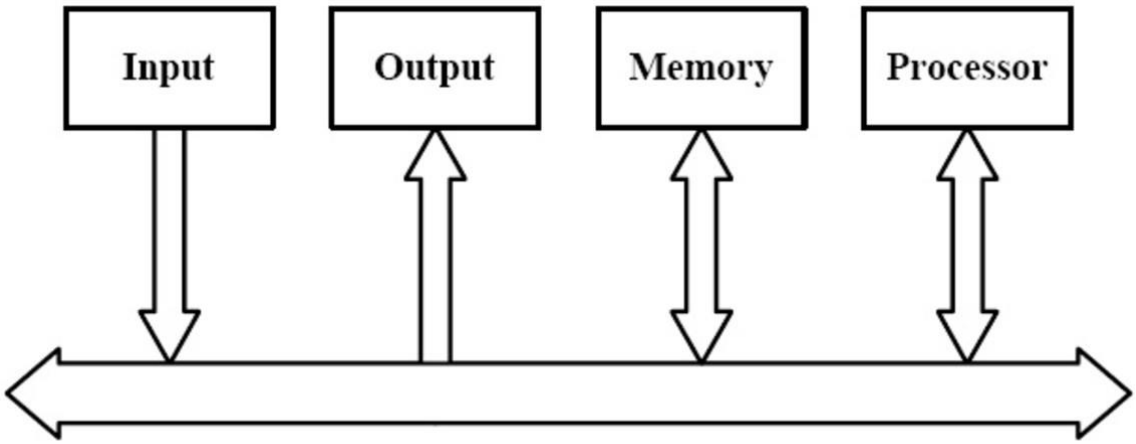
COMPUTER ORGANIZATION

Explain about different bus structures of computer system?

In computer systems, **bus structures** are essential for communication between various components, such as the CPU, memory, and input/output devices. A bus is a set of wires or connections that allows data to be transferred between these components. Different bus structures are used in computer architectures, and each has its unique characteristics. Here are the main types:

1. Single Bus Structure

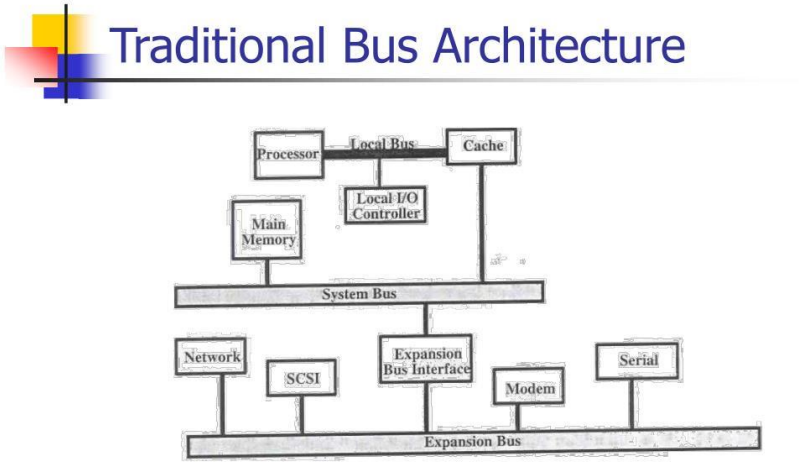
- **Definition**: A single bus connects all major components of the computer (CPU, memory, I/O devices).
- **Characteristics**:
    - **Simple Design**: Easy to implement and cost-effective.
    - **Shared Communication**: All components share the same bus for communication, which simplifies wiring.
    - **Performance Limitations**: Only one device can communicate over the bus at a time, which may create a bottleneck when multiple components need to transfer data simultaneously.
- **Use Cases**: Suitable for small-scale systems with fewer devices, like microcontrollers.



2. Multiple Bus Structure

- **Definition**: Several buses are used, separating communication paths for different subsystems like the memory bus, I/O bus, and CPU bus.
- **Characteristics**:
    - **Parallel Communication**: Multiple components can communicate simultaneously on different buses, increasing system performance.
    - **Higher Complexity**: More difficult to design and implement due to the need for bus controllers and more wiring.

    - **Reduced Bottleneck**: Improved data transfer speeds and efficiency compared to a single bus structure.
- **Use Cases**: Commonly used in modern computers and servers where high-speed data transfer and multitasking are important.



3. Address Bus

- **Definition**: Carries the memory addresses from the CPU to memory or I/O devices.
- **Characteristics**:
    - **Unidirectional**: The address bus only sends information from the CPU to other components.
    - **Width Determines Addressable Memory**: The number of lines (width) in the address bus determines the amount of memory the system can address (e.g., a 32-bit address bus can address 4 GB of memory).
- **Use Cases**: Essential in all computer systems for identifying the specific memory locations or I/O devices.

4. Data Bus

- **Definition**: Transfers actual data between the CPU, memory, and I/O devices.
- **Characteristics**:
    - **Bidirectional**: Data can travel in both directions—read from memory or written to memory.
    - **Width Determines Data Volume**: The width of the data bus (e.g., 8-bit, 16-bit, 32-bit) determines how much data can be transferred at once.
- **Use Cases**: Used in all systems to move data between the processor, memory, and peripheral devices.

5. Control Bus

- **Definition**: Carries control signals (like read/write instructions) from the CPU to other components to coordinate operations.
- **Characteristics**:
    - **Mixed Directionality**: Some control signals are sent from the CPU to other devices (e.g., read/write), while others may come from devices to the CPU (e.g., acknowledging data transfer).
    - **Synchronization**: Ensures that all components perform operations at the right time, according to control signals.
- **Use Cases**: Critical in managing how devices communicate and handle operations in the system.

Basic organization of computer?

The important parts a computer are the processor, main memory, hard disk(secondary memory), keyboard, monitor and peripheral devices. All these different parts of a computer are connected through a single bus called a Backplane Bus. A single bus interconnecting all the components of a computer is usually called a "Backplane Bus". Because it is considered to be a backbone communication medium, to which various components of a computer are attached. Backplane bus was replaced by multiple buses in the modern computers for achieving higher performance.

➢ The basic organization of a computer is shown in the below figure (a).

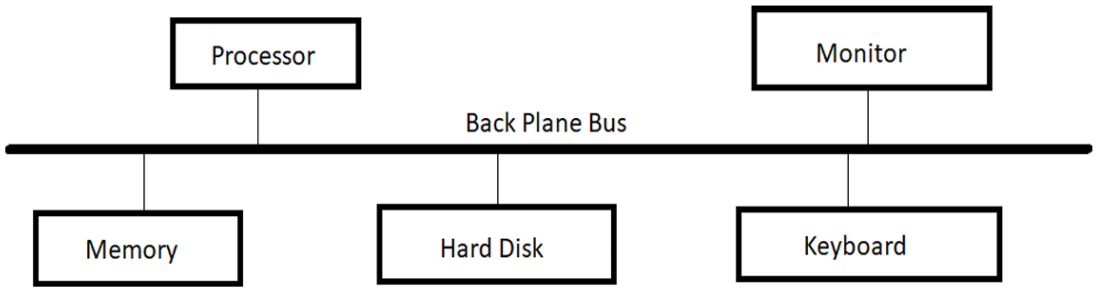


Figure (a): Basic organization of a digital computer

Processor: The processor or central processing unit (CPU) is responsible for fetching an instruction stored in the memory and execute it. It contains an arithmetic and logic unit (ALU) for manipulating data, a number of registers for storing data and a control unit (CU). Digital Computer operations can be described as follows:

• A set of instructions or a program will be stored in the memory.

• The CPU fetches those instructions sequentially from the memory, decodes them and performs the specified operation on associated data or operands in ALU.

- **Processed data and results** will be sent to an **output unit**, if required.
- All activities, like the processing of any operation and **data movement inside the computer**, are governed by the **control unit**.

Main Memory

In a computer, **memory** (whether main or secondary) is used to store data and instructions. The **main memory**, also called **Random Access Memory (RAM)**, allows the CPU to access any location at random to retrieve or store data. Main memory is typically made from **SRAM** (Static RAM) or **DRAM** (Dynamic RAM).
The **secondary memory**, or auxiliary memory, supplements the main memory and comes in various forms such as **CDs**, **DVDs**, **USB devices** (e.g., pen drives, external hard disks), and more.

Monitor

The **monitor** is an **output device** used to display the result or output from the computer. It is the primary output unit. Over time, monitors have evolved from simple text displays to devices capable of showing **high-quality graphics** and **animations**. Other output devices include **printers**, **plotters** (for printed output), and **speakers** (for sound output).

Keyboard

Early computers used card readers as the primary input device, but modern computers rely on the **keyboard**. Other commonly used input devices are the **mouse**, **scanner**, and **camera**.

Peripheral Devices

Several **peripheral devices** can be attached to the **backplane bus**. These devices can be either **output devices** (such as printers, plotters, and speakers) or **input devices** (such as scanners and cameras). The CPU interacts with these devices via the backplane bus to provide output or collect input.

Backplane Bus

The **backplane bus** is a set of wires partitioned into **control**, **address**, and **data wires**. Control wires carry signals to various units in the computer, address wires carry the memory location of data, and data wires carry the data itself.

Error detection code:

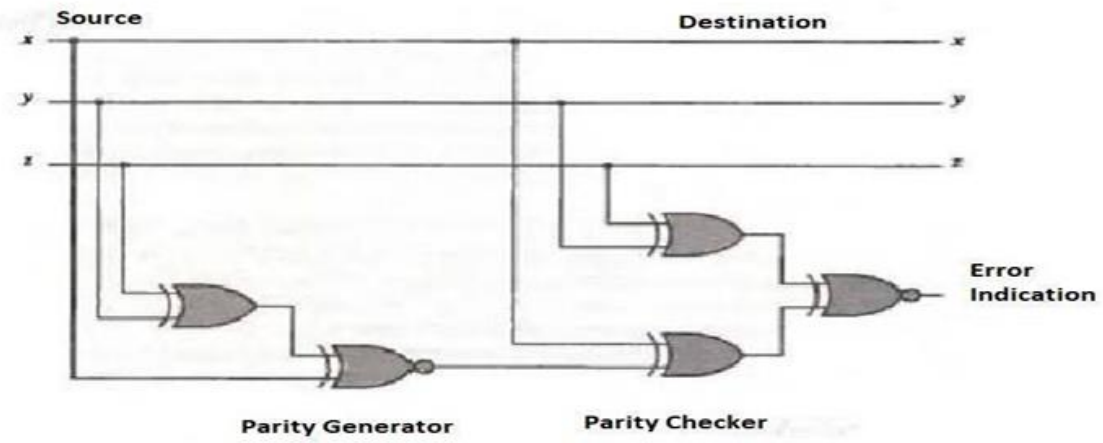
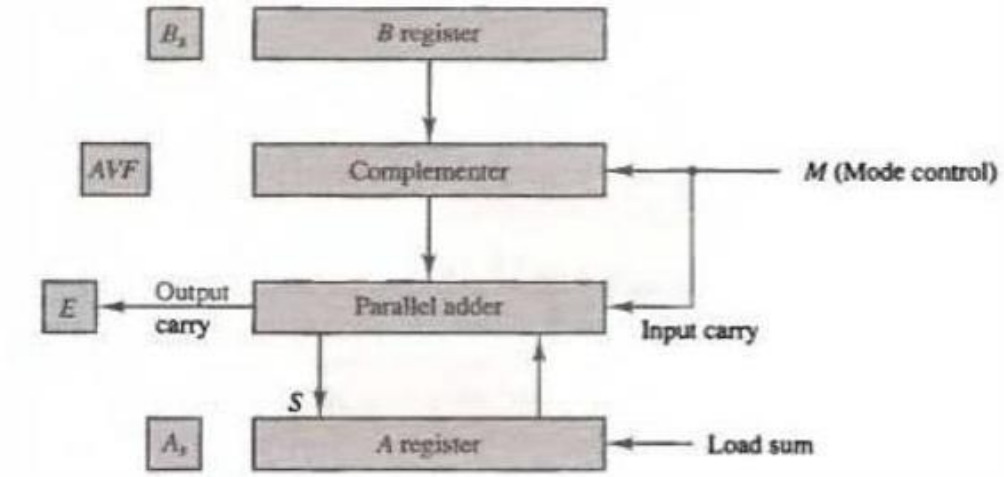
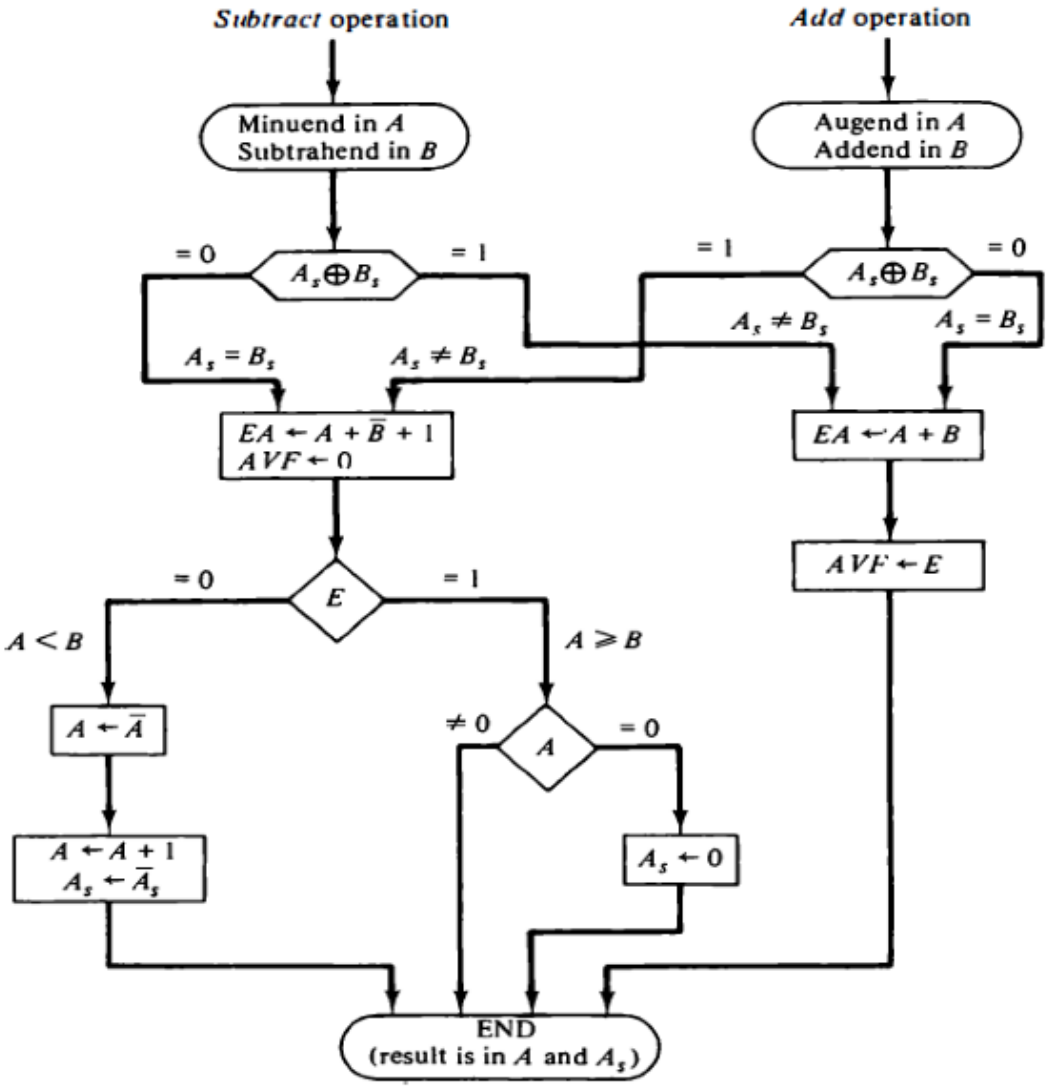


Figure (i): Hardware for addition and subtraction with Signed-Magnitude Data

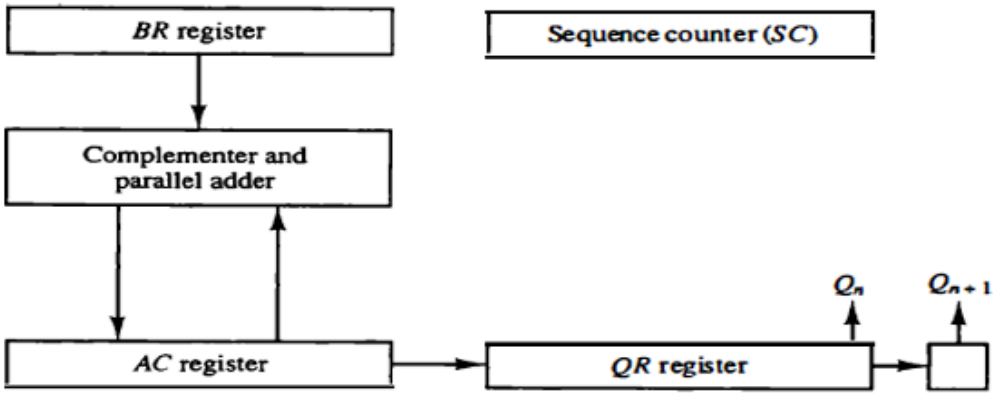


Hardware Algorithm



multiplier bit.

Hardware implementation of Booth algorithm Multiplication:



Example: multiplication of (- 9) x (- 13) = + 117 is shown below. Note that the multiplier in QR is negative and that the multiplicand in BR is also negative. The 10-bit product appears in AC and QR and is positive.

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	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

Figure (p): Example of Multiplication with Booth Algorithm.

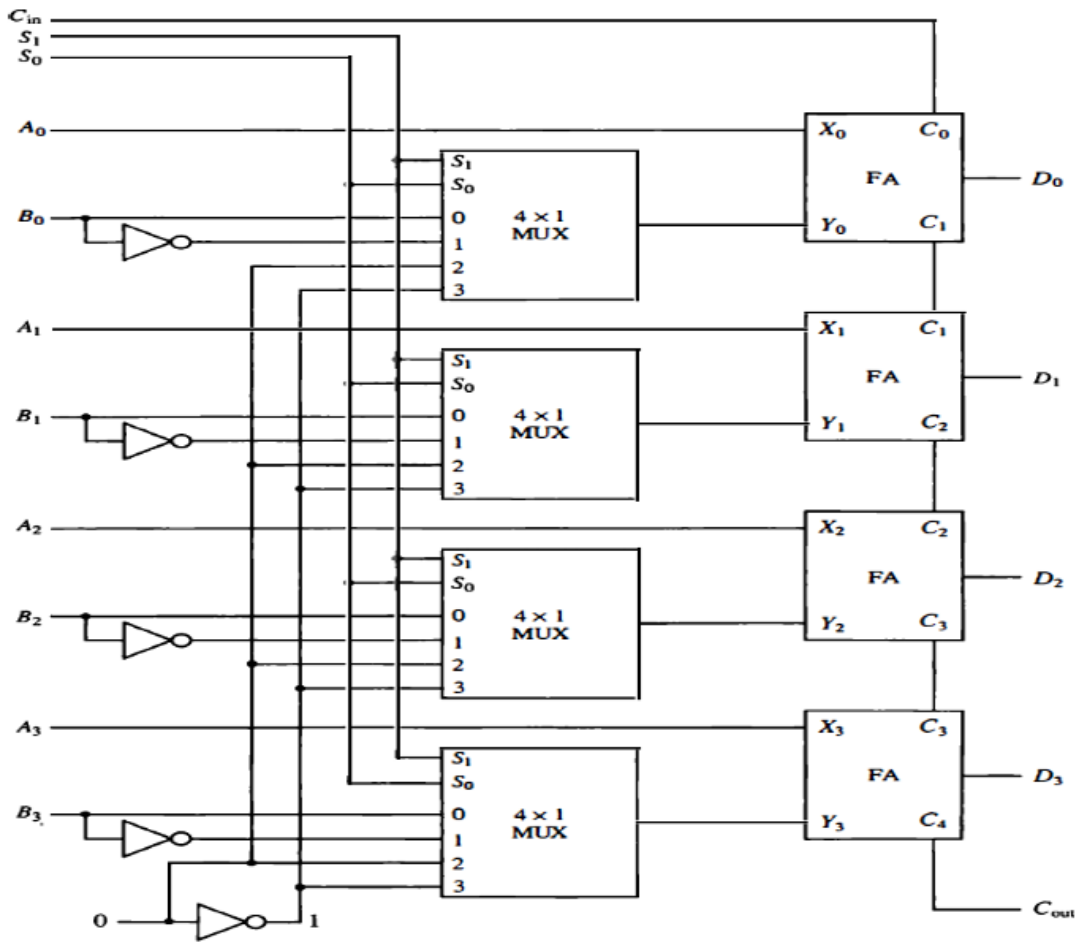


Figure (f): 4-bit Arithmetic Circuit

F 1, it is possible to generate the eight arithmetic microoperations listed in Table (b).

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

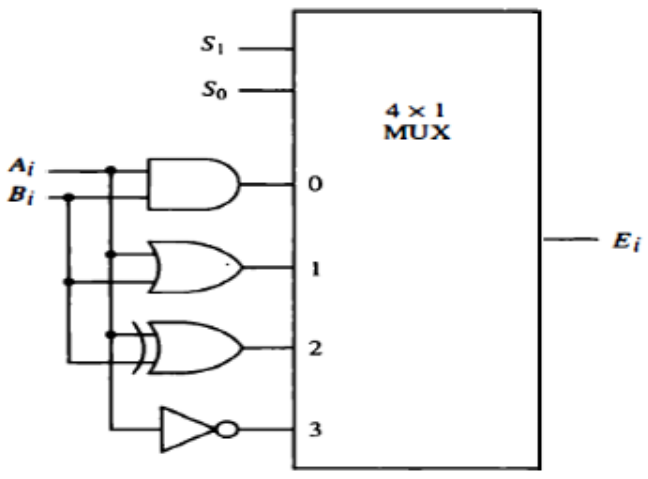
Table (b): Arithmetic Circuit Function Table

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table (c): Truth Tables for 16 functions of two variables

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Table (d): Sixteen Logic Microoperations



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

Figure (g): One stage of logic circuit

bolic notation for the shift microoperations is shown in the below table:

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

metic shift:

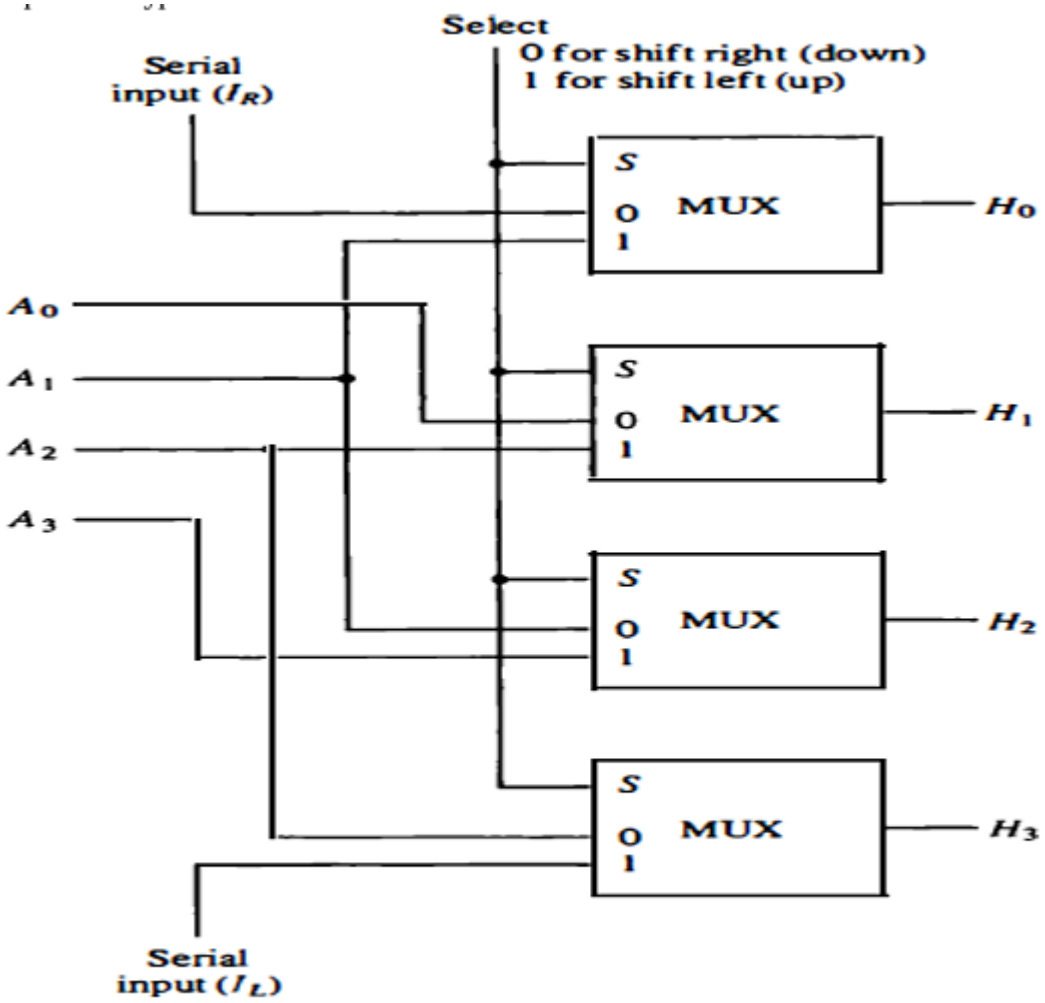


Figure (i): 4-bit combinational Circuit shifter.

Instruction code:

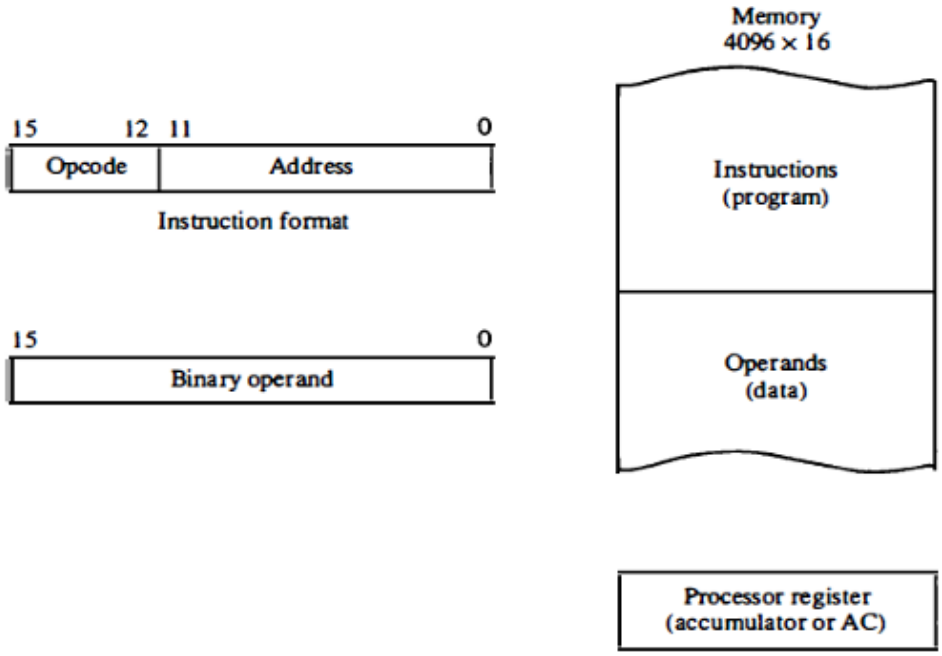


Figure (k): Stored program organization

Direct & Indirect address:-

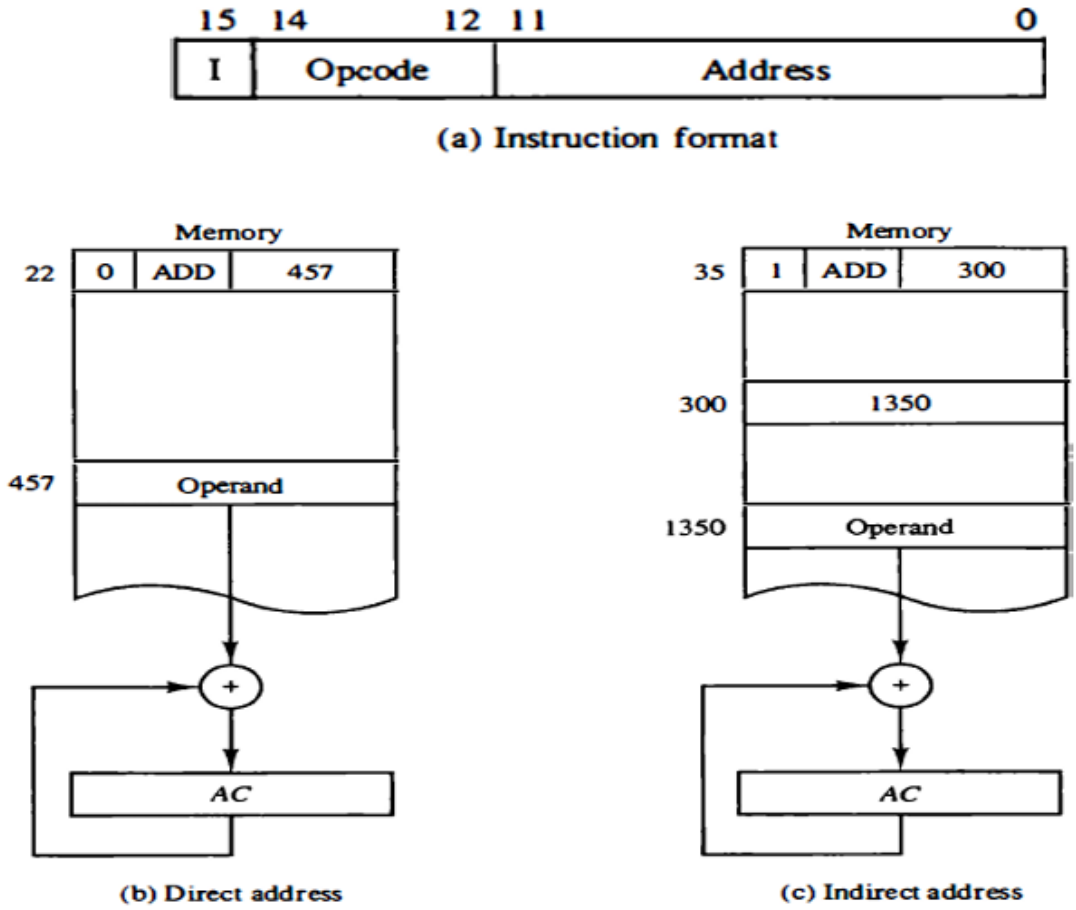


Figure (l): Demonstration of direct and indirect address.

The registers available in the computer are shown in the below figure (m) and table (f), a brief description of their function and the number of bits that they contain also given.

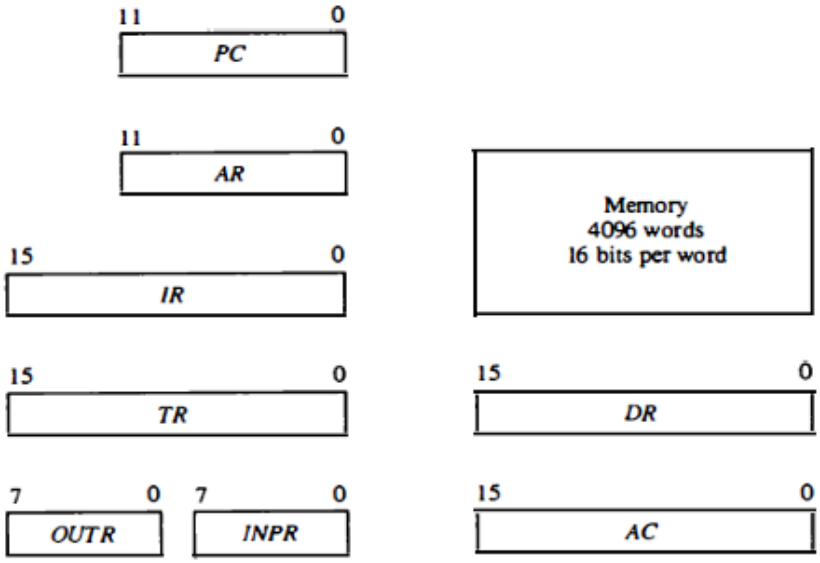


Figure (m): Basic computer registers and memory.

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

Table (f): List of Registers for the Basic computer.

Computer Instructions:

The basic computer has **three types of instruction code formats**,

1. Memory-reference instruction.
2. Register-reference instruction.
3. An input-output instruction.

Each format has 16 bits. 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.

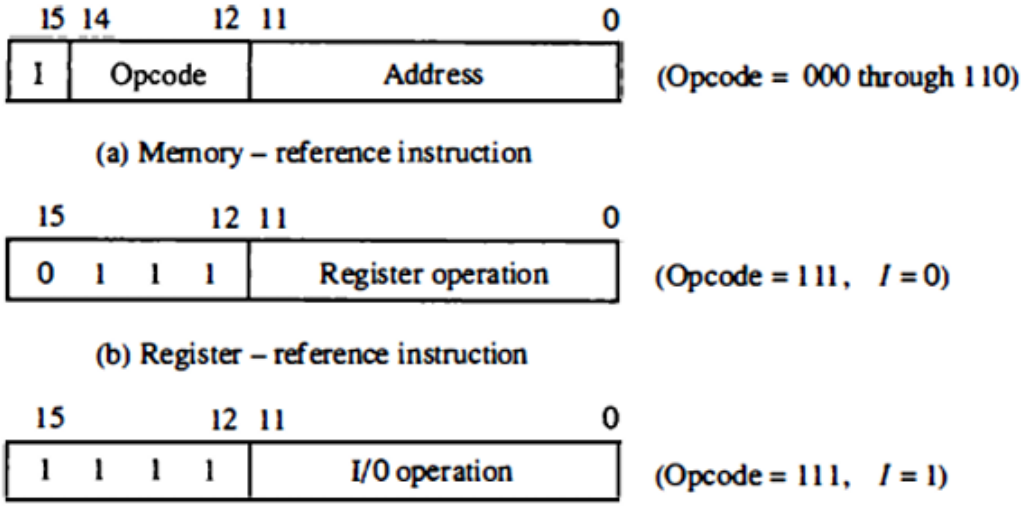


Figure (n): Basic computer instruction formats

The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction.

- If the three opcode bits in positions 12 to 14 are not equal to 111, the instruction is a **memory-reference type** and the bit in position 15 is taken as the addressing mode I. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I = 0 for direct address and I = 1 for indirect address.
- If the 3-bit opcode = 111, control then inspects the bit in position 15. If this bit = 0, the instruction is a **register-reference type**. These instructions use 16 bits to specify an operation.
- If the bit I = 1, the instruction is **an input-output type**. These instructions also use all 16 bits to specify an operation.

The instructions for the computer are listed in Table (g, h, i).

	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

Table (g): Memory-reference instructions

Memory-Reference Instructions:

- ✓ The below Table (k) lists the seven memory-reference instructions. The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table.
- ✓ The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4.

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

Table (k): Memory-Reference Instructions

AND : AND to AC

This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC. The rnicrooperations that execute this instruction are:

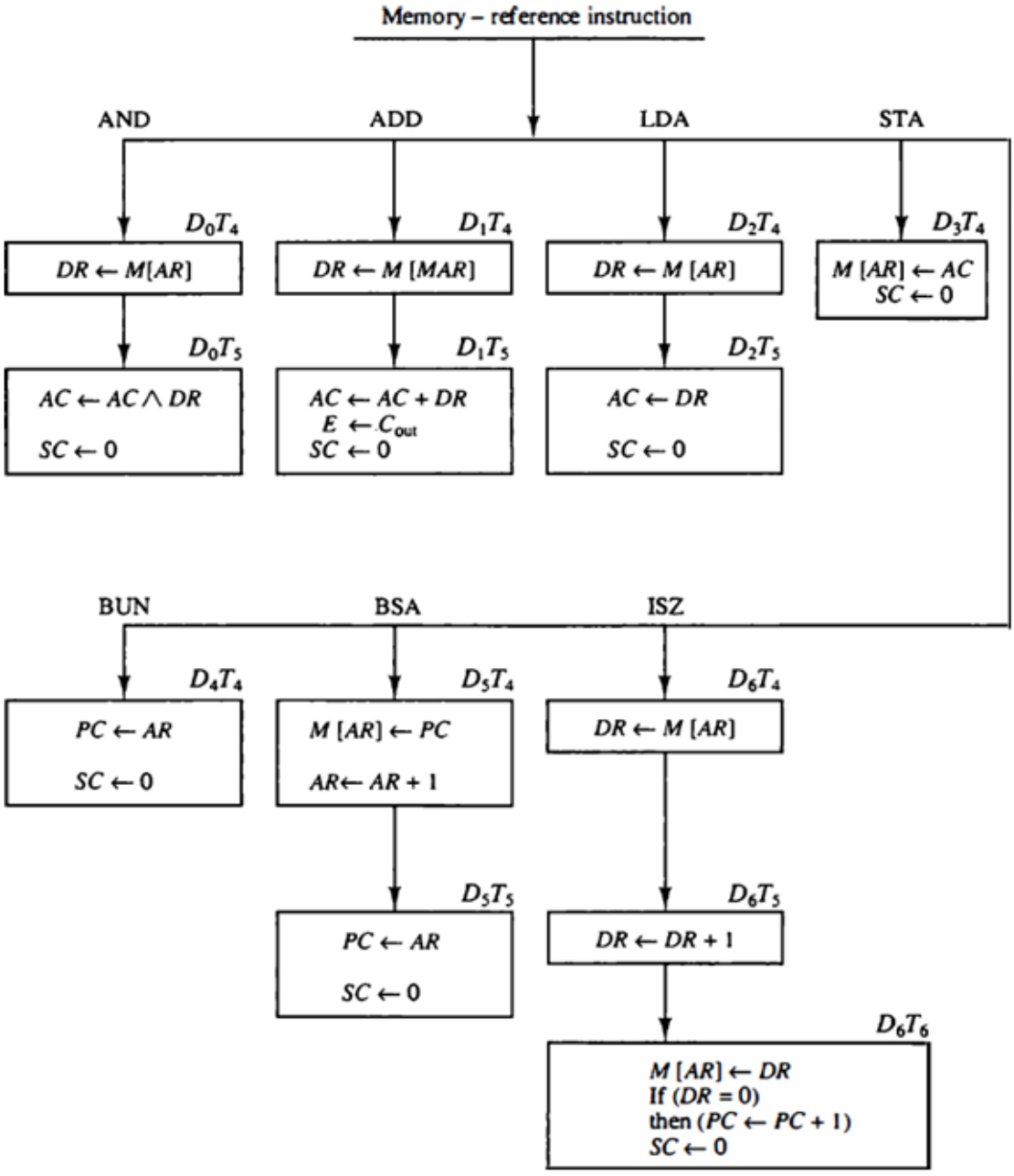


Figure (g): Flowchart for Memory-reference instructions

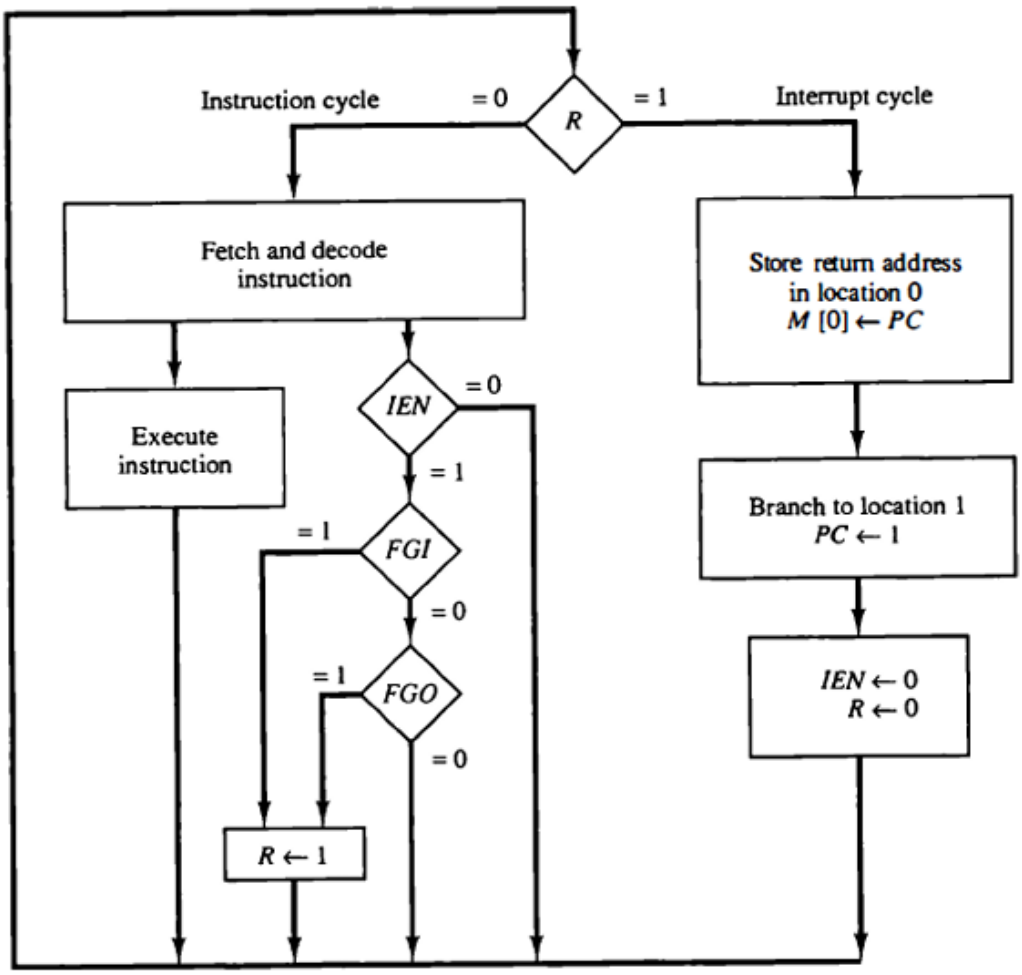


Figure (s): Flowchart for interrupt cycle

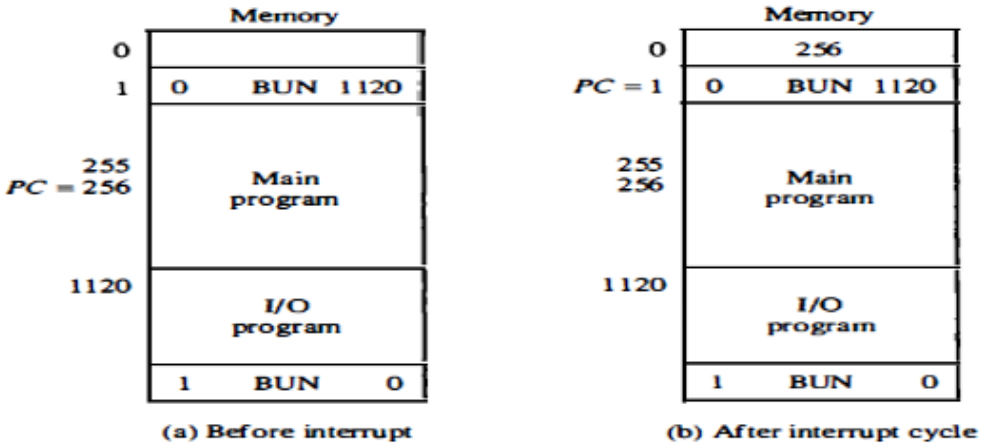


Figure (t): Demonstration of Interrupt Cycle

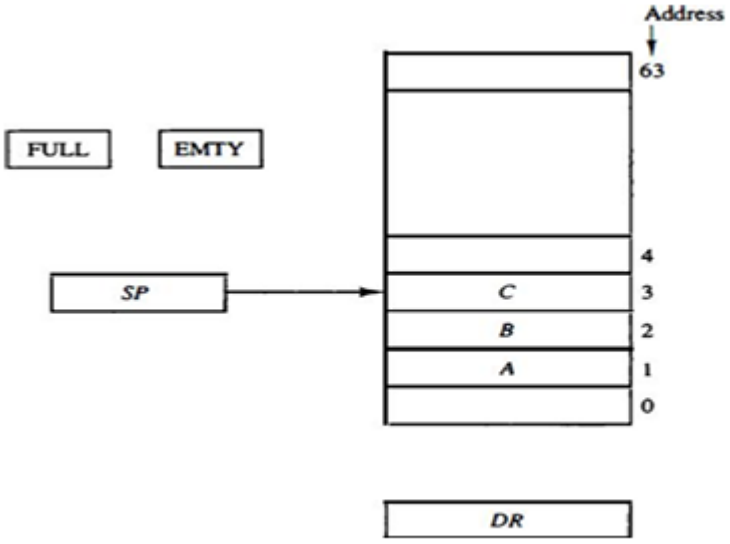


Figure (4): the organization of a 64-word register stack.

Memory Stack

A stack can exist as a stand-alone unit as in Figure (3) or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. Figure (4) shows a portion of computer memory partitioned into three segments: program, data, and stack.

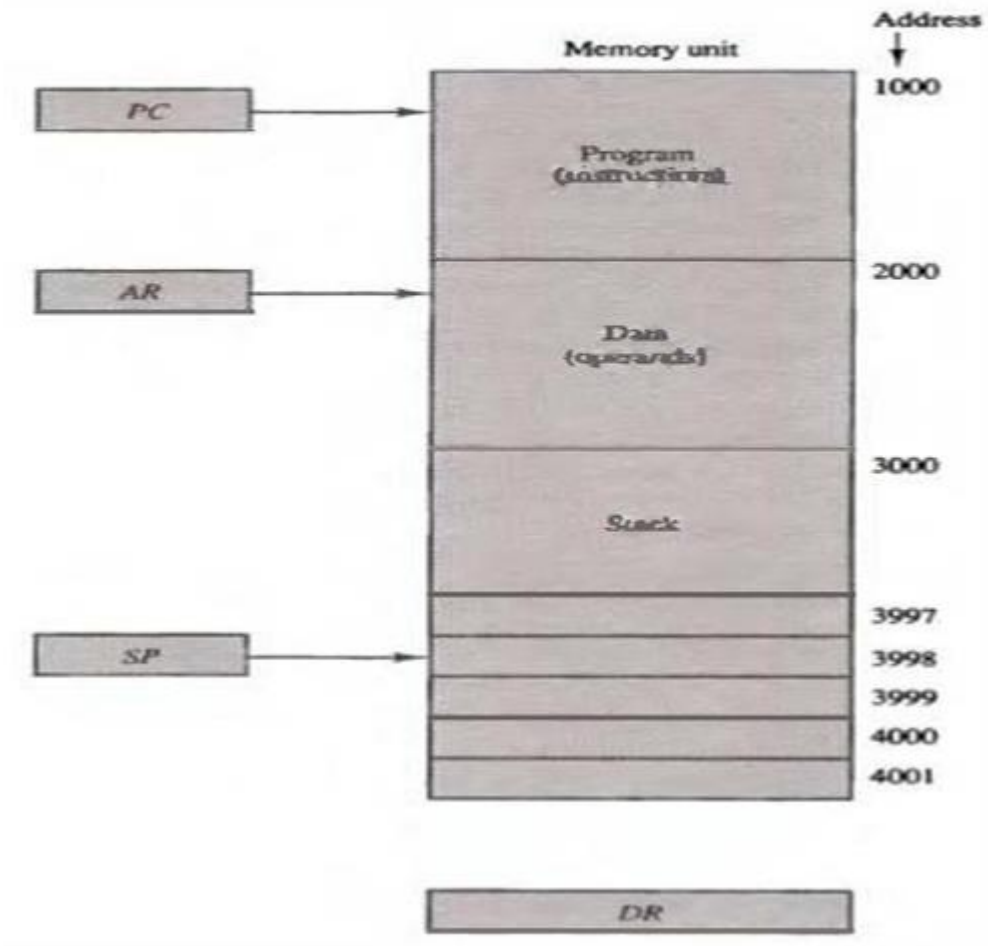


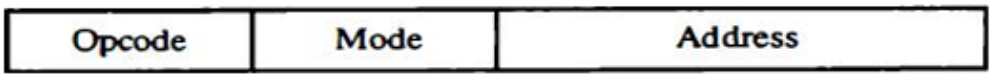
Figure (5): Computer memory with program, data and stack segments

Addressing Modes:

The **operation field** of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The **addressing mode** specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

In simple terms, **Addressing mode** is the way in which the location of an operand can be specified in an instruction. It generates an effective address (the actual address of the operand).

Instruction format with mode field



Types of Addressing Modes:

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode:
5. Autoincrement or Autodecrement Mode
6. Direct Address Mode
7. Indirect Address Mode
8. Relative Address Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

There are two modes that need **no address field at all**. These are the implied and immediate modes.

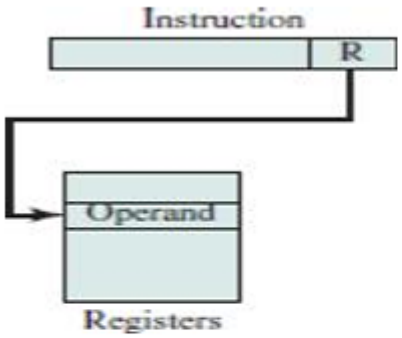
1. **Implied Mode:** In this mode the operands are specified implicitly in the definition ofthe instruction.
   For example, the instruction "complement accumulator (CMA)" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2. **Immediate Mode:** In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.

EX: LDAC #34H
LDAC loads data from memory to accumulator.
Therefore, AC=00110100.

When the address field specifies a processor register, the instruction is said to be in the register mode.

3. **Register Mode:** In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction.



E.g : LDAC R

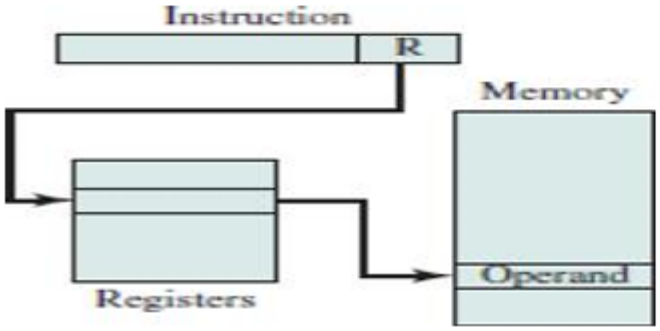
instruction reads data from register R

LDAC R

R : 5 → store value in AC

∴ AC = 5

4. **Register Indirect Mode:** In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself.



EX: **LDAC (R1)**

If R1cotains the address of an operand in the memory, for example: address of an operand is 2000 which contains a value 350. Result: 350 is stored in the AC.

5. **Autoincrement Mode:** This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. The 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 to the next value.

E.g : LDAC (R)

R : 5

5 : 10 → store in AC

instruction reads address from register R .

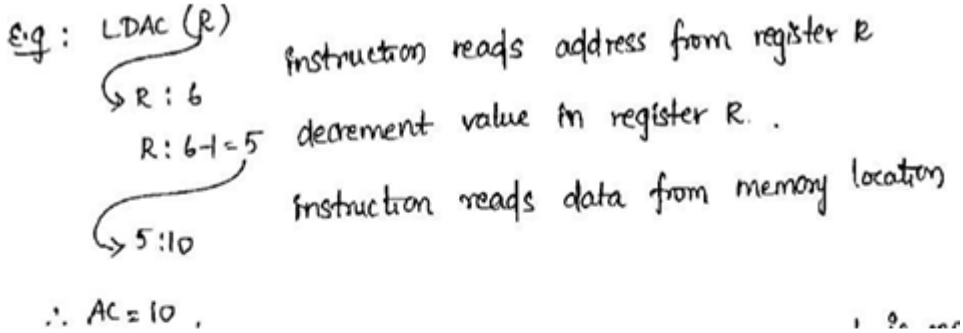
instruction reads data from memory location

AC = 10

R : 5+1 = 6 .

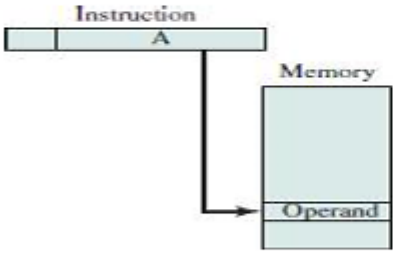
6. **Autodecrement Mode**

The effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented and then the value is accessed.



Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.

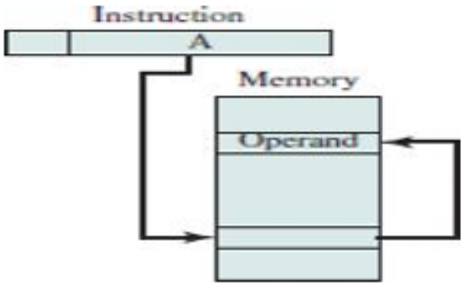
7. **Direct Address Mode:** In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.



Ex: **LDAC 5000**

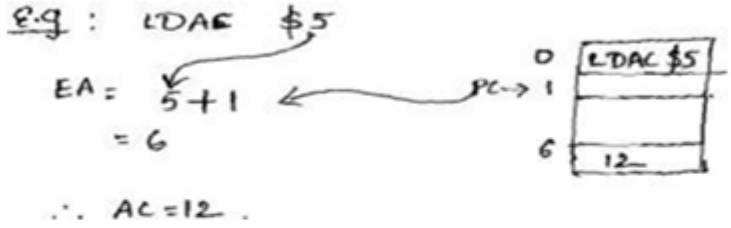
This instruction reads the operand from the Memory location 5000. if the memory location 5000 contains a value 250, then it will be stored in AC.

8. **Indirect Address Mode:** In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.



9. Relative Address Mode:

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.



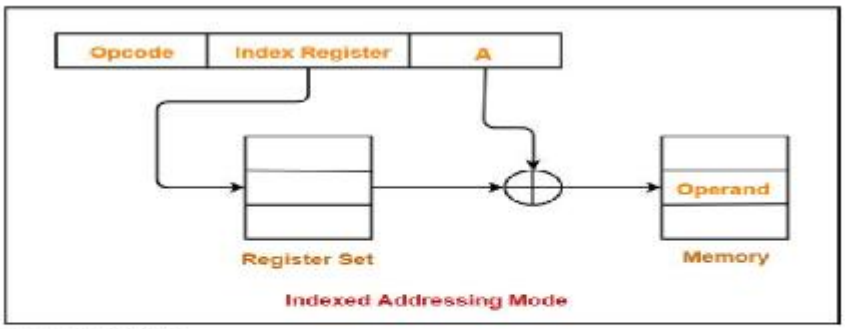
EX:

PC= address of next instruction , i.e., 1. The address given in the instruction is 5

Then EA= 5 + 1= 6 which contains a value 12. Finally AC contains 12.

10. Indexed Addressing Mode:

In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.



Ex: LDAC A(XR)

Assume XR=100, A=500

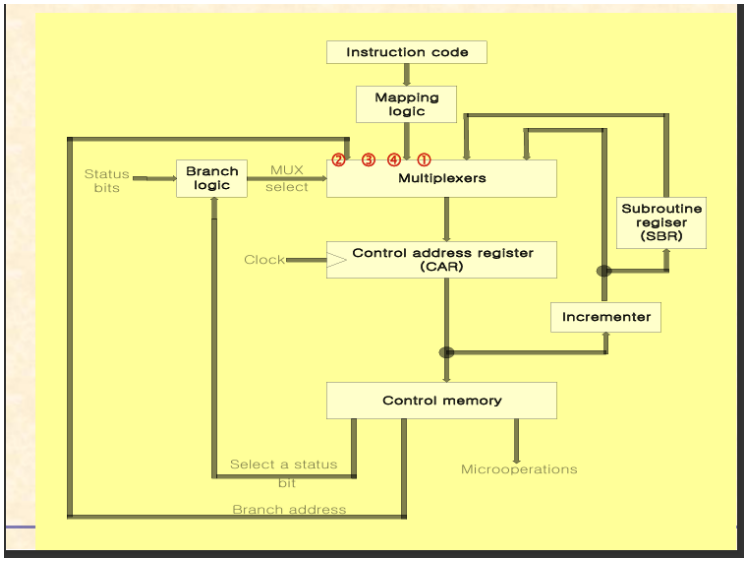
This instruction reads the operand from the effective address (600)

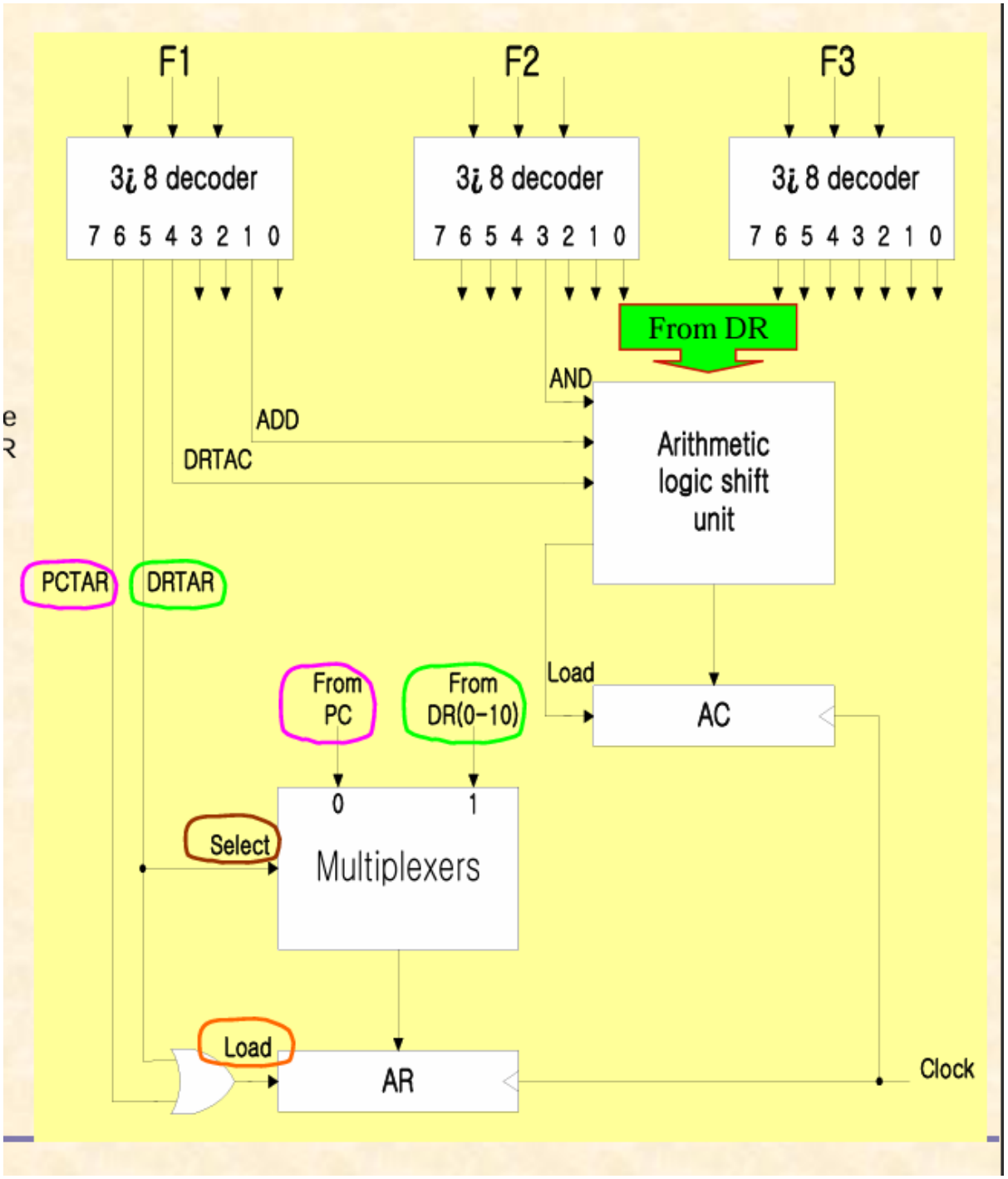
i.e., EA= XR contents (index register) + 500

= 100 + 500=600

If memory location at 600 contains a value 55 (assume), This 55 will be stored in AC.

Address Sequencing:

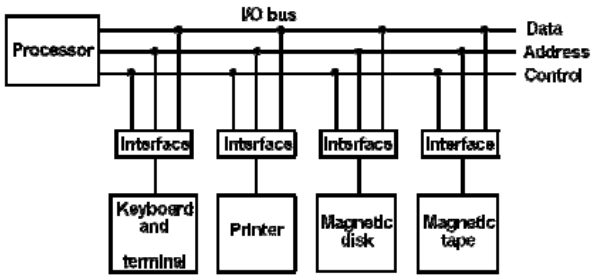



F1
F2
F3
3¡ 8 decoder
7 6 5 4 3 2 1 0
3¡ 8 decoder
7 6 5 4 3 2 1 0
3¡ 8 decoder
7 6 5 4 3 2 1 0
From DR
AND
ADD
DRTAC
Arithmetic
logic shift
unit
PCTAR
DRTAR
From
PC
From
DR(0-10)
Load
AC
0
1
Select
Multiplexers
Load
AR
Clock

I/O BUS AND INTERFACE MODULES

A typical communication link between the processor and several peripherals is shown in Fig.

The I/O bus consists of data lines, address lines, and control lines.



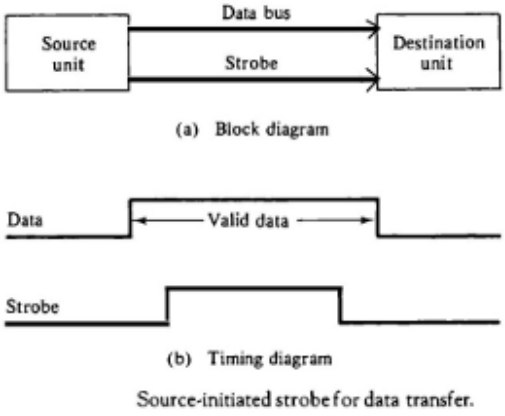
Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device.

To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the

ASYNCHRONOUS DATA TRANSFER:-

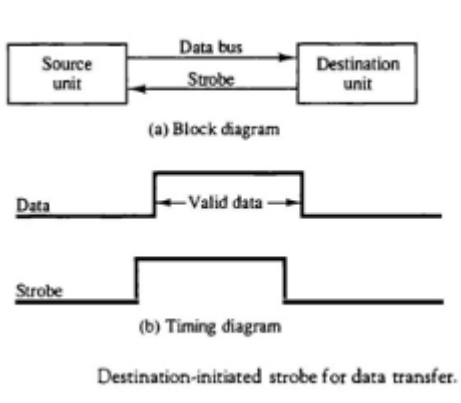
STROBE CONTROL

- 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.
- The data bus carries the binary information from source unit to the 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 in the bus.
- As shown in the timing diagram the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse.



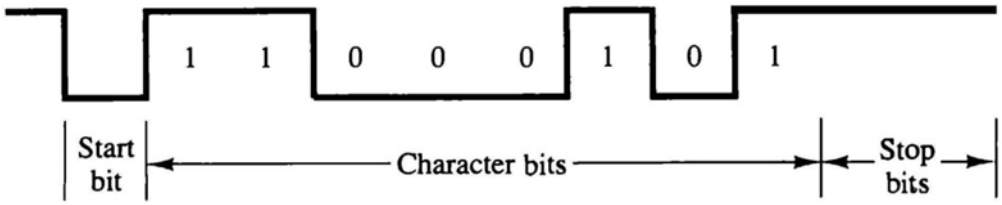
Source-initiated strobe for data transfer.

- The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data. Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.



Destination-initiated strobe for data transfer.

Asynchronous serial transmission.



DMA CONTROLLER

- The following figure shows the block diagram of a typical DMA controller