Computer Architecture

Dr. İlknur Dönmez

Sequential Logic

Combinatorial Logic

- So far we ignored the issue of time
- The inputs were just "there" – fixed and unchanging
- The output was just a function of the input
  - Not of anything that happened "previously"
- The output was computed "instantaneously"
- This is sometimes called "Combinatorial Logic"

Sequential Logic

Hello, Time

- Use the same hardware over time
  - Inputs change and outputs should follow
  - E.g.

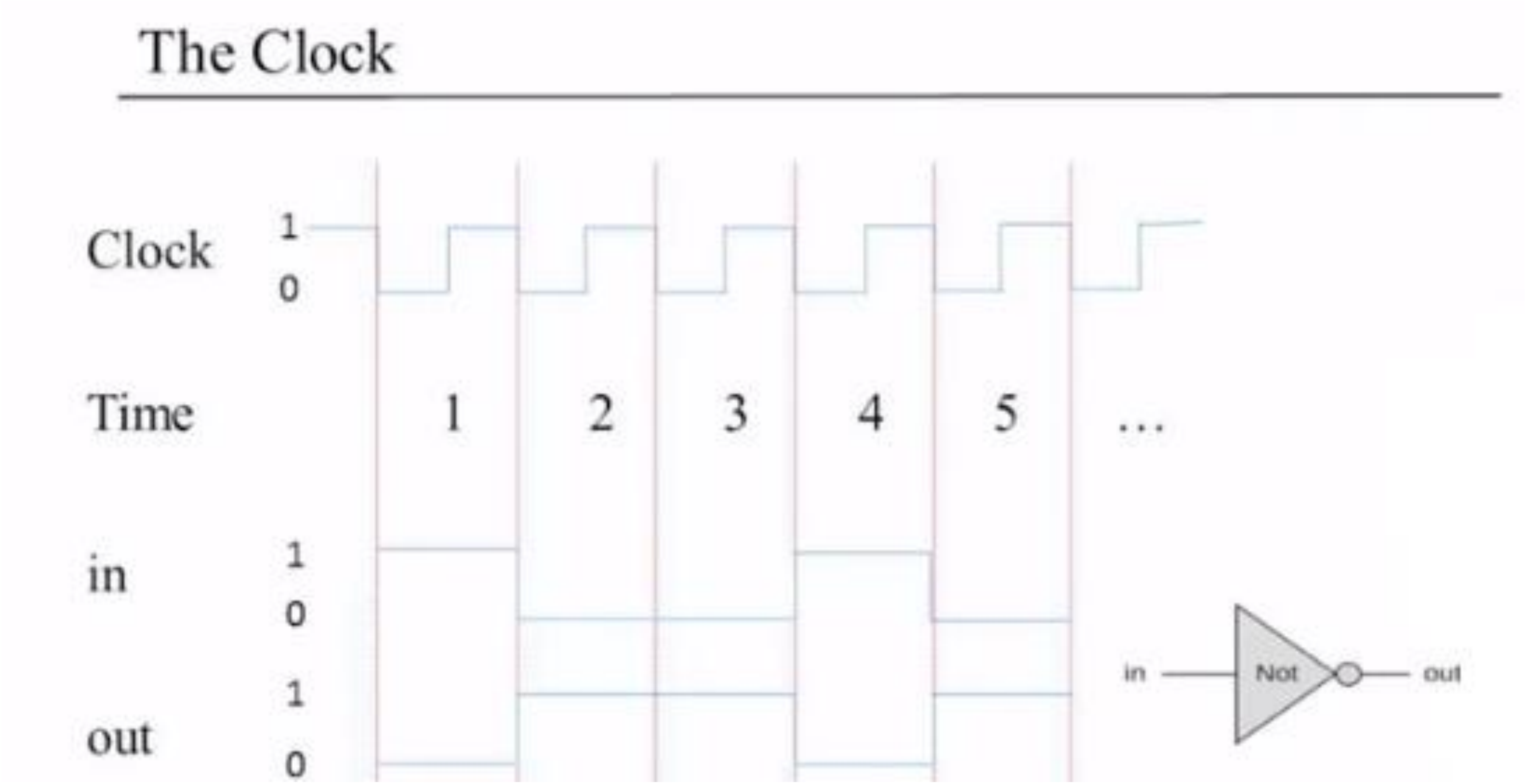
```
For i = 1 ... 100:
    a[i]=b[i]+c[i]
```

- Remember "State"
  - Memory
  - Counters
  - E.g.

```
For i = 1 ... 100:
    sum = sum + i
```

- Deal with speed

Sequential Logic



Combinatorial Logic vs. Sequential Logic

- Combinatorial: *out[t] = function(in[t])*

- Sequential: *out[t] = function(in[t-1])*

Time	1	2	3	4	…
in	a	b	c	d	
	↓	↓	↓	↓	
out	f(a)	f(b)	f(c)	f(d)	

Combinatorial Logic vs. Sequential Logic

- Combinatorial: *out[t] = function(in[t])*

- Sequential: *out[t] = function(in[t-1])*

Time	1	2	3	4	…
in	a	b	c	d	
out		f(a)	f(b)	f(c)	

Combinatorial Logic vs. Sequential Logic

- Combinatorial: *out[t] = function(in[t])*

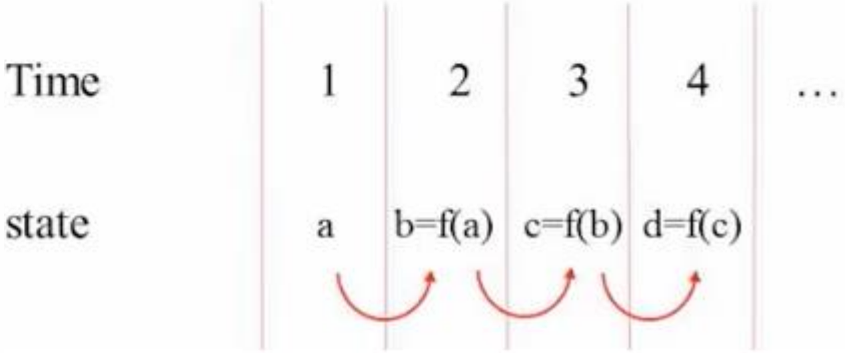
- Sequential: *state[t] = function(state[t-1])*

Time	1	2	3	4	…
state	a	b=f(a)	c=f(b)	d=f(c)	

Remembering State

- Missing ingredient: remember one bit of information from time *t-1* so it can be used at time *t*.
- At the "end of time" *t-1*, such an ingredient can be at either of two states: "remembering 0" or "remembering 1".
- This ingredient remembers by "flipping" between these possible states.
- Gates that can flip between two states are called Flip-Flops.

state[t] = function(state[t-1])



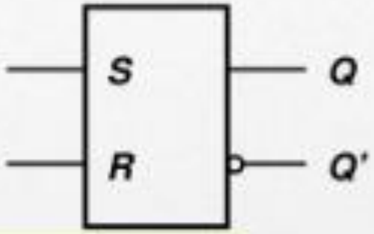
Flip-Flop

- "Flip-flop" is the common name given to two-state devices **which offer basic memory for sequential logic operations**.
- Flip-flops are heavily used for digital data storage and transfer and are commonly used in banks called "registers" for the **storage of binary numerical data.**

Flip-Flop

- Flip-flop are basic storage/memory elements.
- Flip-flop are essentially 1-bit storage devices.
- Types of flip-flops are:
    - **1. SR Flip-flop**
    - **2. JK Flip-flop**
    - **3. D Flip-flop**
    - **4. T Flip-flop**
- Application of flip-flop:
    - 1. Counter
    - 2. Register
    - 3. Memory
    - 4. Logic controller
    - 5. Frequency Divider

SR Flip-Flop



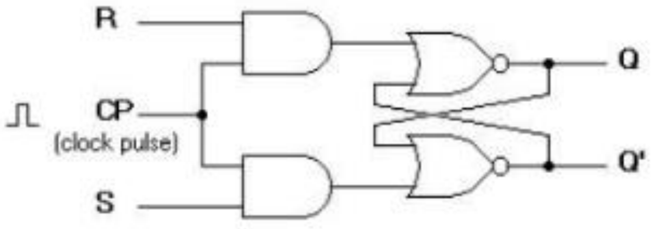
- The **simplest binary storage** device.
- SR Flip-flop have **2 inputs (SET & RESET)** and **2 outputs (Q & Q').**

  NOTE: **Q & Q' are compliments** of each other

- The SR flip flop is sometimes referred to as an **SR latch**. The Term latch refers to its use as a **temporary memory storage device**.

Flip Flops- SR Flip flop (set reset)

- If S and R is 0, Qnext=Q ; Q(t+1)=Q(t)



(a) Logic diagram

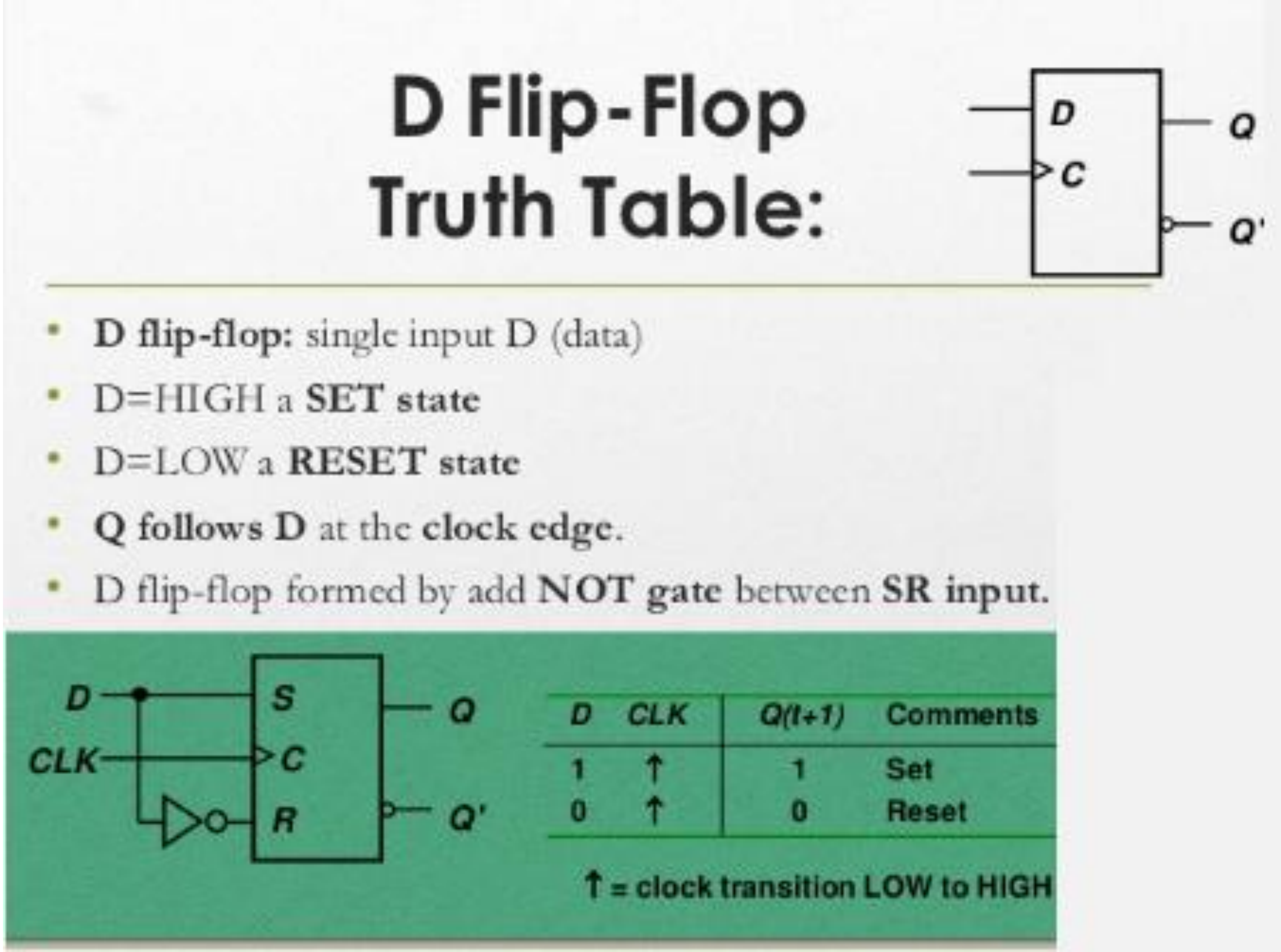
Q S R	Q(t+1)
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	indeterminate
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	indeterminate

(b) Truth table

Clocked SR flip-flop

S	R	Q	Q_{next}	Q'_{next}
0	0	0	0	1
0	0	1	1	0
0	1	x	0	1
1	0	x	1	0
1	1	x	x	x

D-flip flop


D Flip-Flop
Truth Table:
D
C
Q
Q'
- **D flip-flop:** single input D (data)
- D=HIGH a **SET state**
- D=LOW a **RESET state**
- **Q follows D** at the **clock edge**.
- D flip-flop formed by add **NOT gate** between **SR input.**
D
CLK
S
C
R
Q
Q'

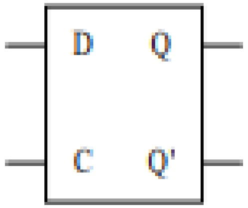
D	CLK	Q(t+1)	Comments
1	↑	1	Set
0	↑	0	Reset

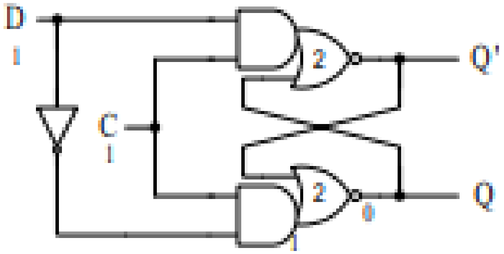
↑ = clock transition LOW to HIGH

Gated D- flip flop

- when the C input is 0, the Q output remanins same.
- The latch "closes", the Q output reta'ns 'ts last value and no longer change 'n response to D

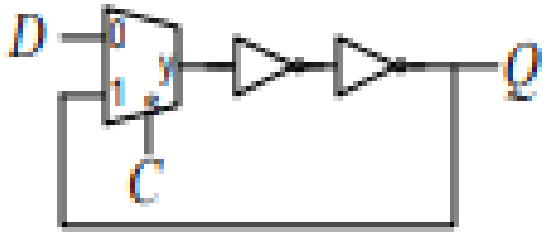
- when the C input is 1, the Q output follows the D input.
- At this stage, the latch is said to be "open" and the path from D input to Q output is "transparent"



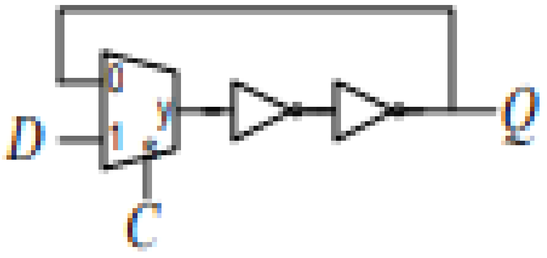


C	D	Q	Q_{next}
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

Gated D latch can also be implemented using a multiplexer.

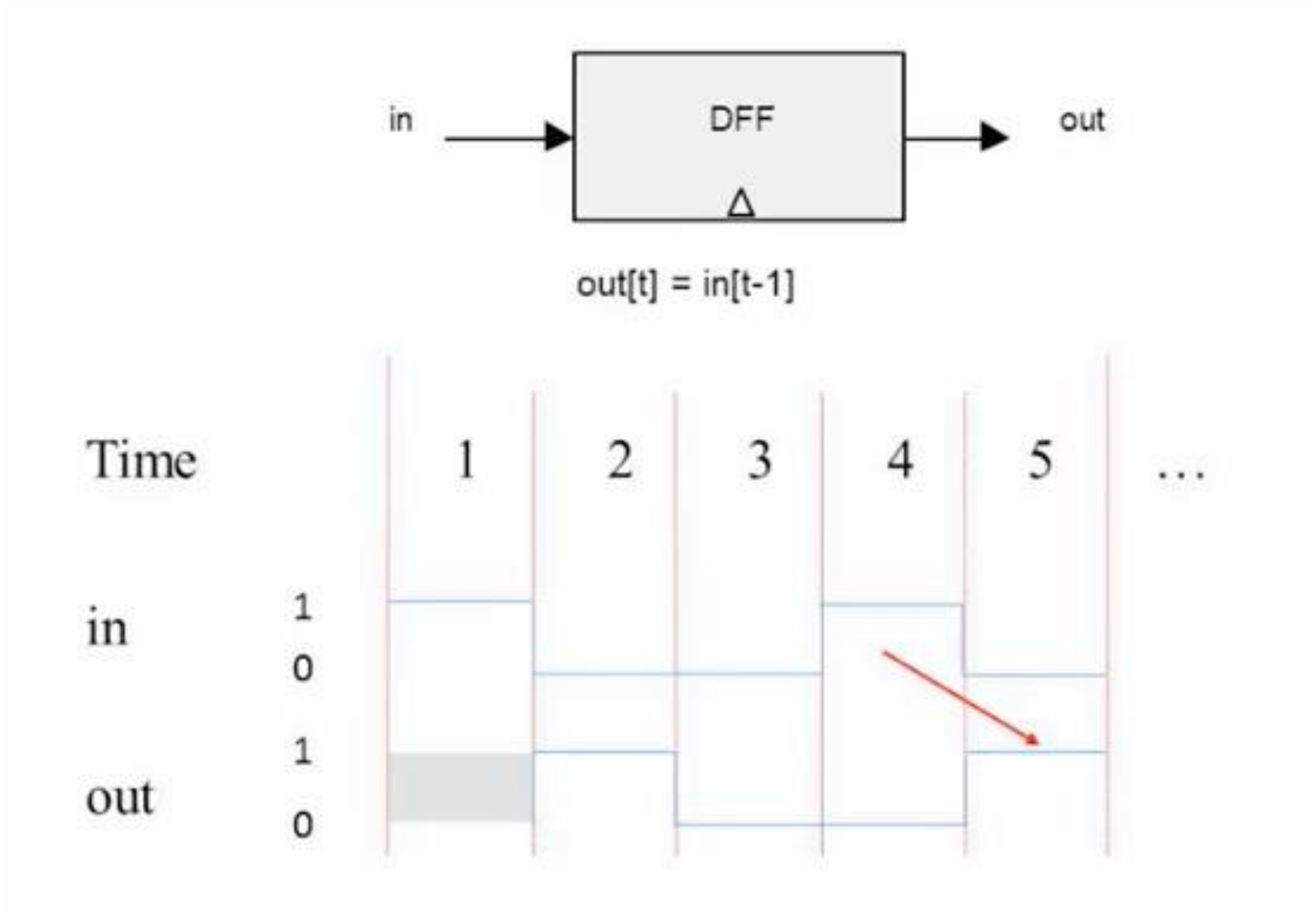


negative latch
D passes to Q when C=0

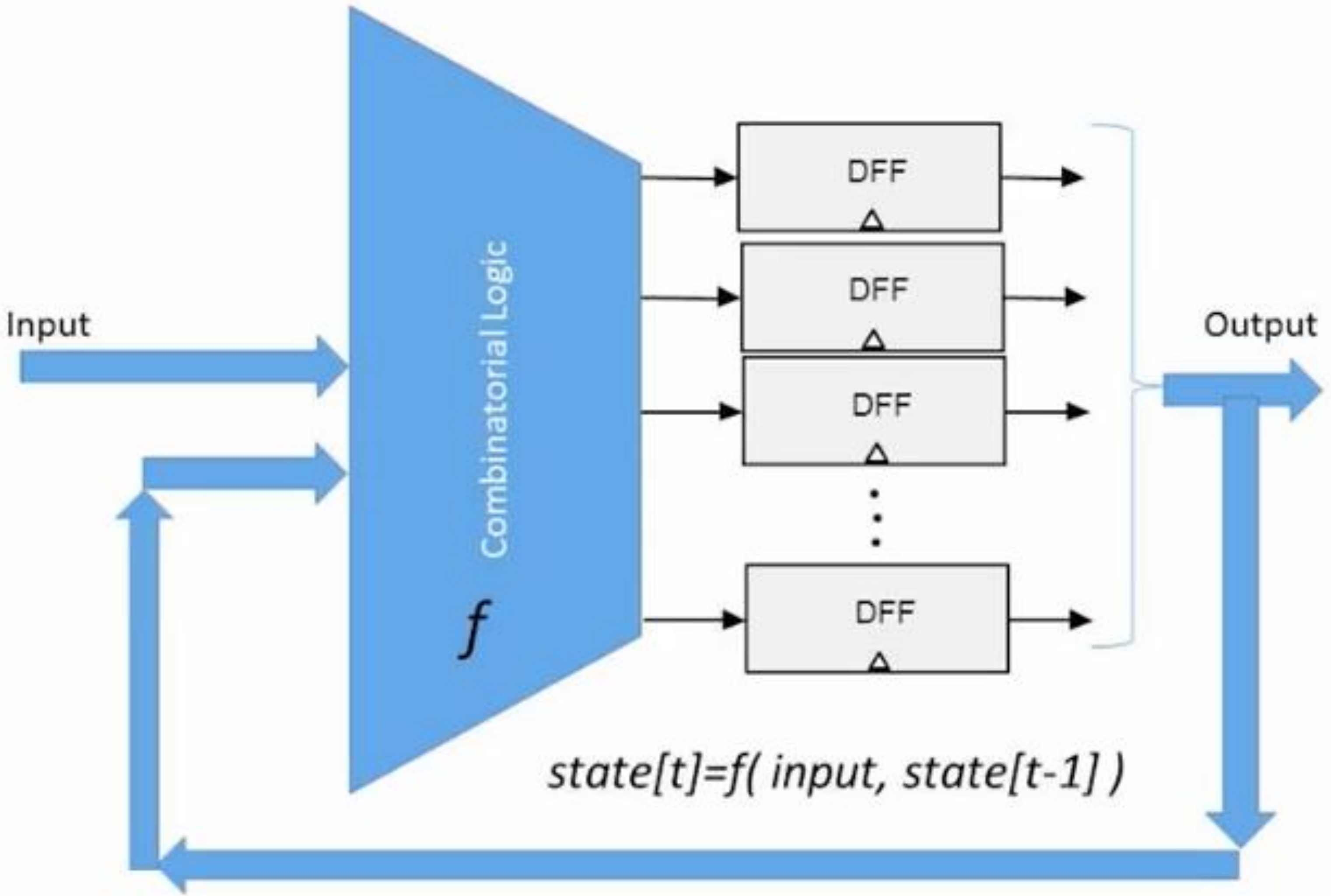


positive latch
D passes to Q when C=1

Flip Flops

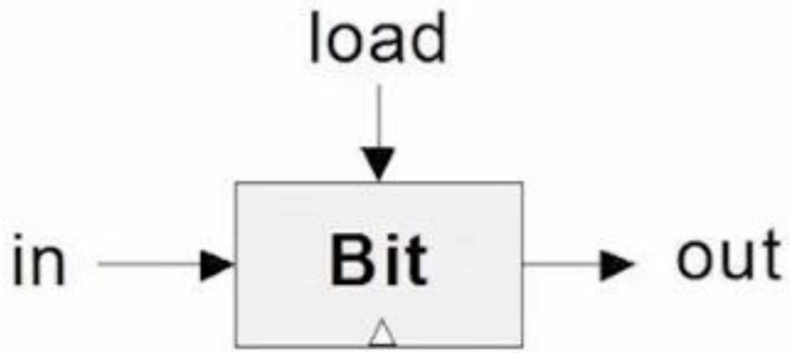


Sequential Logic Implementation



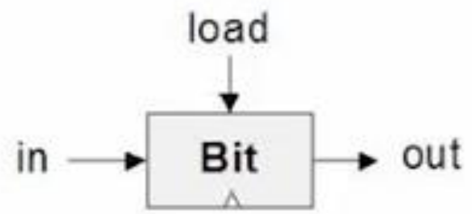
Remembering For Ever: 1-bit Register

- Goal: remember an input bit "forever": until requested to load a new value.

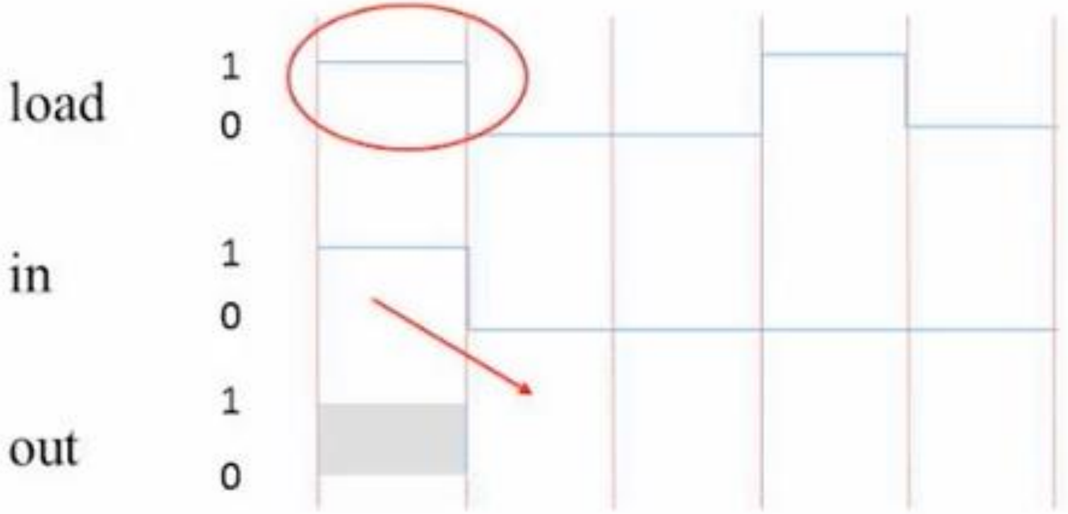


if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

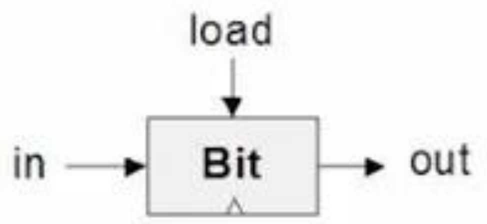
1-Bit Register



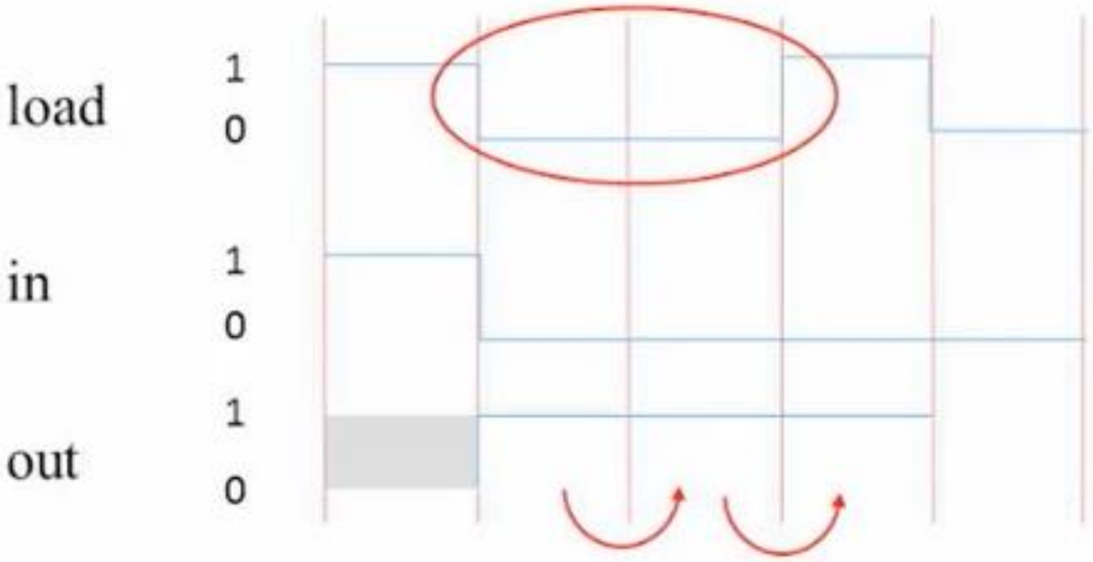
if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)



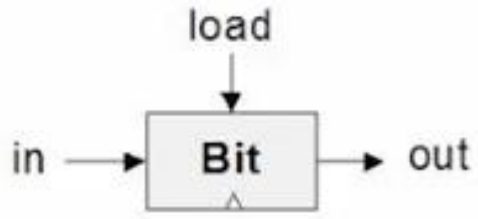
1-Bit Register



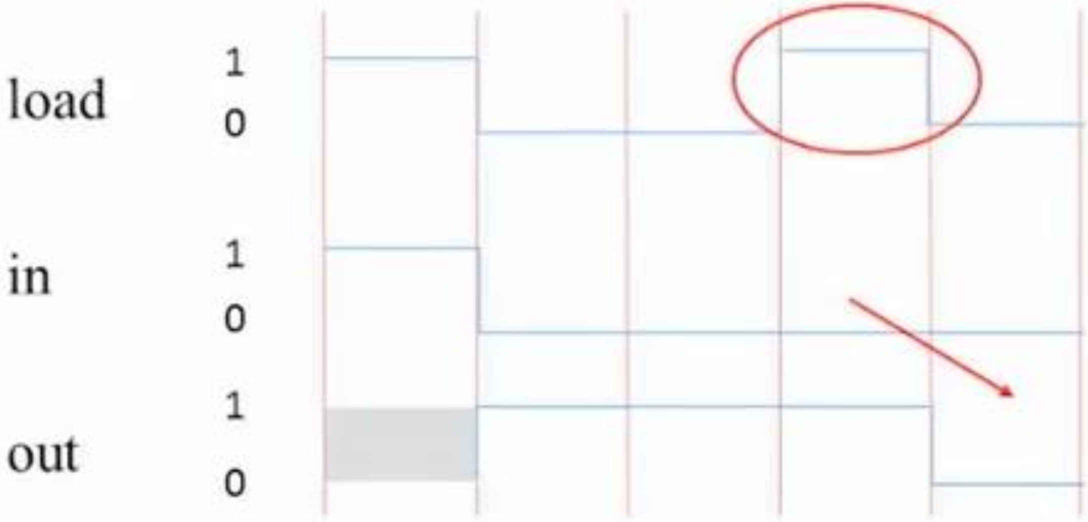
if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

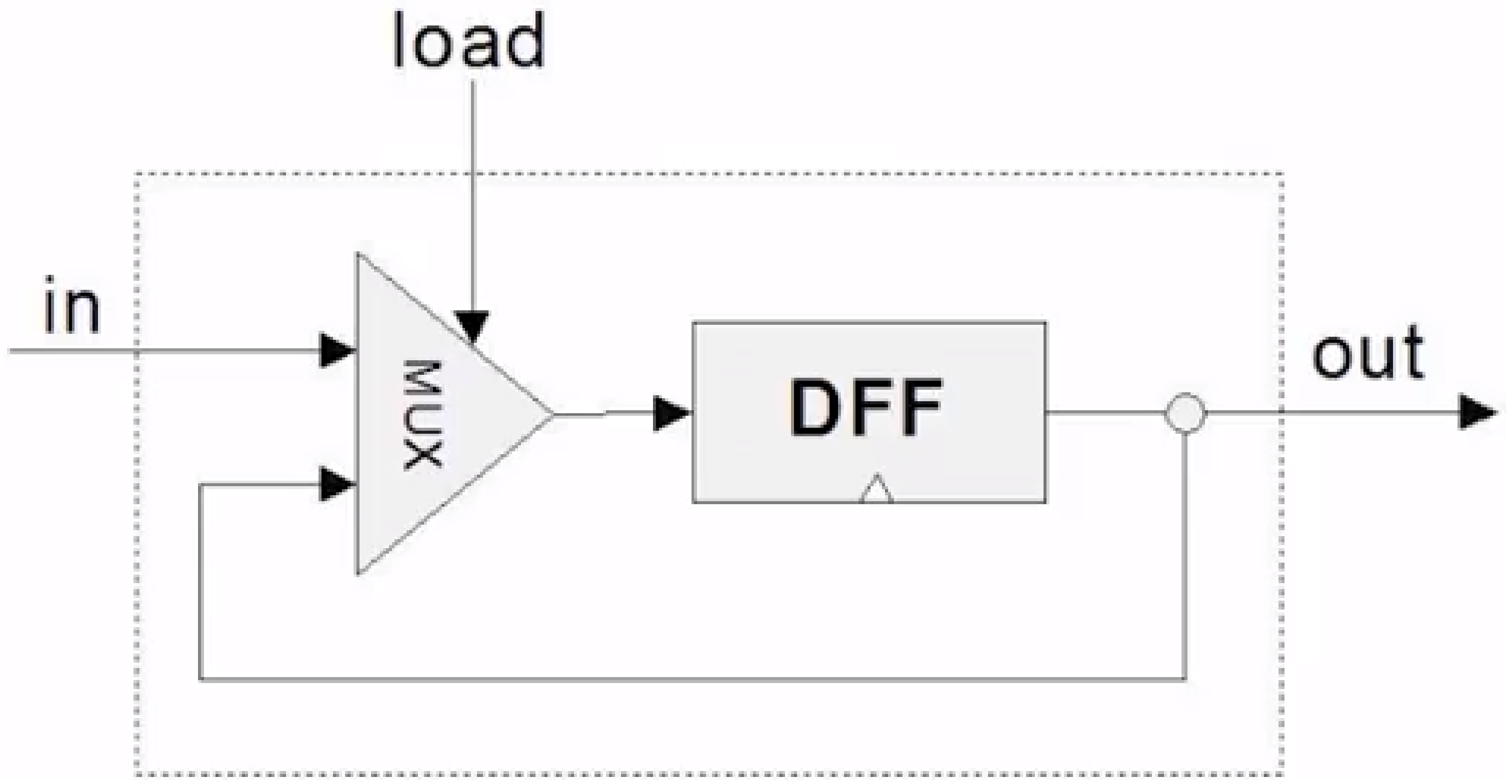


1-Bit Register

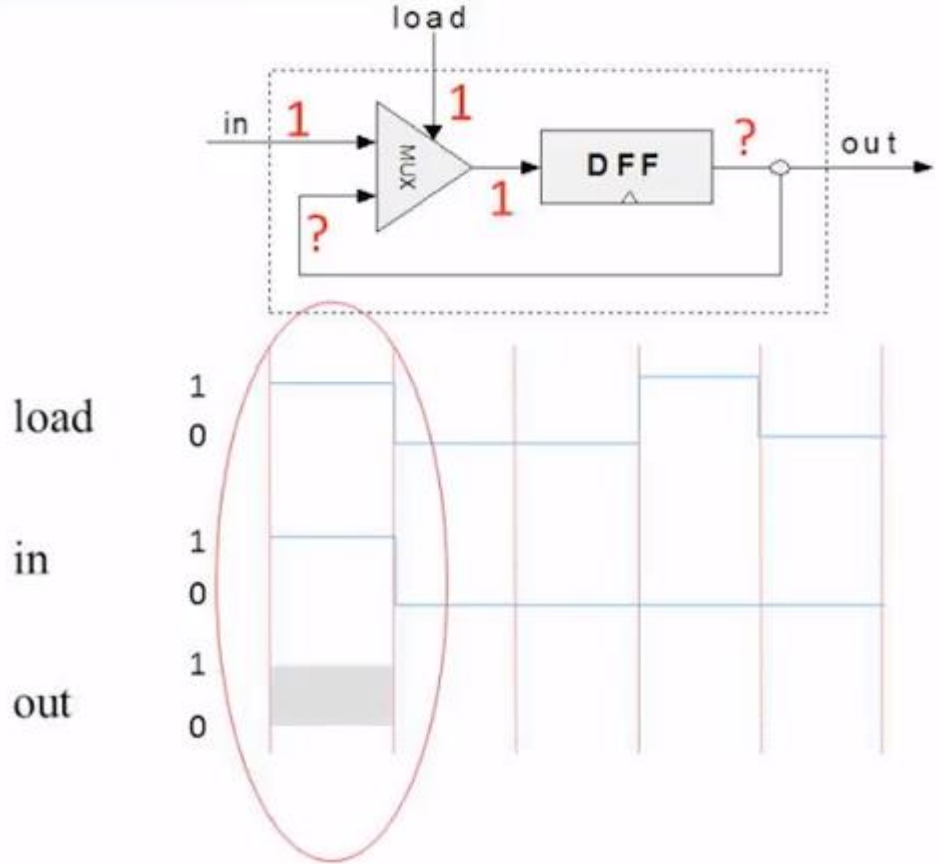


if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

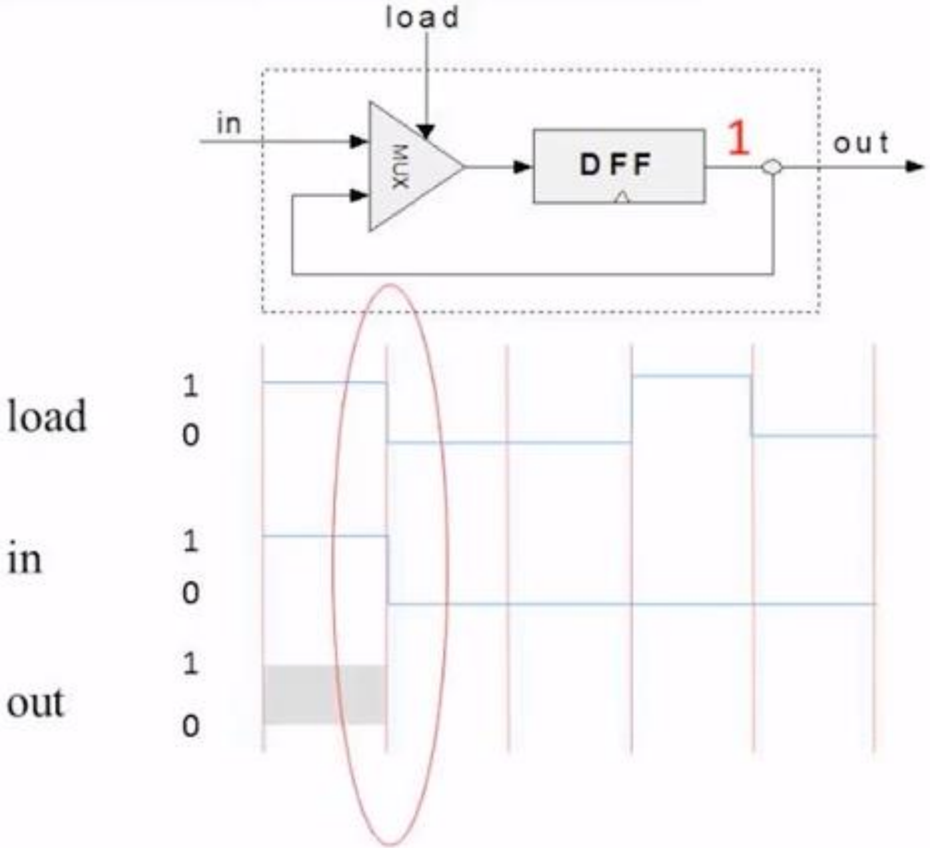



load
in
MUX
DFF
out

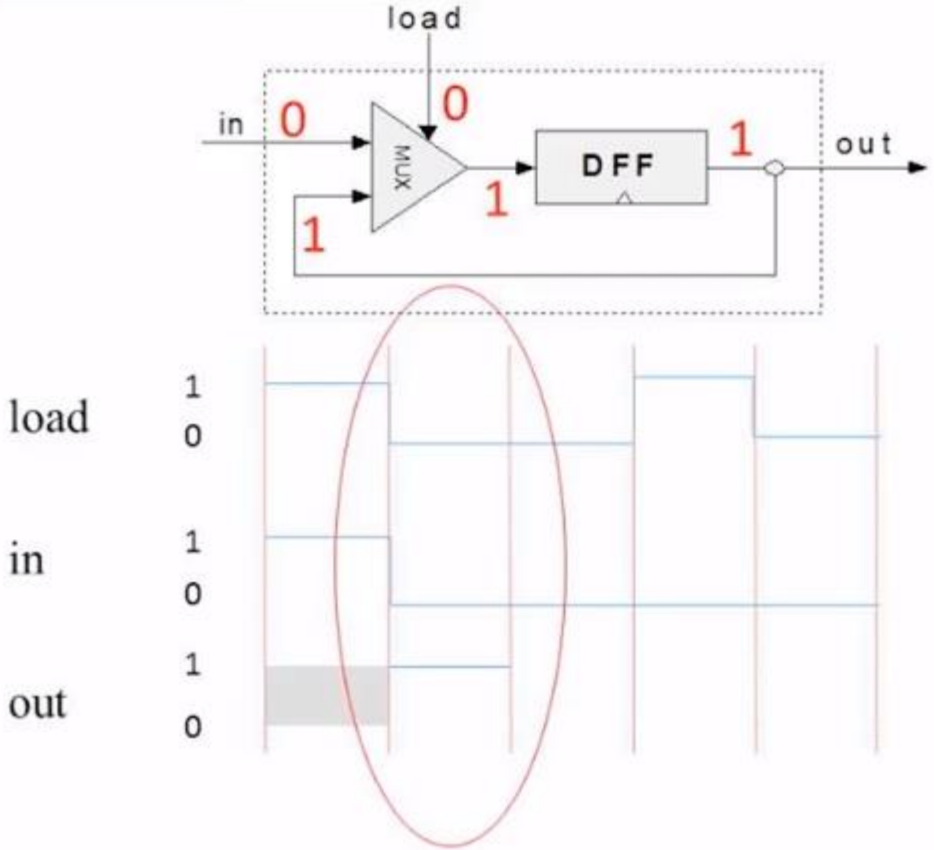
1-Bit Register

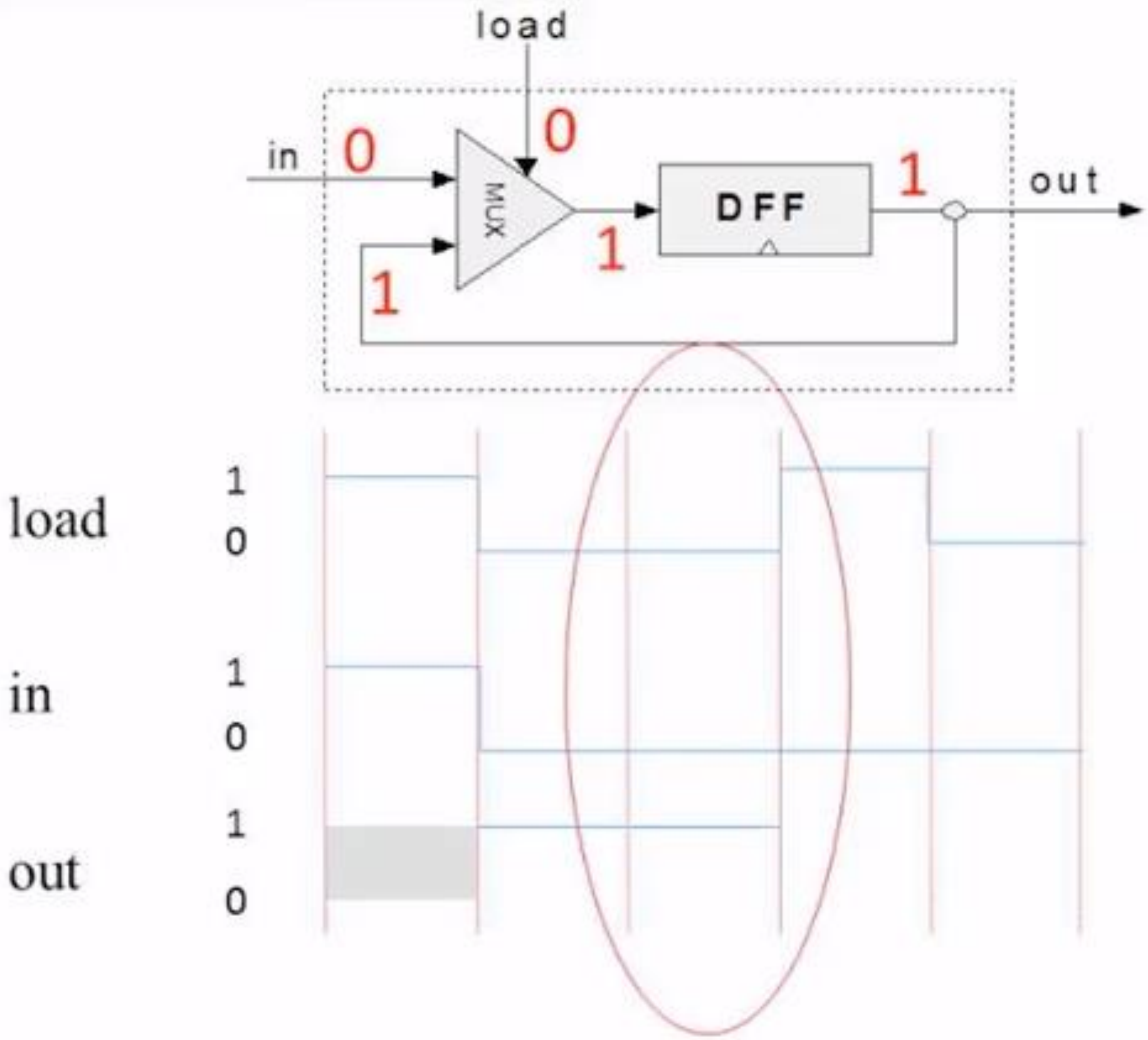


1-Bit Register

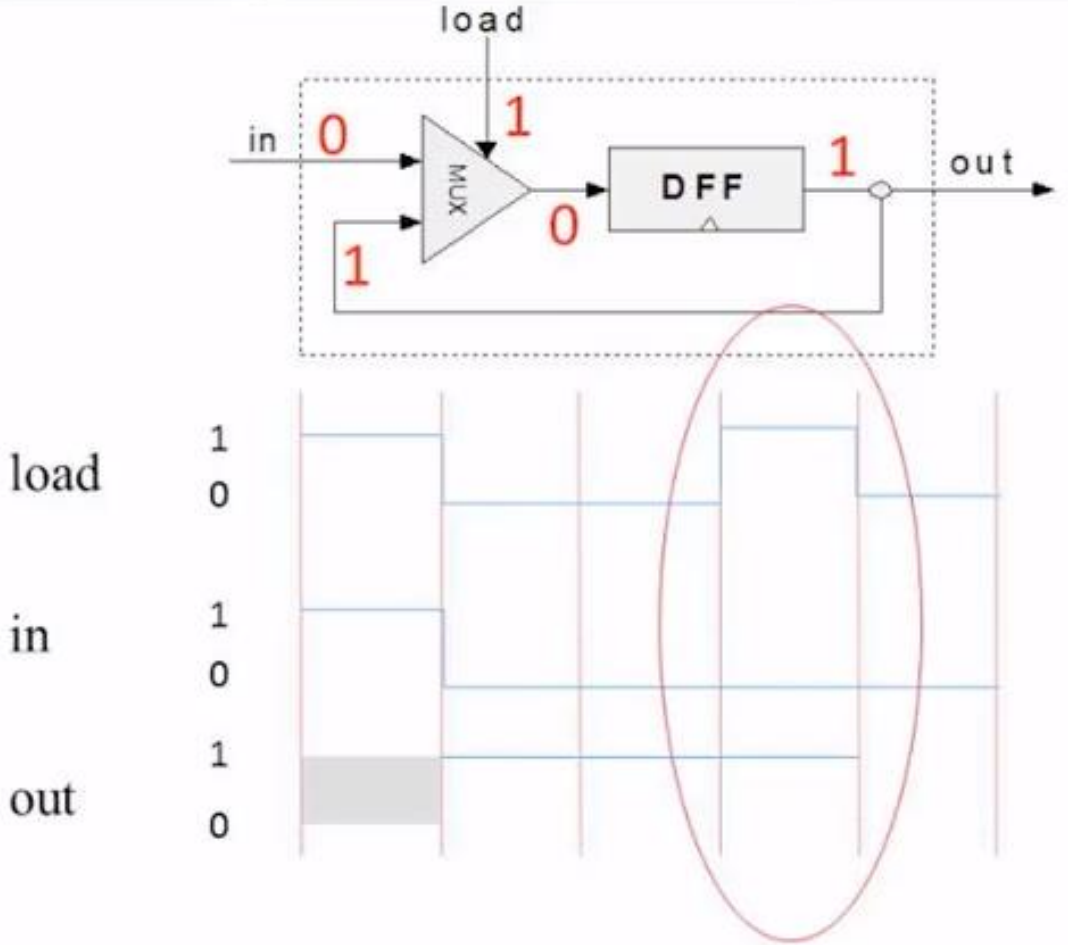


1-Bit Register

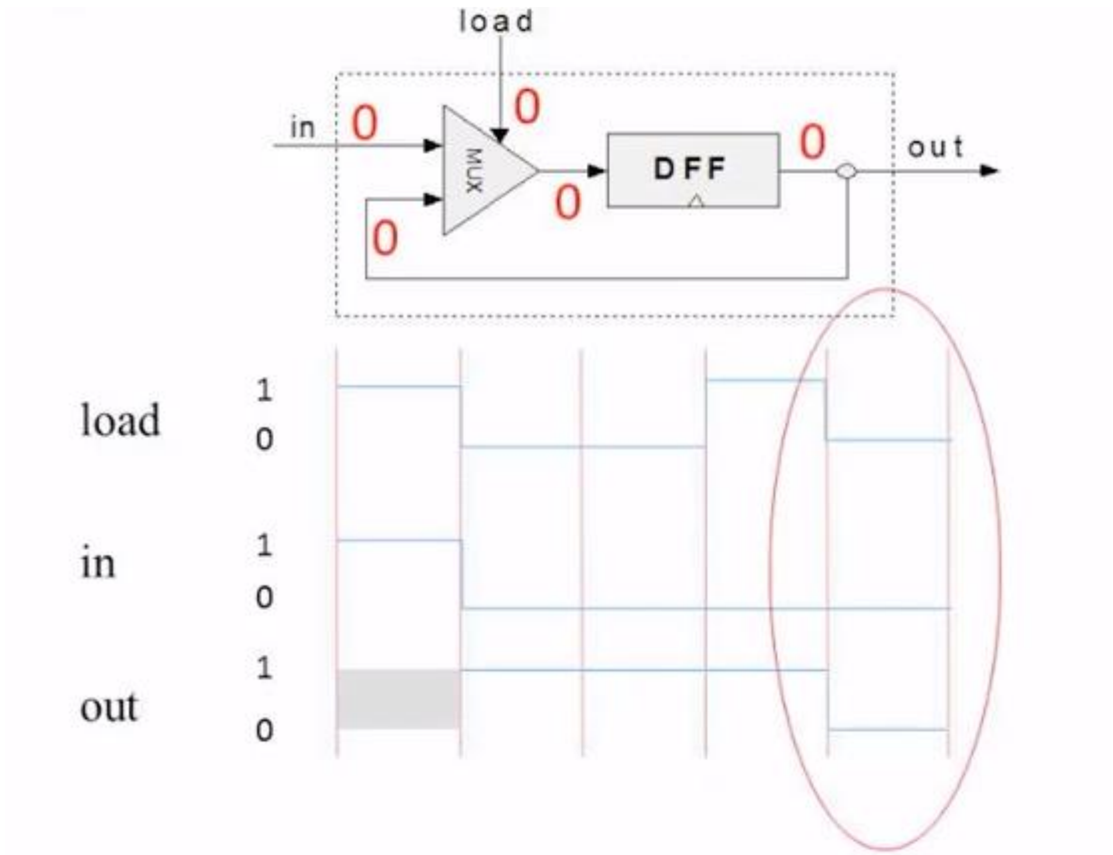



load
in
0
0
MUX
DFF
1
out
1
1
load
1
0
in
1
0
out
1
0

1-Bit Register



If load is 1, it takes the input value; if load is 0 it save the previous state..



- MEMORY

Memory

<u>Memory:</u>

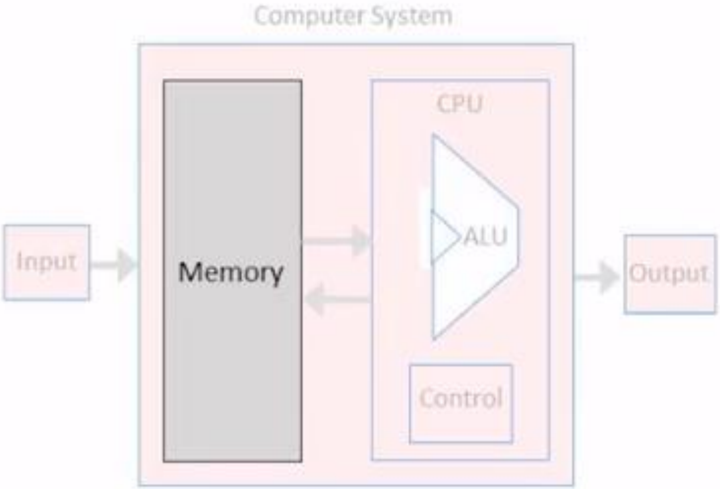
- Main memory: RAM, …
- Secondary memory: disks, …
- Volatile / non-volatile

<u>RAM:</u>

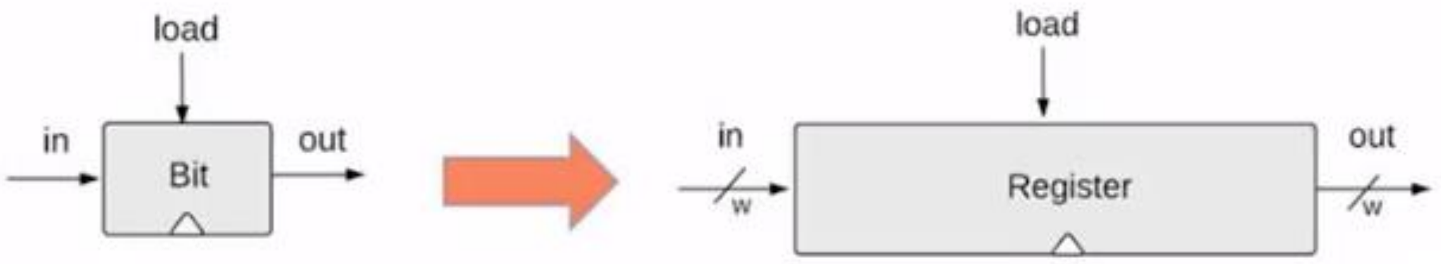
- Data
- Instructions

<u>Perspective:</u>

- Physical
- Logical



The most basic memory element: Register



1-bit
Register

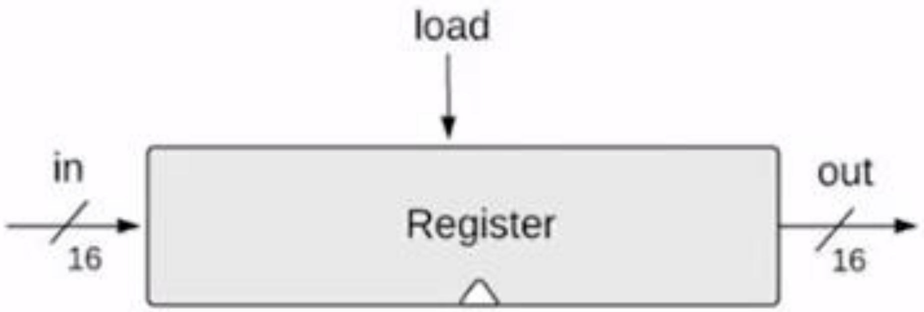
multi-bit
Register

- <u>w (word width):</u> 16-bit, 32-bit, 64-bit, …

  (from now on we will talk about 16-bit registers, without loss of generality)

- <u>Register's state:</u> the value which is currently stored inside the register

Register / read logic



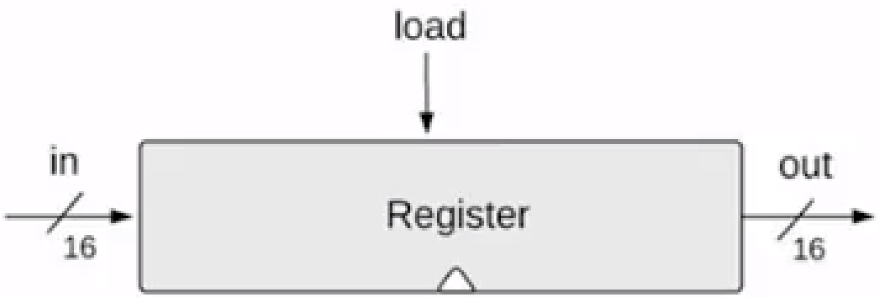
To read the Register:

probe `out`

Result:

`out` emits the Register's state

Register / write logic



To set Register = *v*

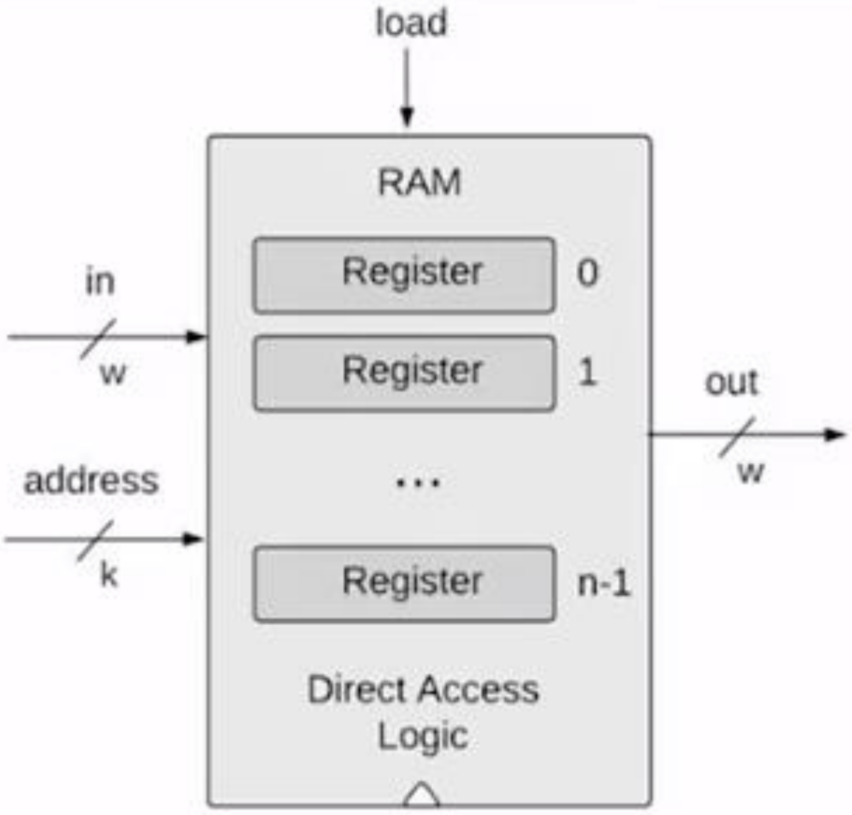
set `in` = v

set `load` = 1

Result:

- The Register's state becomes v
- From the next cycle onward, `out` emits v

RAM unit



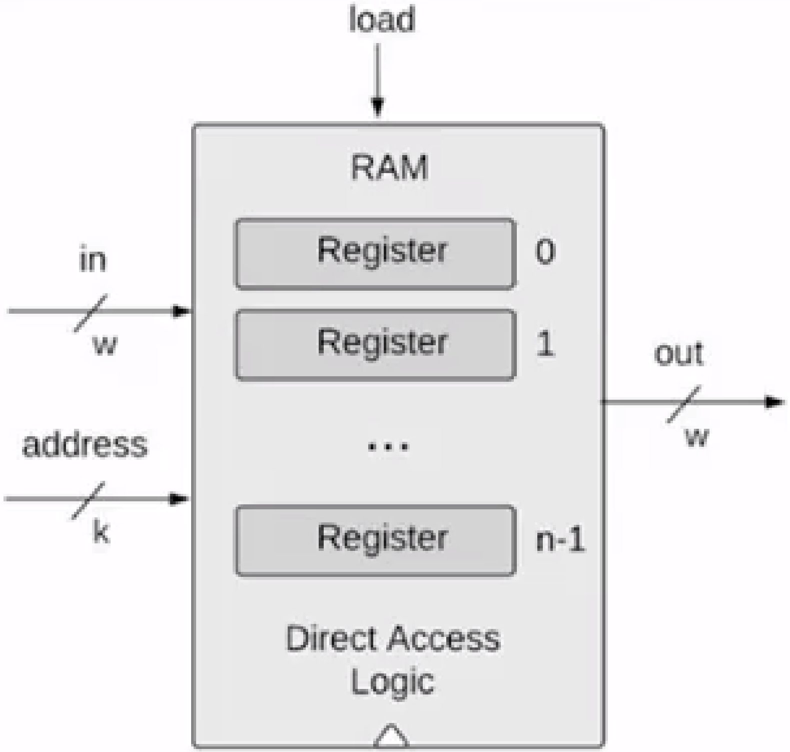
RAM abstraction:
A sequence of n addressable registers, with addresses 0 to n-1

At any given point of time, only *one* register in the RAM is selected

k (width of address input):
$k = \log_2 n$

w (word width):
No impact on the RAM logic
(Hack computer: w=16)

RAM / Read Logic



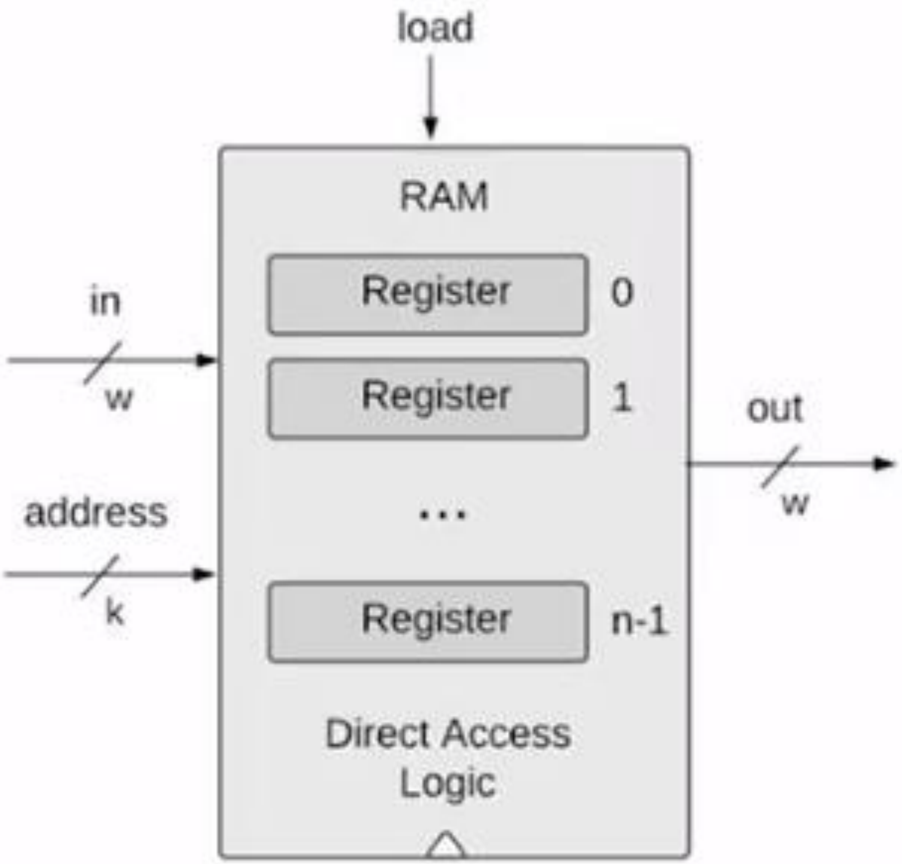
To read `Register` i :

set `address` $= i$

Result:

`out` emits the state of `Register` i

RAM / Write Logic



To set `Register` i to v :

set `address` $= i$

set `in` $= v$

set `load` $= 1$

Result:

- The state of `Register` i becomes v
- From the next cycle onward, `out` emits v