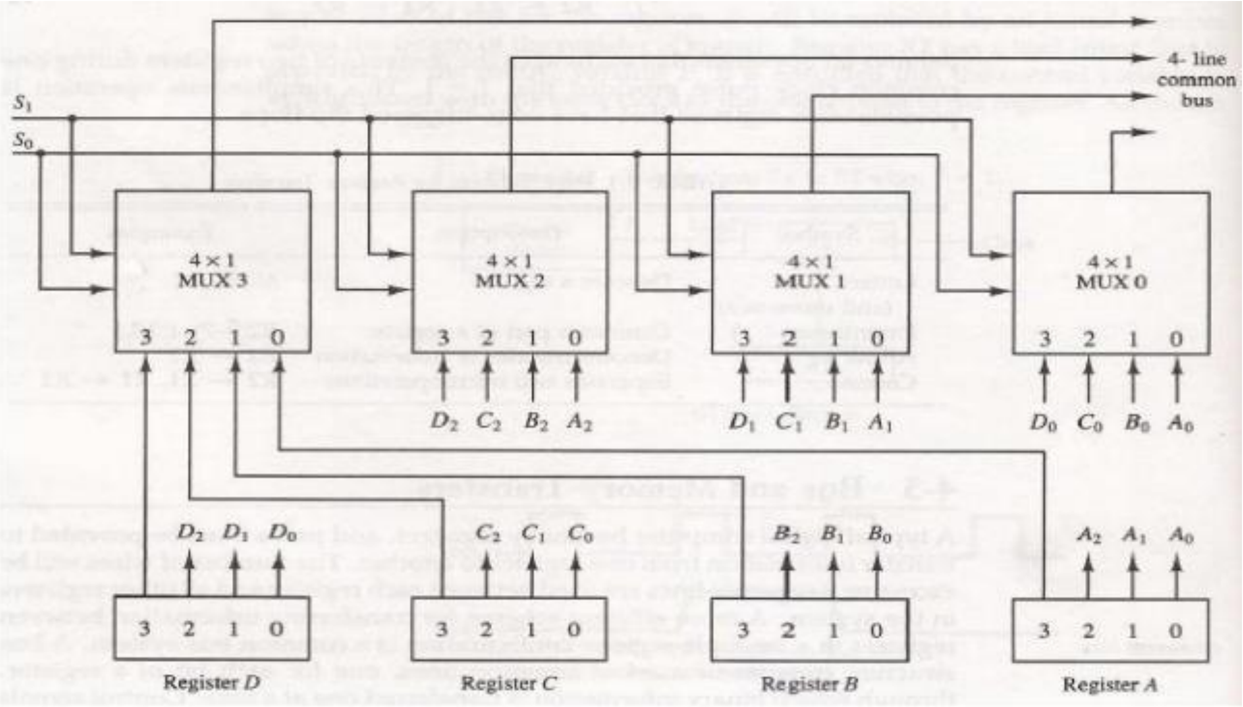
Unit 1

1)**Define register transfer. With neat diagram explain the implementation of register transfer using BUS?**

A) Register transfer:

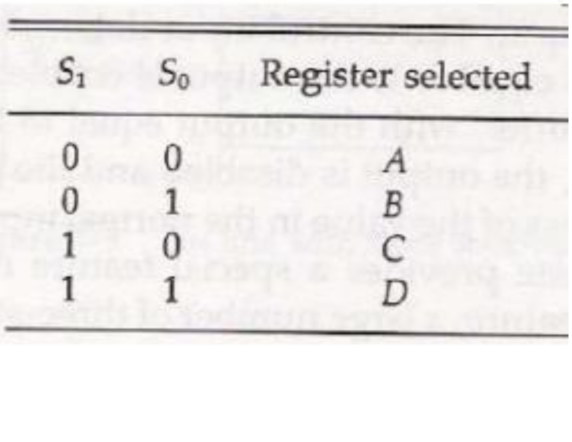
Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

- The statement **R2← R1** denotes a transfer of the content of register **R1** into register **R2**. It designates a replacement of the content of **R2** by the content of **R1** the content of the source register **R1** does not change after the transfer. If we want the transfer to occur only under a predetermined control condition then it can be shown by an if-then statement.
- if **(P=1) then R2← R1** P is the control signal generated by a control section.We can separate the control variables from the register transfer operation by specifying a Control Function. Control function is a Boolean variable that is equal to 0 or 1. control function is included in the statement as **P: R2← R1** Control condition is terminated by a colon implies transfer operation be executed by the hardware only if **P=1** Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



The two selection lines Si and So are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus. When S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

Similarly, register B is selected if S1S0 =01, and so on.



S_1	S_0	Register selected
0	0	*A*
0	1	*B*
1	0	*C*
1	1	*D*

2)**Explain Binary adder-Subtractor**

A)

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- A 4-bit adder-subtractor circuit is shown in Fig

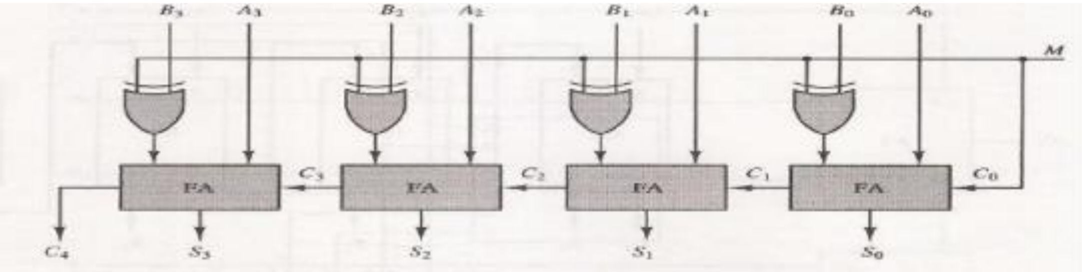


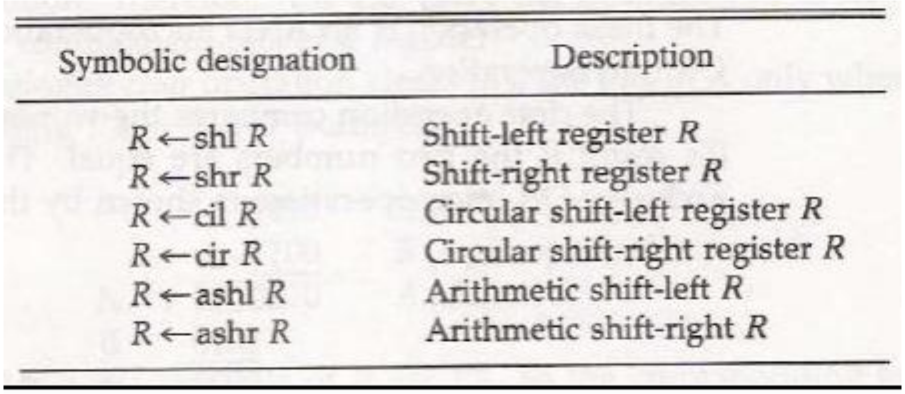
Figure 4-7 4-bit adder-subtractor.

- The mode input M controls the operation. When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.
- Each exclusive-OR gate receives input M and one of the inputs of B
- When M = 0, we have B xor 0 = B. The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
- When M = 1, we have B xor 1 = B' and Co = 1.
- The B inputs are all complemented and a 1 is added through the input carry.
- The circuit performs the operation A plus the 2's complement of B.

3)**Define Shift Operation .List various of Shift operations. Explain the implementation of shl &shr operation?**

A)

- Shift microoperations are used for serial transfer of data.
- The contents of a register can be shifted to the left or the right
- During a shift-left operation the serial input transfers a bit into the rightmost position.
- During a shift-right operation the serial input transfers a bit into the leftmost position.
- There are three types of shifts: logical, circular, and arithmetic.



Symbolic designation	Description
$R \leftarrow \mathrm{shl}\ R$	Shift-left register R
$R \leftarrow \mathrm{shr}\ R$	Shift-right register R
$R \leftarrow \mathrm{cil}\ R$	Circular shift-left register R
$R \leftarrow \mathrm{cir}\ R$	Circular shift-right register R
$R \leftarrow \mathrm{ashl}\ R$	Arithmetic shift-left R
$R \leftarrow \mathrm{ashr}\ R$	Arithmetic shift-right R

- **Logical Shift**: A logical shift is one that transfers 0 through the serial input. The symbols shl and shr for logical shift-left and shift-right microoperations.
- **Circular Shift**: The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by

connecting the serial output of the shift register to its serial input

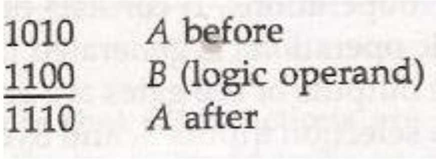
- **Arithmetic Shift**: An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2.

4)with examples explain various applications of logical operations

Logic micro-operations are very useful for manipulating individual bits or a portion of a word stored in a register.

Selective set

- The selective-set operation sets to 1 the bits in register A where there are corresponding l's in register B. It does not affect bit positions that have 0's in B. The following numerical example clarifies this operation:

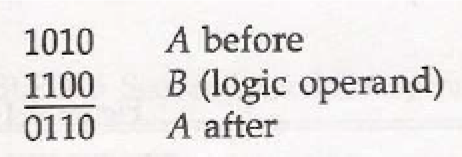


1010	*A* before
1100	*B* (logic operand)
1110	*A* after

- The OR microoperation can be used to selectively set bits of a register.

Selective complement

- The selective-complement operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in B. For example:

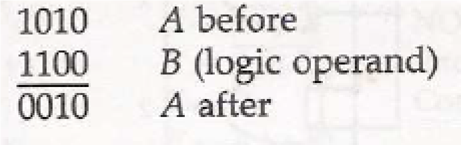


1010	*A* before
1100	*B* (logic operand)
0110	*A* after

- The exclusive-OR microoperation can be used to selectively complement bits of a register.

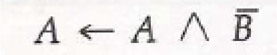
Selective clear

- The selective-clear operation clears to 0 the bits in A only where there are corresponding l's in B. For example:



1010	*A* before
1100	*B* (logic operand)
0010	*A* after

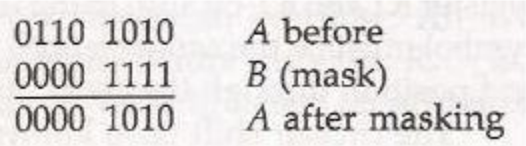
- The corresponding logic microoperation is



$$A \leftarrow A \wedge \overline{B}$$

Mask

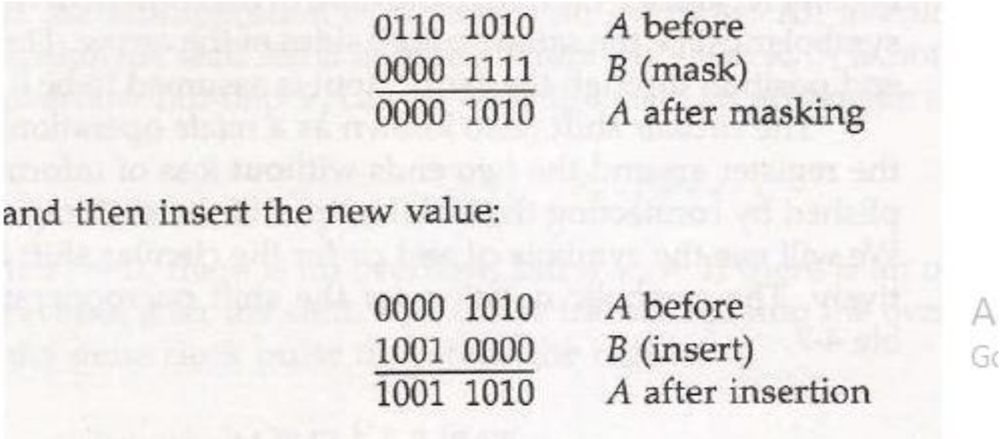
- The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding O's in B . The mask operation is an AND micro operation as seen from the following numerical example:



0110 1010	*A* before
0000 1111	*B* (mask)
0000 1010	*A* after masking

Insert

- The insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value.
- For example, suppose that an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits:



0110 1010	*A* before
0000 1111	*B* (mask)
0000 1010	*A* after masking

and then insert the new value:

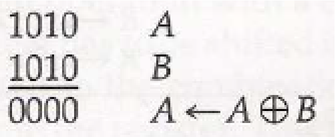
0000 1010	*A* before
1001 0000	*B* (insert)
1001 1010	*A* after insertion

A
Gc

- The mask operation is an AND microoperation and the insert operation is an OR microoperation.

Clear

- The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal. This operation is achieved by an exclusive-OR microoperation as shown by the following example



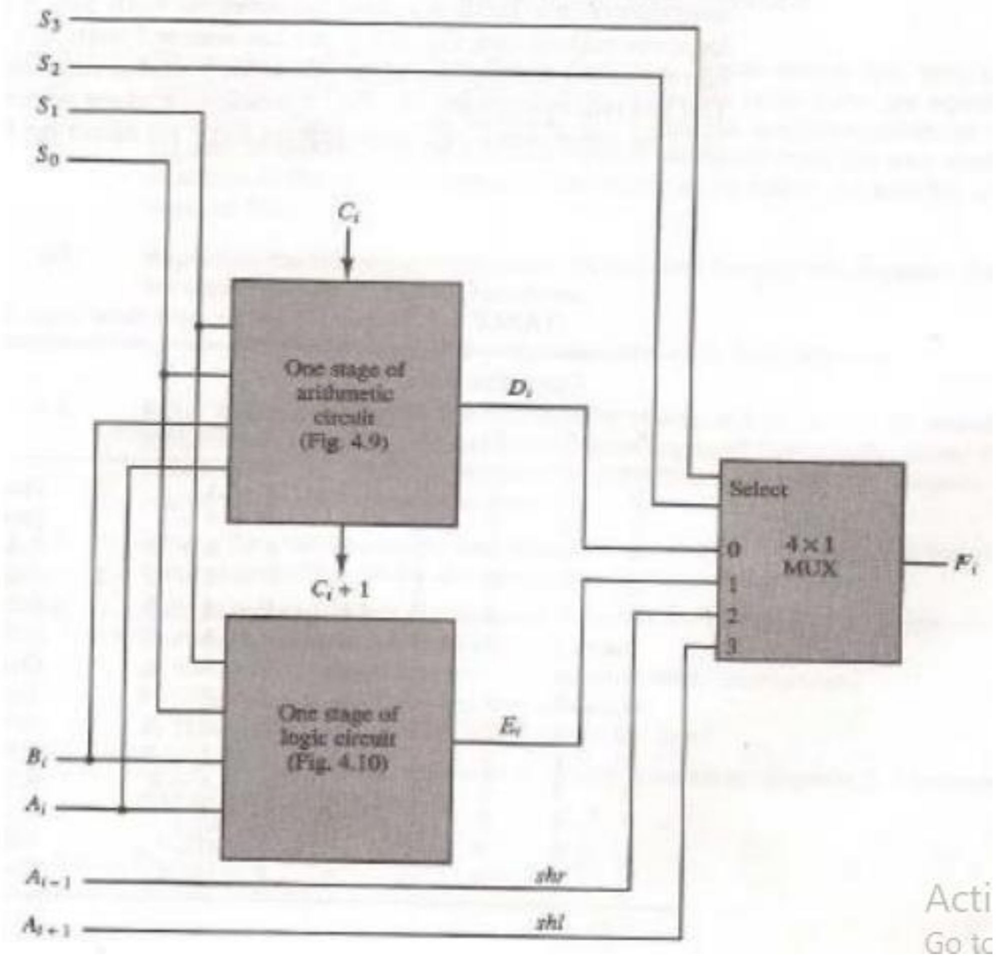
1010	A
1010	B
0000	$A \leftarrow A \oplus B$

5)**With neat diagram illustrate implementation of ALU**

A)

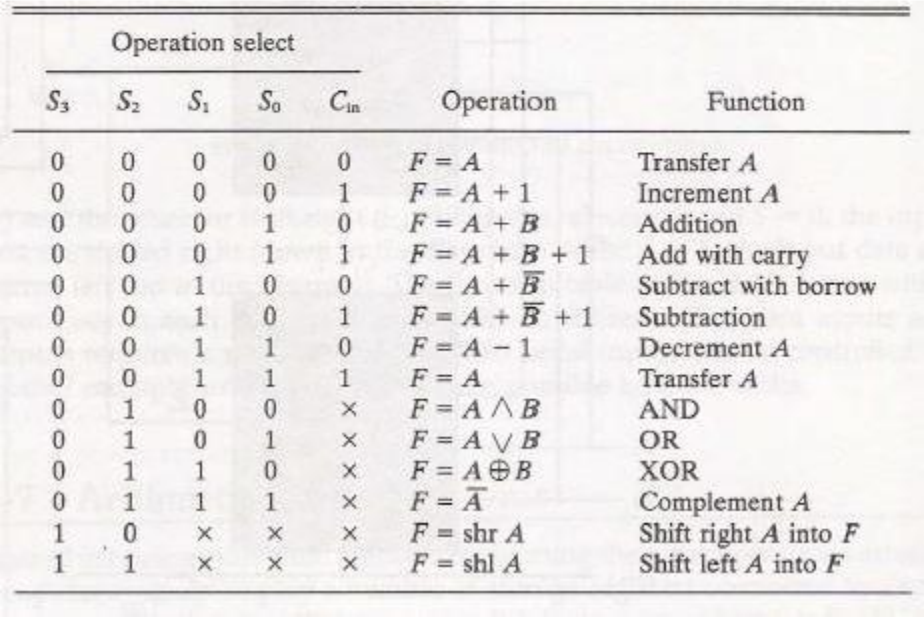
- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.
- The shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.
- Particular microoperation is selected with inputs S1 and S0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in Di and a logic output in E
- The data in the multiplexer are selected with inputs S3 and S2. The other two data inputs to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai+1 for the shift-left operation. ¬ The circuit whose one stage is specified in Fig provides eight arithmetic operation, four logic operations, and two shift operations

Figure 4-13 One stage of arithmetic logic shift unit.



- Each operation is selected with the five variables S3, S2, S1, S0 and C_{in}
- The input carry C_{in} is used for selecting an arithmetic operation only
- Table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S3S2 = 00. The next four are logic and are selected with S3S2 = 01. The input carry has no effect during the logic operations and is marked with don't-care x’s. The last two operations

are shift operations and are selected with S3S2= 10 and 11. The other three selection inputs have no effect on the shift.



Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F

Unit 2

1. Illustrate the implementation of common bus in general purpose Computer.

Ans.In common bus there are ‘8’ registers of different type and ‘1’ memory unit .

11 0

PC

Program counter (12) holds address of instruction (that should be performed next)

11 0

AR

Address Register (12) holds address of memory.

15 0

IR

Instruction Register (16) holds instruction codes.

15 0

TR

Temporary Register (16) holds temporary data.

15 0

DR

Data Register (16) holds memory operand.

15 0

AC

Accumulator (16) Process Register (Stores the result of operation performed)

7 0

OUTR

Output Register (8) holds output character.

7 0

INPR

Input Register (8) holds input character.

Memory unit (4096 words, 16 bits per word)

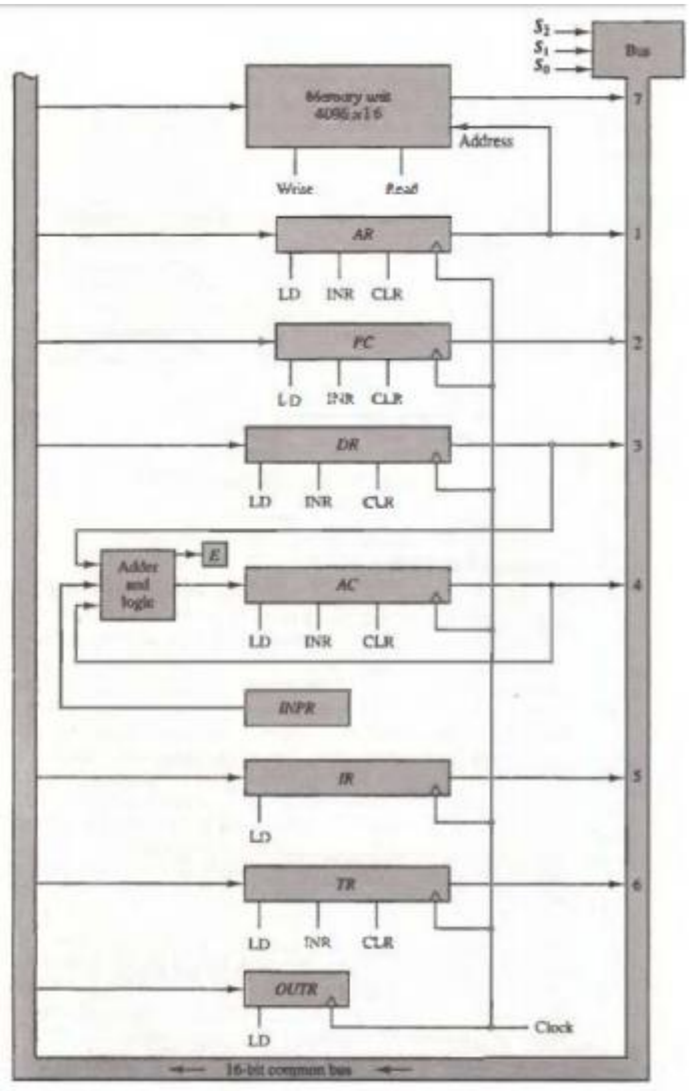


Figure 5-4 Basic computer registers connected to a common bus.

Connections:

The outputs of all the registers except the OUTR (output register) are connected to the common bus. The output selected depends upon the binary value of variables S2, S1 and S0. The lines from common bus are connected to the inputs of the registers and memory. A register receives the information from the bus when its LD (load) input is activated while in case of memory the Write input must be enabled to receive the information. The contents of memory are placed onto the bus when its Read input is activated.

Various Registers:

4 registers DR, AC, IR and TR have 16 bits and 2 registers AR and PC have 12 bits. The INPR and OUTR have 8 bits each. The INPR receives character from input device and delivers it to the AC while the OUTR receives character from AC and transfers it to the output device. 5 registers have 3 control inputs LD (load), INR (increment) and CLR (clear). These types of registers are similar to a binary counter.

Adder and logic circuit:

The adder and logic circuit provides the 16 inputs of AC. This circuit has 3 sets of inputs. One set comes from the outputs of AC which implements register micro operations. The other set comes from the DR (data register) which are used to perform arithmetic and logic micro operations. The result of these operations is sent to AC while the end around carry is stored in E as shown in diagram. The third set of inputs is from INPR.

Note:

The content of any register can be placed on the common bus and an operation can be performed in the adder and logic circuit during the same clock cycle.

2. **Discuss Timing & Control of a general-purpose computer.**

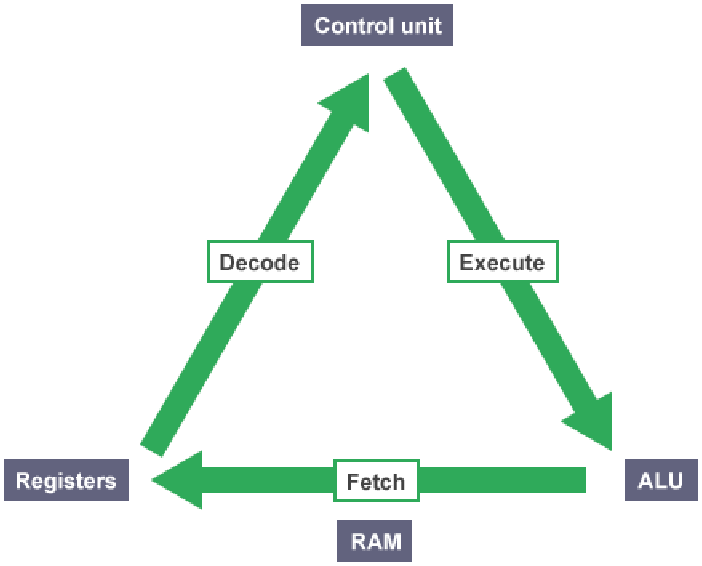
Ans.Timing an control unit are the fundamentals of any computer system. One of the primary work of control unit is to send control signals to system component. Computer programs are number of sequential instructions.The processor are fed with these instruction.Instruction are made of two things opcode and address. Opcode defines the operation to be performed and address defines the place in the memory where the operands are located. For each instruction the CPU needs a certain time to process it known as clock. The operands located in the memory needs to be fetched and brought to the CPU with the help of bus.Similarly processor needs to keep track of next instruction to be executed. All these are managed by controlled unit.The logical operations performed on Gates like AND gate OR gate are also controlled by control unit.

All the calculations are done by ALU unit and management is done by control unit

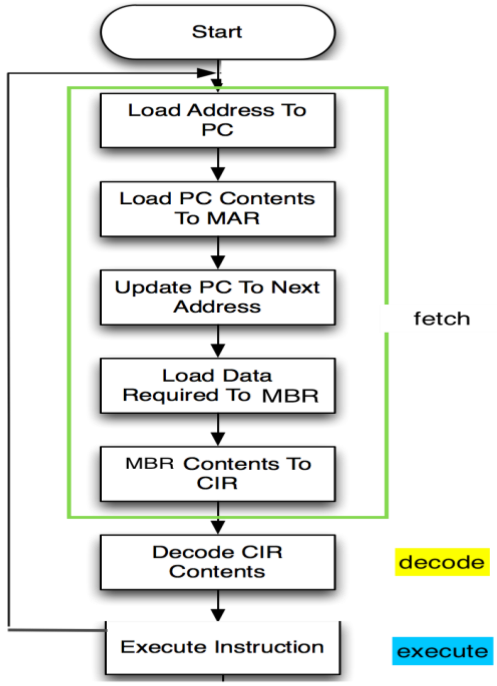
3. **With neat diagram and flow chart illustrate Fetch & Decode phases.**

Ans.

Diagram:


Control unit
Decode
Execute
Registers
Fetch
ALU
RAM

Flowchart:


Start
Load Address To PC
Load PC Contents To MAR
Update PC To Next Address
Load Data Required To MBR
MBR Contents To CIR
fetch
Decode CIR Contents
decode
Execute Instruction
execute

- The processor checks the program counter to see which instruction to run next.
- The program counter gives an address value in the memory of where the next instruction is.
- The processor fetches the instruction value from this memory location.
- Once the instruction has been fetched, it needs to be decoded and executed. For example, this could involve taking one value, putting it into the **ALU**, then taking a different value from a **register** and adding the two together.
- Once this is complete, the processor goes back to the program counter to find the next instruction.
- This cycle is repeated until the program ends.

4. With an example illustrate various microoperations that need to carry in Interrupt cycle.

Ans.

Interrupt cycle:

It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the central processing unit (CPU), from boot up to when the computer is shut down.

The microoperations that need to carry in Interrupt cycle are:

t1 : MBR <= (PC)

t2 : MAR <= Save_Address

PC <= Routine_Address

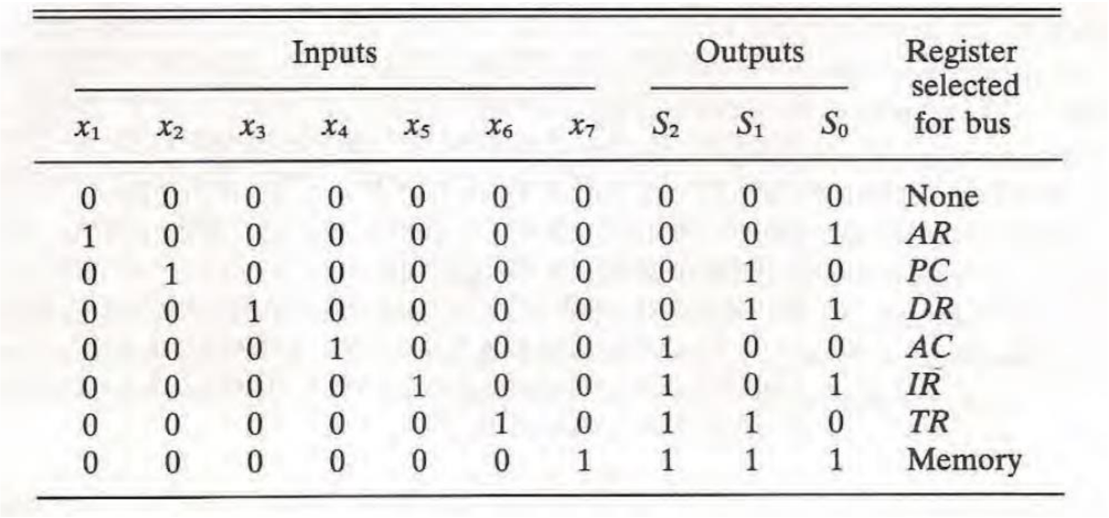
t3: Memory <= (MBR)

In the first step, the contents of the PC are transferred to the MBR, so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may each be a single micro-operation. However, because most processors provide multiple types and/or levels of interrupts, it may lake one or more additional micro-operations to obtain the save_address and the routine_address before they can be transferred to the MAR and PC, respectively. In any case, once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory. The processor is now ready to begin the next instruction cycle.

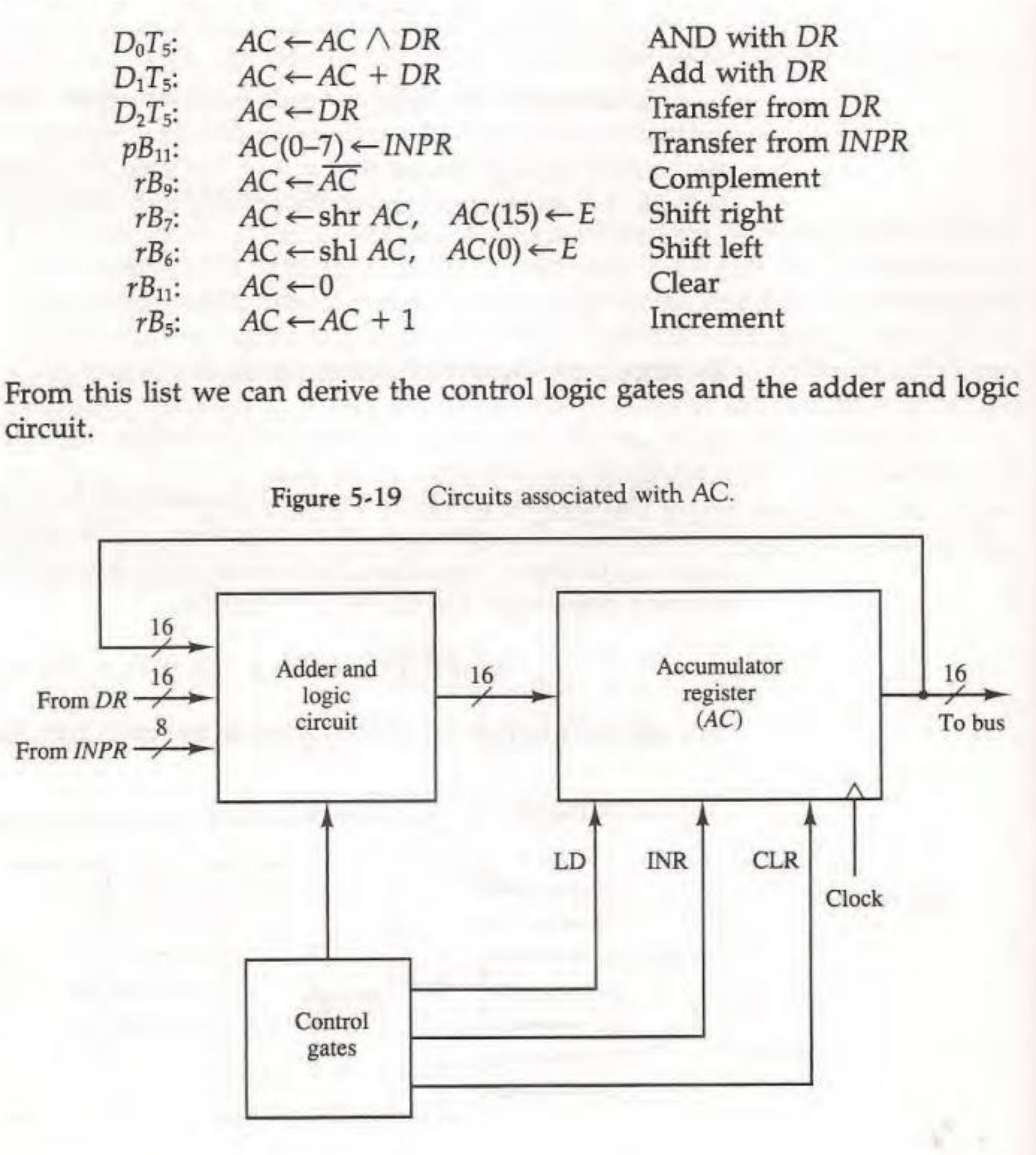
5. Discuss design of Accumulator Logic.

Ans.

The circuits associated with the AC register are shown in below figure. The adder and logic circuit has three sets of inputs. One set of 16 inputs comes from the outputs of AC.Another set of 16 inputs comes from the data register DR. A third set of eight inputs comes from the input register INPR.The outputs of the adder and logic circuit provide the data inputs for the register. In addition, it is necessary to include logic gates for controlling the LD, INR, and CLR in the register and for controlling the operation of the adder and logic circuit.In order to design the logic associated with AC, it is necessary to go over the register transfer statements in below Table and extract all the statements that change the content of AC



Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	*AR*
0	1	0	0	0	0	0	0	1	0	*PC*
0	0	1	0	0	0	0	0	1	1	*DR*
0	0	0	1	0	0	0	1	0	0	*AC*
0	0	0	0	1	0	0	1	0	1	*IR*
0	0	0	0	0	1	0	1	1	0	*TR*
0	0	0	0	0	0	1	1	1	1	Memory



D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with *DR*
D_1T_5:	$AC \leftarrow AC + DR$	Add with *DR*
D_2T_5:	$AC \leftarrow DR$	Transfer from *DR*
pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
rB_9:	$AC \leftarrow \overline{AC}$	Complement
rB_7:	$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment

From this list we can derive the control logic gates and the adder and logic circuit.

Figure 5-19 Circuits associated with AC.

6. Briefly explain address sequencing.

Ans.

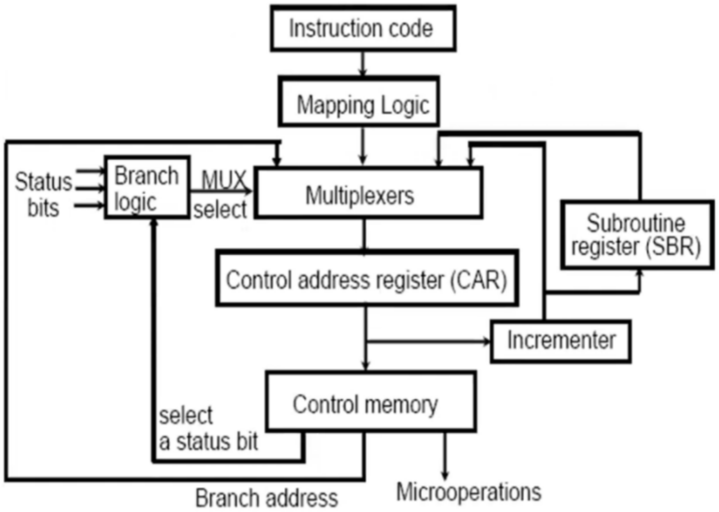
- When computer is switched on, CAR is loaded with an initial address pointing to first microinstruction that activates the fetch routine.

- Fetch routine is sequenced by incrementing the CAR through the rest of micro instructions.
- At end of fetch routine, the instruction is in IR.
- The address of operand needs to be calculated.
- Bits in machine instruction specify various addressing modes.
- Mode bits decide the conditions which choose the branch microinstruction which is used to calculate effective address.
- At end of address calculation routine, the address of operand is in MAR.
- Microinstructions to execute the instruction fetched from memory need to be generated.
- Each instruction has its own microprogram routine stored in a given location of control memory.
- Transformation from the instruction code bits to an address in control memory where the routine is locate is called mapping process.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.
- Once the required routine is reached, the microinstruction that execute the instruction may be sequenced by incrementing the CAR.

- Microprograms that use subroutines require an external register for storing the return address.
- At end of instruction execution, control returns to fetch routine by executing an unconditional branch microinstruction to first address of fetch routine.

Address sequencing capabilities required in control memory are

1. Incrementing of the control address register (CAR)
2. Unconditional and conditional branches
3. A mapping process from the bits of the machine instruction to an address for control memory (CM).
4. A facility for subroutine call and return



Selection of Address for Control Memory

7. **Define Microinstruction. Discuss various fields of microinstruction with an example.**

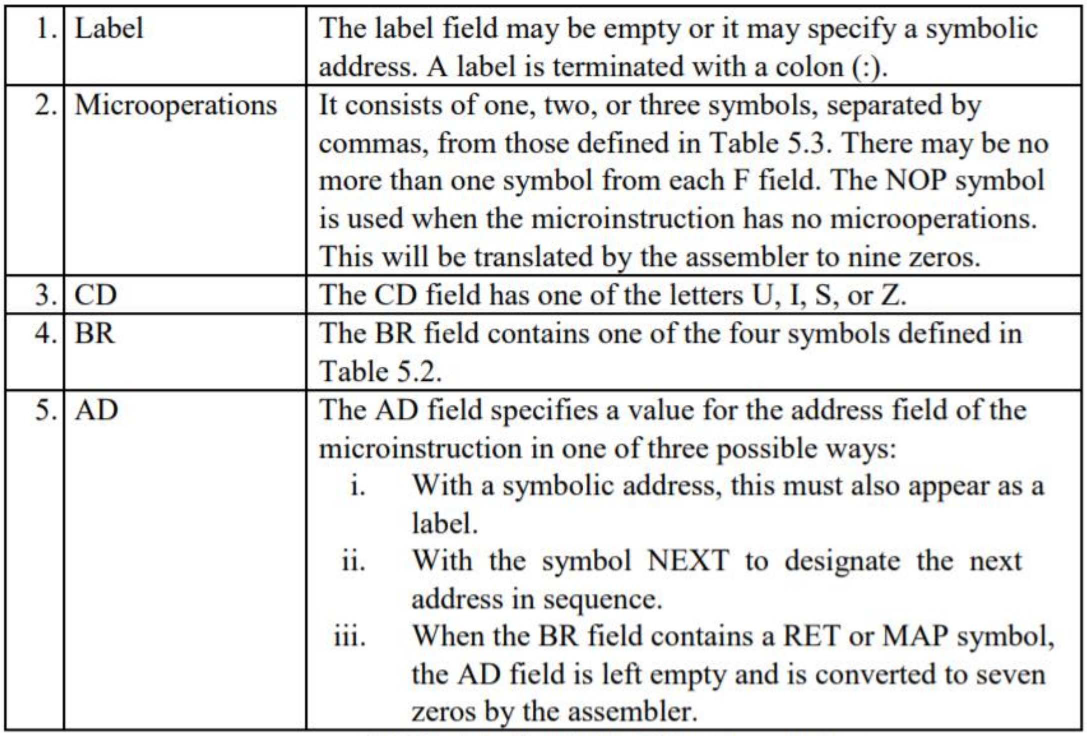
Ans.

Microoperation :

- The operation executed on data stored in registers are called microoperations.
- A microoperation is a elementary operation performed on the information stored in one or more register.
- Examples of microoperations are shift, count, clear and load.

Symbolic Microinstruction. □

- Each line of the assembly language microprogram defines a symbolic microinstruction. □
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. The fields specify the following Table.



1.	Label	The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2.	Microoperations	It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.
3.	CD	The CD field has one of the letters U, I, S, or Z.
4.	BR	The BR field contains one of the four symbols defined in Table 5.2.
5.	AD	The AD field specifies a value for the address field of the microinstruction in one of three possible ways: i. With a symbolic address, this must also appear as a label. ii. With the symbol NEXT to designate the next address in sequence. iii. When the BR field contains a RET or MAP symbol, the AD field is left empty and is converted to seven zeros by the assembler.

Table 4.3: Symbolic Microinstruction

8. Discuss the design of Microprogram sequencer .

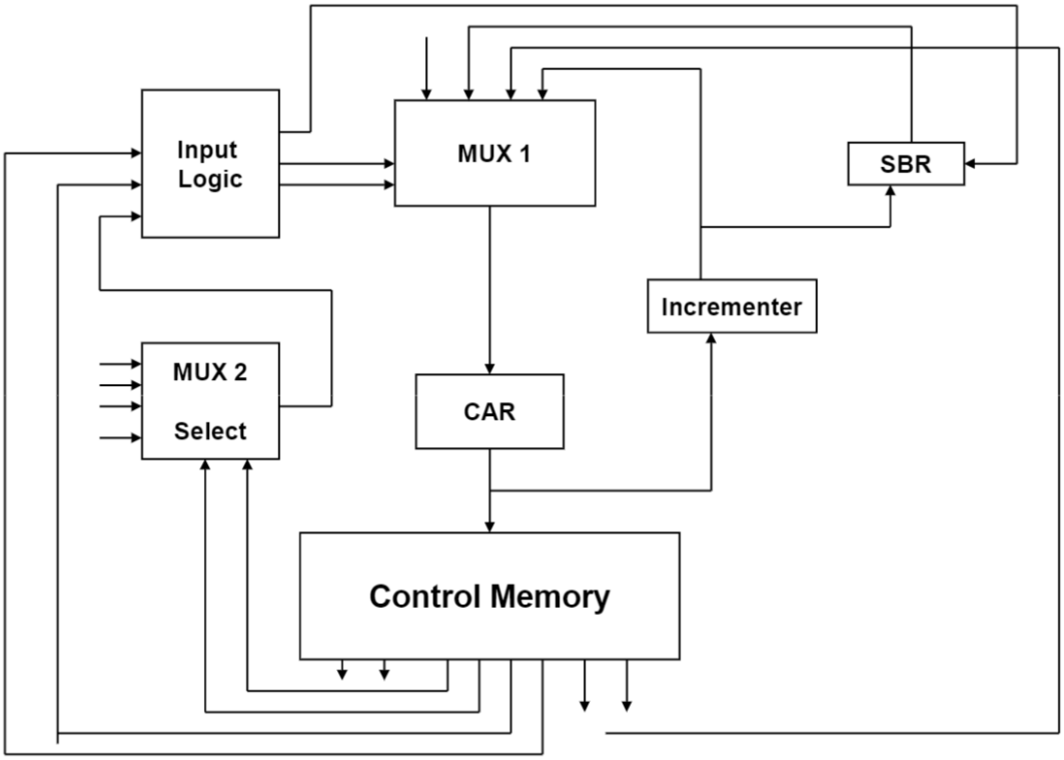
Ans.

- The basic components of microprogrammed control unit are the control memory and the circuits that select the next address.
- The task of Microinstruction sequencing is done by Microprogram sequencer.
- The address selection part is called as microprogram sequencer.
- Microprogram sequencer can be constructed with digital functions to suit a particular application.

Two imp. factors that must be considered while designing the microinstruction sequencer:

1. The size of the microinstruction
2. The address generation time

- The purpose of microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.



Microprogram sequencer for a control Mmemory

Control Address Register(CAR) :

Control address register receives the address from four different paths. For receiving the addresses from four different paths, Multiplexer is used.

Multiplexer :

Multiplexer is a combinational circuit which contains many data inputs and single data output depending on control or select inputs.

Branching :

Branching is achieved by specifying the branch address in one of the fields of the micro instruction. Conditional branching is obtained by using part of the micro-instruction to select a specific status bit in order to determine its condition.

Mapping Logic :

An external address is transferred into control memory via a mapping logic circuit.

Incrementer :

Incrementer increments the content of the control address register by one, to select the next micro-instruction in sequence.

Subroutine Register (SBR) :

The return address for a subroutine is stored in a special register called Subroutine Register whose value is then used when the micro-program wishes to return from the subroutine.

Control Memory :

Control memory is a type of memory which contains addressable storage registers. Data is temporarily stored in control memory. Control memory can be accessed quicker than main memory.

Unit-3

1.Implementations of register stacks and memory stack.

- **Register Stack** A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 3 shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.

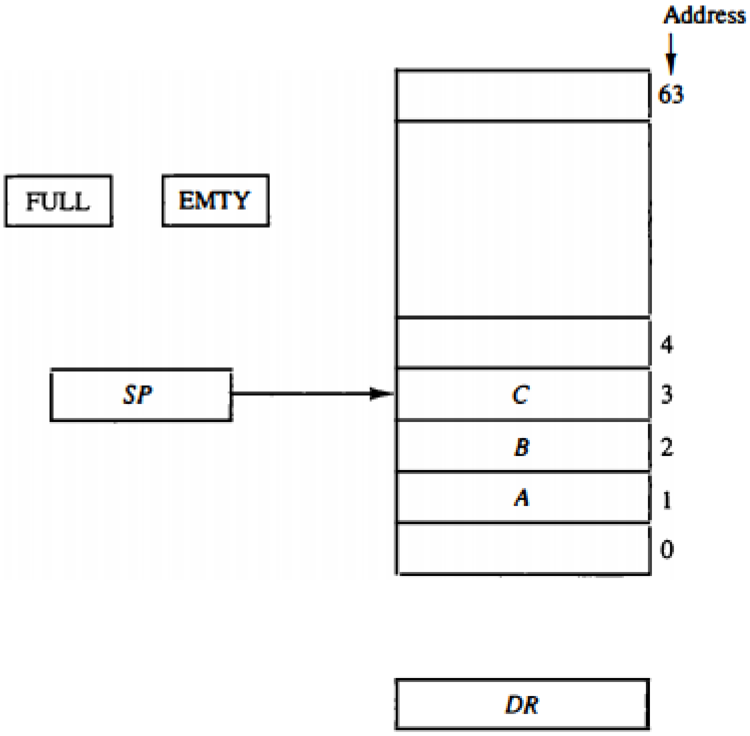


Figure 3 Block diagram of a 64-word stack.

- **To remove the top item**, the stack is popped by reading the memory word at address 3 and decrementing the content of SP.
- **Item B is now on top of the stack since SP holds address 2**. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed.
- **This does not matter because when the stack is pushed**, a new item is written in its place. In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.

- **Since SP has only six bits**, it cannot exceed a number greater than 63 (111111 in binary). When63 is incrementedby 1, the resultis 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.
- **Similarly, when 000000 is decremented by 1**, the result is 111111. The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack.
- **Initially**, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.
- **The push operation** is implemented with the following sequence of microoperations;

- SP ← SP + 1 Increment stack pointer
- M[SP] ← DR   Write item on top of the stack
- If (SP = 0) then (FULL ←1) Check if stack is full
- EMTY ← 0   Mark the stack not empty

- **The stack pointer** is incremented so that it points to the address of the next-higher word. A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.

- **The first item stored** in the stack is at address L The last item is stored at address 0.

- **If SP reaches 0**, the stack is full of items, so FULL is set to L This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.
- **Once an item is stored** in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to
- **A new item** is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of microoperations:

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

- **The top item** is read from the stack into DR . The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1.

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

Memory Stack

- **A stack can exist** as a stand-alone unit as in Fig. 3 or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- **Figure 4** shows a portion of computer memory partitioned into three segments: program, data, and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data.

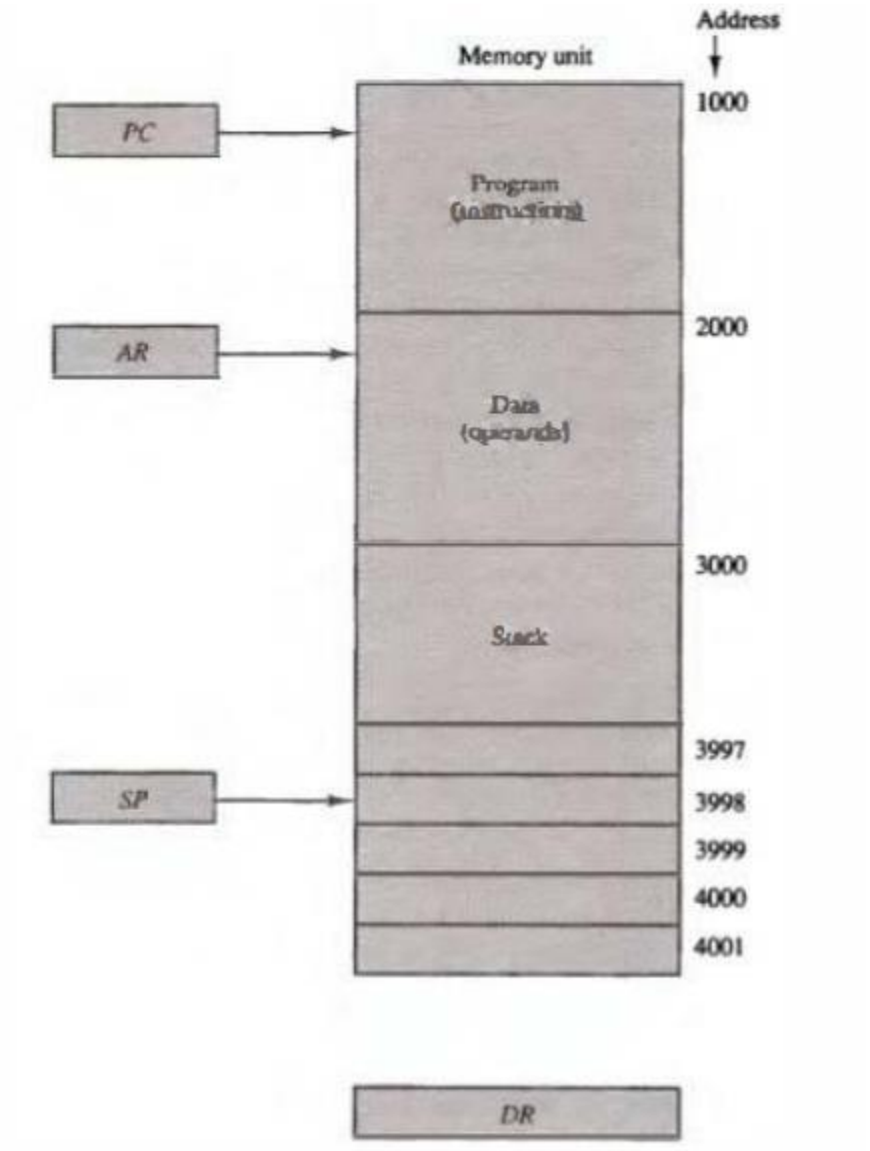


Figure 4 Computer memory with program, data, and stack segments.

- **The stack pointer** SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory.
- **PC is used during the fetch phase** to read an instruction. AR is used during the execute phase to read an operand.

- **SP is used to push or pop items** into or the stack. As shown in Fig. 4, the initial value of SP is 4001 and the stack grows with decreasing addresses.
- **Thus the first item** stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack Is 3000.
- **No provisions** are available for stack limjt checks.
- **We assume that the items** in the stack communicate with a data register DR . A new item is inserted with the push operation as follows:

- SP ← SP - 1
- M[SP] ← DR

- **The stack pointer** is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

- DR ← M[SP]
- SP ← SP + 1

- **The top item** is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.
- **Most computers** do not provide hardware to check for stack overflow (full stack) or underflow (empty stack).

- **The stack limits** can be checked by using two processor registers: one to hold the upper limit (3000 in this case), and the other to hold the lower limit (4001 in this case).

- **After a push operation**, SP is compared with the upper-limit register and after a pop operation, SP is compared with the lower-limit register.
- **The two microoperations** needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP. Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of the stack.
- **In Fig. 4 the stack grows** by decreasing the memory address. The stack may be constructed to grow by increasing the memory address as in Fig. 3.
- **In such a case, SP is incremented** for the push operation and decremented for the pop operation. A stack may be constructed so that SP points at the next empty location above the top of the stack.
- **In this case the sequence** of microoperations must be interchanged. A stack pointer is loaded with an initial value. This initial value must be the bottom address of an assigned stack in memory. Henceforth, SP is automatically decremented or incremented with every push or pop operation.
- **The advantage of a memory stack** is that the CPU can refer to it without having to specify an address, since the

address is always available and automatically updated in the stack pointer.

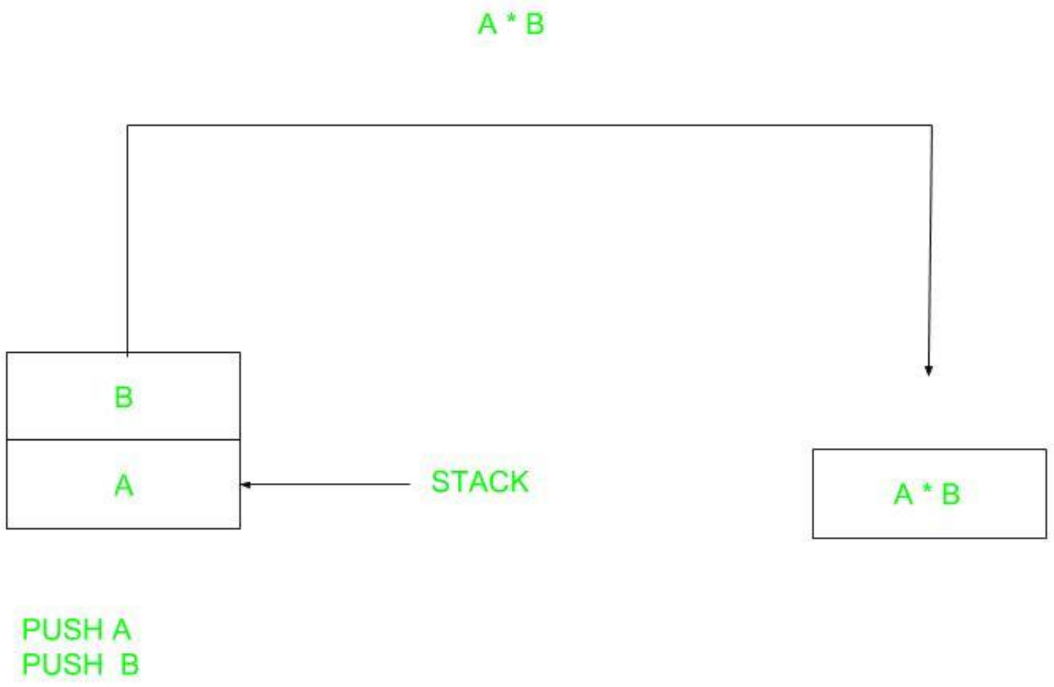
Implementation of Stack
1.Registerstack
2. Memory Stack
Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.
Memory Stack

A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as stack pointer.

2.Evaluate the equation X=(A+B)*(C+D) using various address instruction formats.

0. **Zero Address Instructions –**



A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C

PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

1. **One Address Instructions –**
   This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

opcode	operand/address of operand	mode

Expression: X = (A+B)*(C+D)

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC

```
   LOAD       C           AC = M[C]

    ADD       D       AC = AC + M[D]

    MUL       T       AC = AC * M[T]

  STORE       X           M[X] = AC
```

2. **Two Address Instructions –**
This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

opcode	Destination address	Source address	mode

Here destination address can also contain operand.

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

```
  MOV       R1, A          R1 = M[A]

  ADD       R1, B      R1 = R1 + M[B]
```

MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

3. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

opcode	Destination address	Source address	Source address	mode

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]

MUL X, R1, R2 M[X] = R1 * R2

3.Define addressing mode and illustrate various addressing modes with examples.

Addressing modes specifies the way, the effective address of an operand is represented in the instruction. Some addressing mode efficiently allows referring to a large range of area like a linear array of addresses and list of addresses. Addressing mode describes a **flexible** and **efficient** way to define complex **effective address**.

Types of Addressing Modes

1.Register Addressing Mode

2.Direct Addressing Mode

3.Immediate Addressing Mode

4.Register Indirect Addressing Mode

5.Index Addressing Mode

6.Auto Increment Mode

7.Auto Decrement Mode

8.Relative Addressing Mode

1. Register Addressing Mode

Every instruction includes operands; the operands can be a memory location, a processor register or an I/O device. The instruction which uses processor **registers** to represent operands is the instruction in **register addressing mode**.

Ex: **Add R4, R3**

Load R3, R2

2. Direct Addressing Mode

The direct addressing mode is also known as **Absolute Addressing mode**. Here, the instruction contains the address of the **location in memory** where the value of the operand is stored.

For example, observe the examples below:

Add R2, A

Store R2, B

3. Immediate Addressing Mode

In immediate addressing mode, the value of the operand is **explicitly mentioned** in the instruction. Here, effective address is not required as the operand is explicitly defined in instruction.

Let us see the example of immediate addressing mode:

Add R2, #100

Store R2, 100H

4. Register Indirect Addressing mode

A **processor register** is used to hold the **address of a memory location** where the operand is placed. This addressing mode allows executing the same set of instructions for the different memory location. This can be done by incrementing the content of register thereby pointing the new location each time.

In higher-level language, it is referred to as pointers. The indirect mode is denoted by placing the register inside the parenthesis Now, for example:

Load R3, (R2)   // *Load R2, A*

5. Index Addressing Mode

Index addressing mode is helpful when the instructions in the program are accessing the **array or the large range of memory addresses**. In this mode, the effective address is generated by **adding a constant to the register's content**. The content of the register does not change.

For example, consider the instruction below:

Load R2, A

Load R3, (R2)

Load R4, 4(R2)

6. Auto Increment Addressing Mode

In auto-increment addressing mode once the content of the register is **accessed** by the instruction the register's content is **incremented** to refer the next operand.

Symbolically it is represented as below:(R)+

7. Auto Decrement Addressing Mode

It is just opposite of auto-increment mode. In auto decrement mode the content of the register is **decremented initially** and then the decremented content of the register is used as effective address.

Symbolically it is presented as:**-(R)**

The auto-increment and decrement mode help to implement the **stack structure**.

8. Relative Addressing Mode

In the content above we have discussed the **index addressing mode**. There we were adding a constant to the register content to refer the next operand address. In some computer instead of a register, the **program counter** is used.

The symbolic representation of relative address mode is

X(PC)

The effective address for it would be:

EA = X + (PC)

4.Briefly discuss various program control mechanism

Program Control Instructions are the machine code that are used by machine or in assembly language by user to command the processor act accordingly. These instructions are of various types. These are used in assembly language by user also. But in level language, user code is translated into machine code and thus instructions are passed to instruct the processor do the task.

Types of Program Control Instructions:

There are different types of Program Control Instructions:

1. Compare Instruction:

Compare instruction is specifically provided, which is similar t a subtract instruction except the result is not stored anywhere, but flags are set according to the result.

Example:

CMP R1, R2 ;

2. Unconditional Branch Instruction:

It causes an unconditional change of execution sequence to a new location.

Example:

JUMP L2

Mov R3, R1 goto L2

3. Conditional Branch Instruction:

A conditional branch instruction is used to examine the values stored in the condition code register to determine whether the specific condition exists and to branch if it does.

Example:

Assembly Code : BE R1, R2, L1
Compiler allocates R1 for x and R2 for y
High Level Code: if (x==y) goto L1;

4. Subroutines:

A subroutine is a program fragment that lives in user space, performs a well-defined task. It is invoked by another user program and returns control to the calling program when finished.

Example:

CALL and RET

5. Halting Instructions:

- **NOP Instruction –** NOP is no operation. It cause no change in the processor state other than an advancement of the program counter. It can be used to synchronize timing.
- **HALT –** It brings the processor to an orderly halt, remaining in an idle state until restarted by interrupt, trace, reset or external action.

6. Interrupt Instructions:

Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced.

- **RESET –** It reset the processor. This may include any or all setting registers to an initial value or setting program counter to standard starting location.
- **TRAP –** It is non-maskable edge and level triggered interrupt. TRAP has the highest priority and vectored interrupt.

- **INTR –** It is level triggered and maskable interrupt. It has the lowest priority. It can be disabled by resetting the processor

5.Define Pipelining.Illustrate Arithmetic pipeline with an example.

Ans.

Pipelining:

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing. Pipelining is a technique where multiple instructions are overlapped during execution.

Arithmetic pipeline:

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.

Floating point addition using arithmetic pipeline :

The following sub operations are performed in this case:

1.Compare the exponents.

2.Align the mantissas.
3.Add or subtract the mantissas.
4.Normalize the result

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:

Let us consider two numbers,

X=0.3214*10^3 and Y=0.4500*10^2

Explanation:

First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 times to the right to give

Y=0.0450*10^3

Finally the two numbers are added to produce

Z=0.3664*10^3

As the result is already normalized the result remains the same.

6. Illustrate Instruction Pipelining.Discuss various conflicts of Instruction Pipelining.

Ans.

instruction pipelining: Pipeline processing can occur not only in the data stream but in the instruction stream as well. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence. In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.

1Fetch the instruction from memory.

2. Decode the instruction.

3. Calculate the effective address.

4. Fetch the operands from memory.

5. Execute the instruction.

6. Store the result in the proper place.

The conflicts of instruction pipelining: In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. Resource conflicts caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2. Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

3. Branch difficulties arise from branch and other instructions that change the value of PC.

7.With an algorithm and flow chart discuss implementation of addition and subtraction of signed magnitude numbers.

Ans.

addition and subtraction algorithm:

1.when the signs of A and B are identical (different), add the two magnitudes and attach the sign of A to the result. When the signs of A and B are different (identical), compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be the same as A if A > B or the complement of the sign of A if A < B . If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

2. The two algorithms are similar except for the sign comparison. The procedure to be followed for identical signs in the addition algorithm is the same as for different signs in the subtraction algorithm, and vice versa.

Implementation:

This shows a block diagram of the hardware for implementing the addition and subtraction operations. It consists of registers A and B and sign flip-flops A s and Bs . Subtraction is done by adding A to the 2's complement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add-overflow flip-flop AVF holds the overflow bit when A and B are added. The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm. The addition of A plus B is done through the parallel adder. The S (sum) output of the adder is applied to the input of the A register. The complementer provides an output of B or the complement of B depending on the state of the mode control M. The complementer consists of exclusive-OR gates and the parallel adder consists of full-adder circuits as shown in Fig. 4-7 in Chap. 4. The M signal is also applied to the input carry of the adder. When M = 0, the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum A + B. When M = 1, the l's complement of B is applied to the adder, the input carry is 1, and output S = A + B + 1. This is equal to A plus the 2's complement of B, which is equivalent to the subtraction of A-B.

Algorithm:

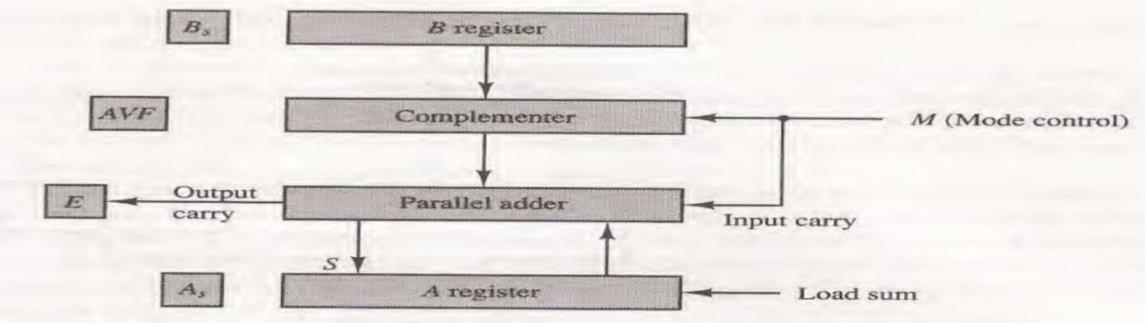


Figure 10-1 Hardware for signed-magnitude addition and subtraction.

The two signs A s and B s are compared by an exclusive-OR gate. If the output of the gate is 0, the signs are identical; if it is 1, the signs are different. For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added. The magnitudes are added with a microoperation EA=EA+1 where EA is a register that combines £ and A . The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF. The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complement of B. No overflow can occur if the numbers are subtracted so AVE is cleared to 0. A 1 in £ indicates that A > B and the number in A is the correct result. If this number is zero, the sign As must be made positive to avoid a negative zero. A 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. This operation can be done with one microoperation A=A’+1. However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations. In other paths of the flowchart, the sign of the result is the same as the sign of A, so no change in As is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement As to obtain. the correct sign. The final result is found in register A and its sign in As . The value in AVF provides an overflow indication. The final value of £ is immaterial.

Flowchart:

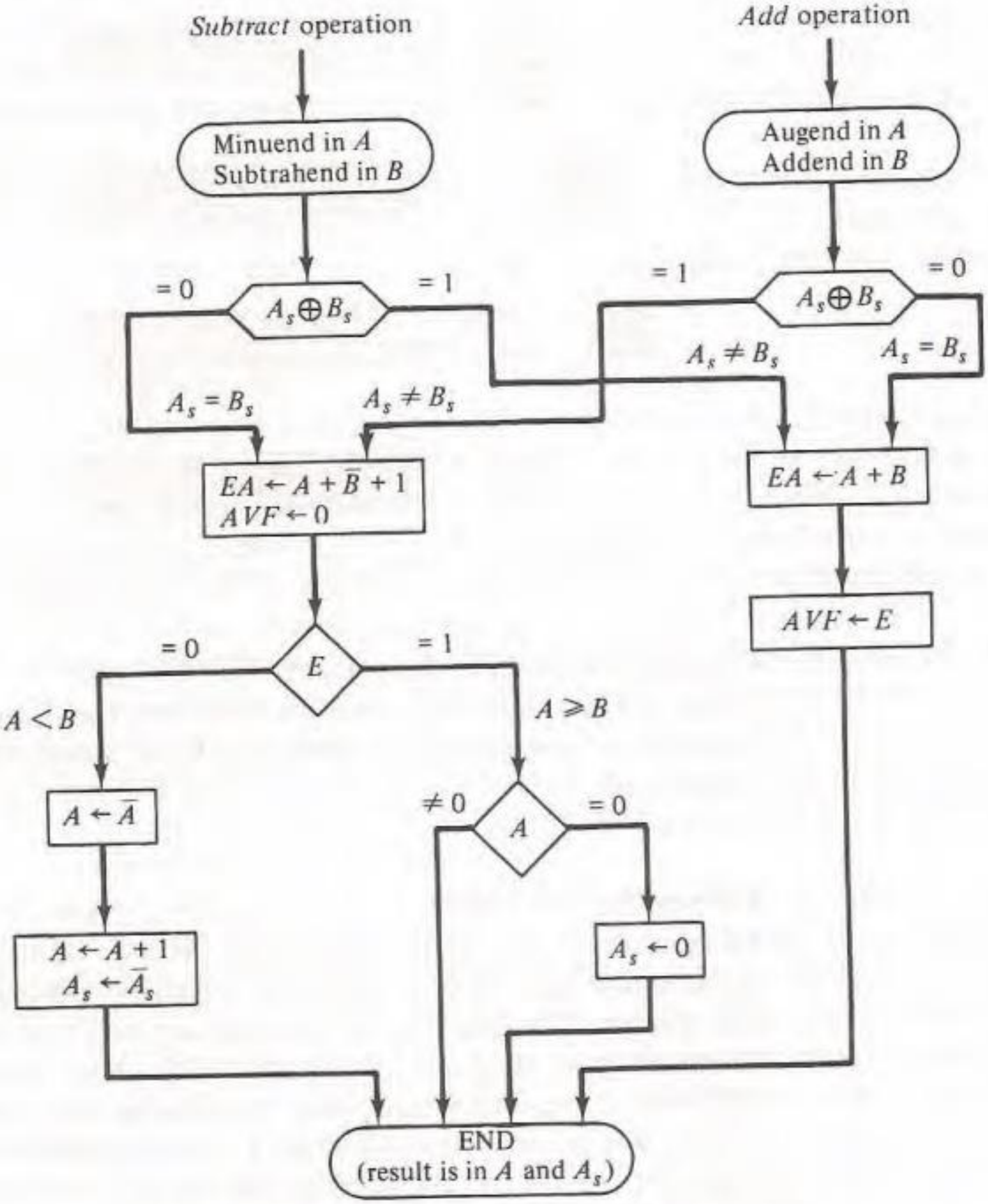


Figure 10-2 Flowchart for add and subtract operations.

8.Explain Booths Algorithm.

Ans.Booth’s algorithm is a powerful algorithm that is used for signed multiplication. It generates a 2n bit product for two n bit signed numbers.

Flowchart:

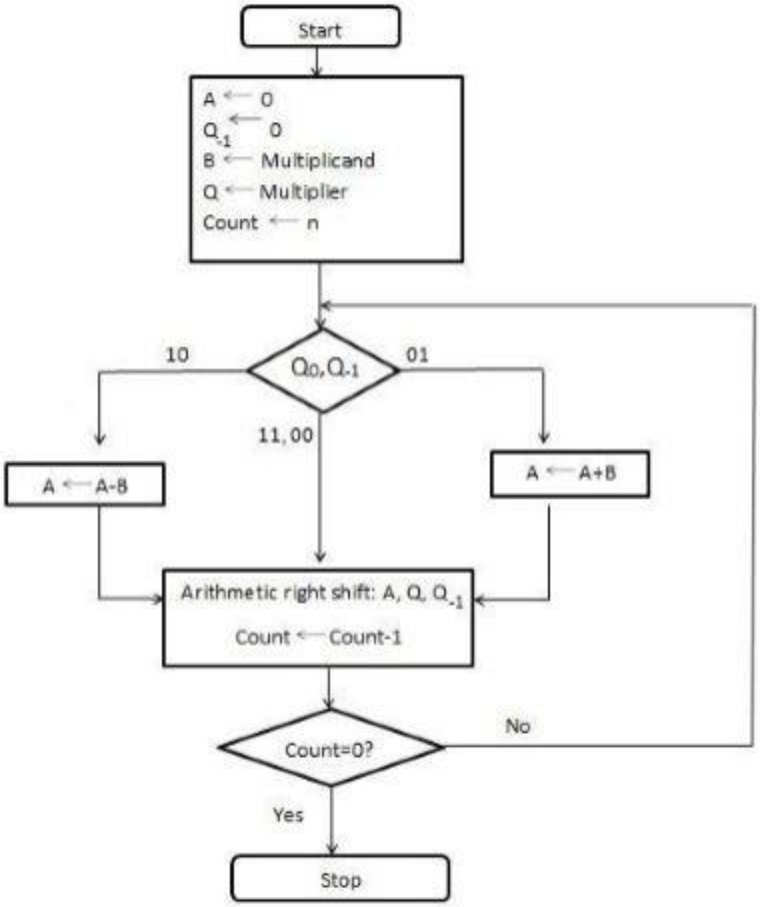


Figure 1

The steps in Booth’s algorithm are as follow:

1) Initialize A,Q−1Q−1 to 0 and count to n

2) Based on the values of Q0 and Q−1Q0 and Q−1 do the following:

a. if Q0,Q−1Q0,Q−1=0,0 then Right shift A,Q,Q−1Q−1 and finally decrement count by 1

b. If Q0,Q−1Q0,Q−1=0,1 then Add A and B store in A, Right shift A,Q,Q−1Q−1 and finally decrement count by 1

c. If Q0,Q−1=1Q0,Q−1=1,0 then Subtract A and B store in A, Right shift A,Q,Q−1Q−1 and finally decrement count by 1

d. If Q0,Q−1=1Q0,Q−1=1,1 then Right shift A,Q,Q−1Q−1 and finally decrement count by 1

3) Repeat step 2 till count does not equal 0.

Using the flowchart, we can solve the given question as follows:

(Q).Multiply (−2)10   and   (−5)10(−2)10   and   (−5)10 **using Booth's algorithm**

(−5)10(−5)10= 1011(in 2's complement)

(−2)10(−2)10 =1110(in 2's complement)

Multiplicand (B) = 1011

Multiplier (Q) =1110

And initially Q-1=0

Count =4

Steps	**A**	**Q**	Q^{1}Q1	**Operation**

Steps	**A**	**Q**	Q^1Q1	**Operation**
Initial	0000	1110	0	-
1	0000	0111	0	Shift Right
2	0101 0010	0111 1011	0 1	A=A-B Shift right
3	0001	0101	1	Shift right
4	0000	1010	1	Shift Right
Result	0000	1010	-	-

Result =(0000 1010)10 or (10)10(0000 1010)10 or (10)10

This is the required and correct result.

9.Briefly discuss Decimal Arithmetic Unit.

The user of a computer prepares data with decimal numbers and receives results in decimal form. A CPU with an arithmetic logic unit can perform arithmetic microoperations with binary data. To perform arithmetic operations with decimal data, it is necessary to convert the input decimal numbers to binary, to perform all calculations with binary numbers, and to convert the results into decimal. This may be an efficient method in applications requiring a large number of calculations and a relatively smaller amount of input and output data. When the application calls for a large amount of input-output and a relatively smaller number of arithmetic calculations, it becomes convenient to do the internal arithmetic directly with the decimal numbers. Computers capable of performing decimal arithmetic must store the decimal data in binarycoded form. The decimal numbers are then applied to a decimal arithmetic unit capable of executing decimal arithmetic microoperations. Electronic calculators invariably use an internal decimal arithmetic unit since inputs and outputs are frequent. There does not seem to be a reason for converting the keyboard input numbers to binary and again converting the displayed results to decimal, since this process requires special circuits and also takes a longer time to execute. Many computers have hardware for arithmetic calculations with both binary and decimal data. Users can specify by programmed instructions whether they want the computer to perform calculations with binary or decimal data.

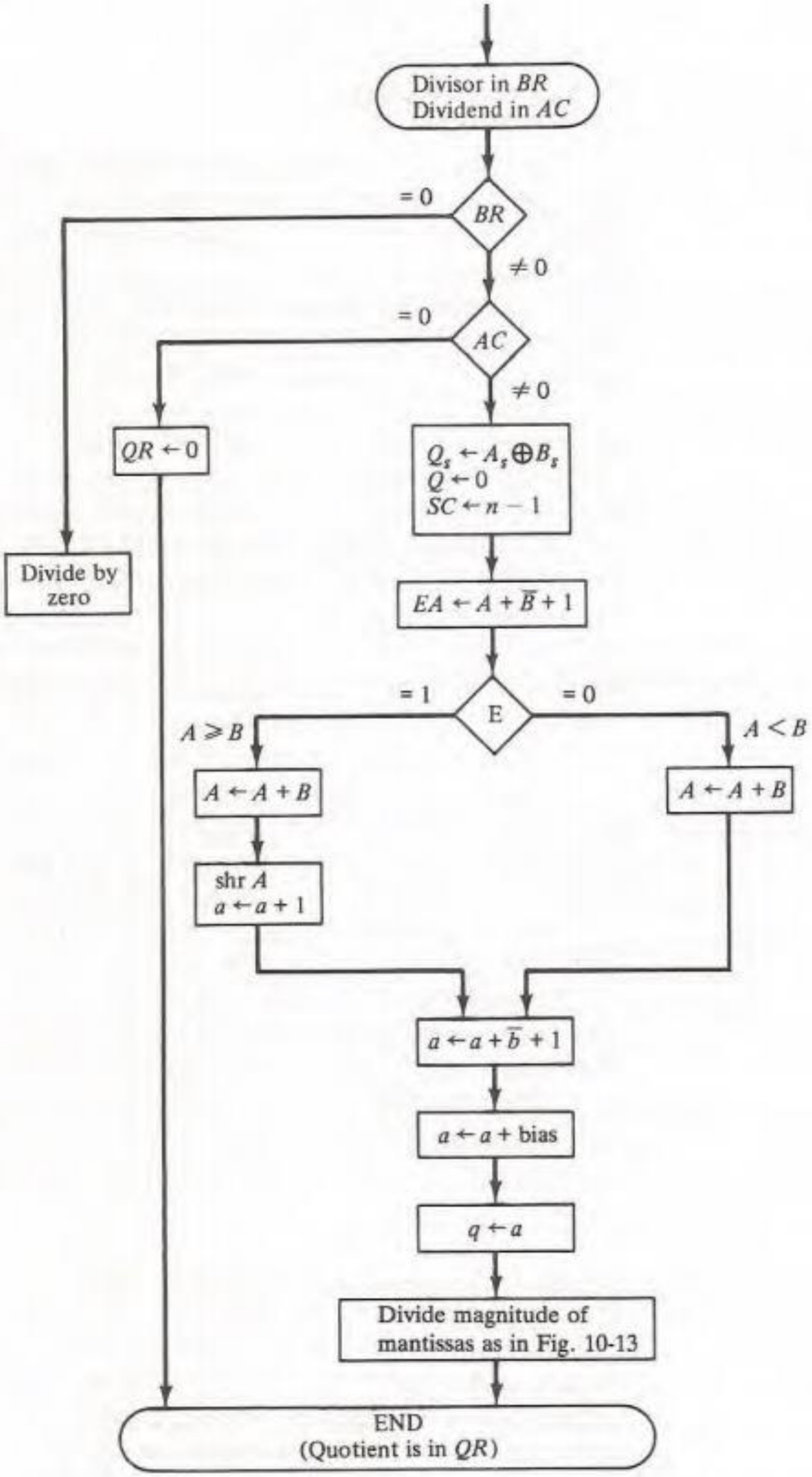


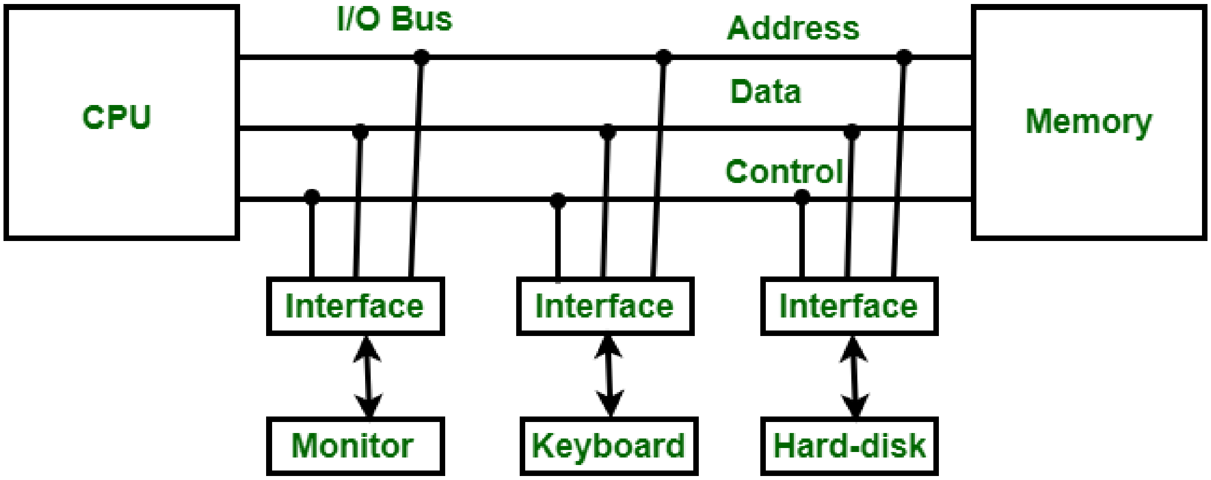
Figure 10-17 Division of floating-point numbers.

Unit 4

1.define I/O interface

Ans:

Input-Output Interface
 is used as an method which helps in transferring of information between the internal storage devices i.e. memory and the external peripheral device . A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices.



To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:

 i. Use two Separate buses , one for memory and other for I/O.

ii. Use one common bus for both memory and I/O but separate control lines for each.

iii. Use one common bus for memory and I/O with common control lines.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output device.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also convert digital data into analog signal and vice-versa.

2.Discuss strobing and handshaking mechanism for implementing asynchronous transfer

Ans.

The strobe pulse and handshaking method of asynchronous data transfer are not restricted to I/O transfer.In fact, they are used extensively on numerous occasion requiring transfer of data between two independent units.So, here we consider the transmitting unit as source and receiving unit as destination.

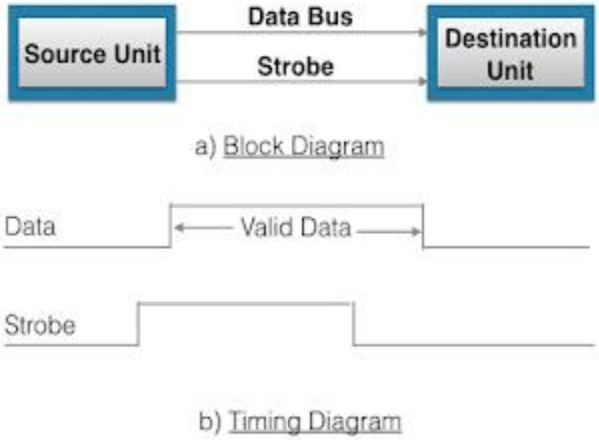
As an example: The CPU, is the source during an output or write transfer and is the destination unit during input or read transfer.

. **<u>Strobe Control:</u>**

The Strobe Control method of asynchronous data transfer employs a single control line to time each transfer .This control line is also known as strobe and it may be achieved either by source or destination, depending on which initiate transfer.

<u>Source initiated strobe for data transfer:</u>

The block diagram and timing diagram of strobe initiated by source unit is shown in figure below:



The disadvantage of strobe method is that source unit that initiates the transfer has no way of knowing whether the destination has actually received the data that was placed in the bus.Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit, has actually placed data on the bus.

This problem can be solved by handshaking method.

Hand shaking method introduce a second control signal line that provides a replay to the unit that initiates the transfer.

In it, one control line is in the same direction as the data flow in the bus from the source to destination .It is used by source unit to inform the destination unit whether there are valid data in the bus.The other control line is in the other direction from destination to the source.It is used by the destination unit to inform the source whether it can accept data.And in it also, sequence of control depends on unit that initiate transfer.Means sequence of control depends whether transfer is initiated by source and destination

3.with a example illustrate programmed I/O

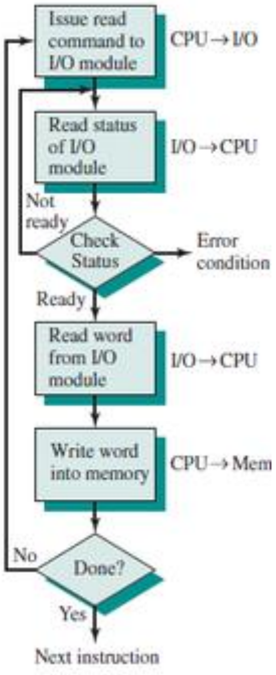
it requires constant monitoring by the CPU of the peripheral devices. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program.

Usually the transfer is from a CPU register and memory. In this case

The overall operation of the programmed I/O can be summaries as follow:

1. The processor is executing a program and encounters an instruction relating to I/O operation.
2. The processor then executes that instruction by issuing a command to the appropriate I/O module.
3. The I/O module will perform the requested action based on the I/O command issued by the processor (READ/WRITE) and set the appropriate bits in the I/O status register.
4. The processor will periodically check the status of the I/O module until it find that the operation is complete.

5.



4.define priority interrupt.with neat diagrams explain parallel priority and series priority interrupt

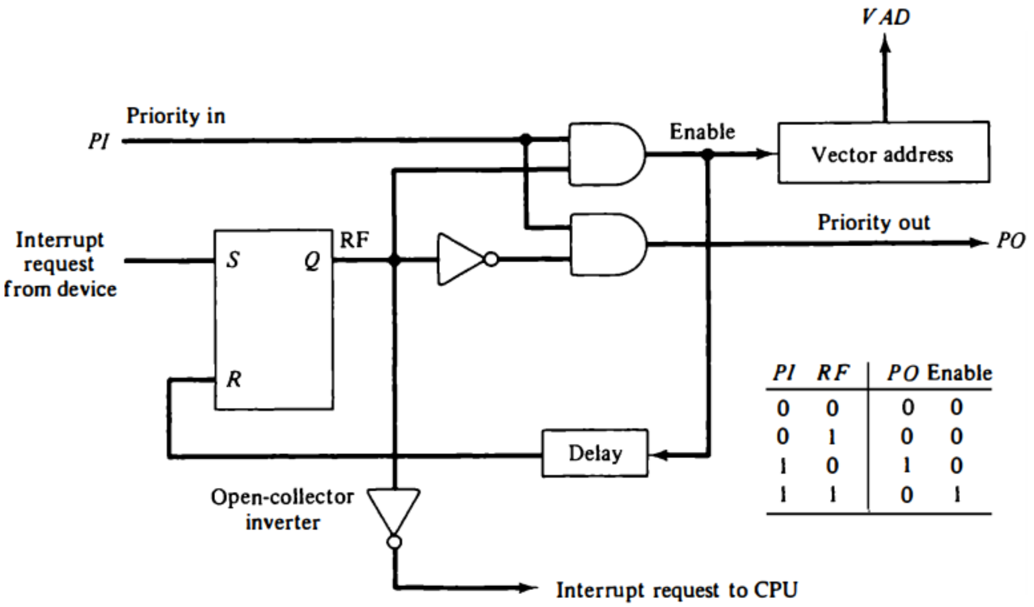
A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU. The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced. Generally, devices with high speed transfer such as *magnetic disks* are given high priority and slow devices such as *keyboards* are given low priority.

Parallel priority:

□
The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device.

□ **Priority** is established according to the position of the bits in the register

□ **In addition** to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request.



PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

- **Figure 13** One stage of the daisy-chain priority arrangement.

- **The mask register** can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced.

- **It can also provide** a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.

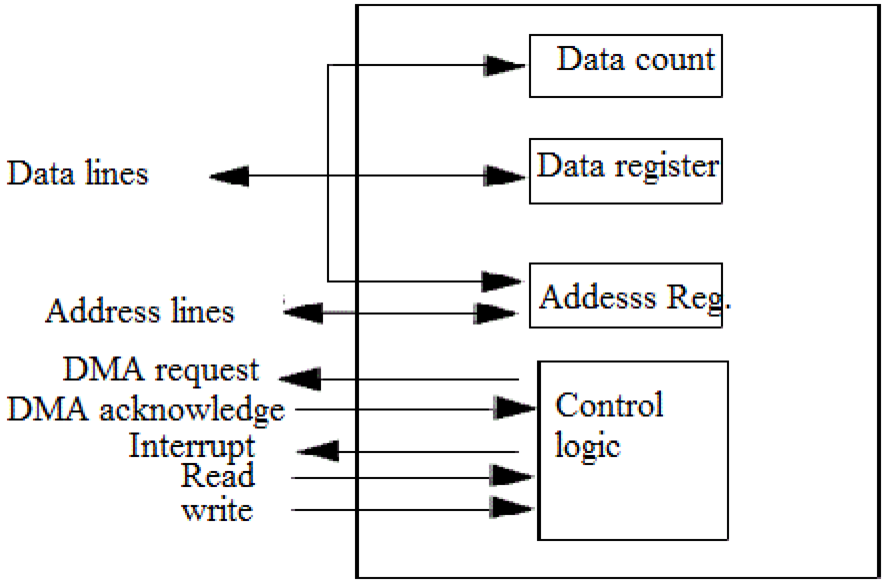
- **The priority logic** for a system of four interrupt sources is shown in Fig. 14.
- **It consists** of an interrupt register whose individual bits are set by external conditions and cleared by program instructions.

- **The magnetic disk,** being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard.
- **The mask register** has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register.
- **Each interrupt bit** and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder.
- **In this way** an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.
- **Another output** from the encoder sets an interrupt status flip-flop lST when an interrupt that is not masked occurs. The interrupt enable flip-flop lEN can be set or cleared by the program to provide an overall control over the interrupt system.
- **The outputs** of IST ANDed with IEN provide a common interrupt signal for the CPU.

5. With neat diagram illustrate DMA operation

Ans.

Dma Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.



Modes of operation

Burst mode

In burst mode, an entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode".

Cycle stealing mode

The cycle stealing mode is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR (Bus Request) and BG (Bus Grant) signals, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on. Data is not transferred as quickly, but CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

Transparent mode

Transparent mode takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. In transparent mode, the DMA controller transfers data only when the CPU is performing operations that do not use the system buses. The primary advantage of transparent mode is that the CPU never stops executing its programs and the DMA transfer is free in terms of time, while the disadvantage is that the hardware needs to determine when the CPU is not using the system buses, which can be complex. This is also called "Hidden DMA data transfer mode".

6.Write short notes IBM 370 i/o channel.

Ans.

- A channel is an independent hardware component that co-ordinate all I/O to a set of controllers. Computer systems that use I/O channel have special hardware components that handle all I/O operations.
- Channels use separate, independent and low cost processors for its functioning which are called Channel Processors.
- Channel processors are simple, but contains sufficient memory to handle all I/O tasks. When I/O transfer is complete or an error is detected, the channel controller communicates with the CPU using an interrupt, and informs CPU about the error or the task completion.
- Each channel supports one or more controllers or devices. **Channel programs** contain list of commands to the channel itself and for various connected controllers or devices. Once the operating system has prepared a list of I/O commands, it executes a single I/O machine instruction to initiate the channel program, the channel then assumes control of the I/O operations until they are completed.

IBM 370 I/O Channel

The I/O processor in the IBM 370 computer is called a Channel. A computer system configuration includes a number of channels which are connected to one or more I/O devices.

Categories of I/O Channels

Following are the different categories of I/O channels:

Multiplexer

The Multiplexer channel can be connected to a number of slow and medium speed devices. It is capable of operating number of I/O devices simultaneously.

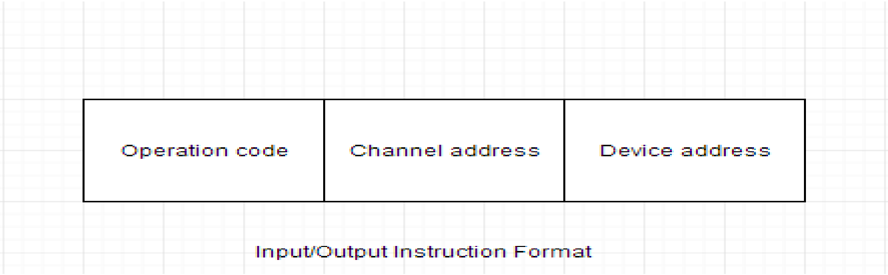
Selector

This channel can handle only one I/O operation at a time and is used to control one high speed device at a time.

Block-Multiplexer

It combines the features of both multiplexer and selector channels.

The CPU directly can communicate with the channels through control lines. Following diagram shows the word format of channel operation.



Input/Output Instruction Format

The computer system may have number of channels and each is assigned an address. Each channel may be connected to several devices and each device is assigned an address.

7.Discuss I/O Processor

Ans.The input/output processor or I/O processor is a processor separate from the CPU designed to handle only input/output processes for a device or the computer.

The I/O processor is capable of performing actions without interruption or intervention from the CPU. The CPU only needs to initiate the I/O processor by telling it what activity to perform. Once the necessary actions are performed, the I/O processor then provides the results to the CPU. Doing these actions allow the I/O processor to act as a bus to the CPU, carrying out activities by directly interacting with memory and other devices in the computer. A more advanced I/O processor may also have memory built into it, allowing it to perform actions and activities more quickly.

For example, without an I/O processor, a computer would require the CPU to perform all actions and activities, reducing overall computer performance. However, a computer with an I/O processor would allow the CPU to send some activities to the I/O processor. While the I/O processor is performing the necessary actions for those activities, the CPU is free to carry out other activities, making the computer more efficient and increase performance.

Unit 5

1. **Explain internal organization of memory chips and static memories**

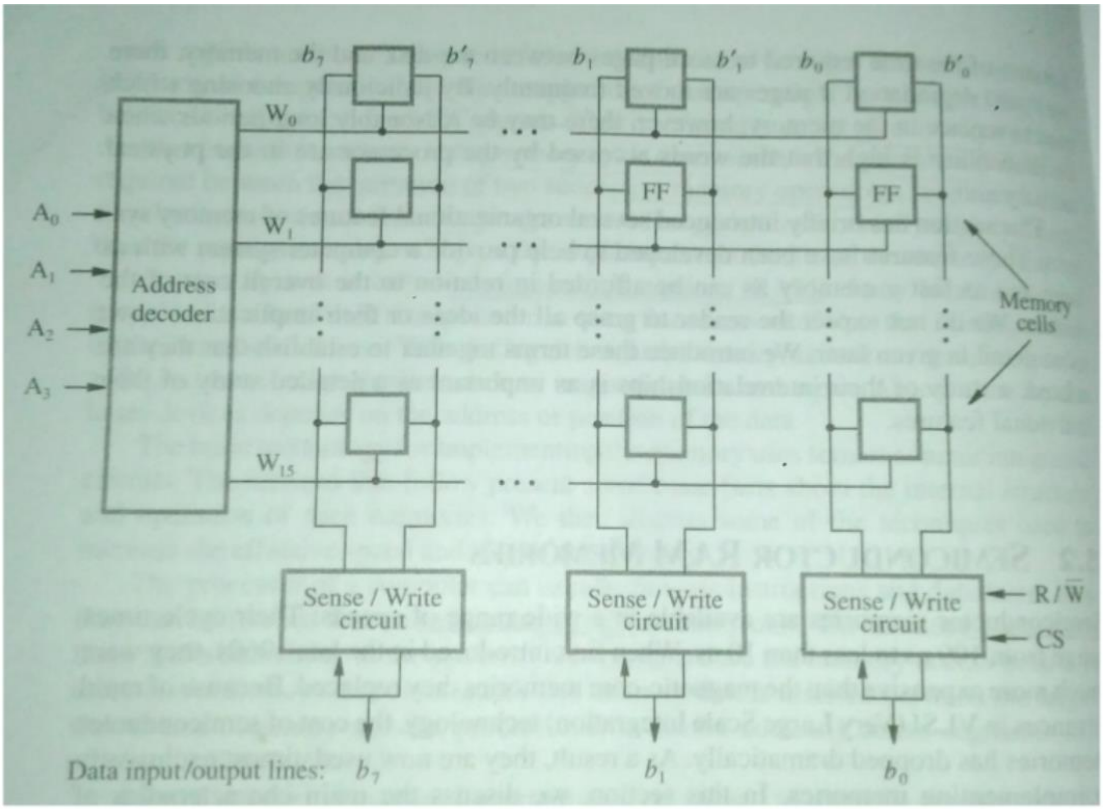
Ans.

Internal Organization of memory chips:

- Memory cells are usually organized in the form of array, in which each cell is capable of storing one bit of information.
- Each row of cells constitutes a memory word and all cells of a row are connected to a common line called as word line.
- The cells in each column are connected to Sense/Write circuit by two bit lines.
- Figure shows the possible arrangements of memory cells.
- The Sense/Write circuits are connected to data input or output lines of the chips.

- During a write operation, the sense/write circuit receives input information and stores it in the cells of the selected word.
- The data input and data output of each sense/Write circuits are connected to a single bidirectional dataline that can be connected to a data bus of the cpu.
- R /W --> specifies the required operation.
- CS --> Chip Select input selects a given chip in the multi-chip memory system.

Organization of bit cells in a memory chip



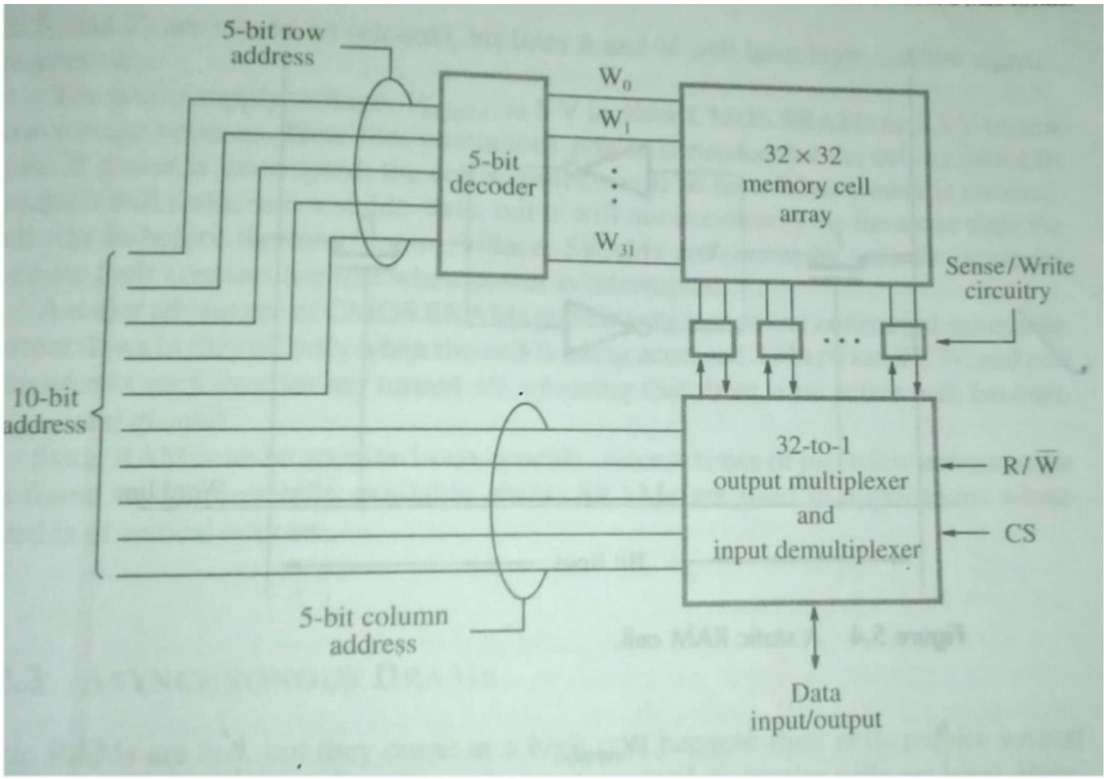
- Memory chip consisting of 16 words of 8 bit each.

- This is referred to as a 16 x 8 organization.
- It can stores 128 bits and requires 14 external connections for address, data and control lines.
- It also needed 2 lines for power supply and ground.
- Consider a memory circuit with 1k cells.this can be organized as a 128 x 8 memory.Requiring a total of 19 external connections
- Same can be organized into a 1k x 1 format

  1. 10 bit address is needed
  2. But thee is only one data line.
  3. Requiring a total of 15 external connections

- The required 10-bit address is divided into 2 groups of 5-bits each to form a row and column addresses for the cell array.
- Row address selects a row of 32 cells, all of which are selected in parallel
- According to the column address only one of these cells is connected to the external data line by the output multiplexer and the input demultiplexer.

Organization of 1k x 1   Memory chip



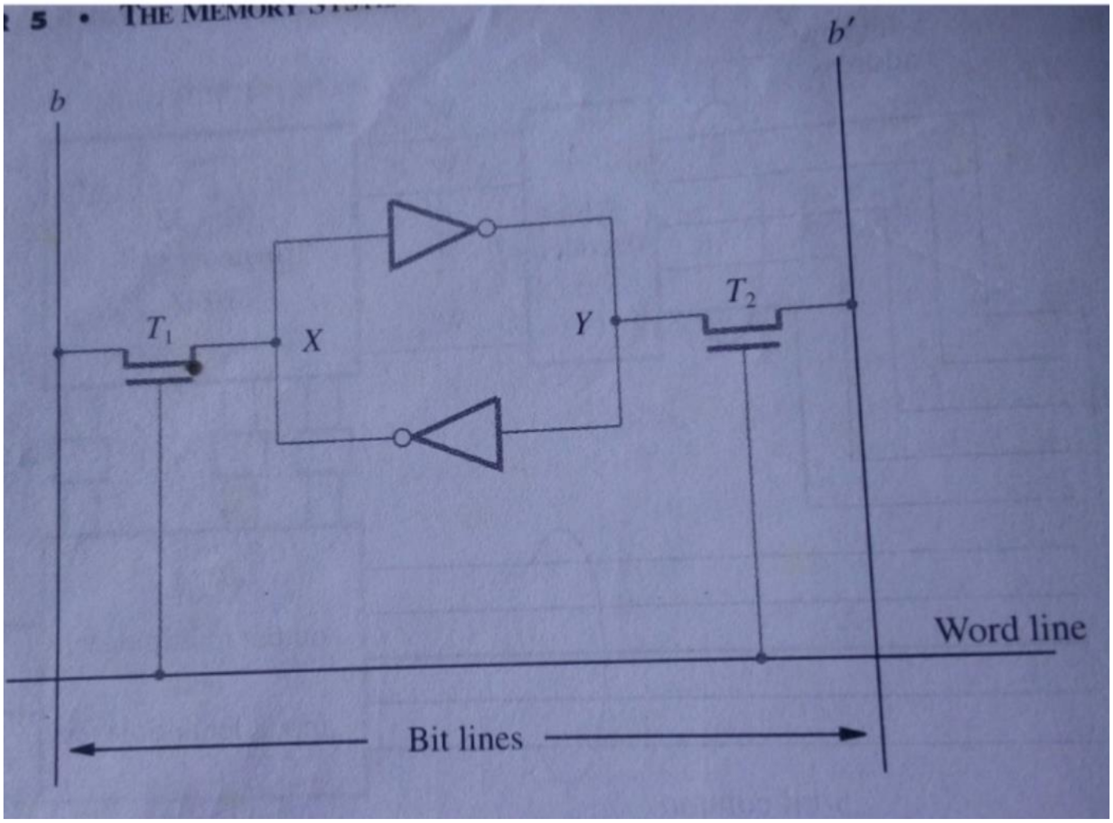
- Eg: A 4M-bit chip may have a 512 k x 8 organization.
    1. 9 address lines
    2. 8 input/output pins are needed.

Static memories

- Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memory.
- Implementation of static memory.
    1. Two inverters are cross connected to form a latch is connected to two bit lines by transistors T1 and T2
    2. These transistors act as switches that can be opened closed under the control of the word line.

3. When the word line is at ground level, the transistors are turned off and the latch retains its state.

Static RAM cell



Example:

- Assume that the cell is in state 1:
    1. Logic value at point X is 1
    2. Logic value at point Y is 0
- This state is maintained as long as the signal on the word line is at ground level.
- Read operation

1. In order to read the state of the SRAM cell, the word line is activated to close switches T! and T2.
2. If the cell is in state 1, the signal on bit line b is high and the signal on the bit line b is low.
3. Thus b and b’ are complement of each other.
4. Sense / Write circuit at the end of the bit line monitors the state of b and b’ and set the output according.

- Write operation
    1. The state of the cell is set by placing he appropriate value an bit line b and its complement on b’ and then activating the word line.
    2. This forces the cell into the corresponding state. The required signal on the bit lines are generated by Sense/Write circuit.

2.Explain replacement algorithms with Example

Ans.

Page Replacement Algorithms-

Page replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.

Various page replacement algorithms are-

- FIFO Page Replacement Algorithm
- LIFO Page Replacement Algorithm
- LRU Page Replacement Algorithm
- Optimal Page Replacement Algorithm
- Random Page Replacement Algorithm

A good page replacement algorithm is one that minimizes the number of page faults.

1.FIFO Page Replacement Algorithm-

As the name suggests, this algorithm works on the principle of "First in First out".

It replaces the oldest page that has been present in the main memory for the longest time.

It is implemented by keeping track of all the pages in a queue.

2.LIFO Page Replacement Algorithm-

As the name suggests, this algorithm works on the principle of “Last in First out“.

It replaces the newest page that arrived at last in the main memory.

It is implemented by keeping track of all the pages in a stack.

NOTE

Only frame is used for page replacement during entire procedure after all the frames get occupied.

3.LRU Page Replacement Algorithm-

As the name suggests, this algorithm works on the principle of “Least Recently Used“.

It replaces the page that has not been referred by the CPU for the longest time.

4.Optimal Page Replacement Algorithm-

This algorithm replaces the page that will not be referred by the CPU in future for the longest time.

It is practically impossible to implement this algorithm.

This is because the pages that will not be used in future for the longest time can not be predicted.

However, it is the best known algorithm and gives the least number of page faults.

Hence, it is used as a performance measure criterion for other algorithms.

5.Random Page Replacement Algorithm-

As the name suggests, this algorithm randomly replaces any page.

So, this algorithm may behave like any other algorithm like FIFO, LIFO, LRU, Optimal etc.

3.Explain performance considerations in memory system

Ans.

- A key design objective of a computer system is to achieve the best possible performance at the lowest possible cost
  - Price/performance ratio is a common measure of success.
- Performance of a processor depends on:
  - How fast machine instructions can be brought into the processor for execution.
  - How fast the instructions can be executed.
- Interleaving
- Methods of address layouts
- Hit rate and Miss penalty
- Caches on the processor chip
- Write buffer
- Prefetching
- Lockup-Free Cache

4.A digital computer has a memory unit of 64*16 and a cache memory of 1K words. The cache uses direct mapping with a block size of four words

i) How many bits are there a in tag, index, block and word fields of address format

ii) How many bits are there in each words of cache and how are they divided into functions? Include a valid bit

iii) How many blocks the cache can accommodate?

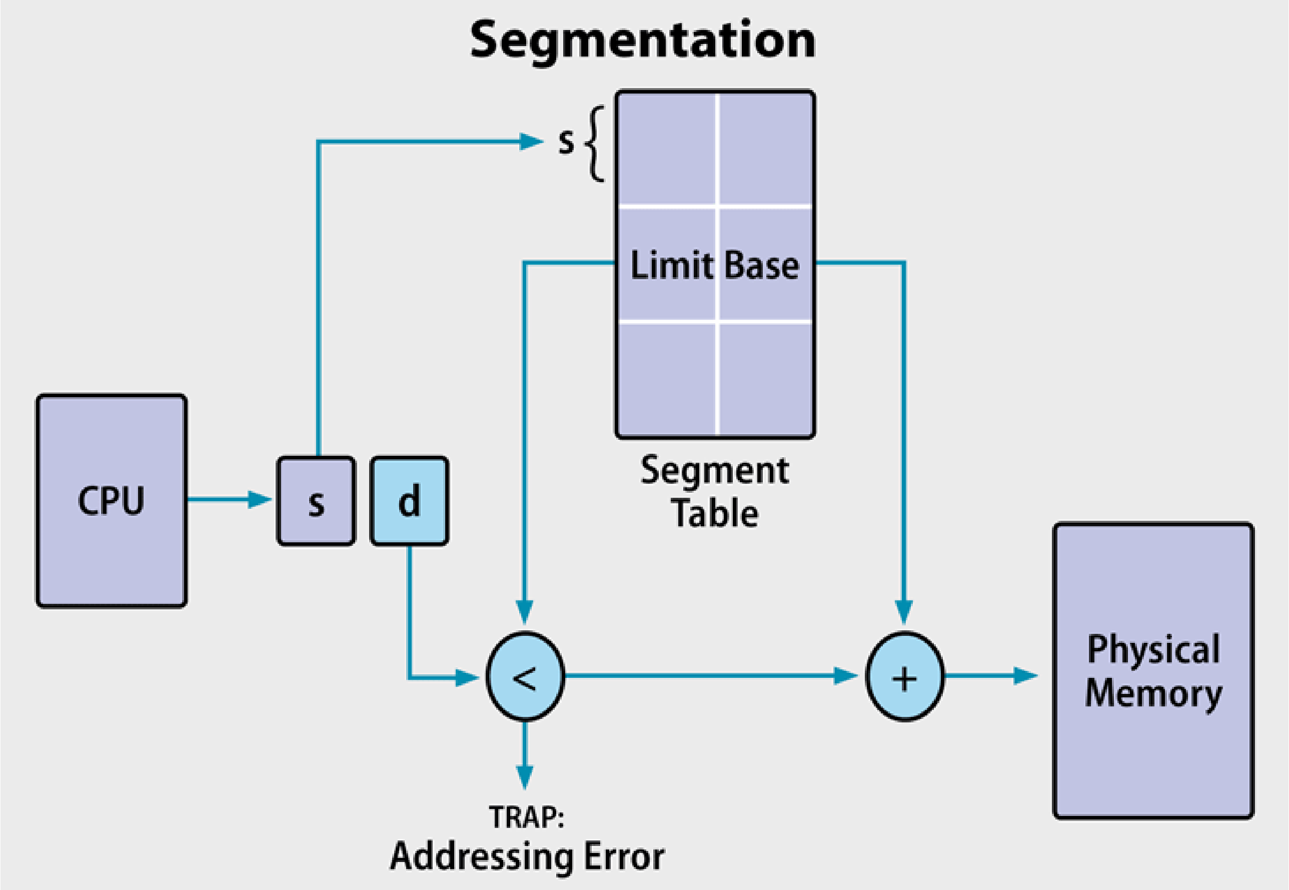
5.Explain segmented page mapping with the help of suitable example

Ans.

Segmentation is a virtual process that creates variable-sized address spaces in computer storage for related data, called segments.

The process known as segmentation is a virtual process that creates address spaces of various sizes in a computer system, called segments. Each segment is a different virtual address space that directly corresponds to process objects.

When a process executes, segmentation assigns related data into segments for faster processing. The segmentation function maintains a segment table that includes physical addresses of the segment, size, and other data.



Segmentation speeds up a computer's information retrieval by assigning related data into a "segment table" between the CPU and the physical memory.

The Segmentation Process

Each segment stores the processes primary function, data structures, and utilities. The CPU keeps a segment map table for every process and memory blocks, along with segment identification and memory locations.

The CPU generates virtual addresses for running processes. Segmentation translates the CPU-generated virtual addresses into physical addresses that refer to a unique physical memory location. The translation is not strictly one-to-one: different virtual addresses can map to the same physical address.

Segmented Paging

Some modern computers use a function called segmented paging. Main memory is divided into variably-sized segments, which are then divided into smaller fixed-size pages on disk. Each segment contains a page table, and there are multiple page tables per process.

Each of the tables contains information on every segment page, while the segment table has information about every segment. Segment tables are mapped to page tables, and page tables are mapped to individual pages within a segment.

Advantages include less memory usage, more flexibility on page sizes, simplified memory allocation, and an additional level of data access security over paging. The process does not cause external fragmentation.

6.Explain the associative memory with a neat diagram and derive the match logic for one word of association memory

Ans.

Associative memory searches stored data only by the data value itself rather by an address. This type of search helps in reducing the search time by a large extent.

working of *associative memory in computer architecture*:

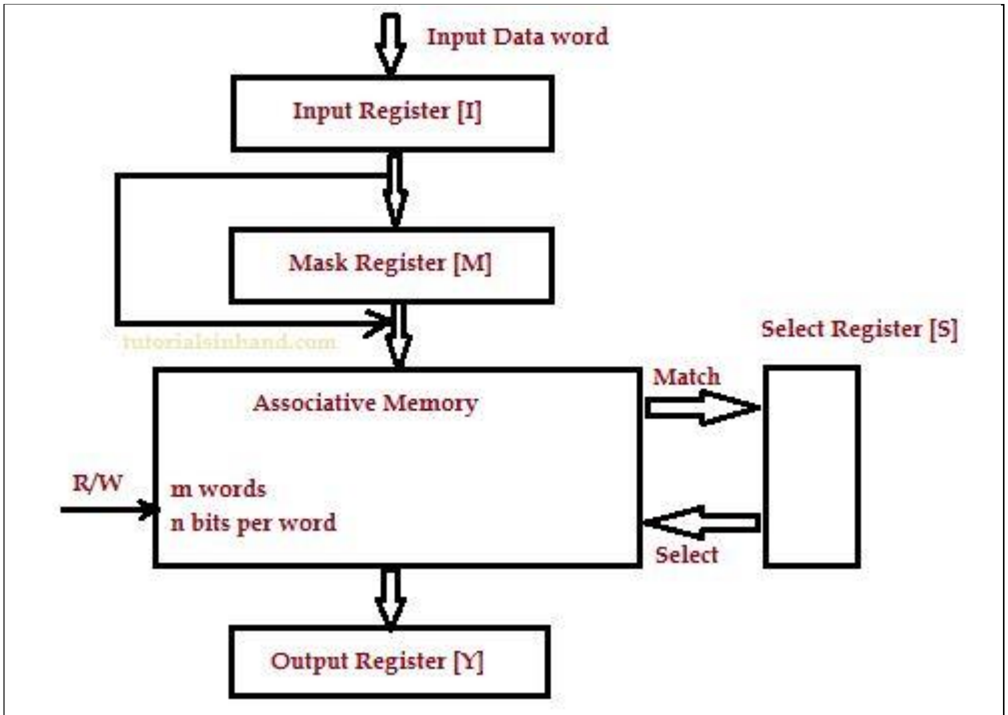
- Data is stored at the very first empty location found in memory.
- In associative memory when data is stored at a particular location then no address is stored along with it.
- When the stored data need to be searched then only the key (i.e. data or part of data) is provided.
- A sequential search is performed in the memory using the specified key to find out the matching key from the memory.
- If the data content is found then it is set for the next reading by the memory.

Associative memory organization

The *associative memory hardware structure* consists of:

- memory array,
- logic for m words with n bits per word, and
- several registers like input resgister, mask register, select register and output register.

The **block diagram showing organization of associative memory** is shown below:



Block Diagram of Associative Memory

Functions of the registers used in associative memory is given below:

- **Input Register (I)** hold the data that is to be written into the associative memory. It is also used to hold the data that is to be searched for. At a particular time it can hold a data containing one word of length (say n).
- **Mask Register (M)** is used to provide a mask for choosing a key or particular field in the input register's word. Since input register can hold a data of one word of length n so the maximum length of mask register can be n.
- **Select Register (S)** contains m bits, one for each memory words. When input data in I register is compared to key in

m register and match is found then that particular bit is set in select register.

- **Output Register (Y)** contains the matched data word that is retrieved from associative memory.

Advantages of associative memory

Advantages of associative memory is given below:

- Associative memory searching process is fast.
- Associative memory is suitable for parallel searches.

Disadvantages of associative memory

Disadvantages of associative memory is given below:

- Associative memory is expensive than RAM

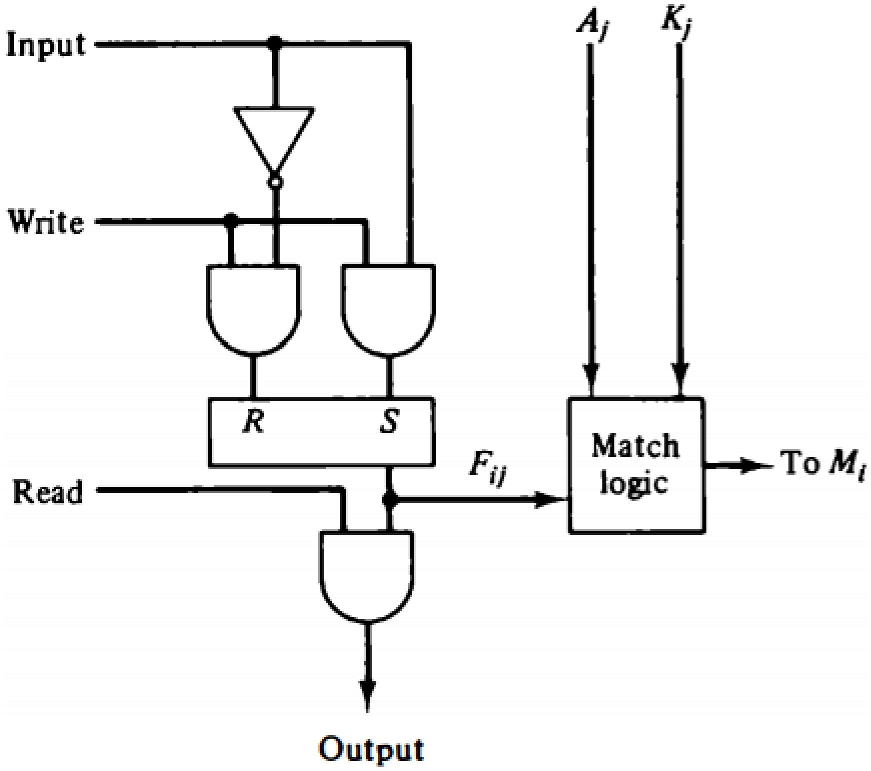
Match Logic

- **The match logic** for each word can be derived from the comparison algorithm for two binary numbers.
- **First**, we neglect the key bits and compare the argument in A with the bits stored in the cells of the words. Word i is equal to the argument in A if Aj = Fij for j = 1, 2, ... , n. Two bits are equal if they are both 1 or both 0.

- **The equality** of two bits can be expressed logically by the Boolean function $xj=AjFij+A'jF'ij$ where $xj = 1$ if the pair of bits in position j are equal; otherwise, $xj = 0$.

- **For a word i** to be equal to the argument in A we must have all xj variables equal to 1.
- **This is the condition** for setting the corresponding match bit Mi to 1. The Boolean function for this condition is $Mi=x1x2x3...xn$ and constitutes the AND operation of all pairs of matched bits in a word.

Figure 8 One cell of associative memory.



- **We now include** the key bit Kj in the comparison logic. The requirement is that if $Kj = 0$, the corresponding bits of Aj and

Fij need no comparison. Only when Kj = 1 must they be compared. This requirement is achieved by ORing each term with K'j thus:

$$x_j + K_j' = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

- **When Kj = 1**, we have K'j = 0 and xj + 0 = xj . When Kj = 0, then K'j = 1 and xj + 1 = 1. A term (xj + K'j) will be in the 1 state if its pair of bits is not compared.
- **This is necessary** because each term is ANDed with all other terms so that an output of 1 will have no effect. The comparison of the bits has an effect only when Kj = 1.
- **The match logic for** word i in an associative memory can now be expressed by the following Boolean function:

$$M_i = (x_1 + K_1')(x_2 + K_2')(x_3 + K_3') \cdots (x_n + K_n')$$

- **Each term** in the expression will be equal to 1 if its corresponding Kj = 0. If Kj = 1, the term will be either 0 or 1 depending on the value of Xj. A match will occur and Mj will be equal to 1 if all terms are equal to 1.
- **If we substitute** the original definition of xj, the Boolean function above can be expressed as follows:

$$M_i = \prod_{j=1}^{n} (A_j F_{ij} + A_j' F_{ij}' + K_j')$$

- **where ∏ is a product** symbol designating the AND operation of all n terms. We need m such functions, one for each word i = 1, 2, 3, ... , m.

- **The circuit for matching** one word is shown in Fig. 9. Each cell requires two AND gates and one OR gate. The inverters for Aj and Kj are needed once for each column and are used for all bits in the column.

- **The output of all OR** gates in the cells of the same word go to the input of a common AND gate to generate the match signal for Mi. Mi will be logic 1 if a match occurs and 0 if no match occurs.

- **Note** that if the key register contains all 0' s, output Mj will be a 1 irrespective of the value of A or the word. This occurrence must be avoided during normal operation.

7. With the help of a diagram explain the working principle of semiconductor RAM

Ans.

RAM - Random Access Memory: As the names suggest, the RAM or random access memory is a form of semiconductor memory technology that is used for reading and writing data in any order - in other words as it is required by the processor. It is used for such applications as the computer or processor memory where variables and other stored and are required on a random basis. Data is stored and read many times to and from this type of memory.

Random access memory is used in huge quantities in computer applications as current day computing and processing technology requires large amounts of memory to enable them to handle the memory hungry applications used today. Many types of RAM including SDRAM with its DDR3, DDR4, and soon DDR5 variants are used in huge quantities.

8.Write short notes on

a) Virtual Memory

b) Cache Memory ,c) Secondary storage

Ans.

a) Virtual Memory

- Virtual memory is a feature of an operating system that enables a computer to be able to compensate shortages of physical memory by transferring pages of data from random access memory to disk storage.
- This process is done temporarily and is designed to work as a combination of RAM and space on the hard disk.

- This means that when RAM runs low, virtual memory can move data from it to a space called a paging file. This process allows for RAM to be freed up so that a computer can complete the task.
- Occasionally a user might be shown a message that says the virtual memory is running low, this means that either more RAM needs to be added, or the size of the paging file needs to be increased.

b) Cache Memory

- **Cache Memory** is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various

different independent caches in a CPU, which store instructions and data.

c) Secondary storage

- A secondary storage device refers to any non-volatile storage device that is internal or external to the computer. It can be any storage device beyond the primary storage that enables permanent data storage.
- A secondary storage device is also known as an auxiliary storage device, backup storage device, tier 2 storage, or external storage.
- Common types of secondary storage
  - magnetic storage devices, such as hard disk drives.
  - optical storage devices, such as CD, DVD and Blu-ray discs.
  - solid state storage devices, such as solid state drives and USB memory sticks.

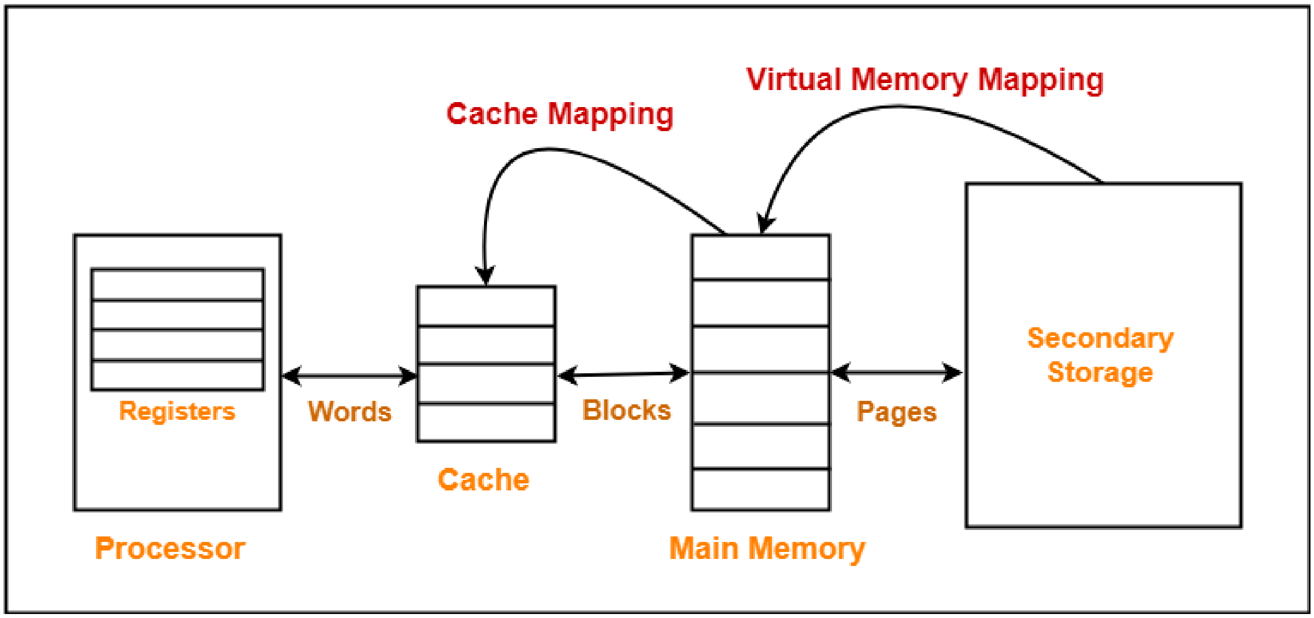
9. Write about different mapping techniques of cache memory.

Ans.

<u>Cache Mapping-</u>

Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

The following diagram illustrates the mapping process-



Cache Mapping Techniques-

Cache mapping is performed using following three different techniques-

1. Direct Mapping
2. Fully Associative Mapping
3. K-way Set Associative Mapping

1. Direct Mapping-

In direct mapping,

- A particular block of main memory can map only to a particular line of the cache.
- The line number of cache to which a particular block can map is given by-

Cache line number

= (Main Memory Block Address) Modulo (Number of lines in Cache)

2. Fully Associative Mapping-

In fully associative mapping,

- A block of main memory can map to any line of the cache that is freely available at that moment.
- This makes fully associative mapping more flexible than direct mapping.

3. K-way Set Associative Mapping-

In k-way set associative mapping,

- Cache lines are grouped into sets where each set contains k number of lines.
- A particular block of main memory can map to only one particular set of the cache.
- However, within that set, the memory block can map any cache line that is freely available.

- The set of the cache to which a particular block of the main memory can map is given by-

Cache set number

= (Main Memory Block Address) Modulo (Number of sets in Cache)

10.Explain about different types of memory and it’s hierarchy.

Ans.

1. The memory in a computer can be divided into five hierarchies based on speed as well as use. The processor can move from one level to another based on its requirements.

2. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes.

3. The first three hierarchies are volatile memories which means when there is no power, and then automatically they lose their stored data.

4. Whereas the last two hierarchies are not volatile which means they store the data permanently.

Memory Hierarchy Design

The memory hierarchy in computers mainly includes the following.

Registers

Most of the processors use a status word register as well as an accumulator. A status word register is used for decision making, and the accumulator is used to store the data like mathematical operation.

Usually, computers like complex instruction set computers have so many registers for accepting main memory, and RISC-reduced instruction set computers have more registers.

Cache Memory

Cache memory can also be found in the processor, however rarely it may be another IC (integrated circuit) that is separated into levels. The cache holds the chunk of data that is frequently used from the main memory.

Main Memory

The main memory in the computer is nothing but, the memory unit in the CPU that communicates directly. It is the main storage unit of the computer. This memory is fast as well as large memory used for storing the data throughout the operations of the computer. This memory is made up of RAM as well as ROM.

Magnetic Disks

The magnetic disks in the computer are circular plates fabricated of plastic otherwise metal by magnetized material. Frequently, two faces of the disk are utilized as well as many disks may be stacked on one spindle by reading or write heads obtainable on every plane. All the disks in the computer turn jointly at high speed. The tracks in the computer are nothing but bits that are stored within the magnetized plane in spots next to concentric circles. These are usually separated into sections that are named as sectors.

Magnetic Tape

This tape is a normal magnetic recording that is designed with a slender magnetizable covering on an extended, plastic film of the thin strip. This is mainly used to back up huge data. Whenever the computer requires to access a strip, first it will mount to access the data. Once the data is allowed, then it will be unmounted. The access time of memory will be slower within a magnetic strip as well as it will take a few minutes for accessing a strip.