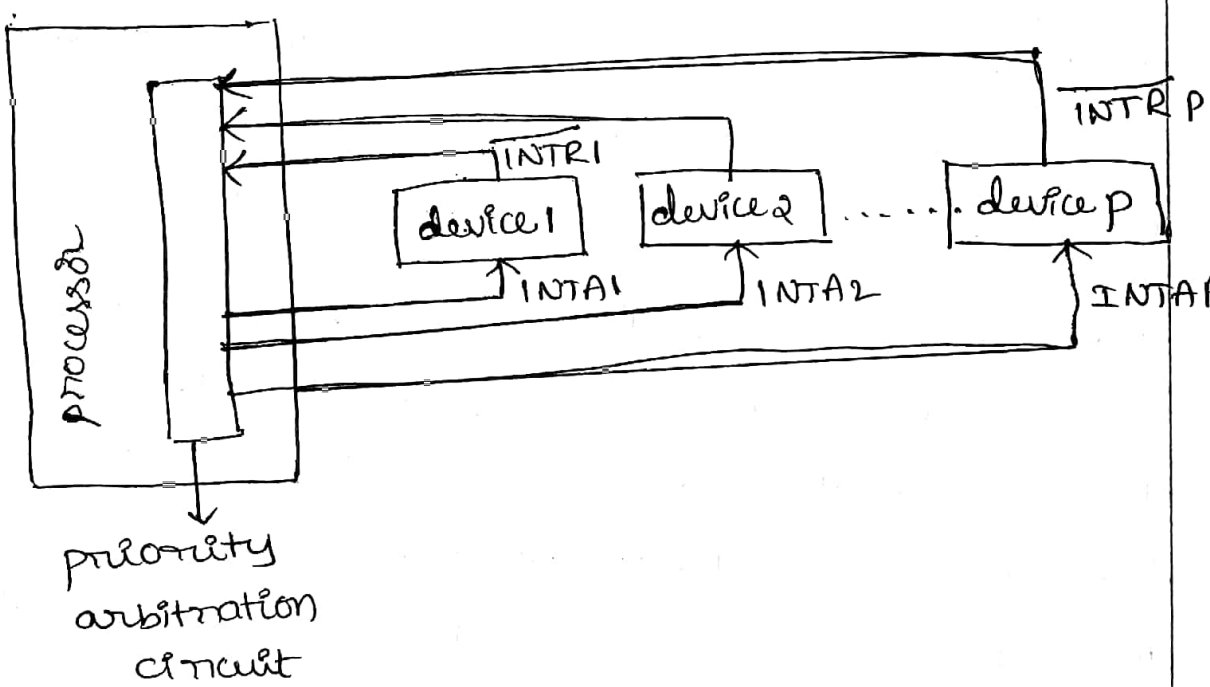
Handling multiple devices: Now consider the situation where a no. of devices capable of initiating interrupts are connected to the processor.

Vectored interrupts: A device requesting an interrupt can identify itself by sending a special code to the processor over the bus. This enables the processor to identify individual devices even if they share a single interrupt request line. The code represent the starting address of the ISR for that device.

The code length is typically in the range of 4 to 8 bits. This arrangement implies that the interrupt service routine for a given device must always start at the same location. The programmer can gain some flexibility by storing in this location an instruction that causes a branch to the appropriate routine.
The location pointed to by the interrupting device is used to store the starting address of the ISR. The processor reads this address, called the interrupt vector, and loads it into the PC. In most computers, I/O devices send the interrupt vector code over the data bus, using the bus control signals.

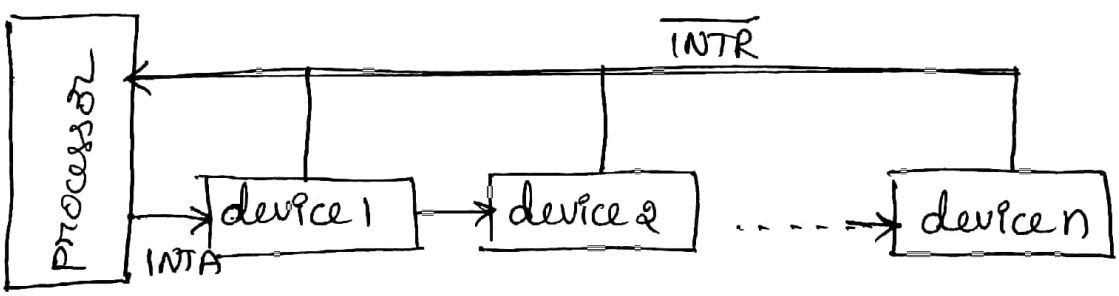
Interrupt Nesting: The interrupts should be disabled during the execution of an ISR, to ensure that a request from one device will not cause more than one interruption. If I/O devices were organized in a priority structure, interrupt requests will be accepted from some devices, but not from others. An interrupt req. from a high priority device should be accepted while the processor is servicing another request from a lower priority device. To implement this scheme, we can assign a priority level to the processor that can be changed under program control.

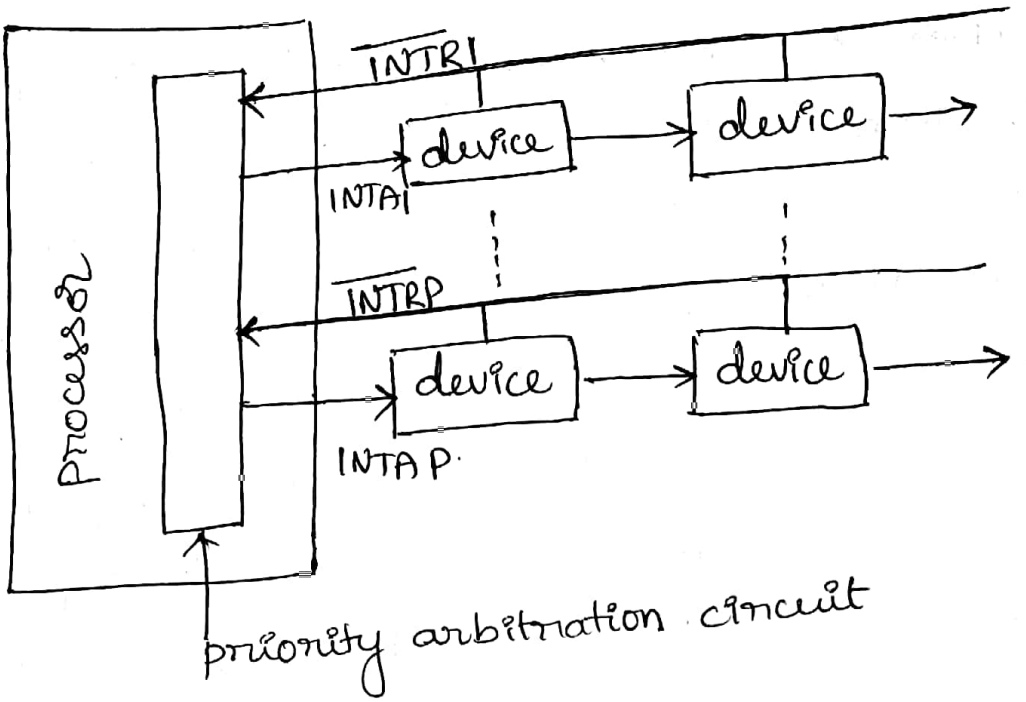
The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the PS. These are privileged instructions, which can be executed only while the processor is running in the supervisor mode. The processor is in the supervisor mode only when executing operating system routines.



A multiple priority scheme can be implemented easily by using separate interrupt request and interrupt acknowlege lines for each device as shown in fig. Each of the interrupt request lines is assigned a different priority level.

Simultaneous interrupts : Let us now consider the problem of simultaneous arrivals of interrupt requests from two or more devices. A widely used scheme is to connect the devices to form a daisy chain as shown in fig. The interrupt request line $\overline{INTR}$ 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 $\overline{INTR}$ line is activated, the processor responds by setting the INTA line to 1. This signal is received by device 1. device 1 passes the signal on to device 2 only if it does not require any service. 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 the daisy chain arrangement, the device that is closest to the processor has the highest priority





The scheme in 1st fig requires considerably fewer wires than the individual connections. ~~ther~~ In above fig devices are organized in groups, and each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.

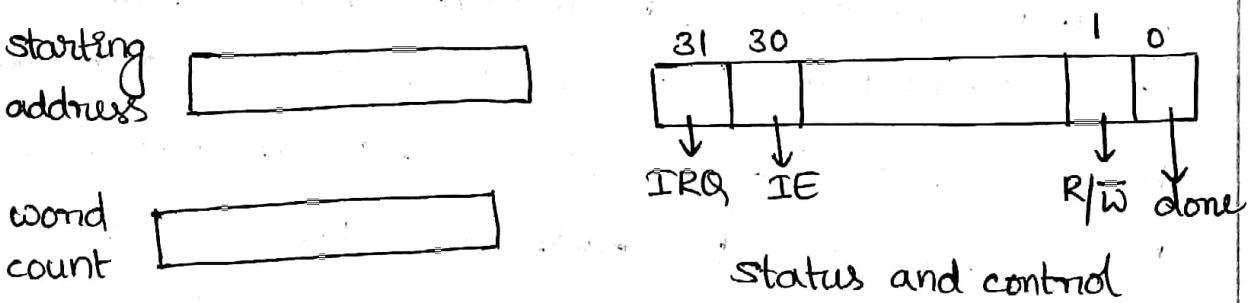
Direct Memory Access (DMA) : To transfer data between the processor and I/O devices the processor either polls a status flag in the device interface or waits for the device to send an interrupt request. In either case, considerable overhead is incurred. To transfer large blocks of data at high speed, an alternative approach is used. 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 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 the functions that would normally be carried out by the processor when accessing the main memory. For each word transferred, it provides the memory address and all the bus signals that control data transfer.

Although a DMA controller can transfer data without intervention by the processor, its operation must be under the control of program executed by the processor. To initiate the transfer of a block of words,

the processor sends the starting address, the no. of words in the block, and the direction of the transfer. On receiving this information, the DMA controller proceeds to perform the requested operation. On receiving this information, the DMA controller proceeds to perform the requested operation. When the entire block has been transferred, the controller informs the processor by raising an interrupt signal.

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.



Status and control

The R/W bit determines the direction of the transfer when R/W = 1, read operation, it transfers data from the memory to the I/O device. When the controller has completed transferring a block of data and is ready to receive another command, it sets the done flag to 1. When interrupt enable (IE) flag is set

to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data. The controller sets the IRQ bit to 1 when it has requested an interrupt.

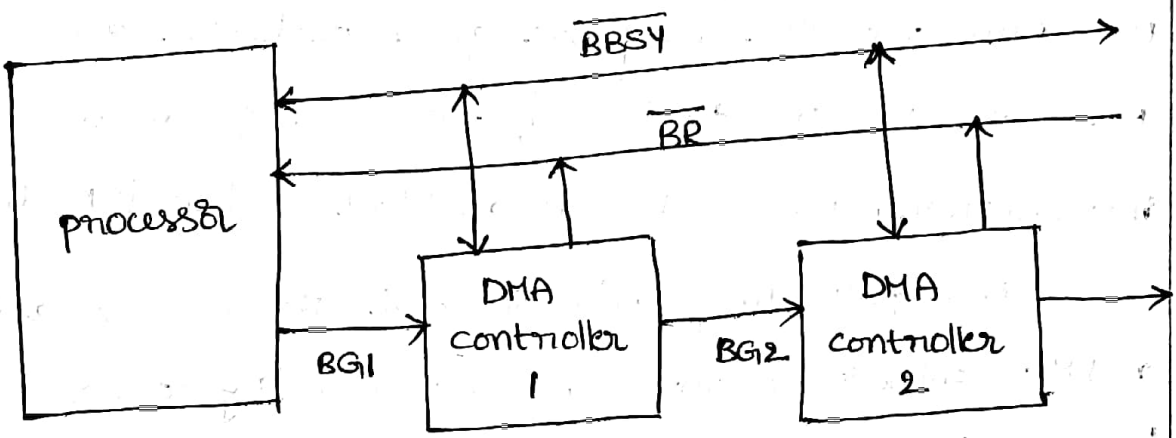
Memory accesses by the processor and the DMA controllers are interwoven. Requests by DMA devices for using the bus are always given higher priority than processor requests. Among different DMA devices, top priority is given to high speed peripherals such as a disk, a high speed network interface or a graphics display device. 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. Alternatively the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as block or burst mode.

A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve these conflicts, an arbitration procedure is implemented on the bus to coordinate the activities of all

devices requesting memory transfers.

Bus arbitration :— The device that is allowed to initiate data transfers on the bus at any given time is called the bus master. Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. There are two approaches to bus arbitration : centralized and 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 : The bus arbiter may be the processor or a separate unit connected to the bus. Figure illustrates a basic arrangement in which the processor contains the bus arbitration circuitry.



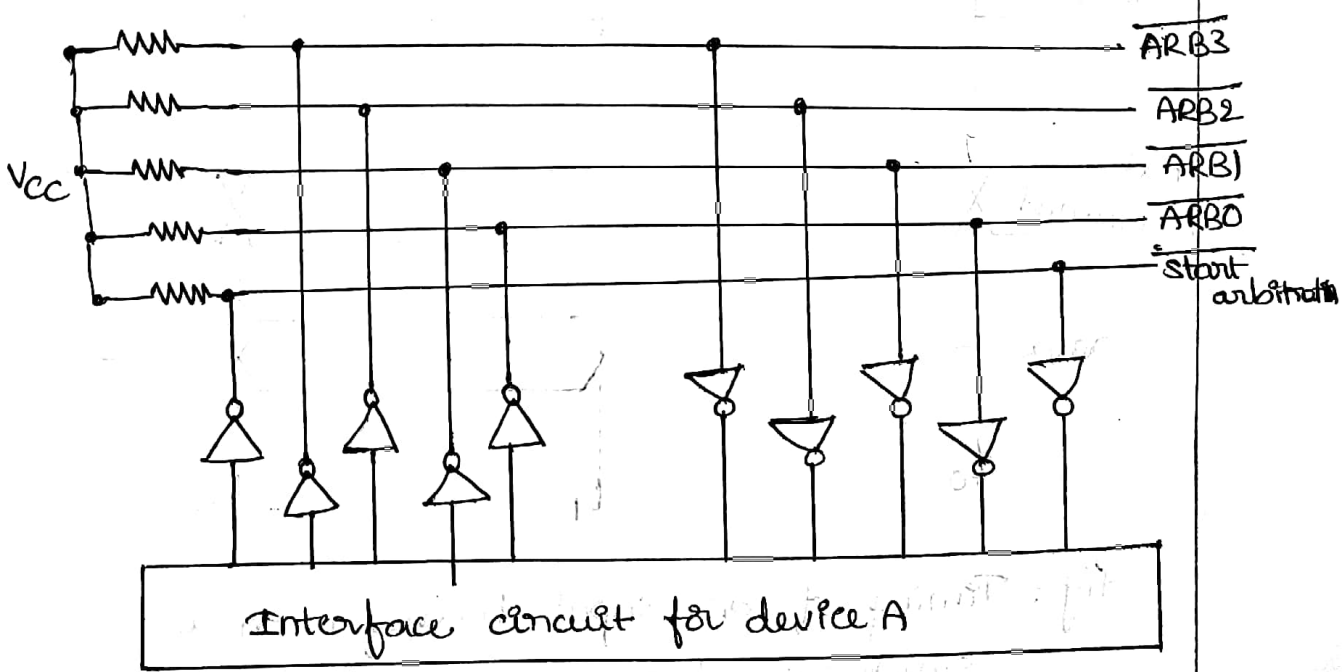
In this case, the processor is normally the bus master unless it grants bus mastership to one of the DMA controllers. A DMA controller indicates that it needs to become the bus master by activating the bus request line $\overline{BR}$. The signal on the bus request line is the logical OR of the bus requests from all the devices connected to it. When $\overline{BR}$ is activated, the processor activates the bus grant signal BG1, indicating to the DMA controllers that they may use the bus when it becomes free. This signal is connected to all DMA controllers using a daisy chain arrangement. Thus, 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 devices that it is using the bus by activating another open collector line called bus busy $\overline{BBSY}$. Hence after receiving the bus grant signal, a DMA controller waits for bus busy to become inactive, then assumes mastership of the bus. At this time, it activates bus busy to prevent other devices from using the bus at the same time.

The above fig shows one bus request line and one bus grant line forming a daisy chain. This arrangement leads to considerable flexibility in determining the order in which requests from different devices are serviced. the arbiter circuit ensures that only one request is granted at any given time, according to a predefined priority scheme. For example BR1, highest priority BR4, lowest priority (if 4 are there). Alternatively, a rotating priority scheme may be used to give all devices an equal chance of being serviced.

Distributed arbitration :- Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter. A simple method for distributed arbitration is illustrated in fig. Each device on the bus is assigned a 4bit identification number. when one or more devices request the bus, they assert the $\overline{\text{start arbitration}}$ signal and place their 4 bit ID numbers on four open collector lines, $\overline{\text{ARB0}}$ through $\overline{\text{ARB3}}$. A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders. The net

outcome is that the code on the four lines represents the request that has the highest ID number.

Assumes that two devices A and B, having ID numbers 5 and 6 respectively are requesting the use of the bus. Devices A transmits the pattern 0101, and device B transmits the pattern 0110. The code seen by both devices is 0111. Each device compares the pattern on the arbitration lines to its own ID, starting from MSB. If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower order bits. It does so by placing a 0 at the input of these drivers. Hence it disables its drivers on lines $\overline{ARB1}$ and $\overline{ARB0}$. This causes the pattern on the arbitration lines to change to 0110, which means that B has won the contention.



Buses: The processor, main memory, and I/O devices can be interconnected by means of a common bus whose primary function is to provide a communication path for the transfer of data. 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, assert control signals and so on.

Synchronous bus: In a synchronous bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals called a bus cycle during which one data transfer can take place.

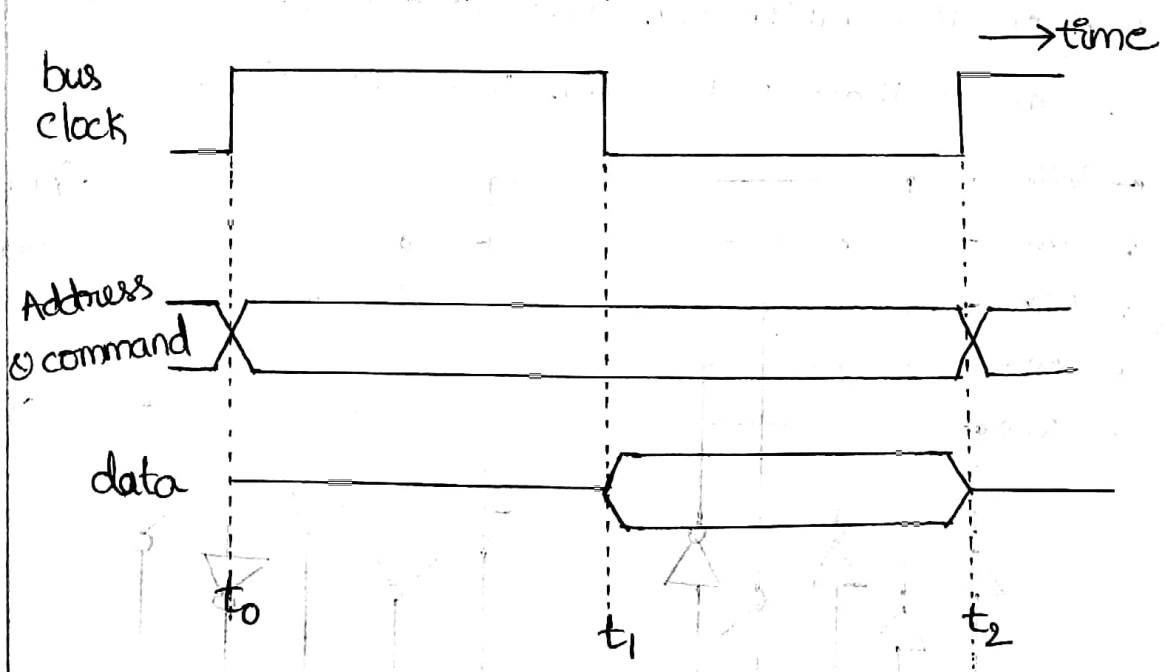


fig: Timing of an input transfer on a synchronous bus

At time t_0, the master places the device address on the address lines and sends an appropriate command on the control lines. The addressed slave places the requested input data on the data lines at time t_1. At time t_2, the master strobes the data on the data lines into its input buffer. A similar procedure is followed for an output operation.

Asynchronous Bus: An alternative scheme for controlling data transfers on the bus is based on the use of a handshake between the master and the slave. The common clock is replaced by two timing control lines, master ready and slave ready.

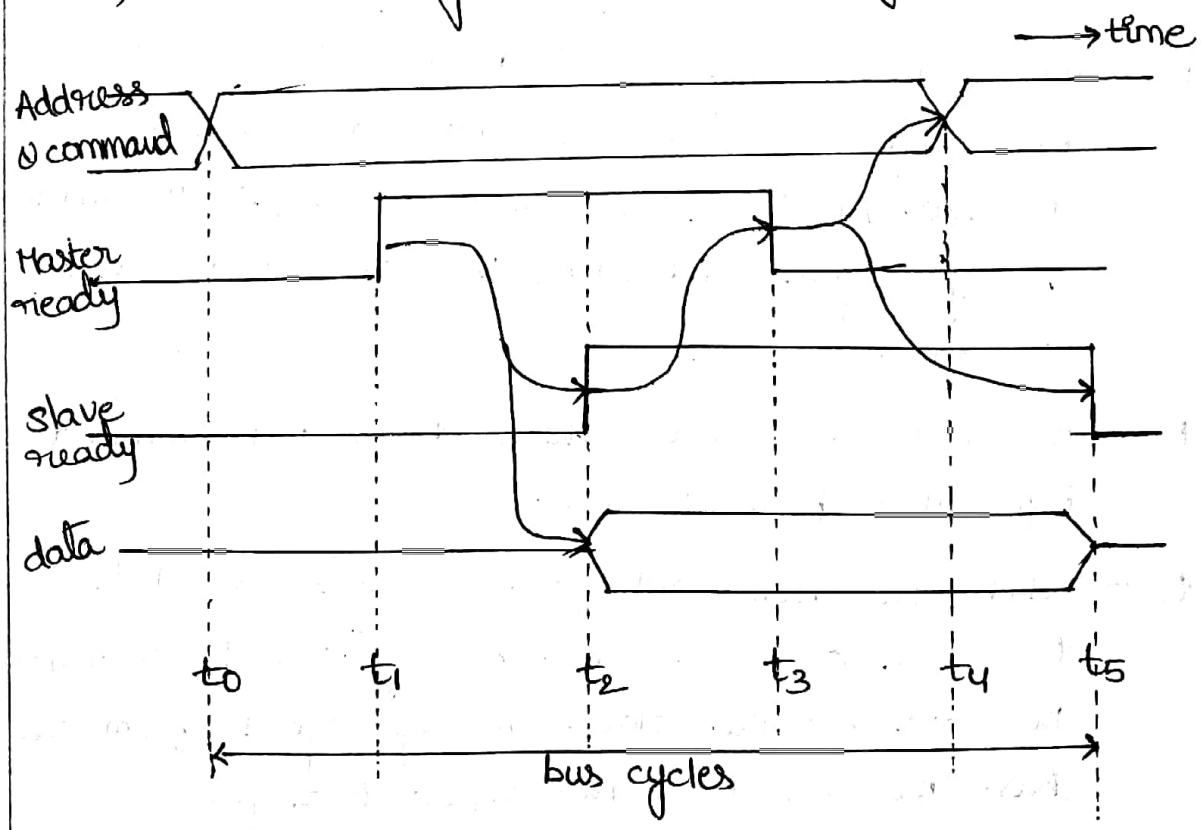


fig: Handshake control of data transfer during an input operation.

t_0 – The master places the address and command information on the bus, and all devices on the bus begin to decode this information.

t_1 – The master sets the master ready line to 1 to inform the I/O devices that the address & command information is ready. When the address information arrives at any device, it is decoded by the interface circuitry.

t_2 – The selected slave, having decoded the address and command information, performs the required input operation by placing the data from its data register on the data lines. At the same time, it sets the slave ready signal to 1.

t_3 – The slave ready signal arrives at the master, indicating that the input data are available on the bus. At the same time, it drops the master ready signal, indicating that it has received the data.

t_4 – The master removes the address and command information from the bus.

t_5 – When the device interface receives the 1 to 0 transition of the master ready signal, it removes the data and the slave ready signal from the bus. This completes the input transfer.

The handshake signals shown in above fig causes a change of state in one signal is followed by a change in the other signal. Hence this is known as a full handshake.

Interface circuits :- An I/O interface consists of the circuitry required to connect an I/O device to a computer bus. On one side of the interface we have the bus signals for address, data & control. On the other side we have a datapath with its associated controls to transfer data between the interface and the I/O device. This side is called a port, and is classified as either a parallel or serial port.

Functions of an I/O interface:

* provides a storage buffer for atleast one word of data
* contains status flags that can be accessed by the processor to determine whether the buffer is full or empty
* contains address decoding circuitry to determine when it is being addressed by the processor
* generates the appropriate timing signals required by the bus control scheme.
* Performs any format conversion (parallel ↔ serial)

Parallel port: A parallel port transfers data in the form of a no of bits (8, 16, 32, --) simultaneously to or from the device. We assume that the interface circuit is connected to a 32 bit processor that uses memory mapped I/O and the asychronous bus protocol. Figure shows the hardware components needed for connecting a keyboard to a processor.

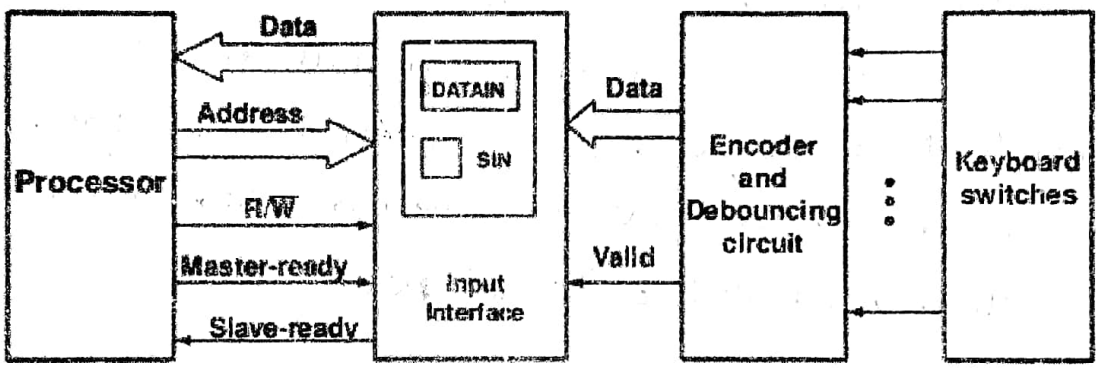


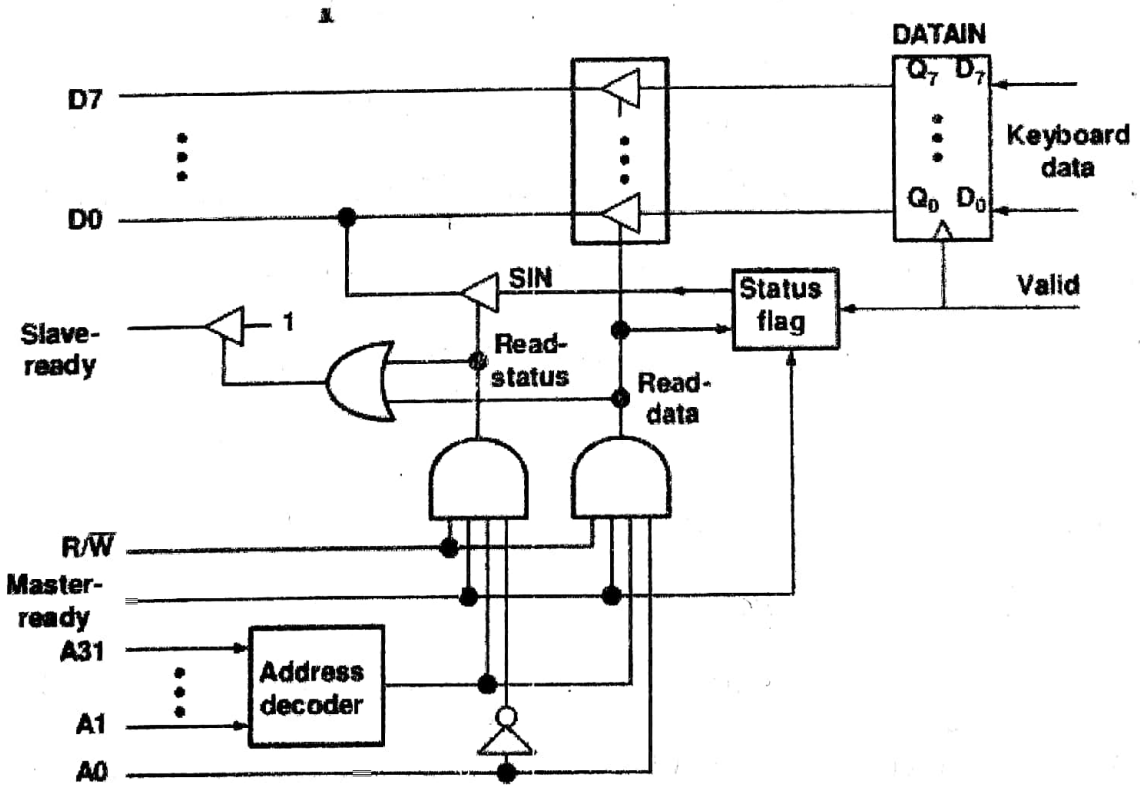
fig: Keyboard to processor connection.

A typical keyboard consists of mechanical switches that are normally open. When a key is pressed, its switch closes & establishes a path for an electrical signal. This signal is detected by an encoder circuit that generates the ASCII code for the corresponding character. The output of the encoder consists of the bits that represent the encoded character and one control signal called valid, which indicates that a key is being pressed. This information

is sent to the interface circuit, which contains a data register, DATAIN and a status flag SIN. When a key is pressed, the valid signal changes from 0 to 1, causing the ASCII code to be loaded into DATAIN and SIN to be set to 1. The status flag SIN is cleared to 0 when the processor reads the contents of the DATAIN register. The interface circuit is connected to an asynchronous bus on which transfers are controlled using the handshake signals master ready and slave ready. The control line R/$\overline{W}$ distinguishes read and write transfers.

Figure shows a suitable circuit for an input interface. The output lines of the DATAIN register are connected to the data lines of the bus by means of three state drivers, which are turned on when the processor issues a read instruction with the address that selects this register. The SIN signal is generated by a status flag circuit. This signal is also sent to the bus through a three state ~~registers~~ driver. It is connected to bit D_0, which means it will appear as bit 0 of the status register. Other bits of this register

do not contain valid information. An address decoder is used to select the input interface when the high order 31 bits of an address correspond to any of the addresses assigned to this interface. Address bit A0 determines whether the status or the data registers is to be read when the master ready signal is active. The control handshake is accomplished by activating the slave ready signal when either read status or read data is equal to 1.



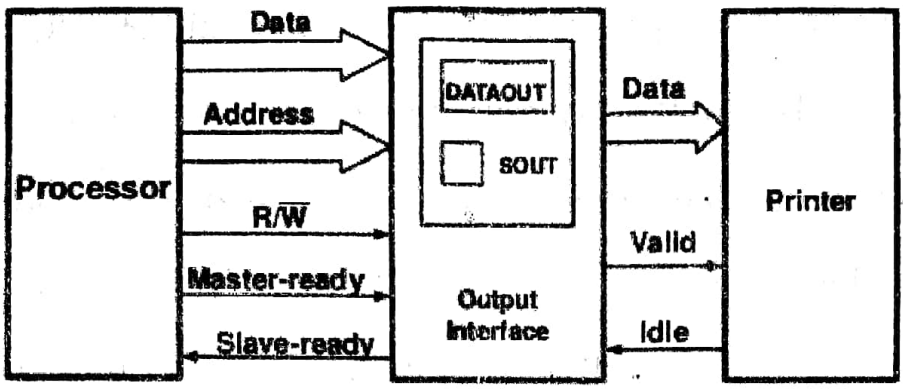


fig : printer to processor connection.

Now consider an output interface that can be used to connect an output device such as a printer, to a processor as shown in figure above.

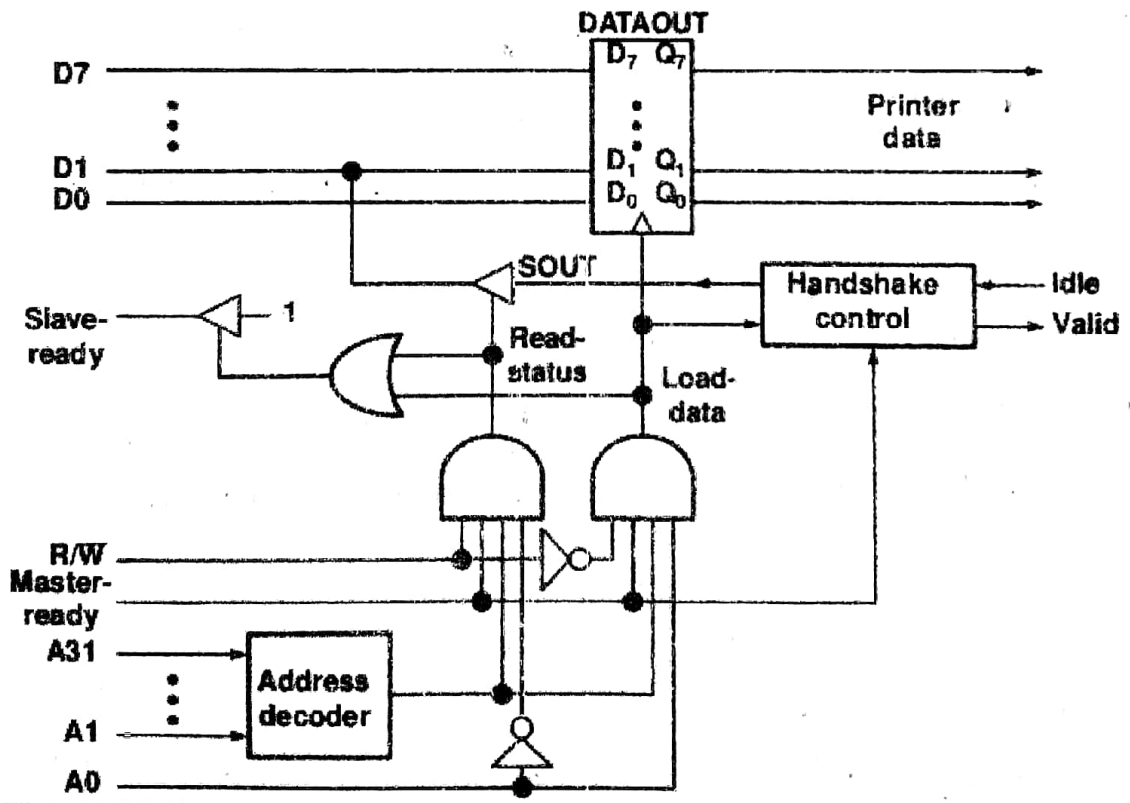


fig : output interface circuit

The printer operates under control of the handshake signals Valid and Idle in a manner similar to the handshake used on the bus with the master ready and slave ready signals. When it is ready to accept a character, the printer asserts its Idle signal. The interface circuit can then place a new character on the data lines and activate the valid signal. The printer starts printing the new character and negates the Idle signal, which in turn causes the interface to deactivate the valid signal.

The interface contains a data register, DATAOUT and a status flag, SOUT. The SOUT flag is set to 1 when the printer is ready to accept another character, and it is cleared to 0 when a new character is loaded into DATAOUT by the processor. The fig shows an implementation of this interface.

The input and output interfaces just described can be combined into a single interface as shown in next page. In this case, the overall interface is selected by the high order 30 bits of the address. Address bits A1 and A0 select one of the three addressable locations in the interface, namely, ~~the~~ the

two data registers and the status register. The status register contains the flags SIN and SOUT in bits 0 and 1, respectively.

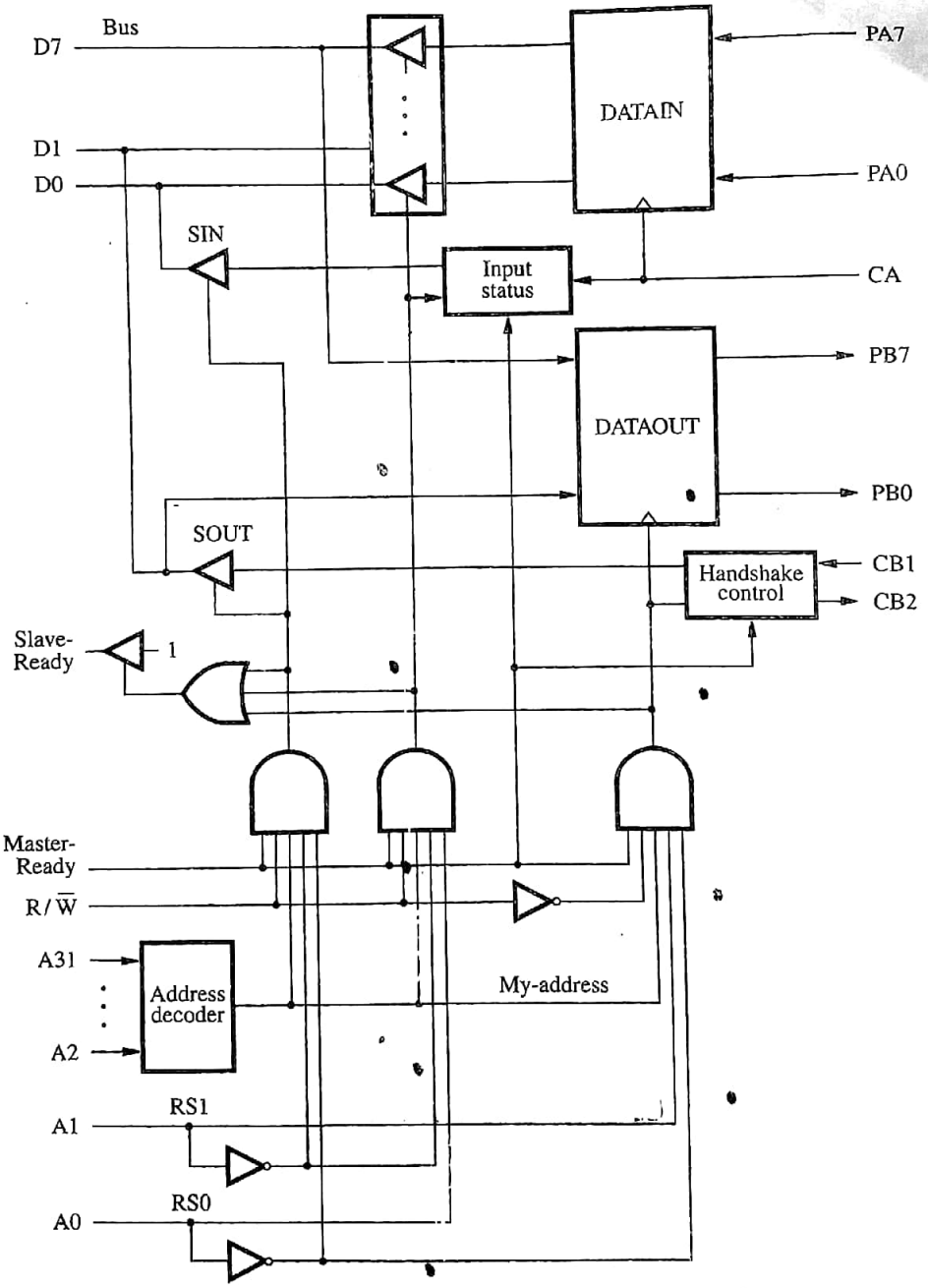
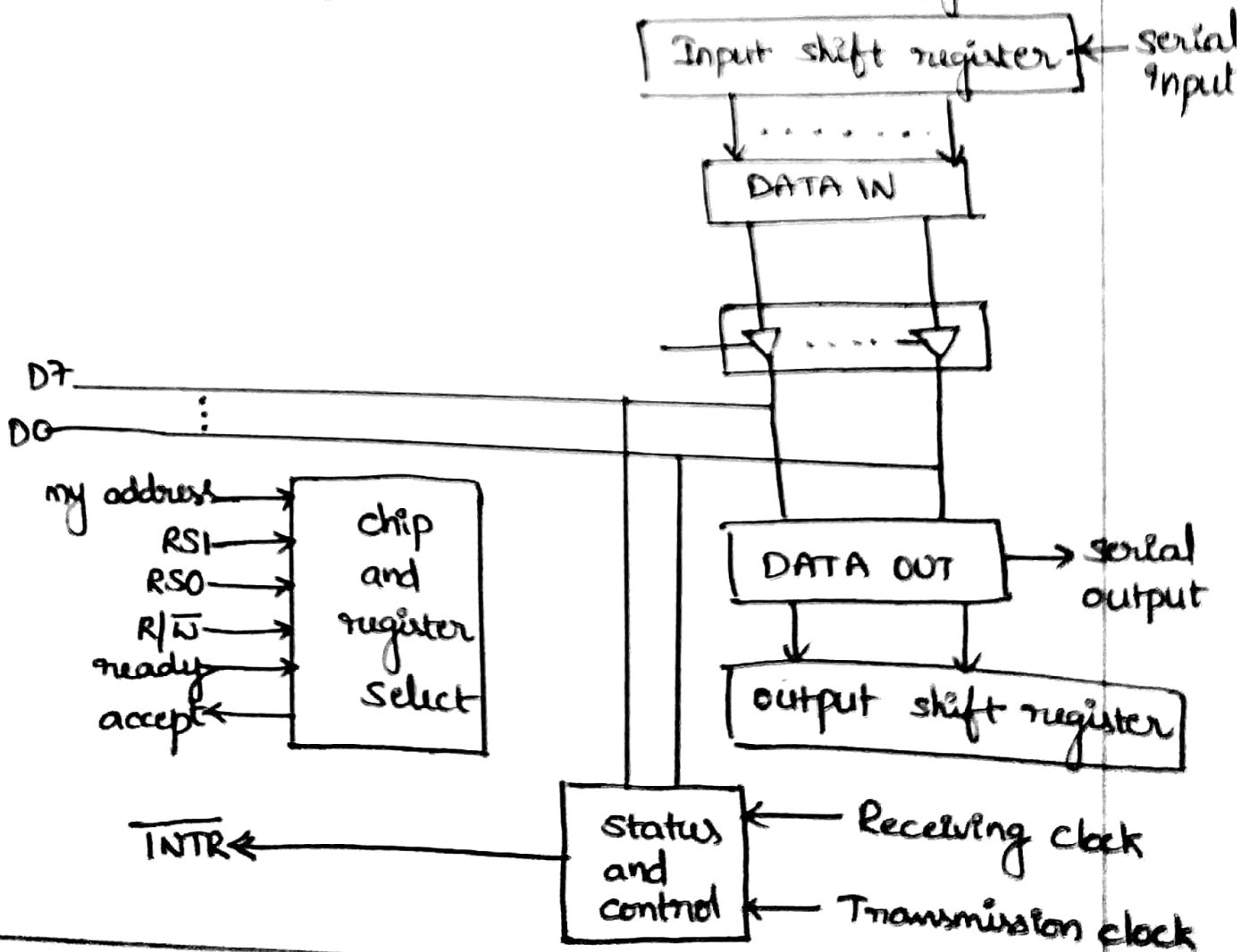


Figure 4.33 Combined input/output interface circuit.

Serial port: A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time. The feature of an interface circuit for a serial port is that it is capable of communicating in a bit serial fashion on the device side and in a bit parallel fashion on the bus side. The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability. A block diagram of a typical serial interface is shown in fig below



It includes the familiar DATAIN and DATAOUT registers. The input shift register accepts bit serial input from the I/O device. When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register. Similarly, output data in the DATAOUT register are loaded into the output shift register, from which the bits are shifted out and sent to the I/O device.

The status flags SIN and SOUT serve similar functions. The SIN flag is set to 1 when new data are loaded in DATAIN, it is cleared to 0 when the processor reads the contents of DATAIN. As soon as the data are transferred from the input shift register into the DATAIN register, the shift register can start accepting the next 8 bit character from the I/O device. The SOUT flag indicates whether the output buffer is available. It is cleared to 0 when the processor writes new data into the DATAOUT register and set to 1 when data are transferred from DATAOUT into the output shift register.

Because it requires fewer wires, serial transmission is convenient for connecting devices that are physically far away from the computer. The speed of transmission, often given as a bit rate, depends on the nature of the devices connected. A standard circuit that includes the features of serial transfer is known as a Universal Asynchronous Receiver and Transmitter (UART). It is intended for use with low speed serial devices. Data transmission is performed using the asynchronous start - stop formate. To facilitate connection to communication links, a popular standard known as RS-232-C was developed.

Standard I/O interfaces: The I/O devices fitted with an interface circuit suitable for one computer may not be usable with other computers. The most practical solution is to develop standard interface signals and protocols.

The processor bus is the bus defined by the signals on the processor chip itself. The motherboard usually provides another bus that can support more devices. the two buses are interconnected by a circuit which we will call a bridge, that translates the signals and protocols of one bus into those of the other.

It is not possible to define a uniform standard for the processor bus because the bus design dependent on architecture of the processor, electrical characteristics of the processor chip (clock speed). The expansion bus is not subject to these limitations, and therefore it can use a standardized signaling scheme. A no of standards have been developed. Some are ISA (Industry Standard Architecture) by IBM, IEEE, ANSI (American National Standards Institute), ISO, have blessed these standards & given them an official status.

Widely used bus standards are

PCI (Peripheral Component Interconnect)

SCSI (Small Computer System Interface)

USB (Universal Serial Bus)

The PCI standard defines an expansion bus on the motherboard. SCSI and USB are used for connecting additional devices, both inside and outside the computer box. The SCSI bus is a high speed parallel bus intended for devices such as disks and video displays. The USB uses serial transmission to connect the equipment like keyboards & game controls.

PCI Bus : Devices connected to the PCI bus appear to the processor as if they were connected directly to the processor bus. The PCI was developed as a low cost bus that is truly processor independent.

Data transfer : PCI is designed primarily to support burst mode data transfer. The bus supports three independent address spaces : memory, I/O, configuration. The I/O address space is intended for use with processors, such as Pentium, that have a separate I/O address space. The system designer may choose to use memory mapped I/O even when a separate I/O address

space is available. The configuration space is intended to give the PCI its plug and play capability.

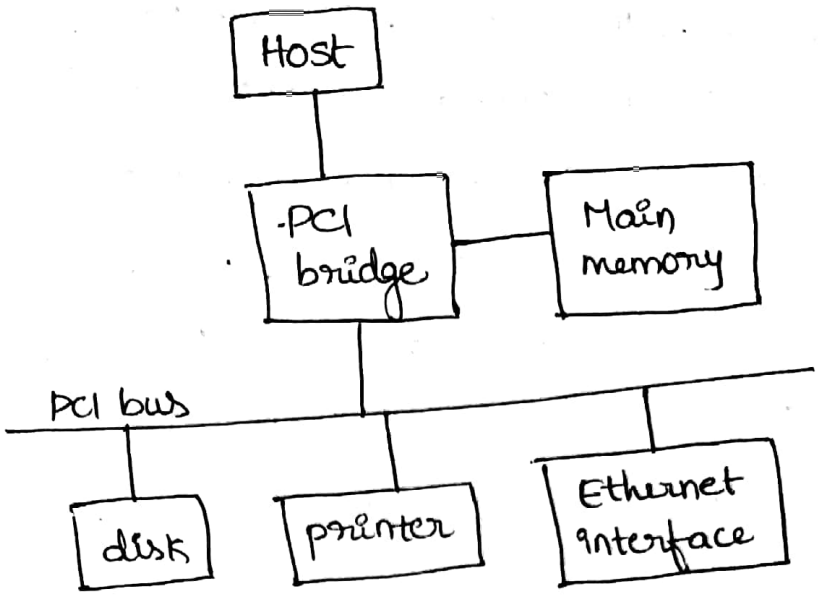


fig: Use of a PCI bus in a computer system.

At any given time, one device is the bus master. A master is called an initiator in PCI terminology. The main bus signals used for transferring data are

Name	Function. (#active low)
clk	A 33MHz or 66MHz clock
Frame #	Sent by the initiator to indicate the duration of a transaction.
AD	32 add/data lines or increased to 64
C/BE#	4 command/byte enable lines
IRDY #, TRDY #	initiator ready and target ready signals
DEVSEL #	A response from the device indicating that it has recognized its address & is ready for a data transfer transaction
IDSEL #	initialization device select

Consider a bus transaction in which the processor reads four 32 bit words from the memory. In this case, the initiator is the processor and the target is the memory. A complete transfer operation on the bus, involving an address and a burst of data is called a transaction. Individual word transfers within a transaction are called phases. The sequence of events on the bus is shown in fig below.

In clock cycle 1, the processor asserts FRAME# to indicate begening of a transaction. At the same time, it sends the address on the

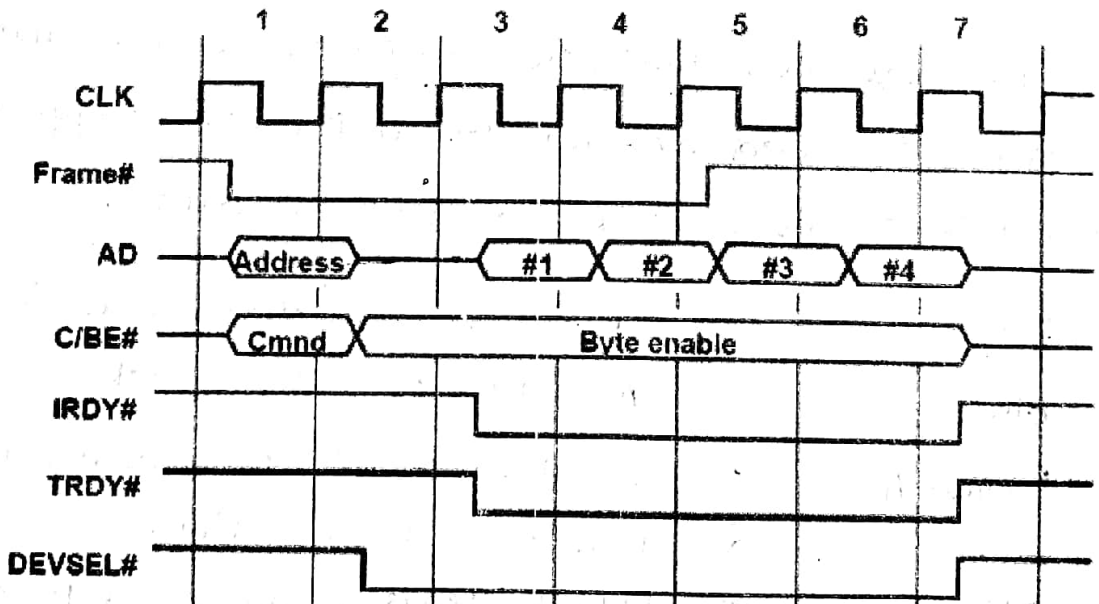


fig : read operation on the PCI Bus

AD lines and a command on the C/BE# lines. clock cycles 2 is used to turn the AD bus lines around. The processor removes the address & disconnects its drivers from the AD lines.

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 target sends three more words of data in clock cycles 4 to 6.

The initiator uses the frame # signal to indicate the duration of the burst. It negates this signal during the second last 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 4th word in clock cycle 6, the target disconnects its drivers and negates DEVSEL# at the beginning of clock cycle 7.

Device configuration: When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it. A typical device interface card for an ISA bus, for example, has a no. of jumpers or switches that have to be set by the user to select

certain options. Once the device is connected, the software needs to know the address of the device. The PCI simplifies this process by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device. The configuration ROMs of all devices are accessible in the configuration address space. The PCI initialization software reads these ROMs whenever the system is powered up or reset. In each case, it determines whether the device is a printer, a keyboard, an ethernet interface or a disk controller. Devices are assigned addresses during the initialization process.

Electrical characteristics : The PCI bus has been defined for operation with either a 5V or 3.3V power supply. The motherboard may be designed to operate with either signalling system. Connectors on expansion boards are designed to ensure that they can be plugged only in a compatible motherboard.

Universal Serial Bus (USB) : It was developed by several computer & communications companies including compaq, Hewlett - Packard (HP), Intel, Lucent, Microsoft, Nortel, Networks, philips.

The USB supports several speeds of operation, called

low speed (1.5Mbps)

full speed (12Mbps)

high speed (480Mbps)

The USB has been designed to meet several objectives

* provide a simple, low cost and easy to use interconnection system that overcomes the difficulties due to the limited no. of I/O ports available on a computer
* accommodate a wide range of data transfer characteristics for I/O devices, including telephone & internet connections.
* enhance user convenience through a plug and play mode of operation.

Port limitation: Basically a few ports are provided in a typical computer. To add new ports, a user must open the computer box to gain access to the internal expansion bus and install new interface card. An objective of the USB is to make it possible to add many devices to a computer system at any time without opening the computer box.

Device characteristics: In the case of a keyboard one byte of data is generated every time a key is pressed, which may happen at any time. Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by the keyboard are called asynchronous which is quite low (1000 bits/second).

Let us consider another source of data. The sound picked up by the microphone produces an analog electrical signal, which must be converted into a digital form before it can be handled by the computer. This is done by sampling the analog signal periodically. The sampling process yields a continuous stream of digitized samples that arrive at regular intervals, synchronized with the sampling clock. Such a data stream is called isochronous, meaning that successive events are separated by equal periods of time. Data transfers for images and video have similar requirements, but at much higher data transfer bandwidth. Large storage devices such as hard disks and CD-ROMs provide a data transfer bandwidth of atleast 40 or 50 Mbps.

Plug and play: the plug and play feature means that a new device, such as an additional speaker can be connected at any time while the system is operating. The system should detect the existence of this new device automatically, identify the appropriate device-driver software and assign establish the appropriate addresses and logical connections to enable them to communicate.

USB architecture: A serial transmission format has been chosen for the USB because a serial bus satisfies the low cost and flexibility requirements. clock and data & information are encoded together and transmitted as a single signal. To accommodate a large no. of devices that can be added or removed at any time, the USB has the tree structure as shown in fig. 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, which are called functions in USB terminology.

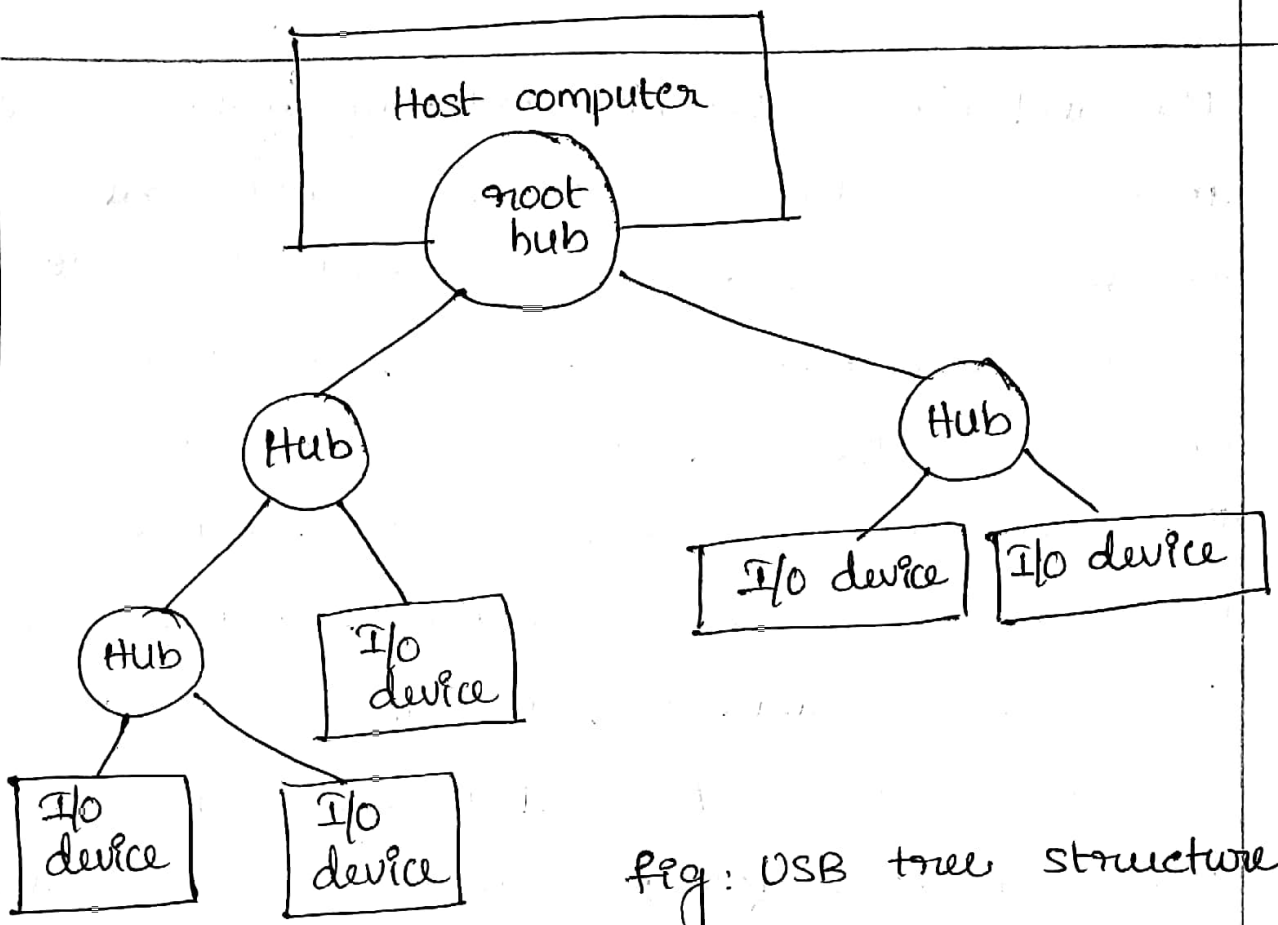


fig: USB tree structure

The tree structure enables many devices to be connected while using only simple point to point serial links. Each hub has a no. of ports where devices may be connected including other hubs. The USB operates strictly on the basis of polling. A device may send a message only in response to a poll message from the host.

The USB standard specifies the hardware details of USB interconnections as well as the organization and requirements of the host software. The purpose of the USB software is to provide bidirectional communication links between application software and

and I/O devices. These links are called pipes. Any data entering at one end of a pipe is delivered at the other end.

Addressing : When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree. Each device on the USB, whether it is a hub or an I/O device, is assigned a 7bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus. When a device is first connected to a hub, or when it is powered on, it has the address 0.

When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.

Locations in the device to or from which data transfer can take place, such as status, control, and data registers are called end points. They are identified by a 4 bit number.

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

A packet consists of one or more fields containing different kinds of information. The 1st field of any packet is called the packet identifier (PID) which identifies the type of that packet. 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.

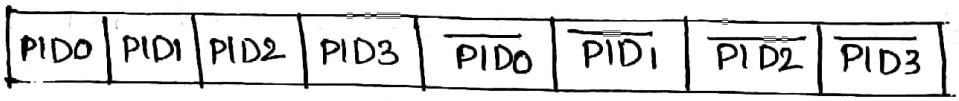


fig: Packet Identifier field

Control packets used for controlling data transfer operations are called token packets. A token packet starts with the PID field, using one of two PID values to distinguish between an IN packet and an OUT packet, which control input and output transfers respectively. The PID field followed by the 7 bit address of a device and the 4 bit end point 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 endpoint fields. By performing an inverse computation, the receiving device can determine whether the packet has been received correctly.

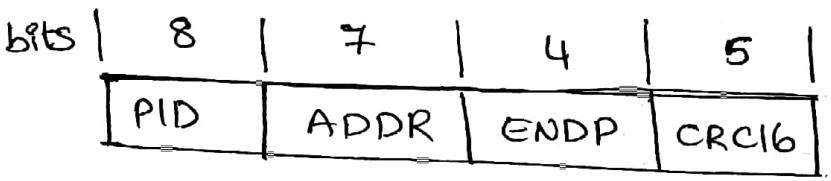


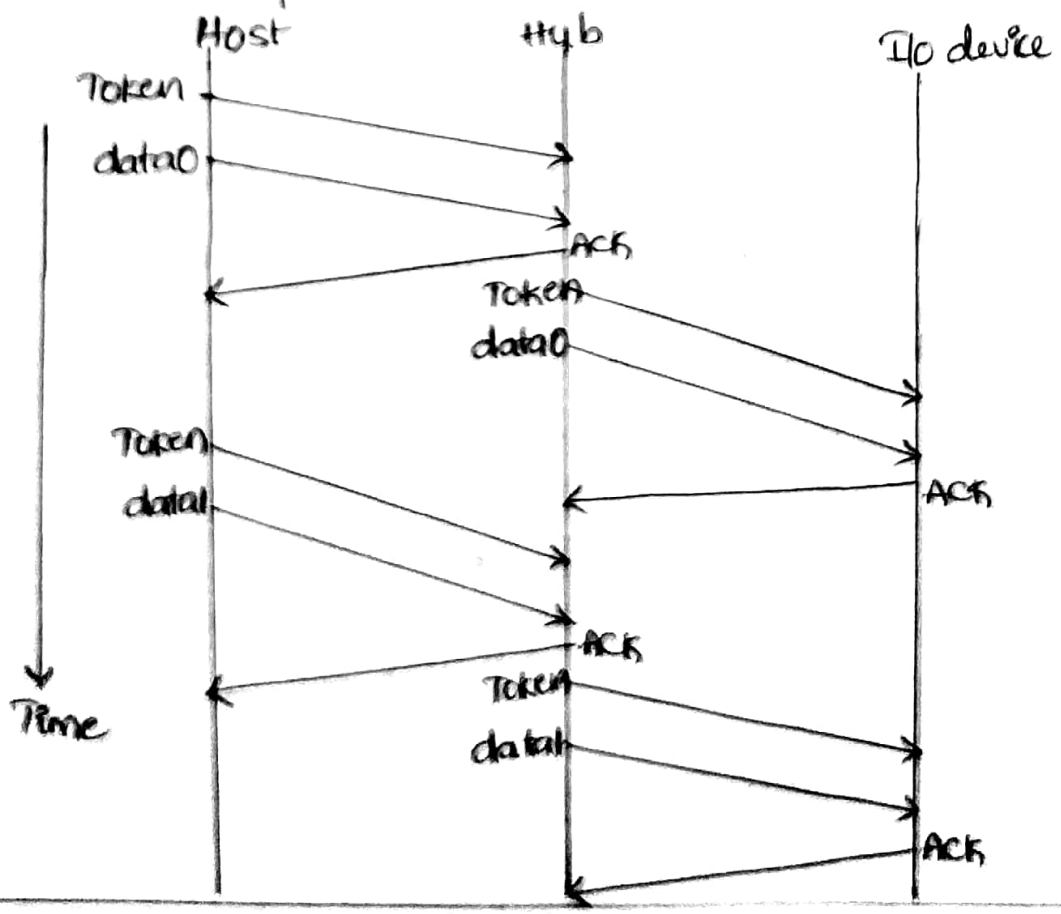
fig: Token packet, IN or OUT

Data packets, which carry input and output data, have the format shown below.

bits	8	0 to 8192	16
	PID	DATA	CRC 16

fig: Data Packet

The packet identifier field is followed by up to 8192 bits of data, then 16 error checking bits. Three different PID patterns are used to identify data packets, so that data packets may be numbered 0, 1 & 2. Data packets do not carry a device address & an end point number.



Isochronous Traffic on USB : One of the key objective of the USB is to support the transfer of isochronous data, such as sampled voice, in a simple manner. Devices that generate or receive isochronous data require a time reference to control the sampling process. To provide this reference, transmission over the USB is divided into frames of equal length. A frame is 1ms long for low and full speed data. The root hub generates a start of frame control packet (SOF) precisely once every 1ms to mark the begenning of a new frame.

The main requirement for isochronous traffic is consistent timing. An occasional error can be tolerated. Hence, there is no need to retransmit packets that are lost or to send acknowledgements.

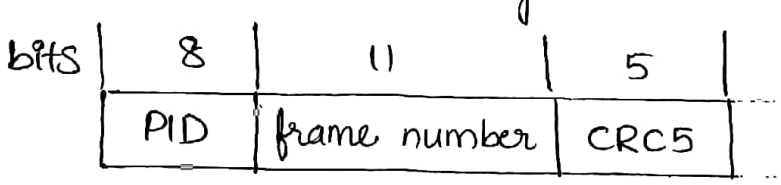
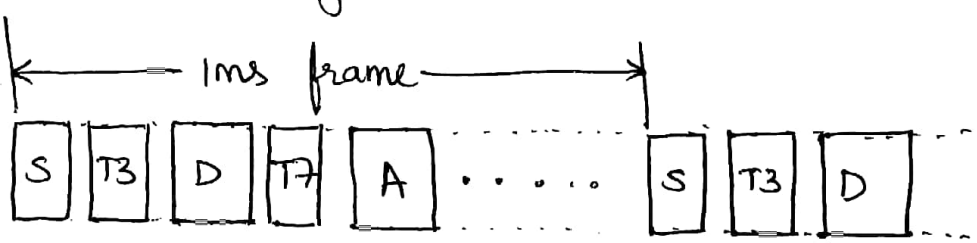


fig : SOF packet



S – Start of frame packet D – data packet
Tn – Token packet, address = n A – ACK packet

Isochronous data are allowed only on full speed and high speed links. The SOF packet is repeated eight times at equal intervals within the ms frame to create eight microframes of 125μs each.

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.

UNIT-V

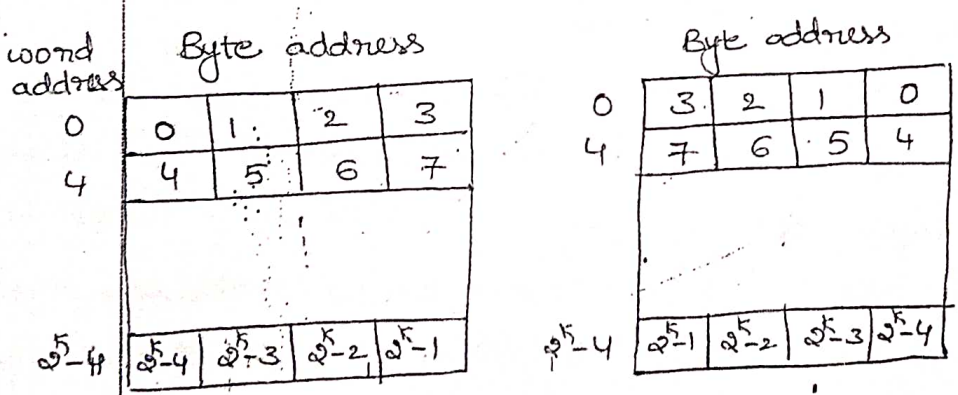
THE MEMORY SYSTEM

Ideally the memory would be fast, large, inexpensive. Unfortunately, it is impossible to meet all three of these requirements simultaneously.

Some basic concepts: The maximum size of the memory that can be used in any computer is determined by the addressing lines. Ex: 16-bit computer generates 16 bit addresses is capable of addressing upto 2^{16} = 64K memory locations.

No. of address lines	size of memory
10	1K (Kilo)
20	1M (Mega)
30	1G (Giga)
40	1T (Tera)

Most modern computers are byte addressable.



Big Endian assignment

Used in Motorola 68000 processor

little endian assignment

Used in Intel processor

ARM uses either arrangement

The memory is usually designed to store and retrieve data in word length quantities. The no of bits actually stored or retrieved in one memory access is the most common definition of the word length of a computer.

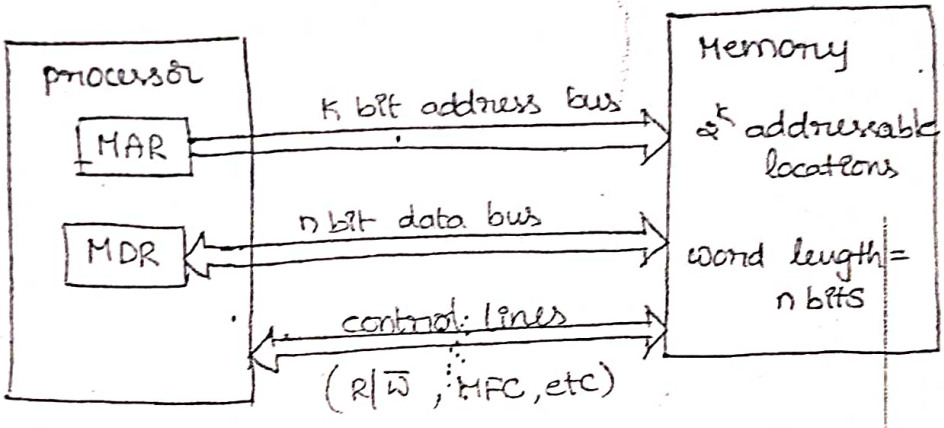


fig : Connection of the memory to the processor.

MFC - Memory Function Completed.
MAR - Memory address register
MDR - " Data "

If read or write operations involve consecutive address locations in the main memory, then a block transfer operation can be performed in which the only address sent to the memory is the one that identifies the first location. The time that elapses between the initiation of an operation and the completion of that operation is referred to as the memory access time

Semiconductor RAM memories: Semiconductor memories are available in a wide range of speeds. Their cycle times range from 100ns to less than 10ns. These are much more expensive than the magnetic core memories. Because of VLSI technology, the cost of semiconductor memories has dropped.

Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information.

Static Memories: Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories. Figure shows that how a static RAM (SRAM) cell may be implemented

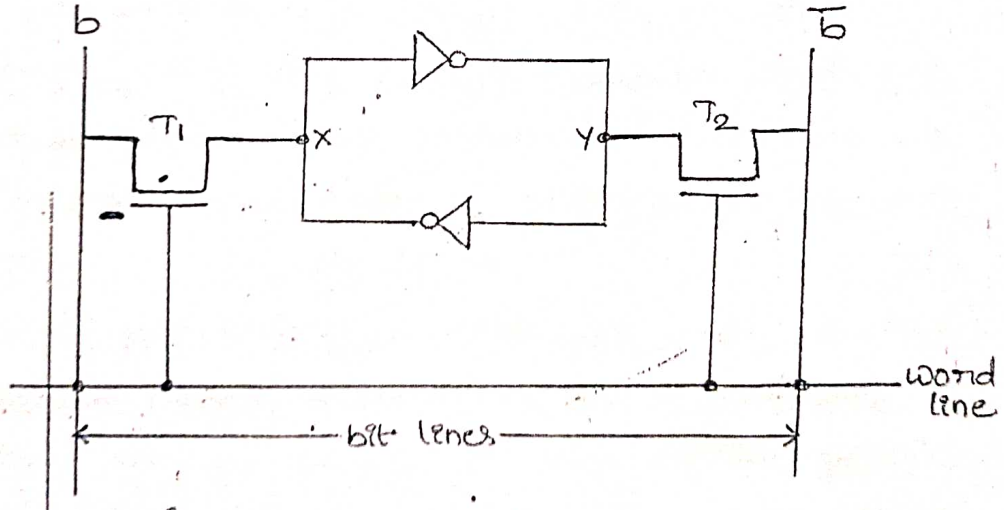


fig: A SRAM cell

(The time between the read and the MFC signals) The minimum time delay required between the initiation of two successive memory operations is called the memory cycle time. (the time between two successive read operations).

A memory unit is called RAM if any location can be accessed for a read or write operation in some fixed amount of time that is independent of the location's address. The basic technology for implementing the memory uses semiconductor integrated circuits. Cache memory is a small, fast memory that is inserted between the larger, slower main memory and the processor. This is used to reduce the memory access time.

Data may be stored in physical memory locations that have addresses different from those specified by the program. The memory control circuitry translates the address specified by the program into an address that can be used to access the physical memory. In such a case, an address generated by the processor is referred to as a virtual or logical address. The virtual address space is mapped onto the physical memory where data are actually stored. The mapping function is implemented by memory management unit (MMU).

Asynchronous DRAMS : static RAMs are fast, but they come at a high cost because their cells require if simpler cells are used. However, such cells do not retain their state indefinitely. hence they are called dynamic RAMs (DRAMs). Information is stored in a dynamic ram cell in the form of a charge on a capacitor, and this charge can be maintained for only 10s of milli seconds. Since the cell is required to store information for a much longer time, its contents must be periodically refreshed by restoring the capacitor charge to its full value.

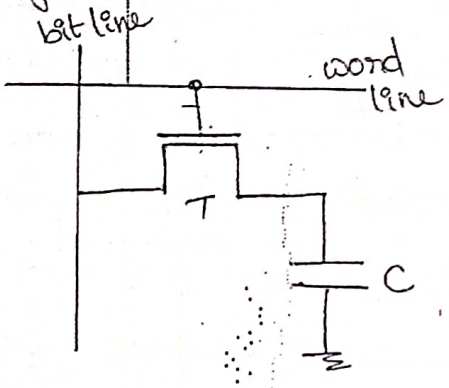


fig : DRAM cell

DRAM cell consist of a capacitor C and transistor T as shown above fig. Inorder to store information in this cell, transistor T is turned on and an appropriate voltage is applied on the bit line. This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, the capacitor begins to discharge. This is caused

Two inverters are cross connected to form a latch. The latch is connected to two bit lines by transistors T1 and T2. These transistors act as switches that can be opened & closed under control of the word line. When the word line is at ground level, the transistors are turned off and the latch retains its state.

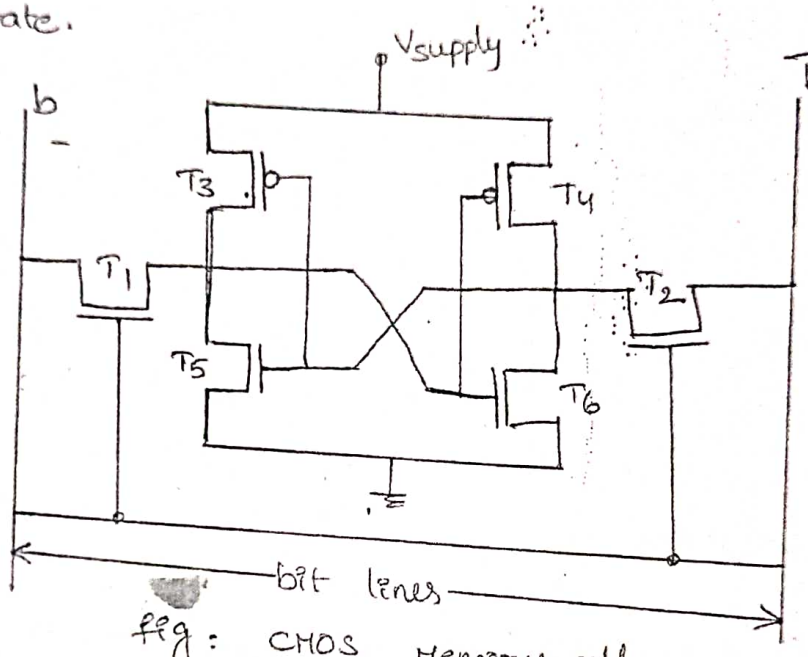


fig: CMOS Memory cell.

Transistor pairs (T_3, T_5), (T_4, T_6) form the inverters in the latch. The power supply voltage V_{supply} is 5V in older CMOS SRAMs & 3.3V in new low voltage versions. Continuous power is needed for the cell to retain its state. SRAMs are said to be volatile memories because their contents are lost when power is interrupted. SRAMs can be accessed very quickly.

by the capacitor's own leakage resistance and the fact that the transistor continues to conduct a tiny amount of current (RAs) (Pico Amp), after it is turned off. Hence the information stored in the cell can be retrieved correctly only if it is read before the charge on the capacitor drops below some threshold. In the DRAM cells, the timing of the memory device is controlled asynchronously. A specialized memory controller circuit provides the necessary control signals RAS (Row Address strobe) and CAS (column Address strobe), that govern the timing. The processor must take into account the delay in response of the memory. Such memories are referred to as asynchronous DRAMs.

The most useful arrangement is to transfer the bytes in sequential order, which is achieved by applying a consecutive sequence of column addresses under the control of successive CAS signals. This scheme allows transferring a block of data at a much faster rate than can be achieved for transfers involving random addresses. The block transfer capability is referred to as the fast page mode feature.

Synchronous DRAMs: If DRAMs operation is directly synchronized with a clock signal, then such memories are known as synchronous DRAMs (SDRAMs). The term memory latency is used to refer to the amount of time it takes to transfer a word of data to or from the memory. The standard SDRAM performs all actions on the rising edge of the clock signal. A similar memory device is available, which transfer data on both edges of the clock. The latency of these devices is the same as for standard SDRAMs, bandwidth is doubled for long burst transfers, such devices are known as double data rate SDRAMs (DDR SDRAMs).

Memory system considerations: The choice of a RAM chip for a given application depends on several factors, like cost, speed, power dissipation, size of the chip.

Memory controller: To reduce the no of pins, the dynamic memory chips use multiplexed address inputs. The address is divided into two parts. The high order address bits, which select a row in the cell array, the low order address bits, which select a column. A typical processor issues all bits of an address at the same time. The required multiplexing of address bits is usually performed by a memory controller unit

which is interposed between the processor and the dynamic memory, as shown in fig. The controller accepts

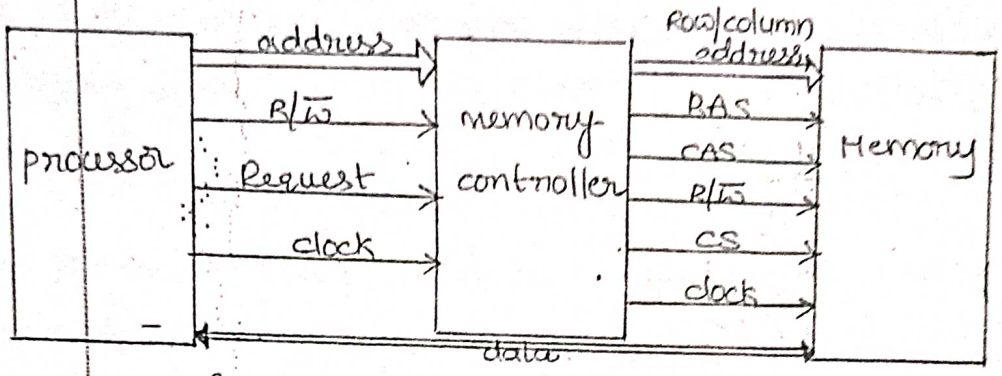


fig: Use of a memory controller.

a complete address and the R/W signal from the processor under control of a request signal which indicates that a memory access operation is needed. The controller then forwards the row and column portions of the address to the memory and generates the $\overline{RAS}$ and $\overline{CAS}$ signals. Thus the controller provides the RAS-CAS timing, in addition to its address multiplexing function. It also sends the R/W and CS signals to the memory. Data lines are connected directly between the processor and the memory. When used with DRAM chips, which do not have self refreshing capability, the memory controller has to provide all the information needed to control the refreshing process.

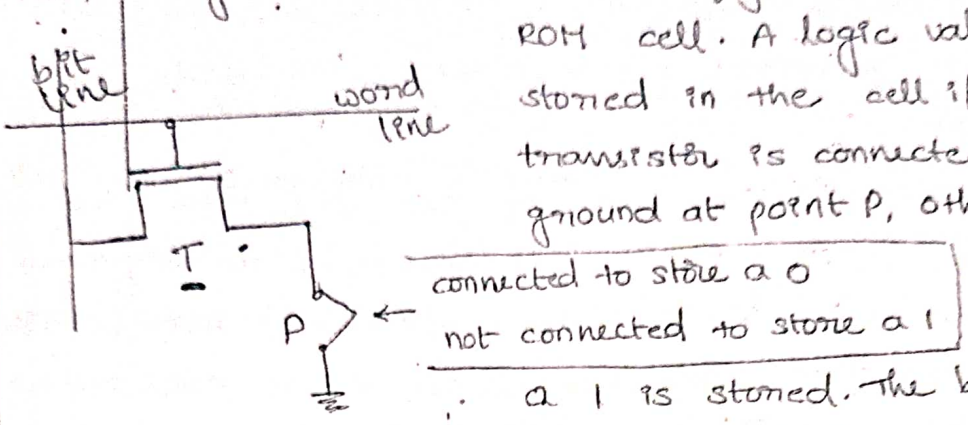
Refresh overhead: Consider an SDRAM whose cells are arranged in 8K(=8192) rows. Suppose that it takes four clock cycles to access each row. Then it takes $8192 \times 4 = 32,768$ cycles to refresh all rows. At a clock rate of 133MHz, the time needed to refresh all rows is $\frac{32,768}{133 \times 10^6} = 246 \times 10^{-6} s = 0.246 ms$

Thus refreshing process occupies 0.246ms in each 64ms time interval. ∴ refresh overhead $= \frac{0.246}{64} = 0.0038$. which is less than 0.4% of the total time available for accessing the memory.

Read only memories: Both SRAM and DRAM chips are volatile. Non volatile memory is used extensively in embedded systems. Such systems typically do not use disk storage devices — the programs are stored in nonvolatile semiconductor memory devices.

Different types of nonvolatile memory have been developed. Generally, the contents of such memory can be read as if they were SRAM or DRAM memories. But a special writing process is needed to place the information into this memory. Since its normal operation involves only reading of stored data, a memory of this type is called read only memory.

ROM : Fig shows a possible configuration for a ROM cell. A logic value 0 is stored in the cell if the transistor is connected to ground at point P, otherwise a 1 is stored. The bit-line is connected through a resistor to the power supply. To read the state of the cell, the word line is activated. Thus the transistor switch is closed and the voltage on the bit line drops to near zero if there is a connection between the transistor and ground. If there is no connection to ground, the bit line remains at the high voltage, indicating a 1. A sense circuit at the end of the bit line generates the proper output value. Data are written into a ROM when it is manufactured.



PROM : Some ROM designs allow the data to be loaded by the user, thus providing a programmable ROM. Programmability is achieved by inserting a fuse at point P as shown in above fig. Before it is programmed the memory contains all 0's. The user can insert 1s at the required locations by burning out the fuses

at these locations using high current pulses. This process is irreversible. PROMs provide flexibility and convenience not available with ROMs with increased cost.

EPROM : Another type of ROM chip allows the stored data to be erased and new data to be loaded. Such an erasable, reprogrammable ROM is usually called an EPROM. An EPROM has a structure similar to the ROM cell. In an EPROM cell, however, the connection to ground is always made at point P and a special transistor is used, which has the ability to function either as a normal transistor or as a disabled transistor that is always turned off. This transistor can be programmed to behave as a permanently open switch, by injecting charge into it that becomes trapped inside. ~~Thus, an EPROM cell~~ The important advantage of EPROM chips is that their contents can be erased and reprogrammed. Erasure requires dissipating the charges trapped in the transistors of memory cells, this can be done by exposing the chip to ultraviolet light. For this reason, EPROM chips are mounted in packages that have transparent windows.

EEPROM : A significant disadvantage of EPROMs is that a chip must be physically removed from the circuit for reprogramming and that its entire contents are erased by the ultraviolet light. Another version of erasable PROMs that can be both programmed and erased electrically. Such chips called EEPROMS, do not have to be removed for erasure. Moreover, it is possible to erase the cell contents selectively. The only disadvantage of EEPROMS is that different voltages are needed for erasing, writing and reading the stored data.

Flash memory: A flash cell is based on a single transistor controlled by trapped charge, just like an EEPROM cell. In EEPROM it is possible to read and write the contents of a single cell. In a flash device it is possible to read the contents of a single cell, but it is only possible to write an entire block of cells. Prior to writing the previous contents of the block are erased. Flash devices have greater density, which leads to higher capacity. They require single power supply voltage, and consume less power in their operation.

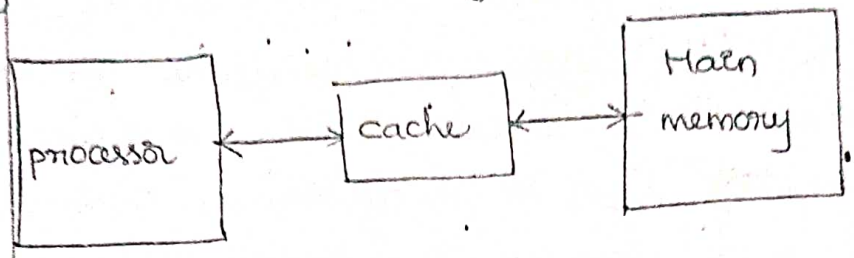
Typical applications include hand held computers, cell phones, digital cameras, and MP3 music players. In handheld computers and cell phones, flash memory holds the software needed to operate the equipment. In digital cameras, flash memory is used to store picture image data. In MP3 players are good examples flash memory stores the data that represent sound.

Single flash chips do not provide sufficient storage capacity for the applications mentioned above. Larger memory modules consisting of a no of chips are need. such modules are flash cards, flash drives.

Cache memories: The speed of the main memory is very low in comparison with the speed of modern processors. Since the speed of the main memory unit is limited by electronic and packaging constraints. An efficient solution is to use a fast cache memory which essentially makes the main memory appear to the processor to be faster that it really is.

The effectiveness of the cache mechanism is based on a property of computer programs called locality of reference. also known as the principle of locality. is the tendency of a processor to access the same set of memory locations respectively over

a short period of time. Consider the simple arrangement as shown in fig.



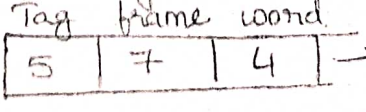
When a read request is received from the processor the contents of a block of memory words containing the location specified are transferred into the cache one word at a time. Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache. The cache memory can store a reasonable no of blocks at any given time, but this number is small compared to the total no. of blocks in the main memory. The correspondence between the main memory blocks and those in the cache is specified by a mapping function. When the cache is full and a memory word is not in the cache is referenced the cache control ~~block that contains the referenced~~ hardware must decide which block should be removed to create space for the new block that contains the

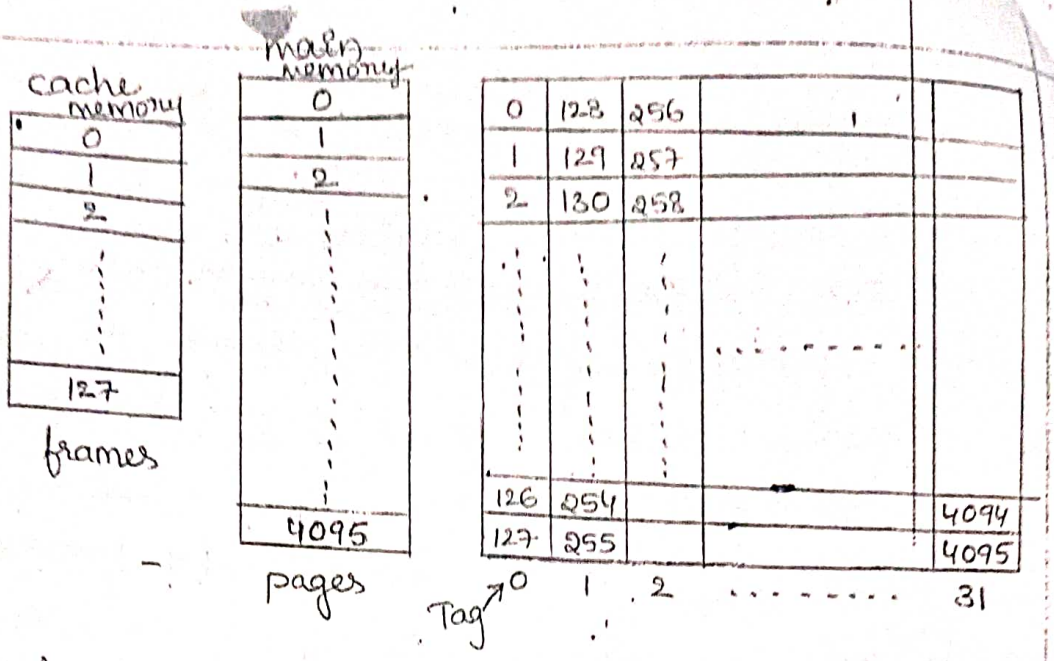
referenced word. The collection of rules for making this decision constitutes the replacement algorithm.

The processor does not need to know explicitly about the existence of the cache. The cache control circuitry determines whether the requested word currently exists in the cache. If it does, the read or write operation is performed on the appropriate cache location. In this case, a read or write hit is said to have occured. In a read operation, the main memory is not involved. For a write operation, the system can proceed in two ways. In the first technique, called the write through protocol, the cache location and the main memory location are updated simultaneously. The second technique is to update only the cache location and to mark it as updated with an associated flag bit, often called the dirty or modified bit. The main memory location of the word is updated later, when the block containing this marked word is to be removed from the cache to make room for a new block. This technique is known as the write back or copy back protocol.

Mapping functions: Consider a cache consisting of 128 frames of 16 words each, a total of 2048 ($128 \times 16 = 2^{11} = 2K$) words and assume main memory is addressable by a 16 bit address. The main memory has 4096 pages, each page having 16 words ($4096 \times 16 = 2^{16} = 64K$).

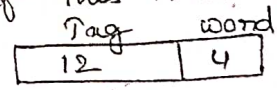
Direct mapping: The simplest way to determine cache locations in which to store memory blocks is the direct mapping technique. In this technique, page j of main memory maps onto ~~block~~ frame j modulo 128 of the cache. Thus whenever one of the main memory pages 0, 128, 256.... is loaded in the cache, it is stored in cache frame 0. Pages 1, 129,--- are stored in cache frame 1, and so on. Since more than one memory page is mapped onto a given cache frame position, contention may arise even for that position even when the cache is not full. Contention is resolved by allowing the new block page to overwrite the currently resident page. Placement of a page in the cache is determined from the memory address. The memory address can be divided into three fields.





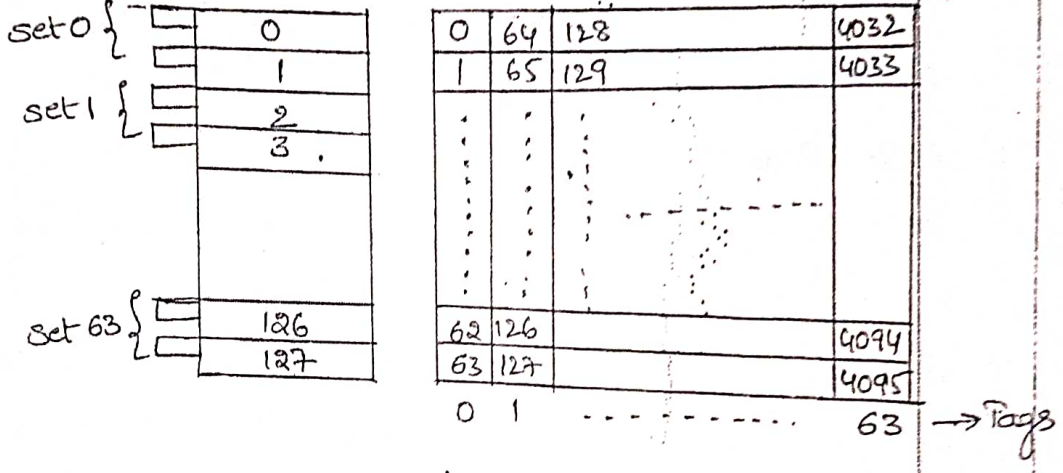
The high order 5 bits of the memory address of the block are stored in 5 tag bits associated with its location in the cache. They identify which of the 32 blocks that are mapped into this cache position are currently resident in the cache. The next 7bit cache block field determines the cache position in which this page must be stored. The high order 5 bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache. It is easy to implement, but it is not very flexible.

Associative mapping : In this technique a main memory page can be placed into any cache frame position. In this case, 12 tag bits are required to identify a memory page when it is resident in the cache. The tag bits of an address received from the processor are compared to tag bits of each frame of the cache to see if the desired page is present. This is called associative mapping technique A new page that has to be brought into the cache has to replace an existing page only if the cache is full. In this case, we need an algorithm to select the block to be replaced. Many replacement algorithms are possible to do this. The cost of an associative cache is higher than the cost of a direct mapped cache because of the need to search all 128 tag patterns to determine whether a given page is in the cache. A search of this kind is called an associative search.



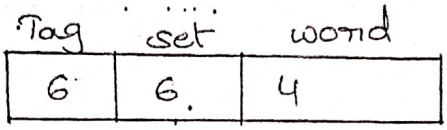
Set associative mapping : A combination of both direct and associative mapping techniques can be used Frames of cache are grouped into sets, and the mapping allows a page of main memory to reside

in any frame of a specific set. Hence, the contention problem of the direct method is eased by having a few choices for page replacement. At the same time, the hardware cost is reduced by decreasing the size of the associative search. An example of this set associative mapping is shown below, for a cache with two blocks per set.



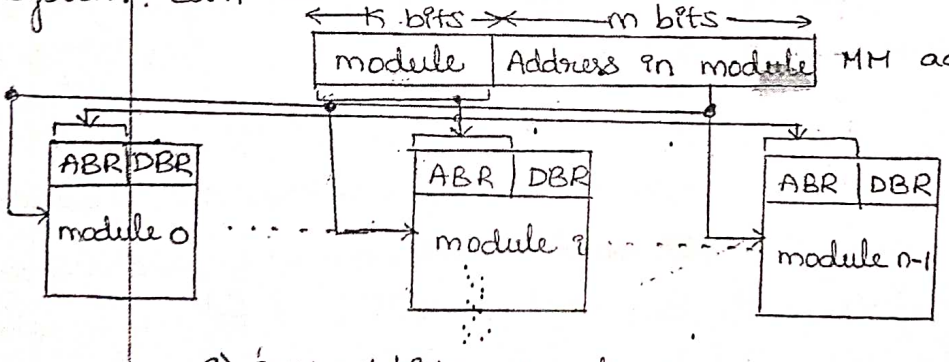
In this case memory pages 0, 64, 128, ... 4032 map into cache set 0, and they can occupy either of the two block positions within this set. Having 64 sets means that the 6 bit set field of the address determines which set of the cache might contain the desired page. The tags field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired page is present.

The no of frames per set is a parameter that can be selected to suit the requirement of a particular computer. For the main memory four frames per set can be accommodated by a 5 bit set field, & frames per set by a 4 bit set field and so on.

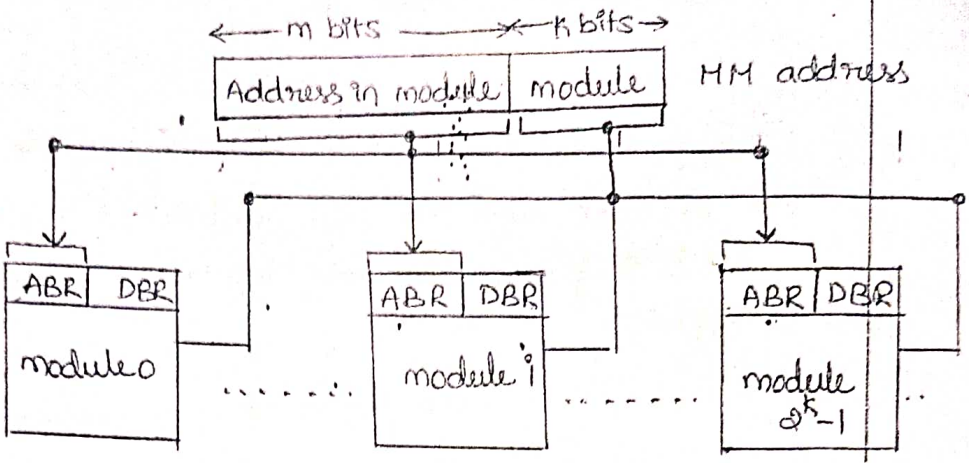


main memory address

Interleaving: If the main memory of a computer is structured as a collection of physically separate modules, each with its own address buffer register (ABR) and data buffer register (DBR), memory access operations may proceed in more than one module at the same time. Thus, the aggregate rate of transmission of words to and from the main memory system can be increased.



a) Consecutive words in a module.



b) consecutive words in consecutive modules

Two a methods of address layout are indicated in above figures. In the 1st case, the memory address generated by the processor is decoded as shown in fig(a). The high order k bits name one of n modules, and the low order m bits name a particular word in that module. When consecutive locations are accessed, as happens when a block of data is transferred to a cache, only one module is involved.

The second way to address the modules is shown in fig(b). It is called memory interleaving. The low order k bits of the memory address select a module, and the high order m bits name a location within that module. In this way consecutive addresses are located in successive modules. Thus any component of the system that generates requests for access to consecutive memory locations can keep several modules

busy at any one time. This results in both faster access to a block of data and higher average utilization of the memory system as a whole. Interleaving is used within a memory module to increase the speed of accessing successive words of data.

Magnetic hard disks: The storage medium in a magnetic disk system consists of one or more disks mounted on a common spindle. A thin magnetic film is deposited on each disk, usually on both sides. The disks are placed on a rotary drive so that the magnetized surfaces move in close proximity to read/write heads as shown in fig. The disks rotate at a uniform speed. Each head consists of magnetic yoke and a magnetizing coil.

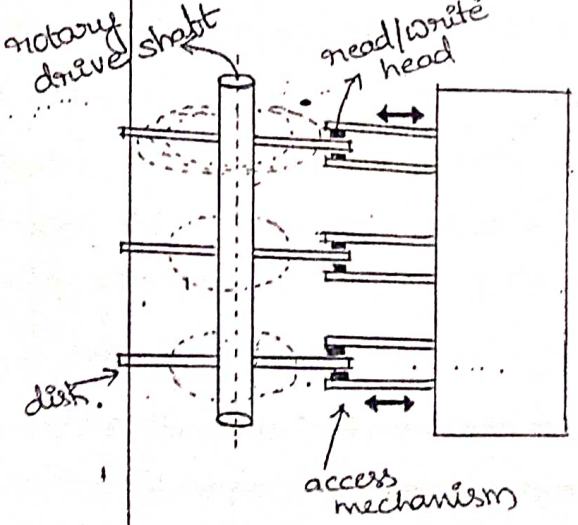


fig: Mechanical structure

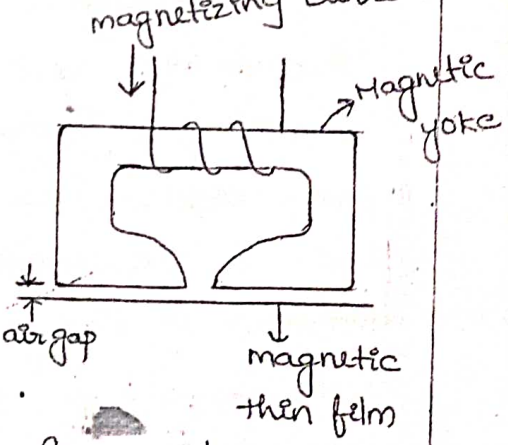


fig: read/write head detail

Digital information can be stored on the magnetic film by applying current pulses of suitable polarity to the magnetizing coil. This causes the magnetization of the film in the area immediately underneath the head to switch to a direction parallel to the applied field. The same head can be used for reading the stored information. In this case changes in the magnetic field in the vicinity of the head caused by the movement of the film relative to the yoke induce a voltage in the coil, which now serves as a sense coil. The polarity of this voltage is monitored by the control circuitry to determine the state of magnetization of the film. Only changes in the magnetic field under the head can be sensed during the read operation. ∴ If the binary states 0 and 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0 to 1 and at 1 to 0 transitions in the bit stream. A long string of 0s or 1s causes an induced voltage only at the beginning and end of the string. To determine no. of 0s and 1's stored, a clock must provide information for synchronization.

In early designs, a clock was stored on a separate track, where a change in magnetization is forced for each bit period. The modern approach is to combine the clocking information with the data. Several different techniques have been developed for such encoding. One simple scheme depicted in fig below is known as phase encoding or Manchester encoding.

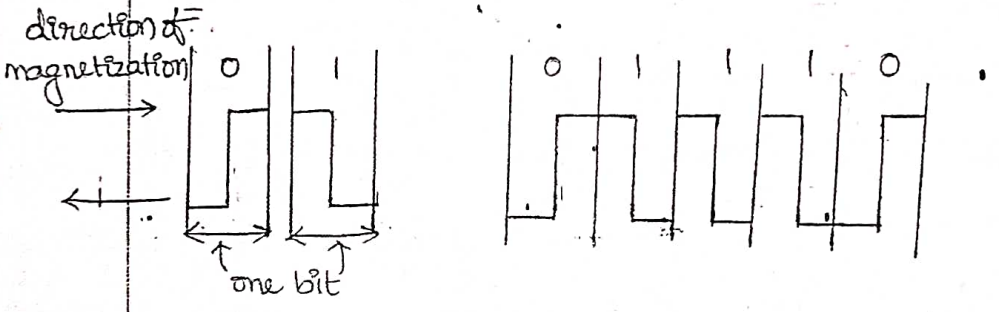


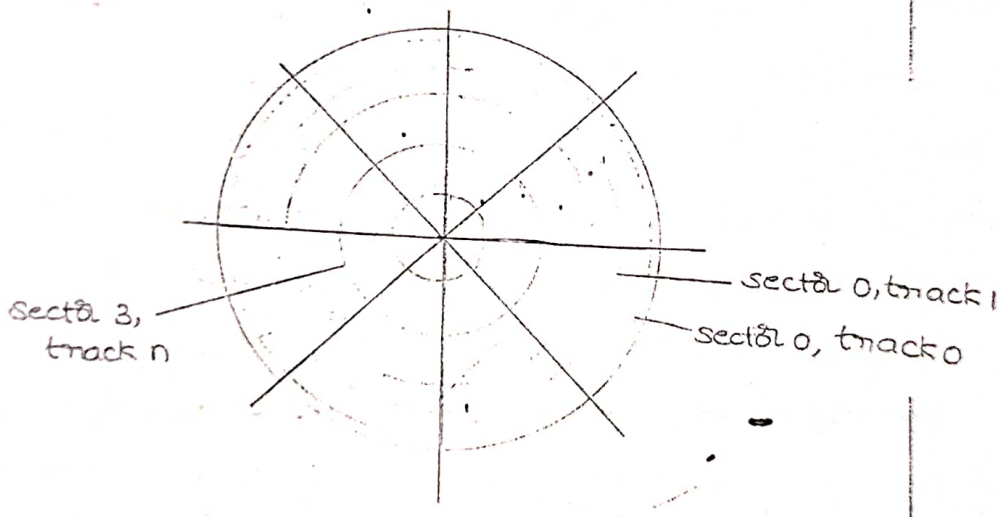
fig : bit representation by phase encoding

In this scheme, changes in magnetization occur for each data bit as shown above. A change in magnetization is guaranteed at the mid point of each bit period, thus providing the clocking information. We use the Manchester encoding example to illustrate how a self clocking scheme may be implemented.

Read / write heads must be maintained at a very small distance from the moving disk surfaces in order to achieve high densities and reliable read / write operations. When the disks are moving at their steady

mate, air pressure develops between the disk surface and the head and forces the head away from the surface. In most modern disk units, the disks and the read/write heads are placed in a sealed, air filtered enclosure. This approach is known as Winchester technology.

The read/write heads of a disk system are movable. There is one head per surface. All heads are mounted on a comb like arm that can move readily across the stack of disks to provide access to individual tracks as shown in below fig.



The disk system consists of three key parts. One part is the assembly of disk platters, which is usually referred to as the disk. The second part comprises the electromechanical mechanism that spins the disk and moves the read/write heads. It is

called the disk drive. The third part is the electronic circuitry that controls the operation of the system, which is called the disk controller.

Organization and accessing of data on a disk:-

The organization of data on a disk is illustrated in above fig. Each surface is divided into concentric tracks and each track is divided into sectors. The set of corresponding tracks on all surfaces of a stack of disks form a logical cylinder. The data on all tracks of a cylinder can be accessed without moving the read/write heads. The data are accessed by specifying the surface number, the track number, and the sector number.

Data bits are stored serially on each track. Each sector usually contains 512 bytes of data, but other sizes may be used. The data are preceded by a sector header that contains identification (addressing) information used to find the desired sector on the selected track. Following the data, there are additional bits that constitute an error correcting code (ECC). The ECC bits are used to detect and correct errors that may have occured in writing or reading of the 512 data bytes. To distinguish between

two consecutive sectors, there is a small intersector gap. Each track has the same no. of sectors. So all tracks have the same storage capacity. Thus, the stored information is packed more densely on inner tracks than on outer tracks. This arrangement is used in many disks because it simplifies the electronic circuits needed to access the data. It is possible to increase storage density on outer tracks with the increase in circuitry.

Access time: There are two components involved in the time delay between receiving an address and the beginning of the actual data transfer. The first, called the seek time, is the time required to move the read/write head to the proper track. This depends on the initial position of the head relative to the track specified in the address. Average values are in the 5 to 8 ms range. The 2nd component is the rotational delay also called latency time. This is the amount of time that elapses after the head is positioned over the correct track until the starting position of the addressed sector passes under the read/write head. The sum of these two delays is called the disk access time.

Typical disks: A 3.5 inch diameter high capacity high data rate, disk have

* 20 data recording surfaces with 15000 tracks/surface
* average of 400 sectors per track & each sector contains 512 bytes of data. Total capacity is ($20 \times 15,000 \times 400 \times 512 = 60 \times 10^9 = 60$ GB)
* average seek time is 6ms. The platters rotate at 10,000 revolutions/min. So average latency is 3ms.
* Internal transfer rate, from a track to the data buffer in the disk controller is 34 Mbps.

Data buffer/cache: A disk drive is connected to the rest of a computer system using some standard interconnection scheme. Normally, a standard bus, such as the SCSI bus is used. A disk drive that incorporates the required SCSI interface circuitry is usually referred to as a iSCSI drive. The SCSI bus is capable of transferring data at much higher rate than the rate at which data can be read from disk tracks. An efficient way to deal with the possible differences in transfer rates between the disk and the SCSI bus is to include a data buffer in the disk unit. This buffer is a semiconductor

memory, capable of storing a few mega bytes of data. The data buffer can also be used to provide a caching mechanism for the disk. When a read request arrives at the disk, the controller can first check to see if the desired data are already available in the cache(buffer). If so, the data can be accessed and placed on the SCSI bus in microseconds rather than milli seconds. Otherwise, the data are read from a disk track in the usual way and stored in the cache.

Disk controller : Operation of a disk drive is controlled by a disk controller circuit, which also provides an interface between the disk drive and the bus that connects it to the rest of the computer system. Fig shows a disk controller which controls two disk drives.

A disk controller that is connected directly to the processor system bus, or to an expansion bus such as PCI, contains a no. of registers that can be read and written by the operating system. The communication between the OS & the disk controller is achieved in the same manner as with any I/O interface, as discussed in chapter 4.

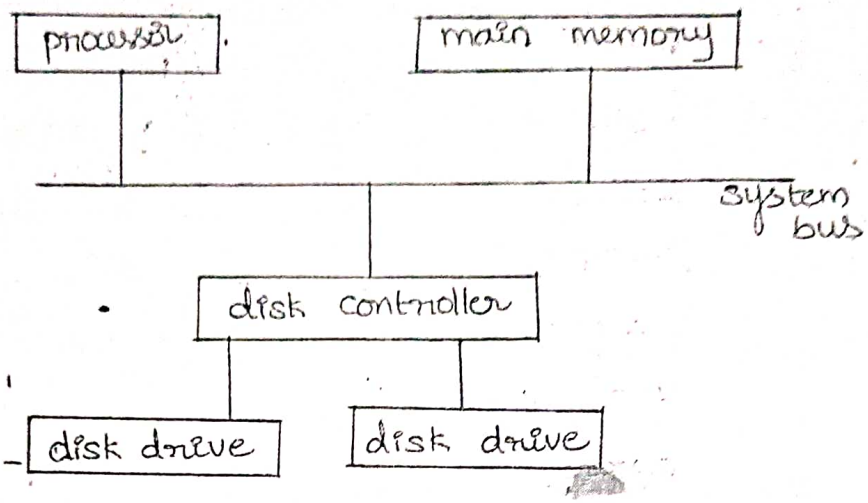


fig: Disks connected to the system bus.

The disk controller uses the DMA scheme to transfer data between the disk and the main memory. Actually, these transfers are from/to the data buffer, which is implemented as a part of the disk controller module. ~~The~~ OS initiates the transfers by issuing read and write requests; which entail loading the controller's registers with the necessary addressing and control information, typically:

Main memory address — the address of the first main memory location of the block of words involved in the transfer.

Disk address — The location of the sector containing the beginning of the desired block of words.

Word count — the no. of words in the block to be transferred.

On the disk drive side, the controllers major functions are:

Seek — causes the disk drive to move the read/write head from its current position to the desired track.

Read — Initiates a read operation, starting at the address specified in the disk address register. Data read serially from the disk are assembled into words and placed into the data buffer for transfer to the main memory. The no. of words is determined by the word count register.

Write — Transfers data to the disk, using a control method similar to that for the read operations.

Error checking — Computes the error correcting code value for the data read from a given sector and compares it with the corresponding ECC value read from the disk. In case of a mismatch, it corrects the error if possible, otherwise it raises an interrupt to inform the OS that an error has occured. During a write operation, the controller computes the ECC value for the data to be written and stores this value on the disk.

Software and operating system implications:

All data transfer activities involving disks are initiated by the operating system. The disk is a non volatile storage medium, so the OS itself is stored on a disk. During normal operation of a computer, parts of the OS are loaded into the main memory and executed as needed.

When power is turned off, the contents of main memory are lost. When the power is turned on again, the OS has to be loaded into the main memory which takes place as part of a process known as booting. To initiate booting, a tiny part of main memory is implemented as a non volatile ROM. The ROM stores a small monitor program that can read and write main memory locations as well as read one block of data stored on the disk at address 0. This block referred to as the boot block, contains a loader program. After boot block is loaded into memory by the ROM monitor program, it loads the main parts of the OS into the main memory.

In a computer system that has multiple disks, the OS may require transfers from several disks. Efficient operation is achieved if the DMA transfer from/to one disk occurs while another is doing a seek.

Floppy disks : The devices previously discussed are known as hard or rigid disk units. Floppy disks are smaller, simpler and cheaper disk units that consist of a flexible, removable, plastic diskette coated with magnetic material.

One of the simplest schemes used in the first floppy disks for recording data is phase or Manchester encoding. Disks encoded in this way are said to have single density. A more complicated variant of this scheme, called double density, is most often used in current standard floppy disks. It increases the storage density by a factor of 2 but also requires more complex circuits in the disk controller.

The main feature of floppy disks is their low cost and shipping convenience. However, they have much smaller storage capacities, longer access times and higher failure rates than hard disks. Current floppy disks are 3.25 inches in diameter and store 1.44 or 2 Mbytes of data. Larger, super-floppy disks are also available & one type is zip disk, can store more than 100 Mbytes.

RAID disk arrays: Processor speeds have increased dramatically during the past decade. Processor performance has doubled every 18 months. Semiconductor memory speeds have improved more modestly. High performance devices tend to be expensive. Sometimes it is possible to achieve very high performance at a reasonable cost by using a no. of low cost devices operating in parallel. Multiple magnetic disk drives can be used to provide a high performance storage unit.

In 1988, researchers at the University of Berkeley proposed a storage system based on multiple disks. They called it RAID, for Redundant Array of Inexpensive Disks. Six different configurations were proposed. They are known as RAID levels even though there is no hierarchy involved.

RAID 0 is the basic configuration intended to enhance performance. A single large file is stored in several separate disk units by breaking the file up into a no. of smaller pieces and storing these pieces on different disks. This is called data striping. When the file is accessed for a read, all disks can deliver their data in parallel. The total transfer time of the file is equal to the transfer time that would be

required in a single disk system divided by the no. of disks used in the array. However, access time is not reduced. Since each disk operates independently of the others, access times vary. This is the simplest possible disk array operation in which only data flow time performance is improved.

RAID 1 is intended to provide better reliability by storing identical copies of data on two disks rather than just one. The two disks are said to be mirrors of each other. Then, if one disk drive fails, all read and write operations are directed to its mirror drive. This is a costly way to improve the reliability because all disks are duplicated.

RAID 2, RAID 3, RAID 4 levels achieve increased reliability through various parity checking schemes without requiring a full duplication of disks. All of the parity information is kept on one disk.

RAID 5 also makes use of a parity based error recovery scheme. However, the parity information is distributed among all disks.

Some hybrid arrangements have been developed. ex RAID 10 is an array that combines the features of RAID 0 and RAID 1.

Commodity disk considerations:

ATA/EIDE disks: (Enhanced Integrated drive electronics) (Advanced Technology Attachment). The present version of a disk interface suitable for connection to the IBM PC bus was become a standard known as EIDE or as ATA. Many disk manufacturers have a range of disks that have EIDE/ATA interfaces. Such disks can be connected directly to the PCI bus which is used in many PCs.

SCSI disks: Many disks have an interface designed for connection to a standard SCSI bus. These disks are more expensive, but they exhibit better performance compared with a PCI bus. Concurrent accesses can be made to multiple disk drives because the drive's interface is actively connected to the SCSI bus only when the drive is ready for a data transfer.

RAID disks: These offer excellent performance and provide a large and reliable storage. They are used in high performance computers & high reliability is required.

Optical disks: Large storage devices can also be implemented using optical means. The familiar (CD) compact disk used in audio systems, was the 1st practical application of this. The 1st generation of CDs was developed in mid 1980s by Sony and Philips companies. This technology is used for digital representation of analog sound signals to provide high quality sound recording and reproduction. 16 bit samples of the analog signal are taken at a rate of 44,100 samples per second.

CD technology: The optical technology that is used for CD systems is based on a laser light source. The laser emits a coherent light beam that is sharply, focussed on the surface of the disk.

A cross section of a small portion of a CD is shown in fig below.

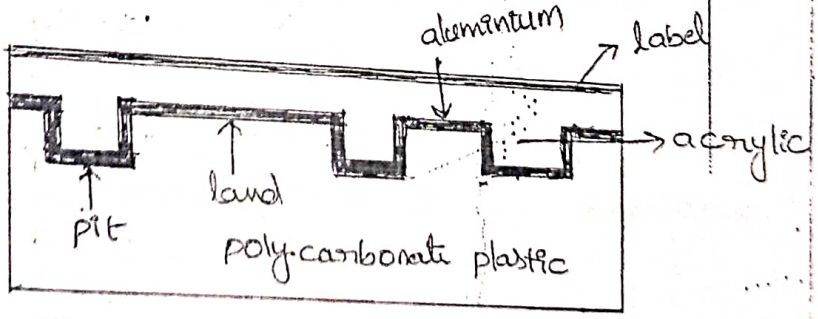
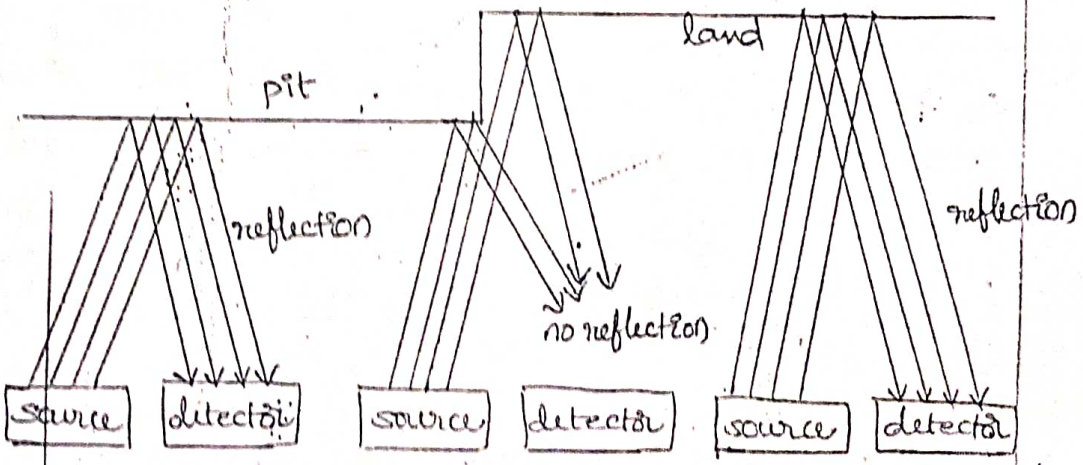


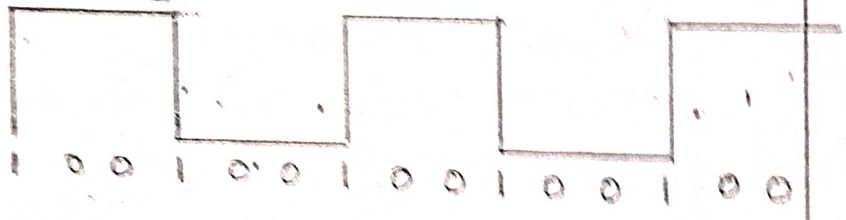
fig: cross section

The bottom layer is polycarbonate plastic, which functions as a clear glass base. The surface of this plastic is programmed to store data by indenting it with pits. The unindented parts are called lands. A thin layer of reflecting aluminium material is placed on top of a programmed disk. The aluminium is then covered by a protective acrylic. Finally, the topmast layer is deposited and stamped with a label. The total thickness of the disk is 1.2mm.

A laser source and the photo detector are positioned below the poly carbonate plastic. The emitted beam travels through this plastic, reflects off the aluminium layer, and travels back toward the photo detector. Fig below shows what happens as the laser beam scans across the disk.



Three different positions of the laser source and the detector are shown, as would occur when the disk is rotating. When the light reflects from the pit or from the land, the detector will see the reflected beam as a bright spot. When the beam moves through the edge the pit is recessed one quarter of the wavelength of the light. Thus, the reflected wave from the pit will be 180° out of phase with the wave reflected from the land, cancelling each other. Hence at transitions the detector will not see a reflected beam and will detect a dark spot.



Above fig depicts several transitions between lands and pits. If each transition, detected as a dark spot, is taken to denote the binary value 1, and the flat portions represent 0s. This pattern is not a direct representation of the stored data. CDs use a complex encoding scheme to represent data, which provides considerable error detection capability.

The CD is 120mm in diameter, 15mm hole in the center. Data are stored on tracks that cover the area from 25mm radius to 58mm radius. The space

between the tracks is 1.6 microns. There are more than 15,000 tracks on a disk. This indicate a track density of about 6000 tracks/cm. The density ranges from 800 to 2000 tracks/cm in hard disks and 240 tracks/cm in floppy disks.

CD-ROM: CDs use additional bits to provide error checking and correcting capability. CDs used in computer applications have such capability. They are called CD-ROMs, because after manufacture their contents can only be read, as with semiconductor ROM chips.

Stored data are organized on CD-ROM tracks in the form of blocks that are called sector. The no of sectors per track is variable, there are more sectors on the longer outer tracks.

CD-Recordables: It was developed in the late 1990s on which data can be easily recorded by a computer user. It is known as CD-Recordable (CD-R). A spiral track is implemented on a disk during the manufacturing process. A laser in a CD-R drive is used to burn pits into an organic dye on the track. When a burned spot is heated beyond a critical temperature, it becomes opaque. Such burned spots reflect less light when subsequently read. The written data are stored permanently. Unused portions of a disk can be used to store additional data at a later time.

CD - Rewritables :- The most flexible CDs are those that can be written multiple times by the user. They are known as CD-RWs. The basic structure of CD-RWs is similar to the structure of CD-Rs, Instead of using an organic dye in the recording layer, an alloy of silver, indium, antimony and tellurium is used. If it is heated above its melting point (560°c) and then cooled down, it goes into an amorphous state in which it absorbs light. But, if it is heated only to about 200°c and this temperature is maintained for an extended period, a process known as annealing takes place, which leaves the alloy in a crystalline state that allows light to pass through. If the crystalline state represents a land area, pits can be created by heating selected spots past the melting point. The stored data can be erased using the annealing process, which returns the alloy to a uniform crystalline state. A reflective material is placed above the recording layer to reflect the light when the disk is read.

The CD-RW drive uses three different laser powers. The highest power is used to record the pits. The middle power is used to put the alloy into its crystalline state, it is referred to as the

erase power. The lowest power is used to read the stored information. Presently 1000 times the CD-RW disk can be rewritten.

DVD technology: The physical size of a DVD disk is much larger than that of CDs by several design changes:

→ A red_light laser with a wavelength of 635nm is used instead of the infrared light laser used in CDs, which has wavelength of 780nm. The shorter wavelength makes it possible to focus the light to a smaller spot.

→ Pits are smaller, having a min length of 0.4 microns

→ Tracks are placed closer together, the distance b/w tracks is 0.74 microns.

Using these improvements leads to a DVD capacity of 4.7 Giga bytes. Further increases in capacity have been achieved by going to two layered and two sided disks. The single layered single sided disk is almost the same as the CD. A double layered disk makes use of two layers on which tracks are implemented on top of each other. The 1st layer is the clear base, as in CD disks. But instead of using reflecting aluminium, the lands and pits of this layer are

covered by a translucent material that acts as a semi reflector. The surface of this material is then also programmed with indented pits to store data. A reflective material is placed on the top of the 2nd layer of pits and lands. The disk is read by focusing the laser beam on the desired layer. When the beam is focused on the 1st layer, sufficient light is reflected by the translucent material to detect the stored binary patterns. When the beam is focused on the 2nd layer, the light reflected by the reflective material corresponds to the information stored on this layer. In both cases, the layer on which the beam is not focused reflects a much smaller amount of light, which is eliminated by the detector circuit as noise. The total storage capacity of both layers is 8.5GB.

DVD-RAM : A rewritable version of DVD devices, known as DVD-RAM. It provides a large storage capacity & only disadvantages are the higher price, relatively slow writing speed. To ensure that the data have been recorded correctly on the disk, a process known as write verification is performed, this is done by DVD-RAM drive, which reads the stored contents & checks them against original data.

UNIT-6: Processing Unit & Micro-programmed control

JNTUK R16 2-2 CSE

Syllabus:

→ Processing Unit
  → Fundamental Concepts
    → Register Transfers ✓
    → Performing an Arithmetic (or) Logic Operation
    → Fetching a word from memory
  → Execution of a Complete Instruction ✓
  → Hardwired Control ✓

→ Micro-programmed Control
  → Micro Instructions
  → Microprogram Sequencing ✓
  → Wide-branch Addressing
  → MicroInstructions with Next-Address Field ✓

① Fundamental concepts:

→ To execute a program, the processor fetches one instruction at a time and performs the operations specified.

→ Instructions are fetched from successive memory locations until a branch (or) Jump instruction is encountered.

→ The processor keeps track of the address of the memory locations containing the next instruction to be fetched using the program counter, PC.

→ Another key register in the processor is the Instruction Register, IR.

→ Suppose that each instruction comprises 4 bytes, and that it is stored in one memory word.

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

* Fetch the contents of the memory location pointed to by the PC. They are loaded into the IR.

$$IR \leftarrow [[PC]]$$

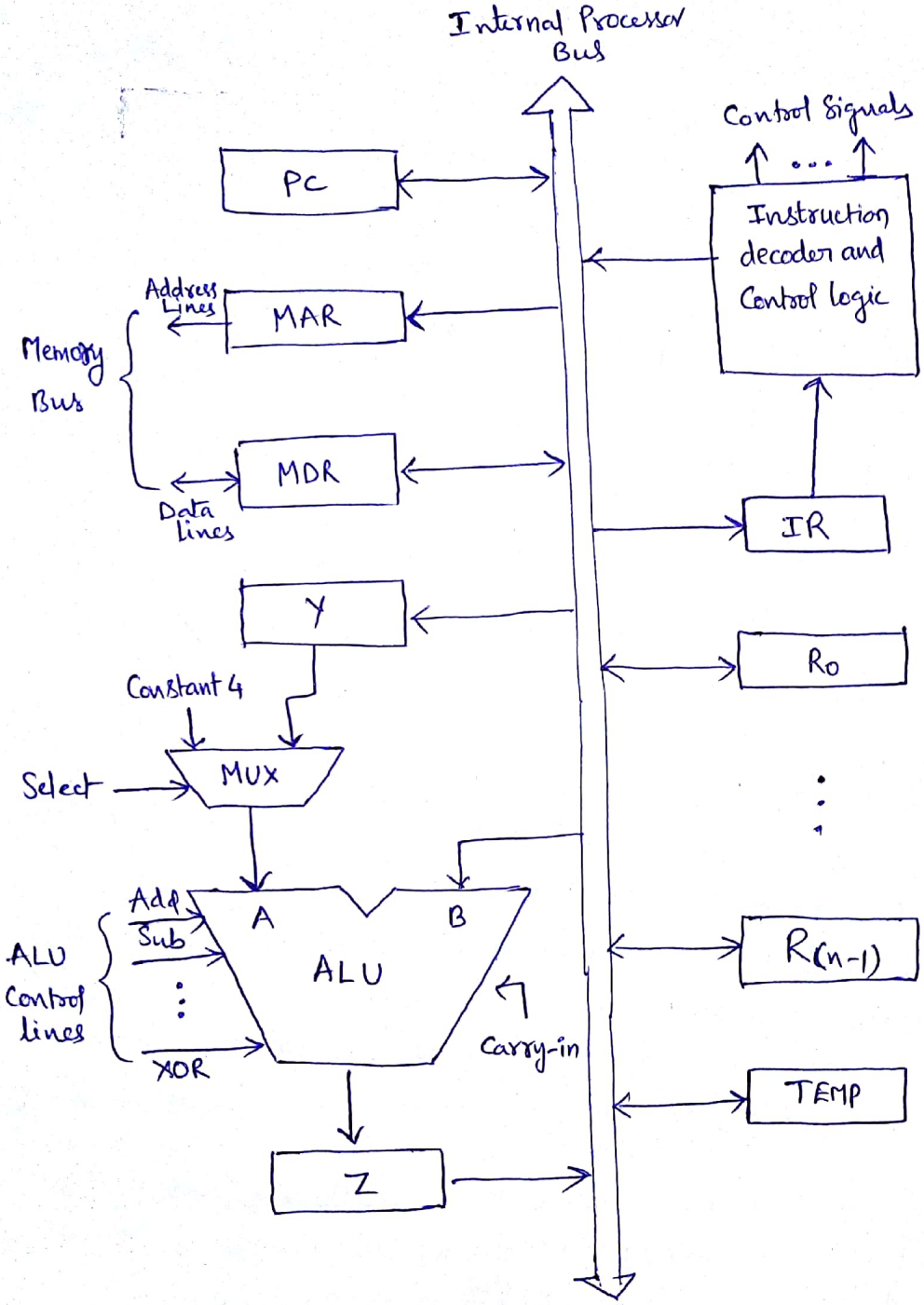
* Assuming that the memory is byte addressable, increment the contents of the PC by 4

$$PC \leftarrow [PC] + 4$$

* Carry out the actions specified by the instruction in the IR.

→ Here, first two steps represents fetch phase, Third step represents execution phase.

→ Single-bus organization of the datapath inside a processor is depicted as,



→ Data may be loaded into MDR either from the memory bus (or) from the internal processor bus.

→ The input of MAR is connected to the internal bus and its output is connected to the external bus.

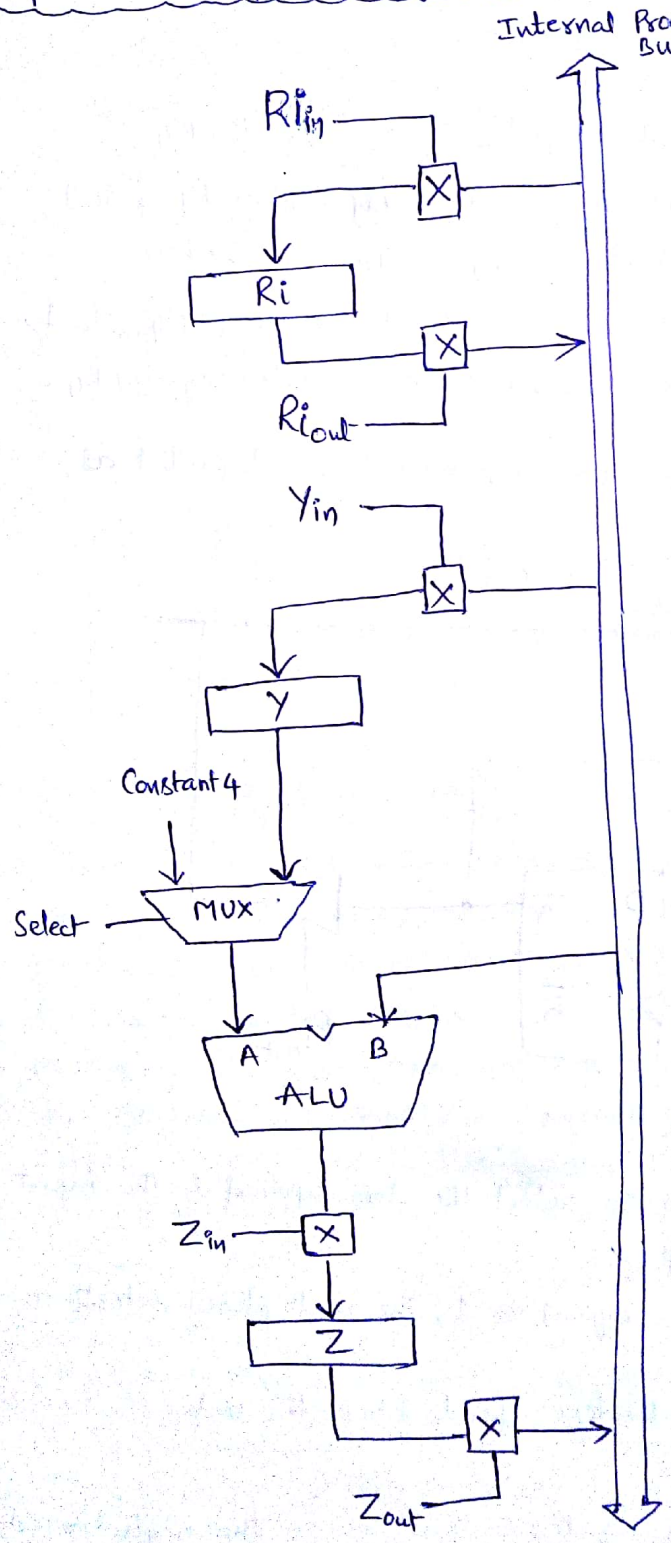
→ The multiplexer MUX selects either the output of register Y (or) a constant value 4 to be provided as input A of the ALU.

(a) Register Transfer :

→ 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) load the data on the bus into the register.

→ Input and output gating for the registers is depicted as,



→ The input and output of register R_i are connected to the bus via switches controlled by the signals $R_{i_{in}}$ and $R_{i_{out}}$, respectively.

→ When $R_{i_{in}}$ is set to 1, the data on the bus are loaded into R_i

→ Similarly, when $R_{i_{out}}$ is set to 1, the contents of Register R_i are placed on the bus.

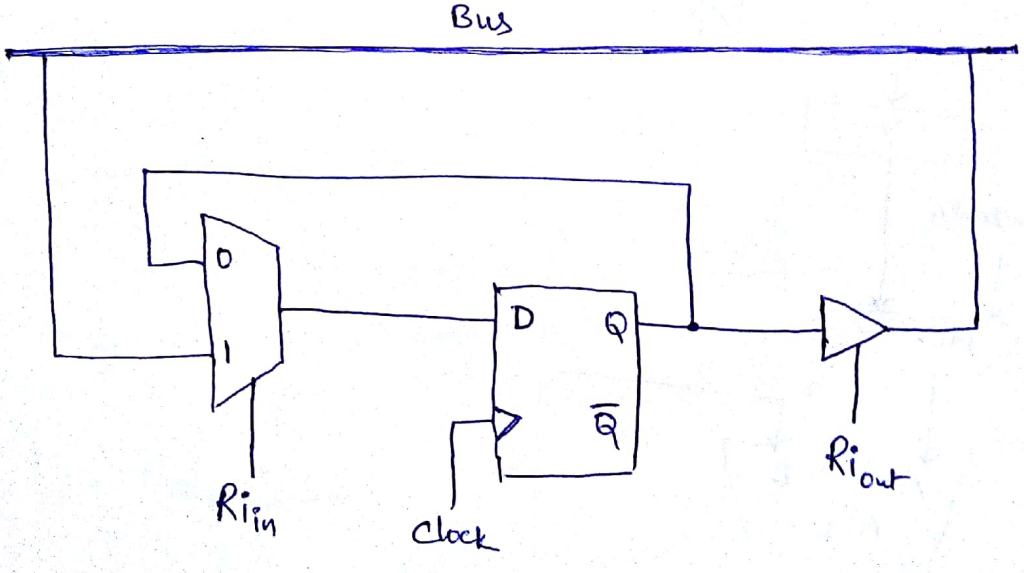
→ While $R_{i_{out}}$ is equal to 0, the bus can be used for transferring data from other registers

Example:

→ To transfer the contents of register R_1 to register R_4,

- Enable the output of register R_1 by setting $R_{1_{out}}$ to 1, This places the contents of R_1 on the processor bus.
- Enable the input of register R_4 by setting $R_{4_{in}}$ to 1, This loads data from the processor bus into register R_4.

→ Input and Output gating for one register bit is depicted as,



→ A two-input multiplexer is used to select the data applied to the input of an edge-triggered D-flipflop

→ When the control input $R_{i_{in}}$ is equal to 1, the multiplexer selects the data on the bus.

→ When $R_{i_{in}}$ equal to 0, the multiplexer feeds back the value currently stored in the flip-flop.

→ When $R_{i_{out}}$ is equal to 0, the gate's output is in the high-impedance (electrically disconnected) state.

→ When $Ri_{out} = 1$, the gate drives the bus to 0 (or) 1, depending on the value of Q.

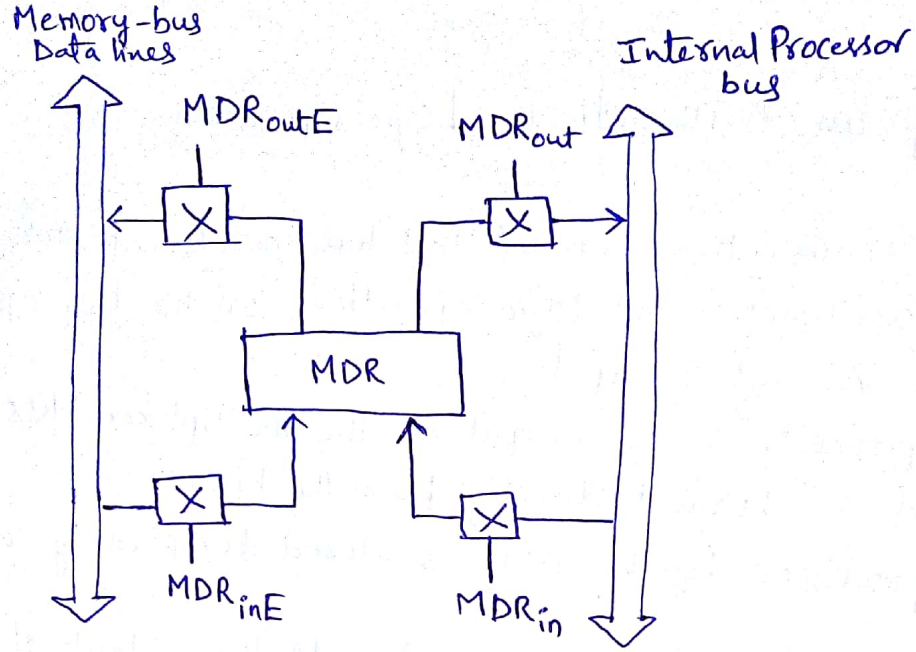
(b) Performing an Arithmetic (or) Logic Operation:

→ 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
→ One of the operands is the output of the multiplexer MUX and the other operand is obtained directly from the bus.
→ The result produced by the ALU is stored temporarily in register Z.
→ Therefore, a sequence of operations to add the contents of register R1 to those of register R2 and store the result in register R3 is,

1. $R1_{out}$, Y_{in}
2. $R2_{out}$, Select Y, Add, Z_{in}
3. Z_{out}, $R3_{in}$

(c) Fetching a word from memory:

→ To fetch a word of information from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation.
→ This applies whether the information to be fetched represents an instruction in a program (or) an operand specified by an instruction.
→ The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus.
→ When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.
→ The connection and control signals for register MDR is depicted as,



→ It has 4 control signals

- MDR$_{in}$ and MDR$_{out}$ control the connection to the internal bus.
- MDR$_{inE}$ and MDR$_{outE}$ control the connection to the external bus

→ As an example of a 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]

(2) Execution of a Complete Instruction:

→ 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.

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

→ The Control sequence for execution of the instruction Add (R3), R1 is given as

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

→ In Step1, 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.

→ From Step 1 through 3 represents instruction fetch 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 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 sum is stored in register Z, then transferred to R1 in Step 7.

→ From step 4 through 7 represents execution phase.

(i) BRANCH Instructions:

→ A branch instruction replaces the contents of the PC with the branch target Address.

→ This address is usually obtained by adding an offset X, which is given in the branch instruction, to the updated value of the PC

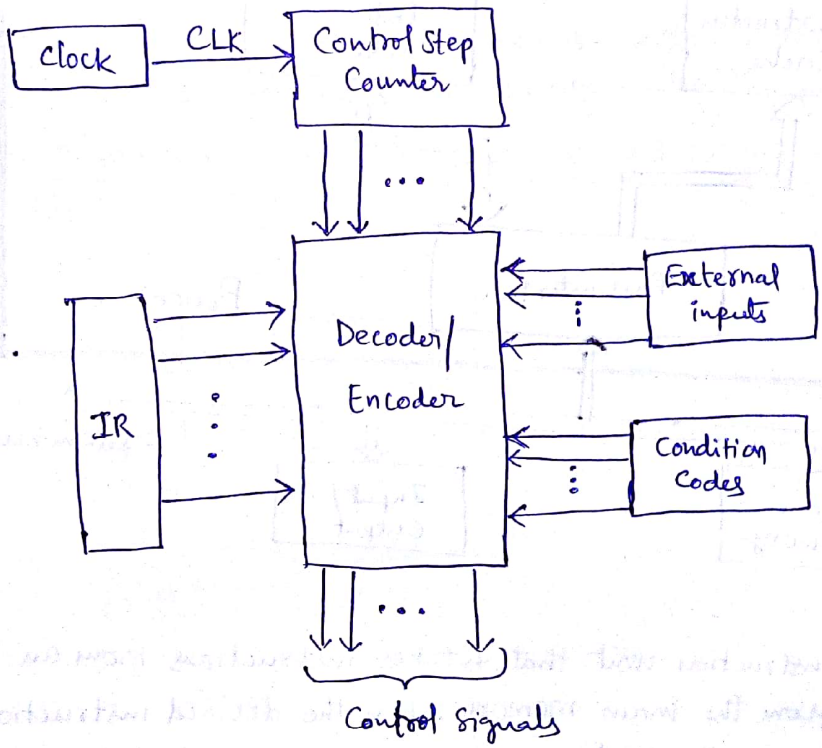
→ Control Sequence for an unconditional Branch instruction is given as

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	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

③ Hardwired Control :

→ To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
→ This approach has two Categories
  * Hardwired Control
  * Micro-programmed control

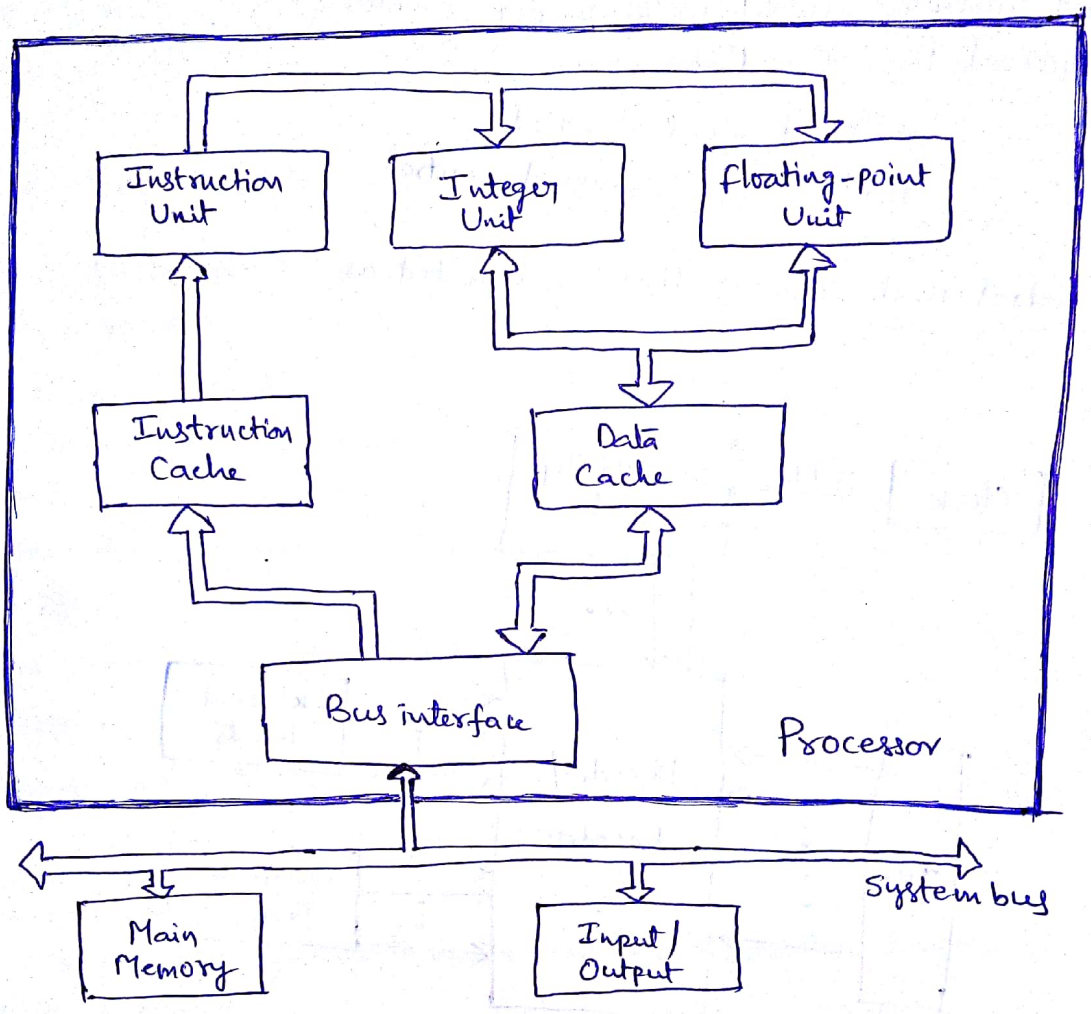
→ The control unit organization is depicted as



→ Consider the sequence of control signals.
→ Each step in this sequence is completed in one clock period.
→ A counter may be used to keep track of the control steps.
→ Each State, (or) count, of this counter corresponds to one control step.
→ 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.

(i) A Complete Processor :

→ The Block diagram of a Complete Processor is depicted as



→ It has an instruction unit that fetches instructions from an Instruction Cache (or) from the main memory when the desired instructions are not already in the Cache.

→ It has seperate processing units to deal with integer data and floating-point data.

→ Using seperate caches for instruction and data is common practice in many processors today.

→ The processor is connected to the System bus and, hence, to the rest of the Computer, by means of a bus interface.

④ Microprogrammed Control:

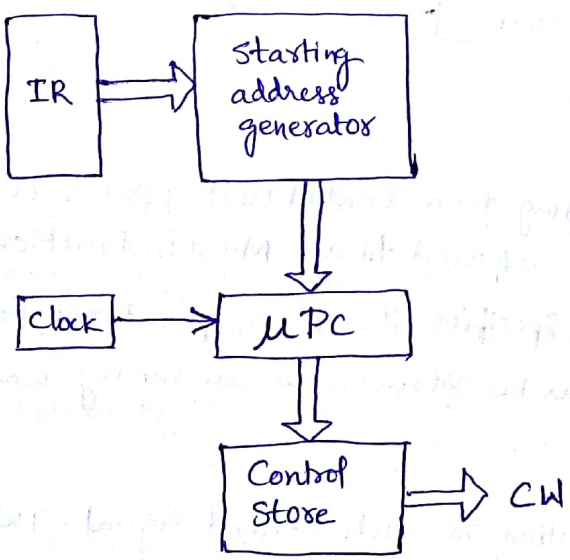
→ The control signals are generated by a program similar to machine language programs represents a scheme called Microprogrammed Control

→ A control word (CW) is a word whose individual bits represent the various control signals

→ A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the microroutine for that instruction, and the individual control words in this microroutine are referred to as microinstructions.

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

→ The Basic organization of a microprogrammed control unit is depicted as,



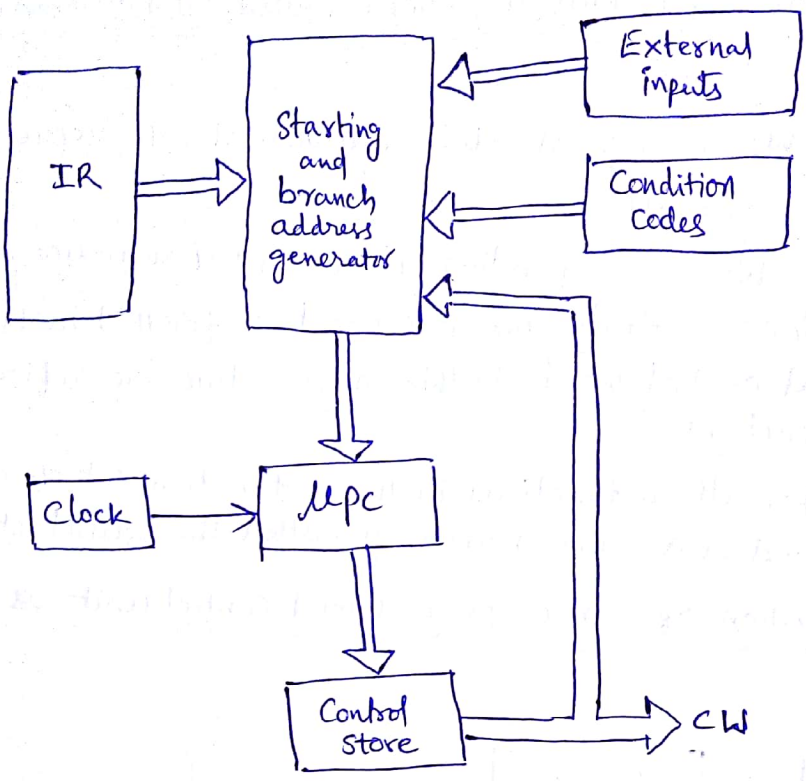
→ To read the control words sequentially from the control store, a micro-program counter (μPC) is used.

→ Everytime 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.

→ Hence, the control signals are delivered to various parts of the processor in the correct sequence.

→ Organization of the control unit to allow conditional branching in the microprogram is depicted as



(i) Microinstructions :

→ The control word belonging to a control unit possess certain instructions, usually referred to as Microinstructions.

→ Each Microinstruction specifies the Microoperations for the System

→ A micro'instruction can be structured in many ways. Some of them are,

- Assign a bit position to each control signal. This scheme is not good enough, as it results in long instructions. And also because of only few bits are set to 1, it does not use the bit space properly
- Encode the microinstructions using bits, with assumptions.
- Group the mutually exclusive control signals into fields.
- Enumerate the patterns of required signals in all microinstructions.

→ There are two types of microinstruction formats.

(a) Horizontal Microinstruction Format
(b) Vertical Microinstruction Format

a) Horizontal Microinstruction format:

Internal CPU Control Signals	System bus Control Signals	Jump Condition (Indirect bit, Zero overflow, Unconditional)	Microinstruction address

→ Horizontal Microinstruction format supports
- Long formats
- express (or) resort to high degree of parallelism
- Low degree of encoding of control information

b) Vertical Microinstruction format:

Function codes	Function Codes	Jump Condition	Microinstruction Address

→ Vertical Microinstruction Format supports
- Less degree of parallelism in case of microoperations.
- Subsequently high encoding in case of control information.
- Relatively Short Format

(ii) Microprogram Sequencing

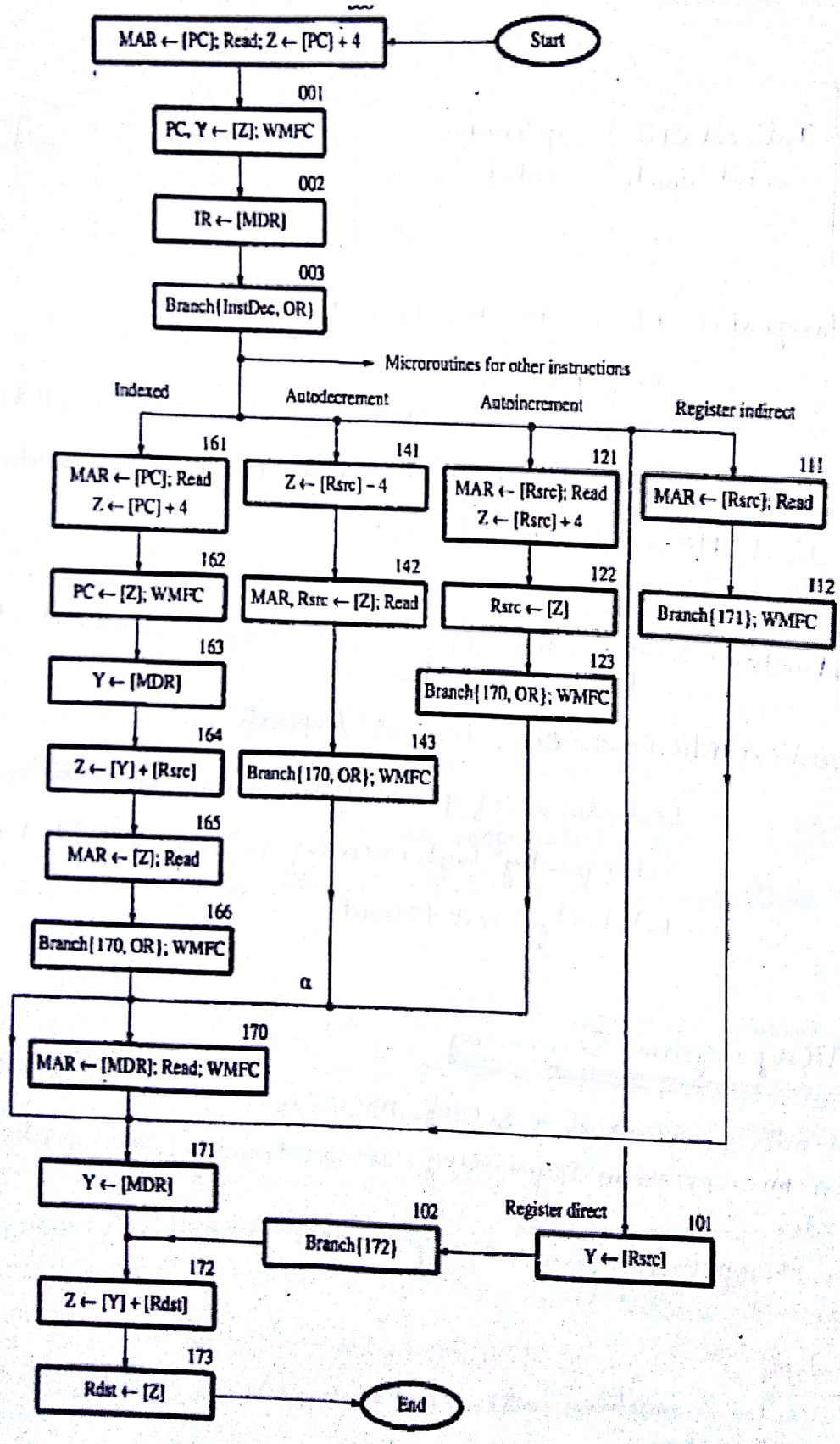
→ A microprogram is a set of microinstructions
→ In microprogram sequencing, microinstructions are executed in the sequential order.
→ The Microprogram Sequencer generates the order of executing microinstructions from the Control Store.

Example:
→ Consider a machine instruction that adds R_s and R_d and stores the result in R_d

Add R_s, R_d

→ Where, R_s is the source operand register and R_d is the destination operand register

→ Flowchart of a microprogram for the Add Rsrc, Rdst instruction is depicted as,



(iii) Wide-Branch Addressing:

→ Generating branch addresses means that the circuitry becomes more Complex

→ Example, the machine instruction fetch is completed, and an appropriate microroutine should be selected according to addressing modes

→ Here, the OPcode of a machine instruction is translated into a starting address.

→ It is possible to issue a wait for MFC (WMFC) Command in a branch micro-instruction.

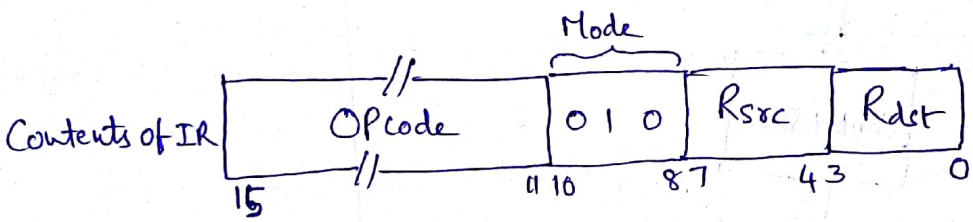
→ The WMFC signal means that the microinstruction may take several clock cycles to complete.

Example:

→ For the instruction

Add (Rsrc)+, Rdst

→ The Format of IR is depicted as



(iv) Microinstructions with Next-Address Field:

→ The purpose of branch microinstructions is to find the address of the next microinstruction to be fetched.

→ That means, they do not perform any useful operation in the data path.

→ This effects the operating speed of the System.

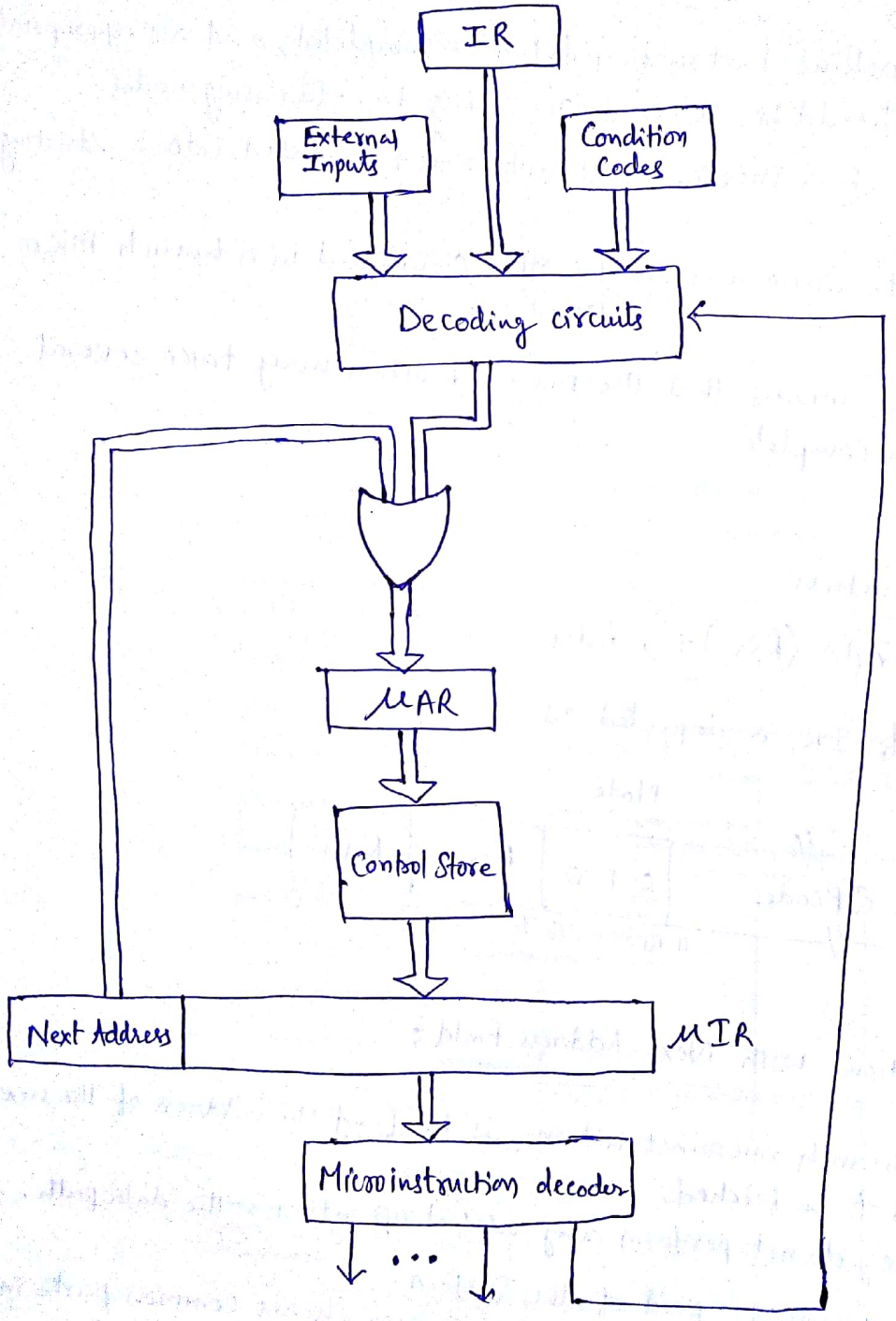
→ The situation becomes worse when the processor shares common parts in microinstructions using branch microinstruction in order to execute sequential instructions.

→ Thus, execution of one simple instruction needs several branch instructions, which badly affects the System's Operating System (OS).

→ One way to overcome the above problem is to modify the sequencing technique based on incrementable µpc.

→ An alternative is to include a special address field called "Next-address field" in each microinstruction in order to specify the address of the next microinstruction

→ The Microinstruction-Sequencing Organization is depicted as,



→ As a result, each microinstruction generates the effect of a branch microinstruction and also performs its intended function.

→ So, there is no need for a seperate μpc to store the address of next instruction.

CO - UNIT 6 - Short Answer Questions

(1) What is register transfer? Discuss..

page 6.3

(2) Briefly discuss unconditional branch instructions.

page 6.8

(3) Explain Hardwired Control

page 6.9

(4) Explain Microprogrammed Control unit

page 6.11

(5) Define a) Micro-operation b) Micro-program c) Micro-instruction

a) Microoperation:

→ A Microoperation refers to a Simple (or) a Complex operation that produces certain output, which is stored in a memory location (or) register.

b) Microprogram:

→ Microprogram refers to a sequence of microinstructions.
→ It is usually stored in control memory.
→ A Microprogram is executed by a micro programmed Control unit.

c) Micro Instruction:

→ The control word belonging to a control unit possess certain instructions, usually referred to as microinstructions.
→ Each Microinstruction Specifies microoperations for the system.

(6) Write Micro instruction formats.

(or)

Write about a) Horizontal Microinstruction Format b) vertical Microinstruction Format

page 6.13

(7) Differentiate between Hardwired Control and Microprogrammed Control.

Microprogrammed control Unit	Hardwired control unit
1. It has slower execution speed	1. It has faster execution speed
2. Its control functions are implemented in software	2. Its control functions are implemented in hardware
3. It can easily accommodate changes such as new system specifications (or) new instructions redesign	3. It is not flexible towards any changes
4. Its design process is systematic	4. Its design process is complicated
5. It usually supports more than 100 instructions.	5. It supports less than 100 instructions
6. It can easily support operating systems and diagnostic features	6. It cannot easily support OS and diagnostic features
7. It can easily handle Large/ Complex instruction sets	7. It cannot easily handle Large/ Complex instruction sets.
8. It is used in Mainframes and Microprocessors	8. It is used in RISC microprocessors