VTU solutions BEC306C Computer Organization and Architecture:

CBCS SCHEME

USN | | | | | | | | | | |

BEC306C

Third Semester B.E./B.Tech. Degree Examination, Dec.2023/Jan.2024
Computer Organization and Architecture

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	With a neat diagram, explain basic operational concept of computer.	10	L1	CO1
	b.	Explain following with an example : i) Three address instruction ii) Two-address instruction iii) One-address instruction	06	L1	CO1
	c.	Explain Big Endian and Little Endian with neat diagram.	04	L1	CO1
		OR			
Q.2	a.	Discuss IEEE standard for single precision and double precision floating point numbers with example.	08	L1	CO1
	b.	What is system software? List functions of system software and explain how the processor is shared between user program and os routine.	08	L1	CO1
	c.	Explain computer basic performance equation.	04	L1	CO1
		Module – 2			
Q.3	a.	What is an addressing mode? Explain any five types of addressing modes with example.	10	L1	CO2
	b.	Write a program to add 'n' number using indirect addressing mode.	05	L2	CO2
	c.	Explain stack operations.	05	L2	CO2
		OR			
Q.4	a.	What are assembler directives? Explain various assembler directives used in assembly language program.	08	L2	CO2
	b.	Explain subroutine linkage with an example using linkage register.	06	L2	CO2
	c.	Explain the shift and rotate operations with example.	06	L2	CO2
		Module – 3			
Q.5	a.	Showing register configuration in I/O Interface, Explain program controlled input/output with program.	08	L2	CO2
	b.	Explain the registers involved in DMA interface.	06	L2	CO2
	c.	What is an interrupt? Explain interrupt hardware.	06	L2	CO2
		OR			
Q.6	a.	Explain the following method of handling interrupts from multiple devices. i) Daisy chain method ii) Priority structure	08	L2	CO3
	b.	What is Bus arbitration? Explain centralized bus arbitration mechanism with a neat diagram.	08	L2	CO3
	c.	Explain the concept of vectored interrupt.	04	L2	CO3
		Module – 4			
Q.7	a.	Explain internal organization of 16×8 memory chip.	08	L2	CO4
	b.	With a neat diagram, explain working principle of magnetic disk.	06	L2	CO4
	c.	With a neat diagram, explain virtual memory organization.	06	L2	CO2
		OR			
Q.8	a.	Explain the internal organization of 2M×8 DRAM chip with neat diagram.	08	L2	CO3
	b.	Explain a static RAM cell with a neat diagram.	06	L2	CO3
	c.	Discuss the concept of cache memory.	06	L2	CO3

		Module – 5			
Q.9	a.	Explain with neat diagram, Single Bus organization of data path inside a processor.	08	L2	CO4
	b.	Discuss the control sequence for execution of instruction ADD (R3), R1.	06	L2	CO4
	c.	Describe the organization of hardwired control unit.	06	L2	CO4
		OR			
Q.10	a.	Explain multiple bus/three bus organization with a neat diagram.	10	L2	CO5
	b.	What is microprogrammed control? Explain its basic organization with suitable diagram and example.	10	L2	CO5

* * * * *

1. A. With a neat diagram, explain the basic operational concept of Computer

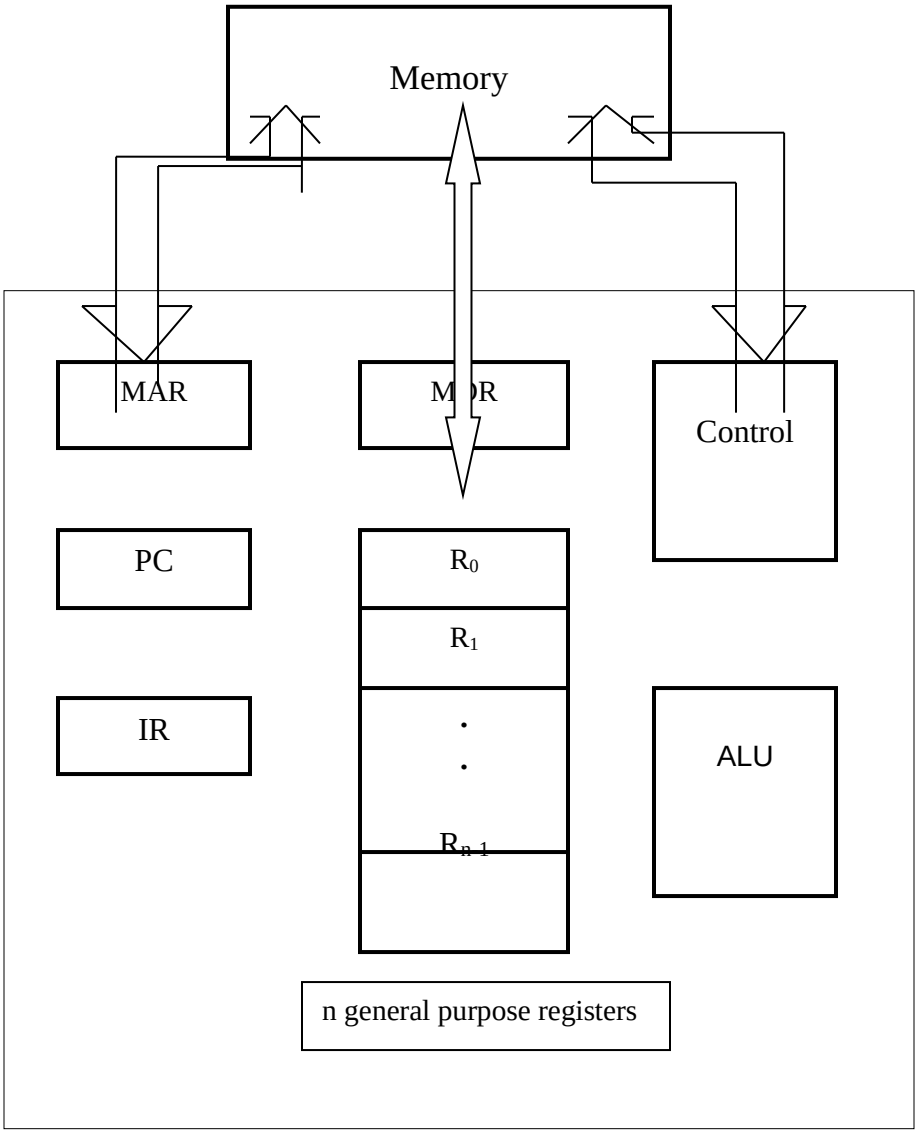
Basic Operational Concepts

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. A typical instruction may be

Add LOCA, R0

This instruction adds the operand at memory location LOCA to the operand in a register in the processor, R0, and places the sum in the register R0. The original contents of location LOCA are preserved whereas those of R0 are overwritten. First the instruction is fetched from the memory into the processor. Next the operand at LOCA is fetched and added to the contents of R0. Finally the resulting sum is stored in register R0.

Transfers between memory and processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data is transferred to or from the memory. The memory and processor connection is shown in Fig 1.2.



The Instruction register (IR) holds the instruction that is currently being executed. Its output is available to control circuits which generate the timing signals that control various processing elements involved in executing the instruction.

The Program Counter (PC) holds the 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. MAR and MDR facilitate communication with the memory.

MAR (Memory Address Register) hold the address of the location to be accessed and MDR (Memory Data Register) contains data written into or read out of the addressed location.

If some devices require urgent servicing then they raise the interrupt signal interrupting the normal execution of the current program. The processor provides the requested service by executing the appropriate interrupt service routine.

1.B. Explain the following with an example: 1) Three Address Instruction 2) Two Address Instruction 3) One Address Instruction

Basic Instruction Types:

Three-Address Instructions: There are 3-operands or three addresses (labels used to specify the location of data) present along with the Op-code in the instruction. In these instructions at-most only two location can be in memory

Eg: Add R1, R2, R3 R3 ← R1 + R2 , Here R1, R2 and R3 are the general purpose registers (GPRs) present in the Proessor Chip.

Two-Address Instructions: There are 2-operands or two addresses (labels used to specify the location of data) present along with the Op-code in the instruction. In these instructions at-most only one location can be in memory

Eg: Add R1, R2 R2 ← R1 + R2, Here R1 and R2 are the general purpose registers (GPRs) present in the Proessor Chip.

One-Address Instructions: There is only 1-operands or 1- address (label used to specify the location of data) present along with the Op-code in the instruction which may be a memory location or any internal Processor Registers (GPRs).

Here all operations encoded in the Op-code of Instruction is carried out with respect to the data in Accumulator-register (AC) present inside the Processor with the Accumulator register also acting as the detination register after the operation.

Eg: Add M AC ← AC + [M] , Here the M represents any memory Location data specified by memory-address M

1.C. Explain Big Endian and Little Endian with neat diagram

Big-Endian and Little-Endian Assignments:

Big-Endian: lower byte addresses are used for storing the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for storing the less significant bytes of the word

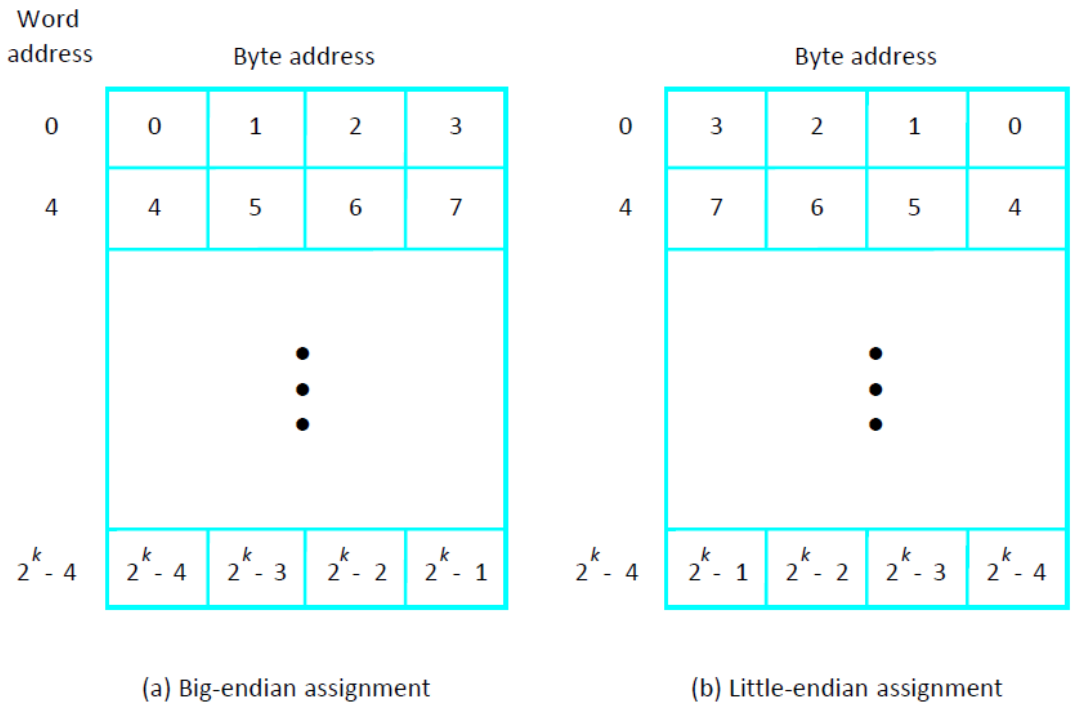
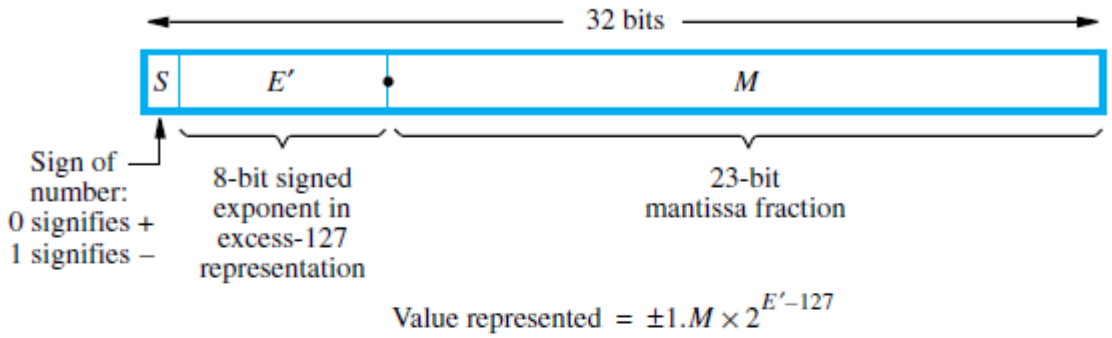


Figure 2.7. Byte and word addressing.

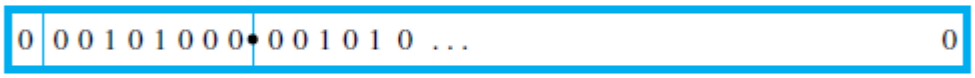
2.A. Discuss the IEEE standard for single precision and double precision floating point numbers with example

The basic IEEE format is a 32-bit representation, shown in Figure 9.26*a*. The leftmost bit represents the sign, S, for the number. The next 8 bits, E', represent the signed exponent of the scale factor (with an implied base of 2), and the remaining 23 bits, M, are the



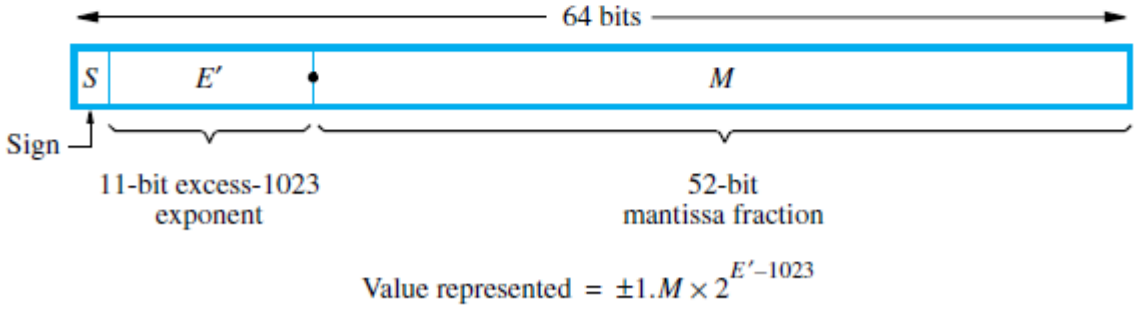
$$\text{Value represented} = \pm 1.M \times 2^{E'-127}$$

(a) Single precision



$$\text{Value represented} = 1.001010 \ldots 0 \times 2^{-87}$$

(b) Example of a single-precision number



$$\text{Value represented} = \pm 1.M \times 2^{E'-1023}$$

(c) Double precision

Figure 9.26 IEEE standard floating-point formats.

fractional part of the significant bits. The full 24-bit string, B, of significant bits, called the *mantissa*, always has a leading 1, with the binary point immediately to its right. Therefore, the mantissa

$$B = 1.M = 1.b_{-1}b_{-2}\ldots b_{-23}$$

has the value

$$V(B) = 1 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \cdots + b_{-23} \times 2^{-23}$$

By convention, when the binary point is placed to the right of the first significant bit, the number is said to be *normalized*. Note that the base, 2, of the scale factor and the leading 1 of the mantissa are both fixed. They do not need to appear explicitly in the representation.

Instead of the actual signed exponent, E, the value stored in the exponent field is an unsigned integer $E' = E + 127$. This is called the *excess*-127 format. Thus, E' is in the range $0 \leq E' \leq 255$. The end values of this range, 0 and 255, are used to represent special values, as described later. Therefore, the range of E' for normal values is $1 \leq E' \leq 254$. This means that the actual exponent, E, is in the range $-126 \leq E \leq 127$. The use of the excess-127 representation for exponents simplifies comparison of the relative sizes of two floating-point numbers. (See Problem 9.23.)

The 32-bit standard representation in Figure 9.26*a* is called a *single-precision* representation because it occupies a single 32-bit word. The scale factor has a range of 2^{-126} to 2^{+127}, which is approximately equal to $10^{\pm 38}$. The 24-bit mantissa provides approximately the same precision as a 7-digit decimal value. An example of a single-precision floating-point number is shown in Figure 9.26*b*.

To provide more precision and range for floating-point numbers, the IEEE standard also specifies a *double-precision* format, as shown in Figure 9.26*c*. The double-precision format has increased exponent and mantissa ranges. The 11-bit excess-1023 exponent E' has the range $1 \leq E' \leq 2046$ for normal values, with 0 and 2047 used to indicate special values, as before. Thus, the actual exponent E is in the range $-1022 \leq E \leq 1023$, providing scale factors of 2^{-1022} to 2^{1023} (approximately $10^{\pm 308}$). The 53-bit mantissa provides a precision equivalent to about 16 decimal digits.

2.B. What is system software? List functions of system software and explain how the processor is shared between user program and OS routine.

Sol:

Software

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

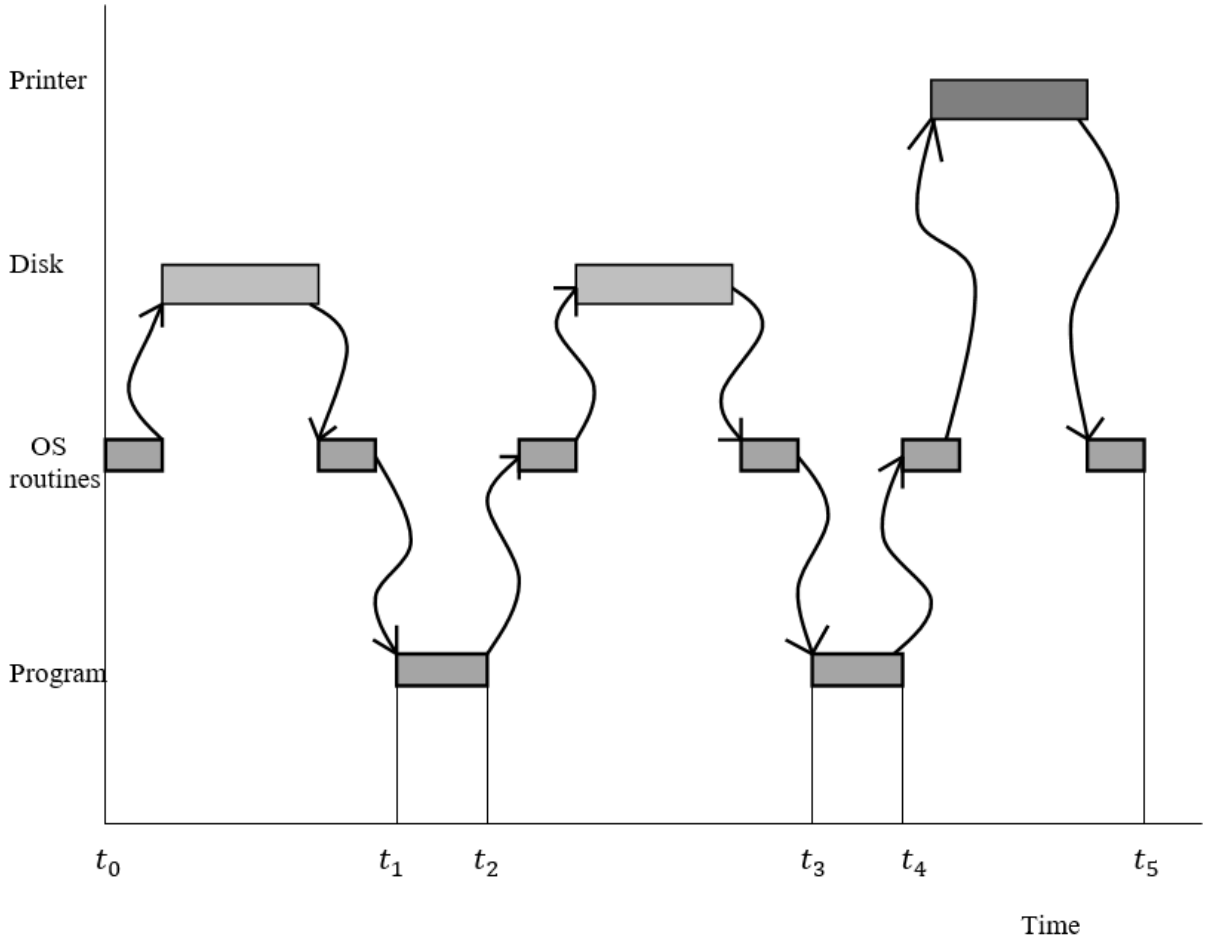
- Receiving and interpreting user commands.

- 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, spreadsheets or games with data supplied by users.
- Controlling I/O units to receive input information and produce output results.
- Translating programs from source form prepared by user into object form consisting of machine instructions.
- Linking and running user written application programs with existing standard library routines such as numerical computational packages.

A compiler is a system software program translates the high level language program in to a suitable machine language program containing instructions such as Add and Load instructions.

Operating system (OS) 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. These tasks include assigning memory and magnetic disk space to program and data files, moving data between memory and disk units and handling I/O operations.

Consider a system with one processor, one disk and one printer. When the application program has been compiled from a high language form to 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 part of the program's task involves reading a data file from the disk in to the memory, performing some computations on the data and printing the results. When the execution of the program reaches the point where the data file is needed, the program requests the operating system to transfer the data file from the disk to the memory. The OS performs this task and passes the execution control back to the application program, which then proceeds to perform the required computation. When the computation is completed and the results are ready to be printed, the application program again sends the request to the operating system. An OS routine is then executed to cause the printer to print the results.



The execution control passes back and forth between application program and OS routines. This sharing of processor execution time is illustrated by a time line diagram as shown in Fig 1.5. During the time period t_0 to t_1, an OS routine initiates the application loading program from the disk to the memory, waits until the transfer is completed, and then passes execution control to the application program. A similar pattern of activity occurs during period t_2 to t_3 and period t_4 to t_5 , when the operating system transfers the data file from the disk and print the results. At t_5, the operating system may load and execute another application program. Notice that the disk and processor are idle during most of the time period t_4 to t_5. The operating system manages the concurrent execution of several application programs to make best possible use of computer resources. This pattern of concurrent execution is called *multiprogramming* or *multitasking*.

2.C. Explain computer basic performance equation.

Basic Performance Equation

Let T be the processor time required to execute a program that has been prepared by some high level language. The compiler generates machine level object program that corresponds to source program. Assume that complete execution of the program requires the execution of N machine language instructions. Suppose that the average number of basic steps needed to execute one machine instruction is S, where each basic step is completed in one clock cycle. If the clock rate is R cycles per second, the program execution time is given by *basic performance equation.*

$$T = \frac{N \times S}{R}$$

To achieve high performance, the value of T must be reduced which can be done by reducing N and S, and increasing R. The value of N is reduced if the source program is compiled in fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform or if the execution of instructions are overlapped. Using a higher-frequency clock increases the value of R which means the time required to complete a basic execution step is reduced.

3.A. What is an addressing mode? Explain any five types of addressing modes with examples.

Sol:

Addressing Modes

The different ways in which the location of an operand is specified in an instruction is known as *addressing modes*. Variables and constants are the simplest data types. In assembly language, a variable is represented by allocating a register or memory location to hold its value. Thus, the value can be changed as needed using appropriate instructions.

- *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 this location is given explicitly in the instruction.

The instruction $Move \quad LOC, R2$
uses two modes. Processor registers are temporary storage locations where data in a register is accessed using the Register mode. Address and data constants can be represented in assembly language using the Immediate mode addressing where the operand is given explicitly in the instruction. For example, the instruction

$$Move \quad 200_{immediate}, R0$$

Places the value 200 in register $R0$. A common convention is to use # in front of the immediate value to indicate that this value is to be used as an immediate operand. Hence we can write the instruction above in the form

$$Move \quad \#200, R0$$

Constant values are used frequently in high-level language programs. The statements $A = B + 6$ contains the constant 6. Assuming that A and B have been declared as variables and may be accessed using Absolute mode.

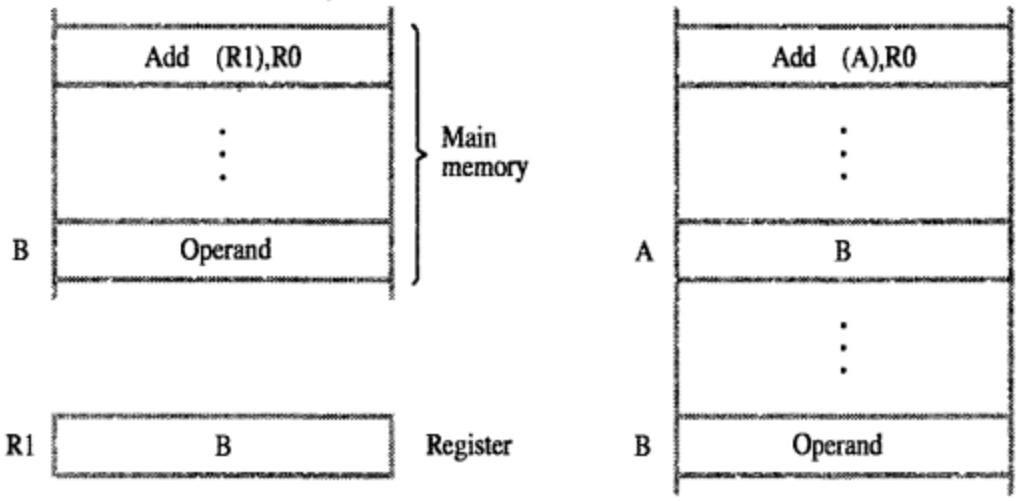
$$Move \quad B, R1$$
$$Add \quad \#6, R1$$
$$Move \quad R1, A$$

Indirection and Pointers

In indirect mode addressing, the instruction does not give the operand or the address explicitly. Instead it provides information from which the memory address of the operand can be determined. This address is referred to as *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.

To execute the Add instruction in Fig 2.1a, the processor uses the value B, which is in the register $R1$, as the effective address of the operand. It requests a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register $R0$. Indirect addressing through a memory location is also possible as shown in Fig 2.1b. In this case, the processor first reads the contents of memory location A, then request the second read operation using the value B as a address to obtain the operand.



a) Through a general purpose register b) Through a memory location

Fig 2.1: Indirect addressing

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

Indexing and Arrays

This addressing mode provides flexibility for accessing operands and is useful in dealing with lists and arrays.

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

We indicate the Index mode symbolically as $X(Ri)$ where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by

$$EA = X + [Ri]$$

Fig 2.2 illustrates two ways of using Index mode. In Fig 2.2a, the index register $R1$ contains the address of the memory location and the value X defines an *offset* or *displacement* from this address to the location where the operand is found.

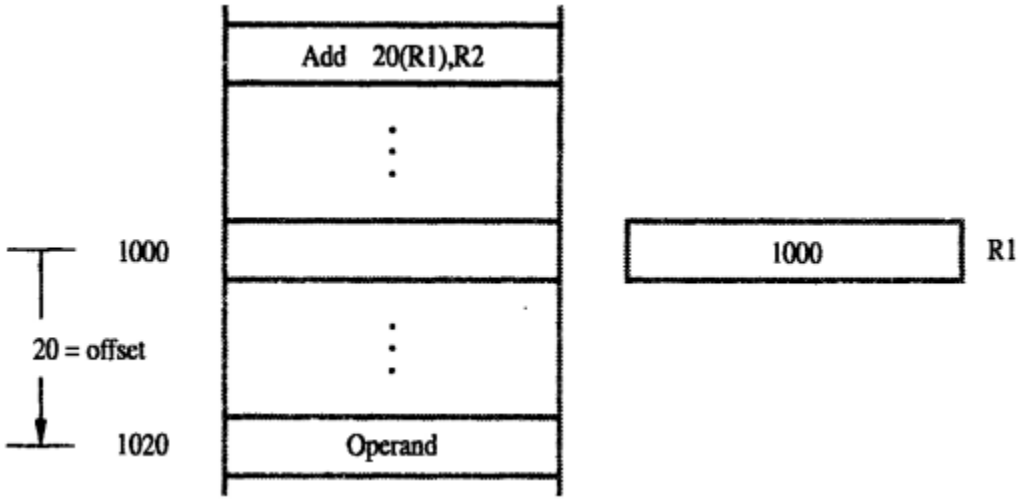


Fig 2.2 a: Offset is given as a constant

An alternate use is illustrated in Fig 2.2b. Here, the constant X corresponds to a memory address and the content of the index register defines the offset to the operand. In either case, the effective address is the sum of two values, one is given explicitly in the instruction and the other is stored in the register.

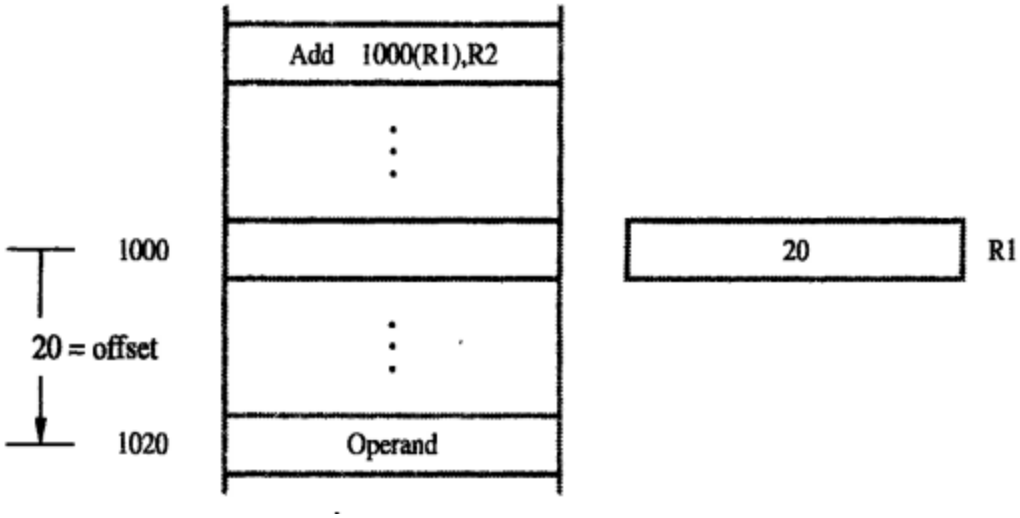


Fig 2.2b: Offset is in the register

Relative Addressing

Here the Program Counter (PC) is used instead of a general purpose register. In *Relative mode*, the effective address is determined by the Index mode using program counter in place of general-purpose register Ri. It's most common use is to specify the target address in branch instructions. An instruction such as

$$Branch > 0 \quad LOOP$$

causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied. This location can be computed by specifying it as an offset from the current value of the program counter. Suppose that Relative mode is used to generate the target branch address LOOP in the Branch instruction of the program

$$\begin{aligned} LOOP: \quad & Add \quad (R2), R0 \\ & Add \quad \#4, R2 \\ & Decrement \quad R1 \\ & Branch > 0 \quad LOOP \end{aligned}$$

Assume that the four instructions of the loop body, starting at LOOP are located at memory locations $1000, 1004, 1008$ and 1012. Hence the updated contents of the PC at the time of branch target address is generated will be 1016. To branch to location LOOP(1000), the offset needed is $X = -16$.

Auto-increment 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 is automatically incremented to point to the next item in the list. The Auto-increment mode is written as

$$(Ri) +$$

The increment is 1 for byte-sized operands, 2 for 16 bit operands and 4 for 32 bit operands.

Auto-decrement mode – The content of a register specified in the instruction is first automatically decremented and then used as effective address of the operand. In this mode, operands are accessed in descending address order.

These two modes can be used together to implement an important data structure called stack.

3.B. Write a program to add 'n' number using indirect addressing mode.
Sol:

The programme is written to add n – numbers located in the memory starting at location address labeled as List and the result is stored in the location labeled as SUM.

We assume a 32-bit word-length and Each number is assumed to be of 32 bits and each general purpose register is of 32 bits

```
        Move        #List, R0
        Move        N, R1
        Clear       R2
Loop:   ADD         (R0), R2
        ADD         #4, R0
        Decrement   R1
        Branch > 0  Loop
        Move R2, SUM
```

3.C. Explain Stack Operations
Sol:

Stack operations are facilitated with Stack pointer. This register should not be used for any other purpose. The stack grows in the direction of decreasing memory addresses. In other words, Stack memory is always defined at the top of memory. It works on Last-in-First-Out (LIFO) technique.

The general instructions used for stack operations are:

1) **PUSH Reg/[MemoryLocation]**
2) **POP Reg/[Memory Location]**

Suppose we define the Stack at 7018 i.e. Stack Pointer Register is currently Loaded with 7019 and if we execute a Push Instruction say,
Eg: PUSH R0:
Step1: The Stack-Pointer Contents are auto-decremented to point to the previous word address.
Step2: The contents of R0 is then saved on to this new Word address in memory pointed out by Stack Pointer contents

Status of Registers and Mem-Locations *before* Executing PUSH R0	**Status of Registers and Mem-Locations *After* Executing PUSH R0**
SP := 7018 H	**SP= 7018-4=7014 H (Assuming a Word-Length of 4-bytes)**
R0:= 0024 H	**R0=0024 H**
[7014]=0005 H	**[7014] = 0024 H**
R3:=0060 H	**R3=0060 H**

Now, if the POP instruction is executed say,
Eg: POP R3
Step1: The contents from the memory-location pointed out by Stack-Pointer i.e the address 7014 is read and pasted into R3 register in the Processor

Step2: The contents of Stack-Pointer is then auto incremented to point to the next word

Status of Registers and Mem-Locations *before* Executing POP R3	**Status of Registers and Mem-Locations *After* Executing POP R3**
SP := 7014 H	**SP= 7018-4=7014 H (Assuming a Word-Length of 4-bytes)**
R0:= 0024 H	**R0=0024 H**
[7014]=0024 H	**[7014] = 0024 H**
R3:=0060 H	**R3=0024 H**

4.A. What are Assembler Directives? Explain various assembler directives used in ALP.
Sol:

Assembler Directives

The assembly language allows the programmer to specify other information needed to translate the source program to object program. Suppose the name SUM is used to represent the value 200. This fact may be conveyed to the assembler program through a statement such as

```
SUM  EQU  200
```

This statement does not denote the instruction that will be executed when the object program is run. It informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements are *assembler directives* (or commands) are used by the assembler when it translates the source program in to a object program.
ORIGIN is a directive that tells the assembler program where in the memory to place the data block.
DATAWORD directive is used to inform the assembler to place the data in the address.
RESERVE directive declares a memory block and does not cause any data to be loaded in these locations.
ORIGIN directive specifies that the instructions of an object program are to be loaded in the memory starting at an address.
END is directive which indicates the end of the source program text. The END directive includes the label START, which is the address of the location at which execution of the program is to begin.
RETURN is an assembler directive that identifies the point at which the execution of the program should be terminated.

4.B. Explain subroutine linkage with an example using linkage register.

Subroutines

It is often necessary to perform a particular subtask many times on different data values. Such subtask is called *subroutine*. When a program branches to a subroutine we call that it is *calling* a subroutine. The instruction that performs this branch operation is called a Call instruction. The subroutine is said to *return* to program that called it by executing a Return instruction. The location where the calling program resumes execution is the location pointed by the updated PC while the Call instruction being executed. Hence the contents of the PC must be saved by the Call instruction to enable correct return to the calling program. This way in which the computer makes it possible to call and return from subroutines is referred to as *subroutine linkage method.*

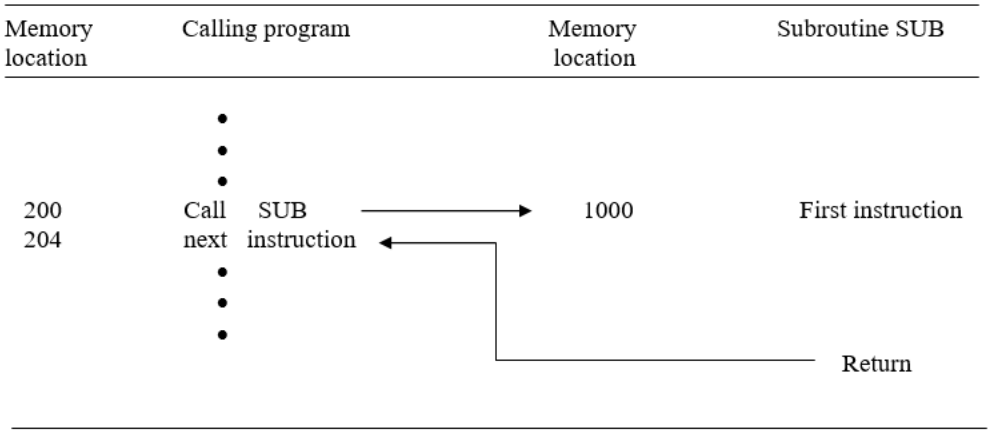
The Call instruction is a special branch instruction that performs the following operations:

1. Store the contents of PC in the link register.
2. Branch to the target address specified by the instruction.

The Return instruction is a special branch instruction that performs the operation:

Branch to the address contained in the link register.

Fig 2.6 illustrates this procedure.



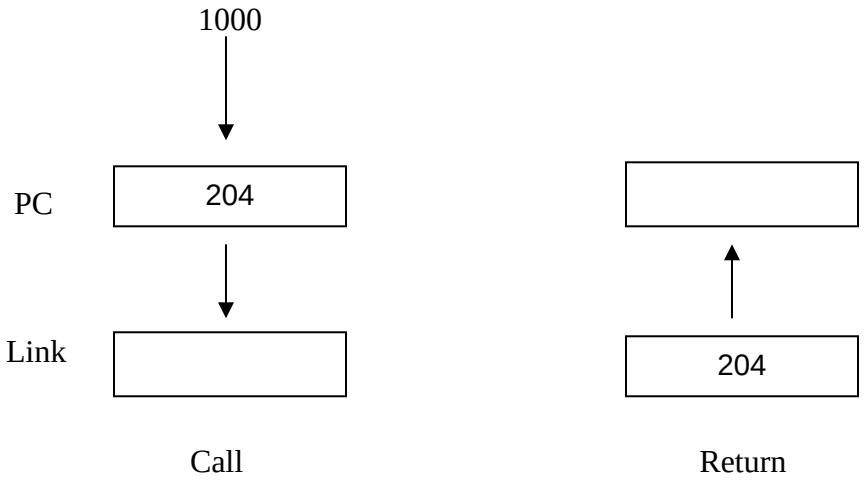


Fig 2.6: Subroutine linkage using a link register

4.C. Explain the shift and rotate operations with examples.

Sol:

Shift and Rotate Instructions

2.10.2 Shift and Rotate Instructions

There are many applications that require the bits of an operand to be shifted right or left some specified number of bit positions. The details of how the shifts are performed depend on whether the operand is a signed number or some more general binary-coded information. 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

Two logical shift instructions are needed, one for shifting left (LShiftL) and another for shifting right (LShiftR). These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction. The general form of a logical left shift instruction is

LShiftL count,dst

The count operand may be given as an immediate operand, or it may be contained in a processor register. To complete the description of the left shift operation, we need to specify the bit values brought into the vacated positions at the right end of the destination operand, and to determine what happens to the bits shifted out of the left end. Vacated positions are filled with zeros, and the bits shifted out are passed through the Carry flag, C, and then dropped. Involving the C flag in shifts is useful in performing arithmetic operations on large numbers that occupy more than one word. Figure 2.30*a* shows an example of shifting the contents of register R0 left by two bit positions. The logical shift right instruction, LShiftR, works in the same manner except that it shifts to the right. Figure 2.30*b* illustrates this operation.

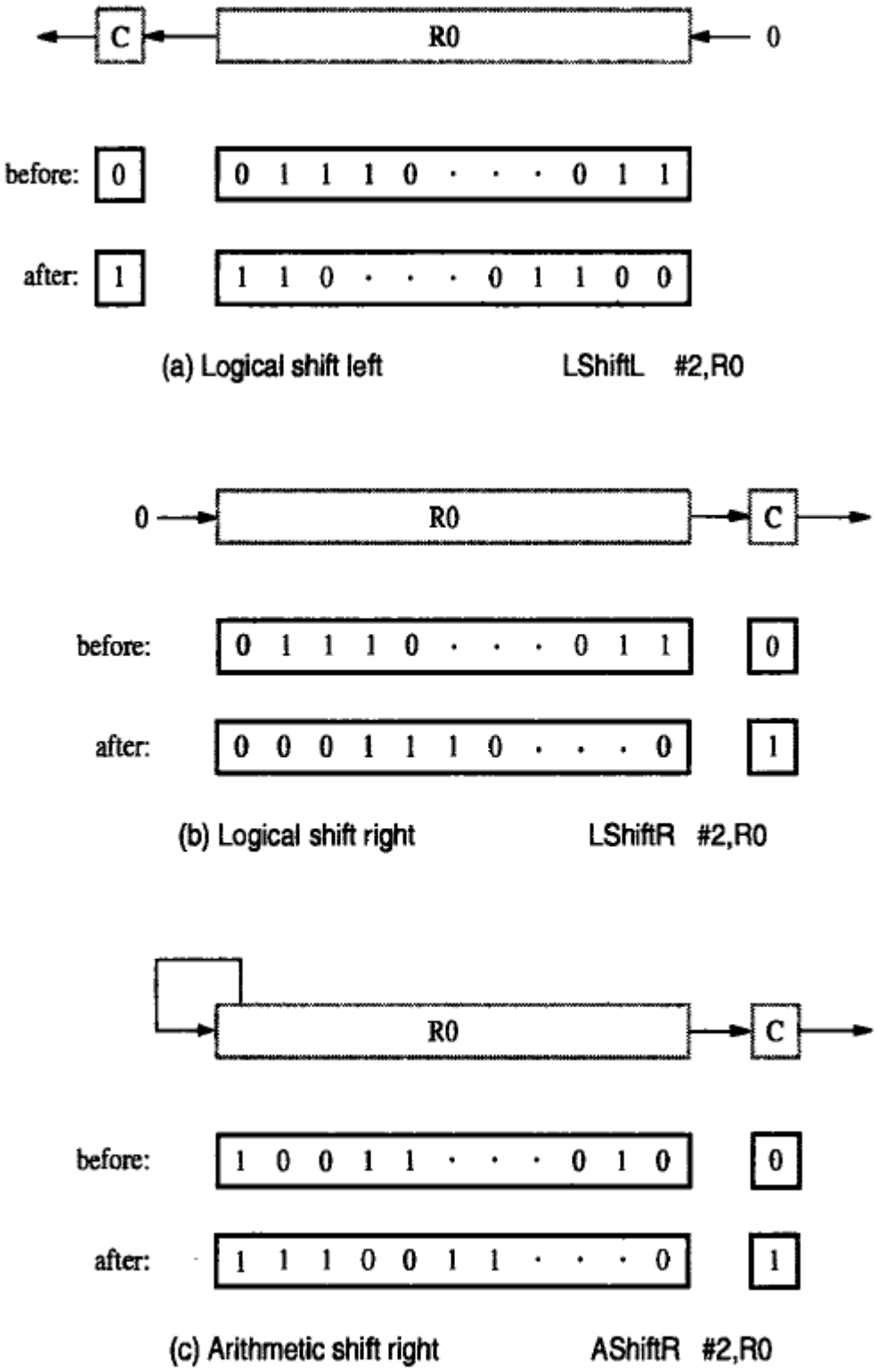


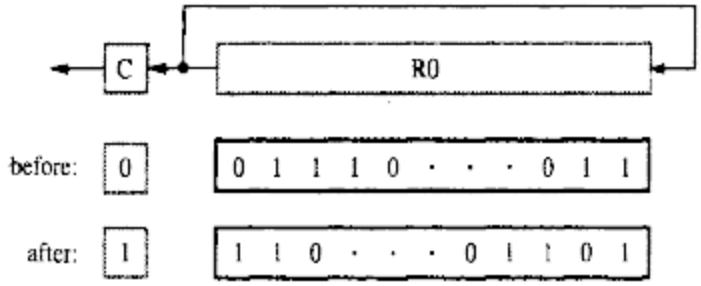
Figure 2.30 Logical and arithmetic shift instructions.

Arithmetic Shifts

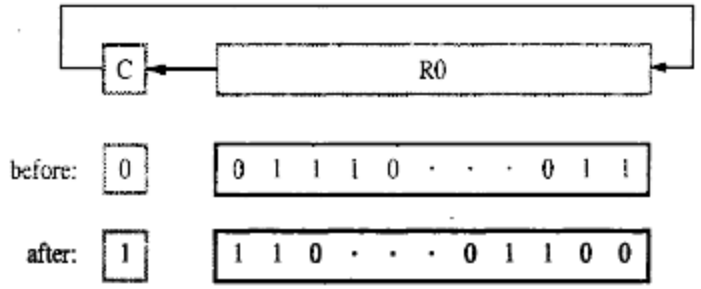
A study of the 2's-complement binary number representation in Figure 2.1 reveals that shifting a number one bit position to the left is equivalent to multiplying it by 2; and shifting it to the right is equivalent to dividing it by 2. Of course, overflow might occur on shifting left, and the remainder is lost in shifting right. Another important observation is that on a right shift the sign bit must be repeated as the fill-in bit for the vacated position. This requirement on right shifting distinguishes arithmetic shifts from logical shifts in which the fill-in bit is always 0. Otherwise, the two types of shifts are very similar. An example of an arithmetic right shift, AShiftR, is shown in Figure 2.30*c*. The arithmetic left shift is exactly the same as the logical left shift.

Rotate Operations

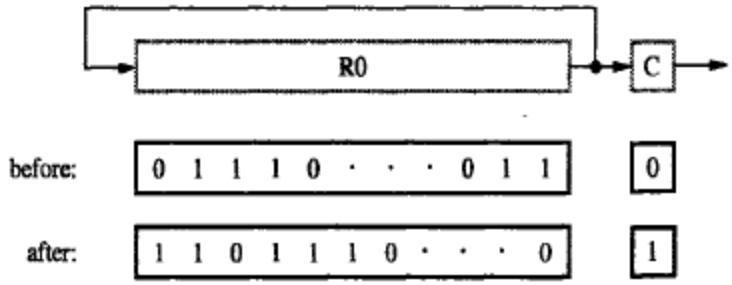
In the shift operations, the bits shifted out of the operand are lost, except for the last bit shifted out which is retained in the Carry flag C. To preserve all bits, a set of rotate instructions can be used. They move the bits that are shifted out of one end of the operand back into the other end. 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. Figure 2.32 shows the left and right rotate operations with and without the C flag being included in the rotation. Note that when the C flag is not included in the rotation, it still retains the last bit shifted out of the end of the register. The mnemonics RotateL, RotateLC, RotateR, and RotateRC, denote the instructions that perform the rotate operations. The main use for Rotate instructions is in algorithms for performing arithmetic operations other than addition and subtraction,



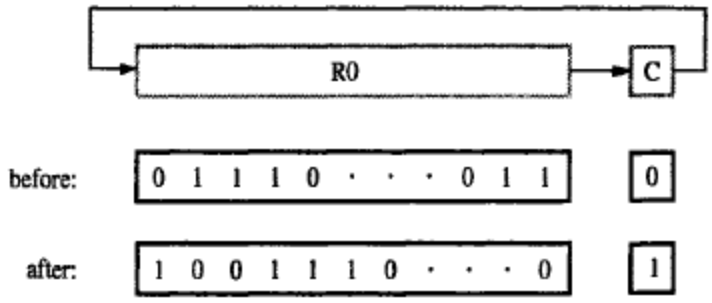
(a) Rotate left without carry RotateL #2,R0



(b) Rotate left with carry RotateLC #2,R0



(c) Rotate right without carry RotateR #2,R0



(d) Rotate right with carry RotateRC #2,R0

Figure 2.32 Rotate instructions.

5.A. Showing register configuration in I/O Interface, Explain program controlled input/output with program.

4.1 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 Figure 4.1. 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 a write operation, and the requested data are transferred over the data lines. As mentioned in Section 2.7, when I/O devices and the memory share the same address space, the arrangement is called *memory-mapped I/O.*

With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device. For example, if DATAIN is the address

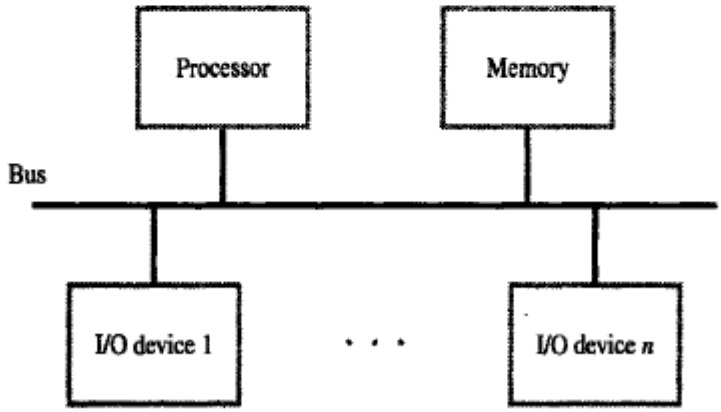


Figure 4.1 A single-bus structure.

of the input buffer associated with the keyboard, the instruction

```
Move    DATAIN,R0
```

reads the data from DATAIN and stores them into processor register R0. Similarly, the instruction

```
Move    R0,DATAOUT
```

sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.

Most computer systems use memory-mapped I/O. Some processors have special In and Out instructions to perform I/O transfers. For example, processors in the Intel family described in Chapter 3 have special I/O instructions and a separate 16-bit address space for I/O devices. When building a computer system based on these processors, the designer has the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space. The latter approach is by far the most common as it leads to simpler software. One advantage of a separate I/O address space is that I/O devices deal with fewer address lines. Note that a separate I/O address space does not necessarily mean that the I/O address lines are physically separate from the memory address lines. A special signal on the bus indicates that the requested read or write transfer is an I/O operation. When this signal is asserted, the memory unit ignores the requested transfer. The I/O devices examine the low-order bits of the address bus to determine whether they should respond.

Figure 4.2 illustrates the hardware required to connect an I/O device to the bus. The address decoder 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 of the I/O device. Both the data and status registers are connected to the data bus and

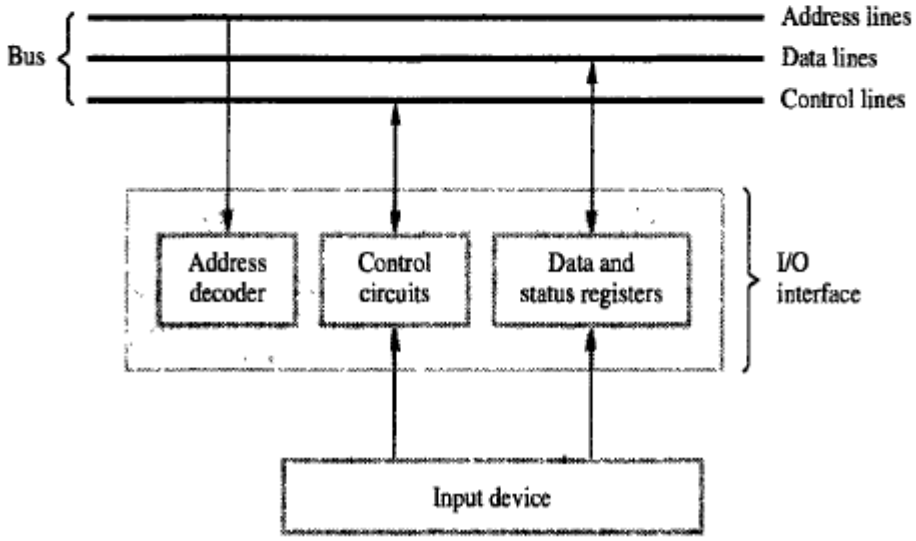


Figure 4.2 I/O interface for an input device.

assigned unique addresses. The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's *interface circuit*.

I/O devices operate at speeds that are vastly different from that of the processor. When a human operator is entering characters at a keyboard, the processor is capable of executing millions of instructions between successive character entries. An instruction that reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface. Also, we must make sure that an input character is read only once.

The basic ideas used for performing input and output operations were introduced in Section 2.7. For an input device such as a keyboard, a status flag, SIN, is included in the interface circuit as part of the status register. This flag is set to 1 when a character is entered at the keyboard and cleared to 0 once this character is read by the processor. Hence, by checking the SIN flag, the software can ensure that it is always reading valid data. This is often accomplished in a program loop that repeatedly reads the status register and checks the state of SIN. When SIN becomes equal to 1, the program reads the input data register. A similar procedure can be used to control output operations using an output status flag, SOUT.

To review the basic concepts, let us consider a simple example of I/O operations involving a keyboard and a display device in a computer system. The four registers shown in Figure 4.3 are used in the data transfer operations. Register STATUS contains two control flags, SIN and SOUT, which provide status information for the keyboard and the display unit, respectively. The two flags KIRQ and DIRQ in this register are used in conjunction with interrupts. They, and the KEN and DEN bits in register CONTROL, will be discussed in Section 4.2. Data from the keyboard are made available

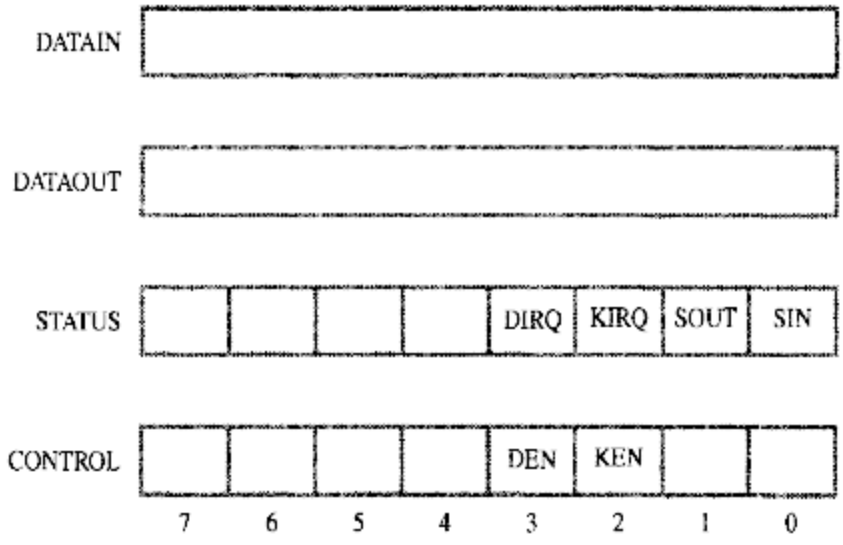


Figure 4.3 Registers in keyboard and display interfaces.

```
        Move        #LINE,R0        Initialize memory pointer.
WAITK   TestBit     #0,STATUS       Test SIN.
        Branch=0    WAITK           Wait for character to be entered.
        Move        DATAIN,R1       Read character.
WAITD   TestBit     #1,STATUS       Test SOUT.
        Branch=0    WAITD           Wait for display to become ready.
        Move        R1,DATAOUT      Send character to display.
        Move        R1,(R0)+        Store charater and advance pointer.
        Compare     #$0D,R1         Check if Carriage Return.
        Branch≠0    WAITK           If not, get another character.
        Move        #$0A,DATAOUT    Otherwise, send Line Feed.
        Call        PROCESS         Call a subroutine to process the
                                      the input line.
```

Figure 4.4 A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

in the DATAIN register, and data sent to the display are stored in the DATAOUT register.

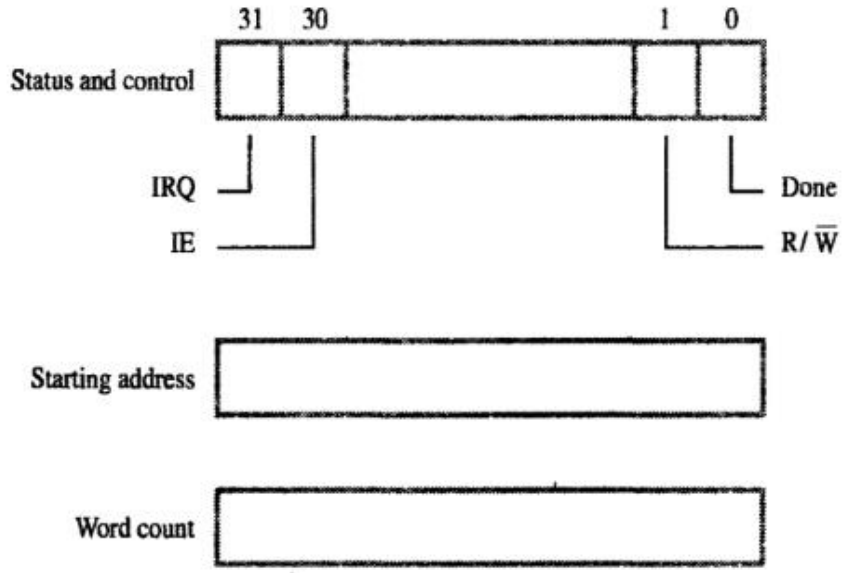
The program in Figure 4.4 is similar to that in Figure 2.20. This program reads a line of characters from the keyboard and stores it in a memory buffer starting at location LINE. Then, it calls a subroutine PROCESS to process the input line. As each character is read, it is *echoed back* to the display. Register R0 is used as a pointer to the memory buffer area. The contents of R0 are updated using the Autoincrement addressing mode so that successive characters are stored in successive memory locations.

Each character is checked to see if it is the Carriage Return (CR) character, which has the ASCII code 0D (hex). If it is, a Line Feed character (ASCII code 0A) is sent to move the cursor one line down on the display and subroutine PROCESS is called. Otherwise, the program loops back to wait for another character from the keyboard.

This example illustrates *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.

5.B. Explain the registers involved in DMA Interface.

Sol:



Registers in a DMA interface.

Figure 4.18 shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations. Two registers are used for storing the starting address and the word count. The third register contains status and control flags. The R/$\overline{W}$ bit determines the direction of the transfer. When this bit is set to 1 by a program instruction, the controller performs a read operation, that is, it transfers data from the memory to the I/O device. Otherwise, it performs a write operation. When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1. Bit 30 is the Interrupt-enable flag, IE. When this flag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data. Finally, the controller sets the IRQ bit to 1 when it has requested an interrupt.

Q5.C: What is an Interrupt? Explain interrupt hardware.
Sol:

Suppose a program enters a wait loop in which it repeatedly tests the device status. During this period, the processor is not performing any useful computation. 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 alert the processor when it becomes ready. It can do so by sending a hardware signal called an *interrupt* to the processor. At least one of the bus control lines, called an *interrupt-request* line, is usually dedicated for this purpose. Since the processor is no longer required to continuously check the status of external devices, it can use the waiting period to perform other useful functions. Indeed, by using interrupts, such waiting periods can ideally be eliminated.

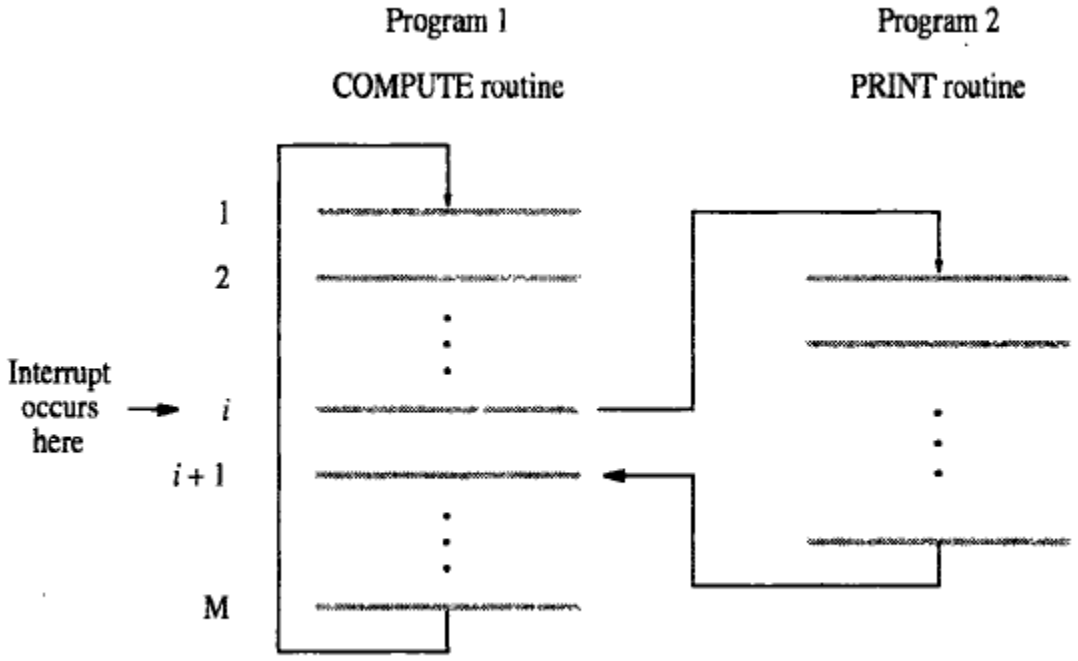


Figure 4.5 Transfer of control through the use of interrupts.

The routine executed in response to an interrupt request is called the Interrupt Service Routine. Interrupts bear considerable resemblance to subroutine calls. Assume that an interrupt request arrives during execution of instruction – 'i' as shown in the above figure, then :

processor first completes execution of instruction i. Then, it loads the program counter with the address of the first instruction of the interrupt-service routine. For the time being, let us assume that this address is hardwired in the processor. After execution of the interrupt-service routine, the processor has to come back to instruction $i + 1$. Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction $i + 1$, must be put in temporary storage in a known location. A Return-from-interrupt instruction at the end of the interrupt-service routine reloads the PC from that temporary storage location, causing execution to resume at instruction $i + 1$. In many processors, the return address is saved on the processor stack. Alternatively, it may be saved in a special location, such as a register provided for this purpose.

An interrupt is more than a simple mechanism for coordinating I/O transfers. In a general sense, interrupts enable transfer of control from one program to another to be initiated by an event external to the computer. Execution of the interrupted program resumes after the execution of the interrupt-service routine has been completed. The concept of interrupts is used in operating systems and in many control applications where processing of certain routines must be accurately timed relative to external events. The latter type of application is referred to as *real-time processing*.

4.2.1 Interrupt Hardware

We pointed out that an I/O device requests an interrupt by activating a bus line called interrupt-request. Most computers are likely to have several I/O devices that can request an interrupt. A single interrupt-request line may be used to serve n devices as depicted in Figure 4.6. All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}. This is the inactive state of the line. When a device requests an interrupt by closing its switch, the voltage on the line drops to 0, causing the interrupt-request signal, INTR, received by the processor to go to 1. Since the closing of one or more switches will cause the line voltage to drop to 0, the value

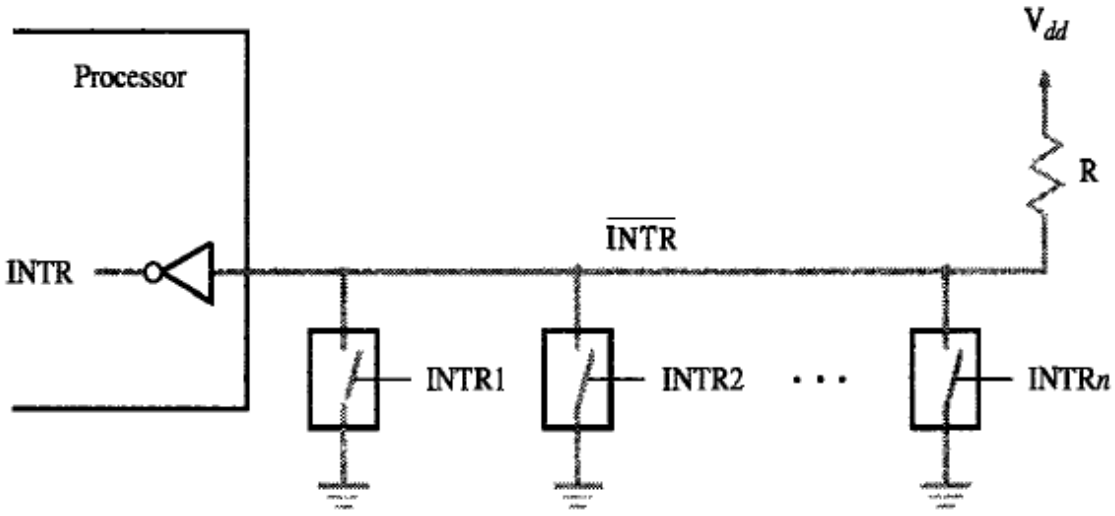


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

of INTR is the logical OR of the requests from individual devices, that is,

$$INTR = INTR_1 + \cdots + INTR_n$$

It is customary to use the complemented form, $\overline{INTR}$, to name the interrupt-request signal on the common line, because this signal is active when in the low-voltage state.

In the electronic implementation of the circuit in Figure 4.6, special gates known as *open-collector* (for bipolar circuits) or *open-drain* (for MOS circuits) are used to drive the $\overline{INTR}$ line. The output of an open-collector or an open-drain gate is equivalent to a switch to ground that is open when the gate's input is in the 0 state and closed when it is in the 1 state. The voltage level, hence the logic state, at the output of the gate is determined by the data applied to all the gates connected to the bus, according to the equation given above. Resistor R is called a *pull-up resistor* because it pulls the line voltage up to the high-voltage state when the switches are open.

6. A. Explain the following method of handling interrupts from multiple devices:
i. Daisy chain method ii. Priority structure

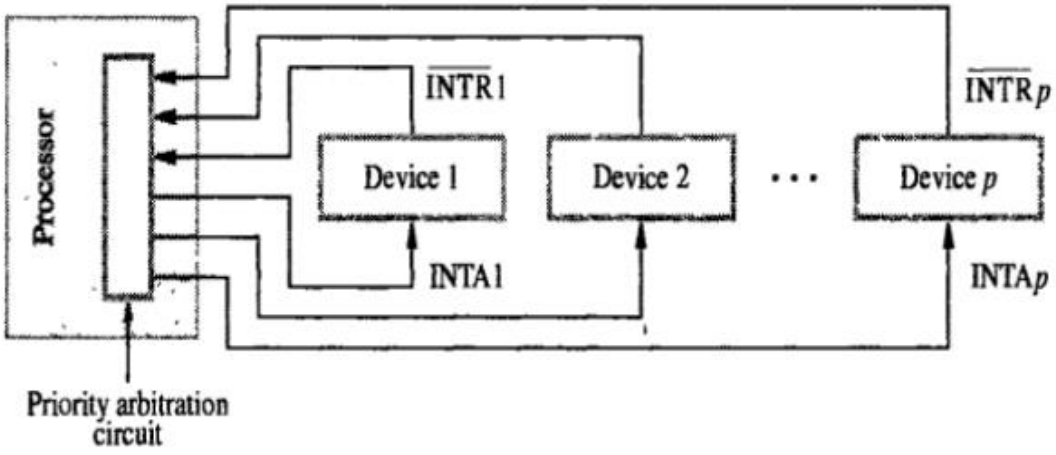


Fig 6: Implementation of interrupt priority using individual interrupt request & acknowledge lines

Simultaneous Requests

If several devices share one interrupt request line, some other mechanism is needed. When several devices raises interrupt request and $\overline{INTR}$ line is activated, the processor responds by setting the INTA line to 1. The signal is received by device 1. Device 1 passes the signal onto device 2 only if it does not require any service. If device 1 has pending request for interrupt, it blocks the INTA signal and proceeds to put its identification code on to data lines. In daisy chain the device that is electrically closest to the processor has the highest priority.

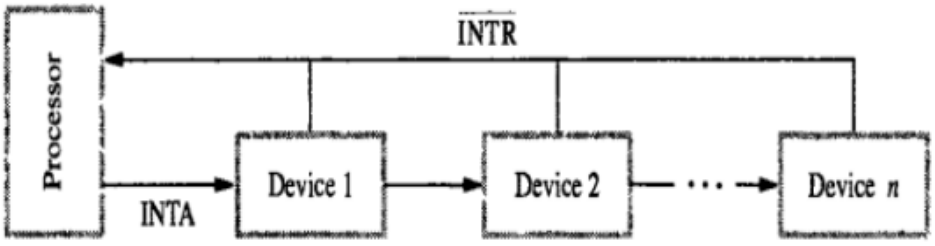


Fig 7: Daisy chain

Devices can be organized in groups and each group is connected at a different priority level. Within group devices are connected in daisy chain.

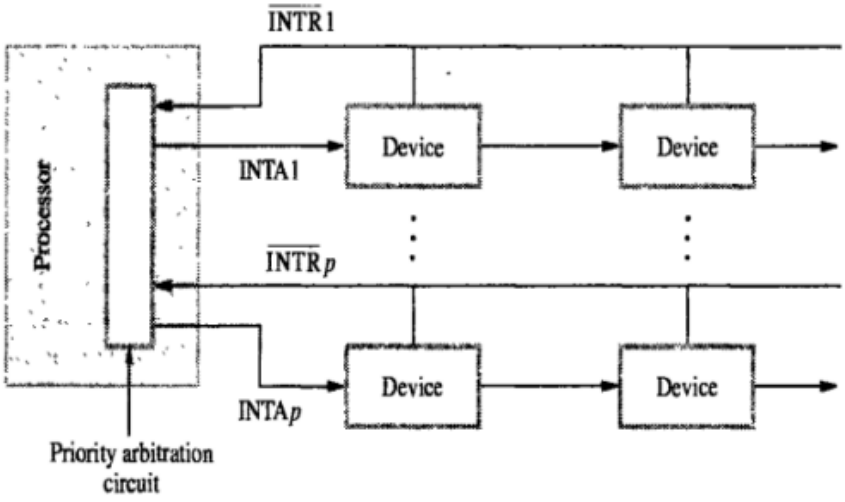


Fig 8: Arrangement of priority groups

6.b What is bus arbitration ? Explain centralized bus arbitration mechanism with a neat diagram.

Bus Arbitration-

Bus Arbitration refers to the process by which the current bus master accesses and then leaves the control of the bus and passes it to the another bus requesting processor unit. The controller that has access to a bus at an instance is known as **Bus master.**

A conflict may arise if the number of DMA controllers or other controllers or processors try to access the common bus at the same time, but access can be given to only one of those. Only one processor or controller can be Bus master at the same point of time. To resolve these conflicts, Bus Arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus. The **Bus Arbiter** decides who would become current bus master.

There are two approaches to bus arbitration:

1. Centralized Arbitration

In centralized bus arbitration, a single bus arbiter performs the required arbitration. The bus arbiter may be the processor or a separate controller connected to the bus.
There are three different arbitration schemes that use the centralized bus arbitration approach. There schemes are:

a) Daisy chaining

b) Polling method

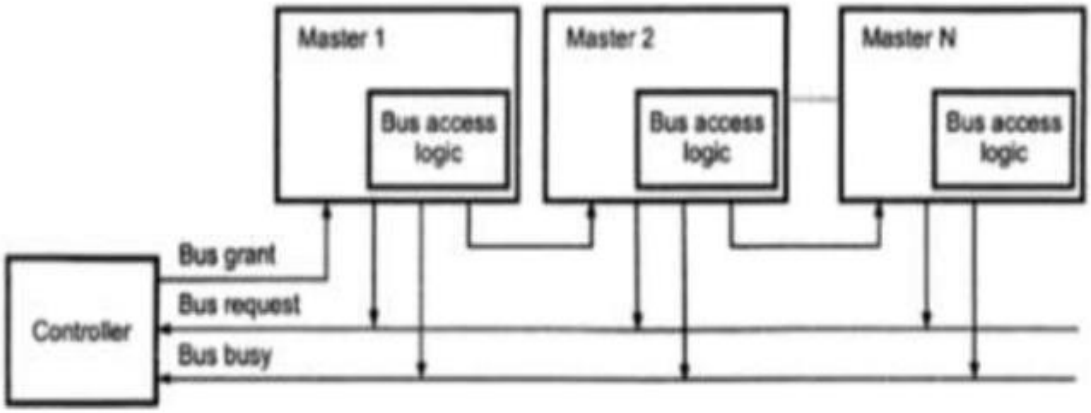
c) Independent request

a) Daisy chaining

The system connections for Daisy chaining method are shown in fig below.

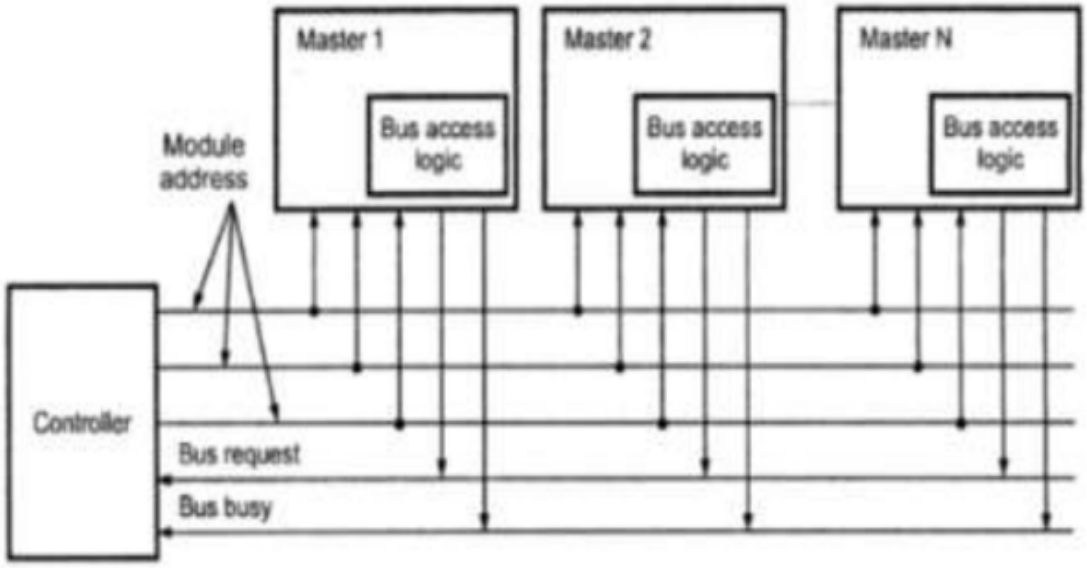
a) Daisy chaining

The system connections for Daisy chaining method are shown in fig below.



It is simple and cheaper method. All masters make use of the same line for bus request. In response to the bus request the controller sends a bus grant if the bus is free. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activities the busy line and gains control of the bus. Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access.

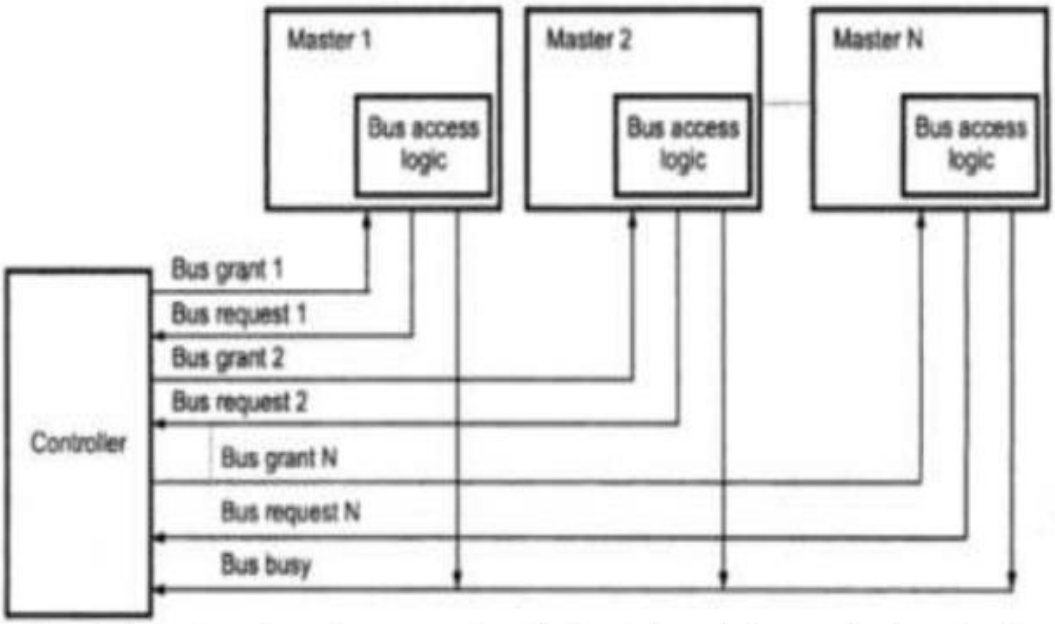
b) Polling method



The system connections for polling method are shown in figure above.

In this the controller is used to generate the addresses for the master. Number of address line required depends on the number of master connected in the system. For example, if there are 8 masters connected in the system, at least three address lines are required. In response to the bus request controller generates a sequence of master address. When the requesting master recognizes its address, it activated the busy line ad begins to use the bus.

C. **Independent request**



The figure below shows the system connections for the independent request scheme. In this scheme each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it. The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

6c Explain the concept of vectored interrupt.

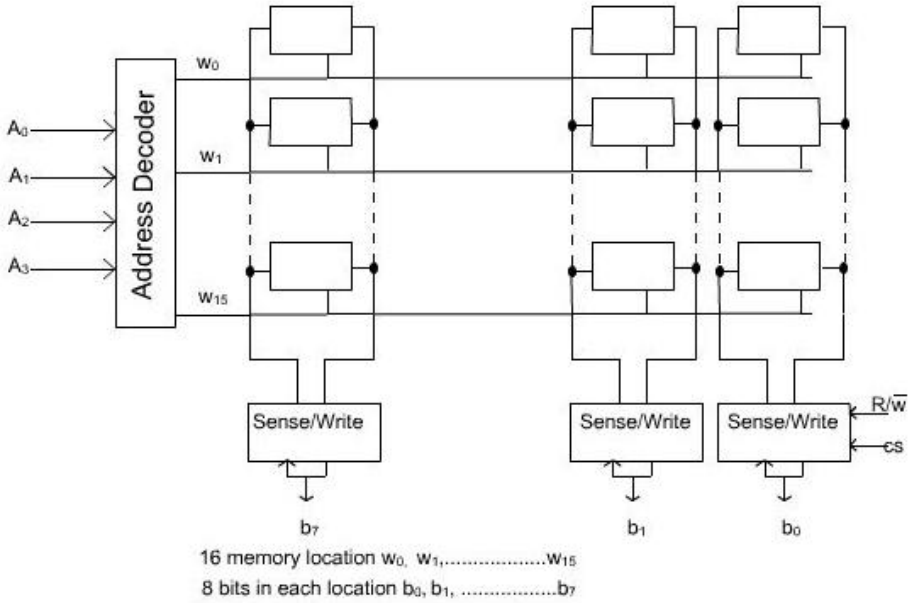
Vectored Interrupts

- A device requesting an interrupt can identify itself by sending a special code to the processor over the bus.
- Interrupt vector
- Avoid bus collision

7.a. Explain internal organization of 16X8 memory chips.

Internal Organization of Memory Chips

A memory cell is capable of storing 1-bit of information. A number of memory cells are organized in the form of a matrix to form the memory chip.

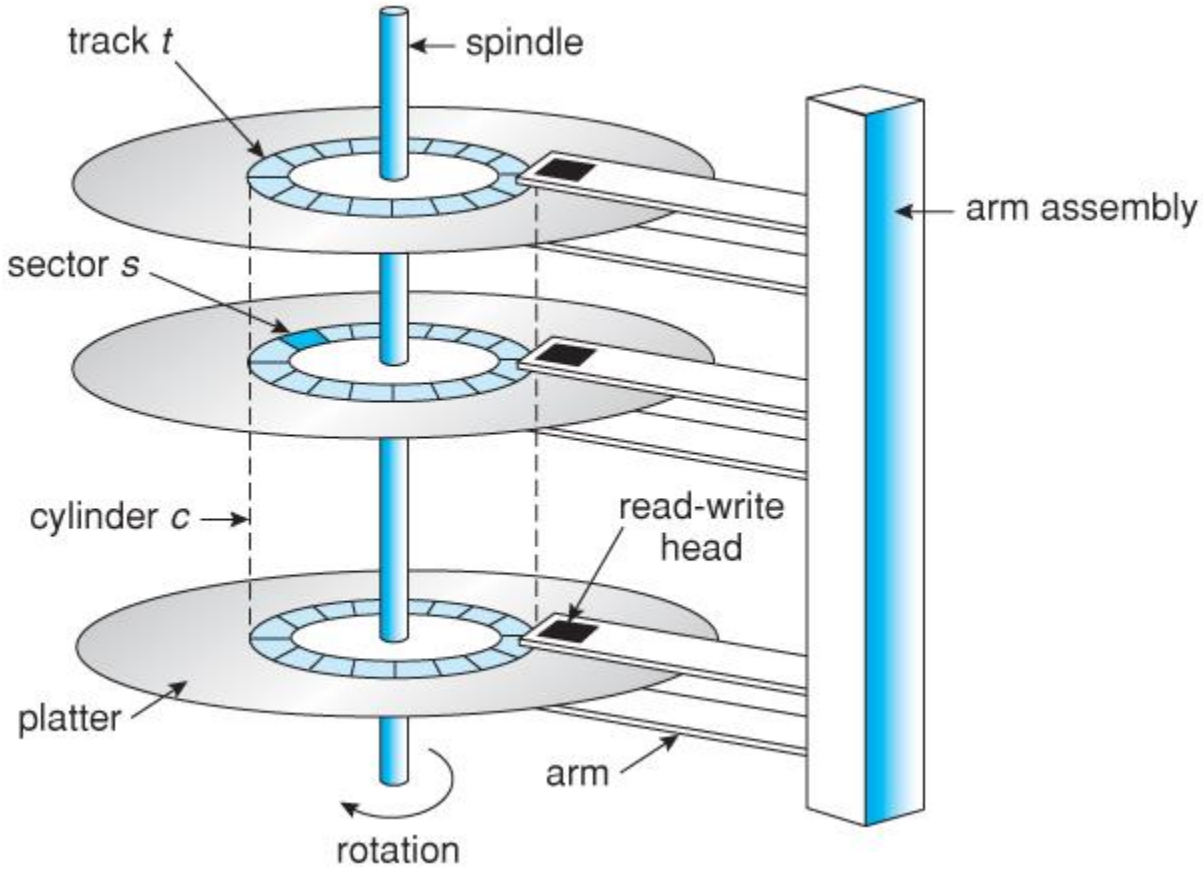


Each row of cells constitutes a memory word, and all cell of a row are connected to a common line which is referred as word line. An address decoder is used to drive the word line. At a particular instant, one word line is enabled depending on the address present in the address bus. The cells in each column are connected by two lines. These are known as bit lines. These bit lines are connected to data input line and data output line through a Sense/Write circuit. During a Read operation, the Sense/Write circuit sense, or read the information stored in the cells selected by a word line and transmit this information to the output data line. During a write operation, the sense/write circuit receive information and store it in the cells of the selected word.

A memory chip consisting of 16 words of 8 bits each, usually referred to as 16 x 8 organization. The data input and data output line of each Sense/Write circuit are connected to a single bidirectional data line in order to reduce the pin required. For 16 words, we need an address bus of size 4. In addition to address and data lines, two control lines, and CS, are provided. The line is to used to specify the required operation about read or write. The CS (Chip Select) line is required to select a given chip in a multi chip memory system.

Consider a slightly larger memory unit that has 1K (1024) memory cells…
128 x 8 memory chips:If it is organised as a 128 x 8 memory chips, then it has got 128 memory words of size 8 bits. So the size of data bus is 8 bits and the size of address bus is 7 bits (2^7=128). The storage organization of 128 x 8 memory chip is shown in the figure 3.6.

7b. With a neat diagram, explain the working principle of magnetic disk.

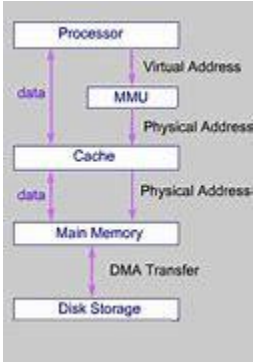


The **disk** has two concentric tracks, each one has various division as sectors for. Information keeps storing on the **disk** in the form of magnetized regions within the sectors. The read/write head of **disk** drive accesses the information by sensing the **magnetic** field. When data allows to write to the **disk**, then read/write head creates a **magnetic** field.

7c. With a neat diagram explain virtual memory organization.\
Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available.

Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main

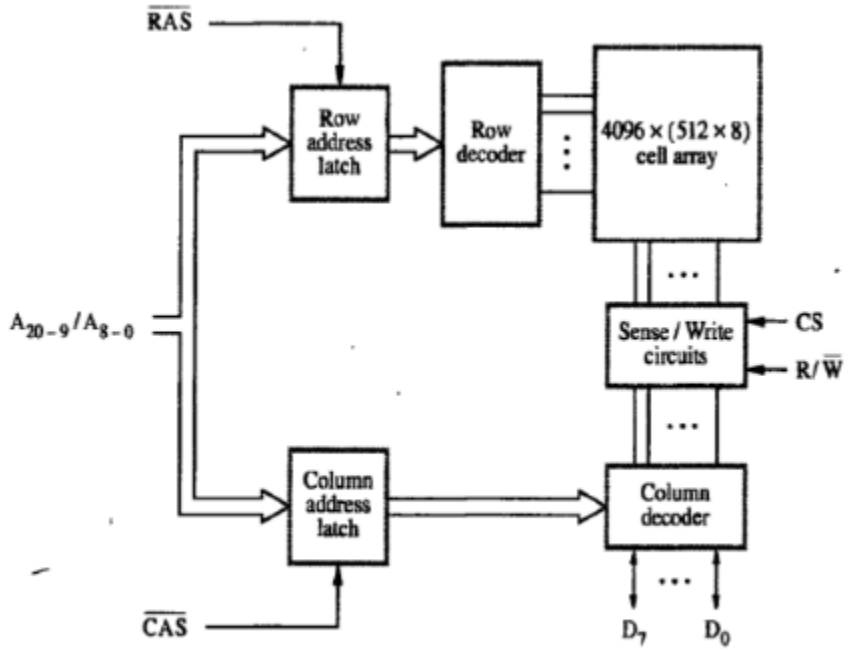
memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.



8a. Explain the internal organization of 2MX8 DRAM memory chip with a neat diagram.

With a neat diagram, explain the organization of 2M X 8 dynamic memory chip. Organized as 4kx4k array. 4096 cells in each row are divided into 512 groups of 8. Each row can store 512 bytes. 12 bits to select a row, and 9 bits to select a group of 8 bits in a row. Total of 21 bits. (2 MB). Reduce the number of bits by multiplexing row and column addresses. First apply the row address, RAS signal latches the row address. Then apply the column address, CAS signal latches the address. Timing of the memory unit is controlled by a specialized unit which generates RAS and CAS. This is asynchronous DRAM. - All the contents of a row are selected based on a row address.Particular byte is selected based on the column address. - Add a latch at the output of the sense circuits in each row. All the latches are loaded when the row is selected. Different column addresses can be applied to select and place different bytes on the data lines. - Consecutive sequence of column addresses can be applied under the control signal CAS, without reselecting the row. Allows a block of data to be transferred at a much faster rate than random accesses. A small collection/group of

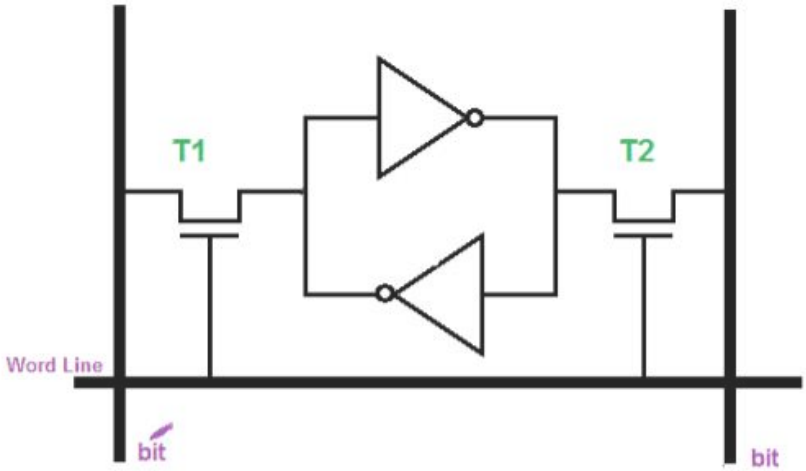
bytes is usually referred to as a block. This transfer capability is referred to as the fast page mode feature.



8b. Explain a static RAM cell with a neat diagram.

Static Random Access Memory cell is designed with two inverters, which are cross-linked like as latch form. This latch is made connection to two bit line along with two transistors T1 and T2. Now both transistors are capable to alter their modes (open or close) under control of word line, and this entire process is controlled by address decoder. When word line goes to ground level then both transistors get turned off, and latch starts to retain own state.

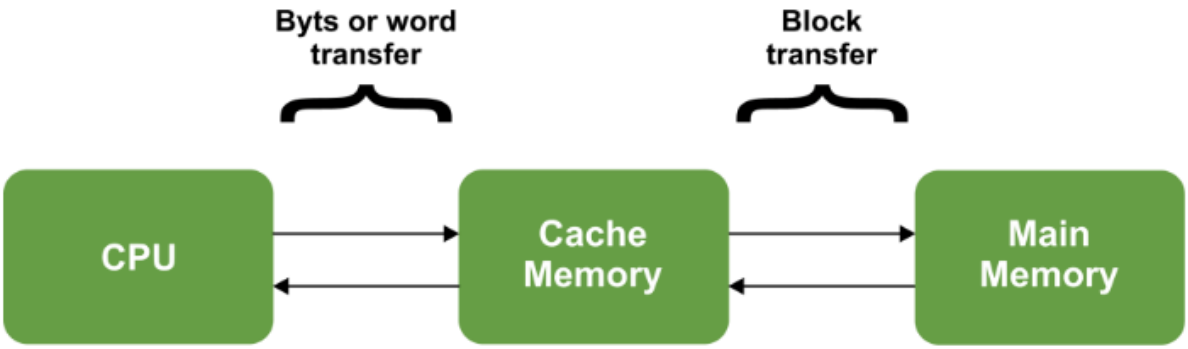
SRAM Circuit Diagram



8c. Discuss the concept of cache memory.

The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time. Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory, then the CPU moves into the main memory.

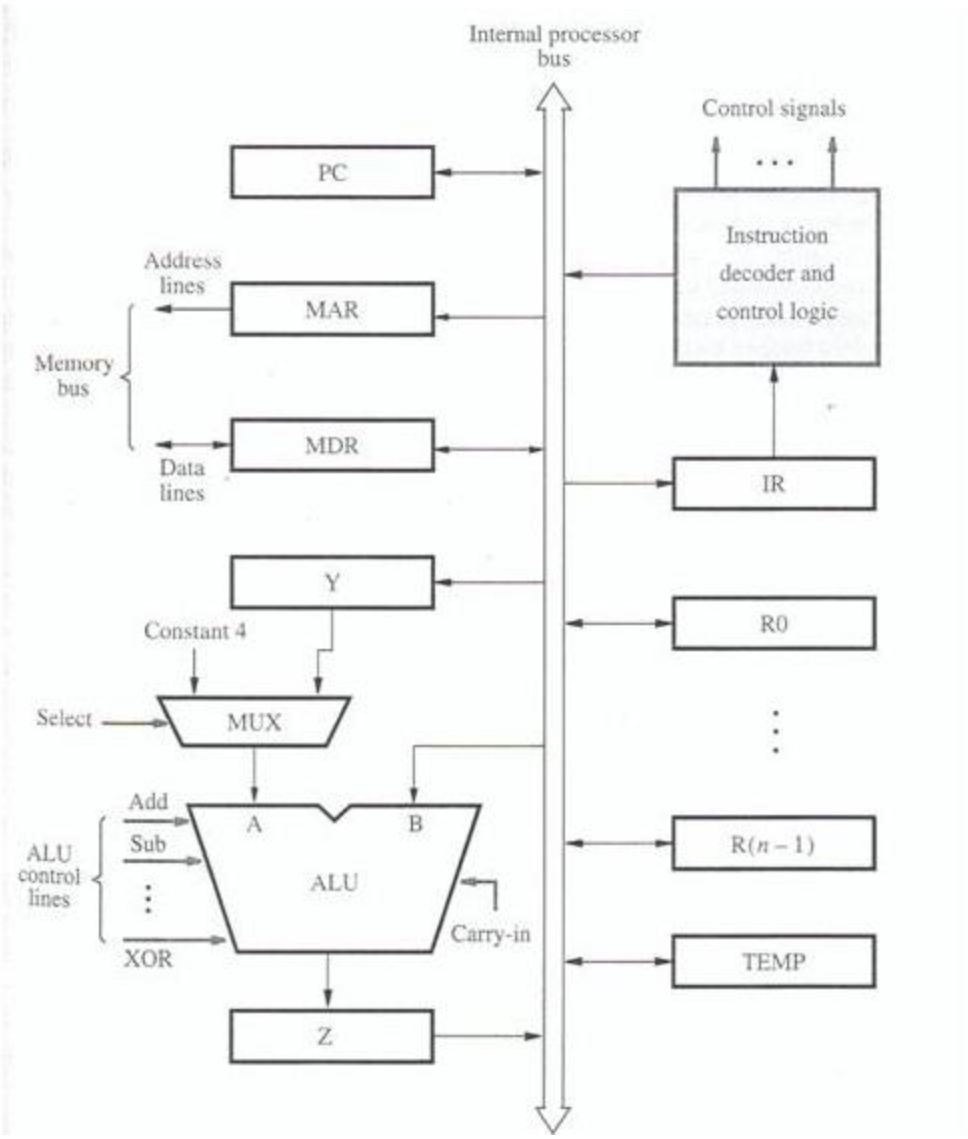
Cache memory is placed between the CPU and the main memory. The block diagram for a cache memory can be represented as:



Cache Memory

9a. Explain with the help of neat diagram single bus organization of a data path inside a processor.

Single-bus Organization of the Datapath inside CPU [1]



The operation or task that must perform by CPU are:
Fetch Instruction: The CPU reads an instruction from memory.

Interpret Instruction: The instruction is decoded to determine what action is required.

Fetch Data: The execution of an instruction may require reading data from memory or I/O module.

Process data: The execution of an instruction may require performing some arithmetic or logical operation on data.

Write data: The result of an execution may require writing data to memory or an I/O module.

To do these tasks, it should be clear that the CPU needs to store some data temporarily. It must remember the location of the last instruction so that it can know where to get the next instruction. It needs to store instructions and data temporarily while an instruction is begin executed. In other words, the CPU needs a small internal memory. These storage location are generally referred as registers.

The major components of the CPU are an arithmetic and logic unit (ALU) and a control unit (CU). The ALU does the actual computation or processing of data. The CU controls the movement of data and instruction into and out of the CPU and controls the operation of the ALU.

The CPU is connected to the rest of the system through system bus. Through system bus, data or information gets transferred between the CPU and the other component of the system. The system bus may have three components:

Data Bus:

Data bus is used to transfer the data between main memory and CPU.

Address Bus:

Address bus is used to access a particular memory location by putting the address of the memory location.

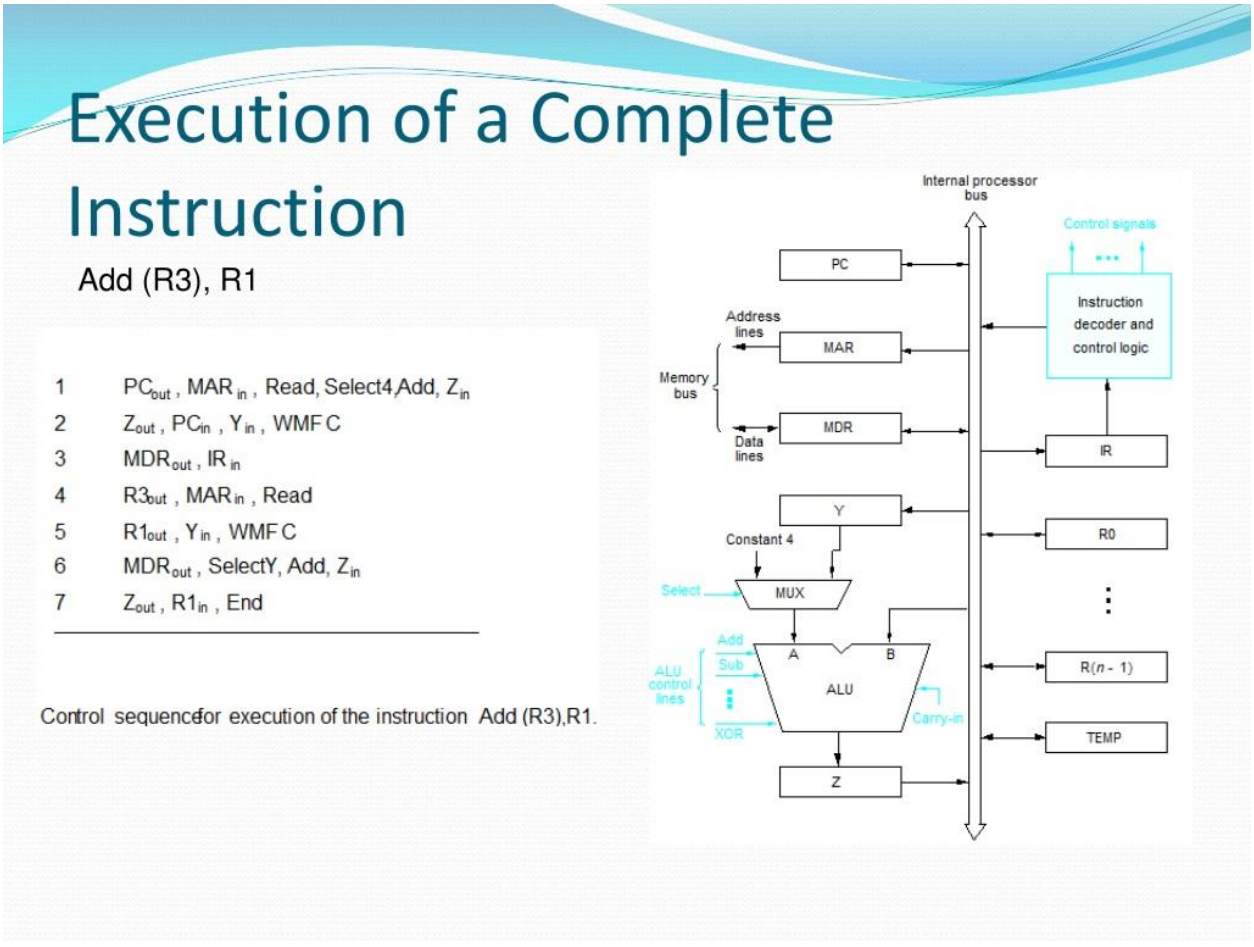
Control Bus:

Control bus is used to provide the different control signal generated by CPU to different

part of the system. As for example, memory read is a signal generated by CPU to indicate that a memory read operation has to be performed. Through control bus this signal is transferred to memory module to indicate the required operation.

There are three basic components of CPU: register bank, ALU and Control Unit. There are several data movements between these units and for that an internal CPU bus is used. Internal CPU bus is needed to transfer data between the various registers and the ALU.

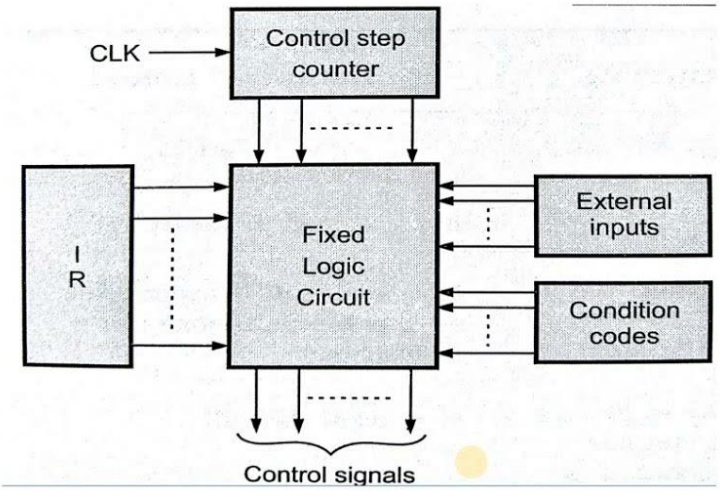
9.b. Discuss the control sequence for the execution of the instruction ADD (R3),R1.



9c. Describe the organization of hardwired control.

HARDWIRED CONTROL UNIT

Diagram shows the typical hardwired control unit.



Fundamentals of Hardwired control

controller of a hardwired control unit is viewed as a sequential logic circuit that implements microoperations using a combinational circuit.

The inputs are from the microperation counter, instruction register and status register, which are transformed into a set of outputs that are the control signals.

10. A. Explain multiple bus organizations with a neat diagram.

Multiple Bus Organization

Multiple bus organization in computer architecture is a design that allows multiple devices to work simultaneously. This reduces the time spent waiting and improves the computer's speed. The main advantage of multiple bus organization is the reduction in the number of cycles required for execution.

In a multiple bus structure, one bus is used to fetch instructions and the other is used to fetch data. The same bus is shared by three units: memory, processor, and I/O units.

In a multiple bus system, each processor-memory pair is linked by various redundant paths. This means that the failure of one or more paths can be tolerated, but it will degrade system performance.

The main reason for having multiple buses in a computer design is to improve performance. Other advantages include:

- Better connectivity
- An increase in the size of the registers

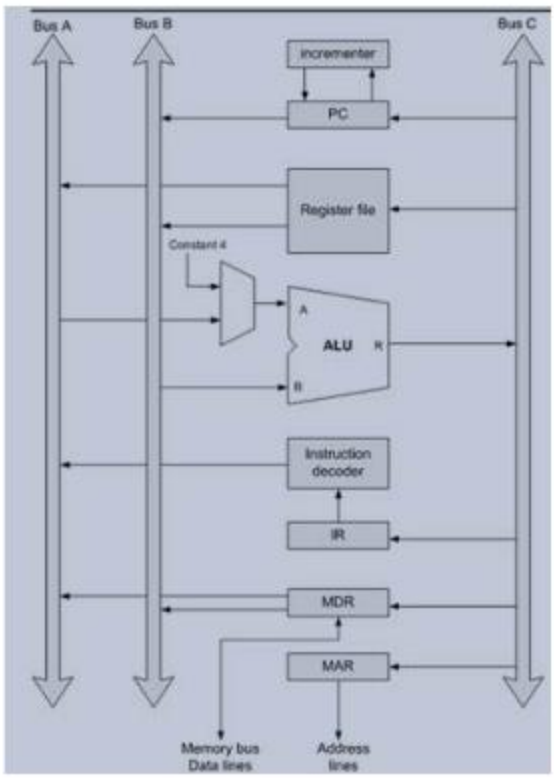
There are three types of bus lines: data bus, address bus, and control bus. Communication over each bus line is performed in cooperation with another.

Three bus organization of data path

In single bus organization, only one data item can be transferred over the bus in a clock cycle. To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.

Figure illustrates a three-bus structure used to connect the registers and the ALU of a processor. All general-purpose registers are combined into a single block called the register file. The register file in Figure is said to have three ports. There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The third port allows the data on bus C to be loaded into a third register during the same clock cycle.

Buses A and B are used to transfer the source operands to the A and B inputs of the ALU, where an arithmetic or logic operation may be performed. The result is transferred to the destination over bus C. If needed, the ALU may simply pass one of its two input operands unmodified to bus C. We will call the ALU control signals for such an operation R=A or R=B. A second feature in Figure is the introduction of the Incrementer unit, which is used to increment the PC by 4. Using the Incrementer eliminates the need to add 4 to the PC using the main ALD, as was done in single bus organization. The source for the constant 4 at the ALU input multiplexer is still useful.



Action:

1. *PCout, R=B, MARin, Read, IncPC*
2. *WFMC*
3. *MDRoutB, R=B, Irin*
4. *R4out, R5outB, SelectA, Add, R6in, End.*

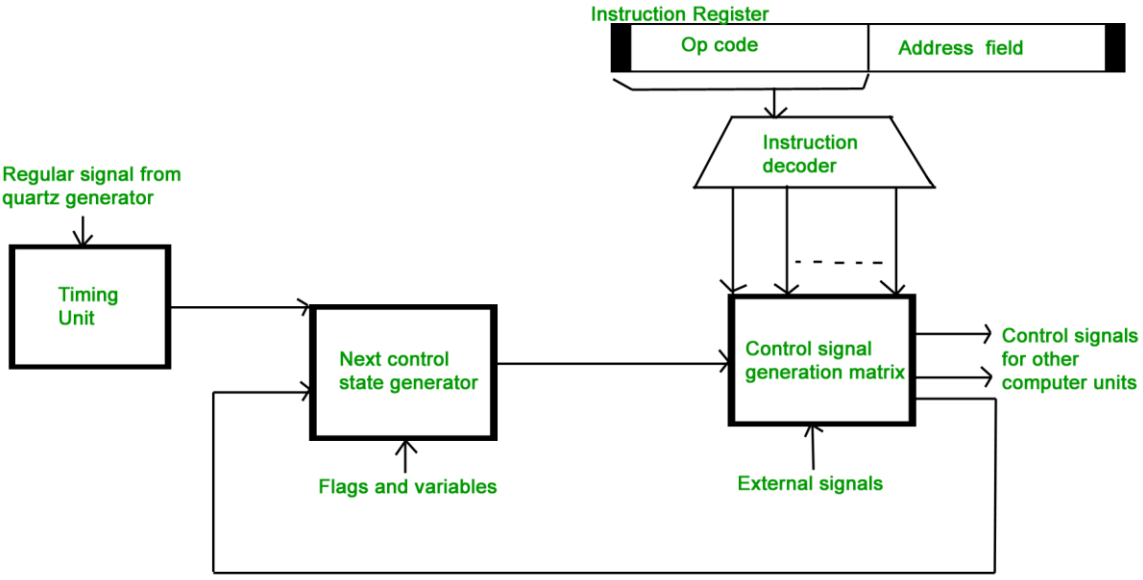
Consider the three-operand instruction **Add R4,R5,R6:**

The control sequence for executing this instruction is given in Figure 2.8.

In step 1, the contents of the PC are passed through the ALU, using the R=B control signal, and loaded into the MAR to start a memory read operation. At the same time the PC is incremented by 4. Note that the value loaded into MAR is the original contents of the PC. The incremented value is loaded into the PC at the end of the clock cycle and will not affect the contents of MAR.

In step 2, the processor waits for MFC and loads the data received into MDR, then transfers them to IR in step 3. Finally, the execution phase of the instruction requires only one control step to complete, step 4. By providing more paths for data transfer a significant reduction in the number of clock cycles needed to execute an instruction is achieved.

10.b What is microprogrammed control ? Explain the organization with suitable diagrams and examples.



A micro-programmed control unit can be described as a simple logic circuit. We can use it in two ways, i.e., it is able to execute each instruction with the help of generating control signals, and it is also able to do sequencing through microinstructions. It will generate the control signals with the help of programs. At the time of evolution of CISC architecture in the past, this approach was very famous. The program which is used to create the control signals is known as the "Micro-program". The micro-program is placed on the

processor chip, which is a type of fast memory. This memory is also known as the control store or control memory.

A micro-program is used to contain a set of microinstructions. Each microinstruction or control word contains different bit patterns. The n bit words are contained by each microinstruction. On the basis of the bit pattern of a control word, every control signals differ from each other.

Like the above, the instruction execution in a micro-programmed control unit is also performed in steps. So for each step, the micro-program contains a control word/ microinstruction. If we want to execute a particular instruction, we need a sequence of microinstructions. This process is known as the micro-routine. The image of a micro-programmed control unit is described as follows. Here, we will learn the organization of micro-program, micro-routine, and control word/ microinstruction.