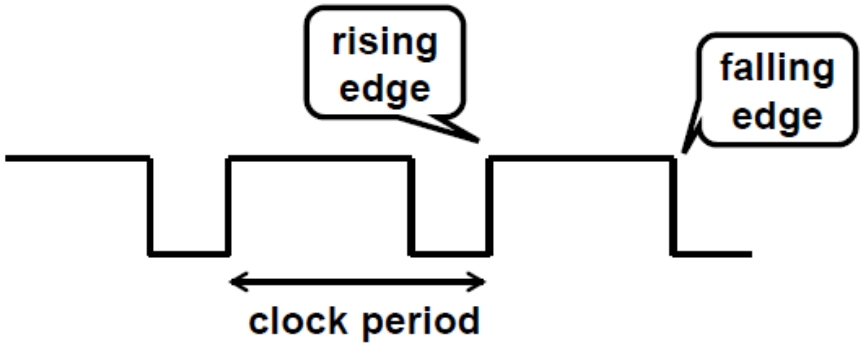
Sequential logic

A clock signal

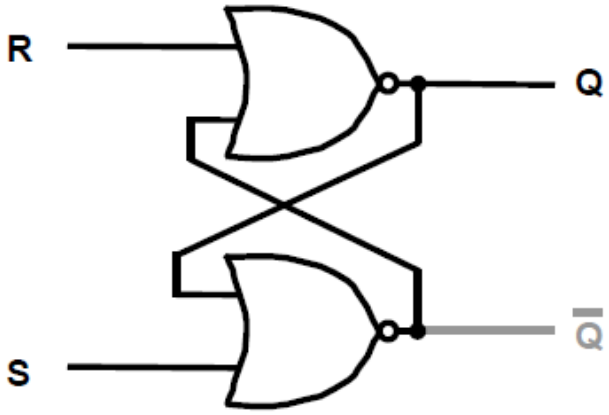
- **Sequential logic** blocks usually update their output on the clock edge



- This is edge-triggered clocking
- Change in inputs at any other time have no effect on the outputs

Set-reset (SR) latch

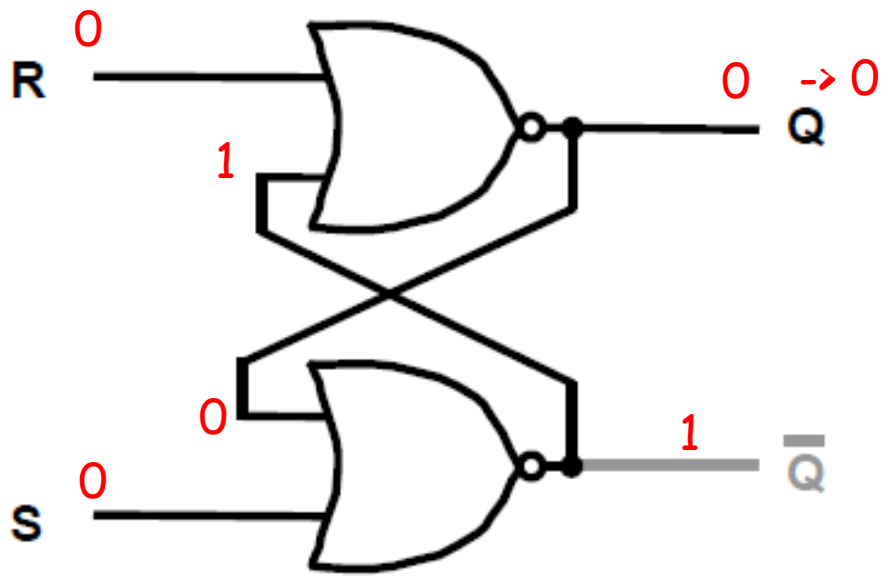
- Inputs
  - Reset
  - Set
- Not clocked
- Output undefined when S and R asserted simultaneously



		NOR
0	0	1
0	1	0
1	0	0
1	1	0

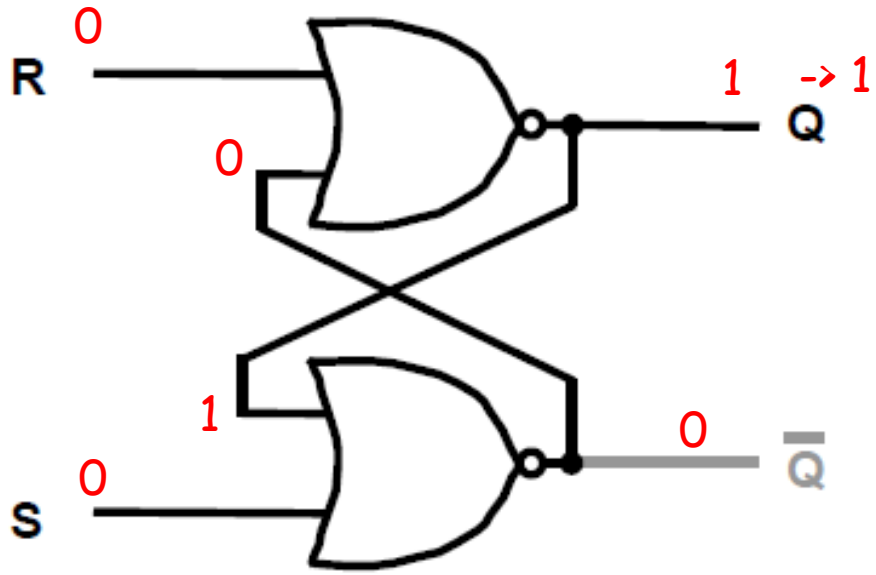
Q	S	R	Q_n
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	?
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	?

State transition table



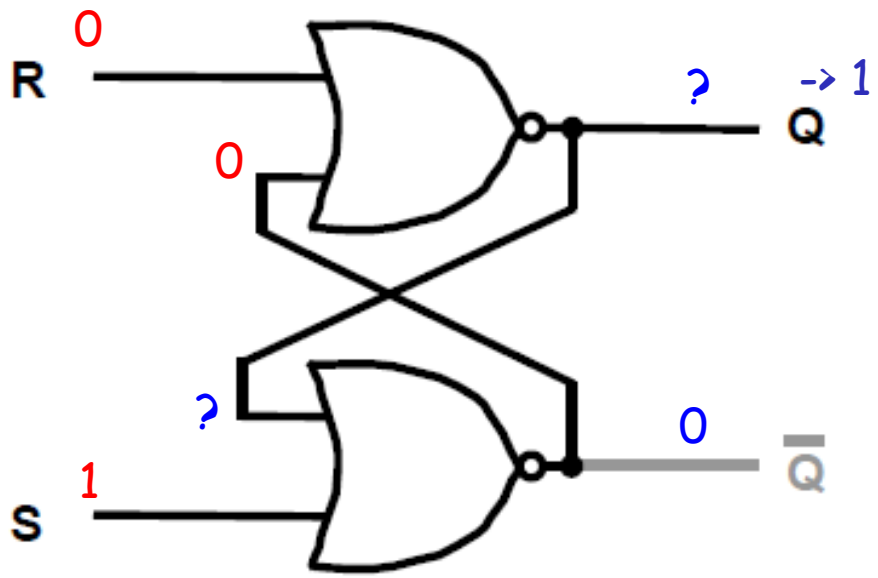
Q	S	R	Q_n
0	0	0	0
0	0	1	
0	1	0	
0	1	1	?
1	0	0	
1	0	1	
1	1	0	
1	1	1	?

State transition table



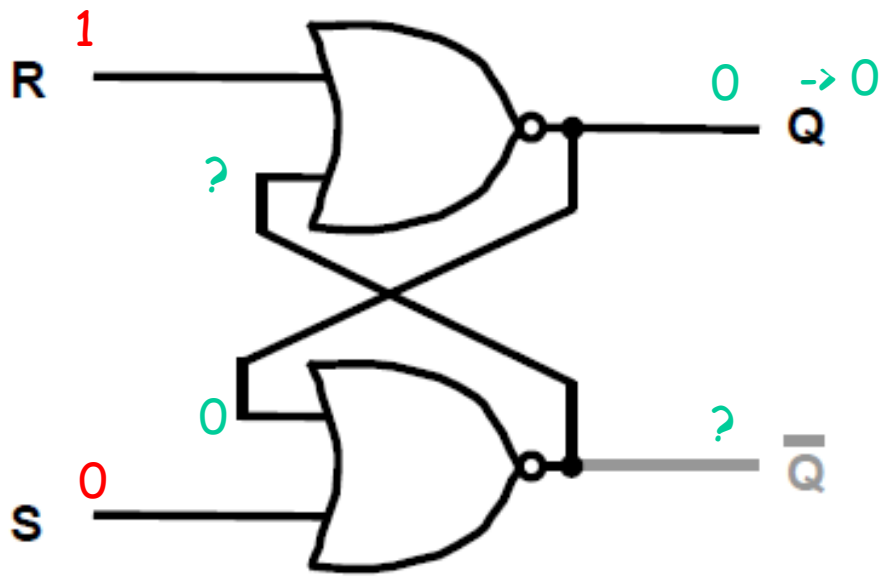
Q	S	R	Q_n
0	0	0	0
0	0	1	
0	1	0	
0	1	1	?
1	0	0	**1**
1	0	1	
1	1	0	
1	1	1	?

State transition table



Q	S	R	Q_n	
0	0	0	0	
0	0	1		
0	1	0	1	←
0	1	1	?	
1	0	0	1	
1	0	1		
1	1	0	1	←
1	1	1	?	

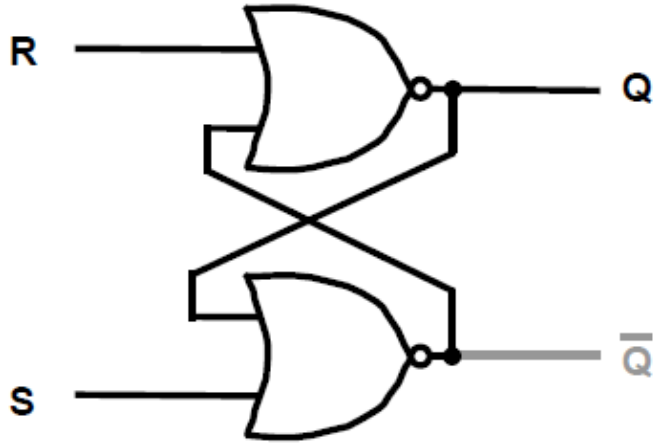
State transition table

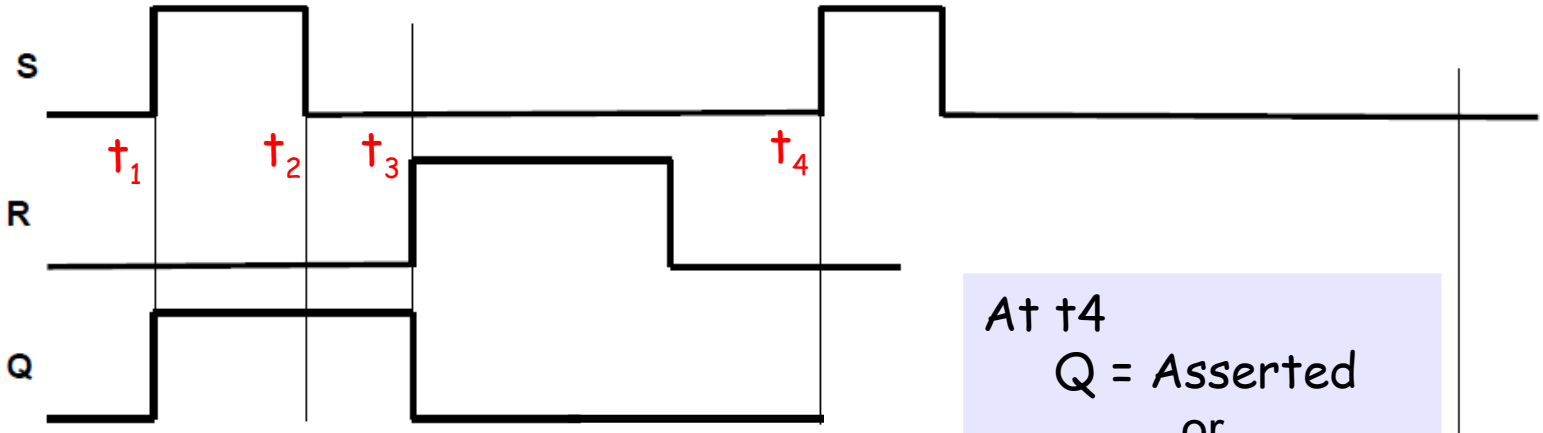


Q	S	R	Q_n
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	?
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	?

State transition table

S-R latch example

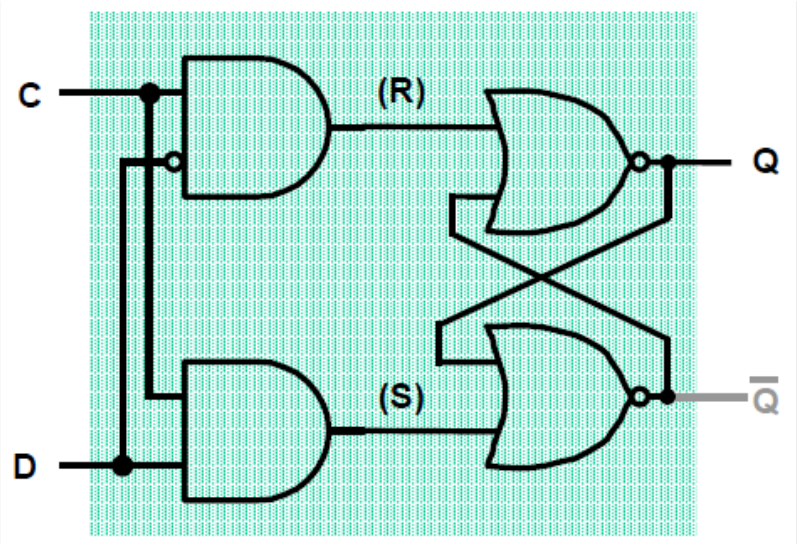


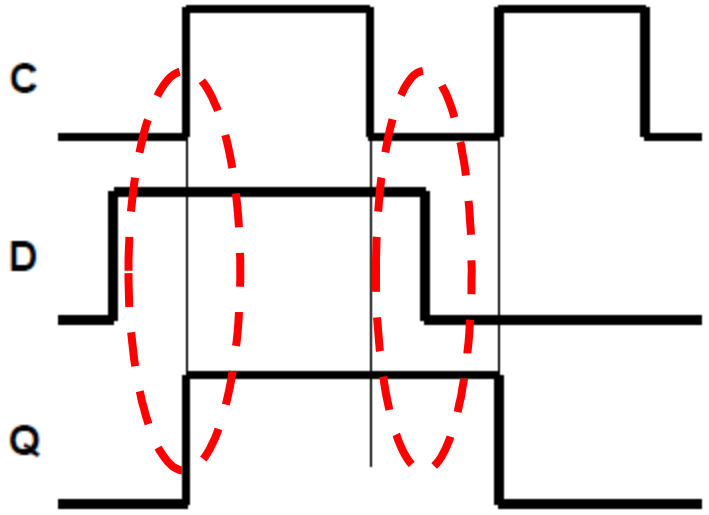


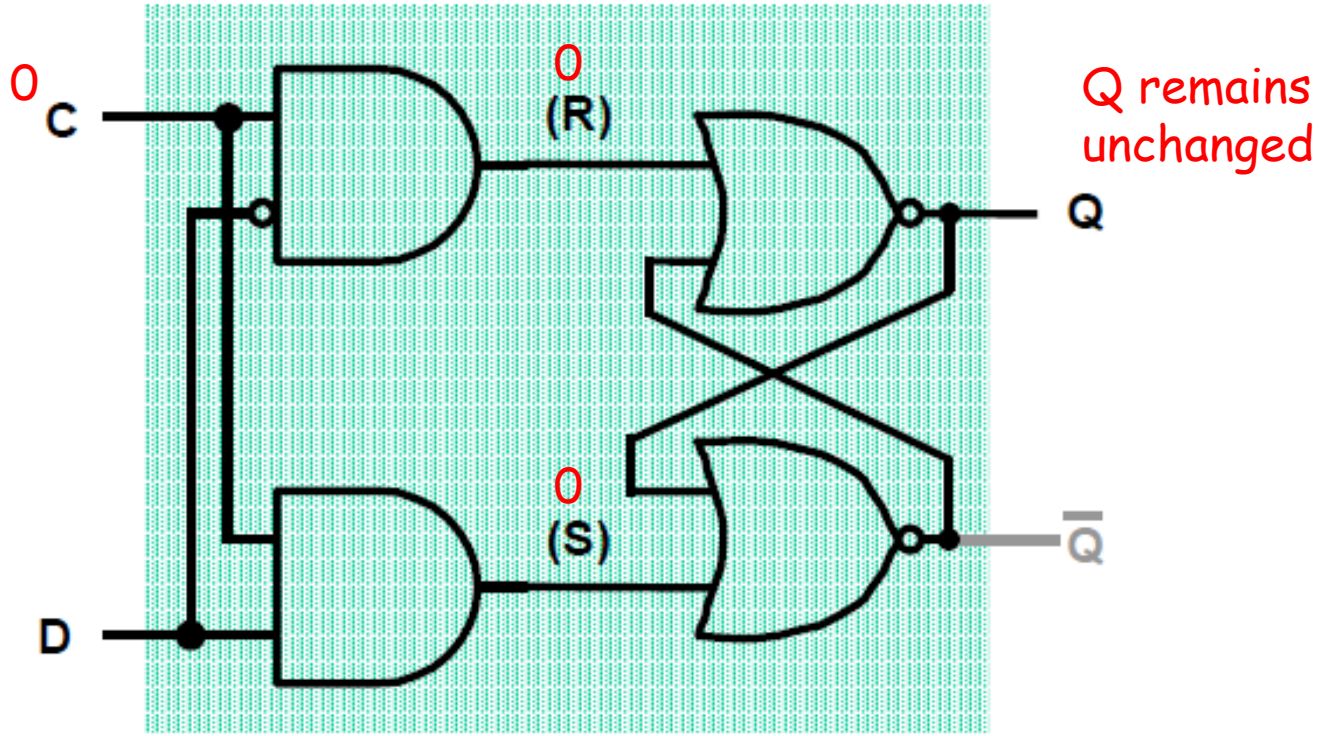
At t4
Q = Asserted
or
Q = Deasserted

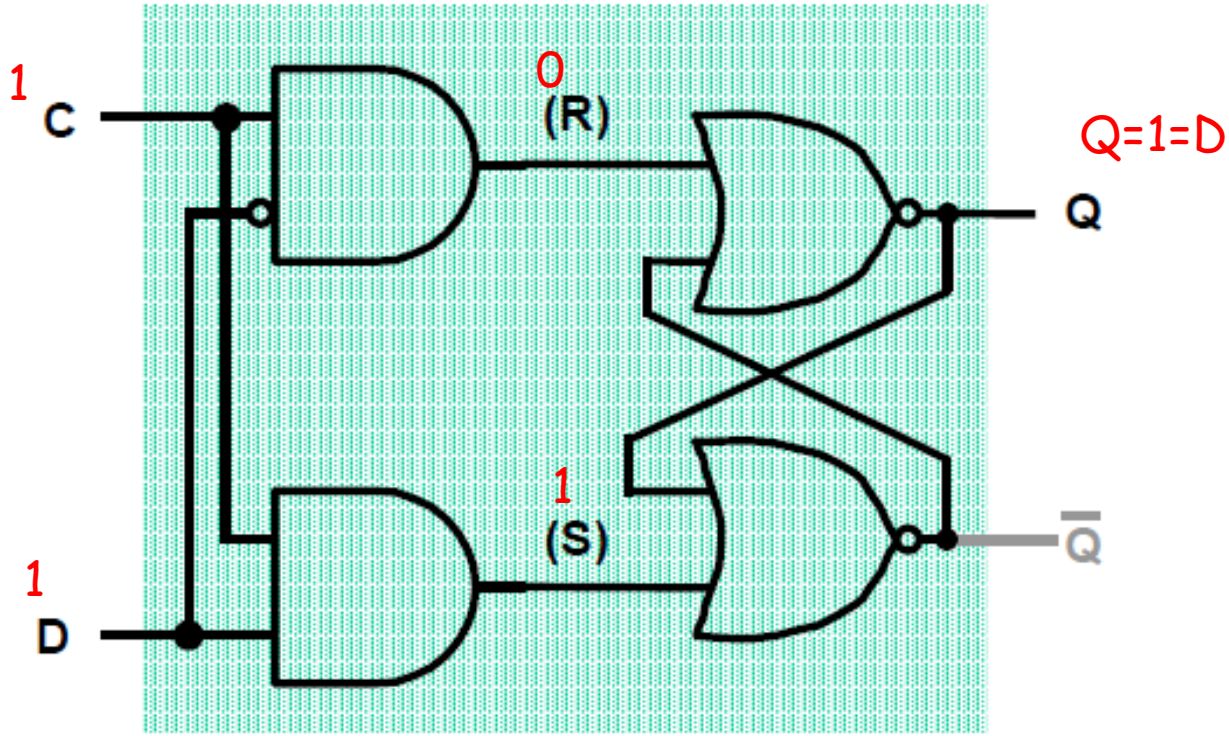
D latch

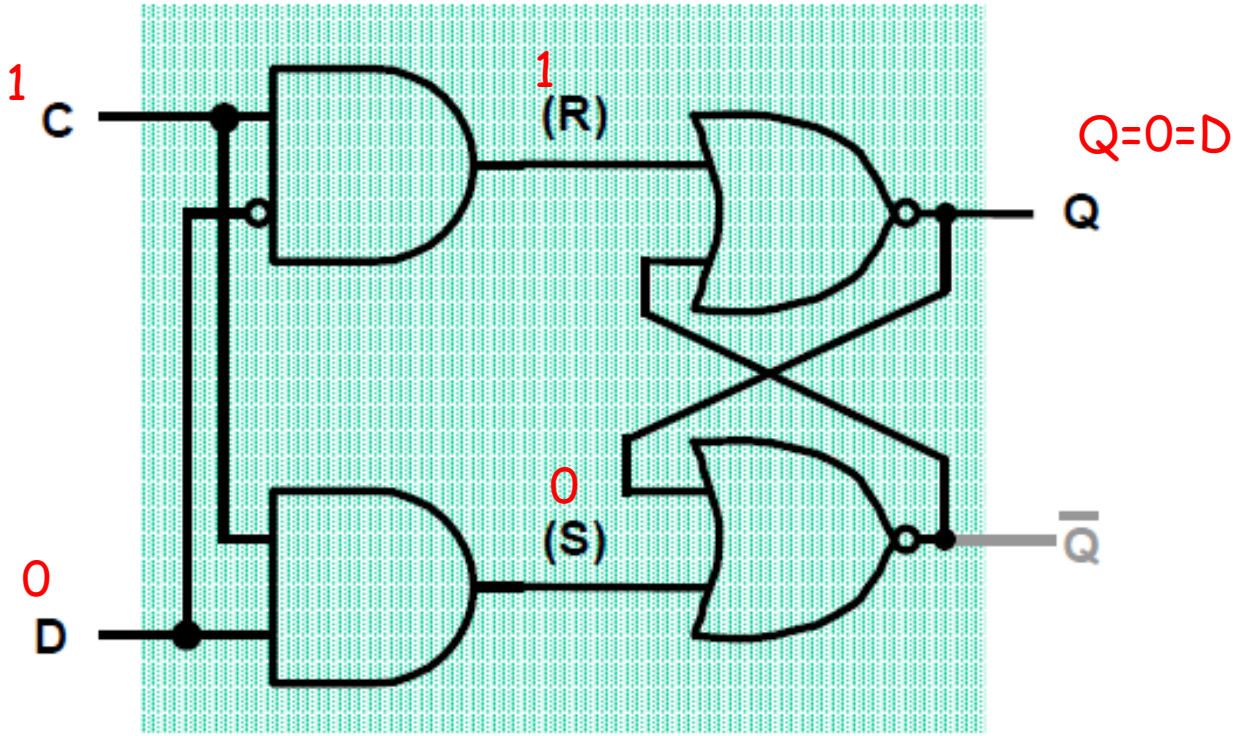
- D latches are level triggered with inputs
  - C, a clock input
  - D, a data input



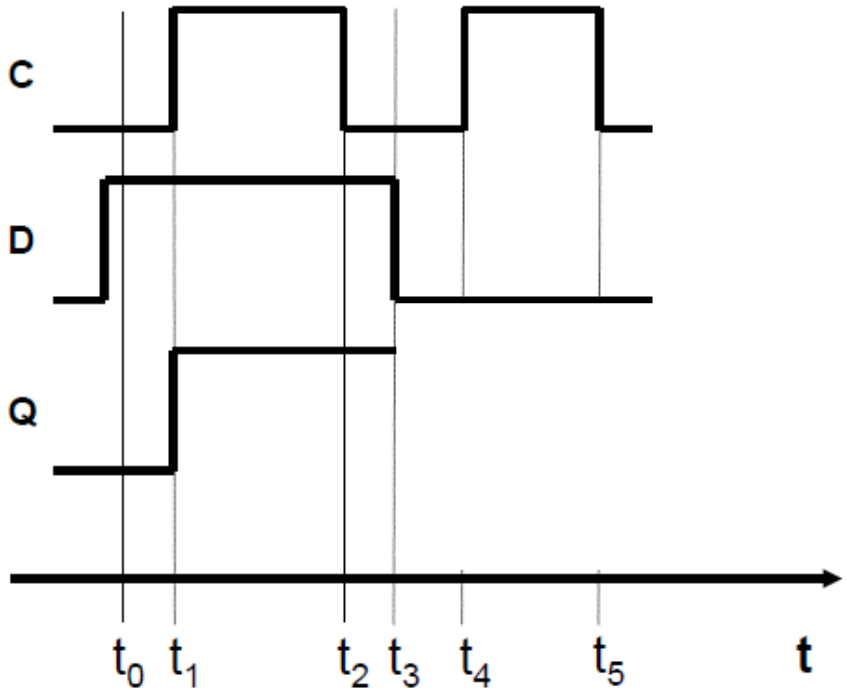



0
C
0
(R)
Q remains
unchanged
Q
0
(S)
$\overline{Q}$
D


1
C
0
(R)
Q=1=D
Q
1
(S)
Q̄
1
D


1
C
1
(R)
Q=0=D
Q
0
(S)
$\overline{Q}$
0
D

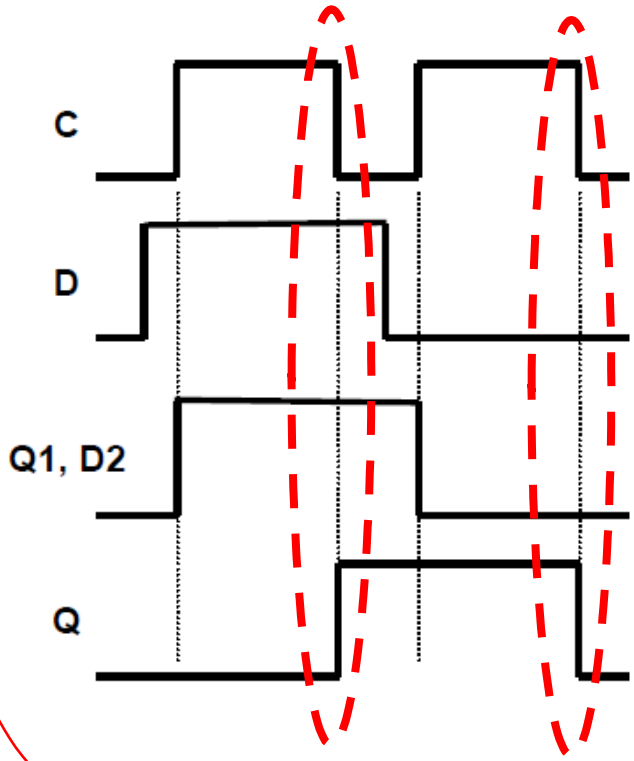
D latch example

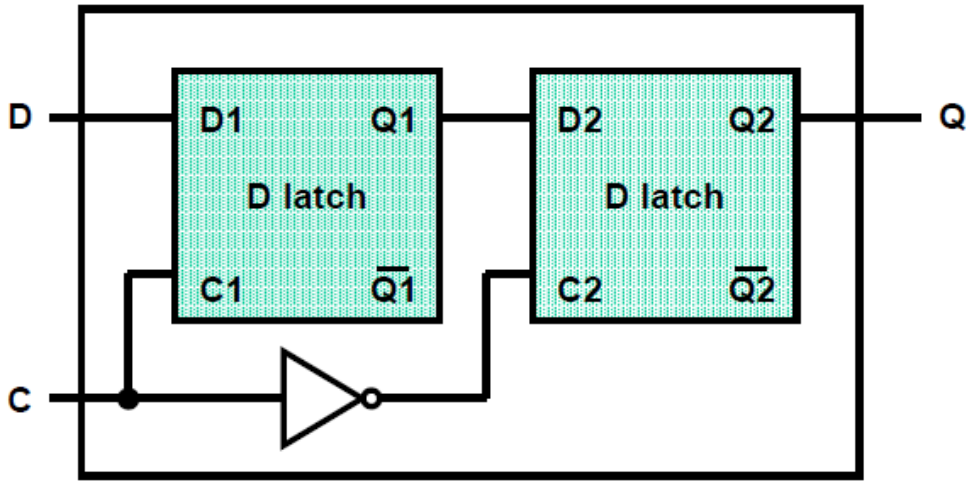


t	C	D	S	R	Q
t_0	0	1	0	0	0
t_1	1	1	1	0	1
t_2	0	1	0	0	1
t_3	0	0	0	0	
t_4	1	0	0	1	
t_5	0	0	0	0	

D flip flops

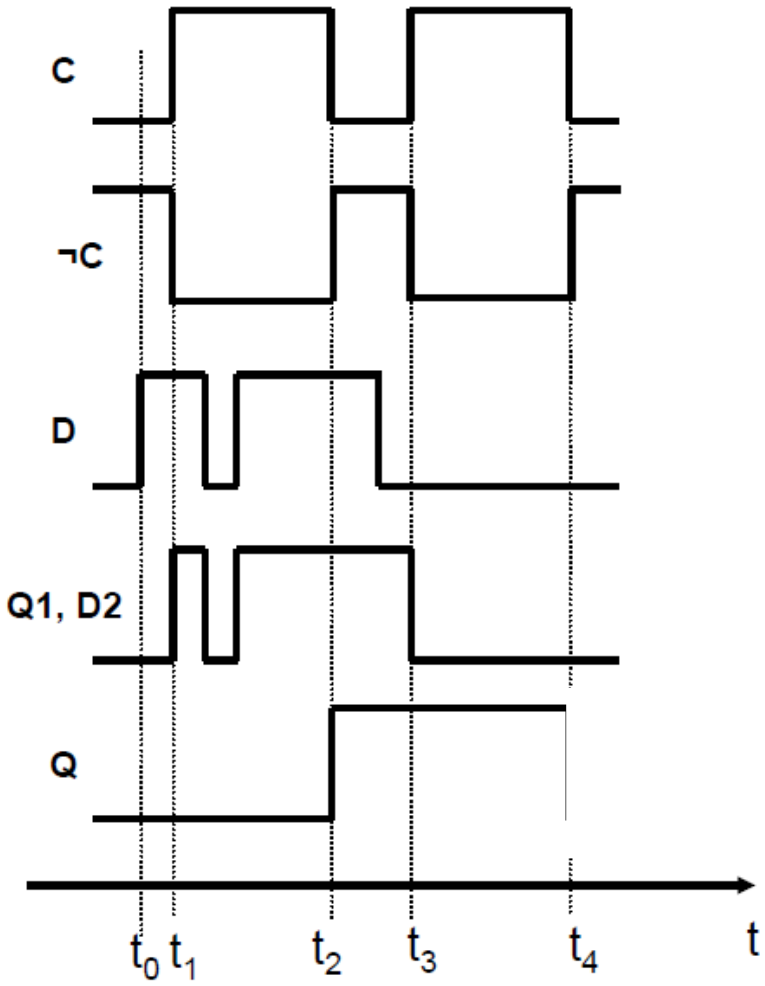
- Flip-flops are non-transparent latches



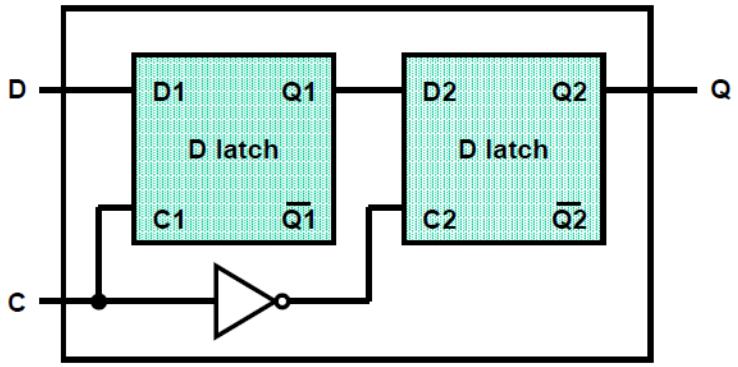


- When C is asserted, the first latch, the master, is open and follows the input D, $Q_1 = D$
- When C changes to deasserted, the first latch is closed, but the second latch, the slave, is open: $Q_2 = Q_1$

Flip-flops example



t	C	$\neg C$	D	$Q1$	Q
t_0	0	1	1	0	0
t_1	1	0	1	1	0
t_2	0	1	1	1	1
t_3	1	0	0	0	1
t_4	0	1	0	0	



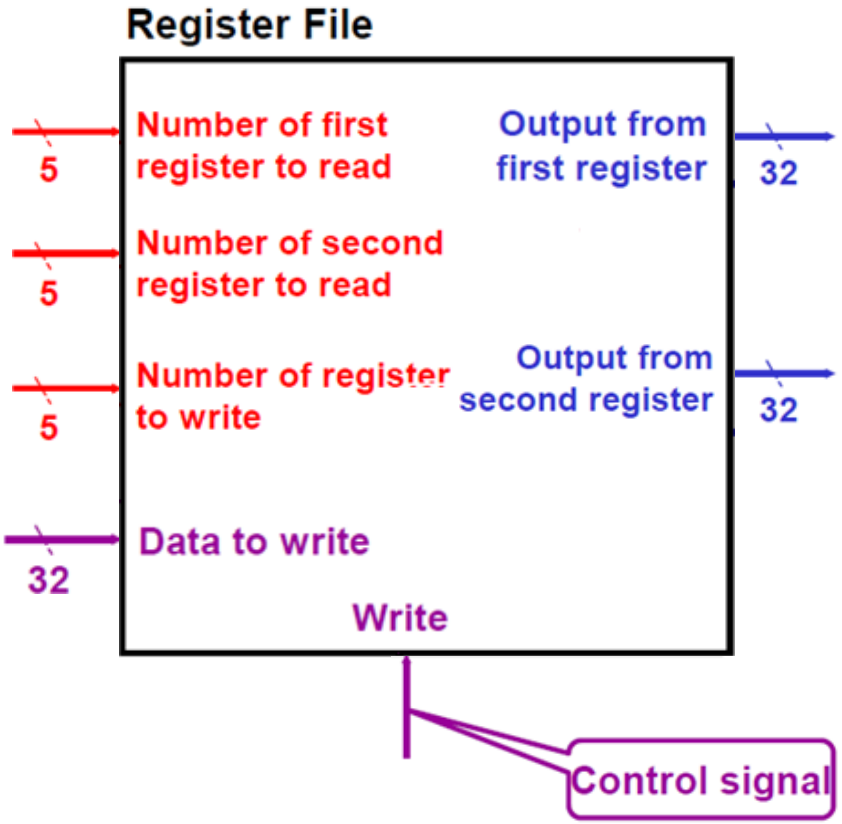
Register file

- A register can be made by combining several flip-flops
  - e.g. 32 flip-flops to make one MIPS register
  - Each flip-flop stores one bit
  - Several registers can then be combined into a register file
  - Each register in the register file gets a number n (address)

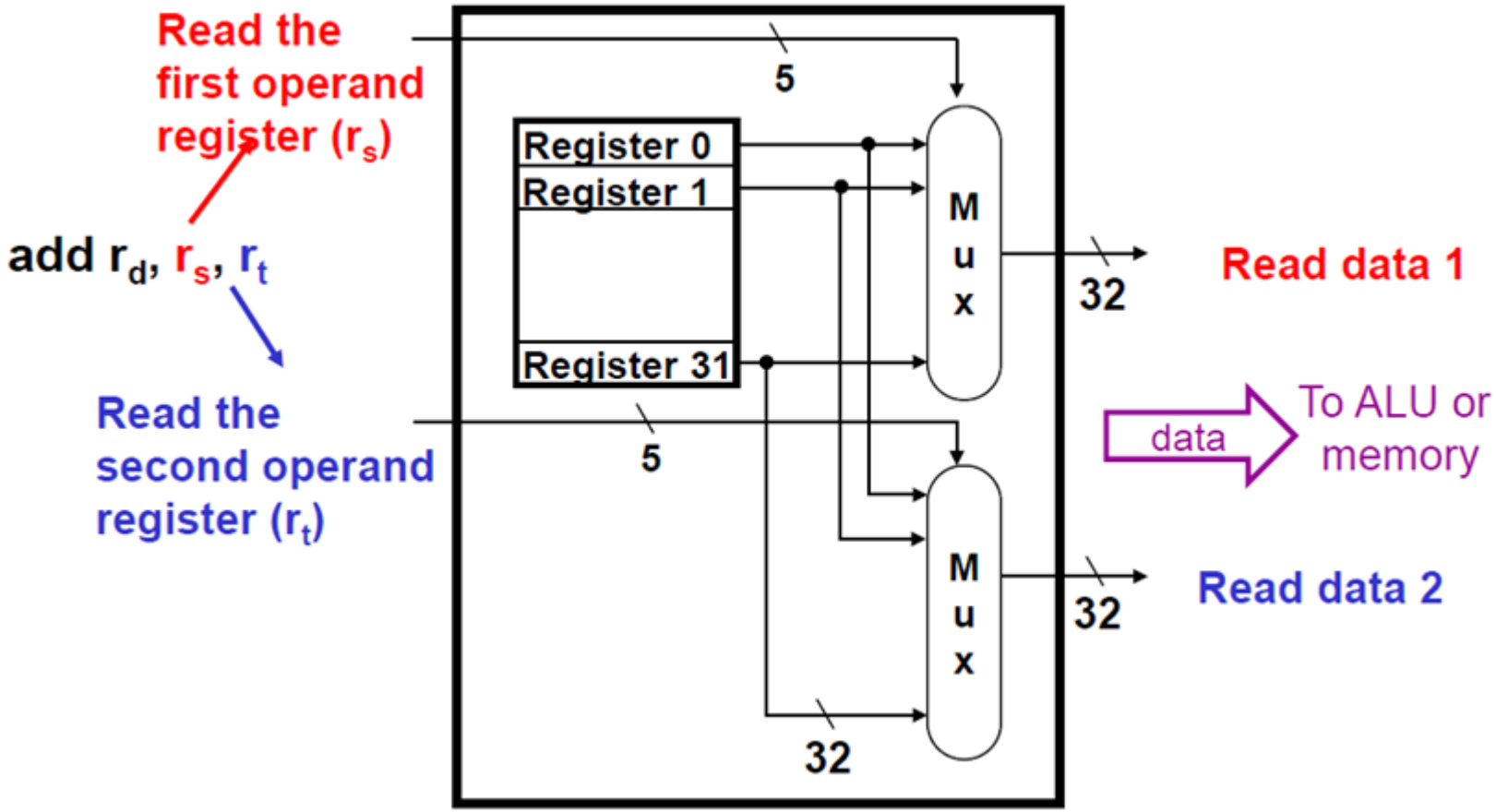
- There are two basic operations
  - read_register(n,d) – delivers data d from register n to ALU or memory
  - write_register(n,d) – inserts data d into the register number n

Register file design

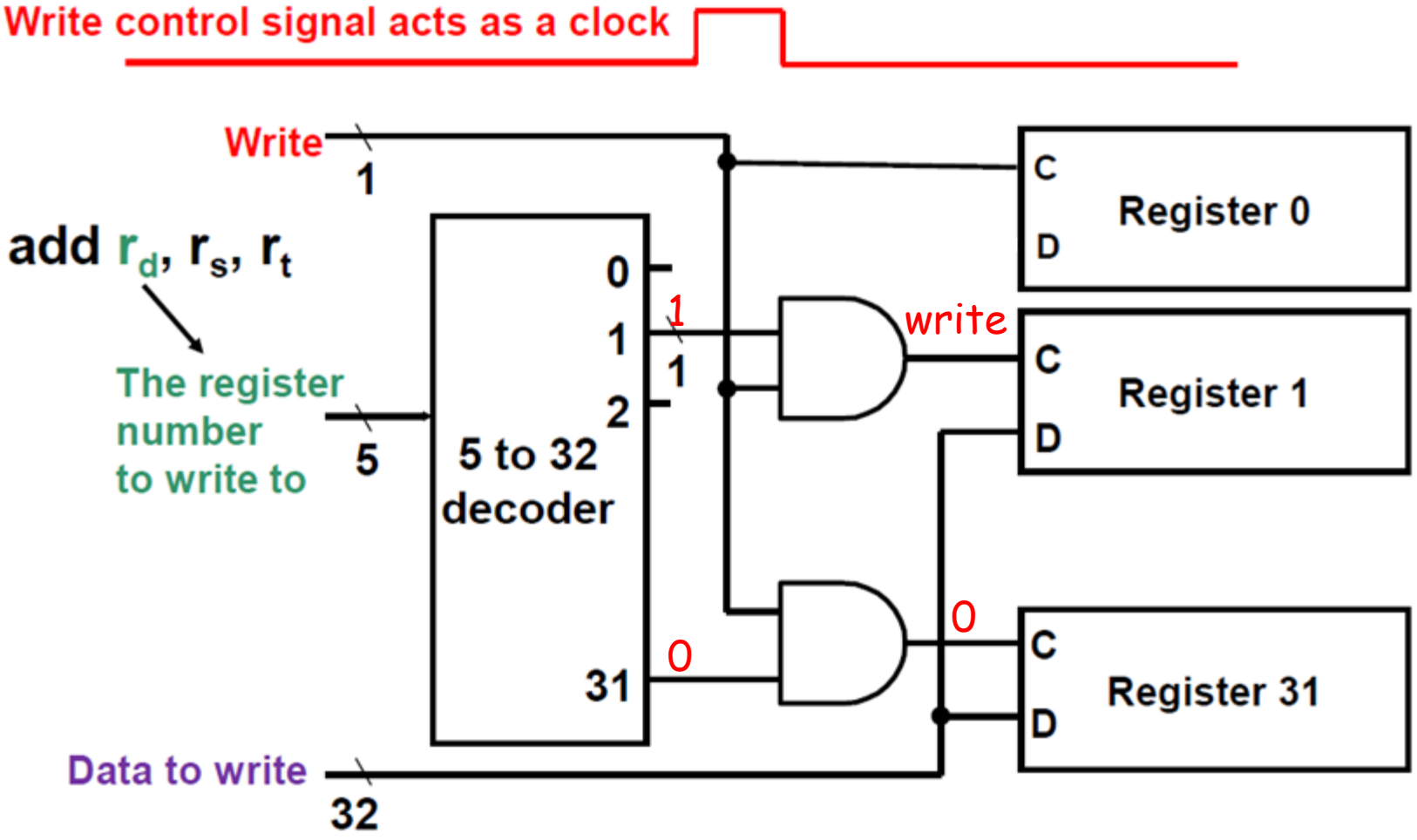
- The register file has:
  - 32 registers
  - Three register address inputs
  - Two output ports
    - Output from source register
    - Output from target register
  - One data input port (data to write)
  - Write control input



Reading from the register file



Writing to the register file



Why so few registers?

- We need 32 flip-flops per register
- We have n=32 registers
- Each flip-flop is made up of 15 gates
  - 7 gates for a D latch + 1 for inverter
- Total gates of the flip-flops: 32 x 32 x 15 = 15,360 gates
- And each gate needs multiple transistors

Summary

- Combinational logic blocks are made of AND, OR, and NOT basic gates
  - The output is determined solely by inputs, it does not depend on what was before (no memory)
    - Multiplexors
    - Decoders
- For sequential logic, the output depends not only on the present input but also on the history of the input
  - Latches and flip-flops are cross-coupled NOR gates
  - S-R latch remembers state until changed
  - D latch (advanced S-R latch)
  - Flip-flops (advanced D latch)
- Register file
  - 32 register (each has 32 flip-flops), 2 multiplexors and 1 decoder
  - Read at most two registers at one time and write at most one.