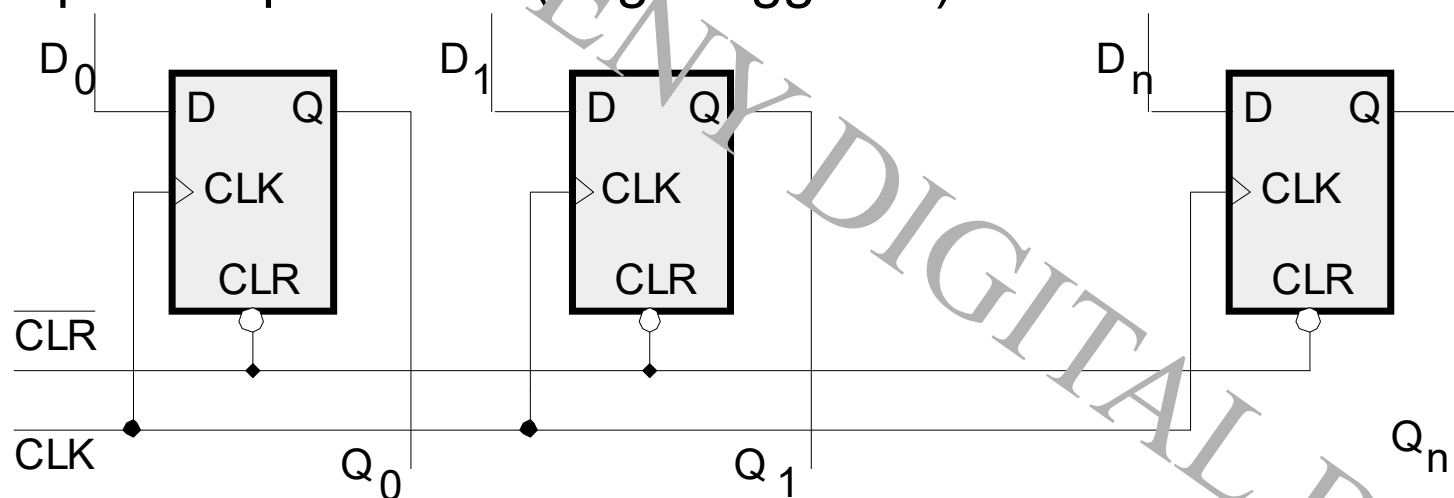

Introducció als Ordinadors:

Capítol 2:

REGISTRES DE LA CPU

Registres

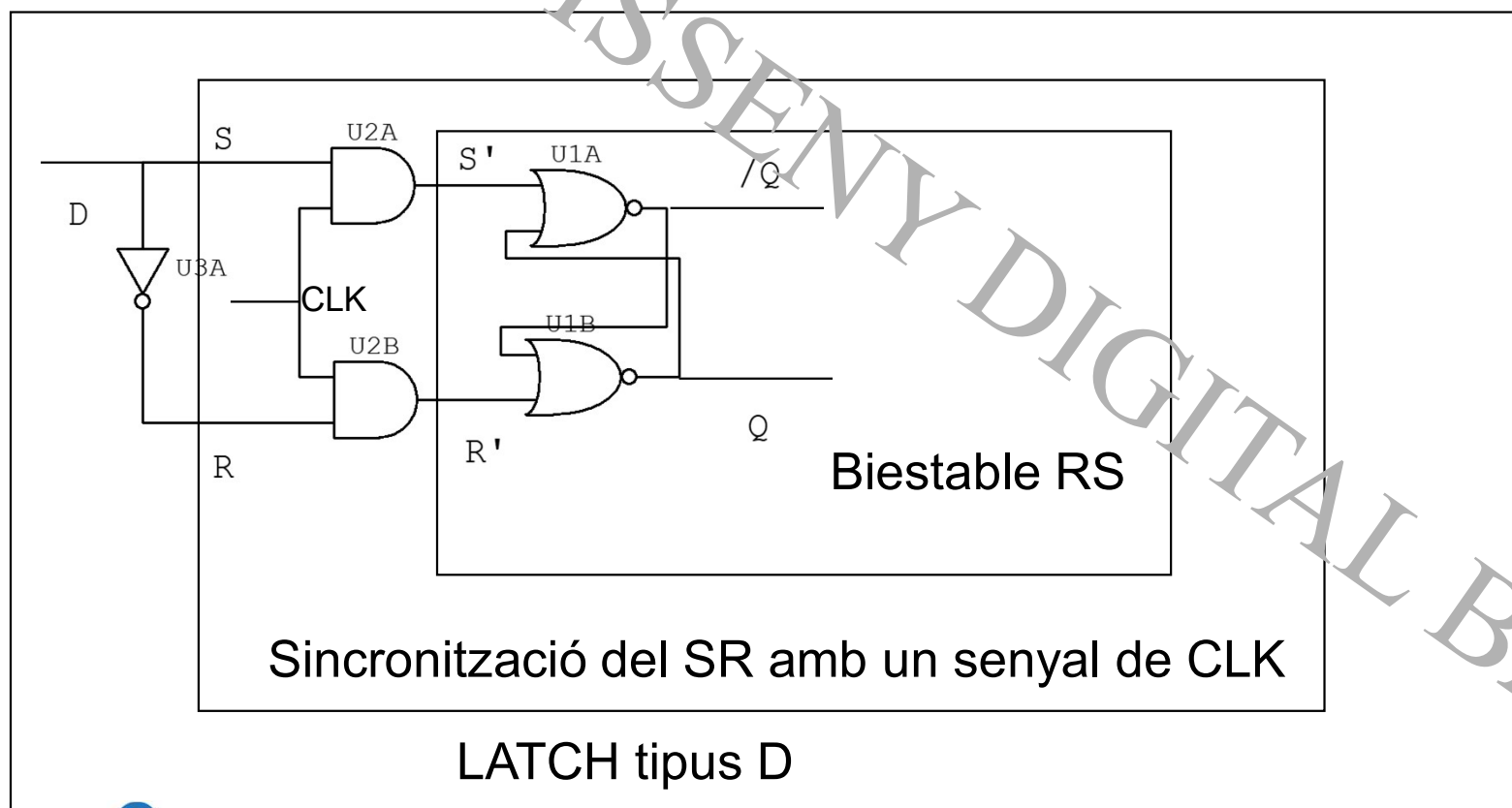
- Conjunt d'elements biestables governats per el mateix clock.
- Un registre és una memòria d'alta velocitat i poca capacitat integrada en el processador que permet guardar temporalment i accedir a dades
- Sincronisme:
- Latch: actiu per nivells (no pot utilitzar-se per operacions del tipus $R_j \leftarrow f(R_j, R_i)$)
- Flip-Flop: actiu per flancs (edge-triggered)



n elements = n bits

LATCH

- Quan el senyal de rellotge està habilitat, l'entrada D passa a la sortida Q
- Es fonamenta en un biestable SR, sincronitzat amb un senyal de rellotge



| Clk | D | Q |
|-----|---|---|
| 0 | 0 | Q |
| 0 | 1 | Q |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

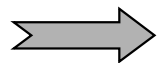
Registres

Flip-flop actiu per flanc (edge-triggered)

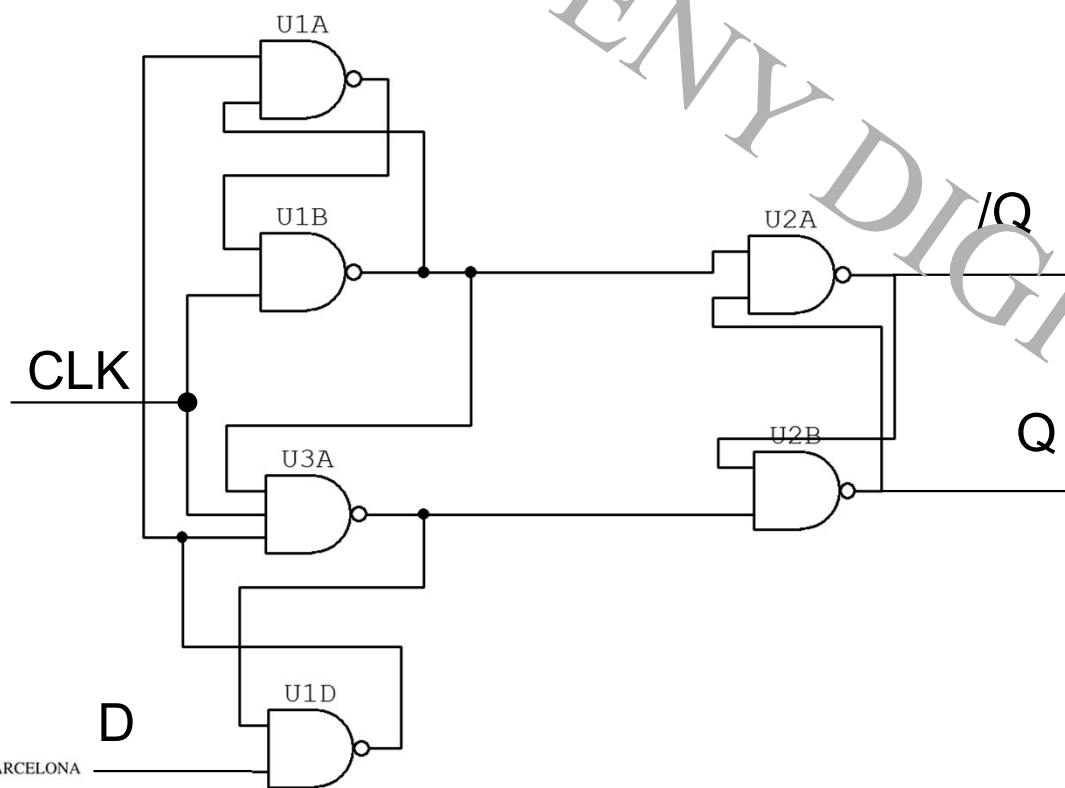
Només permet els canvis durant el temps de transició del senyal de clk

-elimina sorolls

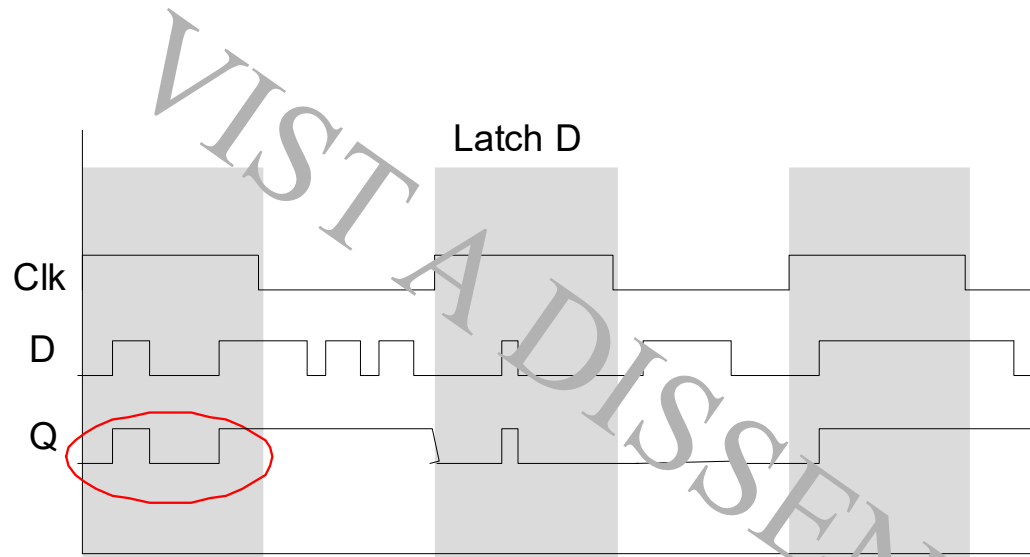
-elimina problemes de transicions en els flip-flops



El canvi es produeix en la transició 0-1 (rise time) o 1-0 (fall time)

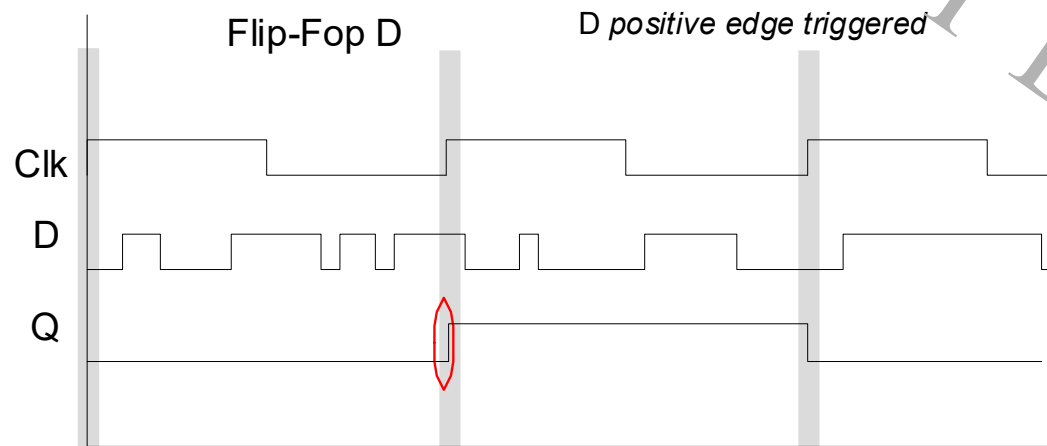


Latch vs. Flip-Flop



Latch

- Actiu per nivells
- Útil per guardar
- No útil per Reg.Desplaçament
- Asincrònic
- No útil per: font i destí alhora



Flip-Flop

- Actiu per canvis de nivells
- Útil per guardar
- Útil per Reg.Desplaçament
- sincrònic

Nivell RTL (*Register Transfer Level*)

Disseny de sistemes digitals **sincrònics**

Les operacions són **RTL**

$$R_i \leq R_j * R_k$$

Registre destí

Registres font

Indica **transferència**, que es produeix quan arriba el flanc actiu del **clk**

$$F = A * B$$

Indica funció combinacional. El valor és estable quan les 2 variables ho són (retard)

Exemple

Desenvolupament de un registre en **System Verilog**

```
module Registre( input logic clk, reset, en,  
                 input logic [7:0] d,  
                 output logic [7:0] q);  
    always_ff@(posedge clk)  
        if(reset) q<=8'b0;  
        else if(en) q<=d;  
endmodule
```

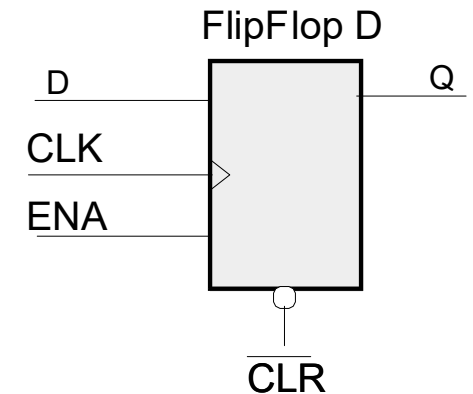
Desenvolupament d'un circuit sumador en **System Verilog**

```
module fulladder(input logic a, b, cin,  
                 output logic s, cout);  
    logic p, g;  
    always_comb  
        begin  
            p = a^b;  
            g = a&b;  
            s = p^cin;  
            cout = g|(p&cin);  
        end  
endmodule
```

Senyals síncrons d'habilitació i RESET

Introduïm un senyal d'habilitació (ENABLE) active high

Introduïm un senyal de clear (RESET) active low



Hem de generar la funció que ens doni l'entrada D del flip flop en funció de les variables EN, /CLR i DI

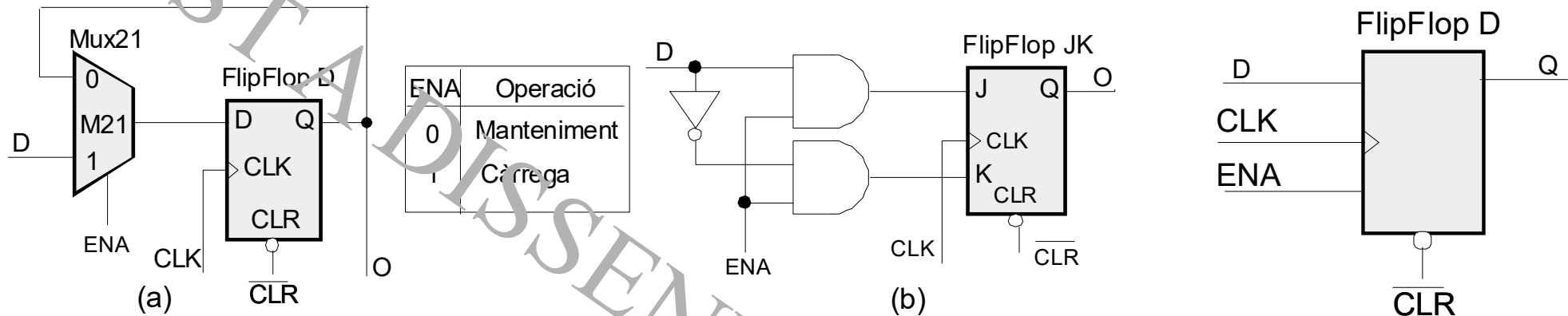
La funció és:

$$D = \neg EN \cdot \neg CLR \cdot Q + EN \cdot \neg CLR \cdot DI$$

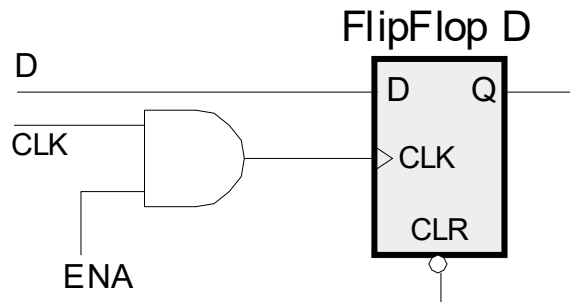
| EN | /CLR | DI | D |
|----|------|----|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | Q |
| 0 | 1 | 1 | Q |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Estructura dels Registres

Funcionament correcte: governat per senyal d'habilitació



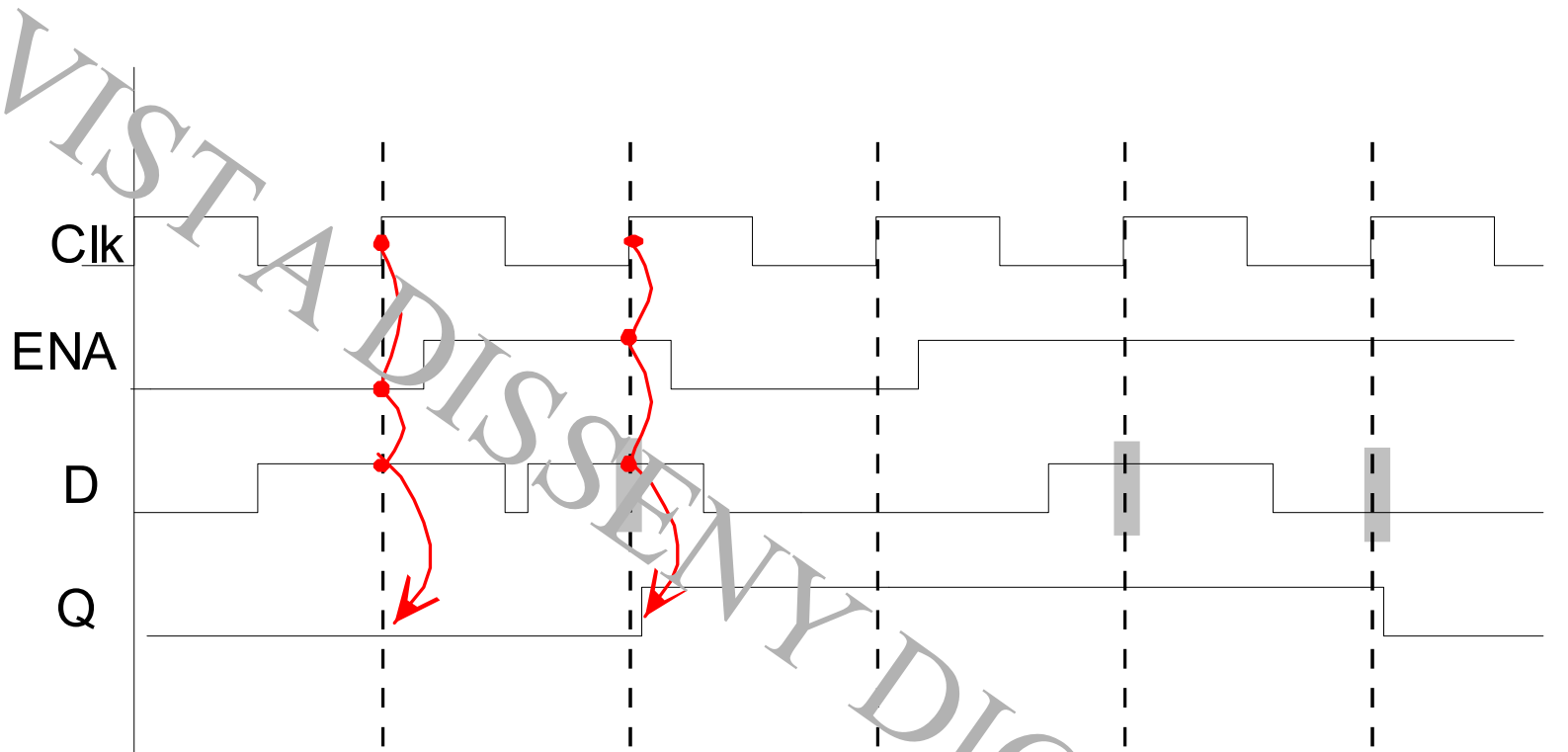
No aconsellable: habilitació actuant sobre el senyal de clk



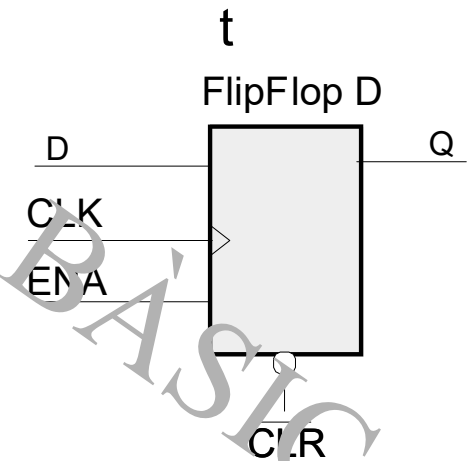
Gated clock

- Retard
- Pèrdua de sincronisme

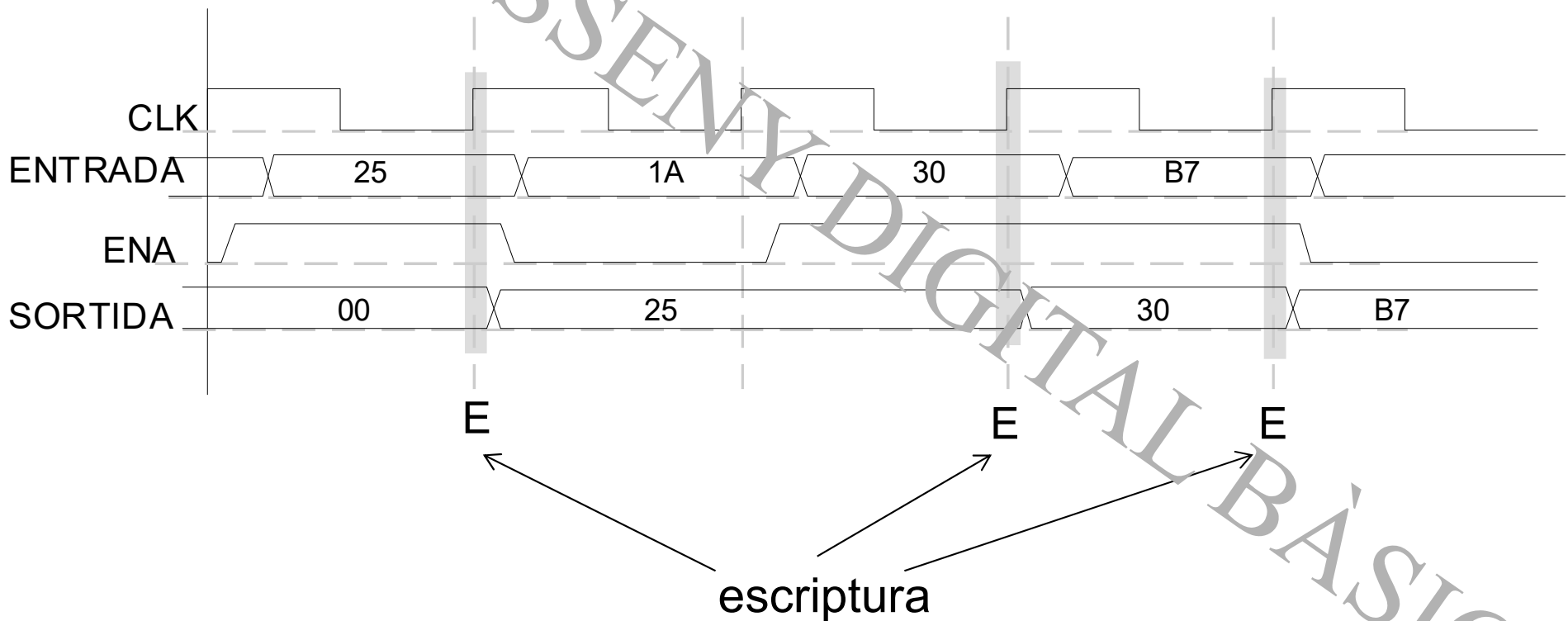
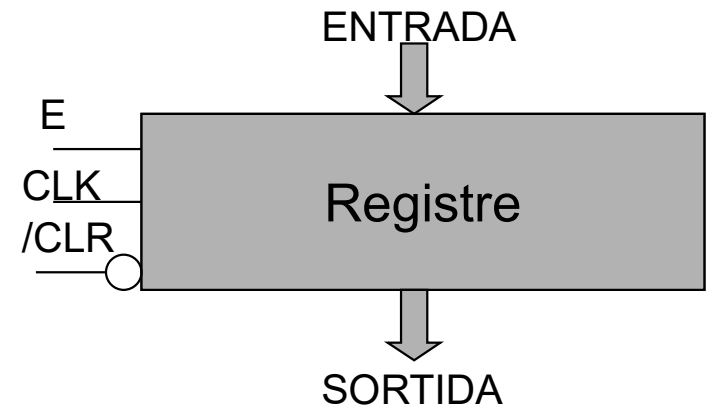
Sincronisme en Registres



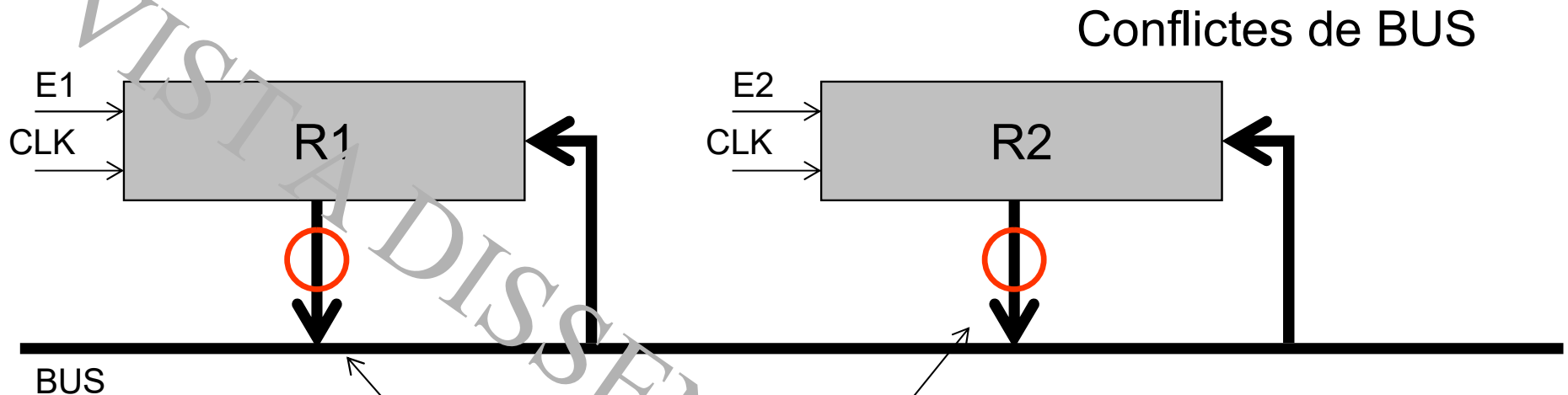
Registre governat per senyal d'habilitació (ENA)



L'ESCRITURA és SÍNCRONA
La LECTURA és ASÍNCRONA

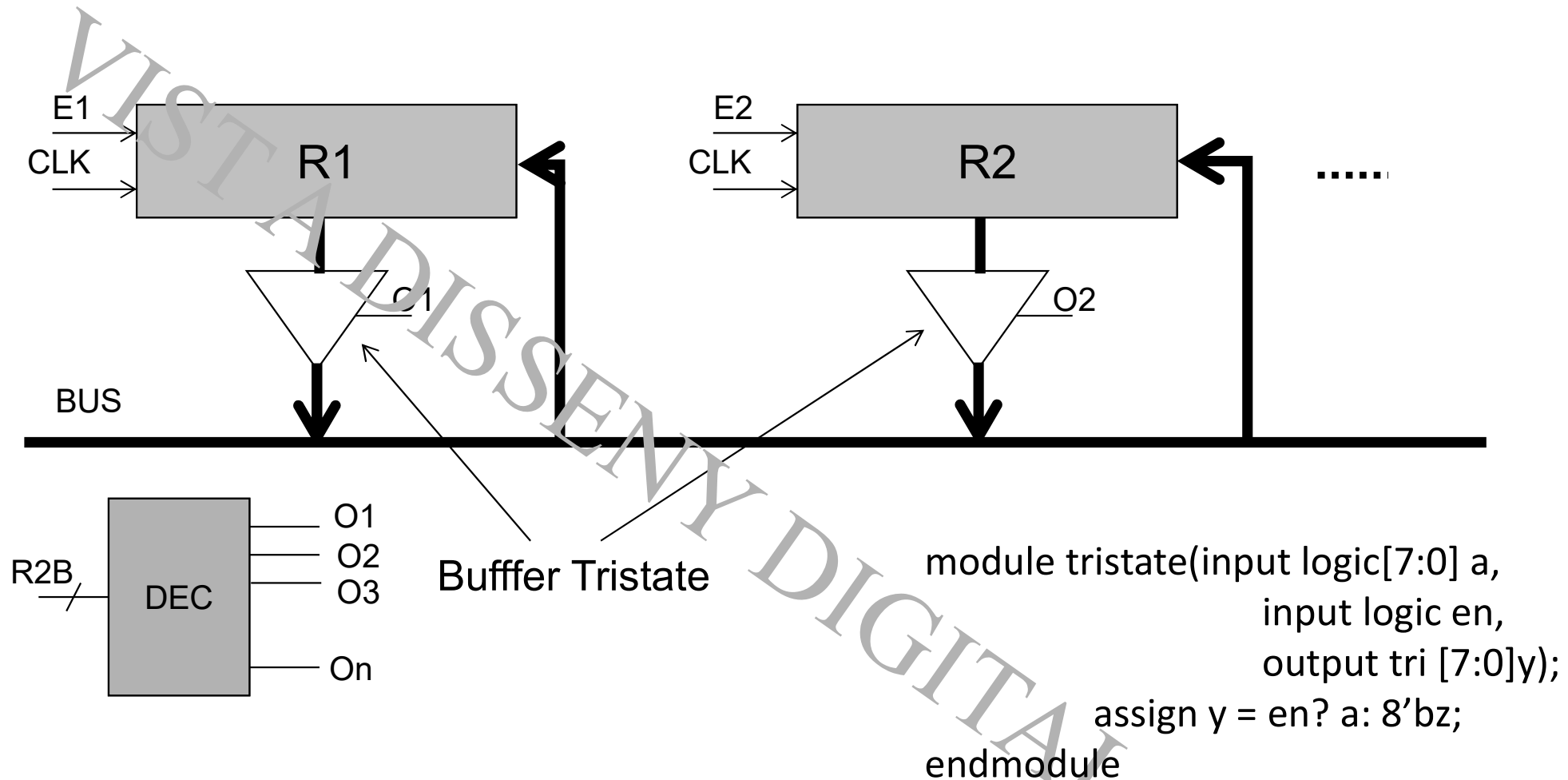


Lectura de REGISTRES

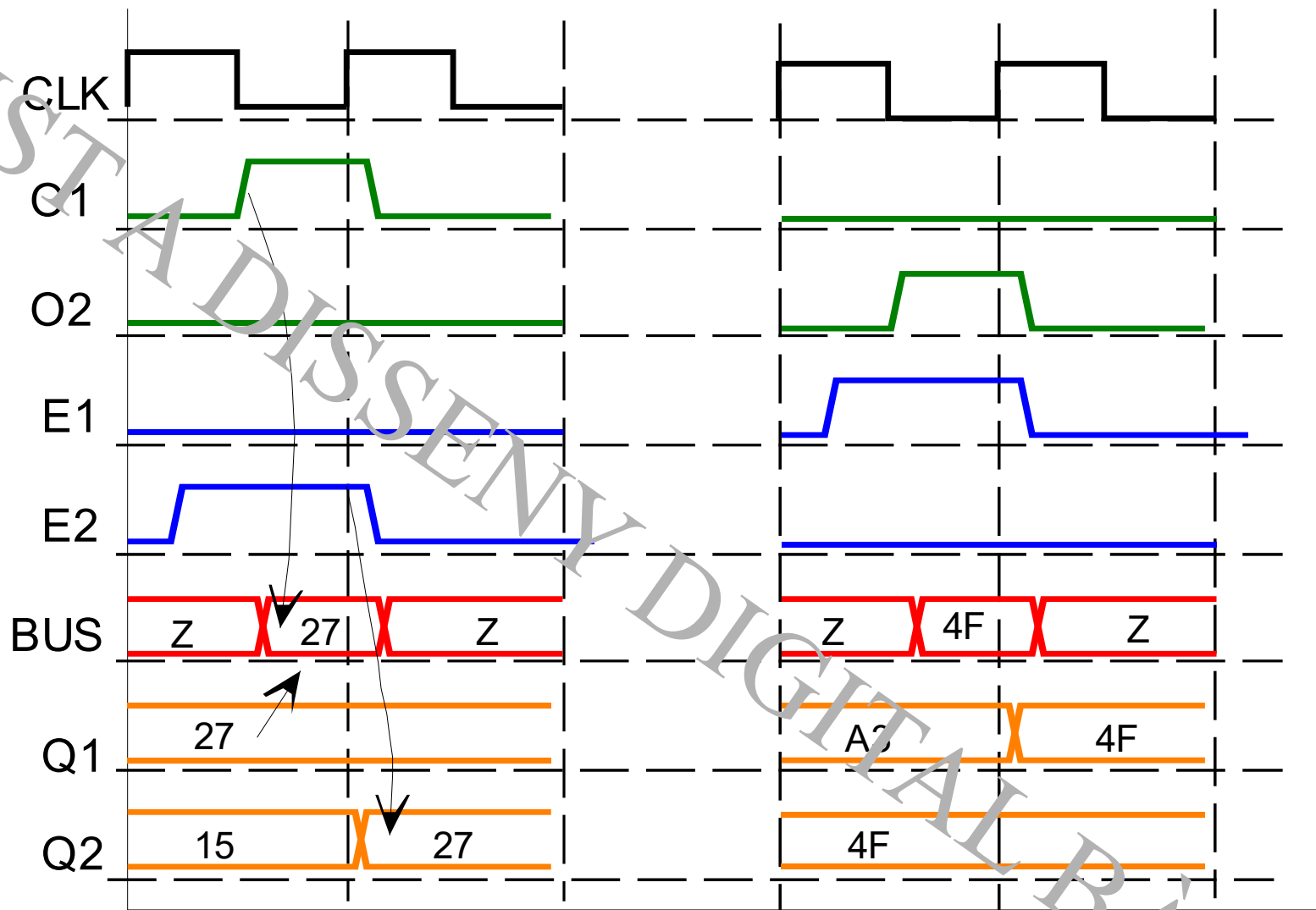


- L'escriptura està controlada per els senyals E_i .
- Però si es vol llegir de dos registres i passar les dades a BUS es produeix un **conflicte de BUS**
- Cal controlar l'accés a BUS

Multiplexat de les sortides dels registres (buffers-tristate)



Una alternativa és un MUX-n. Diferències de cost depenen del nombre de bits



$R2 \leq R1$

$R1 \leq R2$

Depenent de l'arquitectura del processador, es pot actuar sobre els registres directament. Per exemple...

SUB a1,a1,a1

ADDI a1,a1,8

ADD a1,zero,a2

...

On

ADD Rd, Rs1, Rs2

ADDI Rd, Rs1, Immediat

SUB Rd, Rs1, Rs2

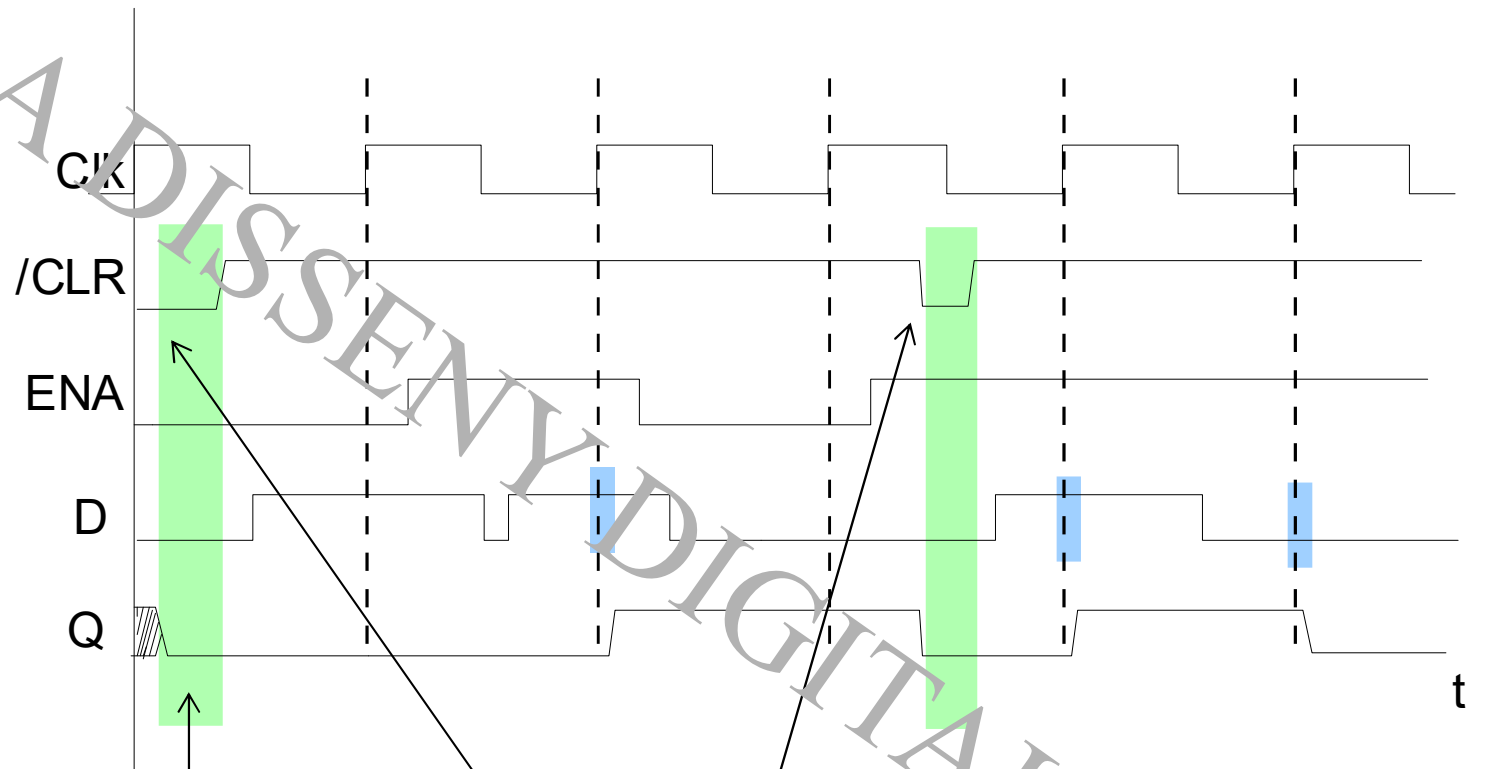
L'OPERACIÓ DE **LECTURA** DE
REGISTRES ÉS **ASÍNCRONA**
L'OPERACIÓ **D'ESCRITURA** ÉS
SÍNCRONA

El processador RISC-V el
registre x0 sempre conté un 0
No es pot modificar

RESET/CLEAR en Registres

Generalment els flip-flops consten de RESET (CLEAR) asíncron

El CLEAR sol
ser **active low**
(/CLR)



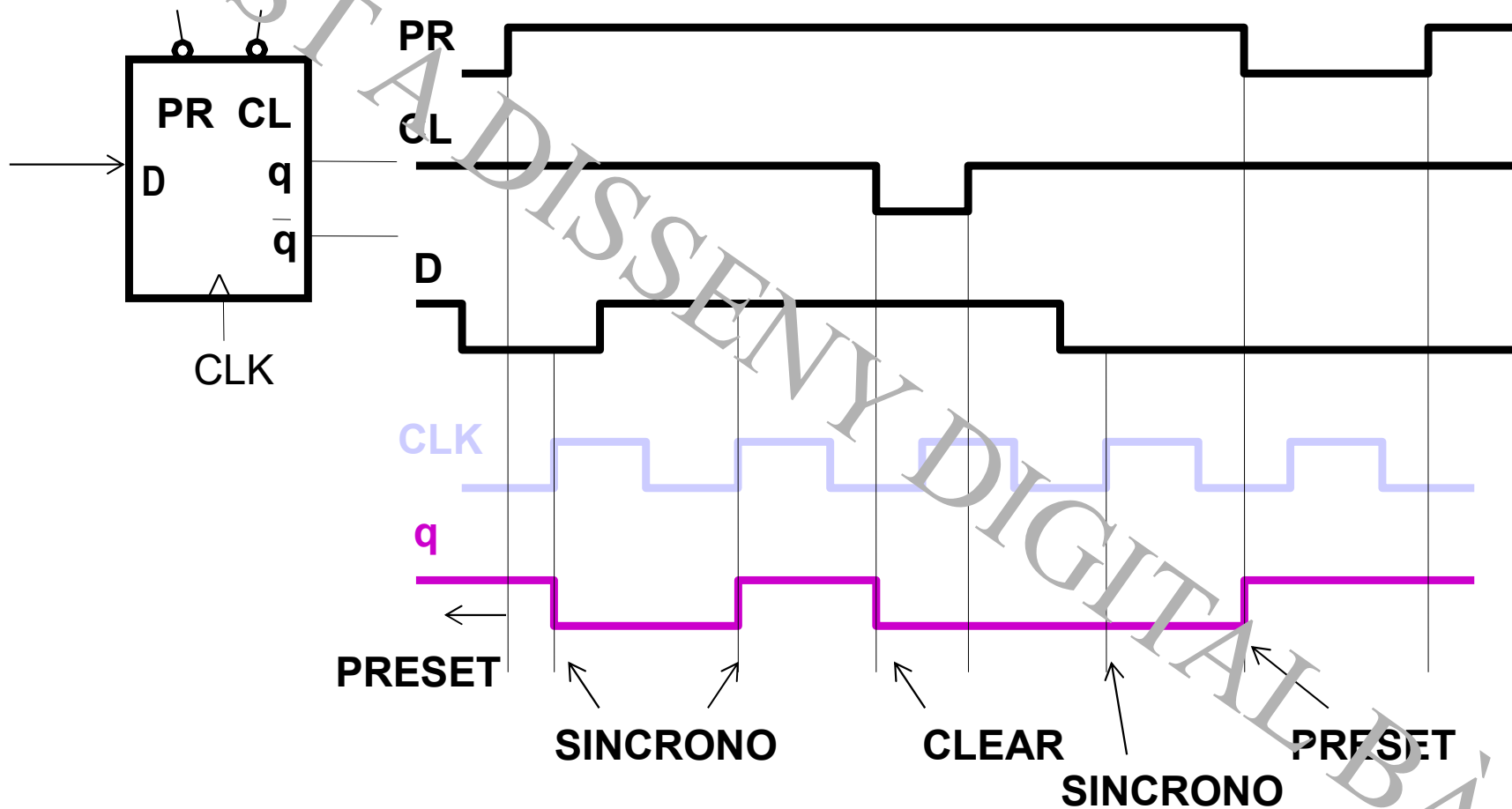
CLEAR asincrònic

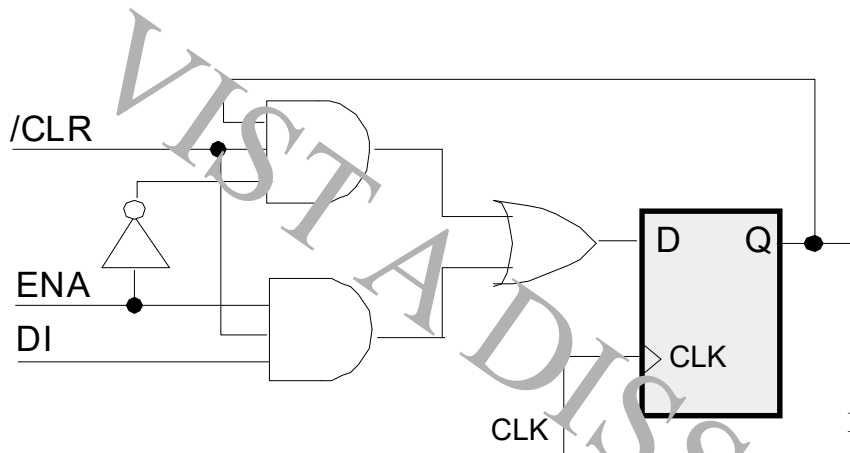
Aconsellable iniciar amb un CLEAR dels registres

De la mateixa manera pot haver-hi senyal de PRESET

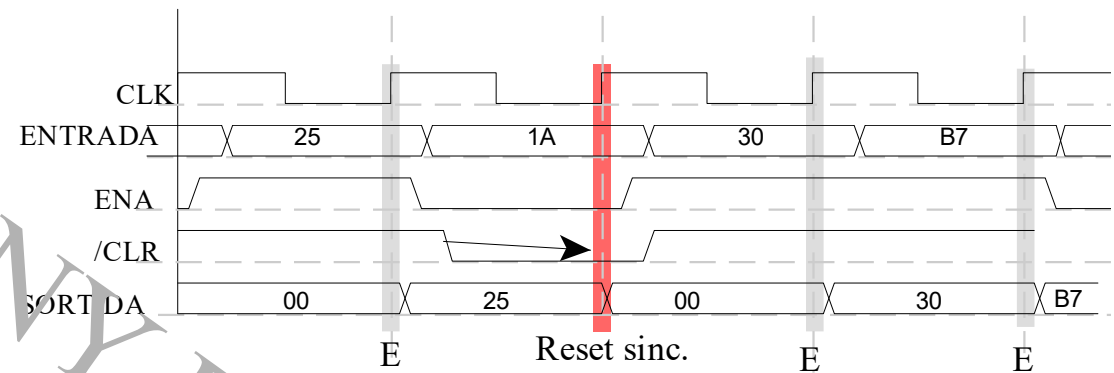
Elementos de memoria

Entradas asíncronas de los biestables



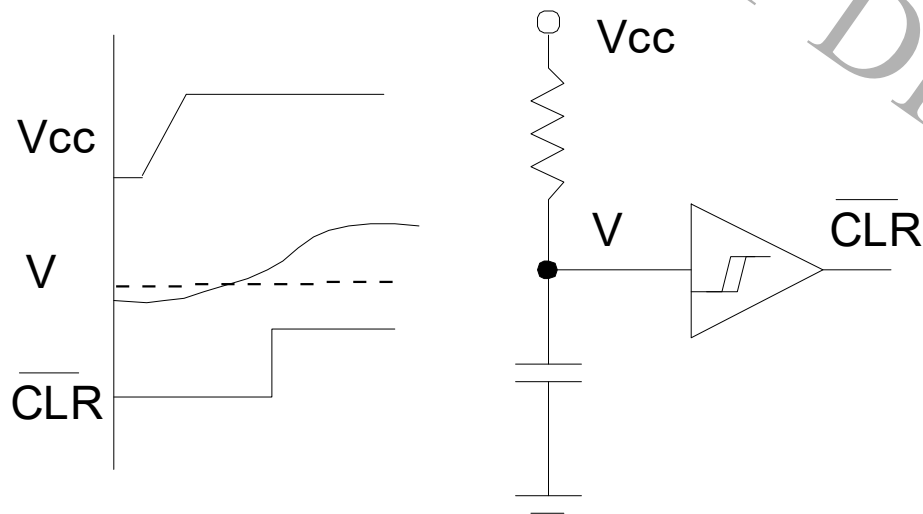


Alguns cops és aconsellable
usar un senyal de CLEAR
sincrònic



Power-On-Reset inicial

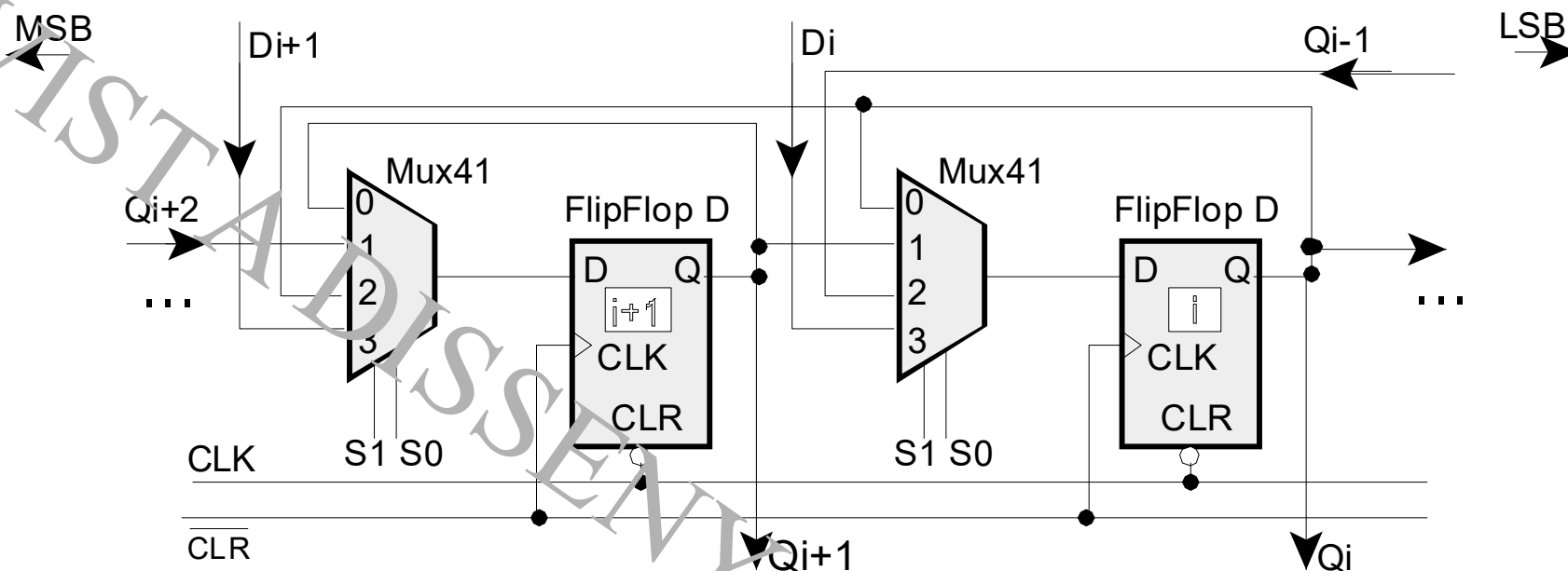
POWER-ON-RESET



Molts Microprocessadors porten
incorporat el Power-On-Reset

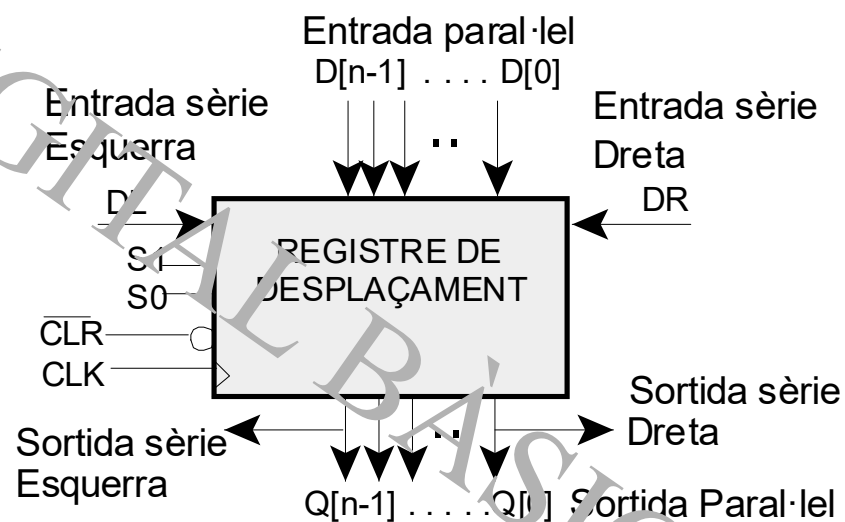
Registres de desplaçament

Cel·la bàsica

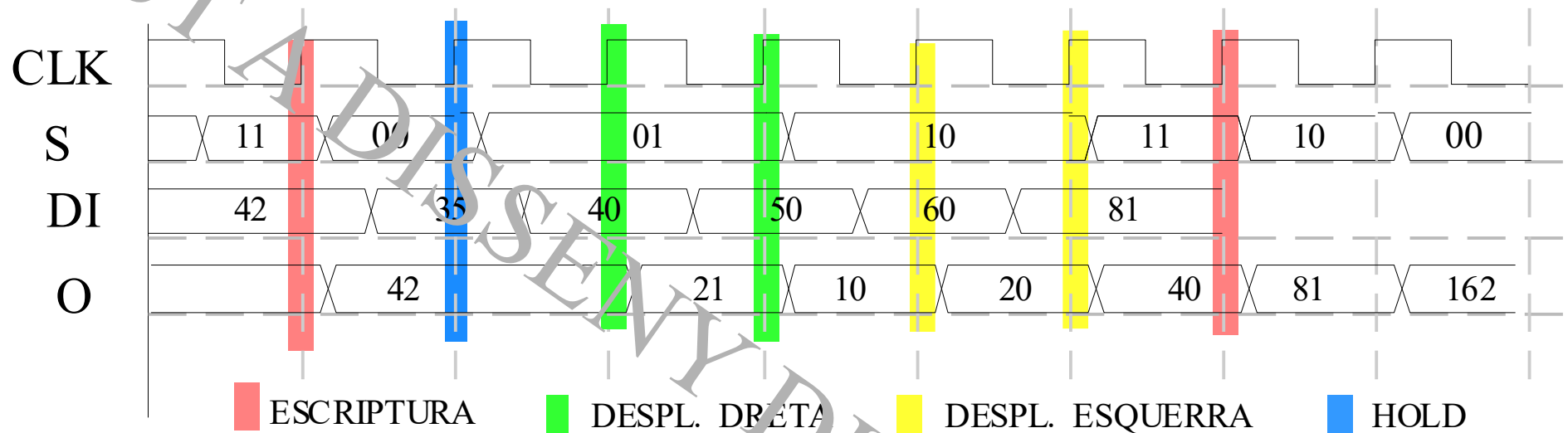


S0 i S1 són l'enable actuant al multiplexor

| Selecció S1 S0 | Operació | Senyals de Sortida Qn-1 Qn-2 Q1 Q0 |
|-------------------|---------------------|---|
| 0 0 | Mantenir HOLD | Qn-1 Qn-2 Q1 Q0 |
| 0 1 | Despl. Dreta SHR | DL Qn-1 ... Q2 Q1 |
| 1 0 | Despl. Esquerra SHL | Qn-2 Qn-3 Q0 DR |
| 1 1 | Càrrega LOAD | Dn-1 Dn-2 D1 D0 |



Funcionament del Registre de Desplaçament



| Selecció S1 S0 | Operació | | Senyals de Sortida Qn-1 Qn-2 Q1 Q0 | | | |
|-------------------|-----------------|------|---|------|-------|-------|
| 0 0 | Mantenir | HOLD | Qn-1 | Qn-2 | | Q1 Q0 |
| 0 1 | Despl. Dreta | SHR | DL | Qn-1 | ... | Q2 Q1 |
| 1 0 | Despl. Esquerra | SHL | Qn-2 | Qn-3 | | Q0 DR |
| 1 1 | Càrrega | LOAD | Dn-1 | Dn-2 | | D1 D0 |

Aplicació directa dels registres de desplaçament en ASM

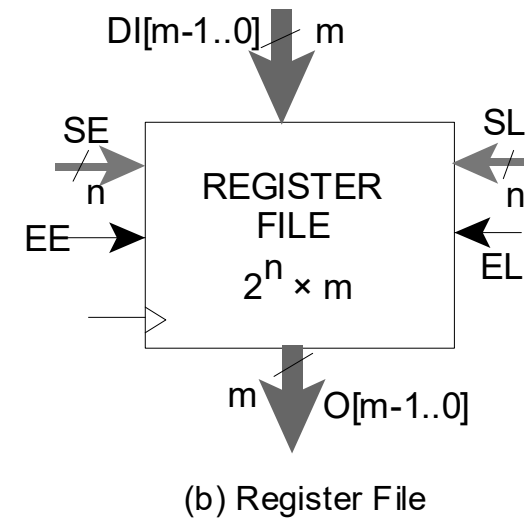
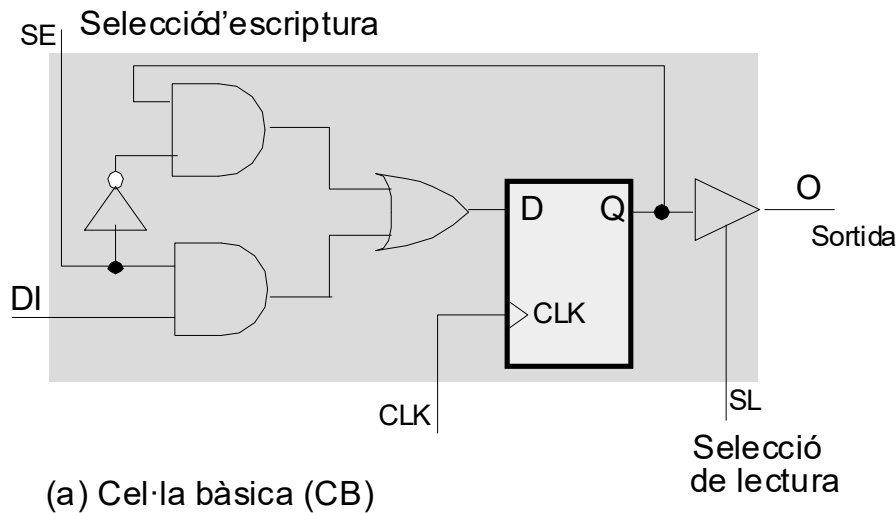
Instruccions de l'estil:

En RISC-V tenim per exemple

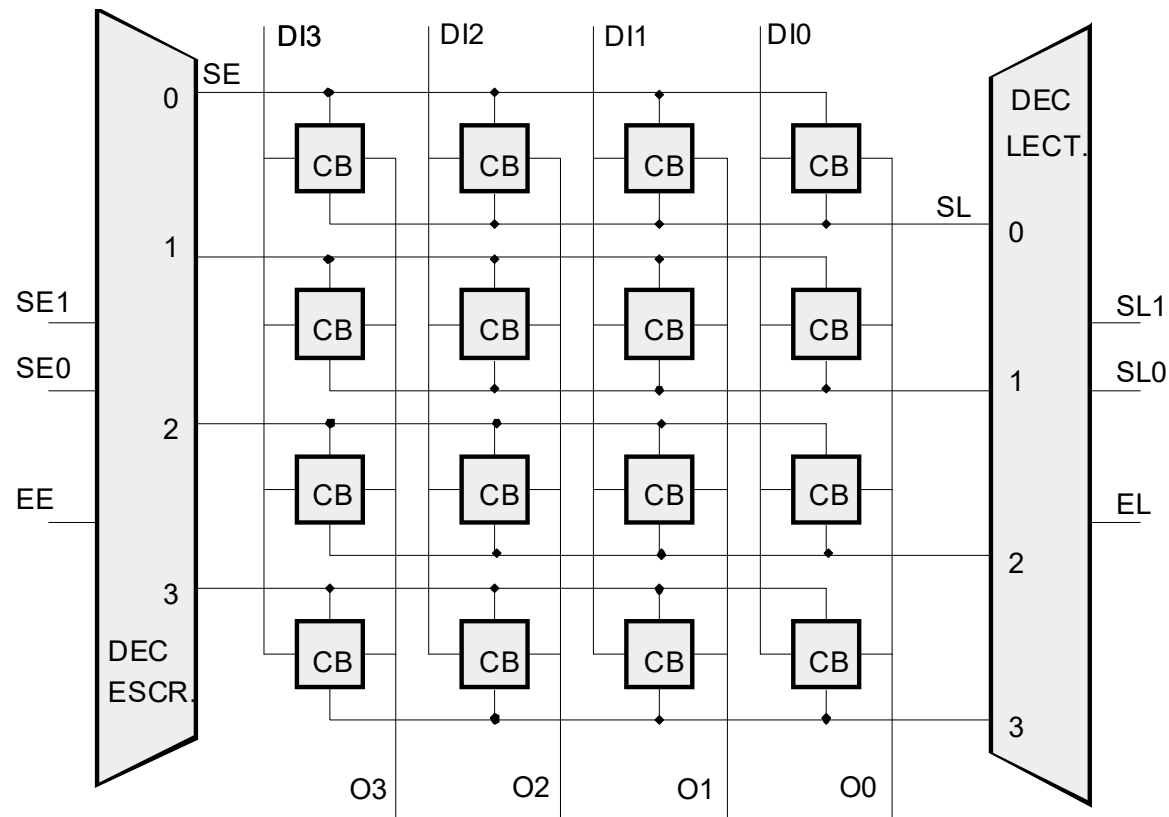
SLL Rd, Rs1, Rs2 => Desplaçament lògic de Rs1 [Rs2] bits
i es guarda en Rd.

SLLI Rd, Rs1, Imm => desplaçament lògic de Rs1 Imm bits

Conjunt de Registres (*Register File*)



CR

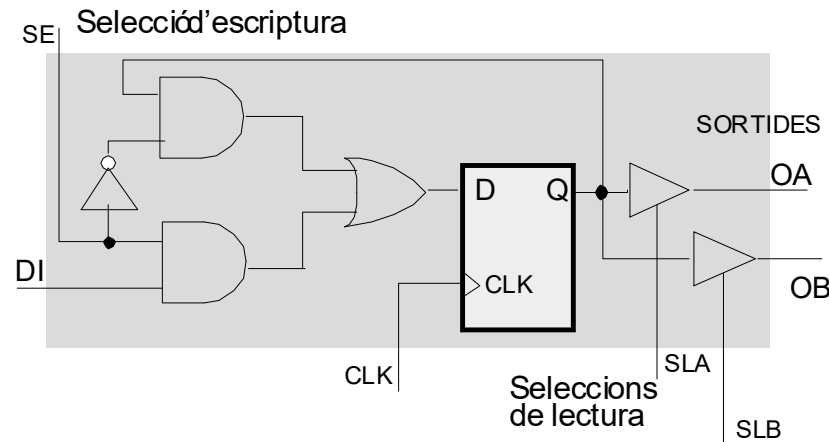


Conjunt de Registres de doble port

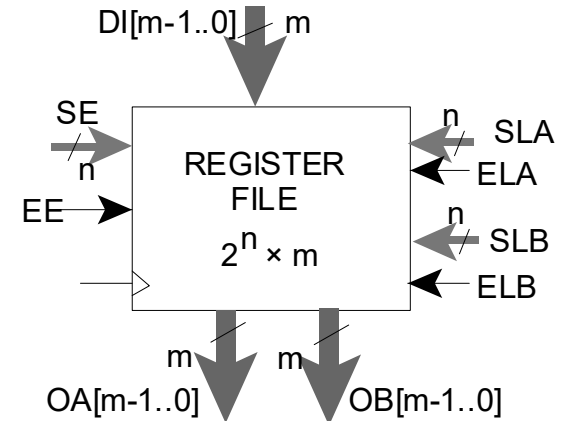
Dual-Port RF

Operacions

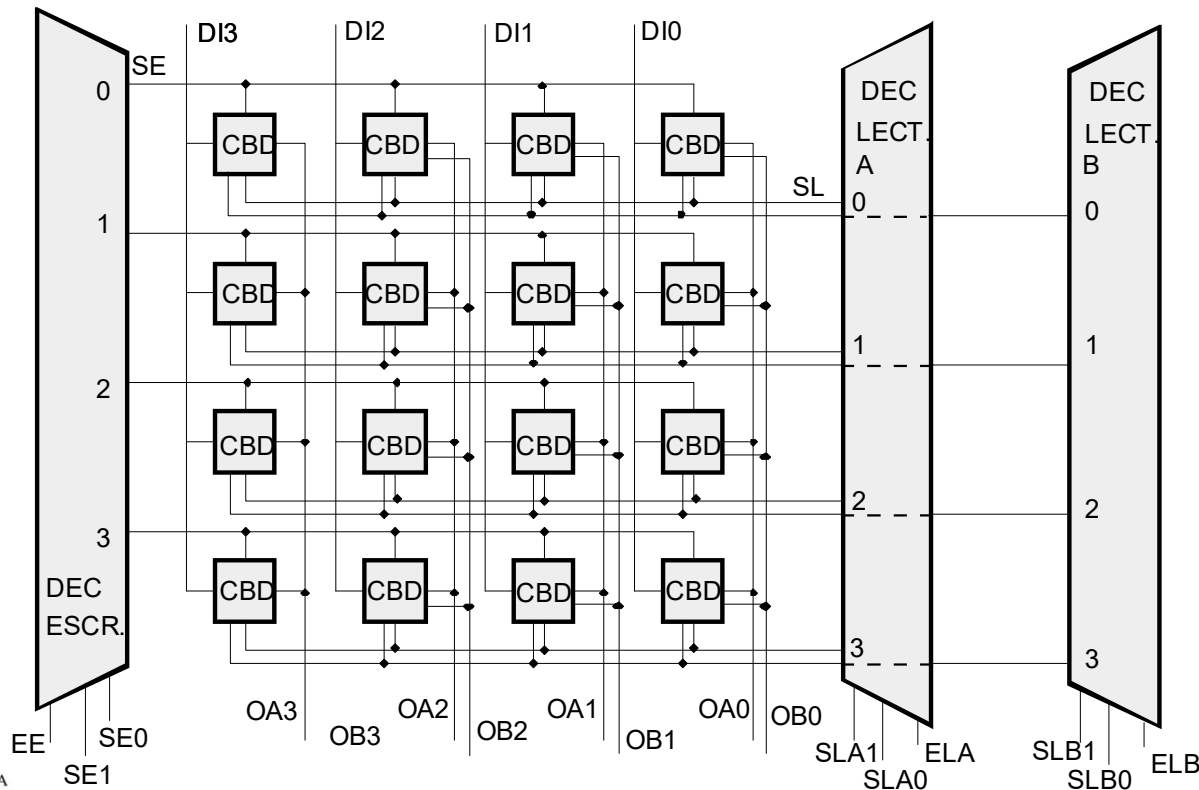
$$R_i \leftarrow R_j * R_k$$



(a) Cel·la bàsica (CBD) doble port de lectura



(b) Register File de doble port de lectura



Límits:

Àrea del CR creix
aprox com el $\#ports^2$

Temps d'accés creix
lineal. amb el $\#ports$

Registres de la CPU

- Tenim dos tipus de registres:

- Registres de propòsit general:

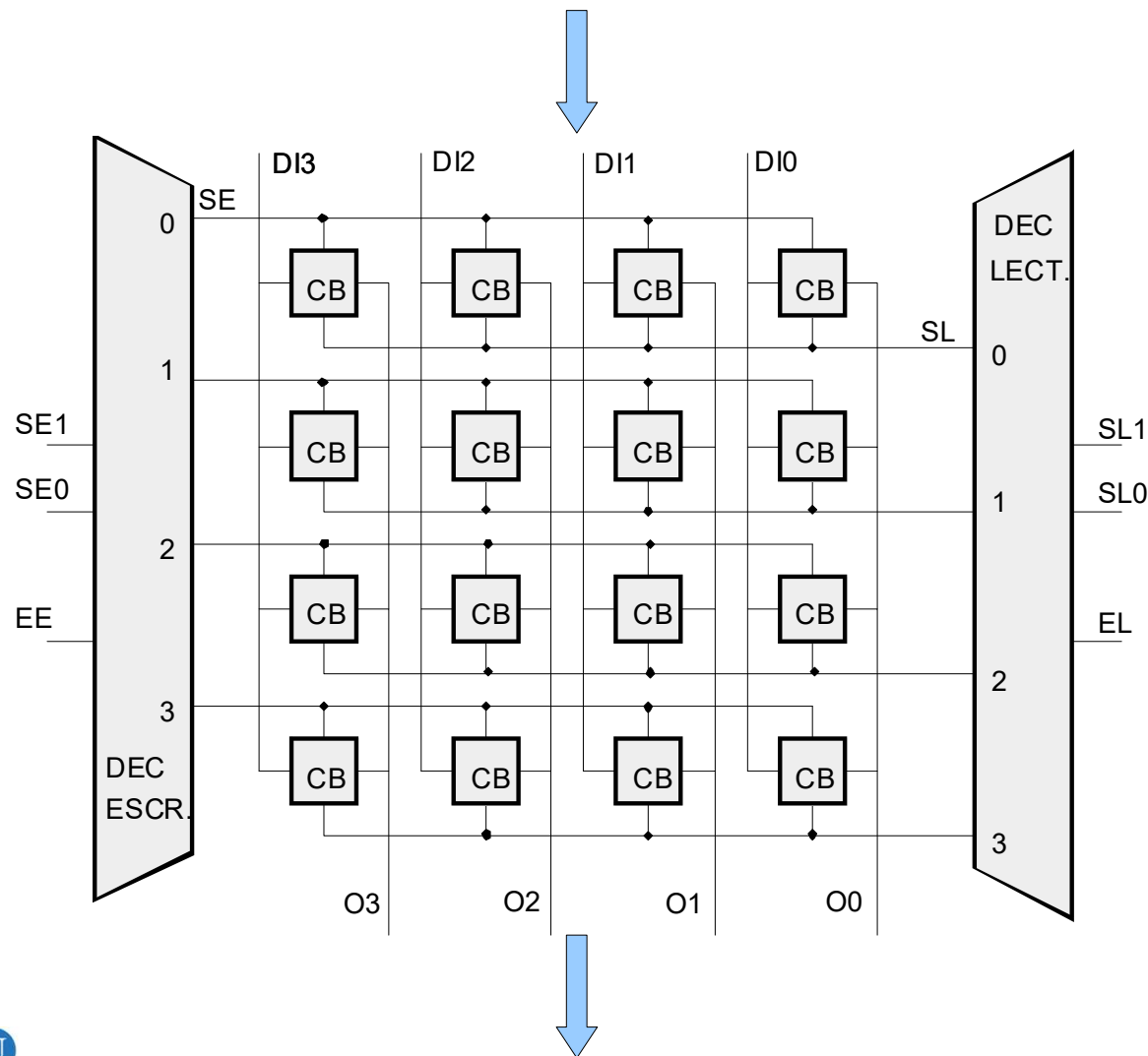
- GPR (General Purpose Registers). Poden guardar qualsevol tipus de dada o adreça de memòria. Fonamentals en l'arquitectura Von Neumann, poden ser accessibles al programador o no.

- Registres de propòsit específic:

- Guarden informació específica de l'estat del sistema. No són accessibles al programador (a nivell d'escriptura, però si a nivell de lectura)

Registres de la CPU

- Registres de propòsit general (ja vist)



Bus de sistema

Bus de sistema

•Conjunt de registres RISC-V

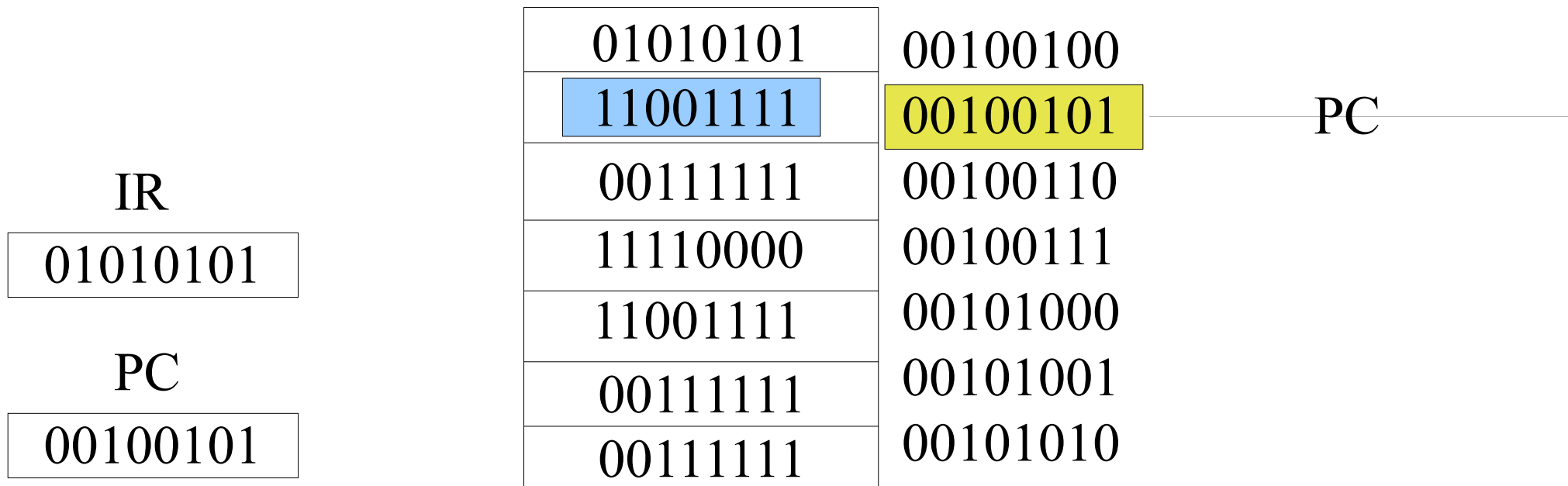
| | |
|--------------|--|
| 31 | 0 |
| x0 / zero | Alambrado a cero |
| x1 / ra | Dirección de retorno |
| x2 / sp | Stack pointer |
| x3 / gp | Global pointer |
| x4 / tp | Thread pointer |
| x5 / t0 | Temporal |
| x6 / t1 | Temporal |
| x7 / t2 | Temporal |
| x8 / s0 / fp | Saved register, frame pointer |
| x9 / s1 | Saved register |
| x10 / a0 | Argumento de función, valor de retorno |
| x11 / a1 | Argumento de función, valor de retorno |
| x12 / a2 | Argumento de función |
| x13 / a3 | Argumento de función |
| x14 / a4 | Argumento de función |
| x15 / a5 | Argumento de función |
| x16 / a6 | Argumento de función |
| x17 / a7 | Argumento de función |
| x18 / s2 | Saved register |
| x19 / s3 | Saved register |
| x20 / s4 | Saved register |
| x21 / s5 | Saved register |
| x22 / s6 | Saved register |
| x23 / s7 | Saved register |
| x24 / s8 | Saved register |
| x25 / s9 | Saved register |
| x26 / s10 | Saved register |
| x27 / s11 | Saved register |
| x28 / t3 | Temporal |
| x29 / t4 | Temporal |
| x30 / t5 | Temporal |
| x31 / t6 | Temporal |
| 32 | |
| 31 | 0 |
| pc | |
| 32 | |

Registres de la CPU

- Registres específics

- PC (Program Counter). Ens indica la posició de memòria on es troba la següent instrucció a executar. És un punter que ens indica l'adreça de memòria de la instrucció, NO EL CONTINGUT

- IR (instrucció Register). Registre de propòsit específic on es guarda el CONTINGUT de l'adreça de memòria on apuntava el PC



Registres de la CPU

- Registres específics (II)

- ACC (Acumulador). És un registre on es guarda el resultat de qualsevol operació que realitza la ALU (Unitat Aritmètic-lògica)

- Pot ser explícit o implícit. En aquest darrer cas es fa servir com ACC un registre de propòsit general

- STAT (Reg. d'estat). És el registre on s'indiquen les incidències que es tenen al fer una determinada operació (en risc-v mstatus)

Exemple:

El contingut del registre R1 és 010011

El contingut del registre R2 és 100110

Fem $R1 + R2 \longrightarrow ACC \leq R1 + R2$
 $ACC = 111001 = -7d$
 $STAT \Rightarrow N = 1, OV = 0, C = 0, Z = 0$

Registres de la CPU

- Registres específics

- Mie (Machine interrupt enable register). Habilita les interrupcions a nivel de màquina

- MAR (Memory Address Register) Registre de adreces de memòria. Es fa servir de buffer per anar a una determinada posició de programa. Es el valor que es carrega al bus d'adreces

Es un

ria
ades

Registres de la CPU

- Registres específics

- SP (Stack Pointer) Registre que ens indica la posició de l'últim valor guardat a la PILA.

- GP (global pointer) Registre per accedir a dades globals i variables estàtiques

@adreça 6

SP

| |
|-----------------|
| |
| |
| |
| #valor 5 |
| #valor 4 |
| #valor 3 |
| #valor 2 |
| #valor 1 |

@adreça 8

@adreça 7

@adreça 6

@adreça 5

@adreça 4

@adreça 3

@adreça 2

@adreça 1

Registres de la CPU

- Registres específics

- TP (Thread Pointer) Registre utilitzat per la gestió d'informació relacionada amb threads (sistemes que soporten multifil o multitasking).

- FP (Frame pointer) Control del marc de pila. Defineix l'adreça base de pila en l'execució d'una subrutina. *El FP s'utilitza per accedir de manera més eficient a les variables locals i als paràmetres d'una funció dins de la pila. Mentre que el Stack Pointer (SP) apunta al cim de la pila (la part més alta), el Frame Pointer (FP) apunta a un punt fix dins del marc de pila de la funció actual, cosa que facilita l'accés als dades locals i paràmetres de la funció de manera més directa.*

@adreça 6

SP

| |
|-----------------|
| |
| |
| |
| #valor 5 |
| #valor 4 |
| #valor 3 |
| #valor 2 |
| #valor 1 |

@adreça 8

@adreça 7

@adreça 6

@adreça 5

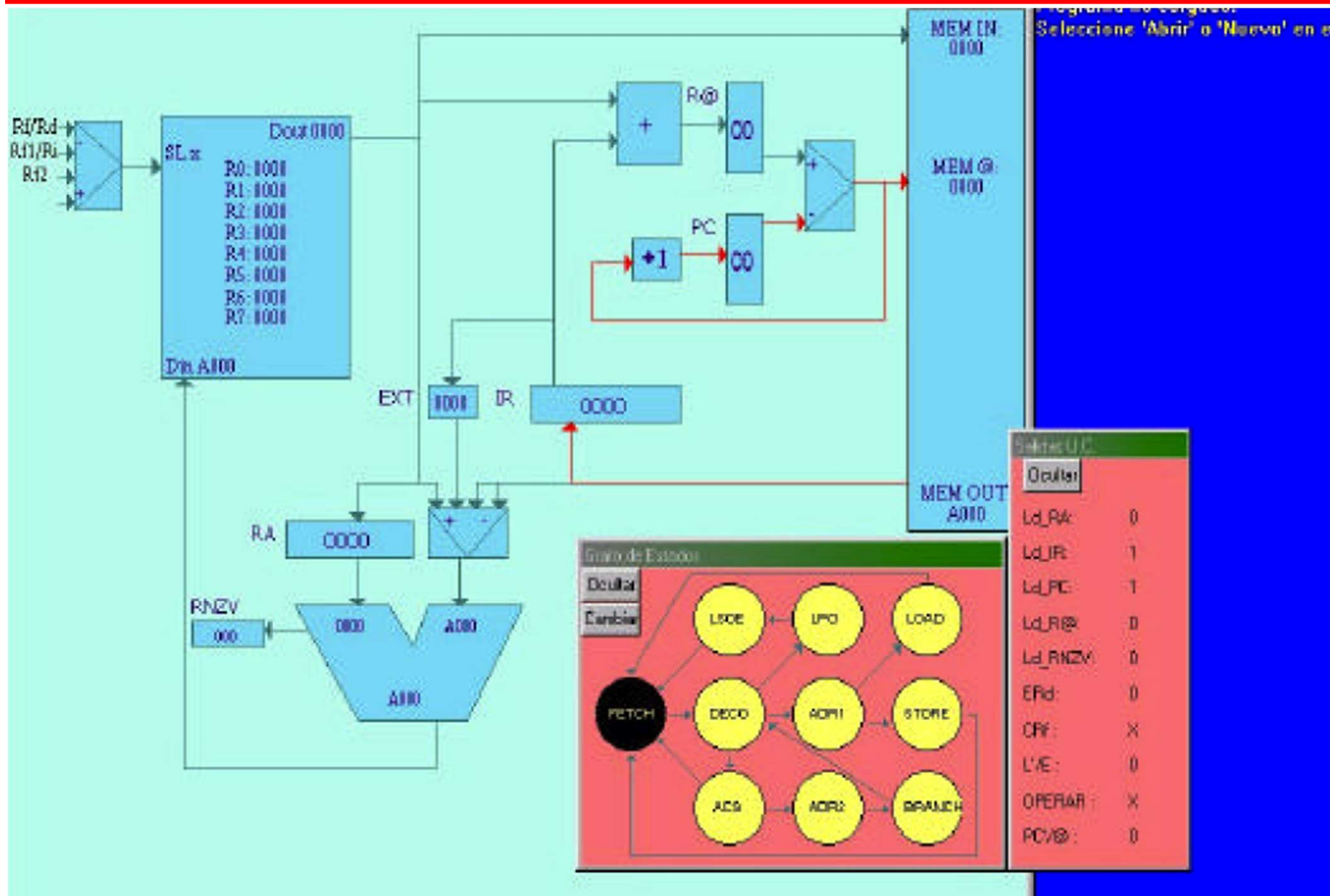
@adreça 4

@adreça 3

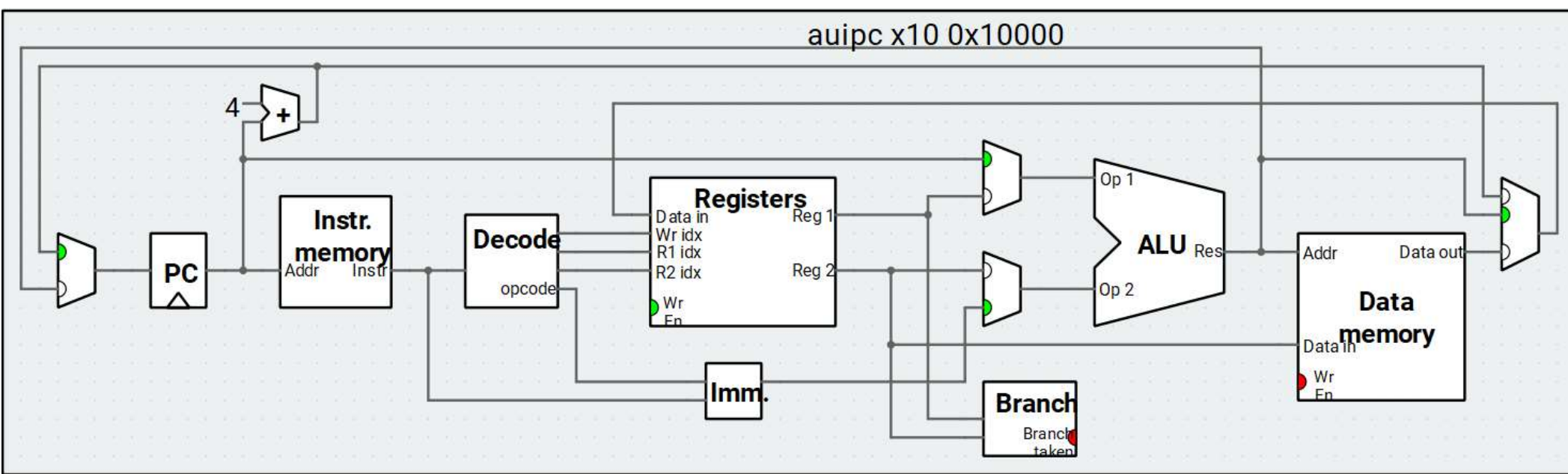
@adreça 2

@adreça 1

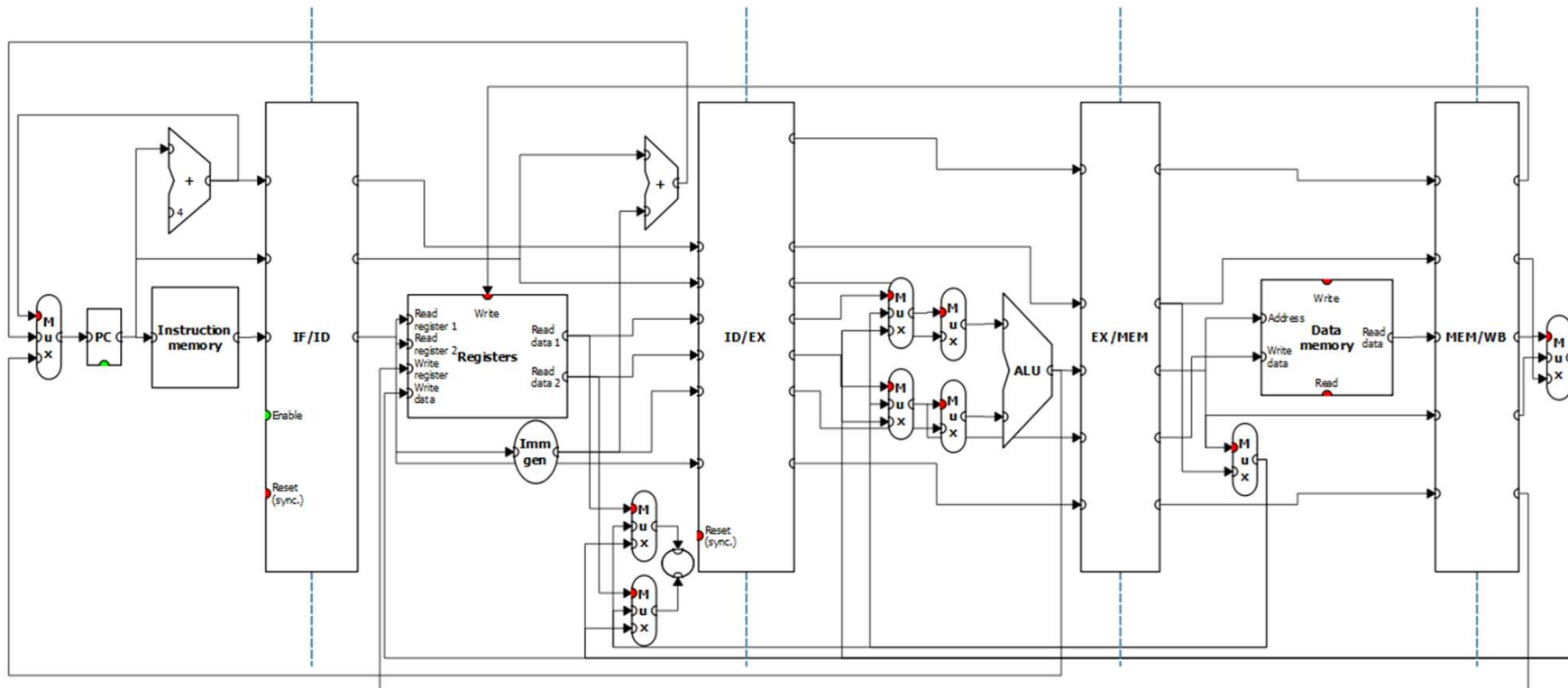
Registres de la CPU: Exemple SiMR



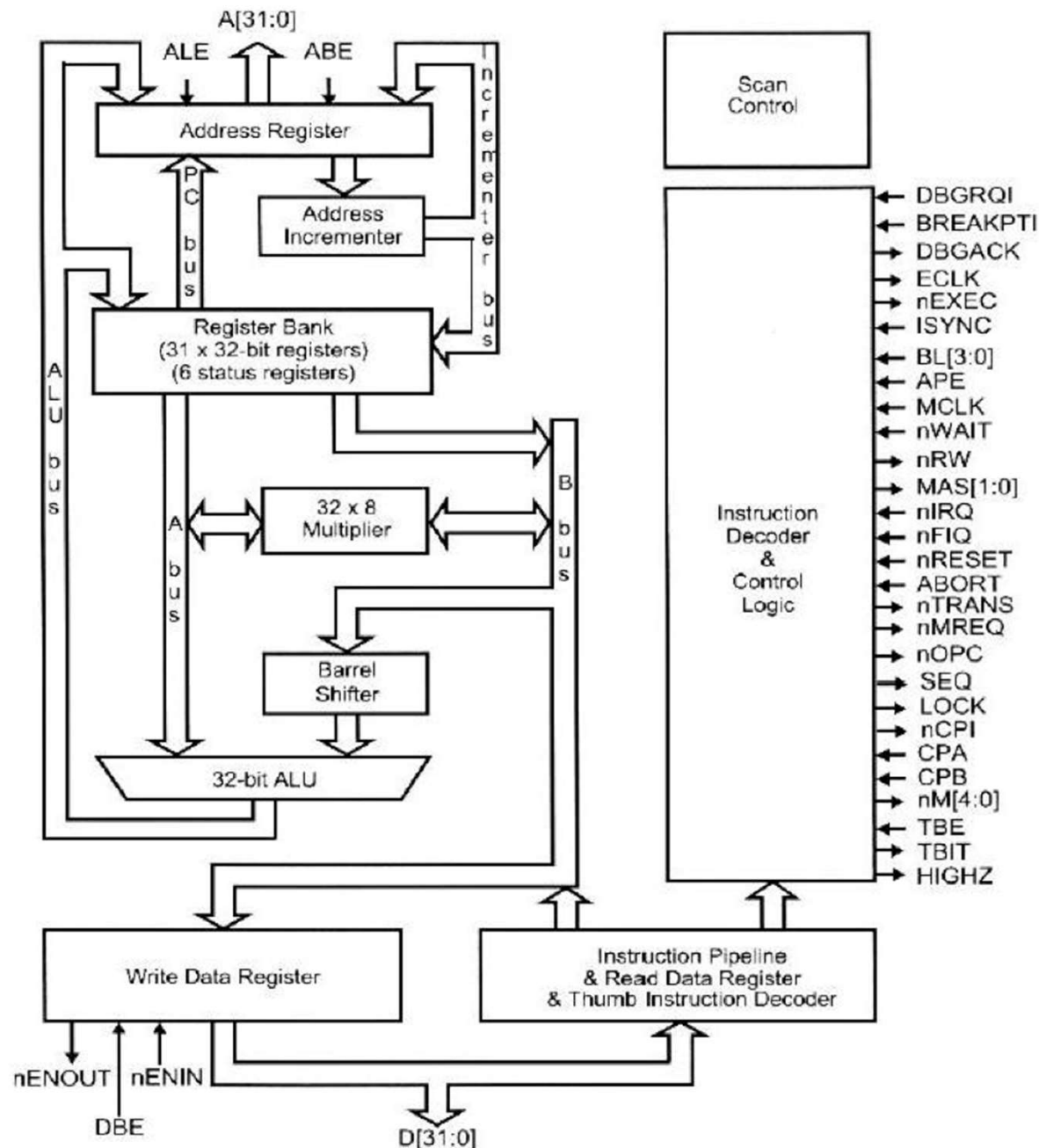
Registres de la CPU: Exemple arquitectura RISC-V



Registres de la CPU: Exemple arquitectura RISC-V



Registres de la CPU: Exemple Arquitectura ARM



Registres de la CPU: Arquitectura IA-32



PENTIUM® PROCESSOR 75/90/100/120/133/150/166/200

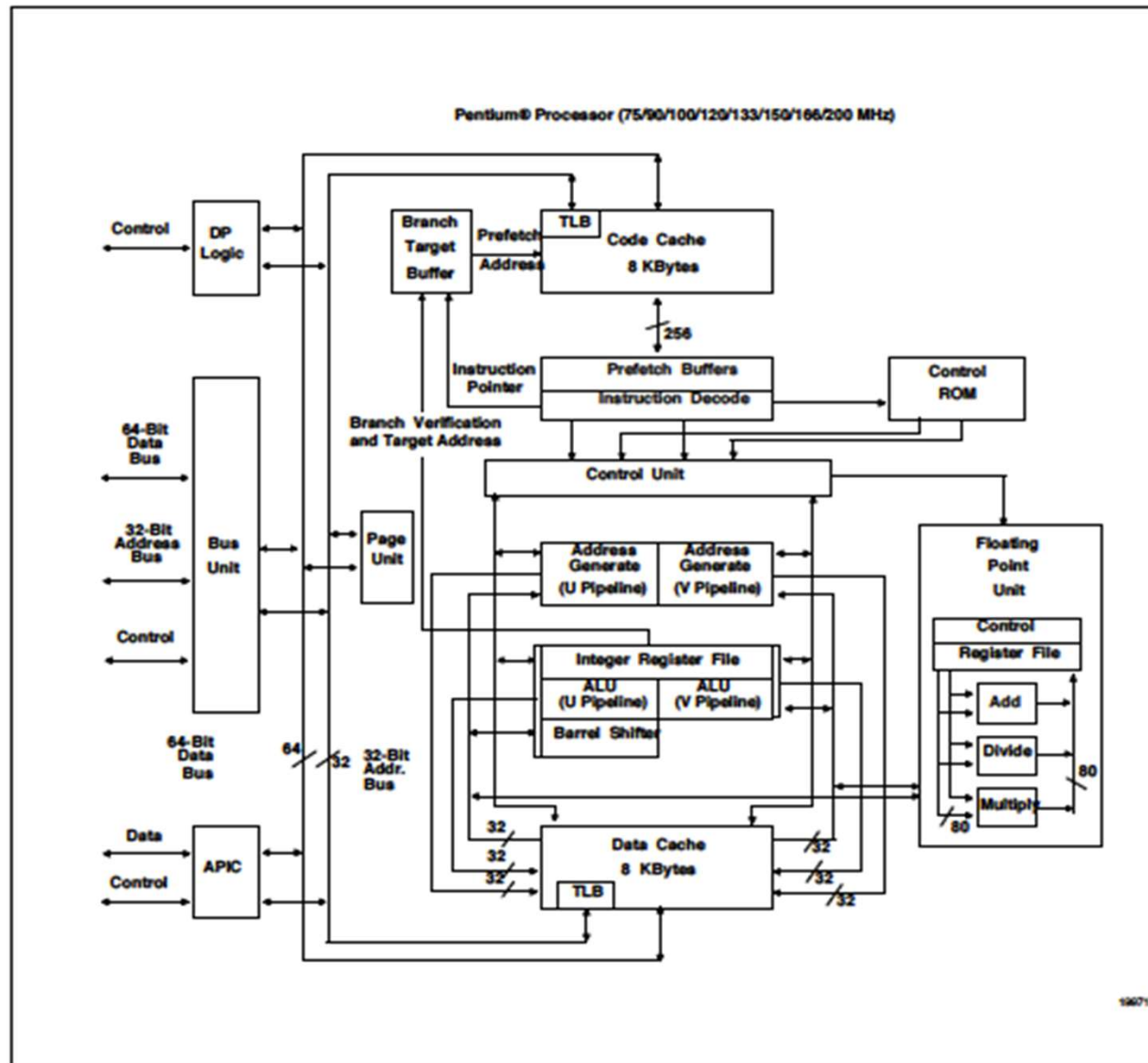


Figure 1. Pentium® Processor Block Diagram