Sequential circuits

USTHB, 01/28/2024

1- Sequential circuits
2- **Memories**
3- Structure and Functioning of a Basic Computer

By L.ABADA

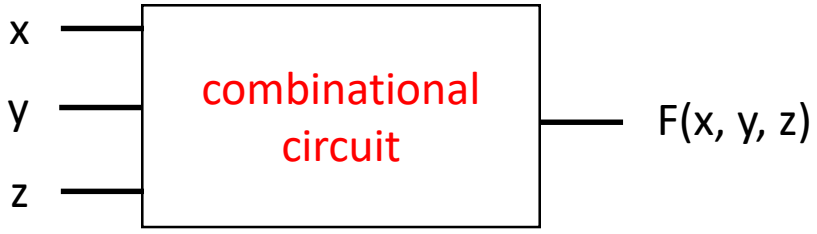
Sequential circuits

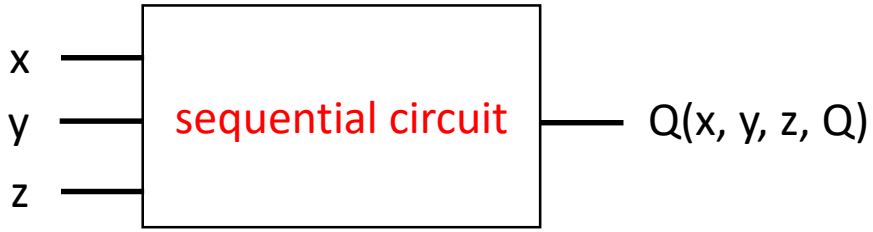
Introduction

In a combinational circuit, an output is only a function of the inputs.

On the other hand, in a sequential circuit, an output is a function of the inputs but also of the outputs of the circuit.

This means that a sequential circuit keeps the memory of past states.





Sequential circuits

Definitions

A flip-flop is a logic circuit capable of storing binary information. This information is represented by the Q output of the flip-flop (flip-flop state).

$Q_{t+1} = f(E_i, Q_t)$

Q_{t+1} Q^+ Q

Qt Q Q^-

E_i : Are the flip-flop inputs.

Q_t :is the state of the flip-flop at time t (old state).

Q_{t+1} : is the state of the flip-flop at time t+1 (new state).

A sequential circuit is an interconnection of flip-flops.

Sequential circuits

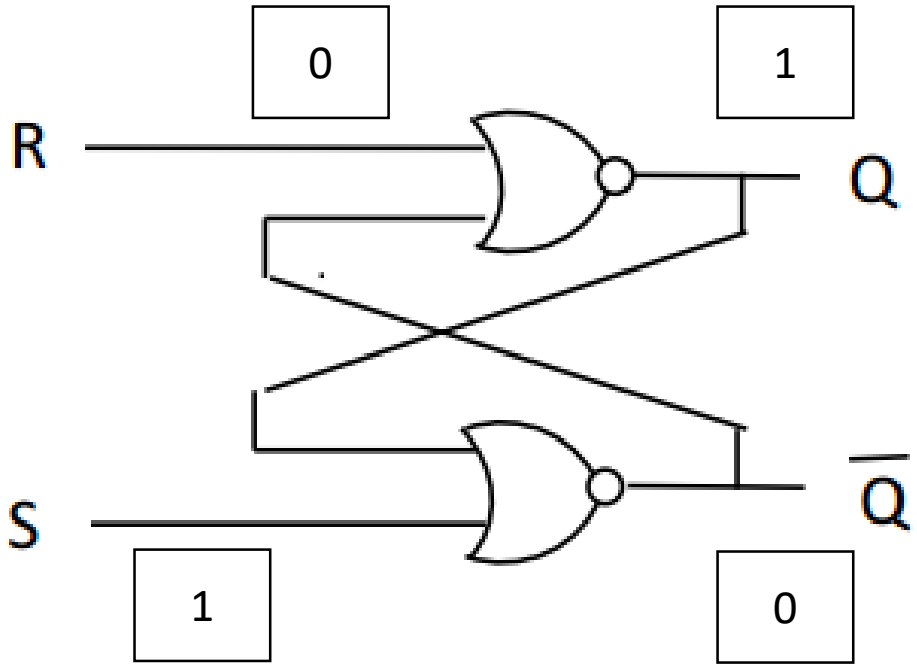
Usual flip-flops

There are four types of flip-flops :

1. **RS (or SR) Set-Reset flip-flop**
2. **JK flip-flop**
3. **D :Data or Delay flip-flop.**
4. **T : Toggle flip-flop.**

Sequential circuits

1. RS (or SR) Set-Reset flip-flop



$$Q^+ = \overline{R + \overline{Q}}$$

$$\overline{Q} = \overline{S + Q}$$

$$Q^+ = R + \overline{S + Q}$$

$$Q^+ = \overline{R}\,(S + Q)$$

$$Q^+ = \overline{R}\,S + \overline{R}\,Q$$

$$\overline{Q}^+ = \overline{S + Q}$$

$$Q = \overline{R + \overline{Q}}$$

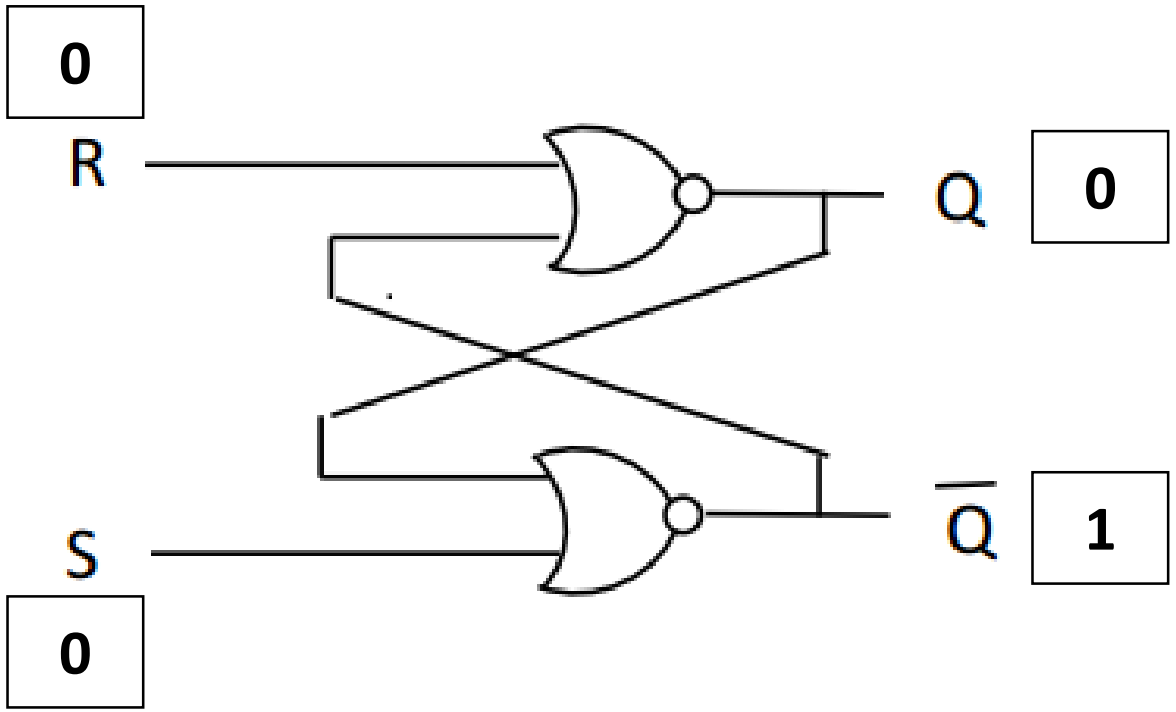
$$\overline{Q}^+ = S + R + \overline{Q}$$

$$\overline{Q}^+ = \overline{S}\,(R + \overline{Q})$$

$$\overline{Q}^+ = \overline{S}\,R + \overline{S}\,\overline{Q}$$

Sequential circuits

1. RS (or SR) Set-Reset flip-flop



R	S	Q	Q^+	$\overline{Q}^+$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

Sequential circuits

The characteristic table of the RS flip-flop is therefore :

R	S	Q^+
0	0	Q
0	1	1
1	0	0
1	1	X

Memorization

Set to 1

Set to 0 or Reset

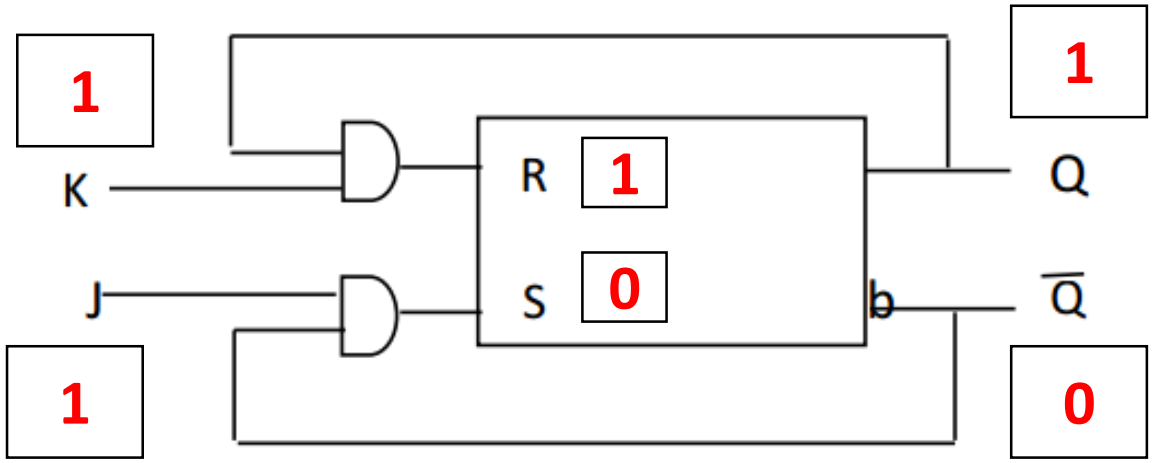
Undetermined

R	S	Q	Q^+	$\overline{Q}^+$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	X	X
1	1	1	X	X

Its output equation is $Q+ = S + \bar{R}Q$

Sequential circuits

1. JK Flip Flop



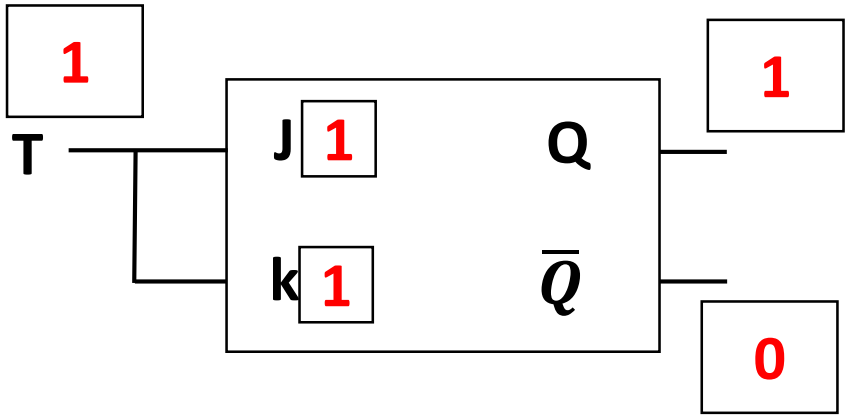
K	J	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

K	J	Q^+
0	0	Q
0	1	1
1	0	0
1	1	$\bar{Q}$

Its output equation is Q+ = J $\bar{Q}$+ $\bar{K}$ Q

Sequential circuits

1. T Flip Flop



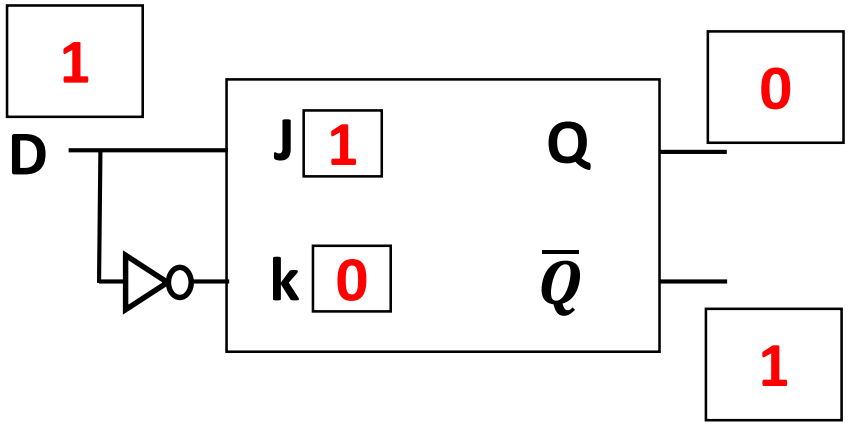
T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

T	Q^+
0	Q
1	$\overline{Q}$

Its output equation is $Q^+ = T \oplus Q$

Sequential circuits

1. D Flip Flop



D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

D	Q^+
0	0
1	1

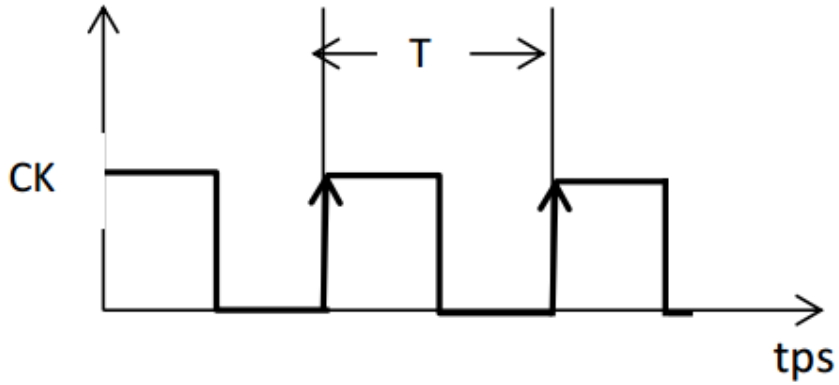
Its output equation is Q+ = D

Sequential circuits

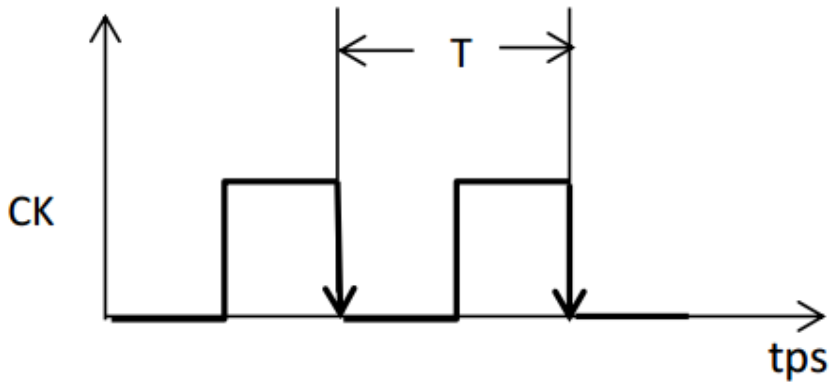
A clock is a logical variable that passes successively from 0 to 1 and from 1 to 0 periodically.

This variable is often used as an input to sequential circuits, the circuit is then said to be synchronous.

The Clock is generally noted CLK or CK



flip-flop activated on a rising edge



flip-flop activated on a falling edge

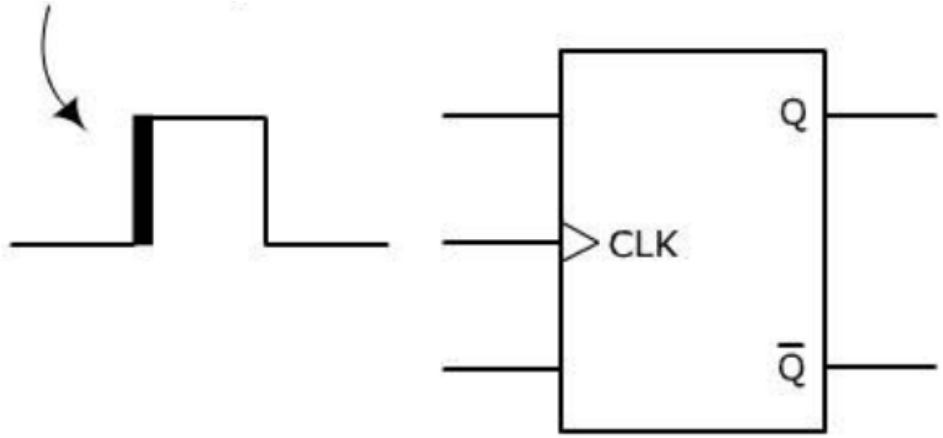
T is the period generally calculated in seconds.
F = 1/T is the frequency calculated in Hertz.

Sequential circuits

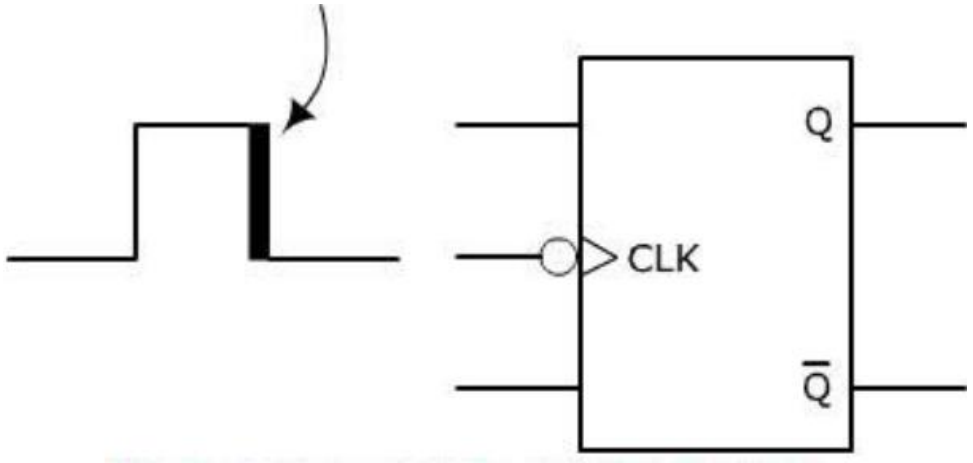
A clock is a logical variable that passes successively from 0 to 1 and from 1 to 0 periodically.

This variable is often used as an input to sequential circuits, the circuit is then said to be synchronous.

The Clock is generally noted CLK or CK



flip-flop activated on a rising edge



flip-flop activated on a falling edge

T is the period generally calculated in seconds.
F = 1/T is the frequency calculated in Hertz.

Sequential circuits

Asynchronous flip-flops.
The outputs of asynchronous flip-flops can change at any time as soon as one or more inputs change.

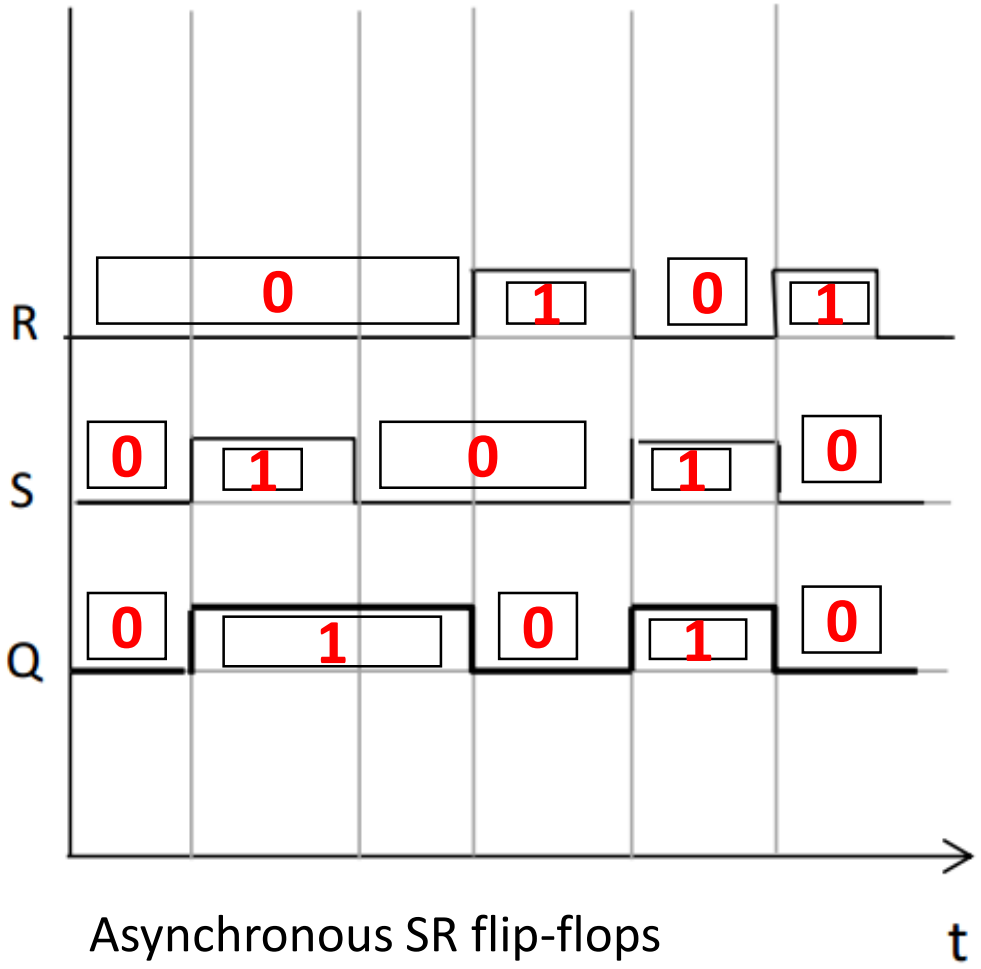
Synchronous flip-flops.

The change on the outputs occurs after the change of the clock.

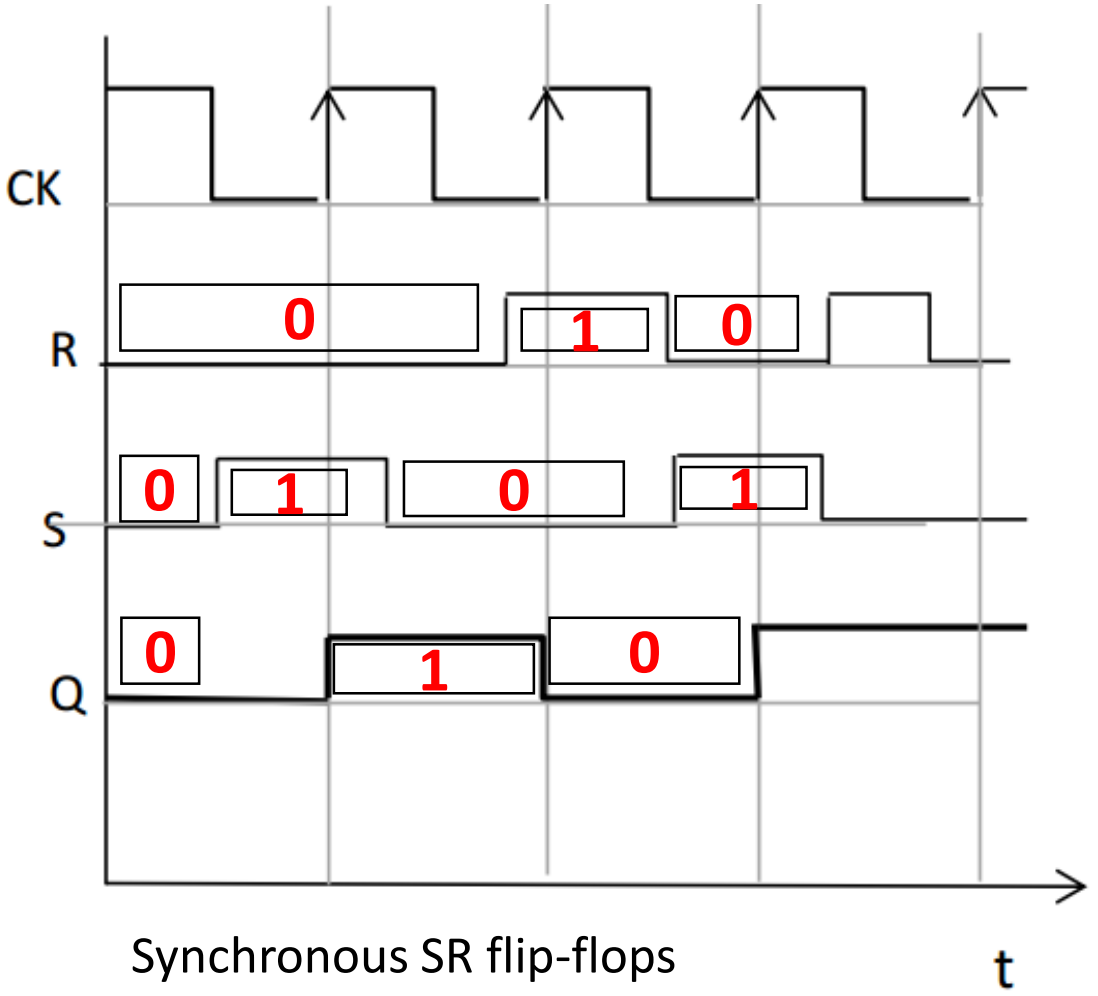
The inputs are used to prepare for the change of output, **but** do not **cause a change** in the outputs. Any change of state is synchronized **by the clock**.

Sequential circuits

Examples of synchronous and asynchronous flip-flops



Asynchronous SR flip-flops



Synchronous SR flip-flops

Sequential circuits

Analysis of a sequential circuit

To analyze a sequential circuit, we must :

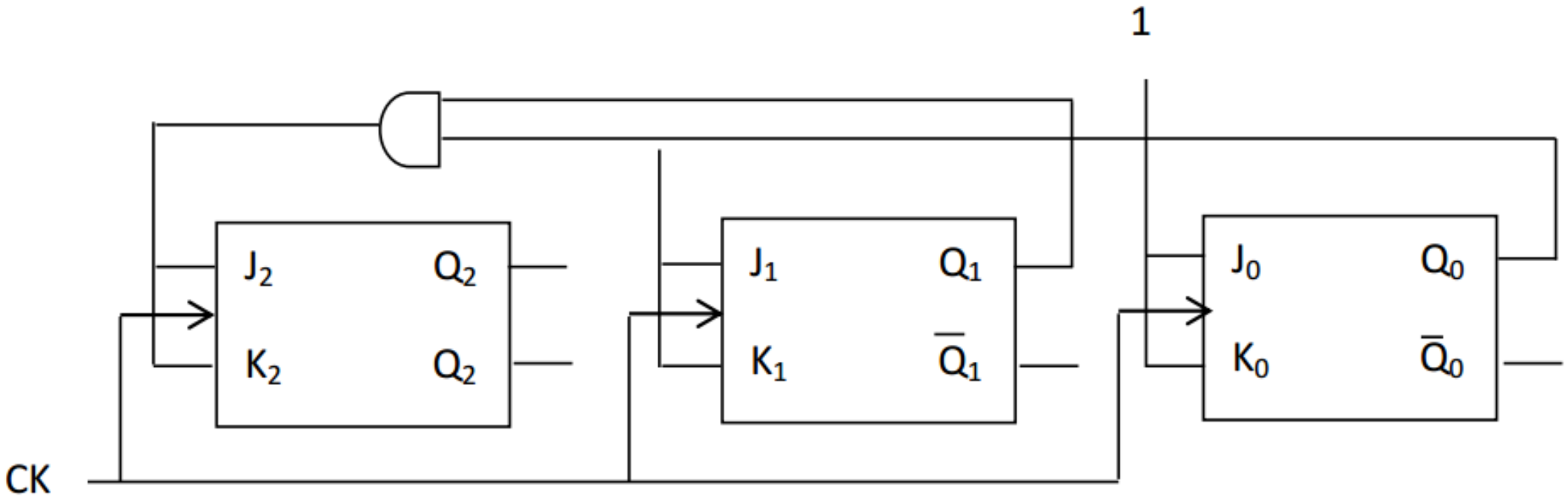
a) Establish the input equations of each flip-flop

b) Create the Truth Table of the circuit (the principle is to find the Qi+ from the values of the input equations).

c) Deduce the state diagram hence the role of the circuit. The state diagram can consist of one or more sequences

Sequential circuits

Example :



$J_2 = Q_0Q_1$ $J_1 = Q_0$ $J_0 = 1$

$K_2 = Q_0Q_1$ $K_1 = Q_0$ $K_0 = 1$

Sequential circuits

Example :

$J_2 = Q_0Q_1$

$K_2 = Q_0Q_1$

$J_1 = Q_0$

$K_1 = Q_0$

$J_0 = 1$

$K_0 = 1$

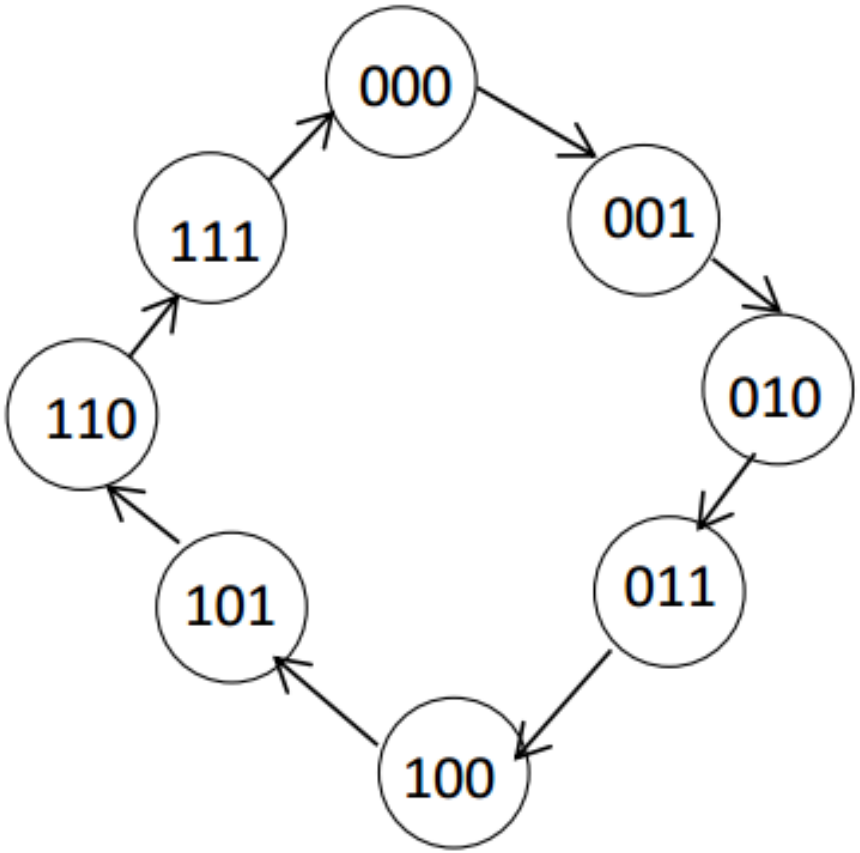
K	J	Q^+
0	0	Q
0	1	1
1	0	0
1	1	$\overline{Q}$

Truth Table

Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	0	0	0	1	1	0	0	1
0	0	1	0	0	1	1	1	1	0	1	0
0	1	0	0	0	0	0	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1	1	1	0	1
1	0	1	0	0	1	1	1	1	1	1	0
1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0	0	0

Sequential circuits

Diagramme des états



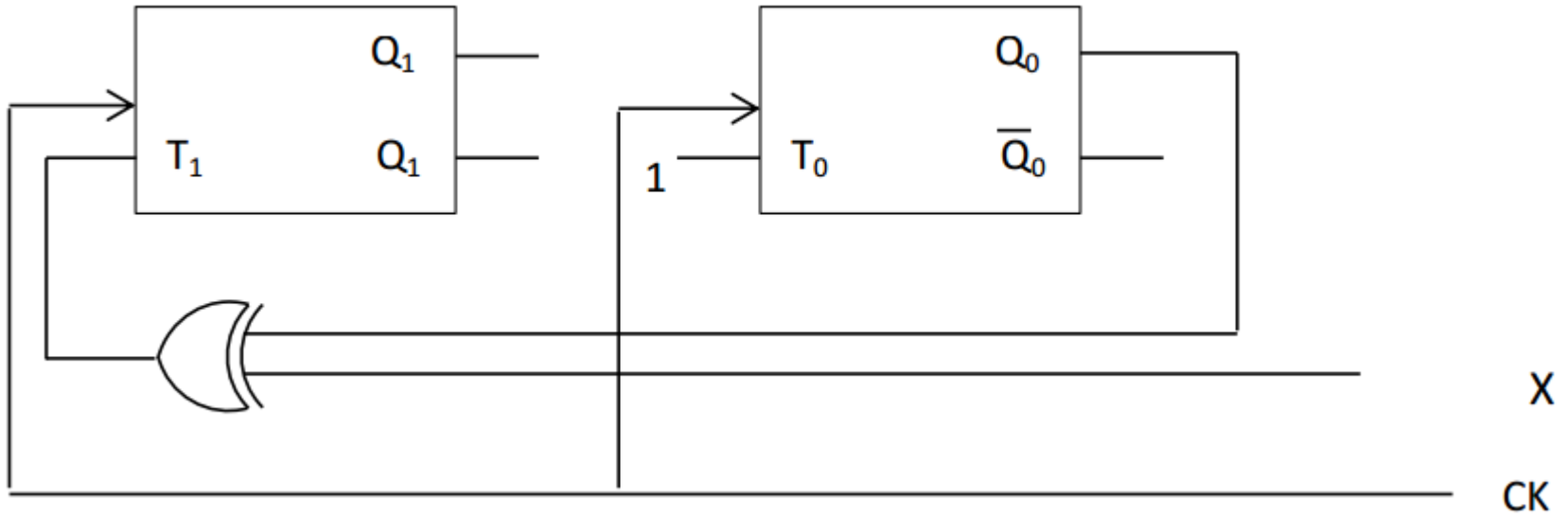
Truth Table

Q_2 Q_1 Q_0	J_2 K_2	J_1 K_1	J_0 K_0	Q_2^+ Q_1^+ Q_0^+
0 0 0	0 0	0 0	1 1	0 0 1
0 0 1	0 0	1 1	1 1	0 1 0
0 1 0	0 0	0 0	1 1	0 1 1
0 1 1	1 1	1 1	1 1	1 0 0
1 0 0	0 0	0 0	1 1	1 0 1
1 0 1	0 0	1 1	1 1	1 1 0
1 1 0	0 0	0 0	1 1	1 1 1
1 1 1	1 1	1 1	1 1	0 0 0

This circuit represents a **modulo** 8 binary up counter, it counts from 0 to 7

Sequential circuits

Example 2 :



$T_1 = Q_0 \oplus x$ $\quad$ $T_0 = 1$

Si x = 0

$T_1 = Q_0$ $\quad$ $T_0 = 1$

Si x = 1

$T_1 = \overline{Q}_0$ $\quad$ $T_0 = 1$

Sequential circuits

Example 2 :

	X	Q_1	Q_0	T_1	T_0	Q_1^+	Q_0^+
X=0	0	0	0	0	1	0	1
	0	0	1	1	1	1	0
	0	1	0	0	1	1	1
	0	1	1	1	1	0	0
X=1	1	0	0	1	1	1	1
	1	0	1	0	1	0	0
	1	1	0	1	1	0	1
	1	1	1	0	1	1	0

T	Q^+
0	Q
1	$\overline{Q}$

$T_1 = Q_0 \oplus x$ **$T_0 = 1$**

Si x = 0

$T_1 = Q_0$ $T_0 = 1$

Si x = 1

$T_1 = \overline{Q}_0$ $T_0 = 1$

Sequential circuits

Example 2 :

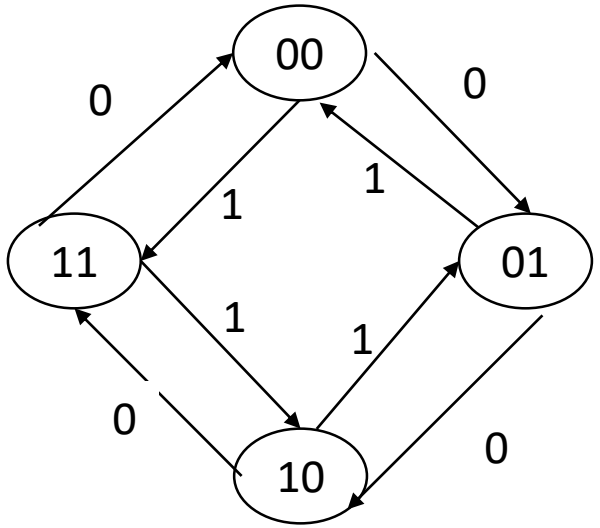
X = 0

00 → 01 → 10 → **11**

X = 1

00 → 11 → 10 → 01

	X	Q_1	Q_0	T_1	T_0	Q_1^+	Q_0^+
X=0	0	0	0	0	1	0	1
	0	0	1	1	1	1	0
	0	1	0	0	1	1	1
	0	1	1	1	1	0	0
X=1	1	0	0	1	1	1	1
	1	0	1	0	1	0	0
	1	1	0	1	1	0	1
	1	1	1	0	1	1	0



For x = 0, this circuit is **a modulo 4 up counter**, and for X = 0, it's **a modulo 4 down counter**

Sequential circuits

Flip-flop excitation tables

An excitation table (or transition table) consists of finding the input values of a flip-flop according to the variation of the output states of this flip-flop.

Sequential circuits

1. RS Flip-Flop

R	S	Q	Q^+	
0	0	0	0	R=0 S=0
0	0	1	1	R=0 S=0
0	1	0	1	R=0 S=1
0	1	1	1	R=0 S=1
1	0	0	0	R=1 S=0
1	0	1	0	R=1 S=0
1	1	0	X	
1	1	1	X	

Excitation table (R S)

Q	Q^+	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

Sequential circuits

2. JK Flip-Flop

K	J	Q	Q^+	
0	0	0	0	
0	0	1	1	J = 0 K=0
0	1	0	1	
0	1	1	1	J = 1 K=0
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	0	

excitation table (J K)

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Sequential circuits

3. T Flip-Flop

T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

excitation table (T)

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

Sequential circuits

4. D Flip-Flop

D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

excitation table (D)

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

Sequential circuits

Synthesis of a sequential circuit :

The synthesis of a sequential circuit consists of finding the input functions of this circuit from a state diagram (sequence). For that it is necessary :

a) Establish the state diagram (or sequence) and give the number of flip-flops necessary.

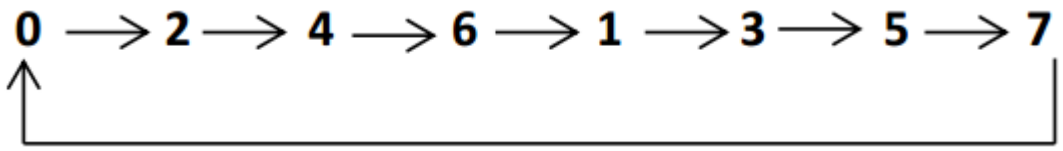
b) Create the transition table

c) Deduce the input equations to the flip-flops

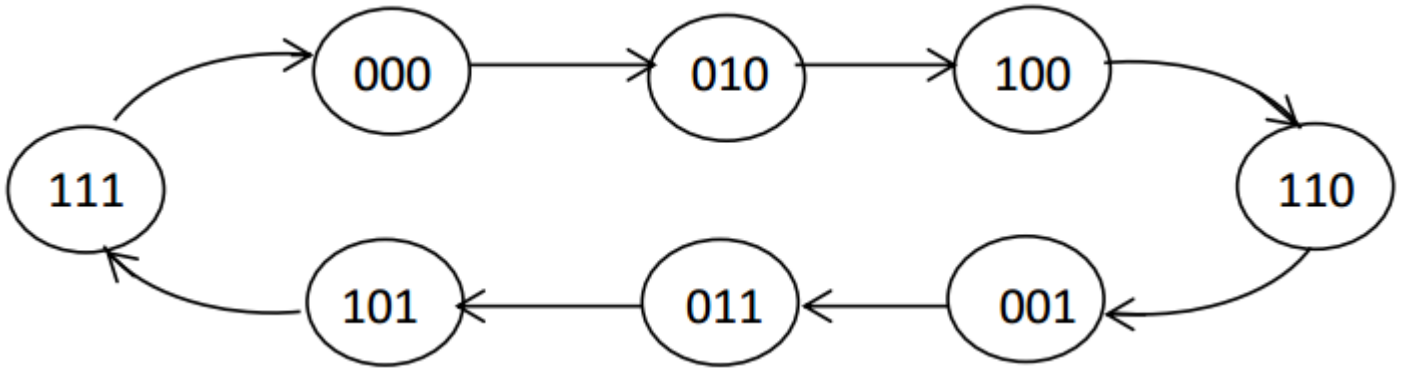
d) Create the corresponding circuit

Sequential circuits

Example :

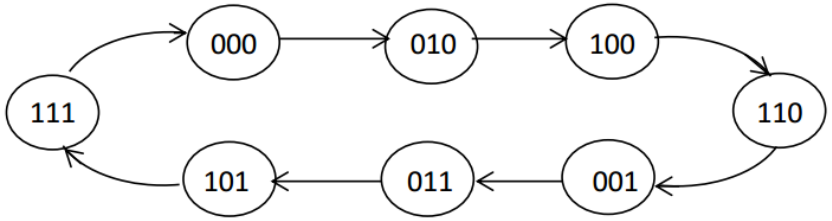


Sequence



Sequential circuits

Transition table



Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q_2 Q_1 Q_0	Q_2^+ Q_1^+ Q_0^+	J_2 K_2	J_1 K_1	J_0 K_0
0 0 0	0 1 0	0 X	1 X	0 X
0 0 1	0 1 1	0 X	1 X	X 0
0 1 0	1 0 0	1 X	X 1	0 X
0 1 1	1 0 1	1 X	X 1	X 0
1 0 0	1 1 0	X 0	1 X	0 X
1 0 1	1 1 1	X 0	1 X	X 0
1 1 0	0 0 1	X 1	X 1	1 X
1 1 1	0 0 0	X 1	X 1	X 1

Transition table

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	1	0	0	1	X	X	1	0	X
0	1	1	1	0	1	1	X	X	1	X	0
1	0	0	1	1	0	X	0	1	X	0	X
1	0	1	1	1	1	X	0	1	X	X	0
1	1	0	0	0	1	X	1	X	1	1	X
1	1	1	0	0	0	X	1	X	1	X	1

$J_2 = Q_1$ $J_1 = 1$ $J_0 = Q_2Q_1$

$K_2 = Q_1$ $K_1 = 1$ $K_0 = Q_2Q_1$

Q_2 \ Q_1Q_0	0 0	0 1	1 1	1 0
0	0	0	1	1
1	X	X	X	X

$J_2 = Q_1$

Q_2 \ Q_1Q_0	0 0	0 1	1 1	1 0
0	X	X	X	X
1	0	0	1	1

$K_2 = Q_1$

Q_2 \ Q_1Q_0	0 0	0 1	1 1	1 0
0	0	X	0	X
1	0	X	X	1

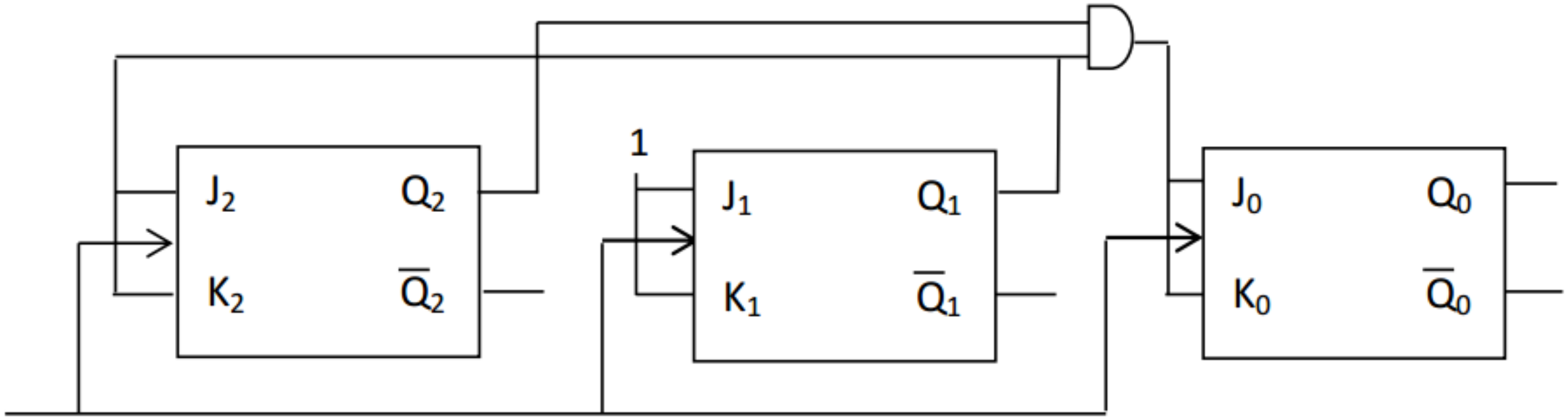
$J_0 = Q_2Q_1$

Q_2 \ Q_1Q_0	0 0	0 1	1 1	1 0
0	X	0	0	X
1	X	0	1	X

$K_0 = Q_2Q_1$

Sequential circuits

Circuit :



Sequential circuits

Example 2 (Diagram with equivalent states)

In a state diagram, it sometimes happens that two or more states are equivalent

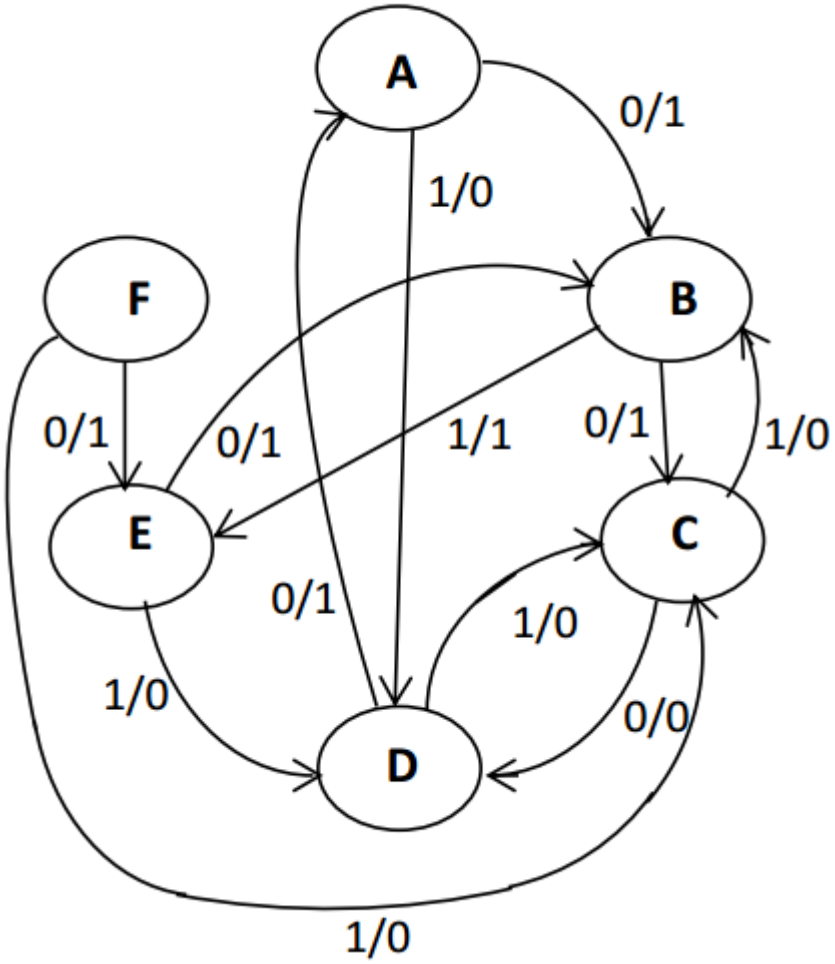
In this case, it must be simplified.

Two states at time (t) are equivalent if :

- they have the same next state (t+1)
- they have the same control variables (input and output)

To simplify the diagram, we represent it in **tabular form**

initial state diagrams



Sequential circuits

representation in tabular form

Example 2 (Diagram with equivalent states)

To simplify the diagram, we represent it in tabular form.

We see that for the same input/output variables, A and E have the same next states, so they are equivalent;

We must then delete one of the 2 states : In this example we delete E and replace all the E in the table with A

	X=0		X=1	
	Y	State (t+1)	Y	State(t+1)
A	1	B	0	D
B	1	C	1	~~E~~ A
C	0	D	0	B
D	1	A	0	C
~~E~~	~~1~~	~~B~~	~~0~~	~~D~~
F	1	~~E~~ A	0	C

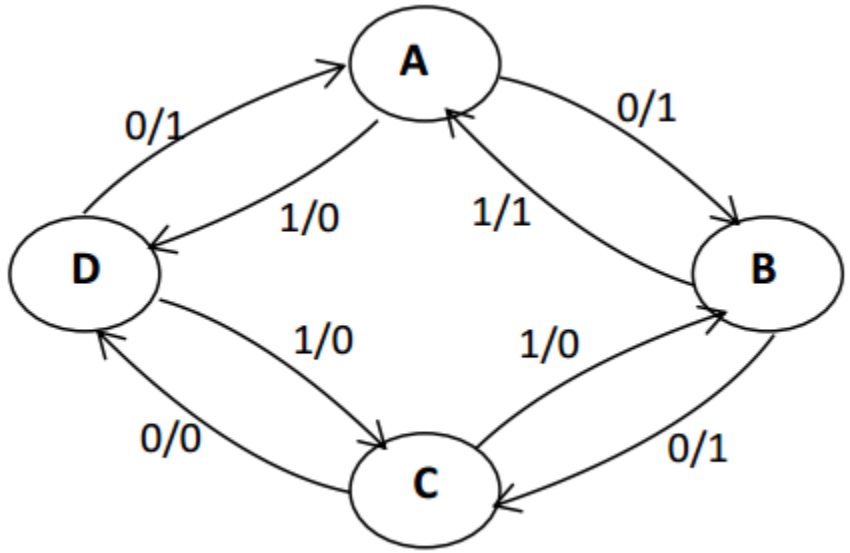
Sequential circuits

Example 2 (Diagram with equivalent states)

We also have the states D and F which are equivalent, we therefore delete F

and we obtain a new table and a new diagram

	X=0		X=1	
	Y	State (t+1)	Y	State(t+1)
A	1	B	0	D
B	1	C	1	A
C	0	D	0	B
D	1	A	0	C
~~F~~	~~1~~	~~A~~	~~0~~	~~C~~



Simplified state diagram

Sequential circuits

Example 2 (Diagram with equivalent states)

To create the circuit it is necessary to code the states.

Generally, we use binary codes increasing in alphabetical order.

A = 00 B = 01 C = 10 D = 11

Max (A,B,C,D) = D = 11

The greatest value is written on 2 bits so you need 2 flip-flops to create this circuit.

Sequential circuits

Example 2 (Diagram with equivalent states)

Transition table

X	Q_1	Q_0	Q_1^+	Q_0^+	T_1	T_0	Y
0	0	0	0	1	0	1	1
0	0	1	1	0	1	1	1
0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0
1	1	1	1	0	0	1	0

Q_1Q_0 / X	0 0	0 1	1 1	1 0
0	0	1	1	0
1	1	0	0	1

$$T_1 = X \oplus Q_0$$

$$T_0 = 1$$

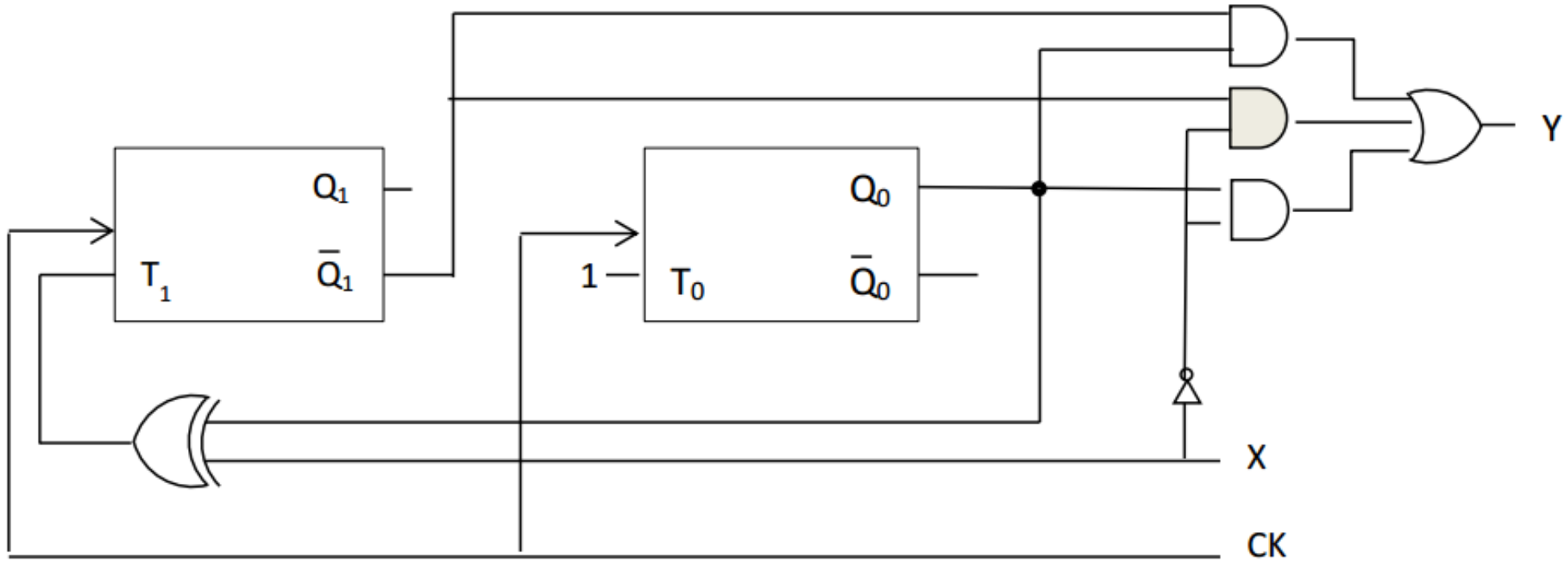
Q_1Q_0 / X	0 0	0 1	1 1	1 0
0	1	1	1	0
1	0	1	0	0

$$Y = \bar{X}\,Q_1 + X\,Q_0 + Q_1\,Q_0$$

Sequential circuits

Example 2 (Diagram with equivalent states)

Corresponding circuit



Timing Diagram (chronogram)

Timing Diagram Timing Diagram is a graphical representation of the evolution of the state of a circuit over time. It uses a clock of period T. There are two types of sequential circuits, synchronous circuits and asynchronous circuits:

- **In a synchronous circuit**, all flip-flops are connected to the same clock.
- **In an asynchronous circuit**, the flip-flops do not have the same clock

- Generally we find the sequence of a circuit using a truth table
- but we can also find this directly using a **Timing Diagram (chronogram).**

Timing Diagram

To produce the timing diagram of a circuit it is necessary:

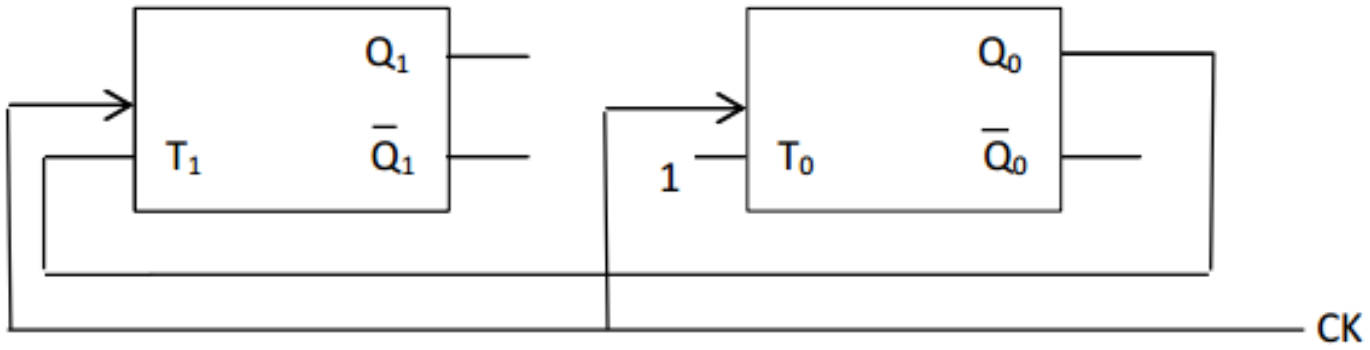
a. Give the initial state of each flip-flop at time (t)

b. Deduce the values of the inputs for each flip-flop at time (t)

c. From of these inputs, find the state of each flip-flop at time (t+1)

d. (t+1) becomes (t) and we start again until we find the initial state

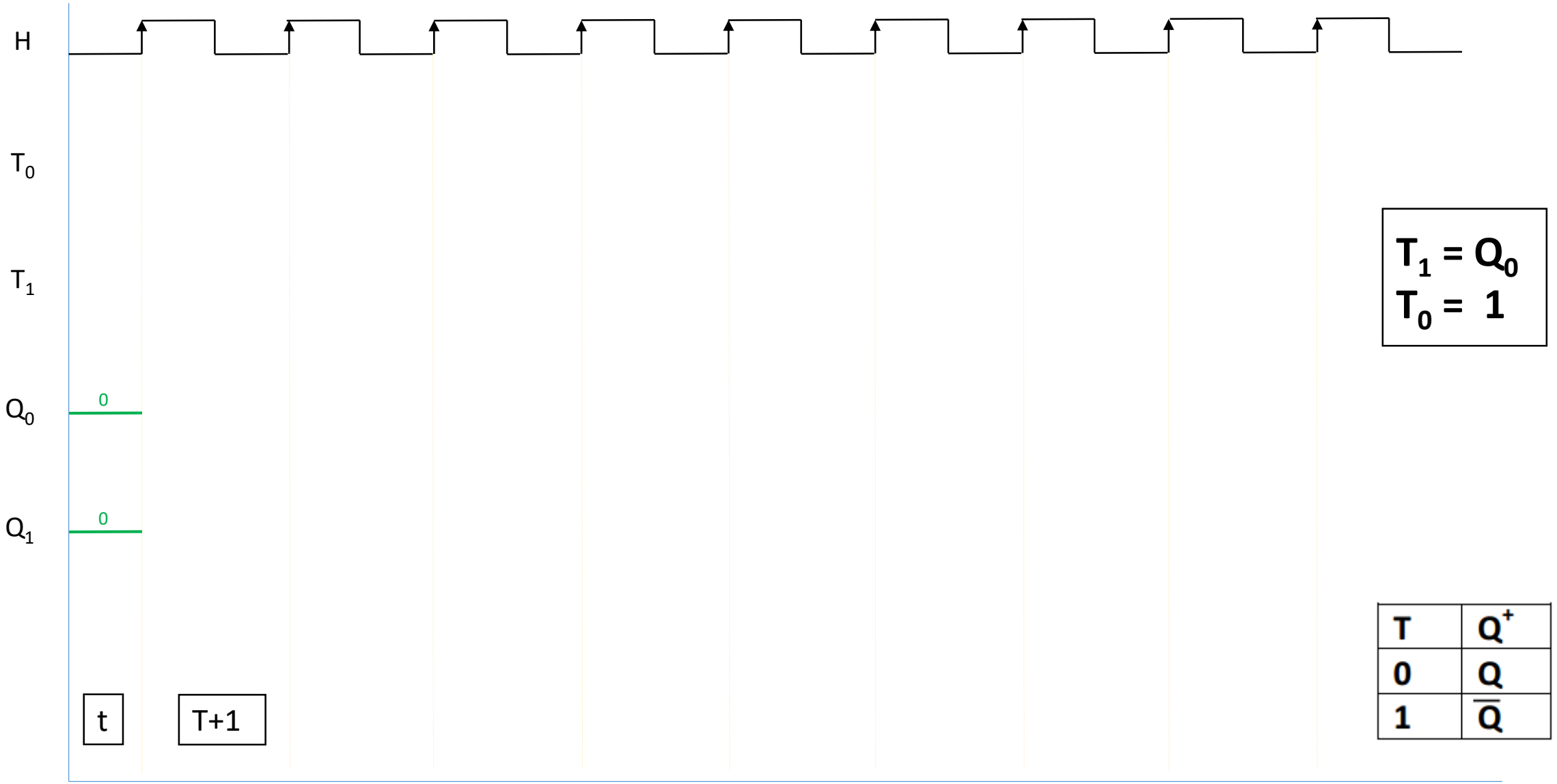
- **Example 1: synchronous circuit**



T0 = 1

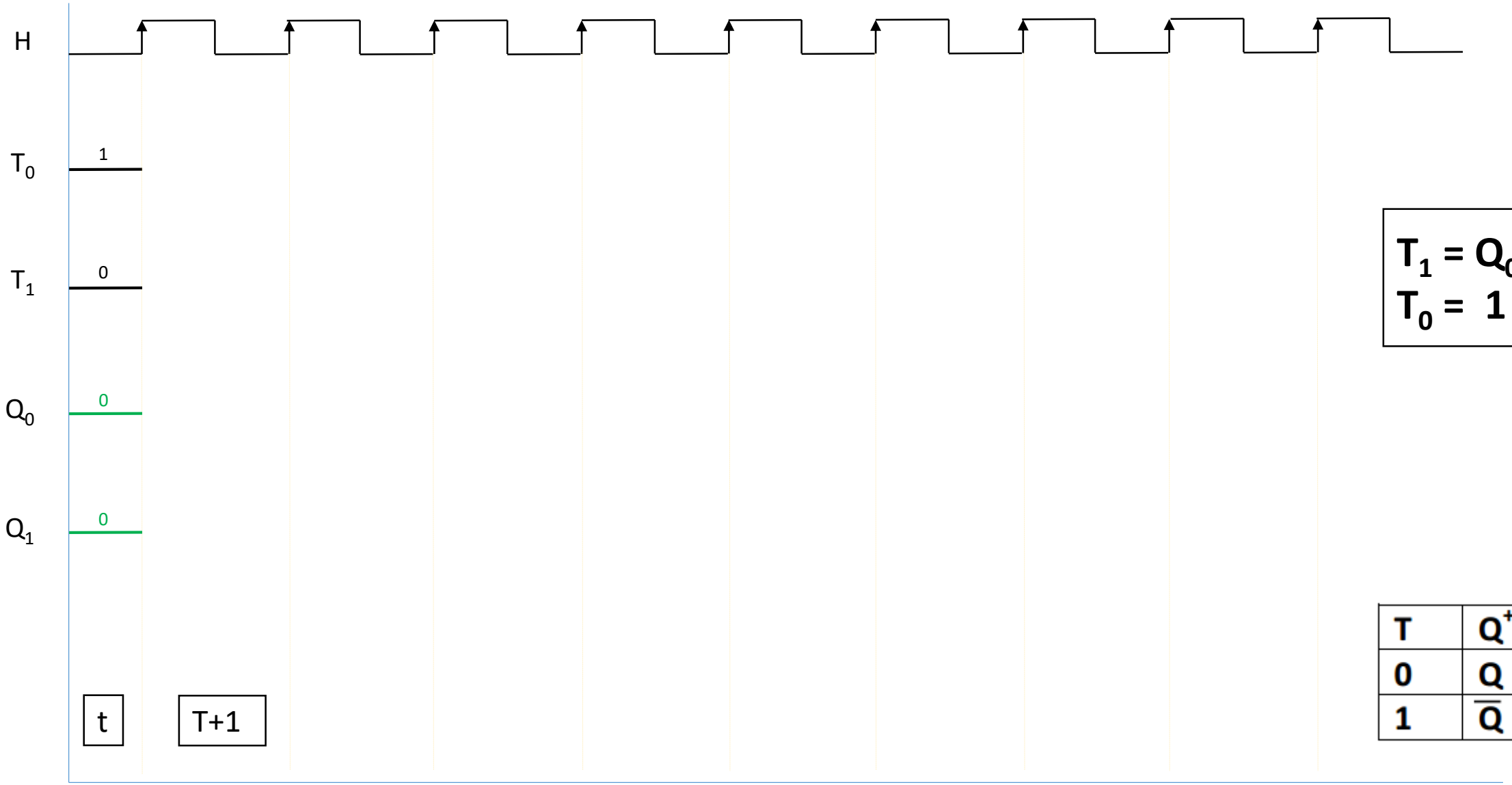
T1 = Q_0

Example 1

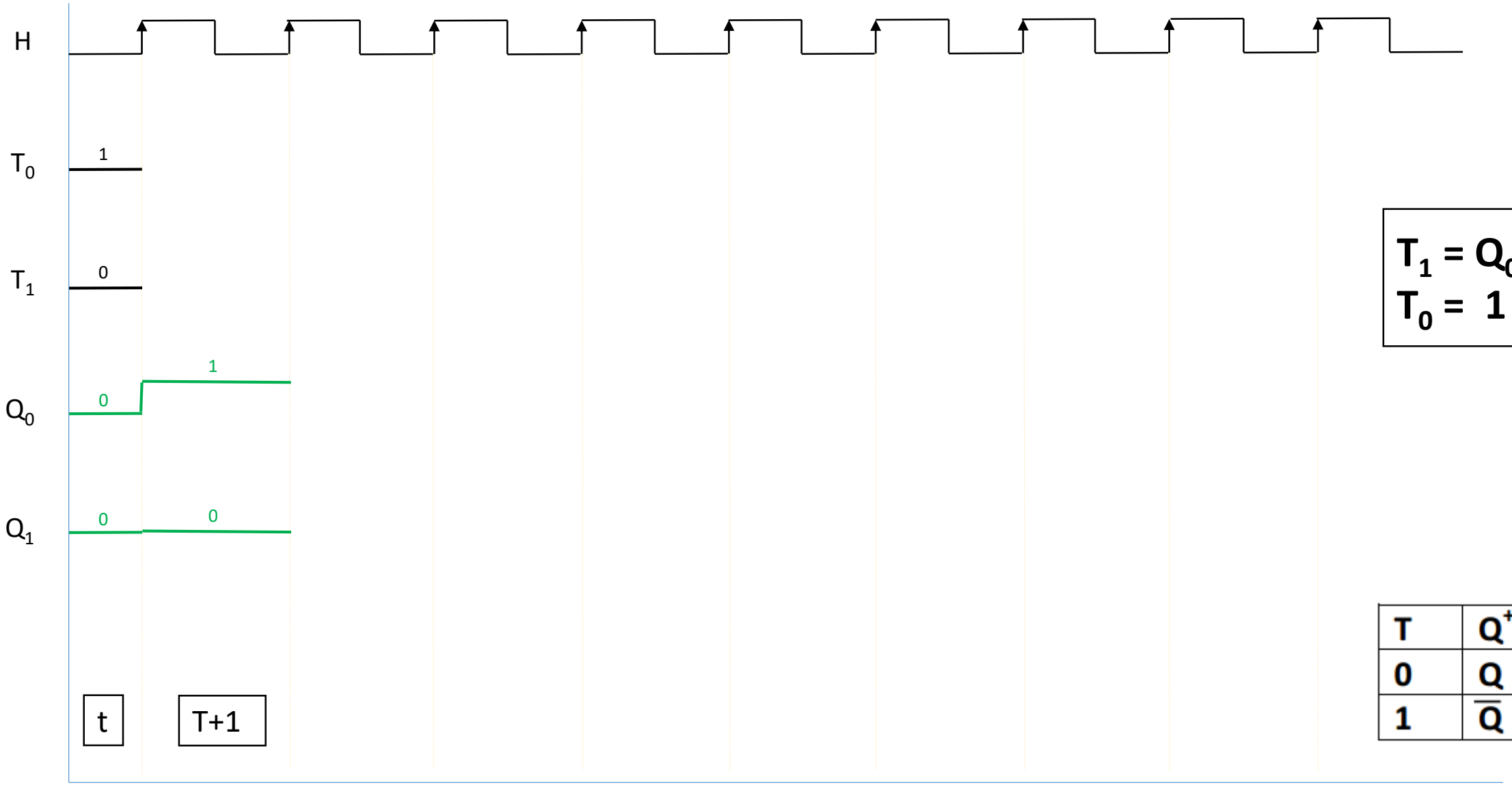


T	Q^+
0	Q
1	$\overline{Q}$

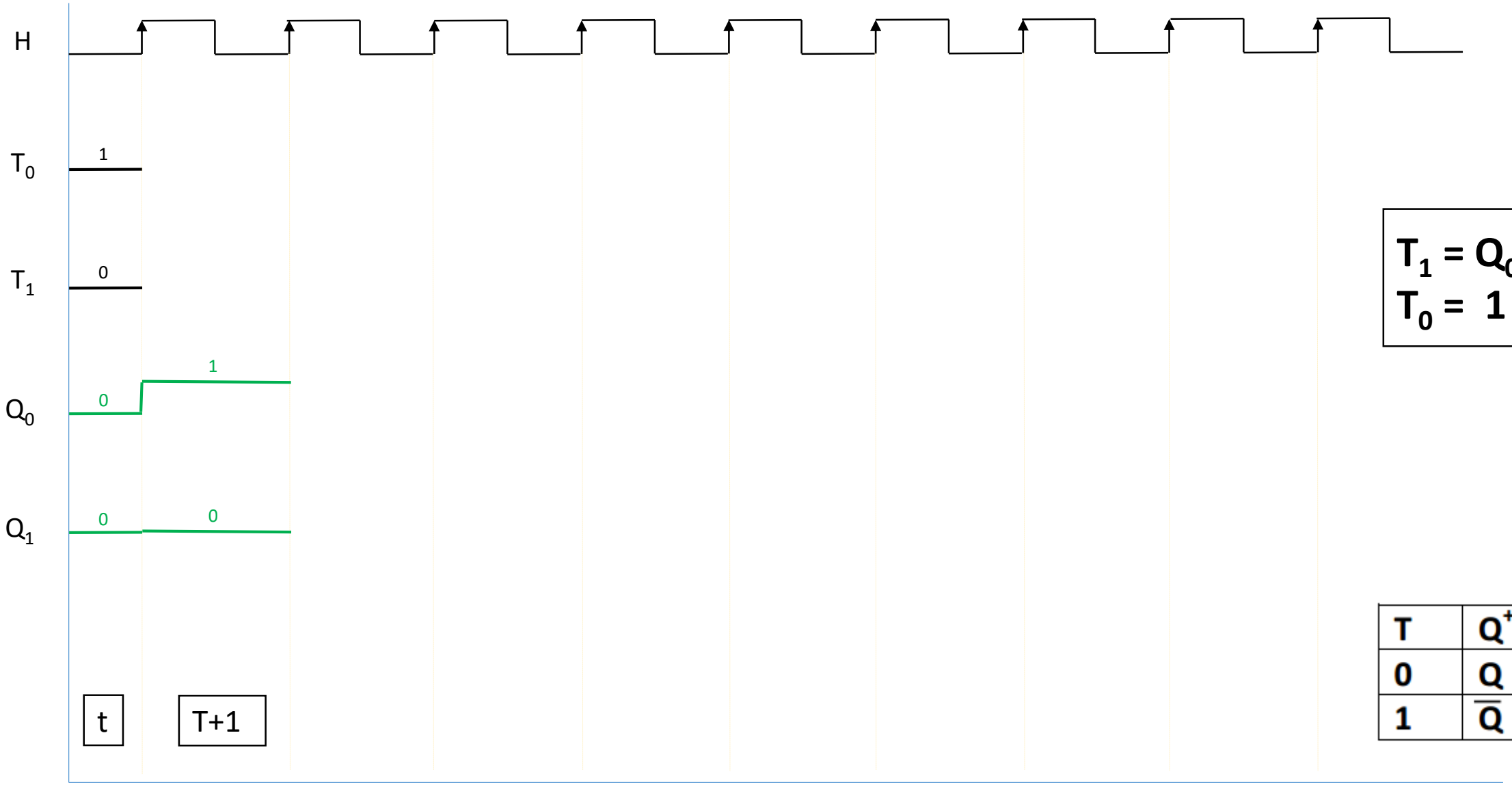
Example 1



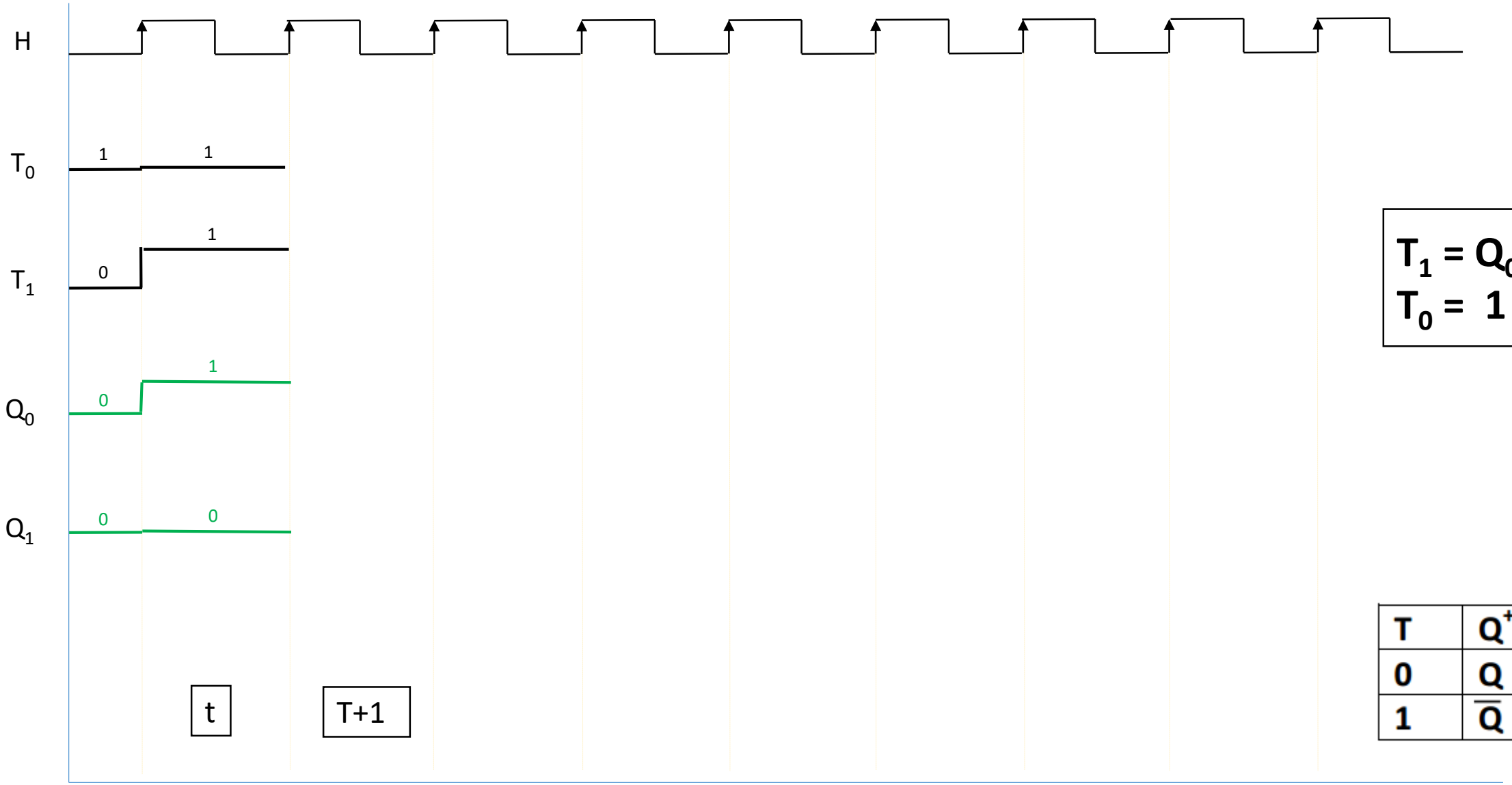
Example 1



Example 1

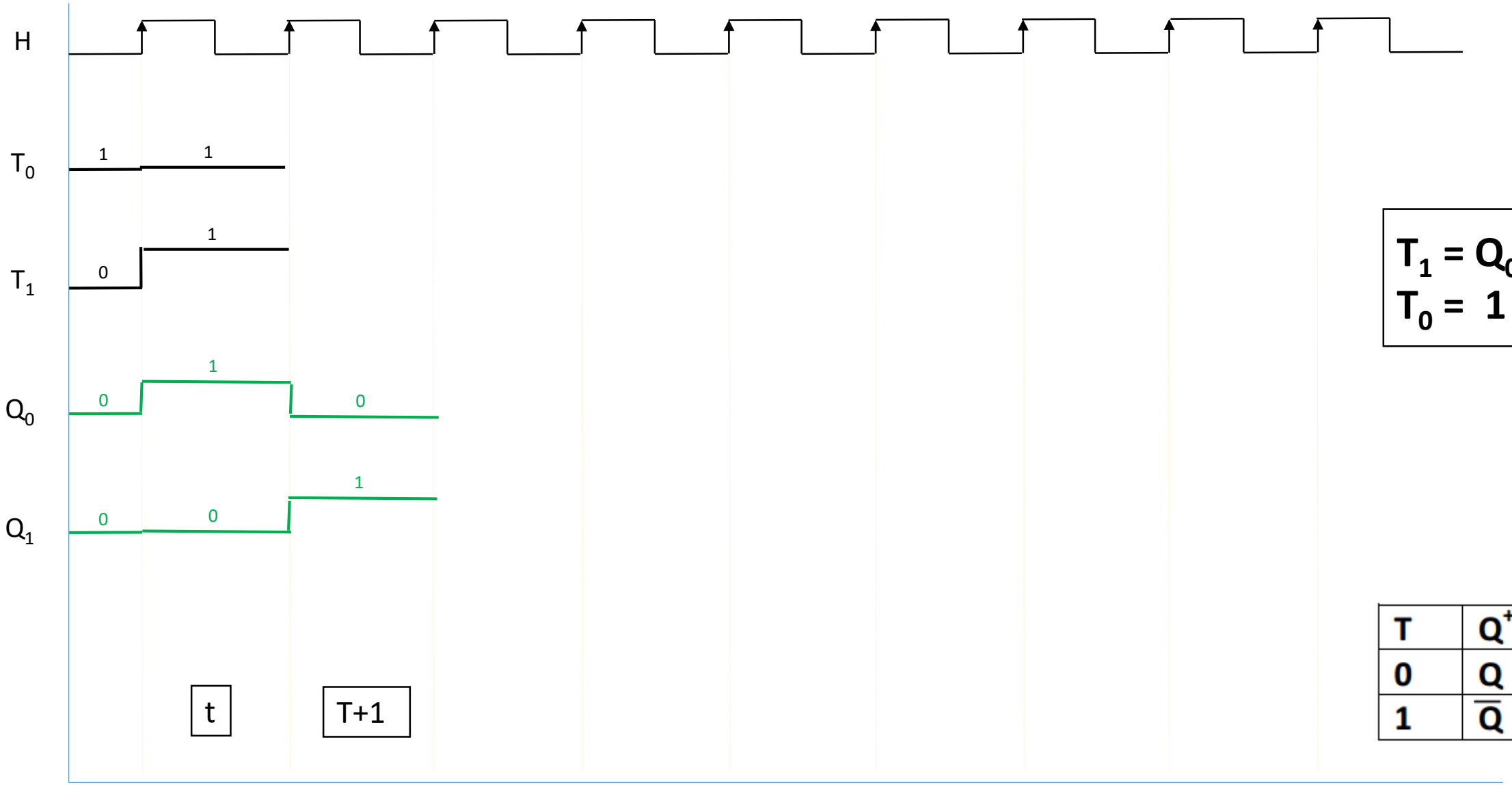


Example 1



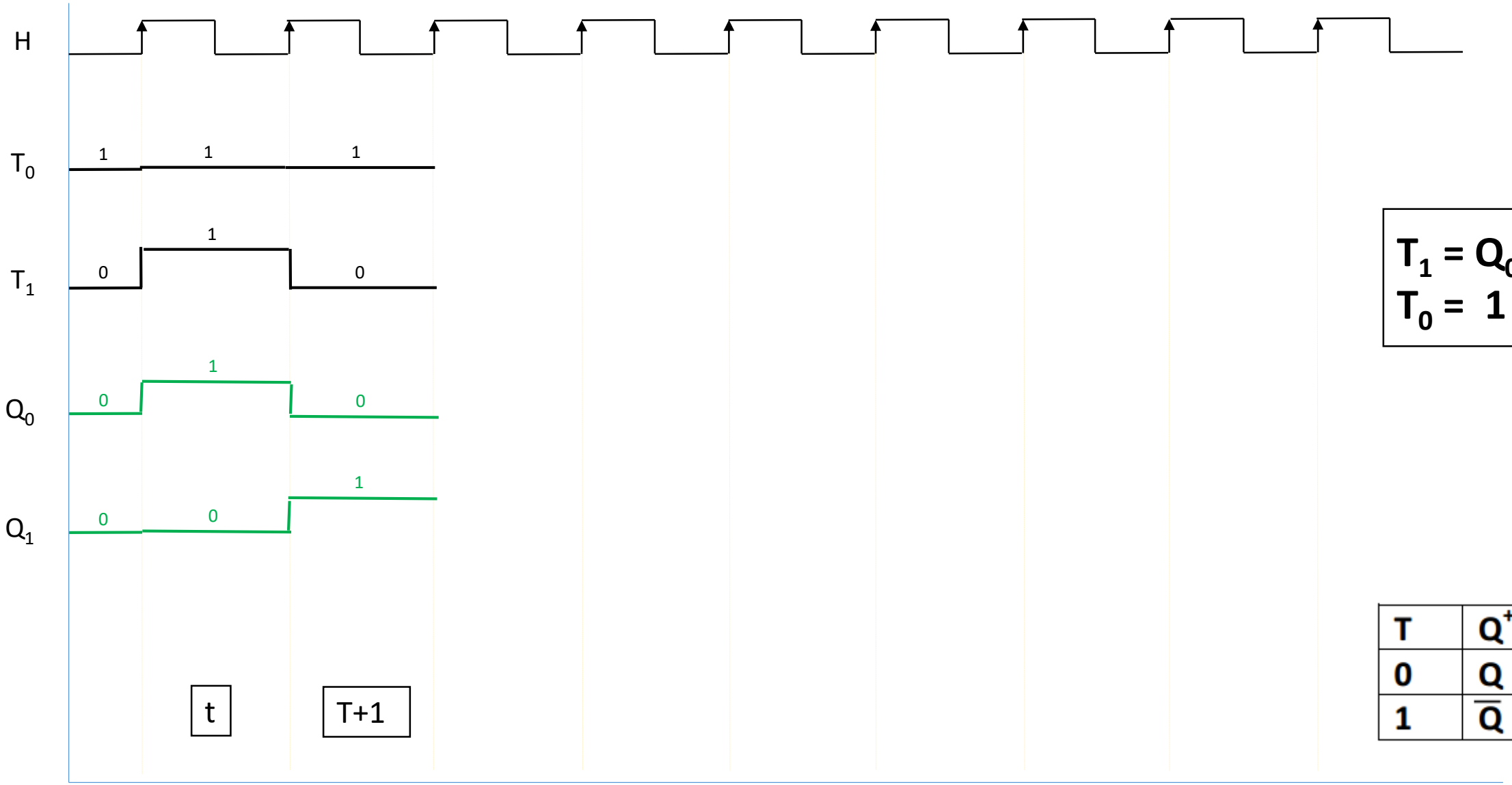
T	Q^+
0	Q
1	$\overline{Q}$

Example 1



T	Q^+
0	Q
1	$\overline{Q}$

Example 1

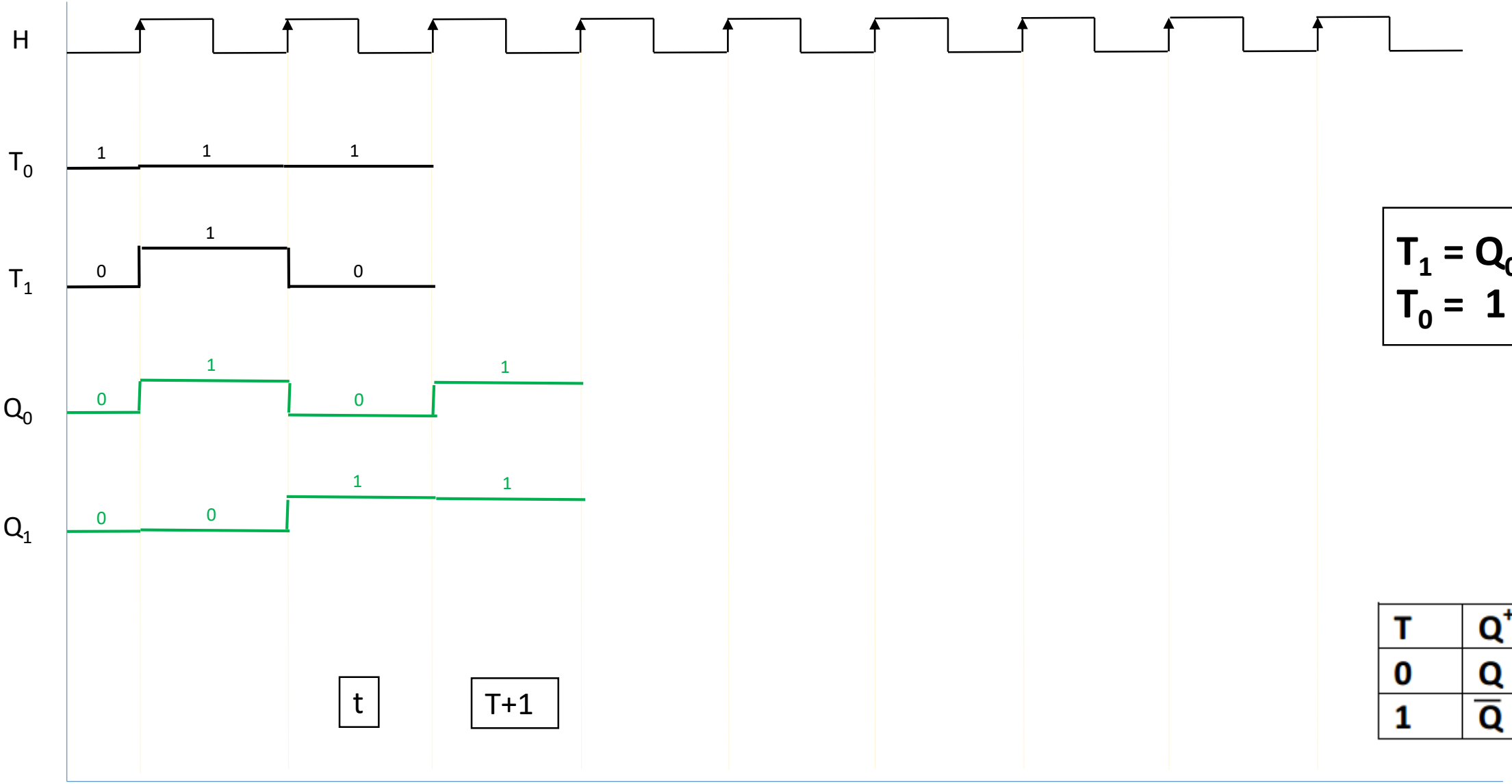


$T_1 = Q_0$

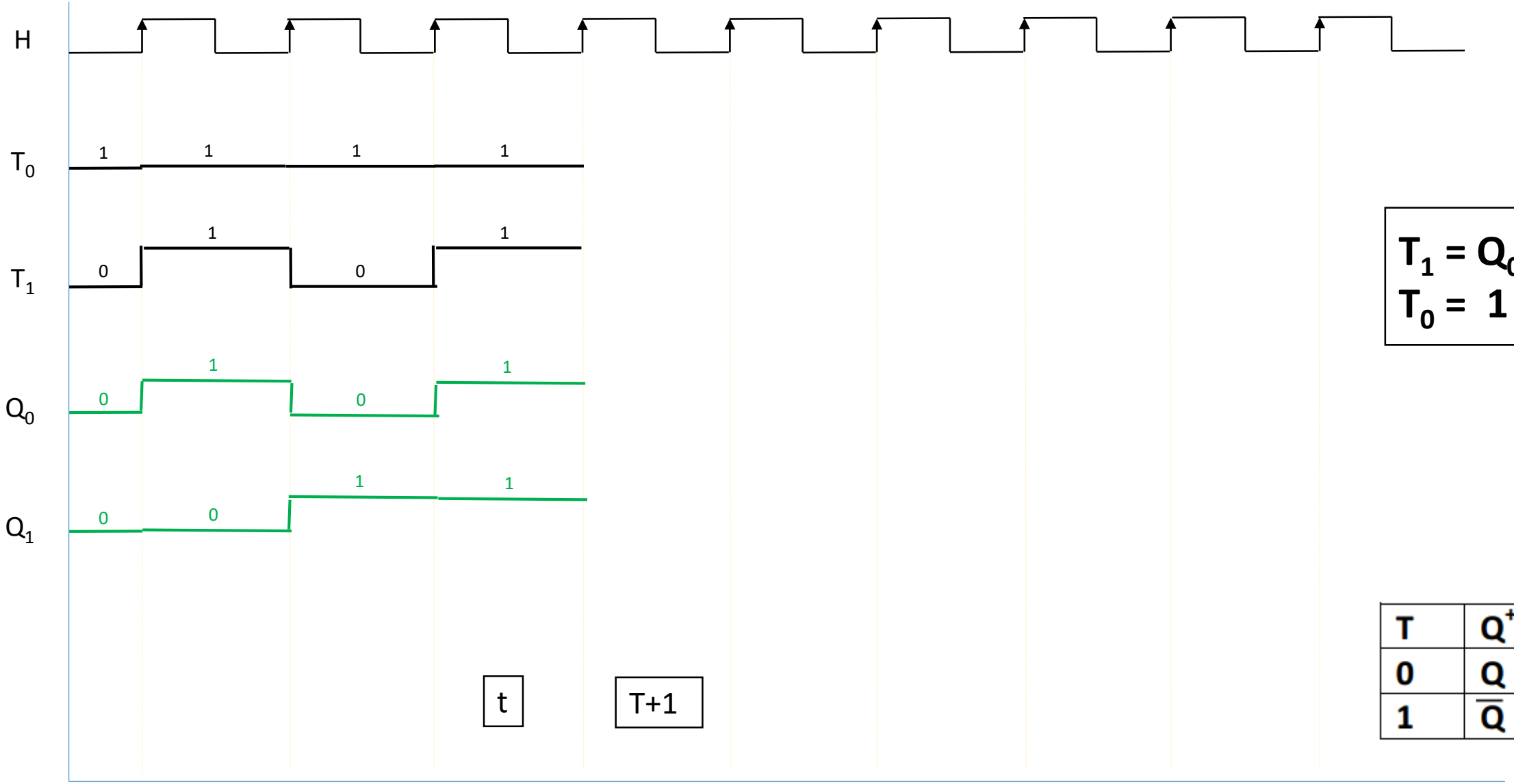
$T_0 = 1$

T	Q^+
0	Q
1	$\overline{Q}$

Example 1



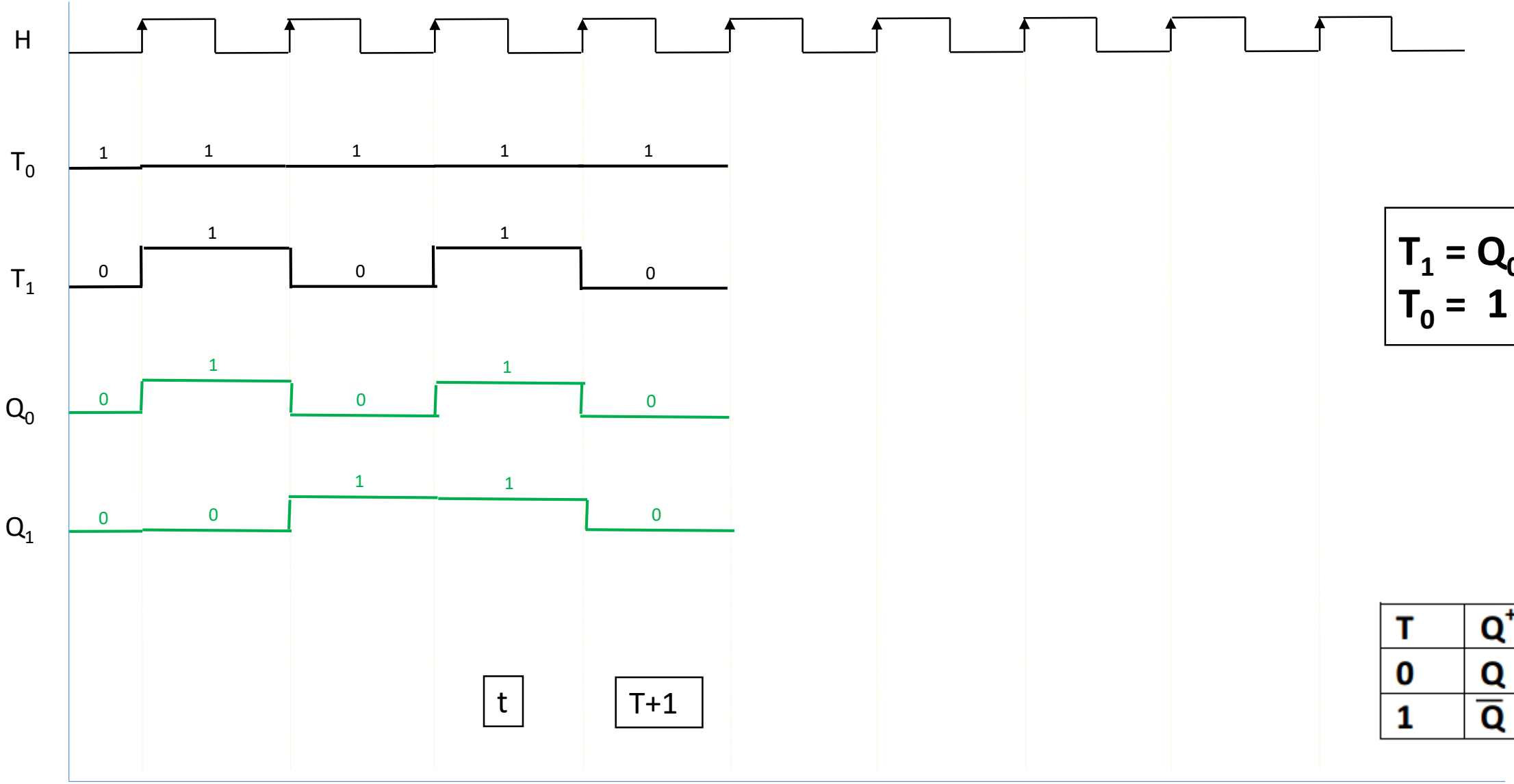
Example 1



$T_1 = Q_0$
$T_0 = 1$

T	Q^+
0	Q
1	$\overline{Q}$

Example 1

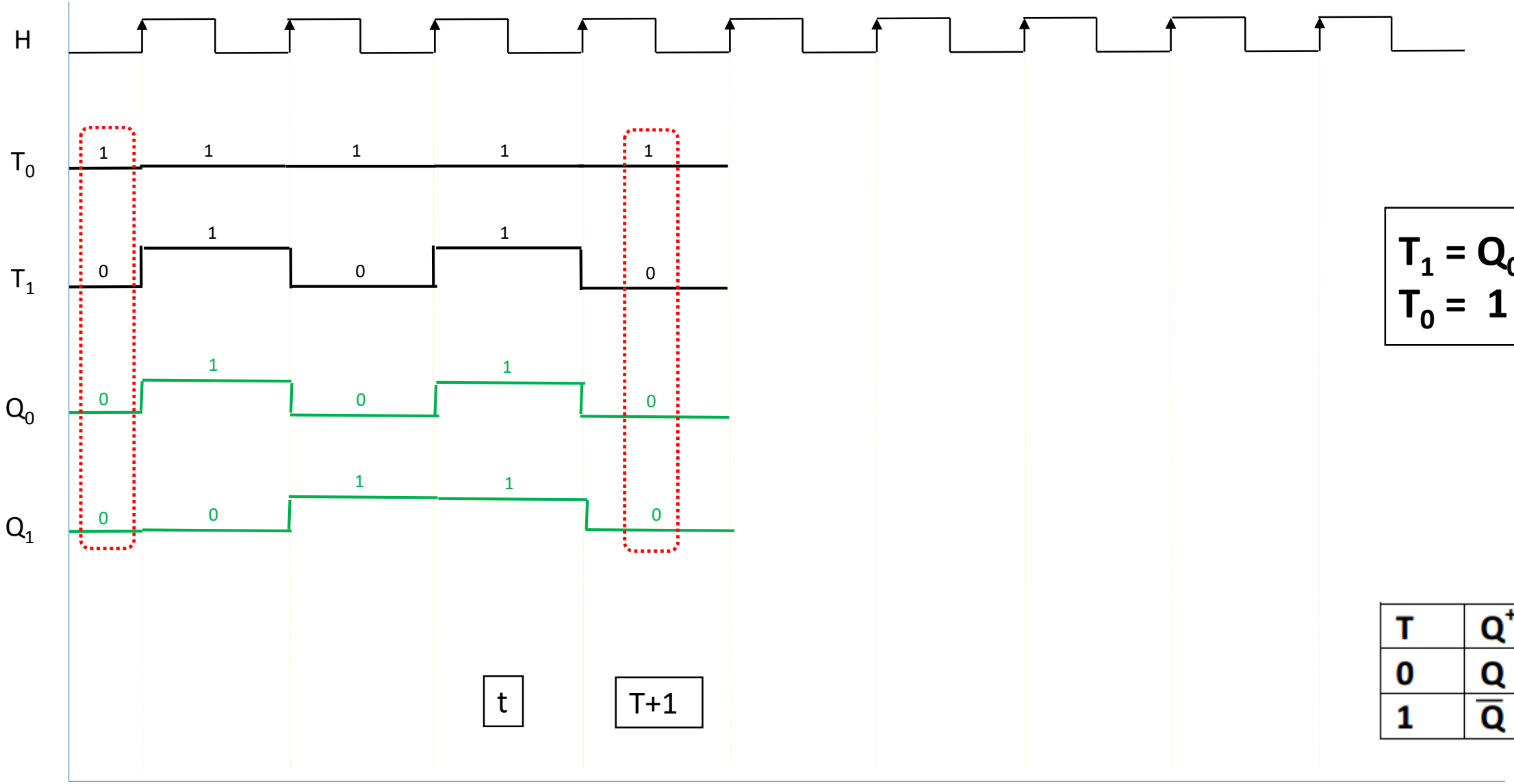


$T_1 = Q_0$

$T_0 = 1$

T	Q^+
0	Q
1	$\overline{Q}$

Example 1

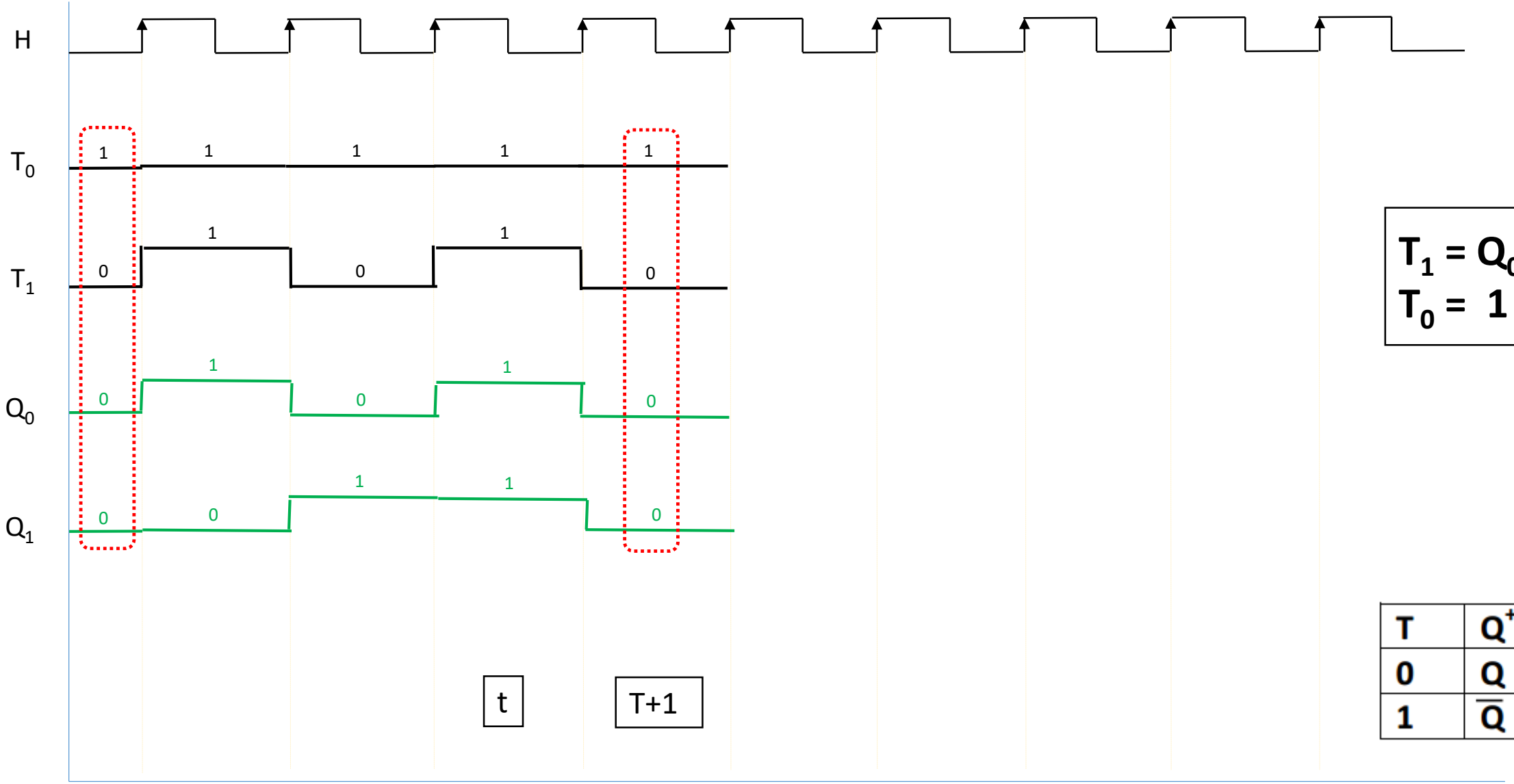


$T_1 = Q_0$

$T_0 = 1$

T	Q^+
0	Q
1	$\overline{Q}$

Example 1

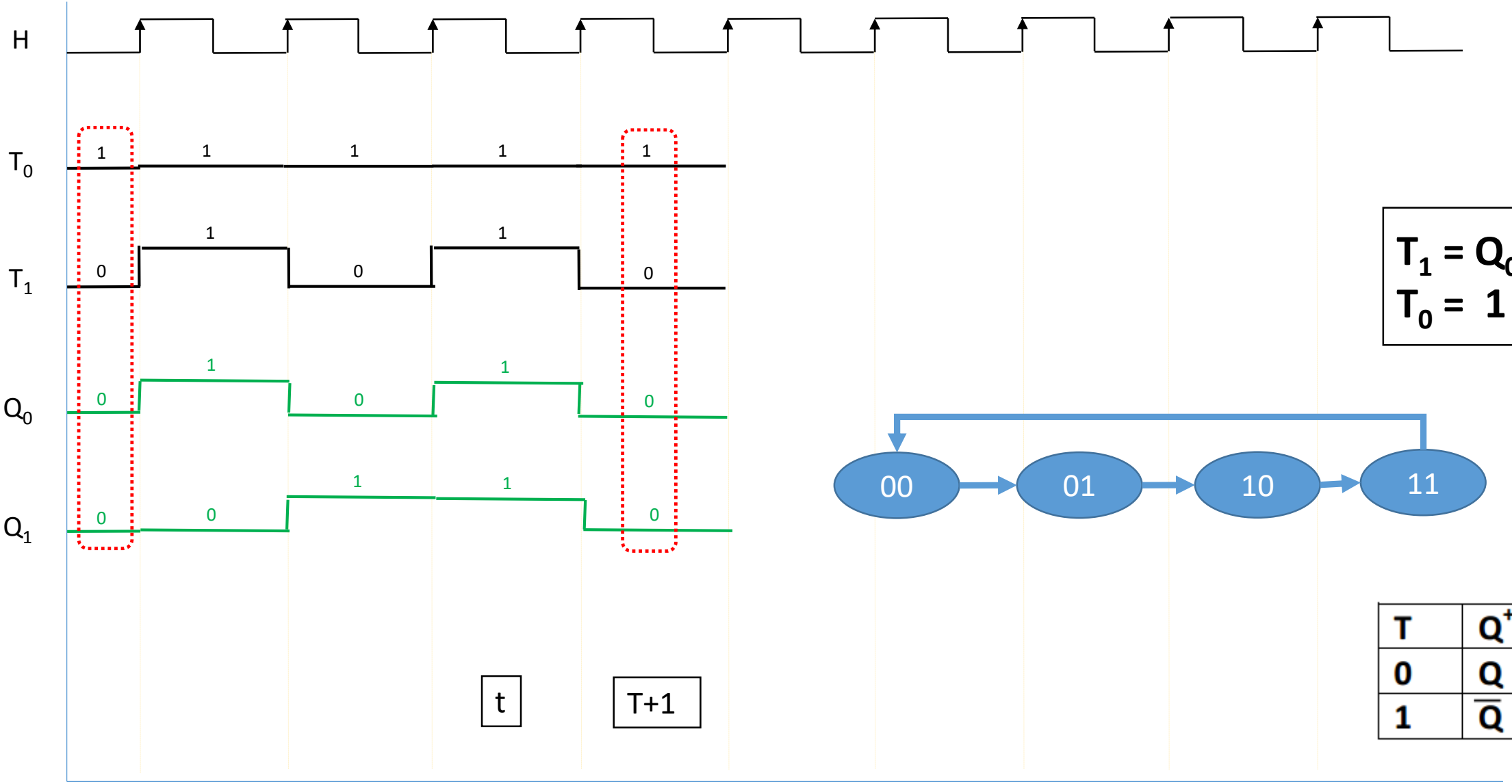


$T_1 = Q_0$

$T_0 = 1$

T	Q^+
0	Q
1	$\overline{Q}$

Example 1



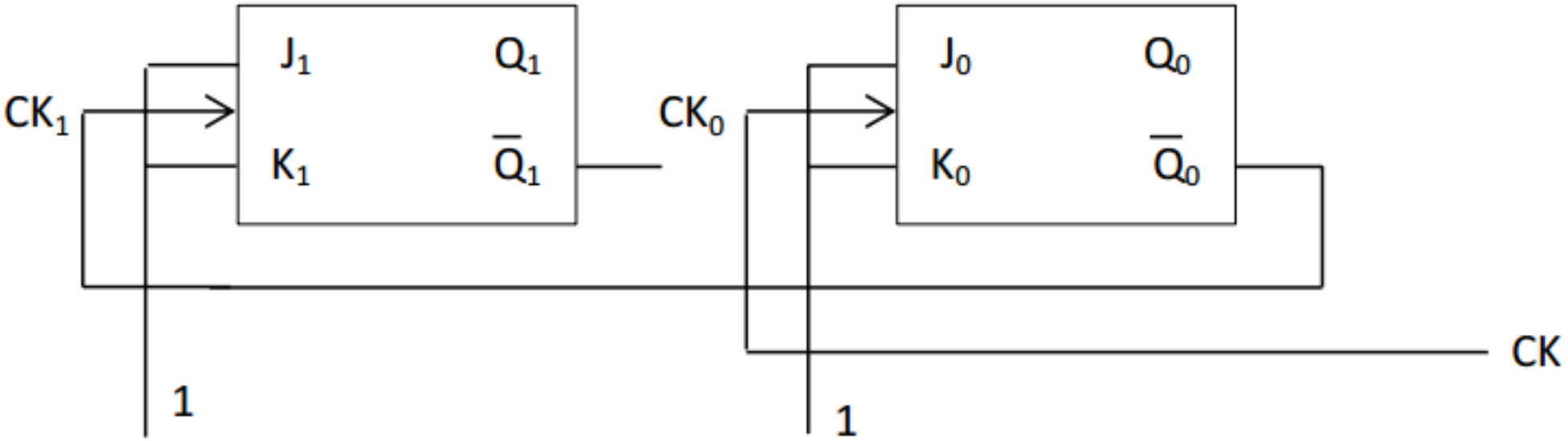
$T_1 = Q_0$

$T_0 = 1$

T	Q^+
0	Q
1	$\overline{Q}$

Timing Diagram (chronogram)

- **Example 2 :** **Asynchronous circuit**



J0 = 1

K0 = 1

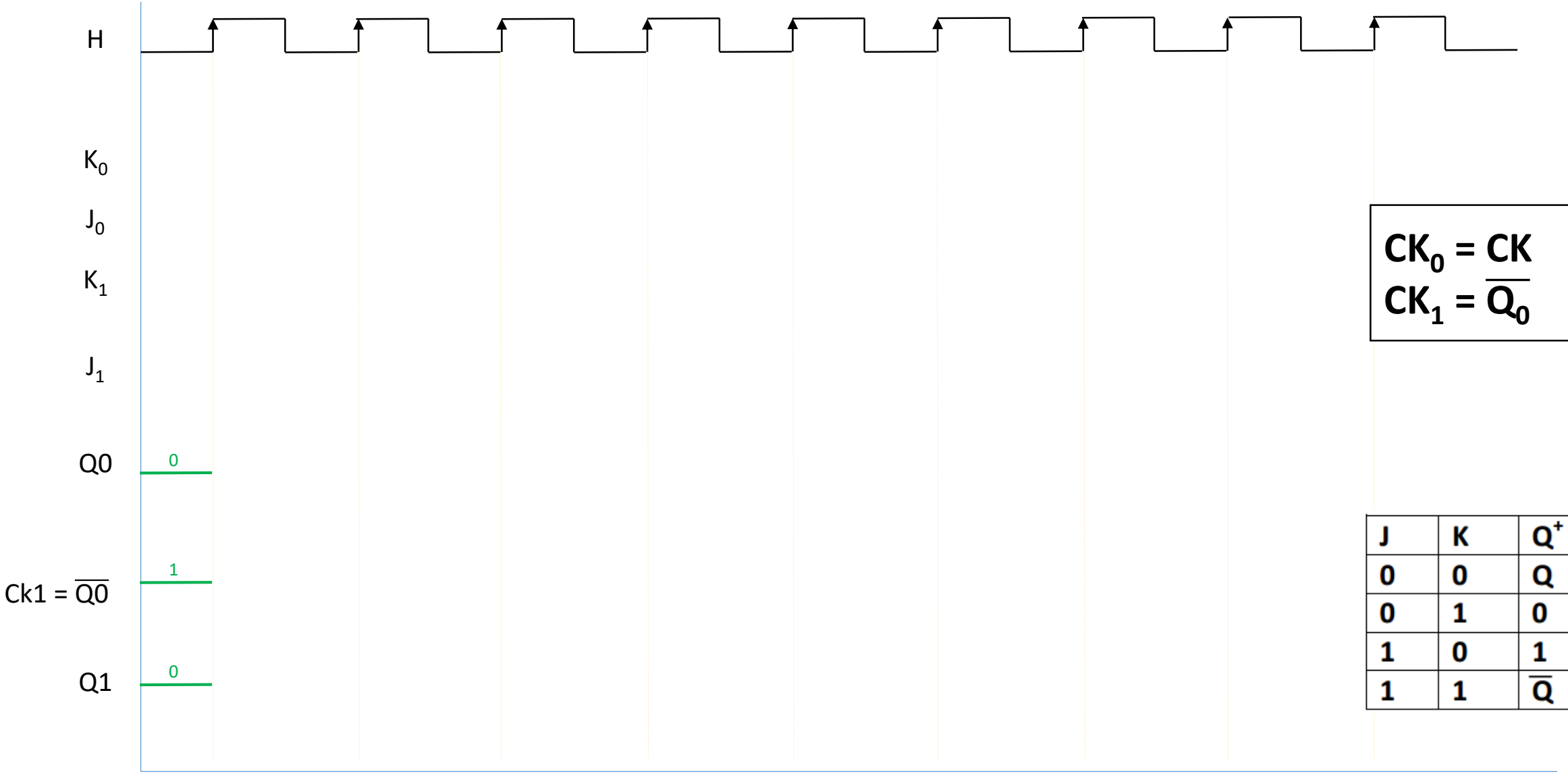
J1 = 1

J1 = 1

CK0 = CK

CK1 = $\overline{Q_0}$

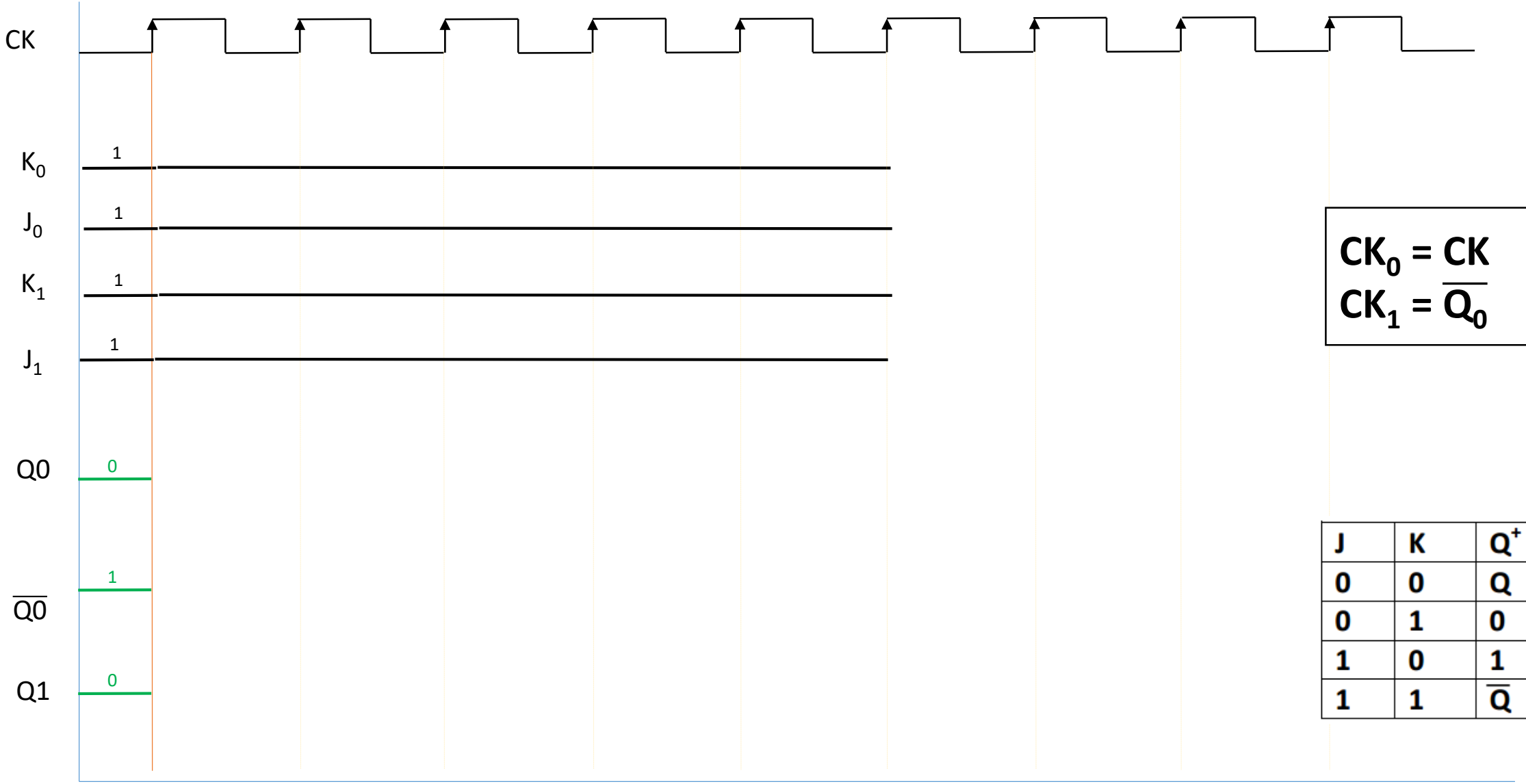
Example 2



$$CK_0 = CK$$
$$CK_1 = \overline{Q_0}$$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

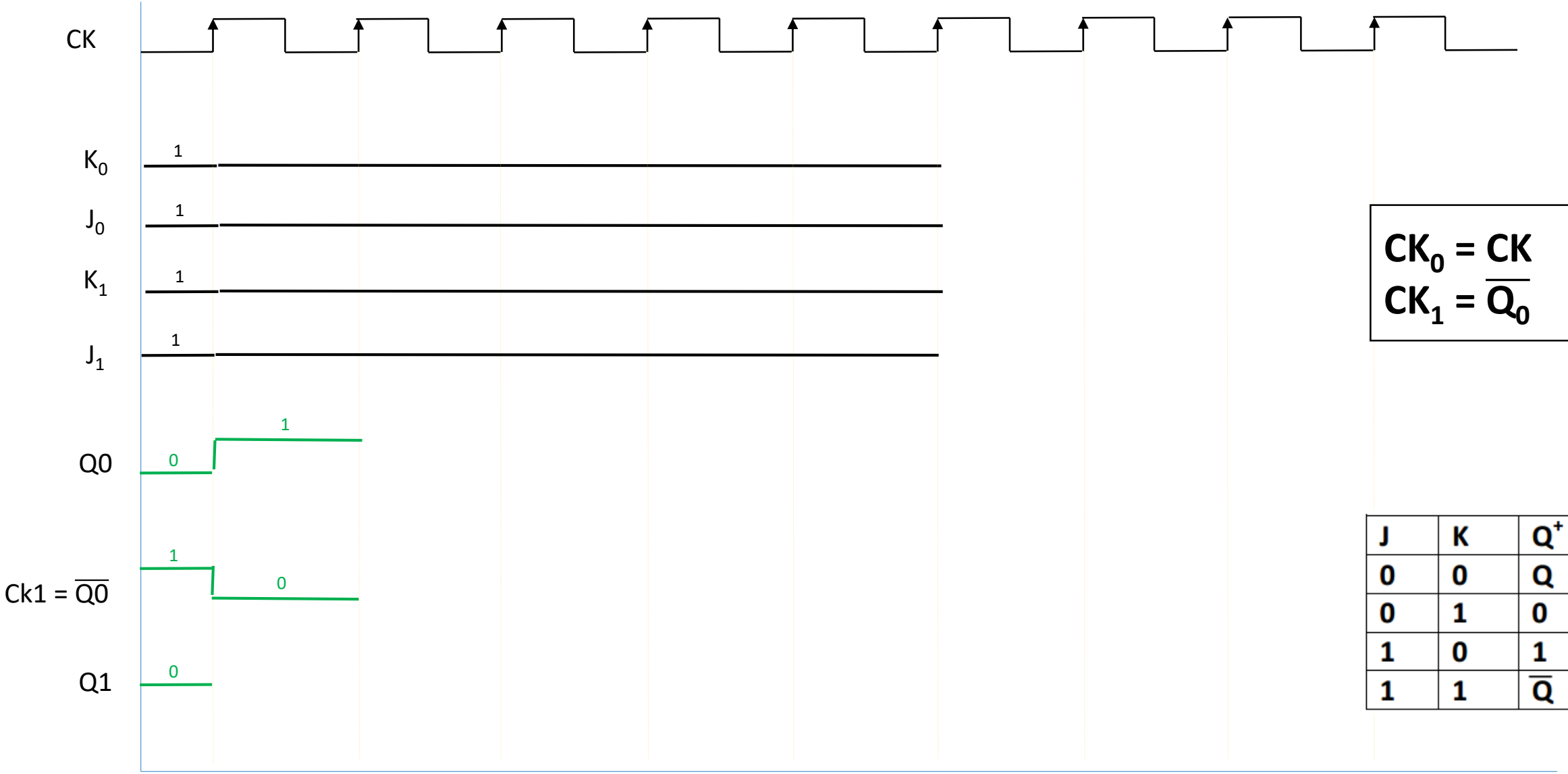
Example 2



$$CK_0 = CK$$
$$CK_1 = \overline{Q_0}$$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Example 2

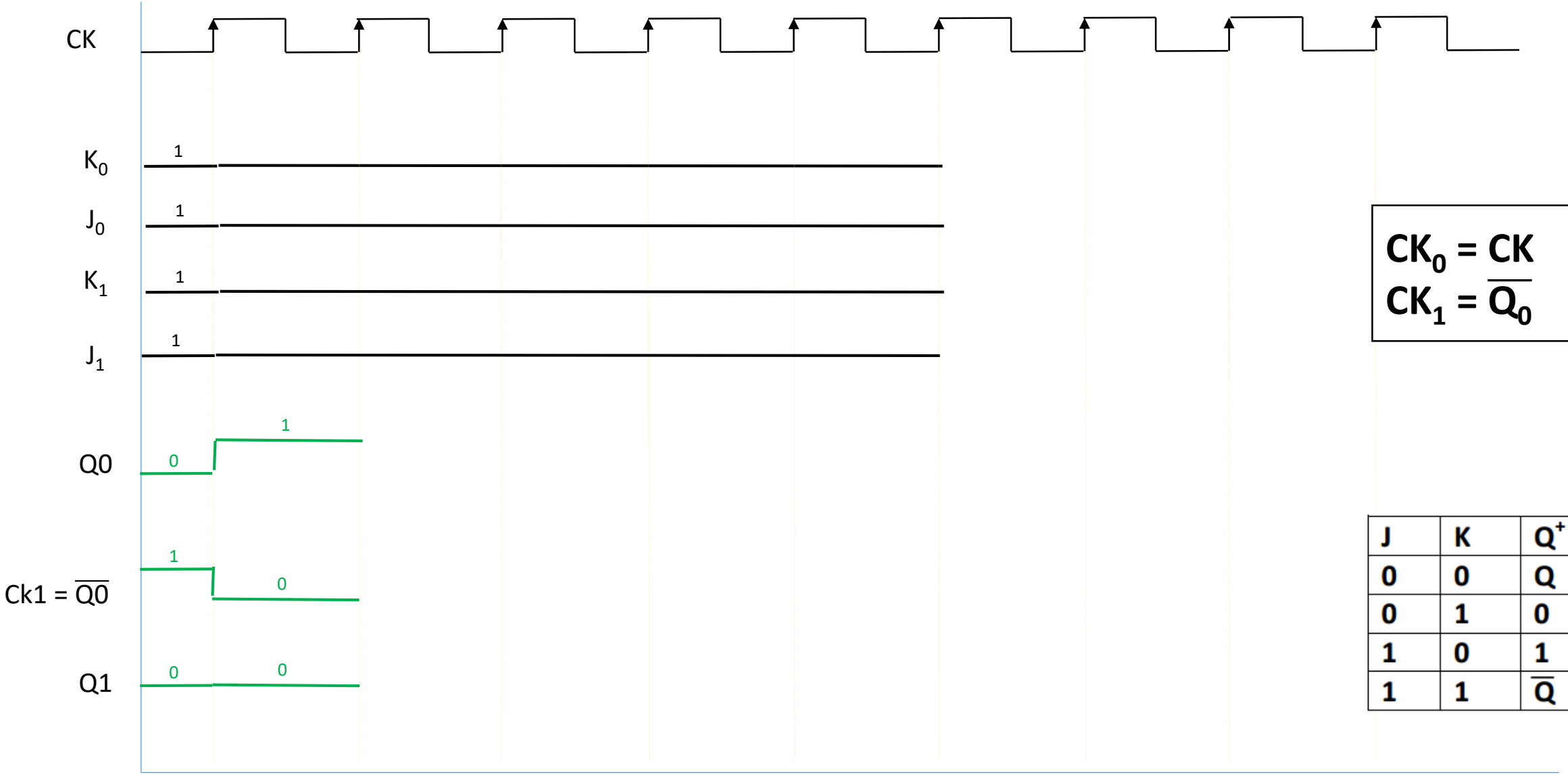


$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Example 2

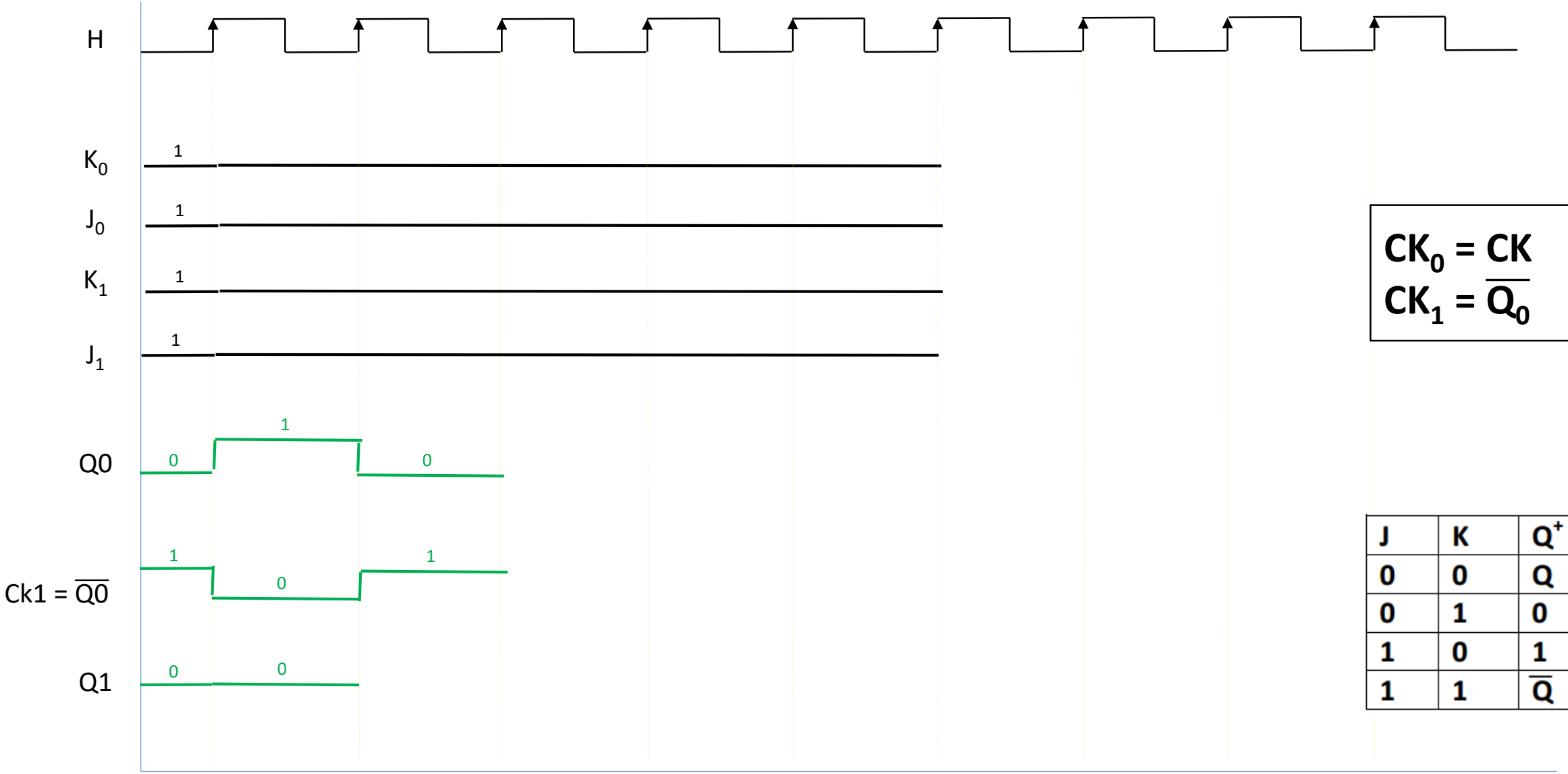


$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Example 2

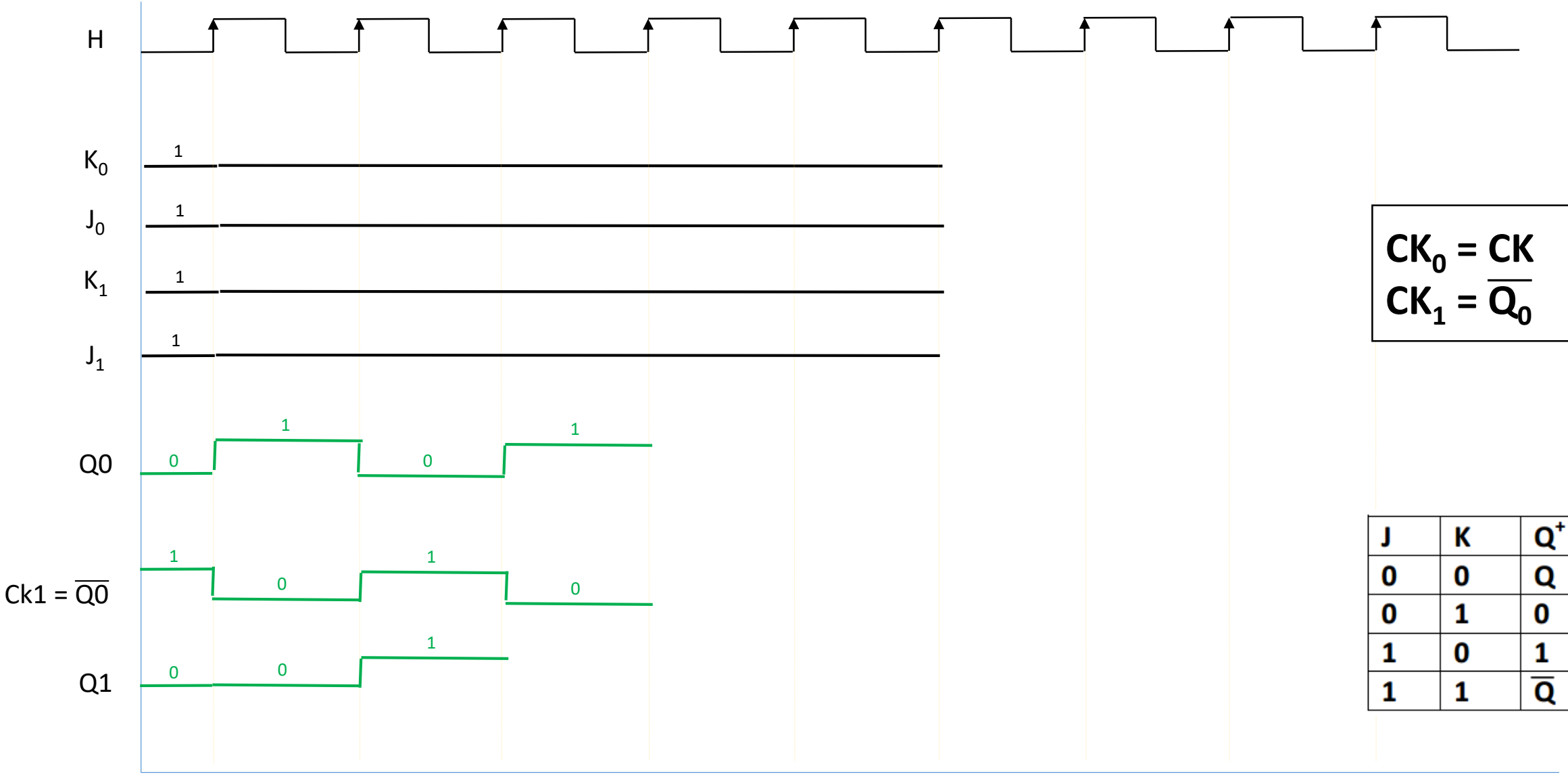


$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Example 2

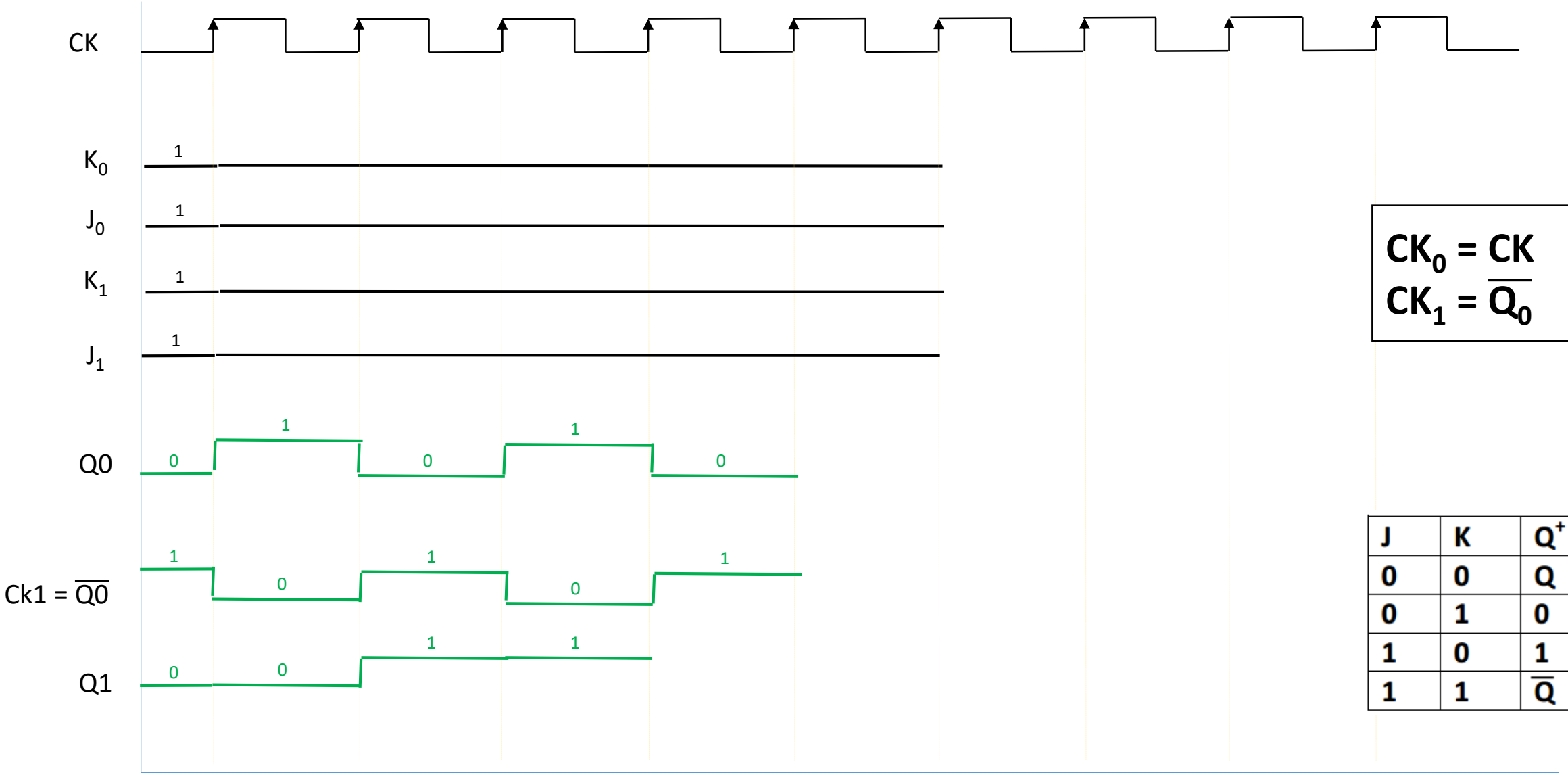


$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Example 2

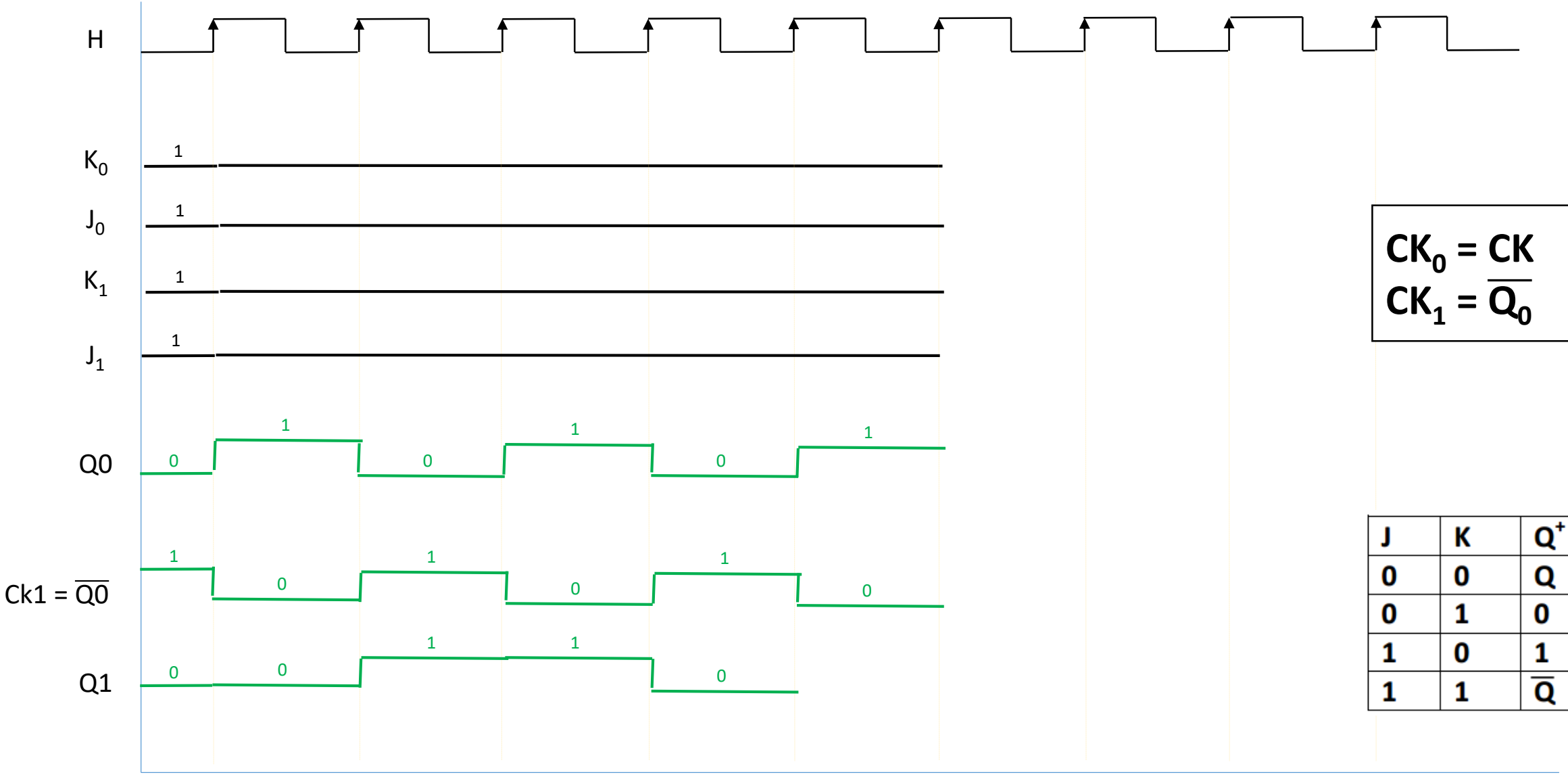


$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Example 2

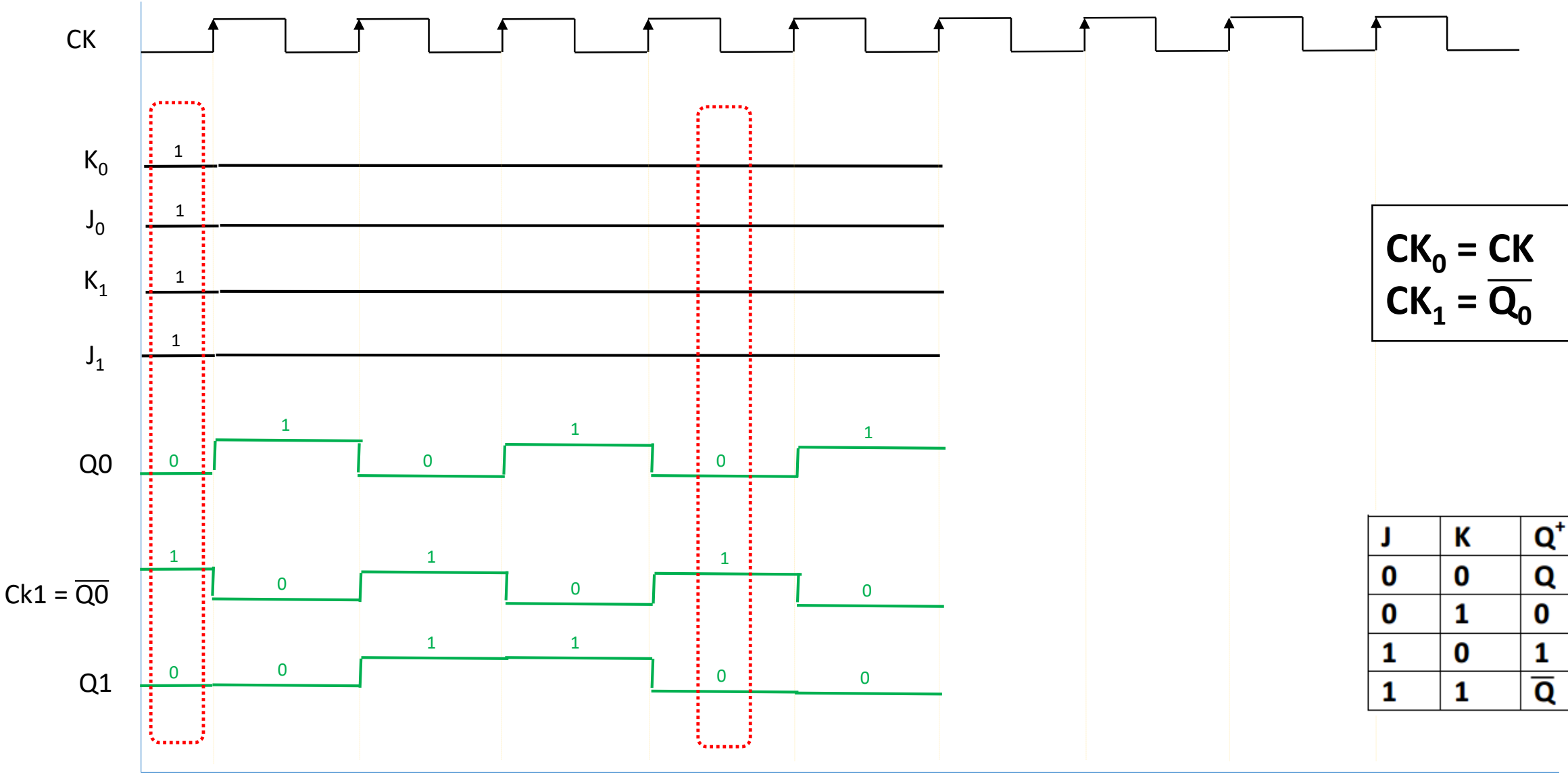


$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Exemple 2



$CK_0 = CK$

$CK_1 = \overline{Q_0}$

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Force functions : Clear et Preset

- **Clear function:**

- Clear is a logic function that allows to reset a flip-flop at any time regardless of its state, its inputs or its clock.

- **In a sequential circuit** when Clear is enabled, all flip-flops instantly display zero regardless of other inputs

- It's the circuit reset function.

Force functions : Clear et Preset

- **Preset function**

- Preset is a logic function that allows to set a flip-flop to 1 at any time regardless of its state, its inputs or its clock.

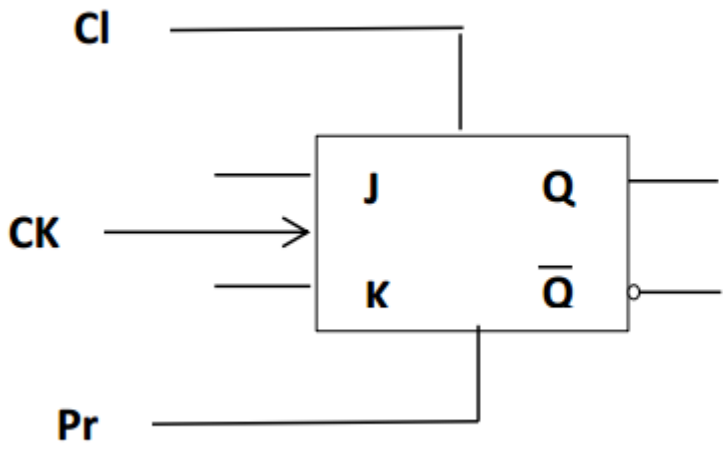
- **In a sequential circuit,** when the Preset function is activated, all flip-flops instantly display 1 regardless of other inputs.

This is the function that set the circuit to 1.

Force functions : Clear et Preset

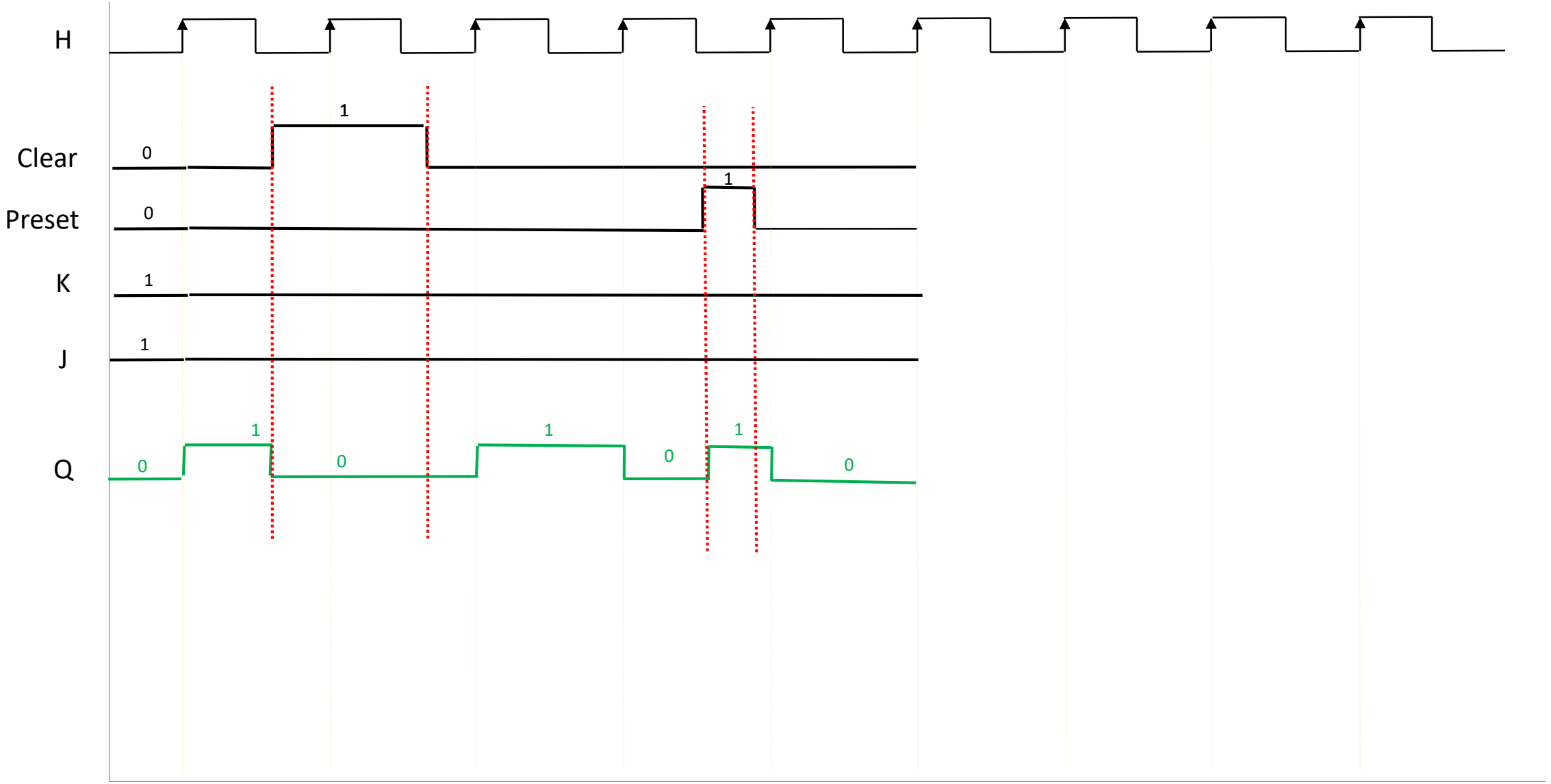
- **Example for a JK flip-flop**

- The Clear and Preset inputs force the flip-flop to go to 0 or 1 without taking into account the values of the J and K inputs or the CK clock.



Clear	Preset	Q_{t+1}
0	0	Dépend de (J K) et CK
0	1	Mise à 1 ∀ (J K) et CK
1	0	Mise à 0 ∀ (J K) et CK
1	1	Interdit

Example 2



Ch1. The first part

Sequential circuits

Analyze

Analysis of a sequential circuit

a) To analyze a sequential circuit, we must :

b) Establish the input equations of each flip-flop

c) Create the Truth Table of the circuit (the principle is to find the Qi+ from the values of the input equations).

d) Deduce the state diagram hence the role of the circuit. The state diagram can consist of one or more sequences

Synchronous circuits

The timing diagram

a) Give the initial state of each flip-flop at time (t)

b) Deduce the values of the inputs for each flip-flop at time (t)

c) From of these inputs, find the state of each flip-flop at time (t+1)

(d) t+1) becomes (t) and we start again until we find the initial state

Synchronous and Asynchronous circuits

Syntheses

Synthesis of a sequential circuit :

a) Establish the state diagram (or sequence) and give the number of flip-flops necessary.

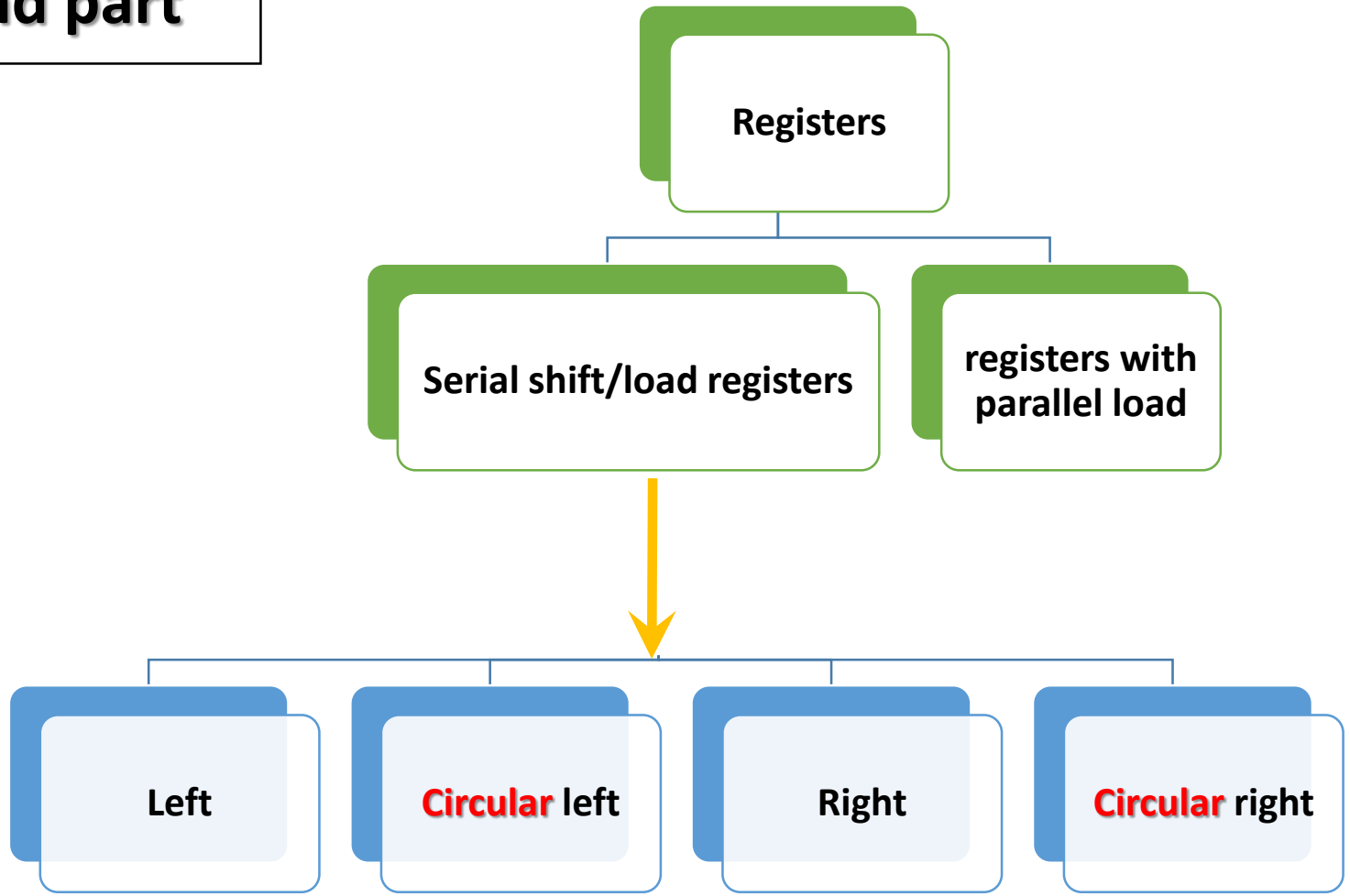
b) Create the transition table

c) Deduce the input equations to the flip-flops

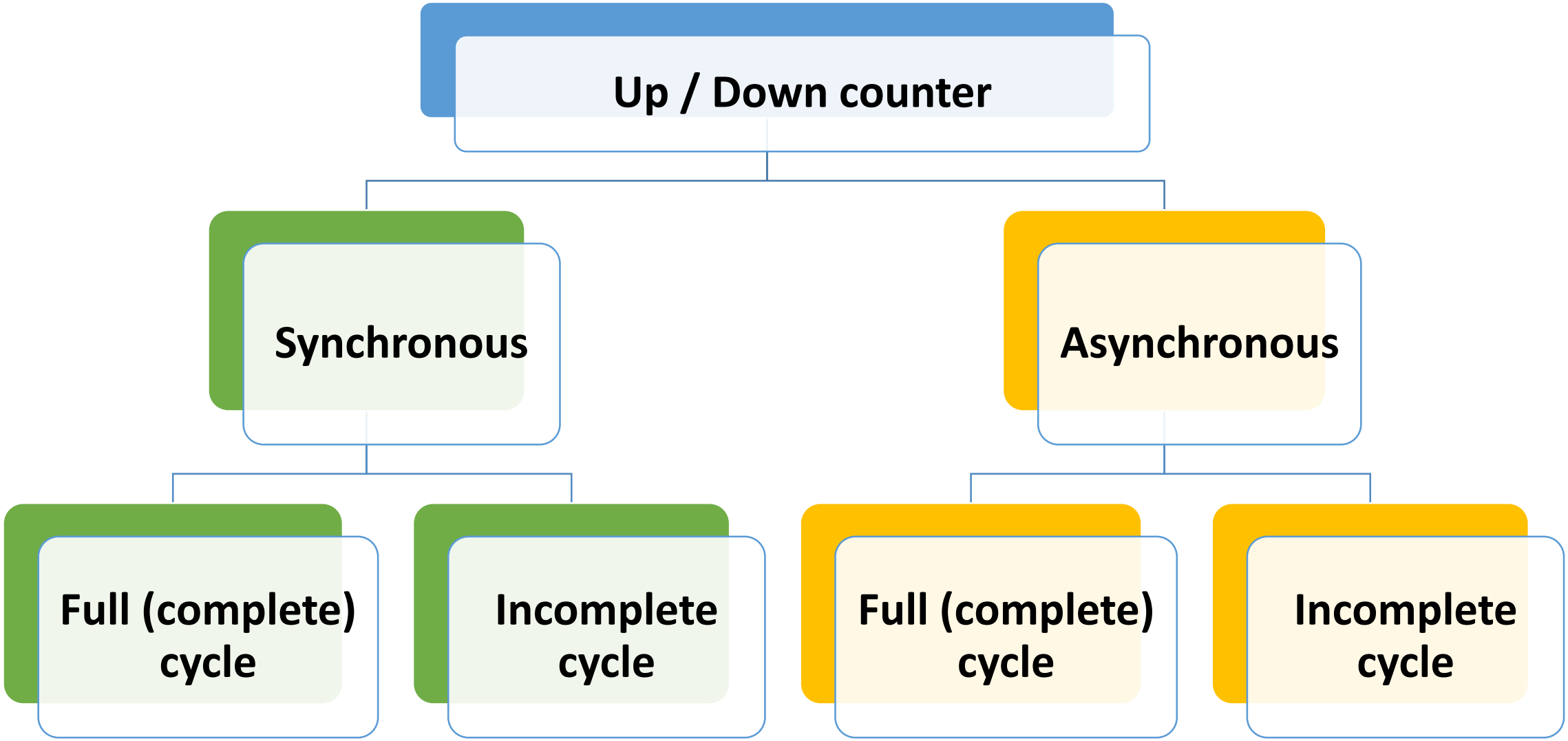
d) Create the corresponding circuit

Synchronous circuits

Ch1. The second part



Ch1. The third part



Registers

Registers: A register is an ordered set of **n flip-flops** which allows information to be stored in **n bits**.

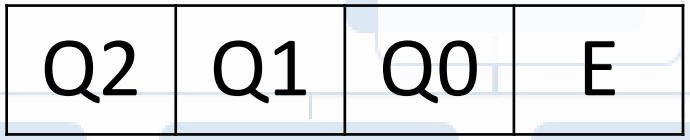
Serial load (shift) registers: Serial **load** or **shift** registers are registers with serial input and serial output.

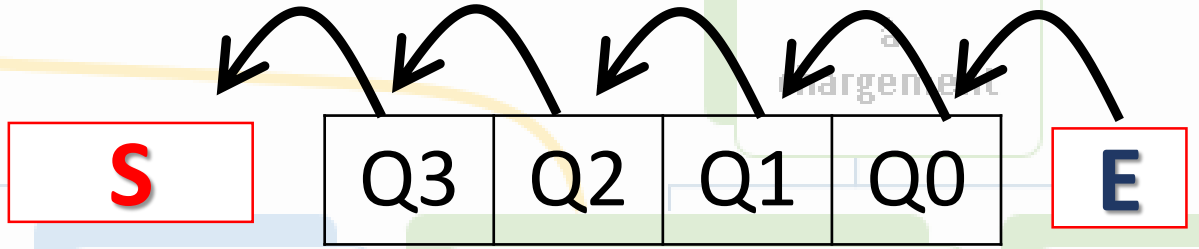
Flip-Flop D (Data)

D	Q^+
0	0
1	1

Registers

Left Shift Register (Left Serial Load Registers) :





The entire register (bits) is shifted one position to the left:

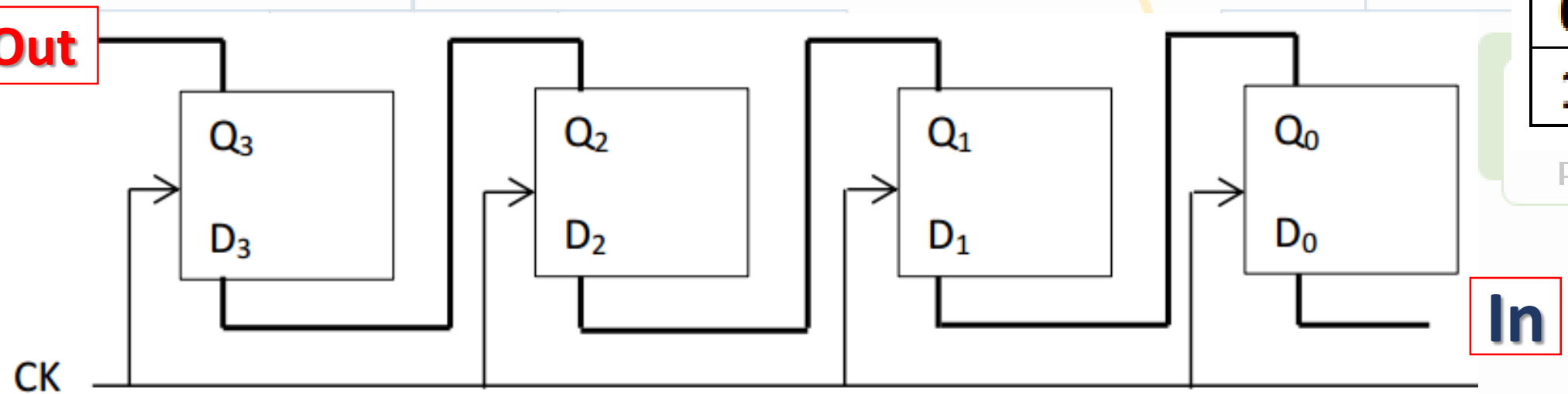
$S = Q3$	$Q3^+ = Q2$	$Q2^+ = Q1$	$Q1^+ = Q0$	$Q0^+ = E$

Registers

Left Shift Register (Left Serial Load Registers) :

S = Q3	D3=$Q3^+$ = Q2	D2=$Q2^+$ = Q1	D1=$Q1^+$ = Q0	D0=$Q0^+$= **E=0**

D	Q^+
0	0
1	1



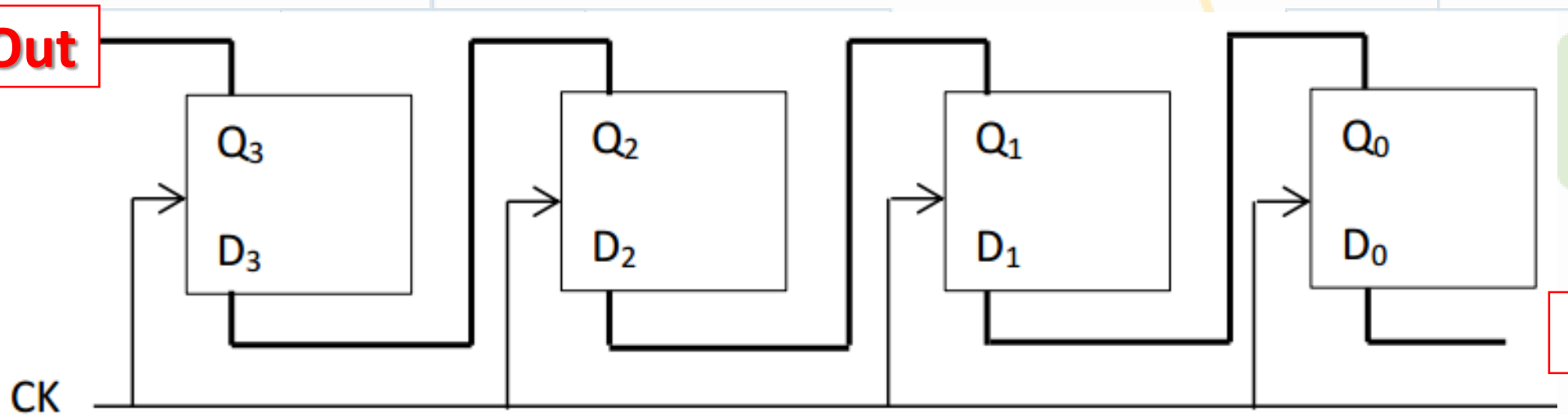
More generally for a register of **n** flip-flops we have :

D_0 = **In** , $D_i = Q_{i-1}$ et **Out** = Q_{n-1}

Registers

Left Shift Register (Left Serial Load Registers) :

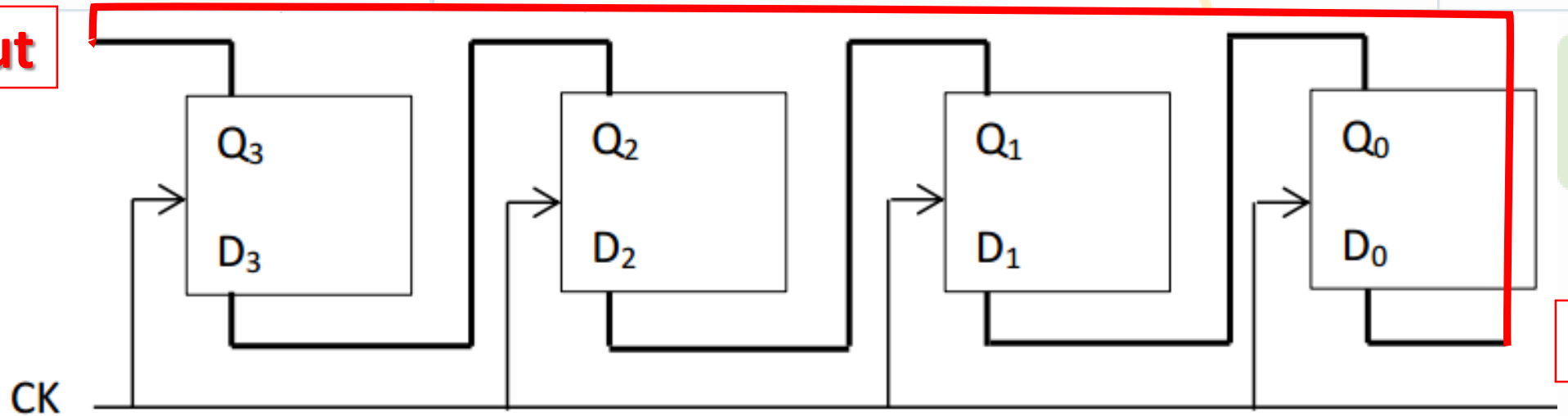
S = Q3	$Q3^+ = Q2$	$Q2^+ = Q1$	$Q1^+ = Q0$	$Q0^+$=**E=0**



Registers

Left circular shift register:

$D3=Q3^{+} = Q2$	$D2=Q2^{+} = Q1$	$D1=Q1^{+} = Q0$	$D0=Q0^{+}=\mathbf{Q3=S}$



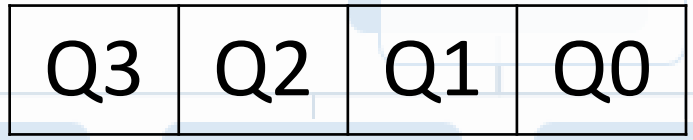
In=Out=Q3

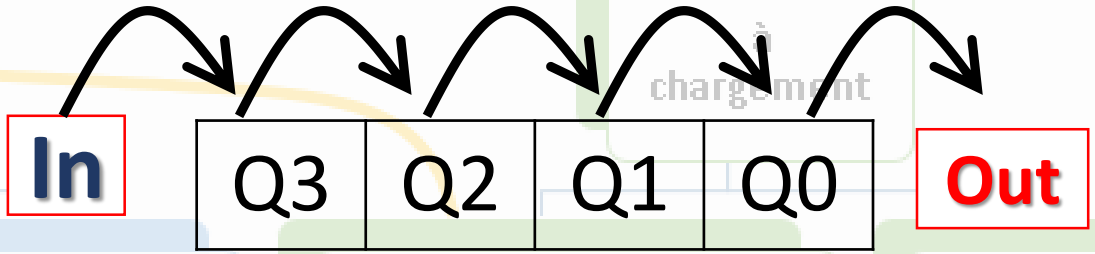
Q3	Q2	Q1	Q0
0	1	1	0

Q3+	Q2+	Q1+	Q0+
1	1	0	0

Registers

Right Shift Register (Right Serial Load Registers) :





The entire register (bits) is shifted one position to the right:

$D3=Q3^{+}=E$	$D2=Q2^{+}=Q3$	$D1=Q1^{+}=Q2$	$D0=Q0^{+}=Q1$	$S=Q0$

D	Q^{+}
0	0
1	1

More generally for a register of **n** flip-flops we have :

$D_{n-1} = In$, $D_{i-1} = Q_i$ et $Out = D_0$

Registers

Parallel Load Registers

Loading information into a register consists of giving this register the value of this information regardless of its previous state.

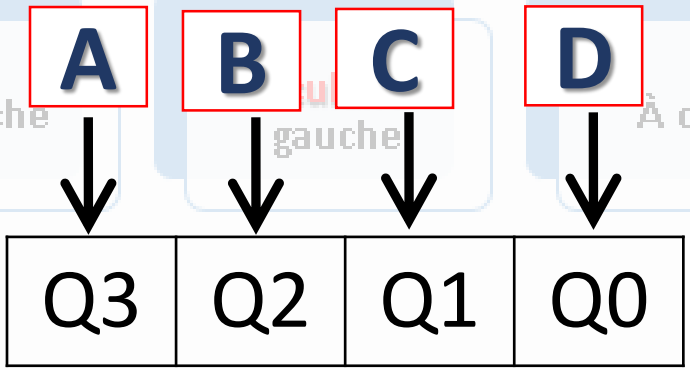
The parallel load register can load n-bit information at the same time.

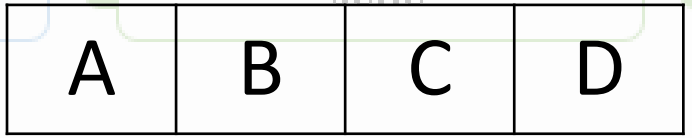
Each flip-flop takes the value of the information contained in the corresponding bit

Registers

Parallel Load Registers

If we want to load, for example, the information A B C D into a 4-bit register, regardless of the state of this register at time t, it will display A B C D at time t+1

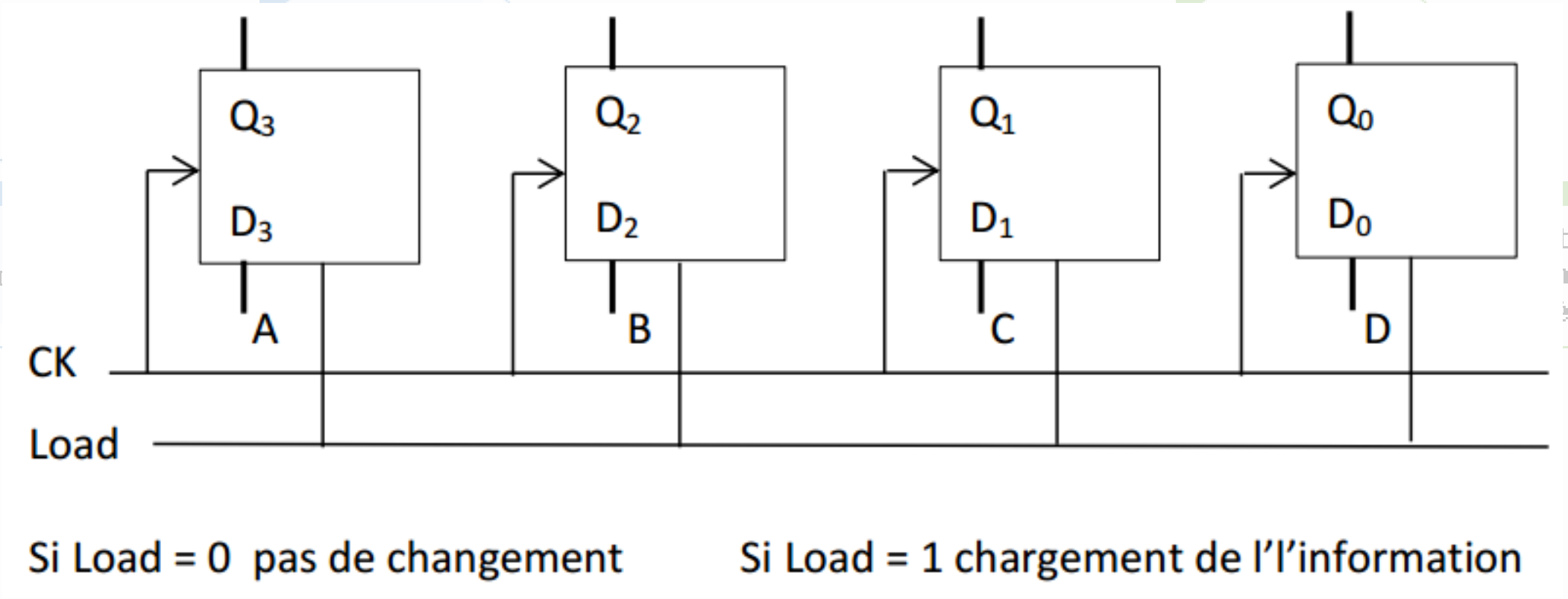




$Q3^+=A$	$Q2^+ = B$	$Q1^+ = C$	$Q0^+ = D$

Registers

Parallel Load Registers

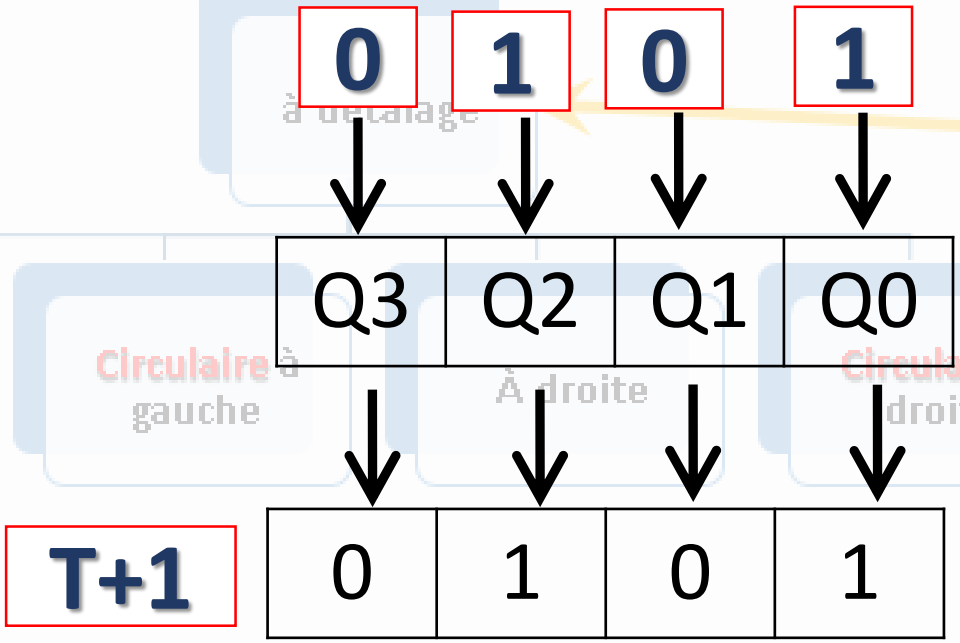


Si Load = 0 pas de changement

Si Load = 1 chargement de l'l'information

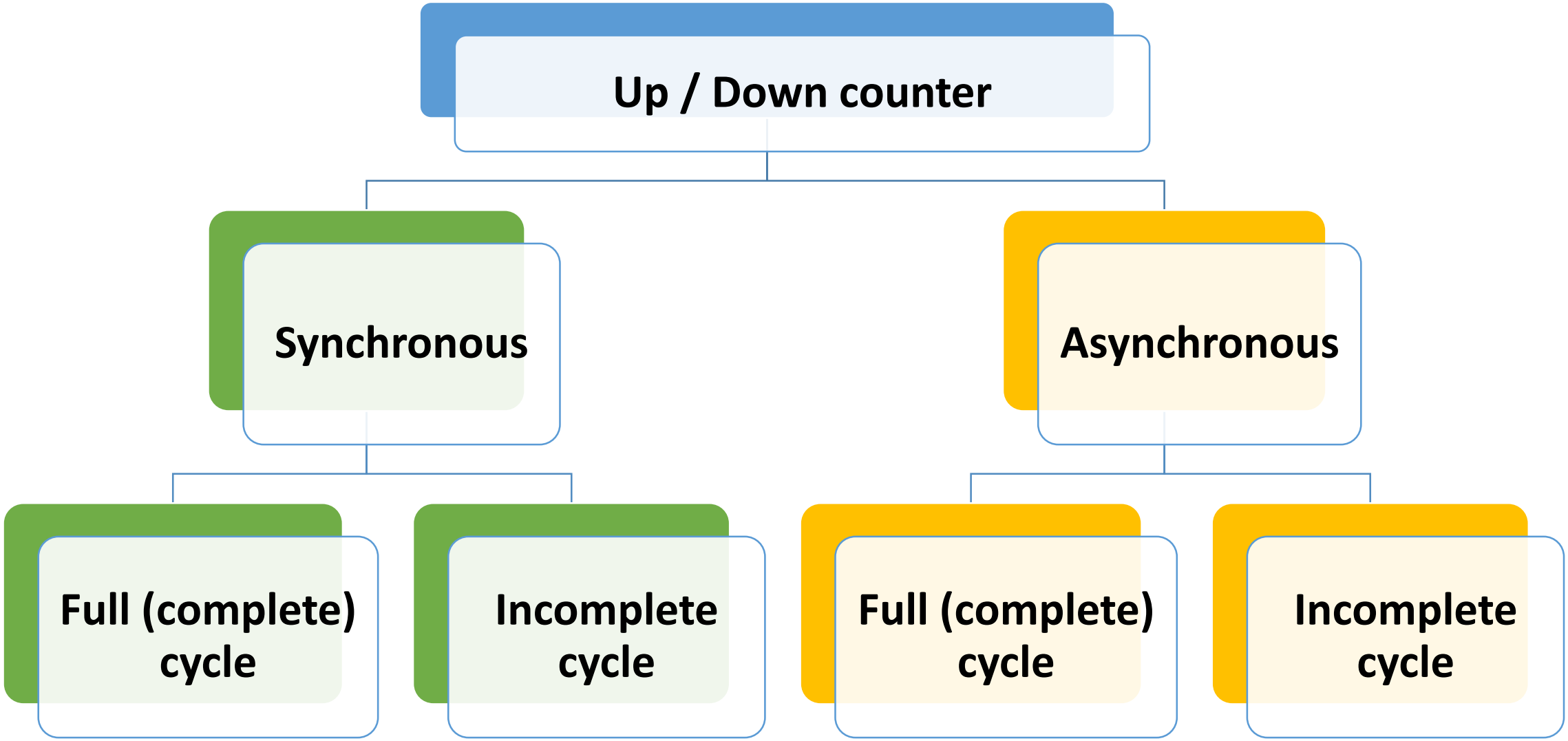
Registers

Parallel Load Registers



$Q3^{+}=0$	$Q2^{+}=1$	$Q1^{+}=0$	$Q0^{+}=1$

Ch1. The third part



Up / Down counter

A up counter is a sequential circuit which has **n** states (0, 1, 2, ……n-1).
At each **clock pulse** it goes from state **i** to state **i+1** and always returns to initial state.

The modulo of an up counter is the number of its states. A modulo 8 counter for example is a counter with **8** states (0, 1, 2, …7)

There is two types of counters : **Synchronous** and **Asynchronous** counters.

Up / Down counter

Synchronous counters

A synchronous counter is a counter where all the flip-flops have the same clock.

The number of flip-flops of a modulo **n** counter is equal to **p** if $2^{p-1} < n \leq 2^p$ is the power of **2** which is immediately greater than **n**

For example, the number of flip-flops of a modulo **6** counter is **3** because

$2^2 < 6 <= 2^3$

Up / Down counter

Full cycle synchronous up counter :

A **full cycle up counter** is a modulo **n** up counter such that $n = 2^p$.

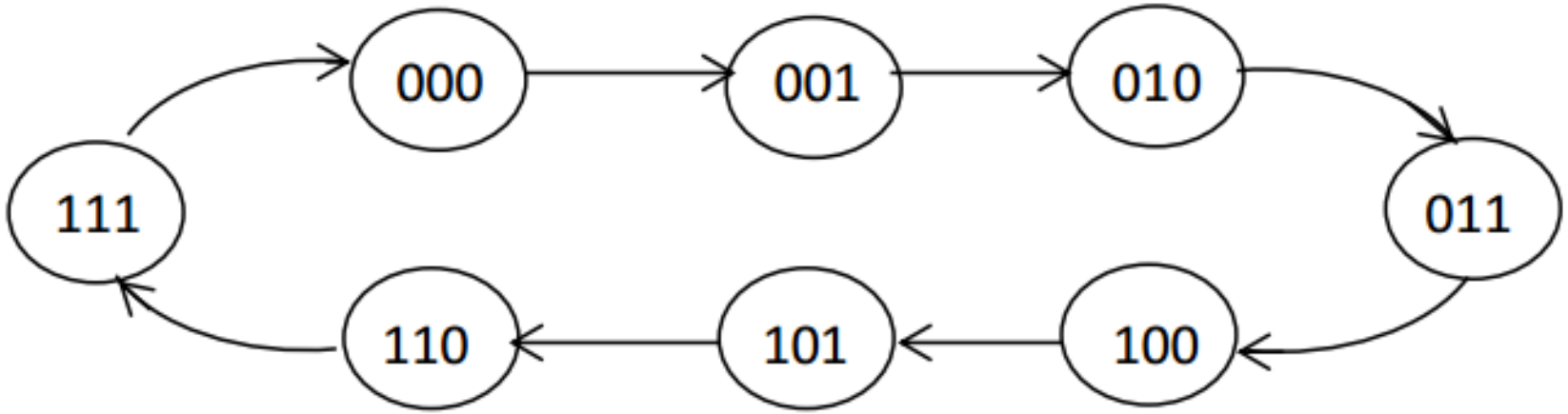
For example the modulo 8 up counter and the modulo 16 up counter are full cycle counters, $8=2^3$ $16=2^4$.

Example: Establishing a modulo **8** up counter using **JK** flip-flops.

Up / Down counter

Example: Establishing a modulo **8** counter using **JK** flip-flops

N=8 ➔ $8 = 2^3$



Up / Down counter

Table de transition

Q_2 Q_1 Q_0	Q_2^+ Q_1^+ Q_0^+	J_2 K_2	J_1 K_1	J_0 K_0
0 0 0	0 0 1	0 X	0 X	1 X
0 0 1	0 1 0	0 X	1 X	X 1
0 1 0	0 1 1	0 X	X 0	1 X
0 1 1	1 0 0	1 X	X 1	X 1
1 0 0	1 0 1	X 0	0 X	1 X
1 0 1	1 1 0	X 0	1 X	X 1
1 1 0	1 1 1	X 0	X 0	1 X
1 1 1	0 0 0	X 1	X 1	X 1

Equations d'entrée :

$J_0 = 1$

$K_0 = 1$

$J_1 = Q_0$

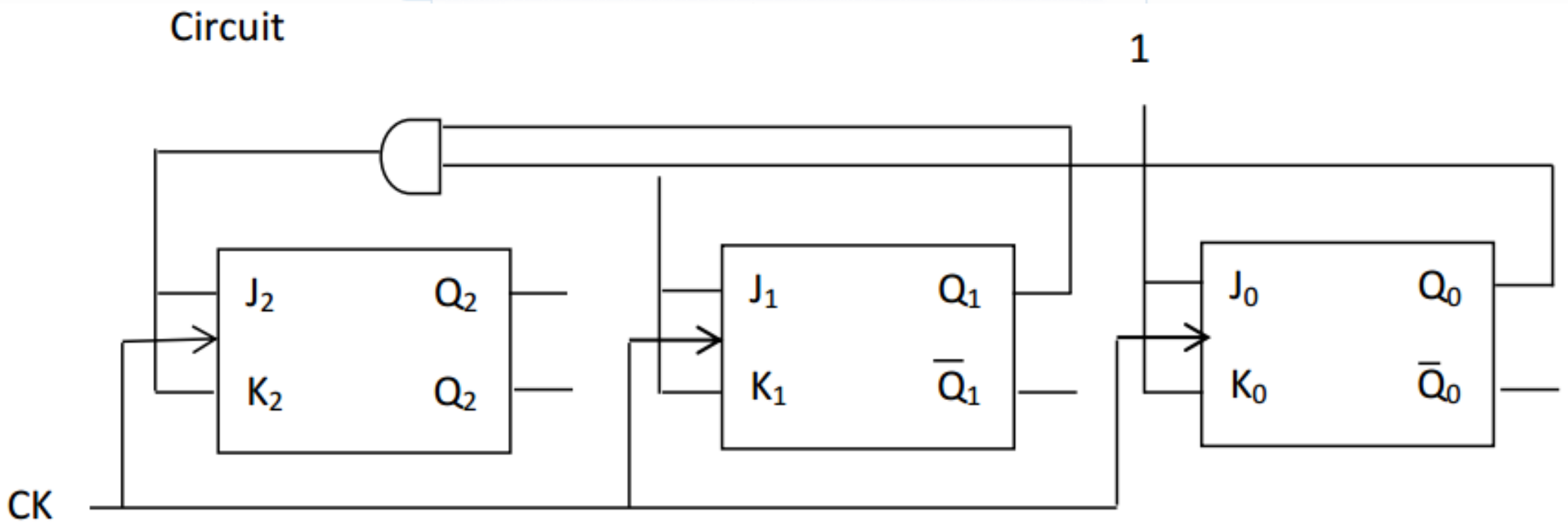
$K_1 = Q_0$

$J_2 = Q_0Q_1$

$K_2 = Q_0Q_1$

Up / Down counter

Circuit



Up / Down counter

Dans un cas plus général on aura :

$J_0 = 1$

$K_0 = 1$

$J_i = Q_0Q_1.....Q_{j-1}$

$K_i = Q_0Q_1.....Q_{j-1}$

à cycle complet

à cycle incomplet

à cycle complet

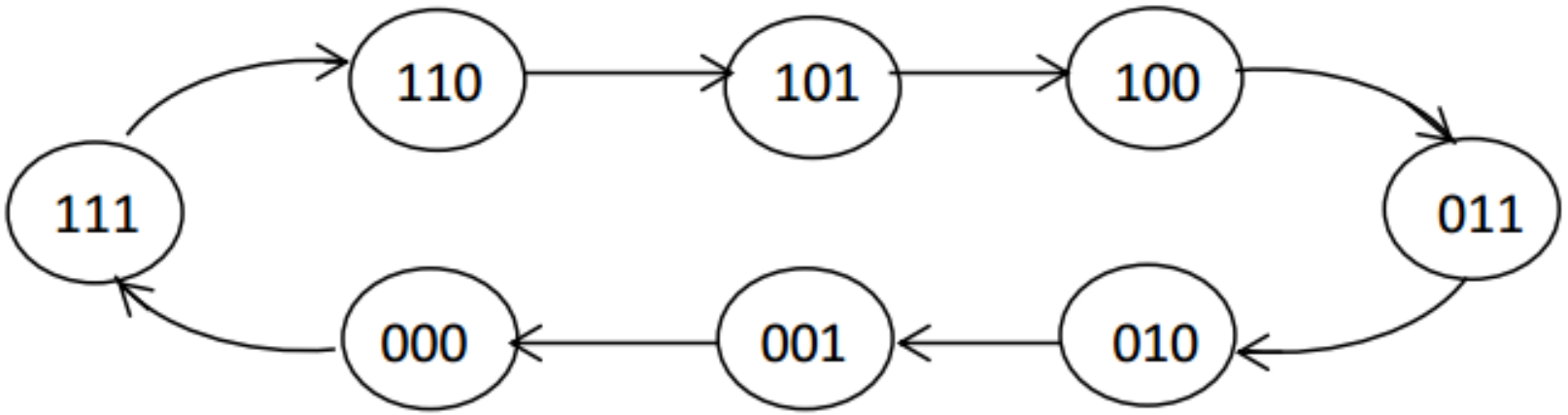
à cycle incomplet

Up / Down counter

Full cycle synchronous Down counter :
A **full cycle down counter** is a modulo **n** down counter such that **$n = 2^p$**.
it goes from state **i** to state **i-1** and always returns to state **n-1**.

For example the modulo 8 up counter and the modulo 16 up counter are full cycle counters,
$8=2^3$ $16=2^4$.

Example: Establishing a modulo **8** down counter using **JK** flip-flops.



Up / Down counter

Table de transition

Q_2 Q_1 Q_0	Q_2^+ Q_1^+ Q_0^+	J_2 K_2	J_1 K_1	J_0 K_0
0 0 0	1 1 1	1 X	1 X	1 X
0 0 1	0 0 0	0 X	0 X	X 1
0 1 0	0 0 1	0 X	X 1	1 X
0 1 1	0 1 0	0 X	X 0	X 1
1 0 0	0 1 1	X 1	1 X	1 X
1 0 1	1 0 0	X 0	0 X	X 1
1 1 0	1 0 1	X 0	X 1	1 X
1 1 1	1 1 0	X 0	X 0	X 1

Equations d'entrée

$J_0 = 1$

$K_0 = 1$

$J_1 = \overline{Q_0}$

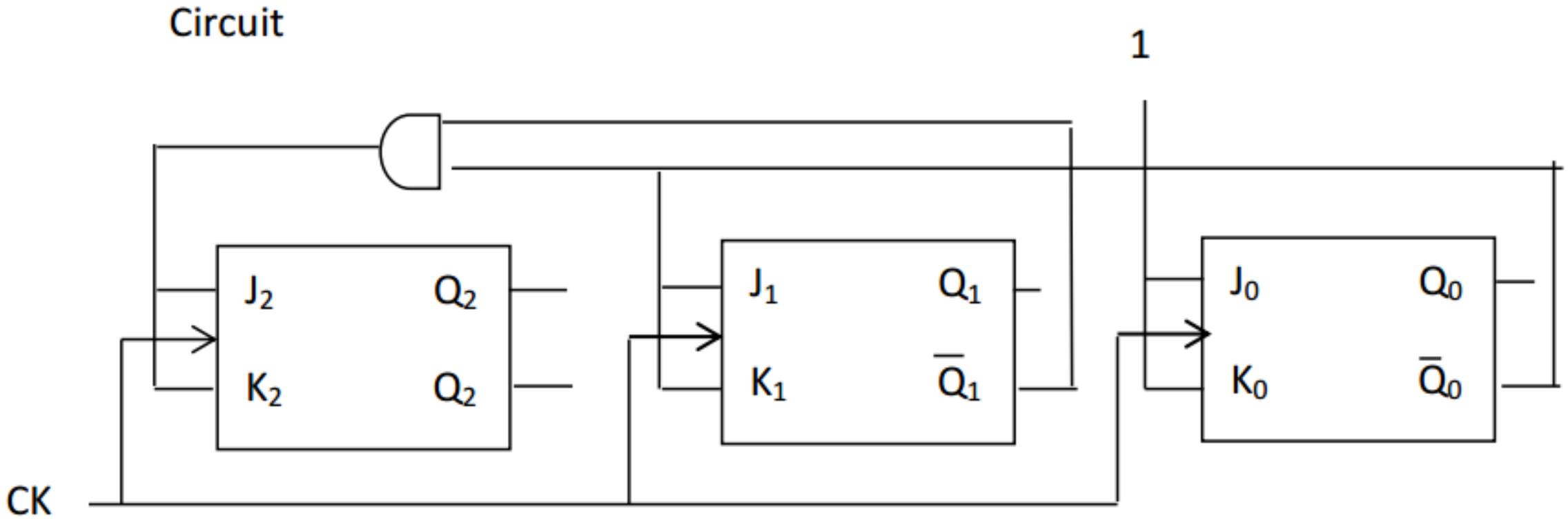
$K_1 = \overline{Q_0}$

$J_2 = \overline{Q_0}\,\overline{Q_1}$

$K_2 = \overline{Q_0}\,\overline{Q_1}$

Up / Down counter

Circuit



Up / Down counter

Dans un cas plus général on aura :

$$J_0 = 1$$
$$K_0 = 1$$

$$J_i = \overline{Q}_0\overline{Q}_1.....\overline{Q}_{j-1}$$
$$K_i = \overline{Q}_0\overline{Q}_1.....\overline{Q}_{j-1}$$

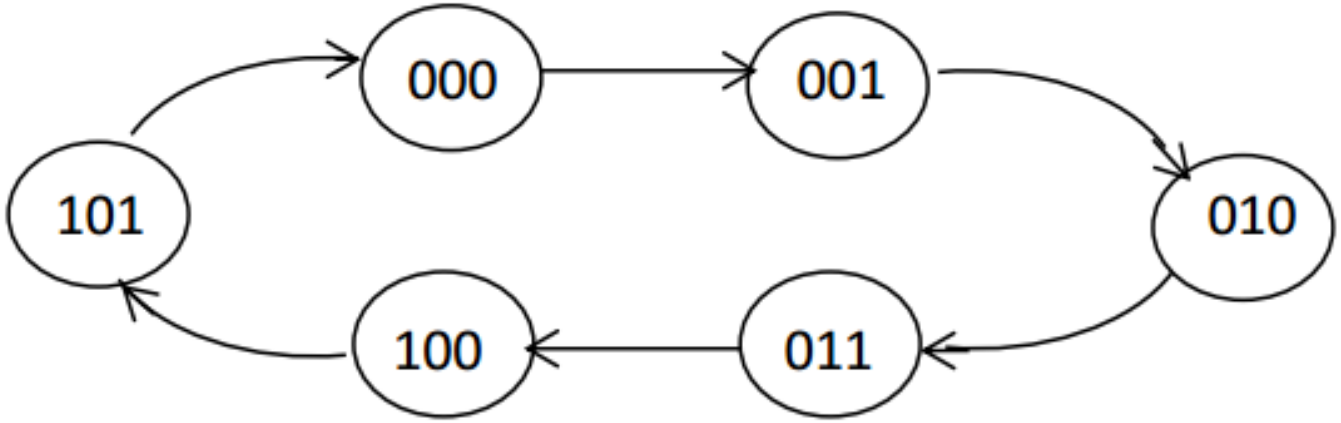
Up / Down counter

Incomplete cycle synchronous counter

An incomplete cycle counter is a counter with **n** states such that $2^{p-1} < n < 2^p$ **(n is not a power of 2)**.

The number of flip-flops of this counter is equal to **p.**

Example: The modulo 6 counter : $4=2^2 < 6 < 8=2^3$



Up / Down counter

Table de transition

Q_2 Q_1 Q_0	Q_2^+ Q_1^+ Q_0^+	J_2 K_2	J_1 K_1	J_0 K_0
0 0 0	0 0 1	0 X	0 X	1 X
0 0 1	0 1 0	0 X	1 X	X 1
0 1 0	0 1 1	0 X	X 0	1 X
0 1 1	1 0 0	1 X	X 1	X 1
1 0 0	1 0 1	X 0	0 X	1 X
1 0 1	0 0 0	X 1	0 X	X 1
1 1 0	X X X	X X	X X	X X
1 1 1	X X X	X X	X X	X X

Equations d'entrée :

$J_0 = 1$

$K_0 = 1$

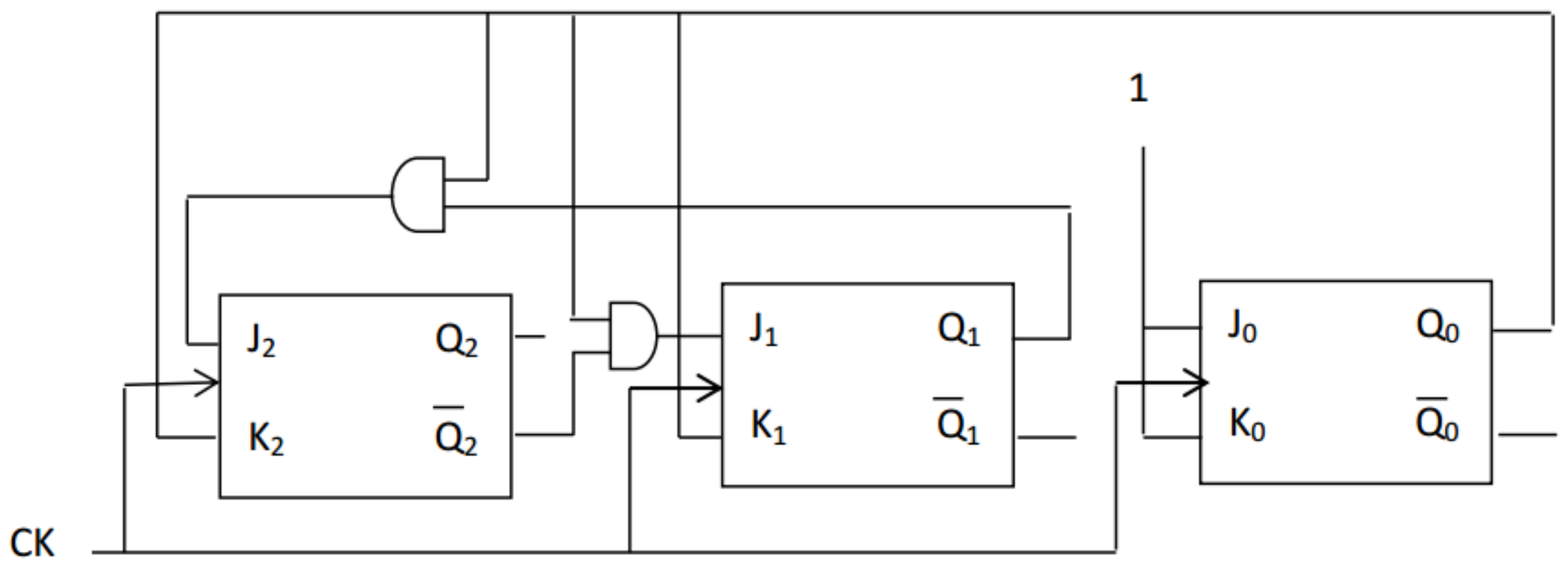
$J_1 = \overline{Q}_2 Q_0$

$K_1 = Q_0$

$J_2 = Q_1 Q_0$

$K_2 = Q_0$

Up / Down counter



Up / Down counter

Asynchronous counters

An asynchronous counter is a sequential circuit where all the flip-flops do not have the same clock.

The clock of each flip-flop depends on the output of the flip-flop that precedes it.

The flip-flop inputs are designed such that their outputs are inverted with each clock pulse;

for example for JK flip-flops we will take J = 1 and K = 1 ($Q_{t+1} = /Q_t$)

To create the circuit of an asynchronous counter, we trace its chronogram then we deduce the circuit

Up / Down counter

Full cycle asynchronous up counter

Like the synchronous counter, the full-cycle asynchronous counter is a modulo n counter such that $\mathbf{n = 2^p}$

Example: modulo 8 asynchronous counter with **rising edge clock**.
For a modulo **8** counter we need **3 flip-flops** because $\mathbf{8 = 2^3}$

The counter is asynchronous so we have :
$J_2 = K_2 = 1$ $J_1 = K_1 = 1$ $J_0 = K_0 = 1$

000 -> 001 -> 010 -> 011 -> 100 -> 101->110 -111 ->000

Up / Down counter

Creating the timing diagram :

1) Trace the functions Q0 Q1 Q2 which carry out the counter sequence
2) Determine the clocks of each Qi function :

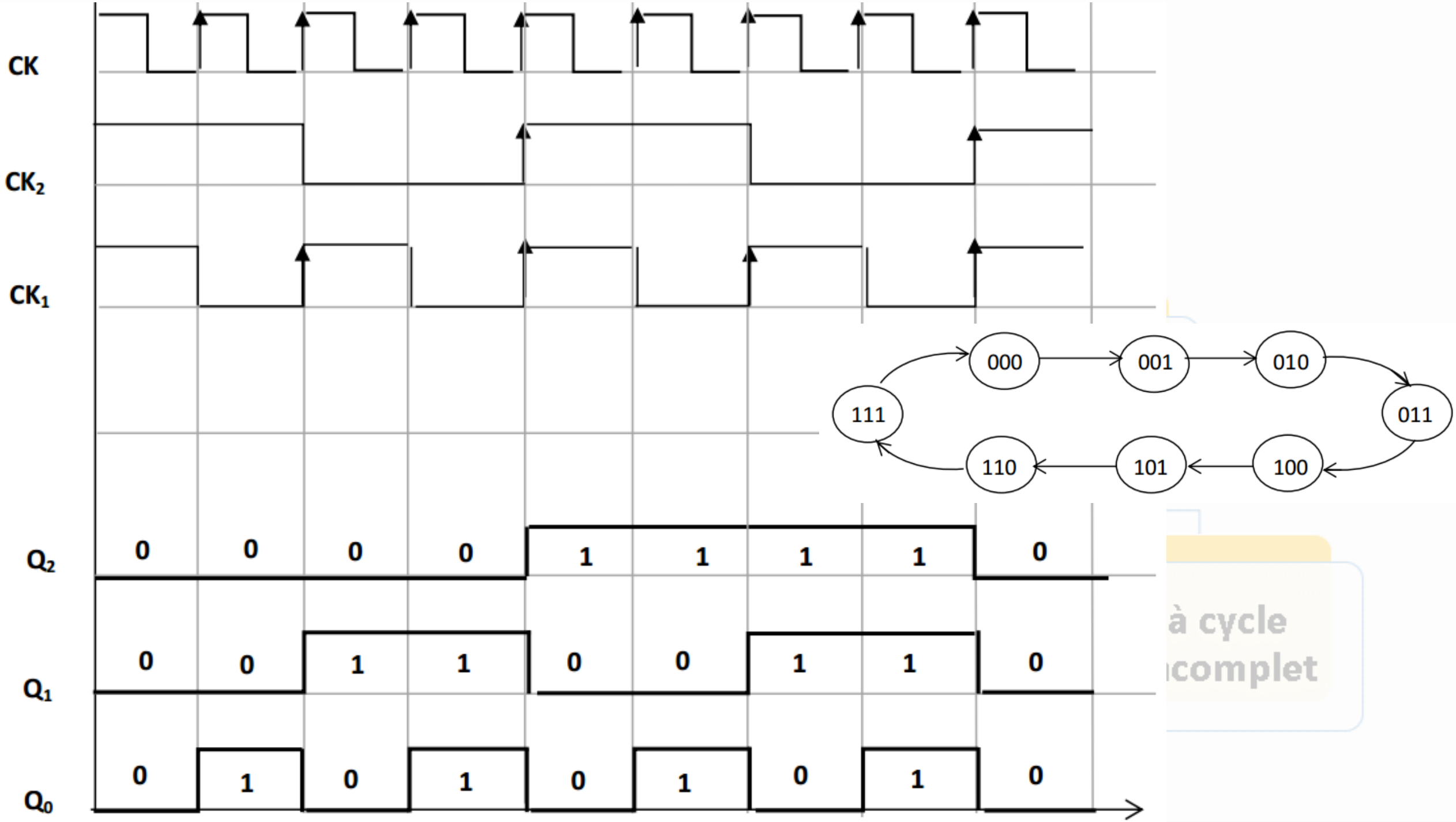
- The change of the state of Q0 occurs at each clock pulse, so CK0 = CK

- The change of state of Q1 takes place every 2 periods so we will have a rising clock edge every 2 periods; this clock corresponds to Q0 so CK1 = Q0

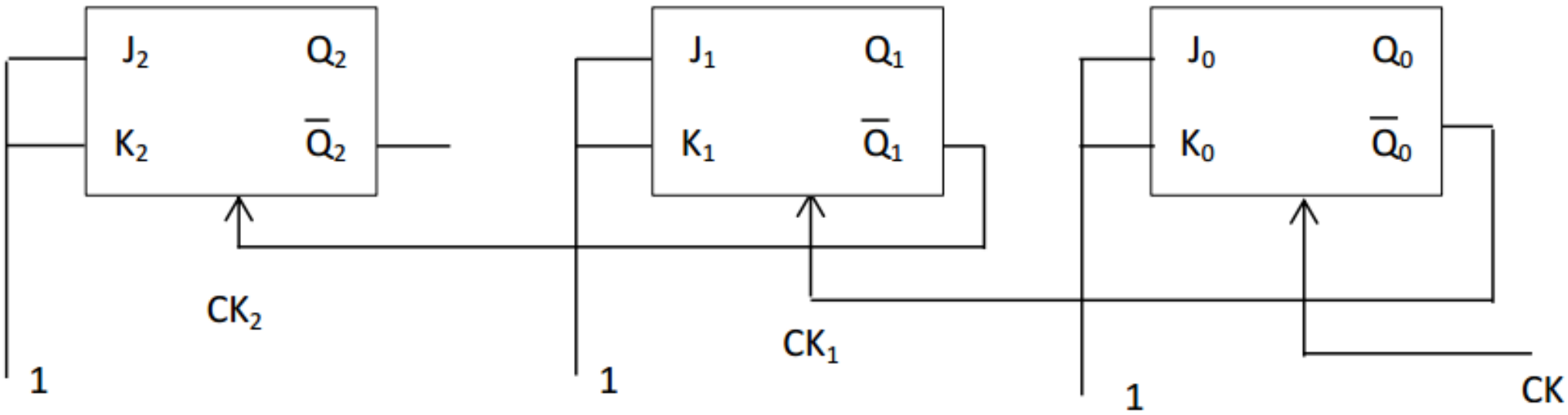
- The change of state of Q2 takes place every 4 periods so we will have a rising clock edge every 4 periods; this clock corresponds to Q1 so CK2 = Q1

For a modulo 8 asynchronous counter we will therefore have :

CK0 = CK CK1 = $\overline{Q0}$ CK2 = $\overline{Q1}$


CK
CK2
CK1
000
001
010
011
100
101
110
111
Q2
Q1
Q0
0 0 0 0 1 1 1 1 0
0 0 1 1 0 0 1 1 0
0 1 0 1 0 1 0 1 0

Circuit



For an asynchronous counter of N flip-flops with rising edge clock we will have :

$CK_0 = CK$ et $CK_i = \overline{Q_{i-1}}$

Up / Down counter

Full cycle asynchronous down counter

We use the same method as for the up counter.
For an asynchronous down counter of N flip-flops with rising edge clock.

we have :
$CK_0 = CK$ et $CK_i = Q_i$

Up / Down counter

Incomplete cycle asynchronous counter

An incomplete cycle counter is a counter with n states such that $2^{p-1} < n < 2^p$ (n is not a power of 2).

The number of flip-flops of this counter is equal to p.
The principle consists of resetting the counter to zero as soon as it reaches n-1.

Example: The modulo 5 up counter 5 (0→4) $4=2^2 < \mathbf{5} < 8=2^3$
The counter has a Clear function which resets it to zero as soon as it reaches 5
(Clear is asynchronous).

Up / Down counter

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	Clear
0	0	0	0	0	1	0
0	0	1	0	1	0	0
0	1	0	0	1	1	0
0	1	1	1	0	0	0
1	0	0	1	0	1	0
1	**0**	**1**	**0**	**0**	**0**	**1**
1	1	0	X	X	X	0
1	1	1	X	X	X	0

The counter resets to zero when it reaches 5.

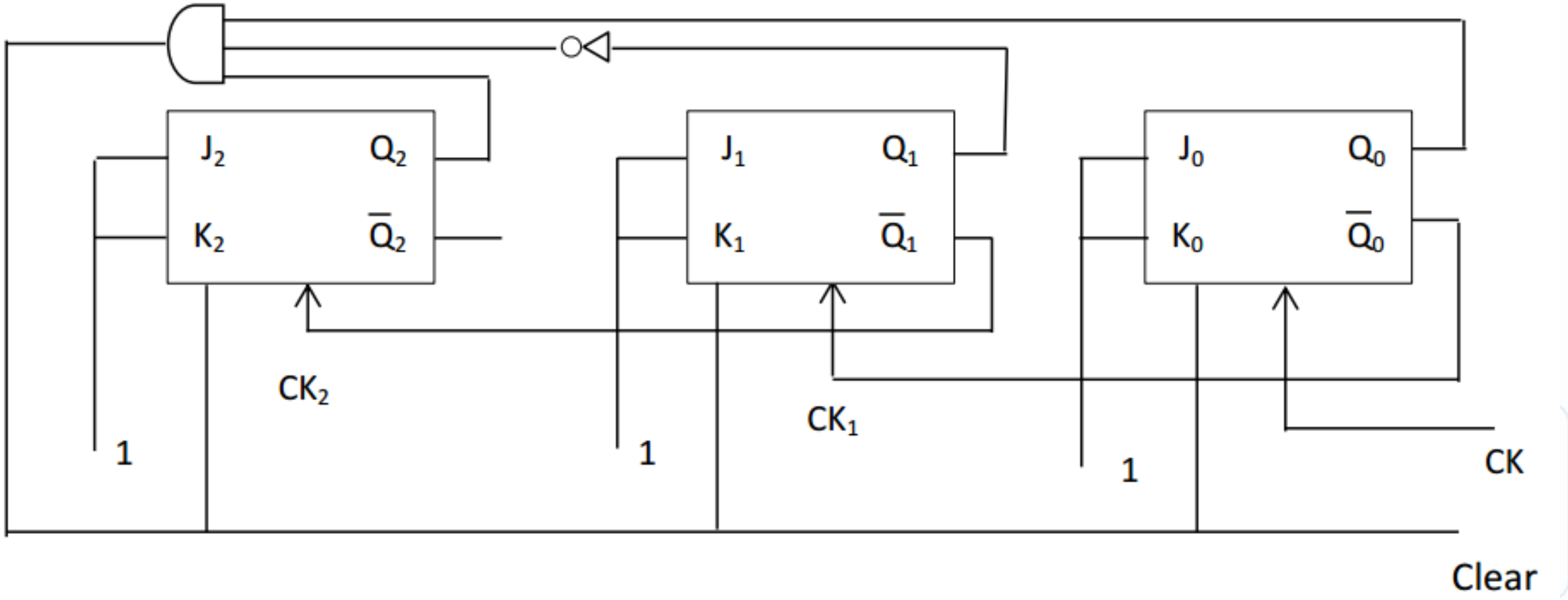
The Clear function is therefore activated when the counter reaches 5.

The equation of the Clear function is given by the truth table:

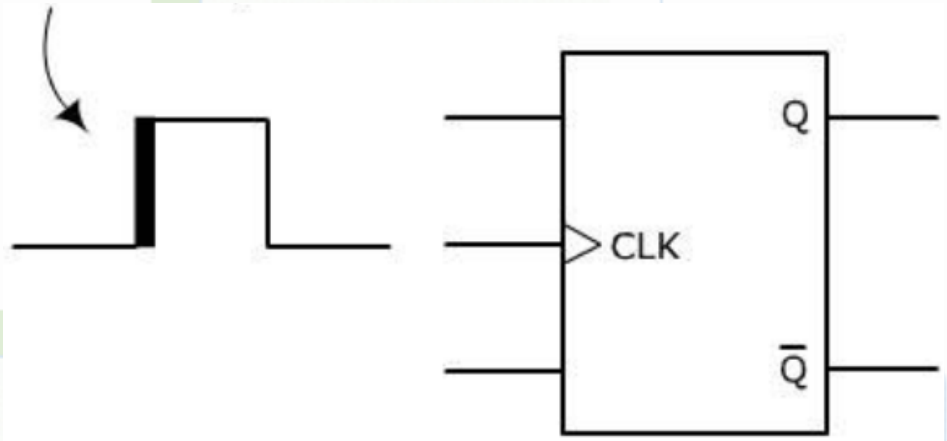
Clear = Q2 $\overline{Q1}$ Q0

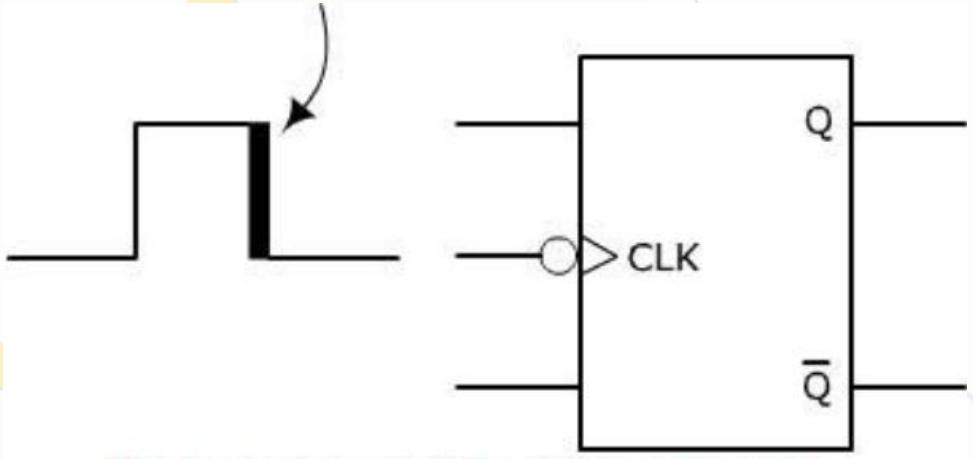
Up / Down counter

Circuit :



Up / Down counter





Chapter 2 : Memories

Definition

A memory is a device capable of storing and retaining information in such a way that a user can access it at any time.

Memoires

Hierarchy of memories

The memory elements of a computer are divided into several levels characterized by their capacity and their access time.

Primary Memory: It is the main organ for storing information used by the CPU (Processor).
To execute a program, it must be loaded into primary memory (instructions and data).

Cache Memory : It is a low capacity memory used as a buffer between the CPU and primary memory. It allows the CPU to make fewer accesses to primary memory and thus save time.

Secondary Memories : Secondary memories are also called mass memories, they are peripheral memories of large capacity and relatively low cost. They serve as a permanent storage element.

Memoires

Organization of information

The information present on a computer is organized according to a certain format whose general characteristics are presented as follows :

Bit : The bit constitutes the basic unit of information. In a memory, the smallest storage element is called a memory point, it stores one bit of information.

Byte : The basic unit, it corresponds to a grouping of 8 bits.

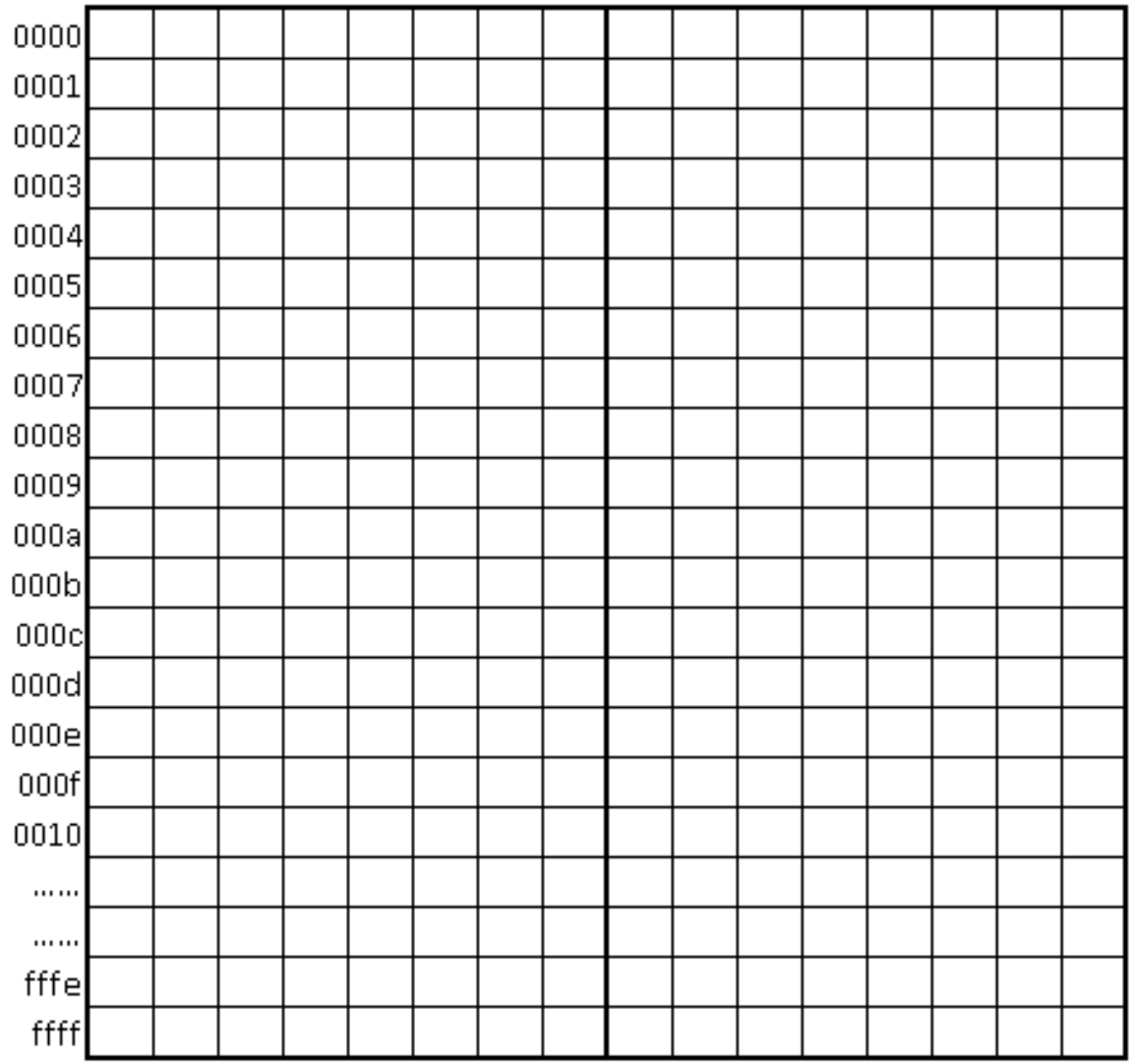
Character (Char) : The character is a group of 8 bits allowing the coding of an alphanumeric character or a special character according to the coding convention (in ASCII code, a character is coded on one byte)

Memoires

Organization of information

Word : The word is a set of bits constituting a unit of information addressable in central memory. It consists of 2, 4 or 8 bytes depending on the computer configuration.

Address: The address is the numerical value designating a memory element (memory word)



Memoires

Memory characteristics

Capacity : Memory capacity is the size of the memory, it corresponds to the number of information it can contain.

This value can be expressed as a function of the number of bits, the number of bytes or the number of words.

Units of measurement of memory capacity:

1 Kb = 1 Kilo Byte = 2^{10} Byte
1 Mb = 1 Méga Byte = 2^{20} Byte
1 Gb = 1 Giga Byte = 2^{30} Byte
1 Tb = 1 Tera Byte = 2^{40} Byte

Access time: Memory access time is the time required between the start of an operation (read or write) and its completion.

Memoires

Memory characteristics

Flow : the flow is the volume of information exchanged per unit of time, it is expressed in bits per second

Volatility : Volatility characterizes the ability of a memory to retain data or not when it is no longer electrically powered.
A volatile memory loses its contents when it is no longer powered by electrical current.
RAM is volatile while **auxiliary memory** such as the **hard drive** is not **volatile**.

Sequential access : Sequential access is relatively long because to access a piece of data you have to go through all the information that precedes it.

Direct access : In this case the information has its own address through which it can be accessed directly.

Memoires

Random access memory (RAM)

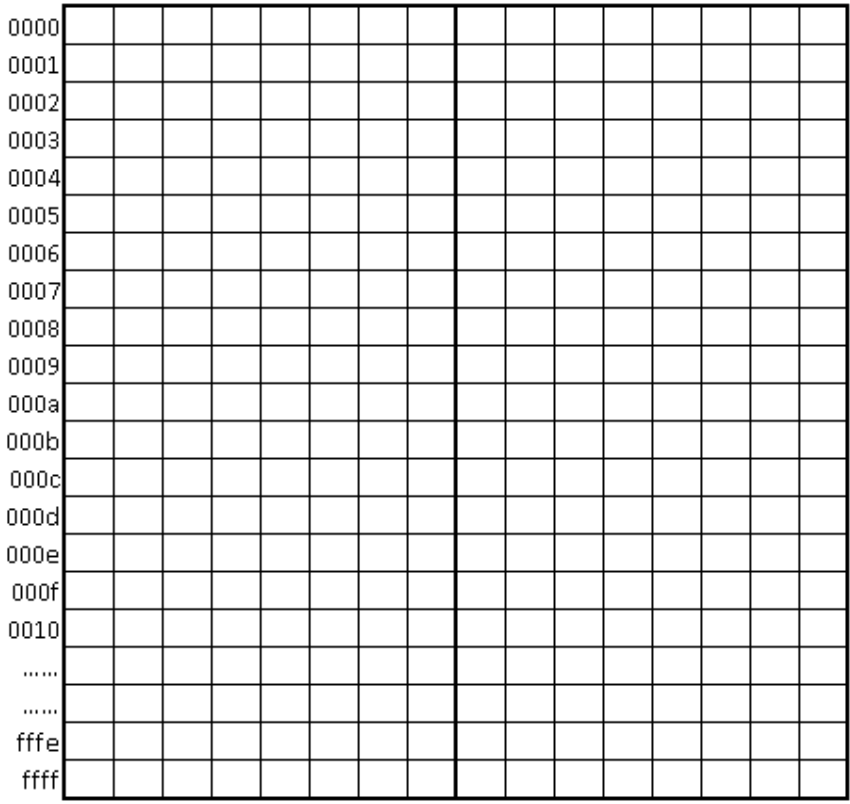
Random access memory (RAM), is direct access memory.

It is the memory in which the computer places the data during its processing (reading or writing).

RAM only works when the computer is turned on.

RAM consists of m words of n bits.

The expression which represents this RAM is: **RAM m x n**



Memoires

Random access memory (RAM)

Each RAM is associated with the following elements:

CS (Chip Select) : the input that allows to activate or deactivate the RAM

MAR (Memory Address Register): It contains an address which allows direct access to a specific word in RAM using a decoder.

MBR (Memory Buffer Register): It is a temporary register through which all data moves from Random Access Memory (RAM) to Central Processing Unit (CPU) and vice versa.

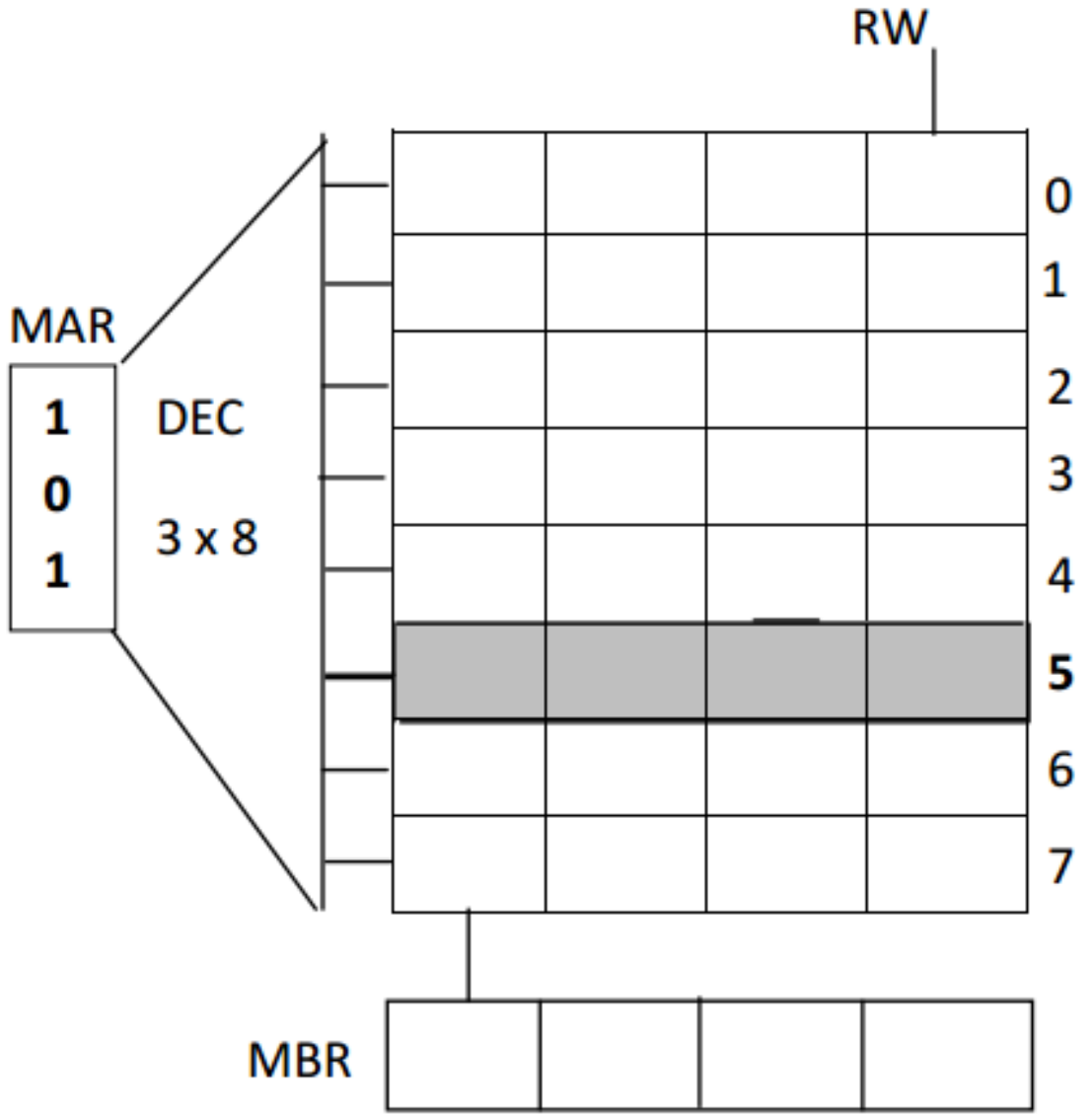
$\boldsymbol{R/\overline{W}}$ (Read/Write) : It is the read-write function (RW = 1 for read and RW = 0 for write)

Memoires

The MAR of this RAM shows the value **(101)** so it is the address **5** of the RAM which is selected in the **DEC 3x8**.

If the **RW function = 1** it is a reading;
The **address word 5** will be transferred to the MBR.

If the **RW function = 0** it is a write;
The word found in the MBR will be transferred to memory at **address 5**

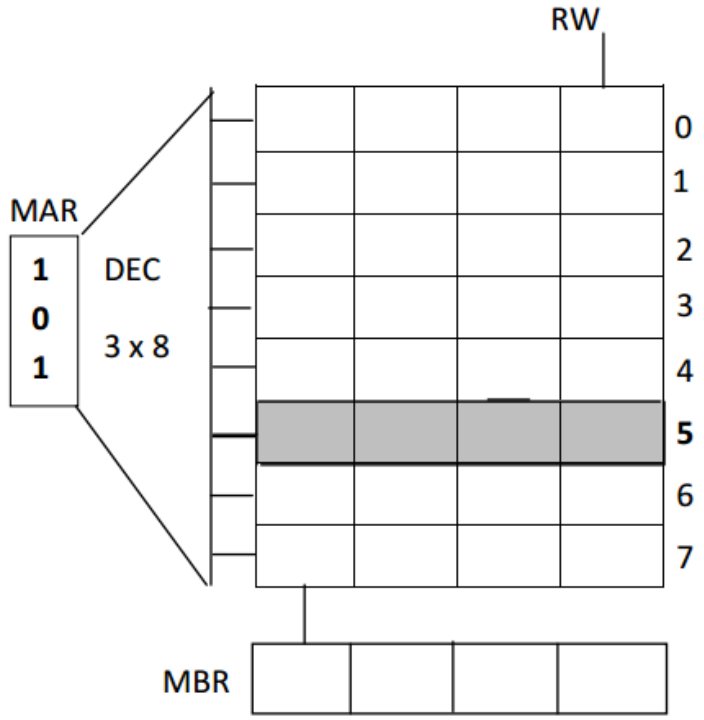


Memoires

Information is transferred between RAM and registers via bus lines;

There is an address bus between the MAR and the RAM, and a data bus between the MBR and the RAM.

The total number of bits in this RAM is 8x4, i.e. 32 bits.



Memoires

Definition of a bus :

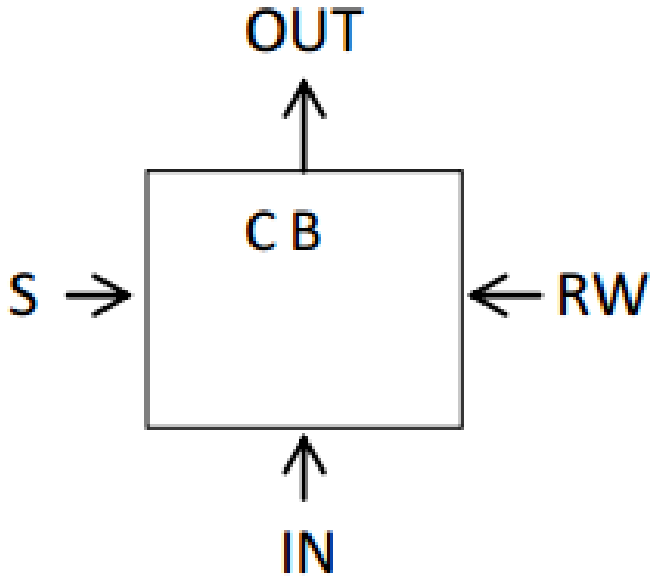
A computer bus is a means of transferring information from one part of the computer to another. It contains as many lines as the registers that use it.

The data bus is bidirectional, consisting of 2 paired lines that allow operation in both directions (read and write).

Memoires

Each bit represents a **binary cell**, and each binary cell has 3 inputs and one output.

S is the decoder line that selects the binary cell**.**
R/$\overline{W}$: is the read/write function.
IN : (Input) represents the information to be written to the binary cell.
OUT : (Output) represents the information to be read from the binary cell.



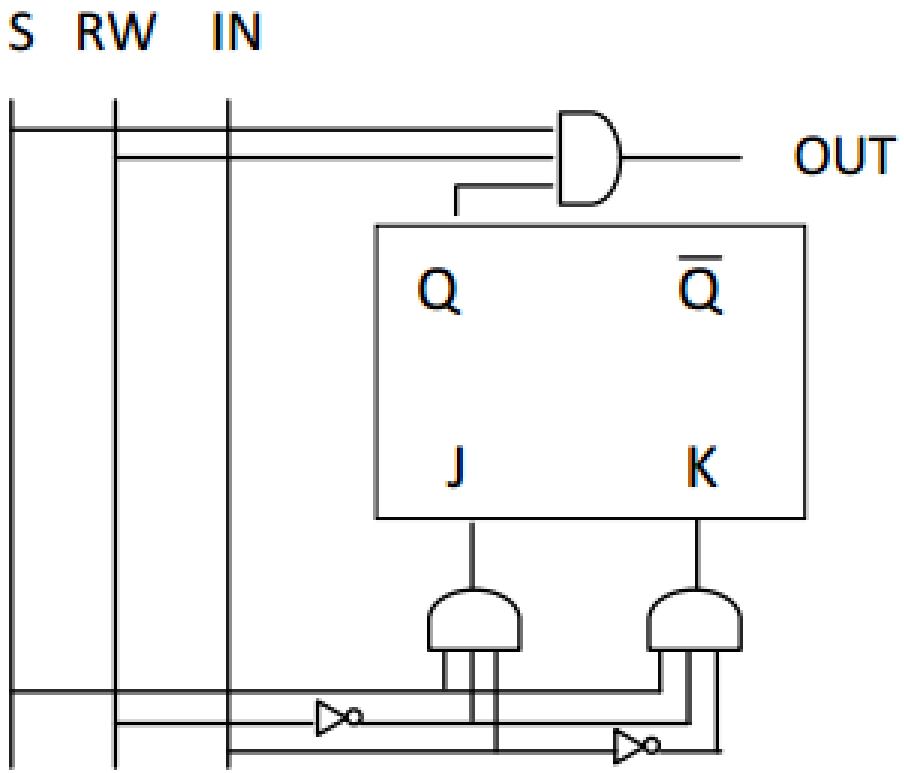
Representation of a binary cell using a JK flip-flop :

1) **S = 0** the binary cell is not selected
There is no change in the JK flip-flop.
J = 0 and K = 0

2) **S = 1** Binary cell is selected (i.e. active)
If **RW = 0**, a write operation is performed.
If **RW = 1**, a read operation is performed.

CB Table of Truth :

S	RW	IN	J	K
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0



Memory expansion (extension)

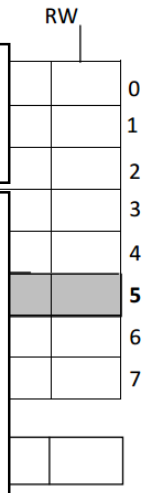
Memory expansion (extension)

To create **large-capacity** memories, **small-capacity** memories can be used and connect them together.

If you want to increase **the number of words**, we connect the memories in parallel by increasing the address bus.

If you want to increase the memory **word length**, connect the memories in series, increasing the data bus.

Memory expansion (extension)



Increasing address space :

If you want to increase the address space of a RAM, you need to connect several RAMs in parallel; to do this, they must have **words of the same length.**

To create **a RAM (2^m x P)** using **RAMs (2^n x P)**, first define the number of RAMs **(2^n x P)** required.

Simply divide the capacity of the RAM required by the capacity of the RAMs proposed:
$(2^m \times P) / (2^n \times P) = 2^{m-n}$

We'll use an address register (MAR) of **n bit**, a data register (MBR) **of P bit** and **(m-n)** complementary addresses.

Example 1 : m = 10 n = 9 and p = 8

Memory expansion (extension)

Example 1 : m = 10, n = 9 and p = 8
We want to create a **2^{10}x8 RAM** from **2^9 x 8** RAM.

The number of RAMs required is :
2^{10}x8 / 2^9 x8 = 2 RAMs

We'll have **a 9-bit MAR** and **(m-n)=1 bit of** complementary address, **the MBR** contains **8 bits**.

Memory expansion (extension)

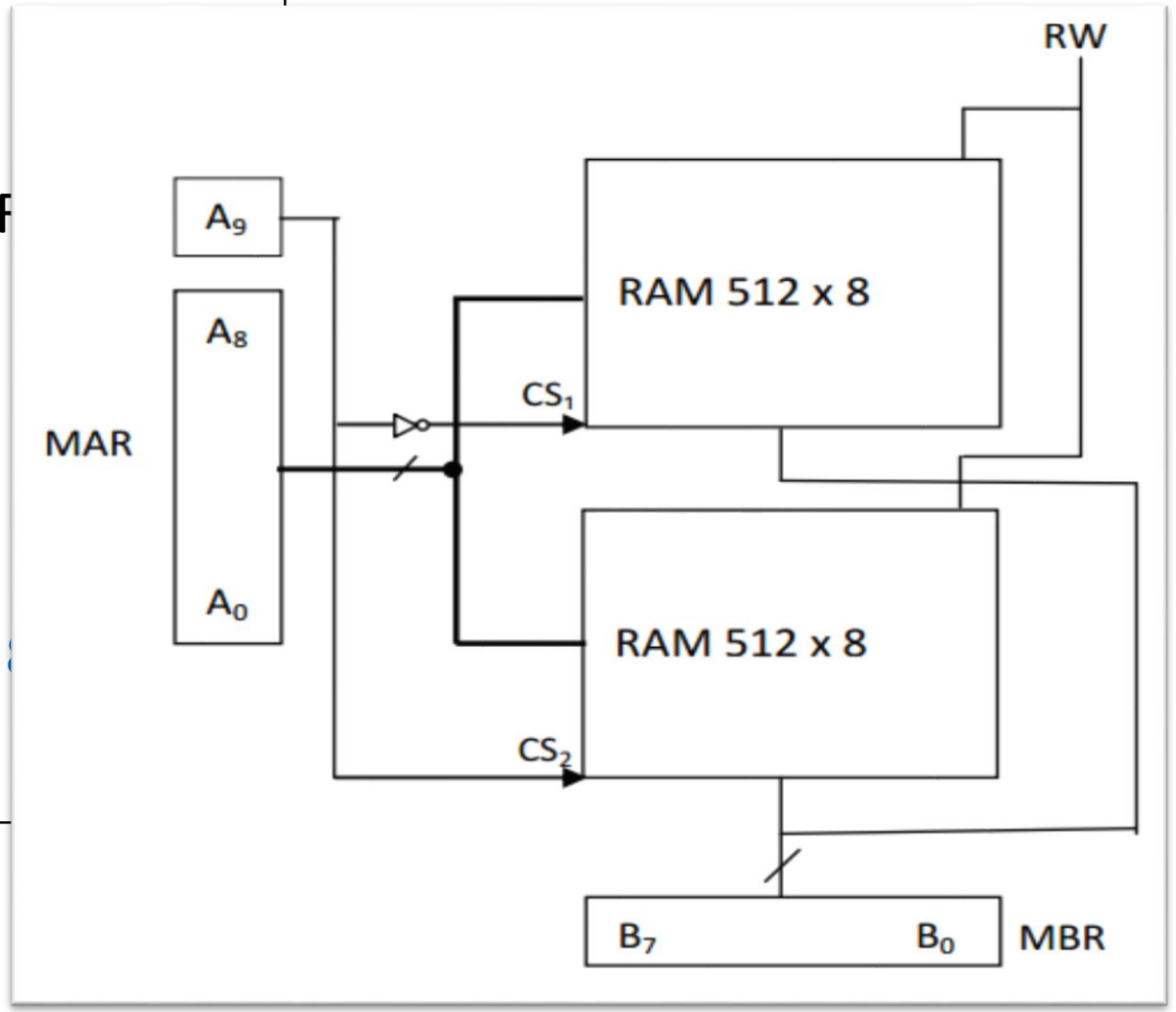
Example 1 : m = 10, n = 9 and p = 8

We want to create a **2^{10}x8 RAM** from **2^9 x 8** R

The number of RAMs required is :

2^{10}x8 / 2^9 x8 = 2 RAMs

We'll have **a 9-bit MAR** and **(m-n)=1 bit of** complementary address, **the MBR** contains



Memory expansion (extension)

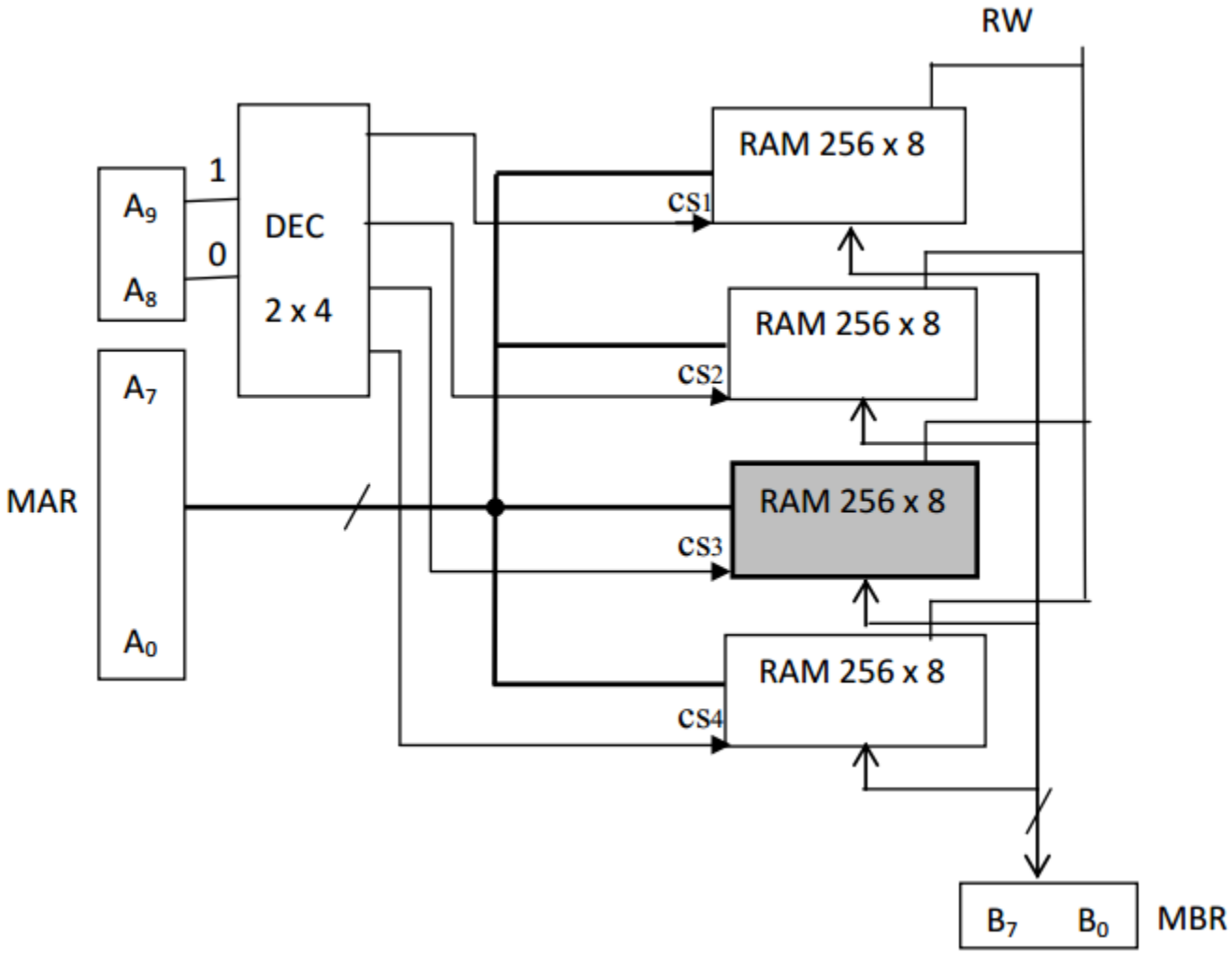
Example 2 : m = 10 n = 8 and p = 8

We want to create a **1K x 8 RAM** from **256 x 8 RAM**.

We want to create a **2^{10} x 8 RAM** using **2^{8} x 8 RAMs**

RAMs used = **2^{10}x 8 / 2^{8} x 8 = 2^{10-8} = 4**

(m-n)=2 bit of complementary address



Memory expansion (extension)

Increasing word length :

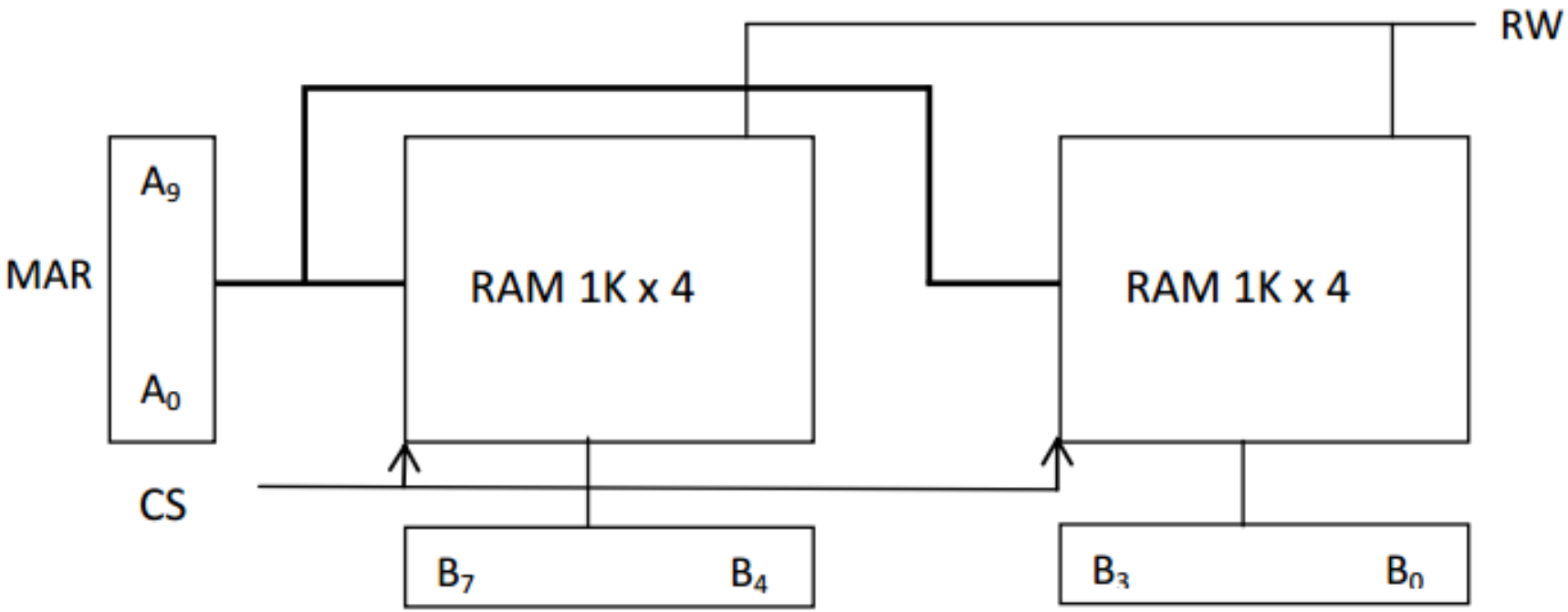
If you want to increase the length of a memory word, you need to connect several **boxes in series**; to do this, they must have the same number of words.

To create a **RAM (2^n x P)** using **RAMs (2^n x Q)**, first define the number of RAMs **(2^n x Q)** required. Simply **divide P by Q**

We'll use **a single address register (MAR)** of **n bit**, **a** virtual **data register of P bit** made up of several registers of **Q bit** (the number of these registers is equal to **P/Q**) and the same **Chip Select** for **both RAMs**.

Memory expansion (extension)

Example: **n = 10 P = 8 and Q = 4**
We want to create a **RAM2^{10} x 8** from **RAM2^{10} x 4**
The number of RAMs required is: **8 / 4 = 2 RAMs**
We'll have a **MAR** of **10-bit** and the **MBR** contains **2 x 4 = 8 bits.**



Memory expansion (extension)

Increased address space and word length:

To create a **RAM (2^m x P)** using **RAMs (2^n x Q)**, we first need to define the total number of **RAMs (2^n x Q)** required. To do this, we divide the number of bits of the RAM we want to create by the number of RAMs proposed:

$(2^m \times P) / (2^n \times Q) = 2^{m-n} \times P/Q$

This gives **2^{m-n} RAMs** in parallel, each composed of **(P/Q) RAMs** in series.

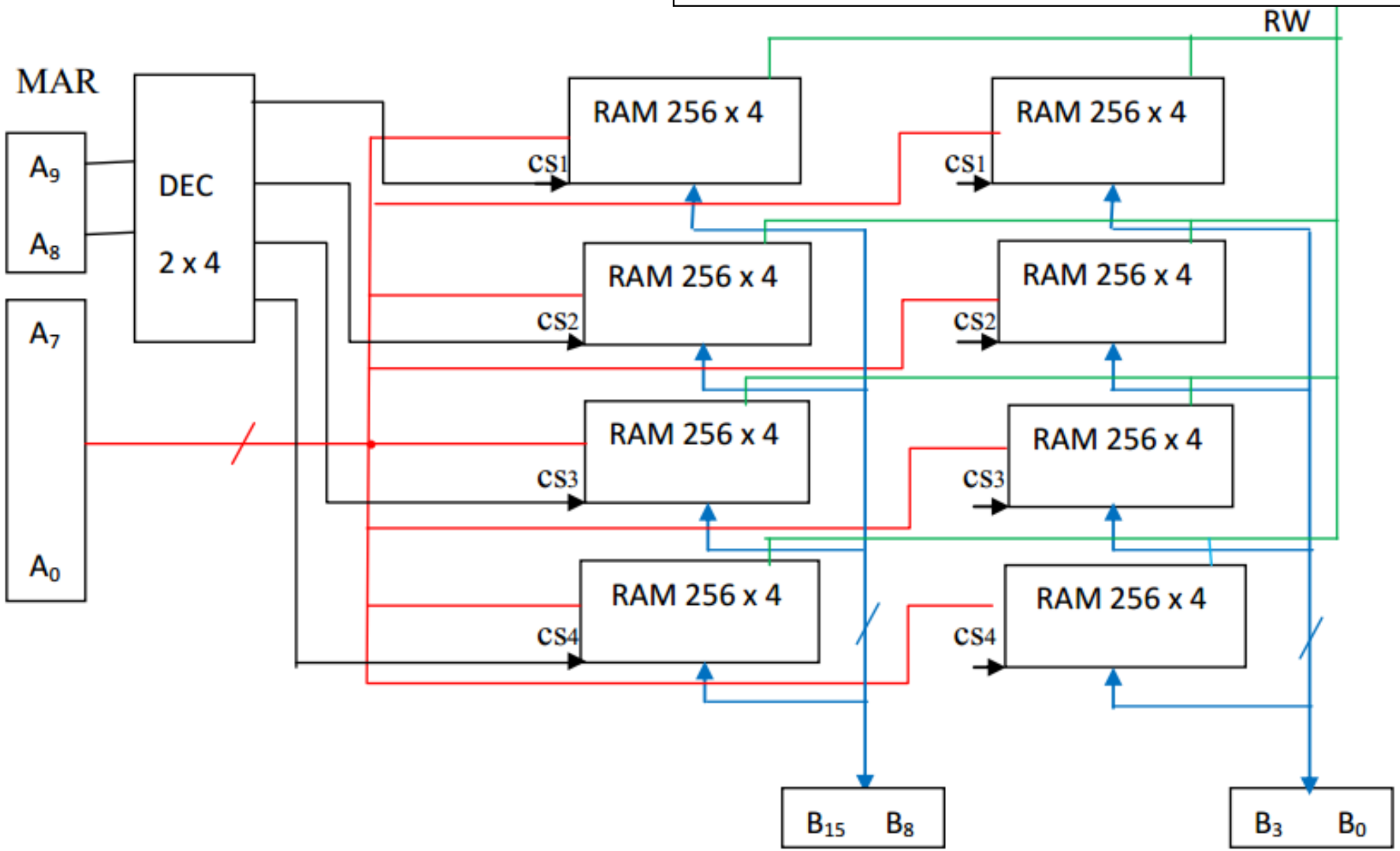
Note: Word numbers and **word lengths** are divided **separately**.

Memory expansion (extension)

Example: **m = 10 P = 8 and n = 8 Q = 4**

We want to create a **2^{10} x 8 RAM** from a **2^{8} x 4 RAM**.
The number of RAMs required is :
Parallel = $2^{10}/2^{8}$ = 4,
Serial = 8/4 = 2,

Memory expansion (extension)



Memoires

ROM read-only memory

ROM (Read Only Memory) is a permanent, non-volatile, and read-only memory, unlike RAM.

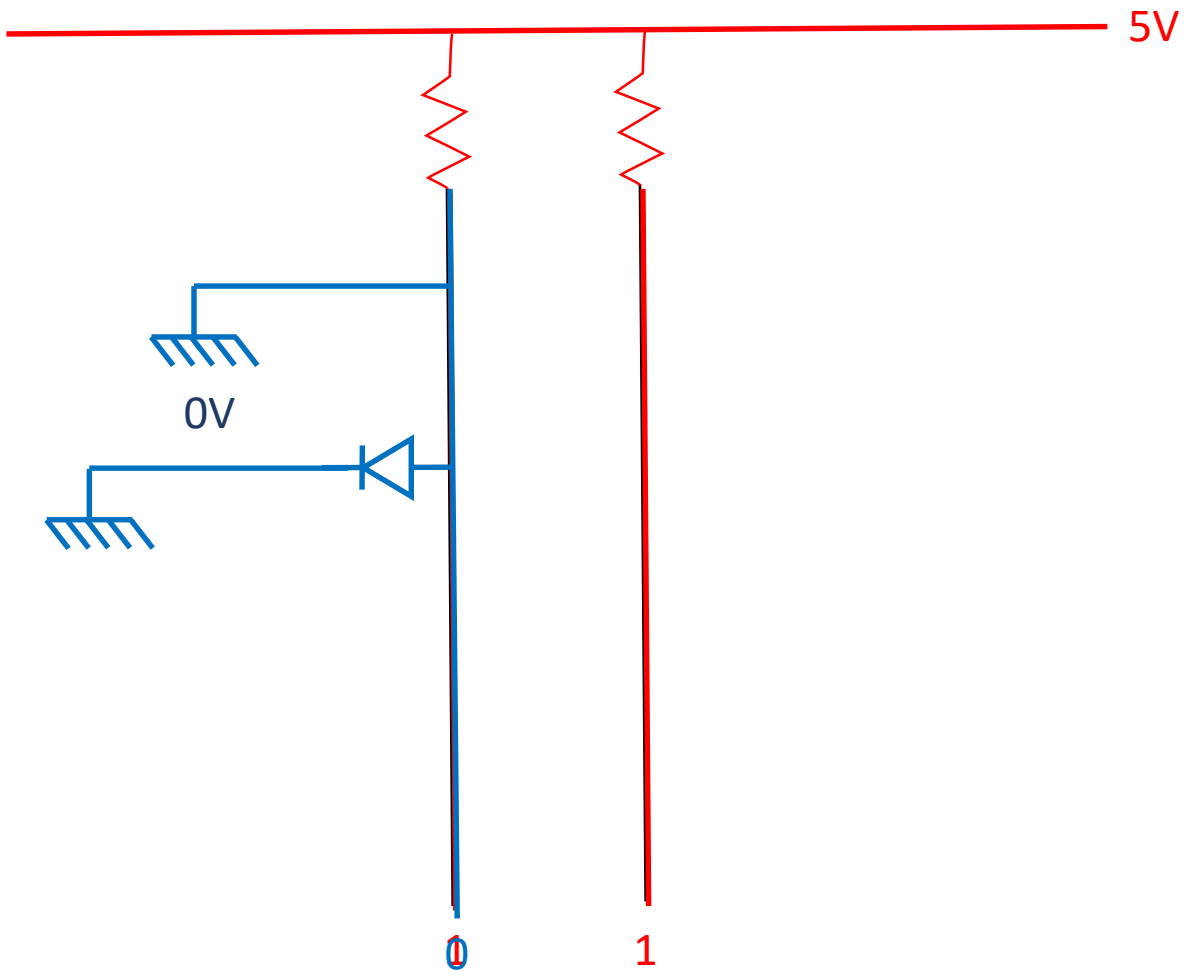
This type of memory is used to store information that is never erased, even when power is removed. It is, however, slower than RAM.

ROM is made up of diodes fitted with small fuses.

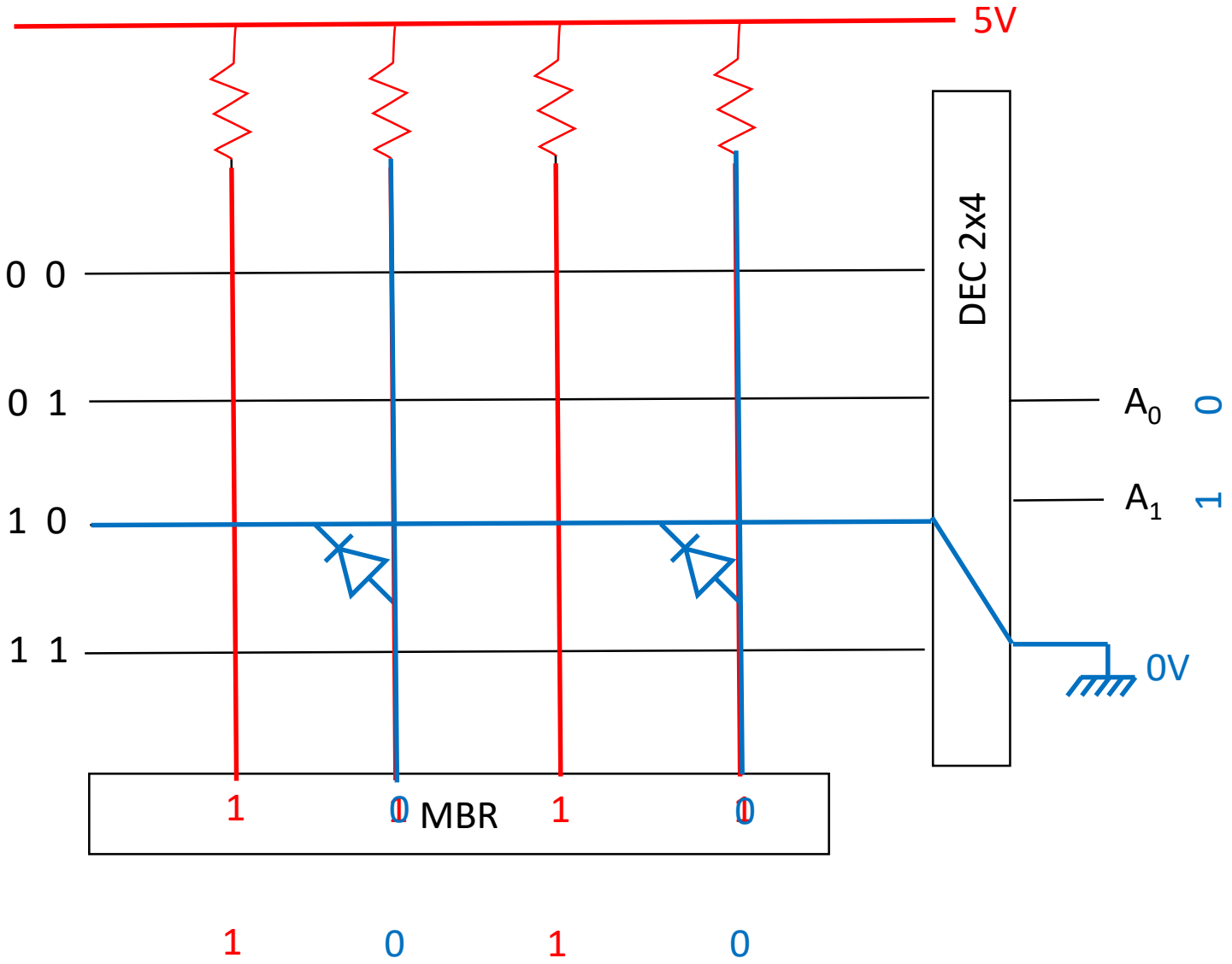
When the fuse is intact, the diode allows current to pass, so it represents 0.

When the fuse does not pass current, we will have 1.

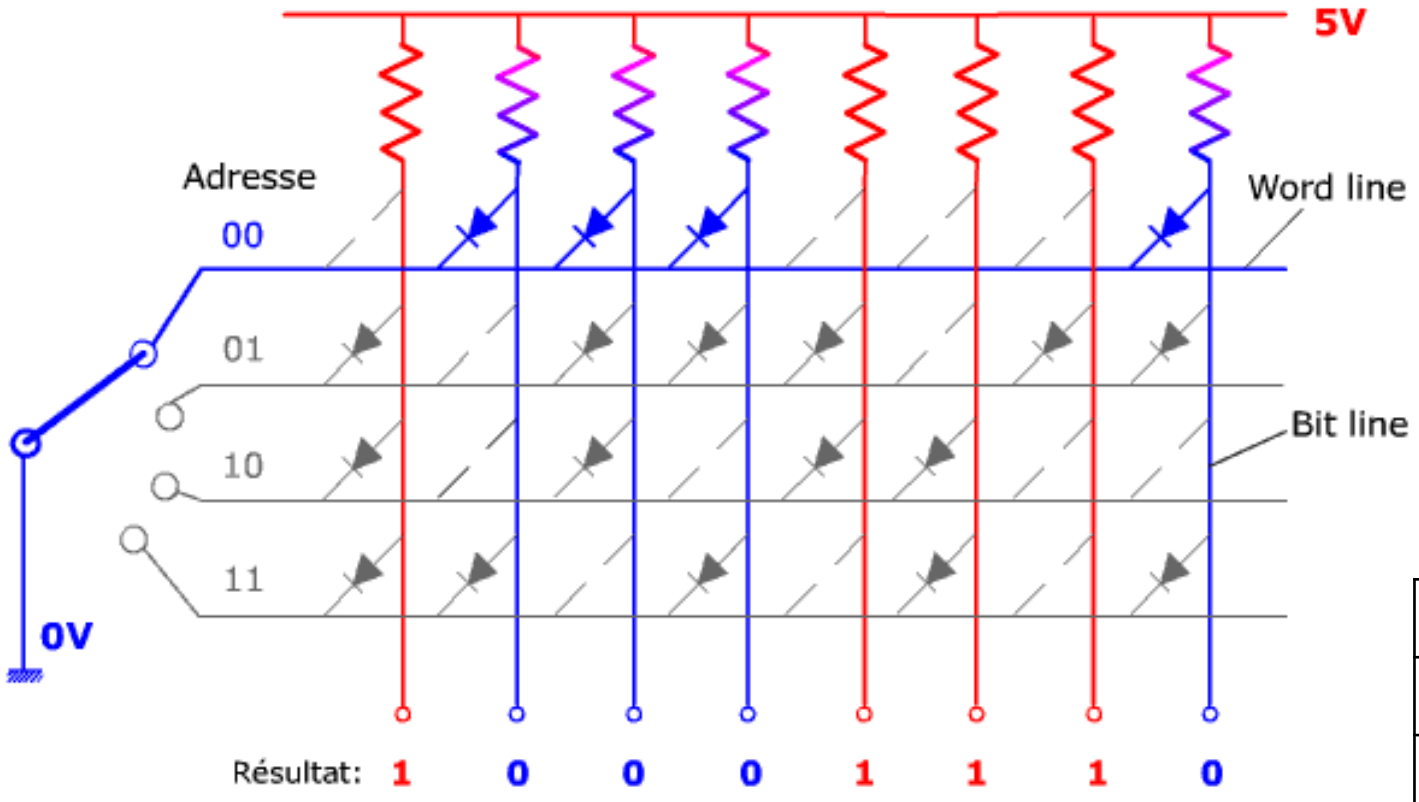
ROM



ROM



ROM



@	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
00								
01								
10								
11								

Memoires

ROM read-only memory

ROM stores the data needed to start up the computer (BIOS).
It can also be used to implement certain combinatorial circuits (logic functions).

There are several types of ROM:
ROMs: their content is defined at the time of manufacture, and cannot be modified.

PROM (Programmable Read Only Memory): programmable only once by the user.

EPROM (Erasable programmable Read Only Memory): can be erased and reprogrammed.
Reset is achieved by reconstituting the fuses after exposure to UV light.

Memoires

ROM read-only memory

This course will focus on the representation of combinatorial functions.

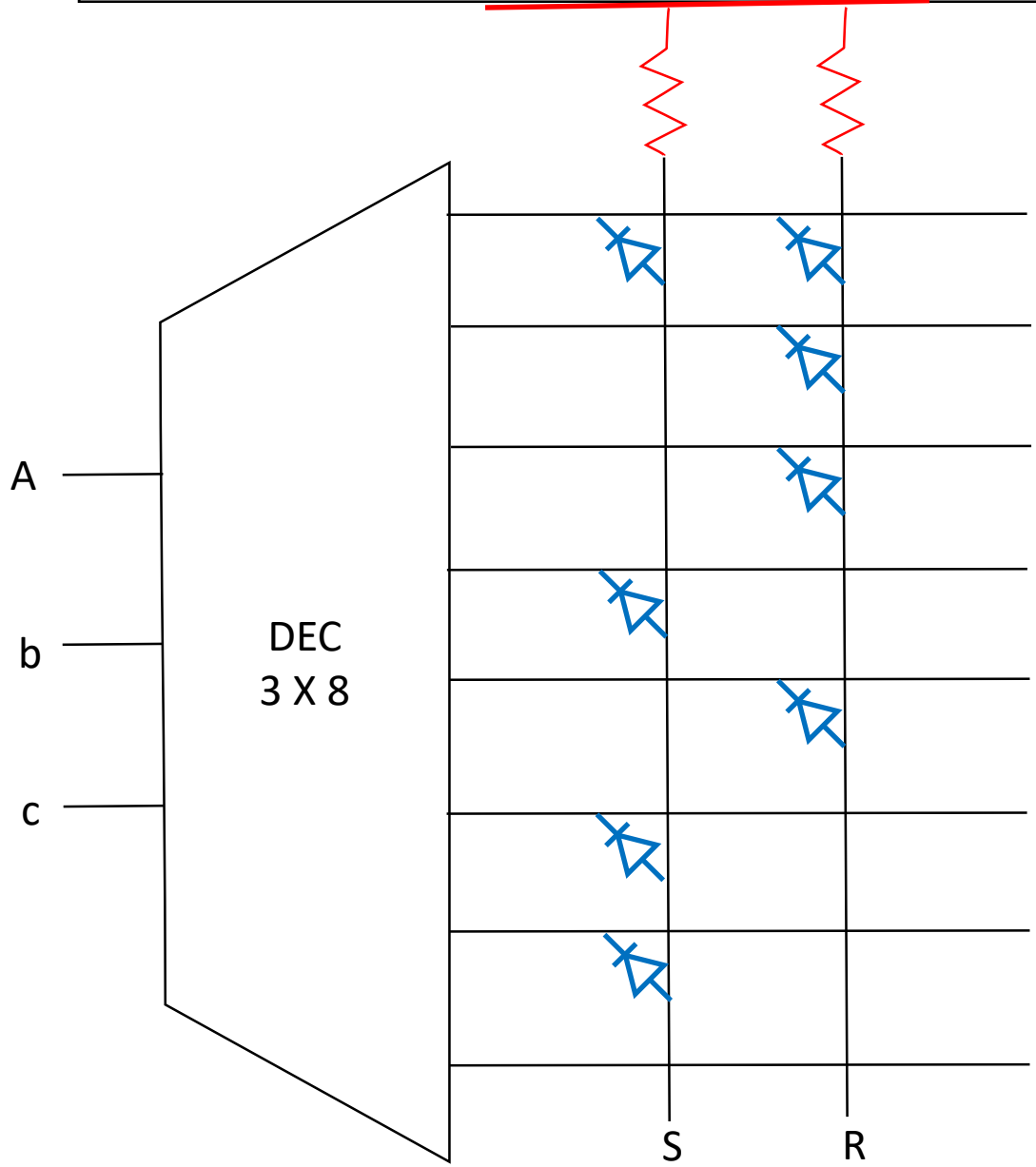
In this case, address bits will be represented by input variables and data lines by output functions.

As an example, we will take a **Full Adder circuit of one bit**.

A	b	c	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Memoires

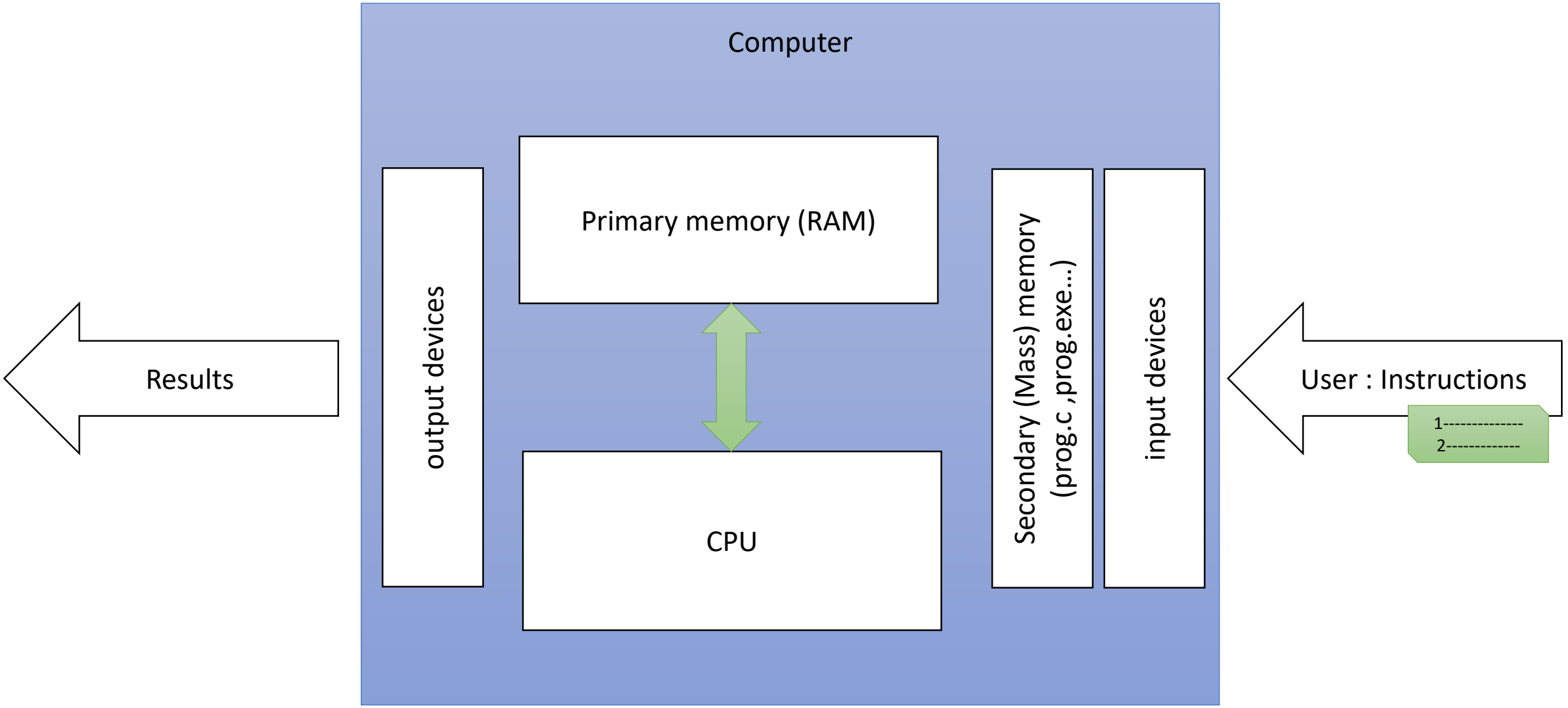
A	b	c	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Pedagogic Machine

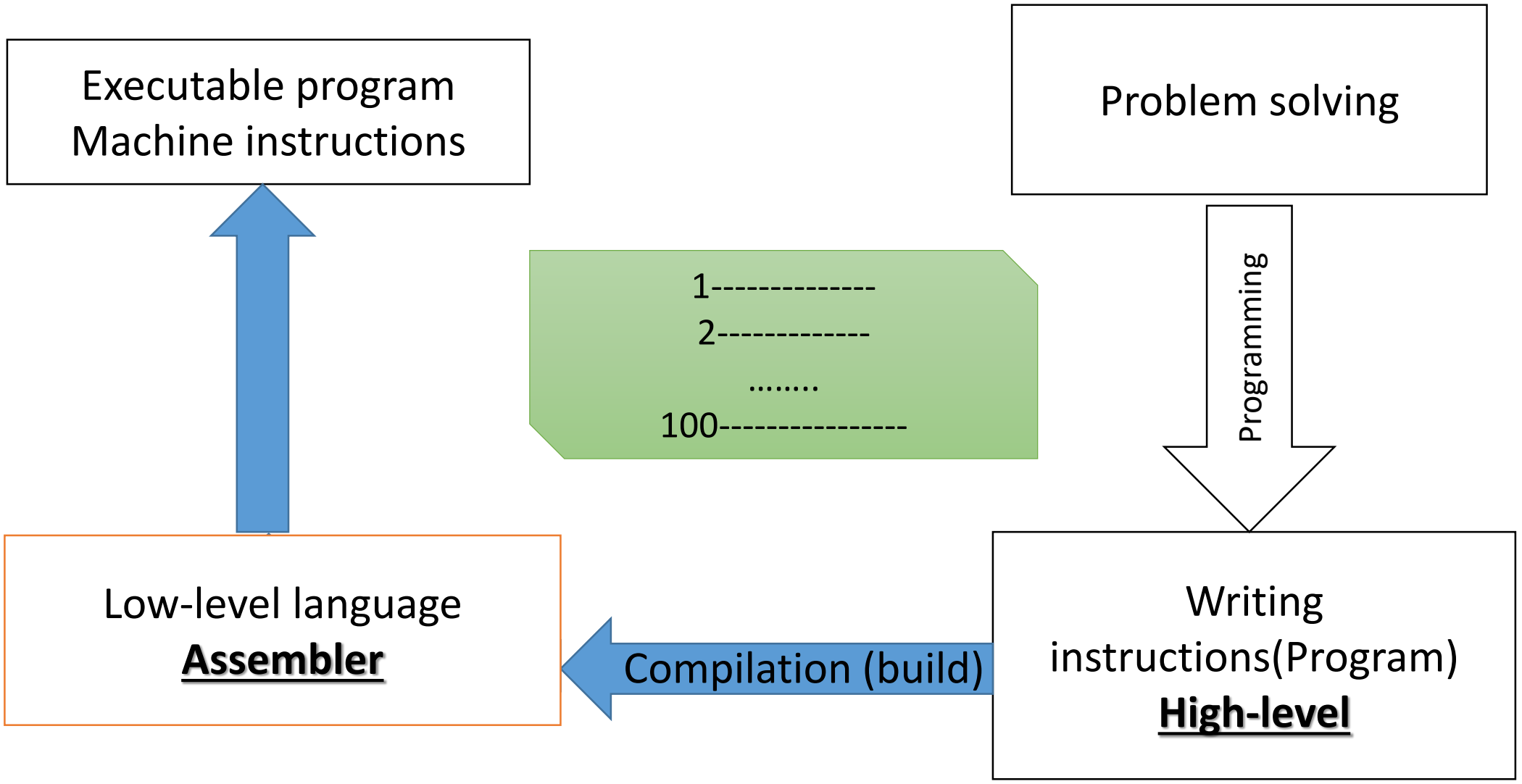
By L.ABADA

Pedagogic Machine



Pedagogic Machine

High-level programme :
a high-level language easily understood by humans

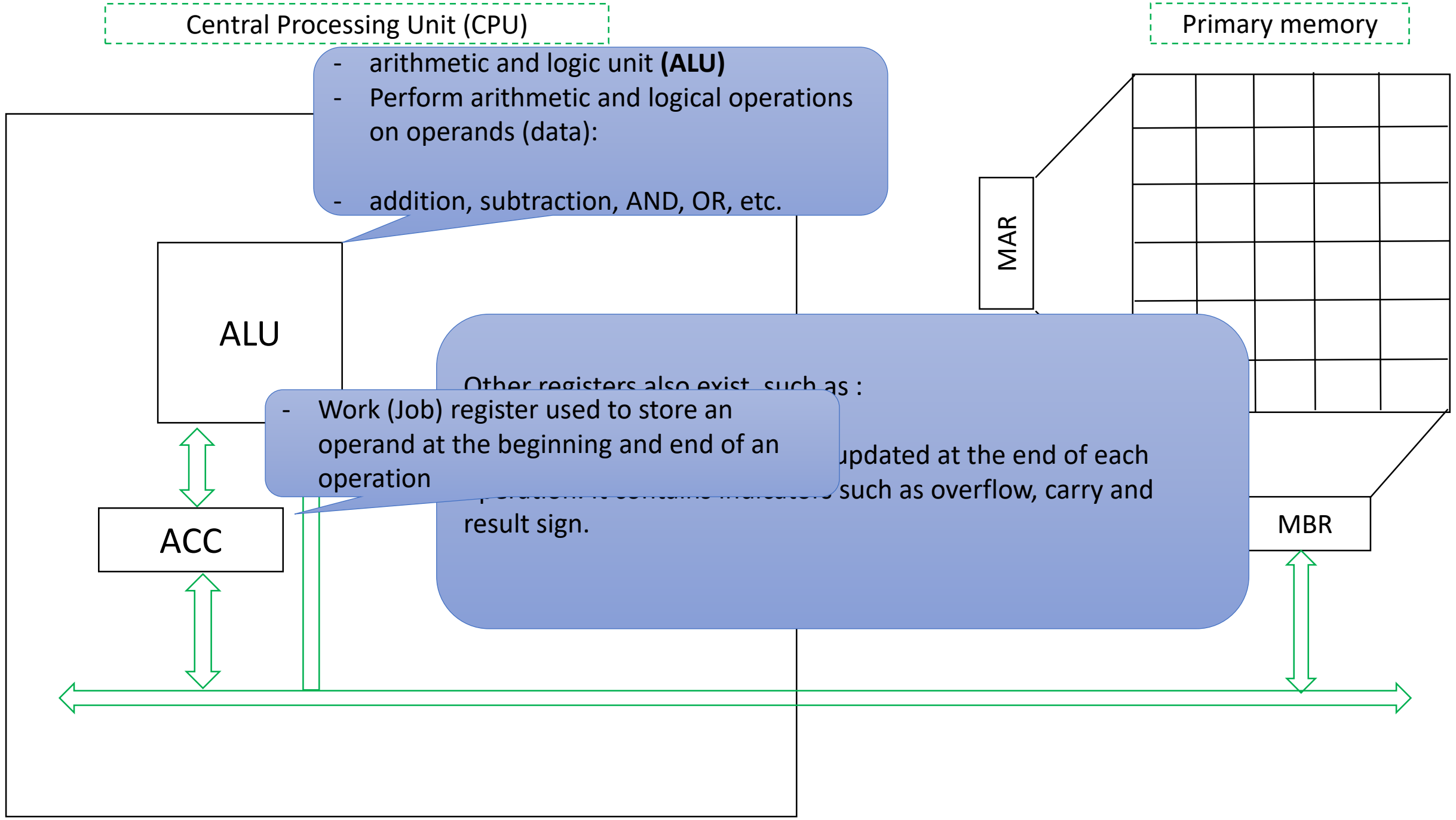


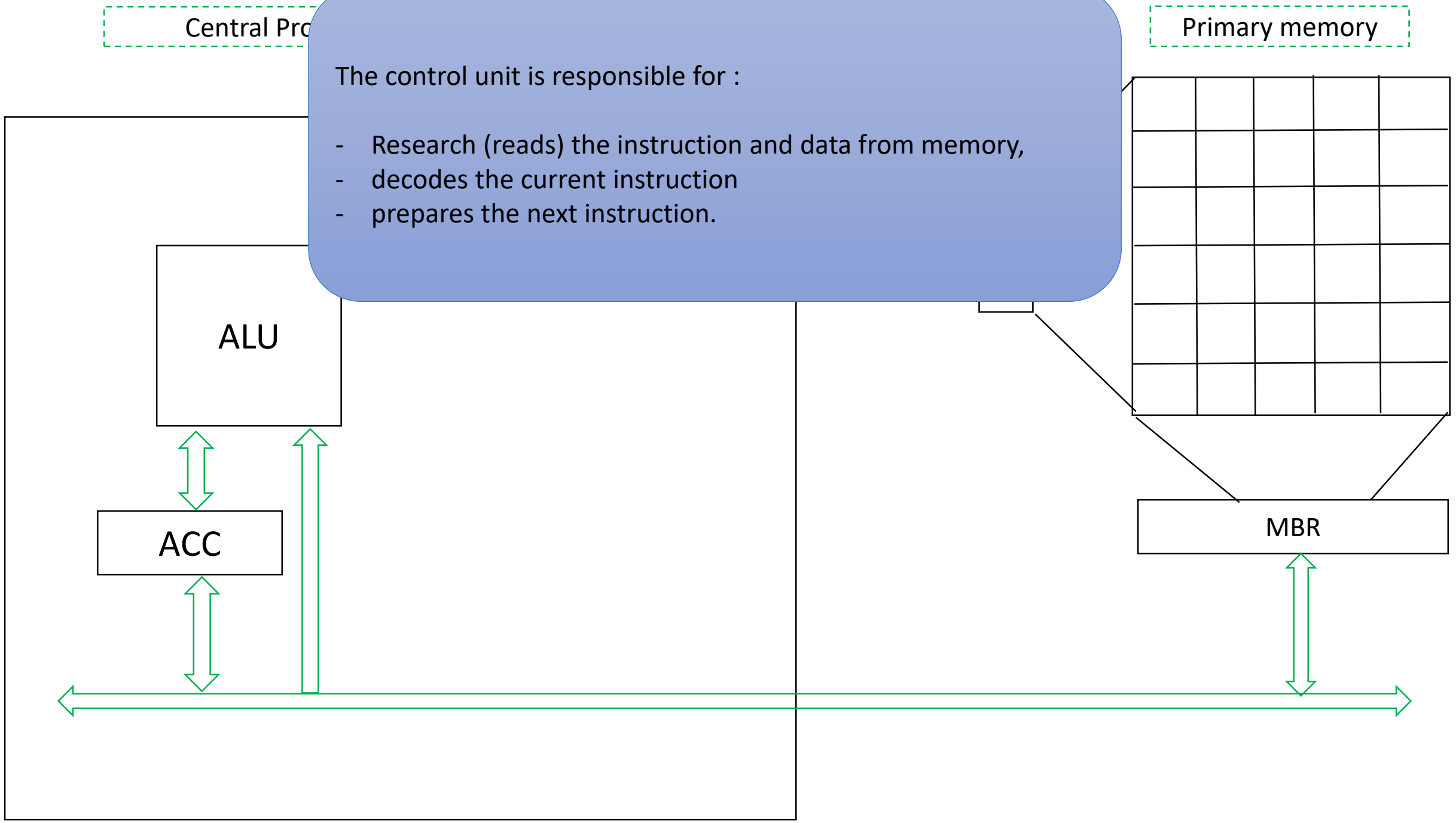
Pedagogic Machine

- assembler is intelligible to humans -
- machine language is purely binary -

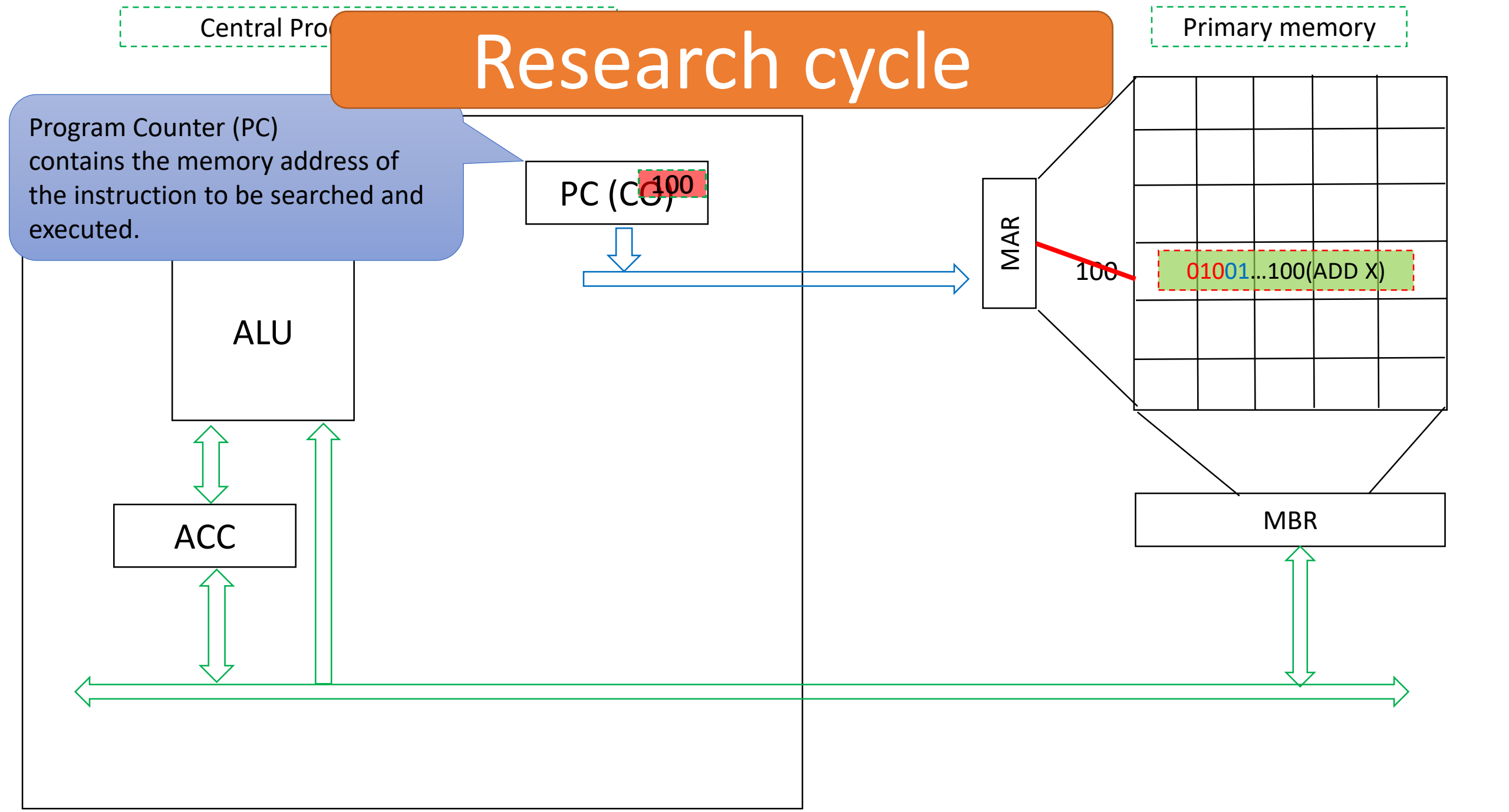
There is a direct correspondence between these 2 languages:

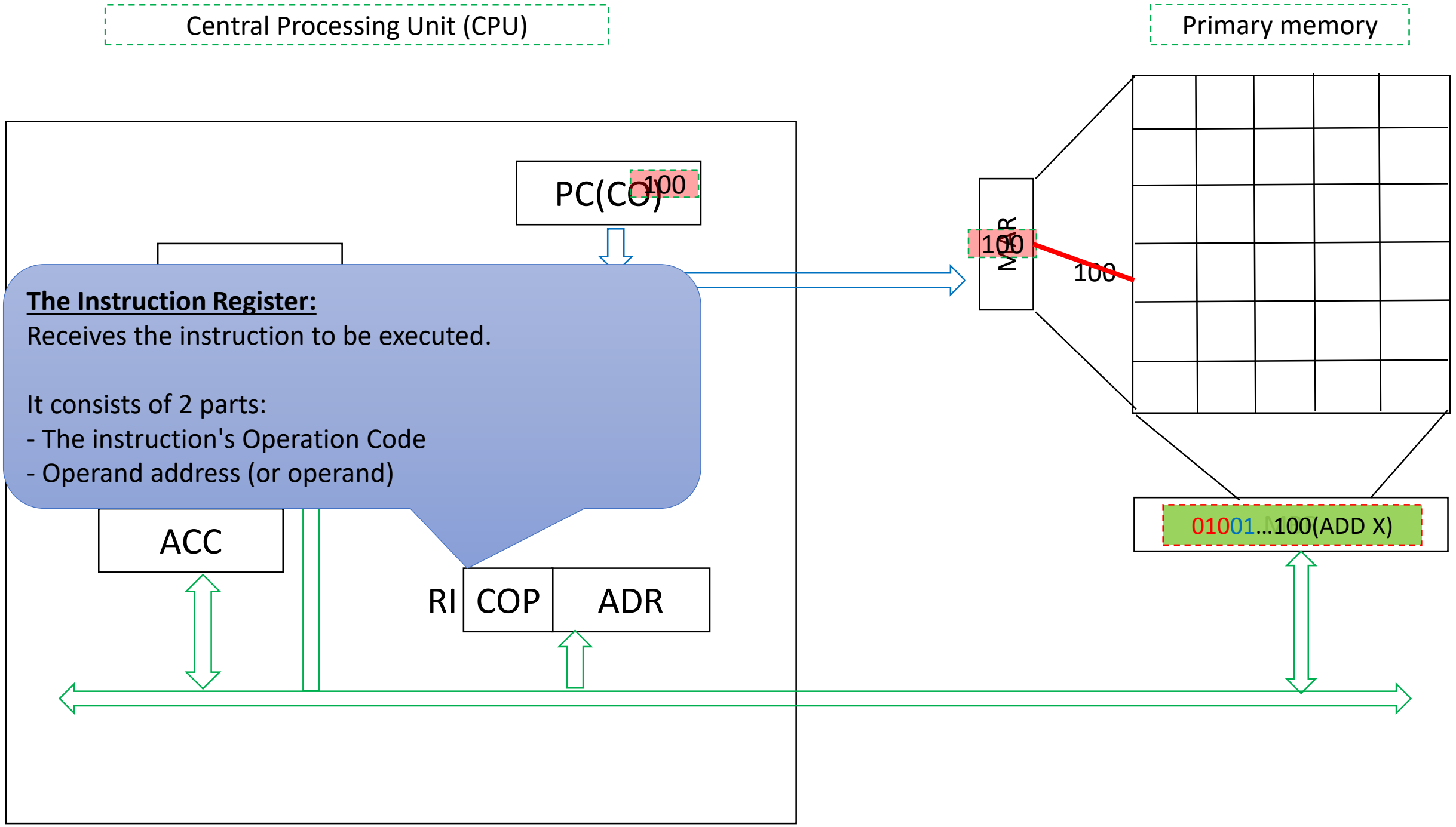
- Each Assembler operation corresponds to a binary code.
- Data and addresses are automatically converted to binary.


Central Processing Unit (CPU)
Primary memory
- arithmetic and logic unit **(ALU)**
- Perform arithmetic and logical operations on operands (data):
- addition, subtraction, AND, OR, etc.
ALU
MAR
- Work (Job) register used to store an operand at the beginning and end of an operation
updated at the end of each
such as overflow, carry and
result sign.
ACC
MBR

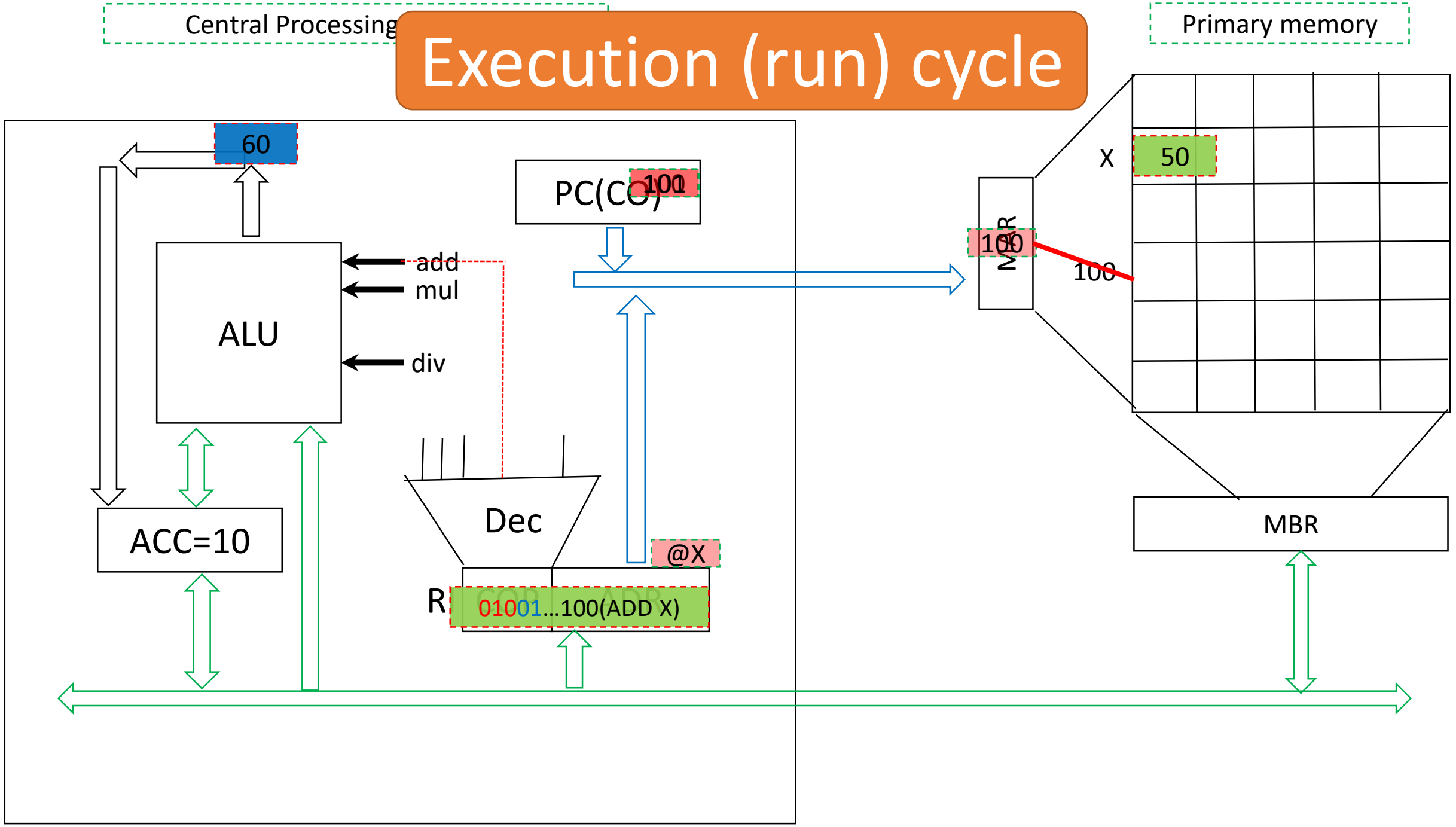

Central Pro
Primary memory
The control unit is responsible for :
- Research (reads) the instruction and data from memory,
- decodes the current instruction
- prepares the next instruction.
ALU
ACC
MBR

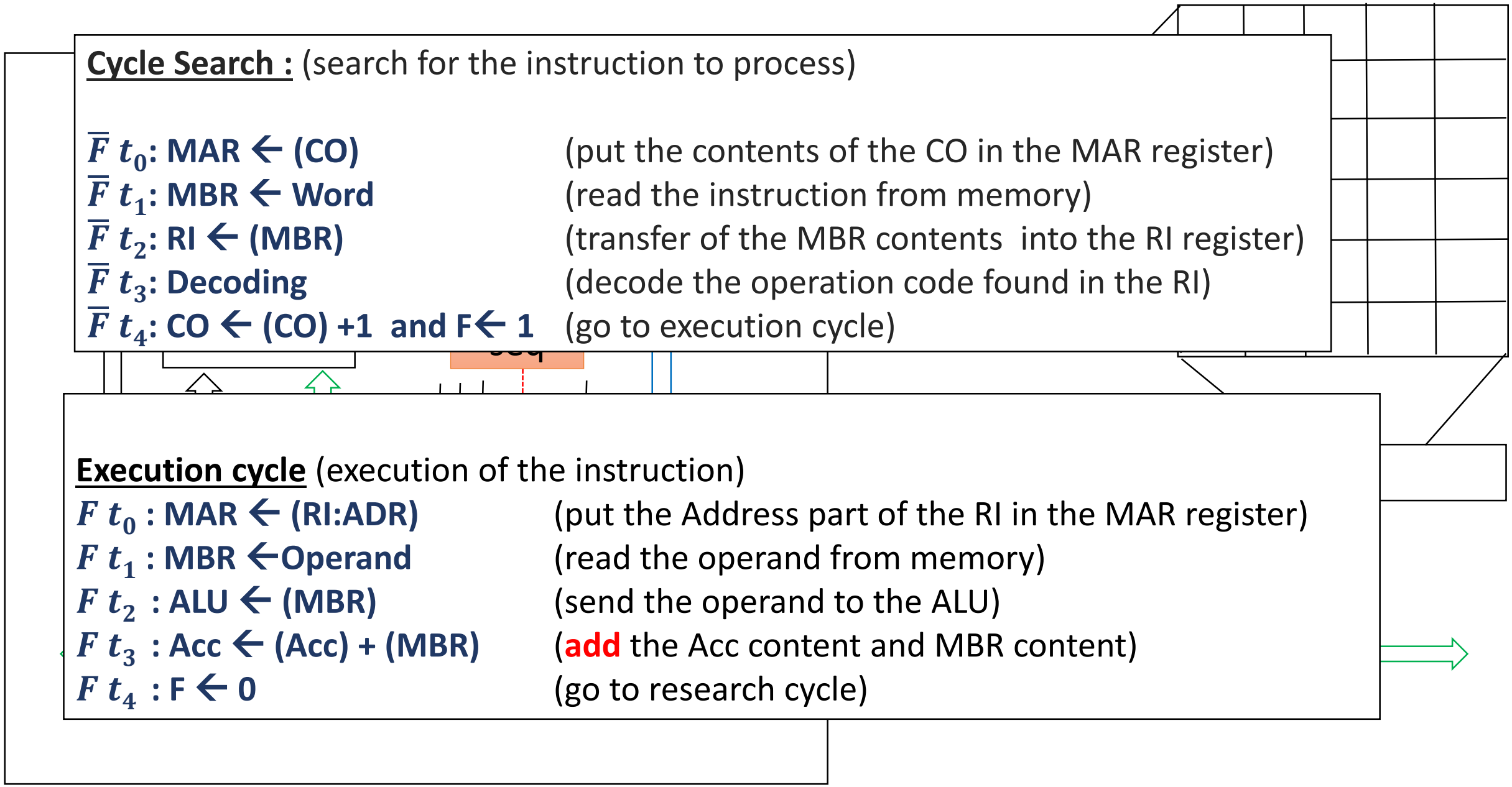
Research cycle

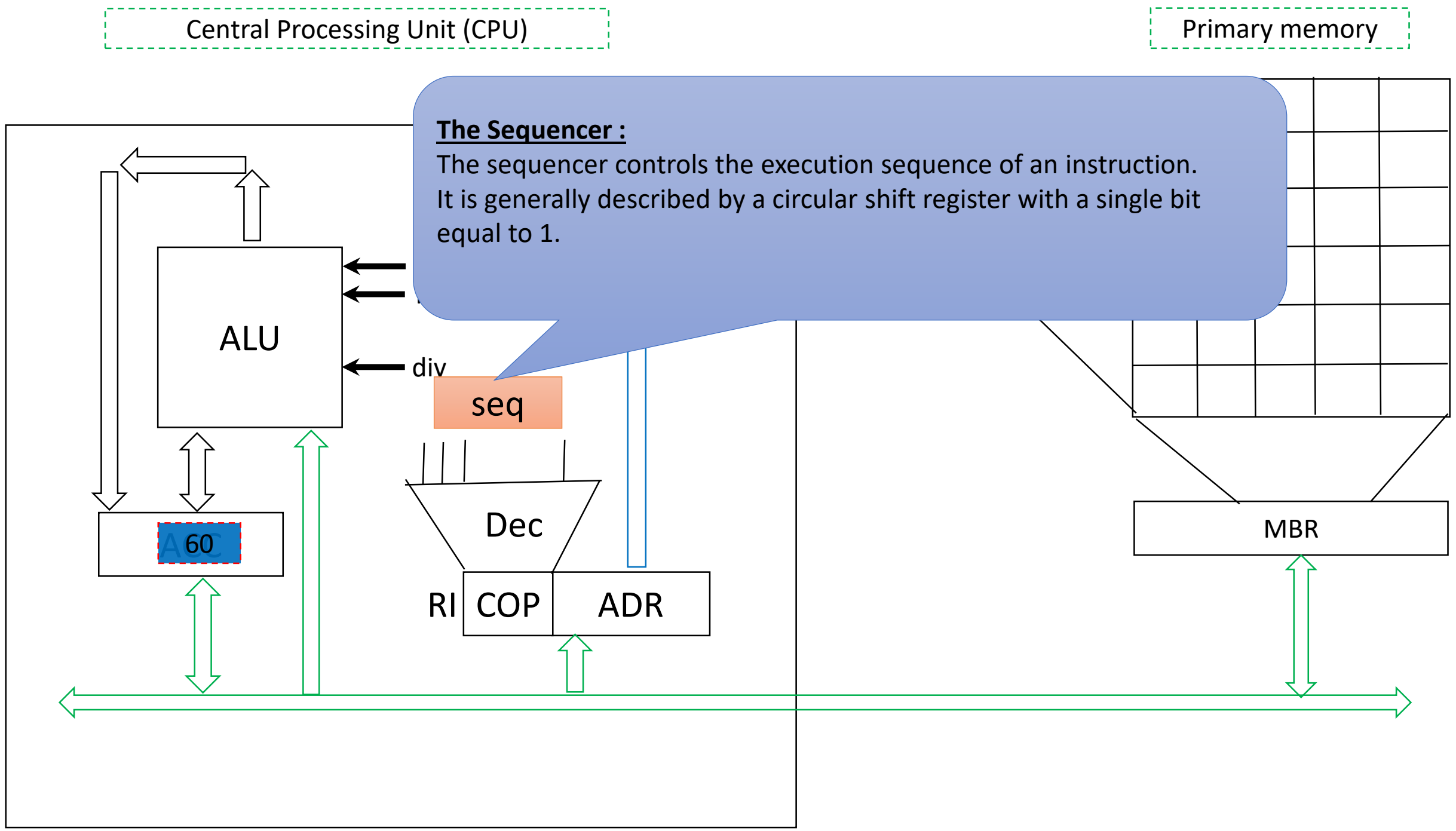


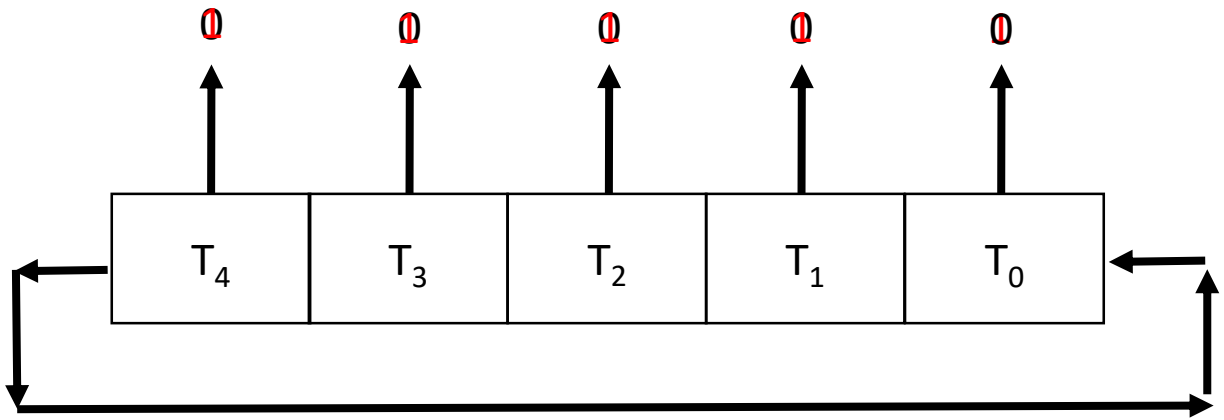

Central Processing Unit (CPU)
Primary memory
PC(CO)
100
The Instruction Register:
Receives the instruction to be executed.
It consists of 2 parts:
- The instruction's Operation Code
- Operand address (or operand)
MAR
100
100
01001…100(ADD X)
ACC
RI
COP
ADR

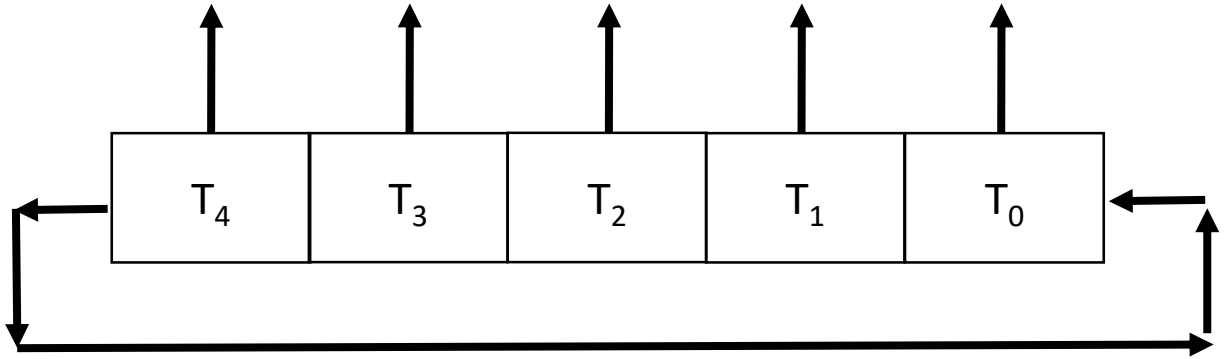
Execution (run) cycle




Central Processing Unit (CPU)
Primary memory
Cycle Search : (search for the instruction to process)
$\overline{F}\ t_0$: MAR ← (CO) (put the contents of the CO in the MAR register)
$\overline{F}\ t_1$: MBR ← Word (read the instruction from memory)
$\overline{F}\ t_2$: RI ← (MBR) (transfer of the MBR contents into the RI register)
$\overline{F}\ t_3$: Decoding (decode the operation code found in the RI)
$\overline{F}\ t_4$: CO ← (CO) +1 and F← 1 (go to execution cycle)
Execution cycle (execution of the instruction)
$F\ t_0$: MAR ← (RI:ADR) (put the Address part of the RI in the MAR register)
$F\ t_1$: MBR ←Operand (read the operand from memory)
$F\ t_2$: ALU ← (MBR) (send the operand to the ALU)
$F\ t_3$: Acc ← (Acc) + (MBR) (add the Acc content and MBR content)
$F\ t_4$: F ← 0 (go to research cycle)

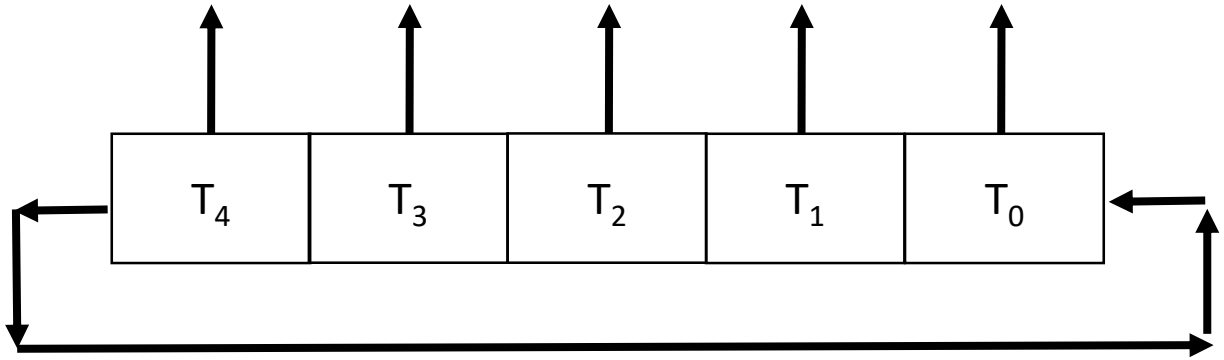

Central Processing Unit (CPU)
Primary memory
The Sequencer :
The sequencer controls the execution sequence of an instruction.
It is generally described by a circular shift register with a single bit equal to 1.
ALU
div
seq
Dec
ACC
60
RI
COP
ADR
MBR


0
0
0
0
0
T4
T3
T2
T1
T0



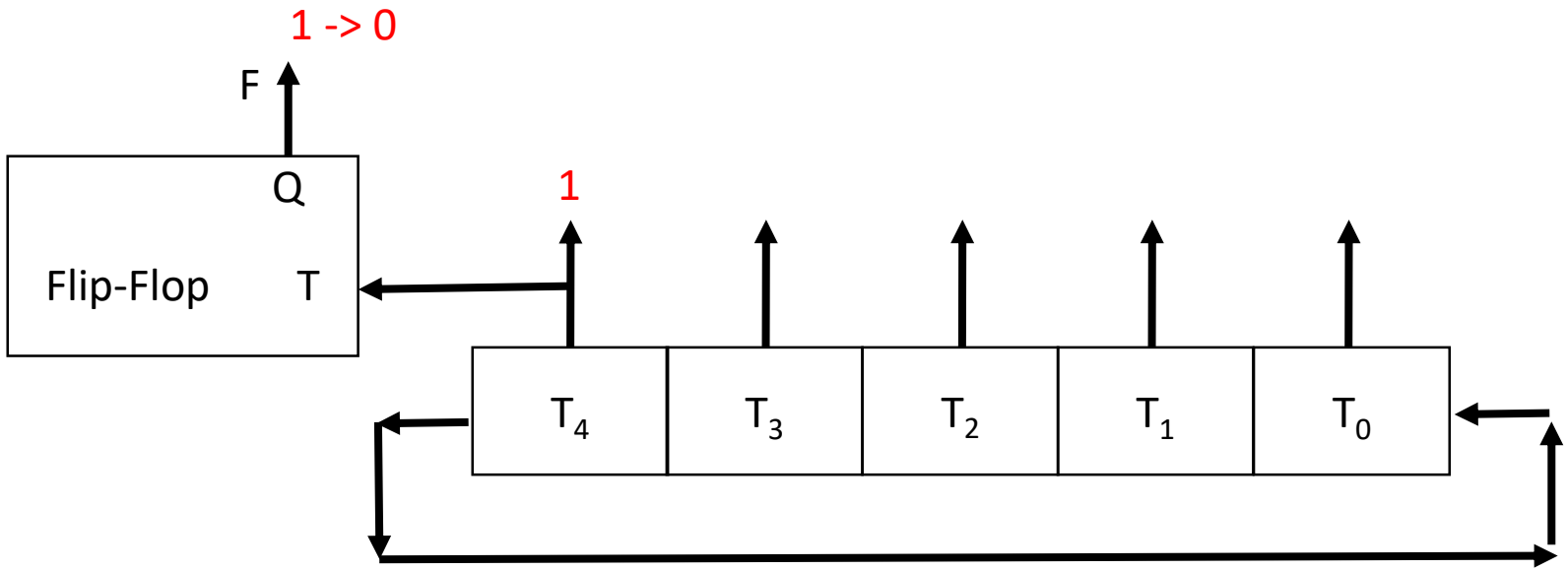
Cycle Search : (search for the instruction to process)

$\overline{F}\ t_0$: MAR ← (CO)	(put the contents of the CO in the MAR register)
$\overline{F}\ t_1$: MBR ← Word	(read the instruction from memory)
$\overline{F}\ t_2$: RI ← (MBR)	(transfer of the MBR contents into the RI register)
$\overline{F}\ t_3$: Decoding	(decode the operation code found in the RI)
$\overline{F}\ t_4$: CO ← (CO) +1 and F← 1	(go to execution cycle)



Execution cycle (execution of the instruction)

$F\ t_0$: MAR ← (RI:ADR)	(put the Address part of the RI in the MAR register)
$F\ t_1$: MBR ←Operand	(read the operand from memory)
$F\ t_2$: ALU ← (MBR)	(send the operand to the ALU)
$F\ t_3$: Acc ← (Acc) + (MBR)	(**add** the Acc content and MBR content)
$F\ t_4$: F ← 0	(go to research cycle)


1 -> 0
F
Q
Flip-Flop
T
1
T4
T3
T2
T1
T0

Instructions

Each microprocessor has a **limited** number of instructions that can be executed.

These instructions are called **Instruction Sets**.

Instructions can be classified into **4 categories:**

1. **Assignment instruction**: transfers data between registers and memory
2. **Arithmetic and Logic** instructions (AND, OR, ADD, ...)
3. **Branch instructions** (conditional and unconditional)
4. **Input/Output** instructions.

Instruction encoding :

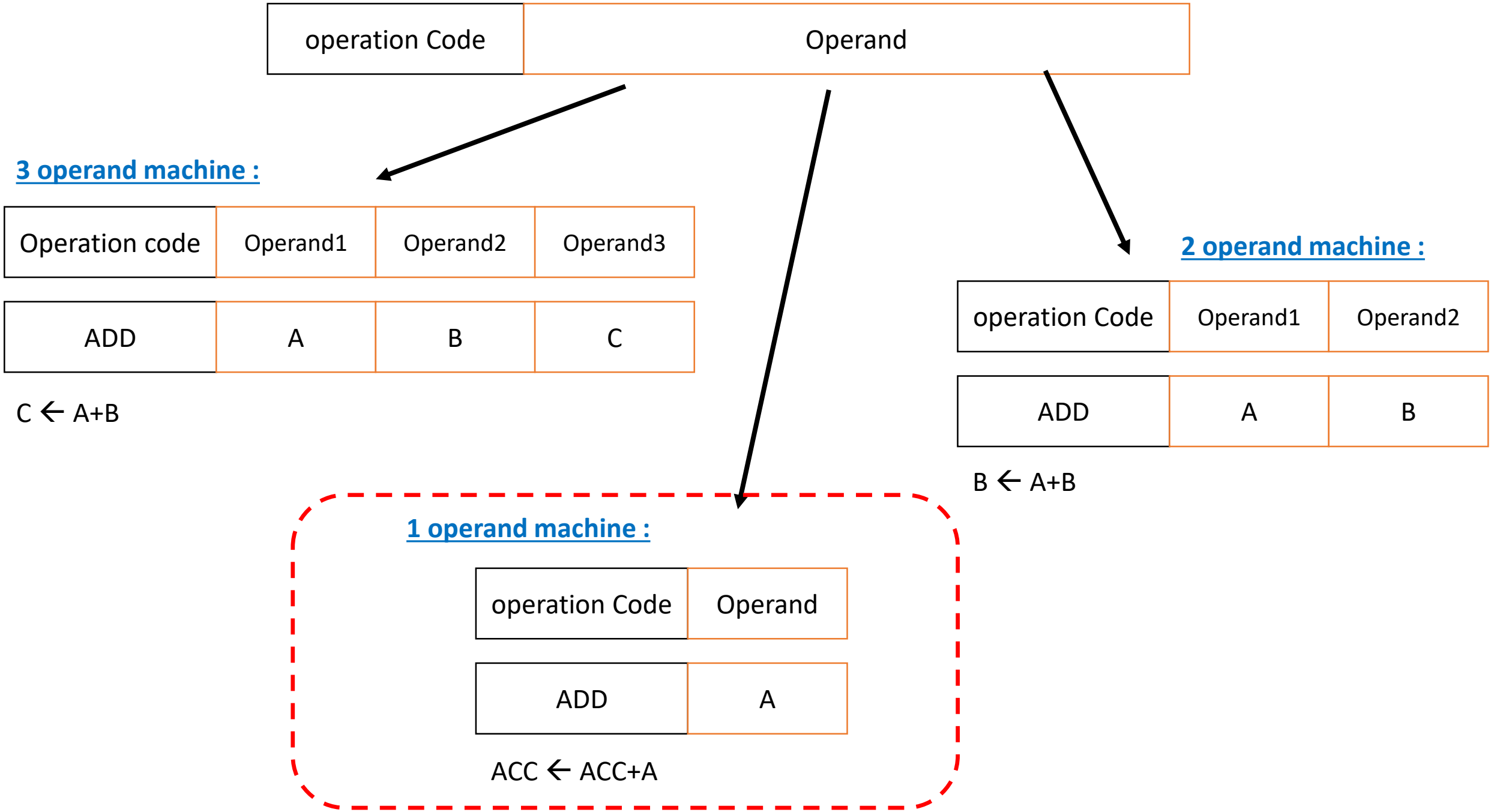
Instructions are stored in memory.

The size of an instruction is equal to the size of a memory word*

The instruction is divided into two parts:

- **Operation code:** indicates which instruction to execute.
- **Operand:** contains the data or address of the data concerned by the operation to be executed.

** only in our course (Pedagogic Machine)*

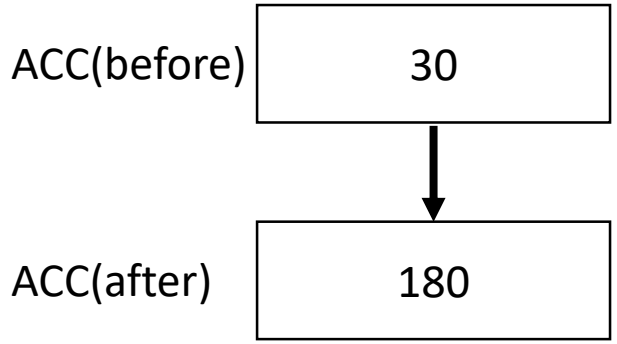

operation Code
Operand
3 operand machine :
Operation code
Operand1
Operand2
Operand3
ADD
A
B
C
C ← A+B
2 operand machine :
operation Code
Operand1
Operand2
ADD
A
B
B ← A+B
1 operand machine :
operation Code
Operand
ADD
A
ACC ← ACC+A

Addressing mode

1- Immediate addressing

operation Code	Operand
ADD	150,IMM

ACC ← (ACC) + 150

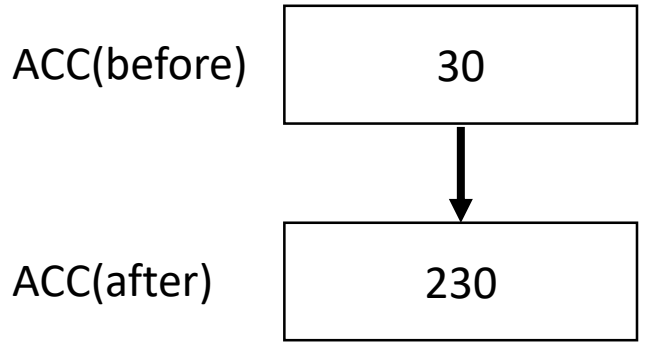


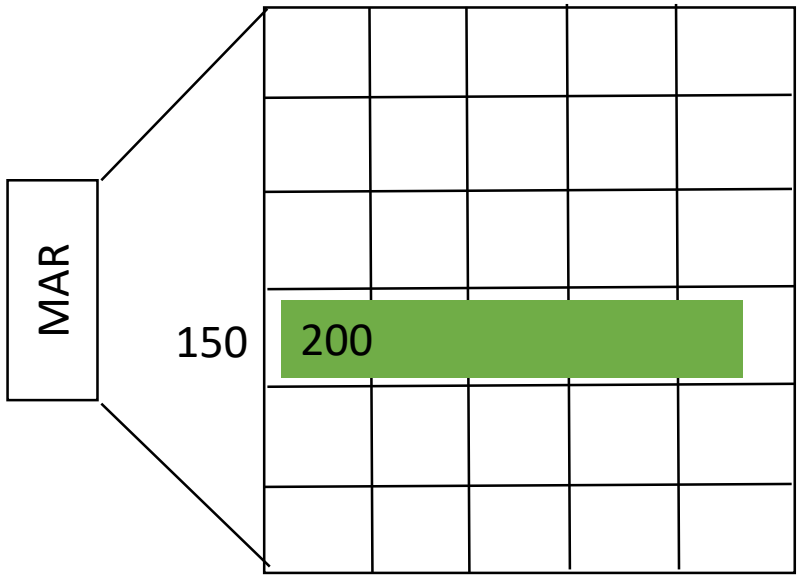
Addressing mode

2- Direct addressing

operation Code	Operand
ADD	150,D

ACC ← (ACC) + (150)





Addressing mode

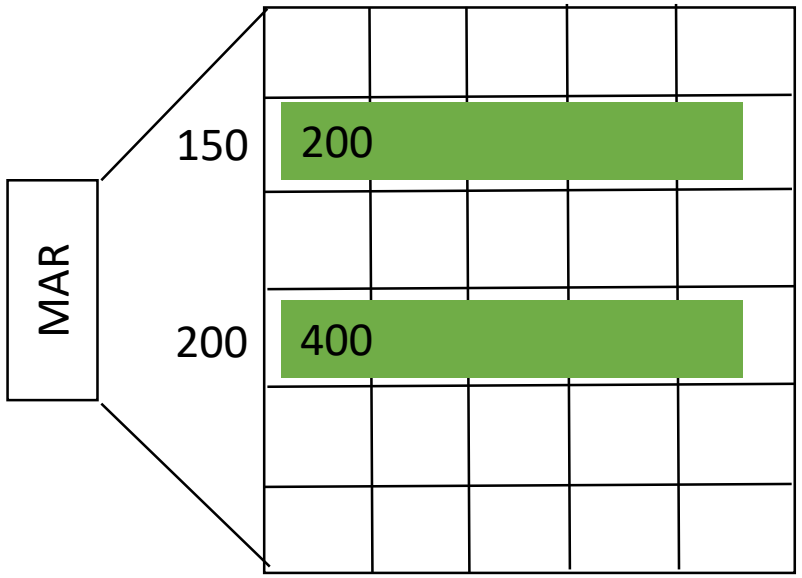
3- Indirect addressing

operation Code	Operand
ADD	150,IND

ACC ← (ACC) + ((150))

ACC(before) 30

ACC(after) 430



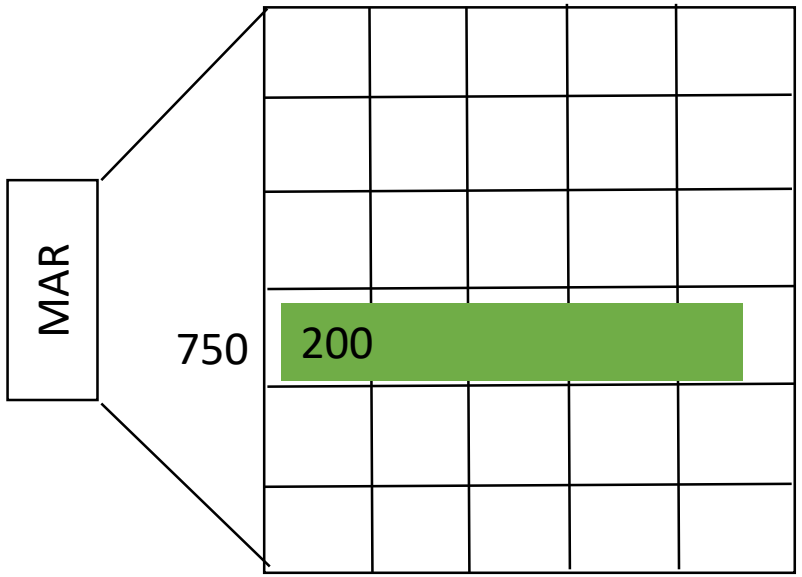
Addressing mode

4- Indexed addressing

operation Code	Operand
ADD	150,XRI

ACC ← (ACC) + (150+(XRI))

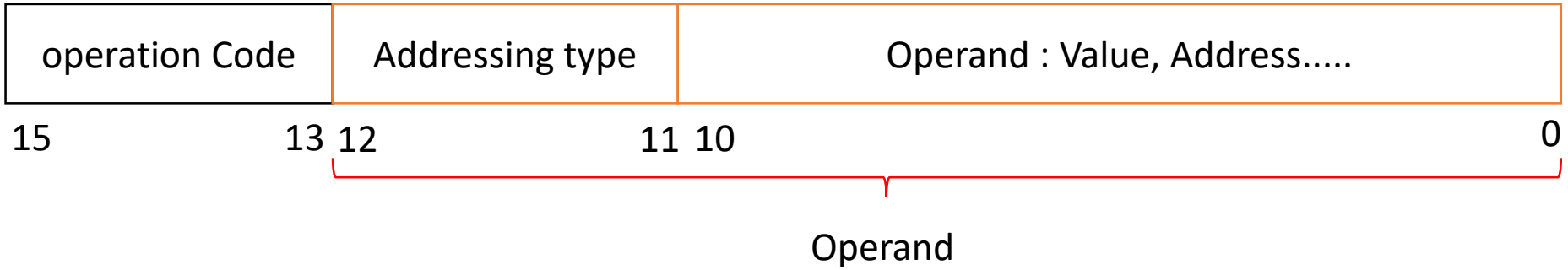
XRI(before)	600
ACC(before)	30
ACC(after)	230



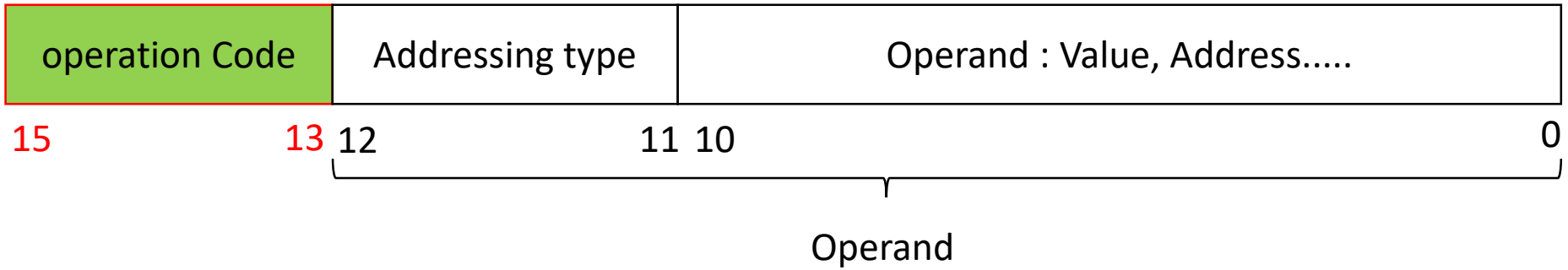
Instruction format

The memory of the machine studied in this course is represented by a 2K x 16 RAM. (RAM with 2^{11} 16-bit words)

Each instruction is encoded in binary on 16 bits according to the following format :



Instruction format

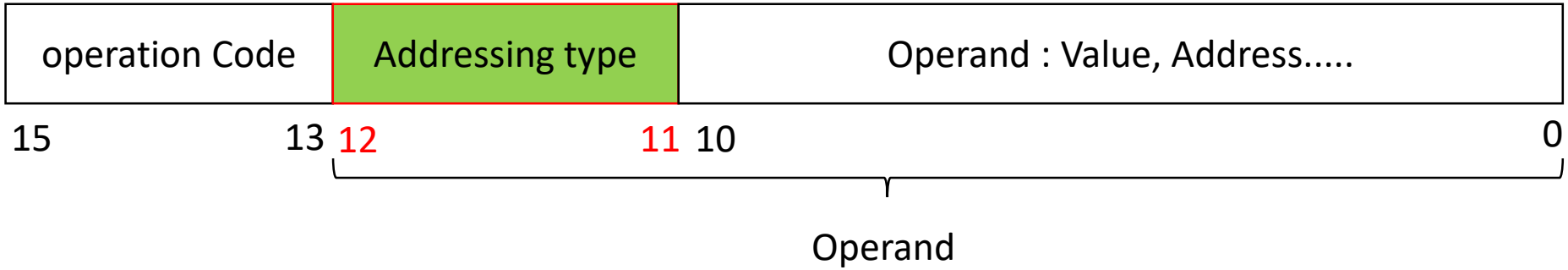


The operation code field is on 3 bits (from 13 to 15).

A maximum of 8 operations can therefore be used on this machine:

000 : LOAD	100 : MUL
001 : STORE	101 : DIV
010 : ADD	110 : READ
011 : SUB	111 : WRITE

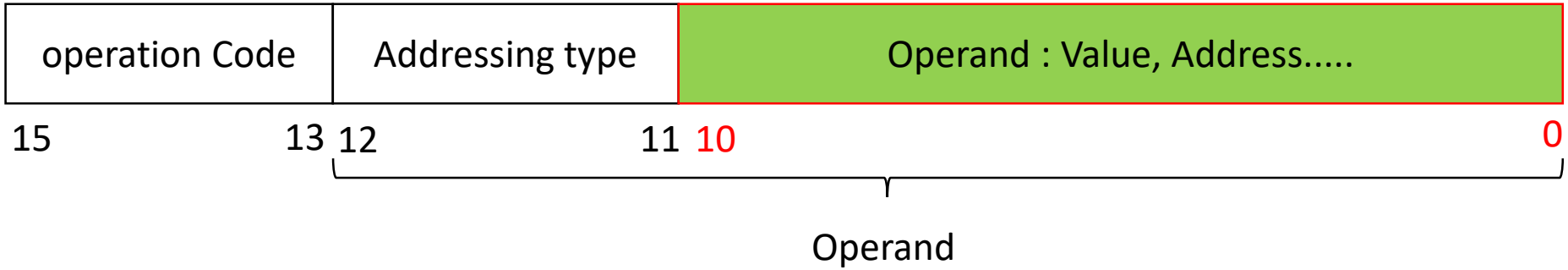
Format d'une instruction



The addressing field is on 2 bits (10, 11).

We use :
00 for immediate addressing ,
01 for direct addressing
10 for indirect addressing
11 for indexed addressing

Format d'une instruction



The operand field is 11 bits long (0 to 10).

It generally contains an address (binary) or sometimes a data item for immediate addressing.

The MBR of this RAM will consist of 11 bits to access 2^{11} addresses.
The MBR of this RAM will consist of 16 bits.

A program is a sequence of instructions.

Format d'une instruction

Instructions	Signification	Code Opération
LOAD X, D	Acc ← (X)	0 0 0
STORE X, D	Mémoire ← (Acc)	0 0 1
ADD X, D	Acc ← (Acc) + (X)	0 1 0
SUB X, D	Acc ← (Acc) - (X)	0 1 1
MUL X, D	Acc ← (Acc) x (X)	1 0 0
DIV X, D	Acc ← (Acc) / (X)	1 0 1
READ	Acc ← val du périphérique d'entrée	1 1 0
WRITE	périphérique de sortie ← (Acc)	1 1 1

Format d'une instruction

ADD val, IMM	Additionner val avec le contenu de l'Acc	**Acc←(Acc)+val**
ADD xxx, D	Additionner le contenu de l'@ xxx au contenu de l'Acc	**Acc←(Acc)+(xxx)**
ADD xxx, IND	Additionner le contenu du contenu de xxx au contenu de l'Acc	**Acc←(Acc)+((xxx))**
ADD xxx, XRI	Additionner le contenu de (xxx+le contenu deXRI) au contenu de l'Acc	**Acc←(Acc)+(xxx+(XRI))**