Process Termination

1- How is the process terminated?
2- What happens after termination?

The process is **terminated** in one of the following scenarios:

- ↳ **executes** all its instructions (**return**) } normal termination
- ↳ **exits** based on a given situation } normal termination
- ↳ **killed** by another **priviliged** process } abnormal termination

Normal Termination

<sys/wait.h>
int exit (int **status**); → return ≡ exit(0).

↳ used to indicate wether the program executed successfully or if there were any errors

Status: a number (1 byte) (0 → 255) used by the process to tell its parent about its exit(status).

N.B: each process running in the system has a PCB. when the process exits, it will stay on the system until his parent gets the info sent by the exit (status)

How the **parent** gets **info** about the **death of the child**?

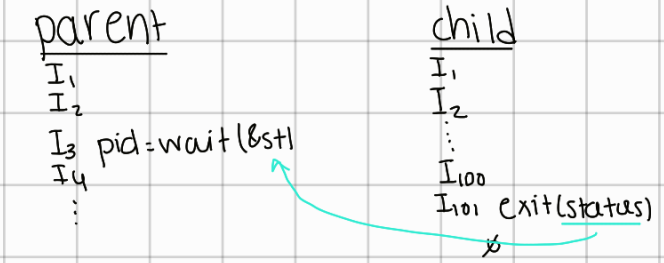
↳ <sys/wait.h>

```
int wait(int *st_ptr);
pid = wait(&str_ptr);
```

Wait Statement:

the wait statement is a **blocking statement**, this means that the parent will **wait** for the **child to exit( )**.

little demo:

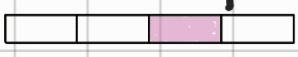


↳ the wait function returns the pid of the exited child.

↳ the wait function is used for the first child that exits.

↳ if the parent has no children, wait returns -1

* **while(wait(&str_pt)!=-1);**
  this means that the parent will wait for all the children to exit.
* the parameter of the wait function is a pointer to int used for receiving the value sent by the child using the function exit(status).
* **wait(null);**
  ignores the return value sent by the child

Note: the returned status is put in the second byte of the integer  so the returned number is stored in ST as a different int, to solve this problem, we use:
ST = ST << 8 or **ST = WEXITSTATUS(ST)** ©
(these are used to shift 8 bits to the right).

wait(&st)

exit & wait: using exit and wait, parent and its child can make some **naïve communication** _ some kind of **synchronization** ©

Example:

array of integers (0 and 255)

int tab[10] = {12, 5, 3, 25, 16, 70, 112, 200, 240, 254}

* determine the max of this vector

parent: 0 → 4 ; child: 5 → 9

display the max on screen.

```
* #include <unistd.h>
  #include <stdlib.h>
  #include <sys/wait.h>

  int main(){
  int tab[10]={12,5,3,25,16,70,112,200,240,254}, j;
  if (pid=fork()){
    int max =0;
    for(int i=0; i<5; i++){
       if(A[i]>max)
          max= A[i];
    } wait (&j);   // j is max1.
     j = j << 8;
    if(max>j)
       printf("%d", max);
    else
       printf("%d", j);
  } else{
    int max1=0;
    for(int i=5; i<10; i++){
       if (A[i]>max1)
          max1=A[i];} exit(max1);
  }
```

int sleep(int nsec)

the process calling this function suspends its execution for nsec.

Zombie Process:

↳ a child process that has completed its execution but still has an entry in the process table (process table entry contains process ID, exit status, etc).

↳ this occurs when the parent process hasn't yet collected the exit status of its child process. The operating system retains the zombie process entry in the process table until the 'wait' system call or similar mech.

↳ once the parent process collects the exit status of the child process to acknowledge their termination.

Orphan Process:

↳ a child process whose parent process has terminated or been killed before the child process has completed its execution.

↳ orphan processes are typically adopted by the init process (process with pid=1). the init process becomes the new parent of the orphaned child process.

↳ orphan processes continue their execution, and when they finish, they become zombie processes until the init process collects their exit status.

↳ orphan processes are usually the result of a parent process creating a child process and then exiting uexpectedly before the child has completed its tasks.

Ex:

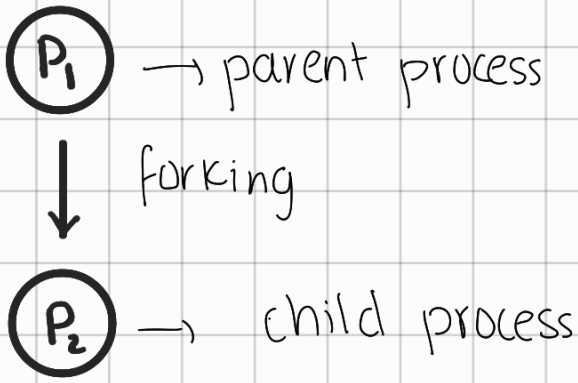
```
void main() {
   int pid;
   printf("I'm the main process with pid=%d\n",
          getpid());
   pid= fork();
   if (!pid){
      printf("I'm the child process with pid=
             %d\n", getpid());
      exit(2);
   }
   else {
      sleep(100);
      wait(null);
   }
}
```

Output

```
child : pid=100
parent: pid=90
```

lecture 3: Process Creation

How the process is created? Unix.



the **main process** in the unix is **Init → pid=1**

- Initially when the **system boots**, a process **(init) with pid = 1 is started.**

Je n'appelle Jiled

- In unix, the **process are created by cloning/ duplication** => one process duplicated

P_1 becomes parent.
P_2 child

- int fork();

```
#include <unistd.h>
int fork();
```

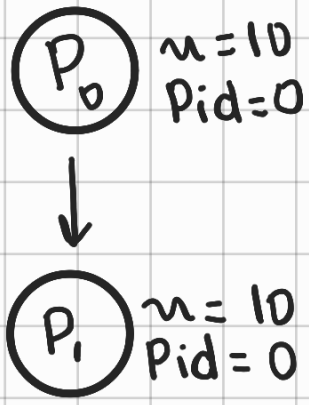

```
int fork()
```

- -1 → failure
- 0 → (child)
- <> 0 → (parent) pid of child.

Ex:

```
void main(){
  int x, pid;
  x = 10;
  if (pid = fork())
    x = x+5;
  printf("x = %d /n", x);
}
```

parent



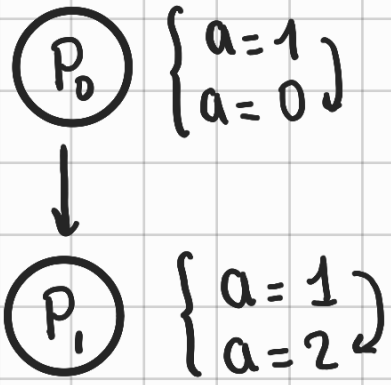

```
if (Pid = fork())
  ① pid = fork();
  ② if (pid) = if(pid!=0)
```

```
void main(){
  int a = 1;
  if(!fork()){
    a++;
    printf("%d\n", a);
  } else
    a++;
  printf("%d\n", a);
}
```

child.

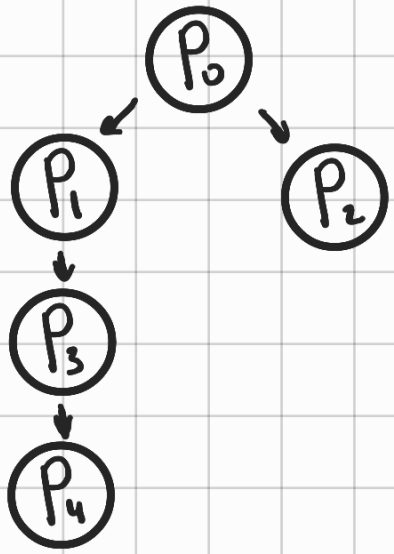
output:		
2	0	2
0	2	2
2	2	0

```
if (!fork())
  → pid = fork()
  → if (pid == 0)
```

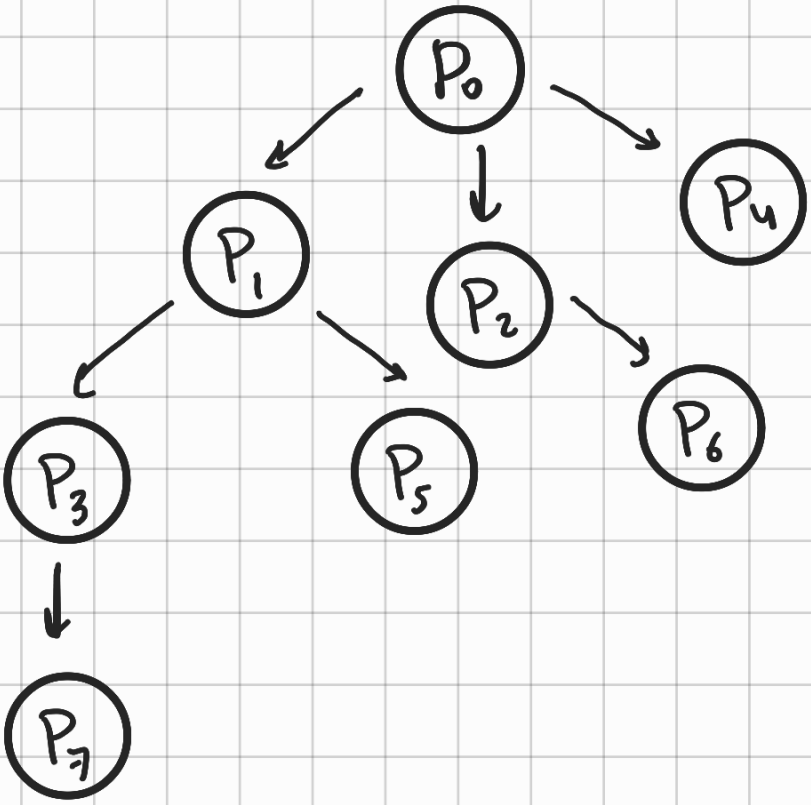


Ex:
```
void main(){
    if(fork())
        fork();
    else if (!fork())
        fork();
}
```

if (fork() => Pid = fork() => P_1
if (Pid <> 0) => P_0.



Ex:
```
void main(){
    fork();
    fork();
    fork();
}
```



Process Identification

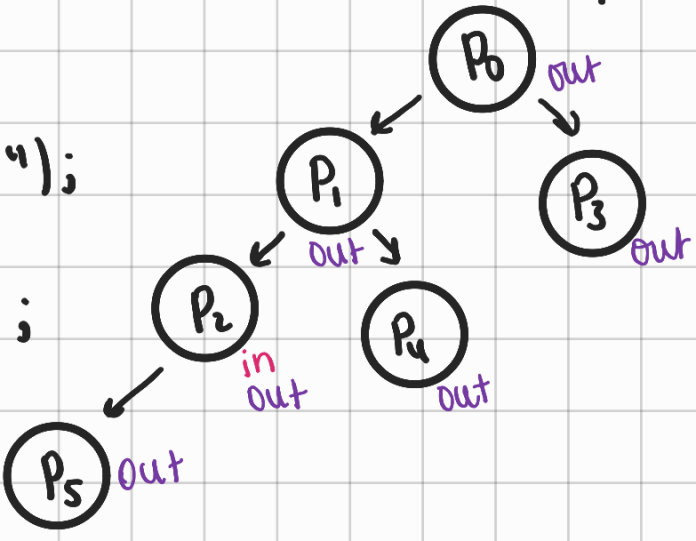
- int getpid()
  returns the pid of the calling process.

- int get ppid()
  returns the pid of the parent of the calling proces.

Extra Exercises:

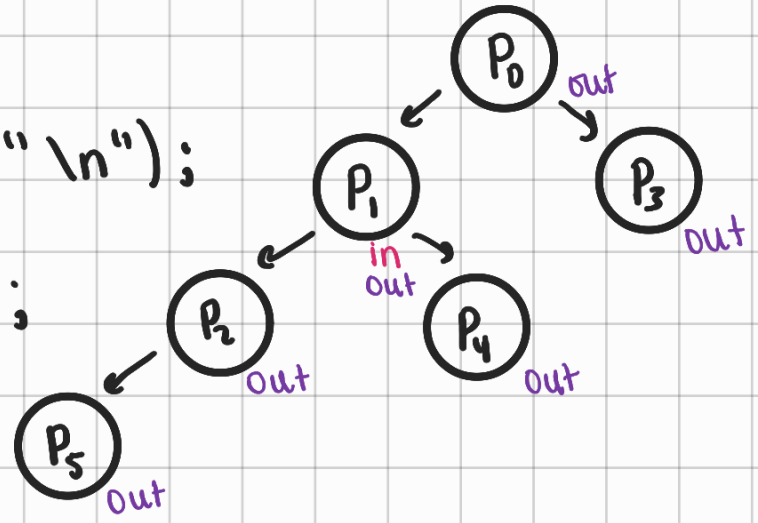
Ex:
```
void main() {
    if (!fork())
        if (!fork())
            printf("I'm in \n");
    fork();
    printf("I'm out \n");
}
```

draw tree + output



-
```
void main(){
    if (! fork())
        if (fork())
            printf ("I'm in" \n");
    fork();
    printf("I'm out \n");
}
```

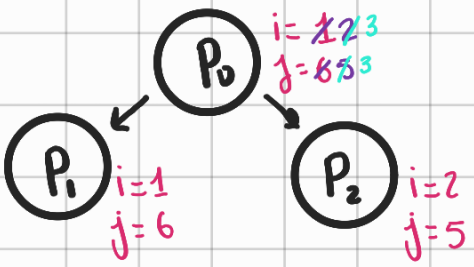
draw tree + output



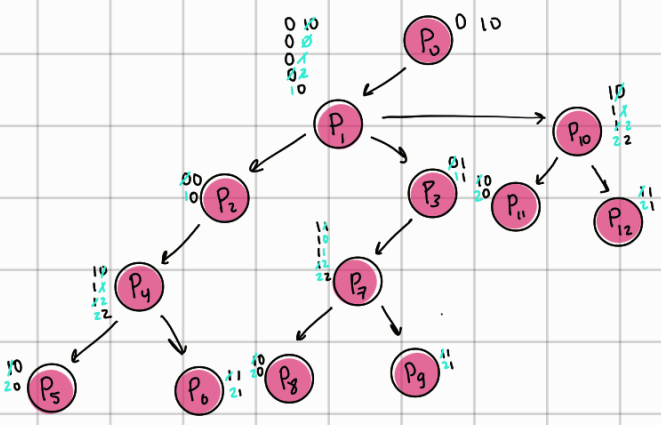
N.B: Break

-
```
void main(){
    int i,j=6;
    for(i=1; i<j; i++){
        if (!fork())
            break;
        else
            j=-i;
    }
    printf("i=%d, j=%d\n", i, j);
}
```

tree of process + output



Ex:
```
void main(){
    int i=10, j=10;
    for(i=0, i<2, i++){
        if (fork()) · if parent break
            break;
        for(j=0, j<2, j++){
            if (!fork()) if child break
                break;
        }
    }
    printf("%d, %d\n", i, j);
}
```



output: 0:0 10 4:22 8:20 12:21

1:12 5:20 9:21

2:10 6:21 10:22

3:11 7:22 11:20

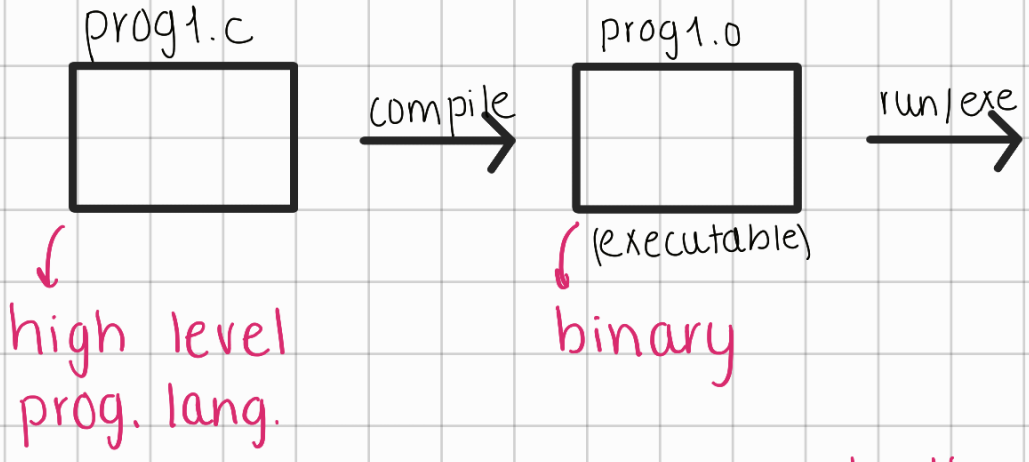
Lecture 2- Process Management.

Resource management
↳ process management
↳ memory management
↳ file system management
↳ I/O devices management
↳ security and protection

} OS responsible for managing these resources.

I- What is a process?

- the process is a **program** in execution.
↳ **program**: set of instructions in machine language



terminal: `>./prog1` → execute the program ↓ process.

Example
program: steps of making a cake.
process: CPU is the baker, he uses the ingredients (data), and follows the recipe (instructions) to make the cake

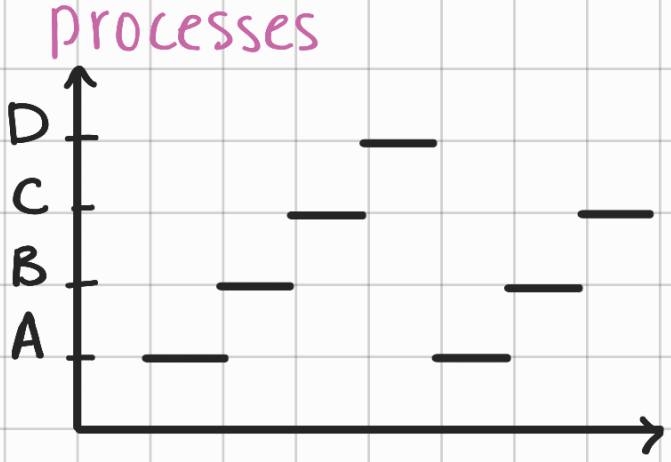
Actual Process Simplified

- When a **program** is run in a computer, its **instructions** are loaded into the **RAM**.
- The **CPU** **fetches** these **instructions** from **memory**, one at a time, **decodes** them, and then **executes** the **instructions**. (process)

Program → static
Process → dynamic

Multi programming mode (time-sharing model)

- execute several processes at the same time
- only one **CPU** → shared resource



(the time is very little, hence unnoticeable)

↳ it seems that 4 independent processes are happening in parallel (**pseudo-parallelism**)

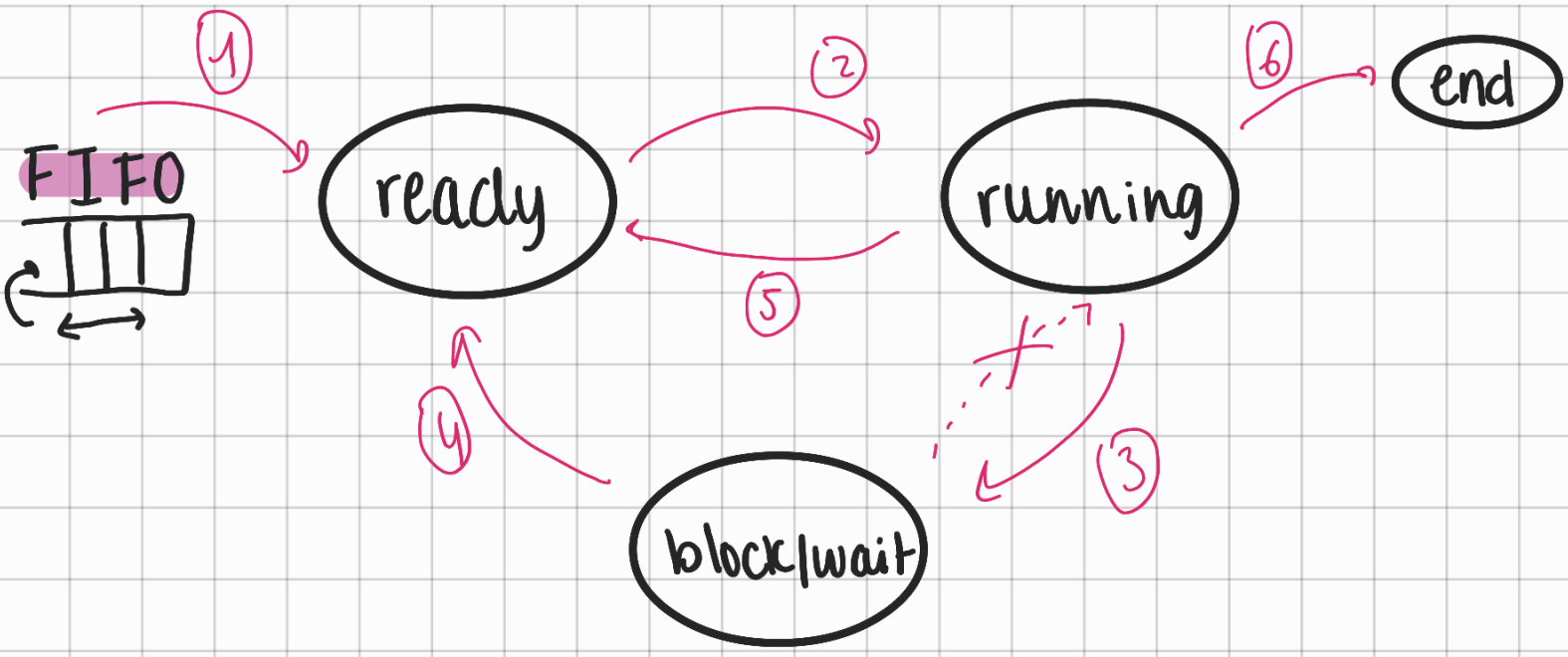
- each process has a programming instruction, input, output, state, set of registers.
- a single processor (CPU) may be shared among several processes
- some scheduling algorithim is used to determin when to terminate a process and serve anotherone

Each process has several attributes:

1- process id → pid (unique identifier)
2- state → to describe
3- priority
4- general purpose registers
   ↳ PC (program counter)
   ↳ IR for decode (instruction register)
   ↳ accumulator
5- list of opened files
6- list of opened devices (I/O devices)
7- protection

State of the process

- refers to the condition or status of a running program / process.
- each process can be in one of several states:
  ↳ **running**: the process is currently being executed by the CPU
  ↳ **ready**: the process was loaded into memory and is prepared to run but is waiting for its turn to execute on the CPU.
  ↳ **blocked / waiting**: the process is in a state of waiting, often because it's waiting for some event, such as user input from an I/O device.
  ↳ **terminated / exit**: the process has completed its execution and has been terminated. It may be waiting for the operating system to clean up resources (freeing memory, closing open files...) or for the parent process to collect exit status (determine out come of the execution



1 the process is created
2 _ the process gets to the CPU
_ the dispatcher elects the process to execute
3 the process makes I/O request so its blocked till the availability of the request
4 I/O request are now available
5 the process is not ended but its quantum time (slice of time) is expired or another process with higher priority arrives to ready queue (context switching)
6 the process is ended

FIFO: "First input, first out put" refers to a data processing principle in which the earliest data or request received is the first to be processed or served.

Types of process

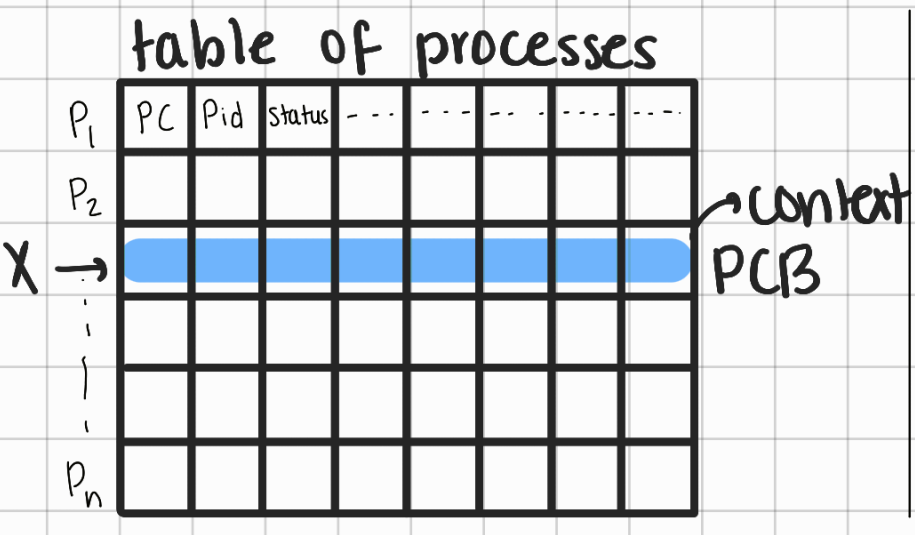
↳ CPU bound → it needs long time CPU (mathematical simulations)

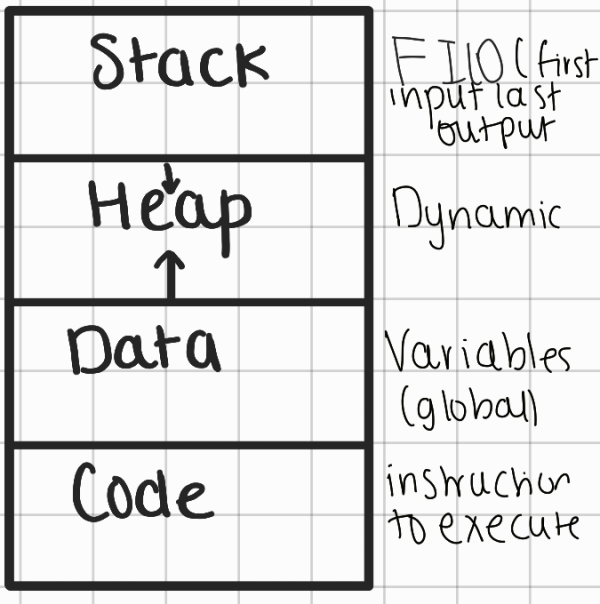
↳ I/O bound → it needs long time I/O (reading files)

Process Creation

PCB: Process Control Block, contains the attributes of each process (pid, state, priority, ...)

in simpler terms: PCB is like a file that the computer uses to keep track of everything it's doing for each running program, such as what it's working on, where it's saved, and how much time it has used so far.





- When a process is created, it is assigned a process control block (PCB), which is stored in the the table of processes located in the operating system's memory. The PCB contains essential info. about the process, including its process identifier (PID), status, priority, and pointers to the memory segments associated with this process, (Stack, Heap, Data, Code)

How Process is Created

1- In UNIX, processes are created using a mechanism called "fork"

2- A process (referred to as the "parent") can create a new process (the "child") by duplicating itself. This duplication includes copying the current state of the parent process such as its code, data, and file descriptors.

3- When you start a UNIX-based system, the first process created has a process identifier (PID) of 1 and is known as "init". Init is responsible for initializing the system and starting other processes.

- The "fork" function is a system call provided by the "unistd.h" library in UNIX. When you call "fork()" in a program, it creates a new child process that is essentially a copy of the current (parent) process. The child process starts executing from the same point in the code as the parent process.

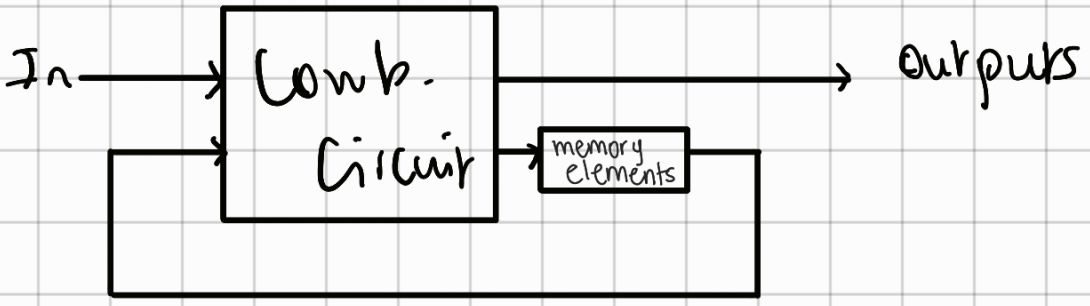
Sequential Circuit

Memory Elements: Devices capable of storing information

State of the Circuit at a Given Moment = The Binary info stored at the el(s) at that moment

Combinational Circuit: **Outputs** = Funct(inp)

Sequential Circuit:
Outputs = Function(inputs, circuit state)
Next State = Function(present state, inputs)



Etat = state
Sorties = output
Suivant = next
entrée = input

} common french terms.

When does the state of Seq. Circ. Change
- ☐ During the change of state of its inputs
  ⇒ Asynchronous Sequential Circuit
- ☐ At discrete moments: Clock Signal.
  ⇒ Synchronous / Clocked Sequential Circuit

Storage elements (memory) used in clocked sequential circuits are called Flip-Flops.

- A flip-flop is a binary storage device capable of storing a bit of information.
- In a stable state, the output of a flip-flop is 0 or 1.
- A sequential circuit can use multiple flip-flops to store as many bits as necessary.
- A change in the state of flip-flops is initiated only by a transition of the clock signal.

Example: the clock signal changes from 0 to 1.

the response time of the combinational circuit must be less than the period of the clock signal

An element of memory in a digital circuit can maintain a binary state indefinitely.

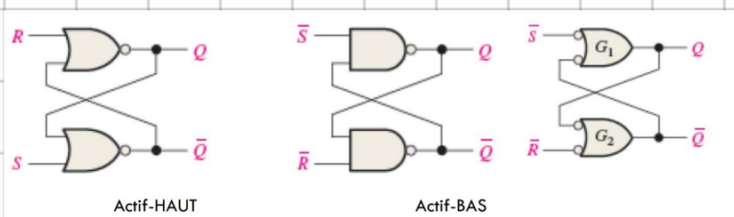
- Power supplied to the circuit
- Change of state according to an input signal

Differences between types of memory elements:

- Number of inputs
- How inputs affect the binary state

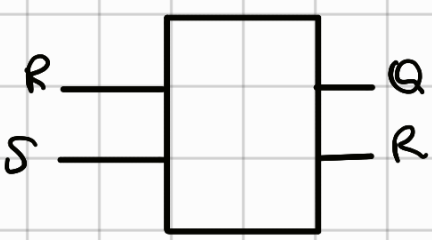
Latches: operate based on signal levels

FlipFlops: controlled by a clock transition



active high

active low

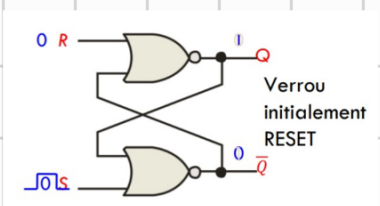


S = Set = 1
R = Reset = 0

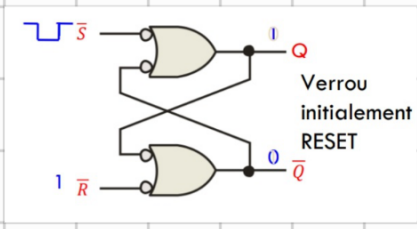
→ An active high SR latch is in stable (locked) condition when both inputs are low

$Q(t+1) = \overline{(R+\bar{Q})}$

$\bar{Q}(t+1) = \overline{(S+Q)}$



latch initially reset



here in stable condition when both inputs are high

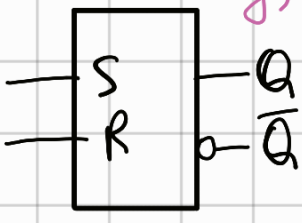
in ⊓ we can't have both high
in ⊔ we can't have both low

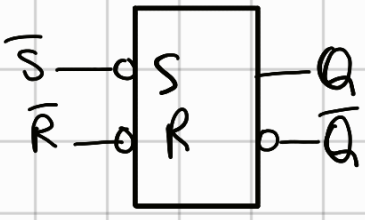
q { $Q \rightarrow$ present
Q { $Q^+, Q^{t+1} \rightarrow$ suivant (next)

Q
$Q(t+1)$

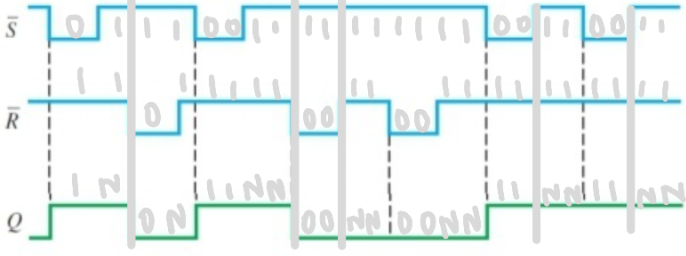
S	R	Q	$\bar{Q}$
0	0	q	$\bar{q}$
0	1	0	1
1	0	1	0
1	1	0	0

No change (memory)
Reset
Set
Invalid





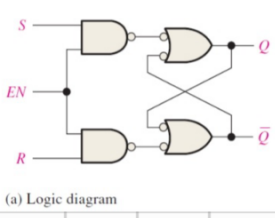
Quelle sera la sortie du Verrou $\bar{S} - \bar{R}$ actif-BAS pour les entrées suivantes?



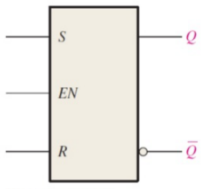
ISSUE: Issue with SR latches: parasitic signal on the inputs causing the latch to change

Solution: Latch with an enable (Gated Latch)
↳ EN must be high for the latch to respond to changes in S and R.

↓



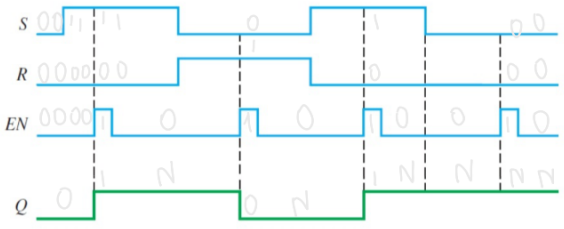
(a) Logic diagram



(b) Logic symbol

En	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

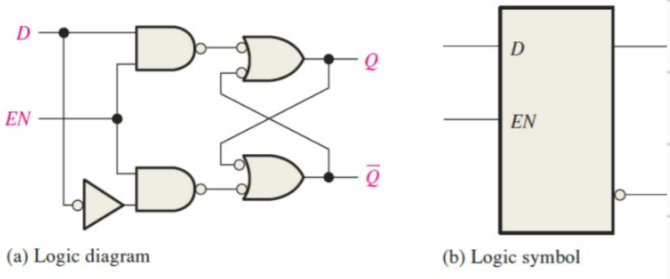
- Quelle est la forme d'onde de sortie Q si les entrées suivantes sont appliquées sur un verrou avec ENABLE étant initialement en état RESET



Issue: Invalid Inputs | Forbidden State

Solution: D Latch

- Single input D (for Data) + EN
- Output Q follows D + EN.
  - → D = high, EN = high ⇒ Latch Set
  - → D = low, EN = high ⇒ Latch Reset



(a) Logic diagram

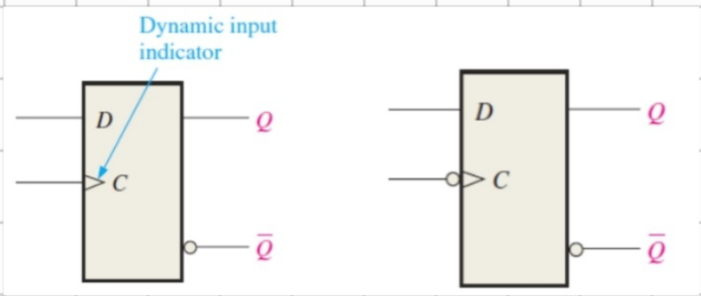
(b) Logic symbol

No change ⇒ EN = 0
Set ⇒ D = 1
Reset ⇒ D = 0

Flip Flops

A flip flop differs from a Latch in the way it changes state

- Clocked device, where only the clocked edge determines when a new bit is entered.
- The active edge can be positive (rising) or negative (falling)

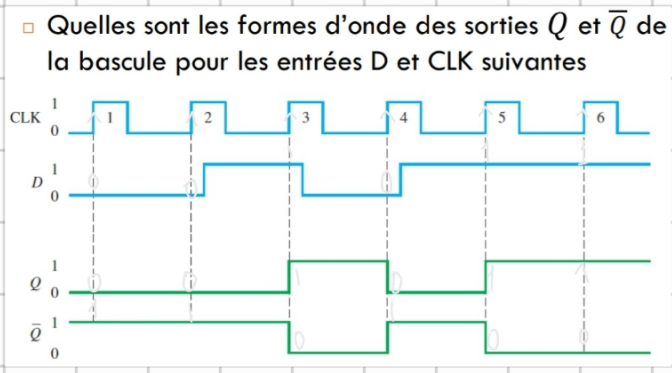


D flip flop

the D input of the D flip flop is synch
↳ the input is transmitted to the flip flop's output on the active edge (rising or falling) of the clock

Truth Table (Rising Edge)

D	CLK	Q	$\overline{Q}$	
0	↑	0	1	Reset
1	↑	1	0	Set



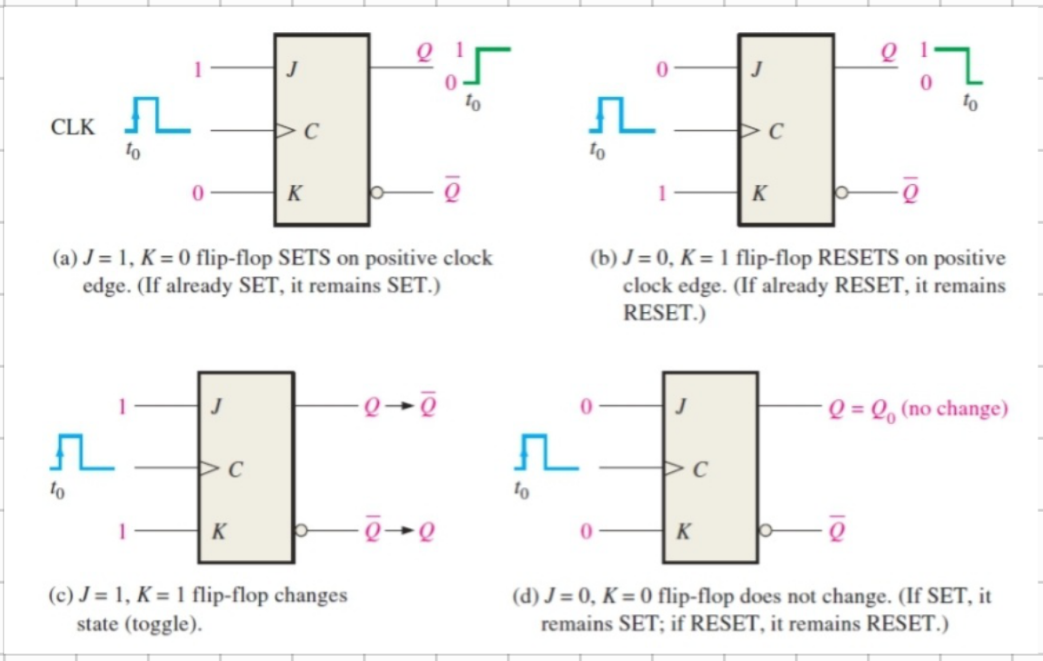
JK Flip Flop

→ D flipflop most used in logic circuits
  → requires fewer gates for its construction
  → other flip flops: JK and T
→ Three operations: set - reset - toggle.
→ D flip flop allows two operations: set and reset.
→ JK flip flop allows all three operations

* The J and K inputs of the JK flip flop are Synchronous.
→ inputs are transmitted to the flipflop's output on the active edge (rising or falling) of the clock.

Truth Table:

J	K	Clk	Q	$\overline{Q}$	
0	0	↑	Q_0	$\overline{Q_0}$	No change (memory)
0	1	↑	0	1	Reset
1	0	↑	1	0	Set
1	1	↑	$\overline{Q_0}$	Q_0	Toggle.



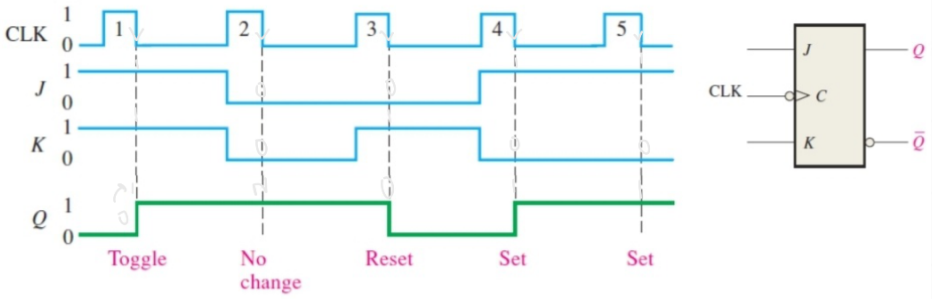
(a) $J = 1$, $K = 0$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)

(b) $J = 0$, $K = 1$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)

(c) $J = 1$, $K = 1$ flip-flop changes state (toggle).

(d) $J = 0$, $K = 0$ flip-flop does not change. (If SET, it remains SET; if RESET, it remains RESET.)

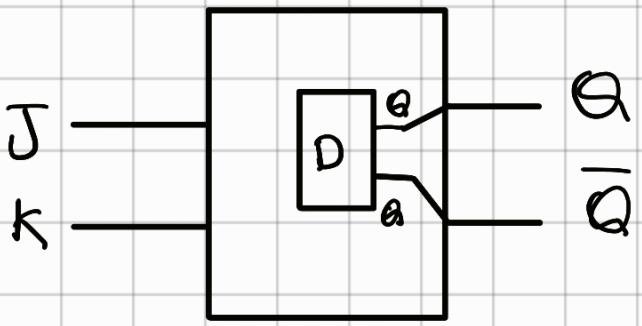
- Quelle est la forme d'onde Q à la sortie de la bascule J-K pour les formes d'onde d'entrée suivantes? (Note: front actif descendant)



(falling edge active)

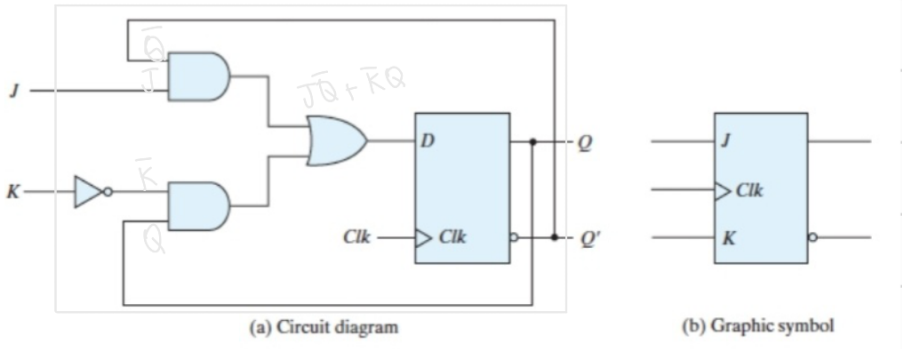
Design and Implement a JK flip flop using a D flipflop and logic gates

J	K	Q	Q*	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0



JK \ Q	0	1
00		1
01		
11	1	
10	1	1

$$D = Q^* = J\bar{Q} + \bar{K}Q$$



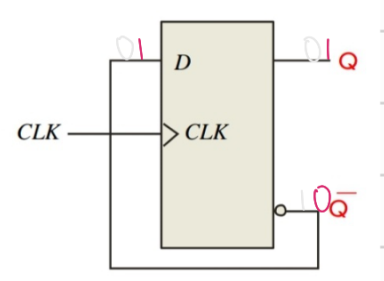
(a) Circuit diagram (b) Graphic symbol

$$D = J\overline{Q} + \overline{K}Q$$

D flipflop in toggle mode

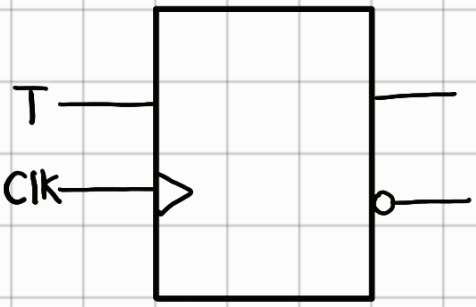
→ D can operate in toggle mode by conn the ourput Q to D

→ the flip flop only changes on the active edge ⇒ the output changes only with each clock purse



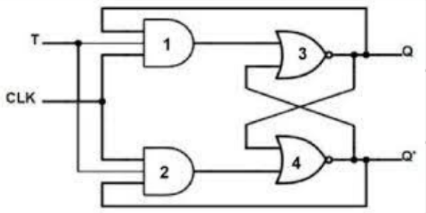
Flip Flop T

T = 0 ⇒ unchanged state
T = 1 ⇒ toggle.



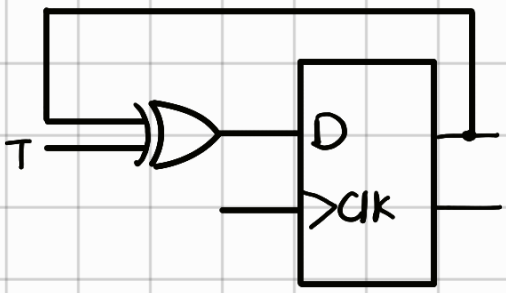
Q	T	clk	Q^+
0	0	↑	0
0	1	↑	1
1	0	↑	1
1	1	↑	0

$Q^+ = Q \oplus T$



Design and implement a T flip flop using
a) D flip flop
b) JK flip flop.

a) $D = Q^+ = Q \oplus T$



b)

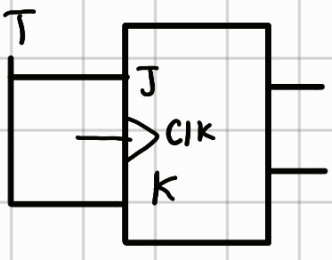
T	Q	Q^+	J	K	J	K
0	0	0	0	0	0	X
0	1	1	0	0	X	0
1	0	1	1	0	1	X
1	1	0	1	1	X	1

T \ Q	0	1
0	0	X
1	1	X

J = T

T \ Q	0	1
0	X	0
1	X	1

K = T



Characteristic Table

relationship btw current state and next state of the flip flop based on inputs

JK:

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

D:

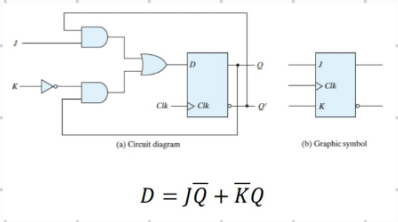
D	Q^*
0	0
1	1

T:

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

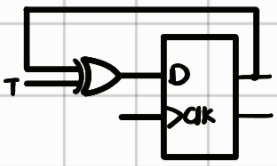
Characteristic Equations:

flip flop D: $Q(t+1) = D$



flip flop JK: $Q(t+1) = J\overline{Q} + \overline{K}Q$

flip flop T: $Q(t+1) = T \oplus Q = T\overline{Q} + \overline{T}Q$



Next State = f(inputs, present state)

Synchronous Inputs

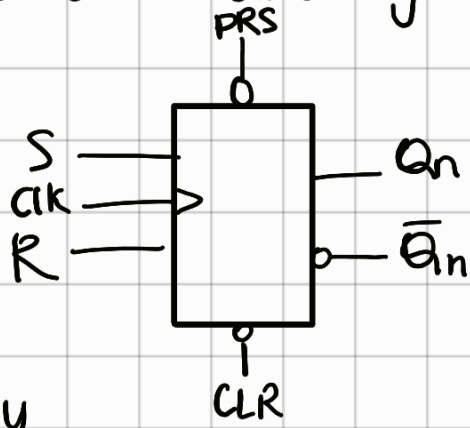
- Synchronous inputs (D or J-K) are transferred to the active edge of the clock.
- Most flip-flops have asynchronous inputs
  - Affect the output independently of the clock.
  - Normally called PRESET (PRE) and CLEAR (CLR)
  - Usually Active-LOW
- Connected to override the input effect synchronous D and the clock.
  - PRESET and CLEAR are not practically low in same time

preset = 0 ⇒ $Q_n = 1$

clear = 0 ⇒ $Q_n = 0$ because $\overline{Q_n} = 1$

regardless of the values of clock and syn. inputs

PR	CLR	Q_n
0	0	INDEFINITE
0	1	1
1	0	0
1	1	will perform normally

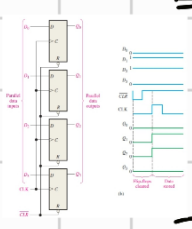


Flip Flop Usage

→ Parallel Storage of Data

⇒ A group of flip flops are connected to data lines parallel and clocked at the same time.

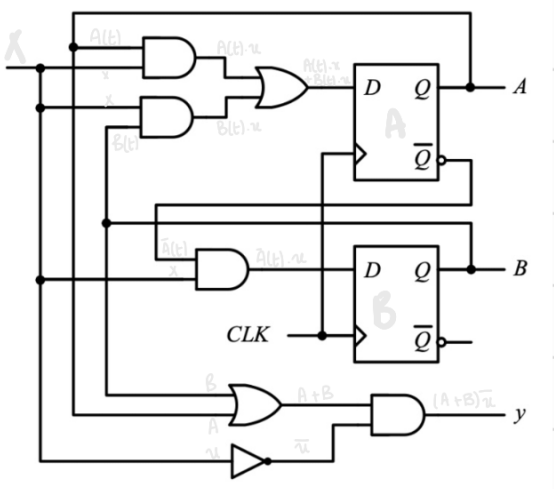
⇒ The Data is stored until the clock pulse next



Analysis of Clocked Sequential Circuits

The analysis of a sequential circuit consists of:

- → obtaining a state equation or transition equation
- → obtaining a table state for the time sequence of inputs, our puts, and internal state.
- → Draw a state diagram
- → Draw a timing diagram



input(s): x

flipflops: 2 D flipflops : D_A, D_B

output(s): y

$$B(t+1) = D$$
$$A(t+1) = Ax + Bx$$
$$B(t+1) = \bar{A}x$$
$$y(t) = (A+B)\bar{x}$$

} state equations

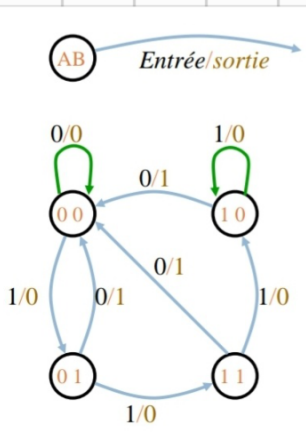
State Table

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0
t			t+1		t

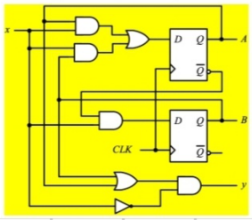
OR

Present State		Next State				Output	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

State Diagram



Etat Présent	Etat Suivant				Sortie	
	$x=0$		$x=1$		$x=0$	$x=1$
A B	A	B	A	B	y	y
0 0	0	0	0	1	0	0
0 1	0	0	1	1	1	0
1 0	0	0	1	0	1	0
1 1	0	0	1	0	1	0



Note: n flipflops → 2^n states.

Example (D flipflop)

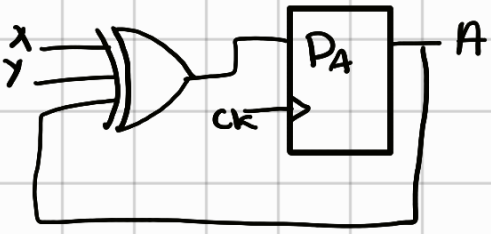
- Analyser le circuit décrit par l'équation d'entrée

$$D_A = A \oplus x \oplus y$$

- Pas d'équation de sortie => la sortie est celle de la bascule => A

No output equation => the output is that of the flipflop => A.

inputs: x, y
flipflops: 1 D flipflop (A)
outputs: A.

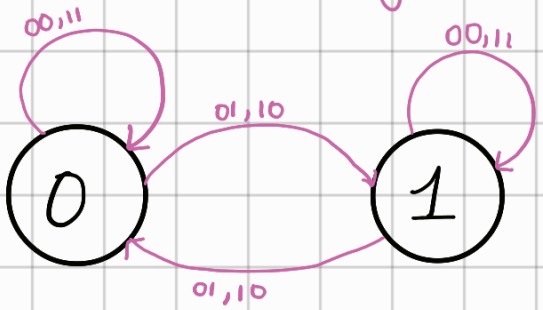


State Equation: $A(t+1) = A^+ = A^* = D_A = A \oplus x \oplus y$

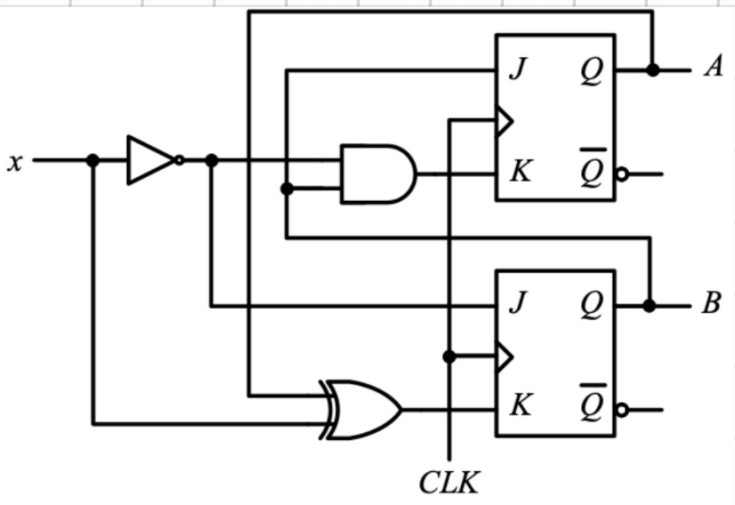
State Table:

A	x	y	A^+
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

State Diagram:



JK Flip Flop Analysis Diagram



Input(s): x
Flip Flop(s): 2 JK ff. A, B
Output(s): no output

State Table:

A	B	x	A^*	B^*	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

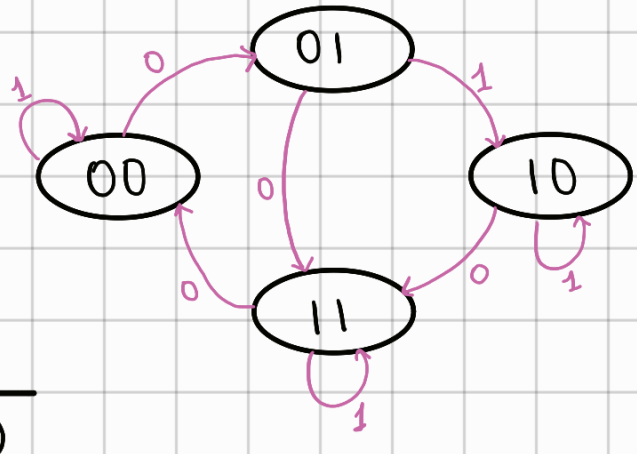
I/O Equations

$J_A = B$

$K_A = B\bar{x}$

$J_B = \bar{x}$

$K_B = A \oplus x$



A^*:

AB \ x	0	1
00	0	0
01	1	1
11	0	1
10	1	1

B^*:

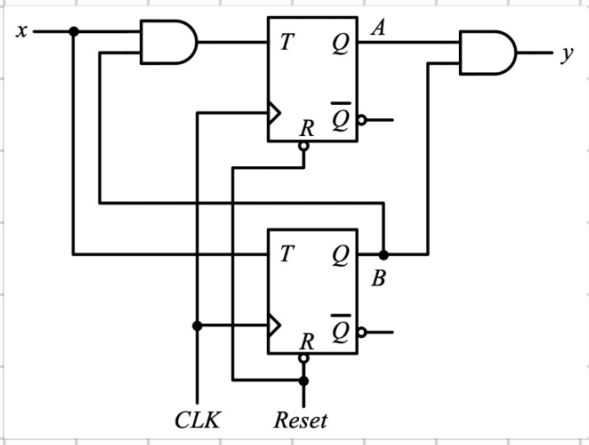
AB \ x	0	1
00	1	0
01	1	0
11	0	1
10	1	0

State Equations

$A^* = \bar{A}B + Bx + A\bar{B}$

$B^* = \bar{A}\bar{x} + \bar{B}\bar{x} + ABx$

T Flip Flop Analysis Diagram



Input(s) : x
Flipflop(s): 2 T flip flops A, B
Output(s) : y

Input Equation:

$T_A = Bx$

$T_B = x$

$y = AB$

A	B	x	A*	B*	T_A	T_B	y
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	0
1	0	0	1	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	1	1

A*:

AB \ x	0	1
00	0	0
01	0	1
11	1	0
10	1	1

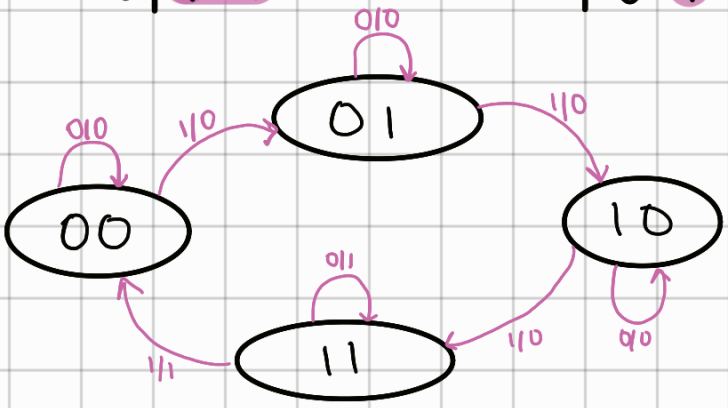
B*:

AB \ x	0	1
00	0	1
01	1	0
11	1	0
10	0	1

State Equations:

$A^* = \bar{A}Bx + A\bar{x} + A\bar{B}$

$B^* = B\bar{x} + \bar{B}x = B \oplus x$



Mealy and Moore

Mealy Model: output is a function of both the present state and the input

Moore Model: output is a function of only the present state

In a **Moore Model** the **outputs** of the seq. circuit are **Synchronized** with the **clock**, because they depend only on flip-flop outputs that are synchronized with the clock

The **output** of the **Mealy Machine** is the **value** that is present immediately before the active edge of the clock

Basically when the output is independent of the input (ex: $y = \overline{Q}_B + Q_A$) it's Moore.

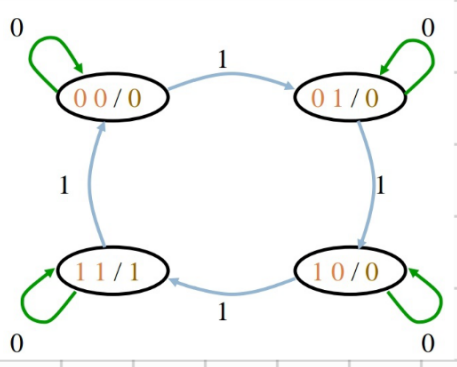
Mealy

Etat Present		E	Etat Suivant		S
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Moore

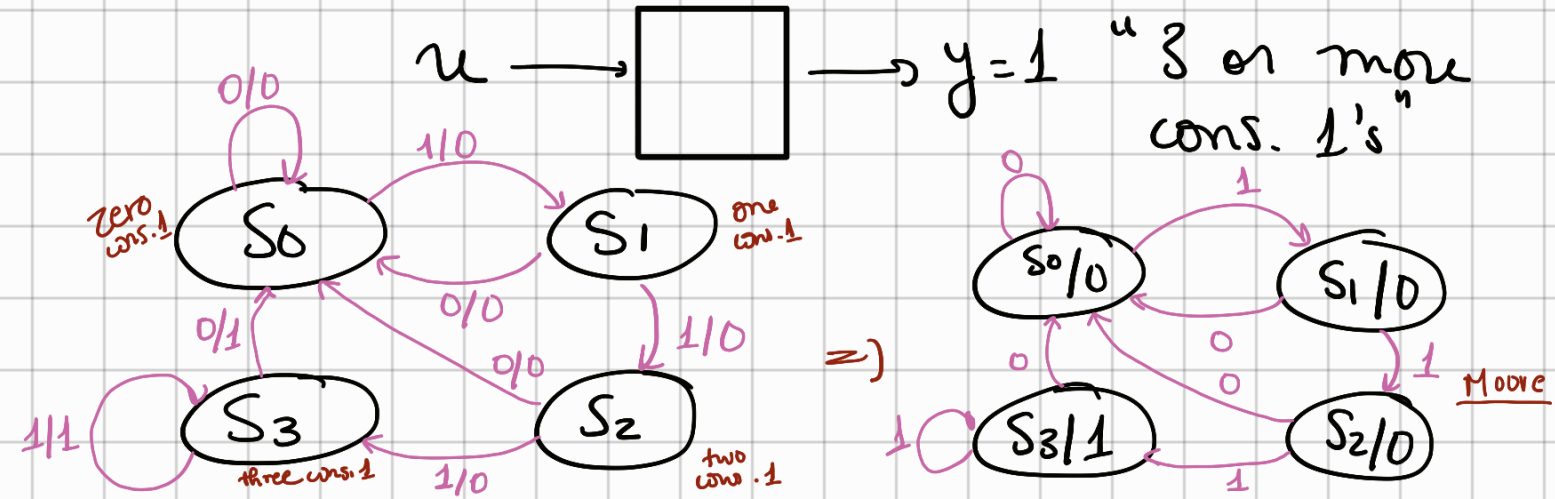
Etat Present		E	Etat Suivant		S
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

Moore

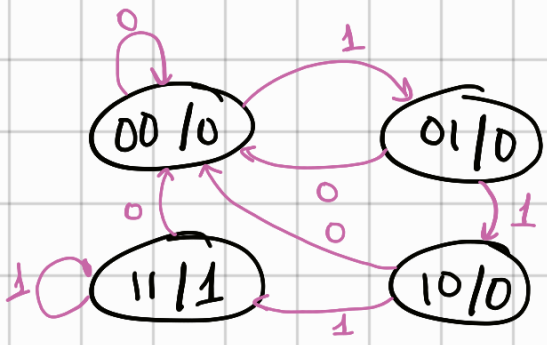


Design of Timed Sequential Circuit

Example:
Detect 3 or more consecutive 1's



4 States ⇒ 2 flipflops



State	A	B
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

Present State		Input	Next State		Output
A	B	u	A^+	B^+	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Flip Flop D: State equation = Input Equation

$A^+ = D_A \quad B^+ = D_B \quad Y = ?$

A^+:

AB \ u	0	1
00	0	0
01	0	1
11	0	1
10	0	1

$A^+ = Bu + Au$

$= u(A+B)$

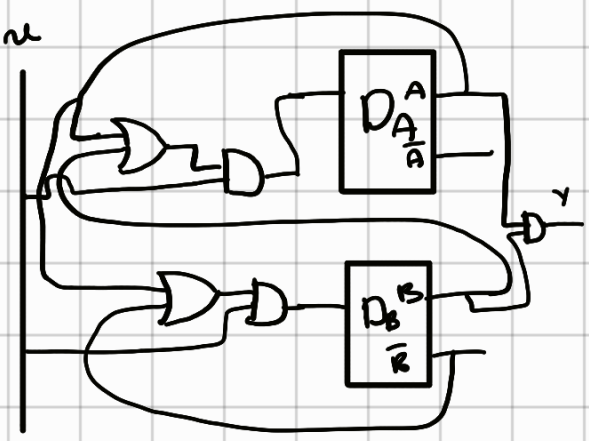
B^+:

AB \ u	0	1
00	0	1
01	0	0
11	0	1
10	0	1

$B^+ = Au + \bar{B}u$

$= u(A+\bar{B})$

$Y = AB$



Excitation Table

Etat Présent	Etat Suivant	Entrée Bascule
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Etat Présent	Etat Suivant	Entrée Bascule	
$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

0 0 (No change)
0 1 (Reset)
1 0 (Set)
1 1 (Toggle)
0 1 (Reset)
1 1 (Toggle)
0 0 (No change)
1 0 (Set)

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

JK Flip Flop: WAY 1

Etat Présent		Entrée	Etat Suivant		Entrée Bascule			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	0	0	x	1	0	x
1	0	1	1	1	x	0	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	1	x	0	x	0

J_A:

AB \ u	0	1
00		
01		1
11	X	X
10	X	X

$J_A = Bu$

K_A:

AB \ u	0	1
00	X	X
01	X	X
11	1	
10	1	

$K_A = \bar{u}$

J_B:

AB \ u	0	1
00		1
01	X	X
11	X	X
10		1

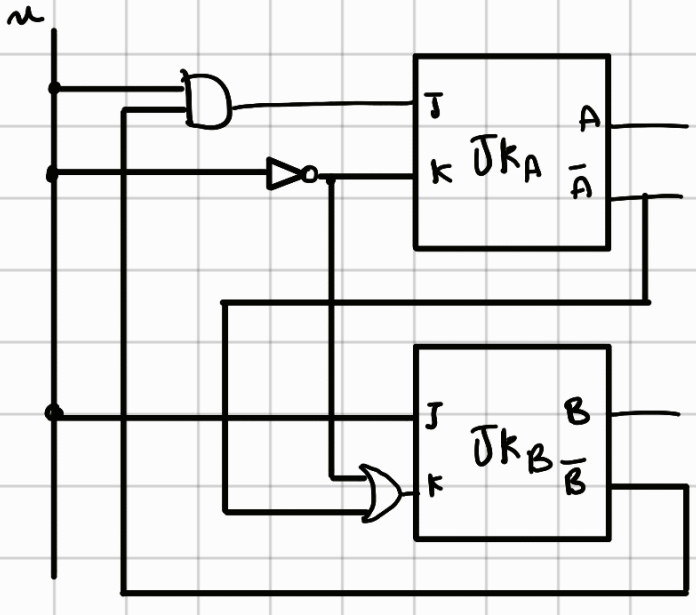
$J_B = u$

K_B:

AB \ u	0	1
00	X	X
01	1	1
11	1	
10	X	X

$K_B = \bar{u} + \bar{A}$

$Y = AB$



WAY 2:

$$Q^+ = J\bar{Q} + \bar{K}Q$$

$Q = 0 \Rightarrow Q^+ = J$

$Q = 1 \Rightarrow Q^+ = \bar{K}$

Present State		Input	Next State		Output
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

$A^+ = J_A\bar{A} + \bar{K}_A A$

$A = 0 \quad J_A = A^+$

$A = 1 \quad K_A = \bar{A}^+$

$B^+ = J_B\bar{B} + \bar{K}_B B$

$B = 0 \quad J_B = B^+$

$B = 1 \quad K_B = \bar{B}^+$

A^+

AB \ x	0	1
00		
01		1
11	///	1
10	///	1

$J_A = \bar{A}Bx = Bx$

A^+

AB \ x	0	1
00	///	///
01	///	1
11		1
10		1

$K_B = \overline{Ax} = \bar{x}$

B^+

AB \ x	0	1
00		1
01	///	///
11	///	1
10		1

$J_A = \bar{B}x = x$

B^+

AB \ x	0	1
00	///	///
01		
11		1
10	///	///

$K_B = \overline{ABx} = \bar{A} + \bar{x}$

T Flip Flop:

Etat Présent		Entrée	Etat Suivant		Entrée Bascule	
A	B	x	A	B	T_A	T_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	1	1	1	0	0

T_A:

AB \ x	0	1
00		
01		1
11	1	
10	1	

$T_A = A\bar{x} + \bar{A}Bx$

T_B:

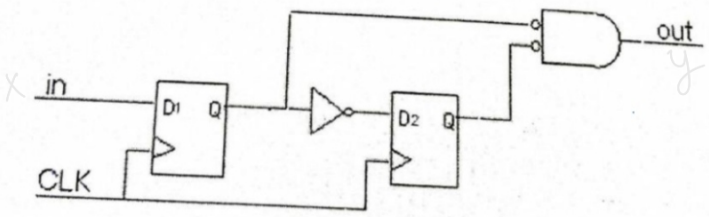
AB \ x	0	1
00		1
01	1	1
11	1	
10		1

$T_B = B\bar{x} + \bar{B}x + \bar{A}x$

$= B \oplus x + \bar{A}x$

or $B \oplus x + \bar{A}B$

Exercise1



a) Give the input and output equations of this circuit.
b) Give the state table of this circuit.
c) Draw the circuit state diagram.
d) Deduce if this circuit is a Mealy or Moore system. Justify your answer.
e) Re-implement the circuit using JK flip-flops

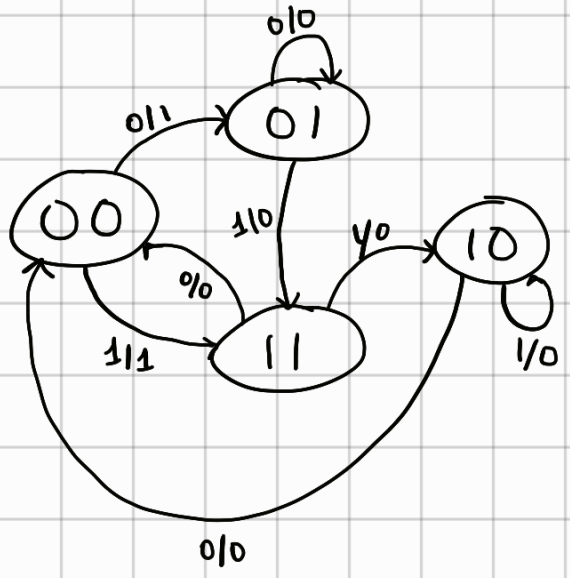
a) input(s): x
flipflops: 2 D flipflops D1, D2.
output(s): y.

$$D_A = x \qquad D_1 = in$$
$$D_B = \bar{A} \qquad D_2 = \bar{Q_1}$$
$$y = \bar{A} \cdot \bar{B} \qquad Y = \bar{Q_1}\,\bar{Q_2}$$

b)

present state		inp	next state		out
A	B	x	A^+	B^+	Y
0	0	0	0	1	1
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	1	0	0

c)



d) Moore since the output isn't influenced by the input, as in the output for each state doesn't change with the change of input.

e) $A^+ = J\bar{A} + \bar{K}A$

$A=0 \quad A^+ = J$

$A=1 \quad A^+ = \bar{K}$

$B^+ = J\bar{B} + \bar{K}B$

$B=0 \quad B^+ = J$

$B=1 \quad B^+ = \bar{K}$

A^+

AB \ x	0	1
00	0	1
01	0	1
11	0	1
10	0	1

B^+

AB \ x	0	1
00	1	1
01	1	1
11	0	0
10	0	0

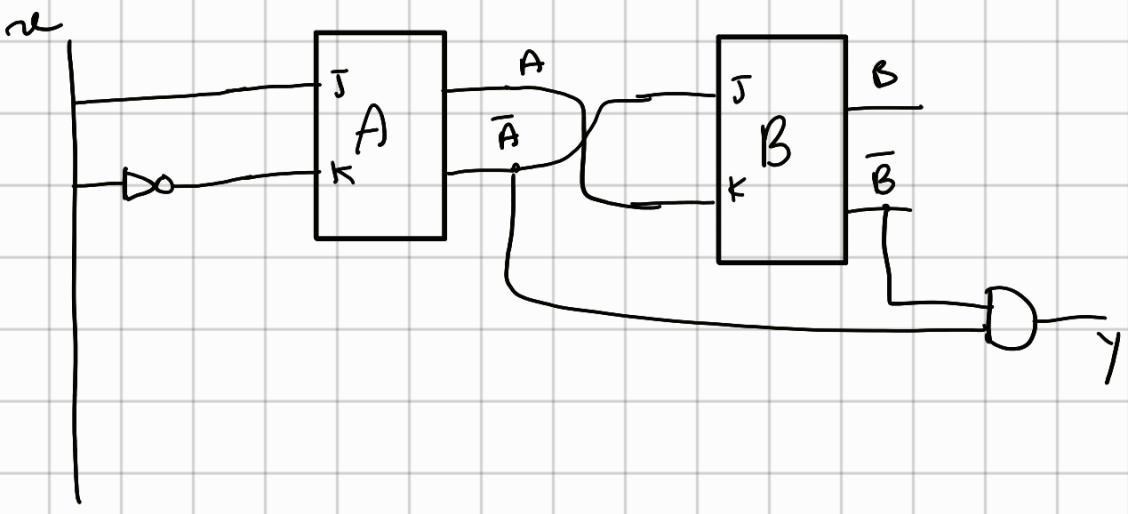
$J_A = \bar{A}x = x$

$K_A = \overline{Ax} = \bar{A} + \bar{x} = \bar{x}$

$J_B = \bar{A}\bar{B} = \bar{A}$

$K_B = \overline{\bar{A}B} = A + \bar{B} = A$

$Y = \bar{A}\bar{B}$

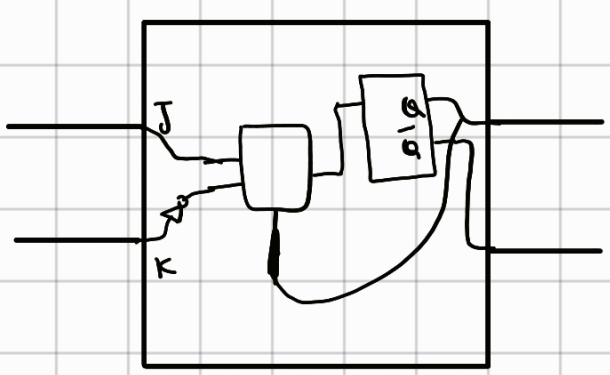


Construct a *JK* flip-flop using a *D* flip-flop, a two-to-one-line multiplexer, and an inverter.

Show that the characteristic equation for the complement output of a *JK* flip-flop is

$$Q'(t+1) = J'Q' + KQ$$

A *PN* flip-flop has four operations: clear to 0, no change, complement, and set to 1, when inputs *P* and *N* are 00, 01, 10, and 11, respectively.
(a) Tabulate the characteristic table. (b)* Derive the characteristic equation.

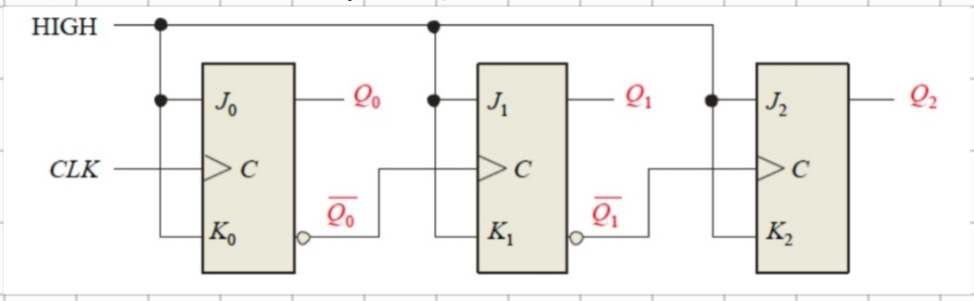


JK: $Q^+ = J\bar{Q} + \bar{K}Q$

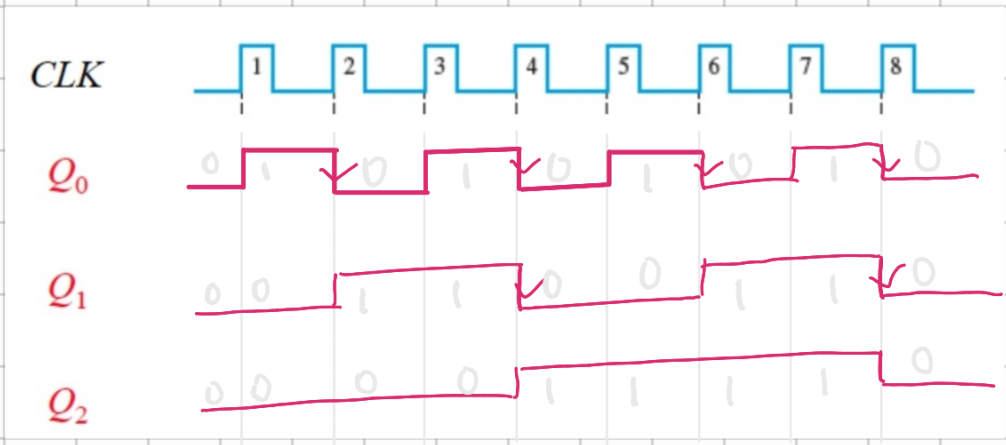
D: $Q^+ = D$

Counters

Asynchronous

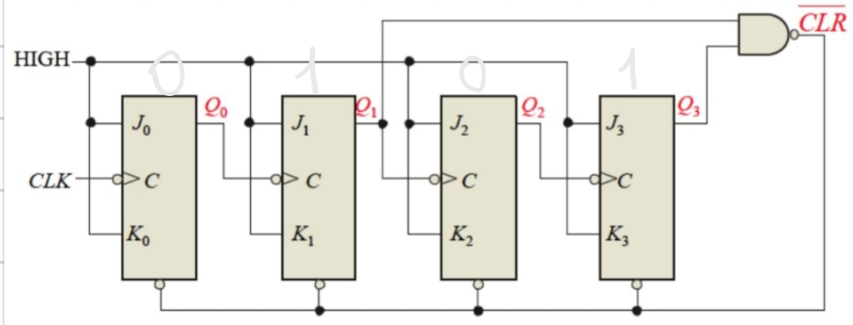


dividers freq. by 2.
counts from 0-7

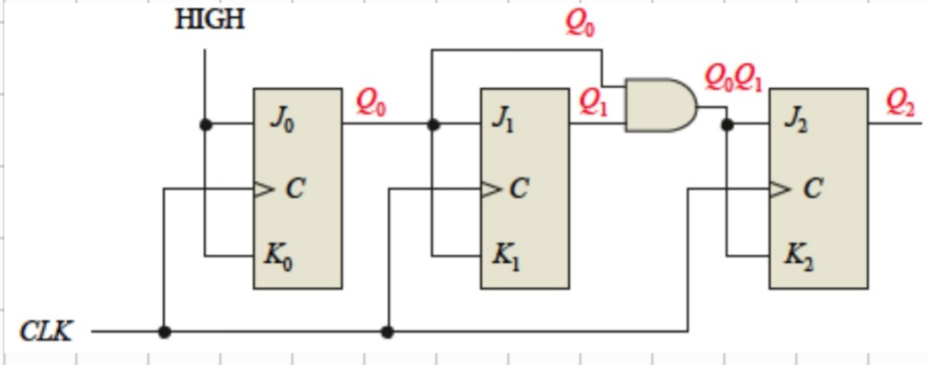


Asynchronous Decade Counters

Count from 0-9, when it reaches 1010, the counter resets



Synchronous

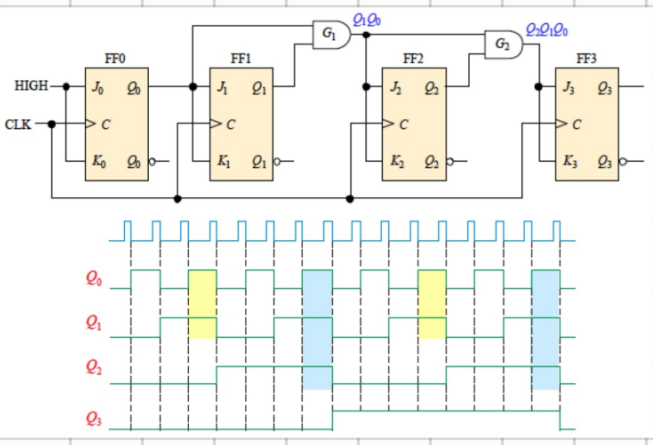


Q_2	Q_1	Q_0	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	$Q_2 = 1$ when $Q_1Q_0 = 1$
1	0	1	
1	1	0	
1	1	1	

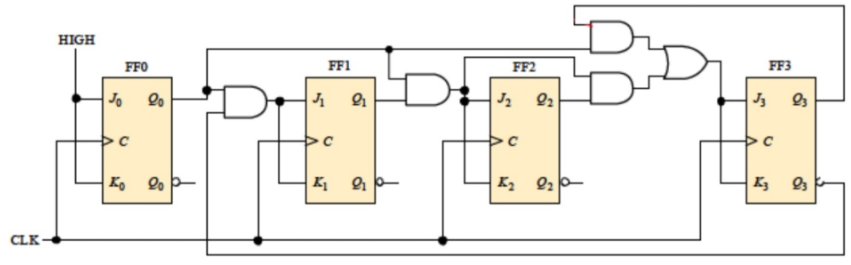
Analyse: circuit → Input Equation → State Eq → State Table → State Diagram → function

Design: Problem → State Diagram → State Table → State & Inp. eq. → Scheme circ.

4bits counter



Synchronous BCD Counter

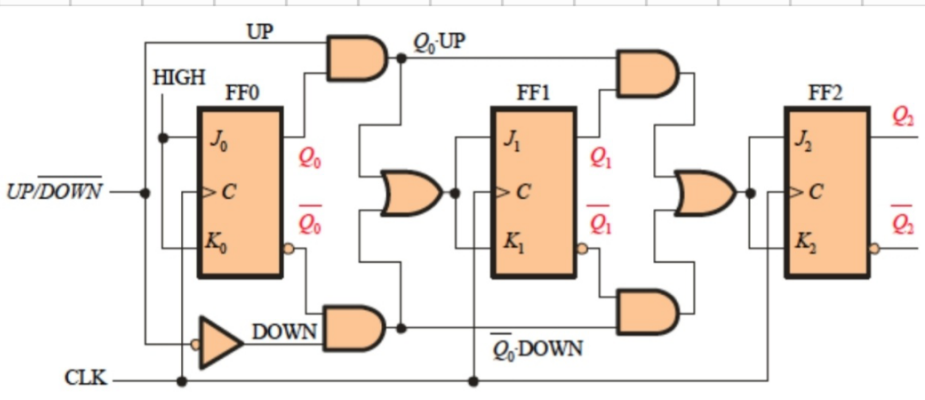


- Q_0 toggles on each clock pulse ⇒ $J_0 = K_0 = 1$
- Q_1 toggles on next clock pulse each time $Q_0 = 1$ & $Q_1 = 0$ ⇒ $J_1 = K_1 = Q_0 \cdot \overline{Q_3}$
- Q_2 toggles on next clock pulse each time both Q_0 & $Q_1 = 1$ ⇒ $J_2 = K_2 = Q_0 Q_1$

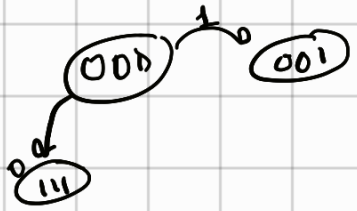
Q_3 toggles for 2 conditions:

1) $Q_0 = 1$, $Q_1 = 1$, $Q_2 = 1$.
2) $Q_0 = 1$, $Q_3 = 1$

⇒ $J_3 = K_3 = Q_0 Q_1 Q_2 + Q_0 Q_3$



counter up or down



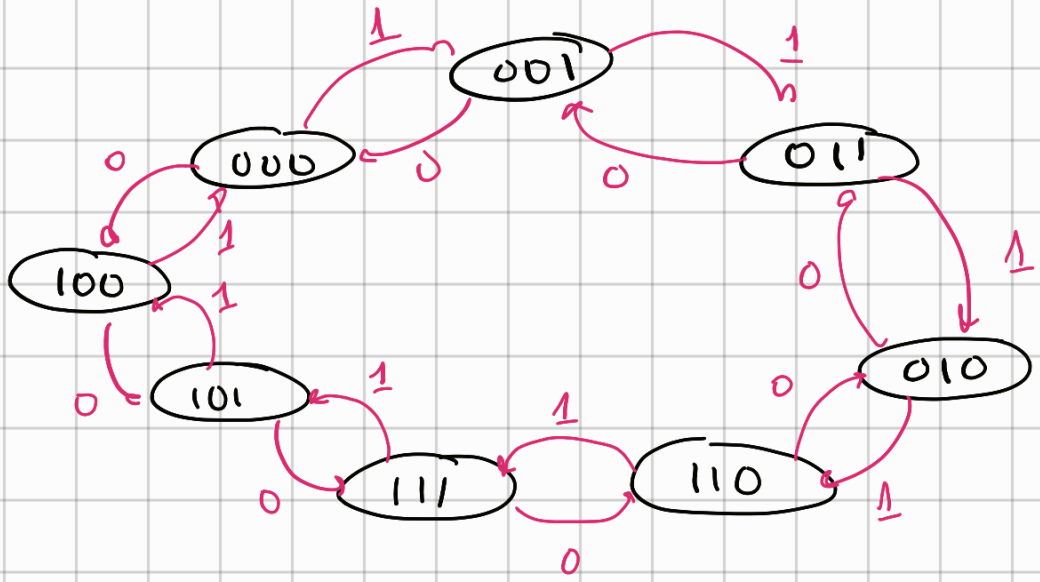
Exercise

Develop a 3-bit synchronous up/down counter for a sequence of UP/DOWN counter for a sequence of a 3-bit gray code in using JK flipflops. The counter must count UP when the UP/DOWN input is 1 and down if input is 0.

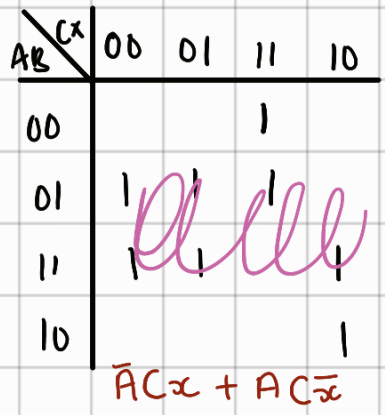
→

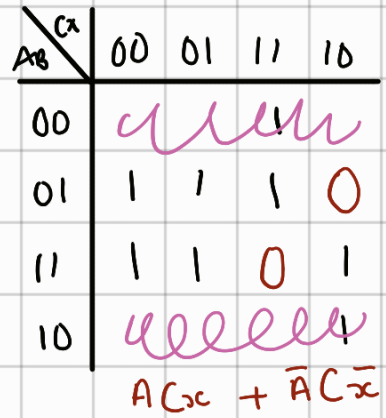
Binary	Gray
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 1
0 1 1	0 1 0
1 0 0	1 1 0
1 0 1	1 1 1
1 1 0	1 0 1
1 1 1	1 0 0

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0



A	B	C	X	A^+	B^+	C^+
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	0	1	1	1	0
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	1	1	1
1	0	1	1	1	0	0
1	1	0	0	0	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	0
1	1	1	1	1	0	1





$J_A = \bar{B}\bar{C}\bar{x} + B\bar{C}x$

$J_B = \bar{A}Cx + AC\bar{x}$

$J_C = B\bar{x} + \bar{B}x = B \oplus x$

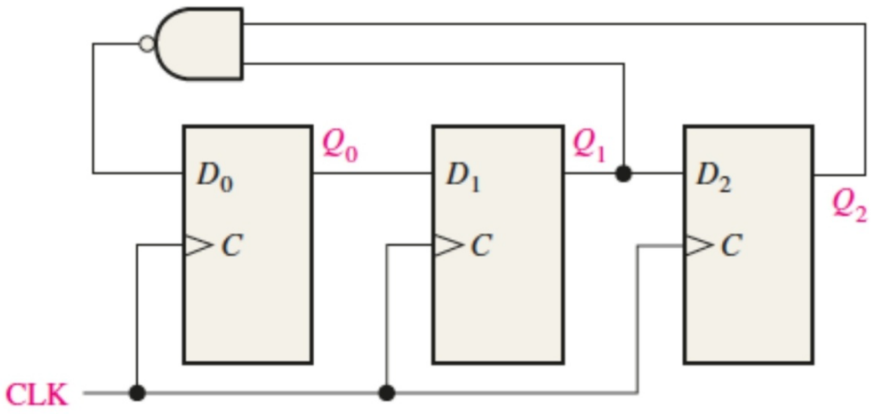
$K_A = B\bar{C}\bar{x} + \bar{B}\bar{C}x$

$K_B = ACx + \bar{A}C\bar{x}$

$K_C = Bx + \bar{B}\bar{x} = B \odot x$

Exercise

Develop the sequence of the following counter.

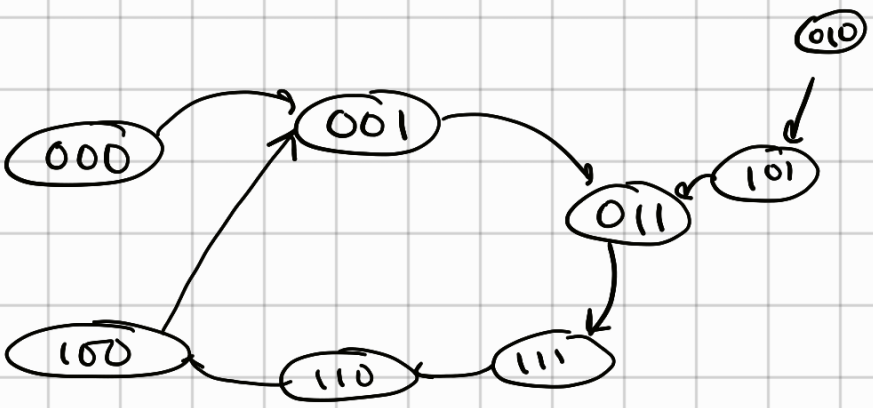


$D_0 = \overline{Q_1 Q_2}$ $\quad Q_2^* = D_2$

$D_1 = Q_0$ $\quad Q_1^* = D_1$

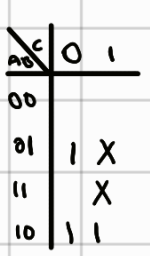
$D_2 = Q_1$ $\quad Q_0^* = D_0$

Q_2	Q_1	Q_0	Q_2^*	Q_1^*	Q_0^*
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	1	1	1
1	1	1	1	1	0
1	1	0	1	0	0
1	0	0	0	0	1
0	1	0	1	0	1
1	0	1	0	1	1

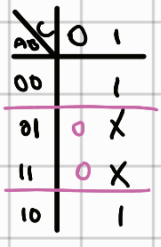


Use JK flip-flops to design a 3-bit self-correctiong counter with the repeated binary sequence: 0, 1, 2, 4, 5, 6. Show what happened if the counter is started in any of unused states.

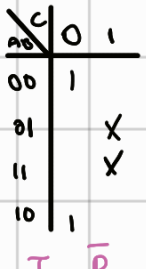
A	B	C	A^*	B^*	C^*
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	X1	X0	X0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	X0	X0	X0



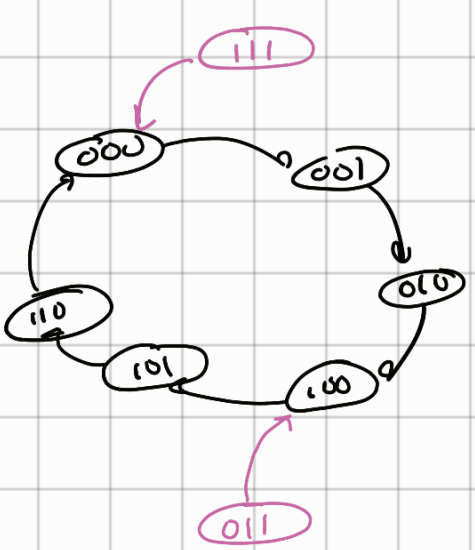
$J_A = B$
$K_A = B$



$J_B = C$
$K_B = 1$

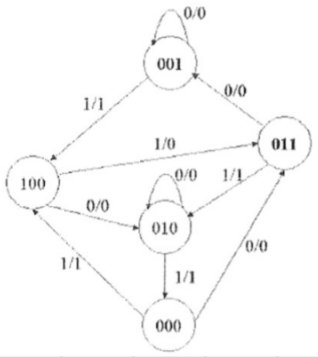


$J_C = \overline{B}$
$J_C = 1$



A sequential circuit has three flip-flops A, B, C; one input x; and one output, y. The state diagram is shown in next figure.
The circuit is to be designed by treating the unused states as ***don't-care*** conditions.
Analyze the circuit obtained from the design to ***determine*** the effect of the unused states.

1) Use D flip-flops in the design.
2) Use J-K flip-flops in the design.



A	B	C	x	A*	B*	C*	y
0	0	0	0	0	1	1	0
0	0	0	1	1	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	1	0	0	1
0	1	0	0	0	1	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	1	0	1
1	0	0	0	0	1	0	0
1	0	0	1	0	1	1	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

AB \ Cx	00	01	11	10
00		1	1	
01				
11	X	X	X	X
10			X	X

AB \ Cx	00	01	11	10
00		1	1	
01				
11	X	X	X	X
10	0	0	X	X

AB \ Cx	00	01	11	10
00	1			
01	1		1	
11	X	X	X	X
10	1	1	X	X

AB \ Cx	00	01	11	10
00	1			
01	1	0	1	0
11	X	X	X	X
10			X	X

AB \ Cx	00	01	11	10
00	1			1
01				1
11	X	X	X	X
10		1	X	X

AB \ Cx	00	01	11	10
00	1		0	1
01			0	1
11	X	X	X	X
10		1	X	X

$J_A = \bar{B}x$ $\quad K_A = 1$

$J_B = \bar{C}\bar{x} + A$ $\quad K_B = \bar{C}x + C\bar{x} = C \oplus x$

$J_C = \bar{A}\bar{B}\bar{x} + Ax$ $\quad K_C = Cx$

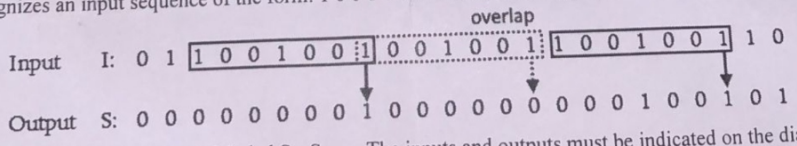
$y = \bar{A}x$

AB \ Cx	00	01	11	10
00		1	1	
01		1	1	
11	X	X	X	X
10			X	X

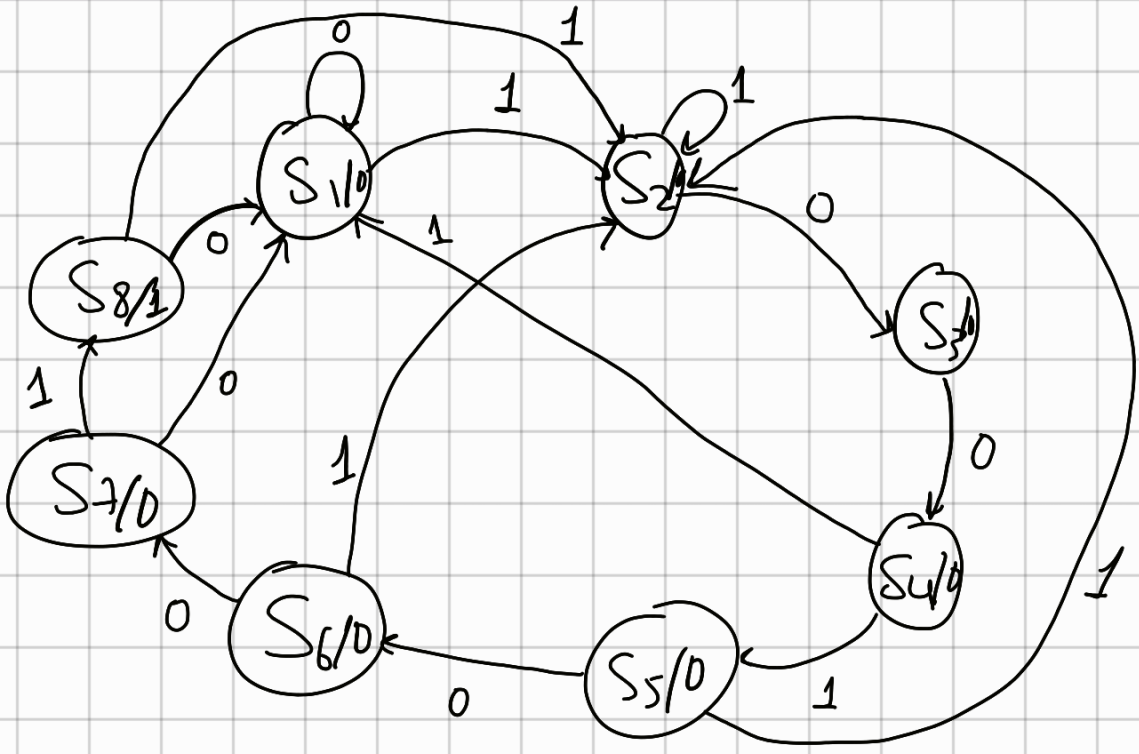
Draw a table with missing states and add them to diagram

Exercise IV (15 pts)

Draw the state diagram of a Moore-type sequential circuit with one input (I) and one output (S) that recognizes an input sequence of the form: 1 0 0 1 0 0 1 **without overlapping**.



The states of the diagram are labeled S_1, S_2, … The inputs and outputs must be indicated on the diagram.



Design a logic circuit that controls the elevator door in a three-story building. The circuit has four inputs. M is a logic signal that indicates when the elevator is moving (M=1) or stopped (M=0). F1, F2, and F3 are floor indicator signals that are normally LOW, and they go HIGH only when the elevator is positioned at the level of that particular floor. For example, when the elevator is lined up level with the second floor, F2=1 and F1=F3=0. The circuit output is the OPEN signal, which is normaly LOW and will go HIGH when the elevator door is to be opened. Find the simplified design and draw the diagram of the elevator circuit.

Note: Becaise the elevator cannot be lined up with more than one floor at a time, the cases when more than one floor inputs can be treated as don't care condition. When the elevator reaches at on floor and stops, the door opens.

1 XXX 0

M	F_1	F_2	F_3	Output
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

MF1 \ F2F3	00	01	11	10
00		1	X	1
01	1	X	X	X
11		X	X	X
10			X	

$$\overline{M}F_1 + \overline{M}F_3 + \overline{M}F_2$$
$$= \overline{M}(F_1 + F_2 + F_3)$$

Julia Abdallah

Computer Science Student at LU

Bekaa, Lebanon +961 70672268 juliaAbdallah4@gmail.com

Summary

I am a Computer Science student dedicated to my studies. Currently looking for a job that would allow me to further enhance my skills in Web Development, Software Engineering, Data Science, or any related fields.

EDUCATION

Computer Engineering | *Lebanese University Faculty of Engineering*
2020 – 2023

Bachelor's in Computer Science | *Lebanese University Faculty of Sciences*
2023 – Present

EXPERIENCE

Private Math, Physics, and Chemistry Teacher
Feb 2019 – Oct 2019

Coding Instructor | *BrightChamps*
Jan 2022 – Jan 2024

Team Lead | *BrightChamps*
July 2022 – Jan 2024

HR Manager | *BrightChamps*
June 2023 – August 2023

SKILLS

- **Programming:** C, C++, Java, C#, HTML, CSS, JavaScript, Python
- **Software:** Office, AutoCAD, MATLAB
- **Languages:** English (fluent), French (fluent), Arabic (native)
- **Teamwork:** Project management, conflict resolution, leadership
- **Communication:** Public speaking, presentation, adaptability