Basic CPU – Control Path

Based on 컴퓨터구조론 (개정 6 판), 김종현 , 생능출판사

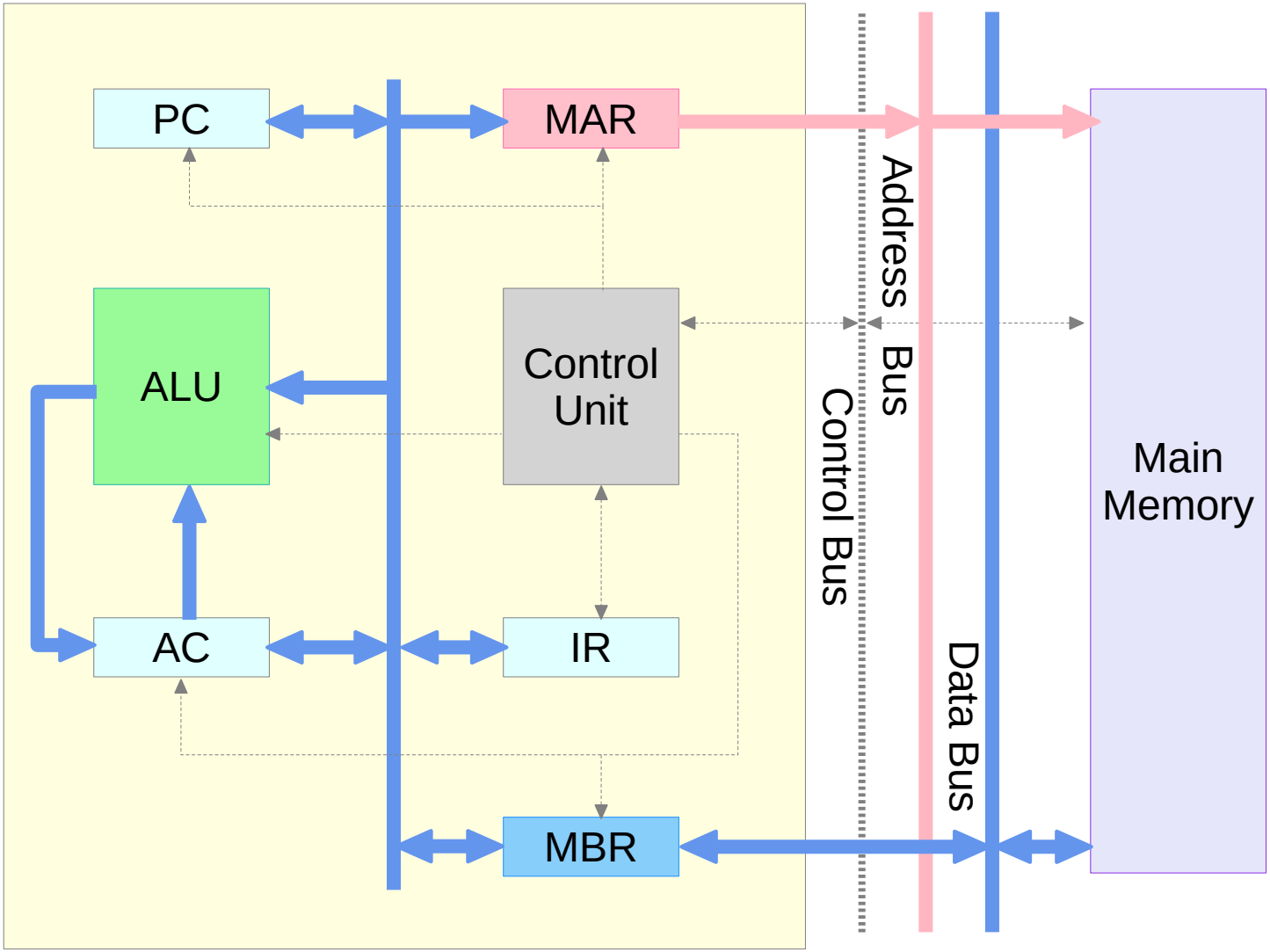
(Computer Architecture, 6th ed, Jonghyun Kim, Life & Power Press, Korea)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

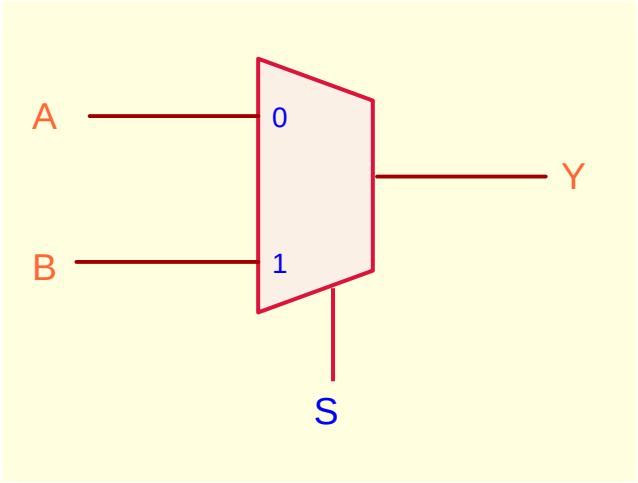
This document was produced by using LibreOffice and Octave.

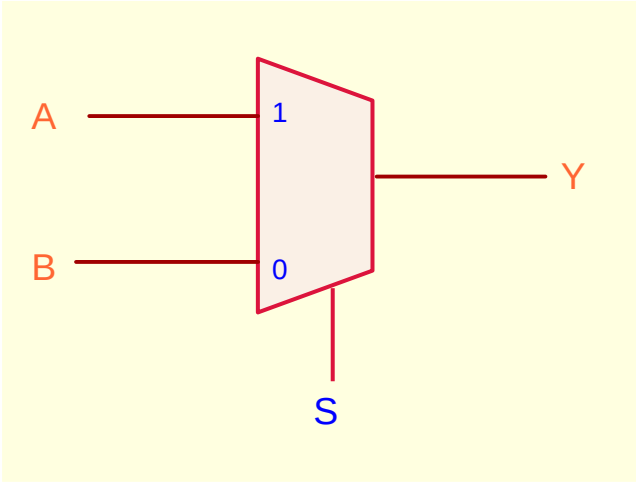
Data Path

Basic CPU



2:1 Multiplexer



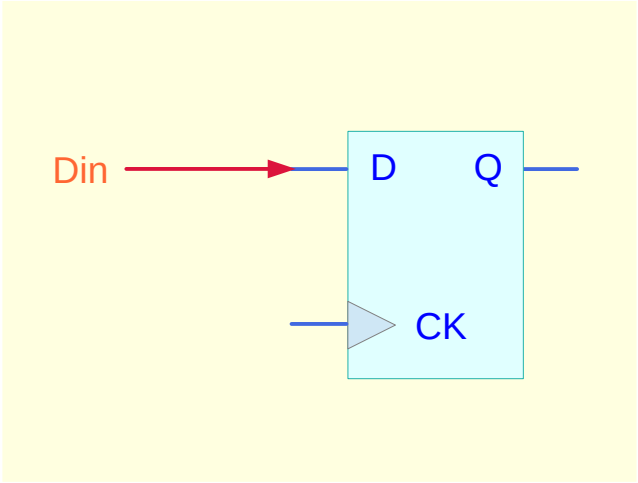


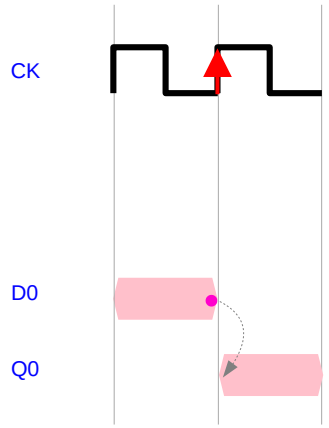
0, 1 are labels
for the two inputs,
one of which are
to be selected by S

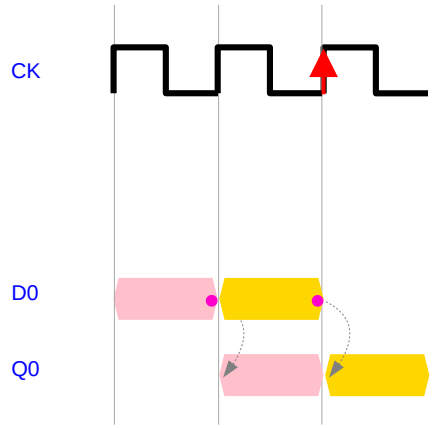
Y = B when S = 1
Y = A when S = 0

Y = A when S = 1
Y = B when S = 0

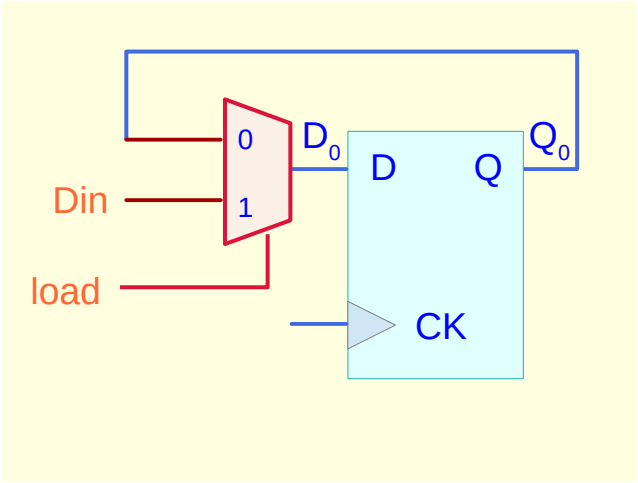
Using D FlipFlops

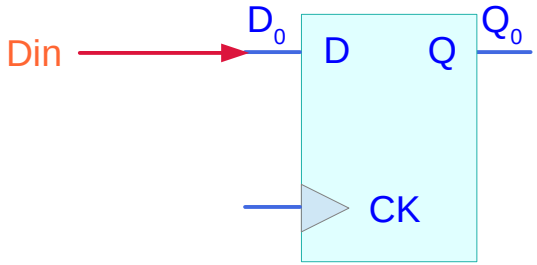




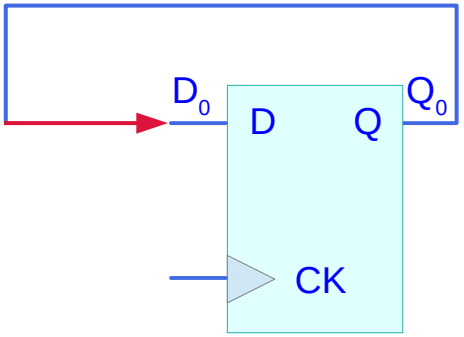


Using D FlipFlops with a MUX

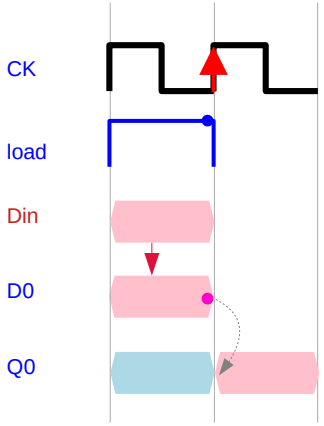


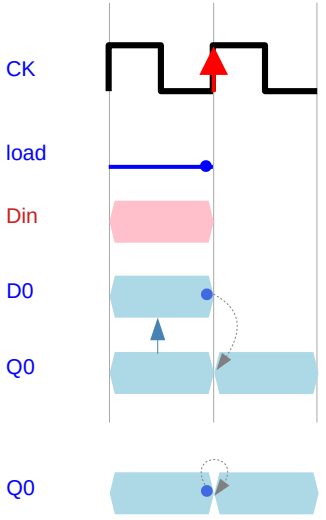


when load =1

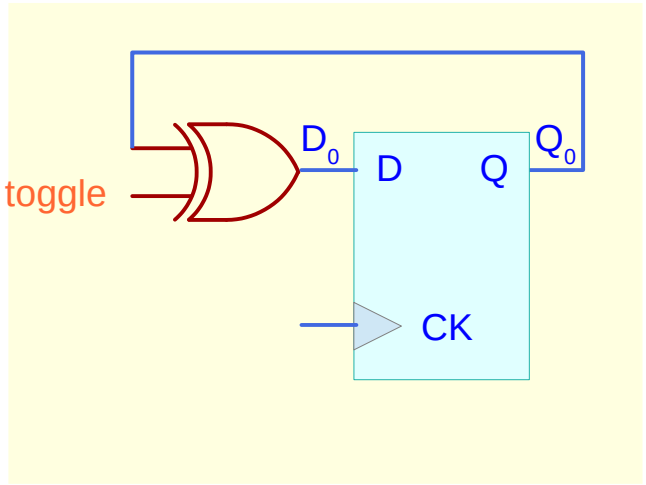


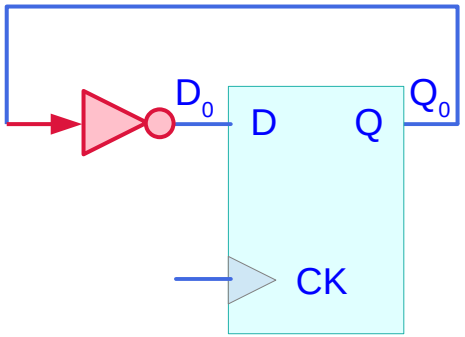
when load =0



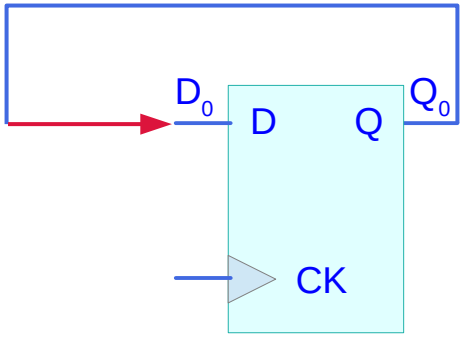


Using D FlipFlops with an XOR

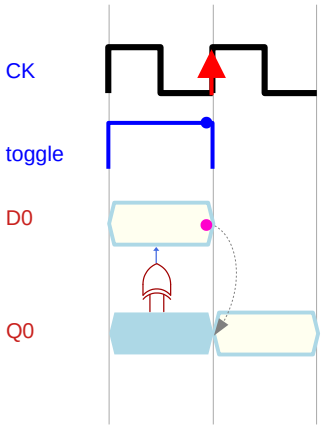


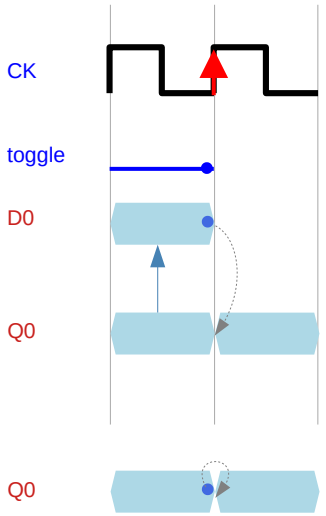


when toggle =1

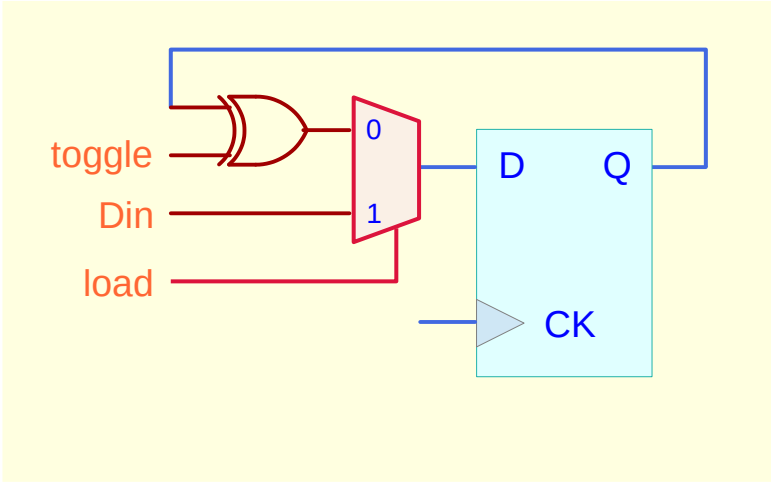


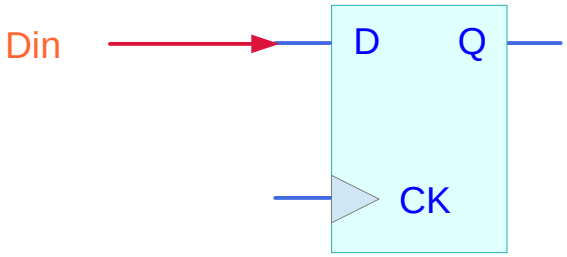
when toggle =0



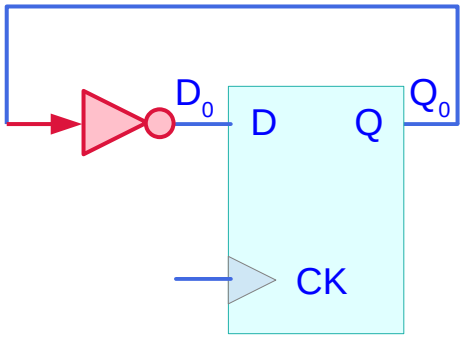


Using D FlipFlops with a MUX and an XOR

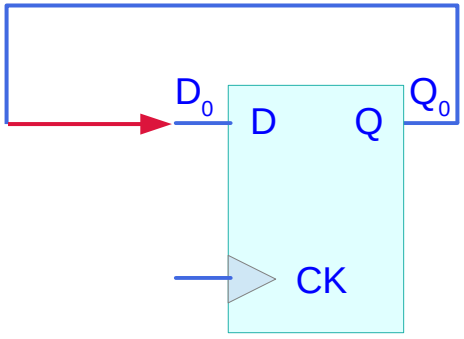




when load =1

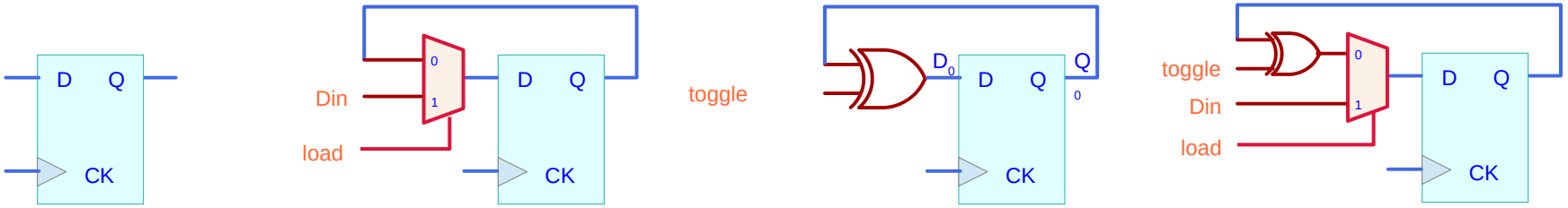


when load =0 & toggle =1

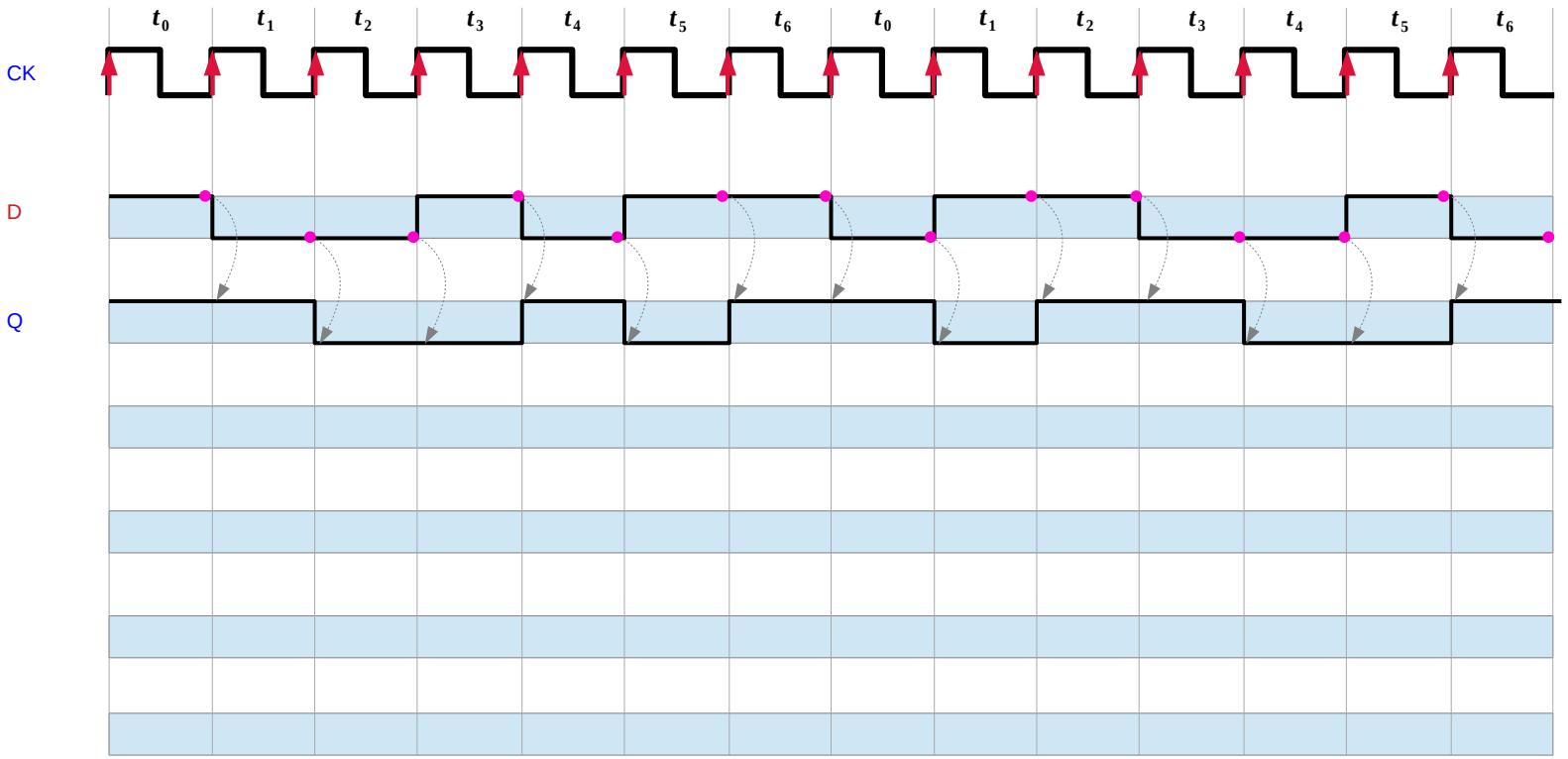


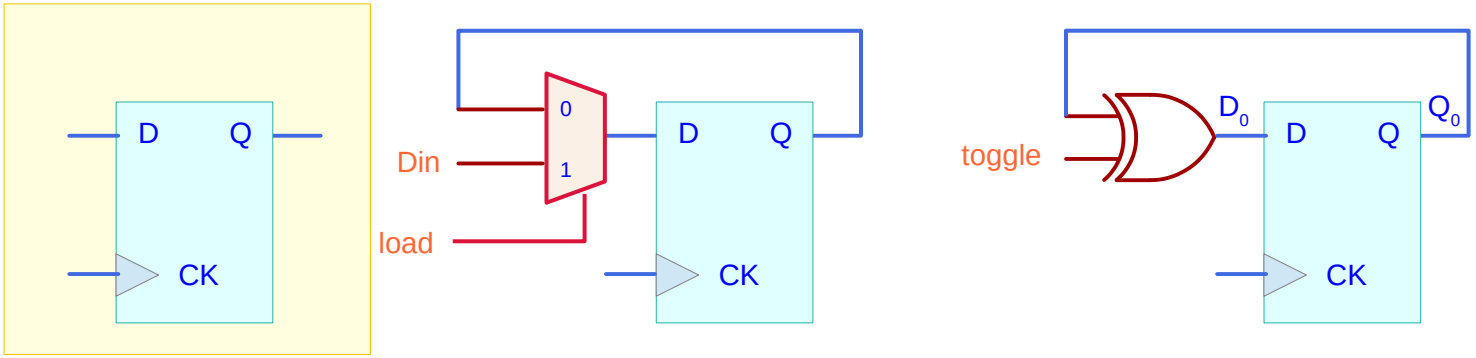
when load =0 & toggle =0

Using D FlipFlops

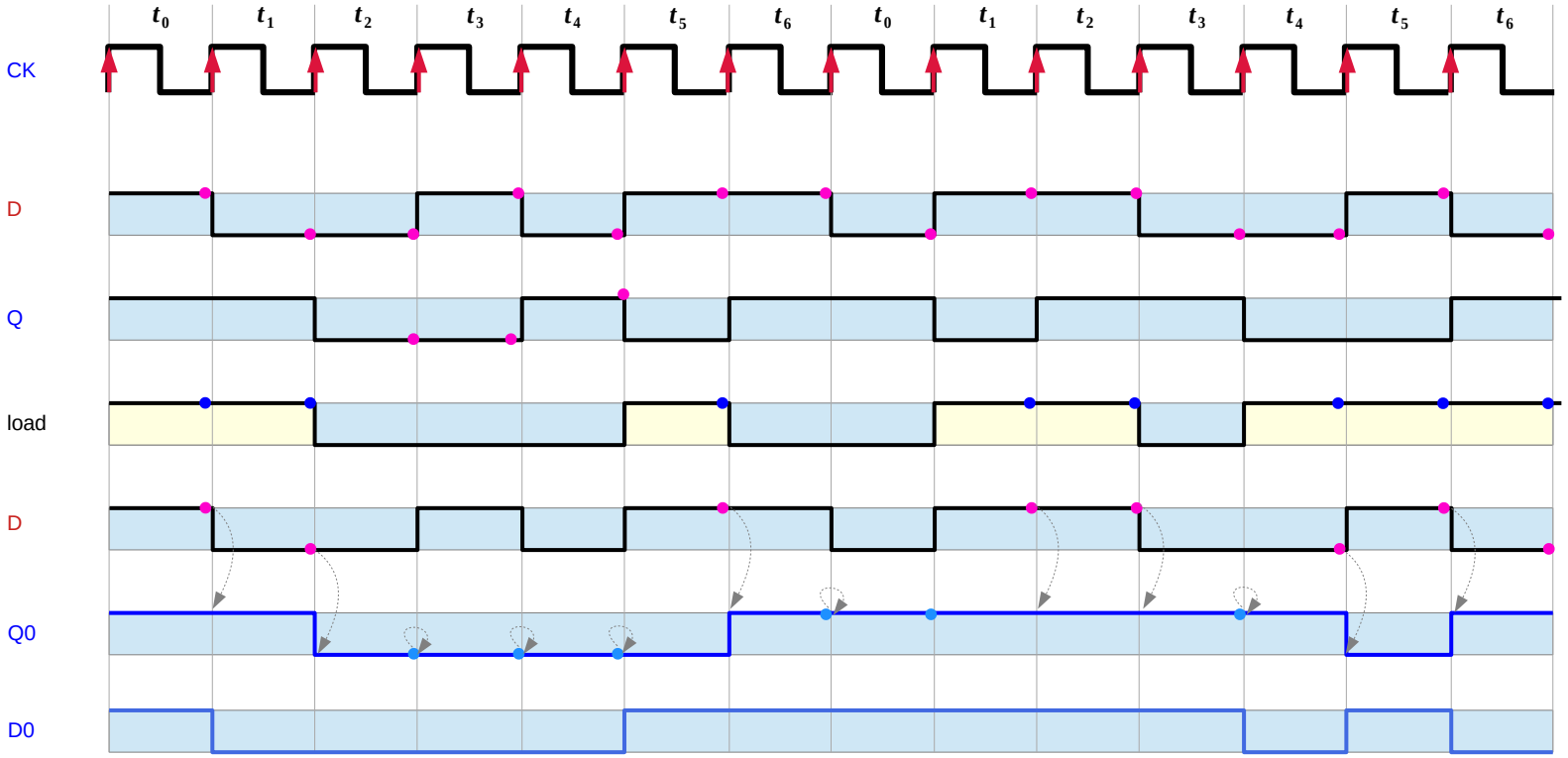


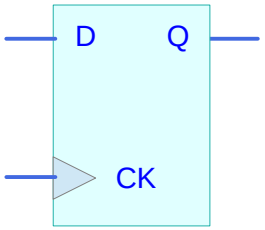
D FlipFlop

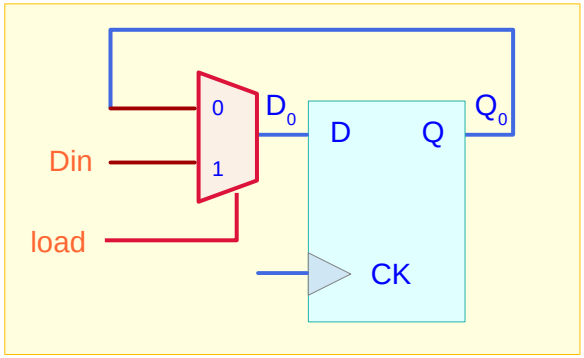

t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_0 t_1 t_2 t_3 t_4 t_5 t_6
CK
D
Q

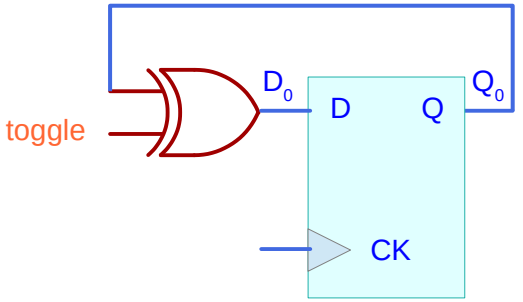

D
Q
CK
0
1
Din
load
D
Q
CK
toggle
D_0
D
Q
Q_0
CK

D FlipFlop with a load input

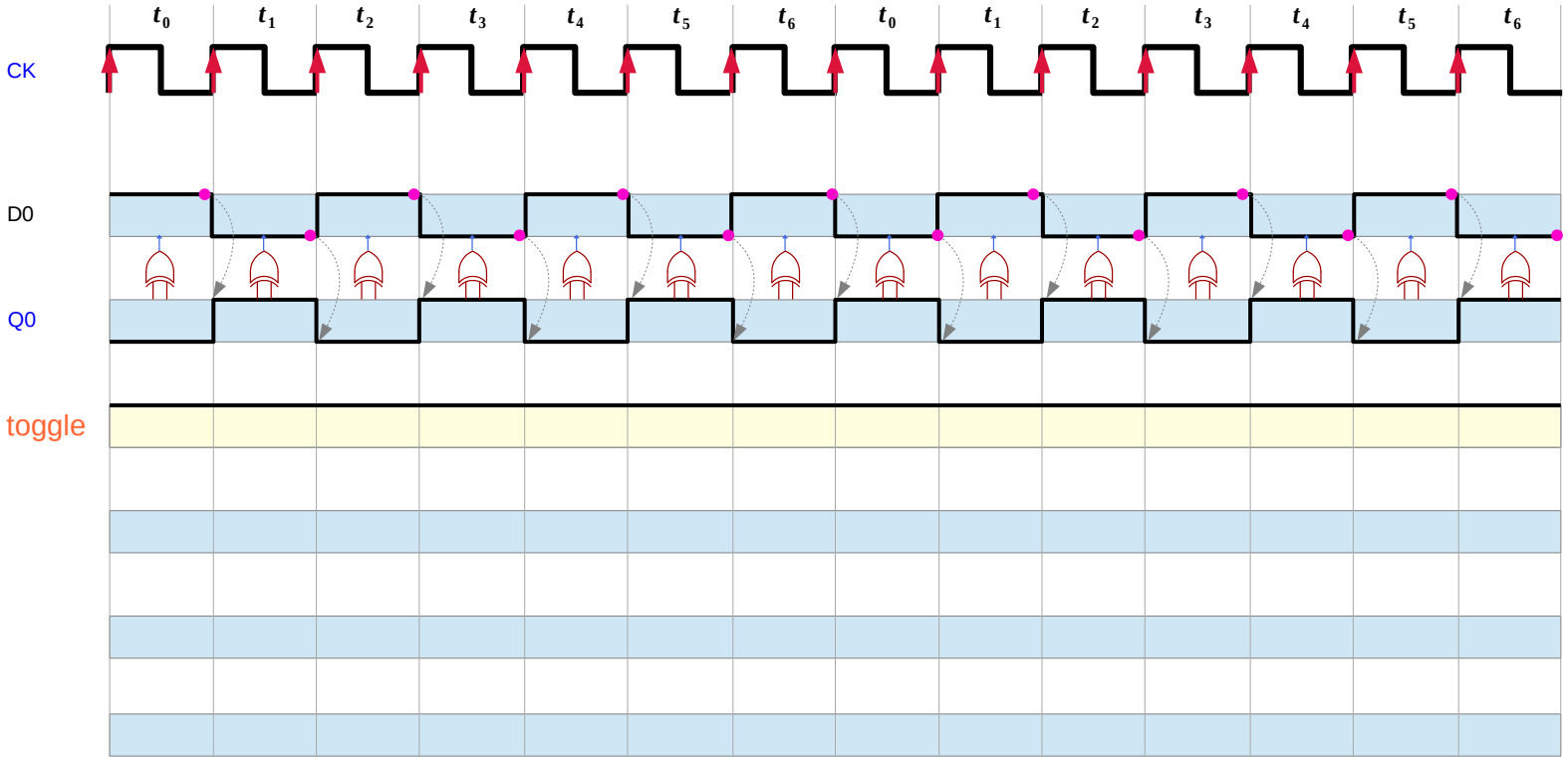




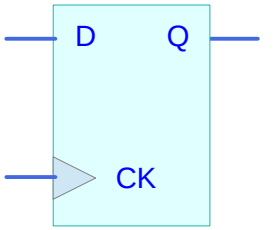


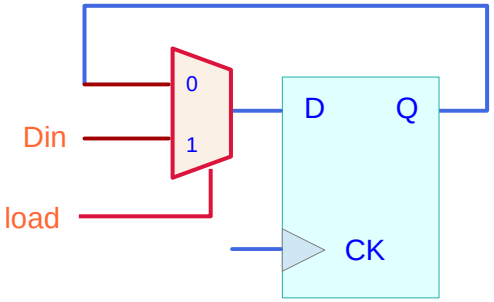


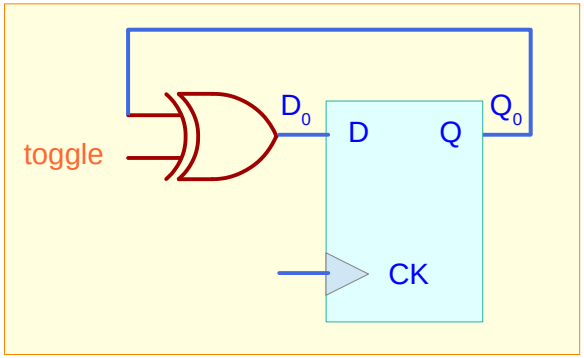
D FlipFlop with a toggle input



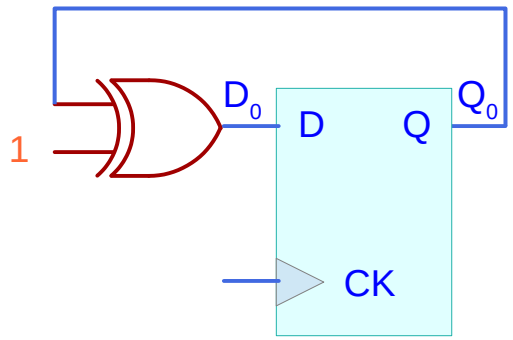
Din = $\overline{Q0}$

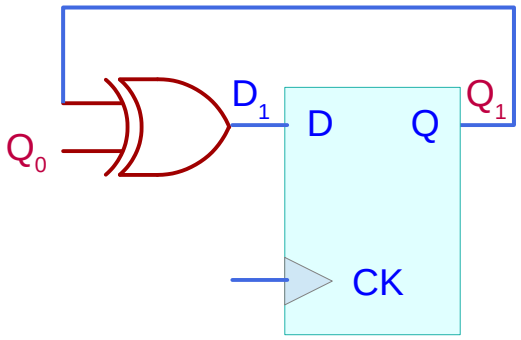


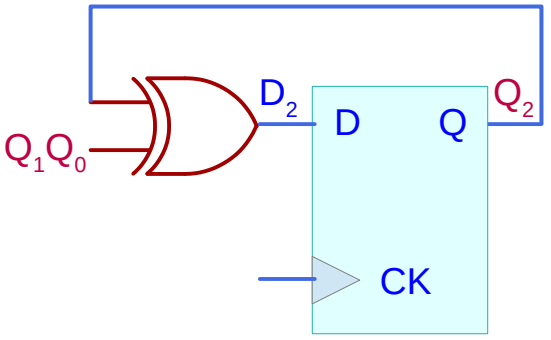




A Binary Counter using D FlipFlops (1)







$D_0 = \overline{Q}_0$

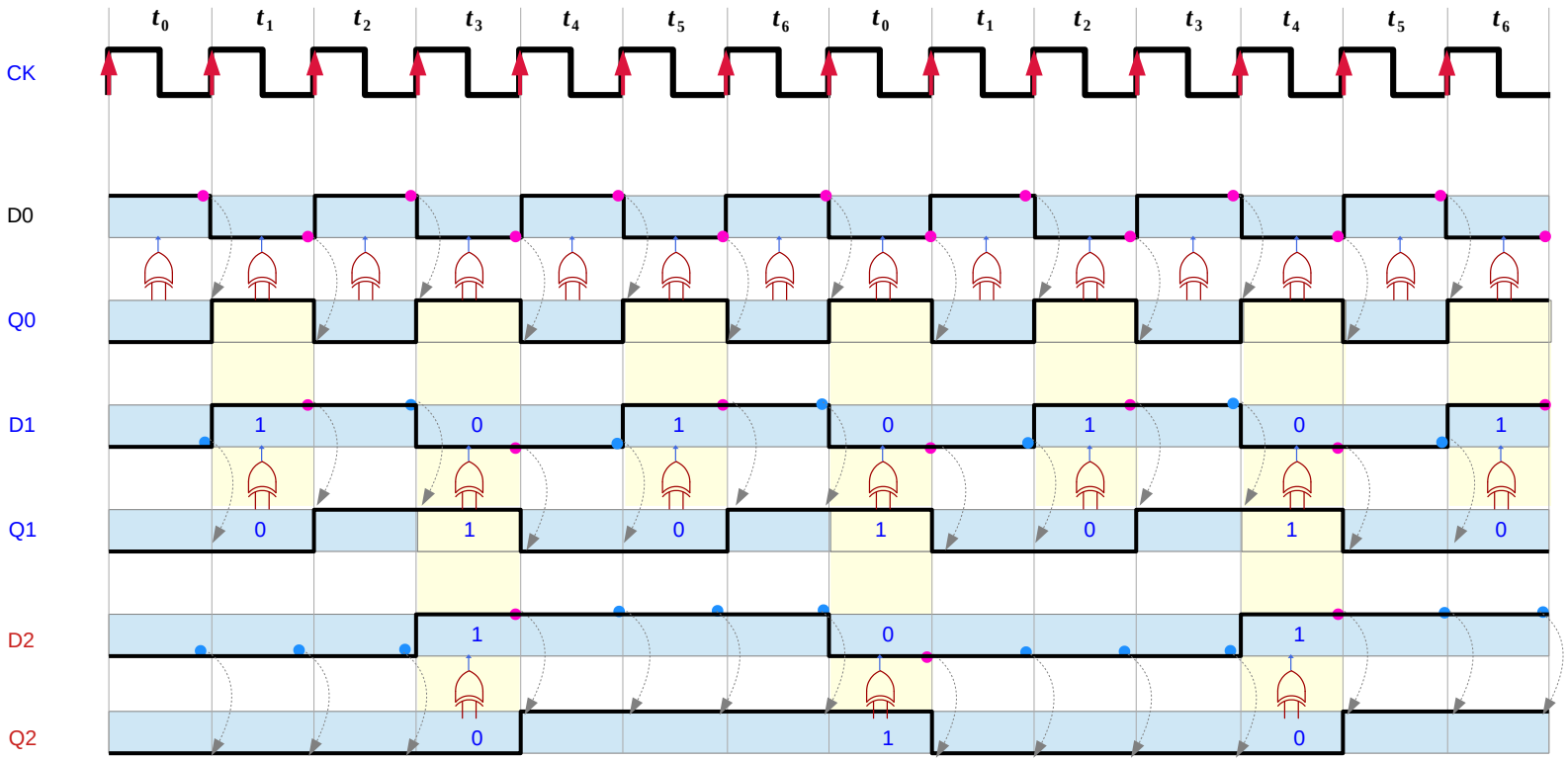
$D_1 = \overline{Q}_1$ when Q_0

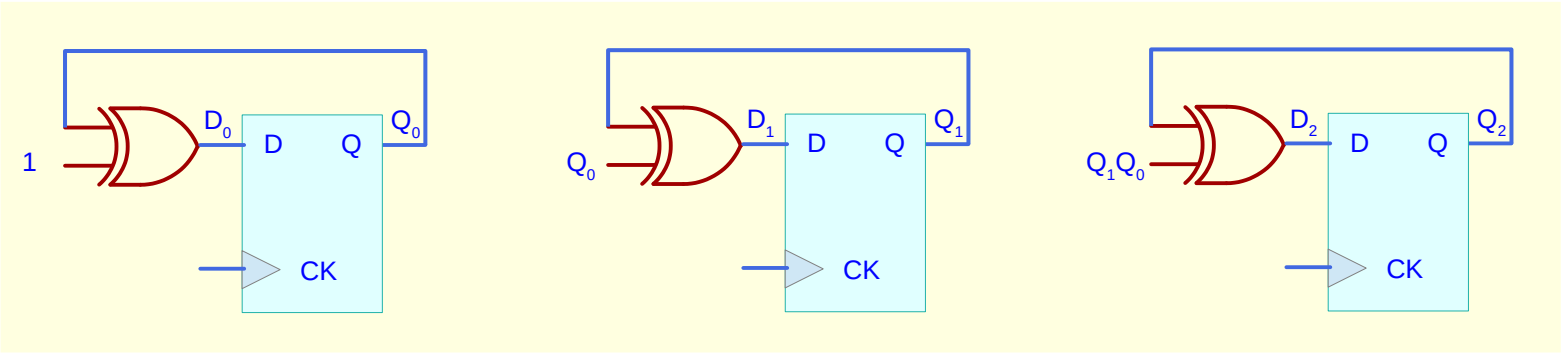
$D_1 = Q_1$ otherwise

$D_2 = \overline{Q}_2$ when Q_1Q_0

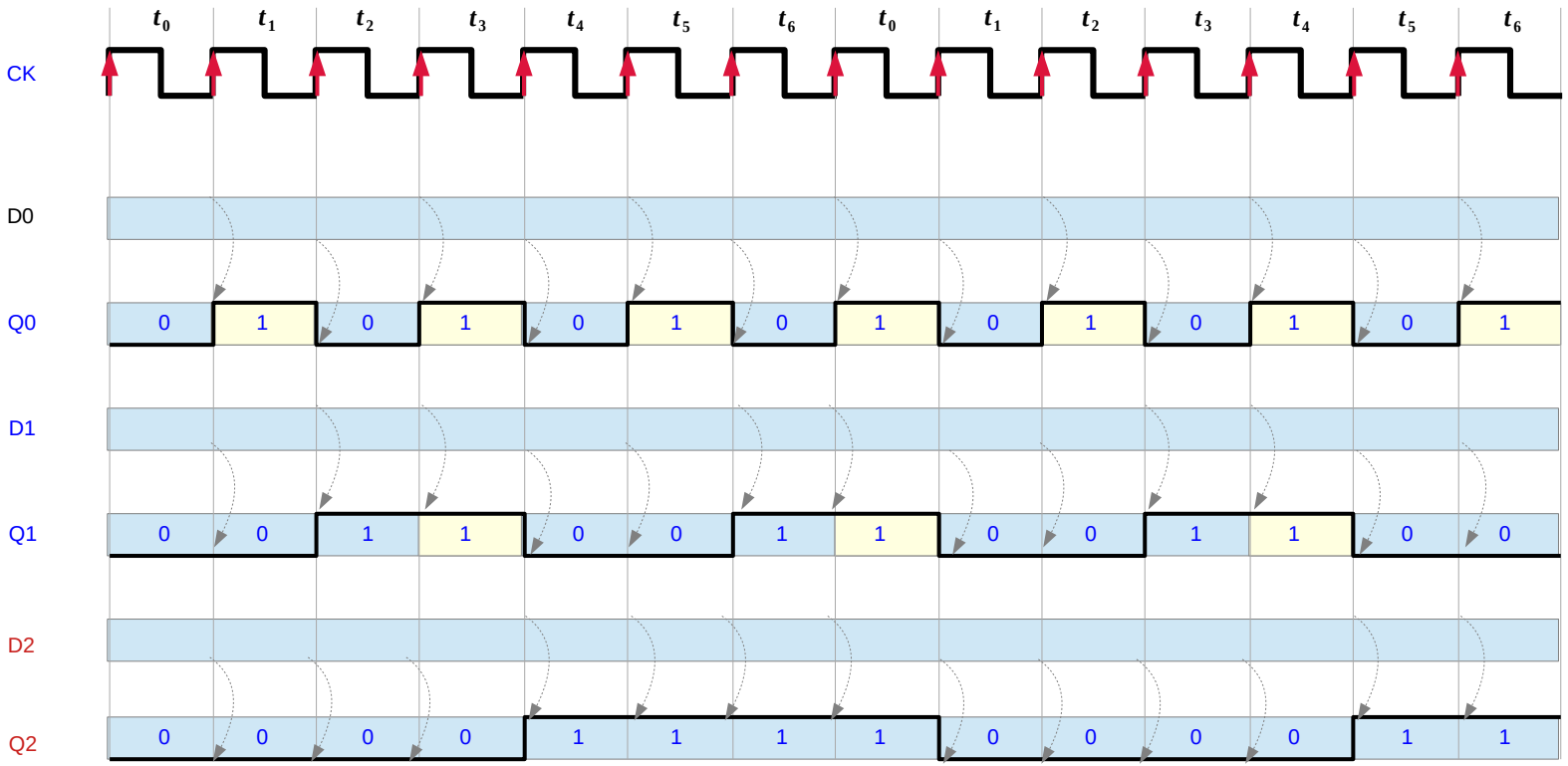
$D_2 = Q_2$ otherwise

A Binary Counter using D FlipFlops (2)





A Binary Counter using D FlipFlops (3)



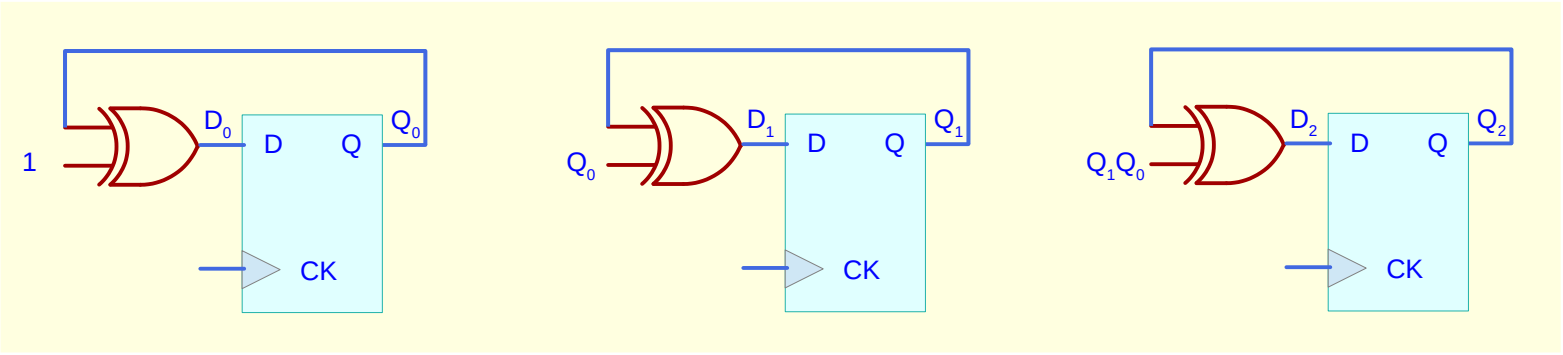
D0 = $\overline{Q0}$

D1 = $\overline{Q1}$ when Q0

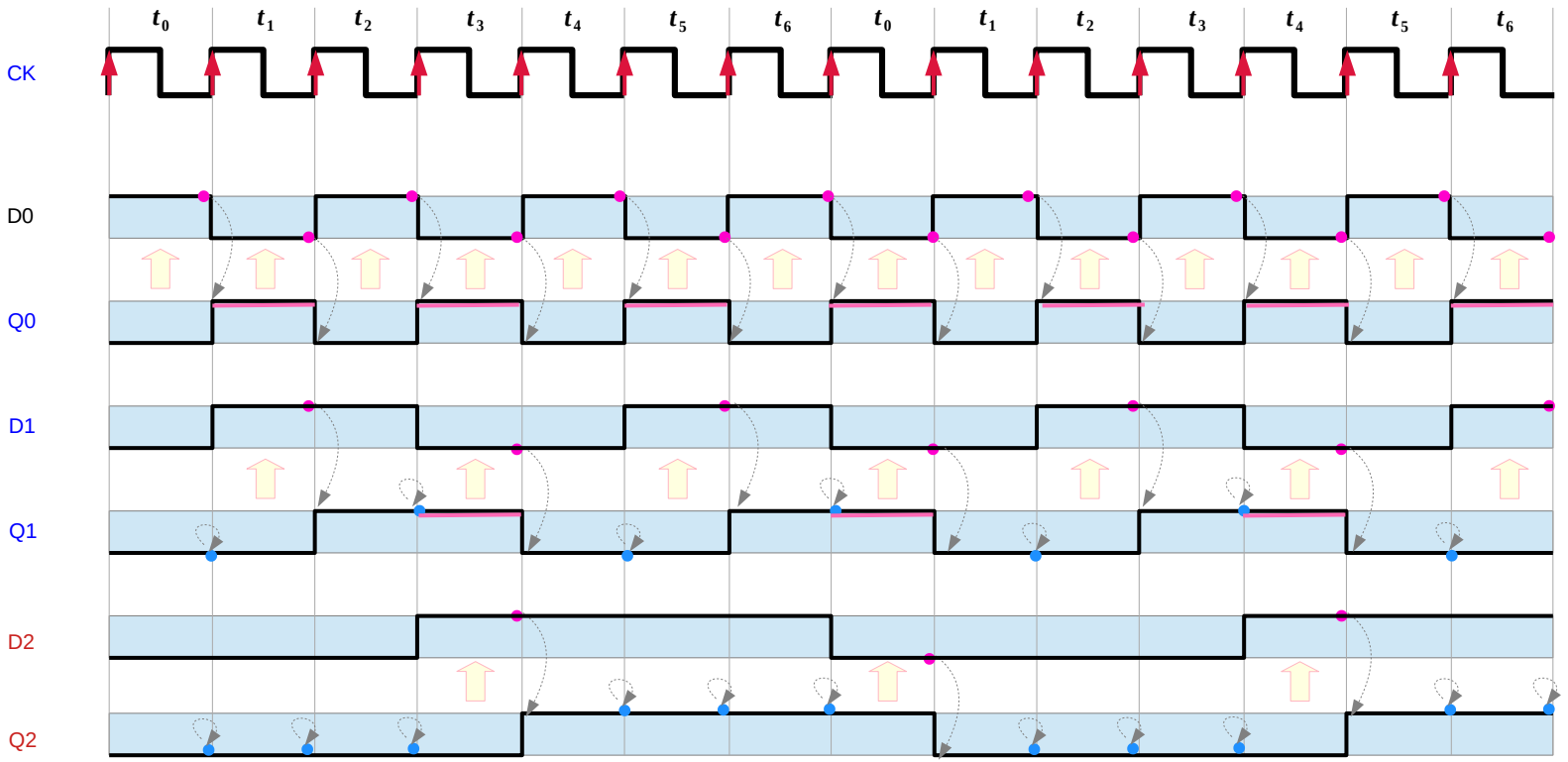
D1 = Q1 otherwise

D2 = $\overline{Q2}$ when Q1Q0

D2 = Q2 otherwise



A Binary Counter using D FlipFlops (4)



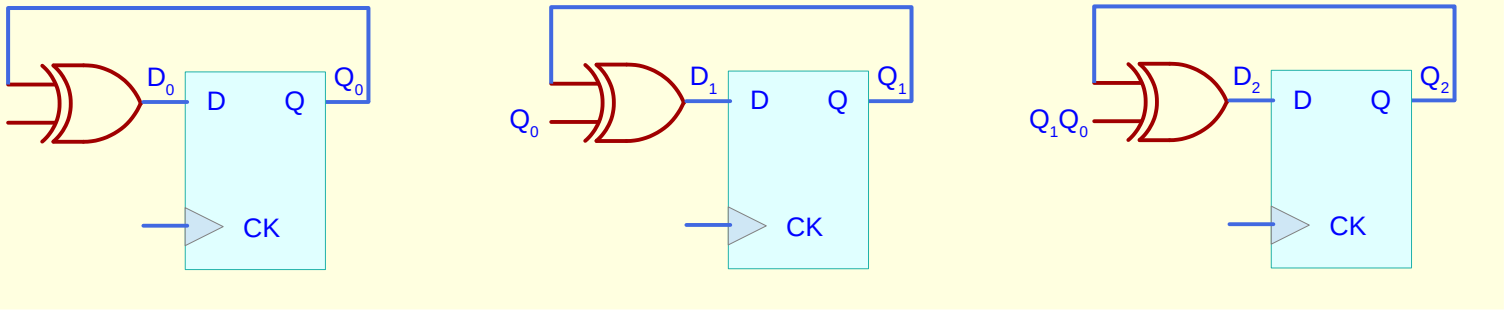
D0 = $\overline{Q0}$

D1 = $\overline{Q1}$ when Q0

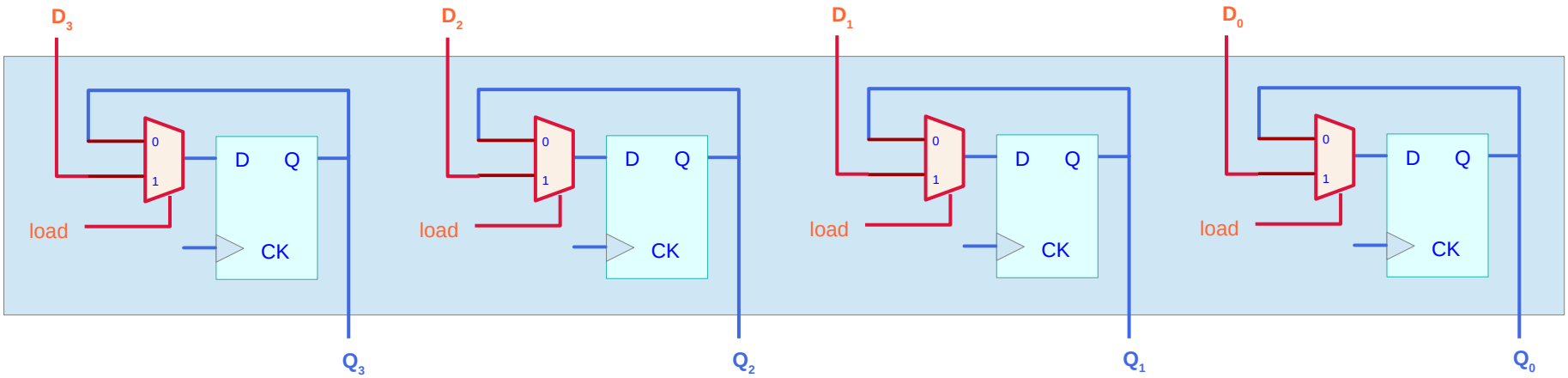
D1 = Q1 otherwise

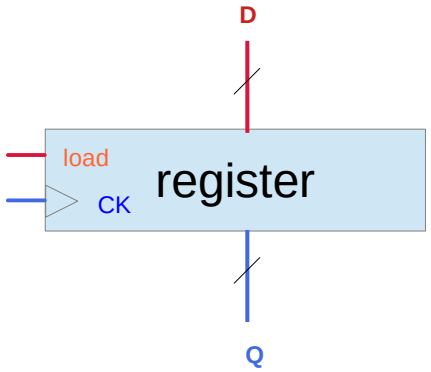
D2 = $\overline{Q2}$ when Q1Q0

D2 = Q2 otherwise

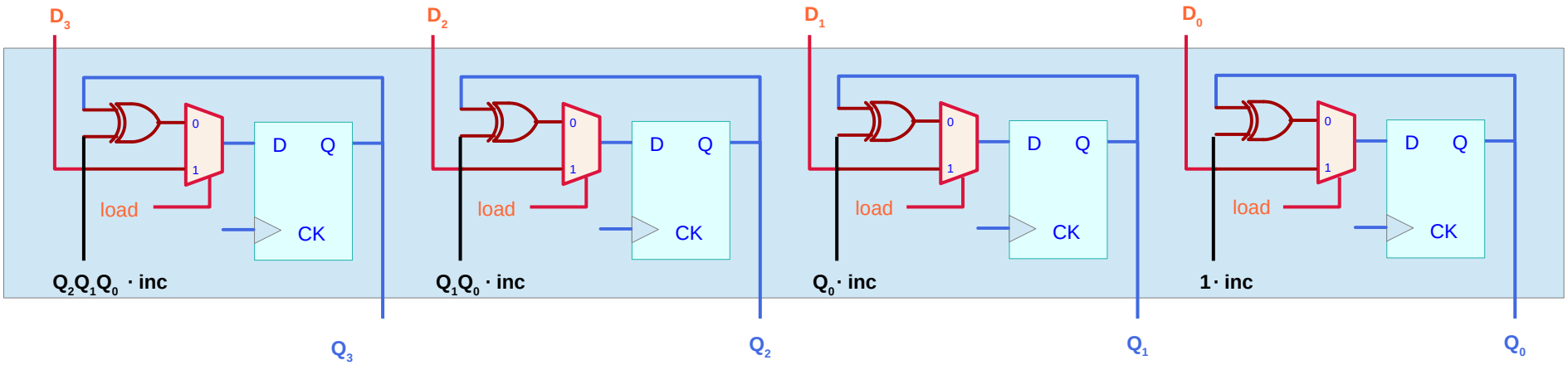


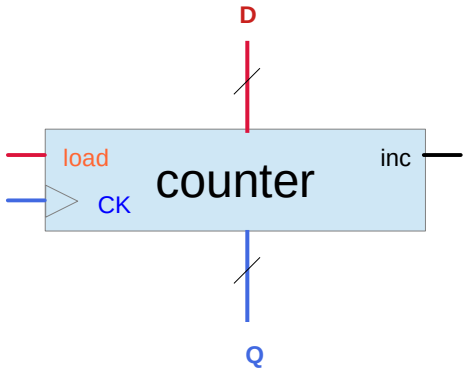
A Register using DFF



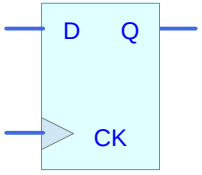


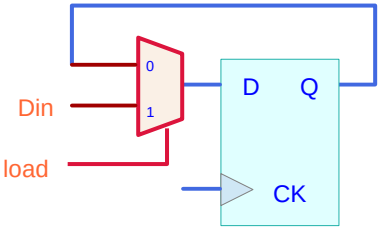
A Binary Counter using DFF

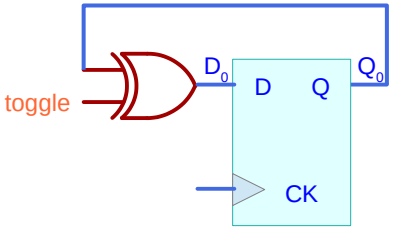


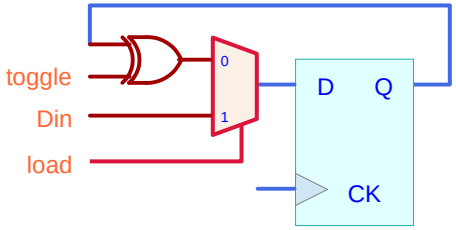


Data Path


D
Q
CK

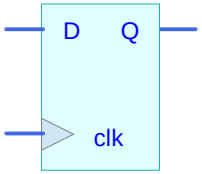

Din
load
0
1
D
Q
CK

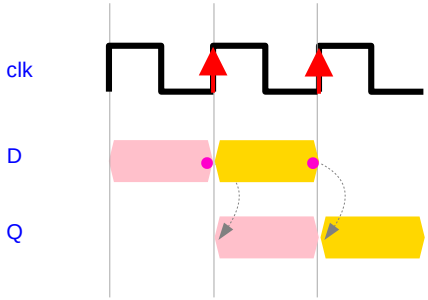

toggle
D0
Q0
D
Q
CK


toggle
Din
load
0
1
D
Q
CK

DFF

Rising Edge DFF





Ideal timing :
just before the edge, read the value of D
just after the edge, write it to Q

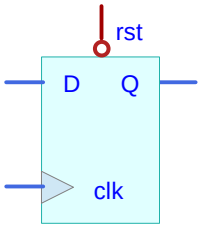
Actual Timing:
D and Q are delayed
D should not change during (t_{setup}, t_{hold}) time
$t_{edge} - t_{setup} < t < t_{edge} + t_{hold}$

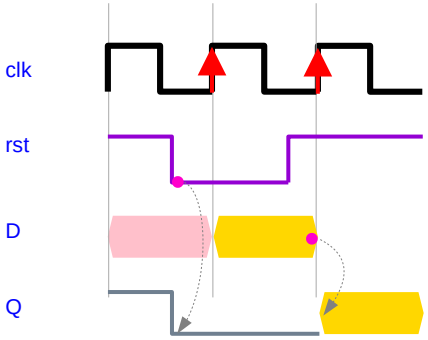
```
library ieee;
use ieee.std_logic_1164.all;

entity DFF is
        port (
                clk : in  std_logic ;
                D   : in  std_logic ;
                Q   : out  std_logic
        );
end DFF;
```

```
architecture behav of DFF is
begin
        process (clk)
        begin
                if rising_edge (clk) then
                        Q <= D;
                end if;
        end process;
end behav;
```

DFF with Asynchronous Reset

Rising Edge DFF with **Asynch Reset**

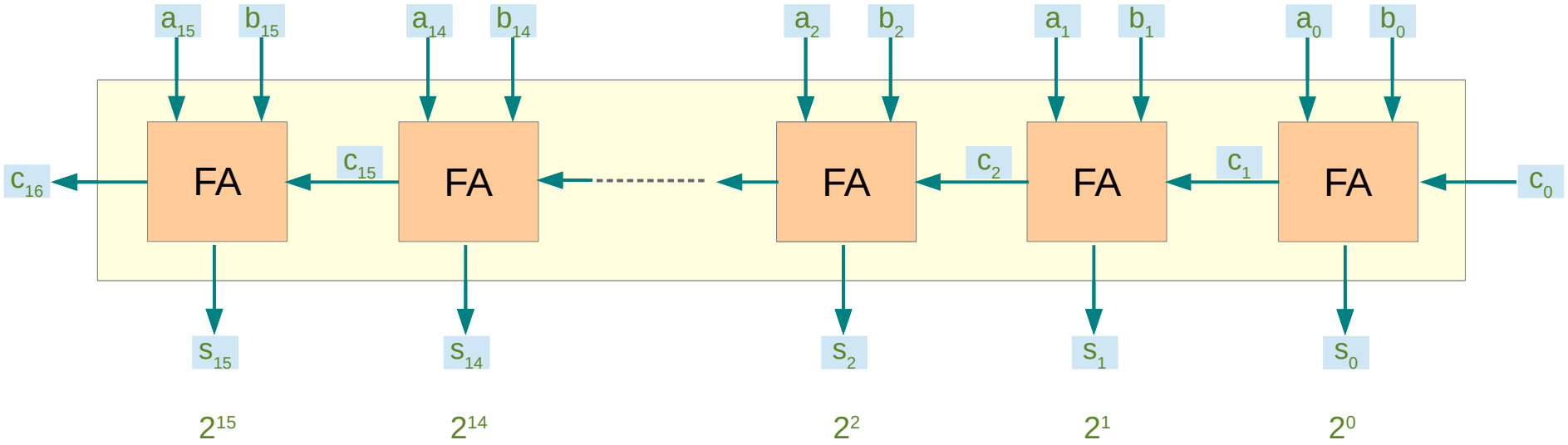




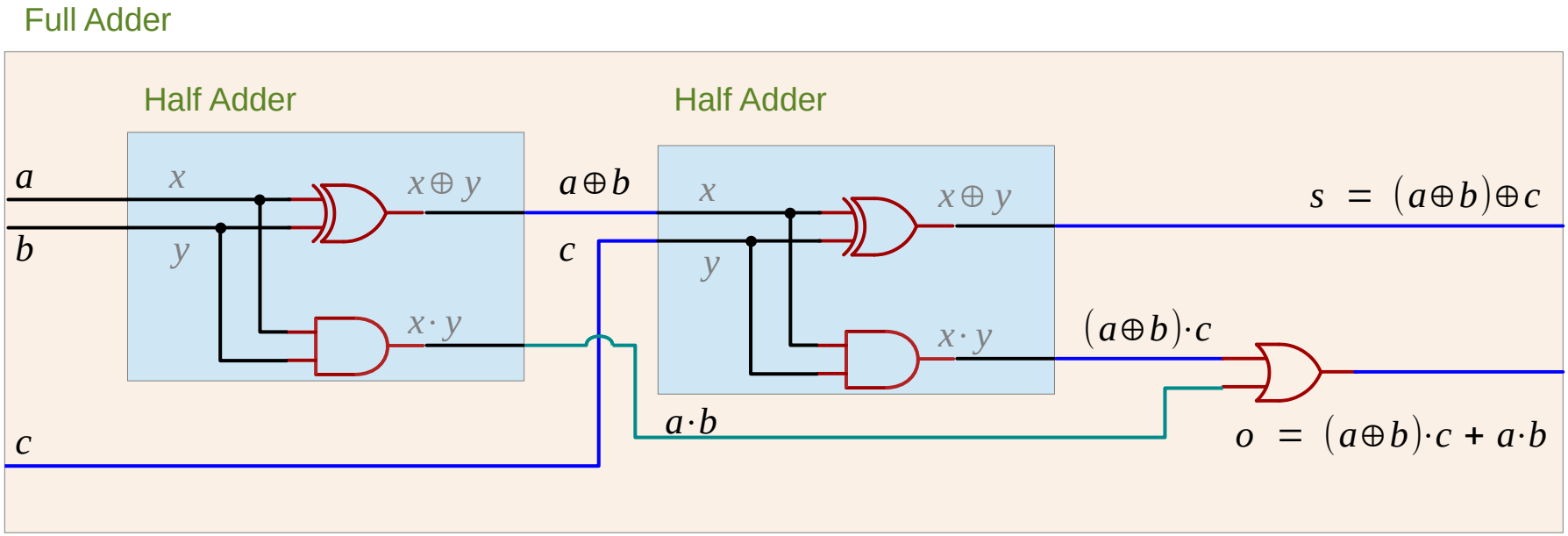

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF_rst is
        port (
                clk : in  std_logic ;
                rst : in  std_logic ;
                D   : in  std_logic ;
                Q   : out  std_logic
        );
end DFF_rst;
```

```
architecture behav of DFF_rst is
begin
        process (clk ,rst)
        begin
                if rst = '0' then
                        Q <= '0';
                elsif rising_edge (clk) then
                        Q <= D;
                end if;
        end process;
end behav;
```

ALU – Ripple Carry Adder



ALU – Full Adder



Control signals

T0	Fetch t_0 cycle
T1	Fetch t_1 cycle
T2	Fetch t_2 cycle
T3	Execution t_3 cycle
T4	Execution t_4 cycle
T5	Execution t_5 cycle
T3 · load	LOAD Execution t_3 cycle
T4 · load	LOAD Execution t_4 cycle
T5 · load	LOAD Execution t_5 cycle
T3 · sta	STA Execution t_3 cycle
T4 · sta	STA Execution t_4 cycle
T5 · sa	STA Execution t_5 cycle
T3 · add	ADD Execution t_3 cycle
T4 · add	ADD Execution t_4 cycle
T5 · add	ADD Execution t_5 cycle
T3 · jump	JUMP Execution t_3 cycle
T4 · jump	JUMP Execution t_4 cycle
T5 · jump	JUMP Execution t_5 cycle

PCin	Inbound to PC
ACin	Inbound to AC
IRin	Inbound to IR
MARin	Inbound to MAR
MBRin	Inbound to MBR
PCout	Outboud from PC
ACout	Outbound from AC
IRout	Outbound from IR
MARout	Outbound from MAR
MBRout	Outbound from MBR
PCld	To write to PC
ACld	To write to AC
IRld	To write to IR
MARld	To write to MAR
MBRld	To write to MBR

Instruction Cycles (1)

FETCH

t_0 : $MAR \leftarrow PC$

t_1 : $MBR \leftarrow M[MAR]$, $PC \leftarrow PC + 1$

t_2 : $IR \leftarrow MBR$

EXECUTION : *LOAD addr*

t_0 : $MAR \leftarrow IR(addr)$

t_1 : $MBR \leftarrow M[MAR]$

t_2 : $AC \leftarrow MBR$

EXECUTION : *ADD addr*

t_0 : $MAR \leftarrow IR(addr)$

t_1 : $MBR \leftarrow M[MAR]$

t_2 : $AC \leftarrow AC + MBR$

EXECUTION : *STA addr*

t_0 : $MAR \leftarrow IR(addr)$

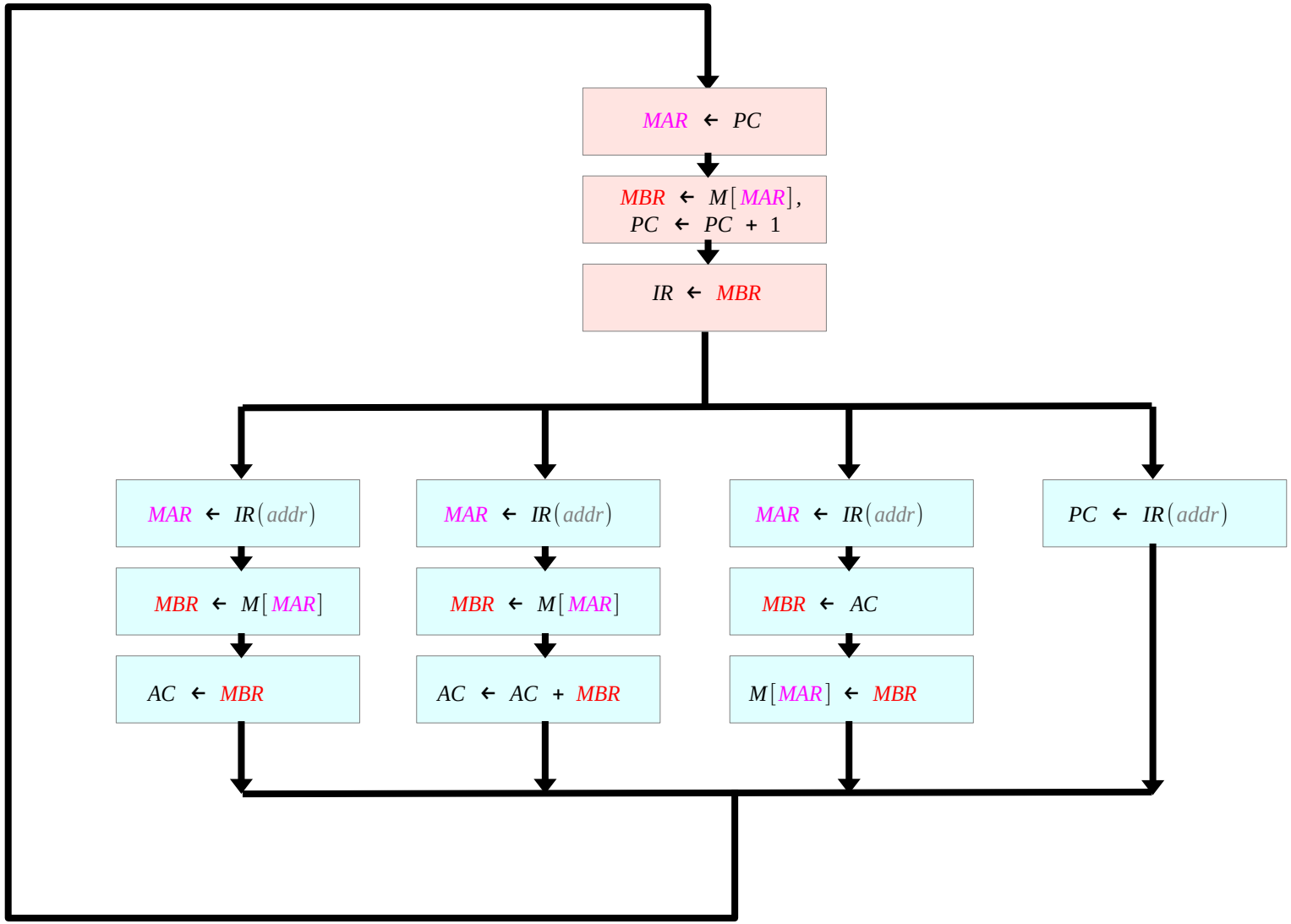
t_1 : $MBR \leftarrow AC$

t_2 : $M[MAR] \leftarrow MBR$

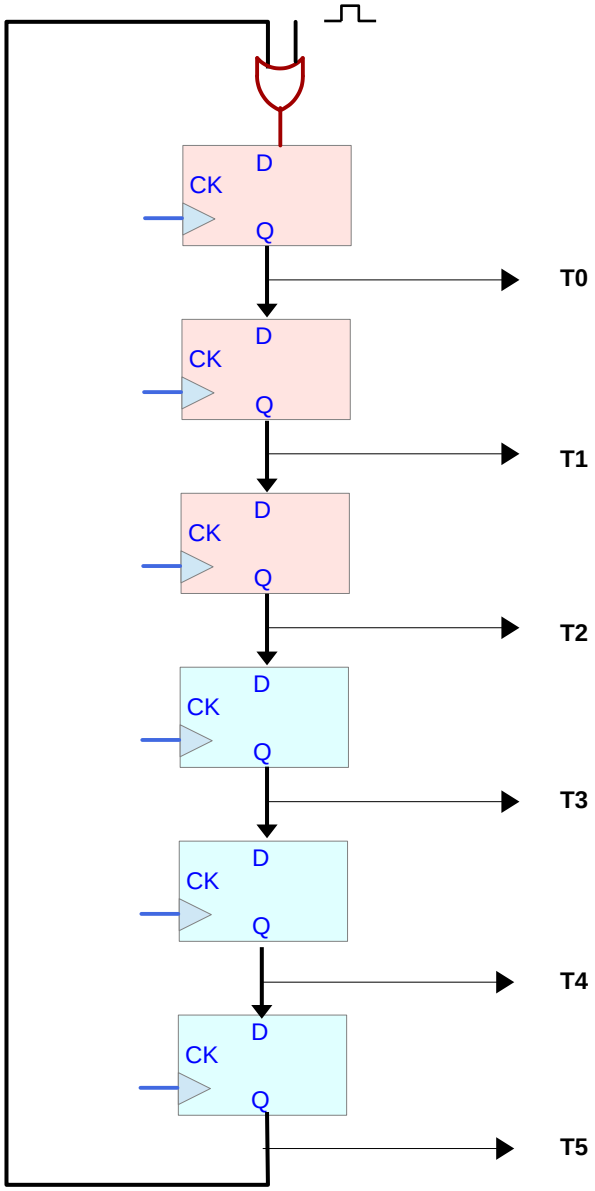
EXECUTION : *JUMP addr*

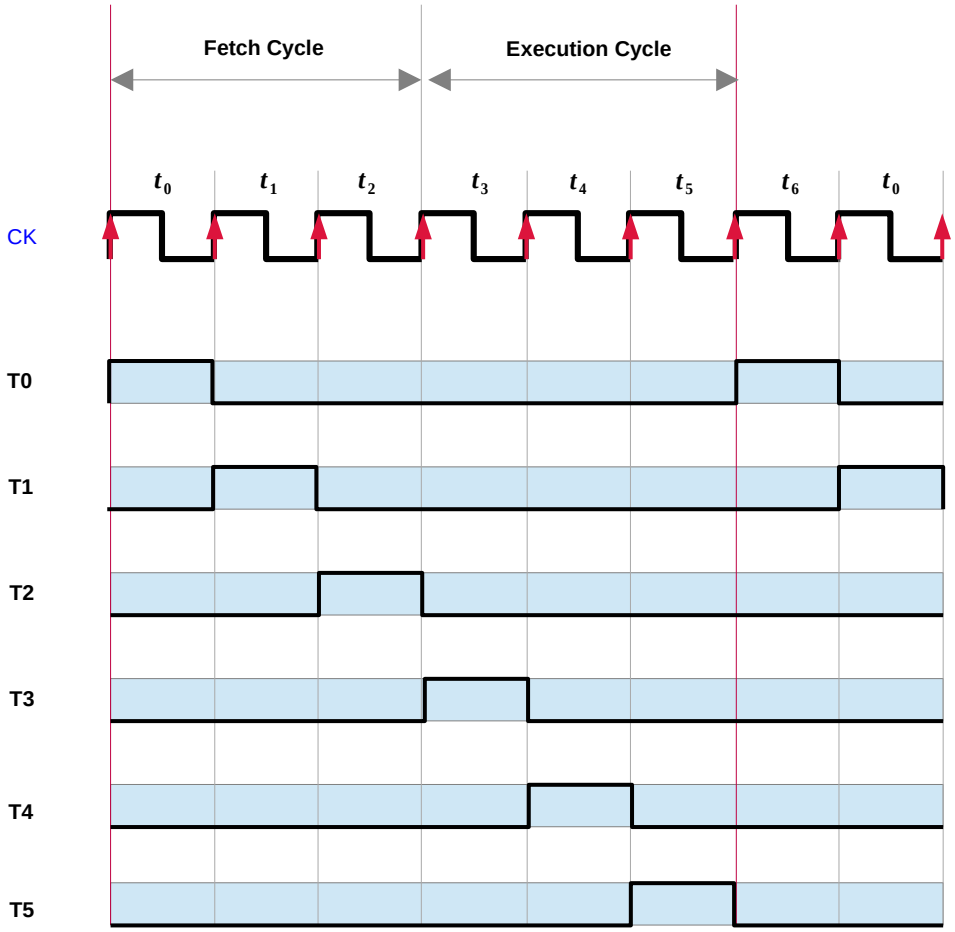
t_0 : $PC \leftarrow IR(addr)$

Instruction Cycles (2)

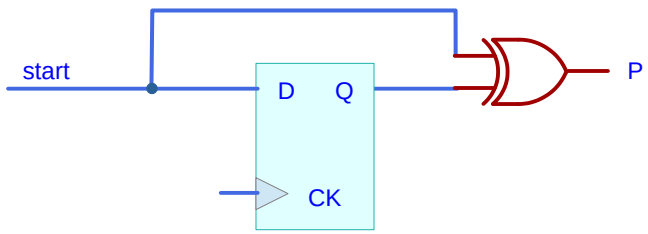


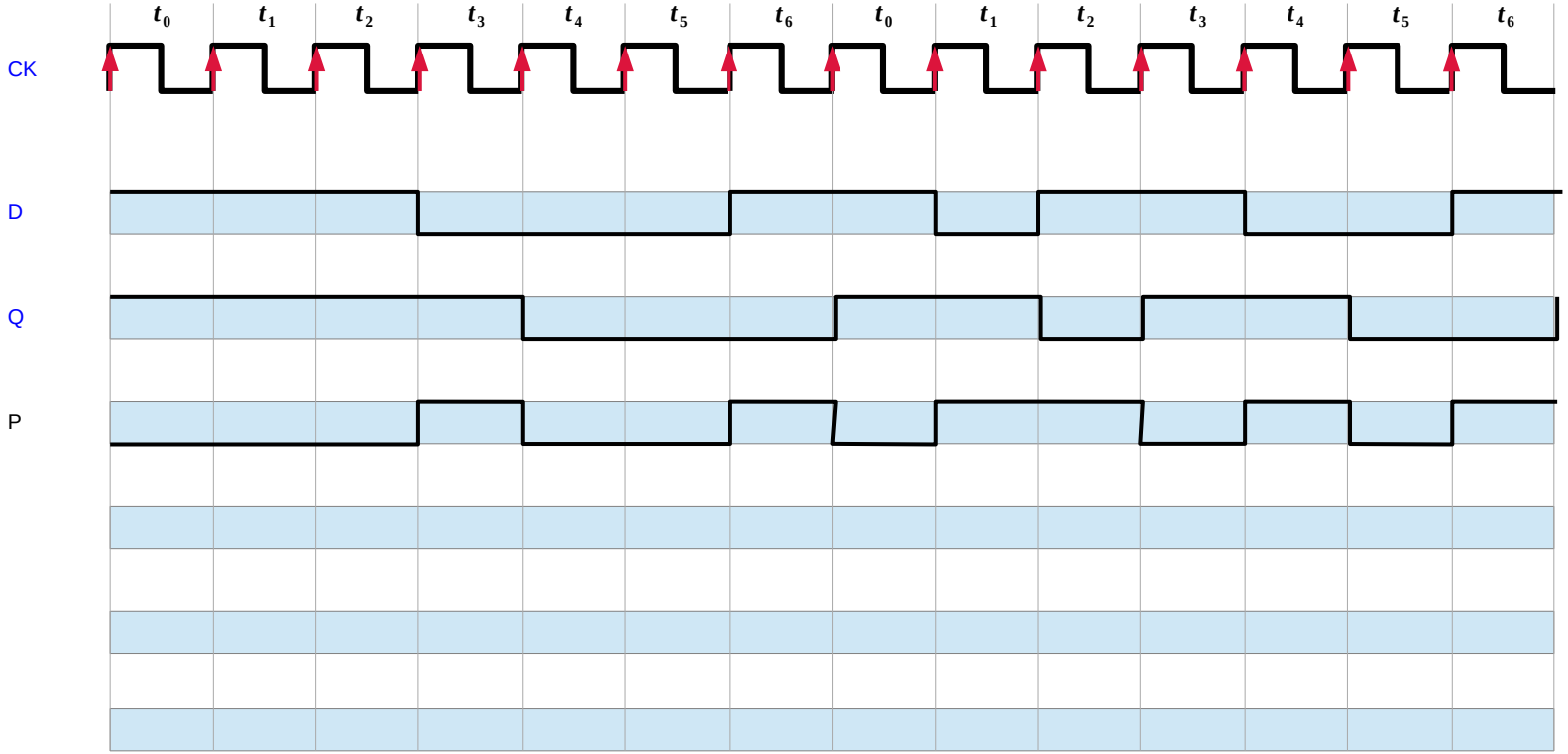
One Hot Method



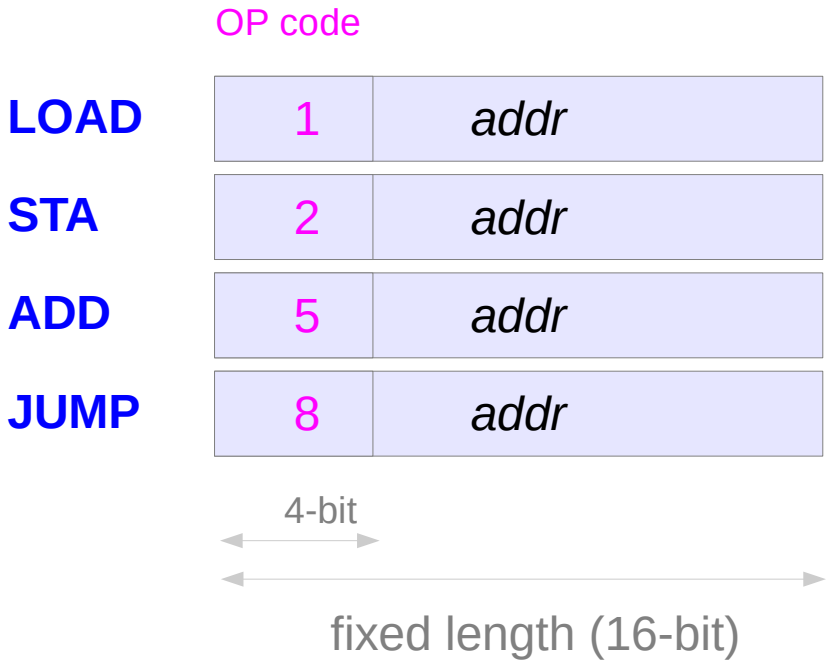


Start Pulse





Instruction Set Architecture



LOAD ***addr*** ; AC ← M[*addr*]

STA ***addr*** ; M[*addr*] ← AC

ADD ***addr*** ; AC ← AC + M[*addr*]

JUMP ***addr*** ; PC ← *addr*

Instruction Decoder

OP code

LOAD	1	*addr*	0001
STA	2	*addr*	0010
ADD	5	*addr*	0101
JUMP	8	*addr*	1000

4-bit

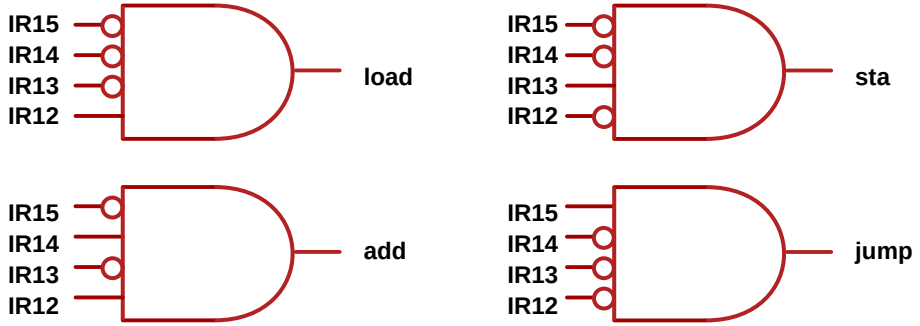
fixed length (16-bit)

load = $\overline{IR[15]} \cdot \overline{IR[14]} \cdot \overline{IR[13]} \cdot IR[12]$

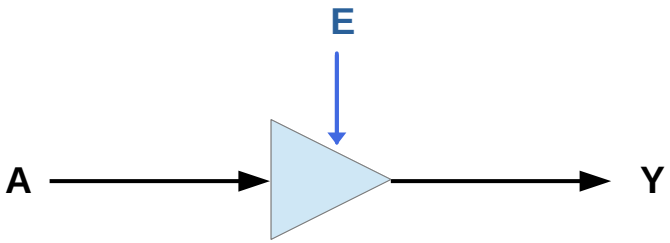
sta = $\overline{IR[15]} \cdot \overline{IR[14]} \cdot IR[13] \cdot \overline{IR[12]}$

add = $\overline{IR[15]} \cdot IR[14] \cdot \overline{IR[13]} \cdot IR[12]$

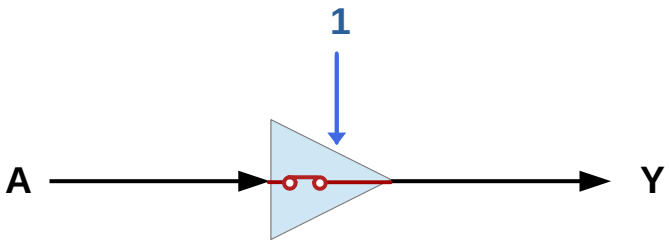
jump = $IR[15] \cdot \overline{IR[14]} \cdot \overline{IR[13]} \cdot \overline{IR[12]}$

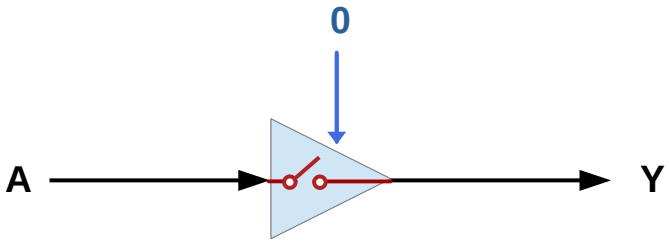


Bus using Tri-State Buffers



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1



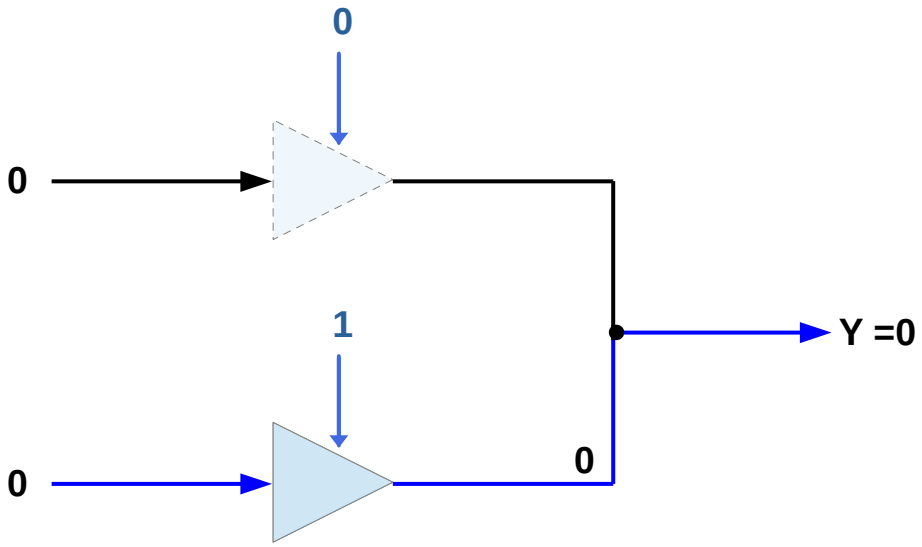


Floating, high impedance, open, high Z

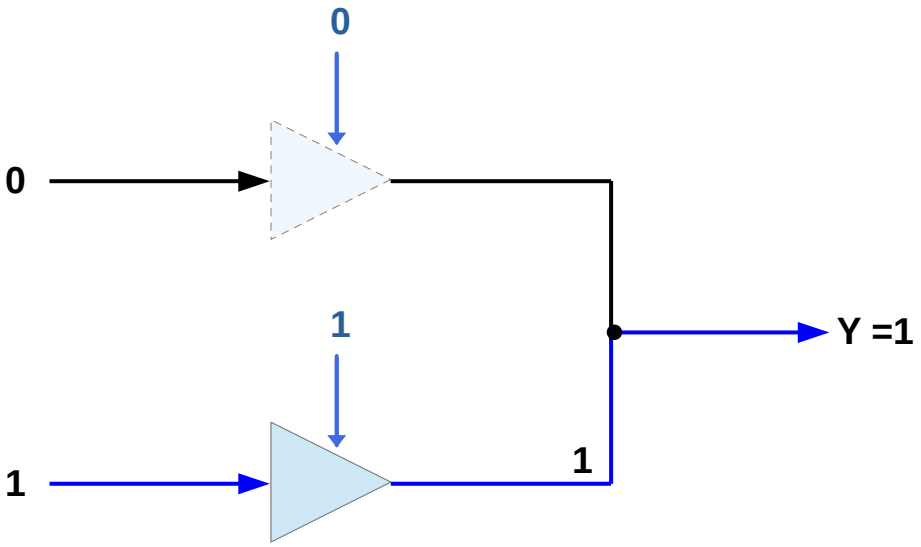
• Floating output might be
0, 1, or somewhere in between
– a voltmeter won't indicate
whether a node is floating

No contention

Bus using Tri-State Buffers



No contention

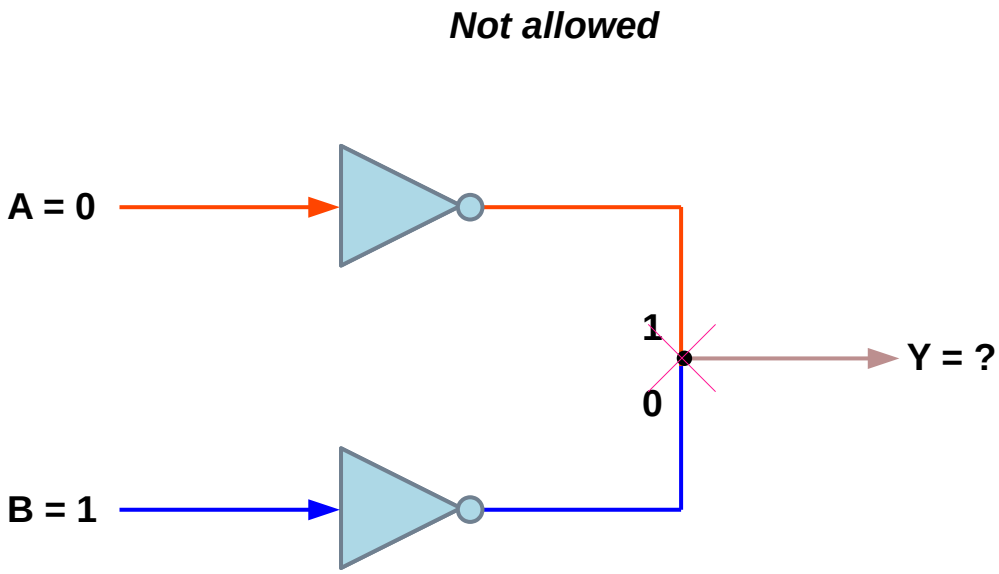


No contention

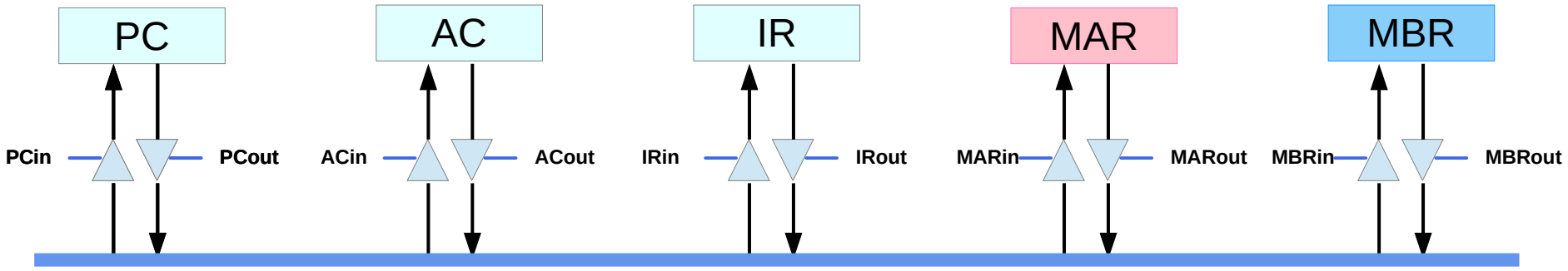
Contention X

Contention:
a circuit tries to drive output to 1 and 0
– actual value somewhere in between
– could be 0, 1, or in forbidden zone
– might change with
voltage, temperature, time, noise
– often causes excessive power dissipation

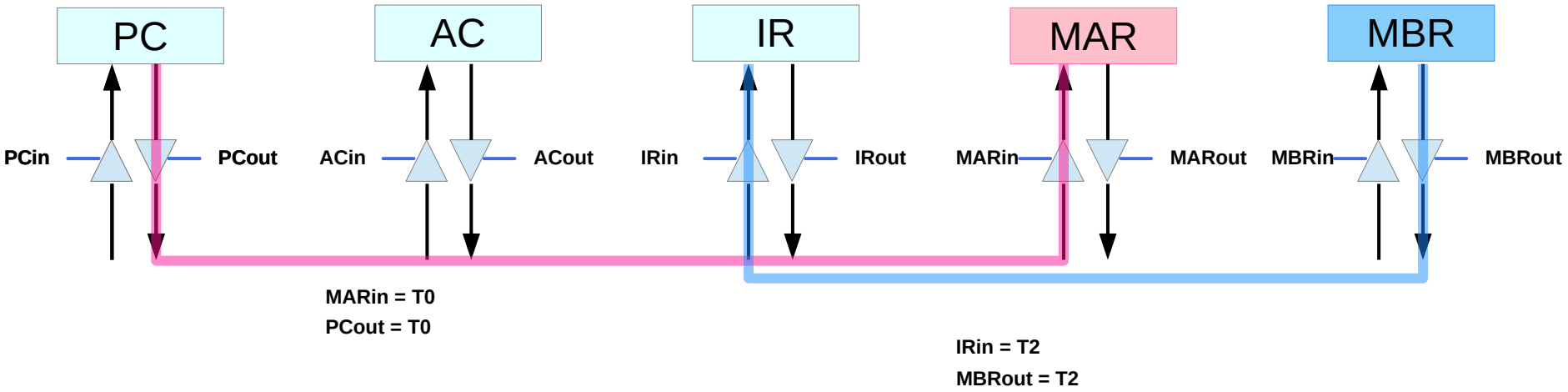
• Warnings:
– Contention usually indicates a bug.
– X is used for “don’t care” in K-map and contention - look at the context to tell them apart



Internal Bus using Tri-State Buffers



Internal Bus Examples (1)



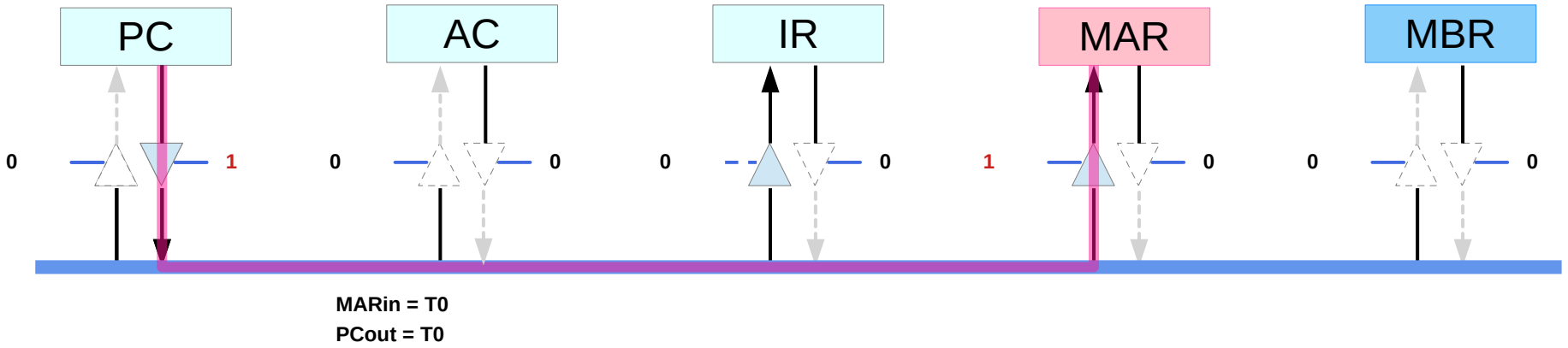
FETCH

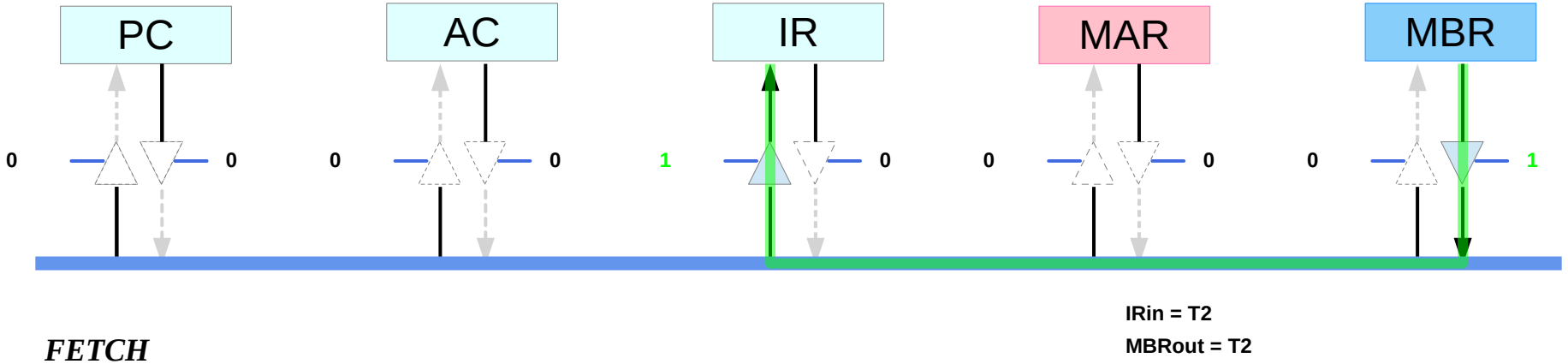
t_0 : $MAR \leftarrow PC$

t_1 : $MBR \leftarrow M[MAR],\ PC \leftarrow PC + 1$

t_2 : $IR \leftarrow MBR$

Internal Bus Examples (2)





FETCH

t_0 : $MAR \leftarrow PC$

t_1 : $MBR \leftarrow M[MAR],\ PC \leftarrow PC + 1$

t_2 : $IR \leftarrow MBR$

Instruction Cycles

FETCH

t_0 : $MAR \leftarrow PC$

t_1 : $MBR \leftarrow M[MAR],\ PC \leftarrow PC + 1$

t_2 : $IR \leftarrow MBR$

EXECUTION : *LOAD addr*

t_3 : $MAR \leftarrow IR(addr)$

t_4 : $MBR \leftarrow M[MAR]$

t_5 : $AC \leftarrow MBR$

EXECUTION : *STA addr*

t_3 : $MAR \leftarrow IR(addr)$

t_4 : $MBR \leftarrow AC$

t_5 : $M[MAR] \leftarrow MBR$

EXECUTION : *ADD addr*

t_3 : $MAR \leftarrow IR(addr)$

t_4 : $MBR \leftarrow M[MAR]$

t_5 : $AC \leftarrow AC + MBR$

EXECUTION : *JUMP addr*

t_3 : $PC \leftarrow IR(addr)$

t_0 : $MAR \leftarrow PC$

t_3 : $MAR \leftarrow IR(addr)$

t_3 : $MAR \leftarrow IR(addr)$

t_3 : $MAR \leftarrow IR(addr)$

t_1 : $MBR \leftarrow M[MAR]$

t_4 : $MBR \leftarrow M[MAR]$

t_4 : $MBR \leftarrow AC$

t_4 : $MBR \leftarrow M[MAR]$

t_5 : $AC \leftarrow MBR$

t_5 : $AC \leftarrow AC + MBR$

t_2 : $IR \leftarrow MBR\ \ FETCH$

t_5 : $M[MAR] \leftarrow MBR$

t_1 : $PC \leftarrow PC + 1$

t_3 : $PC \leftarrow IR(addr)$

t_0 : $(FETCH)$

t_3 : $(EXE\ \ LOAD)$

t_3 : $(EXE\ \ STA)$

t_3 : $(EXE\ \ ADD)$

t_1 : $(FETCH)$

t_4 : $(EXE\ \ LOAD)$

t_4 : $(EXE\ \ STA)$

t_4 : $(EXE\ \ ADD)$

t_5 : $(EXE\ \ LOAD)$

t_5 : $(EXE\ \ ADD)$

t_2 : $(FETCH)$

t_5 : $(EXE\ \ STA)$

t_1 : $(FETCH)$

t_3 : $(EXE\ \ JUMP)$

Inbound Bus Control

t_0 : MAR ← PC $(FETCH)$

t_3 : MAR ← $IR(addr)$ $(EXE\ \ LOAD)$

t_3 : MAR ← $IR(addr)$ $(EXE\ \ STA)$

t_3 : MAR ← $IR(addr)$ $(EXE\ \ ADD)$

MARId = **MARin = T0 + T3 · (load + sta + add)**

t_1 : MBR ← $M[MAR]$ $(FETCH)$

t_4 : MBR ← $M[MAR]$ $(EXE\ \ LOAD)$

t_4 : MBR ← AC $(EXE\ \ STA)$

t_4 : MBR ← $M[MAR]$ $(EXE\ \ ADD)$

MBRId = **MBRin = T1 + T4 · (load + sta + add)**

t_5 : AC ← MBR $(EXE\ \ LOAD)$

t_5 : AC ← $AC + MBR$ $(EXE\ \ ADD)$

ACId = **ACin = T5 · (load + add)**

t_2 : IR ← MBR $(FETCH)$

IRId = **IRin = T2**

t_5 : $M[MAR]$ ← MBR $(EXE\ \ STA)$

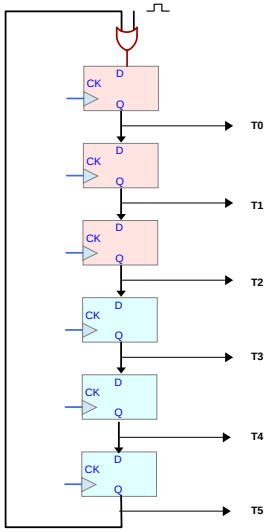
MEMwr = T5 · sta

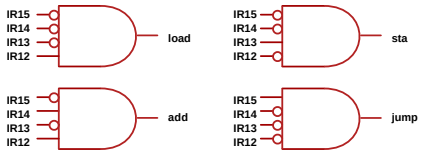
t_1 : PC ← $PC + 1$ $(FETCH)$

PCinc = T1

t_3 : PC ← $IR(addr)$ $(EXE\ \ JUMP)$

PCId = **PCin = T3 · jump**





Outbound Bus Control

$t_0 : MAR \leftarrow PC \quad (FETCH)$

$t_3 : MAR \leftarrow IR(addr) \quad (EXE\ LOAD)$
$t_3 : MAR \leftarrow IR(addr) \quad (EXE\ STA)$
$t_3 : MAR \leftarrow IR(addr) \quad (EXE\ ADD)$
$t_3 : PC \leftarrow IR(addr) \quad (EXE\ JUMP)$

$t_4 : MBR \leftarrow AC \quad (EXE\ STA)$
$t_5 : AC \leftarrow AC + MBR \quad (EXE\ ADD)$

$t_1 : MBR \leftarrow M[MAR] \quad (FETCH)$
$t_4 : MBR \leftarrow M[MAR] \quad (EXE\ LOAD)$
$t_4 : MBR \leftarrow M[MAR] \quad (EXE\ ADD)$

$t_2 : IR \leftarrow MBR \quad (FETCH)$
$t_5 : AC \leftarrow MBR + AC \quad (EXE\ ADD)$
$t_5 : AC \leftarrow MBR \quad (EXE\ LOAD)$
$t_5 : M[MAR] \leftarrow MBR \quad (EXE\ STA)$

$t_1 : PC \leftarrow PC + 1 \quad (FETCH)$

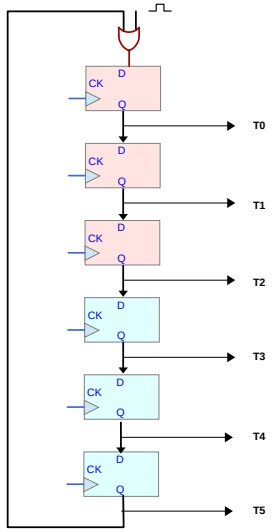
PCout = T0

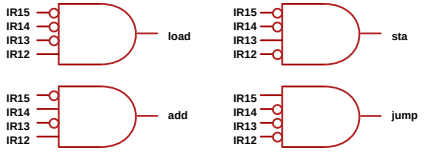
IRout = T3 · (load + sta + add + jump)

ACout = T4 · sta + T5 · add

MEMrd = T1 + T4 · (load + add)

MBRout = T2 + T5 · (add + load + sta)





Instruction Cycles

Instruction Cycles

t_0 : $MAR \leftarrow PC$ $(FETCH)$
t_3 : $MAR \leftarrow IR(addr)$ $(EXE\ \ LOAD,\ EXE\ \ STA,\ EXE\ \ ADD)$

t_1 : $MBR \leftarrow M[MAR]$ $(FETCH,\ EXE\ \ LOAD,\ EXE\ ADD)$
t_4 : $MBR \leftarrow AC\ \ EXE\ \ STA$

t_5 : $AC \leftarrow MBR$ $(EXE\ \ LOAD)$
t_5 : $AC \leftarrow AC + MBR$ $(EXE\ \ ADD)$

MARin = T0 + T3
MBRin = T1 + T
PCout = T0

t_2 : $IR \leftarrow MBR\ \ FETCH$

t_5 : $M[MAR] \leftarrow MBR$ $(EXE\ \ STA)$

t_1 : $PC \leftarrow PC + 1$ $(FETCH)$
t_3 : $PC \leftarrow IR(addr)$ $(EXE\ \ JUMP)$

References

[1] http://en.wikipedia.org/
[2] https://en.wikiversity.org/wiki/The_necessities_in_SOC_Design
[3] https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design
[4] https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design
[5] https://en.wikiversity.org/wiki/The_necessities_in_Computer_Architecture
[6] https://en.wikiversity.org/wiki/The_necessities_in_Computer_Organization
[7] https://en.wikiversity.org/wiki/Understanding_Embedded_Software
[8] Digital Systems, Hill, Peterson, 1987
[9] http://en.wikipedia.org/
[10] http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf