COMPUTER ORGANIZATION

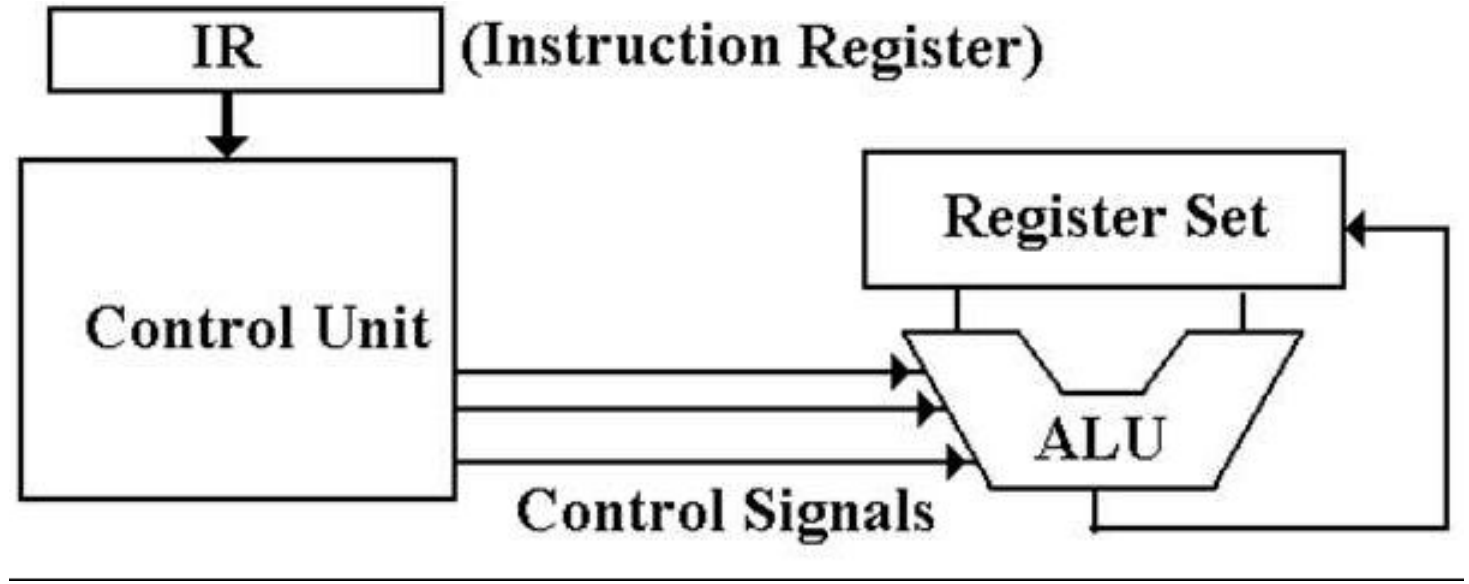
B.TECH III SEM

CSE-4

N SANTOSHI
ASSISTANT PROFESSOR
DEPARTMENT OF ECE

CONTROL UNIT

- CPU is partitioned into Arithmetic Logic Unit (ALU) and Control Unit (CU).
- The function of control unit is to generate relevant timing and control signals to all operations in the computer.
- It controls the flow of data between the processor and memory and peripherals



There are two methods to implement the control unit:

- **Hardwired:** The control signals are generated as an output of a set of basic logic gates, the input of which derives from the binary bits in the Instruction Register.
- **Microprogrammed:** The control signals are generated by a microprogram that is stored in Control Read Only Memory.
- The Hardwired and Microprogrammed control unit generates the **control signals to** fetch and execute instructions.

Hardwired Control Unit

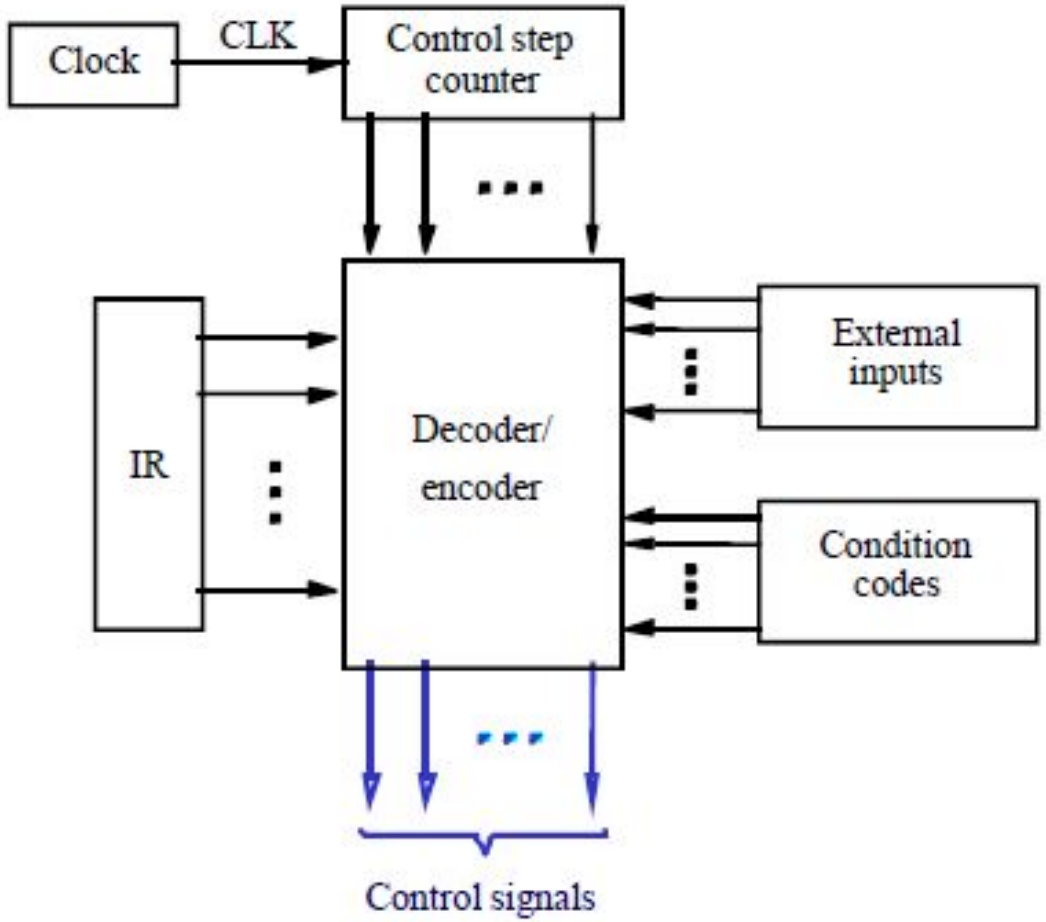
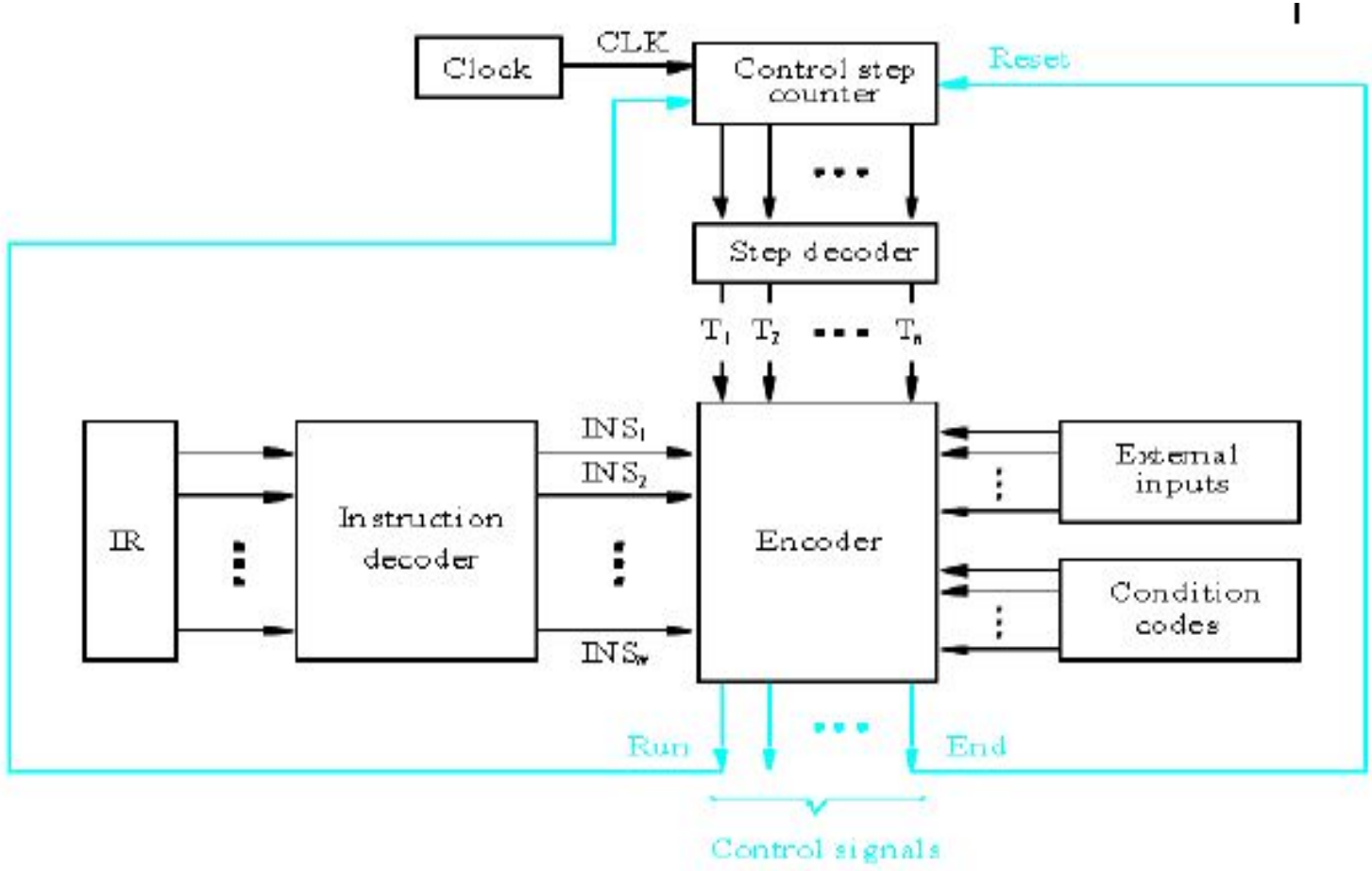


Figure 7.10 Control unit organization.

Hardwired Control Unit

- The decoder/encoder is a combinational circuit that generates a set of required control signals.
- A control step counter is used to keep track of the control steps.
- Each count of this counter corresponds to one control step.
- The required control signals are determined by the following information:
    1. contents of the control step counter
    2. contents of IR register
    3. contents of the condition code flags
    4. External input signals, like MFC and interrupt request.

Detailed Block Description



Detailed Block Description

- The step decoder generate a separate clock signal for each step, or time slot, in the control sequence.
- The instruction decoder decodes the instruction loaded in IR.
- The output of the instruction decoder consists of a separate line for each of the ‘m’ machine instruction.
- According to the code in the IR, only one line amongst all output lines of decoder is set to 1 and all other lines are set to 0.
- The input signals to encoder are combined to generate the individual control signals like add, read, etc.
- The End signal starts the a new instruction fetch cycle by resetting the control step counter to its starting value.
- When run=1, the counter to be incremented by one at the end of every clock cycle.
- When run=0, the counter stops counting and this is needed whenever the WMFC signal is activated.

Hardwired Control Unit

- Advantages of Hardwired Control Unit:

  1. Fast because control signals are generated by combinational circuits.

  2. The delay in generation of control signals depends upon the number of gates.

- Disadvantages of Hardwired Control Unit:

  1. More the control signals required by CPU, more complex will be the design of control unit.

  2. No Flexibility. Modification in control signal is very difficult i.e. it requires rearranging of wires in the hardware circuit.

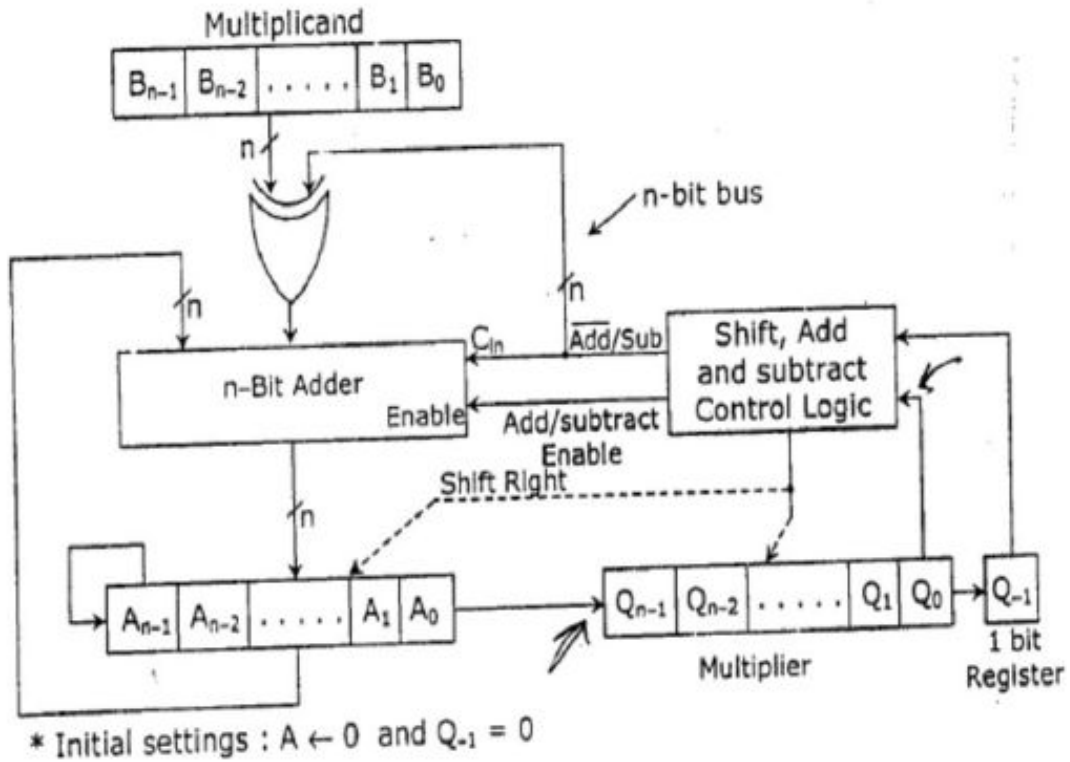
  3. Difficult to add new feature in existing design of control unit.

Hardwired Control Unit Design Methods

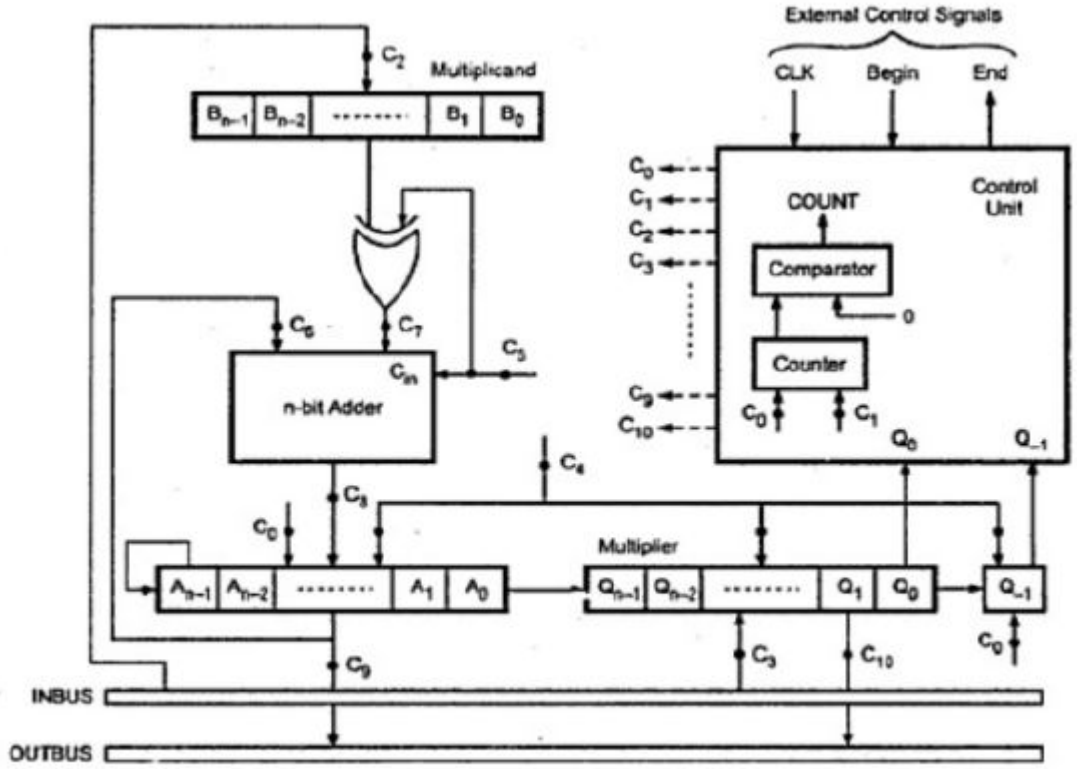
1. State-table Method
2. Delay-element Method: Uses clocked delay element (D-Flip Flop)
3. Sequence-counter Method: Uses counter for timing purposes
4. PLA Method: Uses programmable logic array

Hardwired Control Unit Design Methods (cont.)

- Let us consider 2's complement multiplier control circuit for illustration.



Hardwired Control Unit Design Methods (cont.)



Multiplier control circuit with control signals

Hardwired Control Unit Design Methods (cont.)

Control signal	Operation controlled
C_0	Clear A, Q_{-1} and count ← n
C_1	Decrement count
C_2	Transfer word on INBUS to B
C_3	Transfer word on INBUS to Q
C_4	Shift right register A, Q and Q_{-1}
C_5	2's complement multiplicand
C_6	Transfer A to left input of adder
C_7	Transfer B to right input of adder
C_8	Transfer adder output to A
C_9	Transfer A to OUTBUS
C_{10}	Transfer Q to OUTBUS

Control signals for 2's complement multiplier

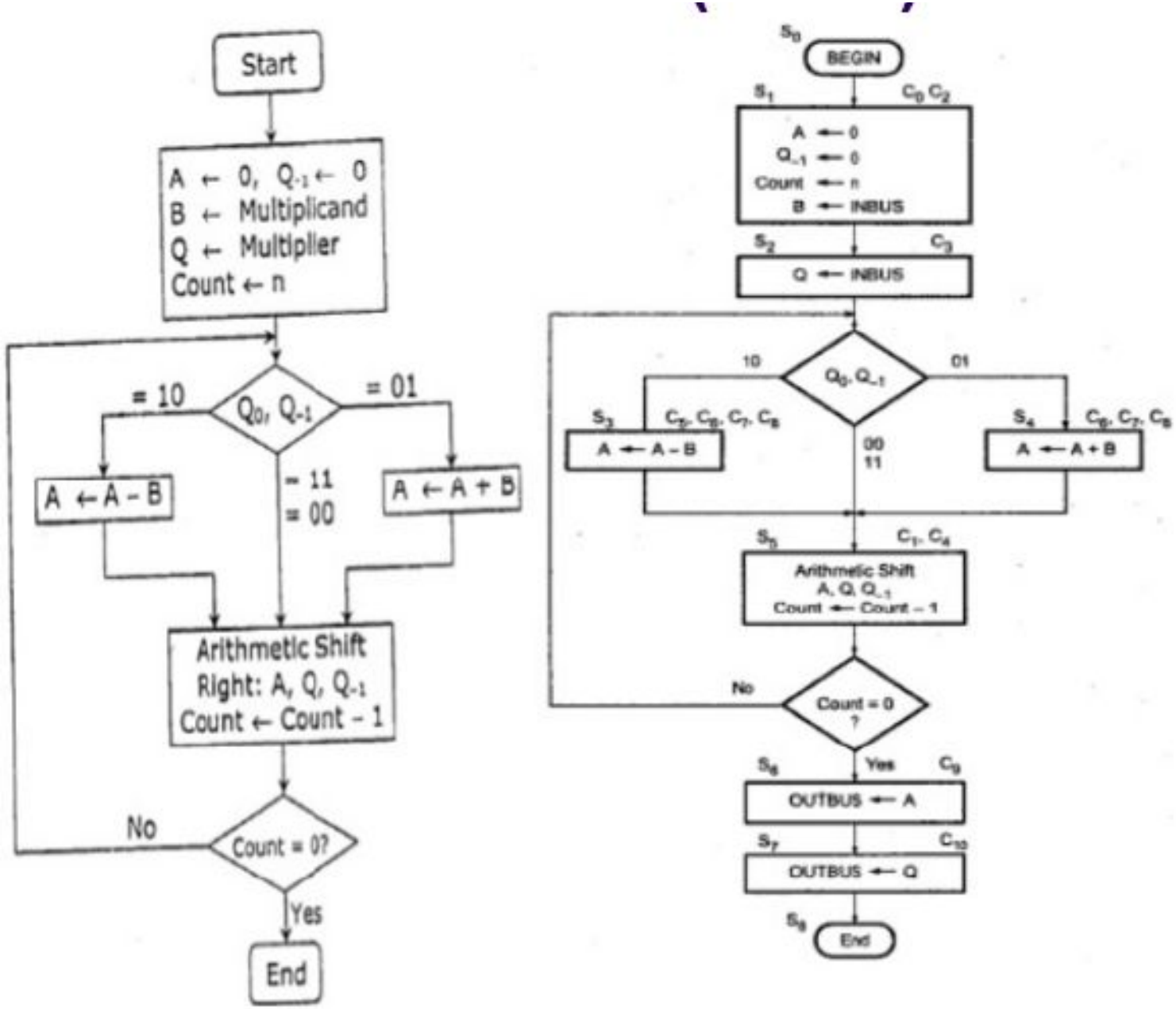
State-table Method

- Classical method of sequential circuit design.
- Attempts to minimize the amount of hardware.
- It starts with the construction of state transition table. In every state the control unit generates a set of control signals.
- Control unit transmits from one state to another state depending on its:
  1. Current state
  2. Input to the controller

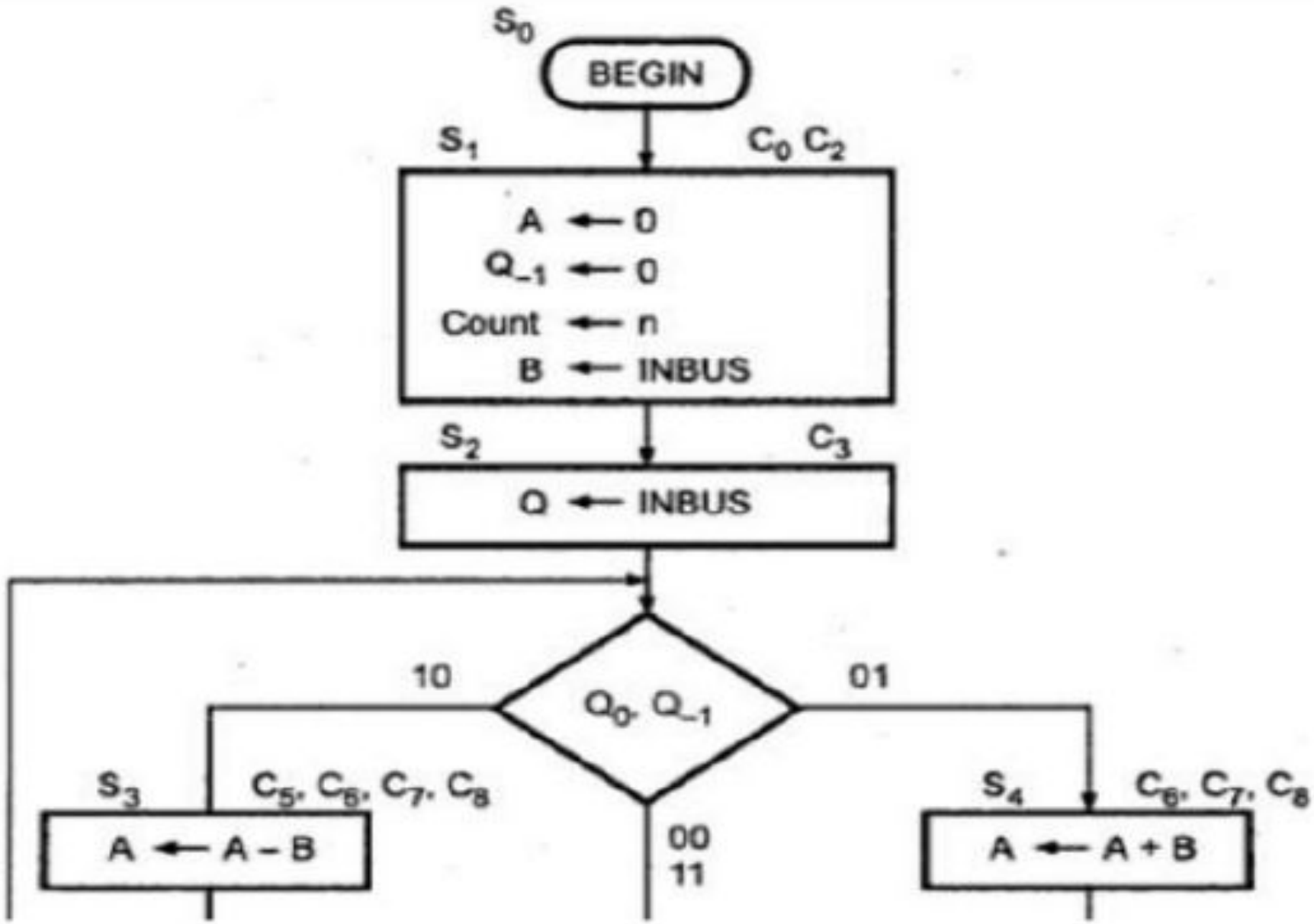
- State Assignment: States are assigned as S1,S2,S3…;each such assignment specifies a particular state of the controller at the specific time step. State table derived from state assignment

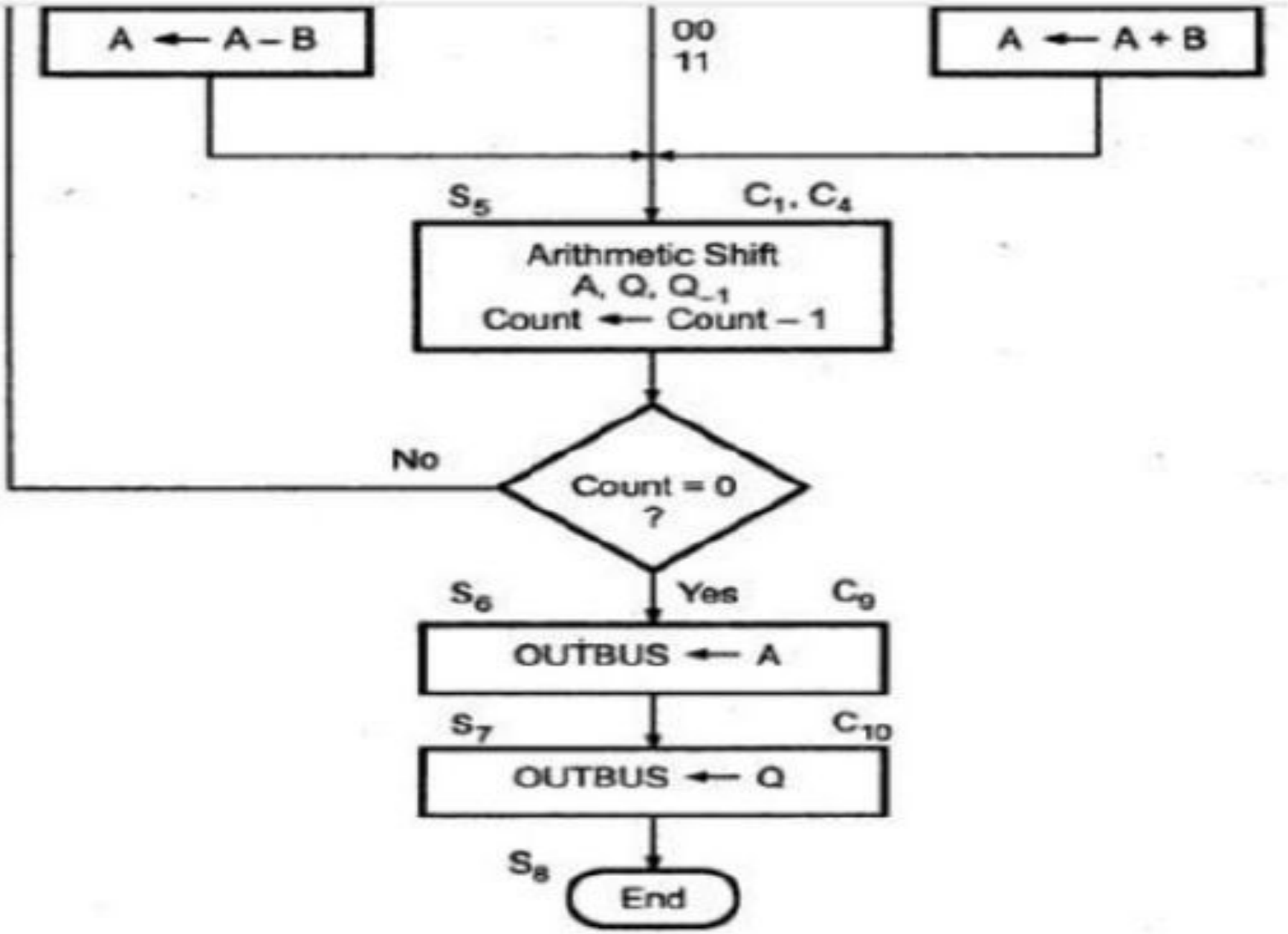
2’s complement multiplier

- State table construction is necessary.
- Associate a state with every micro-operation block giving nine states from S0 through S8.
- There are four primary input signals BEGIN, COUNT, Q0 and Q-1, so sixteen possible input combinations.
- Each entry in the table indicates next state followed by list of control signals that are activated.



Flowchart for 2's complement multiplication


S_0
BEGIN
S_1
C_0 C_2
A ← 0
Q_{-1} ← 0
Count ← n
B ← INBUS
S_2
C_3
Q ← INBUS
10
Q_0, Q_{-1}
01
00
11
S_3
C_5, C_6, C_7, C_8
A ← A − B
S_4
C_6, C_7, C_8
A ← A + B


A ← A – B
00
11
A ← A + B
S5
C1, C4
Arithmetic Shift
A, Q, Q–1
Count ← Count – 1
No
Count = 0
?
S6
Yes
C9
OUTBUS ← A
S7
C10
OUTBUS ← Q
S8
End

Inputs				States								
BEGIN	COUNT	Q_0	Q_{-1}	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
0	0	0		S_0	S_2, C_0 C_2	S_5, C_3			S_5, C_1, C_4			
0	0	1		S_0	S_2, C_0 C_2	S_4, C_3		S_5, C_6 C_7, C_8	S_4, C_1, C_4			
0	1	0		S_0	S_2, C_0 C_2	S_3, C_3	S_5, C_5 C_6, C_7 C_8		S_3, C_1 C_4			
0	1	1		S_0	S_2, C_0 C_2	S_5, C_3			S_5, C_1 C_4			
0	1	0		S_0	S_2, C_0 C_2				S_8, C_1 C_4	S_7, C_9	S_8, C_{10}	S_0, END
0	0	1	1	S_0	S_2, C_0 C_2				S_6, C_1 C_4	S_7, C_9	S_8, C_{10}	S_0, END
0	1	0	1	S_0	S_2, C_0 C_2				S_8, C_1, C_4	S_7, C_9	S_8, C_{10}	S_0, END

State table for multiplier control unit

0 0 1 1	S_0	S_2, C_0 C_2				S_6, C_1 C_4	S_7, C_9	S_8, C_{10}	S_0, END
0 1 1 0	S_0	S_2, C_0 C_2				S_6, C_1, C_4	S_7, C_9	S_8, C_{10}	S_0, END
0 1 1 1	S_0	S_2, C_0 C_2				S_6, C_1 C_4	S_7, C_9	S_8, C_{10}	S_0, END
1 0 0 0	S_1								
1 0 0 1	S_1								
1 1 0 0	S_1								
1 1 0 1	S_1								
1 0 1 0	S_1								
1 0 1 1	S_1								
1 1 1 0	S_1								
1 1 1 1	S_1								

State table for multiplier control unit (cont.)

Disadvantages

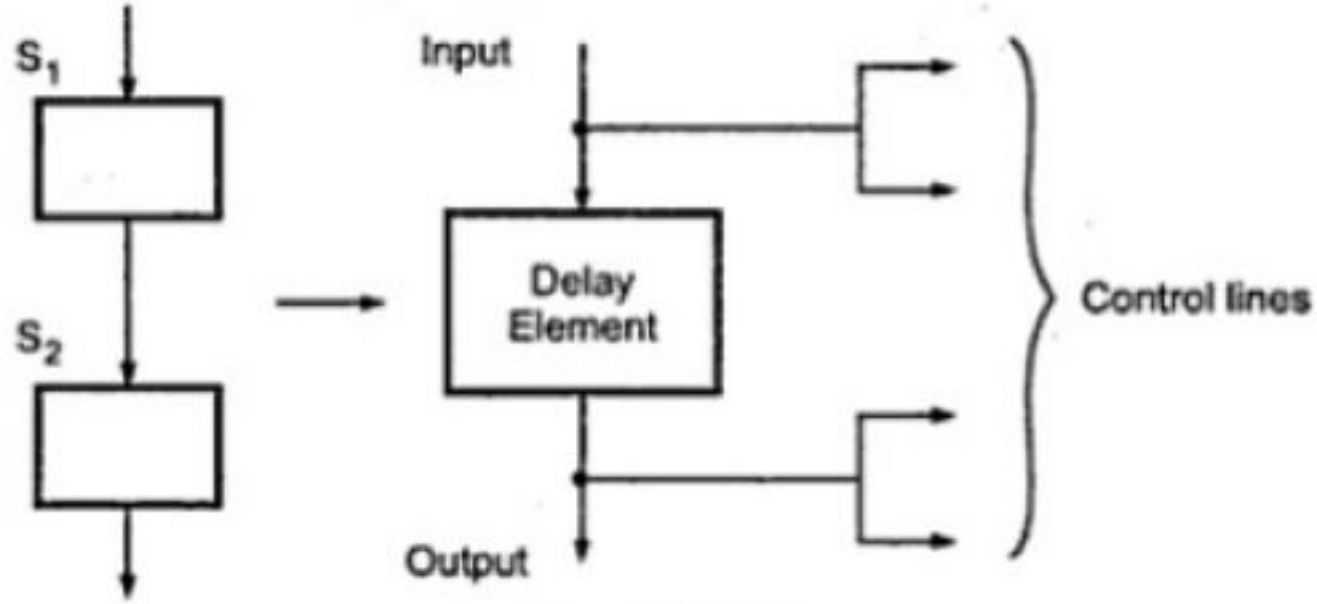
- When multiple instructions or multiple states then this table will have huge size which is not easy to implement.
- It can not deal with looping

Delay element method

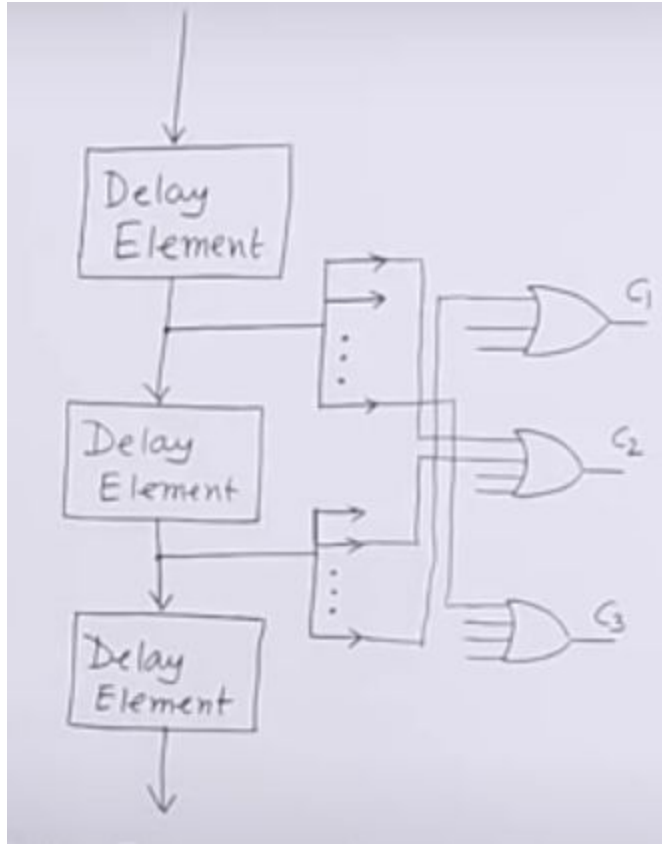
- Control signals or groups of control signals are activated in proper sequence.
- There is a specific time delay between activation of two consecutive control signals or groups of control signal.
- For synchronous operation, delay elements are implemented using D flip-flops and controlled by common clock signal.

Rules for Delay-element Method

1. Each sequence of two successive micro-operations requires a delay element.



- 2.The signals that are intended to activate same control lines are logically ORed to get one common output signals.



3. n lines in the flowchart merge to a common line are transformed into n input OR gate.

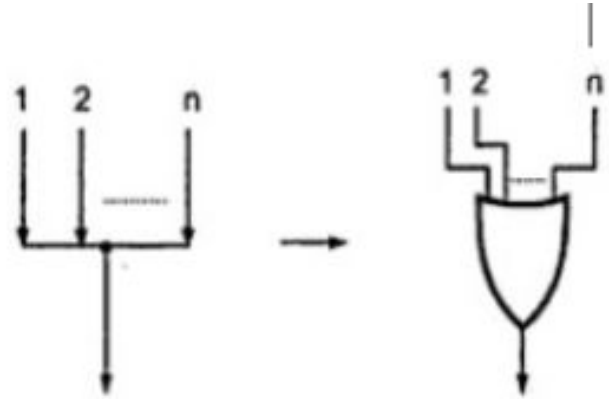


Figure a

4. A decision box can be implemented by two AND gates as shown in figure b

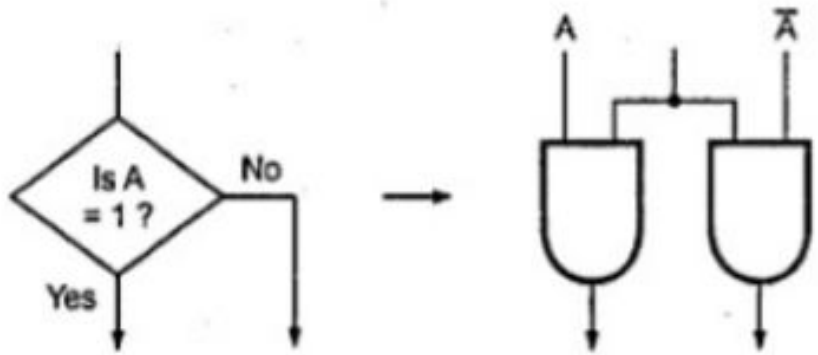
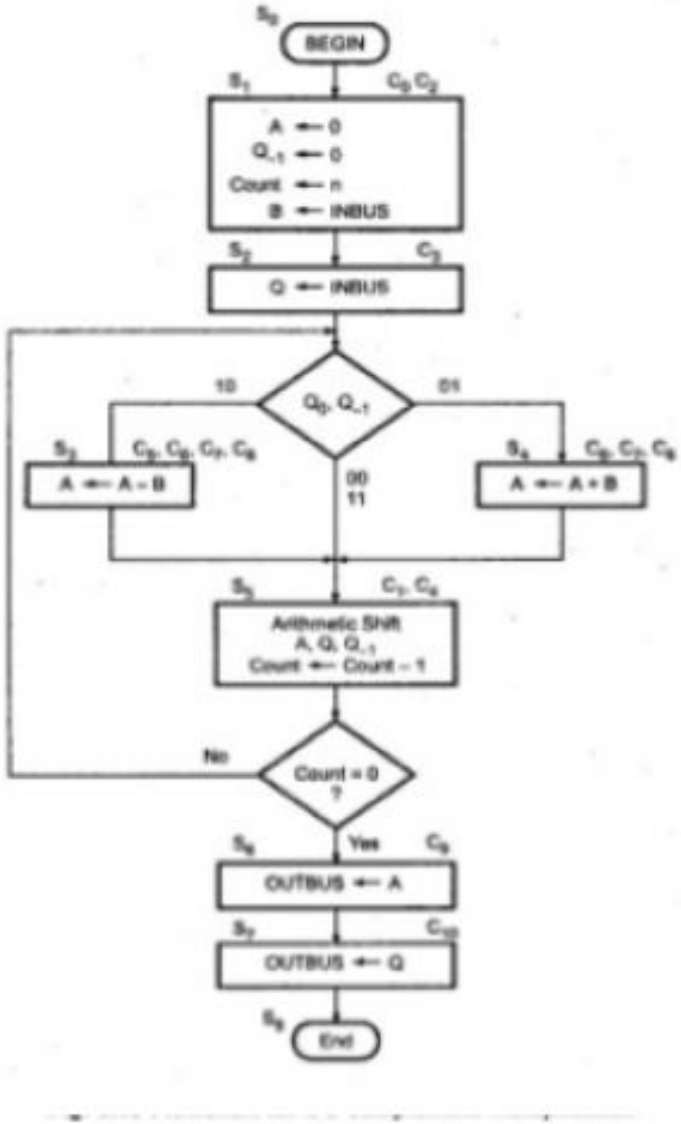
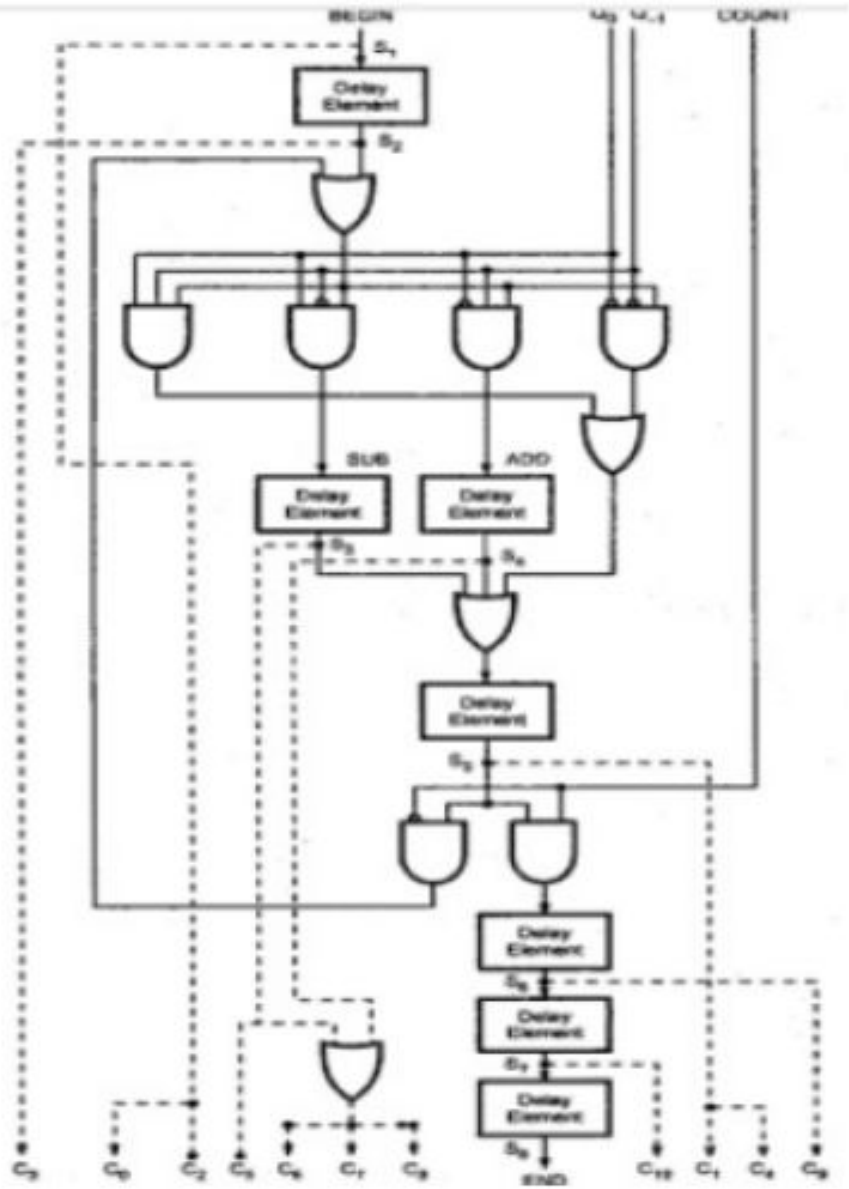
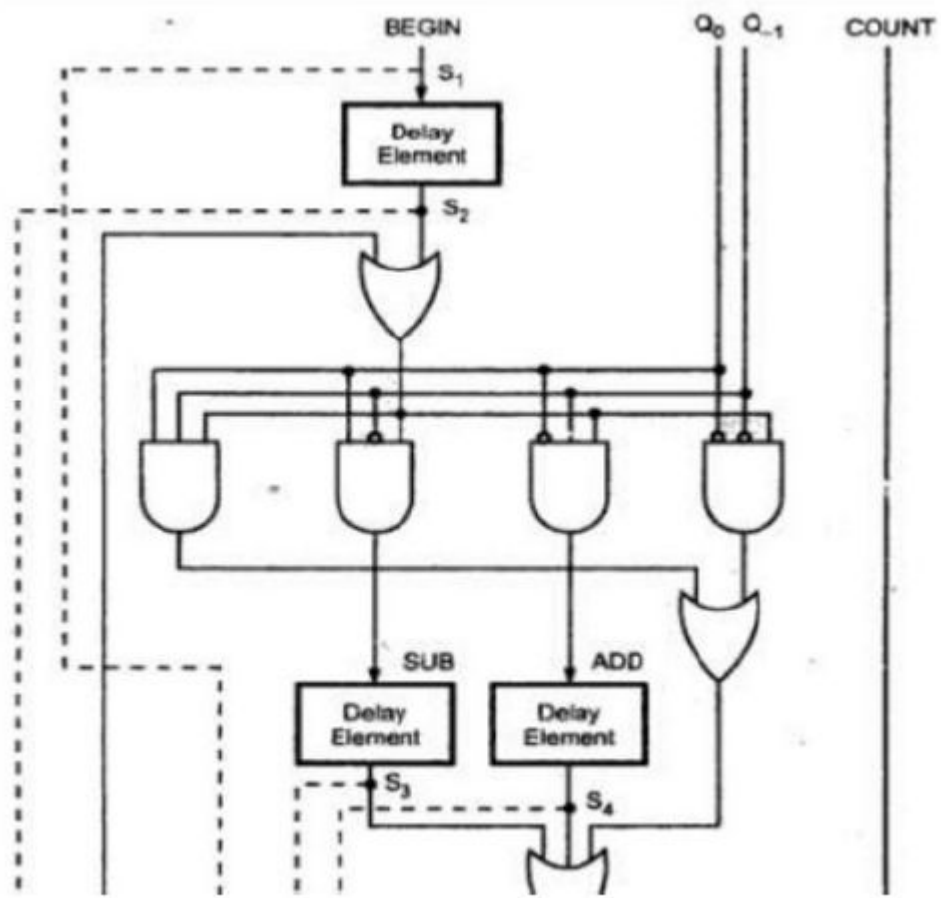


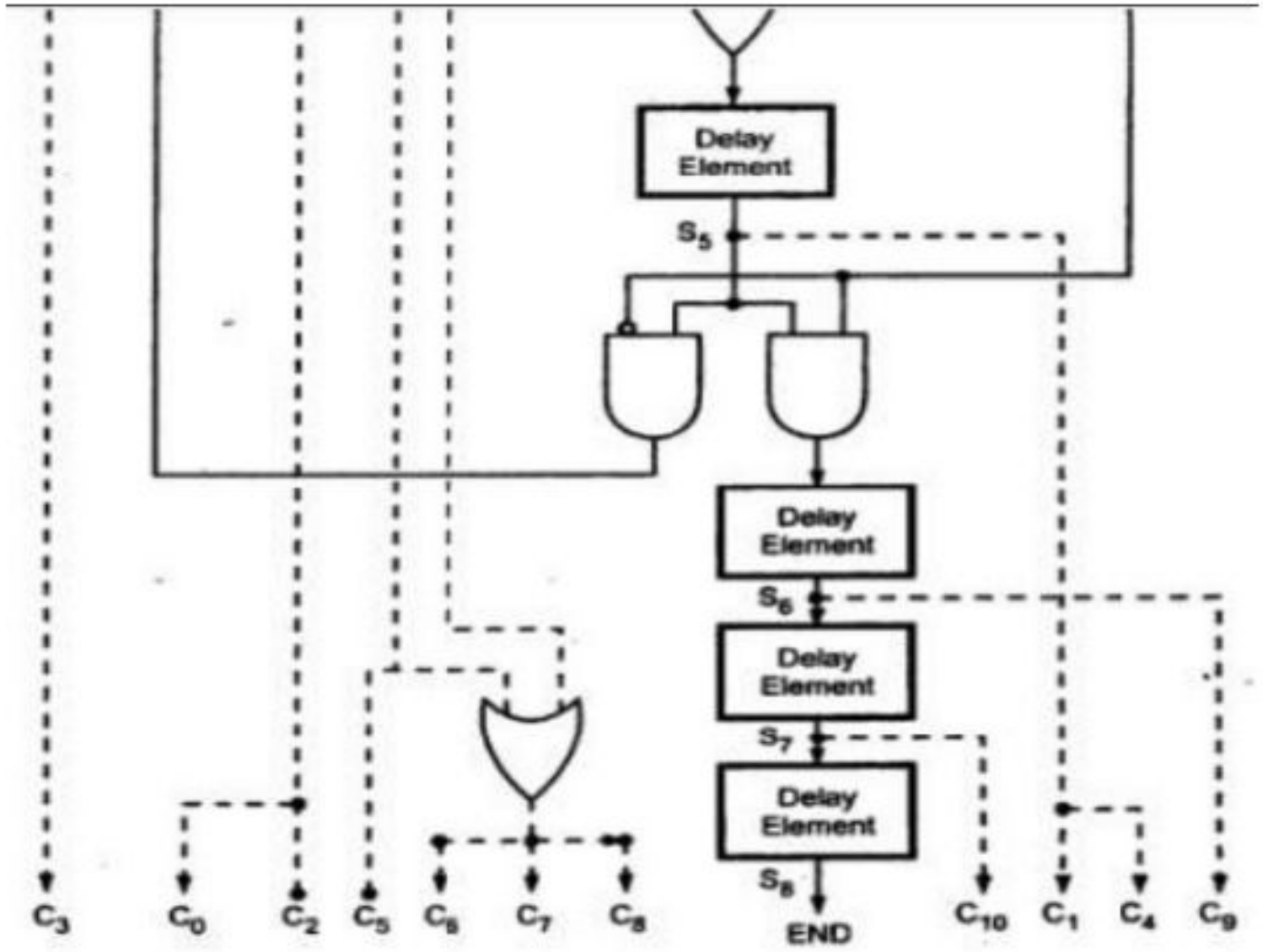
Figure b


BEGIN
A ← 0
Q-1 ← 0
Count ← n
B ← INBUS
Q ← INBUS
Q0, Q-1
10
01
00
11
A ← A - B
A ← A + B
Arithmetic SHR
A, Q, Q-1
Count ← Count - 1
Count = 0 ?
No
Yes
OUTBUS ← A
OUTBUS ← Q
End


BEGIN
COUNT
Delay Element
SUB
ADD
END

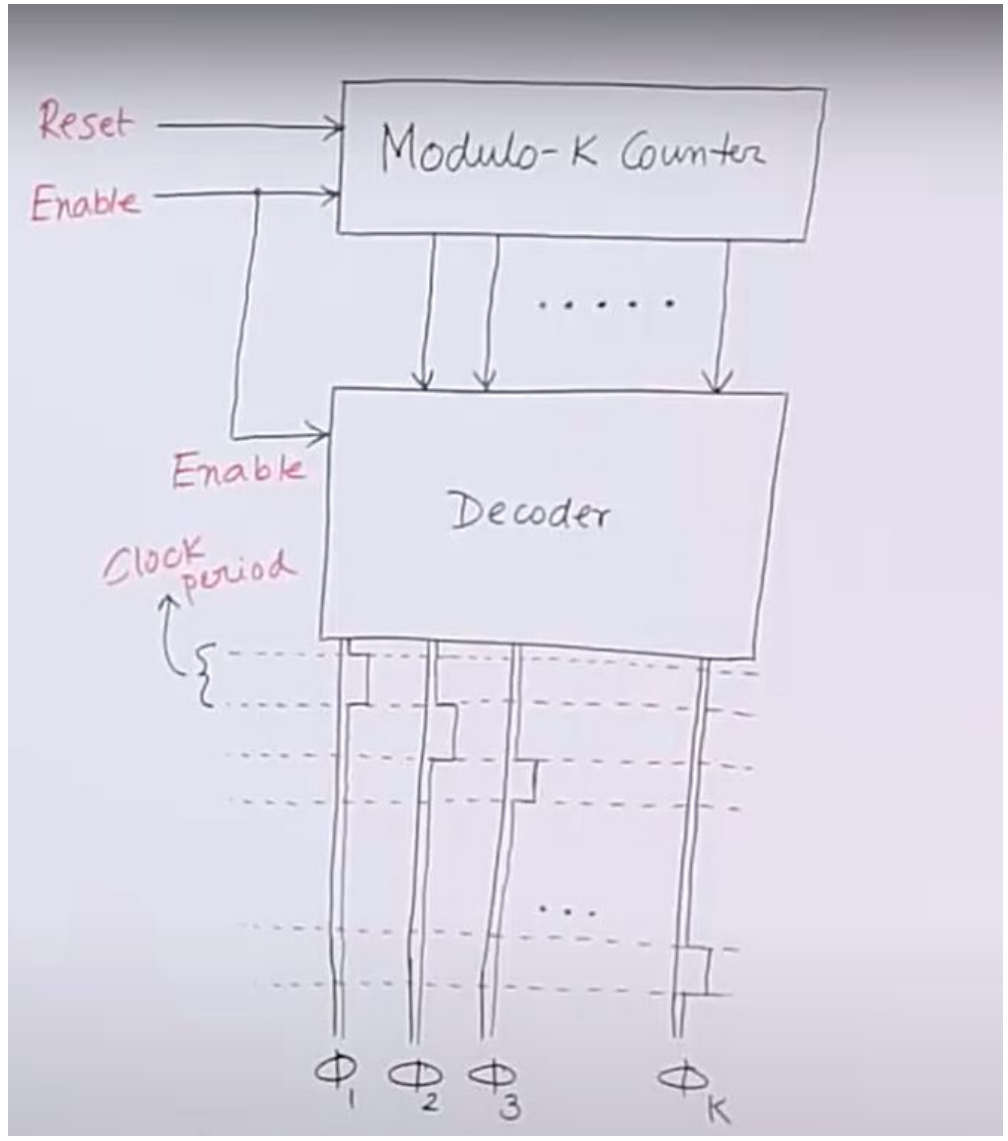


Multiplier control unit using delay elements

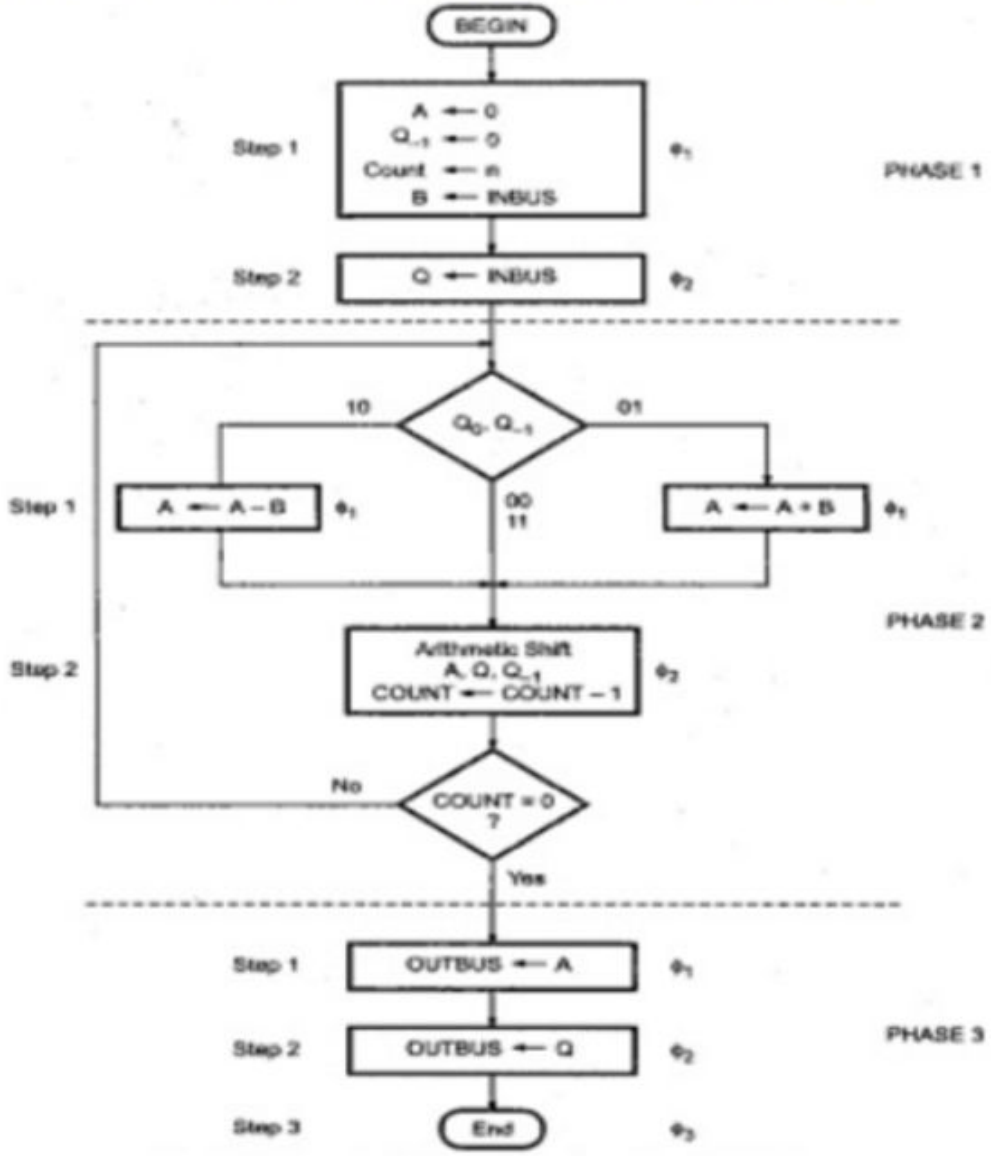


Multiplier control unit using delay elements (cont.)

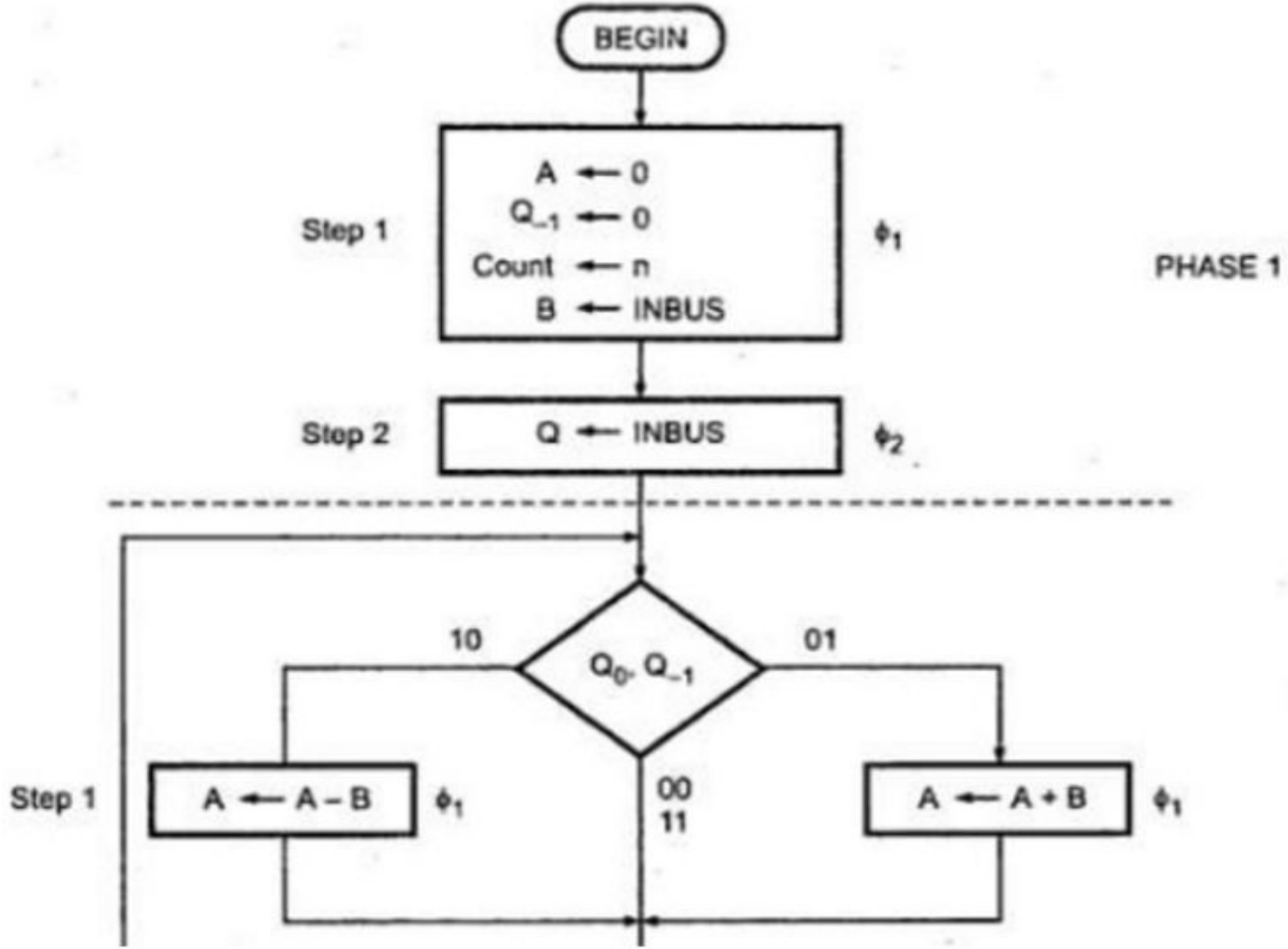
Sequence-counter Method



Sequence-counter Method

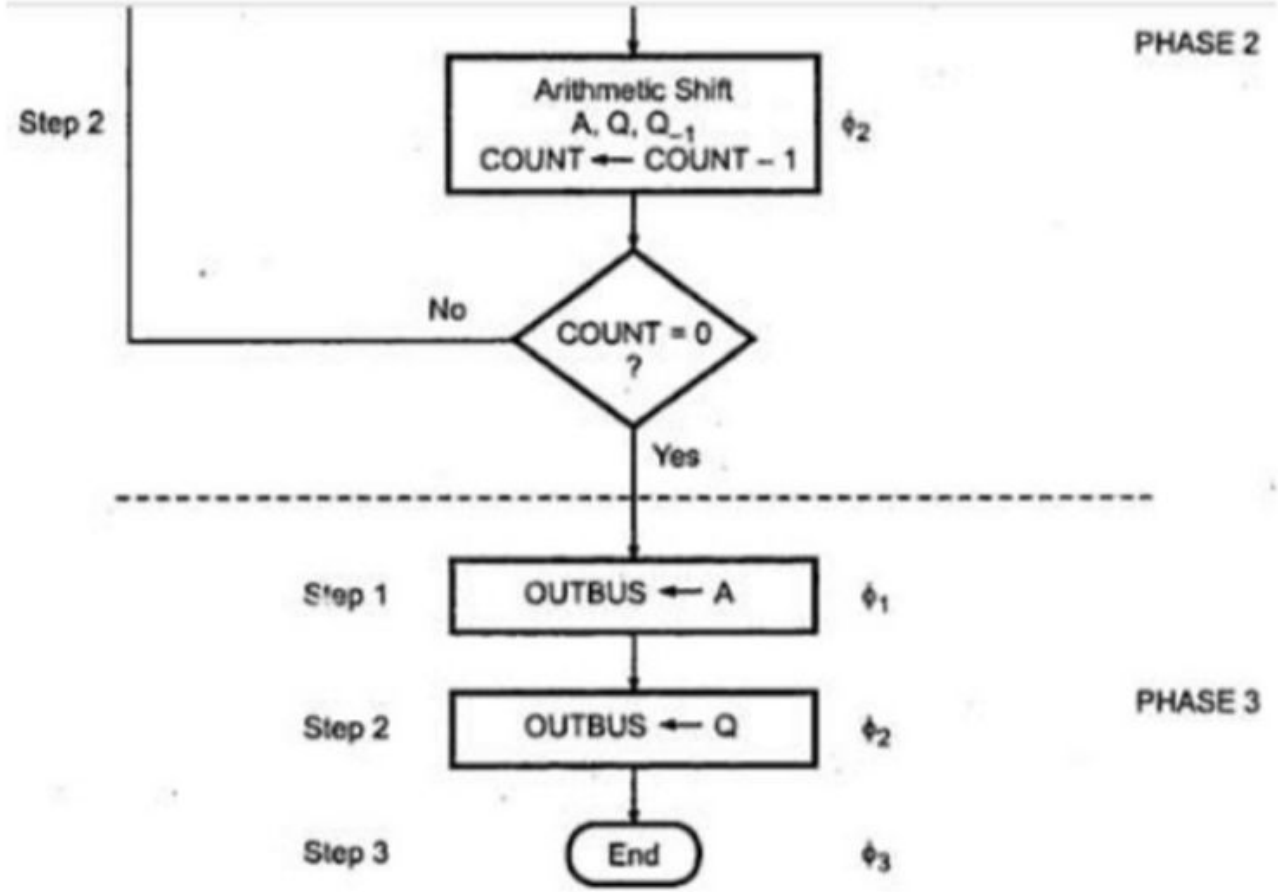


Flowchart for 2's complement multiplication



Flowchart for 2's complement multiplication (cont.)

Sequence-counter Method



Flowchart for 2's complement multiplication (cont.)

Sequence-counter Method

The phase 1 does initial settings. It does this in two steps :

Step 1 ($A \leftarrow 0$, $Q_{-1} \leftarrow 0$, COUNT $\leftarrow$ n and B $\leftarrow$ INBUS) and step 2 ($Q \leftarrow$ INBUS). Phase 2 is repeated n times and it also has two steps :

Step 1 ($A \leftarrow A - B$ or $A \leftarrow A + B$) and

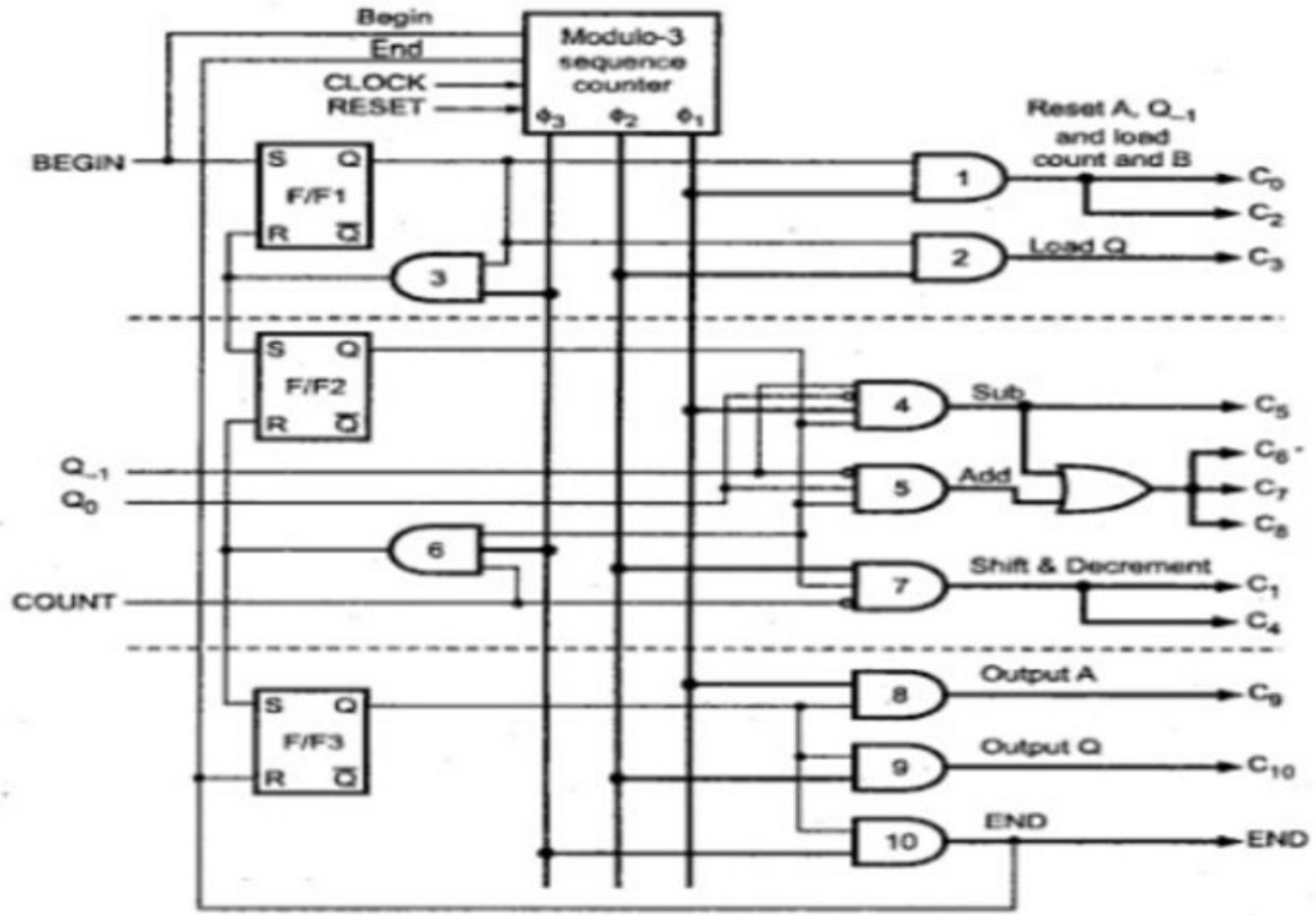
Step 2 (Arithmetic shift A, Q, Q_{-1} and COUNT $\leftarrow$ COUNT – 1).

Phase 3 involves three steps : step 1

Step 1 (OUTBUS $\leftarrow$ A), step 2 (OUTBUS $\leftarrow$ Q) and step 3 (Activate END).

Sequence-counter Method

In each step of particular phase, the corresponding phase signal is used to activate particular set of control lines, as shown in the Fig.



Multiplier control unit using Sequence counter method

Sequence-counter Method

Begin signal sets the output of F/F_1 to logic 1, activating the phase 1. In phase 1 counter goes through steps 0 to 2 and activates ϕ_1, ϕ_2 and ϕ_3 signals in sequence. This action activates control signals as shown in Table 3.3.

Phase 1 Phase signal	Activated control lines	Enabled AND gates
	C_0 and C_2	1
	C_3	2
	Resets F/F_1 and sets F/F_2	3

The phase signal resets F/F_1 and sets F/F_2 activating phase 2. In phase 2, phase signals to are activated again to activate control signals as shown in Table 3.4.

Phase 2 Phase signal	Activated control lines	Enabled AND gates
	In case of subtraction : C_5, C_6, C_7 and C_8	4
	In case of addition : C_6, C_7 and C_8	5
	C_1 and C_4	7
	Resets F/F_2 and sets F/F_3, if COUNT = 1	6

Sequence-counter Method

The phase 2 is repeated until COUNT = 0. When COUNT signal goes high, phase signal resets F/F_2 and sets F/F_3 activating phase 3. In phase 3, phase signals are activated again to activate control signals as shown in the table 3.5.

Phase 3 Phase signal	Activated control lines	Enabled AND gates
	C_9	8
	C_{10}	9
	END	10

Microprogrammed Control Unit

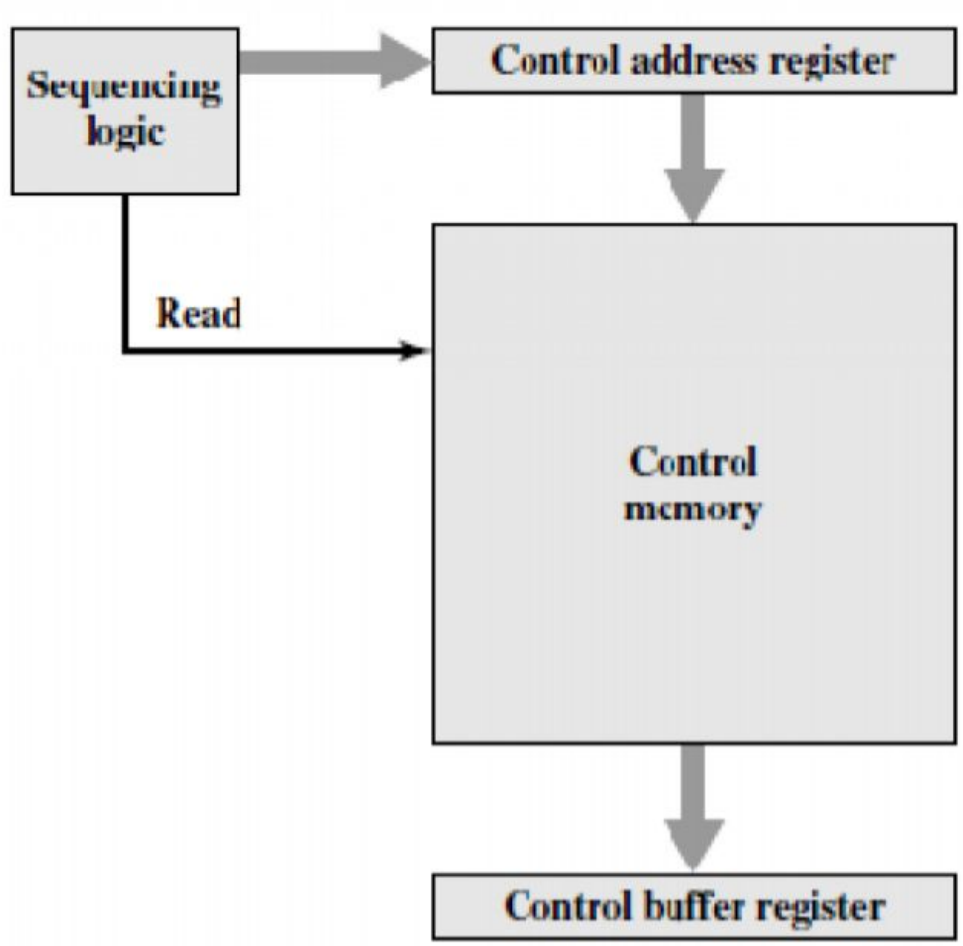
- Control signals are generated by a program similar to machine language programs.
- Sequenceof control steps required to perform ADD(R3),R1operationsare

Micro-instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Microprogrammed Control Unit

- Control Word(CW): A word where individual bits represents the various control signals.
- Microroutine: A sequence of CWs corresponding to the control sequence of a machine instruction.
- Microinstruction: The individual control word in microroutine.
- Control Store: The microroutine for all instructions in instruction set of a computer are stored in a special memory called control store.

Basic Organization of Micro programmed Control UniT



- Key elements of control unit microarchitecture are
- Control memory- stores set of microinstructions.
- Control address register- contains the address of the next microinstruction to be read.
- Control buffer register- contains microinstruction after reading it from control memory.
- reading a microinstruction from the control memory is the same as executing that microinstruction.
- Sequencing unit- loads the control address register and issues a read command

Functioning of Microprogrammed Control Unit

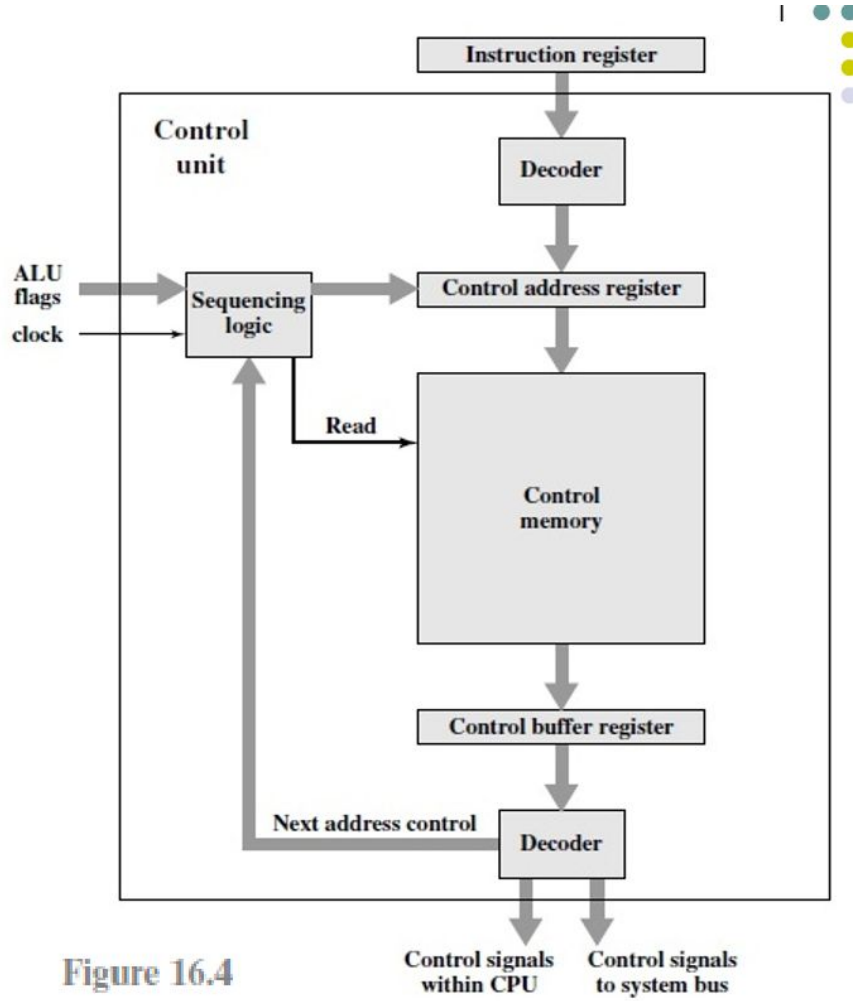


Figure 16.4

- The control unit functions as follows:

1. To execute an instruction, the sequencing logic unit issues a READ command to the control memory.
2. The word whose address is specified in the control address register is read into the control buffer register.
3. The content of the control buffer register generates control signals and next address information for the sequencing logic unit.
4. The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags.

Depending on the value of the ALU flags and the control buffer register, one of three decisions is made:

- Get the next instruction: Add 1 to the control address register.
- Jump to a new routine based on a jump microinstruction: Load the address field of the control buffer register into the control address register.
- Jump to a machine instruction routine: Load the control address register based on the opcode in the IR.

Microprogrammed control unit

- Advantages:

1. It simplifies the design of control unit. Thus it is both, cheaper and less error prone to implement.
2. Control functions are implemented in software rather than hardware.
3. More flexible, can be changed to accommodate new system specification or to correct design error.
4. Debugging and maintenance of a microprogrammed CPU is easy.

Disadvantages:

1. Slower than the hardwired control unit, because time is required to access the microinstruction from control store.
2. The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.
3. The design duration of microprogrammed control unit is more than hardwired control unit for smaller CPU.

Comparison between Hardwired and Microprogrammed Control Unit

Sr. No	Attribute	Hardwired	Microprogrammed
1	Speed	Fast	Slow
2	Cost of implementation	More	Cheaper
3	Implementation approach	Sequential circuit	Programming
4	Flexibility	Not flexible	Flexible
5	Ability to handle complex instruction	Difficult	Easier
6	Design process	Complicated	Systematic
7	Decoding and sequencing logic	Complex	Easy
8	Application	RISC µp	CISC µp
9	Control memory	Absent	Present
10	Chip area	Less	more

MICROPROGRAMMED COMPUTER ORGANIZATION

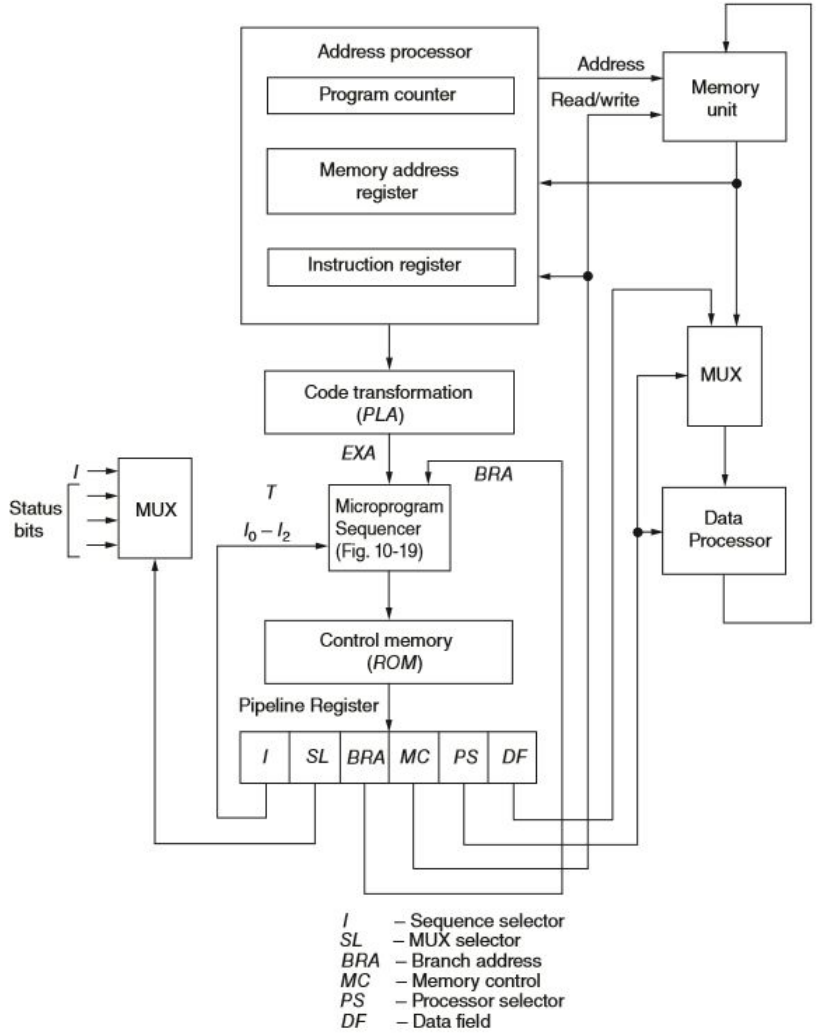


Figure 10-21 Microprogrammed computer organization

- A digital computer consists of a central processor unit (CPU), a memory unit, and input-output devices.
- It consists of a memory unit, two processor units, a microprogram sequencer, a control memory, and few other digital functions.
- The memory unit stores the instructions and data supplied by the user through an input device.
- The data processor manipulates the data, and the address processor manipulates the address information received from memory

- An instruction extracted from memory during the fetch cycle goes into the instruction register. The instruction-code bits in the instruction register specify a macrooperation for control memory.
- A code transformation is sometimes needed to convert the operation-code bits of an instruction into a starting address for the control memory. This code transformation constitutes a mapping function and can be implemented with a ROM or a PLA.

- The mapping concept provides a flexibility for adding instructions or macrooperations for control memory as the need arises. The address generated in the code transformation mapping function is applied to the external address (EXA) input of the sequencer.
- Control unit consists of control memory for storing the microinstructions, a multiplexer, and a pipeline register.

- The memory control (MC) field controls the address processor and the read and write operations in the memory unit.
- The processor select (PS) field controls the operations in the data processor unit.
- The last field is a data field (DF) used to introduce constants into the processor.

Input output subsystem

- The input-output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment.
- Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user.
- Input or output devices attached to the computer are also called peripherals.
- Among the most common peripherals are keyboards, display units, and printers.

- Video monitors are the most commonly used peripherals. They consist of a keyboard as the input device and a display unit as the output device.
- There are different types of video monitors, but the most popular use a cathode ray tube (CRT).
- Printers provide a permanent record on paper of computer output data or text. There are three basic types of character printers: daisywheel, dot matrix, and laser printers.

Magnetic tapes are used mostly for storing files of data: for example, a company's payroll record.

It is one of the cheapest and slowest methods for storage and has the advantage that tapes can be removed when not in use. Magnetic disks have high-speed rotational surfaces coated with magnetic material.

Input-Output Interface

Input-output interface provides a method for transferring information between internal storage and external I/0 devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.

The major differences are or characteristics of I/O

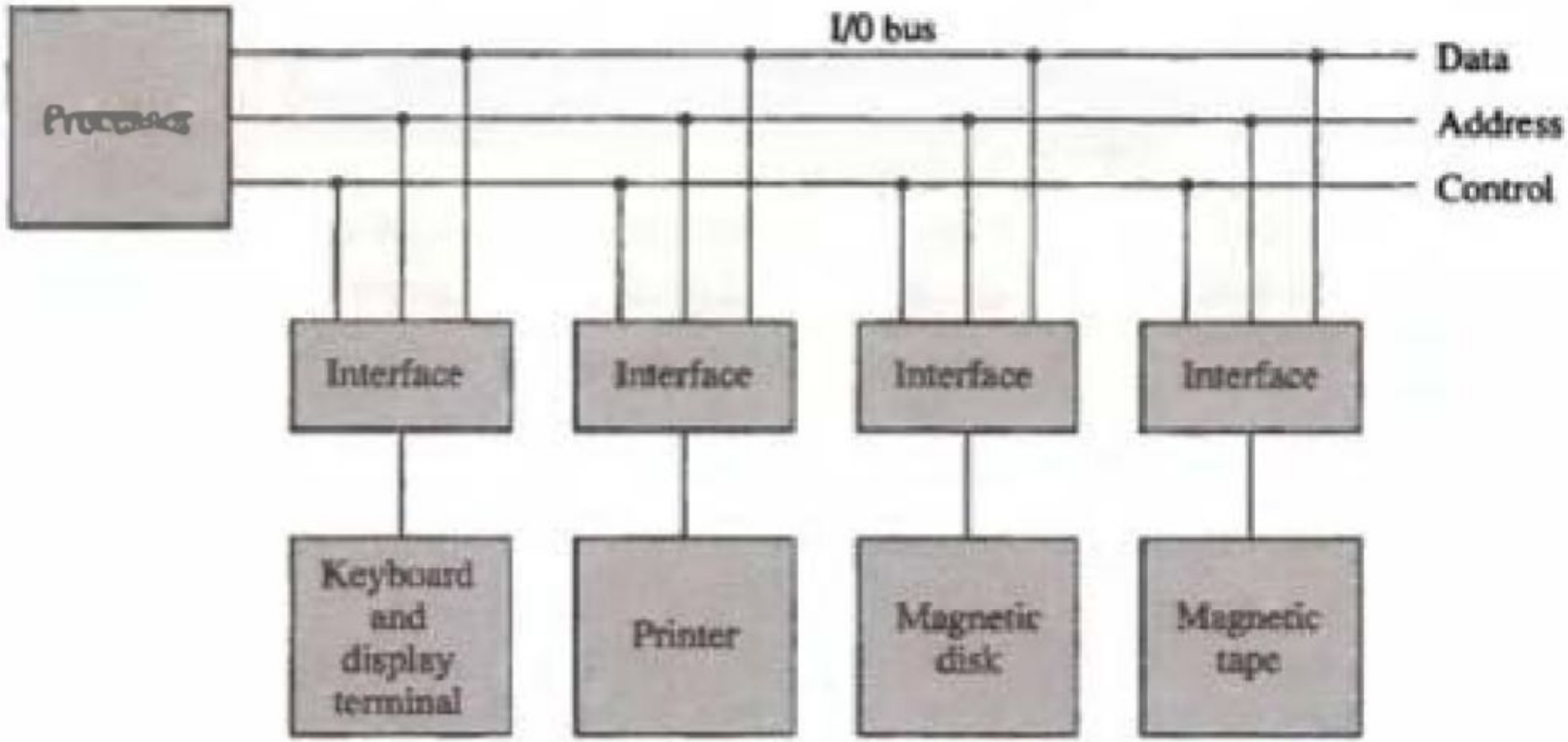
1.Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.

2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.

3. Data codes and formats in peripherals differ from the word format in the CPU and memory.

4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

connection of I/O bus to input output device



- The I/O bus consists of data lines, address lines, and control lines.
- The I/O bus from the procesrs is attached to all peripheral interfaces.
- To communicate with a particular device, the procesrs places a device address on the address lines.
- Each interface attached to the I/O bus contains an address decoder that monitors the address lines.
-

When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.

Type of commands

There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.

1.control command

A control command is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.

2.Status command

A status command is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

3.output data

- A data output command causes the interface to respond by transferring data from the bus into one of its registers.
- example: tape
- The processor then monitors the status of the tape by means of a status command. When the tape is in the correct position, the processor issues a data output command. The interface responds to the address and command and transfers the information from the data lines in the bus to its bufferregister.The interface then communicates with the tape controller and sends the data to be stored on tape.

4.**Input data**

- The data input command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register.
- The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.
-

I/O versus Memory Bus

There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.

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

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

Isolated versus Memory-Mapped I/O

- Many computers use one common bus to transfer information between memory or I/O and the CPU.
- The distinction between a memory transfer and I/O transfer is made through separate read and write lines.
- The I/O read and I/O write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer.

Isolated I/O

- In the isolated I/O configuration, the CPU has distinct input and output instructions, and each of these instructions is associated with the address of an interface register.
- When the CPU fetches and decodes the operation code of an input or output instruction, it places the address associated with the instruction into the common address lines. At the same time, it enables the I/O read (for input) or I/O write (for output) control line. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word.

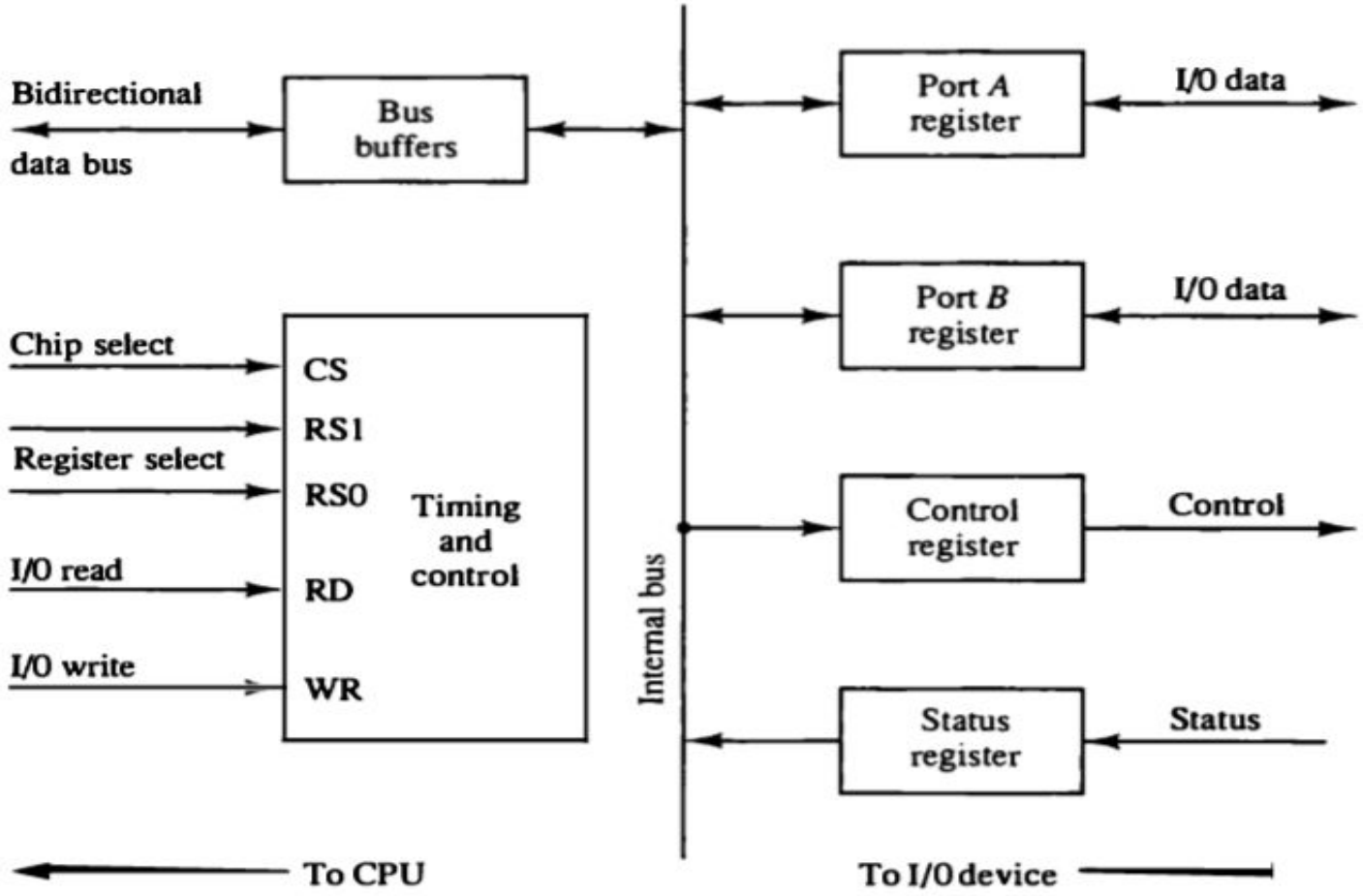
- On the other hand, when the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line. This informs the external components that the address is for a memory word and not for an I/O interface.
- The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space.

Memory-mapped

- The other alternative is to use the same address space for both memory and 1/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and 1/O addresses. This configuration is referred to as memory-mapped 1/O.
- In a memory-mapped I/0 organization there are no specific input or output instructions.

- Computers with memory-mapped I/0 can use memory-type instructions to access I/0 data.
- It allows the computer to use the same instructions for either input-output transfers or for memory transfers.
- The advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/0 registers.

Example of I/O Interface



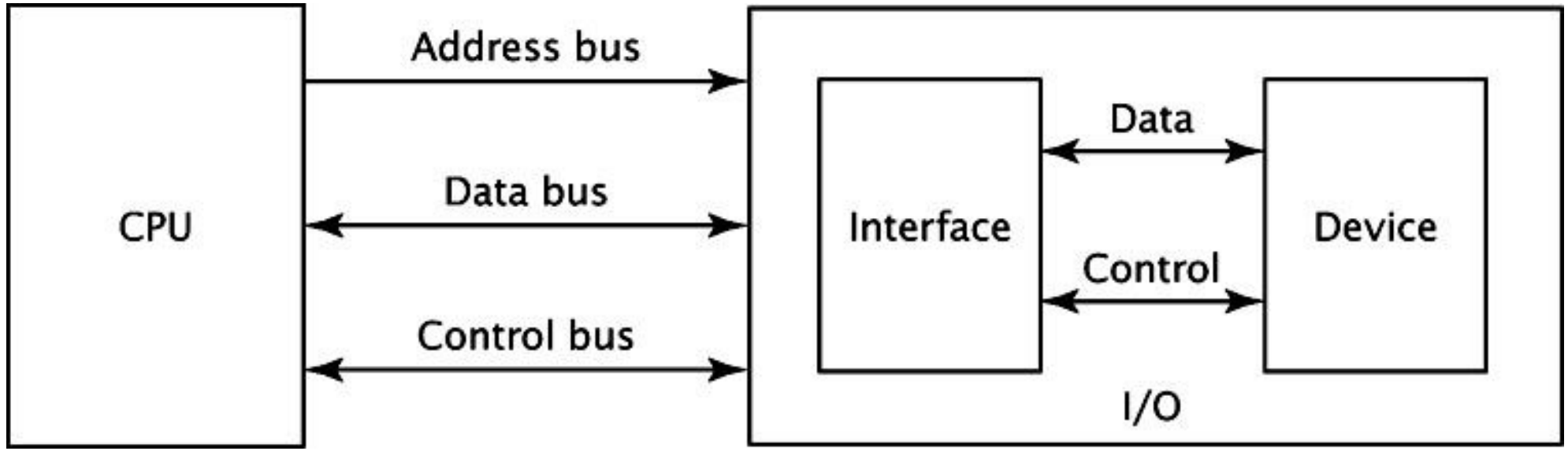
- It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits.
- The chip select and register select inputs determine the address assigned to the interface.
- The I/0 read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/0 device attached to the interface.

CS	RS1	RS0	Register selected
0	×	×	None: data bus in high–impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

Asynchronous Data Transfer

- In a computer system, CPU and an I/O interface are designed independently of each other.
- When internal timing in each unit is independent from the other and when registers in interface and registers of CPU uses its own private clock.
- In that case the two units are said to be asynchronous to each other. CPU and I/O device must coordinate for data transfers.

- The most significant aspect of asynchronous communications is that data is not transmitted at regular intervals, thus making possible variable bit rate.
- And that the transmitter and receiver clock generators do not have to be exactly synchronized all the time.
- Hence we need a start and stop signal to send and receive data.


CPU
Address bus
Data bus
Control bus
Interface
Data
Control
Device
I/O

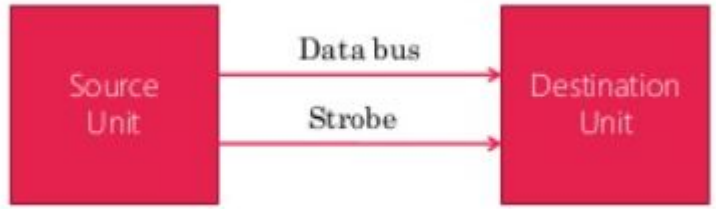
METHODS USED IN ASYNCHRONOUS DATA TRANSFER

- Strobe Control: This is one way of transfer i.e. by means of strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- Handshaking: This method is used to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data.

STROBE CONTROL

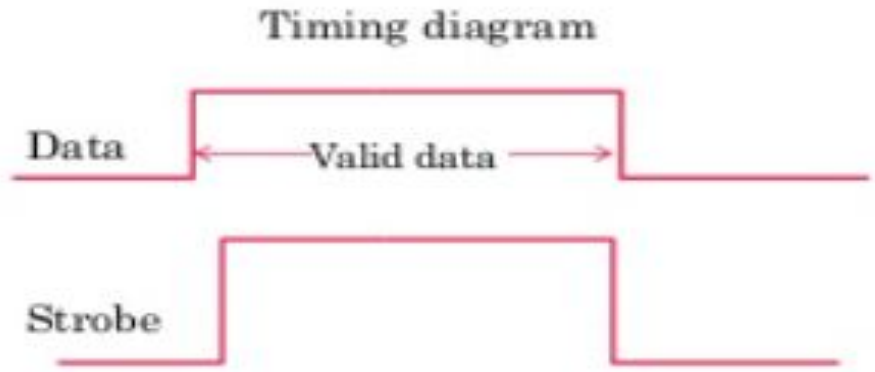
Strobe control method of data transfer uses a single control signal for each transfer. The strobe may be activated by either the source unit or the destination unit.

1. Source Initiated Strobe
2. Destination Initiated Strobe



SOURCE INITIATED STROBE

- The data bus carries the binary information from source unit to the destination unit as shown below.
- The strobe is a single line that informs the destination unit when a valid data word is available in the bus.



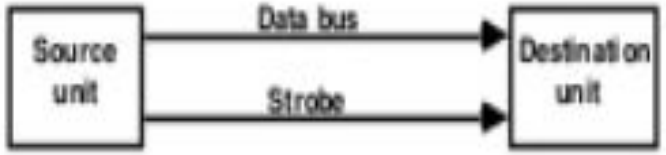
- The source unit first places the data on the bus. After a brief delay to ensure that the data settle to a steady value, the source activities the strobe pulse.
- The information of the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- The source removes the data from the bus for a brief period of time after it disables its strobe pulse.

DESTINATION INITIATED STROB

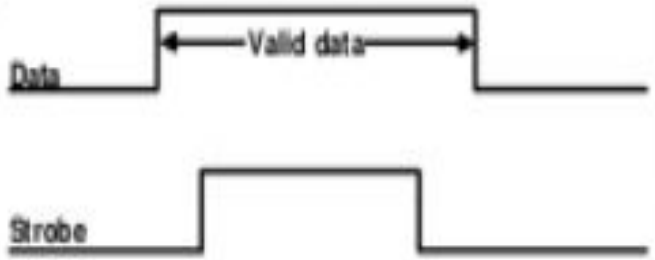
- First, the destination unit activates the strobe pulse, informing the source to provide the data.
- The source unit responds by placing the requested binary information on the unit to accept it.
- The data must be valid and remain in the bus long enough for the destination unit to accept it.
- The falling edge of the strobe pulse can be used again to trigger a destination register.
- The destination unit then disables the strobe.
- The source removes the data from the bus after a predetermined time interval.

Source-Initiated Strobe for Data Transfer

Block Diagram

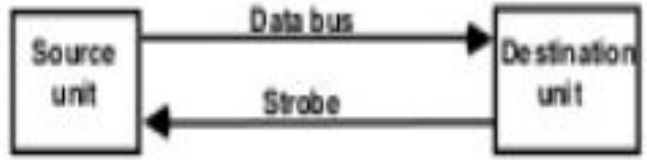


Timing Diagram

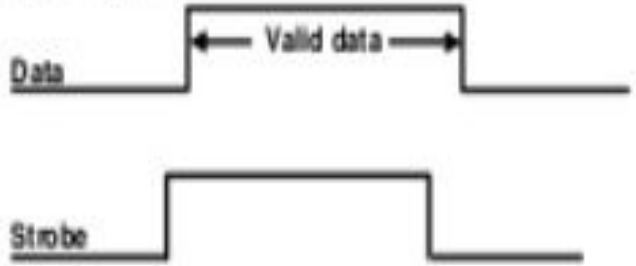


Destination-Initiated Strobe for Data Transfer

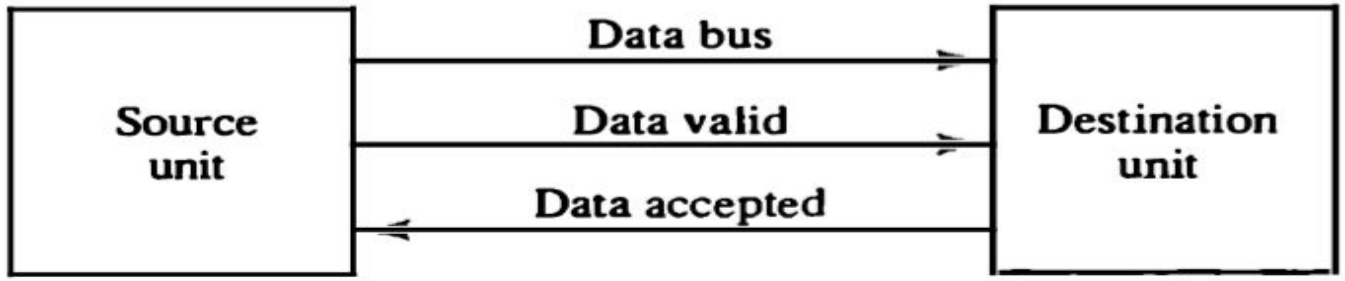
Block Diagram



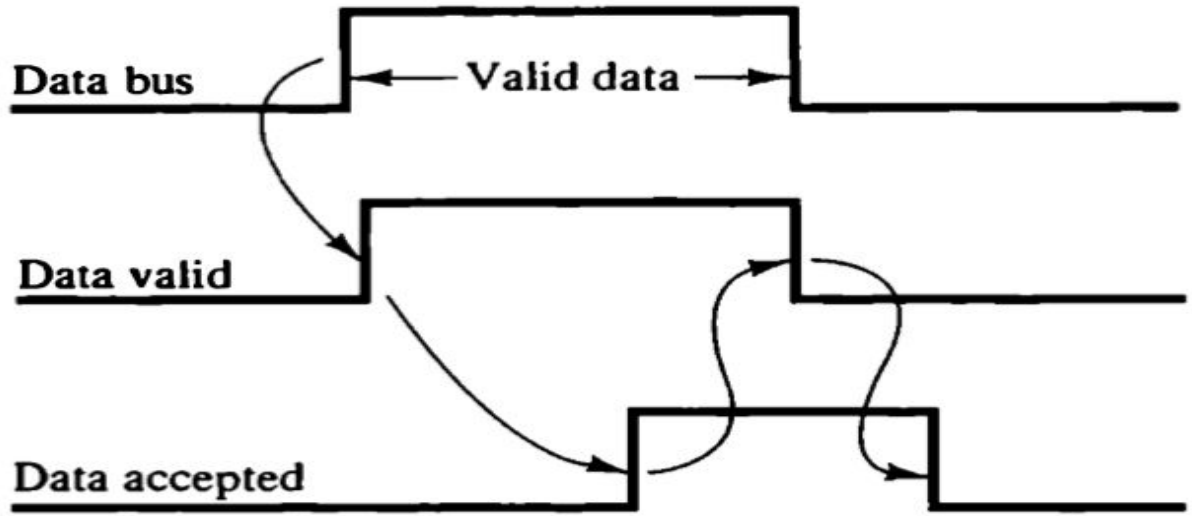
Timing Diagram



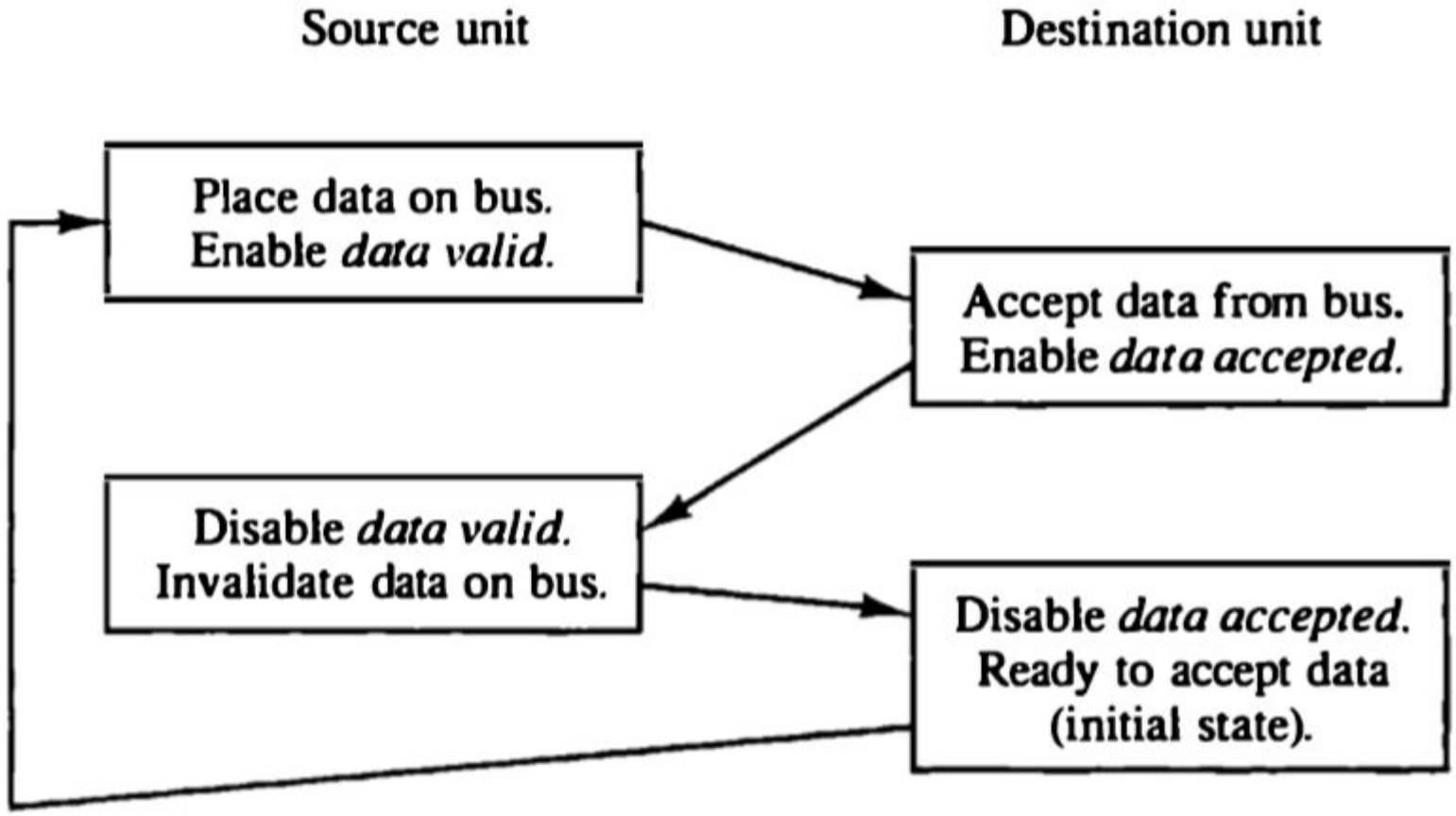
Source initiated transfer using handshaking



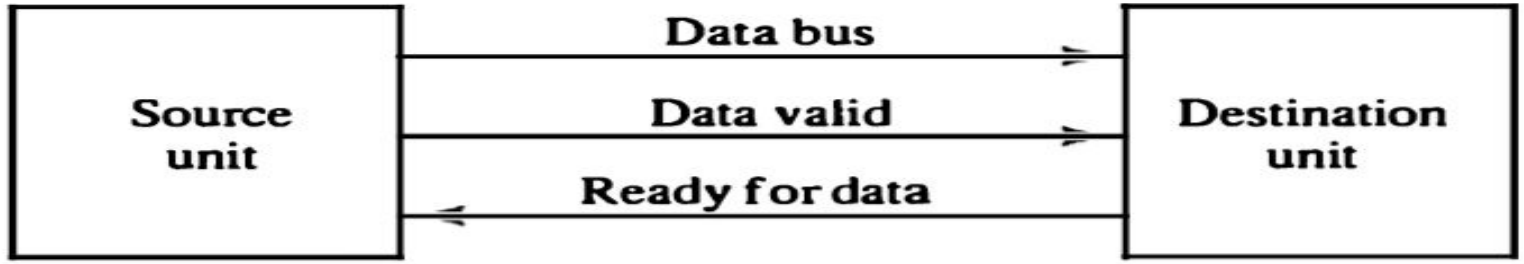
(a) Block diagram



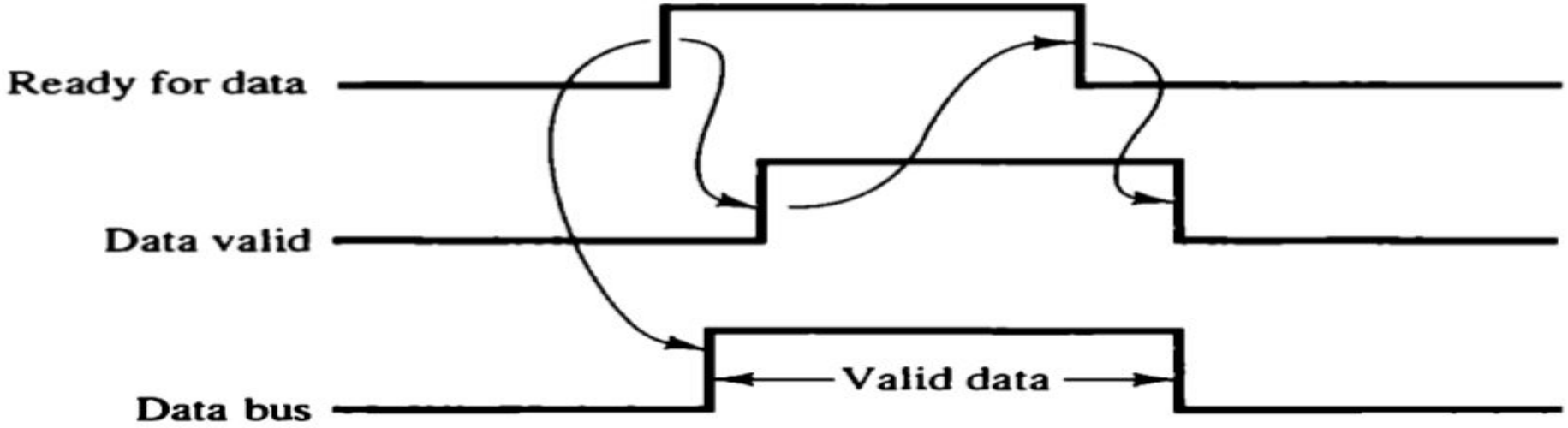
(b) Timing diagram


Source unit
Destination unit
Place data on bus.
Enable data valid.
Accept data from bus.
Enable data accepted.
Disable data valid.
Invalidate data on bus.
Disable data accepted.
Ready to accept data
(initial state).

Destination initiated transfer using handshaking

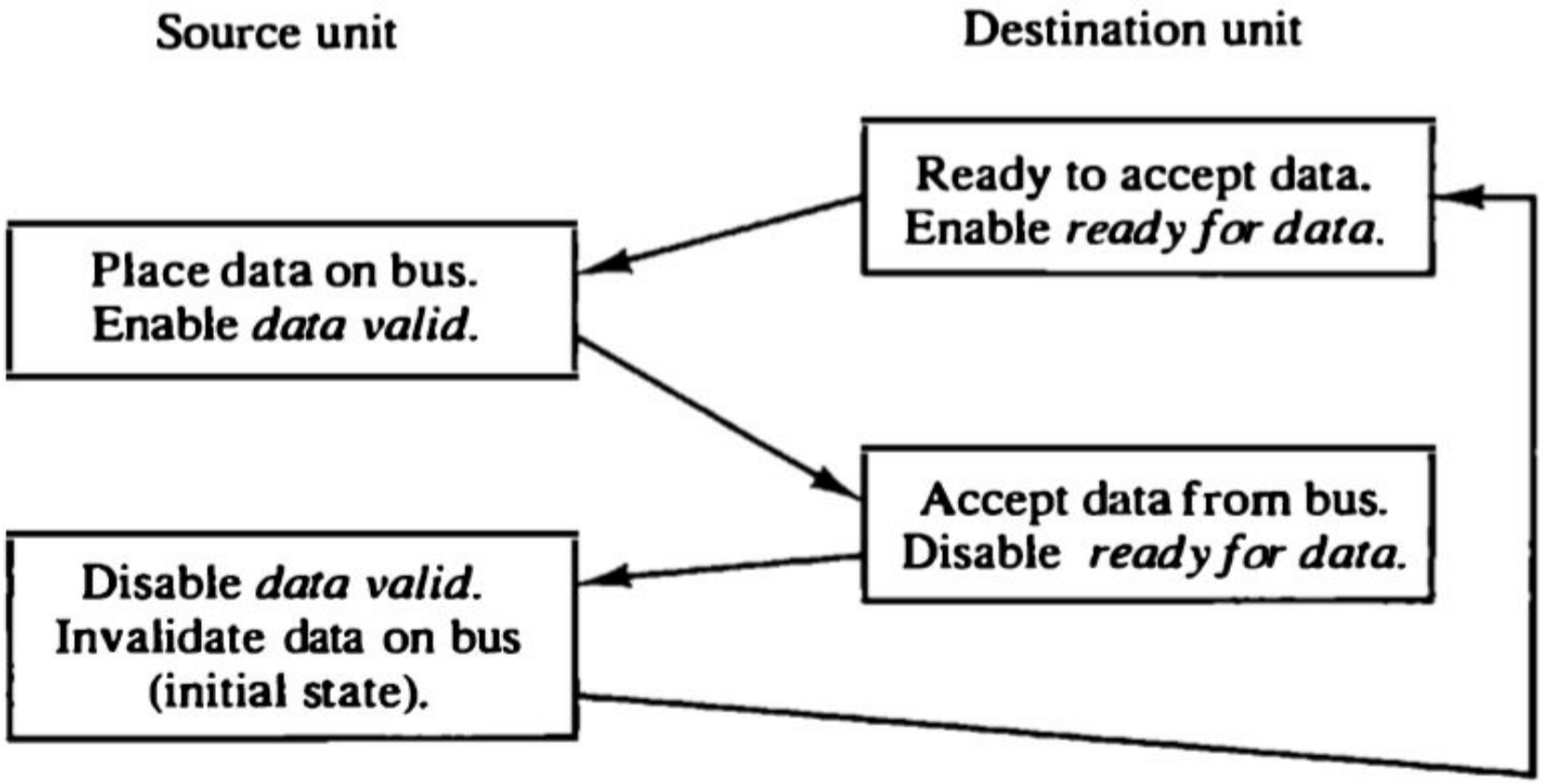


(a) Block diagram



(b) Timing diagram

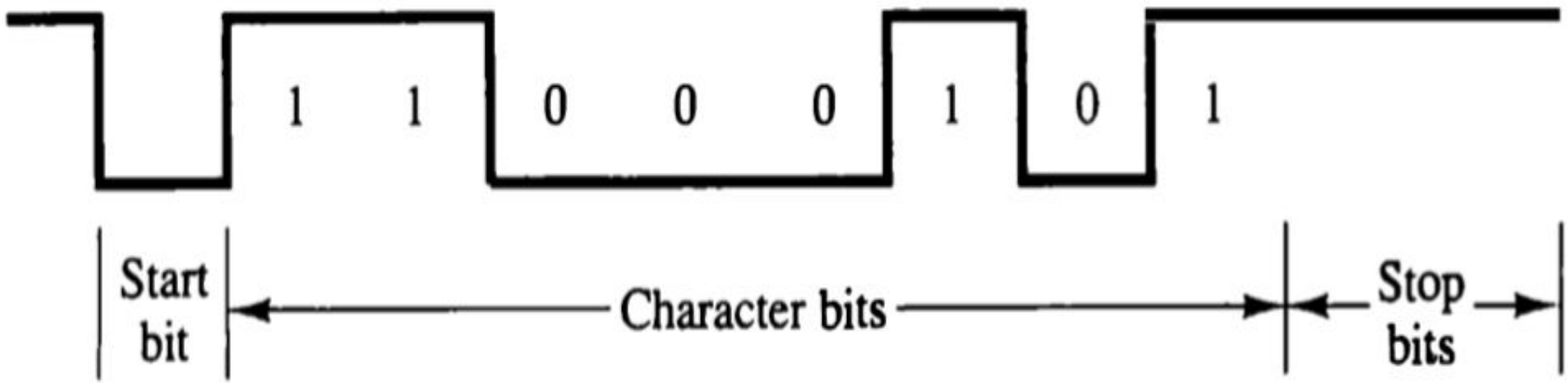
Sequence of events



(c) Sequence of events

Asynchronous Serial Transfer

Each character consists of three parts: a start bit, the character bits, and stop bits.



A transmitted character can be detected by the receiver from knowledge of the transmission rules:

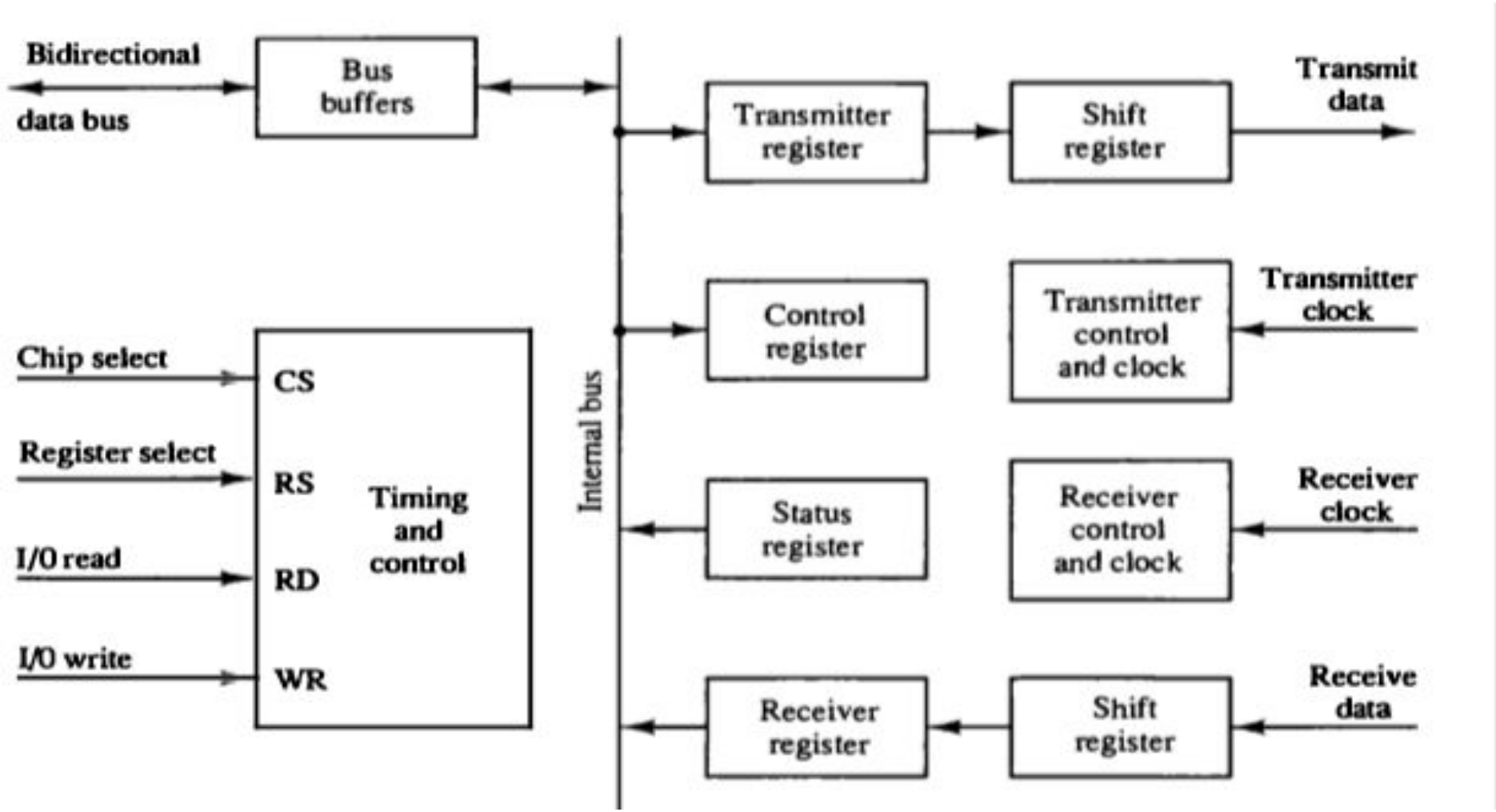
1. When a character is not being sent, the line is kept in the 1-state.

2. The initiation of a character transmission is detected from the start bit, which is always 0.

3. The character bits always follow the start bit.

4. After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time.

Asynchronous Communication Interface



CS	RS	Operation	Register selected
0	×	×	None: data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Transmitter

The CPU reads the status register and checks the flag to see if the transmitter register is empty.

If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.

The first bit in the transmitter shift register is set to 0 to generate a start bit. The character is transferred in parallel from the transmitter register to the shift register and the appropriate number of stop bits are appended into the shift register. The transmitter register is then marked empty.

The character can now be transmitted one bit at a time by shifting the data in the shift register at the specified baud rate.

Receiver

The receive data input is in the 1-state when the line is idle. The receiver control monitors the receive-data line for a 0 signal to detect the occurrence of a start bit.

Once a start bit has been detected, the incoming bits of the character are shifted into the shift register at the prescribed baud rate.

After receiving the data bits, the interface checks for the parity and stop bits.

The character without the start and stop bits is then transferred in parallel from the shift register to the receiver register.

The flag in the status register is set to indicate that the receiver register is full. The CPU reads the status register and checks the flag, and if set, it reads the data from the receiver register.

Errors

The CPU can read the status register at any time to check if any errors have occurred.

Three possible errors that the interface checks during transmission are parity error, framing error, and overrun error.

Modes of Transfer

Data transfer to and from peripherals may be handled in one of three possible modes:

1) Programmed I/O
2) Interrupt-initiated I/O
3) Direct memory access (DMA)

Programmed I/O

Programmed i/o refers to data transfers initiated by a (cpu) under driver software control to access registers or memory on a device.

The cpu issues a command then waits for i/o operations to be complete. As the cpu is faster than the i/o module, the problem with programmed i/o is that the cpu has to wait along time for the i/o module of concern to be ready for either reception or transmission of data . The cpu, while waiting must repeatedly check the status of the i/o module, and this process is known as polling . As a result , the level of the performance of the entire system is severely degraded.

- CPU requests I/O operation.
- I/O module performs operation.
- I/O module sets status bits.
- CPU checks status bits periodically.
- I/O module does not inform CPU directly.
- I/O module does not interrupt CPU.
- CPU may wait or come back later.

Interrupt Driven I/O

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.

Interrupt Driven I/O Basic Operation

- CPU issues read command.
- I/O module gets data from peripheral whilst CPU does other work.
- I/O module interrupts CPU.
- CPU requests data.
- I/O module transfers data.

Types of Interrupts

- Hardware interrupts: if the signal for the processor is from external device or hardware is called hardware interrupts.

- Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are:

- Maskable interrupt: the hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.

- Non maskable interrupt: the hardware which cannot be delayed and should process by the processor immediately.

Software Interrupts: Software interrupt can also divided into two types.

They are:

•Normal Interrupts: the interrupts which are caused by the software instructions are called software instructions.

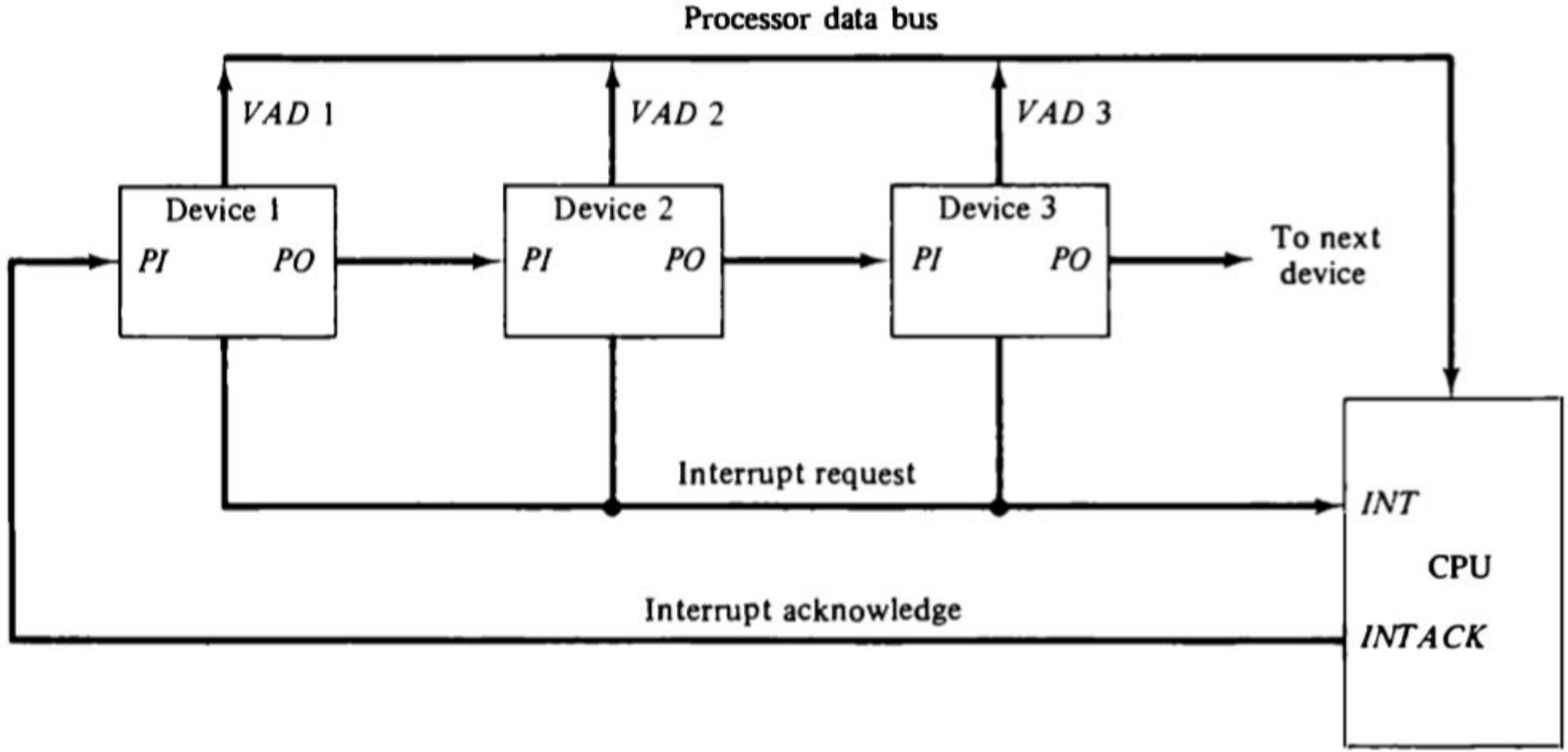
Exception: unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

Design issues

- Two design issues arise in implementing interrupt I/O. First, because there will almost invariably be multiple I/O modules, how does the processor determine which device issued the interrupt?
- And second, if multiple interrupts have occurred, how does the processor decide which one to process?
- Four general categories of techniques are in common use:
  - Multiple interrupt lines
  - Software poll
  - Daisy chain (hardware poll, vectored)
  - Bus arbitration (vectored) Design Issues for Interrupt I/O

- Multiple Interrupt Lines : The most straightforward approach to the problem is to provide multiple interrupt lines between the processor and the I/O modules. However, it is impractical .
- Software polling :
  - When the processor detects an interrupt, it branches to an interrupt-service routine whose job it is to poll each I/O module to determine which module caused the interrupt.
  - The processor then reads the status register of each I/O module to identify the interrupting module.
  - The disadvantage of the software poll is that it is time consuming. A more efficient technique is to use a daisy chain, which provides, in effect, a hardware poll.

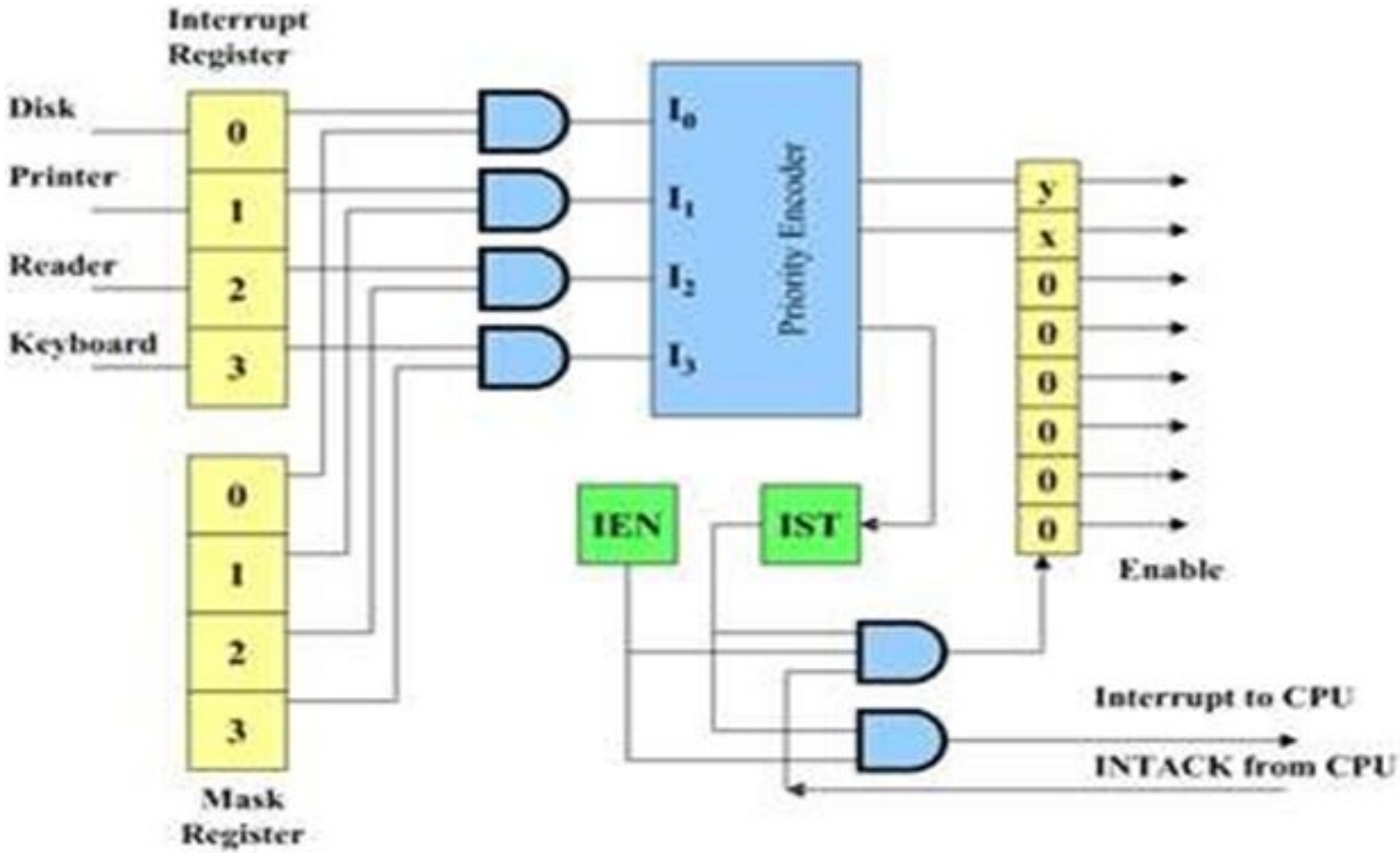
Daisy Chain Priority Interrupt



Daisy Chaining :

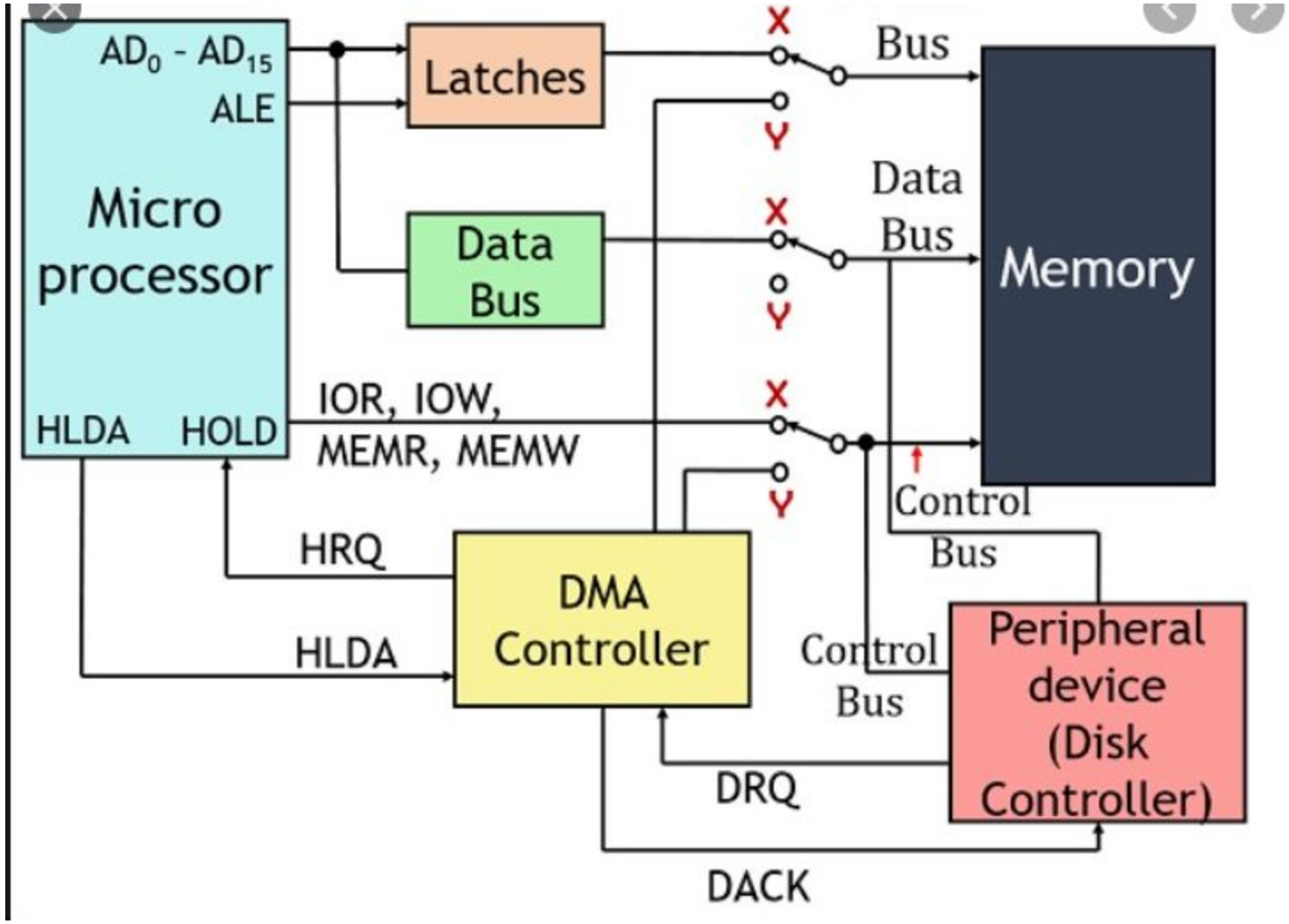
- For interrupts, all I/O modules share a common interrupt request line.
- The interrupt acknowledge line is daisy chained through the modules.
- When the processor senses an interrupt, it sends out an interrupt acknowledge.
- This signal propagates through a series of I/O modules until it gets to a requesting module.
- The requesting module typically responds by placing a word on the data lines. This word is referred to as a vector and is either the address of the I/O module or some other unique identifier.
- In either case, the processor uses the vector as a pointer to the appropriate device-service routine.
- This avoids the need to execute a general interrupt-service routine first. This technique is called a vectored interrupt.

priority Interrupt



Direct Memory Access (DMA)

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred . Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.


AD0 - AD15
ALE
Micro processor
HLDA
HOLD
Latches
Data Bus
IOR, IOW,
MEMR, MEMW
HRQ
HLDA
DMA Controller
DRQ
DACK
X
Y
Bus
Data Bus
Memory
Control Bus
Control Bus
Peripheral device (Disk Controller)

DMA transfer types

The DMA controller functions as a bus master and bus slave. It performs data transfer operations. DMA controlled input/output is further divided into the following categories

1. Burst or Block Transfer DMA
2. Cycle Steal or Single Byte Transfer DMA
3. Transparent or Hidden DMA Transfer

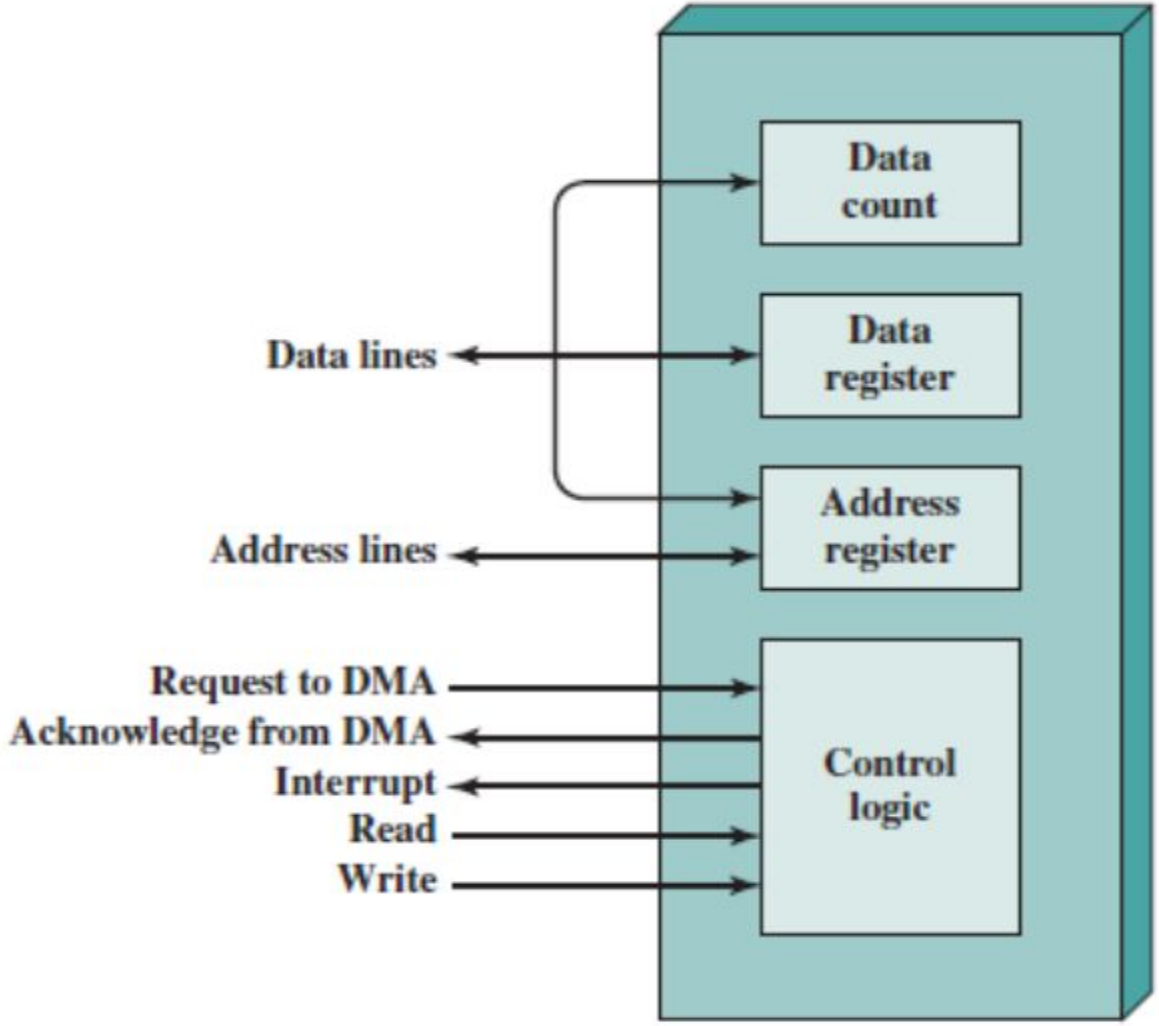
Burst or Block Transfer DMA

- It is the fastest DMA mode.
- In this mode, two or more data bytes are transferred continuously.
- The microprocessor is disconnected from the system bus during DMA transfer i.e. the microprocessor cannot execute its own program during this transfer.
- N number of DMA cycles are added into the machine cycles of the microprocessor where N is number of bytes to be transferred.
- In this mode, the DMA controller sends HOLD signal to the microprocessor and waits for HLDA signal.
- After receiving HLDA signal, the DMA controller gains control of the system bus and execute a DMA cycle to transfer one byte.
- After transferring one byte, it increments memory address, decrements counter and transfers next byte.
- In this way, it transfers all data bytes between memory and I/O devices. After transferring all data bytes, the DMA controller disables HOLD signal and enters in to slave mode.

Cycle Steal or Single Byte Transfer DMA

- In cycle steal transfer only one byte of data is transferred at a time.
- This type of DMA is slower than burst DMA.
- In this mode, only one DMA cycle is added between two machine cycles of the microprocessor, hence the instruction execution speed of the microprocessor is reduced slightly.
- In this mode the DMA controller sends HOLD signal to the microprocessor and waits for HLDA signal.
- After receiving HLDA signal, the DMA controller gains control of the system bus and execute only one DMA cycle.
- After transferring one byte, it disables HOLD signal and enters into slave mode.
- The microprocessor then gains control of the system bus and executes next machine cycle. If the count is not zero and next data is available then the DMA controller sends HOLD signal to the microprocessor and transfers next byte of data block.

DMA Module Diagram



DMA Operation

CPU carries on with other work.

CPU tells DMA controller.

- Read/Write.
- Device address.
- Starting address of memory block for data.

Amount of data to be transferred.

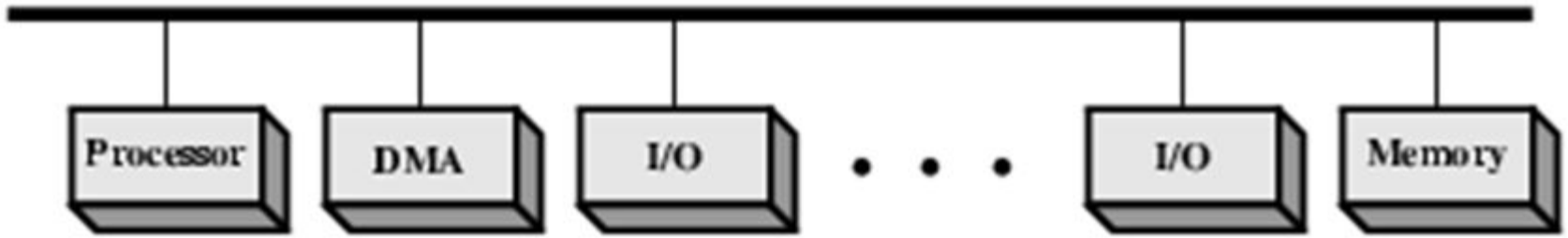
DMA controller deals with transfer.

DMA controller sends interrupt when finished.

DMA mechanism can be configured in a variety of ways, which are:

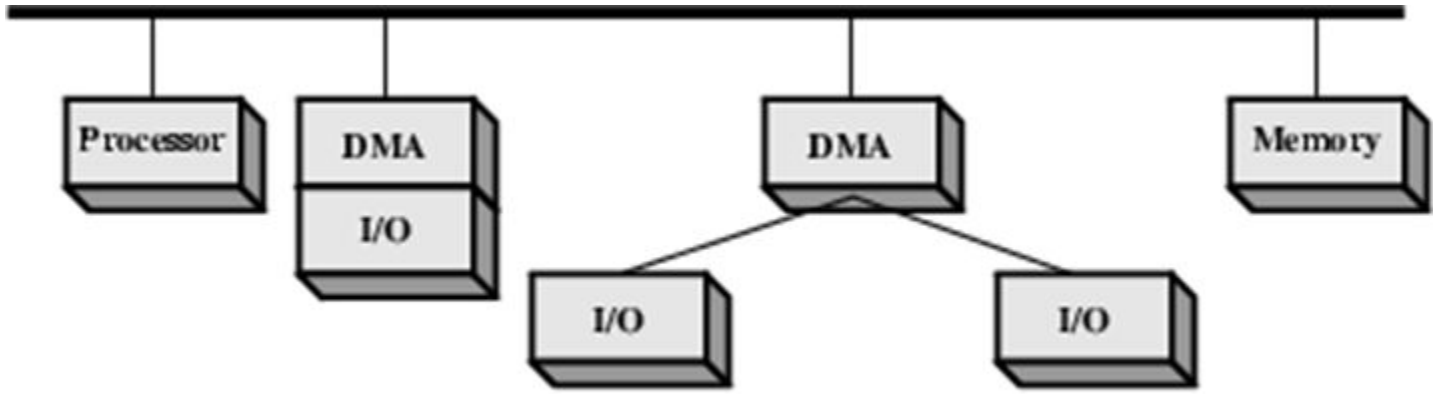
- Single-bus, detached DMA
- Single-bus, integrated DMA-I/O
- I/O bus

Single-bus, detached DMA



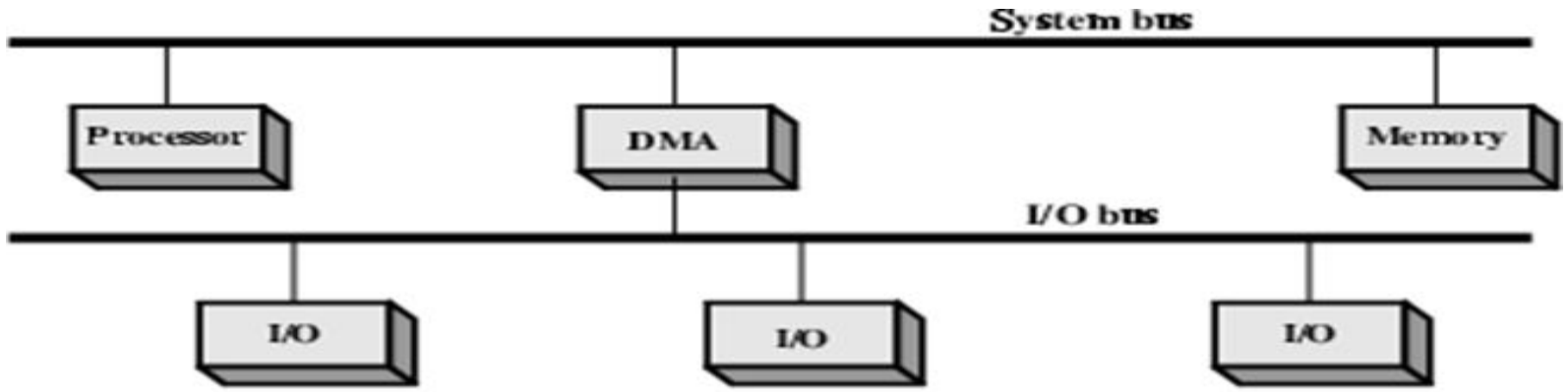
- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - I/O to DMA then DMAto memory
- CPU is suspended twice

Single-bus, integrated DMA



(b) Single-bus, Integrated DMA-I/O

- Single Bus, Integrated DMA controller
- Controller may support >1 device
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once



(c) I/O bus

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
- DMA to memory
- CPU is suspended once

Serial communication

DATA CAN BE TRANSMITTED IN THREE WAYS:

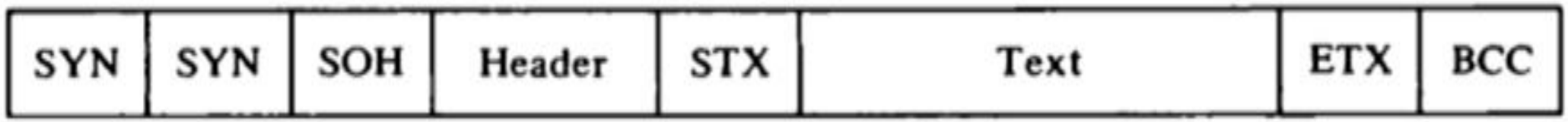
- Simplex
- Half duplex
- Full duplex

Character-Oriented Protocol

In character - oriented framing, data is transmitted as a sequence of bytes, from an 8-bit coding system like ASCII. The parts of a frame in a character - oriented framing are −

- Frame header − it contains the source and the destination addresses of the frame in form of bytes.
- Payload field − it contains the message to be delivered. It is a variable sequence of data bytes.
- Trailer − it contains the bytes for error detection and error correction.
- Flags − flags are the frame delimiters signaling the start and end of the frame. It is of 1- byte denoting a protocol - dependent special character.

CHARACTER ORIENTED FRAME FORMAT



Code	Symbol	Meaning	Function
0010110	SYN	Synchronous idle	Establishes synchronism
0000001	SOH	Start of heading	Heading of block message
0000010	STX	Start of text	Precedes block of text
0000011	ETX	End of text	Terminates block of text
0000100	EOT	End of transmission	Concludes transmission
0000110	ACK	Acknowledge	Affirmative acknowledgement
0010101	NAK	Negative acknowledge	Negative acknowledgement
0000101	ENQ	Inquiry	Inquire if terminal is on
0010111	ETB	End of transmission block	End of block of data
0010000	DLE	Data link escape	Special control character

Typical Transmission from a Terminal to Processor

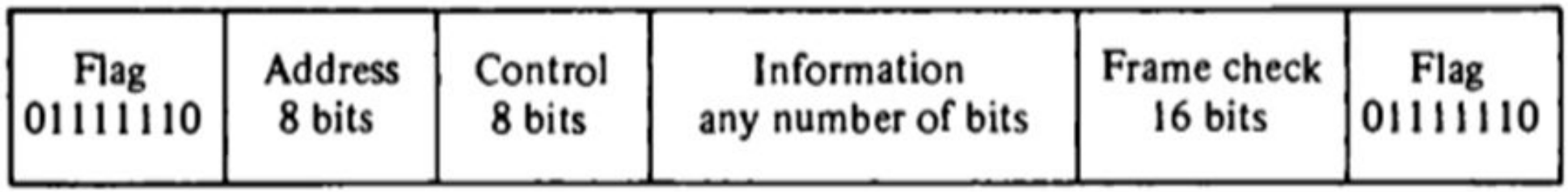
Code	Symbol	Comments
0001 0110	SYN	First sync character
0001 0110	SYN	Second sync character
0000 0001	SOH	Start of heading
0101 0100	T	Address of terminal is T4
0011 0100	4	
0000 0010	STX	Start of text transmission
0101 0010		
0100 0101	request	Text sent is a request to respond with the balance of
.	balance	account number 1234
.	of account	
.	No. 1234	
1011 0011		
0011 0100		
1000 0011	ETX	End of text transmission
0111 0000	LRC	Longitudinal parity character

Typical Transmission from Processor to Terminal

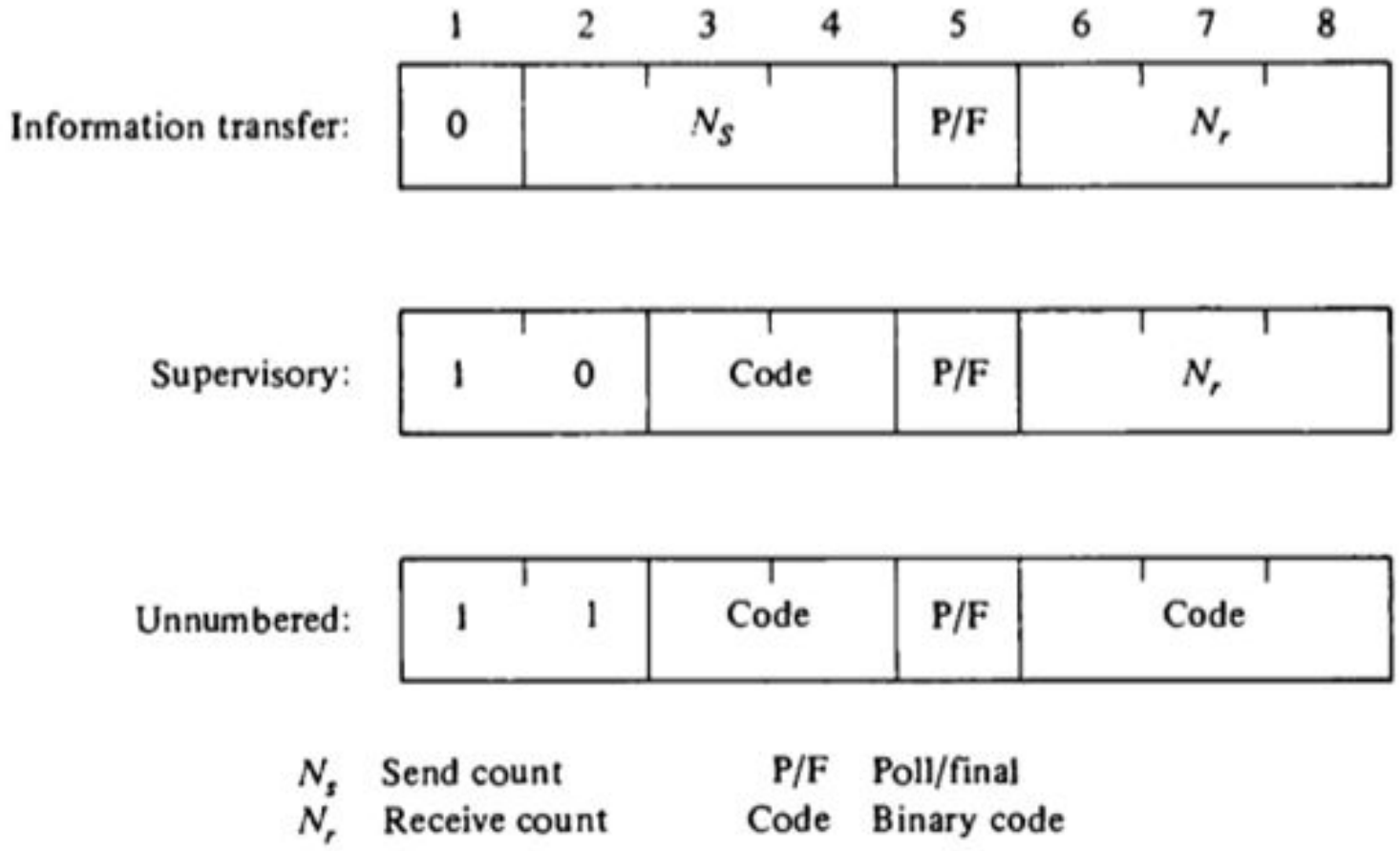
Code	Symbol	Comments
0001 0110	SYN	First sync character
0001 0110	SYN	Second sync character
1000 0110	ACK	Processor acknowledges previous message
0001 0110	SYN	Line is idling
.	.	
.	.	
.	.	
0001 0110	SYN	Line is idling
0000 0001	SOH	Start of heading
0101 0100	T	Address of terminal is T4
0011 0100	4	
0000 0010	STX	Start of text transmission
1100 0010		
1100 0001	balance	Text sent is a response from the computer giving the
.	is	balance of account
.	$100.00	
.		
.		
1011 0000		
1000 0011	ETX	End of text transmission
1101 0101	LRC	Longitudinal parity character

Bit-Oriented Protocol

Frame format



Control field format in bit-oriented protocol



In any operating system, it is necessary to have dual mode operation to ensure protection and security of the system from unauthorized or errant users . this dual mode separates the user mode from the system mode or kernel mode.

Privileged instructions

The Instructions that can run only in Kernel Mode are called Privileged Instructions .

Privileged Instructions possess the following characteristics :

(i)If any attempt is made to execute a Privileged Instruction in User Mode, then it will not be executed and treated as an illegal instruction. The Hardware traps it to the Operating System.

(ii) Before transferring the control to any User Program, it is the responsibility of the Operating System to ensure that the **Timer** is set to interrupt. Thus, if the timer interrupts then the Operating System regains the control.

Thus, any instruction which can modify the contents of the Timer is a Privileged Instruction.

(iii) Privileged Instructions are used by the Operating System in order to achieve correct operation.

(iv) Various examples of Privileged Instructions include:

- I/O instructions and Halt instructions
- Turn off all Interrupts
- Set the Timer
- Context Switching
- Clear the Memory or Remove a process from the Memory
- Modify entries in Device-status table

NON PRIVILEGED INSTRUCTIONS

The Instructions that can run only in User Mode are called Non-Privileged Instructions .

Various examples of Non-Privileged Instructions include:

- Reading the status of Processor
- Reading the System Time
- Generate any Trap Instruction

Also, it is important to note that in order to change the mode from Privileged to Non-Privileged, we require a Non-privileged Instruction that does not generate any interrupt.

Program and processes

Program:

when we execute a program that was just compiled, the OS will generate a process to execute the program. Execution of the program starts via GUI mouse clicks, command line entry of its name, etc. A program is a passive entity as it resides in the secondary memory, such as the contents of a file stored on disk. One program can have several processes.

Process:

the term process (job) refers to program code that has been loaded into a computer's memory so that it can be executed by the central processing unit (CPU). A process can be described as an instance of a program running on a computer or as an entity that can be assigned to and executed on a processor. A program becomes a process when loaded into memory and thus is an active entity

Program vs process

SR.NO.	PROGRAM	PROCESS
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
6.	Program does not have any control block.	Process has its own control block called Process Control Block.