REGISTERS, COUNTERS AND THE MEMORY UNIT

A register includes a set of flip-flops. Since each flip-flop is capable of storing one bit, an n-bit register includes n flip-flops and can store n bits of binary information. In general, a register consists of a group of flip-flops and gates that affect their state transitions and perform data-processing tasks.

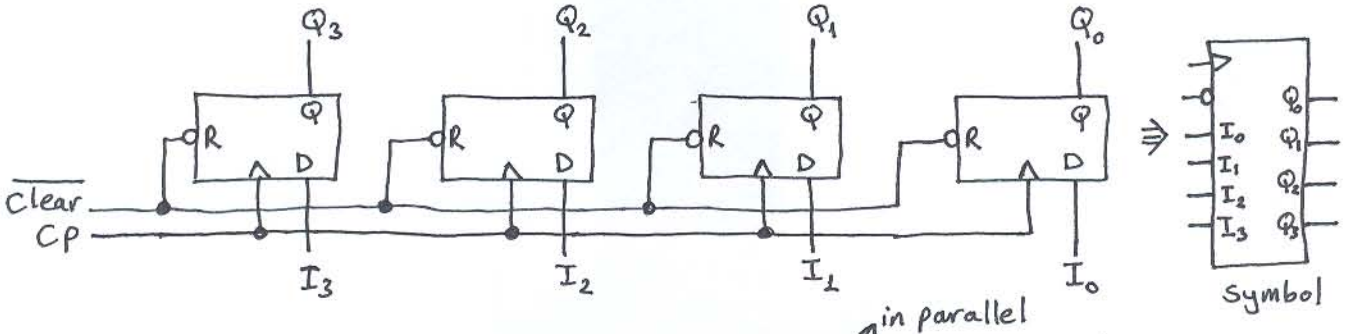
A counter is a register that goes through a predetermined sequence of states upon the application of clock pulses.

Therefore, registers and counters are sequential functional blocks, which are used extensively in the design of digital systems.

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of storage. A random-access memory (RAM) differs from a read-only memory (ROM) in that a RAM can transfer the stored information out (read) and is also capable of receiving new information in for storage (write).

REGISTERS:

The simplest register is a register that consists of only flip-flops without external gates. A 4-bit register constructed with D flip-flops is shown below.



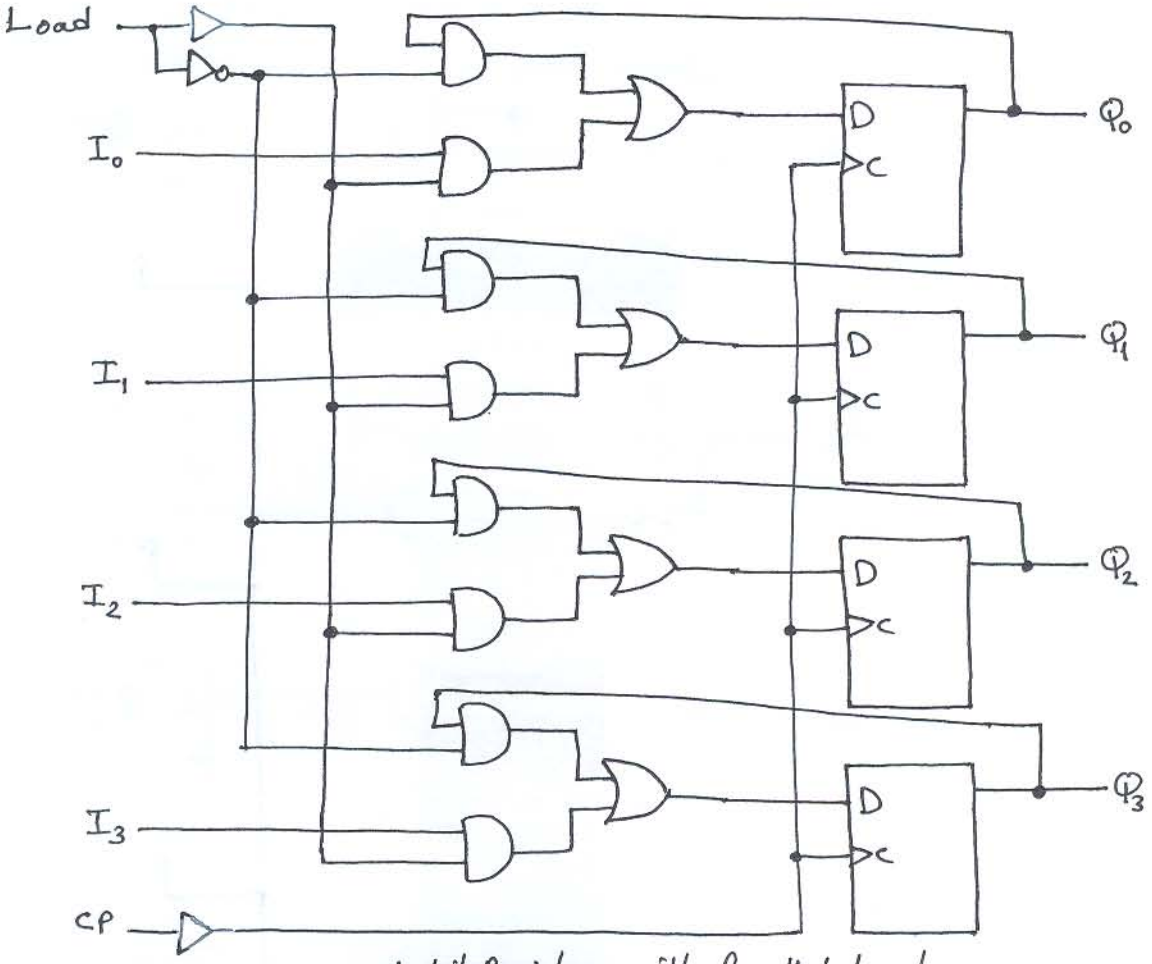
The binary data $I_3\ I_2\ I_1\ I_0$ are transferred into the 4-bit register on the rising edge of each clock pulse. This data is retained at the output until the next rising edge of the clock.

The $\overline{\text{clear}}$ input is an active-low input and used for clearing the register to all 0's prior to its clocked operation. Thus, we maintain $\overline{\text{clear}}$ at Logic 1 during normal clocked operation.

The transfer of new information into a register is referred to as Loading the register.

Register with Parallel Load:

A 4-bit register with a control input Load that determines the action to be taken with each clock pulse is shown below.



4-bit Register with Parallel Load

When Load is 1, the data on the four inputs is transferred into the register with the next positive edge of the clock.

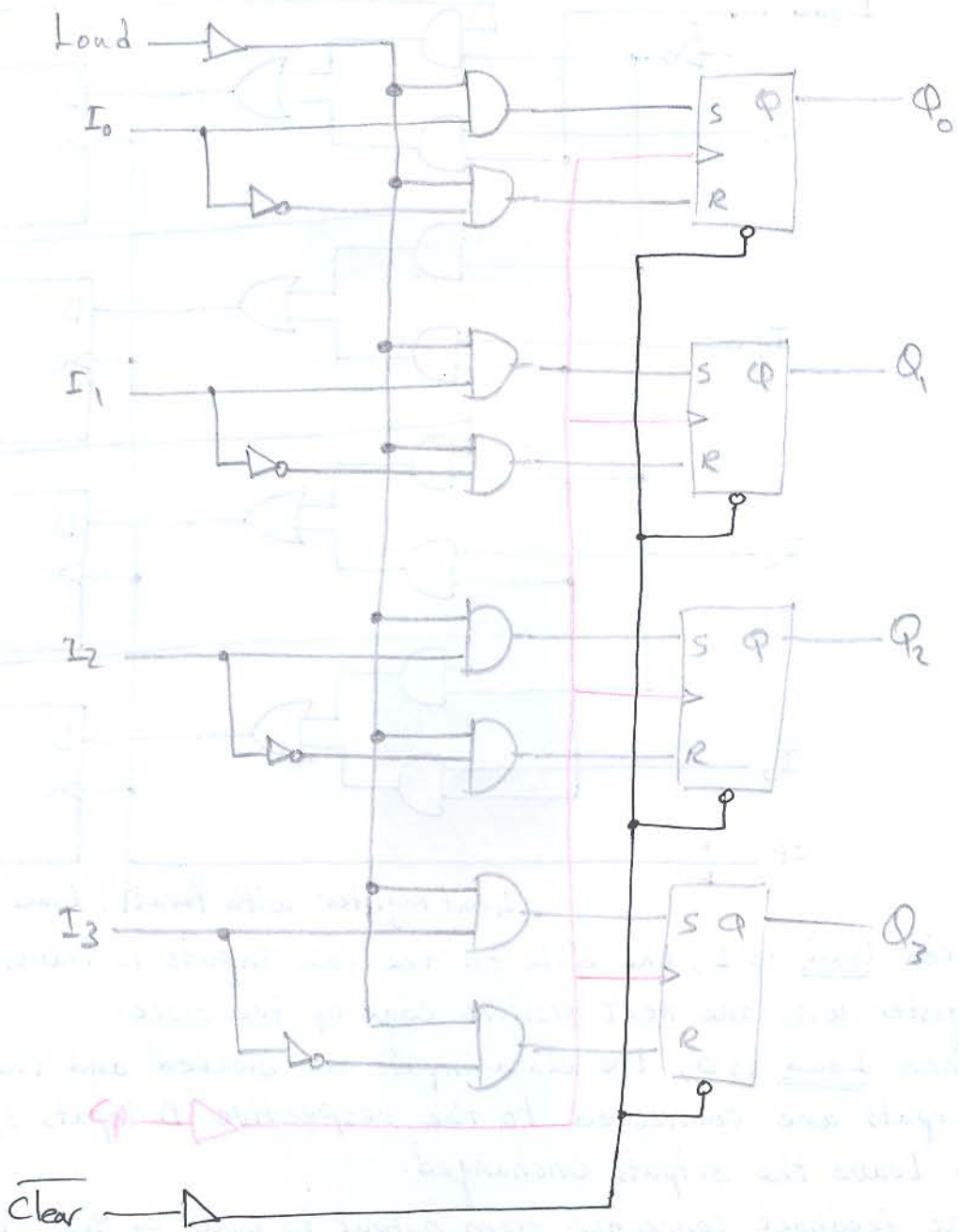
When Load is 0, the data inputs are blocked and the flip-flop outputs are connected to the respective D inputs of the flip-flops, to leave the outputs unchanged.

The feedback connection from output to input of the flip-flop is necessary because the D flip-flop does not have a "no change" input condition: with each clock pulse, the D input determines the next state of the output.

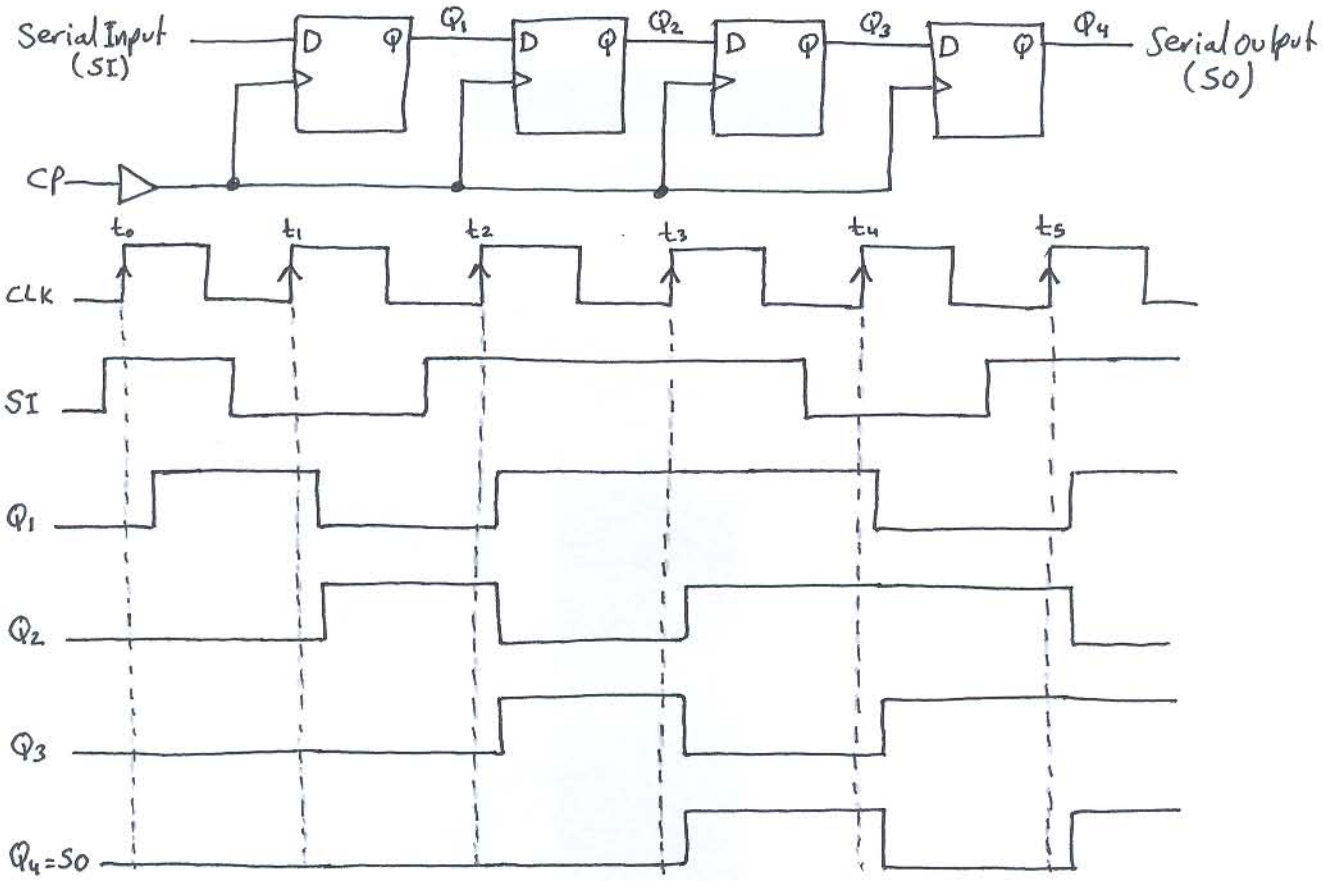
Exercise: Draw the Logic diagram of a 4-bit register with parallel load using RS FFs.

SHIFT REGISTERS:

In computer systems, it is often necessary to transfer n-bit data items. This may be done by transmitting all bits at once using n separate wires, in which we say that the transfer is performed in parallel.


Load
I_0
I_1
I_2
I_3
S
Q
R
Q_0
Q_1
Q_2
Q_3
CP
$\overline{Clear}$

It is also possible to transfer all bits using a single wire, by performing the transfer one bit at a time in n consecutive clock cycles, in which case we say that the transfer is performed in <u>serial</u>.
To do a serial transfer, we need a shift register.
The simplest possible shift register is one that uses only flip-flops as shown below.

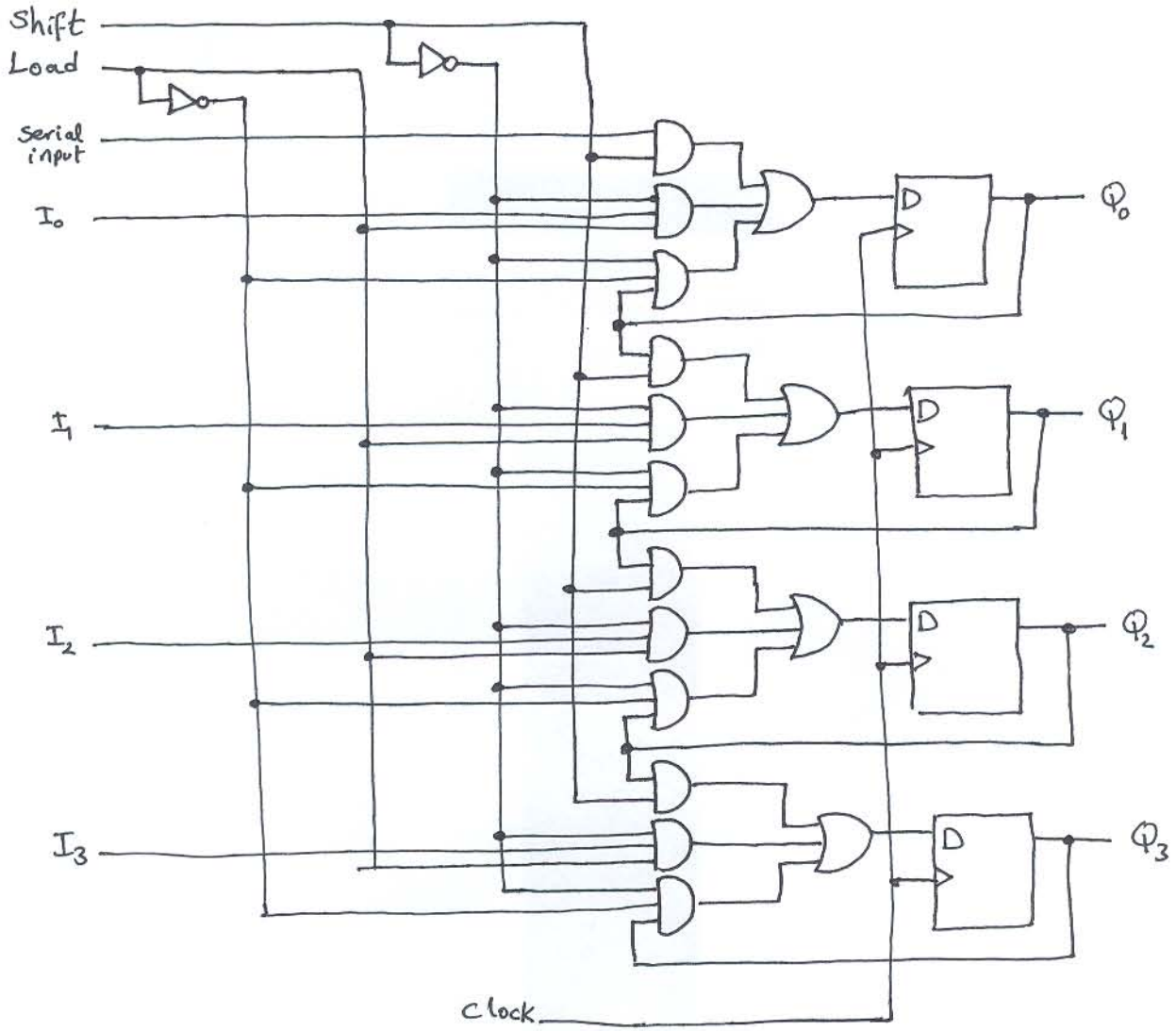


	SI	Q_1	Q_2	Q_3	Q_4=SO
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	0	1	1	0	1
t_5	1	0	1	1	0

A symbol of the shift register is as follows:

Shift Register with Parallel Load:

A shift register with accessible flip-flop outputs and parallel load can be used for converting incoming parallel data to outgoing serial data and vice versa. A 4-bit shift register with parallel load is shown below:

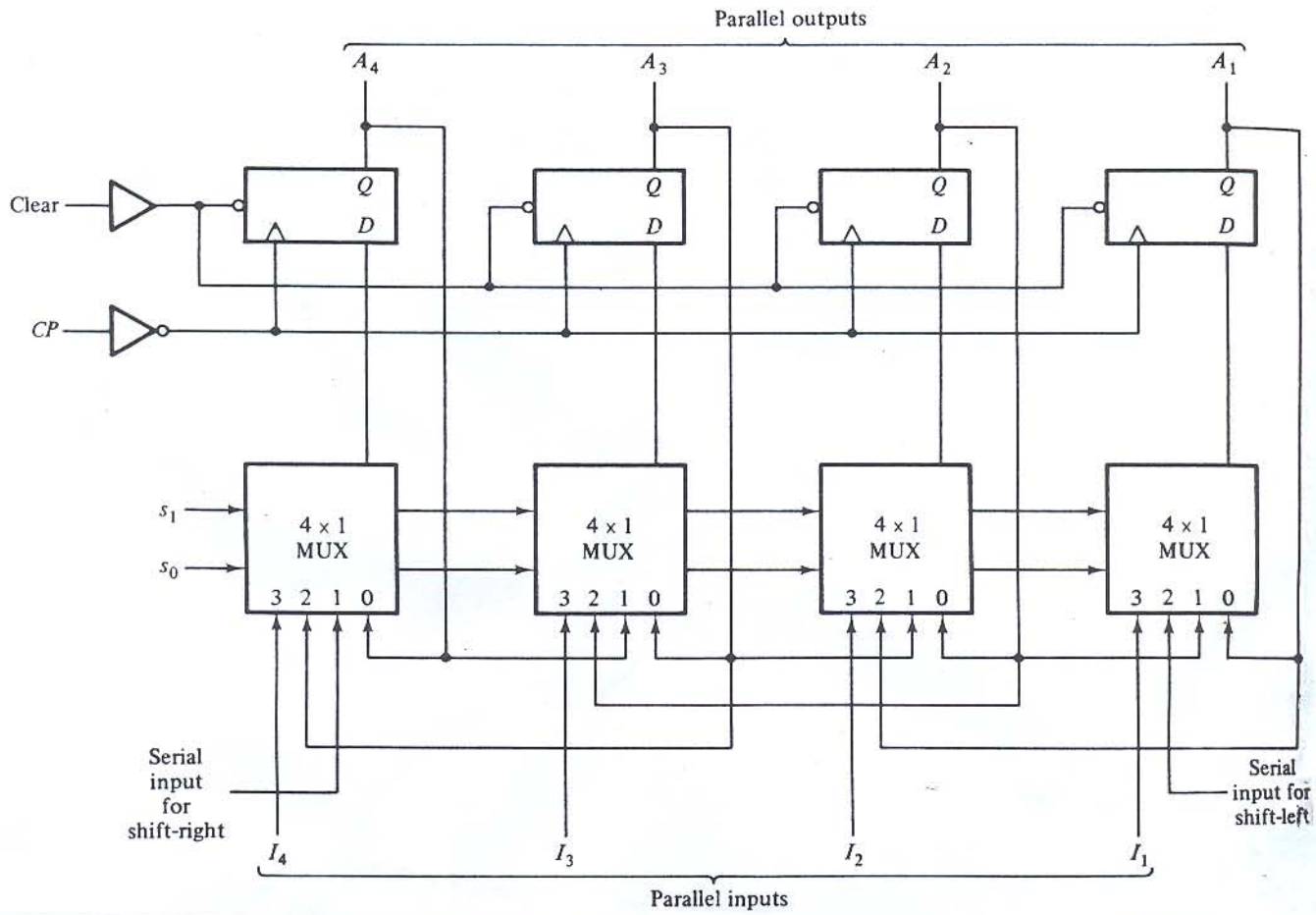


There are two control inputs, one for the shift and the other for the load. In each stage, the first AND gate enables the shift operation; the second AND gate enables the input data (load); and the third AND gate restores the contents of the register when no operation is required.

Shift	Load	operation
0	0	No change
0	1	Load Parallel data ($I_0 \rightarrow Q_0$; $I_1 \rightarrow Q_1$; $I_2 \rightarrow Q_2$; $I_3 \rightarrow Q_3$)
1	X	Shift ($SI \rightarrow Q_0$; $Q_0 \rightarrow Q_1$; $Q_1 \rightarrow Q_2$; $Q_2 \rightarrow Q_3$)

Bidirectional shift Register with Parallel Load:

Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities. A 4-bit bidirectional shift register with parallel load is shown below.



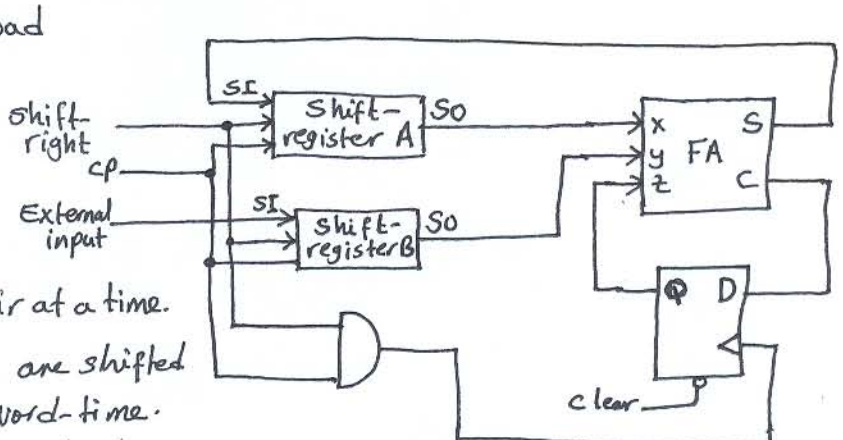
. Mode Control

S_1	S_0	Register operation
0	0	No change
0	1	Shift right
1	0	shift left
1	1	Parallel load

Note: The four multiplexers (MUX) are part of the register.

Serial Adder:

The two numbers are stored in the two shift registers.



Bits are added one pair at a time. The two shift registers are shifted to the right for one word-time. the A register holds the result also.

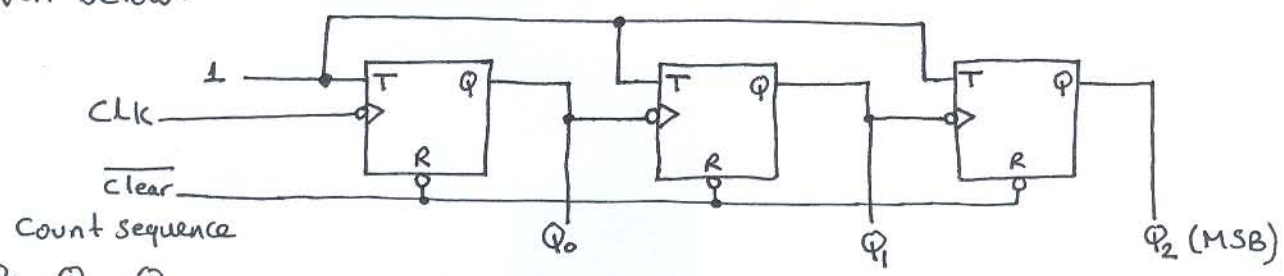
RIPPLE COUNTERS:

In general, counters are available in two categories: ripple counters (or asynchronous) and synchronous counters.

In a ripple counter, the flip-flop output transition serves as a source for triggering other flip-flops. i.e., the C input of some or all of the flip-flops is triggered not by the common clock pulses, but by the transition that occurs in other flip-flop outputs.

In a synchronous counter, the C inputs of all of the flip-flops are triggered by the common clock pulse, and the change of state is determined from the present state of the counter.

The logic diagram of a 3-bit binary ripple (up-) counter using T FFs is given below:



Count sequence

Q_2	Q_1	Q_0	
0	0	0	
0	0	1	
0	1	0	Q_0 goes from 1 to 0 ↓ and complement Q_1.
0	1	1	
1	0	0	Q_0 goes ↓ ⇒ it complements Q_1. Also Q_1 goes ↓ ⇒ it complements Q_2.
1	0	1	
1	1	0	
1	1	1	

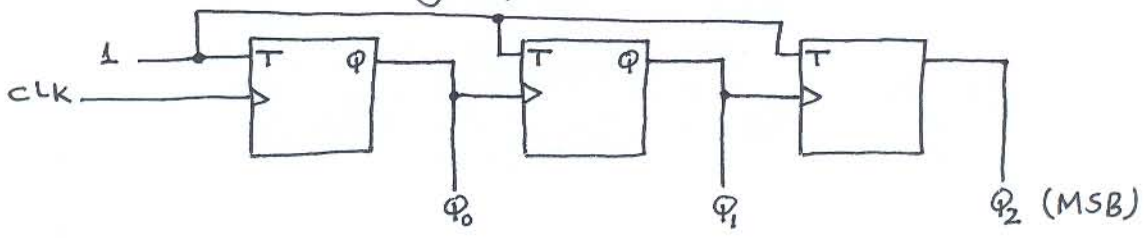
i.e. Q_0 is complemented with each clock pulse input.

Every time that Q_0 goes from 1 to 0 ↓, it complements Q_1.

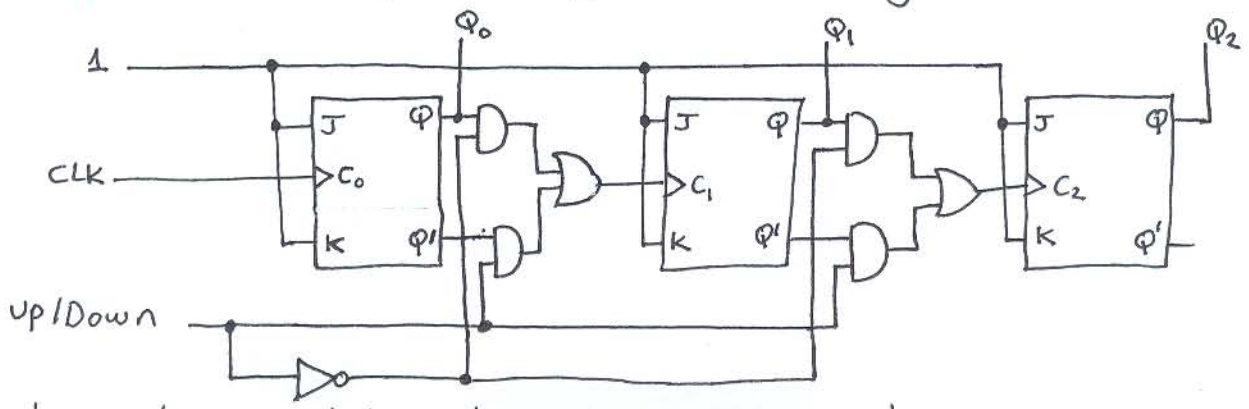
Every time that Q_1 goes from 1 to 0 ↓, it complements Q_2.

For normal operation, the $\overline{\text{clear}}$ input should be disabled.

Exercise: Consider the following ripple counter and find its count sequence.



Example: 3-bit up/Down ripple counter using JK FFs.

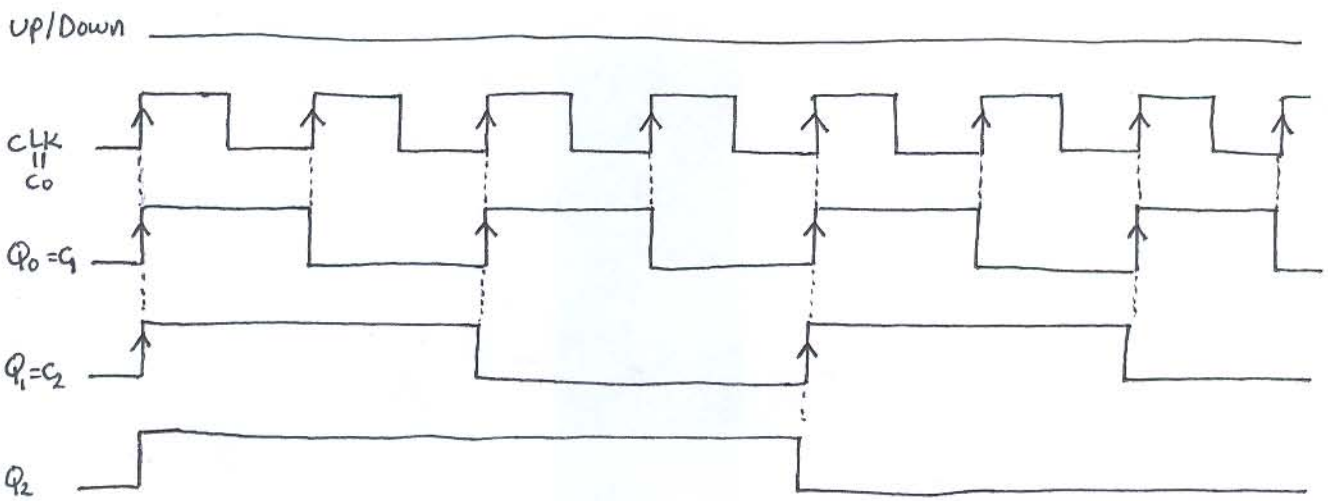


When up/Down control input is 0 ⇒ Down counter.

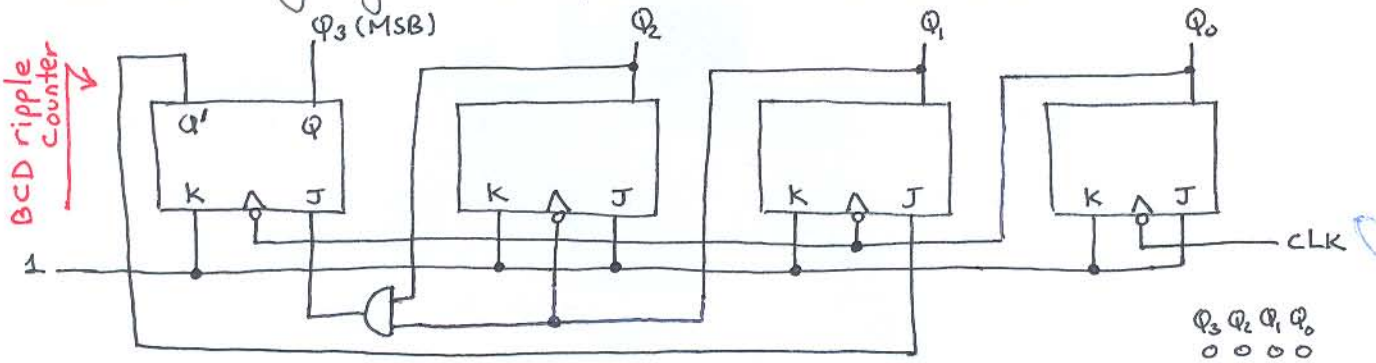
" " " " " " 1 ⇒ up counter.

Note that the FFs toggle when their C inputs go from 0 to 1 ↑ i.e (positive-edge triggered).

Consider the case when up/Down = 0 ⇒



Draw the timing diagram for up/Down = 1 case. (Exercise!)



Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	0
0	0	0	0
:	:	:	:
:	:	:	:

Q_0 is complemented on the negative edge of every CLK pulse.

Q_1 is complemented if $Q_3 = 0$ and Q_0 goes from 1 to 0, and it is cleared if $Q_3 = 1$ and Q_0 goes from 1 to 0. (⇒ J=0, K=1)

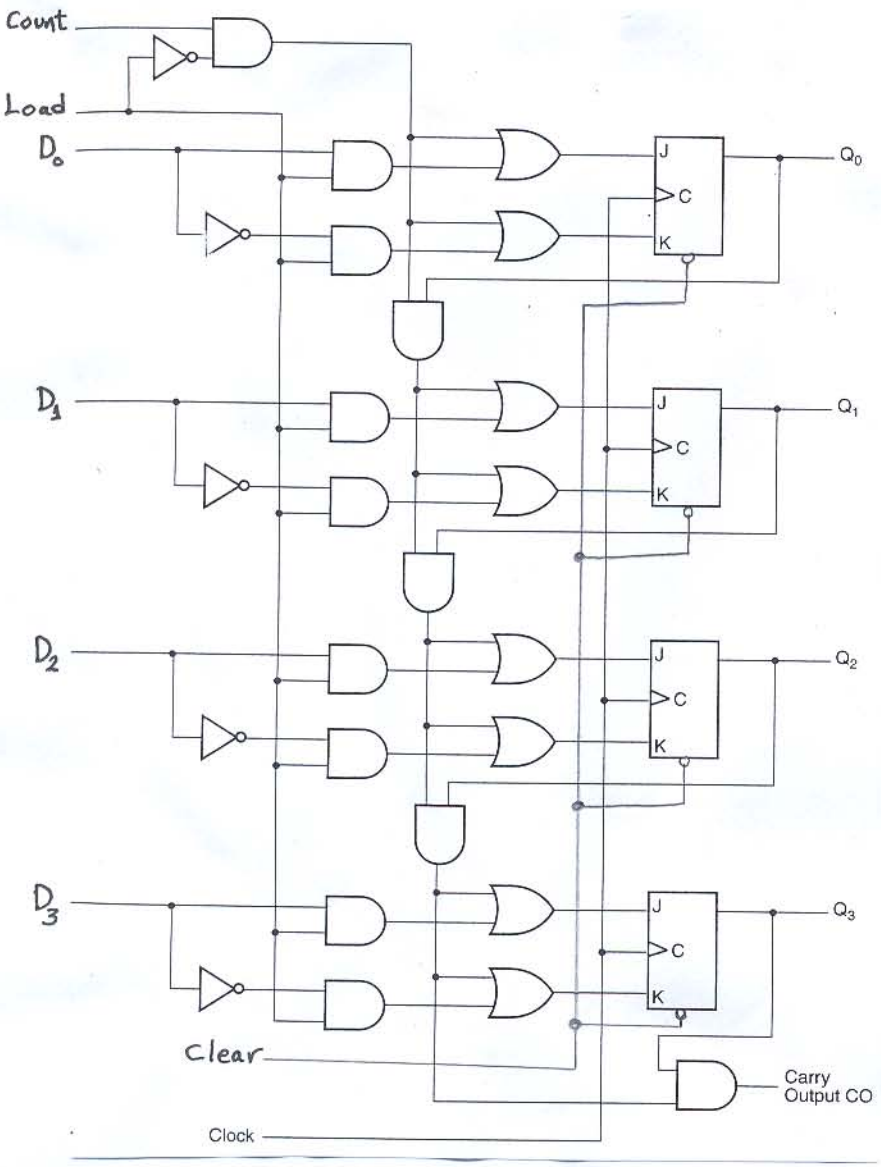
Q_2 is complemented if Q_1 goes from 1 to 0.

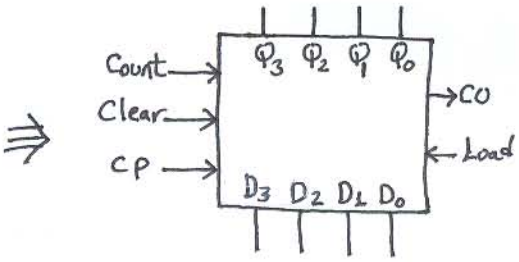
Q_3 is complemented when $Q_2 Q_1 = 11$ and Q_0 goes from 1 to 0; and it is cleared if either Q_2 or Q_1 is 0 and Q_0 goes from 1 to 0. (J=0, K=1)

⇒ Not straightforward !!! ingenuity and imagination are required !!!

Note: The design procedure for a synchronous counter is the same as with other synchronous sequential circuits. A counter may operate without an external input, except for the clock pulse which is applied to the C input of all flip-flops.

The logic diagram of a 4-bit binary synchronous counter with parallel load is given below



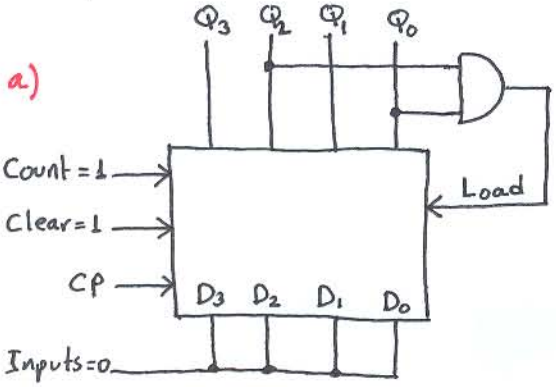


Clear	CP	Load	Count	Function
0	X	X	X	Clear to 0
1	X	0	0	No change
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state

The carry out (CO) can be used to extend the counter to more stages.

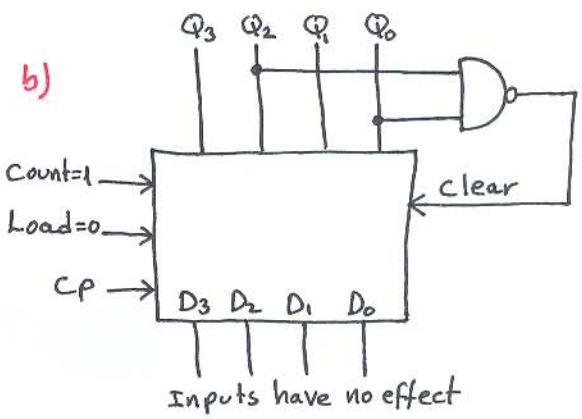
Example: construct a mod-6 counter using the MSI circuit of the previous 4-bit counter with parallel load.

a)



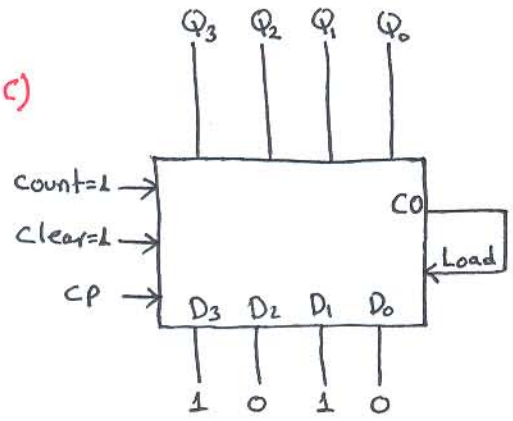
Binary sequence: 0, 1, 2, 3, 4, 5, 0, 1 ...

b)



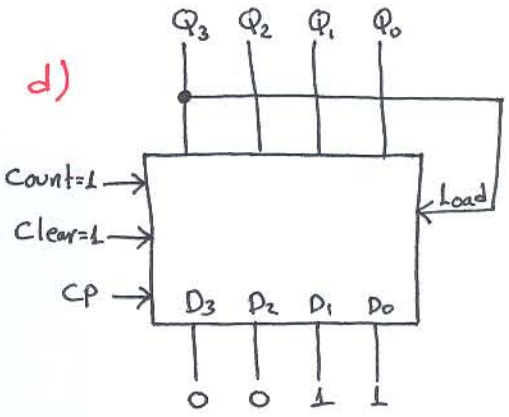
Binary sequence: 0, 1, 2, 3, 4, 5, 0, 1, ..

c)



Binary sequence: 10, 11, 12, 13, 14, 15, 10, 11, ...

d)



Binary sequence: 3, 4, 5, 6, 7, 8, 3, 4, ...

Although synchronous counters can be designed using the traditional design procedure of synchronous sequential circuit, it is sometimes possible and more time saving to use intution method.

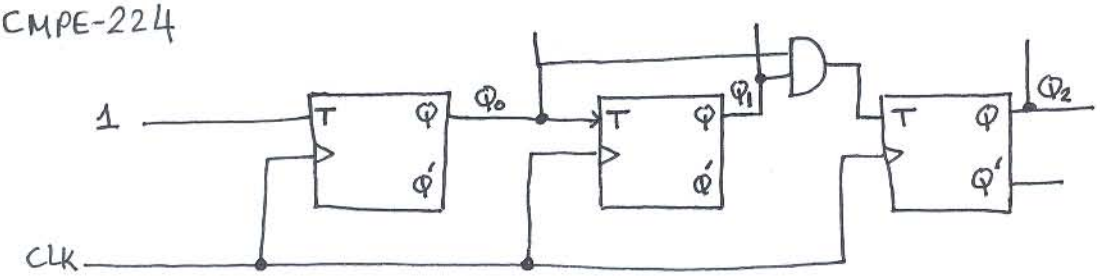
Example:

Using T FFs, design a 3-bit synchronous up-counter, that counts the binary sequence: 0, 1, 2, 3, 4, 5, 6, 7, 0, ...

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0
⋮	⋮	⋮

Note that:
- Q_0 toggles (changes) on each clock cycle. $\Rightarrow T_0 = 1$
- Q_1 changes only when $Q_0 = 1$ prior to the clock pulse $\Rightarrow T_1 = Q_0$.
- Q_2 changes only when $Q_1 Q_0 = 11$ prior to the clock pulse $\Rightarrow T_2 = Q_1 Q_0$

$\Longrightarrow$ The circuit:

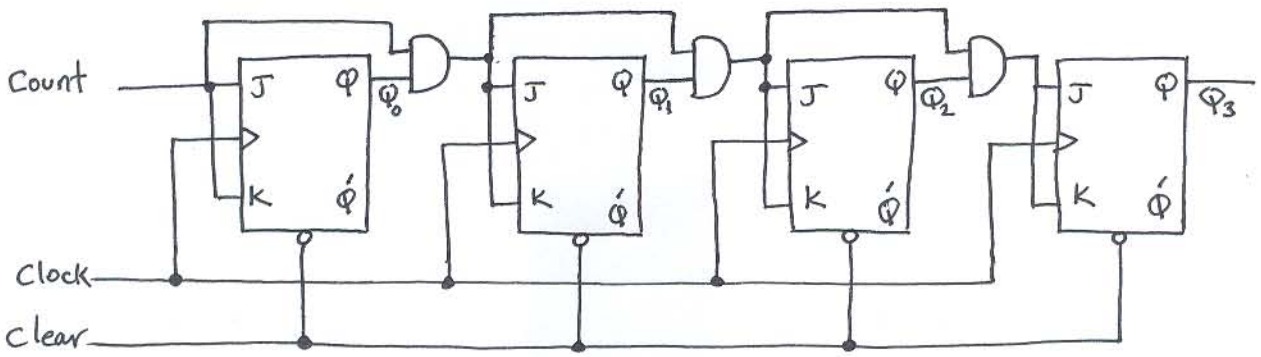


Similarly, for an n-bit counter we have:

$T_0 = 1$; $T_1 = Q_0$; $T_2 = Q_0 Q_1$; $T_3 = Q_0 Q_1 Q_2$ ..... ; $T_n = Q_0 Q_1 \cdots Q_{n-1}$.

Of course, it is possible to inhibit counting by using the control input "count" or "Enable", together with the clear input.

A 4-bit synchronous counter with the above capabilities, and using JK FF's is shown below:



⇒ If count=0 ⇒ all J, K inputs are zero ⇒ No change.

If count=1 ⇒ $J_0 = K_0 = 1$

$J_1 = K_1 = Q_0$

$J_2 = K_2 = Q_0 Q_1$

$J_3 = K_3 = Q_0 Q_1 Q_2$

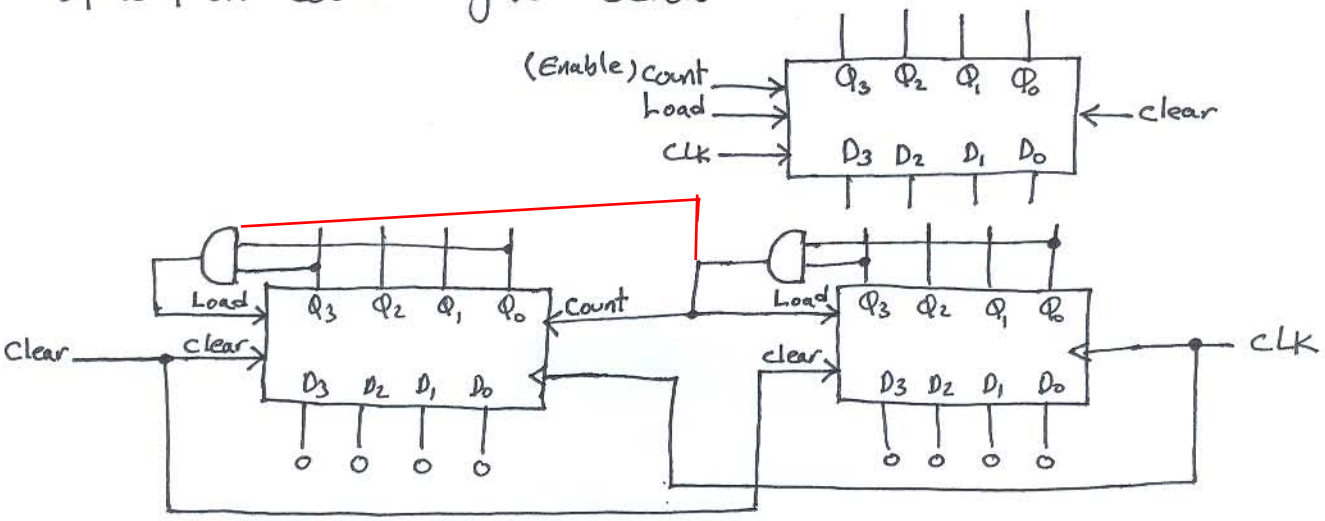
⇒ counting sequence: 0, 1, ..., 15

GO TO page (37') ...

MEMORY AND PROGRAMMABLE LOGIC DEVICES:

- Memory is a collection of cells capable of storing binary information. In addition to these cells, memory contains electronic circuits for storing and retrieving the information.
- Two types of memories are used in various parts of a digital system: Random Access Memory (RAM) and Read-only Memory (ROM). RAM accepts new information for storage to be available later for use.
- The process of storing new information in memory is referred to as a memory Write operation.

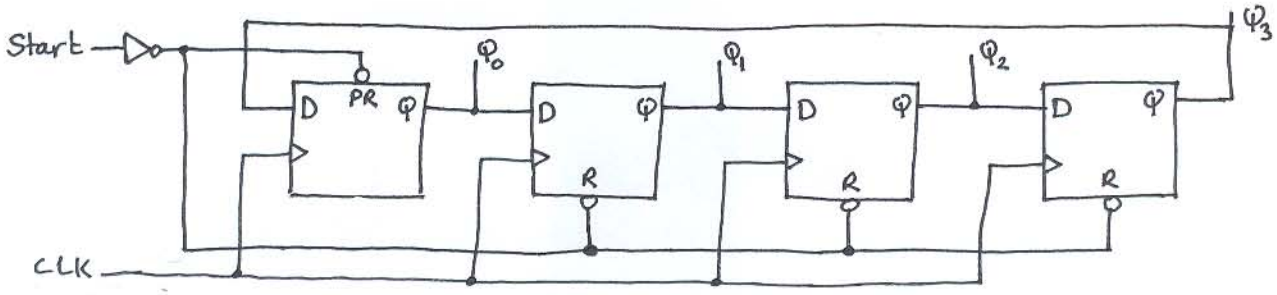
Example: Design a 2-digit BCD counter using the MSI circuit of a 4-bit counter given below:



The above circuit consists of two modulo-10 counters, one for each BCD digit. After reaching "9", it is necessary to reset the counter.

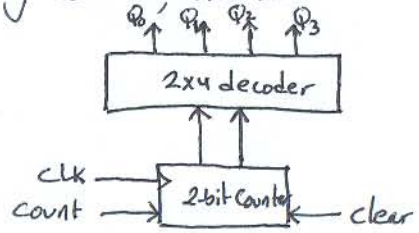
Example: (RING COUNTER)

A ring counter is a circular shift register with only one flip-flop being set at any particular time; all others are cleared.
A 4-bit ring counter is shown below.



Q_3	Q_2	Q_1	Q_0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

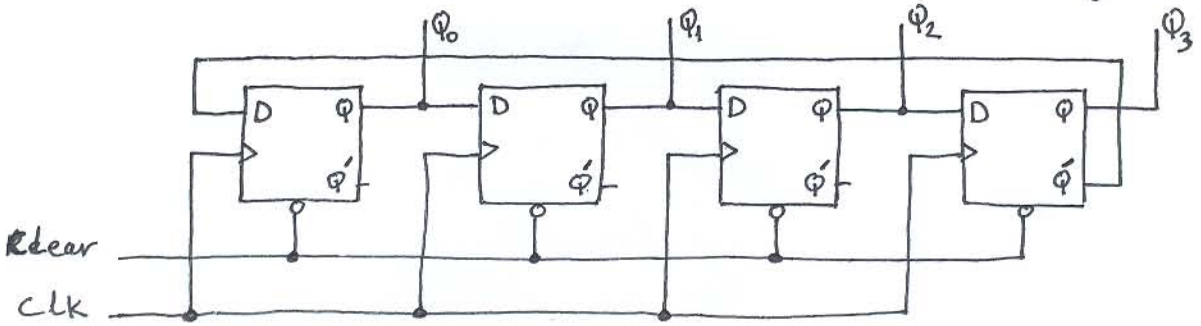
A ring counter can also be constructed by using a counter and a decoder. For example, to construct a 4-bit ring counter we need a 2-bit counter and a 2x4 decoder as follows:



Johnson Counter:

A k-bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states. The number of states can be doubled if the shift register is connected as a switch-tail ring counter. That is, the complement output of the last flip-flop is connected to the input of the first flip-flop.

A 4-bit Johnson counter has 8 states and its circuit is given below:



Q_3	Q_2	Q_1	Q_0	
0	0	0	0	← by applying clear
0	0	0	1	clear is disabled.
0	0	1	1	⋮
0	1	1	1	
1	1	1	1	
1	1	1	0	
1	1	0	0	
1	0	0	0	

Note that if you follow the traditional procedure of sequential circuit design, you will end up with the same circuit. i.e:

PS				NS							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D_{Q_3}	D_{Q_2}	D_{Q_1}	D_{Q_0}
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1	1
0	0	1	1	0	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	0
1	1	1	0	1	1	0	0	1	1	0	0
1	1	0	0	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0

$\Rightarrow D_{Q_3} = Q_2$ (by intuition)

$D_{Q_2} = Q_1$

$D_{Q_1} = Q_0$

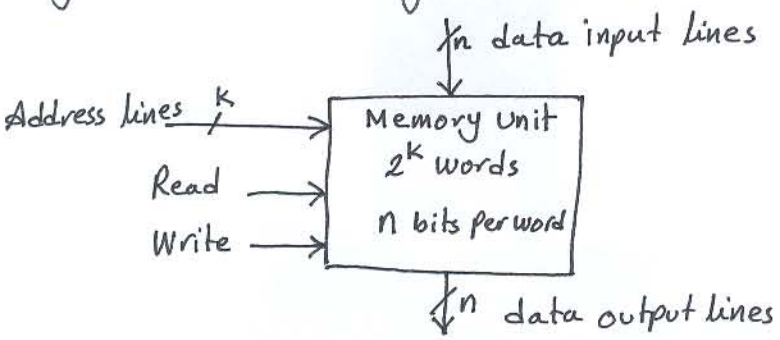
$D_{Q_0} = Q_3'$

Exercise: Modify the above Johnson counter so that it becomes self-correcting.

Note: Ring counters and Johnson counters are usually used to generate timing sequences.

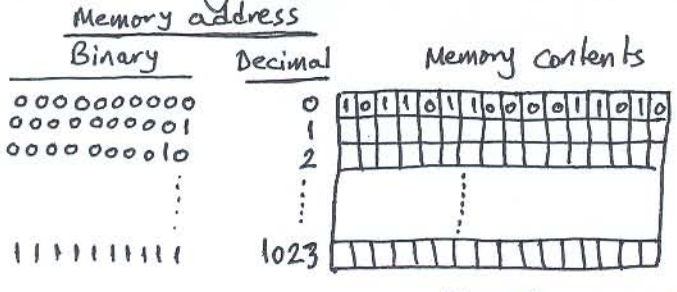
- The process of transferring the stored information out of memory is referred to as a memory read operation.
- RAM can perform both the write and the read operations. RAMs may range in size from hundreds to billions of bits.
- ROM is a programable Logic device (PLD). The binary information that is stored within such a device specified and then embedded in hardware. This process is referred as programming the device. Unlike RAM, ROM can perform only the memory read operation.
- A memory unit stores the binary information in groups of bits called words. A word is an entity of bits that moves in and out of memory as a unit — a group of 1's and 0's that represents a number, an instruction, a character, or other binary-coded information.
- A group of 8 bits is called a byte.
- Most Computer memories use words that are multiples of eight bits in length. Thus a 32-bit word is made up of 4 bytes ... etc.
- The capacity of a memory unit is usually stated as the total number of bytes that it can store.
- The communication between a memory and its environment is acheived through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

A block diagram of the memory unit is shown below:



The k address lines specify the particular word chosen throug a decodor. For example, a memory unit with a capacity of 1K words of 16 bits each:

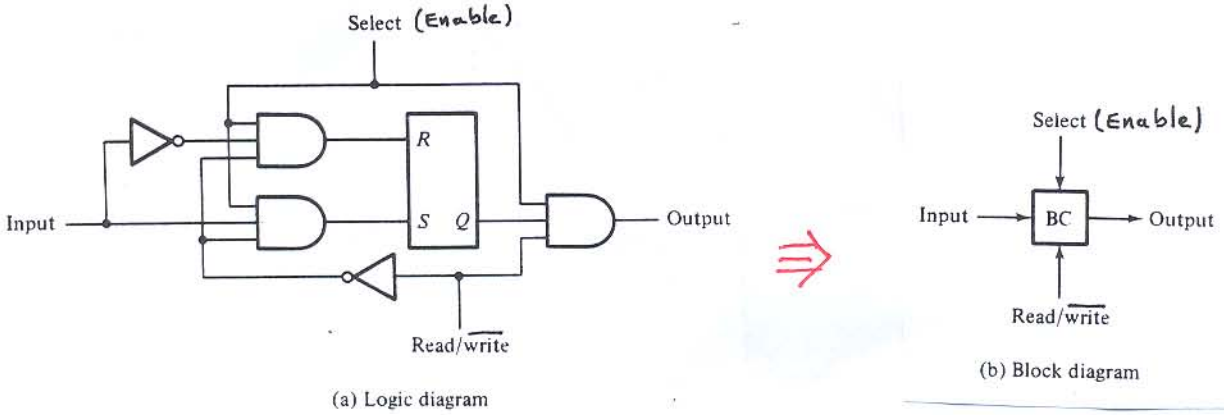
$1K = 2^{10}$



$\therefore$ k = 10
n = 16

1K X 16 Memory.

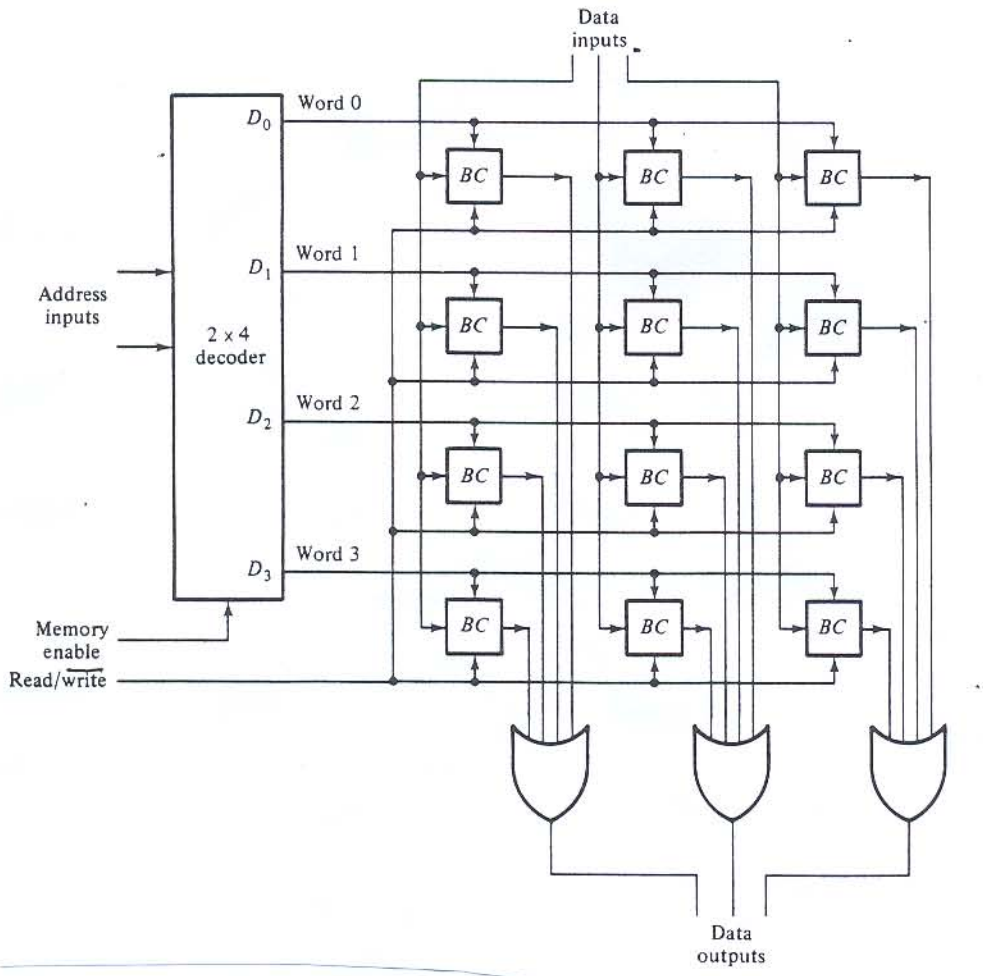
In a RAM unit, the equivalent logic of a binary cell (BC) can be shown as follows:



(a) Logic diagram

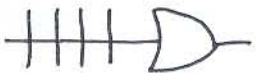
(b) Block diagram

The logical construction of a 4x3 RAM is shown below:



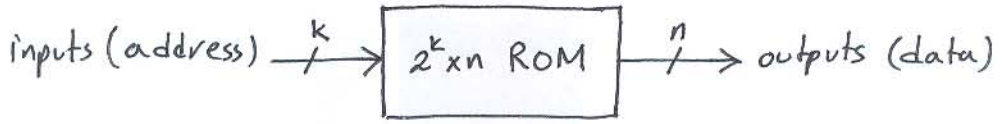
Remember that ROM is just one example of a PLD. Other such devices are the programmable logic array (PLA), the programmable array logic (PAL) device, the complex programmable logic device (CPLD), and the field-programmable gate array (FPGA).
In general, a PLD is an IC with internal logic gates and/or connection that can in some way be changed by a programming process. One of the simplest technologies employs fuses. Originally, all fuses are intact. Programming the device involves blowing some of the fuses.

conventional symbol of OR gate



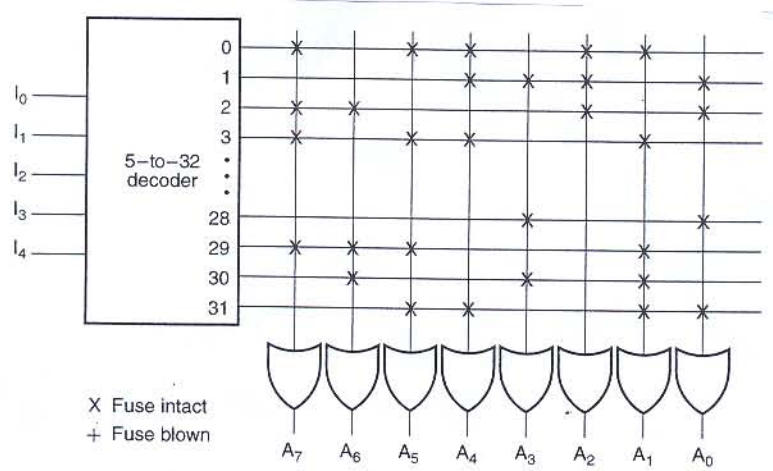
Array Logic symbol of OR gate

If an x is present at the intersection of two lines ⇒ there is a connection.
If an x is not present ⇒ there is No connection.
The block diagram of a $2^k \times n$ ROM is as follows:



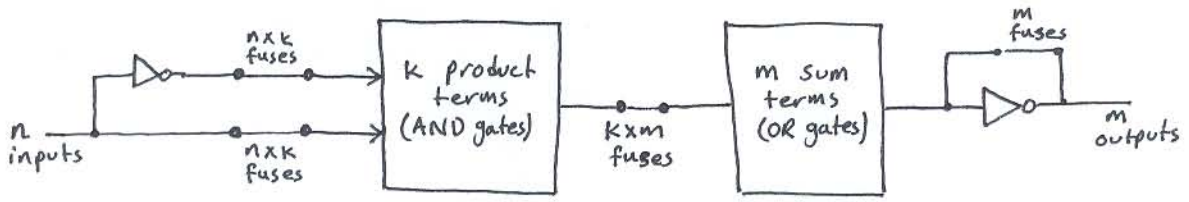
An example of programming a 32x8 ROM (i.e $2^5 \times 8$) according to the given table is shown below:

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1



PROGRAMMABLE LOGIC ARRAY (PLA):

PLA is a general purpose programmable combinational circuit whose block diagram is given below.

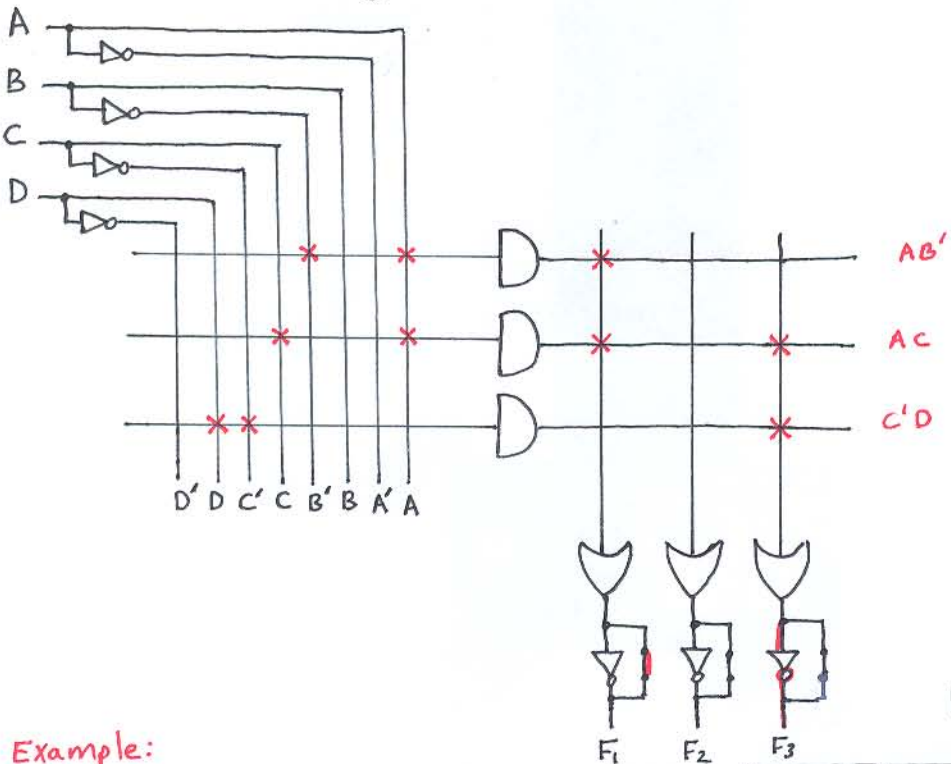


It consists of n inputs, m outputs, k product terms, and m sum terms. The product terms constitute a group of k AND gates and the sum terms constitute a group of m OR gates. Fuses are inserted between all n inputs and their complements to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates. Another set of fuses at the output allows the output functions to be generated either in AND-OR form or in AND-OR-INVERT form (i.e complements of functions).

(Note: If the OR gates are not programmable [i.e no kxm fuses] ⇒ PAL).

Example:

Consider the following PLA which has 4 inputs, 3 product terms and 3 outputs.



PLA program table

	Product term	Inputs A B C D	Outputs F1 F2 F3
AB'	1	1 0 – –	1 – –
AC	2	1 – 1 –	1 1 –
C'D	3	– – 0 1	– 1 –
		T C –	T/C

Example:

Implement the following functions using the above PLA.

$F_1 = AB' + AC$

$F_2 = (AC + C'D)'$

HW #2: 7.5; 7.7; 7.8; 7.14; 7.21; 7.32; 7.33