* Registers, Counters, and the Memory Unit

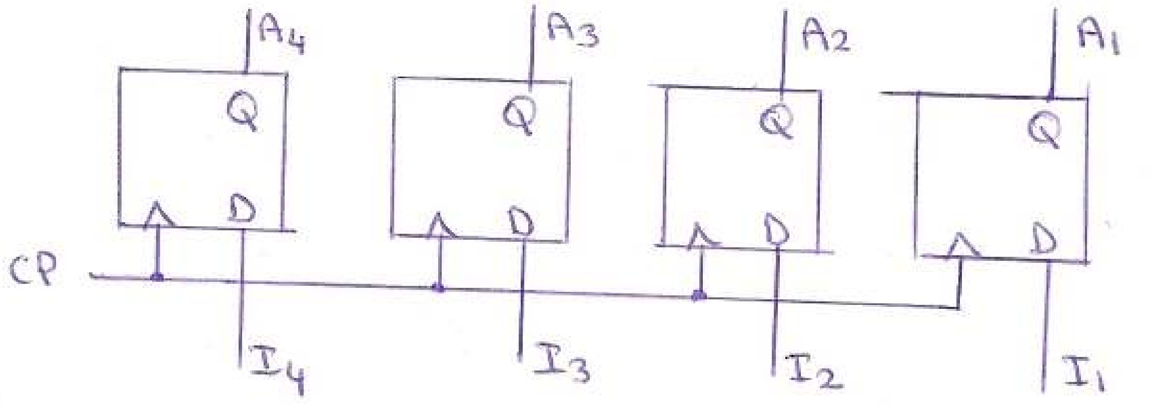
- A register is a group of flip-flops. Since each flip-flop is capable of storing one bit, an n-bit register has a group of n flip-flops and can store n bits of binary information. In general, a register consists of a group of flip-flops and gates that affect their state transition.

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

- 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). However, ROM can only provide read operation

Registers :

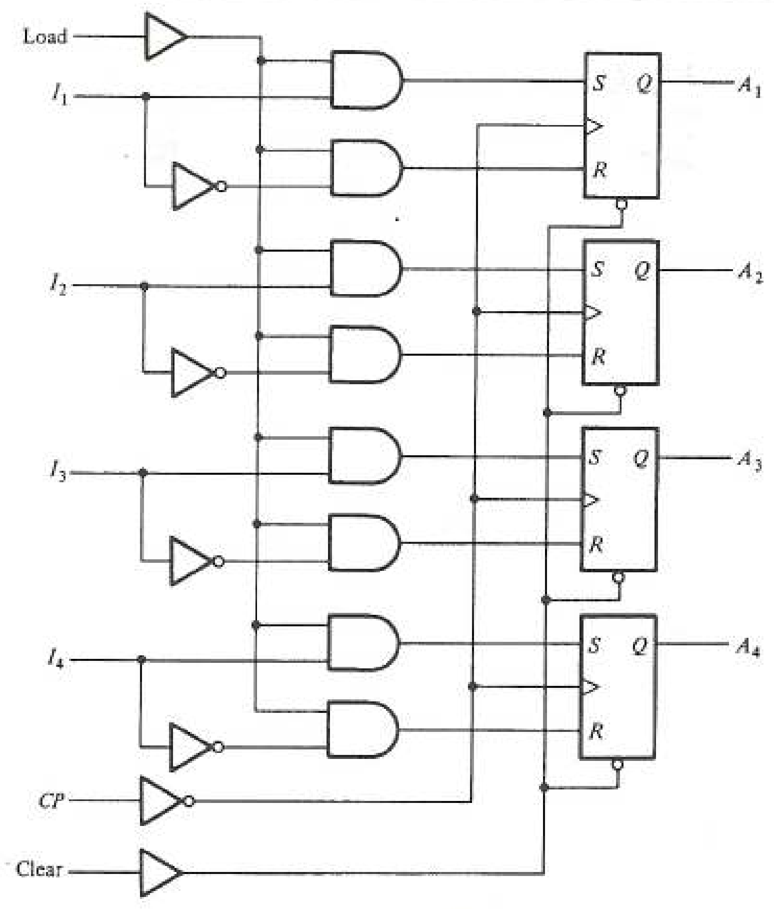
The simplest register is a register that consists of only flip-flops without any external gates. A 4-bit register with D FFs is shown below:



The binary data available at the four inputs are transferred in parallel into the 4-bit register when CP goes from 0 ~~to~~ 1 ($\uparrow$, on the rising edge). This data is retained at the output until the next rising edge of the clock.

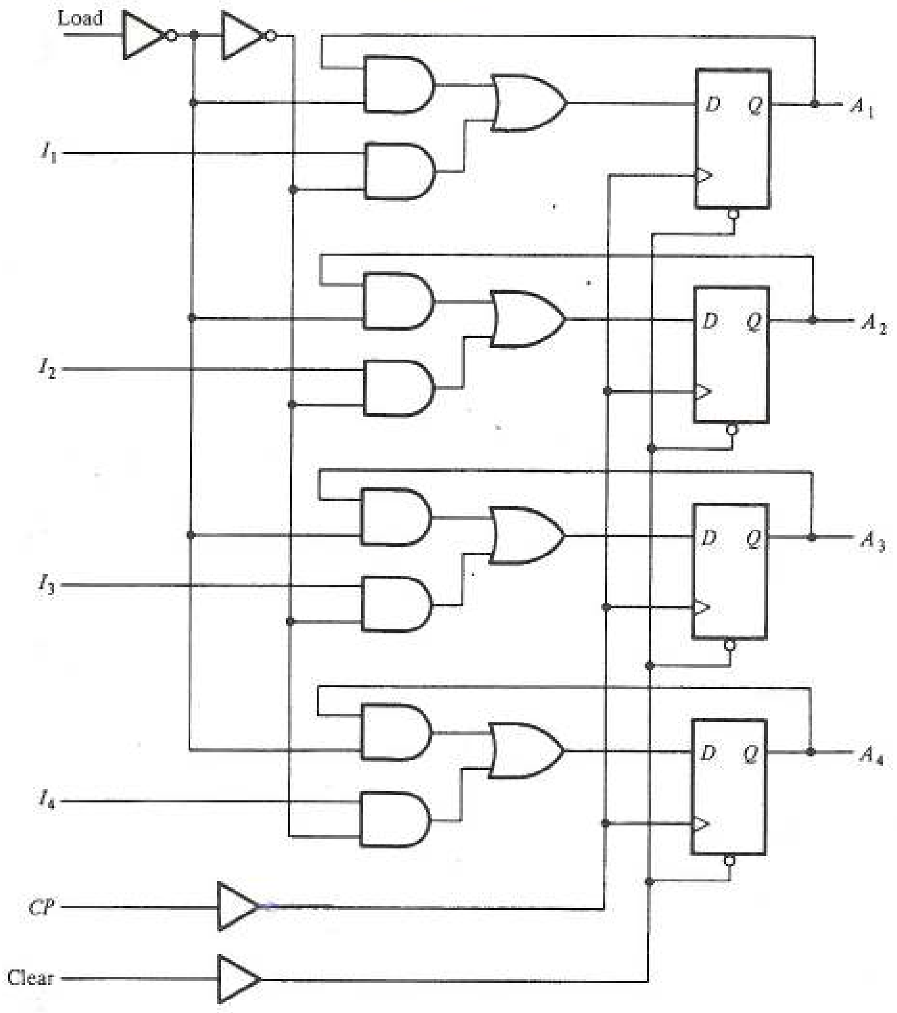
Register with Parallel Load:

- The transfer of new information into a register is referred to as <u>loading</u> the registers.
- A 4-bit register with a <u>load</u> control input using RS FFs is shown below:



- The <u>clear</u> input is an active-low input and used for clearing the register to all ∅'s prior to its clocked operation. Thus, the clear input must be maintained at 1 during normal clocked operations.
- The <u>load</u> input controls the operation of the register. When load is 1, the data on the four inputs is transferred into the register with the next clock pulse.
  When load is 0, both R and S are 0, and no change of state occurs with the next clock pulse.

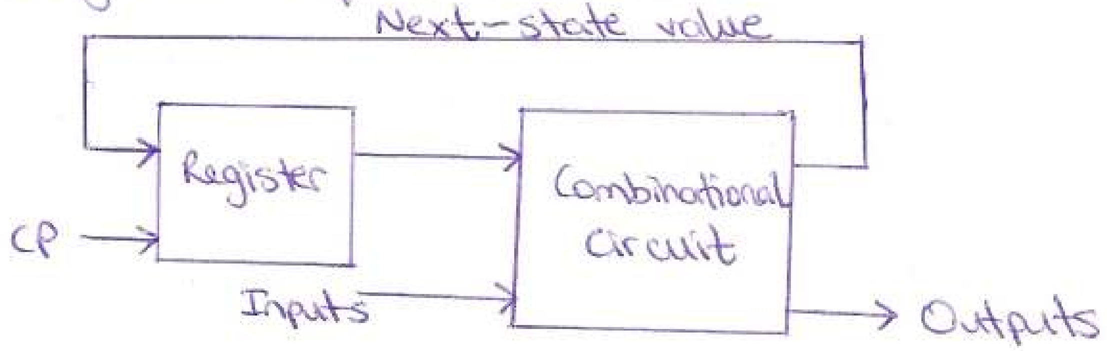
- A register with parallel load can be constructed with D FFs as shown below:



when Load is 1, the data on the four inputs is transferred with the next clock pulse.
when load is 0, the data inputs are blocked and the D flip-flops are reloaded with their present values, leaving the outputs unchanged
The feedback connection from output to input in each flip-flop is necessary because the D flip-flop does not have a "no change" input condition.

Sequential-Logic Implementation :

It is possible to employ registers instead of FFs in the design of sequential circuits.



– Block diagram of a sequential circuit –

Example: Design the sequential circuit whose state table is given below:

Present State		Input	Next State		Output
A_1	A_2	X	A_1	A_2	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

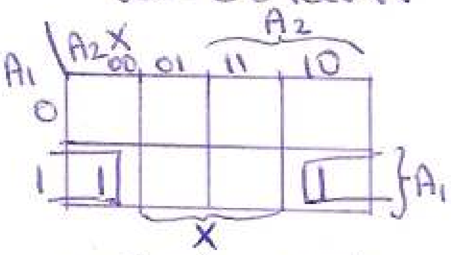
From the table we obtain:

$A_1(t+1) = \Sigma(4,6)$

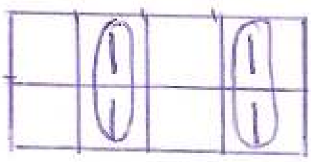
$A_2(t+1) = \Sigma(1,2,5,6)$

$y(A_1, A_2, X) = \Sigma(3,7)$

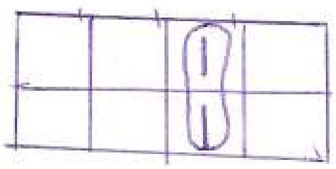
Simplifying these by map we obtain:



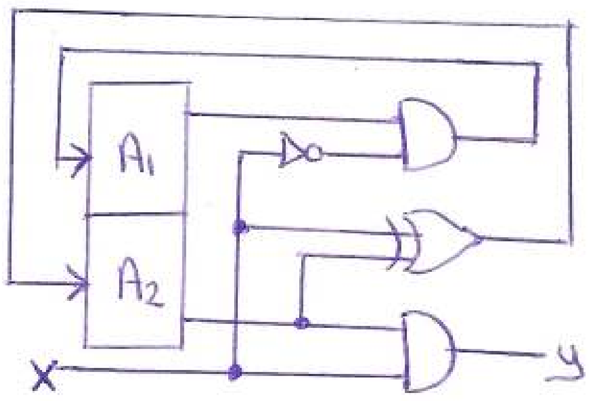
$A_1(t+1) = A_1X'$



$A_2(t+1) = A_2 \oplus X$



$y = A_2X$



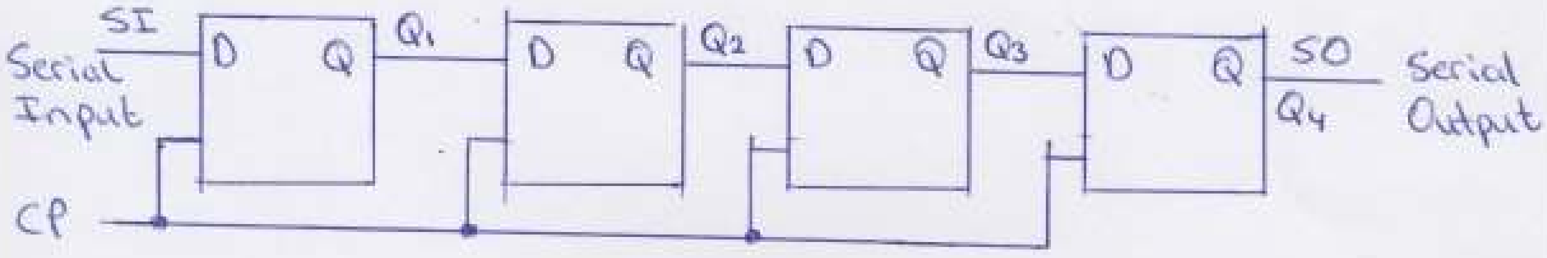
– Logic Diagram –

Shift Registers

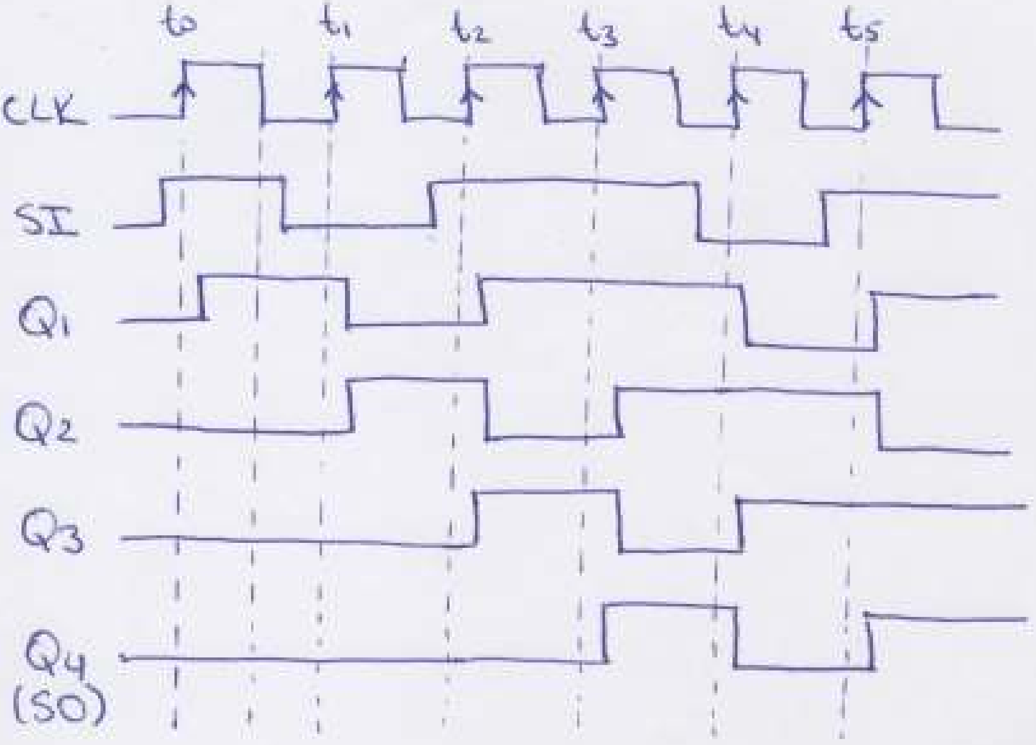
Transmitting all bits at once using n separate wires is called parallel transfer.

Transmitting all bits using a single wire, by performing the transfer one bit at a time in n consecutive clock cycles, is called serial transfer.

To do a serial transfer, a shift register is required. A shift register is a register capable of shifting its binary information either to the right or to the left.



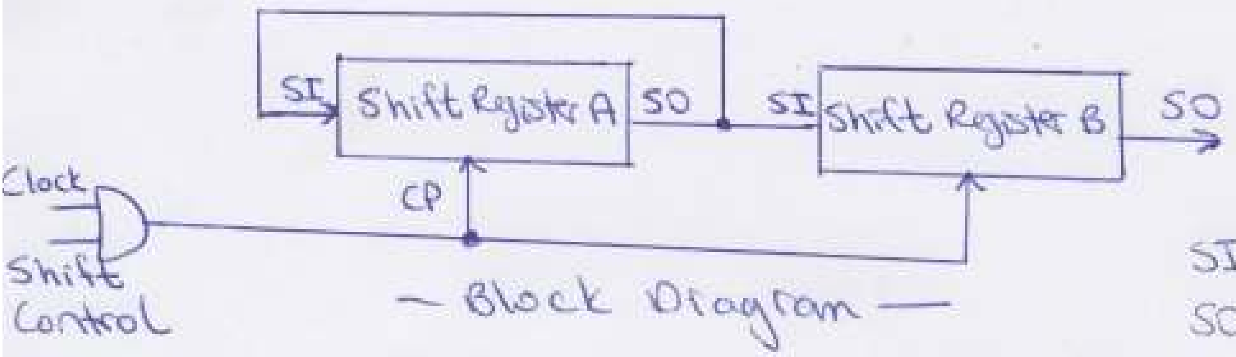
—4 bit shift register with D FFs—



	SI	Q1	Q2	Q3	Q4 (SO)
t0	1	0	0	0	0
t1	0	1	0	0	0
t2	1	0	1	0	0
t3	1	1	0	1	0
t4	0	1	1	0	1
t5	1	0	1	1	0

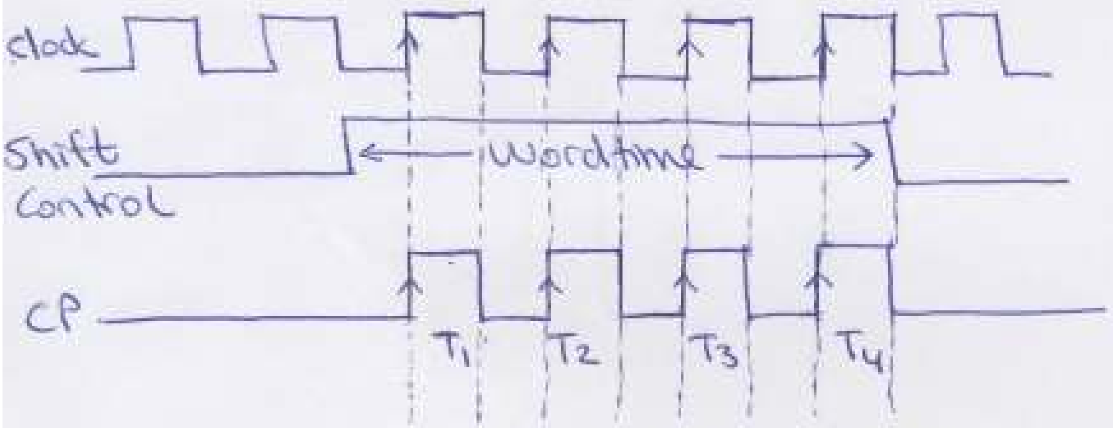
Serial Transfer:

The serial transfer of information from register A to register B is done with shift registers as shown below:



— Block Diagram —

SI → Serial Input
SO → Serial Output



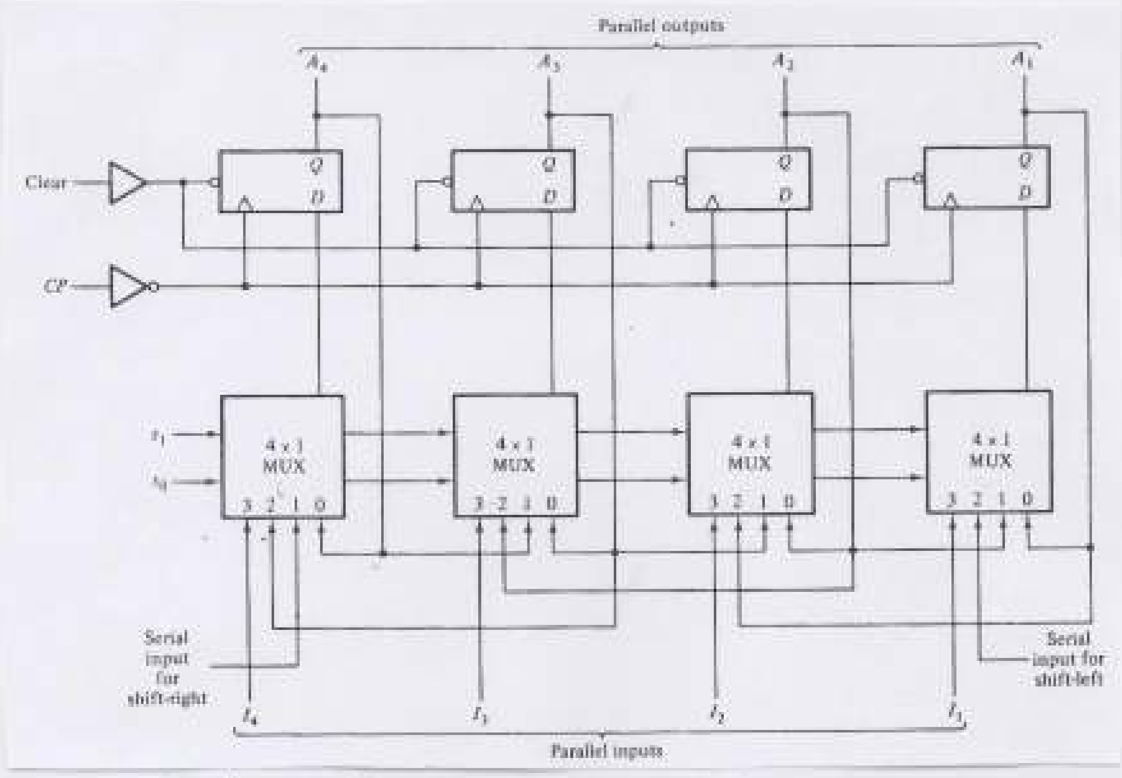
— Timing Diagram —

Timing Pulse	Shift Register A	Shift Register B
Initial Value	1 0 1 1	0 0 1 0
After 1	1 1 0 1	1 0 0 1
After 2	1 1 1 0	1 1 0 0
After 3	0 1 1 1	0 1 1 0
After 4	1 0 1 1	1 0 1 1

- The shift-control input determines when and how many times the registers are shifted.

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:



— 4-bit Bidirectional Shift Register with Parallel Load —

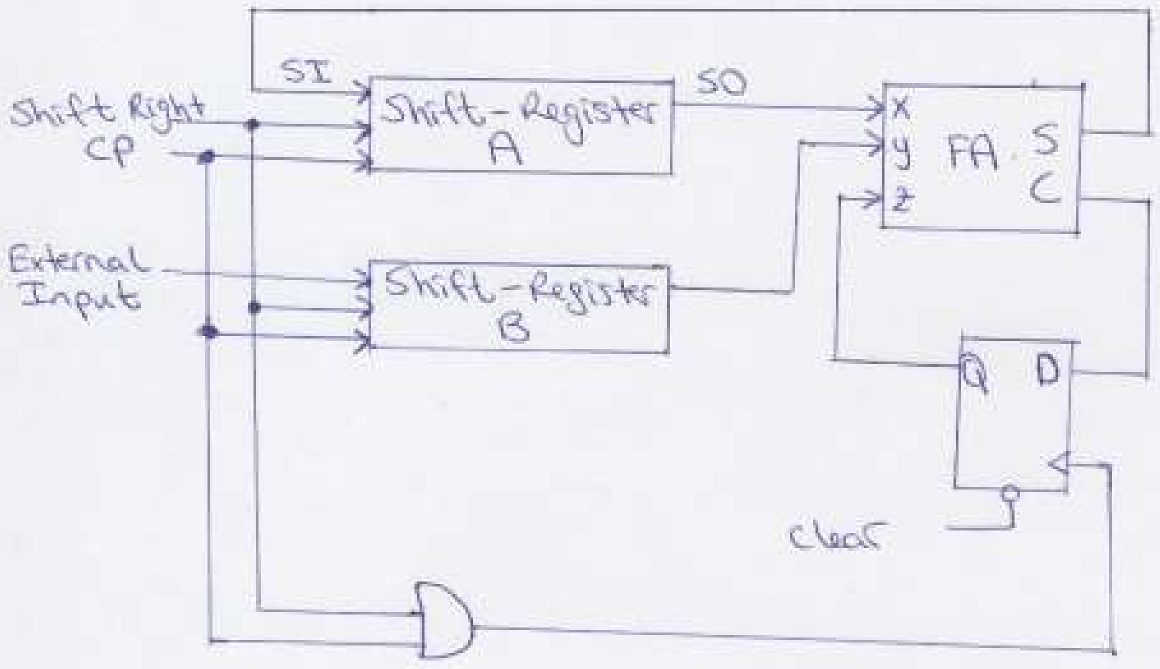
It has following properties:

1- A clear control to clear the register to ∅.
2- A CP input to synchronize all operations.
3- A shift-right control associated with serial input/output lines.
4- A shift-left control " " " " " " .
5- A parallel-load control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6- n-parallel output lines
7- A hold state control.

Mode Control

CLR	S_1	S_0	Register Operation
1	0	0	No change
1	0	1	Shift Right
1	1	0	Shift Left
1	1	1	[illegible]

Serial Addition:



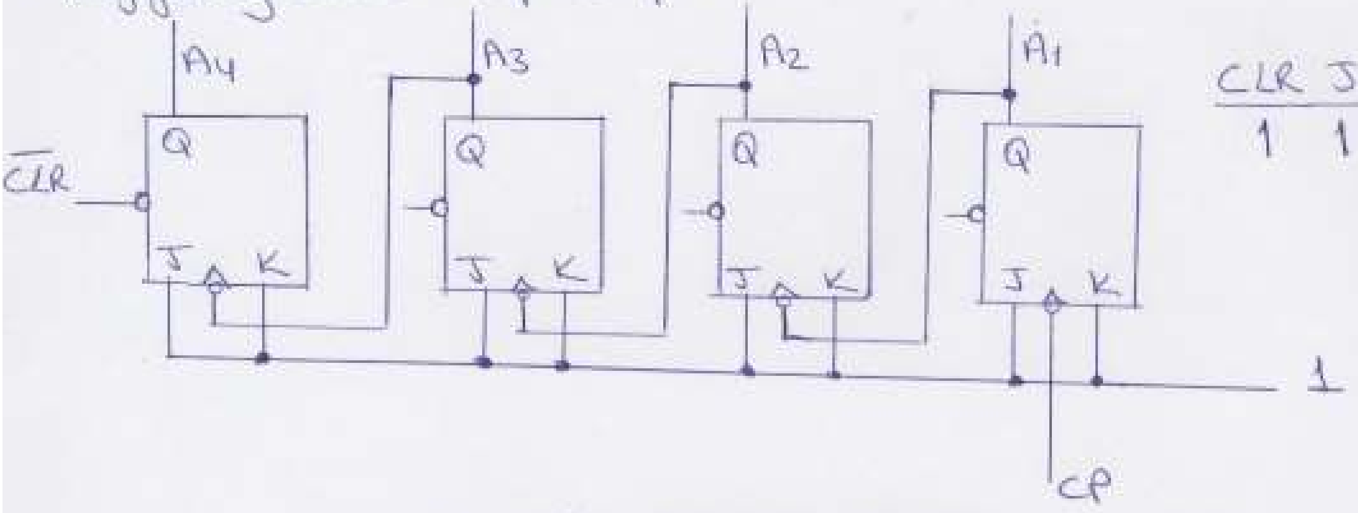
- The two numbers are stored in two shift registers.
- Bits are added, one pair at a time through a Full Adder.
- The two shift registers are shifted to the right for one word-time.
- Register A holds the sum.

A	B	x y z	SC
0101	0110	1 0 0	1 0
1010	0011	0 1 0	1 0
1101	0001	1 1 0	0 1
0110	0000	0 0 1	1 0
1011	0000	1 0 0	1 0

Counters

Ripple Counters :

The flip-flop output transition serves as a source for triggering other flip-flops.



CLR	J	K	Q(t+1)
1	1	1	Q'(t)

Toggle

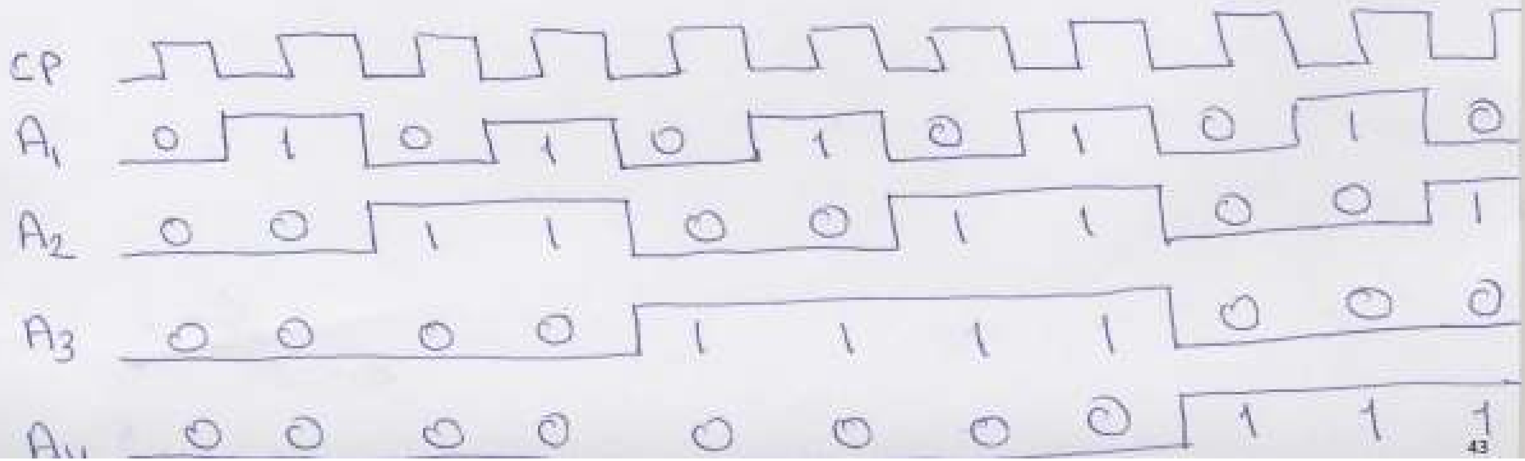
— 4-bit Binary Ripple Counter —
using JK FFs

Count Sequence

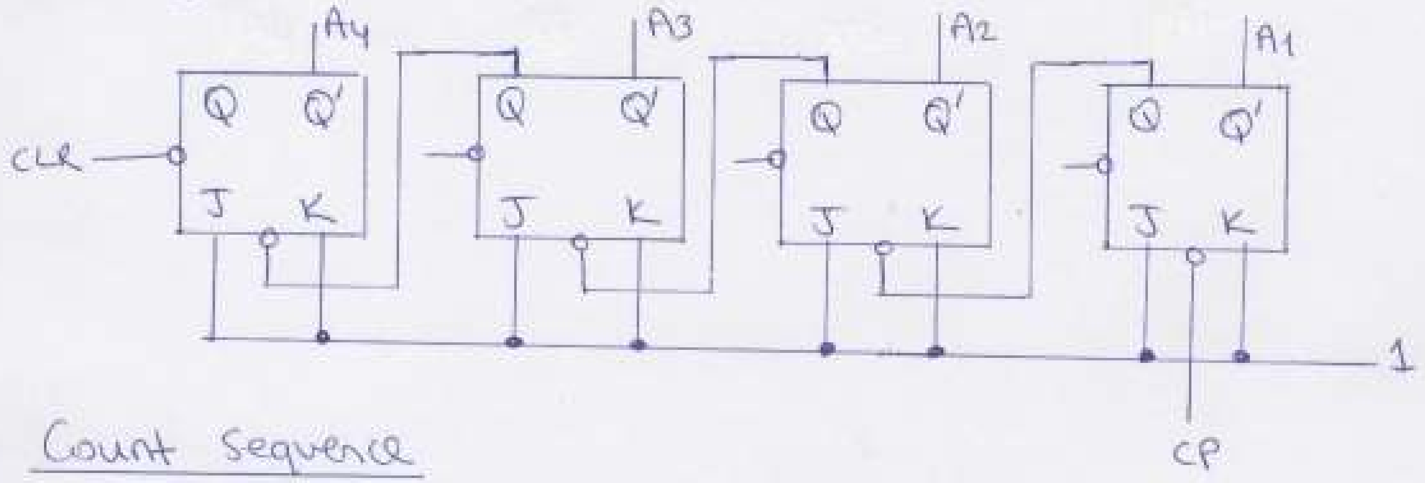
A_4	A_3	A_2	A_1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
	⋮		

- A_1 is complemented with each clock pulse
- Every time A_1 goes from 1 to 0, ↘, it complements A_2.
- Every time A_2 goes from 1 to 0, ↘, it complements A_3.
- Every time A_3 goes from 1 to 0, ↘, it complements A_4.

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



Binary Down Ripple Counter using JK FFs:



Count Sequence

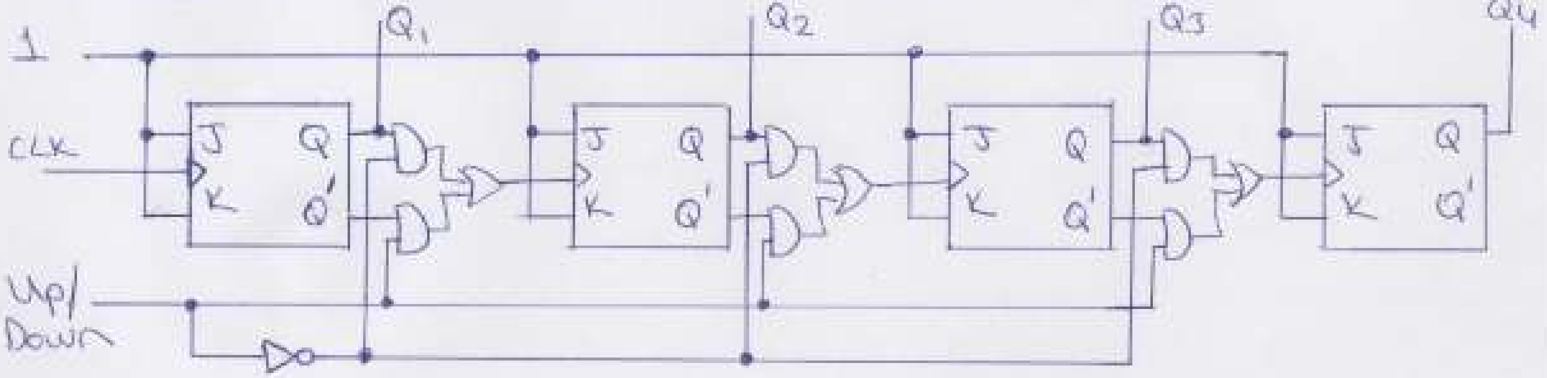
A_4	A_3	A_2	A_1
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
	⋮		

- A_1 is complemented with each clock pulse
- Every time A_1 goes from 0 to 1, ⎍, it complements A_2.
- Every time A_2 goes from 0 to 1, ⎍, it complements A_3

... so on.

* But if JK flip-flop has negative clock?

Binary Up/Down Ripple Counter using JK FFs:

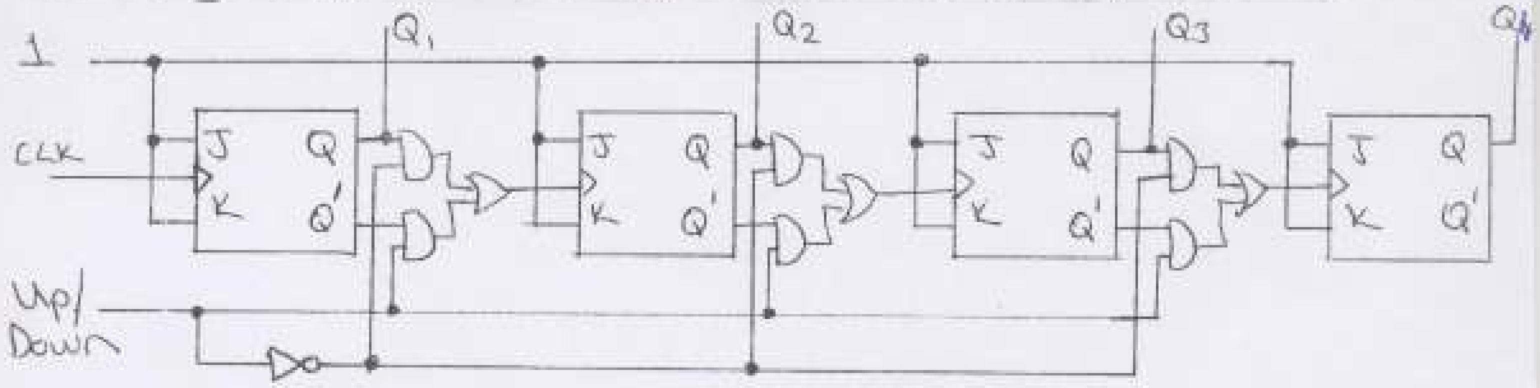


When Up/Down control is 0 ⟹ Down Counter

" " " " 1 ⟹ Up Counter

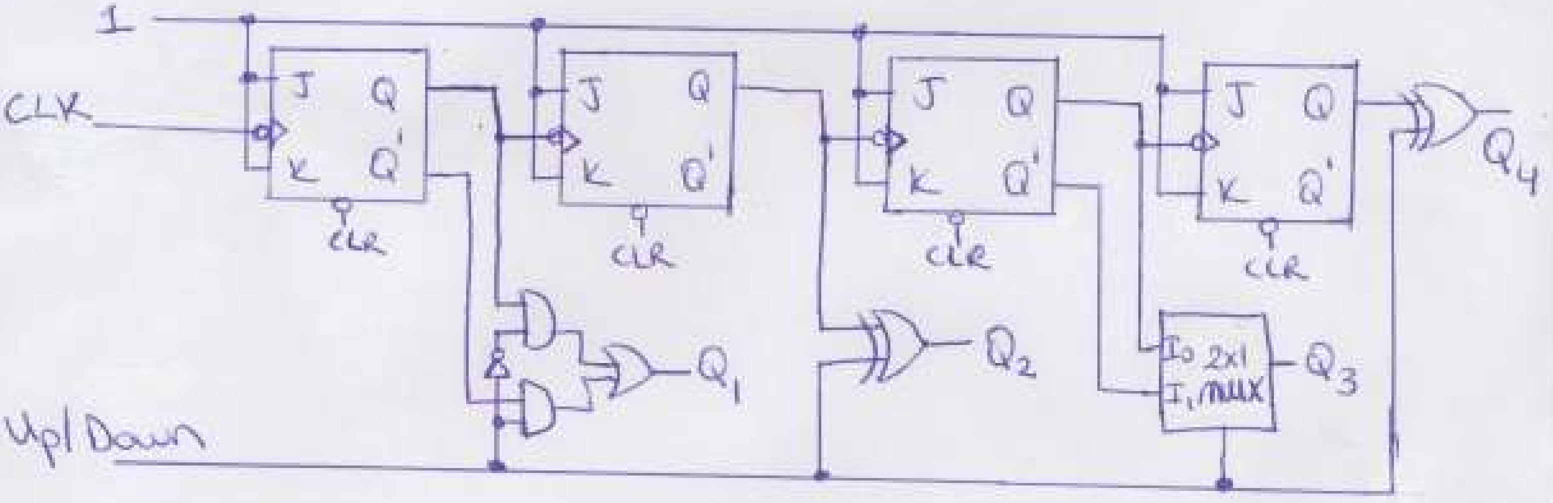
– Note that the FFs toggle when the clock inputs go from 0 to 1, ⎍, (i.e. positive-edge triggered).

Binary Up/Down Ripple Counter using JK FFs:



when Up/Down control is 0 ⇒ Down Counter
" " " " 1 ⇒ Up Counter

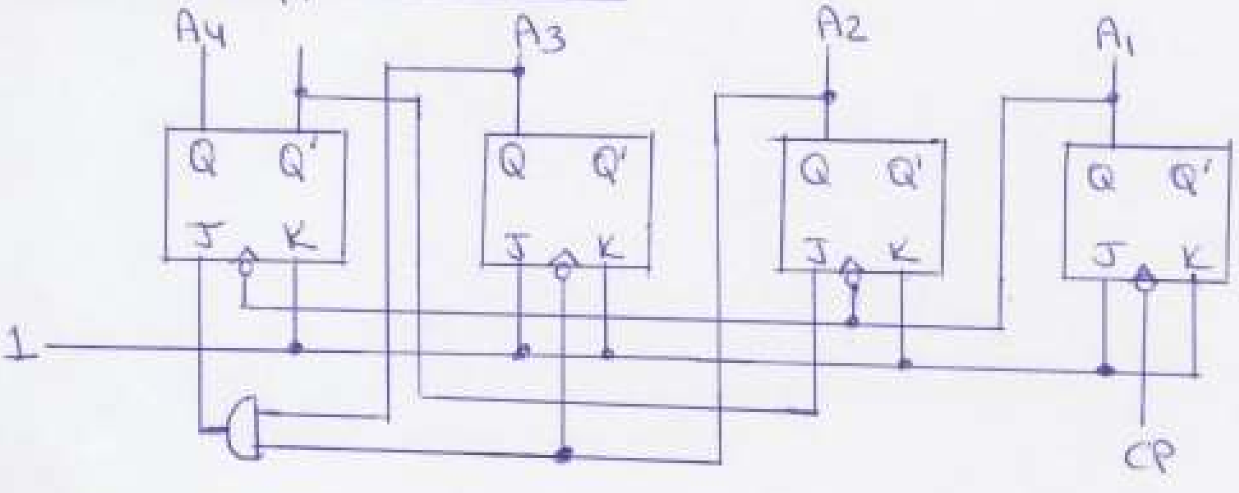
- Note that the FFs toggle when the clock inputs go from 0 to 1, ⎍, (i.e. positive-edge triggered).



when Up/Down control is 0 ⇒ Up Counter
" " " " 1 ⇒ Down Counter

- Note that the FFs toggle when the clock inputs go from 1 to 0, ⎍ (i.e. negative-edge triggered).

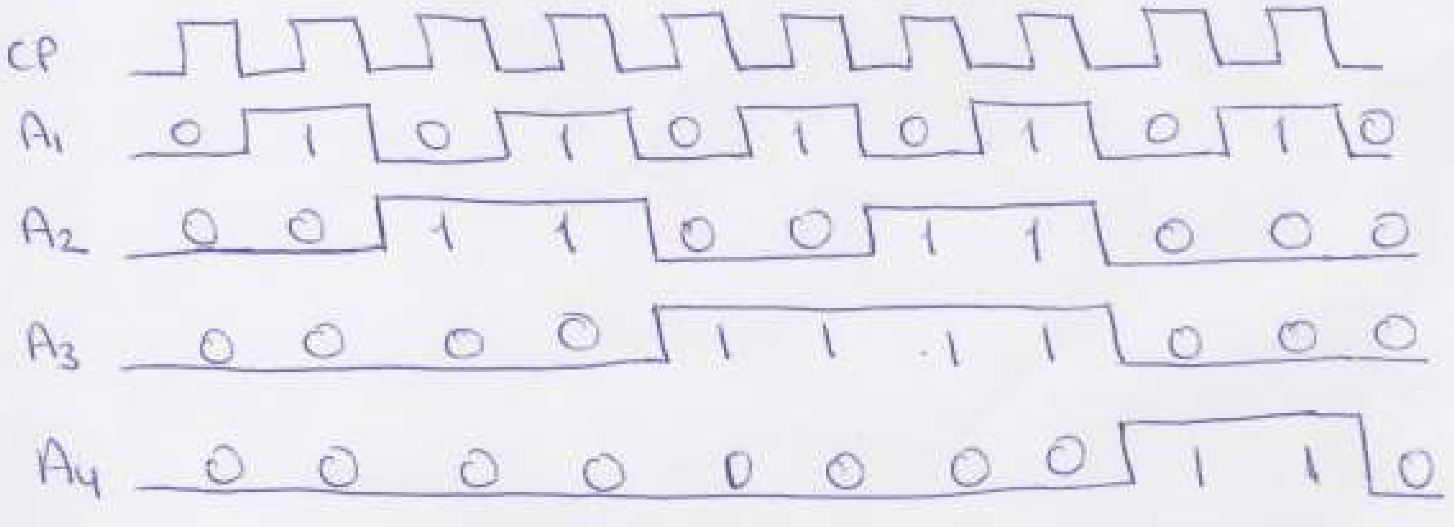
BCD Ripple Counter:



	A4	A3	A2	A1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
0	0	0	0	0

*Counts from 0 to 9 and returns to 0 after 9.

- A_1 is complemented on the negative edge of every clock pulse.
- A_2 is complemented if $A_4=0$ and A_1 goes from 1 to 0, Ƶ, and A_2 is cleared if $A_4=1$ and A_1 goes from 1 to 0.
- A_3 is complemented when A_2 goes from 1 to 0.
- A_4 is complemented when $A_3A_2=1$ and A_1 goes from 1 to 0. A_4 is cleared if either A_3 or A_2 is 0 and A_1 goes from 1 to 0.



— Timing Diagram —

Synchronous Counters

- The design procedure for any type of synchronous counter is the same as with other synchronous sequential circuits.
- In a synchronous counter, the common pulse triggers all the flip-flops simultaneously.

Binary Counter:

- The flip-flop in the lowest-order position is complemented with every clock pulse ($J=K=1$).
- A flip-flop in any other position is complemented with a clock pulse provided that all the lower order bits are equal to 1.

If count $=0 \Rightarrow$ all J,K inputs are zero $\Rightarrow$ No Change

If count $=1 \Rightarrow J_1=K_1=1$

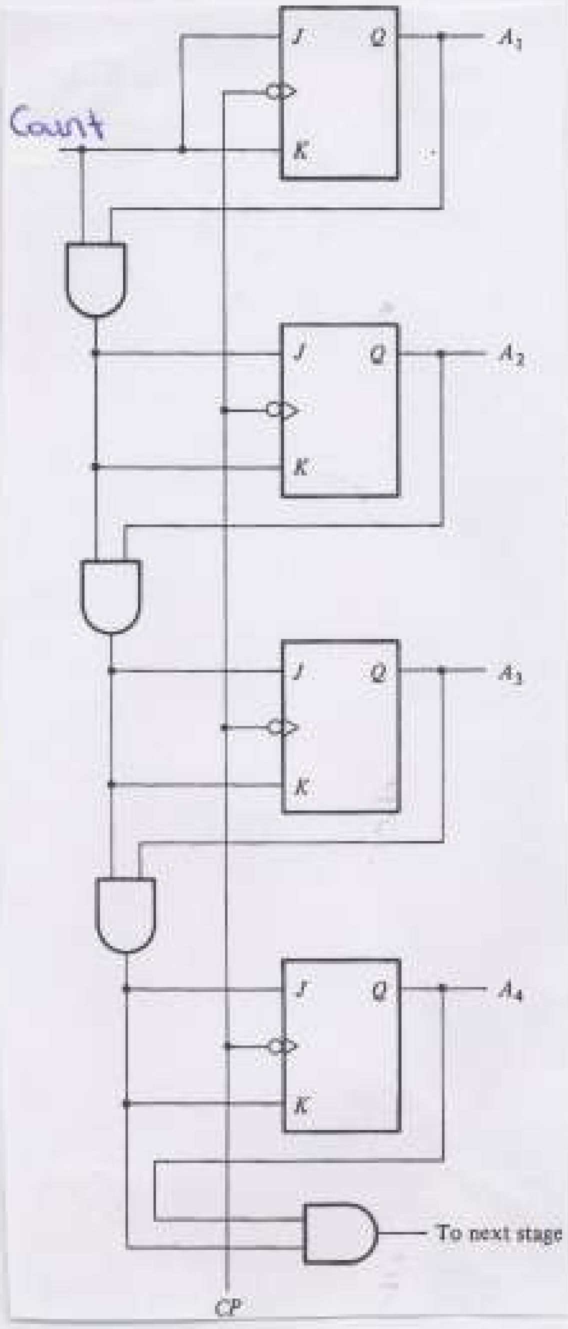
$J_2=K_2=A_1$

$J_3=K_3=A_1A_2$

$J_4=K_4=A_1A_2A_3$

$A_4A_3A_2A_1$
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Counting Sequence: 0, 1, ---, 15

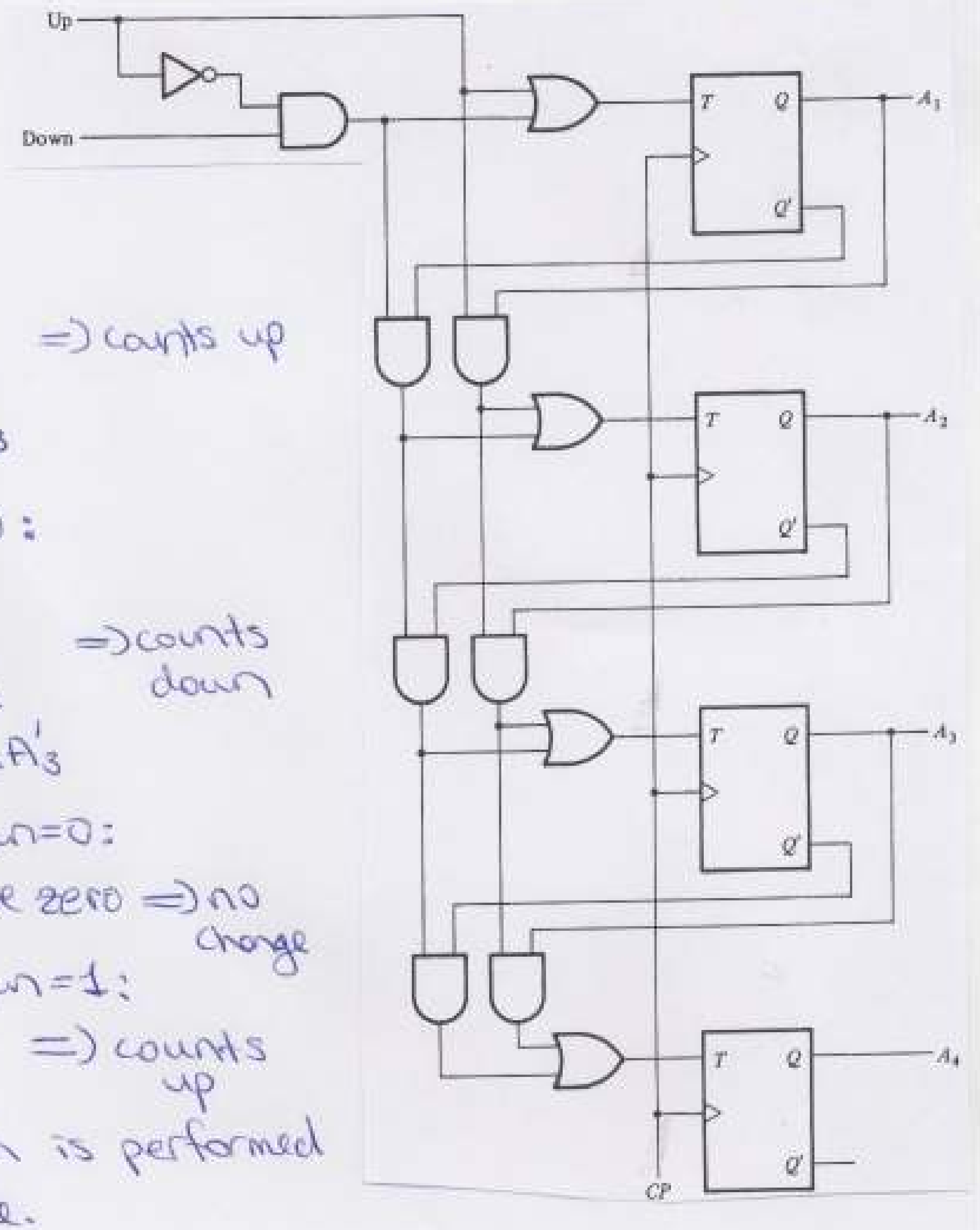


- A count-down binary counter can be constructed by taking the inputs to the AND gates from the complement outputs Q' (not from the normal outputs Q) of the previous flip-flops

Binary Up-Down Counter:

A binary counter capable of counting either up or down is shown below. The T flip-flops used in this circuit may be considered as JK flip-flops with the J and K inputs tied together.

T	Q(t+1)
0	Q(t)
1	Q'(t)



- If up=1 : $T_1 = 1$
  $T_2 = A_1$
  $T_3 = A_1A_2$ ⟹ counts up
  $T_4 = A_1A_2A_3$

- If down=1 and up=0 :
  $T_1 = 1$
  $T_2 = A_1'$ ⟹ counts down
  $T_3 = A_1'A_2'$
  $T_4 = A_1'A_2'A_3'$

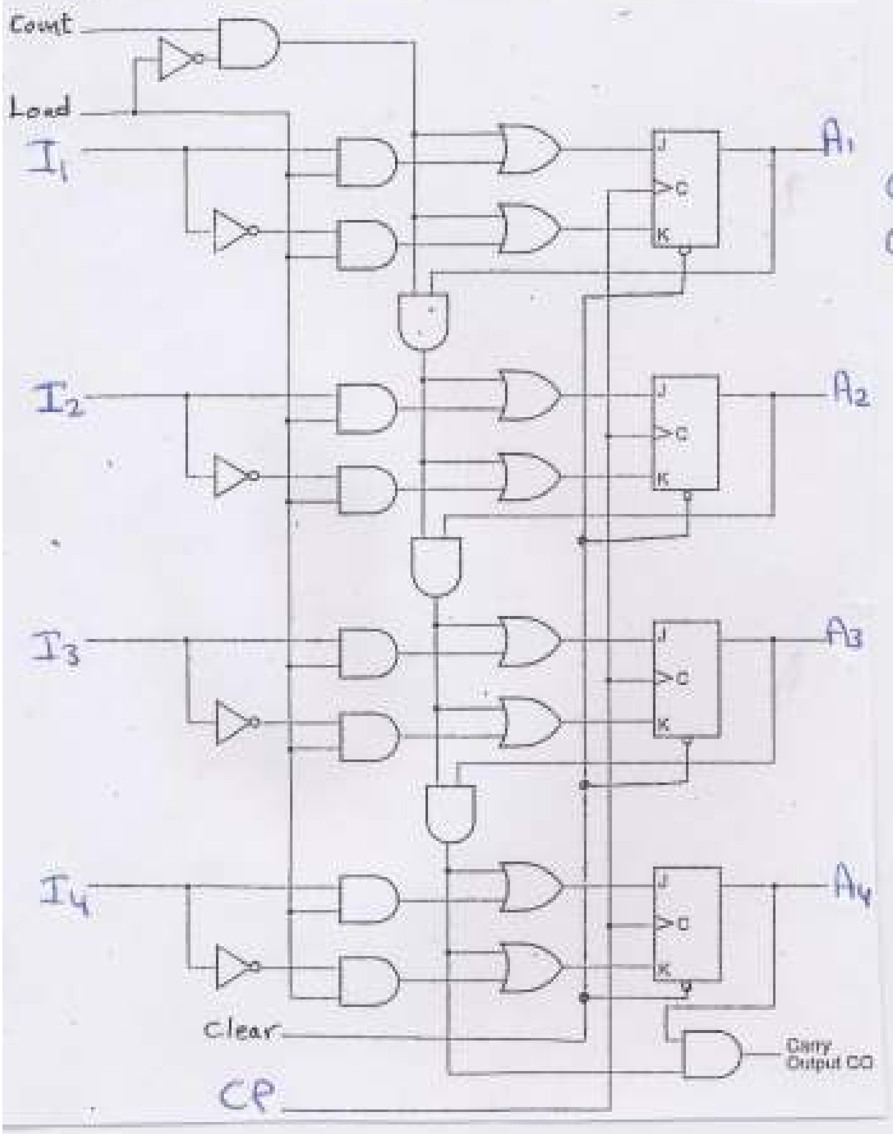
- If up=0 and down=0:
  all T inputs are zero ⟹ no change

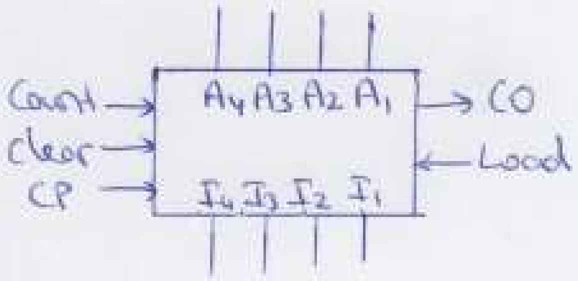
- If up=1 and down=1:
  ⟹ counts up

* only one operation is performed at any given time.

Binary Counter with Parallel Load:

A counter may require a parallel-load capability for transferring an initial binary number prior to the count operation. The logic diagram of a 4-bit binary synchronous counter with parallel load is given below:





Function table

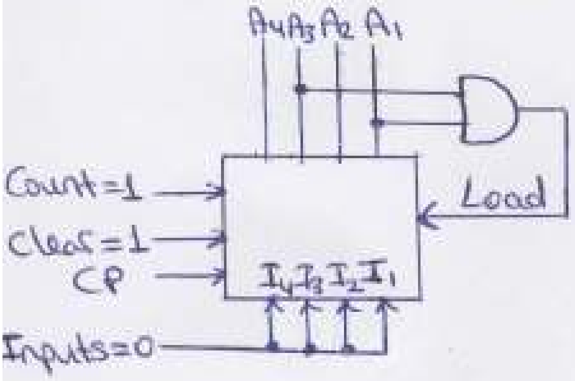
Clear	CP	Load	Count	Function
0	X	X	X	Clear to 0
1	X	0	0	No change
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state

The Carry Out (CO) can be used to expand the counter to more than four bits.

- A counter with parallel load can be used to generate any desired number of count sequences. A modulo-N (mod-N) counter is a counter that goes through a repeated sequence of N counts. Eg. a BCD counter is a mod-10 counter.
- A counter with parallel load can be used to construct any mod_N counter.

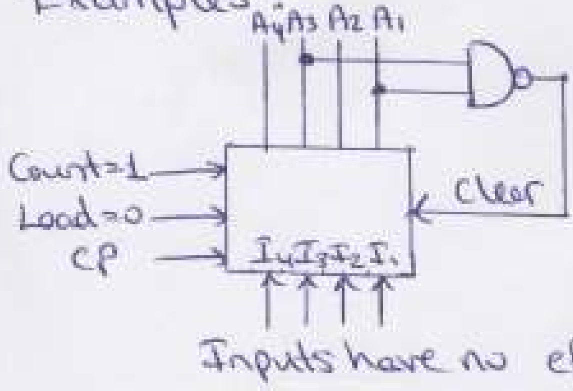
Example: Construct a mod-6 counter that counts 0, 1, 2, 3, 4, 5. Use a counter with parallel load.

4-bit

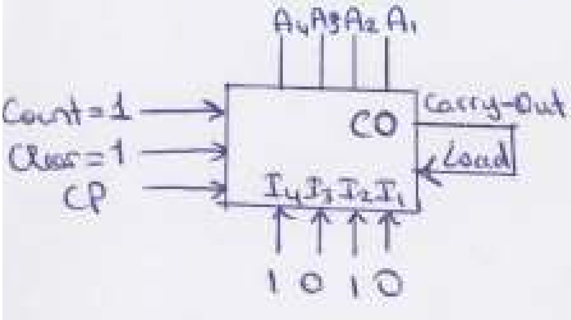


Binary sequence: $\underbrace{0,1,2,3,4,5}$,0,1,2..

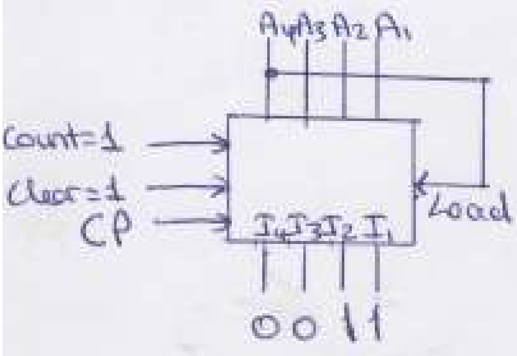
Examples:



Binary sequence: $\underbrace{0,1,2,3,4,5}$,0,1,..



Binary sequence: $\underbrace{10,11,12,13,14,15}$,10,11,..



Binary sequence: $\underbrace{3,4,5,6,7,8}$, 3,4,..