PROFE: TONI CORTES

- TEMA 1 : REPRESENTACIÓ IN -

1.1. DIFERENTS REPRESENTACIONS DEL MATEIX VALOR

$4352 \equiv 2 \times 1 = 2 \ (\cdot 10^0)$

$5 \times 10 = 50 \ (\cdot 10^1)$

$3 \times 100 = 300 \ (\cdot 10^2)$

$4 \times 1000 = 4000 \ (\cdot 10^3)$

$\equiv \sum_{i=0}^{n-1} x_i 10^i$ ex: $\sum_{i=0}^{3} x_i 10^i = 4 \cdot 10^3 + 3 \cdot 10^2 + 5 \cdot 10^1 + 2 \cdot 10^0 = 4352$

$1011101 \equiv \sum_{i=0}^{n-1} x_i 2^i = 93$;

$10000001 \equiv \sum_{i=0}^{n-1} x_i 2^i = 129$;

128 64 32 16 8 4 2 1

2^0 2^1 2^2 2^3 2^4 2^5 2^6

* multiples de dos

$0 - 9 A B C D E F \rightarrow 0 - A3 \rightarrow A \times 16^1 + 3 \cdot 16^0 \Rightarrow 160 + 3 = 163$

10 11 12 13 14 16

$V_{(b)} = \sum_{i=0}^{n-1} x_i (b)^i$

* Donat x número de bits quin rang podem representar?

* Quants bits puc representar en x número?

RANG : interval del 0 a n. $\begin{cases} n \\ b \end{cases} \Rightarrow \text{Rang} = [0, b^n - 1]$

ex: binari $\begin{cases} \text{si } n = 3 \\ \text{en base } b = 2 \end{cases} \Rightarrow [0, \overset{(7)}{111}] \Rightarrow [0, 2^3 - 1]$

$\begin{cases} n = 3 \\ b = 10 \end{cases} \Rightarrow [0, 999] \Rightarrow [0, 10^3 - 1]$

$\hookrightarrow = (999+1) - 1 = 1000 - 1 = 10^3 - 1$

* Passem el 9 a binari ($b_{10} \rightarrow b_2$)

$9 \equiv 1001$

* Passem de $b_2 \rightarrow b_{16}$

* separem en 4 bits

8 2

1001 1010 $\rightarrow$ 9A

9 A

1B $\rightarrow$ 0001 1011

PROBLEMA :

b_{10}	b_{16}	b_2 (8 bits)
10	A	00001010
26	1A	00011010
49	31	00110001

$10 \rightarrow A \rightarrow$ 10 : 2 = 5 r 0; 5 : 2 = 2 r 1; 2 : 2 = 1 r 0; 1 : 2 = 0 r 1

; $1A \rightarrow 0001\ 1010 \rightarrow 1 \times 16 + A \times 1 = 16 + 10 = 26$

2^1 2^0

0011 0001 $\rightarrow$ 31 $\rightarrow$ $32 + 16 + 1 = 3 \cdot 16 + 1 = 49$

* Ejercicio video atenea: passa els seguents numeros de binari a decimal

$x = 1010 \rightarrow \begin{array}{cccc} 1 & 0 & 1 & 0 \\ 8 & & 2 & \end{array} = \begin{array}{r} 8 \\ +\ 2 \\ \hline 10 \end{array}$

$x = 11011100 \rightarrow \begin{array}{cccccccc} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 128 & 64 & 16 & 8 & 4 & & & \end{array} = \begin{array}{r} 11 \\ 128 \\ +\ 92 \\ \hline 220 \end{array}$

$x = 0011000111110101 \rightarrow \begin{array}{cccccccccccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 8192 & 4096 & & & 256 & 128 & 64 & 32 & 16 & & 4 & & 1 & & & \end{array} = \begin{array}{r} 1 \\ 8192 \\ 4096 \\ +\ 501 \\ \hline 12.789 \end{array}$

* De decimal a binário

$x_4 = 426$;

426 | 2
02 213 | 2
06 013 106 | 2
0) 1) 06 53 | 2
0) 13 26 | 2
1) 06 13 | 2
0) 1) 6 | 2
0) 3 | 2
1) 1

$x = 110101010$

$x_4 = 35$;

35 | 2
15 17 | 2
1) 1) 8 | 2
0) 4 | 2
0) 2 | 2
0) 1

$x = 00100011$

$x_4 = 145$;

145 | 2
05 72 | 2
1) 12 36 | 2
0) 16 18 | 2
0) 0) 9 | 2
1) 4 | 2
0) 2 | 2
0) 1

$x = 10010001$

* Com funciona la b_{16}

b_{10}	b_{16}
0	0
1	1
⋮	⋮
9	9
10	A
11	B
12	C
13	D
14	E
15	F

* MÉTODE:

$b_{10} = 699 \rightarrow b_{16} = ?$

699 | 16
59 43 | 16
11) 11) 2 | 16
2) 0

$b_{16} = 2BB$

- ET 2 -

1. $x = x_{n-1} \cdot x_{n-2} \cdots x_1 x_0 \rightarrow \sum_{i=0}^{n-1} x_i b^i$

2. Rang de $\begin{cases} n = 3 \\ b = 16 \end{cases} \rightarrow [0, b^n - 1] \equiv [0, 16^3 - 1] = [0, 4095]$

3. Rang de $\begin{cases} n = 6 \\ b = 2 \end{cases} \rightarrow [0, b^n - 1] \equiv [0, 2^6 - 1] = [0, 63]$

4. Rang de $\begin{cases} n = 3 \\ b = 8 \end{cases} \rightarrow [0, b^n - 1] \equiv [0, 8^3 - 1] = [0, 511]$

5. Rang de $\begin{cases} n = 6 \\ b = 3 \end{cases} \rightarrow [0, b^n - 1] \equiv [0, 3^6 - 1] = [0, 728]$

6. $x = 10001 \rightarrow x_4 = 1 \cdot 2^0 + 1 \cdot 2^4 = 17$

7. $x = 10110 \rightarrow x_4 = 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^4 = 22$

8. $x = 10110 \rightarrow x_4 = 22$;

22	16
6	1
1	0

; $b_{16} = 16$

9. $x = \underbrace{10011011}_{9} \underbrace{}_{11} \rightarrow b_{16} = 9B$

10. $x_4 = 69$;

69	2					
09	34	2				
1	14	17	2			
	0	1	8	2		
			0	4	2	
				0	2	2
					0	1

$x = 1000101$

11. $x_4 = 2047$;

2047	2									
0047	1023	2								
047	023	511	2							
07	03	111	255	2						
1	1	11	055	127	2					
		1	15	07	63	2				
			1	1	03	31	2			
					1	11	15	2		
						1	1	7	2	
							1	3	2	
								1	1	2
									1	0

$x = 11111111111$

12. $x_4 = 2047$;

2047	16	
447	127	16
127	15	7
15	7	0

$b_{16} = 7FF$

13. $x_4 = 428$;

428	16	
108	26	16
12	10	1
1	0	

$b_{16} = 1AC$

14. $b_{16} = 3F$; $x_4 = 3 \cdot 16^1 + F \cdot 16^0 = 63$

63	2					
03	31	2				
1	11	15	2			
	1	1	7	2		
			1	3	2	
				1	1	

15. b_{16} = C37A ; $X_4 = C \cdot 16^3 + 3 \cdot 16^2 + 7 \cdot 16^1 \cdot A \cdot 16^0 = 50042$

X = 1100001101111010

16. b_{16} = A3C ; X_4 = 2620

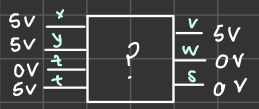
17. b_{16} = 2ED ; X_4 = 749

19. 71 amb 3 dígits més 00071

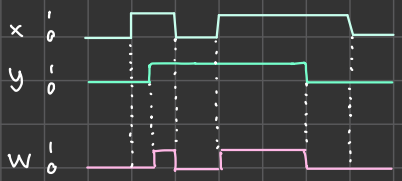
20. ESCRIU BA13 amb 3 dígits més 000 BA13

- TEMA 2: CIRCUITS COMBINACIONALS -

Un circuit combinacional és un circuit amb sortides que depenen únicament de les entrades i sempre dona el mateix.

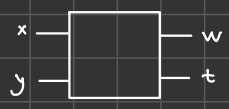


Per interpretar aquest circuits normalment utilitzem CRONOGRAMES



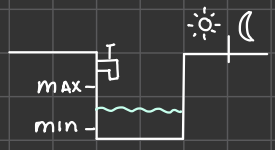
* Aquest sistema es inefectiu quan tenim varies entrades i sortides

Per ser més exactes utilitzem una versió numérica dels cronogrames coneguda com TAULA DE VERITAT



x	y	w	t
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

exemple:



Hem de fer un circuit que no deixi que l'aigua baixi del mínim ni passi del máxim i intenti no ompir de dia i ompli a la nit al máxim de lo possible.

max / min { 0 sec, 1 moll

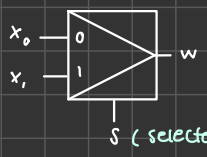
dn { 0 dia, 1 nit

aixeta { 0 tancada, 1 oberta

TAULA DE VERITAT

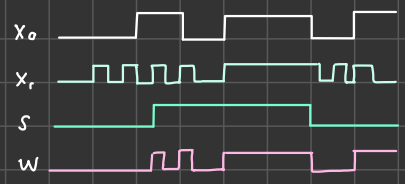
max	min	dn	a	
0	0	0	1	
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	x	→ Opció impossible
1	0	1	x	→ Opció impossible
1	1	0	0	
1	1	1	0	

MULTIPLEXOR



Un multiplexor actua com un semàfor.
* Quan S val 0 w es comporta com x_0 i quan S val 1 w es comporta com x_1.

exemple:



⇒ TAULA DE VERITAT ⇒

S	x_0	x_1	w
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

2.1. PORTES LÒGIQUES BÀSIQUES

NOT

x	y
0	1
1	0

AND

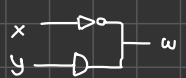
x	y	w
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	w
0	0	0
0	1	1
1	0	1
1	1	1

2.1.1 normes bàsiques

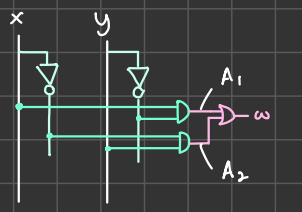
- NO es poden conectar dues sortides



- No poden haber-hi entrades lliures

- NO poden haber cicles

exemple:



⇒

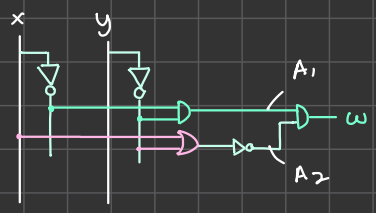
x	y	$\bar{x}$	$\bar{y}$	A_1	A_2	w
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

⇒

XOR → OR exclusiva

x	y	w
0	0	0
0	1	1
1	0	1
1	1	0

exemple:



⇒

x	y	$\bar{x}$	$\bar{y}$	A_1	A_2	$\bar{A_2}$	w
0	0	1	1	1	1	0	0
0	1	1	0	0	0	1	0
1	0	0	1	0	1	0	0
1	1	0	0	0	1	0	0

⇒

x	y	w
0	0	0
0	1	0
1	0	0
1	1	0

— ET 3 —

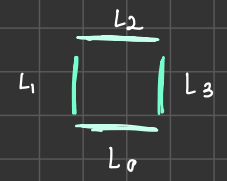
1. CLC és aquell en el qual (e) els valors dels senyals de sortida depenen dels valors dels senyals d'entrada en un matcix moment.

2. $x, y, z \rightarrow f(x,y,z) = 1$ quan una de les condicions es compleixi $\begin{cases} x=0 \text{ o } y=1 \\ y=0 \text{ i } z=1 \\ x,y,z=1 \end{cases}$

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

* + sólo la d.

3. Panell format per 4 llums



si $a=0 \rightarrow$  si $a=1 \rightarrow$ 

si $a=2 \rightarrow$  si $a=3 \rightarrow$ 

Tenim una entrada "a", cada llum s'encen amb un 1

L_0	L_1	L_2	L_3	a_1	a_2
1	1	1	1	0	0
0	0	0	1	0	1
0	1	0	1	1	0
1	1	0	1	1	1

11.

x	y	z	v	w
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

12.

x	y	z	v	w
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

2.2. SISTEMA EN SUMA DE MINTERMS

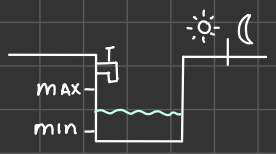
→ Son aquelle funcions amb valor 1.

Per implementar qualsevol minterm es necessita una sola porta AND i varies portes NOT.

PROPIETATS DE AND (i OR)

- Commutatiu $AB = BA$
- Associativa $(AB)C = A(BC) = ABC$
- Element neutre = 1

exemple:



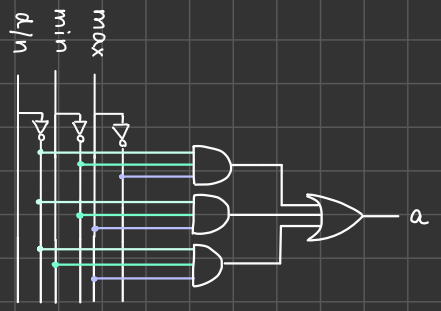
Hem de fer un circuit que no deixi que l'aigua baixi del mínim ni passi del màxim i intenti no omplir de dia i omplir a la nit al màxim de lo possible.

dn { 0 dia, 1 nit }

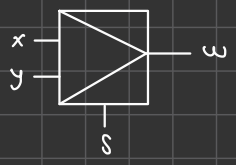
max / min { 0 sec, 1 moll } ⇒ aixeta { 0 tancada, 1 oberta }

TAULA DE VERITAT

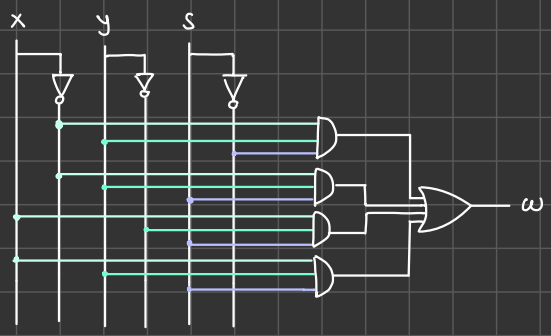
max	min	dn	a	
0	0	0	1	→ mint 1
0	0	1	1	→ mint 2
0	1	0	0	
0	1	1	0	→ mint 3
1	0	0	x	
1	0	1	x	
1	1	0	0	
1	1	1	0	



MULTIPLEXOR

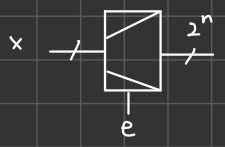


x	y	s	w	
0	0	0	0	
0	0	1	0	
0	1	0	1	→ m_1
0	1	1	1	→ m_2
1	0	0	0	
1	0	1	1	→ m_3
1	1	0	0	
1	1	1	1	→ m_4



DECODIFICADOR

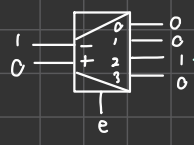
És un aparell de n entrades que té 2^n sortides, es un dispositiu que implementa cada un dels 2^n minterms de n entrades. La sortida d_j és la funció $minterm_j$ de las n entrades del decodificador.



* si e → 0 totes les sortides valen 0.
si e → 1 es comporta normal.

e	x_1	x_0	w_3	w_2	w_1	w_0	
0	0	0	0	0	0	0	
0	0	1	0	0	0	0	
0	1	0	0	0	0	0	
0	1	1	0	0	0	0	
1	0	0	0	0	0	1	→ m_1
1	0	1	0	0	1	0	→ m_2
1	1	0	0	1	0	0	→ m_3
1	1	1	1	0	0	0	→ m_4

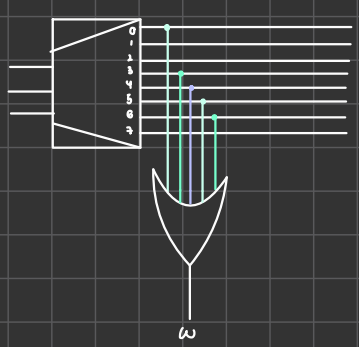
⇒



→ s'encen la llumeta

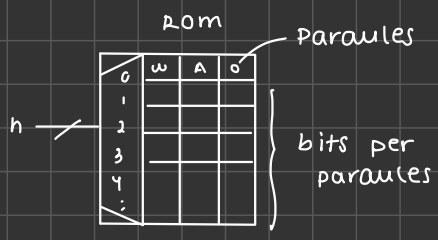
x	y	z	w
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

⇒



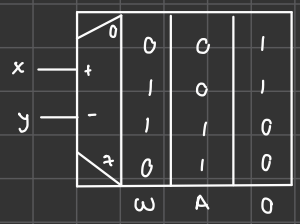
ROM

Per solucionar un circuit combinacional amb més de 4-5 entrades.



x	y	w	A	0
0	0	0	0	1
0	1	1	0	1
1	0	1	1	0
1	1	0	1	0

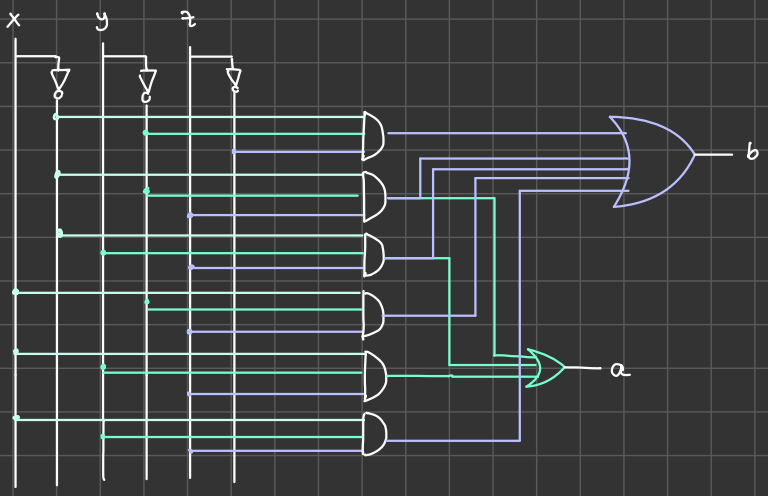
⇒



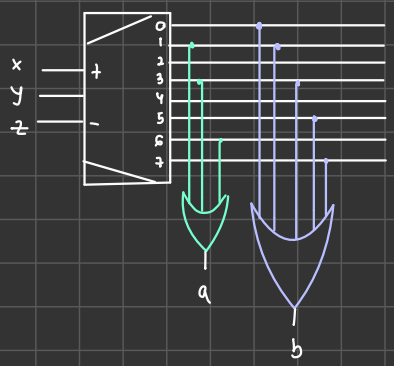
ex:

x	y	z	a	b
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

a)



b)



— ET3 b —

1.

x	y	w
0	0	1
0	1	1
1	0	0
1	1	1

→ $w = \bar{x}\bar{y} + \bar{x}y + xy$

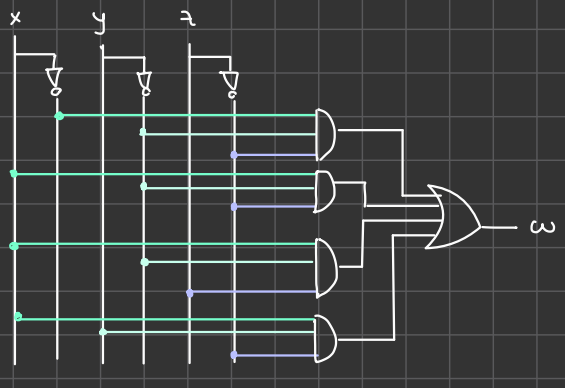
2.

x	y	z	w
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

→ $w = \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}yz$

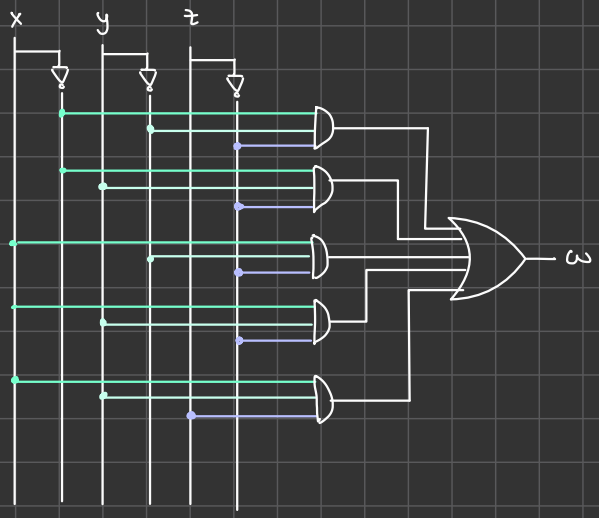
3.

x	y	z	w
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



4.

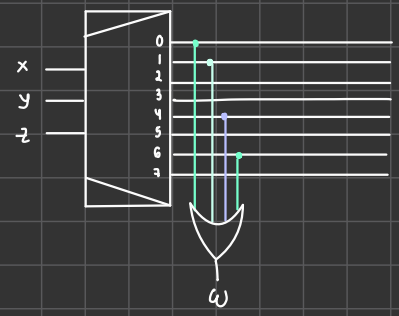
x	y	z	w
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



9.

x	y	z	w
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

⇒



2.3. ANÁLISIS TEMPORAL DE LES PORTES BÁSIQUES

→ tenen retards

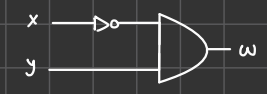
TEMPS DE PROPAGACIÓ

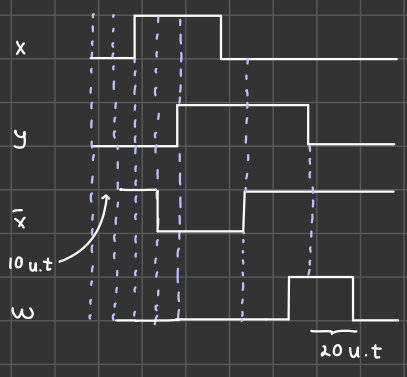
T_p (NOT) = 10 u.t.

T_p (OR) = 20 u.t.

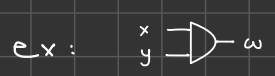
T_p (AND) = 20 u.t.

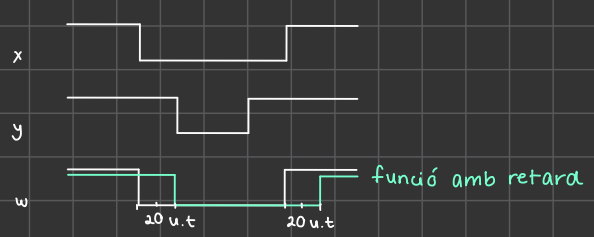
ex:



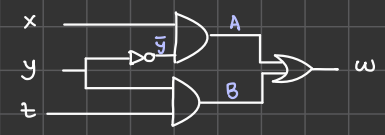


ex:

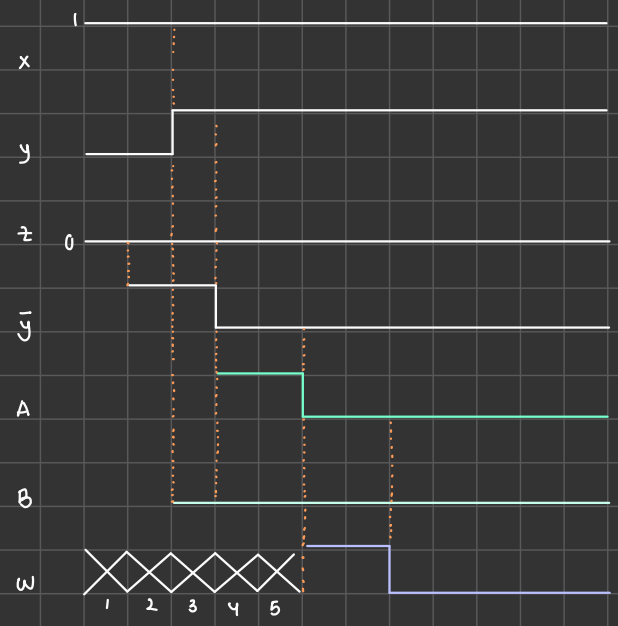




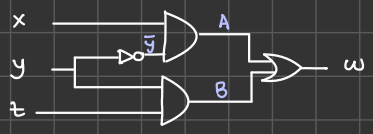
Ex.

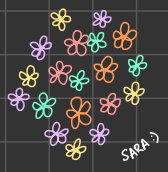


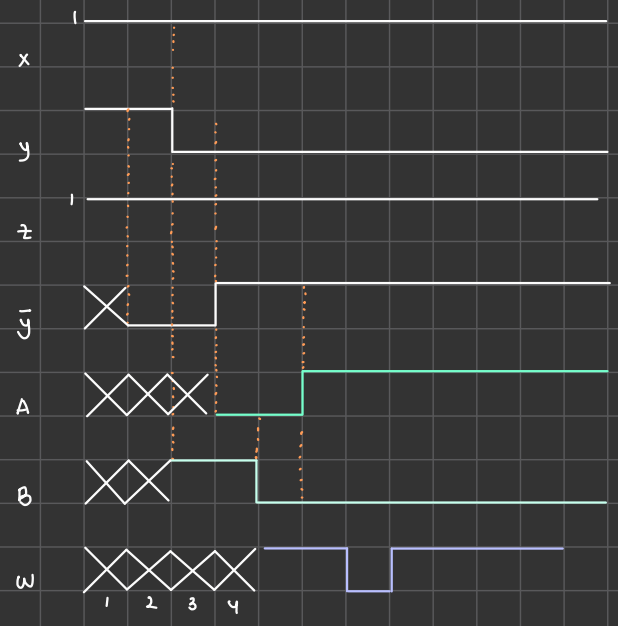
considero que 1 cuad. = 10 u.t.



T_p (TOTAL) = 50 u.t







T_p (TOTAL) = 40 u.t.

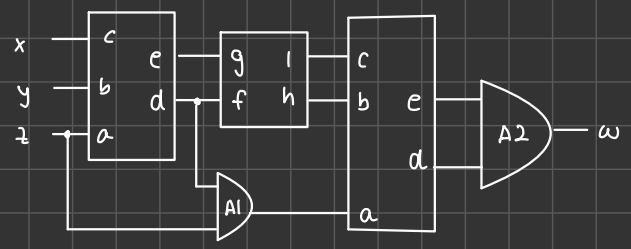
x — A1 (20) — 0 (20) — w — 40 u.t

y — NOT (10) — A1 (20) — 0 (20) — w — 50 u.t

y — A2 (20) — 0 (20) — w — 40 u.t

z — A2 (20) — 0 (20) — w — 40 u.t

ex:



T_p (AND) = 20 u.t

T_p	d	e
a	20	25
b	10	15
c	15	20

T_p	h	i
f	10	5
g	5	10

* camí crític: el que tarda més temps en finalitzar el programa.

$x - c \begin{cases} c \xrightarrow{20} g \begin{cases} i \xrightarrow{10} c \begin{cases} c \xrightarrow{20} A2 - \omega & 70 \text{ u.t} \\ d \xrightarrow{15} A2 - \omega & 65 \text{ u.t} \end{cases} \\ h \xrightarrow{5} b \begin{cases} e \xrightarrow{15} A2 \xrightarrow{20} \omega & 60 \text{ u.t} \\ d \xrightarrow{10} A2 \xrightarrow{20} \omega & 55 \text{ u.t} \end{cases} \end{cases} \\ d \xrightarrow{15} f \begin{cases} i \xrightarrow{5} c \\ h \xrightarrow{10} b \end{cases} \end{cases}$ ...

- ET3c -

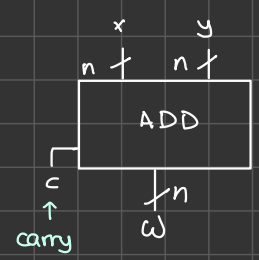
1.

a	b	c	avb	j	k	f	g
x	x	x	x	x	x	x	x
0	1	1	x	x	x	x	x
1	1	1	x	x	0	x	x
0	1	0	1	1	0	x	x
0	0	0	1	1	1	0	x
1	0	1	1	1	1	0	x
0	0	1	0	1	0	0	x
x	x	x	1	1	0	1	1
x	x	x	0	1	x	0	1
x	x	x	x	1	x	1	1
x	x	x	x	x	x	x	1
x	x	x	x	x	x	x	1

2.

a	b	c	j	k	i	f	g
x	x	x	x	x	x	x	x
0	1	1	x	x	x	x	x
1	1	1	x	x	0	x	x
0	1	0	0	x	0	x	x
0	0	0	1	x	1	0	x
1	0	1	0	1	1	0	x
0	0	1	0	1	0	x	1
x	x	x	1	1	0	1	1
x	x	x	0	1	x	0	1
x	x	x	x	1	x	0	1
x	x	x	x	1	x	x	1
x	x	x	x	1	x	x	1

- TEMA 3 : REPRESENTACIÓ NATURALS -



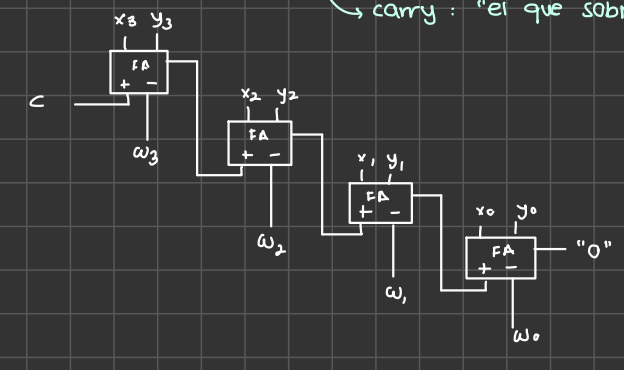
n = 64 → nº de bits

* No es pot fer una taula de la veritat perque tenen moltes entrades.

$$\begin{array}{r} 110 \\ 1572 \\ + \quad 0540 \\ \hline 2112 \end{array} \rightarrow \begin{array}{r} 1\,1\,1\,1\,0 \\ 0\,1\,1\,0\,1\,1 \\ + \quad 1\,0\,1\,1\,1\,0 \\ \hline 1\,0\,0\,1\,0\,0\,1 \end{array}$$

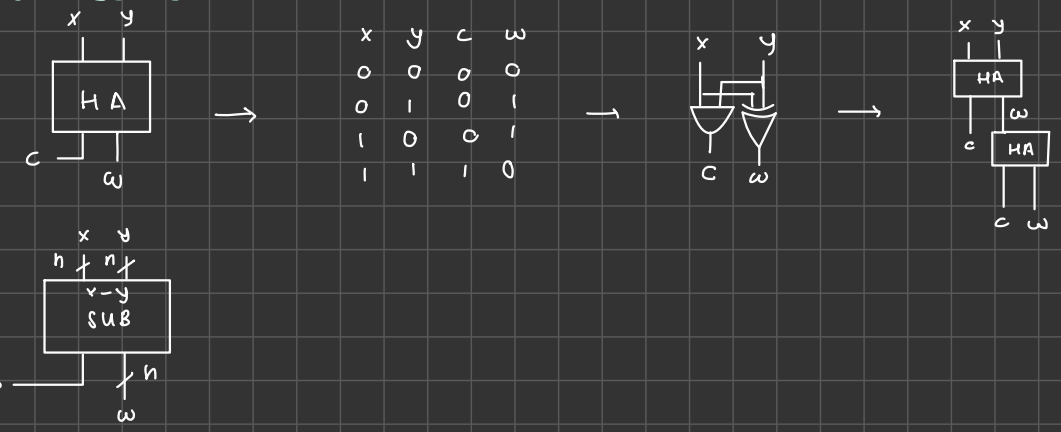
↳ carry: "el que sobra"

$$\begin{array}{r} 1\,1\,1\,0\,1 \\ 0\,1\,1\,0\,1\,1 \\ + \quad 1\,1\,1\,0\,1\,0 \\ \hline 1\,0\,1\,0\,1\,0\,1 \end{array} \rightarrow$$



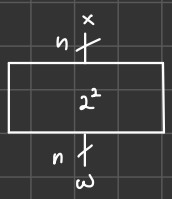
x_i	y_i	c_{i-1}	c_i	ω_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

HALF ADDER

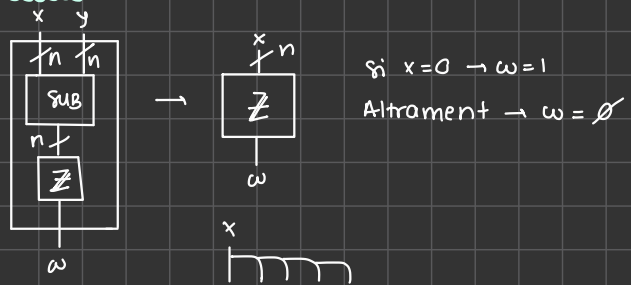


MULTIPLICADOR

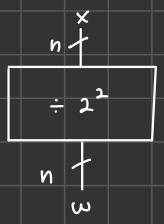
$11 \times 2^2 = 1100$

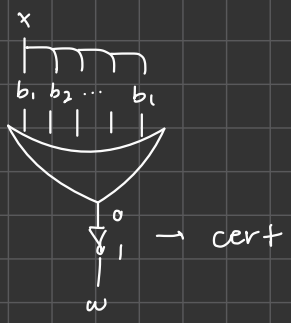


IGUAL (EQ)

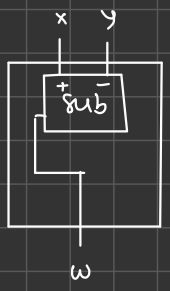


DIVISOR



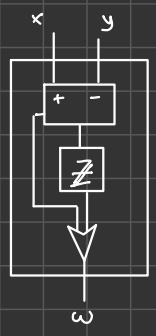


LESS THAN (LTU)



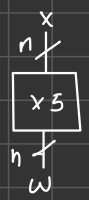
si $x < y \rightarrow \omega = 1$ (cert)
Altrament $\rightarrow \omega = \emptyset$ (fals)

LET (LETU)

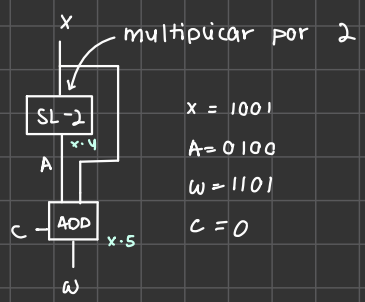


ex.

Fes un component que

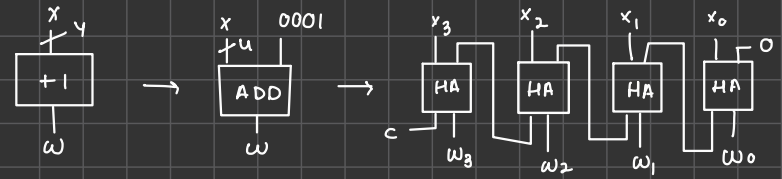


$x \cdot 5 = x \cdot 4 + x$



x = 1001
A = 0100
ω = 1101
c = 0

Fes un component que



- ETY - * Reglas bàsiques $\begin{cases} 0-0=0 \\ 0-1=1 \\ 1-0=1 \\ 1-1=0 \end{cases}$ * arrastras 1

1. $\begin{array}{r} 1\,1\,\;1\;\;1\,1\,1 \\ 1\,1\,0\;1\,0\,1\,1\,0 \\ +\;0\,1\,0\;1\,1\,1\,1\,0 \\ \hline 1\,0\,0\,1\,1\,0\,1\,0\,0 \end{array}$

2. $\begin{array}{r} 1\;\;1\,1\,1\,1\,1\,1\,1 \\ 0\;1\,0\,0\,1\,1\,0\,1 \\ +\;0\,1\,1\,1\,0\,1\,1\,1 \\ \hline 1\,1\,0\,0\,0\,1\,0\,0 \end{array}$

6. $\begin{array}{r} 1\;\;1\,1\,1\,1\,1 \\ 1\,1\,0\;1\,1\,0\,1\,0\,1\,0\,1\,1 \\ -\;1\,0\,1\,1\,1\,1\,1\,1\,1 \\ \hline 1\,1\,1\,0\,0\,0\,1\,0\,0 \end{array}$

7. $\begin{array}{r} 1^{0}\,0\,1^{0}\,1^{0}\,1^{0}\,0\,1 \\ -\;0\,1\,0\,1\,1\,1\,1\,0 \\ \hline 0\,1\,0\,1\,1\,1\,1\,1 \end{array}$

9. $0\,0\,0\,0\,0\,1\,0\,1 \times 2^{6}$

$x_4 = 5$; $5 \cdot 2^6 = 320$;

320 | 2 → 160 | 2 → 80 | 2 → 40 | 2 → 20 | 2 → 10 | 2 → 5 | 2 → 2 | 2 → 1 | 2 → 0 | 1

NR

10. $0000\,1100 \times 2^{5}$

$x_4 = 12$; $12 \cdot 2^5 = 384$ → 10000000 NR

- TEMA 4: REPRESENTACIÓ ENTERS -

N	ℤ
0	000 0
1	001 1
2	010 2
3	011 3
4	100 -4
5	101 -3
6	110 -2
7	111 -1

Ca 2

si + ⇒ $V_S = \sum_{i=0}^{n-2} x_i 2^i$

si − ⇒ $V_S = -2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$

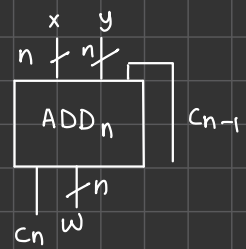
$\Big\} V_S = -2^{n-1} \cdot x_{n-1} + \sum_{i=0}^{n-2} x_i 2^i$

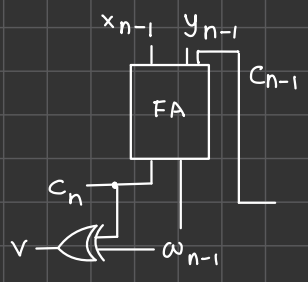
Rang representable amb n bits : $[-2^{n-1}, 2^{n-1}-1]$

Si tractem amb numeros negatius si ens demanan representar els numeros en n bits possem 1 devant.

4.1. HARDWARE PARA REPRESENTAR OPERACIONES CON NÚMEROS REALES

SUMADOR : amb nombres reals el carry no ens idica si es representable o no





+ + < + R, − NR

[+ −, − +] — Aquests casos SEMPRE són representables

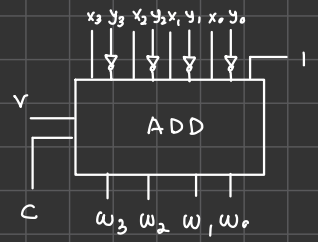
− − < + NR, − R

x_{n-1}	y_{n-1}	C_{n-1}	C_n	ω_{n-1}	V	
0	0	0	0	0	0	
[0+	0+	1−]	0	1]	1	NR
0	1	0	0	1	0	
0	1	1	1	0	0	≡ XOR (C_{n-1}, C_n)
1	0	0	0	1	0	
1	0	1	1	0	0	
[1−	1−	0+]	1	0]	1	NR
1	1	1	1	1	0	

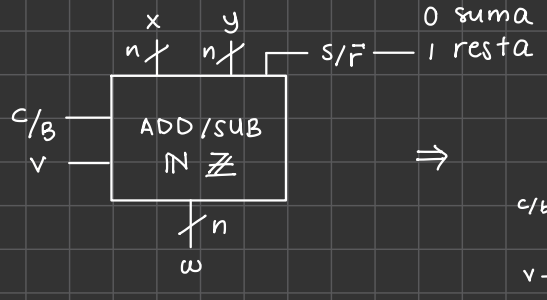
¿ COM PASSAR DE $n \to -n$?

$3 \to -3$; 011 (3) $\to$ 101 (-3) ; $0 \to 1$, $1 \to 0$, $+1$; 011 (3) $\to$ $100 + 1 = 101$ (-3) ; 101 (-3) $\to$ $010 + 1 = 011$ (3)

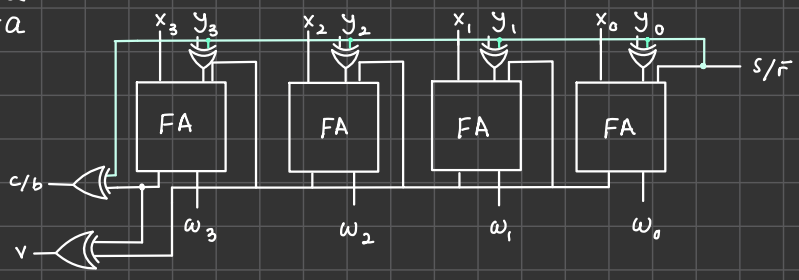
RESTADOR (de $\mathbb{N}$ i $\mathbb{Z}$)



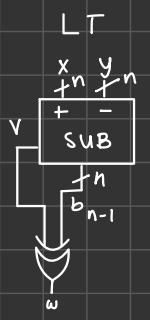
SUMADOR I RESTADOR DE $\mathbb{N}$ i $\mathbb{Z}$



$\Rightarrow$



COMPARADORS ($\mathbb{Z}$)



$$\frac{\begin{array}{l}SL-X\\SRL\\SLL\end{array}}{} \Rightarrow \mathbb{N}$$

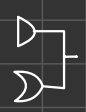
$$\frac{\begin{array}{l}SRA\\SLA\end{array}}{} \Rightarrow \mathbb{Z}$$

- TEMA 5 : CIRCUITS SEQUENCIALS -

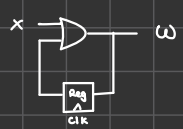
REGLES :

1 No podem tenir entrades lliures

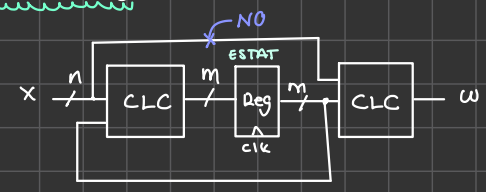
2. No podem conectar dos sortides



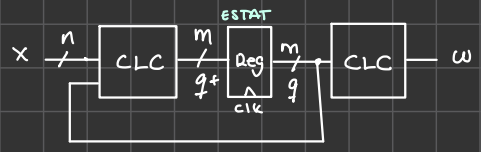
3. Es poden fer cicles sempre i quan hi hagi un diestable en el camí



MODEL MEALY



MODEL MOORE



★ Moore : elimina el cable que fa que la sortida depengui de la entrada.

TAULA SORTIDA

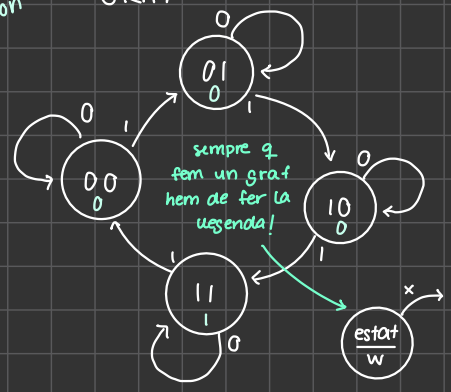
q_1	q_0	w
0	0	0
0	1	0
1	0	0
1	1	1

TAULA ESTAT SEGUENT

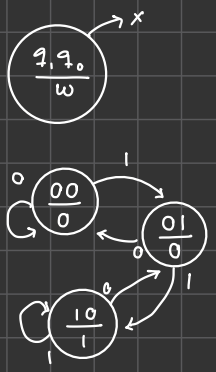
q_1	q_0	x	q_1^+	q_0^+
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

són equivalents

GRAF



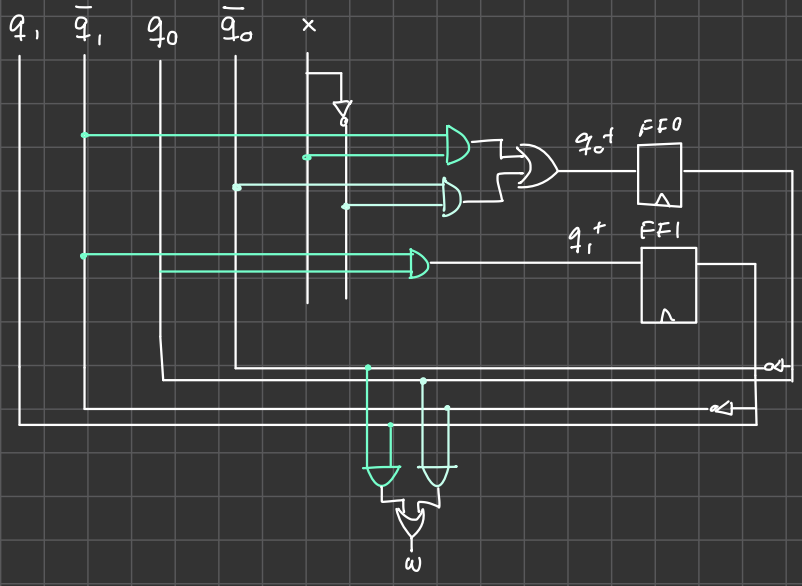
ex:



q_1	q_0	w
0	0	0
0	1	0
1	0	1
1	1	x

q_1	q_0	x	q_1^+	q_0^+
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	x	x
1	1	1	x	x

ex:



q_1	q_0	w
0	0	0
0	1	1
1	0	1
1	1	0

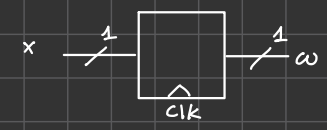
q_1	q_0	x	$\bar{q}_1$	$\bar{q}_0$	$\bar{x}$	q_1^+	q_0^+
0	0	0	1	1	1	0	1
0	0	1	1	1	0	0	1
0	1	0	1	0	1	1	0
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	1
1	0	1	0	1	0	0	0
1	1	0	0	0	1	0	0
1	1	1	0	0	0	0	0

Pasar de text a el circuit

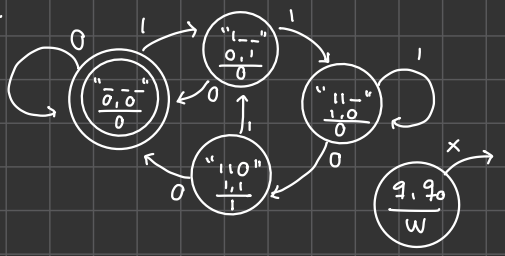
ex: (110)

x 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 0

w 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0



L'Estat és el que porto reconegut de la secuència 4 casos.



¿Quin és el teu estat inicial?

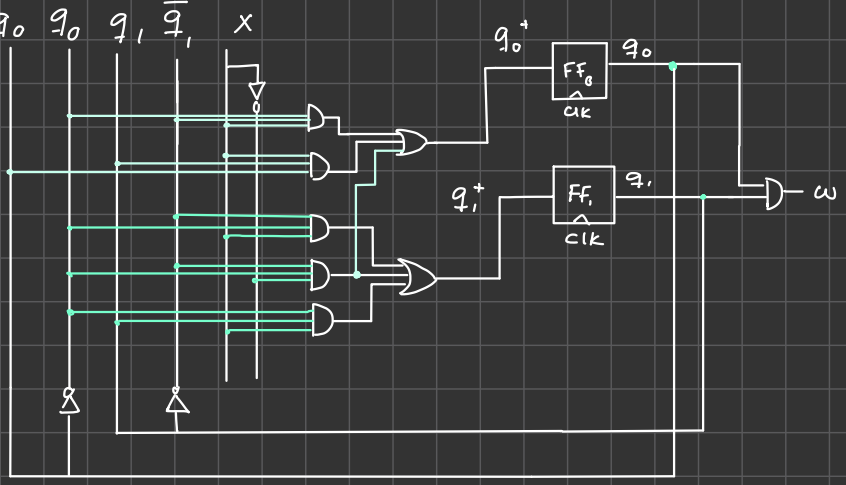
→ És marca amb una doble rodona

q_1	q_0	w
0	0	0
0	1	0
1	0	0
1	1	1

⇒

q_1	q_0	x	q_1^+	q_0^+
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

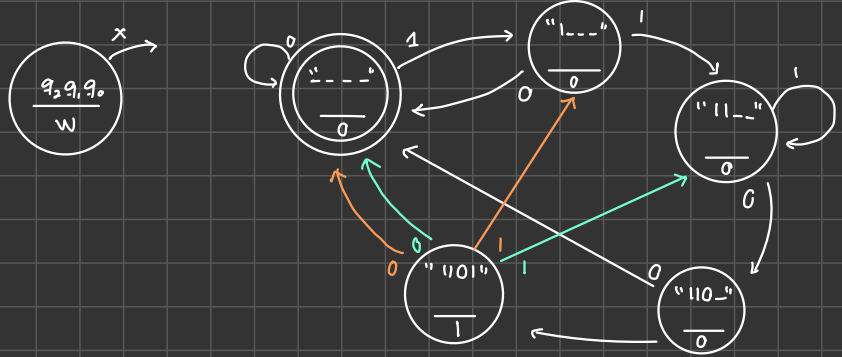
⇒



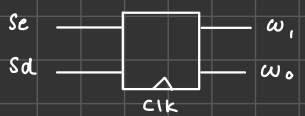
ex: (1101)

1000110110100110100001000

solapament

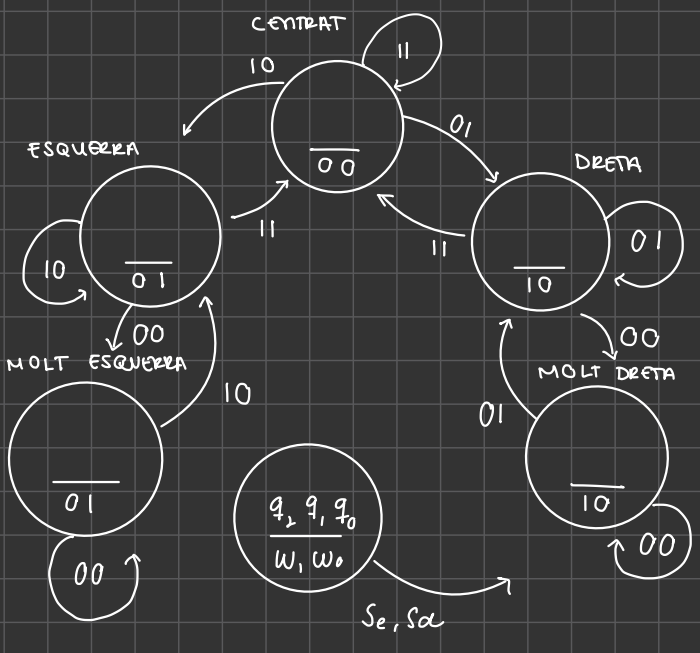


ex:



w_1	w_0	
0	0	recte
0	1	cap a la dreta
1	0	cap a l'esquerra
1	1	—

q_1	q_0	Se	Sd	w_1	w_0
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		



ANÁLISIS TEMPORAL CLS

- camino crítico: camino con mayor tp. de biestable a biestable, de entrada a biestable o de biestable a salida

- tiempo de ciclo (periodo): el tiempo de ciclo tiene que ser mayor al camino crítico para que el sistema funcione bien.

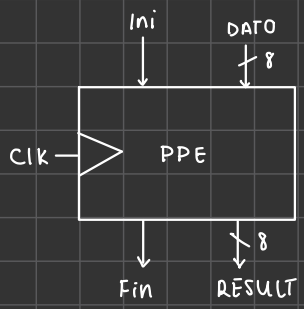
- TEMA 7: PROCESSADORS DE PROPÒSIT ESPECÍFIC -

* són CLS que requieren muchos estados

ej: diseña un CLS que realice la suma módulo 2^8 de una secuencia de 4 números naturales codificados en binário con 8 bits cada uno.

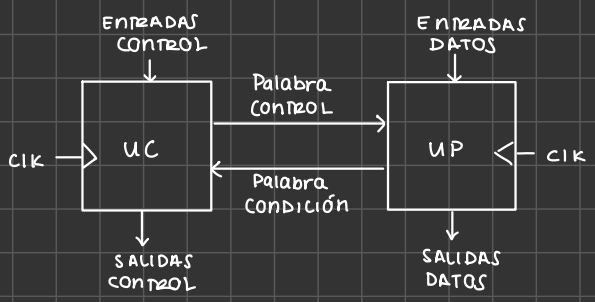
señal de entrada { DATO
Ini = los valores de DATO entran a razón de un num por ciclo empezando cuando Ini vale 1.

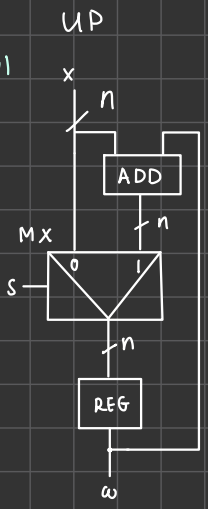
salida { RESULT
Fin = cuando Fin vale 1 se muestra el valor real de RESULT.



* Ini i Fin són entradas y saudas de control (validan los datos de DATO y RETURN)

Diseño del PPE con UP (unidad de proceso) y UC (unidad de control)



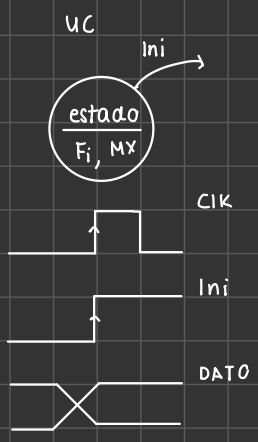


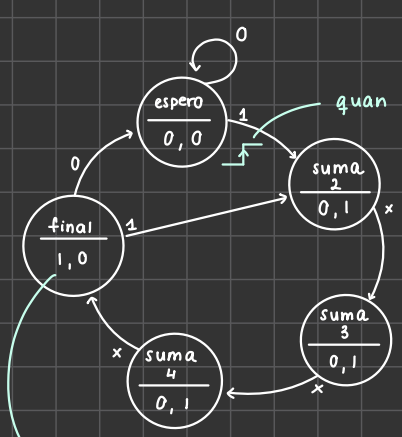
1r ciclo: MX = 0 deja pasar x.

2o ciclo: MX = 1 deja pasar el resultado del ADD de x con w (1r ciclo)

UC decide el valor de MX al principio en este caso

↳ Podria pasar que en el proceso el circuito aceptara otro Ini

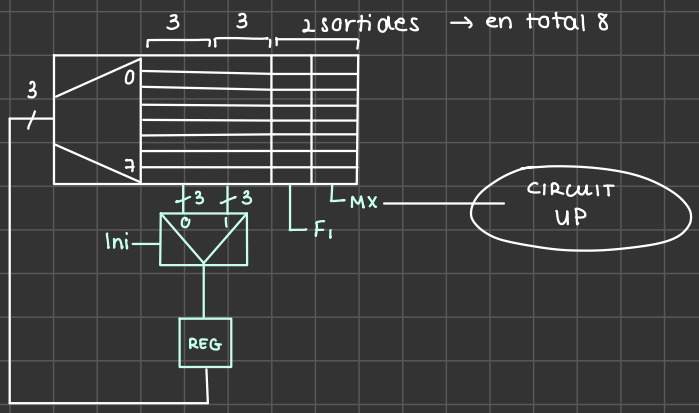




per passar directament a la suma del 2n número MX = 0 (per que deixi passar valors de x).

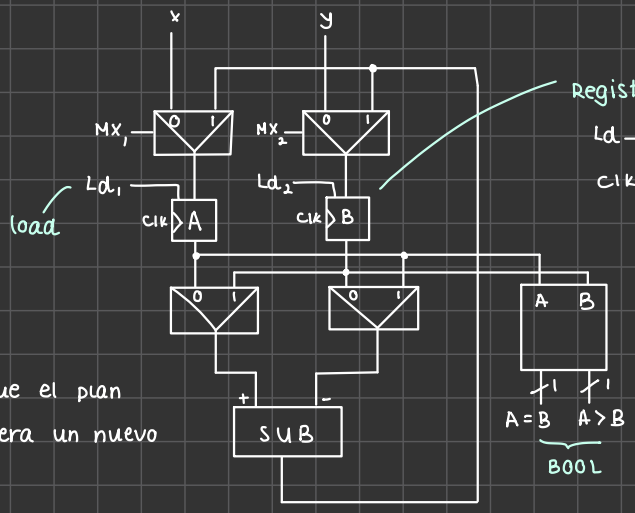
IMPLEMENTACIÓ UNITAT DE CONTROL

* 5 estats → necessito 3 bits per representarlos

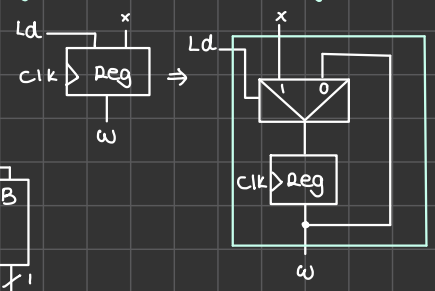


CIRCUIT MCD

```
A = x ; B = y
while ( A <> B ) {
  if A > B → A = A - B
  else B = B - A
}
Return = A
```



Registro con señal de carga

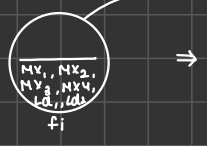


* LOAD · Ld = 1 cuando llegue el pian ascendente del clk, aprendera un nuevo valor.

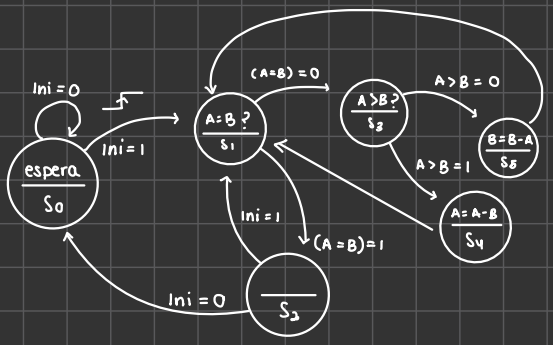
  Ld = 0 se queda el valor que ya tenia (independientemente del clk).

* Si el REG tiene señal de carga, al calcular el tp hay que añadir el tiempo del multiplexor.

GRAF: Ini / A=B ; A>B



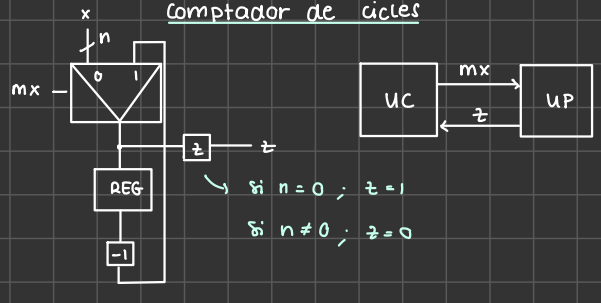
	MX_1	MX_2	MX_3	MX_4	Ld_1	Ld_2	Fi
S_0	0	0	x	x	1	1	0
S_1	x	x	x	x	0	0	0
S_2	0	0	x	x	1	1	1
S_3	x	x	x	x	0	0	0
S_4	1	x	0	1	1	0	0
S_5	x	1	1	0	0	1	0

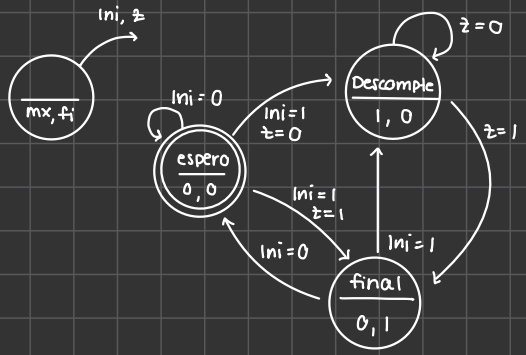


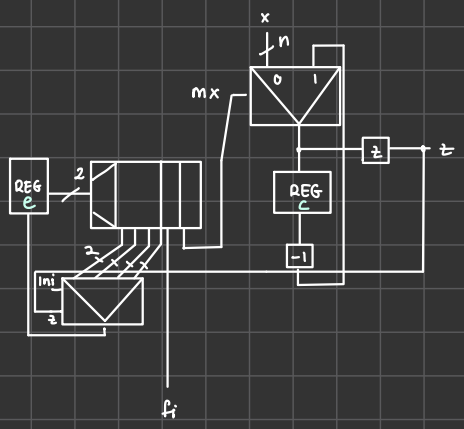
Es pot optimitzar saltant-he S_3. (unió S_1 - S_3)

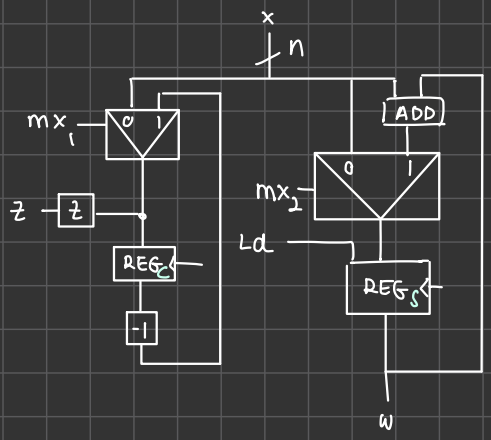
* Quan acabes (S_4 o S_5) tornes a començar (S_1)

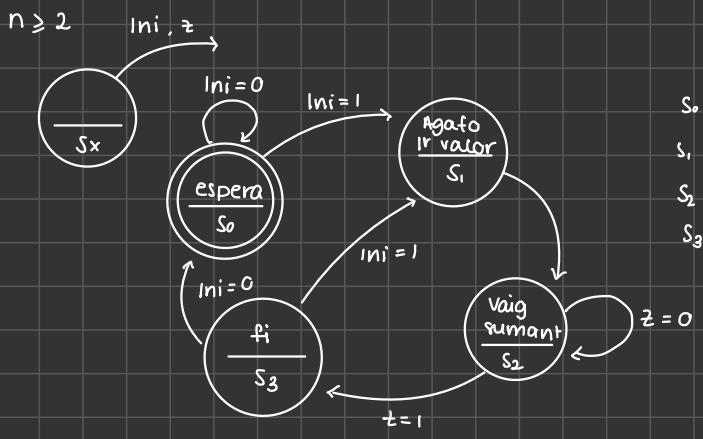
ex: Comptador de cicles











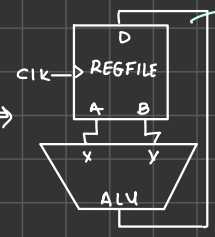
	mx_1	mx_2	Ld	fi
S_0	0	x	x	0
S_1	1	0	1	0
S_2	1	1	1	0
S_3	0	x	x	1

- TEMA 8 : UNITAT DE PROCESS GENERAL -

* És una unitat genérica que resol diversos problemes pel preu de que tardi una mica més de temps.

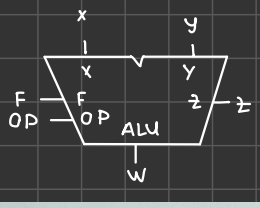
FINS ARA HEM VIST
- → Registres
- → Operadors

} Ho unim per fer la UPG ⇒

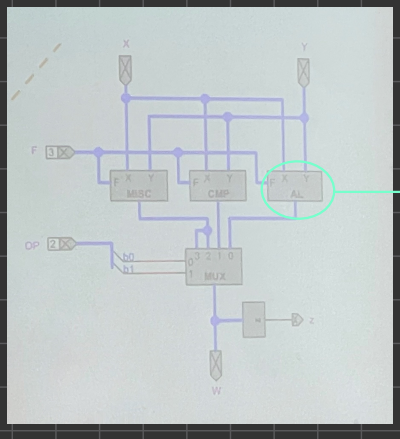


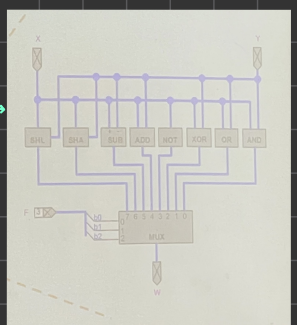
conte 8 REG
* A cada cicle pots llegir 2 REG. Operarlos i guardar el resultat a un dels 8 REG.

ALU mes afons (unitat aritmético lógica)



CIRCUIT INTERN

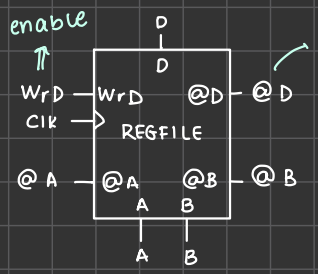




seqüência que t'indica el que pots fer

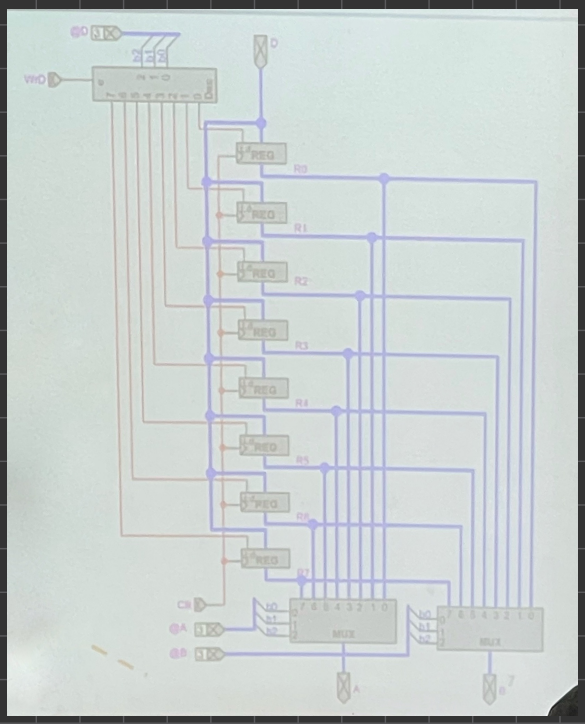
F	OP = 11	OP = 10	OP = 01	OP = 00
000		x	CMPLT (x,y)	AND
001		y	CMPLE (x,y)	OR
010				XOR
011			CMPEQ (x,y)	NOT
100			CMPLTU (x,y)	ADD
101			CMPLTE (x,y)	SUB
110				SHA
111				SHL

RegFile més afons (Banc de registres)



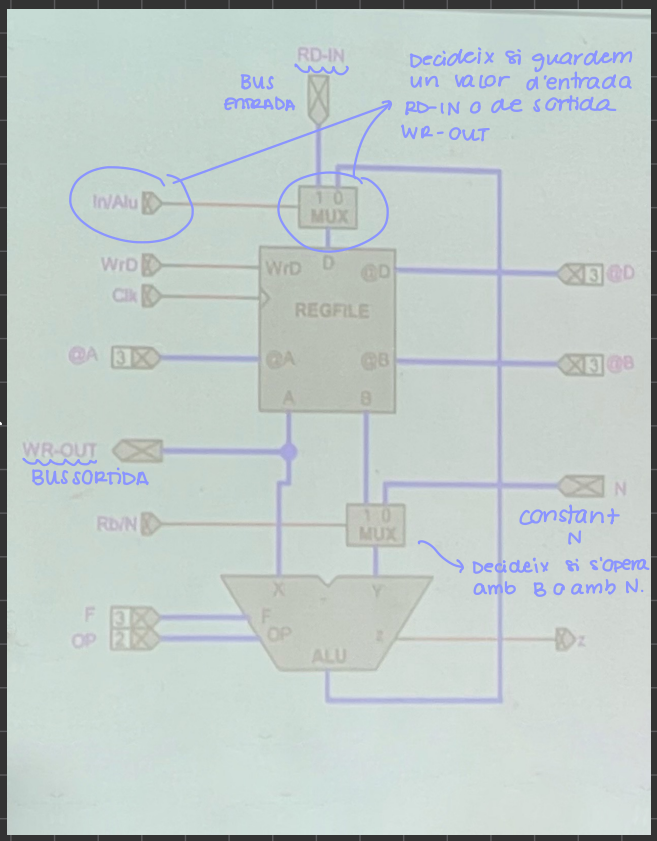
Aquí es guarda el valor

* WrD = 0 No modifica cap REG, es a dir, no és guarda el nou valor
* WrD = 1 Modifica un REG per guardar el nou valor



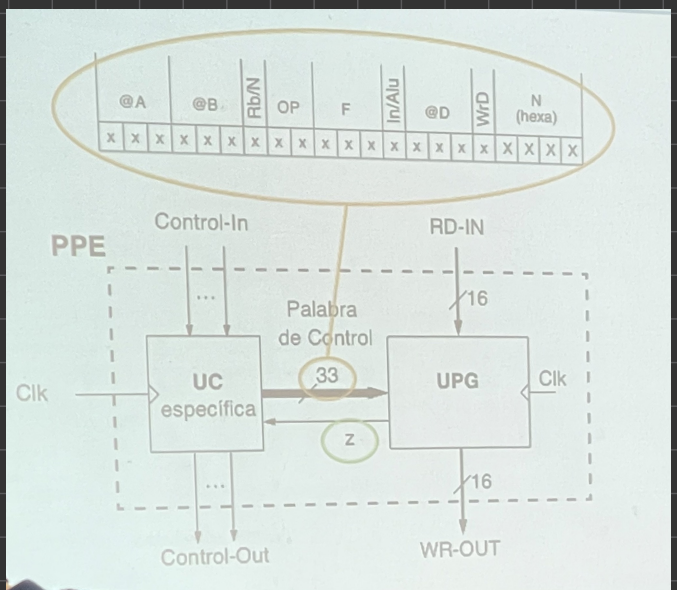
→ Si mirem el circuit així es guardaria sempre el valor d'entrada però això es fals.
Per això quan l'enable (WrD) del decodificador @D si val 0 tot val 0 i si val 1 tenim un funcionament normal

UPG (COMPLETA)



FUNCIONALITAT EXTRA
- Bus d'entrada RD-IN
- Bus de sortida WR-OUT

PARAULA DE CONTROL → 33bits
PARAULA DE CONDICIÓ → 1 bit (Z)



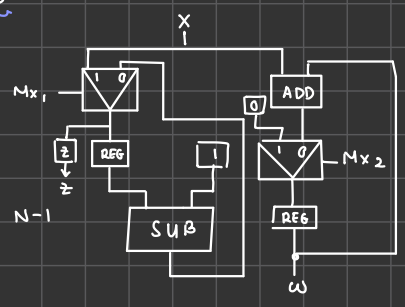
RESOLUCIÓ ET 7b

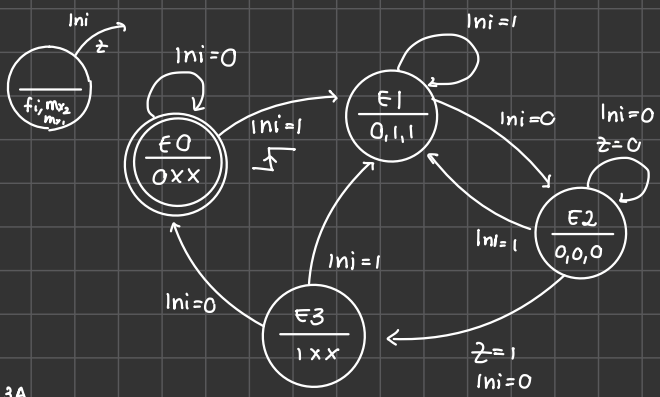
Ini CC) = 1

x CC+1) = N

x CC+2) = DADA 0

x CC+N+1) = DADA N-1





CICLE	00	1	2	3	4	5	6	7	8
X	0x32	0x02	0x34	0x7A	0x50	0xFA	0x03	0xC0	0x3A
Ini	1	0	0	1	0	1	0	0	0
fi	0	0	0	0	0	0	0	0	0
W	xx	xx	0x00	0x34	0xAE	0x00	0xFA	0x00	0xC0
E	xx	E1	E2	E2	E1	E_2	E_1	E_2	E_2
N	xx	xx	0x02	0x01	0x00	0x50	0x4F	0x03	0x02

MEMOTÈCNICS

* Aritmètico-lògiques

ADD Rd, Ra, Rb → Rd = ADD (Ra, Rb)

NOT Rd, Ra → Rd = NOT (Ra)

ANDI Rd, Ra, N → no operas amb registre sino amb constant, AND (Ra, N) = Rd
↳inmediat

AND- Ra, Rb → AND (Ra, Rb) es calcula però no es guarda

SHL (x, y) ⇒ IN
SHA (x, y) ⇒ Z

x → el que vols shiftejar
y → quants bits vols desplaçar-ho ⇒ si { y>0 ← (*) ; y<0 → (÷)

* Comparació

CMPLT Rd, Ra, Rb → Rd = CMPLT (Ra, Rb)

* moviment

MOVE Rd, Ra → Rd = Ra

MOVE I Rd, N → Rd = N

MOVE - Ra → passa Ra per l'ALU

* Entrada / sortida

IN Rd

OUT Ra

In/ALU	WrD	@D	@A	OP	F	Rb/N	@B	N (nexa)
0	1	000	100	01	001	1	001	xxxx
0	1	000	xxx	10	001	0	xxx	0000
0	1	000	xxx	xx	xxx	x	xxx	xxxx
0	1	xxx	100	00	100	0	xxx	0001

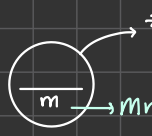
CMPEQ −, R3, R5

@A	@B	Rb/N	OP	F	In/Alu	@D	WrD	N
011,	101,	1,	01,	011,	x,	xxx,	0,	xxxx

MOVEI R4, 0x12

@A	@B	Rb/N	OP	F	In/ALu	@D	WrD	N
xxx,	xxx,	0,	10,	001,	0,	100,	1,	0012

GRAF UPG

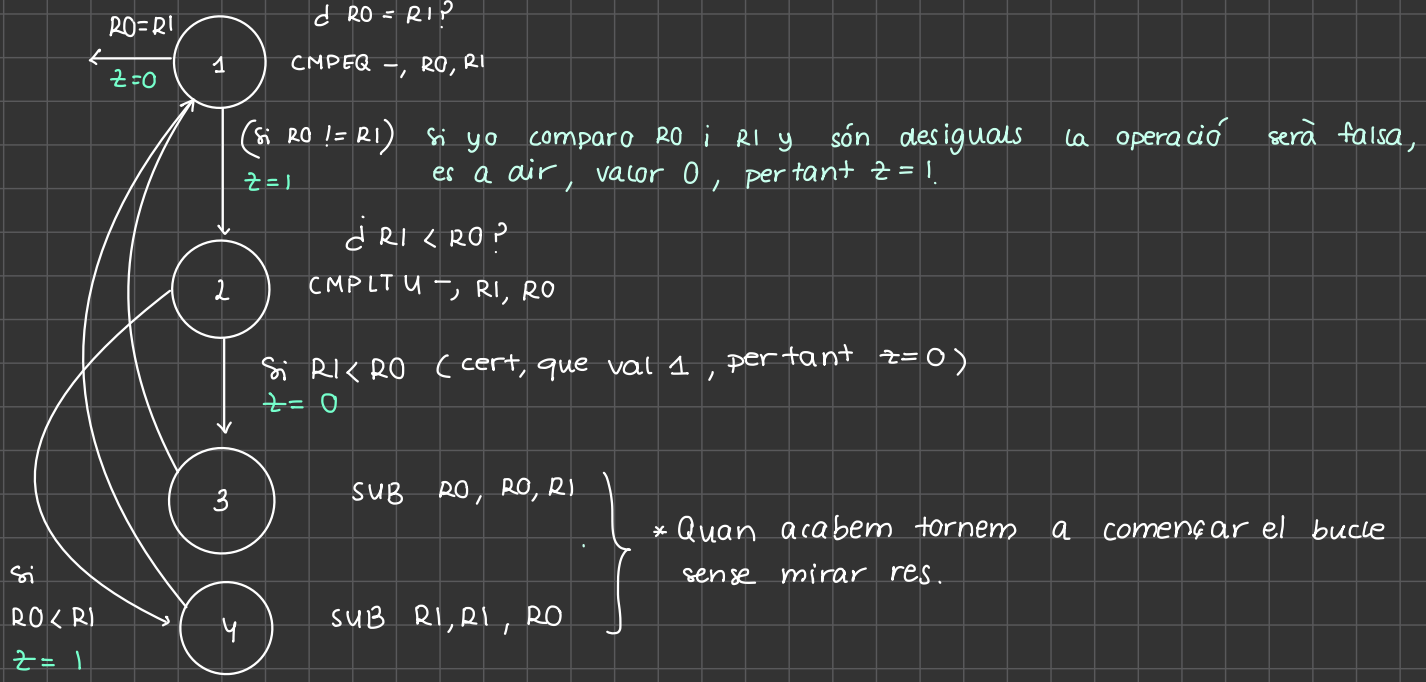


Solo w //out juntos

★ TAULA ATENEA (Power point)

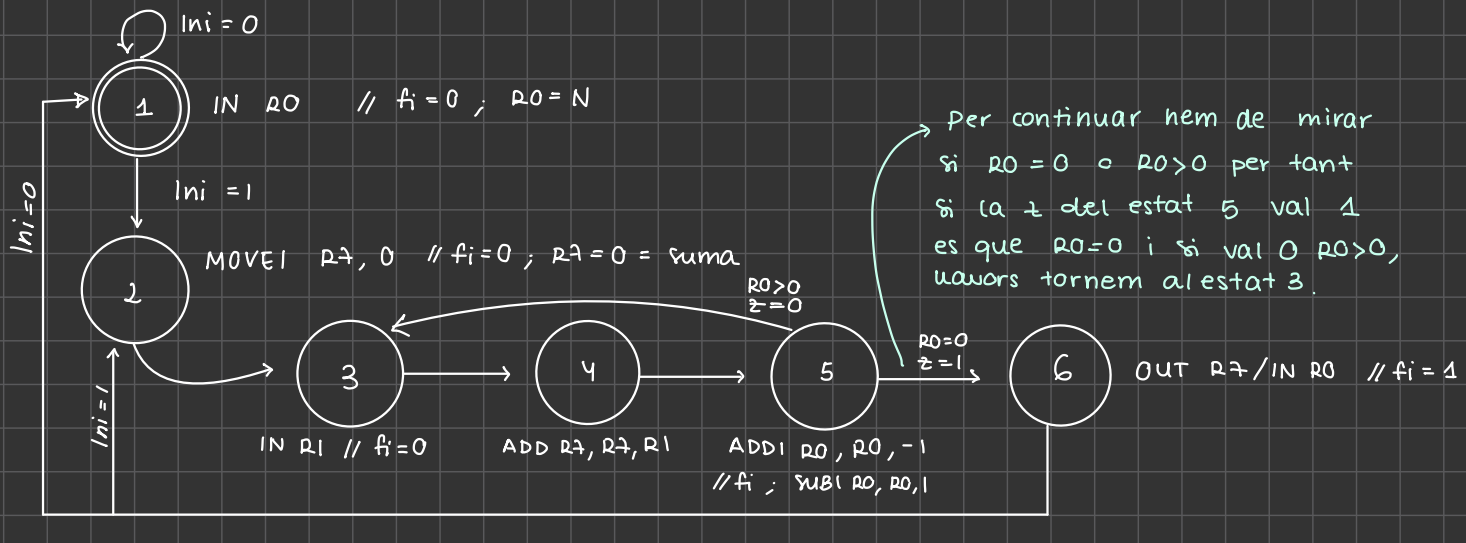
Del codi al graf (IMPORTANTE!)

```
while (R0 != R1) {
    if (R0 > R1) R0 = R0 - R1;
    else R1 = R1 - R0;
}
```

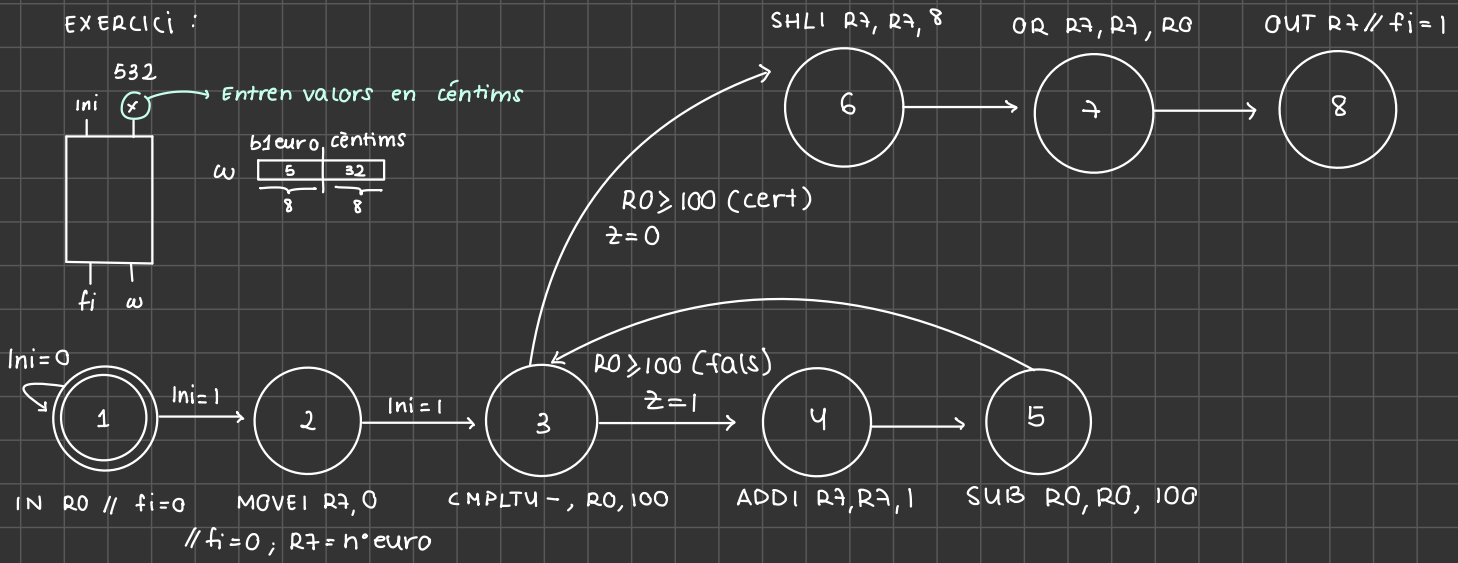


SUMA N VALORS

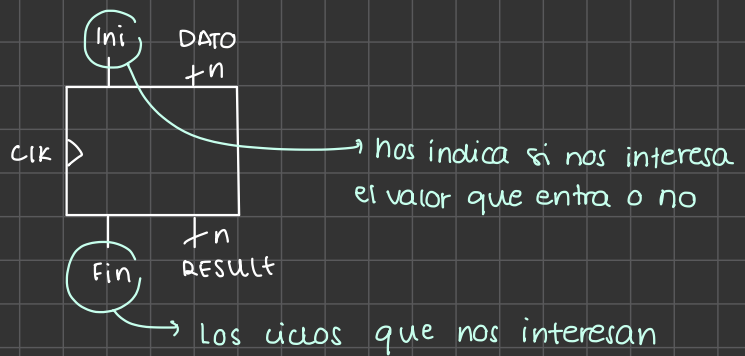
Per Ini=1 → nº de valors que sumem; N≥2

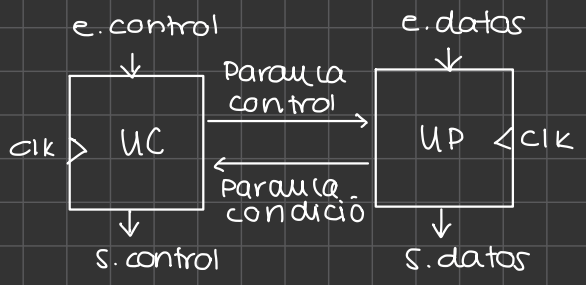


EXERCICI :



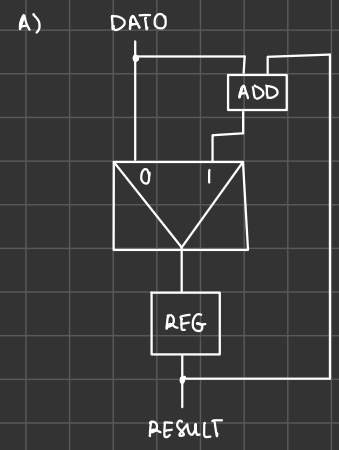
REPASO TEMA 7

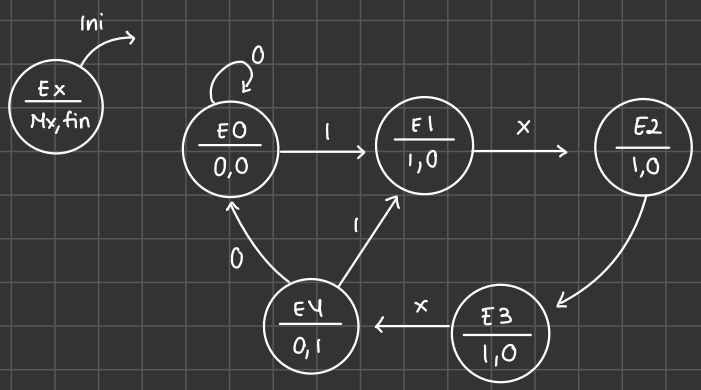




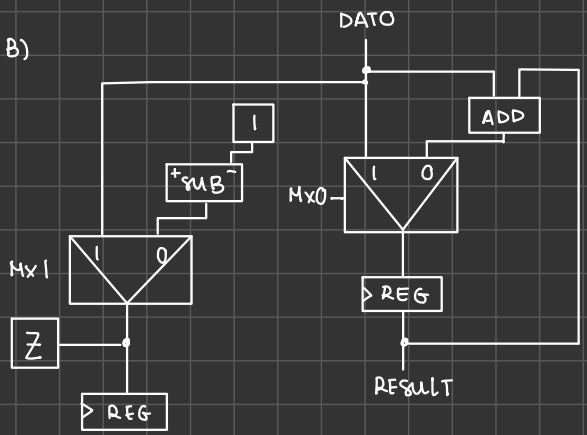
circuits d'exemple:

A)





B)



1)

Ini (C) = 1 → X (C+1) = N

X (C+2) = DADA 0

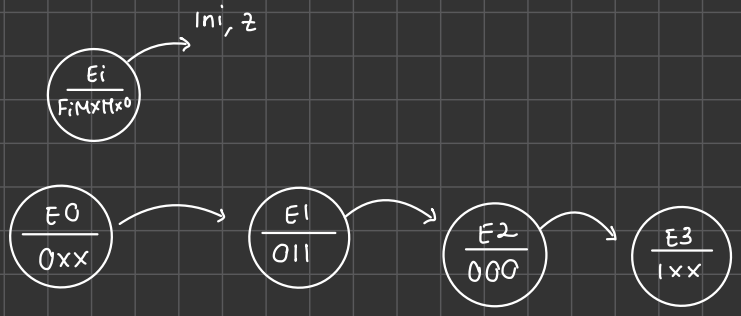
X (C+N+1) = DADA N-1

W (C+N+2) = Resultat → Fi = 1

Cicle	00	01	02	03	04	05	06	07	08	09	10	11	12
x	0x3A	0x02	0x34	0x7A	0x50	0xFA	0x03	0x04	0xC8	0x23	0x1F	0xFc	0x9E
Ini	1	0	1	0	0	0	1	0	0	1	1	0	1
fi	0	0	0	0	1	0	0	0	0	0	0	0	1
W	xx	xx	0x00	0x34	0xAE	xx	xx	xx	0x00	0xC8	0xEB	0x0A	0x06

2)

E	Ini	Z	E+
E0	0	0	E0
E0	0	1	E0
E0	1	0	E1
E0	1	1	E1
E1	0	0	E2
E1	0	1	E2
E1	1	0	E1
E1	1	1	E1
E2	0	0	E2
E2	0	1	E3
E2	1	0	E1
E2	1	1	E1
E3	0	0	E0
E3	0	1	E0
E3	1	0	E1
E3	1	1	E1



Ini (C) = 1 → X (C+1) = N

X (C+2) = DADA 0 ; X (C+N+1) = DADA N-1

W (C+N+2) = RESULTAT → fi = 1

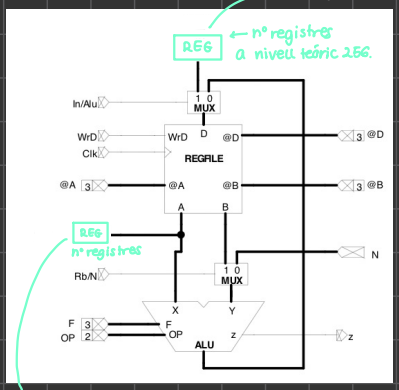
Tenim en compte Ini, si Ini = 1 tornem a començar

Cicle	00	01	02	03	04	05	06	07	08	09	10	11	12
x	0x3A	0x02	0x34	0x7A	0x01	0x01	0x0A	0x12	0x05	0x30	0x8D	0xF1	0x52
Ini	1	0	1	0	1	0	0	1	0	0	0	0	1
fi	0	0	0	0	0	0	0	1	0	0	0	0	0
W	xx	xx	0x00	0x34	0x00	0x01	0x00	0x0A	xx	0x00	0x30	0x BD	0x AE

- TEMA 9: UPG ENTRADES I SORTIDES -

* PERIFÈRICS

↳ els volem per interactuar amb l'exterior (usuaris, medi) i són dispositius asincrons



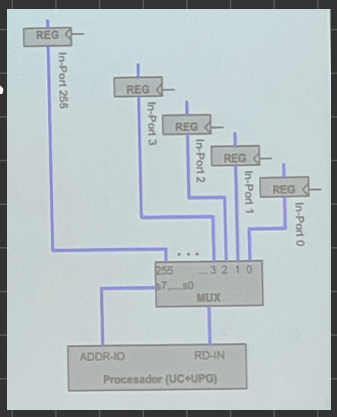
Es connecta a un teclat per exemple

Conectat a una impresora per exemple

* Aquests registres d'entrada i sortida els anomenarem ports i cada un dels n registres està conectat a un element diferent i s'anomenen del 0-256.

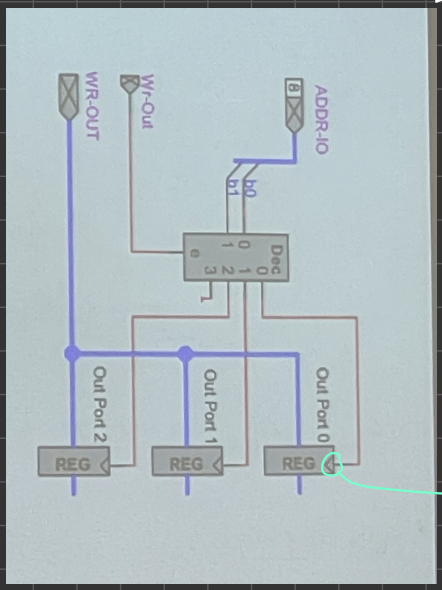
ex: IN 3 Rd1 → Reg 1
↳ port 3

PORTS ENTRADA ⟹



Aquests ports estaràn conectats a un multiplexor de 8 bits per què $2^8 = 256$!

PORTS DE SORTIDA



Volem decidir quan s'escriuen els ports (wrd)

* Si no volem modificar cap port wrd = 0.
* Si ho volem modificar wrd = 1, no es pot guardar valors al mateix port. S'ha de reseteajar el valor del port (0 → 1) ✓ (1 → 1) ✗.

→ Ho considerem una senyal de CÀRREGA!!

★ EL CLOCK D'AQUESTS PORTS ÉS DIFERENT AL DE LA RESTA DE REGISTRES DE LA UPG.

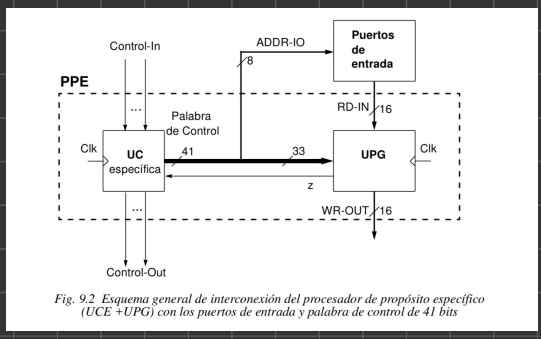


Fig. 9.2 Esquema general de interconexión del procesador de propósito específico (UCE +UPG) con los puertos de entrada y palabra de control de 41 bits

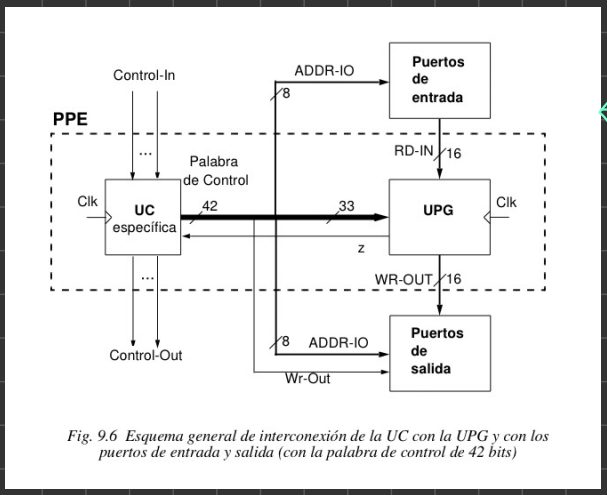


Fig. 9.6 Esquema general de interconexión de la UC con la UPG y con los puertos de entrada y salida (con la palabra de control de 42 bits)

* REUTILITZEM EL ADDR-IO per introduir els 8 bits d'entrada i de sortida

Compodem observar hem pasat de 41 bits de UC a 42 bits de UC això es per el ADDR-IO necesitem un bit extra per indicar si va als ports d'entrada o de sortida.

NOVA PARAULA DE CONTROL

@A	@B	Rb/N	OP	F	In/ALU	@D	WrD	Wr-Out	N	ADDR-IO (nexa)
xxx	xxx	x	xx	xxx	x	xxx	x	x	xxxx	xx

PROTOCOL HANDSHAKE DE 4 PASES

(Aka: protocol comunicació asincron)

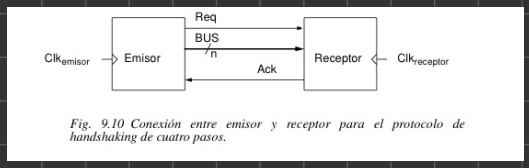


Fig. 9.10 Conexión entre emisor y receptor para el protocolo de handshaking de cuatro pasos.

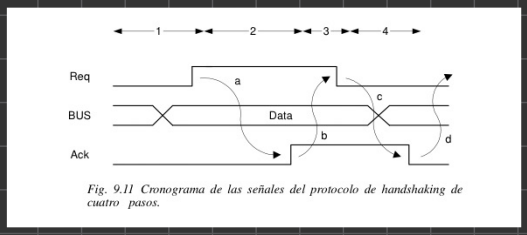


Fig. 9.11 Cronograma de las señales del protocolo de handshaking de cuatro pasos.

* EL EMISOR MAI PODRÁ ENVIAR UN ALTRE DADA FINS QUE Req = 0 i Ack = 0.

- Quan Req = 1 el emisor avisa al receptor de que entra una dada quan el receptor veu i tracta aquesta dada el Ack = 1.
- Quan el ACK = 1 tornem el Req = 0 com a espera, fins que Ack no torni al valor 0 Req es manté en 0.
- Quan els dos passen a 0 uavors tornem a l'estat d'inici.

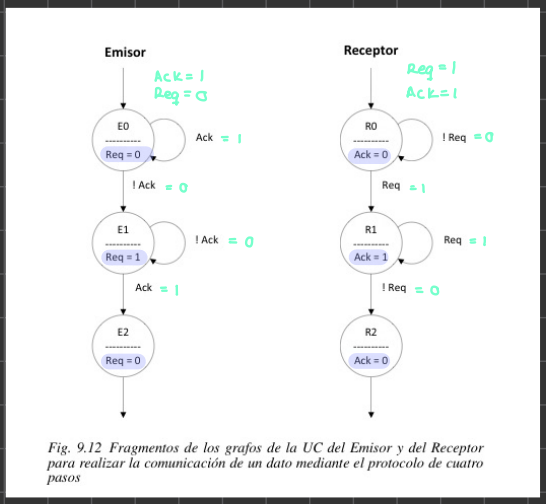
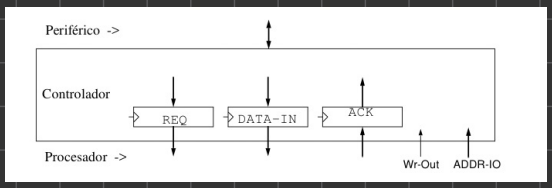


Fig. 9.12 Fragmentos de los grafos de la UC del Emisor y del Receptor para realizar la comunicación de un dato mediante el protocolo de cuatro pasos



HANDSHAKING amb ports

1 port per Req

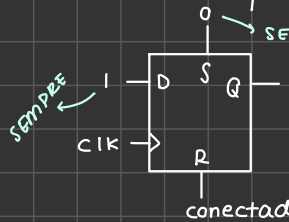
* GRAF UC HANDSHAKE

```
IN   R2, REQ          ; lectura Req
CMPEQI  -, R2, 1      ; ¿Req = 1?
IN   R3, DATA-IN      ; lectura dato
OUT  ACK, R2          ; Ack = 1
IN   R2, REQ          ; lectura Req
CMPEQI  -, R2, 0      ; ¿Req = 0?
OUT  ACK, R2          ; Ack = 0
```

NOU TIPUS DE BIESTABLE

BIESTABLE R/S — R de reset

En aquest biestable es pot forçar un valor sense tenir en compte el CLK.



Si S = 1 Q passa a valer 1

Si R = 1 Q pasa a valer 0

conectada a un senyar

SIMPLIFICAR L'ACCES ALS PORTS

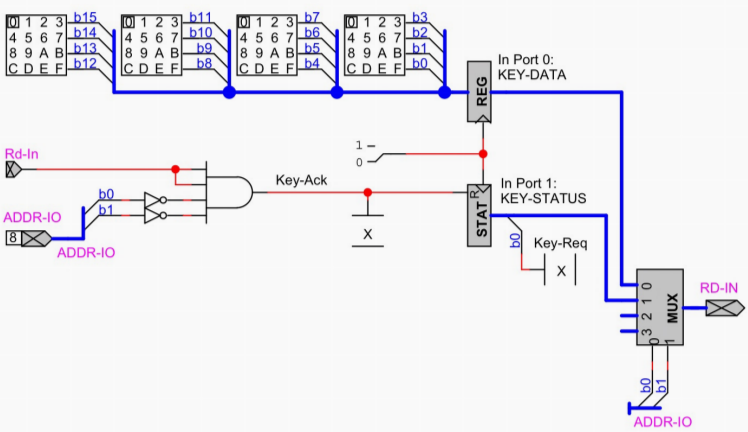


Fig. 9.17 Hardware de un modelo simplificado de teclado y su controlador. Como efecto lateral de la lectura del puerto de datos KEY-DATA *se pone a 0 el bit 0 del puerto* KEY-STATUS.

¡IMPORTANT!

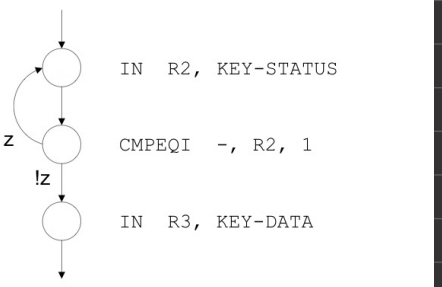


Fig. 9.19 Fragmento del grafo de estados para la entrada de un dato desde el teclado.

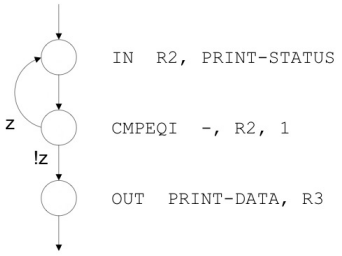
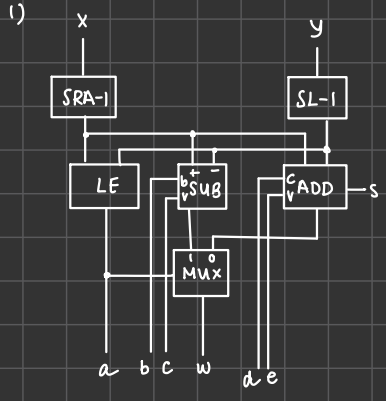


Fig. 9.21 Fragmento de grafo de estados para la salida de un dato por la impresora.

* CORRECCIÓ EXÁMEN PARCIAL

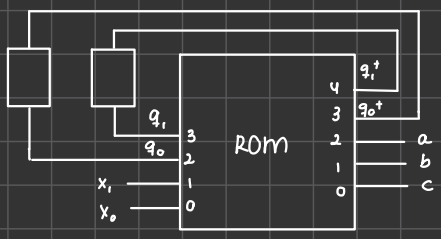
1)

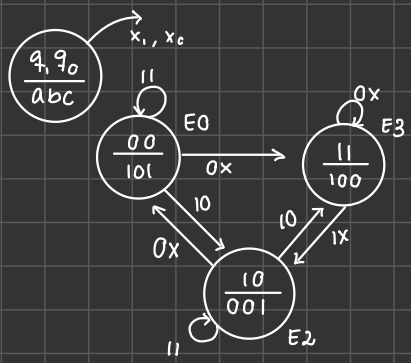


	s	x	y	a	b	c	d	e	w	w_u	w_s
①	0	1010	1010	1	0	0	1	0	1001	9	-7
②	1	1001	0000	1	0	0	0	0	1100	12	-4

① $1101 - 0100 = 1001$ (− ÷ −, OK) $\quad$ $1101 + 0100 = 10001$ (+ ÷ +, OK) $\quad$ LE = 1

2)

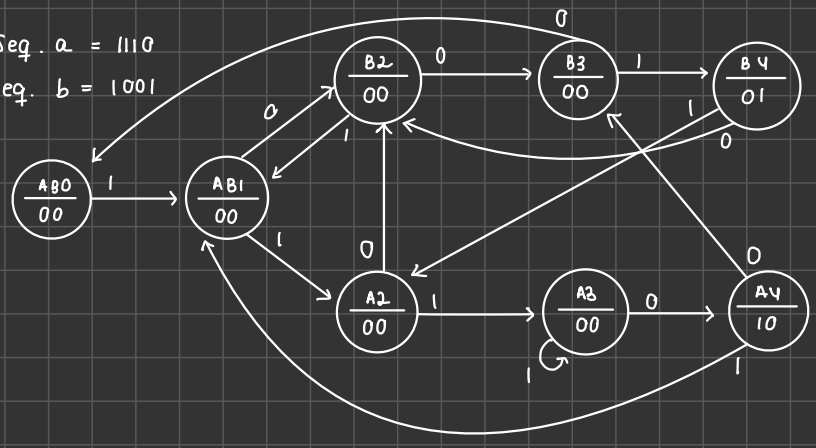




q_1	q_0	x_1	x_0	q_1^+	q_0^+	a b c
0	0	0	0	1	1	101
0	0	0	1	1	1	
0	0	1	0	1	0	
0	0	1	1	0	0	
0	1	0	0	x	x	xxx
0	1	0	1	x	x	
0	1	1	0	x	x	
0	1	1	1	x	x	
1	0	0	0	0	0	001
1	0	0	1	0	0	
1	0	1	0	0	1	
1	0	1	1	0	0	
1	1	0	0	1	1	100
1	1	0	1	1	1	
1	1	1	0	1	0	
1	1	1	1	1	0	

3) Seq. a = 1110

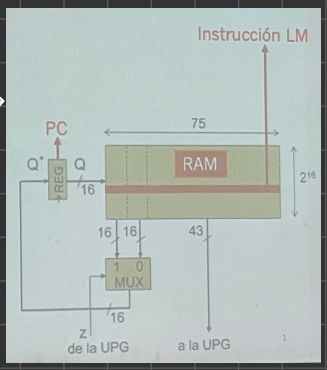
Seq. b = 1001

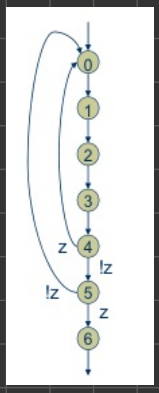


- TEMA 10: UCG -

* Per poder fer tot tipus de grafs canviant la ROM per una RAM.
* Ja no parlem d'estats

Model Harvard → UPDATE → MODEL DE VON NEWMAN
↳ De 75 bits a 16 bits





* Graf típic

Direcció (hexa)	contingut (hexa)	(hexa)	(no codificat)
0000	0001	0001	Pal. control nodo 0
0001	0002	0002	Pal. control nodo 1
0002	0003	0003	Pal. control nodo 2
0003	0004	0004	Pal. control nodo 3
0004	0005	0005	Pal. control nodo 4
0005	0006	0006	Pal. control nodo 5
0006	0007	0007	Pal. control nodo 6

con1	con0	z	TknBr
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

El PC es el que et diu en quin estat estas

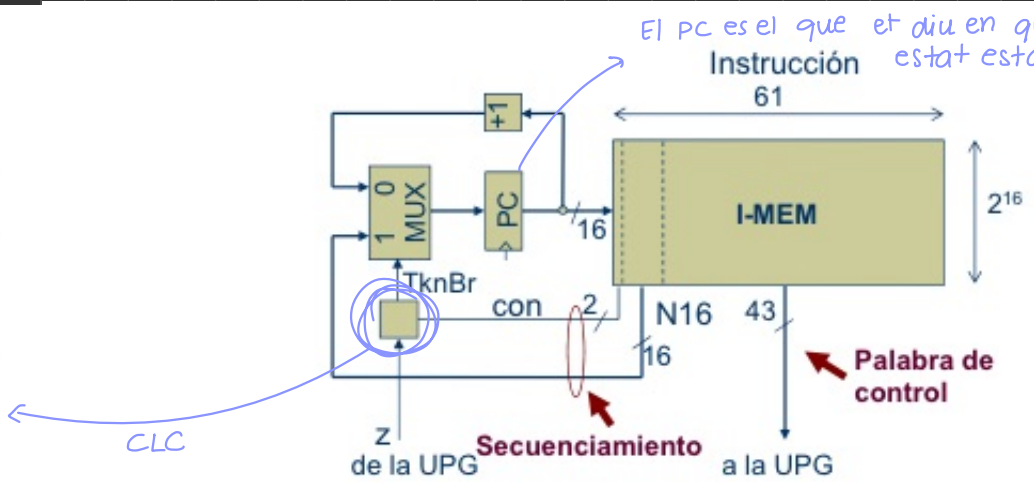
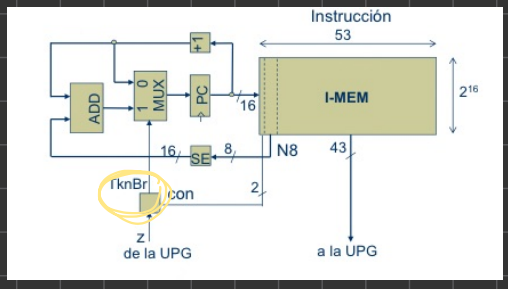
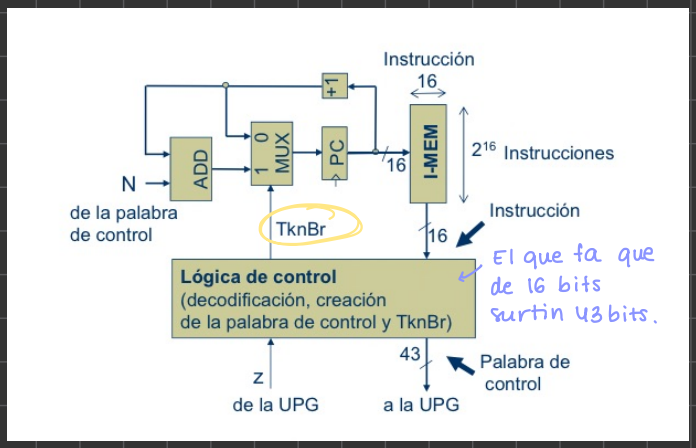
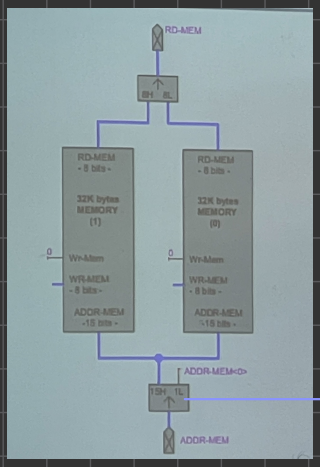


Fig. 10.7 Unidad de control con secuenciamiento semiimplícito absoluto. Otra forma de dibujar el circuito.

* La majoria de salts son curts
↳ Podem guardar la distància i no la direcció absoluta
↳ PC = PC + 1 + SE(N8) enter
↳ Bloc sign extension







* 2 memòries de 8bits juntes per aconseguir 16 bits

PC = PC + 2
Cambia el bloc +1 a +2
→ bit de menor pes

F la F depen de la instrucció

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0	Name	Mnemonic	Format
0 0 0 0	a a a	b b b	d d d	f f f	Logic and Aritmetic Operations	AND, OR, XOR, NOT, ADD, SUB, SHA, SHL	3R
0 0 0 1	a a a	b b b	d d d	f f f	Compare Signed and Unsigned	CMPLT, CMPLE, -, CMPEQ, CMPLTU, CMPLEU, -, -	
0 0 1 0	a a a	d d d	n n n n n n		Add Immediate	ADDI	
0 0 1 1							
0 1 0 0							
0 1 0 1					Memory future extensions		2R
0 1 1 0							
0 1 1 1					Branch future extension		
1 0 0 0	a a a	0	n n n n n n n n		Branch on Zero	BZ	1R
		1			Branch on Not Zero	BNZ	
1 0 0 1	d d d	0	n n n n n n n n		Move Immediate	MOVI	
	a a a	1			Move Immediate High	MOVHI	
	d d d						
1 0 1 0	d d d	0	n n n n n n n n		Input	IN	
	a a a	1			Output	OUT	
1 0 1 1	x x x	x x x x x x x x x x			Future extensions		
1 1 x x							

n de 6 bits

Per fer un Branch tmb F

Pero te la has d'inventar

n de 8 bits

* TAULA RESUM DEL FORAMAT I CODIFICACIÓ INSTRUCCIONS SISA

★ Falten els bits de la OP per tant haurem de inventarcos a través de la ROM!

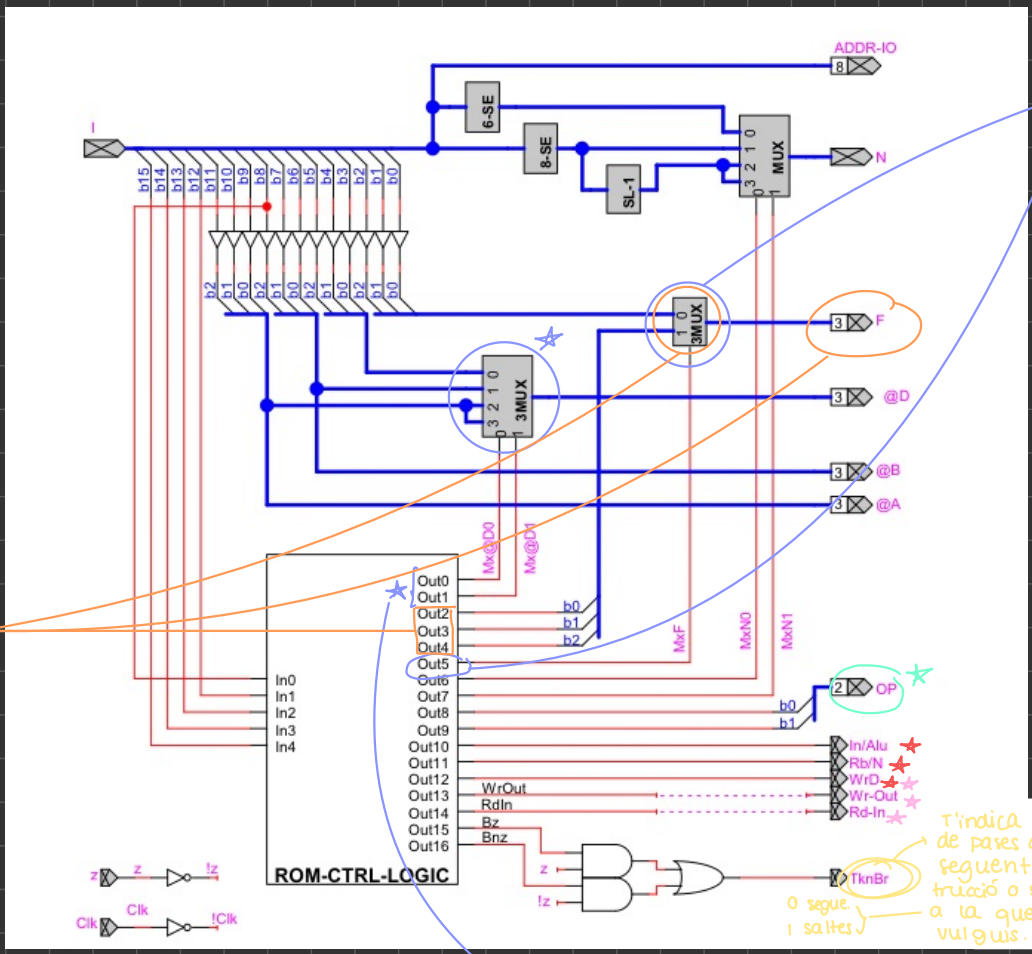
★ També ens falten les dades In/Alu, Rb/N, Wrd tmb a la ROM

★ Wrd, Wr-Out, Rd-In mai son x sempre seràn o "0" o "1" a la ROM.

★ @D pot estar als bits 3-5, 6-8, 9-11 depenent de la instrucció

això es gestiona amb un MUX ★

n de 8 bits multiplicat per 2. Es multiplica per dos pq és un salt (una instrucció requereix 2 bytes → 16 bits!)



MUX QUE DONA LA N (ja que poden ser 6 o 8 bits)
Tmb fem servir la ROM

* Tmb USEM UN MUX
* Aquests bits es fan servir per trovar la F en aquelles operacions on no està però es necesita (Branch, MOV...)

TknBr: t'indica si has de passar a la següent instrucció o saltes a la que vulguis.
0 segueix
1 salta

→ També afegim una ROM per gestionar @D.

→ Tria la D que has de triar

TAULA DE CONTINGUTS DE LA ROM-CTRL-LOGIC (HI HA QUE SABESELA!!)

Dirección					Contenido																	
In4	In3	In2	In1	In0	Out16	Out15	Out14	Out13	Out12	Out11	Out10	Out9	Out8	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0	
I<15>	I<14>	I<13>	I<12>	I<8>	Bnz z=0	Bz z=1	RdIn	WrOut	WrD	Rb/N	In/Alu	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	
0	0	0	0	x	0	0	0	0	1	1	0	0	0	x	x	0	x	x	x	0	0	AL
0	0	0	1	x	0	0	0	0	1	1	0	0	1	x	x	0	x	x	x	0	0	CMP
0	0	1	0	x	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	ADDI
0	0	1	1	x																		-
0	1	0	0	x																		-
0	1	0	1	x																		-
0	1	1	0	x																		-
0	1	1	1	x																		-
1	0	0	0	0	0	1	0	0	0	x	x	1	0	1	0	1	0	0	0	x	x	BZ
1	0	0	0	1	1	0	0	0	0	x	x	1	0	1	0	1	0	0	0	x	x	BNZ
1	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	1	1	0	MOVI
1	0	0	1	1	0	0	0	0	1	0	0	1	0	0	1	1	0	1	0	1	0	MOVHI
1	0	1	0	0	0	0	1	0	1	x	1	x	x	x	x	x	x	x	x	1	0	IN
1	0	1	0	1	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	OUT
1	0	1	1	x																		-
1	1	x	x	x	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	NOP

z=1 ★ Per fer salt

z=0

Fig. 10.35 Tabla de verdad compacta del contenido de la ROM-CTRL-LOGIC de la lógica de control.

★ Per augmentar la capacitat afegim un mòdul de memoria més lent i d'acces diferent.

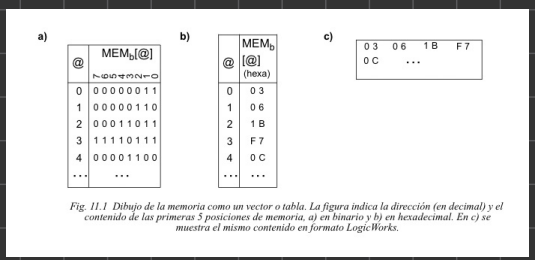
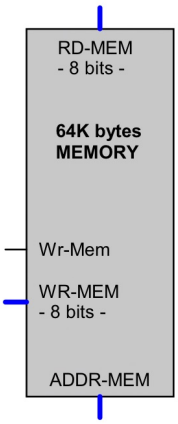


Fig. 11.1 Dibujo de la memoria como un vector o tabla. La figura indica la dirección (en decimal) y el contenido de las primeras 5 posiciones de memoria, a) en binario y b) en hexadecimal. En c) se muestra el mismo contenido en formato LogicWorks.

En un cicle només podem o llegir o escriure un byte.



RD-MEM-8bits (llegir memoria): bus de 8 bits de sortida del mòdul

WR-MEM-8 bits (escriure memoria): Entrada al mòdul

ADDR-MEM (direcció memoria): bus de 16 bits d'entrada (però la usem de 15)

Wr-mem (permís d'escritura). senyal d'un bit d'entrada

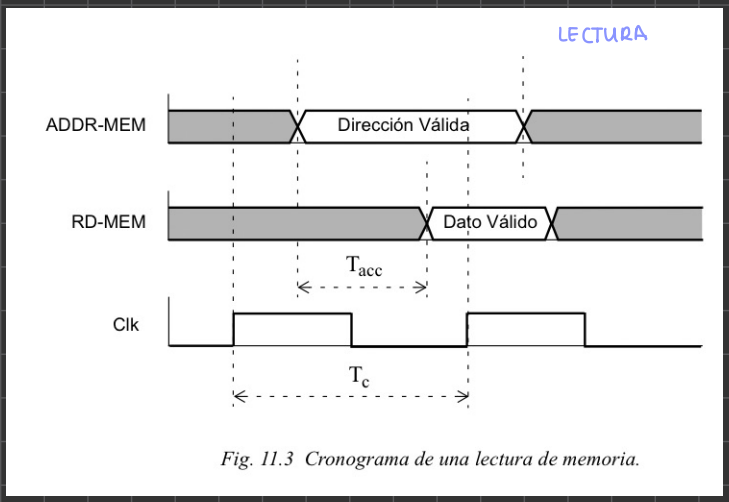


Fig. 11.3 Cronograma de una lectura de memoria.

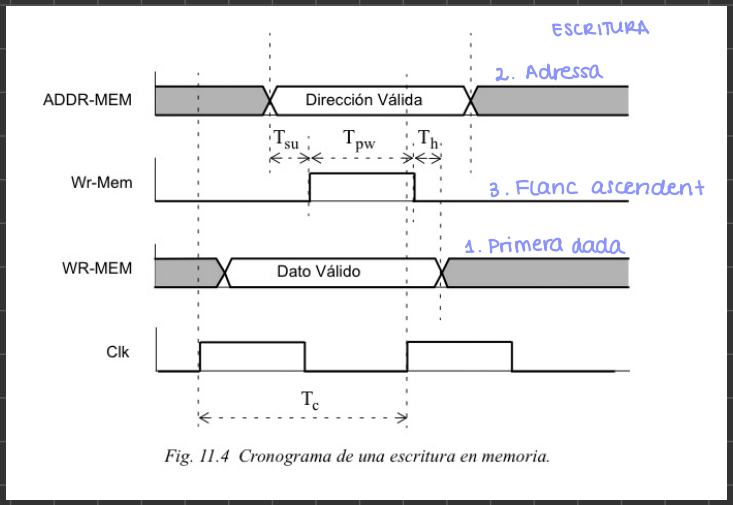


Fig. 11.4 Cronograma de una escritura en memoria.

LECTURES

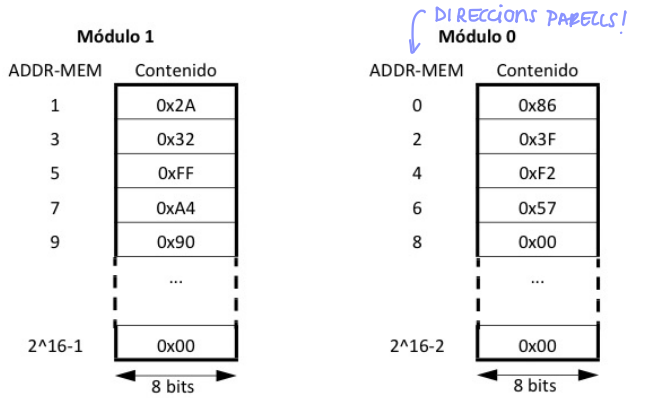
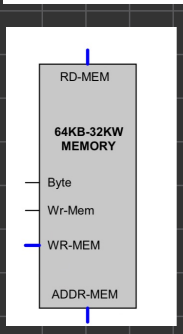


Fig. 11.5 Memoria entrelazada. Asignación de direcciones de memoria (de 16 bits) a dos módulos de memoria. Cada uno con una capacidad de 2^15 bytes.

* Conceptualment necessites dos moduls per guardar 16 bits (el hardware específic el veurem més endevant).
Per llegir 16 bits les dades han d'estar alineades. I la direcció ha de començar per direccions parells.



RD-MEM: Read Memory. Bus de 16 bits de salida del bloque.
WR-MEM: Write Memory. Bus de 16 bits de entrada al bloque.
ADDR-MEM: Address Memory. Bus de 16 bits de entrada al bloque.
Wr-Mem: Permiso de escritura. Señal de 1 bit de entrada al bloque.
Byte: Control de acceso a Byte o a Word. Señal de 1 bit de entrada al bloque.

8 bits (Byte = 1) — 16 bits (Byte = 0)

* La direcció es fà dividint la posició del modul entre 2.

* bits high (H)
* bits low (L)

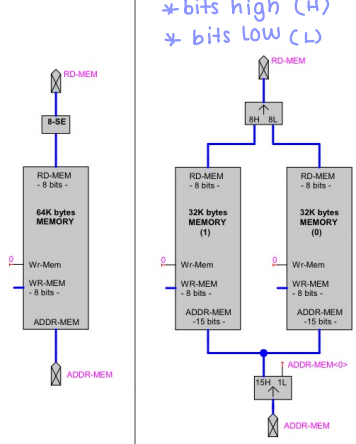


Fig. 11.7
Fase 1 paso 1.
Solo lectura
de solo byte

Fig. 11.8
Fase 1 paso 2.
Solo lectura
de solo word

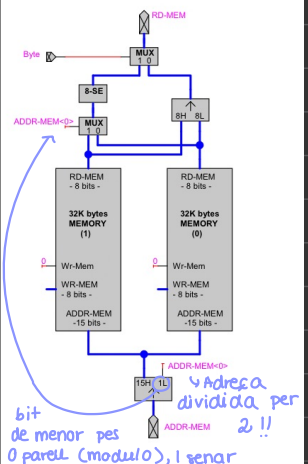


Fig. 11.9
Fase 1 paso 3.
Solo lectura
de byte o de word

* memòries de 2^{15}.

ESCRIPTURES

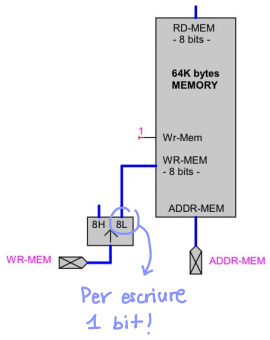


Fig. 11.10
Fase 2 paso 1.
Solo escritura
de solo byte

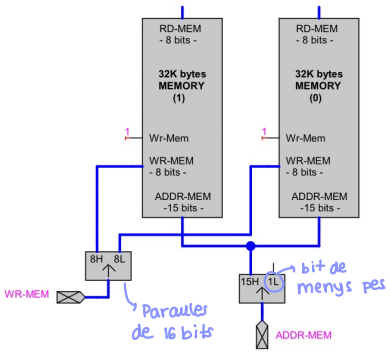


Fig. 11.11
Fase 2 paso 2.
Solo escritura
de solo word

El circuito interno para la fase 2 (solo escritura) paso 3 (de bytes o de words) se ve en la figura 11.12

★ El bus d'entrada SEMPRE serà de 16 bits per tant quan vols escriure un byte (8 bits) només agafem els 8 bits de menys pes

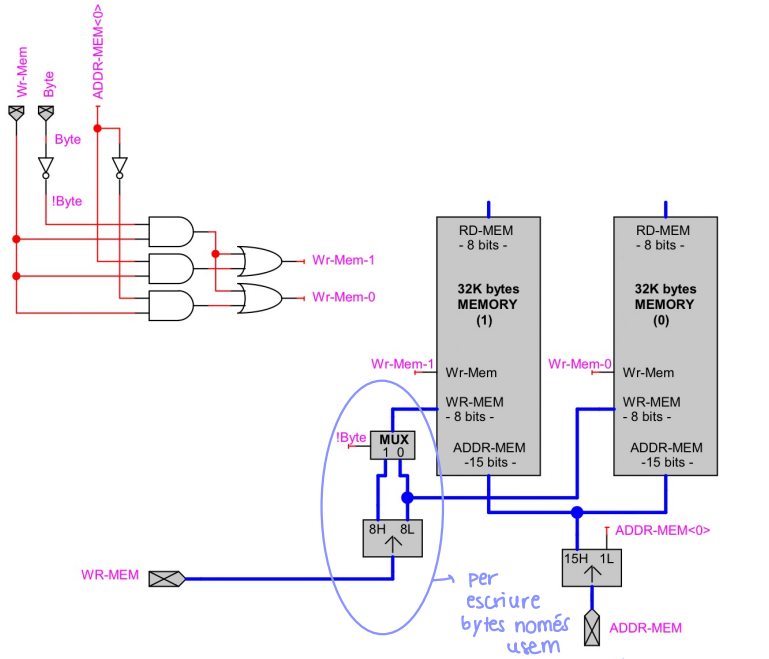


Fig. 11.12 Fase 2 paso 3. Solo escritura de byte o word.

★ Si la adreça es senar Wr-mem-1 ha de valdre 1 i Wr-mem-0 ha de valdre 0. Si es parell (16 bits) els dor Wr-mem valen 1.

Wr-Mem	Byte	ADDR-MEM<0>	Wr-Mem-1	Wr-Mem-0
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	1	0

Fig. 11.13 Tabla de verdad de las señales de permiso de escritura de cada módulo de memoria.

UPG + IO + ADDR-MEM

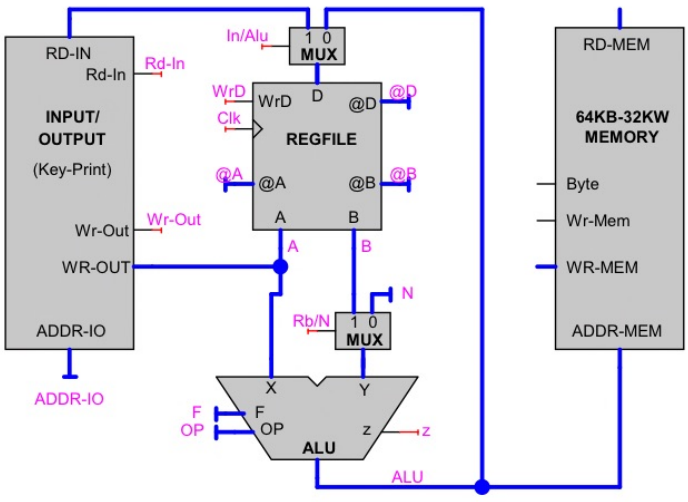


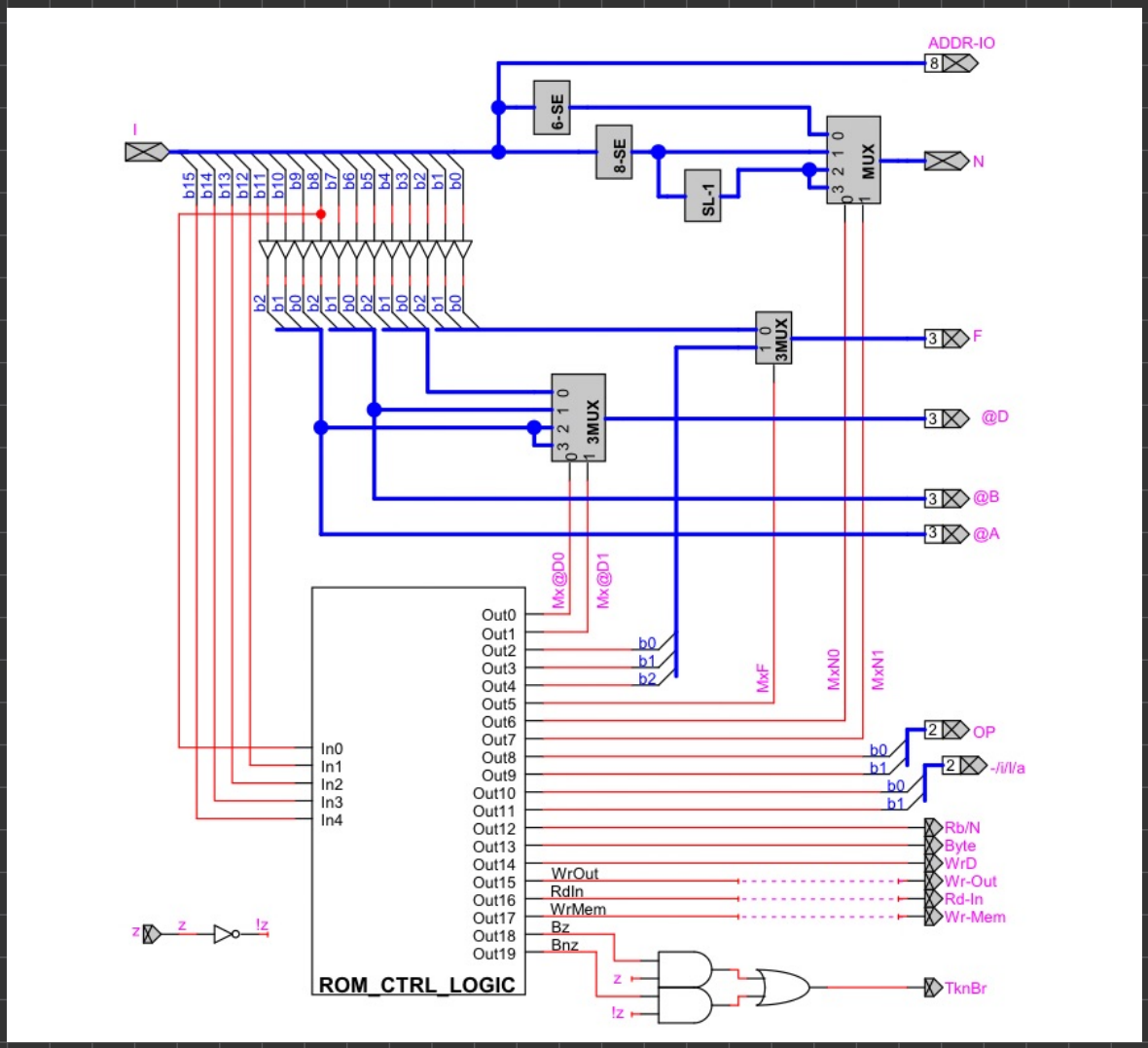
Fig. 11.18 Conexión de la UPG+IO *con el bus de memoria* ADDR-MEM.

LD R2, 10(R6)

@A	@B	Rb/n	OP	F	-/i/l/a	@D	Wrd	Wr-out	Rd-in	Wr-mem	Byte	TknBr	N	ADDR-10
110	xxx	0	00	100	01	010	1	0	0	0	0	0	00A	xx

STB -3(R4), R7

@A	@B	Rb/n	OP	F	-/i/l/a	@D	Wrd	Wr-out	Rd-in	Wr-mem	Byte	TknBr	N	ADDR-10
100	111	0	00	100	xx	xxx	0	0	0	1	1	0	FFFD	xx



* LD (load)
* LDB (load byte)

} de Mem a Rg

* ST (storage)
* STB (storage byte)

} de Rg a mem

SEMÀNTICA

$Rd \leftarrow Mem_w[((Ra + SE(N6)) \; i \; (\sim 1))] \equiv Ra + SE(N6)$ AND 1111 1111 1111 1110

Rd: direcció

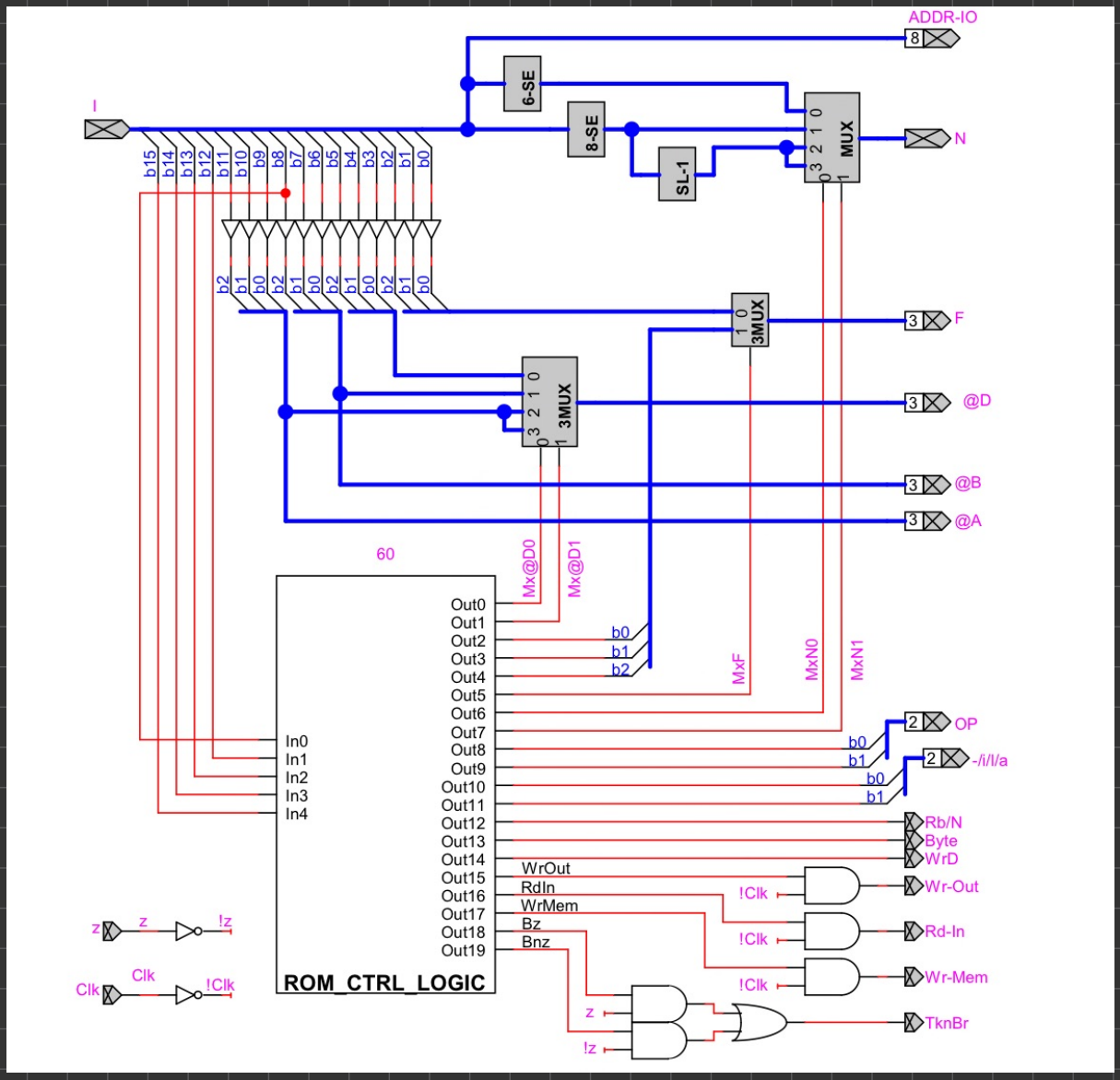
Adreça: $((Ra + SE(N6)) \; i \; (\sim 1))$

word (16 bits)

negat!!

$Ra + SE(N6)$: 0000 0000 0000 0111 / 0000 0000 0000 0110

DISENY ACABAT DE LA ARQUITECTURA PONT (INTRO A LA ARQUITECTURA FINAL)



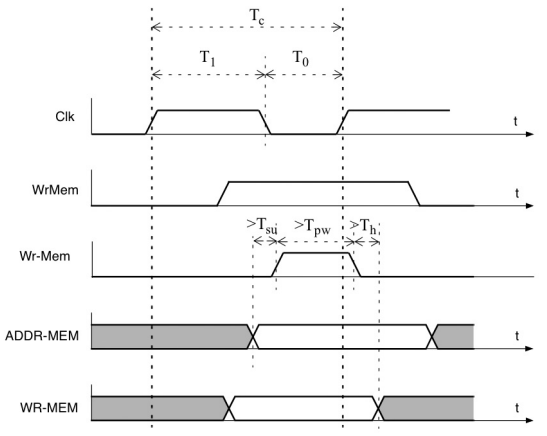


Fig. 12.6 Cronograma de una escritura en la memoria de datos del SISC Harvard Unicíclo.

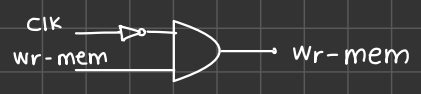
← CRONOGRAMA DEL UNICICLE

T_p Wr-Mem = quan es possa a 1 ⇒ 960 ut

T_p Wr-mem = s'estabilitza a 1030 ut

T_p Addr-Mem = s'estabilitza a 1950 ut

* Solució a aquest problema de T_p:

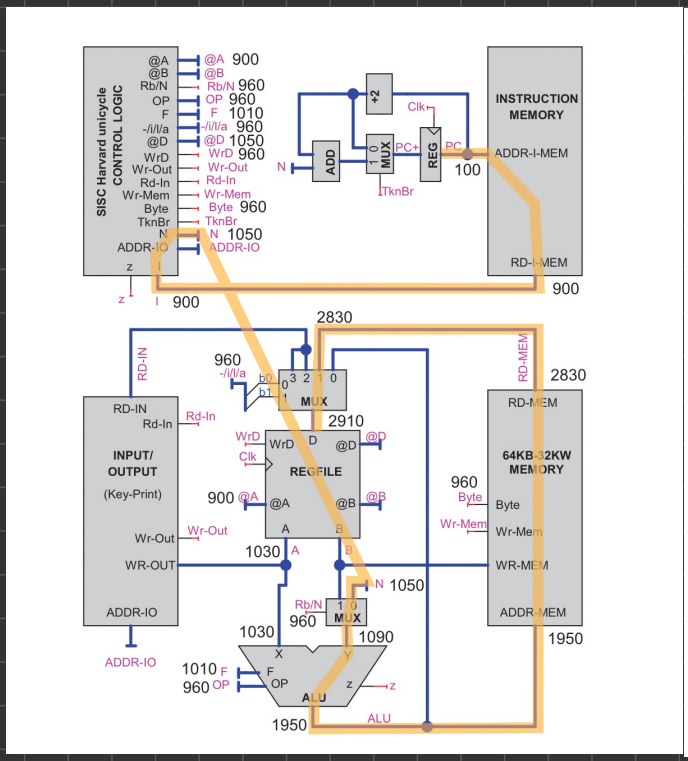


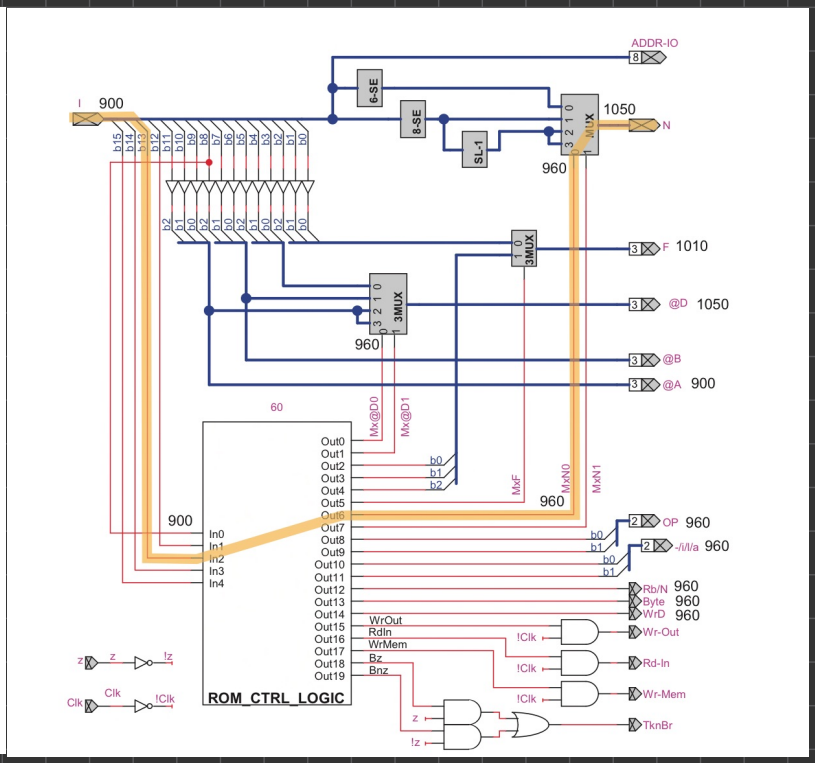
* També ens apareixen problemes amb Rd-In i Wr-Out per tant usem el mateix mètode que per solucionar el T_p de Wr-mem

Rd-In = !Clk · Rd-In

Wr-Out = !Clk · Wr · Out

Camí crític del hardware definitiu (TP = 2200 ut)





* El temps de cicle normal és 3000 ut
  ↳ ADDR-MEM s'estabilitza a 1950 ut després de l'inici del cicle, per tant farem 2000 + 1000 clk aritmètic.

TEMPS D'EXECUCIÓ:

Aquestes instruccions s'executen 1 vegada

```
        MOVI    R1, 0x00
        MOVHI   R1, 0x50        ; R1: puntero al primer elemento de A
        MOVI    R2, 0x00
        MOVHI   R2, 0x80        ; R2: puntero al primer elemento de B
        MOVI    R3, 0xE8
        MOVHI   R3, 0x03        ; R3: Número elementos, número de bytes
        ADD     R3, R3, R1      ; R3: punt. al último elem. de A, + 1
Bucle:  LDB     R4, 0(R1)       ; Lectura del elemento de A
        STB     0(R2), R4       ; Escritura en B
        ADDI    R1, R1, 1       ;Incremento puntero a A
        ADDI    R2, R2, 1       ; Incremento puntero a B
        CMPLTU  R5, R1, R3      ; ¿Se copió el último elemento?
        BNZ     R5, -6  ; Si no se copio, ir a bucle
```

Aquestes s'executen 1000 vegades

Per tant executem 6007 instruccions

$T_P = (7 + 6 \cdot 1000) \cdot 3000$

3000 ut / instrucció

Dividim les instruccions en inst. ràpides i inst. lentes

(1) inst. ràpides ⇒ 3 cicles → $3 \cdot 750 = 2250$ ut

(2) inst. lentes ⇒ 4 cicles → $4 \cdot 750 = 3000$ ut

Exemple: senyal per carregar el PC (al final sempre)

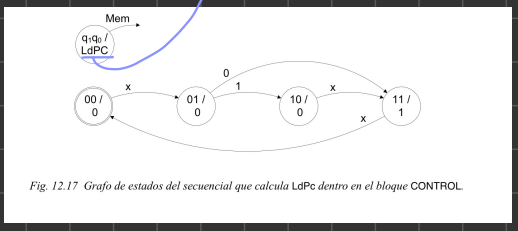
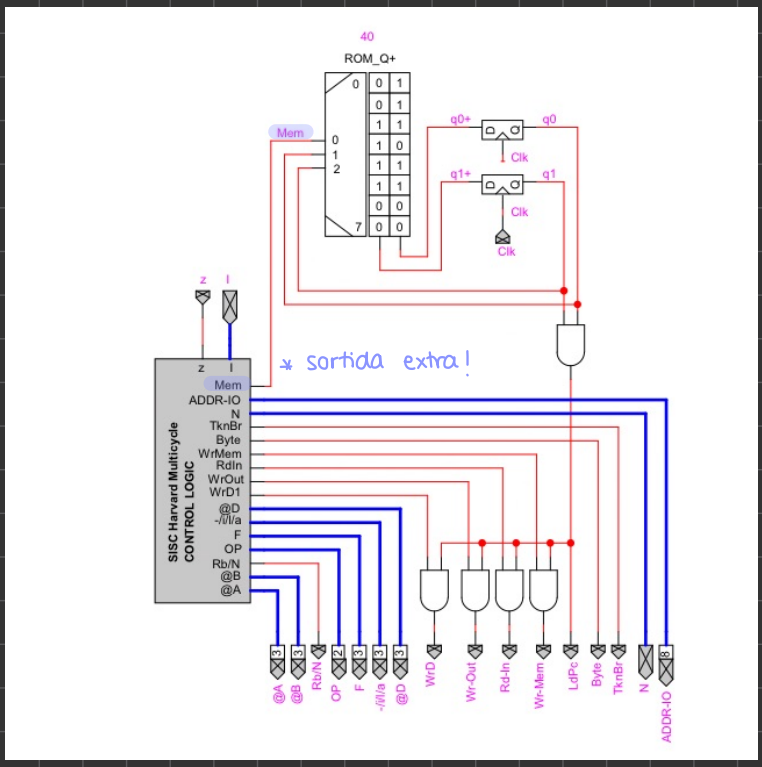


Fig. 12.17 Grafo de estados del secuencial que calcula LdPc *dentro en el bloque* CONTROL.



* Ens falta l'entrada d'aquest circuit (mem) pot ser inst. de memória o no. Per tant la Rom ens-ho indicarà, si mem=1 fem 4 cicles i és una instrucció de memória i si mem=0 fem 3 cicles i no ho és.

TEMPS D'EXECUCIÓ

```
                MOVI    R1, 0x00
                MOVHI   R1, 0x50        ; R1: puntero al primer elemento de A
                MOVI    R2, 0x00
                MOVHI   R2, 0x80        ; R2: puntero al primer elemento de B
                MOVI    R3, 0xE8
                MOVHI   R3, 0x03        ; R3: Número elementos, número de bytes
                ADD     R3, R3, R1      ; R3: punt. al último elem. de A, + 1
Bucle:          LDB     R4, 0(R1)       ; Lectura del elemento de A
                STB     0(R2), R4       ; Escritura en B
                ADDI    R1, R1, 1       ;Incremento puntero a A
                ADDI    R2, R2, 1       ; Incremento puntero a B
                CMPLTU  R5, R1, R3      ; ¿Se copió el último elemento?
                BNZ     R5, -6 ; Si no se copio, ir a bucle
```

7 ràpides

2 lentes 4 ràpides

Totes les instruccions de memória son lentes i les altres ràpidas a no ser que t'indiquin algo diferent a l'enunciat.

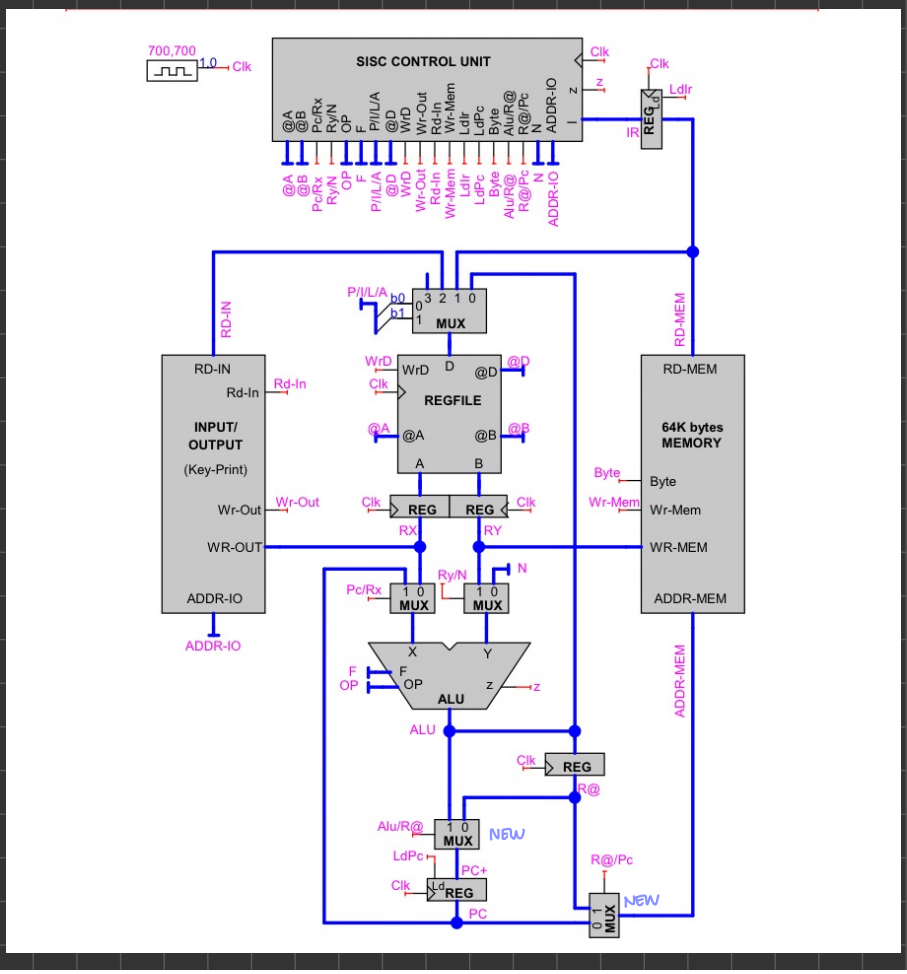
$T_P = 7R + 0L + (4R + 2L)N =$

$=$

★IMPORTANT
- Saber calcular el temps d'execució de les instruccions
- Entendre que una instrucció podem executarla en "n" cicles.

COMPUTADOR DE VON NEUMAN

* Característiques
  - Instruccions de 3 i 4 cicles
  - Té només un sumador
  - Cada cicle té una paraula de control diferent
* Ventatges
  - Alta eficiència (reutilització de components)
  - Alta versatilitat (fàcil afegir noves operacions)

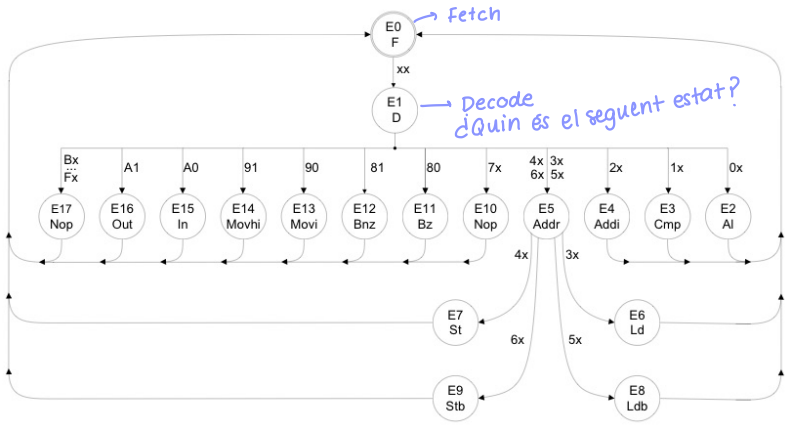


Lleugera modificació a la nostre unitat de procès:
- → Ara no tenim memòria d'instruccions
- → No tenim el bloc +2
- → No tenim el sumador que calcula la direcció de la següent instrucció

★ El PC està a la ALU
- ↳ PC ⇒ PC/Rx = 1
- ↳ Rx ⇒ PC/Rx = 0

★ Nou registre @R que guarda el que surt de la alu durant un cicle.

Canvis a la Unitat de control
FETCH i DECODE



CO,e → Q / ROMout

CO: Código de operación de la Instrucción en hexadecimal ($I_{15}I_{14}I_{13}I_{12}$)
e: Extensión del código de operación (I_8)
Q: Estado en hexadecimal
ROMout: Mnemotécnico que indica la salida de la ROM_OUT del CONTROL

Fig. 13.1 Grafo de estados de la unidad de control del computador SISA (sin JALR).

★ *Fetch*, F. A la vez que se va a buscar la instrucción actual se incrementa el PC en la ALU, dejándolo ya apuntando a la siguiente instrucción del programa. Si la instrucción actual no es de salto, después del ciclo F ya no debe modificarse más el PC, hasta que se vuelva otra vez a esta fase F cuando se inicie la búsqueda de la siguiente instrucción. Se está aprovechando este ciclo en el que la ALU no se usaba para nada para hacer un trabajo, incremento del PC, que debe hacerse en todas las instrucciones y que en esta máquina requeriría un ciclo extra de ejecución.

↳ Basicament entra la instrucció i a la vegada calcula el PC de la següent instrucció a la ALU. PC = PC+2

★ *Decode*, D. A la vez que se decide cuál será el siguiente ciclo/nodo de ejecución en función de la instrucción, en la ALU se suma el PC, que ya se ha dejado apuntando a la siguiente instrucción después del ciclo anterior, más el desplazamiento, N. Pero el valor calculado no debe cargarse en el PC al final de este ciclo ya que solo debe hacerse esto si se trata de una instrucción de salto (y eso no se sabe hasta el siguiente ciclo en el que ya se ha decodificado la instrucción) y si el salto es tomado (lo que se sabrá cuando el ciclo de ejecución propiamente dicho de la instrucción de salto, Bz o Bnz, se deje pasar el contenido del registro Ra por la ALU y la unidad de control sepa si se debe romper la secuencia o no). Por ello, se pone un nuevo registro en la unidad de proceso, que denominamos R@, el cuál se carga al final de D con el valor calculado de la dirección de la instrucción de salto tomado, por si luego hay que usar esta dirección para cargarla en el PC antes de pasar al *Fetch* de la siguiente instrucción. Podemos decir que, aprovechando que la ALU no tiene que hacer nada en el ciclo D, se adelanta el cálculo de la dirección de salto tomado a antes de saber, ni siquiera, si se está ejecutando una instrucción de salto. Si no se hiciera así, las instrucciones de salto necesitarían dos ciclos para su ejecución propiamente dicha, ya que en los dos se necesita la ALU: uno para dejar pasar Ra y saber si es cero o distinto de cero y el otro para calcular la dirección del salto tomado.

↳ Basicament llegeix la instrucció i calcula la direcció de salt (per si la instrucció fos de salt)

@R = PC+2 + SE(N8)·2

8 bits de menys pes de la instrucció, els extenc a 16 bits i els multipuco per 2.

* Suposem que el Decode llegeix una instrucció aritmetico lògica (AL)

Rd = RX AL RY → la OP està a la instrucció

Fem servir la Alu, com no es tracta d'un salt ignorem el PC i @R calculat anteriorment, i guardem el valor de la operació al registre destí.

* Suposem que el Decode llegeix una instrucció MOVI

Rd = SE(N8)

* Instrucció BZ

if (RX = 0) PC = @R

↓

Z=1

* Instrucció acces memòria

Estat Addr

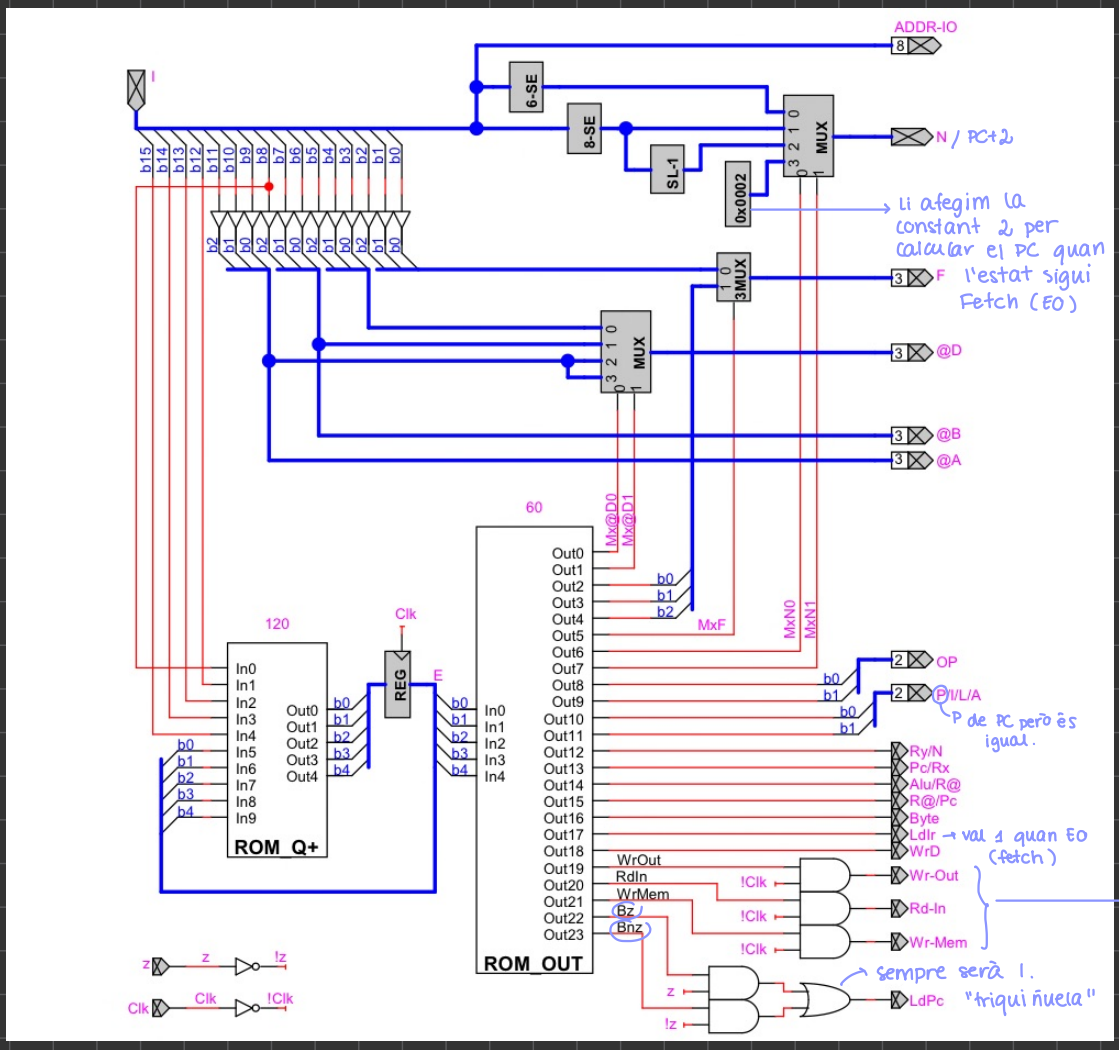
@R = RX + SE(N6)

LD / LDB / ST / STB

Rd = Mem_w [@R]

16-bit Instruction																Mnemonic	Format
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	a	a	a	b	b	b	d	d	d	f	f	f	AND, OR, XOR, NOT, ADD, SUB, SHA, SHL	3R
0	0	0	1	a	a	a	b	b	b	d	d	d	f	f	f	CMPLT, CMPLE, -,CMPEQ, CMPLTU,CMPLEU, -, -	
0	0	1	0	a	a	a	d	d	d	n	n	n	n	n	n	ADDI	2R
0	0	1	1	a	a	a	d	d	d	n	n	n	n	n	n	LD	
0	1	0	0	a	a	a	b	b	b	n	n	n	n	n	n	ST	
0	1	0	1	a	a	a	d	d	d	n	n	n	n	n	n	LDB	
0	1	1	0	a	a	a	b	b	b	n	n	n	n	n	n	STB	
0	1	1	1	a	a	a	d	d	d	x	x	x	x	x	x	JALR	
1	0	0	0	a	a	a	0	n	n	n	n	n	n	n	n	BZ	1R
							1									BNZ	
1	0	0	1	d	d	d	0	n	n	n	n	n	n	n	n	MOVI	
							1									MOVHI	
1	0	1	0	d	d	d	0	n	n	n	n	n	n	n	n	IN	
				a	a	a	1									OUT	

@A	@B	Pc/Rx	Ry/N	OP	F	P/I/L/A	@D	WrD	Wr-Out	Rd-In	Wr-Mem	Ldlr	LdPc	Byte	Alu/R@	R@/Pc	N (hexa)	ADDR-IO (hexa)



* El que decideix la paraula de control és l'estat en el que estas.

* I altrament es l'estat el que determina quina serà la paraula de control.

* Els multiplexors nous que han aparegut depenaràn de l'estat per tant sortiràn de la ROM.

Accions codig

MOVI R3, 2 → estat i : Fetch → IR = Mem_w[PC]
PC = PC+2

estat i+1 : Decode → RX = Ra , RY = Rb
R@ = PC + SE(N8)·2

estat i+2 : MOVI → R3 ← SE(2)

ADDI R3, R3, -1 → estat i+3 : Fetch → IR = Mem_w[PC]
PC = PC+2

estat i+4 : Decode → Rx = Ra , RY = Rb
R@ = PC + SE(N8)·2

estat i+5 : ADDI → R3 = 2 - 1 = 1 * ja que MOVI R3, 2

BNZ R3, -2 → estat i+6 : Fetch → IR = mem_w[PC]
PC = PC+2

estat i+7 : Decode → Rx = Ra , RY = Rb
R@ = PC + SE(N8)·2 = @ADDI

estat i+8 : BNZ → R3 = 1 ≠ 0 → saltem → PC = @R = @ ADDI

ADDI R3, R3, -1 → estat i+9 : fetch → IR = mem_w[PC]
PC = PC + 2

estat i+10 : Decode → Rx = Ra , RY = Rb
R@ = PC + SE(N8)·2

estat i+11 : ADDI → R3 = 1 - 1 = 0

BNZ R3, -2 → estat i+12 : Fetch → IR = mem_w[PC]
PC = PC + 2

estat i+13 : Decode → Rx = Ra , RY = Rb
R@ = PC + SE(N8)·2

estat i+14 : BNZ → R3 = 0

LD R3, 10(R3) → estat i+15 : Fetch → IR = mem_w[PC]
PC = PC + 2

estat i+16 : Decode → Rx = Ra , RY = Rb
R@ = PC + SE(N8)·2

estat i+17 : ADDR → Re = 0 + 10 = 10

estat i+18 : R3 = mem_w[10]

FETCH

R@/PC = 0 , Byte = 0 , LdIr = 1 , PC/RX = 1 , N = 0x0002 , Ry/N = 0
OP = 00 , F = 100 , AW/R@ = 1 , LdPC = 1

DECODE

- Decodificar instrucció
- Rx = Ra i RY = Rb
- R@ = PC + SE(N8)2

PARAULA DE CONTROL →

N = SE(IR<7..0>)2
Pc/Rx = 1, Ry/N = 0, OP = 00
F = 100
@A = I<b9 - b11> = 010
@B = I<b8 - b6> = 001

ARITMÈTICO LÒGIQUES

- Rd = RX AL RY → PARAULA DE CONTROL → Pc/Rx = 0, RY/N = 1, P/i/l/a = 00, OP = 00, WrD = 1

MOVI

- Rd = SE(N8) → PARAULA DE CONTROL → N = SE(IR<7...0>); Ry/N = 0, OP = 10, F = 001, P/i/L/a = 00, WrD = 1; @D = IR<11...9>

Bz

- if (RX == 0) PC = R@ → PARAULA DE CONTROL → PC/Rx = 0, OP = 10, F = 000, Alu/@R = 0; LdPC = z

ACCÉS MEMÒRIA

- Addr R@ = RX + SE(N6) → PARAULA DE CONTROL → N = SE(IR<5...0>); Pc/Rx = 0, Ry/N = 0, OP = 00, F = 100

LD

- Rd = Mem_w[R@] → PARAULA DE CONTROL → R@/PC = 1, Byte = 0, P/i/L/a = 01, WrD = 1

→ com que sempre LdPC = 1 (Pq calculem el PC) llavors Bz i Bnz = 1!

@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	
0	1	1	0	0	0	0	1	0	0	1	1	0	x	x	0	0	1	1	1	1	0	0	x	x	F
1	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	0	1	1	0	0	x	x	D
2	0	0	0	0	0	1	x	x	x	x	0	1	0	0	0	0	x	x	0	x	x	x	0	0	Al
3	0	0	0	0	0	1	x	x	x	x	0	1	0	0	0	1	x	x	0	x	x	x	0	0	Cmp
4	0	0	0	0	0	1	x	x	x	x	0	0	0	0	0	0	0	0	1	1	0	0	0	1	Addi
5	0	0	0	0	0	0	0	x	x	x	0	0	x	x	0	0	0	0	1	1	0	0	x	x	Addr
6	0	0	0	0	0	1	x	0	1	x	x	x	0	1	x	x	x	x	x	x	x	x	0	1	Ld
7	0	0	1	0	0	0	x	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	St
8	0	0	0	0	0	1	x	1	1	x	x	x	0	1	x	x	x	x	x	x	x	x	0	1	Ldb
9	0	0	1	0	0	0	x	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Stb
10	1	1	0	0	0	1	x	x	x	1	0	x	1	1	1	0	x	x	1	0	1	1	0	1	Jalr
11	0	1	0	0	0	0	x	x	x	0	0	x	x	x	1	0	x	x	1	0	0	0	x	x	Bz
12	1	0	0	0	0	0	x	x	x	0	0	x	x	x	1	0	x	x	1	0	0	0	x	x	Bnz
13	0	0	0	0	0	1	x	x	x	x	x	0	0	0	1	0	0	1	1	0	0	1	1	0	Movi
14	0	0	0	0	0	1	x	x	x	x	0	0	0	0	1	0	0	1	1	0	1	0	1	0	Movhi
15	0	0	0	1	0	1	x	x	x	x	x	x	1	0	x	x	x	x	x	x	x	x	1	0	In
16	0	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Out
17..31	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Nop

Tabla 13.1 Contenido completo de la ROM_OUT.

ROM Q+ [0x216] → 100001 0110 → E16 OUT

Q = 16 OUT

INSTRUCCIÓ JALR

- Salts incondicionals (NO depen de la z), pot tornar a la posició desde la que ha saltat
- Guarda el PC d'on ha saltat

JALR Rd, Ra ← direcció a la que vull saltar

PC = PC + 2

tmp = Ra and (~1) ← $bit_0 = 0$

Rd = PC

PC = tmp ← temporal

ex: MOVI R1, 27 — va a la
JALR Rd, R1 direcció de memória 27

* LLENGUATGE DE MÀQUINA 0111 aaa ddd xxxxxx ← NO S'UTILITZEN

PC = RX & (~1) // Rd = PC

```
ex: [0x1000] Subr: ADD R0, R1, R2
                   [Return]
                   JALR R7, R7
             main: MOVI R1, 0x2
                   MOVI R2, 0x5
                   [Call Subr]
                   MOVI R7 0x00
                   MOVHI R7, 0x10
                   JALR R7, R0
```

TEMPS DE CICLE DE LA VON NEUMAN → $T_c = 1400$

- TEMA IV: LLENGUATGE SISA -

SECCIONS

- Data
- Text
- End

DIRECTIVES

- Set Num_elem, 100 (no ocupa espai)
- Space <size>, <fill> → vector: space <5>, <0> 00000
- Byte <fill-1>, <fill-2>, ..., <fill-n> (8 bits)
- Word <fill-1>, <fill-2>, ..., <fill-n> (16 bits)
- Even (actua en espais parells)

ETIQUETES

```
    LD R1, 0(R3)
    BZ R1, etiq   ← salta a la etiqueta
    ADD R2, R0, R1
etiq: AND R1, R2, R
```

```
N = 1000
...
MOVI R2, lo(N)    lo ← low
MOVHI R2, hi(N)   hi ← high
```

lo(N) → pilla 8 bits de menys pes

hi(N) → pilla 8 bits de més pes