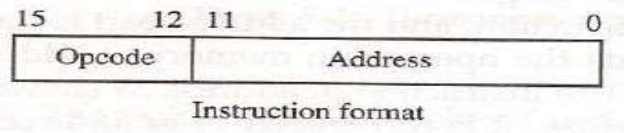
Unit-1

BASIC COMPUTER ORGANIZATION AND DESIGN

1. Instruction Codes:

- The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses.
- ***Program:*** set of instructions that specify the operations, operands, and the sequence by which processing has to occur.
- ***Instruction****:* a binary code that specifies a sequence of micro-operations for the computer.
- **Instruction Code:** group of bits that instruct the computer to perform specific operation.



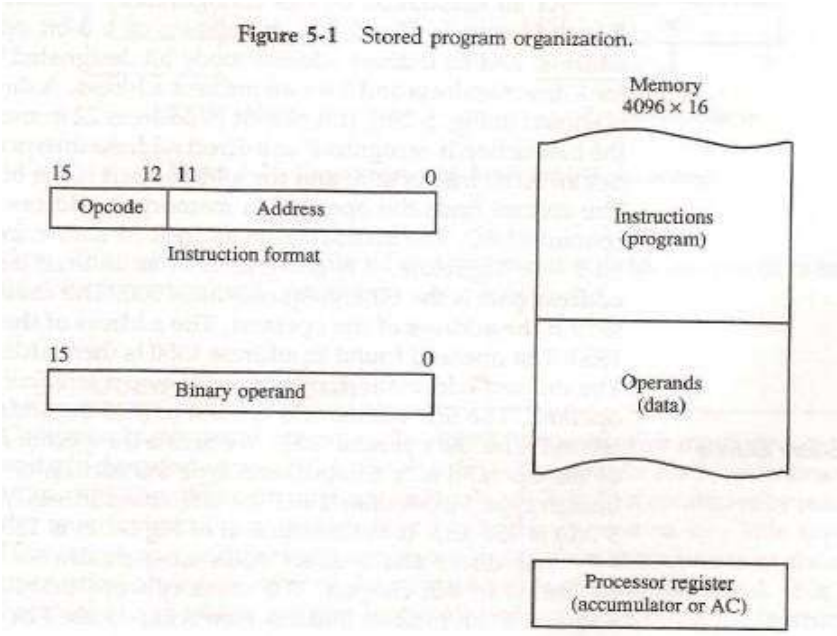
Instruction format

- Instruction code is usually divided into two parts: Opcode and address(operand)
  - *Operation Code (opcode):*
    - group of bits that define the operation
    - Eg: add, subtract, multiply, shift, complement.
    - No. of bits required for opcode depends on no. of operations available in computer.
    - n bit opcode >= 2^n (or less) operations
  - Address (operand):
    - specifies the location of operands (registers or memory words)
    - Memory words are specified by their address
    - Registers are specified by their k-bit binary code
    - k-bit address >= 2^k registers

Stored Program Organization:

- The ability to store and execute instructions is the most important property of a general-purpose computer. That type of stored program concept is called stored program organization.
- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address.

- The below figure shows the stored program organization



- Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words we need 12 bits to specify an address since 2^{12} = 4096.
- If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
- ***Accumulator (AC):***
  - ✓ Computers that have a single-processor register usually assign to it the name accumulator and label it AC.
  - ✓ The operation is performed with the memory operand and the content of AC.

Addressing of Operand:

- Sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand.
- When the second part of an instruction code specifies an operand, the instruction is said to have an***immediate operand.***
- When the second part specifies the address of an operand, the instruction is said to have a ***direct address.***
- When second part of the instruction designate an address of a memory word in which the address of the operand is found such instruction have ***indirect address.***
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- The instruction code format shown in Fig. 5-2(a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The mode bit is 0 for a direct address and 1 for an indirect address.

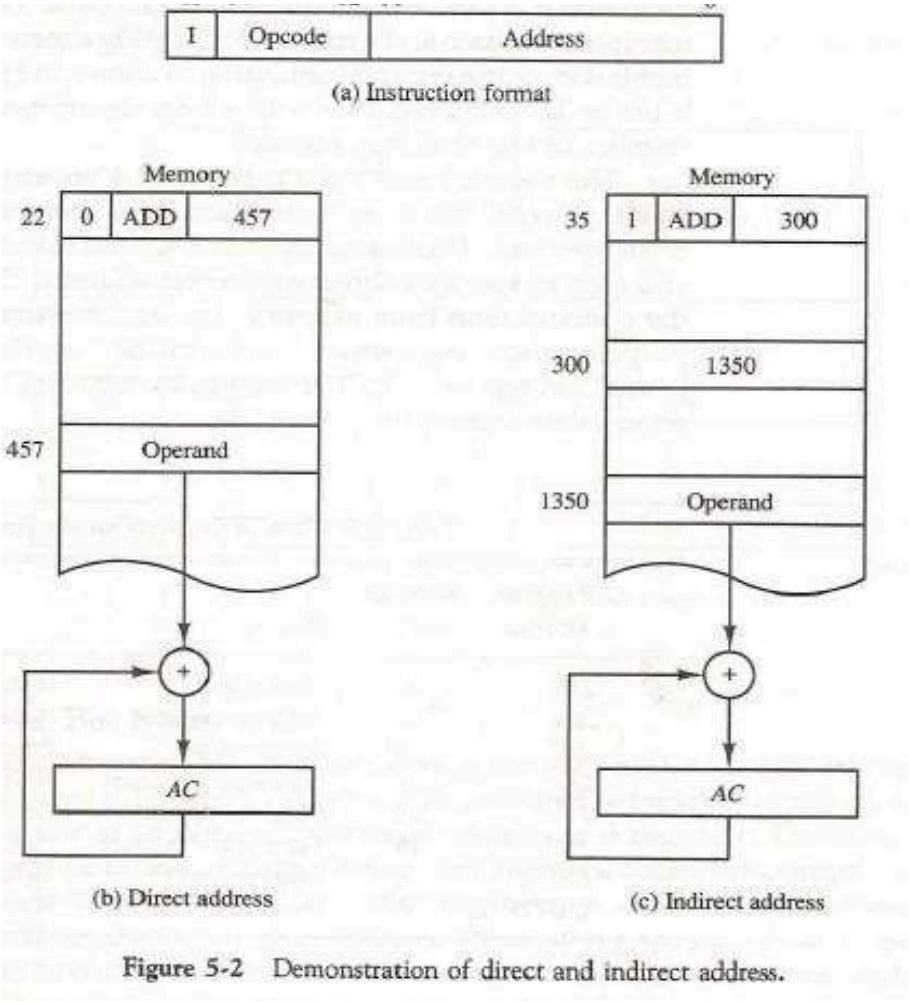


Figure 5-2 Demonstration of direct and indirect address.

- A direct address instruction is shown in Fig. 5-2(b).
- It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in Fig. 5-2(c) has a mode bit I = 1.
- Therefore, it is recognized as an indirect address instruction.
- The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350.
- The operand found in address 1350 is then added to the content of *AC.*
- The ***effective address*** to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction.
- Thus the effective address in the instruction of Fig. 5-2(b) is 457 and in the instruction of Fig 5-2(c) is 1350.

2. Computer Registers:

✓ Need of Computer Registers

- Instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed **(PC)**.
- Necessary to provide a register in the control unit for storing the instruction code after it is read from memory **(IR)**.
- Needs processor registers for manipulating data (AC and TR) and a register for holding a memory address **(AR)**.

✓ The above requirements dictate the registers, their configuration, number of bits and their uses

TABLE 5-1 List of Registers for the Basic Computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

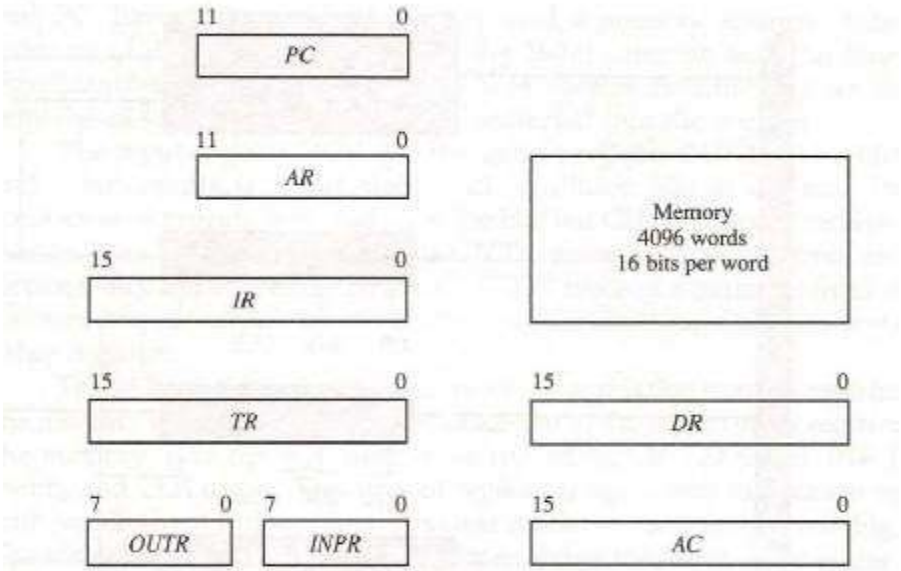


Figure 5-3 Basic computer registers and memory.

✓ The *data register (DR)* holds the operand read from memory.

✓ The *accumulator (AC)* register is a general purpose processing register.

✓ The instruction read from memory is placed in the *instruction register (IR)*.

- ✓ The *temporary register (TR)* is used for holding temporary data during the processing.
- ✓ The *memory address register (AR)* has 12 bits since this is the width of a memory address.
- ✓ The *program counter (PC)* also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.
- ✓ Two registers are used for input and output.
  - The *input register (INPR)* receives an 8-bit character from an input device.
  - The *output register (OUTR)* holds an 8-bit character for an output device.

Common Bus System:

- ✓ The basic computer has eight registers, a memory unit, and a control unit
- ✓ Paths must be provided to transfer information from one register to another and between memory and registers.

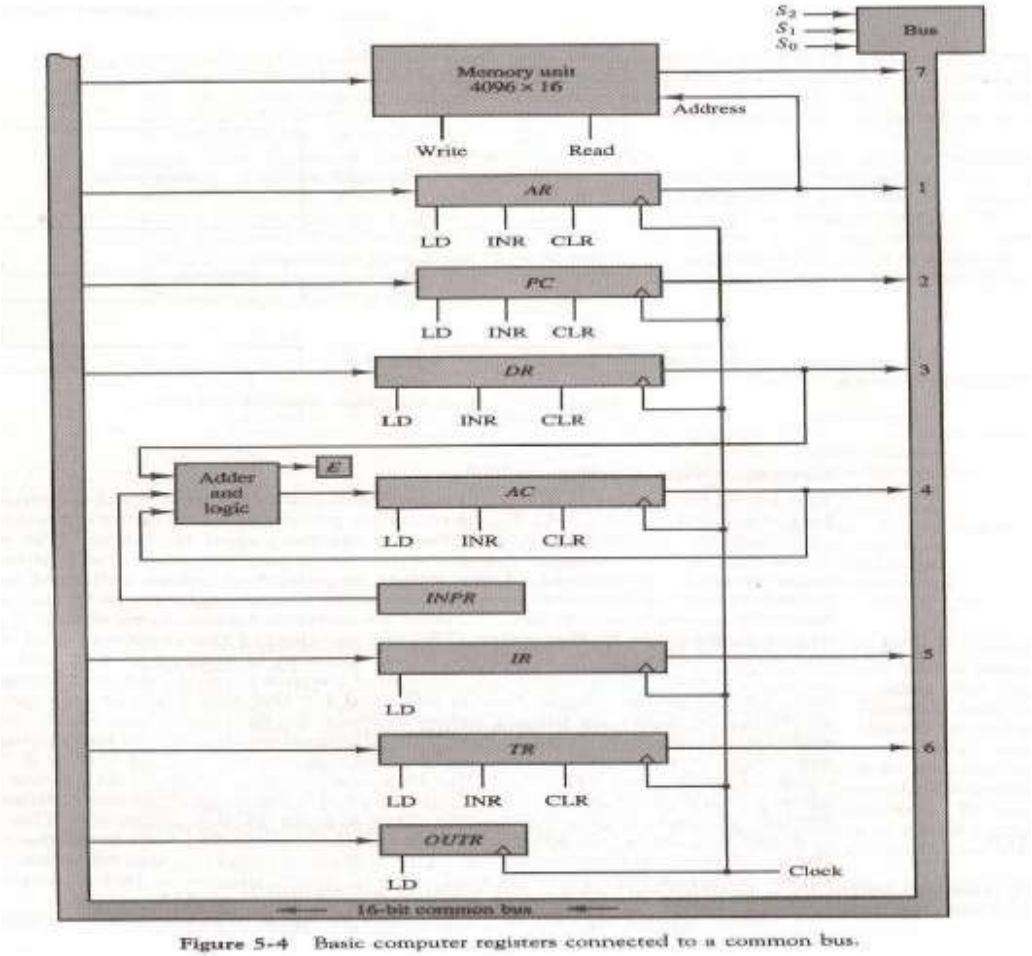


Figure 5-4 Basic computer registers connected to a common bus.

- ➢ A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- ➢ The connection of the registers and memory of the basic computer to a

common bus system is shown in Fig. 5-4.

- The outputs of seven registers and memory are connected to the common bus.
- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.
- The number along each output shows the decimal equivalent of the required binary selection.
- For example, the number along the output of *DR* is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0 = 011$.
- The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.
- The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$.
- Two registers, AR and *PC*, have 12 bits each since they hold a memory address.
- When the contents of *AR* or *PC* are applied to the 16-bit common bus, the four most significant bits are set to 0's.
- When AR or *PC* receives information from the bus, only the 12 least significant bits are transferred into the register.
- The input register *INPR* and the output register OUTR have 8 bits each.
- They communicate with the eight least significant bits in the bus.
- INPR is connected to provide information to the bus but OUTR can only receive information from the bus.
- This is because INPR receives a character from an input device which is then transferred to *AC*.
- *OUTR* receives a character from AC and delivers it to an output device.
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).
- This type of register is equivalent to a binary counter with parallel load and synchronous clear.
- Two registers have only a LD input.
- The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR.
- Therefore, AR must always be used to specify a memory address.
- The 16 inputs of *AC* come from an adder and logic circuit. This circuit has three sets of inputs.
    - One set of 16-bit inputs come from the outputs of *AC*.
    - Another set of 16-bit inputs come from the data register *DR*.

- The result of an addition is transferred to *AC* and the end carry-out of the addition is transferred to flip-flop E (extended *AC* bit).
- A third set of 8-bit inputs come from the input register INPR.

➢ The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.

➢ For example, the two microoperations DR□AC and AC □ DR can be executed at the same time.

➢ This can be done by placing the content of *AC* on the bus (with $S_2S_1S_0 = 100$), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of *AC*, all during the same clock cycle.

3. Computer Instructions:

✓ The basic computer has three instruction code formats, as shown in Fig. 5-5. Each format has 16 bits.

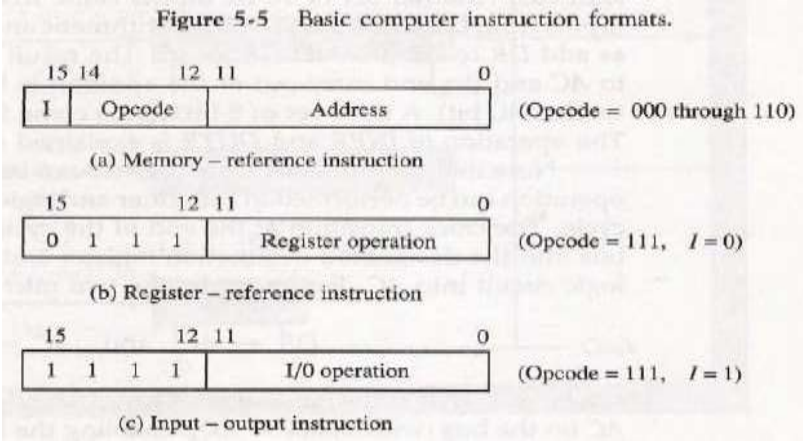


Figure 5-5 Basic computer instruction formats.

✓ The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

✓ A **memory-reference instruction** uses 12 bits(0-11) to specify an address, next 3 bits for operation code(opcode) and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.

Memory Reference Instructions

	Hexadecimal code		
Symbol	I = 0	I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	ADD memory word to AC
LDA	2xxx	Axxx	LOAD Memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and Skip if zero

- ✓ The **register-reference instructions** are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.

  ✓ A register-reference instruction specifies an operation on the AC register. So an operand from memory is not needed. Therefore, the other 12(0-11) bits are used to specify the operation to be executed.

Register Reference Instructions

Symbol	Hexadecimal code	Description
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right AC and E
CIL	7040	Circulate left AC and E
INC	7020	Increment AC
SPA	7010	Skip next instruction if AC positive
SNA	7008	Skip next instruction is AC is negative
SZA	7004	Skip next instruction is AC is 0
SZE	7002	Skip next instruction is E is 0
HLT	7001	Halt computer

- ✓ An **input—output instruction** does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction.
- ✓ The remaining 12 bits are used to specify the type of input—output operation.

I/O Reference Instructions

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

Instruction Set Completeness:

- ✓ A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function.
- ✓ The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- Arithmetic, logical, and shift instructions
- Data Instructions (for moving information to and from memory and processor registers)
- Program control or Brach
- Input and output instructions

✓ There is one arithmetic instruction, ADD, and two related instructions, complement AC(CMA) and increment AC(INC). With these three instructions we can add and subtract binary numbers when negative numbers are in signed-2's complement representation.

✓ The circulate instructions, CIR and CIL; can be used for arithmetic shifts as well as any other type of shifts desired.

✓ There are three logic operations: AND, complement AC (CMA), and clear AC(CLA). The AND and complement provide a NAND operation.

✓ Moving information from memory to *AC* is accomplished with the load AC (LDA) instruction. Storing information from AC into memory is done with the store AC (STA) instruction.

✓ The branch instructions BUN, BSA, and ISZ, together with the four skip instructions, provide capabilities for program control and checking of status conditions.

✓ The input (INP} and output (OUT) instructions cause information to be transferred between the computer and external devices.

4. Timing and Control:

✓ The timing for all registers in the basic computer is controlled by a master clock generator.

✓ The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.

✓ The clock pulses do not change the state of a register unless the register is enabled by a control signal.

✓ The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.

- There are two major types of control organization: *Hardwired control and Microprogrammed control*

✓ The differences between hardwired and microprogrammed control are

Hardwired control	Microprogrammed control
✓ The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.	✓ The control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
✓ The advantage that it can be optimized to produce a fast mode of operation.	✓ Compared with the hardwired control operation is slow.
✓ Requires changes in the wiring among the various components if the design has to be modified or changed.	✓ Required changes or modifications can be done by updating the microprogram in control memory.

- ✓ The block diagram of the hardwired control unit is shown in Fig. 5-6.
- ✓ It consists of two decoders, a sequence counter, and a number of control logic gates.
- ✓ An instruction read from memory is placed in the instruction register (IR). It is divided into three parts: The I bit, the operation code, and bits 0 through 11.
- ✓ The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7.
- ✓ Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- ✓ Bits 0 through 11 are applied to the control logic gates.
- ✓ The 4-bit sequence counter can count in binary from 0 through 15.

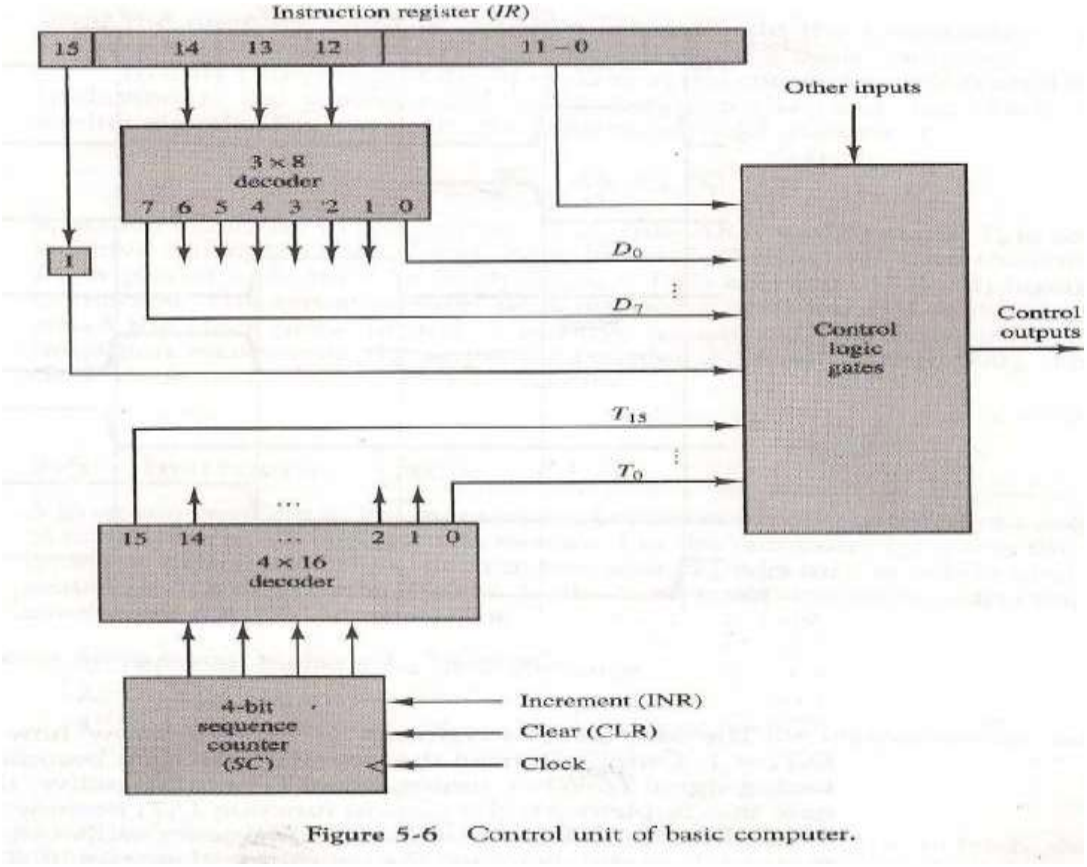


Figure 5-6 Control unit of basic computer.

- ✓ The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.
- ✓ The sequence counter *SC* can be incremented or cleared synchronously.
- ✓ The counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder.
- ✓ As an example, consider the case where *SC* is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, *SC* is cleared to 0 if decoder output D3 is active.
- ✓ This is expressed symbolically by the statement

  D_3T_4: SC□0
- ✓ The timing diagram of Fig. 5-7 shows the time relationship of the control signals.
- ✓ The sequence counter *SC* responds to the positive transition of the clock.
- ✓ Initially, the CLR input of *SC* is active. The first positive transition of the clock clears *SC* to 0, which in turn activates the timing signal T_0 out of the decoder. T_0 is active during one clock cycle.
- ✓ SC is incremented with every positive clock transition, unless its CLR input is active.
- ✓ This produces the sequence of timing signals T_0, T_1, T_2, T_3, $T_4$and so on, as shown in the diagram.
- ✓ The last three waveforms in Fig.5-7 show how *SC* is cleared when $D_3T_4 = 1$.
- ✓ Output D_3 from the operation decoder becomes active at the end of timing signal T_2.
- ✓ When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active.
- ✓ This signal is applied to the CLR input of *SC*. On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to 0.
- ✓ This causes the timing signal T_0 to become active instead of T_5 that would have been active if *SC* were incremented instead of cleared.

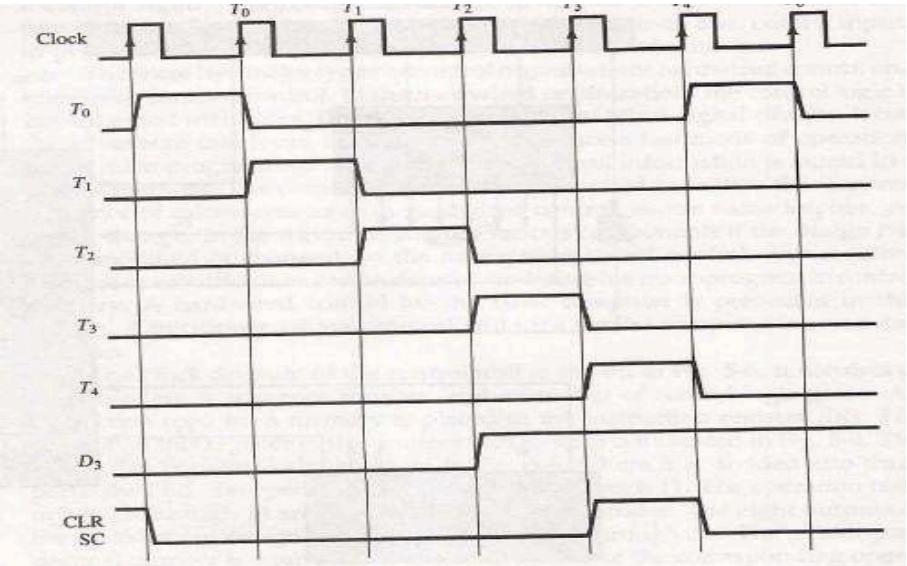


Figure 5-7 Example of control timing signals.

5. Instruction Cycle:

- ✓ A program residing in the memory unit of the computer consists of a sequence of instructions.
- ✓ The program is executed in the computer by going through a cycle for each instruction.
- ✓ Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.
- ✓ In the basic computer each instruction cycle consists of the following phases:
    1. Fetch an instruction from memory.
    2. Decode the instruction.
    3. Read the effective address from memory if the instruction has an indirect address.
    4. Execute the instruction.
- ✓ Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.

Fetch and Decode:

- ✓ Initially, the program counter PC is loaded with the address of the first instruction in the program.
- ✓ The sequence counter *SC* is cleared to 0, providing a decoded timing signal T_0.
- ✓ The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

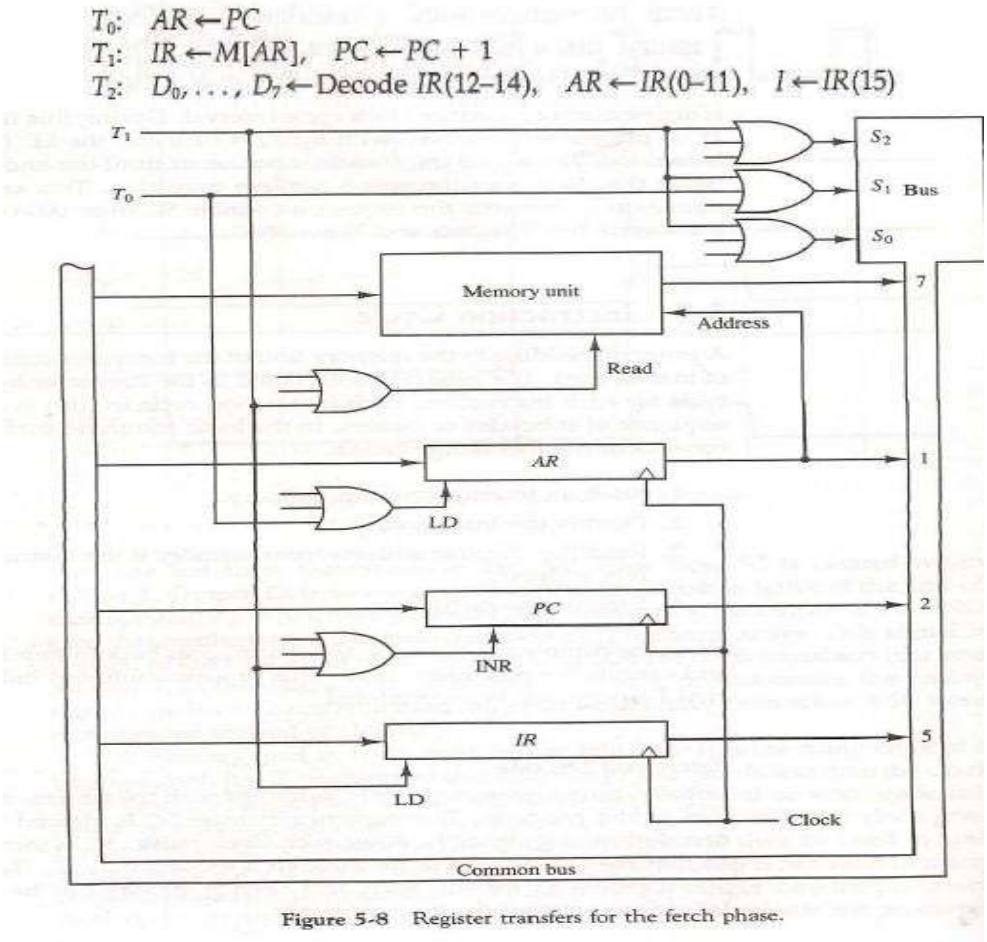


Figure 5-8 Register transfers for the fetch phase.

- ✓ Figure 5-8 shows how the first two register transfer statements are implemented in the bus system.
- ✓ To provide the data path for the transfer of *PC* to AR we must apply timing signal T_0 to achieve the following connection:
    - Place the content of *PC* onto the bus by making the bus selection inputs S_2, S_1, S_0 equal to 010.
    - Transfer the content of the bus to AR by enabling the LD input of AR.
- ✓ In order to implement the second statement it is necessary to use timing signal T_1 to provide the following connections in the bus system.
    - Enable the read input of memory.
    - Place the content of memory onto the bus by making $S_2S_1S_0$=111.
    - Transfer the content of the bus to IR by enabling the LD input of IR.
    - Increment PC by enabling the INR input of PC.
- ✓ Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the Type of Instruction:

- Initialise SC =0.

- Fetch phase

- At time T0 get the address of the next instruction to be executed from the Program Counter (PC).

$$AR \leftarrow PC$$

- At time T1 get the instruction from the memory address in AR into the Instruction Register (IR) and increment PC.

$$IR \leftarrow M[AR]$$
$$PC \leftarrow PC + 1$$

- Decode phase

  - At time T2 decode the instruction by considering 12-14 bits of IR as Opcode, copy 0-11 bits of IR to AR and 15^{th} bit to I flip flop.

- Decoding Phase at time T3 determines the type of instruction read from memory.

- Decoder output D_7 is equal to 1 the instruction is a register reference or memory reference instruction.

  - If I = 0 it is a register reference instruction.

  - If I = 1 it is a I/O instruction. Execute the instruction and set SC =0.

- If D7 = 0 it is a memory reference instruction and the mode of address is determined from the I value.

  - If I = 1, we have a memory reference instruction with an indirect address. The effective address is read from memory using the microoperation

$$AR \leftarrow M[AR]$$

  - If D7=0 and I=0 Nothing is done . Execute the instruction and set SC=0.

After executing the instruction control shifts to process the next instruction.

$D_7' IT_3$: $AR \leftarrow M[AR]$
$D_7' I' T_3$: Nothing
$D_7 I' T_3$: Execute a register-reference instruction
$D_7 IT_3$: Execute an input–output instruction

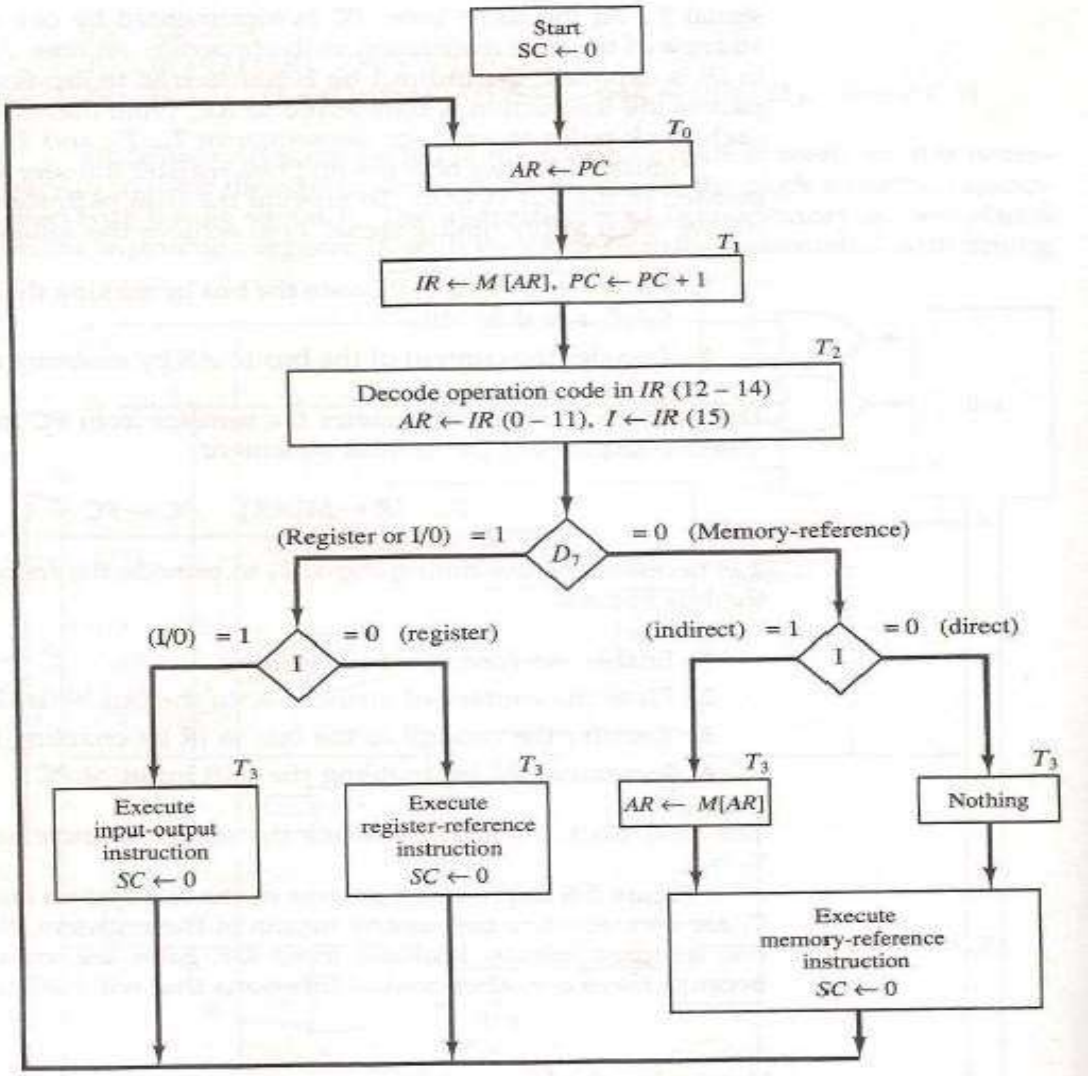


Figure 5-9 Flowchart for instruction cycle (initial configuration).

6. Microprogrammed Control (Control Memory)

- **Microinstruction** : Each word in control memory contains within it a microinstruction.
- **Microoperation** : A microinstruction specifies one or more microoperations.
- **Microprogram** : A sequence of microinstructions forms what is called a microprogram.
- A computer that employs a microprogrammed control unit will have two separate memories:

  1. The main memory : This memory is available to the user for storing programs. The user's program in main memory consists of machine instructions and data.

  2. The control memory : This memory contains a fixed microprogram that cannot be alter by the user. The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations.

- Block diagram of microprogrammed control unit

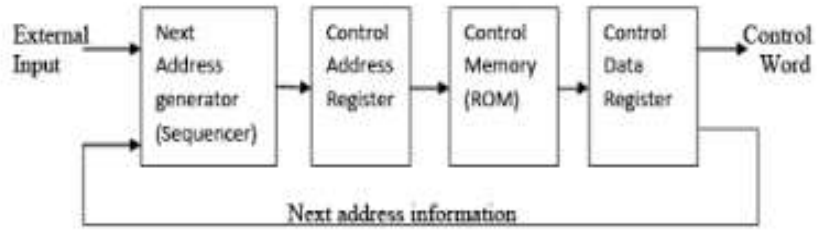


Fig 3-1: Microprogrammed Control Organization

- **Next Address Generator(Sequencer)**: It is used to generate the address of the next instruction to compute. The microinstruction contains bits to initiate the microoperations and bits to determine the address sequence for control memory. The next address can be determined as

  - Loading an initial address to start the control operations.

  - Incrementing the control address register

  - Loading an address from control memory

  - Transferring an external address.

- **Control Address Register** : It is used to hold the address of the instruction to be executed. It passes the address to the control memory.
- **Control Data Register**: It is holds the present microinstruction . It is also called Pipeline register. It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.
- **Control Memory** : Control Memory is the storage in the microprogrammed control unit to store the microprogram. It is a ROM in which all control information is permanently stored.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address.

- **Advantage**: Once the hardware configuration is established no need for further hardware or wiring changes. For different control sequence a different set of microinstructions is used in control memory.

7. Address Sequencing

- Microinstructions are usually stored in groups where each group specifies a routine, where each routine specifies how to carry out an instruction.
- Each routine must be able to branch to the next routine in the sequence.
- An initial address is loaded into the CAR when power is turned on; this is usually the address of the first microinstruction in the instruction fetch routine.
- Next, the control unit must determine the effective address of the instruction.

Mapping :

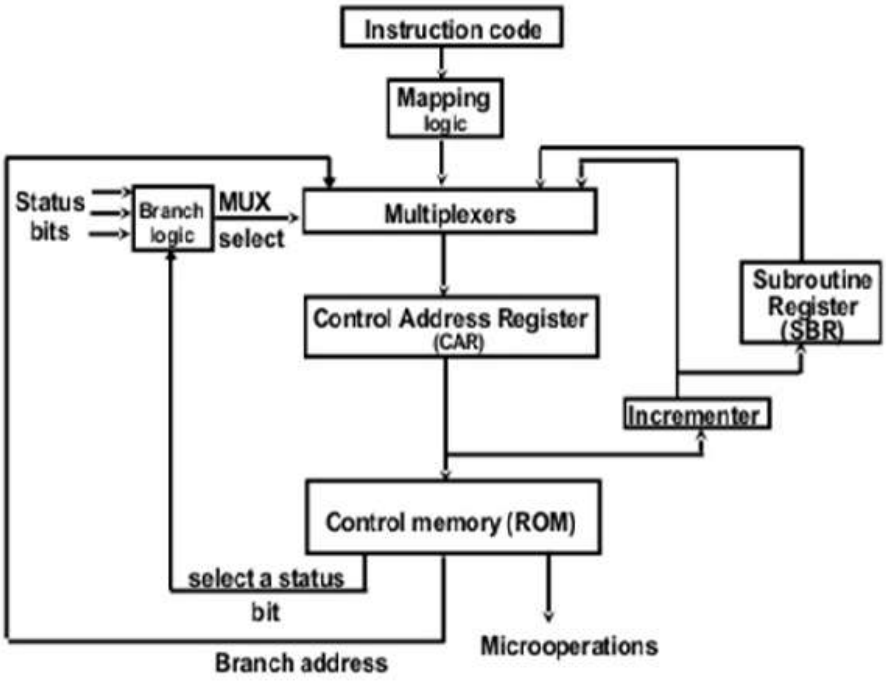
- The next step is to generate the microoperations that executed the instruction.

  – This involves taking the instruction's opcode and transforming it into an address for the the instruction's microprogram in control memory. This process is called mapping.

  – While microinstruction sequences are usually determined by incrementing the CAR, this is not always the case. If the processor's control unit can support subroutines in a microprogram, it will need an external register for storing return addresses.

- When instruction execution is finished, control must be return to the fetch routine. This is done using an unconditional branch.

- Addressing sequencing capabilities of control memory include:
  – Incrementing the CAR
  – Unconditional and conditional branching (depending on status bit).
  – Mapping instruction bits into control memory addresses
  – Handling subroutine calls and returns.



Conditional Branching

- Status bits

  – provide parameter information such as the carry-out from the adder, sign of a number, mode bits of an instruction, etc.

  – control the conditional branch decisions made by the branch logic together with the field in the microinstruction that specifies a branch address.

Branch Logic

- Branch Logic - may be implemented in one of several ways:

– The simplest way is to test the specified condition and branch if the condition is true; else increment the address register.

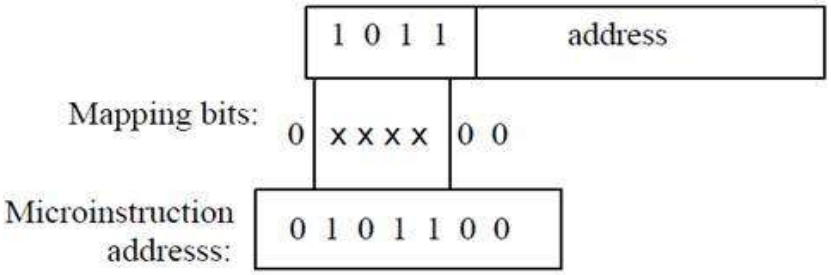
– This is implemented using a multiplexer:

- If the status bit is one of eight status bits, it is indicated by a 3-bit select number.
- If the select status bit is 1, the output is 0; else it is 0.
- A 1 generates the control signal for the branch; a 0 generates the signal to increment the CAR.
- Unconditional branching occurs by fixing the status bit as always being 1.

Mapping of Instruction

- Branching to the first word of a microprogram is a special type of branch. The branch is indicated by the opcode of the instruction.
- The mapping scheme shown in the figure allows for four microinstruction as well as overflow space from 1000000 to 1111111.

Mapping of Instruction Code to Microoperation address



Subroutines

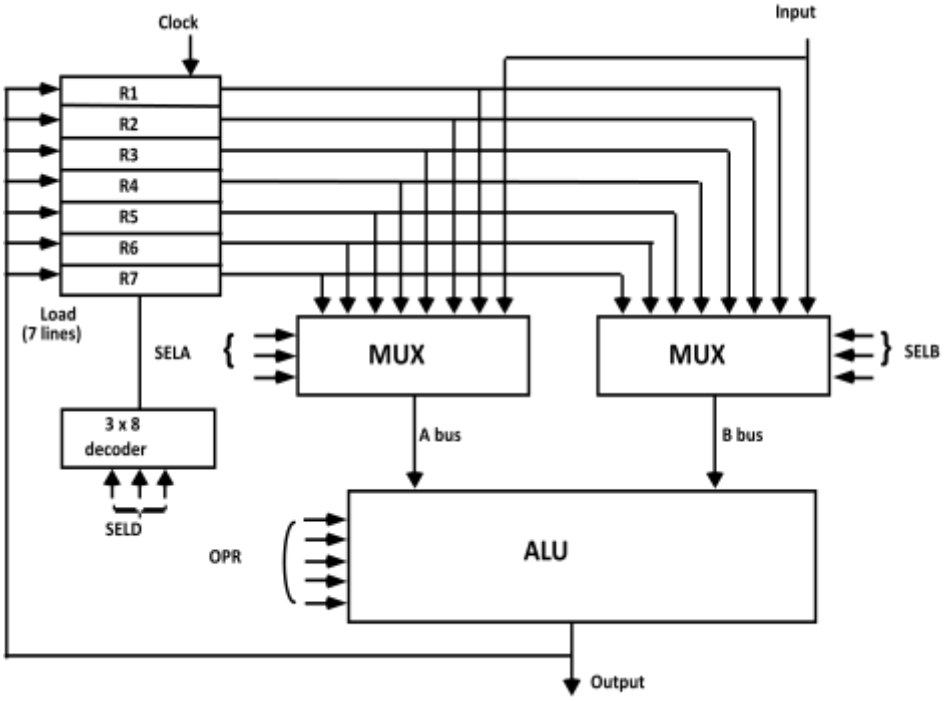
- Subroutine calls are a special type of branch where we return to one instruction below the calling instruction.

– Provision must be made to save the return address, since it cannot be written into ROM

UNIT – II

CENTRAL PROCESSING UNIT

1. GENERAL REGISTER ORGANIZATION



- Generally CPU has seven general registers.
- Register organization show how registers are selected and how data flow between register and ALU.
- A decoder is used to select a particular register.The output of each register is connected to two multiplexers to form the two buses A and B.
- The selection lines in each multiplexer select the input data for the particular bus.
- The A and B buses form the two inputs of an ALU.The operation select lines decide the micro operation to be performed by ALU.
- The result of the micro operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

OPERATION OF CONTROL UNIT

Example: R1 <- R2 + R3

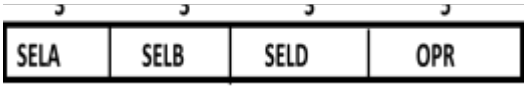
[1] MUX A selector (SELA): BUS A ← R2

[2] MUX B selector (SELB): BUS B ← R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 ← Out Bus

Control Word



Encoding of register selection fields

Binary	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

ALU CONTROL

Encoding of ALU Operations:

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

Examples of ALU Microoperations

TABLE 3 Examples of Microoperations for the CPU

Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

2. STACK ORGANIZATION

- The two operations that are performed on stack are the insertion and deletion.
  - The operation of insertion is called **PUSH.**
  - The operation of deletion is called **POP**.
- These operations are simulated by incrementing and decrementing the **stack pointer register (SP)**.

Adv:

- Efficient computation of complex arithmetic expressions.
- Execution of instructions is fast because operand data are stored in consecutive memory locations.
- Length of instruction is short as they do not have address field.

Disadv:

- The size of the program increases.

REGISTER STACK ORGANIZATION

- 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.
- The figure shows the organization of a 64-word register stack

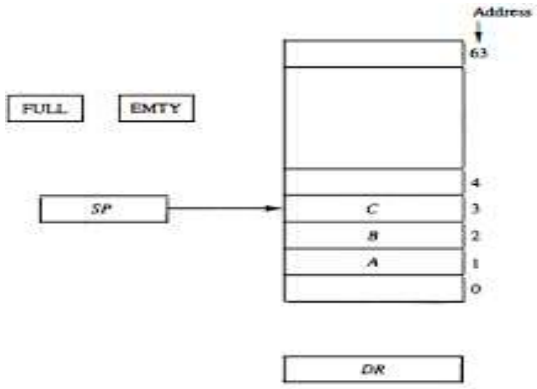


Figure 3 Block diagram of a 64-word stack.

- The stack pointer register SP contains a binary number whose value is equal to the address of the word is currently on top of the stack.
- Three items are placed in the stack: A, B, C, in that order.
- In above figure C is on top of the stack so that the content of SP is 3.
- For removing the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of stack SP.
- Now the top of the stack is B, so that the content of SP is 2.
- Similarly for inserting the new item, the stack is pushed by incrementing SP and writing a word in the nexthigher location in the stack.
- In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.
- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary).

- When 63 is incremented by 1, the result is 0 since 111111 + 1. = 1000000 in binary, but SP can accommodate only the six least significant bits.
- Then the one-bit register FULL is set to 1, when the stack is full.
- Similarly when 000000 is decremented by 1, the result is 111111, and then the one-bit register EMTY is set 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.

PUSH:

- 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:

$$\boldsymbol{SP \leftarrow SP + 1}$$

$$\boldsymbol{M[SP] \leftarrow DR}$$

$$\boldsymbol{IF(SP = 0) then\ (FULL \leftarrow 1)}$$

$$\boldsymbol{EMTY \leftarrow 0}$$

- The stack pointer is incremented so that it points to the address of next-higher word.
- A memory write operation inserts the word from DR the top of the stack.
- The first item stored in the stack is at address 1. The last item is stored at address 0.
- If SP reaches 0, the stack is full of items, so FULL is to 1. This condition is reached if the top item prior to the last push way 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, so the EMTY is cleared to 0.

POP:

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

$$\boldsymbol{DR \leftarrow M[SP]}$$

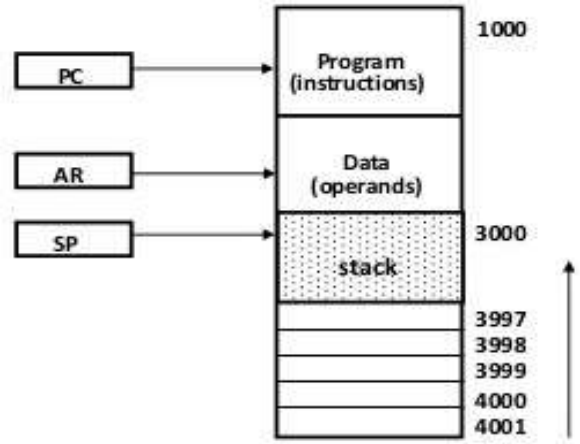
$$\boldsymbol{SP \leftarrow SP + 1}$$

$$\boldsymbol{IF(SP = 0) then\ (EMTY \leftarrow 1)}$$

$$\boldsymbol{FULL \leftarrow 0}$$

- 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 1. This condition is reached if the item read was in location 1. Once this it is read out, SP is decremented and reaches the value 0, which is the initial value of SP.
- If a pop operation reads the item from location 0 and then is decremented, SP changes to 111111, which is equivalent to decimal 63 in above configuration, the word in address 0 receives the last item in the stack.

MEMORY STACK ORGANIZATION



ı a

- A portion of memory is used as a stack with a processor register as a stack pointer
- The program counter PC points at the address of the next instruction in program. PC is used during the fetch phase to read an instruction.
- The address register AR points at an array of data. AR is used during the exec phase to read an operand.
- The stack pointer SP points at the top of the stack. SP is used to push or pop items into or from stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory.
- As shown in Fig., the initial value of SP is 4001 and the stack grows with decreasing addresses.¬ Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last¬ address that can be used for the stack is 3000. No provisions are available for stack limit checks.
- A new item is inserted by the PUSH operation. 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 stack.¬

  $$SP \leftarrow SP - 1$$

  $$M[SP] \leftarrow DR$$

- An element is deleted from the stack using POP operation. The top item is read from the stack into DR. The stack pointer is then decremented to point at the next item in the stack.

  $$DR \leftarrow M[SP]$$

  $$SP \leftarrow SP + 1$$

- Most computers do not provide hardware to check stack overflow (full stack) or underflow(empty stack).

Reverse Polish Notation:

A stack organization is very effective for evaluating arithmetic expressions.

The common arithmetic expressions are written in infix notation, with each operator written between the operands. An expression can also be expressed in Prefix or Postfix notation as follows:

Eg: A+B -----> Infix notation

+AB -----> Prefix or Polish notation

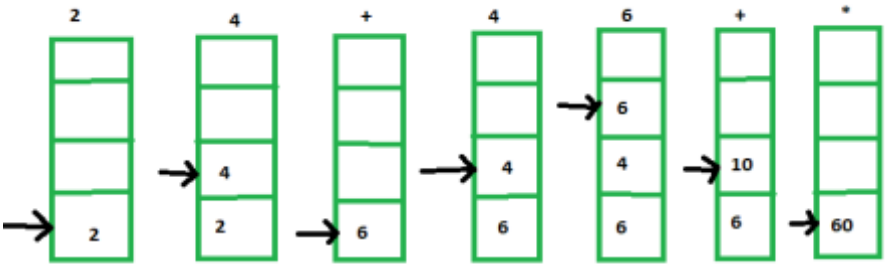
AB+ ----> Post or reverse Polish notation

Evaluation of Arithmetic Expressions:

- Convert the expression from infix to postfix form.
- Scan the expression from left to right.
- When an operand is found PUSH it into the stack.
- When an operator is encountered POP two operands from the stack and perform the operation and PUSH the result into the stack.
- When the expression is scanned completely the final result is found in the TOP of the Stack.
- Consider the arithmetic expression

  (2+4) * (4+6)
- In reverse polish notation, it is expressed as

  24 + 46 + *



Stack operations to evaluate (2+4)*(4+6)

3. Instruction Formats:

- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:
- An operation code field that specifies the operation to be perform
- An address field that designates a memory address or a processor register.
- A mode field that specifies the way the operand or the effective address is determined.
- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruct format of a computer depends on the internal organization of its registers.

- Most computers fall into one of three types of CPU organizations:

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

Single Accumulator Organization:

- In an accumulator type organization all the operations are performed with an implied accumulator register.
- The instruction format in this type of computer uses one address field.
- For example, the instruction that specifies an arithmetic addition defined by an assembly language instruction as

  ADD X← AC

  where X is the address of the operand. The ADD instruction in this case results in the operation AC +M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.
- General register organization:
- The instruction format in this type of computer needs three register address fields.
- Thus the instruction for an arithmetic addition may be written in an assembly language as

  ADD R1, R2, R3

  to denote the operation R1← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Only register addresses for R1 and R2 need be specified in this instruction.

Stack organization:

- The stack-organized CPU has PUSH and POP instructions which require an address field.
- Thus the instruction PUSH X will push the word at address X to the top of the stack.
- The stack pointer is updated automatically.
- Stack type instructions do not need an address field in stack-organized computers since all operands are implied to be in the top of the stack.

Most computers fall into one of the three types of organizations. Some computers combine features from more than one organizational structure.

- On the basis of number of address, instruction are classified as:

  - Three Address Instructions
  - Two Address Instructions
  - One Address Instructions
  - Zero Address Instructions

Three Address Instruction

- 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.
- Makes Program creation easier.
- Programs run much slow because instruction contains too many bits of information
- Expression: X = (A+B)*(C+D)
- R1, R2 are registers. M[] is any memory location
- It can be implemented using three address instruction as

```
ADD   R1, A, B      //R1 = M[A] + M[B]
ADD   R2, C, D      //R2 = M[C] + M[D]
MUL   X, R1, R2     //M[X] = R1 * R2
```

Two Address Instruction

- This is common in commercial computers. Here two address can be specified in the instruction.
- Expression: X = (A+B)*(C+D)
- R1, R2 are registers . M[] is any memory location.
- Program using two address instruction

```
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
```

One address instruction

- 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.
- Expression: X = (A+B)*(C+D)
- AC is accumulator. M[] is any memory location M[T] is temporary location.
- Program using one address instruction

```
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
```

Zero address instruction

- 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
- Program using zero address instruction

```
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
```

4. ADDRESSING MODES

- Addressing modes refers to the way in which the operand of an instruction is specified.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

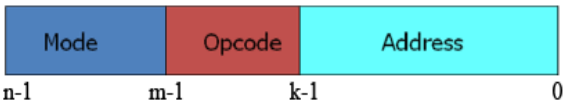


Fig: Instruction format with mode field

- The purpose of using addressing modes is as follows:
  - To give the programming versatility to the user.
  - To reduce the number of bits in addressing field of instruction.

Types Of Addressing Mode

- Implied Mode
- Immediate Mode
- Register Mode
- Register Indirect Mode
- Auto-Increment or Auto-Decrement Mode
- Direct Mode
- Indirect Mode
- Relative Addressing Mode
- Indexed Addressing mode
- Base Register Addressing Mode

Implied Mode

In this addressing mode, the instruction itself specifies the operands implicitly. It is also called as implicit addressing mode. All register-reference instructions that make use of the Accumulator are Implied Mode instructions.

Examples:

- The instruction "Complement Accumulator" is an implied mode instruction.
- RAL – Rotate Left with Carry

Immediate Mode

In this addressing mode, the operand is specified in the instruction explicitly. Instead of address field, an operand field is present that contains the operand.

- Examples:
  - ADD 10 will increment the value stored in the accumulator by 10.
  - MOV R,20 initializes register R to a constant value 20.

Register Mode

In this mode the operand is stored in the register and this register is present in CPU. The instruction has the address of the Register where the operand is stored.

Advantages

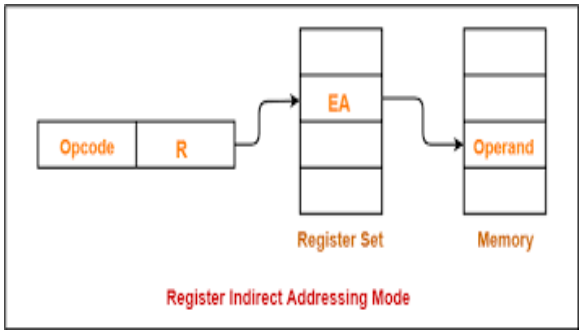
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

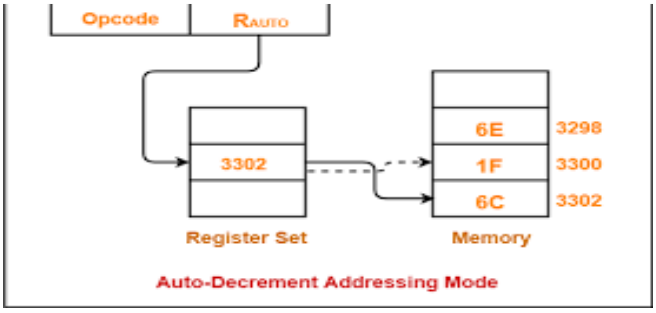
Register Indirect Mode

- In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.



Auto increment or auto decrement mode

Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of the register are automatically incremented or automatically decremented to the next consecutive memory location. Example: Add R1, (R2)+, Add R1,-(R2)



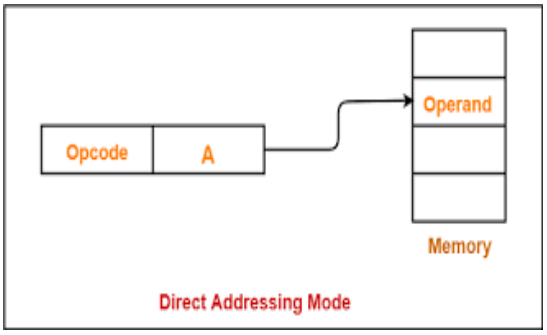
Effective Address

- The memory address of an operand consists of two components:
  - Starting address of memory segment.
  - Effective address or Offset: An offset is determined by adding any combination of three address elements: displacement, base and index.
- Displacement: It is an 8 bit or 16 bit immediate value given in the instruction.
- Base: Contents of base register, BX or BP.
- Index: Content of index register SI or DI.

Direct Addressing Mode

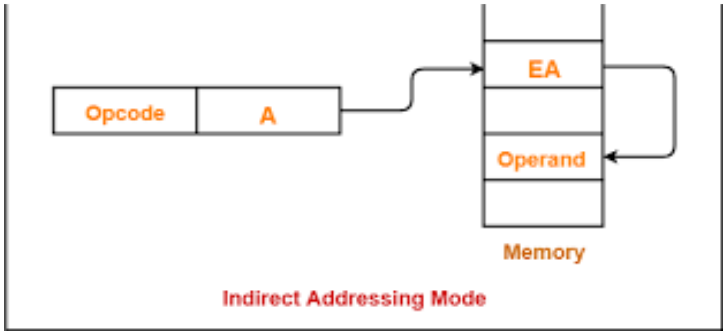
- In this mode, effective address of operand is present in instruction itself. Single memory reference to access data.No additional calculations to find the effective address of the operand.

  Example: ADD R1, 4000 - In this the 4000 is effective address of operand.



Indirect Addressing Mode

- In this, the address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.

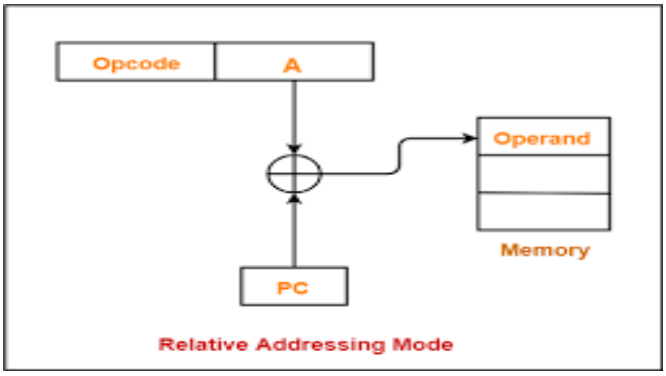


Relative Addressing Mode

In this the contents of PC(Program Counter) is added to address part of instruction to obtain the effective address.

EA = A + (PC), where EA is effective address and PC is program counter.

The operand is A cells away from the current cell(the one pointed to by PC)

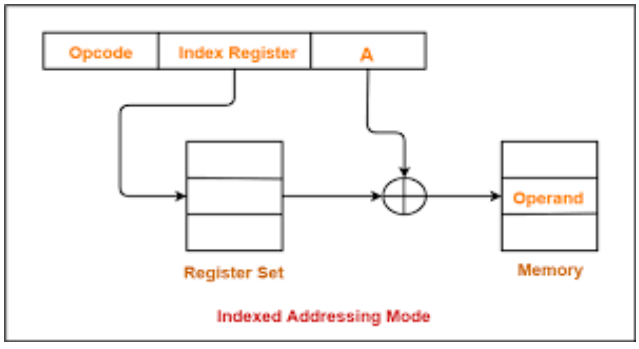


Indexed Addressing Mode

- In this the contents of the indexed register is added to the Address part of the instruction, to obtain the effective address of operand.

  EA = A + (R),

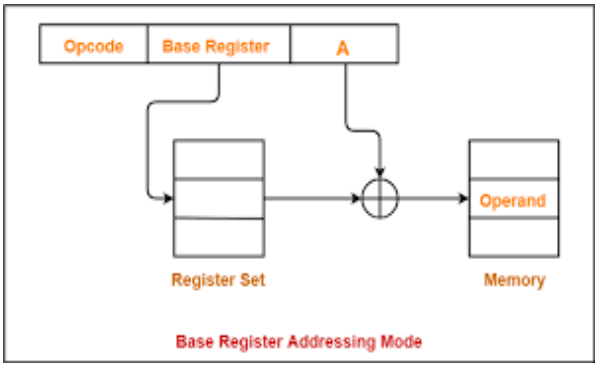
  In this the address field holds two values, A(which is the base value) and R(that holds the displacement), or vice versa.



Base Register Addressing Mode

- It is again a version of Displacement addressing mode. This can be defined

as EA = A + (R), where A is displacement and R holds pointer to base address.



5. DATA TRANSFER AND MANIPULATION

Most computer instructions can be classified into three categories

- Data transfer,
- Data manipulation,
- Program control instructions

Data Transfer Instruction

- Data transfer instructions move data from one place in the computer to another without changing the data content
- The most common transfers are between memory and processor registers,between processor registers and input or output,and between the processor registers themselves.

Typical Data Transfer Instruction :

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

- **Load** : transfer from memory to a processor register, usually an AC (memory read)
- **Store** : transfer from a processor register into memory (memory write)
- **Move** : transfer from one register to another register » Exchange : swap information between two registers or a register and a memory word
- **Input/Output** : transfer data among processor registers and input/output device
- **Push/Pop** : transfer data between processor registers and a memory stack

Data Manipulation Instruction

- Data Manipulation Instructions perform operations on data and provide the computational capabilities for the computer.
- It is divided into three basic types:
    1) Arithmetic,
    2) Logical and bit manipulation,
    3) Shift Instruction

Arithmetic Instructions

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and bit manipulation Instructions

- Logical instructions performs binary operations on strings of bits stored in registers.
- They are useful for manipulating individual bits or group of bits that represent binary coded information.

Name	Mnemonic
CLEAR	CLR
COMPLEMENT	COM
AND	AND
OR	OR
EXCLUSIVE OR	XOR
CLEAR CARRY	CLRC
SET CARRY	SETC
COMPLEMENT CARRY	COMC
ENABLE INTERRUPT	EI
DISABLE INTERRUPT	DI

Shift Instructions

- Shift are operations in which bits of a word are moved left or right.

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

6. PROGRAM CONTROL

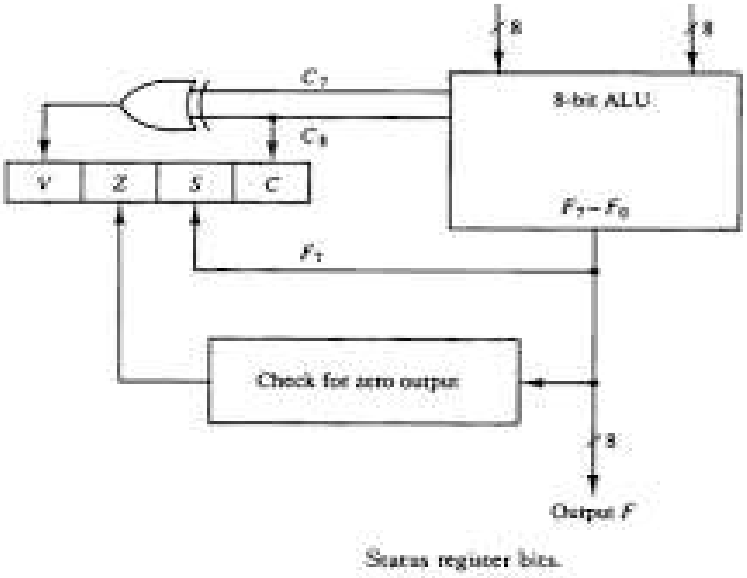
- Program control instructions specify conditions for altering the content of the program counter , while data transfer and manipulation instructions specify condtions for data-processing operations.

Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip next instruction	SKP
Call procedure	CALL
Return from procedure	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TEST

Status Bit Conditions

- It is convenient to supplement the ALU circuit in the CPU with a status register where status bit condition can be stored for further analysis.
- Status bits are also called condition code bit or flag bit.
- The four status bits are symbolized by C,S,Z and V.
- N bit is set if result of operation in negative (MSB = 1)
- Z bit is set if result of operation is zero (All bits = 0, AC=0)
- V bit is set if operation produced an overflow
- C bit is set if operation produced a carry (borrow on subtraction
- The bits are set or cleared as a result of an operation performed in the ALU.



CONDITIONAL BRANCH INSTRUCTIONS

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
	Signed compare conditions (A - B)	
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

22

Subroutine Call and Return

- It is a self-contained sequence of instructions that performs a given computational task.
- During the execution of a program, a subroutine may be called, when it is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions.
- After the subroutine has been executed,a branch is made back to the main program.
- A subroutine call is implemented with the following microoperations:

CALL:

SP← SP-1	// Decrement stack point
M[SP] ←PC	// Push content of PC onto the stack
PC←Effective Address	//Transfer control to the subroutine

RETURN:

PC ← M[SP]	// Pop stack and transfer to PC
SP ← SP+1	// Increment stack pointer

Program Interrupt

- Transfer program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed

Types of Interrupts

1) **External Interrupts**

Arises from I/O device, from a timing device, from a circuit monitoring the power supply, or from any other external source

2) **Internal Interrupts or TRAP**

Caused by register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation

3) **Software Interrupts**

Initiated by executing an instruction (INT or RST) » used by the programmer to initiate an interrupt procedure at any desired point in the program

Reduced Instruction Set Computer(RISC):

A computer with large number instructions is classified as a complex instruction set computer, abbreviated as CISC.

The computer which having the fewer instructions is classified as a reduced instruction set computer, abbreviated as RISC.

CISC Characteristics:

- A large number of instructions--typically from 100 to 250 instructions.
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes—typically from 5 to 20 differ modes.
- Variable-length instruction formats
- Instructions that manipulate operands in memory

RISC Characteristics:

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control
- A relatively large number of registers in the processor unit
- Efficient instruction pipeline

UNIT III

COMPUTER ARITHMETIC

Arithmetic Processor

- Arithmetic instruction in digital computers manipulate data to produce results necessary for the solution of the computational problems.
- An arithmetic processor is the part of a processor unit that execute arithmetic instruction.
- An arithmetic instruction may specify binary or decimal data, and it may be represented in, Fixed point (integer or fraction) OR floating point form.
- An arithmetic processor is simple for binary fixed point add instruction
- Data types considered for the arithmetic operations are,
    - Fixed-point binary data in signed magnitude representation
    - Fixed-point binary data in signed-2's complement representation
    - Floating point binary data
    - Binary –coded decimal (BCD) data
- Negative fixed point binary number can be represented in three ways,
    - Signed magnitude (most computers use for floating point operations)
    - Signed 1's complement
    - Signed 2's complement(most computer use for integers)

1. ADDITION AND SUBTRACTION WITH SIGNED-MAGNITUDE DATA

Eight different conditions to consider for addition and subtraction

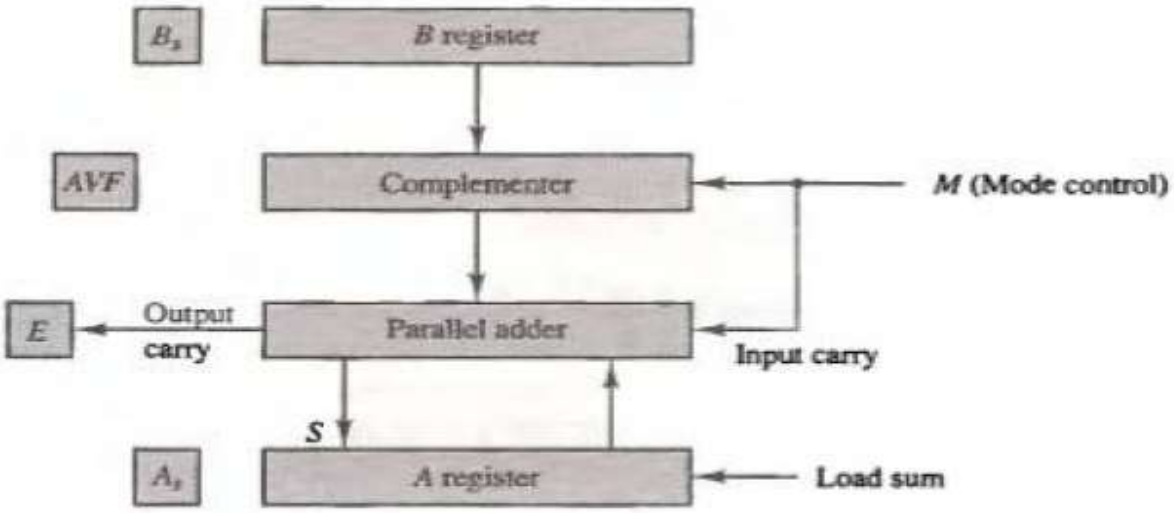
Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Addition(Subtraction) algorithm

- 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 sign 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 same as A if A > B or the complement of the sign of A if A < B.
- For equal magnitude subtract B from A and make the sign of the result

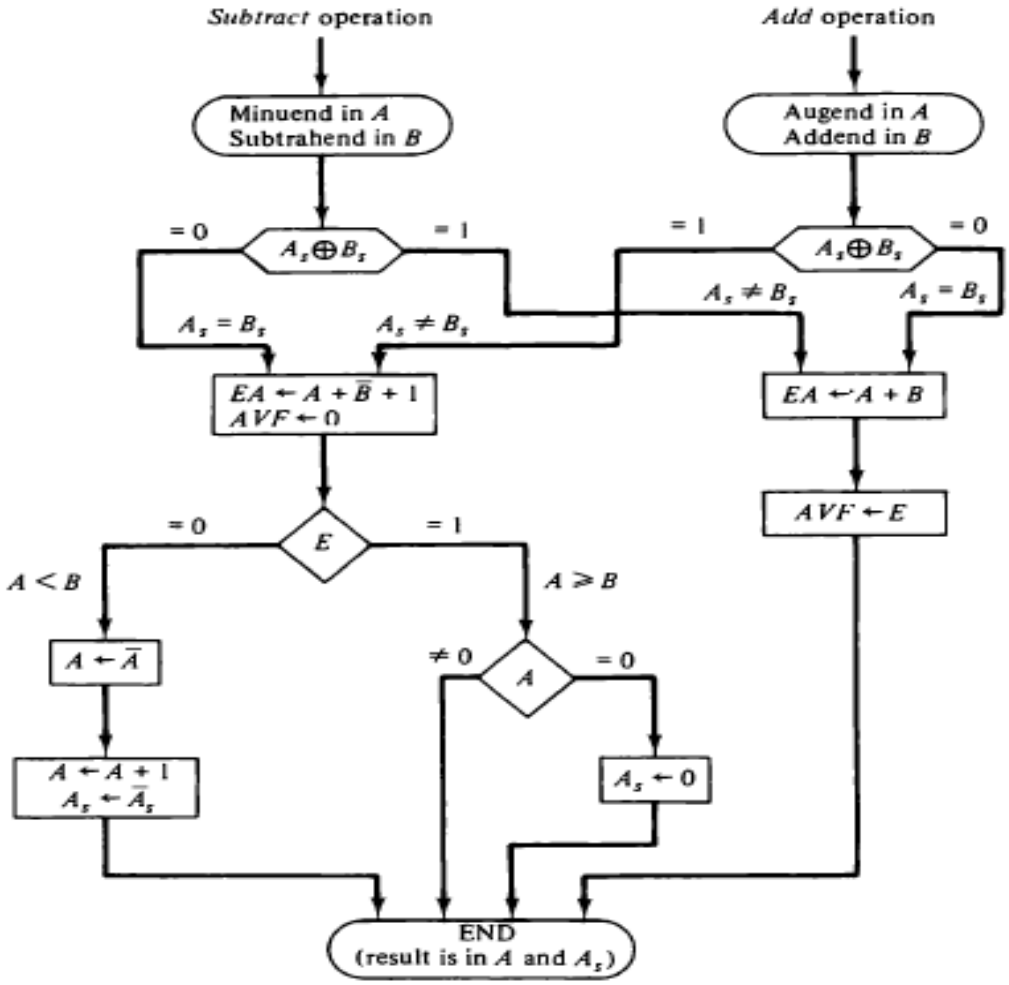
Hardware Implementation

- A and B be two registers that hold the magnitudes of No
- As and Bs be two flip-flops that hold the corresponding signs
- The Result is transferred into A and As.



- Parallel adder is needed to perform the micro operation A + B.
- Parallel subtractor are needed to perform A – B or B – A.
- Can be accomplished by means of complement and add
- Comparator circuit is needed to establish if A > B, A = B or A < B
- Comparison can be determine from the end carry after the subtraction
- The sign relationship can be determine from an exclusive-OR gate with As and Bs as inputs
- Output carry are transferred to E flip-flop
- Where it can be checked to determine the relative magnitude of the Nos.
- Add overflow flip-flop (AVF) holds the overflow bit when A and B are added.

Hardware Algorithm



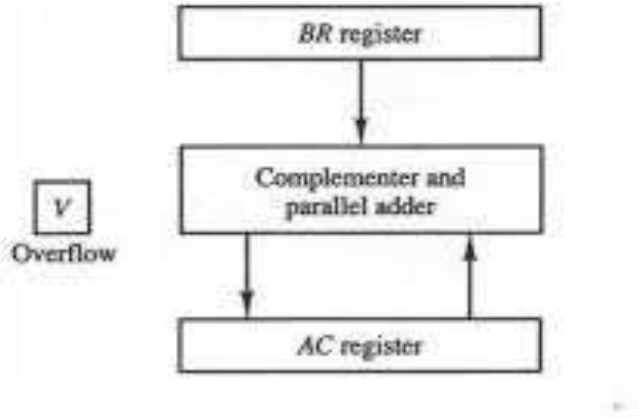
- The two signs A, and B, are compared by an exclusive-OR gate.
- If the output of the gate is 0 the signs are identical;
- If it is 1, the signs are different.
- For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs indicate that the magnitudes be added.
- The magnitudes are added with a microoperation EA← A + B, where EA is a register that combines E 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 complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.
- E=1 indicates that A >= B and the number in A is the correct result. If this number is zero, the sign A must be made positive to avoid a negative zero.
- E=0 indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation A ←A' +1.

- The sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A
- The final result is found in register A and its sign in As.

Addition and Subtraction with signed2's complement data

- The left most bit of binary number represents the sign bit; 0 for positive and 1 for negative.
- If the sign bit is 1, the entire the entire number is represented in 2's complement form.
- The addition of two numbers in signed-2's complement form consists of adding the number with the sign bits treated the same as the other bits of the number . A carry out of the sign bit position is discarded .
- The subtraction consists of first taking the 2's compliment of the subtrahend and then adding it to the minuend When two numbers of n digits each are added and the sum occupies n+1 Digits, we say that an overflow occurred.
- When the two carries are applied to an exclusive-OR gate, the overflow is detected when the output of the gate is equal to 1.

Hardware for signed-2's Complement addition and subtraction



- The left most bit in AC and BR represents the sign bits of the numbers
- The over flow flip-flops V is set to 1 if there is an overflow. The output carry in this case is discarded.

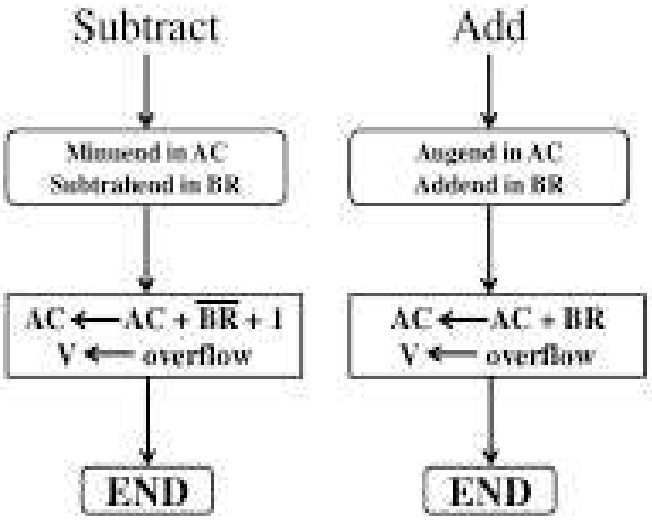


Figure: Algorithm for adding and subtracting numbers in signed-2's complement representation.

- The sum is obtained by adding the contents of AC and BR(including their sign bits).
- The overflow bit V is set to 1 if the ex-OR of the last two carries is 1,and it is cleared to 0 otherwise.

2. Multiplication algorithms

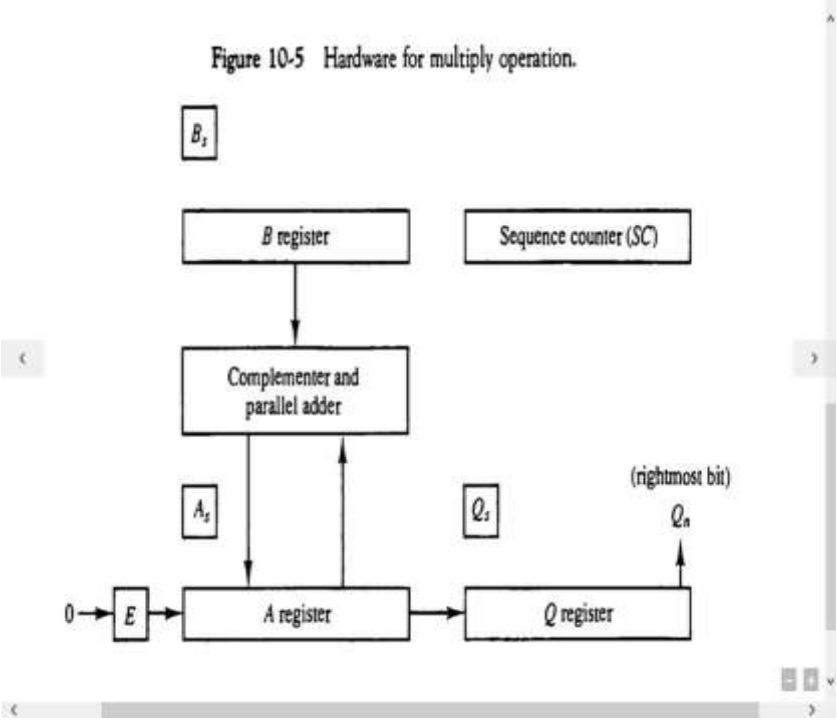
Multiplication of two fixed point binary numbers in signed magnitude representation is done with paper and pencil of successive shift and add operation if the multiplier bit is a 1,the multiplicand is copied down; otherwise zero are copied down.

```
 23      10111    Multiplicand
 19    × 10011    Multiplier
         10111
        10111
       00000   +
      00000
     10111
437  110110101    Product
```

Hardware Implementation for Signed-Magnitude data

- Multiplication is conveniently implemented with two changes in the process
  - Use an adder to add binary numbers as there are bits in multiplier and successively accumulate the partial products in a register.
  - Instead of shifting the multiplicand to the left , the partial product is shifted to the right
- The hardware for multiplication consists of registers Q, A and B.
- The multiplier stored in the Q register and its sign in Qs.

- The multiplicand is stored in B register and its sign in Bs.
- The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product
- The sum of A and B forms a partial product which is transferred to the EA register .
- The shift will be denoted by the statement shr EAQ to designate the
- right shift depicted .
- The least significant bit of A is shifted into the most significant position of Q.



Hardware Algorithm

- Initially the multiplicand is in B and the multiplier in Q their corresponding signs are in Bs and Qs respectively.
- Register A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.
- After the initialization , the low order bit of the multiplier in Qn is tested . If it is 1, the multiplicand In B is added to the present partial product in A . If it is 0, nothing is done .
- Register EAQ shifted once to the right to form the new partial product.
- The process stops when SC=0.
- Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces multiplier.
- The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.

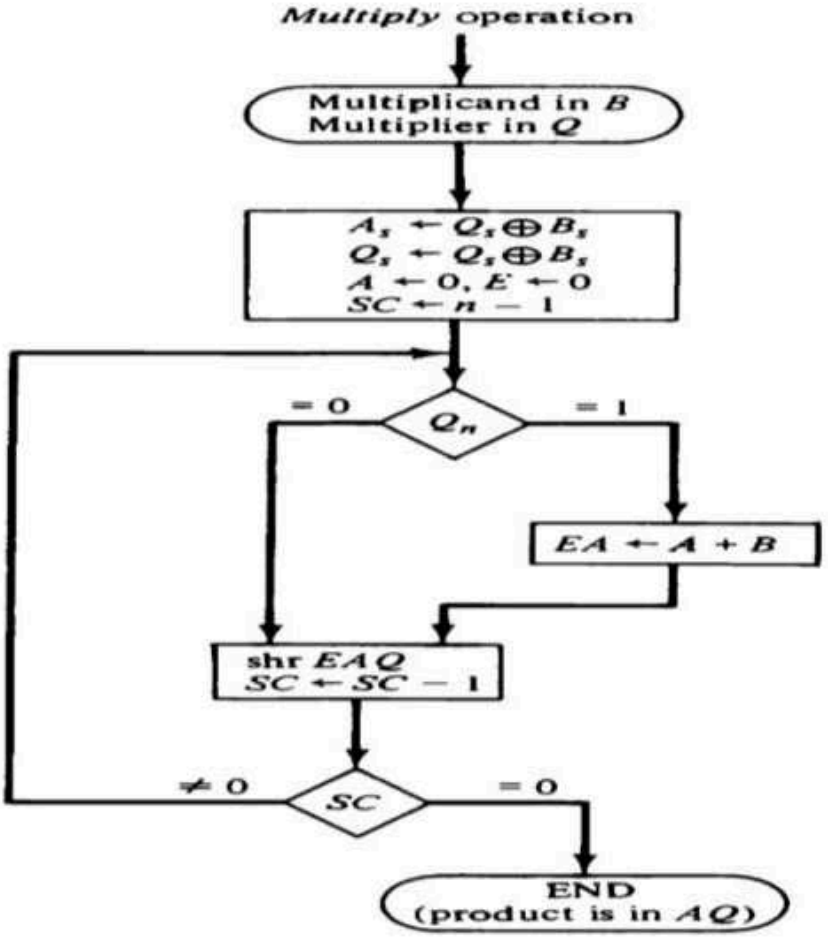


TABLE 10-2 Numerical Example for Binary Multiplier

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Q_n = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Q_n = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Q_n = 0; shift right EAQ	0	01000	10110	010
Q_n = 0; shift right EAQ	0	00100	01011	001
Q_n = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

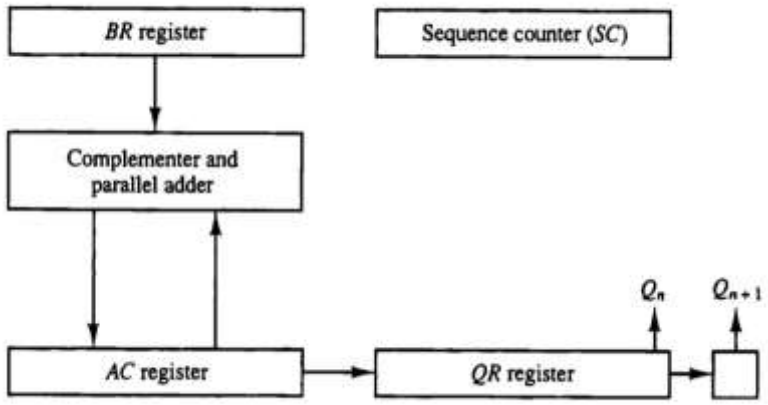
3. Booth's Multiplication Algorithm

- Booth algorithm gives a procedure for **multiplying binary integers** in **signed 2's complement representation in efficient way**, i.e., less number of additions/subtractions required.
- In this method the multiplier or multiplicand if it is negative number is represented in 2's complement representation.

Hardware Implementation of Booths Algorithm

- The hardware implementation of the booth algorithm requires the register configuration shown in the figure below.
- The register used are AC, BR and QR respectively.
- Q_n designates the least significant bit of multiplier in the register QR.
- An extra flip-flop $Q_{n+1}$is appended to QR to facilitate a double inspection of the multiplier

Figure 10-7 Hardware for Booth algorithm.



Algorithm

- Store the multiplicand in BR register and multiplier in QR register.
- Initialise AC and Q_{n+1} to zero. Initialise SC to the number of bits in the multiplier.
- The two bits of multiplier in Q_n and Q_{n+1} are inspected.
- If QnQn+1= 10
  - subtract BR from AC ie Add 2's complement of BR to AC .
  - perform arithmetic right shift on AC & BR using ashr instruction.
- If QnQn+1=00 or 11 perform arithmetic right shift on AC & BR.
- If QnQn+1= 01
  - Add BR to AC
  - perform arithmetic right shift on AC & BR.
- Decrement SC after each shift operation.
- Continue the steps until SC becomes zero.
- The product of the multiplication is present in AC and QR.
- Arithmetic right shift is shifts a bit to the right, it retains the sign(first) bit as such.

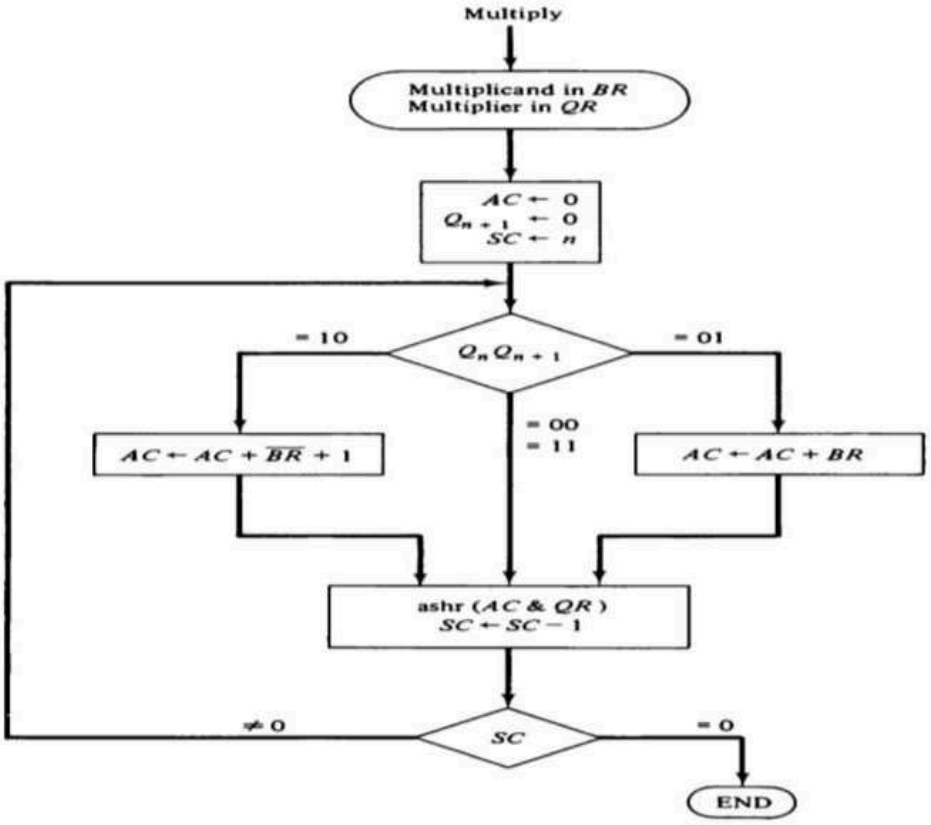


Figure 10-8 Booth algorithm for multiplication of signed-2's complement numbers.

Example

- Multiplying -9 and -13.
- Since both are negative numbers they are represented in 2's complement form.
- Binary of 9 = 01001
  1's complement of 9 = 10110
  2's complement of 9 = 10110 +1 = 10111 = -9
- Binary of 13 = 01101
  1's complement of 13 =10010
  2's complement of 13 = 10010+1 = 10011= -13
- The result is in AC & BR 0001110101 = 117

Q_nQ_{n+1}	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

4. DIVISION ALGORITHMS

Division of two fixed-point binary numbers in signed magnitude representation is done with paper and pencil by a process of successive compare, shift, and subtract operations .

Binary Division

Figure 10-11 Example of binary division.

Divisor:		
	11010	Quotient = Q
B = 10001	)0111000000	Dividend = A
	01110	5 bits of $A < B$, quotient has 5 bits
	011100	6 bits of $A \geq B$
	-10001	Shift right B and subtract; enter 1 in Q
	-010110	7 bits of remainder $\geq B$
	--10001	Shift right B and subtract; enter 1 in Q
	--001010	Remainder $< B$; enter 0 in Q; shift right B
	---010100	Remainder $\geq B$
	----10001	Shift right B and subtract; enter 1 in Q
	----000110	Remainder $< B$; enter 0 in Q
	-----00110	Final remainder

HARDWARE IMPLEMENTATION OF SIGNED MAGNITUDE DATA

- Two modifications is done to the division process to facilitate hardware implementation.
  - Instead of shifting the divisor to the right, the dividend or partial remainder, is shifted to the left.
  - Subtraction is done by adding A to the 2's complement of B.
- The divisor is stored in the B register and the double length dividend is stored in register A and Q.
- The information about relative magnitude is available in E.
- If E=1,it signifies that A ≥B. A quotient bit 1 is inserted into Qn and the partial remainder is shifted left to repeat the process. 2's complement of B is added to partial product. (Qn=1, shl EAQ, Add 2's complement of B)
- If E=0, it signifies that A<B so the quotient in Qn remains a 0. The partial product is restored to its previous value by adding B to it. (Qn=0, Add B, shl EAQ, Add 2's complement of B)
- The process is repeated until all five quotient bits are formed.
- The sign of the remainder is the same as the sign of the dividend .

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B}$ + 1		01111		
E = 1	1	01011		
Set Q_n = 1	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B}$ + 1		01111		
E = 1	1	00101		
Set Q_n = 1	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B}$ + 1		01111		
E = 0; leave Q_n = 0	0	11001	00110	
Add B		10001		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\overline{B}$ + 1		01111		
E = 1	1	00011		
Set Q_n = 1	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B}$ + 1		01111		
E = 0; leave Q_n = 0	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Figure 10-12 Example of binary division with digital hardware.

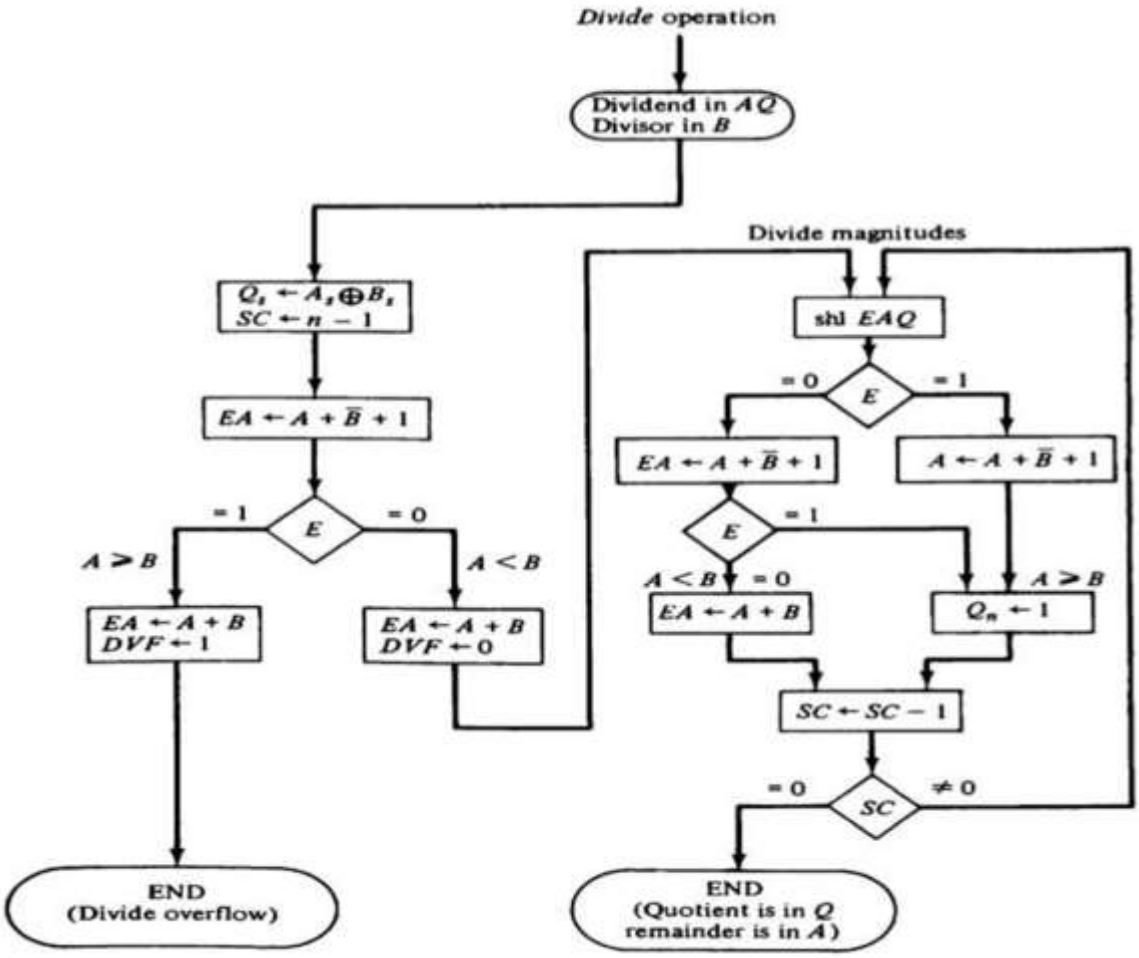
Divide overflow

- A divide overflow condition occurs if the higher order half bits of the dividend constitute a number greater than or equal to the divisor.
- It may result in a quotient with an overflow.The length of the registers is finite and could not hold number of bits greater than its length.
- Overflow condition is usually detected when a special flip-flop is set. Which will call it a divide overflow flip-flop and label it DVF.
- In some computers it is the responsibility of the programmers to check if DVF is set after each divide instruction.
- In older computers the occurrence of a divide overflow stopped the computer and this condition was referred to as a DIVIDE STOP.
- The best way to avoid a divide overflow is to use floating point data. The divide overflow can be handled very simply if numbers are in floating point representation.

Hardware algorithm

- The dividend is in A and Q and the divisor in B. The sign of the results transferred into Qs to be part of quotient.
- A divide overflow condition is tested by subtracting divisor in B from half of the bits of the dividend stored in A. if A≥B, the divide overflow flip-flop DVF set and the operation is terminated prematurely.
- By doing the process as shown in the flowchart the quotient magnitude is formed in register Q and the remainder is found in the register A. The quotient sign is in Qs and the sign of the remainder in As is the same as the original sign of the dividend.

Figure 10-13 Flowchart for divide operation.



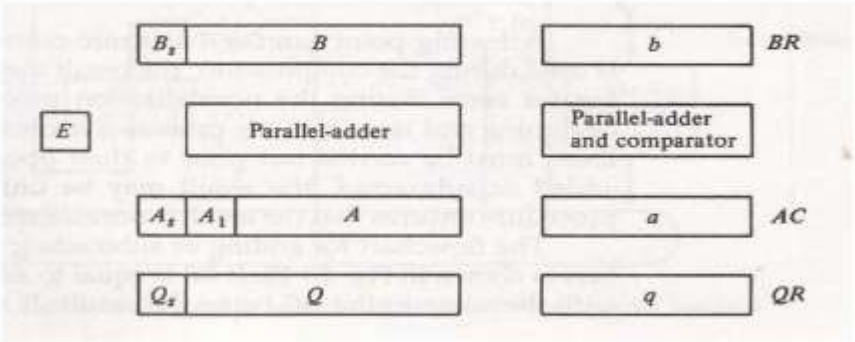
FLOATING POINT ARITHMETIC OPERATIONS

Register Configuration

- Three registers are there, BR, AC, and QR.
- Each register is subdivided into two parts – mantissa part (uppercase symbol) and exponent part (lowercase symbol.
- The AC has a mantissa whose sign is in A_s, and a magnitude that is in A. The most significant bit of A is A1. This bit must be a 1 to normalize the number. 'a' is the exponent part of AC.
- AC is a combination of A_s, A and a.
- Register BR is subdivided into Bs, B, and b and QR into Qs, Q and q.
- A parallel-adder adds the two mantissas and loads the sum into A and the carry into E. A separate parallel adder can be used for the exponents.
- The exponents do not have a distinct sign bit because they are biased but are represented as a biased positive quantity

- The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.
- The numbers in the registers should initially be normalized. After each arithmetic operation, the result will be normalized. Thus all floating-point operands are always normalized.

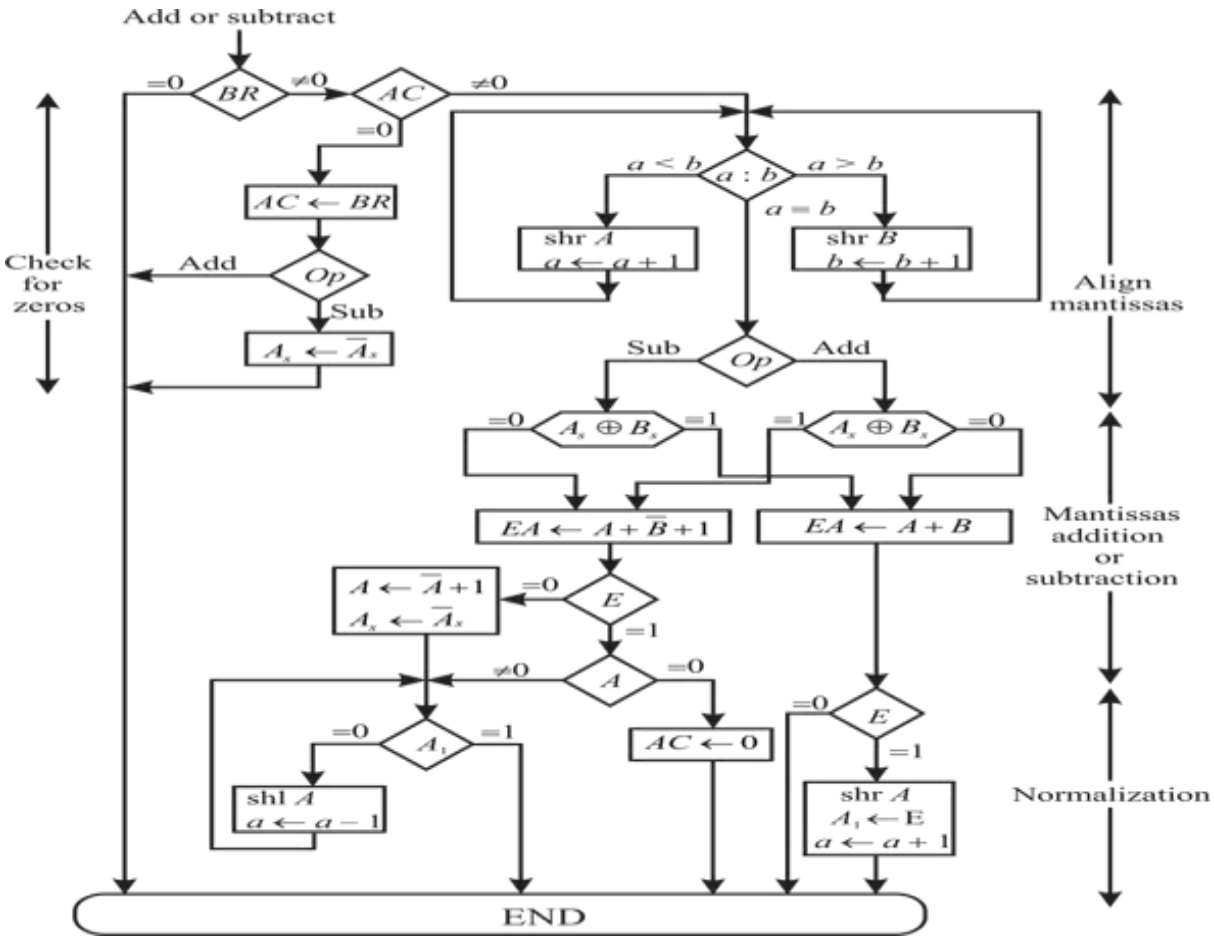
Registers for floating point arithmetic operations



5.ADDITION AND SUBTRACTION OF FLOATING POINT NUMBERS

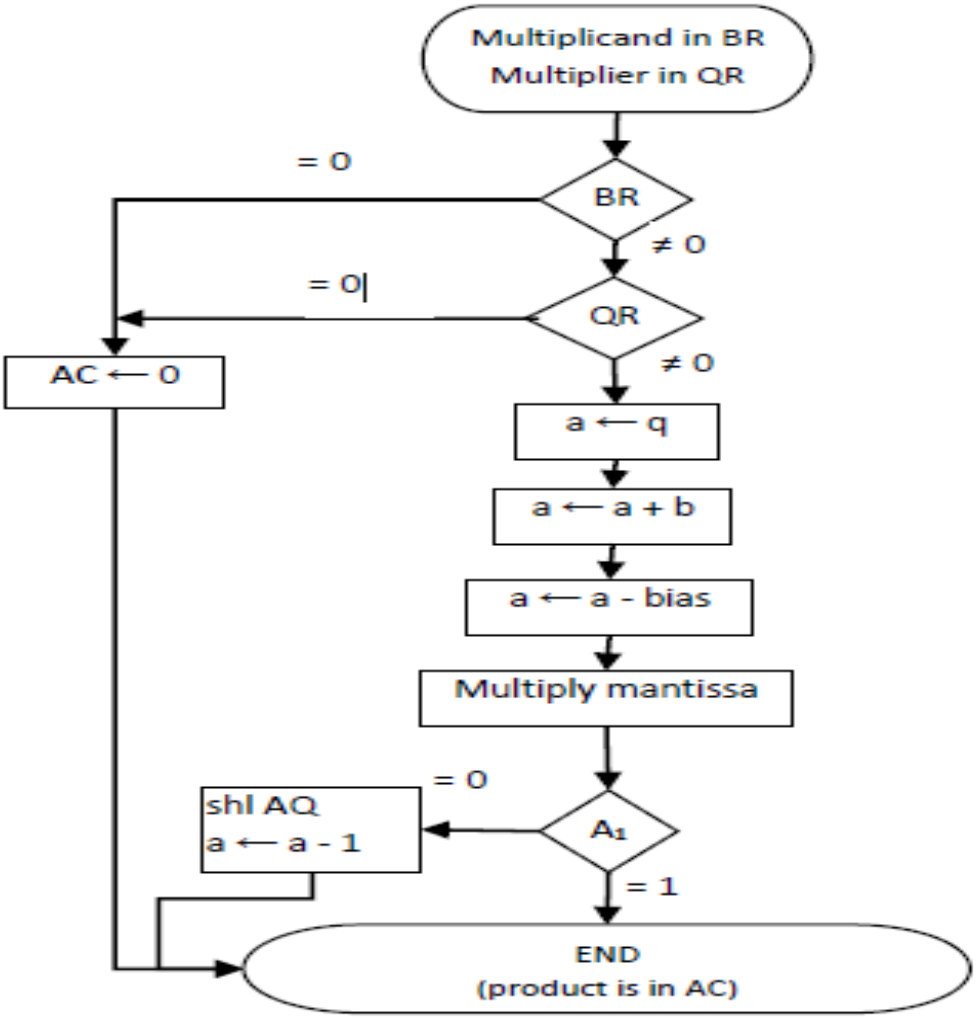
- During addition or subtraction, the two floating-point operands are kept in AC and BR. The sum or difference is formed in the AC.
- The algorithm can be divided into four consecutive parts:
  - Check for zeros.
  - Align the mantissas.
  - Add or subtract the mantissas
  - Normalize the result
- A floating-point number cannot be normalized, if it is 0. If this number is used for computation, the result may also be zero. Instead of checking for zeros during the normalization process we check for zeros at the beginning and terminate the process if necessary.
- The alignment of the mantissas must be carried out prior to their operation.
- After the mantissas are added or subtracted, the result may be un-normalized. The normalization procedure ensures that the result is normalized before it is transferred to memory. If the magnitudes were subtracted, there may be zero or may have an underflow in the result.
- If the mantissa is equal to zero the entire floating-point number in the AC is cleared to zero. Otherwise, the mantissa must have at least one bit that is equal to 1.
- The mantissa has an underflow if the most significant bit in position A1, is 0. In that case, the mantissa is shifted left and the exponent decremented.

- The bit in A1 is checked again and the process is repeated until A1 = 1. When A1 = 1, the mantissa is normalized and the operation is completed



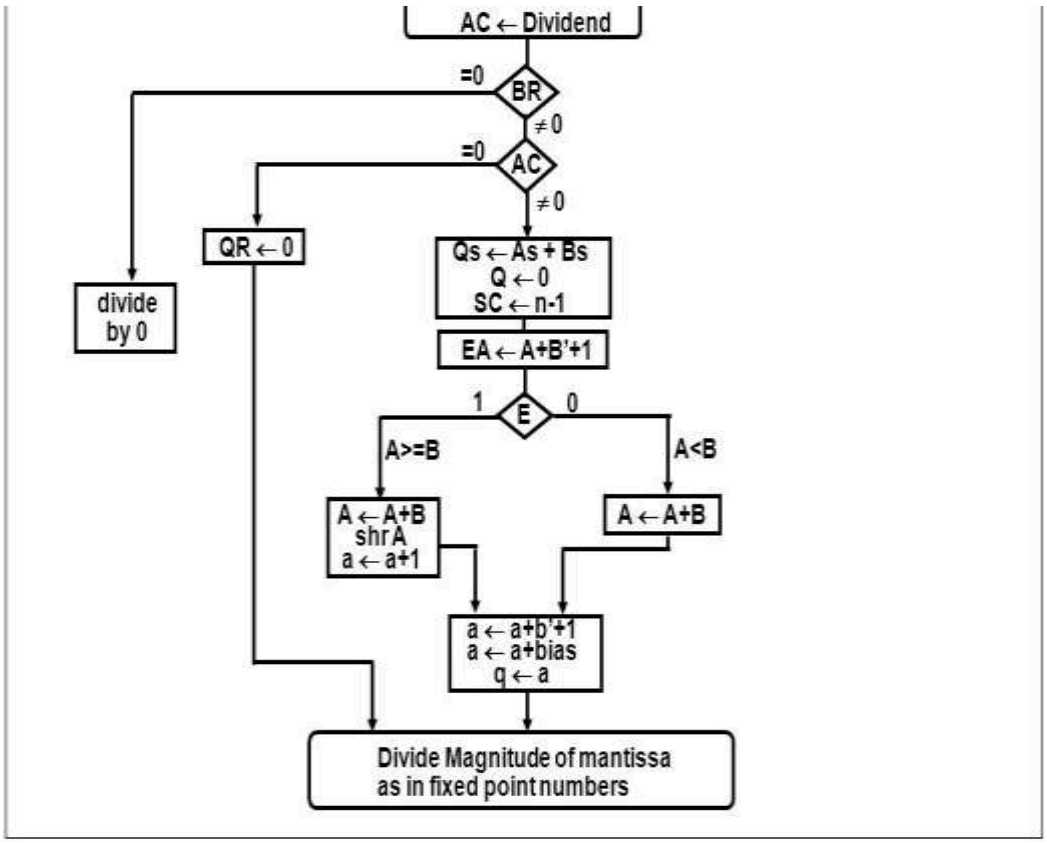
6. FLOATING POINT MULTIPLICATION

- Multiply the mantissas and add the exponents.
- No comparison of exponents or alignment of mantissas is necessary.
- Four components:
    1. Check for zeros.
    2. Add the exponents.
    3. Multiply the mantissas.
    4. Normalize the product.



7.FLOATING POINT DIVISION

- Floating-point division requires that the exponents be subtracted and the mantissas divided as in fixed point division.
- Steps
    - Check for zeros.
    - Initialize registers and evaluate the sign
    - Align the dividend(check overflow).
    - Subtract the exponents.
    - Divide the mantissas.


AC ← Dividend
=0
BR
≠ 0
=0
AC
≠ 0
QR ← 0
divide by 0
Qs ← As + Bs
Q ← 0
SC ← n-1
EA ← A+B'+1
1
E
0
A>=B
A<B
A ← A+B shr A
a ← a+1
A ← A+B
a ← a+b'+1
a ← a+bias
q ← a
Divide Magnitude of mantissa as in fixed point numbers

UNIT-IV

INPUT-OUTPUT ORGANIZATION

Peripheral Devices:

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment

The most common input output devices are: Monitor, Keyboard, Mouse, Printer, Magnetic tapes etc.The devices that are under the direct control of the computer are said to be connected online.

1. INPUT - OUTPUT INTERFACE

- Input Output Interface provides a method for transferring information between internal storage and external I/O devices.
- Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.
- The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.
- The Major Differences are:-
- Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
- The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.
- Data codes and formats in the peripherals differ from the word format in the CPU and memory.
- The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.
- To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervises and synchronizes all input and out transfers. These components are called **Interface Units** because they interface between the processor bus and the peripheral devices.

I/O BUS and Interface Module

- It defines the typical link between the processor and several peripherals.
- The I/O Bus consists of data lines, address lines and control lines. The I/O bus from the processor is attached to all peripherals interface.
- To communicate with a particular device, the processor places a device address on address lines.
- Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller.
- It is also synchronizes the data flow and supervises the transfer between peripheral and processor.
- Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing The control lines are referred as I/O command. The commands are as following:

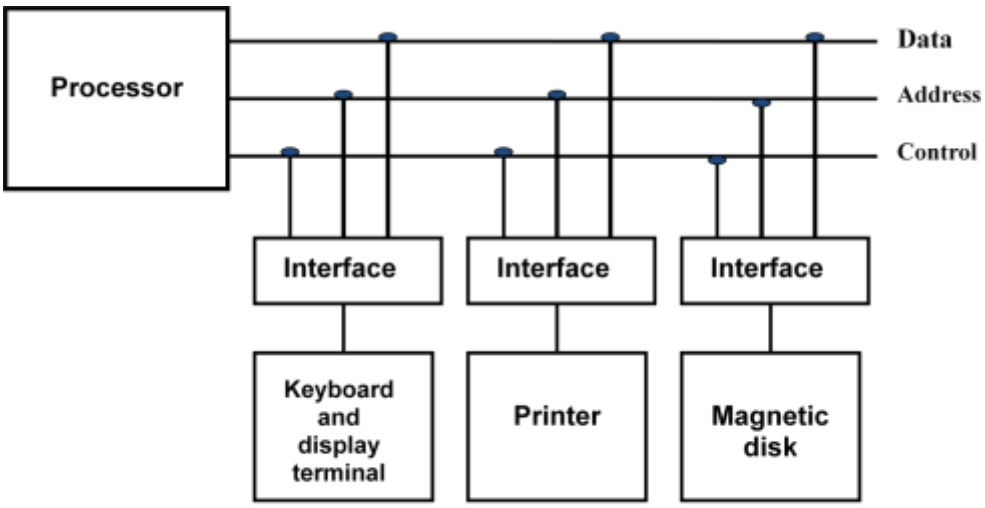
Control command- A control command is issued to activate the peripheral and to inform it what to do.

Status command- A status command is used to test various status conditions in the interface and the peripheral.

Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register. I/O Versus Memory Bus



Connection of I/O bus to 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. I/O Processor
- In the first method, the computer has independent sets of data, address and control buses one for accessing memory and other for I/O. This is done in computers that provides a separate I/O processor (IOP). The purpose of IOP is to provide an independent pathway for the transfer of information between external device and internal memory.

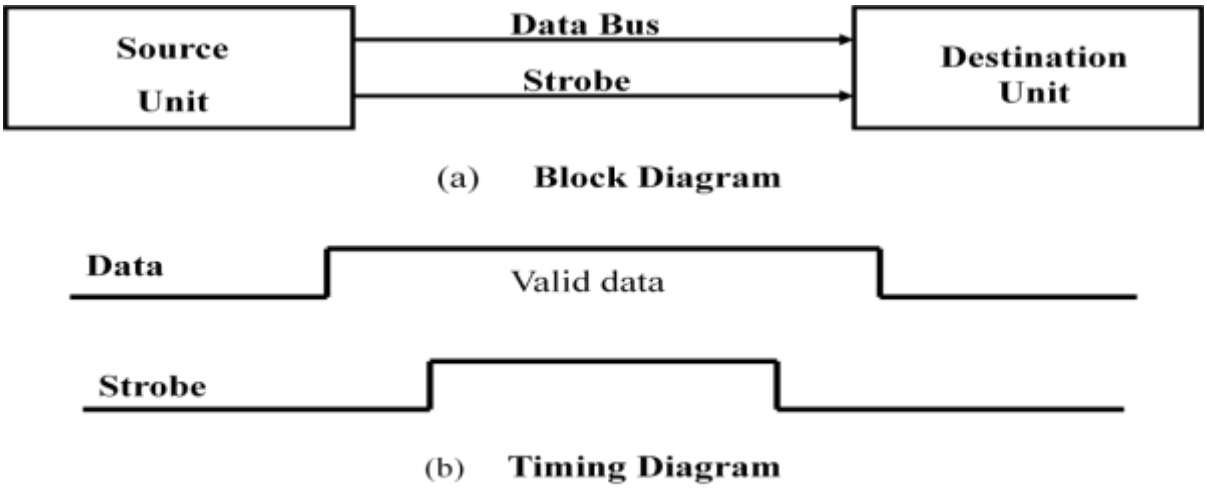
2. ASYNCHRONOUS DATA TRANSFER :

- This Scheme is used when speed of I/O devices do not match with microprocessor, and timing characteristics of I/O devices is not predictable.
- In this method, process initiates the device and check its status. As a result, CPU has to wait till I/O device is ready to transfer data.
- When device is ready CPU issues instruction for I/O transfer. In this method two types of techniques are used based on signals before data transfer.
  i. Strobe Control
  ii. Handshaking

STROBE CONTROL :

The strobe control method of Asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit.

Source-Initiated Data Transfer:

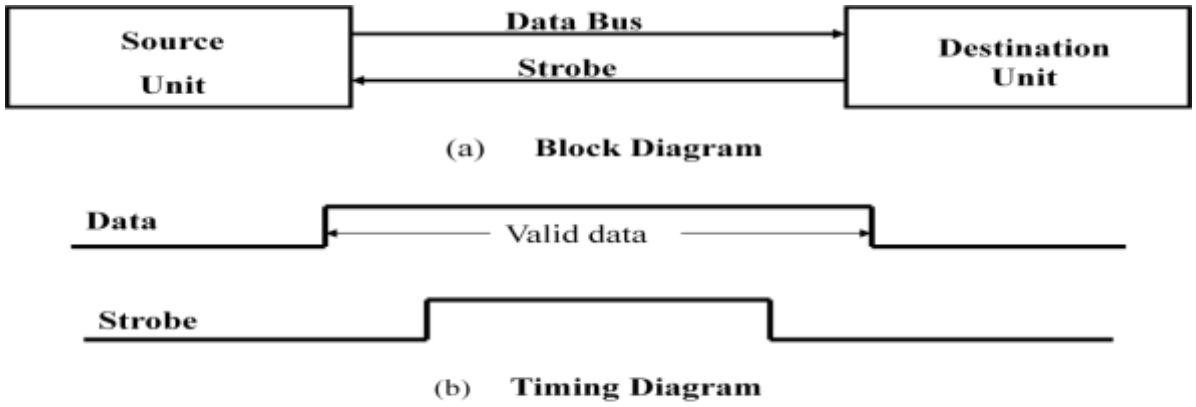


Source-Initiated strobe for Data Transfer

- In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available.
- The timing diagram fig. (b) the source unit first places the data on the data bus. The information on the data bus and strobe signal remain in the active state to allow the destination unit to receive the data.

Destination-Initiated Data Transfer:

- In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond by placing the requested binary information on the data bus.
- The data must be valid and remain in the bus long enough for the destination unit to accept it. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.



Destination-Initiated strobe for Data Transfer

Disadvantage of Strobe Signal :

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

HANDSHAKING:

The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

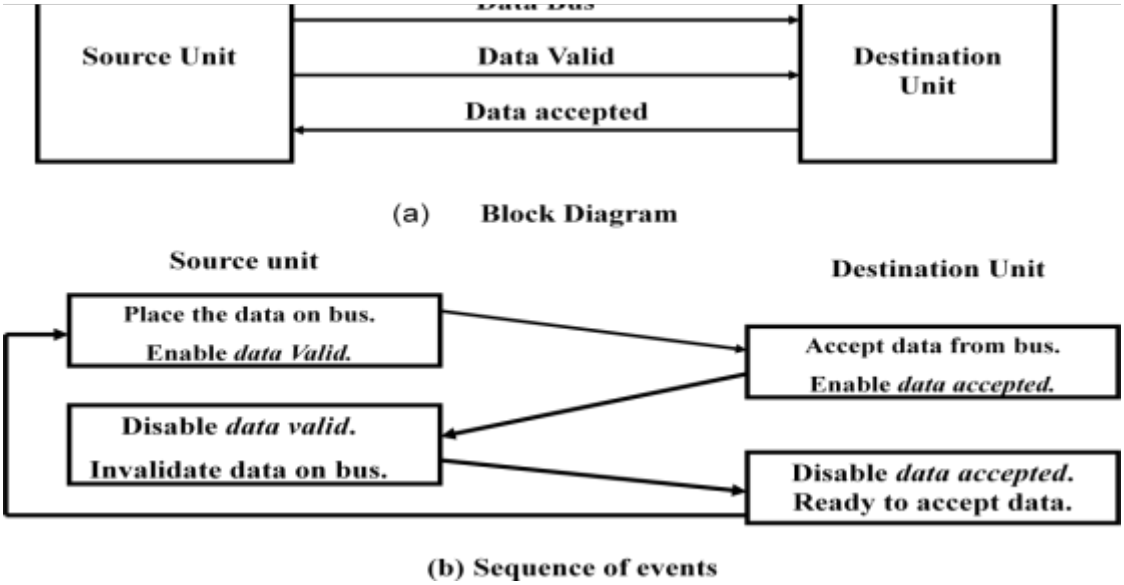
Principle of Handshaking:

The basic principle of the two-wire handshaking method of data transfer is as follow:

One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

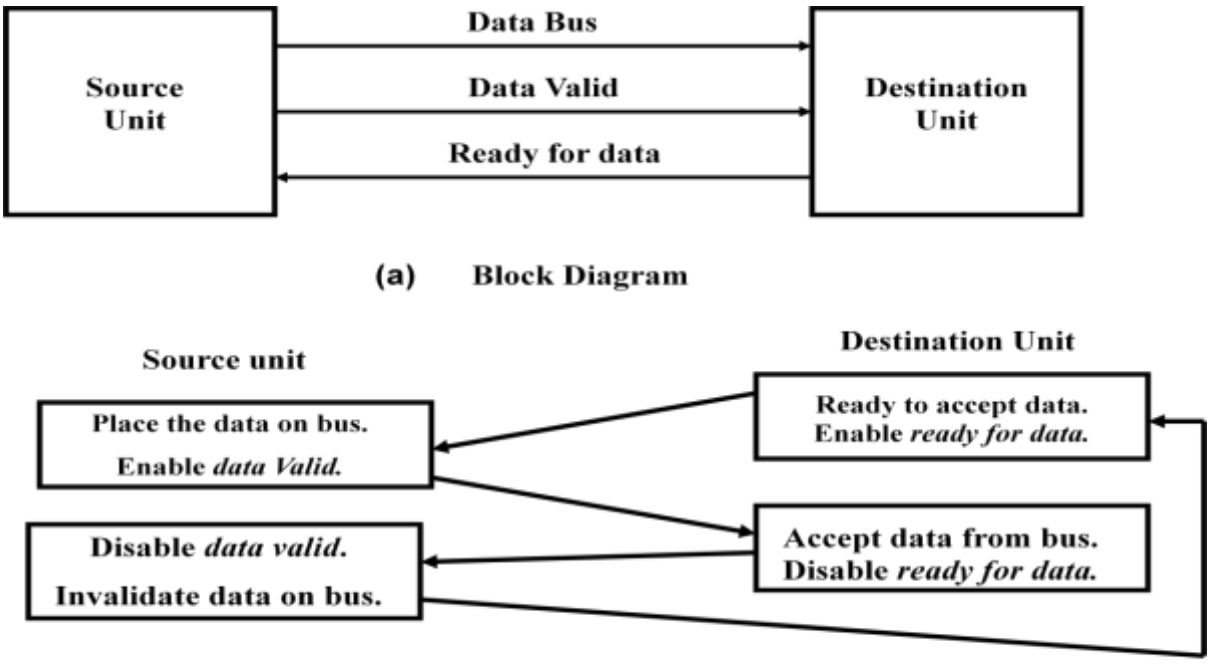
The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its *data valid* signal. The *data accepted* signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its *data accepted* signal and the system goes into its initial state.



Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to *ready for data* to reflects its new meaning. The source unit in this case does not place data on the bus until after it receives the *ready for data* signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case.

The only difference between the Source Initiated and the Destination Initiated transfer is in their choice of Initial sate.



Destination-Initiated transfer using Handshaking

Advantage of the Handshaking method:

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a *Timeout mechanism* which provides an alarm if the data is not completed within time.

ASYNCHRONOUS SERIAL TRANSMISSION:

- The transfer of data between two units is serial or parallel.
- In parallel data transmission, n bit in the message must be transmitted through n separate conductor path.
- In serial transmission, each bit in the message is sent in sequence one at a time.
- Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.
- In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.
- In this technique each character consists of three points :

  i. Start bit

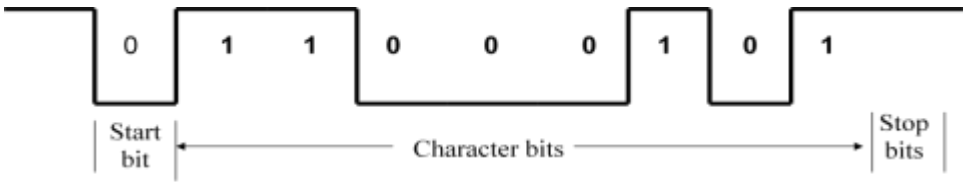
  ii. Character bit

  iii. Stop bit

i. Start Bit- First bit, called start bit is always zero and used to indicate the beginning character.

ii. Stop Bit- Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.

iii. Character Bit- Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.



Asynchronous Serial Transmission

Serial Transmission of Asynchronous is done by two ways:

a) Asynchronous Communication Interface

b) First In First out Buffer

ASYNCHRONOUS COMMUNICATION INTERFACE:

- It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register.
- The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission.
- The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register.
- CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

First In First Out Buffer (FIFO):

- A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates.
- When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate.
- If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously.

3. MODES OF DATA TRANSFER :

Transfer of data is required between CPU and peripherals or memory or sometimes between any two devices or units of your computer system. To transfer a data from one unit to another one should be sure that both units have proper connection and at the time of data transfer the receiving unit is not busy. This data transfer with the computer is Internal Operation.

All the internal operations in a digital system are synchronized by means of clock pulses supplied by a common clock pulse Generator. The data transfer can be

i. Synchronous or
ii. Asynchronous

- When both the transmitting and receiving units use same clock pulse then such a data transfer is called Synchronous process.

- On the other hand, if the there is no concept of clock pulses and the sender operates at different moment than the receiver then such a data transfer is called Asynchronous data transfer.

- The data transfer can be handled by various modes. some of the modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit and this can be handled by 3 following ways:

    i. Programmed I/O
    ii. Interrupt-Initiated I/O
    iii. Direct Memory Access (DMA)

Programmed I/O Mode:

- In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa.

- Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.

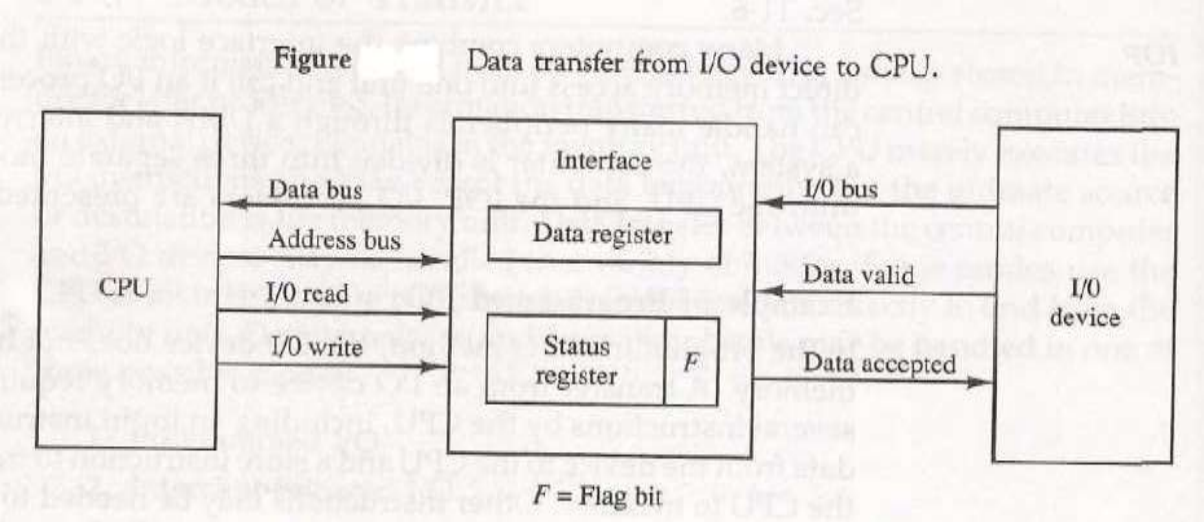
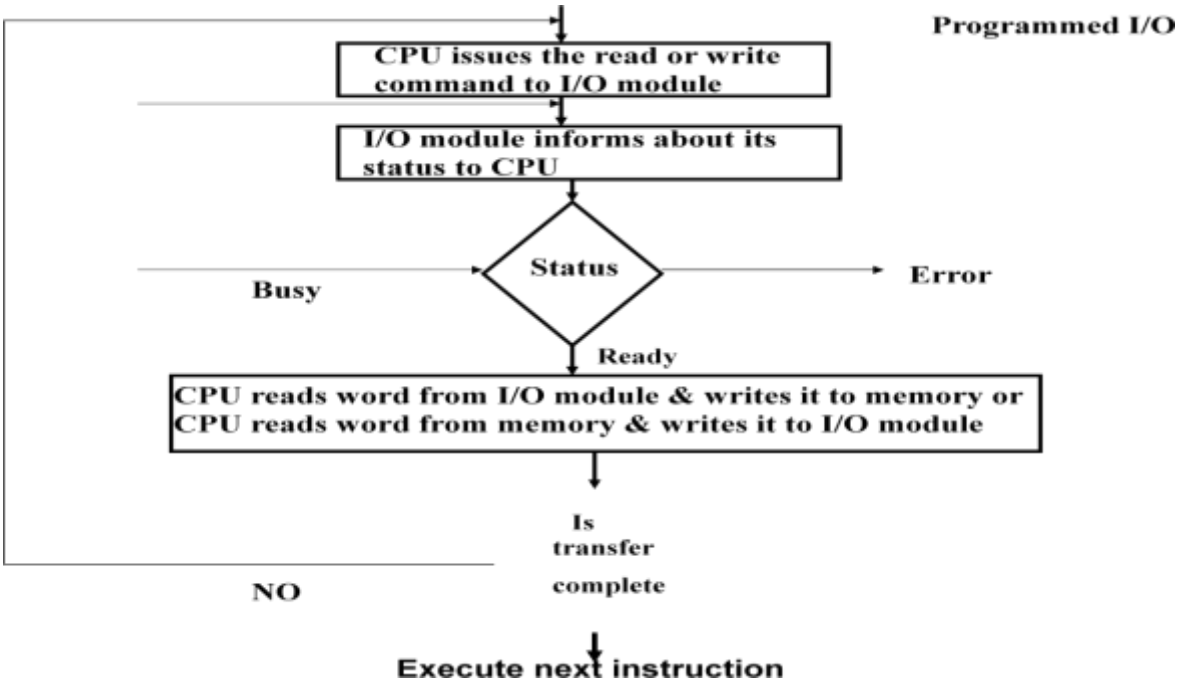


Figure Data transfer from I/O device to CPU.

- The transfer of data requires three instructions:

1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Flowchart-:



Drawback of the Programmed I/O :

- The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process and the CPU time is wasted a lot in keeping an eye to the executing of program.
- To remove this problem an Interrupt facility and special commands are used.

Interrupt-Initiated I/O :

- In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer.
- When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.
- In this type of IO, computer does not check the flag. It continue to perform its task.
- Whenever any device wants the attention, it sends the interrupt signal to the CPU.
- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.
- There are two ways of choosing the branch address:
    - Vectored Interrupt
    - Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

4. PRIORITY INTERRUPT:

- There are number of IO devices attached to the computer.
- They are all capable of generating the interrupt.
- When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.
- Devices with high speed transfer are given higher priority and slow devices are given lower priority.
- Establishing the priority can be done in two ways:
    - Using Software
    - Using Hardware
- A pooling procedure is used to identify highest priority in software means.

Polling Procedure :

- There is one common branch address for all interrupts.
- Branch address contain the code that polls the interrupt sources in sequence. The highest priority is tested first.
- The particular service routine of the highest priority device is served.
- The disadvantage is that time required to poll them can exceed the time to serve them in large number of IO devices.

Using Hardware:

- Hardware priority system function as an overall manager.
- It accepts interrupt request and determine the priorities.
- To speed up the operation each interrupting devices has its own interrupt vector.
- No polling is required, all decision are established by hardware priority interrupt unit.
- It can be established by serial or parallel connection of interrupt lines.

Serial or Daisy Chaining Priority:

- Device with highest priority is placed first.
- Device that wants the attention send the interrupt request to the CPU.
- CPU then sends the INTACK signal which is applied to PI(priority in) of the first device.
- If it had requested the attention, it place its VAD(vector address) on the bus. And it block the signal by placing 0 in PO(priority out)
- If not it pass the signal to next device through PO(priority out) by placing 1.
- This process is continued until appropriate device is found.
- The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

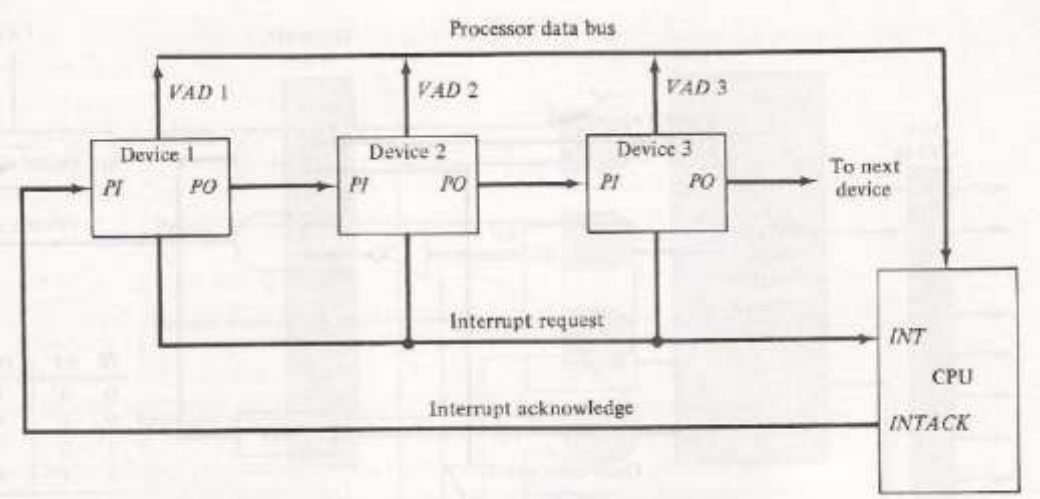


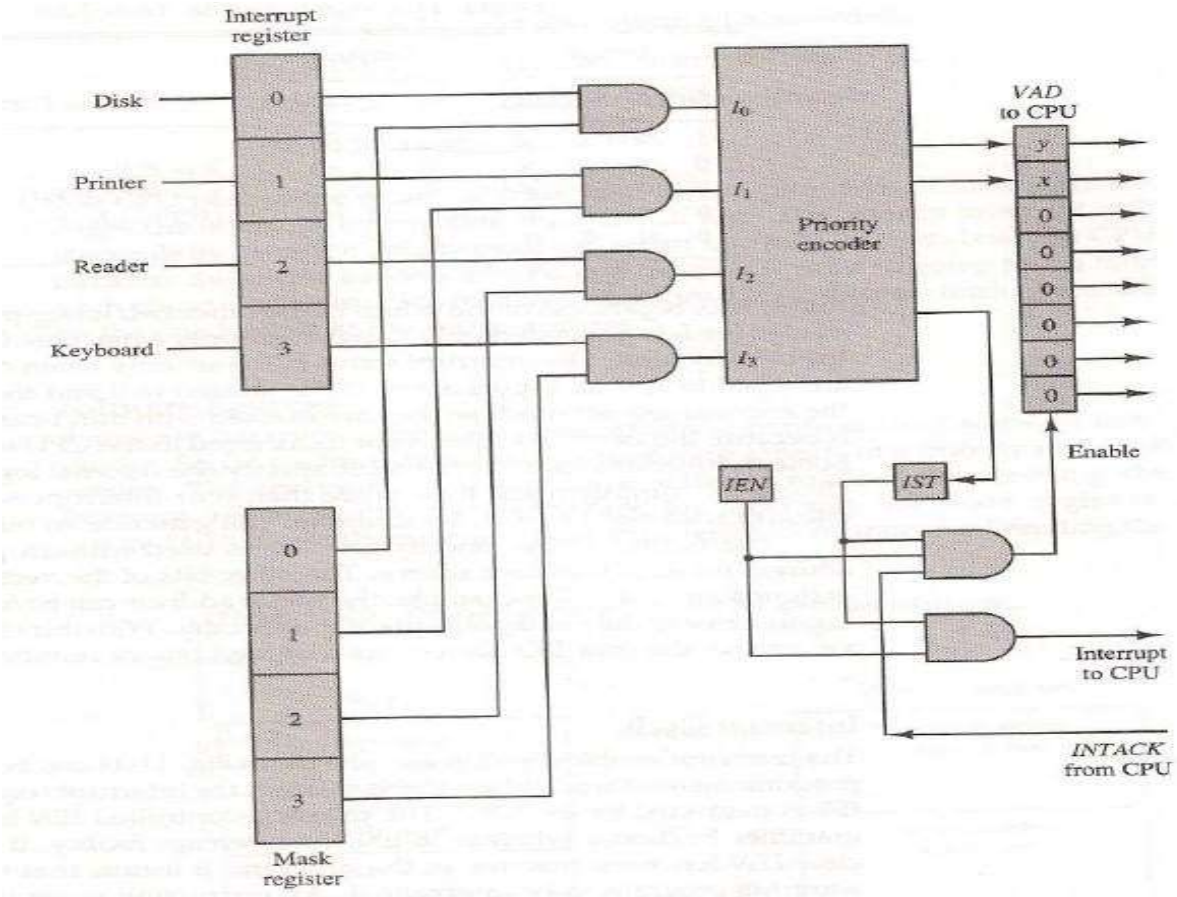
Figure Daisy-chain priority interrupt.

Parallel Priority Interrupt :

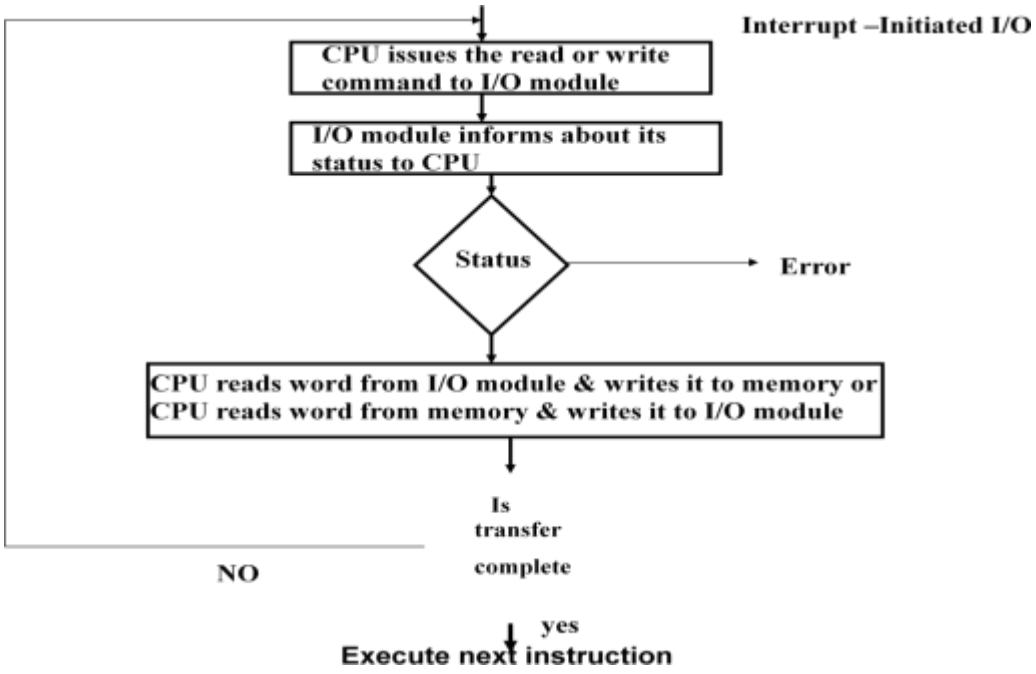
- It consist of interrupt register whose bits are set separately by the interrupting devices.
- Priority is established according to the position of the bits in the register.
- Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being serviced.
- Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.
- Priority encoder generates two bits of vector address.
- Another output from it sets IST(interrupt status flip flop).

Priority Encoder Truth Table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I_0' I_1'$
0	0	1	×	1	0	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	



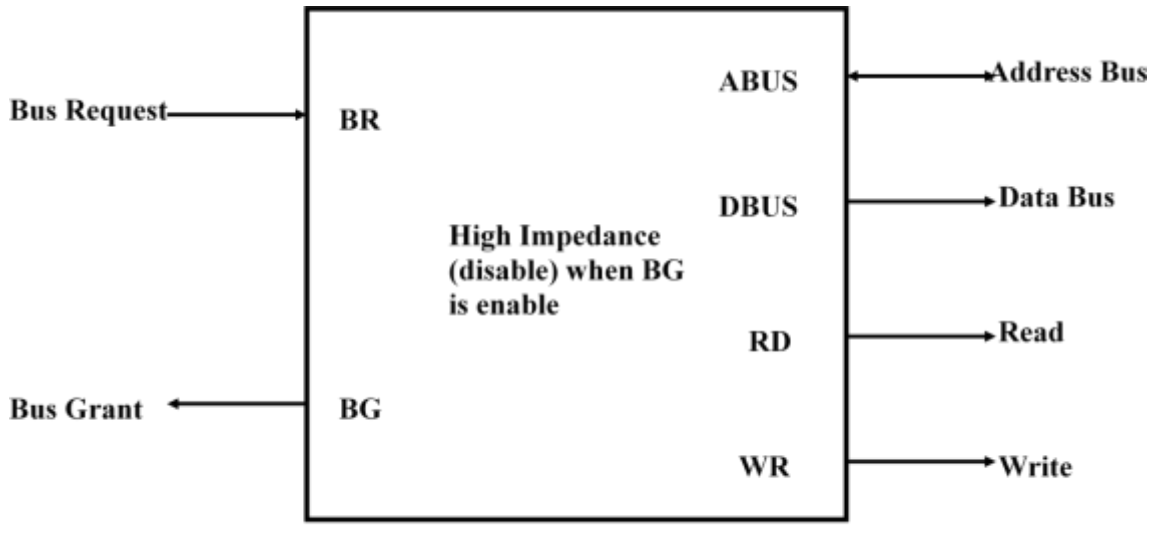
The Execution process of Interrupt–Initiated I/O is represented in the flowchart:



5.Direct Memory Access (DMA):

- In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).
- During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.
- The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:
    - Bus Request (BR)
    - Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The *Bus Request (BR)* input is used by the *DMA controller* to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a *high Impedance state.* High Impedance state means that the output is disconnected.



CPU bus Signals for DMA Transfer

- The CPU activates the *Bus Grant (BG)* output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor.

- When the DMA terminates the transfer, it disables the *Bus Request (BR)* line. The CPU disables the *Bus Grant (BG)*, takes control of the buses and return to its normal operation.

- The transfer can be made in several ways that are:
  - DMA Burst
  - Cycle Stealing

- DMA Burst :- In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

- Cycle Stealing :- Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

DMA Controller:

- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

  i. Address Register
  ii. Word Count Register
  iii. Control Register

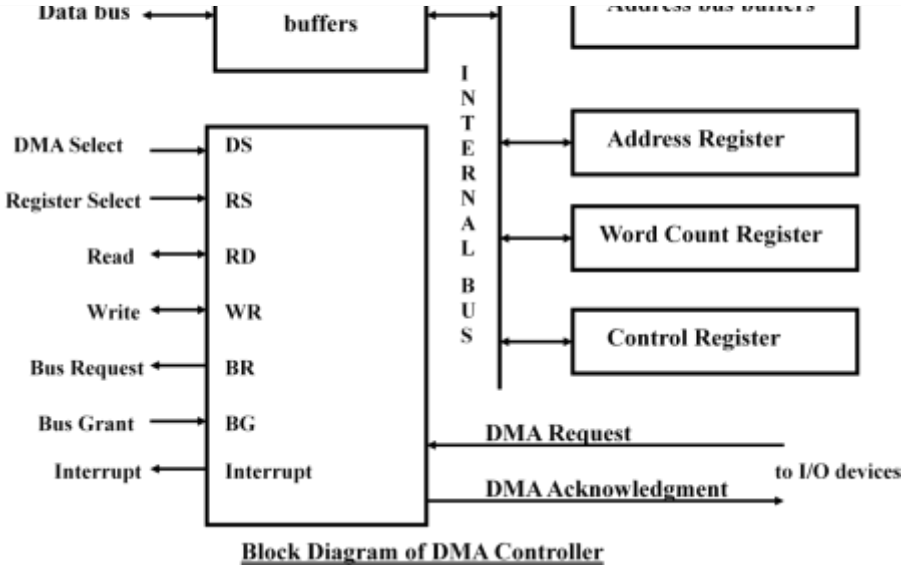
  i. Address Register :- Address Register contains an address to specify the desired location in memory.

  ii. Word Count Register :- WC holds the number of words to be transferred. The register is incre/decre by one after each word transfer and internally tested for zero.

  iii. Control Register :- Control Register specifies the mode of transfer

- The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional.

- When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.



Block Diagram of DMA Controller

DMA Transfer:

- The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory.
- When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.

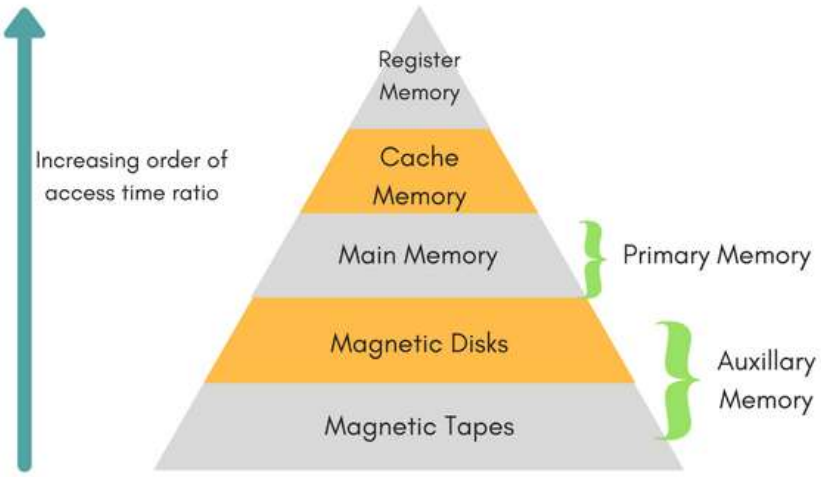
UNIT – 5

Memory Organization

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

- **Volatile Memory**: This loses its data, when power is switched off.
- **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.

Memory Hierarchy



The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.

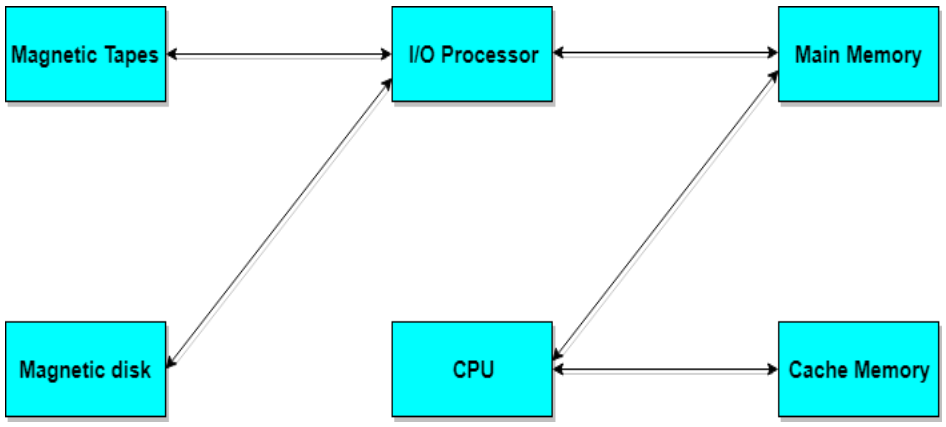
Auxillary memory are devices that provide backup storage. Its access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.

The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).

When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.

The **cache memory** is a high speed memory used to store frequently accessed program data which is currently being executed in the CPU. It is used to increase the speed of processing by making data available for the

CPU at a rapid rate. Approximate access time ratio between cache memory and main memory is about **1 to 7~10**



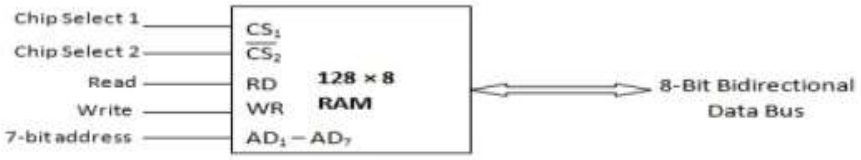
Main Memory

The memory unit that communicates directly within the CPU, auxillary memory and cache memory is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**.

- **RAM: Random Access Memory**
  - **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
  - **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
  - **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.
- **ROM**: Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on. **PROM**(Programmable ROM), **EPROM**(Erasable PROM) and **EEPROM**(Electrically Erasable PROM) are some commonly used ROMs.

RAM Chip

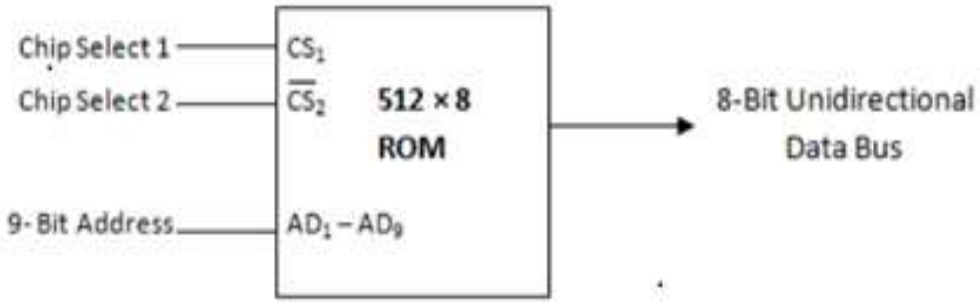
- A RAM chip has one or more control inputs(CS1, CS2) on select the chip only when needed.
- A bidirectional data bus allows data transfer from memory to cpu during read operation and cpu to memory during write operation. It has a three state buffer with the states 0, 1 or high impedance state.
- It uses a 7 bit address bus(AD7). The read(RD) and write(WR) inputs and the chip select input decide the operation. When the chip is selected either RD or WR input is active and the operation is carried out.
- The block diagram and function table of RAM is



CS_1	CS_2	RD	WR	Memory Operation	State of Data Bus
0	0	X	X	No Operation	High Impedance
0	1	X	X	No Operation	High Impedance
1	0	0	0	No Operation	High Impedance
1	0	0	1	Write Operation	Input Data to RAM
1	0	1	X	Read Operation	Output Data from RAM
1	1	X	X	No Operation	High Impedance

ROM Chip

- A ROM has two chip select input(CS1,$\overline{CS2}$) to select the chips.
- Since a ROM is read only the databus can only be in output mode.
- A nine bit address line(AD9) is used to specify the one of the 512 bytes stored in it.
- The block diagram is



Auxiliary Memory

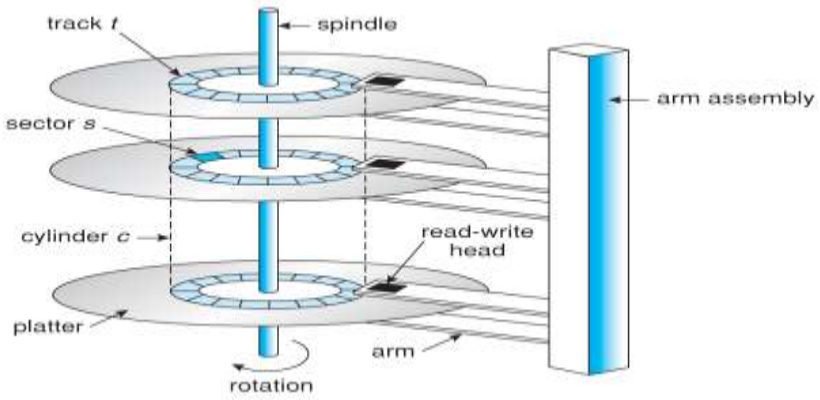
Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks. It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

Magnetic Disks:

- It is a circular plate made of metal or plastic coated with magnetized material. Many disks may be stacked on a spindle.
- The plate is divided into concentric circles called tracks. Tracks are divided into sections called sectors.
- A read/write head is used to point a specific track in the disk. The disk address provides the disk number, disk surface, track number and sector

number. When a track is reached the disk rotates to the specified sector and starts data transfer.

- Tracks near the circumference is longer than the center of the disk. To ensure equal number of bits in all tracks it stores data in variable density.
- Disks that are permanently attached to the unit assembly and are not removed often are called hard disk.
- Removable disk is called floppy disc.

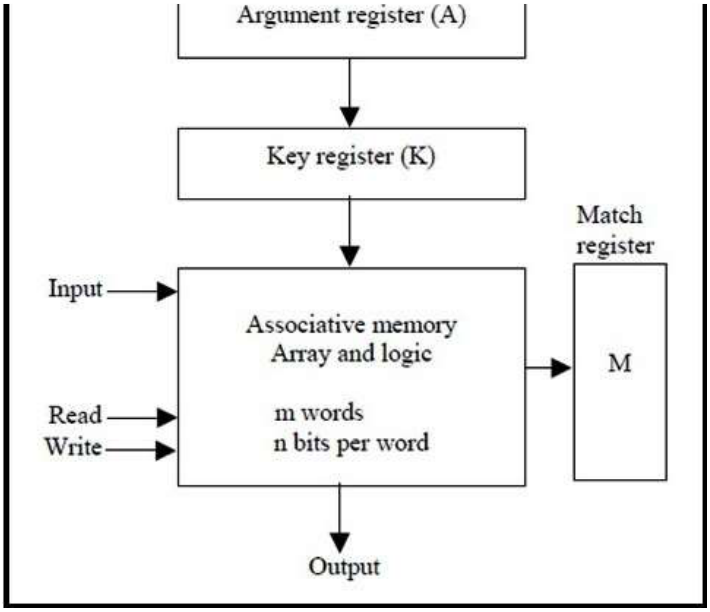


Magnetic Tape

- Magnetic tape consists of electrical, mechanical and electrical components to transfer data.
- Tape is a plastic coated with magnetic recording medium. Bits are recorded as magnetic spots along the tracks. Read/Write heads are mounted on each track to record and read data as a sequence in each track.
- Tape cannot be started or stopped inbetween characters hence gaps are inserted between records where tape can be stopped.
- Each record in the tape has an identification bit in the beginning.
- A tape address specifies record number and number of characters in the record. Records may be fixed or variable length.

Associative Memory

- Associative memory can also be called as Content Addressable Memory(CAM).
- CAM is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- Associative memory is more expensive than a RAM because each cell must have storage capability as well as logic circuits.
- Hardware organization of Associative memory



- The argument register contains the value to search for, and therefore is n bits wide to match the size of a word in the memory.
- The key register holds a mask that allows searching based on part of argument. If a bit in the key register is 1, then the corresponding bit in the argument and each memory word must be the same to be considered a match. If a bit in the key register is 0, then the corresponding bit is considered a match whether or not the argument and memory word are equal for that bit. This allows searches for words where any subset of the bits match the argument register.
- The match register, M, is m bits wide (could be huge), and will contain a 1 for each word that matches the masked argument, and a 0 for each word that does not.

Cache Memory

• Cache memory is small, high speed RAM buffer located between CPU and the main memory.
• Cache memory hold copy of the instructions (instruction cache) or Data (Operand or Data cache) currently being used by the CPU.

• The data or contents of the main memory that are frequently used by the CPU are stored in the cache memory so data can be accessed faster.

• 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 onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accomodate the new one.

• If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from chache

- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

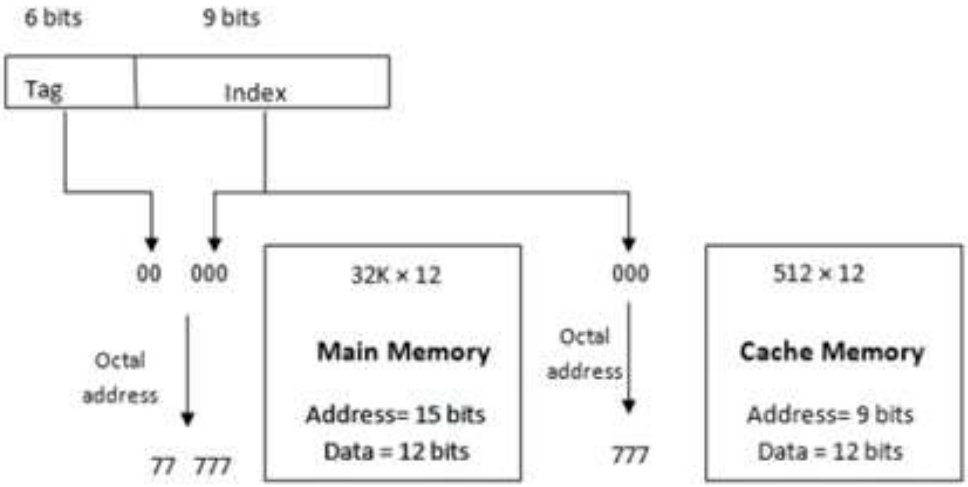
- The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

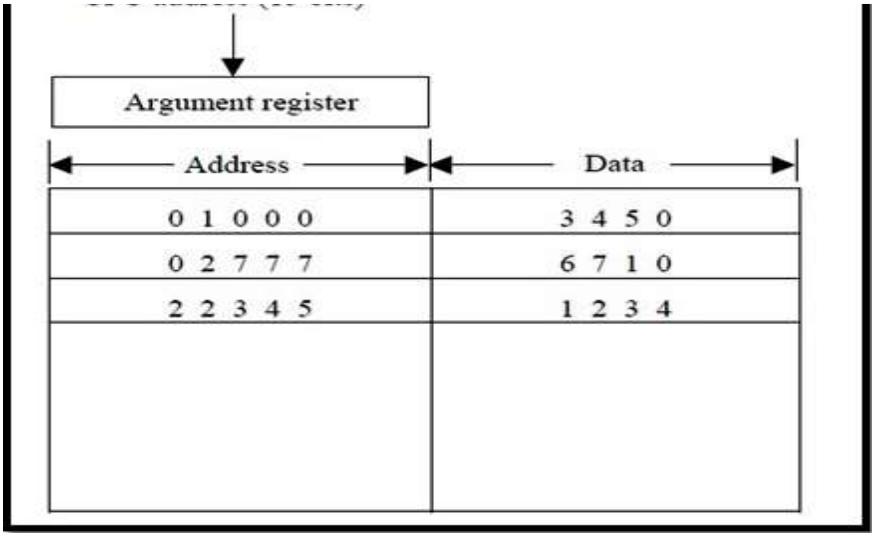
- Transformation of data from main memory to cache memory is refered as mapping process. The types of mapping procedures are

- Direct mapping
- Associative mapping,
- Set-Associative mapping.

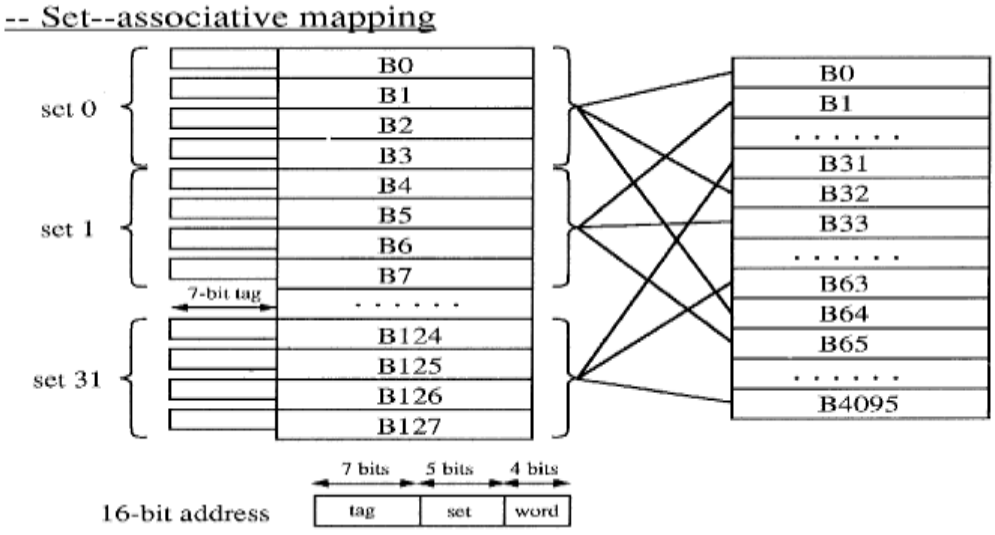
Direct mapping: In direct mapping assigned each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.



Associative mapping: In this type of mapping the associative memory is used to store content and addresses both of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of the any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.



Set-associative mapping: This form of mapping is a enhanced form of the direct mapping where the drawbacks of direct mapping is removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set..Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques



- The most common replacement algorithm used in cache are Random replacement, First-In First Out(FIFO) and Least Recently Used(LRU) algorithm.
- During the write operation if the main memory and cache memory are updated in paralled it is called Write-Through. If the cache memory is updated first and the main memory is updated only when the word is

removed from cache then it is a Write-Back method. This method is used when the data is updated several times when it is in cache.

Virtual Memory

- **Virtual Memory provides the illusion of a large memory** It allows programs to run regardless of actual physical memory size.
- It also allows many processes to run on a single machine simultaneously. It provides each process its own memory space.
- It has two addresses- Virtual Address and Physical address. Virtual address space used by the programmer. Physical Address is the actual physical memory address space.
- This concept divides memory (virtual and physical) into fixed size blocks called Pages in Virtual space and Frames in Physical space. Its Page size is equal to Frame size. All pages in the virtual address space are contiguous. Pages can be mapped into physical frames in any order. The program provides the virtual memory address space. The hardware converts this virtual address to physical address using a page table. Then the page is accessed from the main memory.
- If a new page is to be loaded into memory an older page is replaced by the new page. The page to be replaced is selected based on a Page Replacement algorithm like LRU, FIFO etc.
- Advantages:
  - Illusion of having more physical memory
  - Flexible program relocation
  - Provides protection of one process from another
  - Enables Virtual Address to Physical Addresses Translation