Representació de Naturals (N)

Representació decimal (Base 10)

5213 → $V_{10}(5213) = 5 \cdot 10^3 + 2 \cdot 10^2 + 1 \cdot 10 + 3 \cdot 1$

$5 \cdot 10^3$; $2 \cdot 10^2$; $1 \cdot 10$; 3

$$V_{10}(x) = \sum_{i=0}^{n-1} 10^i \cdot x_i$$

Representació binaria (base 2)

1001

$$V_{10}(x) = \sum_{i=0}^{n-1} 2^i \cdot x_i$$

Representació hexadecimal (base 16)

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

OF2A

$$V_{10}(x) = \sum_{i=0}^{n-1} 16^i \cdot x_i$$

Rang de representació

$b=10 \rightarrow [0,99]$

$b=2 \rightarrow [0,11]$

$b=16 \rightarrow [0,FF]$

$nbits = [0, b^n - 1]$

$b_{10} \rightarrow b_2$

33 | 2
1 16 | 2
0 8 | 2
0 4 | 2
0 2 | 2
0 1

100001

$b_2 \rightarrow b_{10}$

$100001 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5$

$= 33$

$b_2 \leftrightarrow b_{16}$

0001 0110 1111
1 6 F

A 2 3
1010 0010 0011

$b_{16} \rightarrow b_{10}$

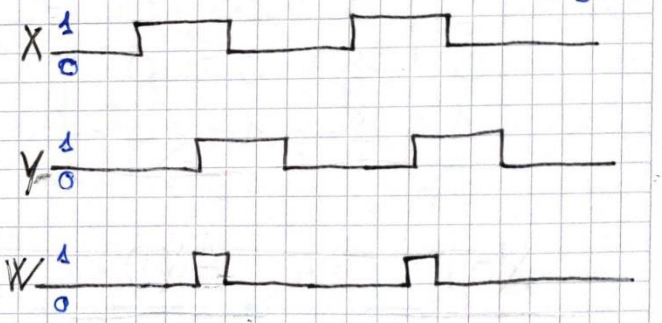
$A23 = 3 \cdot 16^0 + 2 \cdot 16^1 + 11 \cdot 16^2$

$= 2851$

0X23 ← Hexadecimal

Tema 3a — Circuits Combinacionals

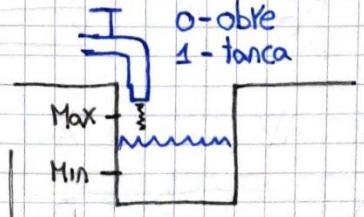
Cronograma



Taula de la veritat

X	Y	W
0	0	0
0	1	0
1	0	0
1	1	1

Ex:



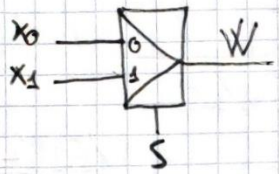
0-obre
1-tanca

⊗ 0-dia
1-nit

R	Max	Min	
0	0	0	1
0	0	1	X
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	0
1	1	1	0

0-humit
1-Sec

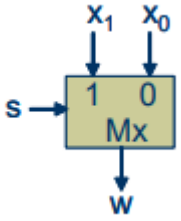
Multiplexor



Si $S=0 \rightarrow W = X_0$

Si $S=1 \rightarrow W = X_1$

S	X_0	X_1	W
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	0
1	1	1	1

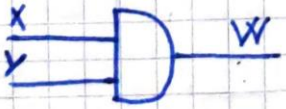


NOT

X	W
0	1
1	0

$W = !X$

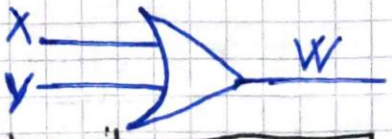
AND



X	Y	W
0	0	0
0	1	0
1	0	0
1	1	1

$W = X \cdot Y$

OR

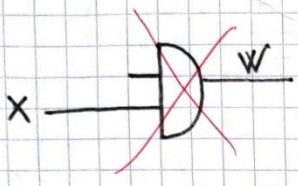


X	Y	W
0	0	0
0	1	1
1	0	1
1	1	1

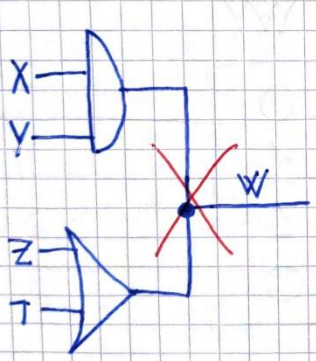
$W = X + Y$

Regles

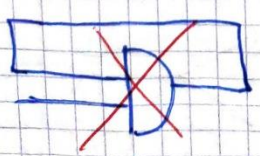
- No entrades Sense Conectar



- No Connectar dues Sortides directament

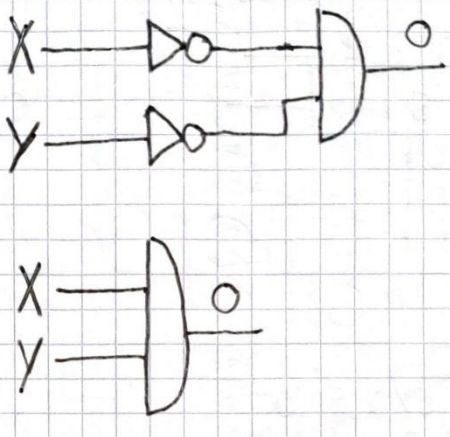


- No podem tenir cicles



Síntesi d'un Circuit Combinacional

	X	Y	W
M_0	0	0	1
	0	1	0
	1	0	0
M_3	1	1	1



Mintermn

$M_0 + M_3$

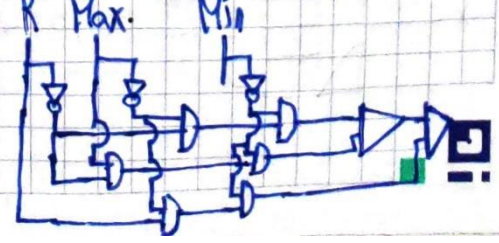
$$\overline{X} \cdot \overline{Y} + X \cdot Y$$

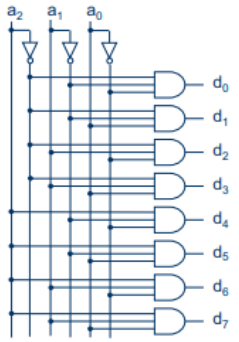
1) Multiplicació de Variables que donin 1
2) Summa de totes aquestes multiplicacions

R	Max	Min	A	E
0	0	0	1	0
0	0	1	X	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	X	0
1	1	0	0	1
1	1	1	0	0

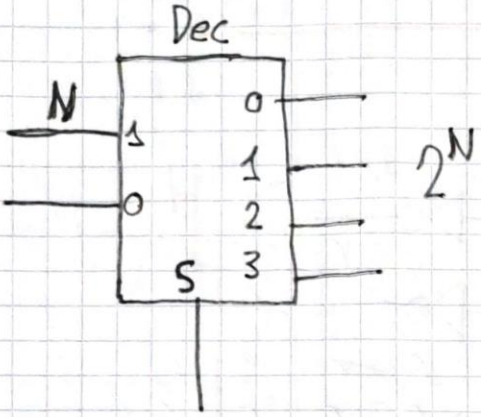
$$\overline{R} \cdot \overline{Max} \cdot \overline{Min} + \overline{R} \cdot Max \cdot \overline{Min} + R \cdot \overline{Max} \cdot \overline{Min}$$

$$R \cdot Max \cdot \overline{Min} + R \cdot \overline{Max} \cdot Min$$





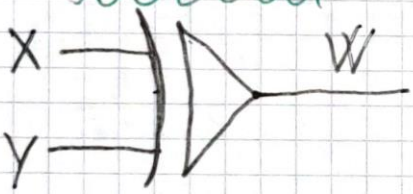
Decodificador



Si S=0 → Totes les Sortides són 0

Si S=1 → La sortida =1 quan el nombre en binari equival a la sortida natural

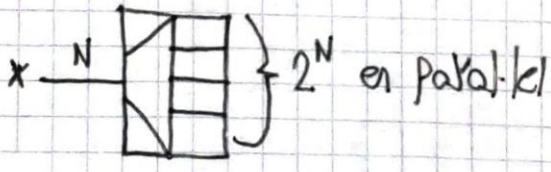
OR exclusiva

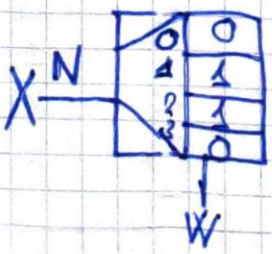


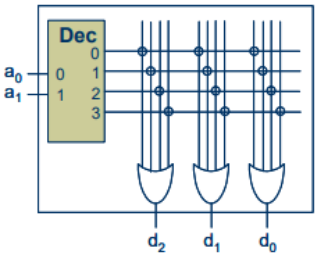
X	Y	W
0	0	0
0	1	1
1	0	1
1	1	0

$$W = A \cdot \bar{B} + \bar{A} \cdot B$$

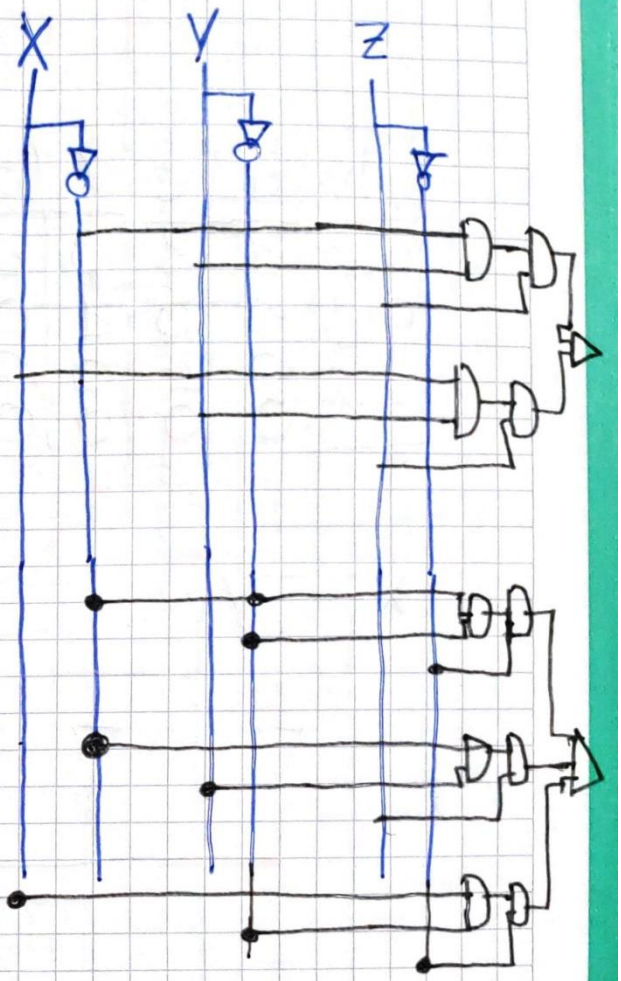
Romm

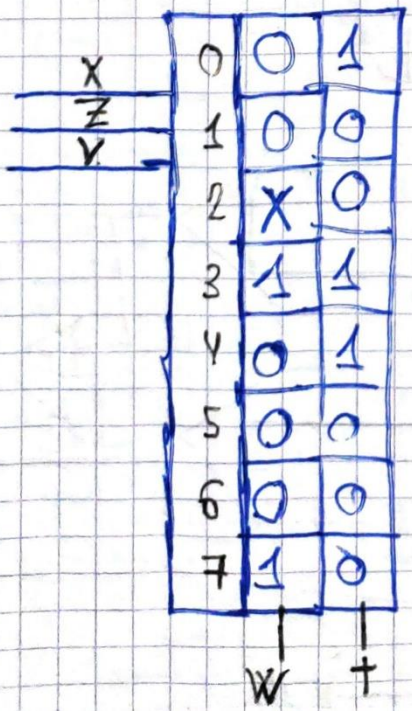


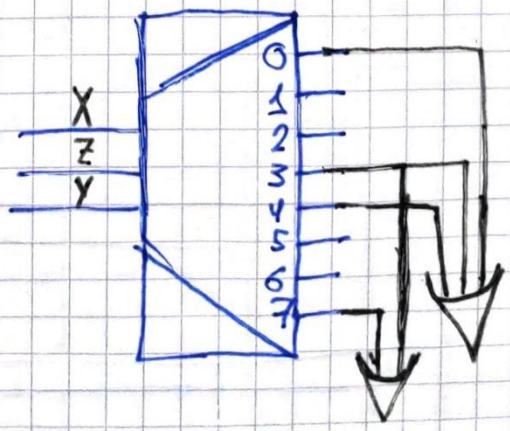


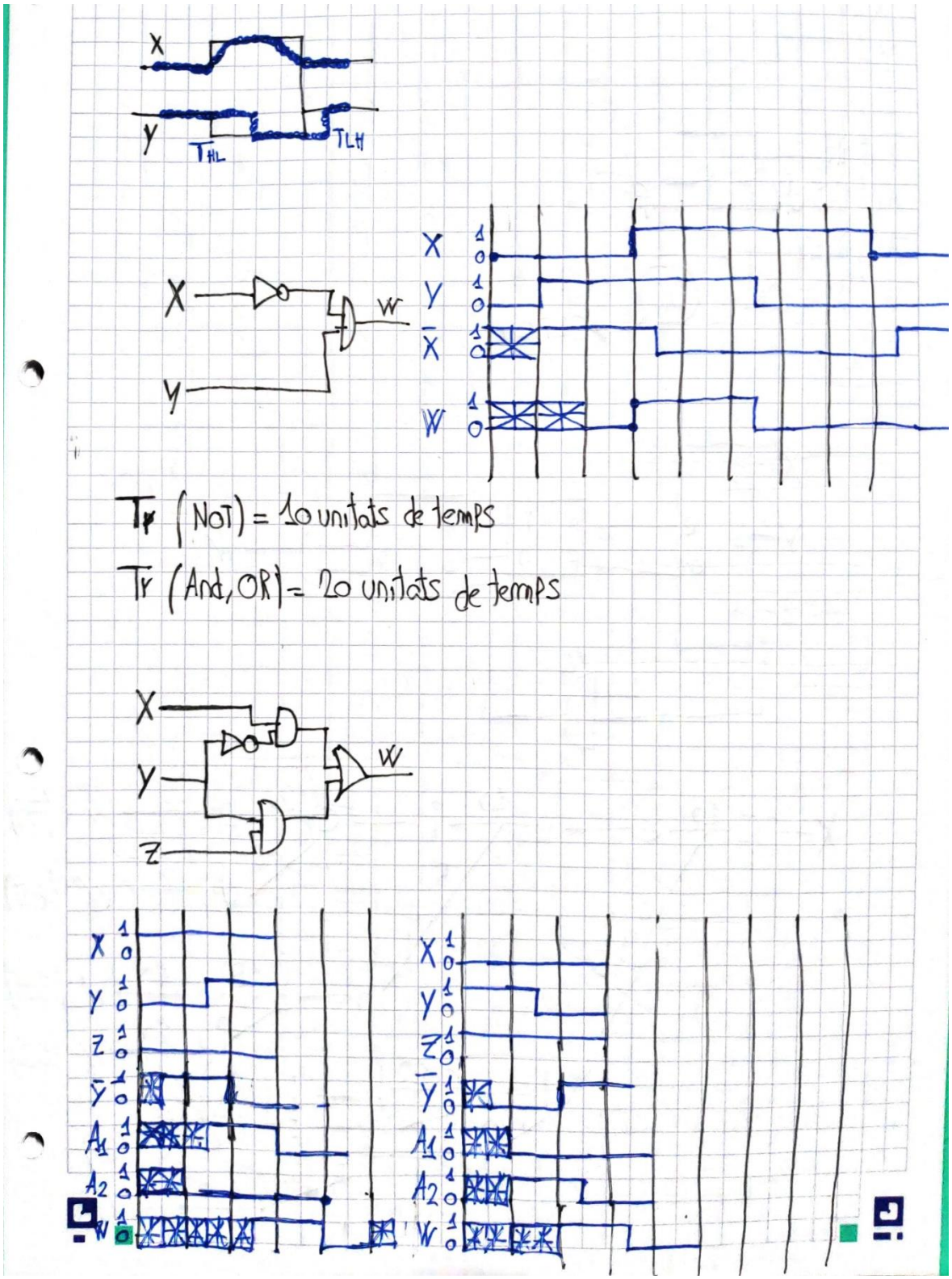


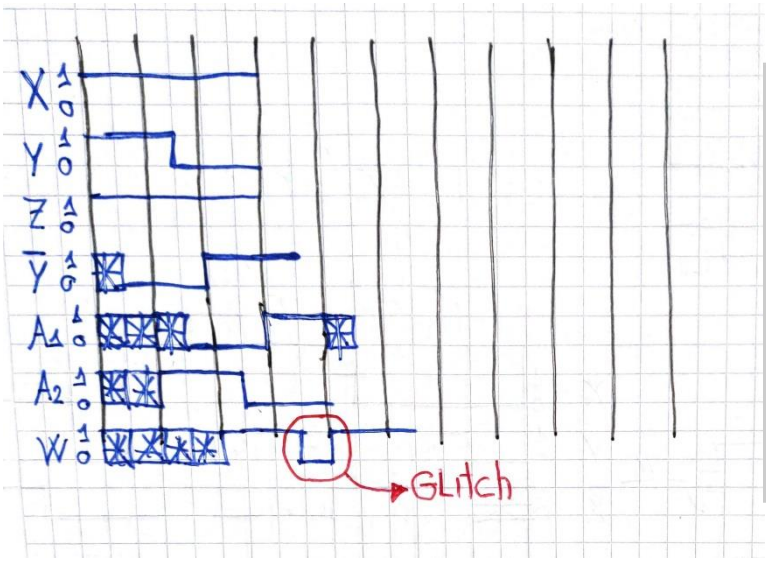
X	Y	Z	W	t
0	0	0	0	1
0	0	1	0	0
0	1	0	x	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	0











Un **camino** desde la entrada **e** a la salida **s** de un circuito combinacional es una trayectoria o recorrido válido[a] desde la entrada **e** a la salida **s**, pasando por dispositivos y las líneas que los conectan.

Puede haber varios caminos de una entrada **e** a una salida **s**. Para cada camino definimos el **tiempo de propagación del camino** como la suma de los tiempos de propagación de las puertas (o en general dispositivos combinacionales) que se encuentran en el camino.[b]

Definimos también el **camino crítico** de la entrada **e** a la salida **s** de un circuito combinacional como el camino de **e** a **s** con mayor tiempo de propagación (el camino más largo en tiempo).

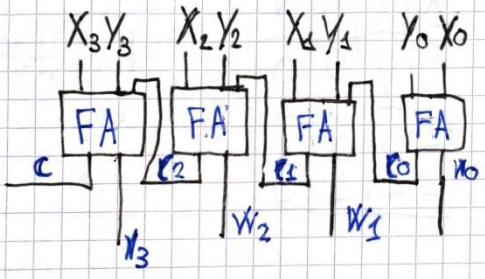
El **tiempo de propagación** desde una entrada **e** a una salida **s**, $\mathbf{Tp_{e\text{-}s}}$, de un circuito combinacional es el tiempo del camino crítico de **e** a **s**.

Tema 4 Aritmetics N

Suma

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 1001110 \\ +\ 1101011 \\ \hline 10111001 \end{array} \qquad \begin{array}{r} {\scriptstyle 1\ 1} \\ 1001 \\ +\ 1011 \\ \hline 10100 \end{array}$$

Full Adder



FA → Full Adder

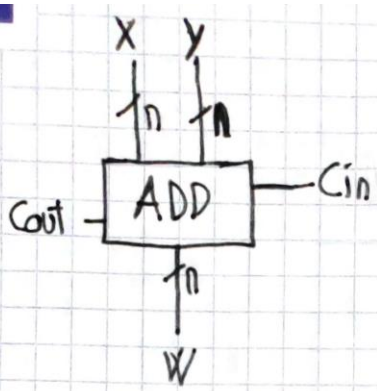
X	Y	C_{i-1}	C_i	W
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full-adder. El Full-adder, **Fa**, es un dispositivo combinacional con tres entradas (x, y, z) y dos salidas (c, s). El vector de salida W = (c, s) codifica en binario el número de entradas con valor igual a 1, esto es:

$$W_u = X_u + Y_u + Z_u = x + y + z$$

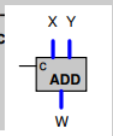
con $W_u = c \times 2^1 + s \times 2^0$ y siendo los operadores + y x la suma y el producto de la aritmética sobre números naturales. Ya que las tres entradas del Fa son conmutativas (a nivel aritmético y lógico) no las etiquetamos en el símbolo del Fa, como se muestra en la figura, pero sí que etiquetamos las salidas c y s, que no son intercambiables. Otra forma de definir el bloque Fa es mediante su tabla de verdad, que se ve en la figura.

x y z	c s
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

Símbolo y circuito interno del sumador con Full-adders en propagación del acarreo, ADD, con salida de resultado no representable, c (para n=16 bits). .



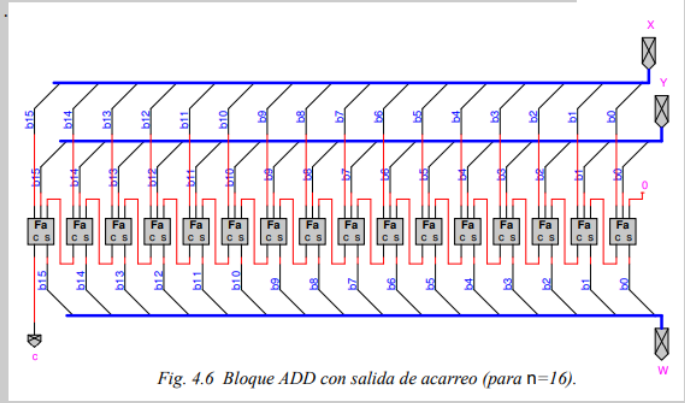
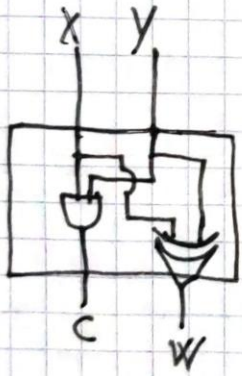


Fig. 4.6 Bloque ADD con salida de acarreo (para n=16).

X	Y	C	W
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

HA- Half Adder



HA

Half-adder: tabla de verdad, símbolo y circuito interno

x y	c s
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

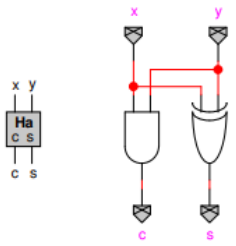
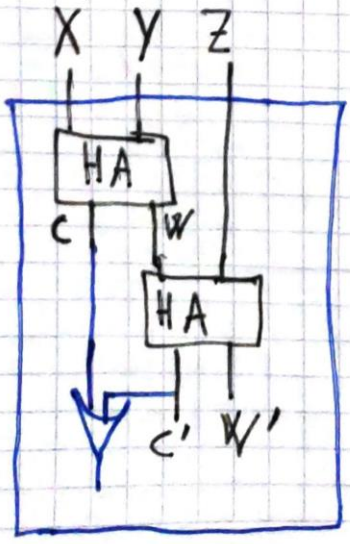


Fig. 4.7 Tabla de verdad, símbolo y circuito interno del Half-adder.



FA

Full-adder construido con dos Half-adder y una Or-2

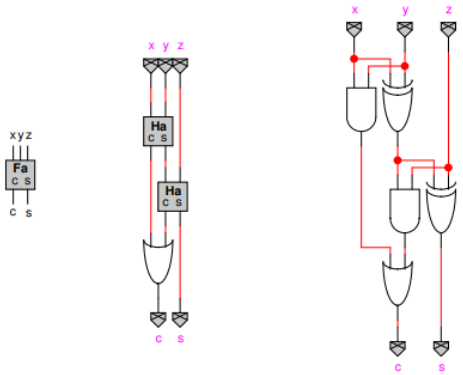
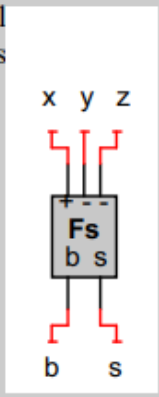


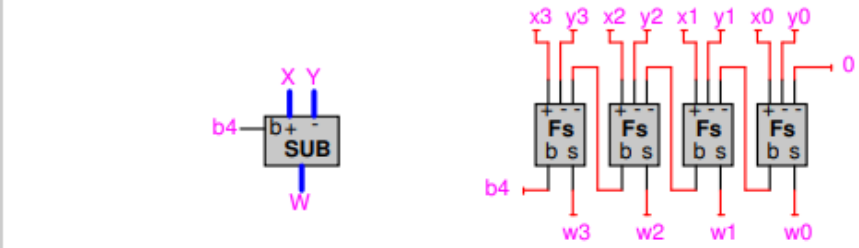
Fig. 4.9 Full-adder con dos Half-adders y una Or-2.

Full-subtractor. El Full-subtractor, **Fs**, es un dispositivo combinacional con tres entradas (x, y, z) y dos salidas (b, s) (cuya función la denotamos (b, s) = Fs(x, y, z)) que tiene la siguiente tabla de verdad.

x y z	b s
0 0 0	0 0
0 0 1	1 1
0 1 0	1 1
0 1 1	1 0
1 0 0	0 1
1 0 1	0 0
1 1 0	0 0
1 1 1	1 1



Símbolo y circuito del restador con Full-subtractors en propagación del débito.



Ejemplo 3

Aplicamos el algoritmo aritmético de resta en binario que acabamos de encontrar a los cuatro ejemplos que se muestran en la figura 4.3 para n = 8 bits (los valores del borrow no se muestran). Sólo en el caso d) los n bits del resultado no expresan el resultado correcto de la resta, ya que el acarreo (mejor débito) de salida, b_8, es 1.

a)

```
  1 1 1 0 1 0 0 0
- 0 1 1 0 1 0 0 1
-----------------
  0 1 1 1 1 1 1 1
```

b)

```
  1 1 1 0 1 0 0 1
- 0 0 1 1 1 0 0 0
-----------------
  1 0 1 1 0 0 0 1
```

c)

```
  0 1 1 0 1 1 0 0
- 0 0 0 0 1 1 1 1
-----------------
  0 1 0 1 1 1 0 1
```

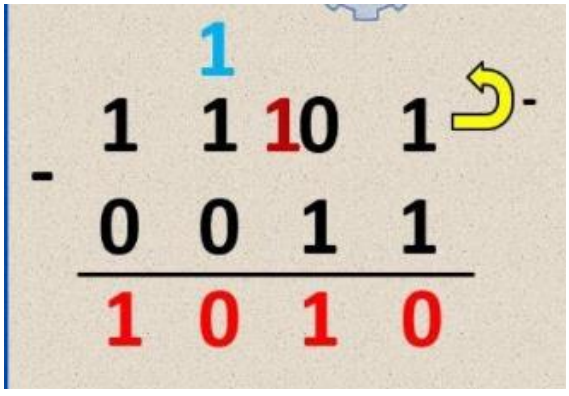
d)

```
  0 0 1 0 1 0 0 0
- 0 1 0 1 0 0 1 0
-----------------
  1 1 0 1 0 1 1 0
```

$b_8 = 1$

No representable

Fig. 4.10 Tablas sencillas resultado de aplicar el algoritmo aritmético de suma en binario para cuatro casos concretos.



Símbolo del Mx-8-1 y circuito interno usando Mx-2-1.

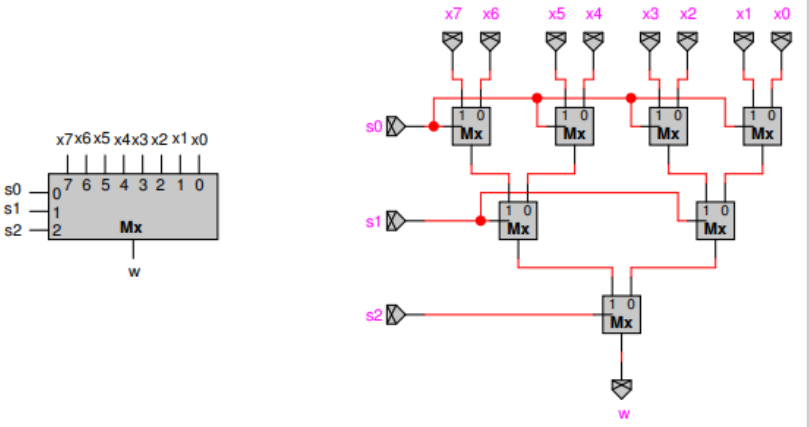
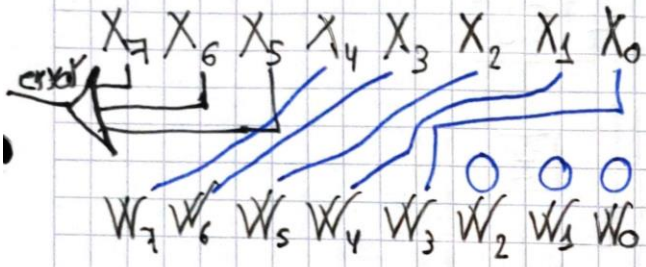


Fig. 4.24 Símbolo y esquema lógico interno del multiplexor Mx-8-1.

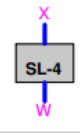
Multiplicació

2^N

$101 \times 2^3 \rightarrow 101000$



Símbolo y circuito interno del desplazador de k bits a la izquierda, SL-k (Shift Left- k), para k = 4. Implementa la operación $W_u = X_u 2^k$ siempre que los k bits de mayor peso de X sean 0. El símbolo no tiene salida de resultado no representable con n bits ya que en el computador que diseñaremos no la utilizaremos. No obstante, en el esquema interno si se ha implementado con una puerta Or.



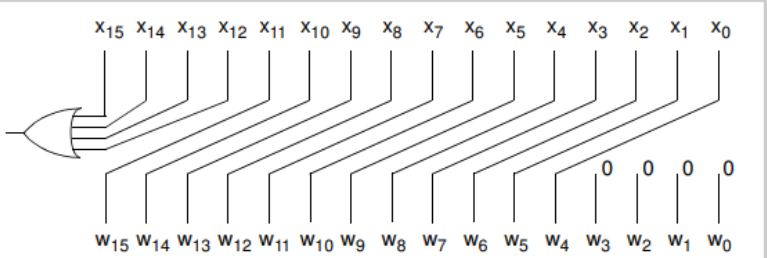
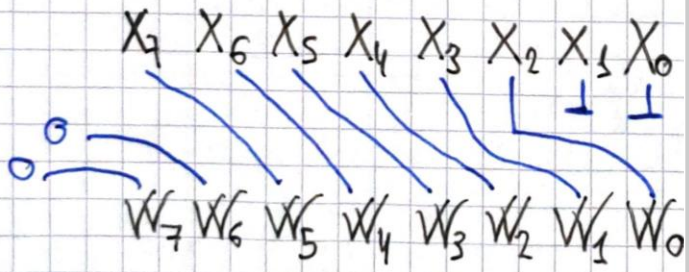


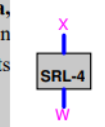
Fig. 4.12 Símbolo del desplazador de 4 bits a la izquierda (Shift Left - 4) y esquema interno para n=16. El símbolo no tiene la salida de resultado no representable.

Divisió

$X \div 2^N$



Símbolo y circuito interno del desplazador de k bits a la derecha, SRL-k (Shift Rigth Logically - k), para k = 4. Implementa la operación $W_u = X_u/2^k$. El resultado siempre es representable con los mismos bits que el operando.



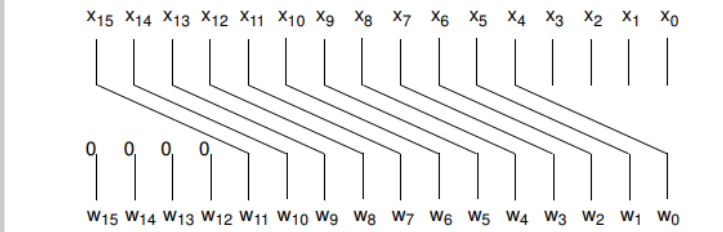
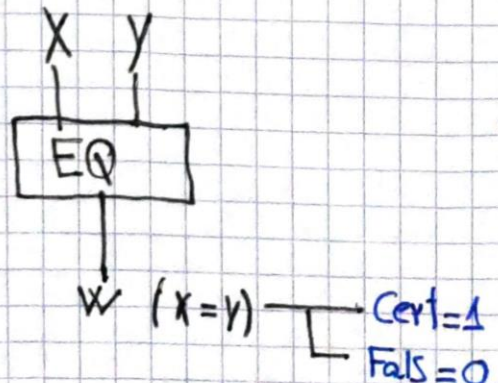
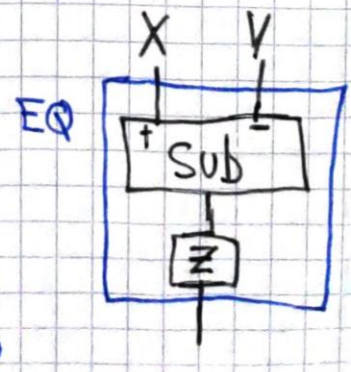


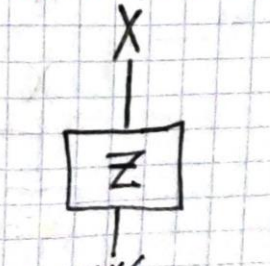
Fig. 4.14 Símbolo del desplazador lógico de 4 bits a la derecha (Shift Right Logicaly - 4) y esquema interno para n=16.

Comparador EQ





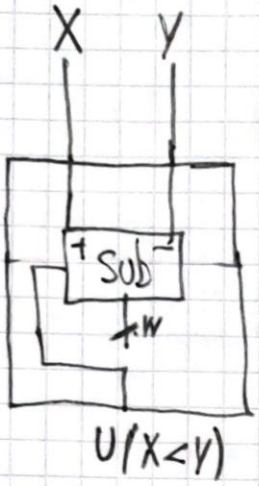
Component Z



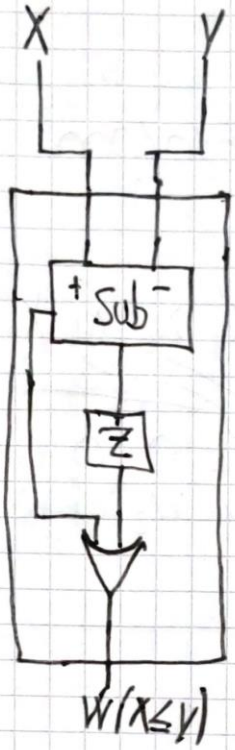
Si tots els bits de X=0
↳ W=1

Si hi ha algun X=1 → W=0

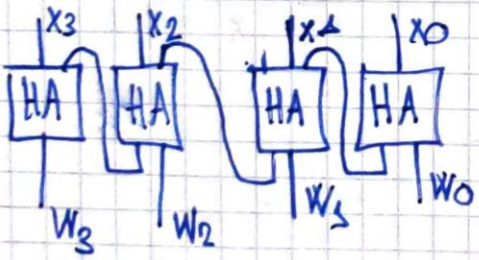
Less than (LT)



Less or Equal (LEU)

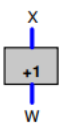


$INC \rightarrow W = X + 1$



$W = X \cdot 5$

Símbolo y circuito interno del incrementador, INC, con Half-adders.



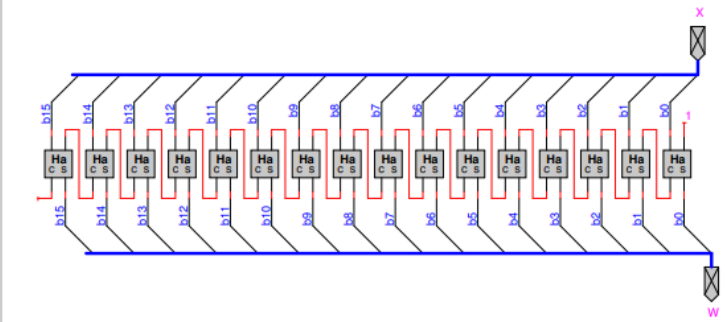


Fig. 4.27 Símbolo y esquema lógico interno del incrementador binario con Half-adders.

Tema 5 — Aritmetics d'enters

Complement a 2

	X	Y	Z	
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	-4
5	1	0	1	-3
6	1	1	0	-2
7	1	1	1	-1

$$V_s(X) = \begin{cases} \text{Si } X \geq 0 \rightarrow \sum_{i=0}^{n-2} X_i \cdot 2^i \\ \text{Si } X < 0 \rightarrow \sum_{i=0}^{n-2} X_i \cdot 2^i - 2^{n-1} \end{cases}$$

$$\boxed{V_s(X) = -2^{n-1} \cdot X_{n-1} + \sum_{i=0}^{n-2} X_i \cdot 2^i}$$

Ex 1011 $n=4$

$$V_s(X) = -2^3 \cdot 1 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$$= \boxed{-5}$$

Rang

$$\left[-2^{n-1},\ 2^{n-1}-1\right]$$

Las ventajas de usar dentro del computador la representación en Ca2 para los números enteros frente a la de signo y magnitud en base dos (más parecida al sistema que usamos los humanos) son las siguientes:

- El sumador en Ca2 es el mismo que el sumador para naturales representados en binario (excepto la detección de resultado no representable con n bits) por lo que no harán falta dos sumadores distintos en el computador: con uno solo podremos sumar tanto números naturales como enteros.
- El sumador binario es más sencillo (menos hardware y menor tiempo de propagación) que el sumador en signo y magnitud en base dos (aunque esto no lo hemos visto en detalle).

SUMADOR BINARIO PARA NATURALES Y ENTEROS CON DETECCIÓN DE RESULTADO NO REPRESENTABLE

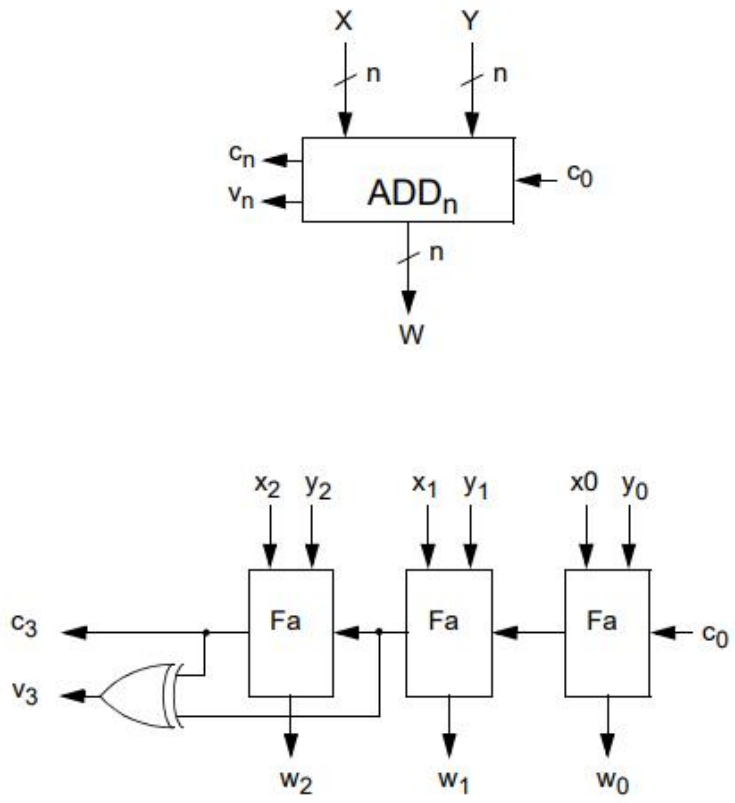


Fig. 5.10 Sumador de dos números naturales o enteros representados en binario o en Ca2 con 3 bits en Ca2 usando Full-adders con propagación del acarreo. Detección de resultado no representable con 3 bits para ambas representaciones.

5.5 Cambio de signo

Con el objetivo de diseñar un circuito negador, primero vamos a encontrar el algoritmo de la operación aritmética de cambio de signo (negación) de un número entero representado en Ca2 con n bits. Esto es, vamos a encontrar las operaciones lógicas a seguir para obtener el vector W de n bits a partir de los n bits del vector X, tales que:

$$W_s = -X_s$$

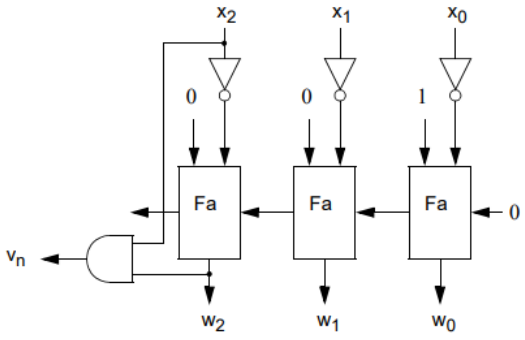


Fig. 5.12 Negador de 3 bits en Ca2 usando Full-Adders.

011 → a negatiu

1- Cambiar 0 per 1

2- Sumar 1

```
  100
    1
 -----
  101
```

Xor

Si X=0 → W=Y

Si X=1 → W=!Y

Restador

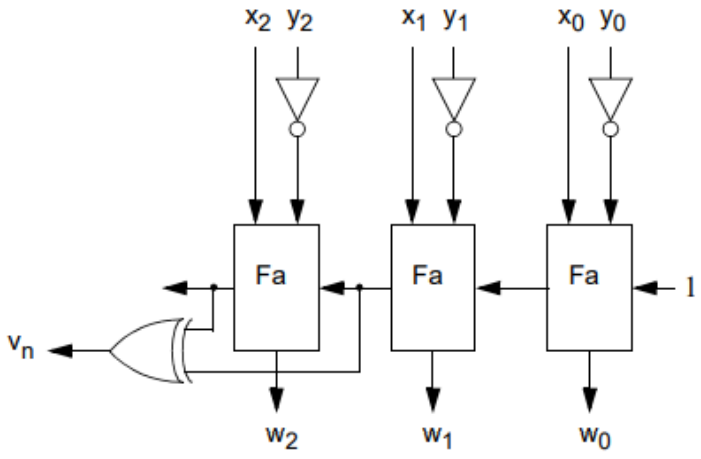
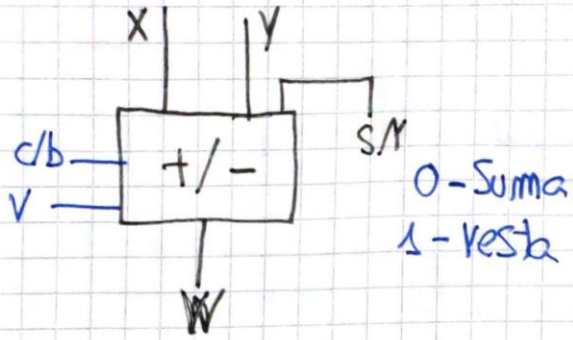


Fig. 5.16 Restador de enteros de 3 bits en Ca2 usando Full-adders con propagación del acarreo.

Sumador - Restador



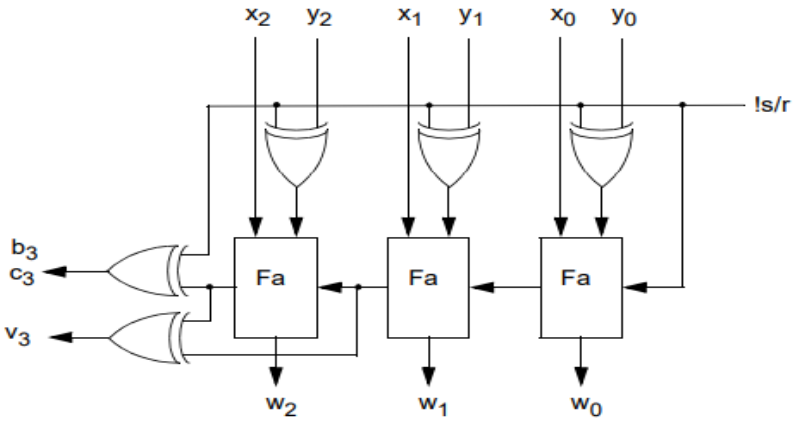


Fig. 5.17 Sumador/restador de 3 bits tanto para números naturales en binario como para enteros en Ca2 con detección de resultado no representable para las dos representaciones.

SL-n → S+ ≡ N

S- error si es perd un 0

SR-n →

Shift logic - N

SLR Right

SLL Left

SAR

SAL

Aritmetic Z

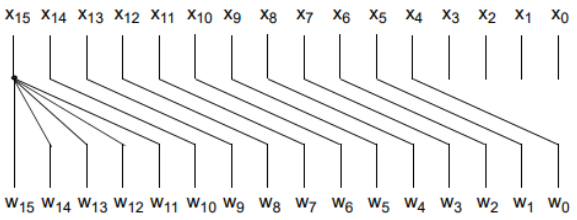
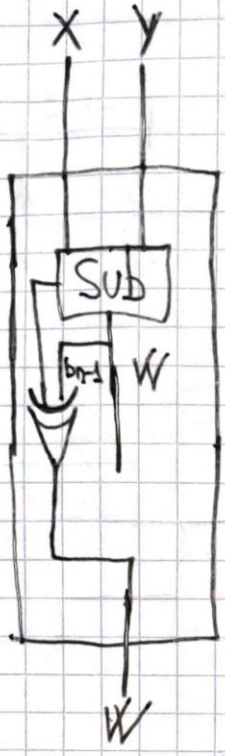
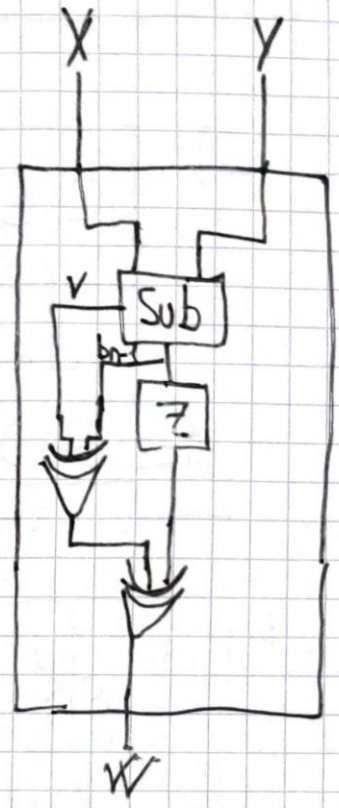


Fig. 5.22 Símbolo del desplazador aritmético de 4 bits a la derecha (Shift Right Arithmetically - 4) y esquema interno para n=16.

LT



LET

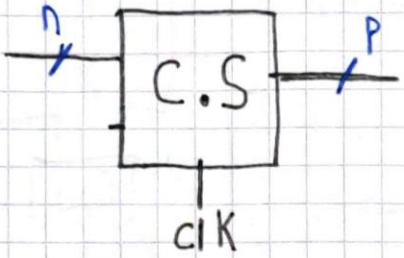


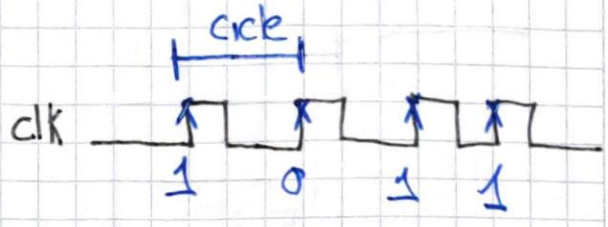
$$X_S = -X_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} X_i \cdot 2^i$$

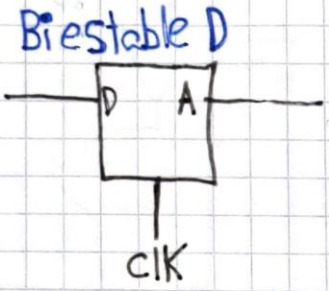
$$0 + 1 \cdot 2^2 + 1 \cdot 2^5 + 1 \cdot 2^6 = 100$$

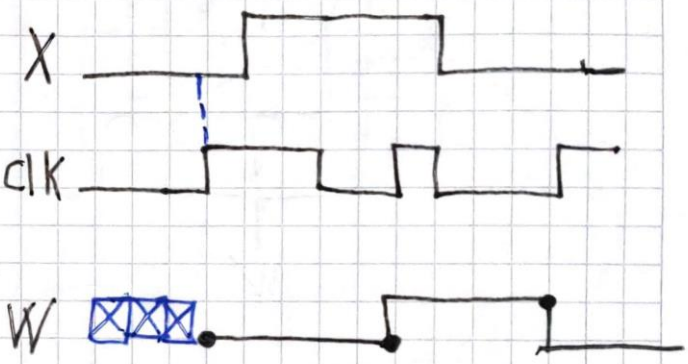
$$-1 \cdot 2^7 + 1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^3 + 2^4 = -103$$

Tema 6 — Circuit Sequencials









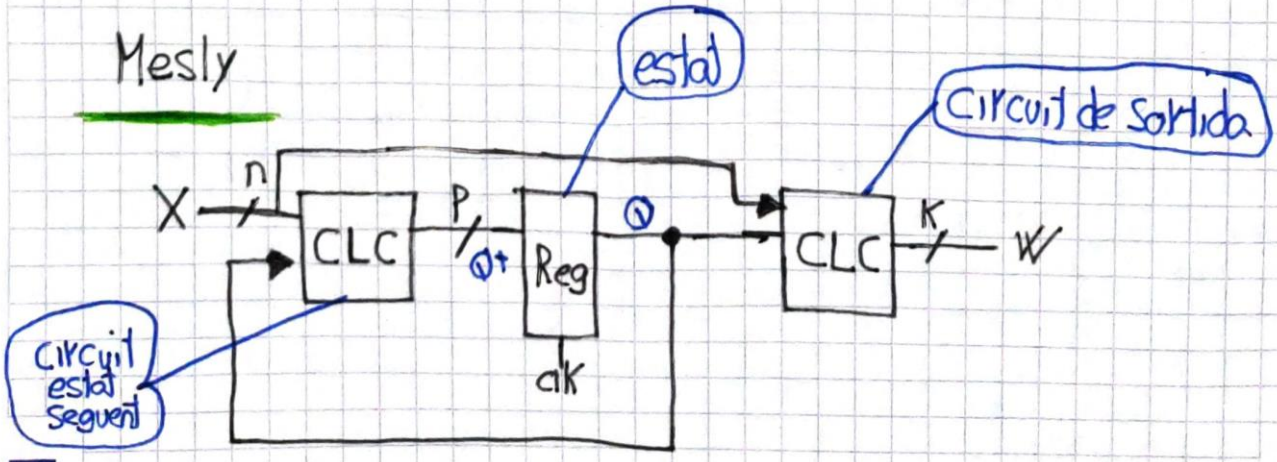
Registre = Conjunt de biestables

Normes

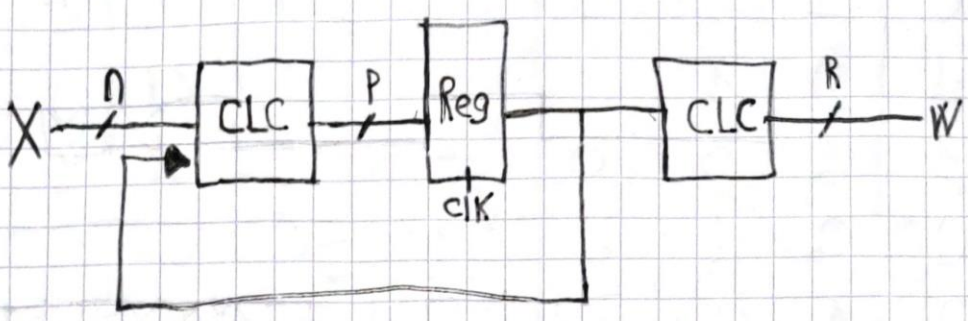
1- Tota entrada ha d'estar conectada

2- No podem conectar 2 sortides

Mesly



Moore



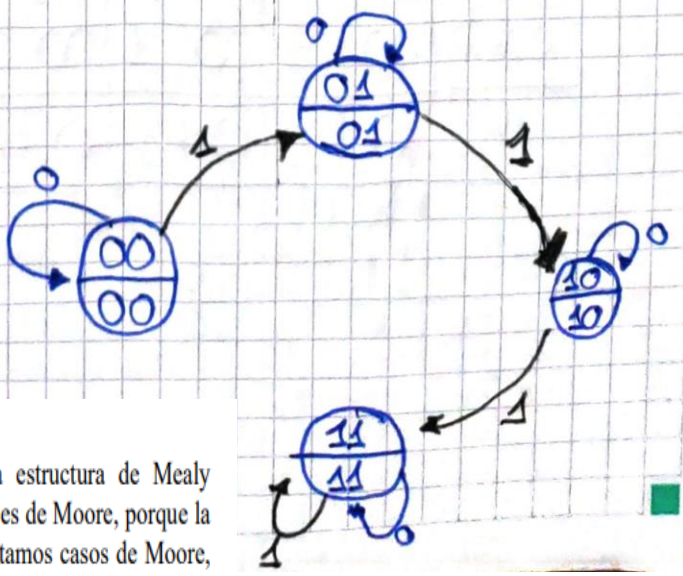
En un Circuit Sequencial → la taula de la veritat del Circuit d'estat Seguent i el Circuit de resultat

estat seguent

X	q_1	q_0	q_1^+	q_0^+
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Sortida

q_1	q_0	W_1	W_0
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

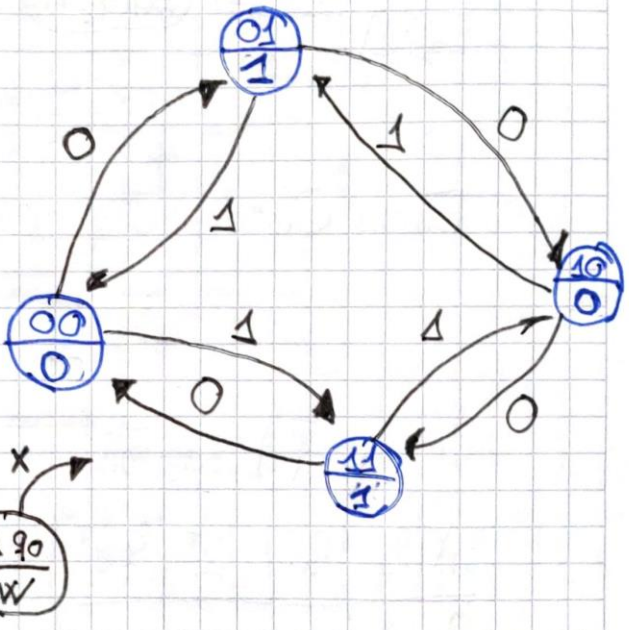


¿Circuito de Mealy o de Moore?

Una vez hecho esto debemos identificar si el esquema lógico sigue una estructura de Mealy (figura 6.13 o figura 6.14) o una estructura de Moore (figura 6.17). El ejemplo es de Moore, porque la salida w es solamente función del estado, pero no de la entrada. Aquí solo tratamos casos de Moore, pero es conveniente comprobarlo antes de continuar.

X	q_1	q_0	q_1^+	q_0^+
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

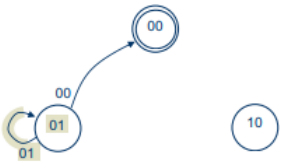
q_1	q_0	W
0	0	0
0	1	1
1	0	0
1	1	1



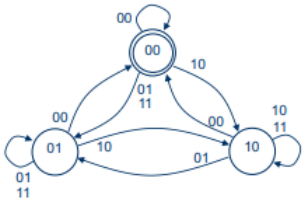
$Q^+ = \mathcal{G}(X, Q)$:

$q_1 q_0 x_1 x_0$	q_1^+	q_0^+
0 0 0 0	0	0
0 0 0 1	0	1
0 0 1 0	1	0
0 0 1 1	0	1
0 1 0 0	0	0
0 1 0 1	0	1
0 1 1 0	1	0
0 1 1 1	0	1
1 0 0 0	0	0
1 0 0 1	0	1
1 0 1 0	1	0
1 0 1 1	1	0
1 1 0 0	x	x
1 1 0 1	x	x
1 1 1 0	x	x
1 1 1 1	x	x

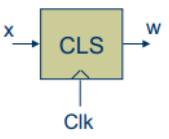
a)

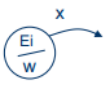


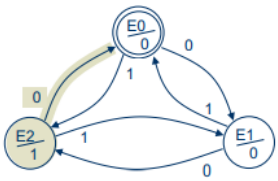
b)



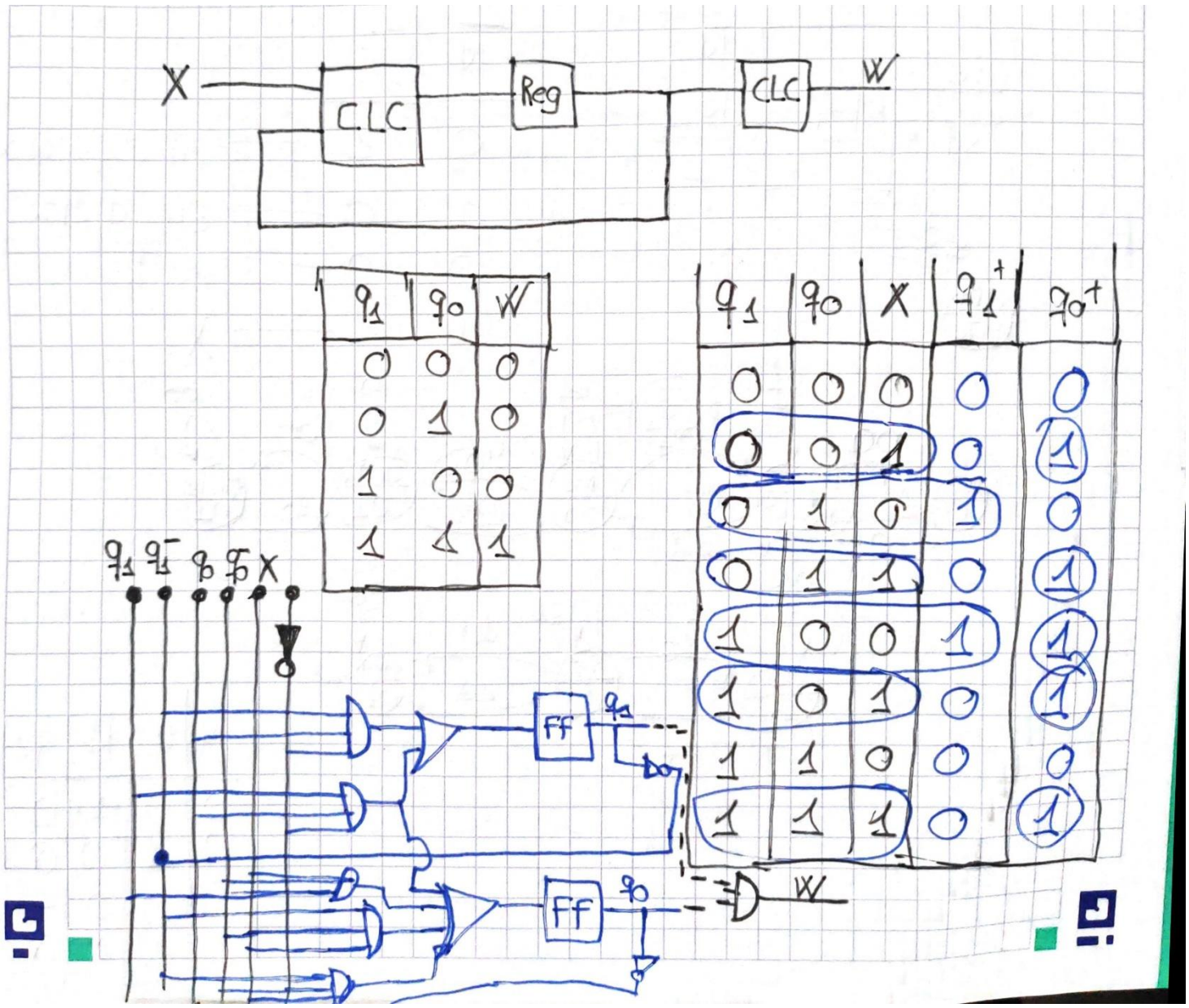
Cicle	0	1	2	3	4	5	6
X	0	1	1	0	0	0	1
estat	00	01	00	11	00	01	10
W	0	1	0	1	0	1	0







Núm. Ciclo	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Estado	E0	E2	E0	E1	E2	E0	E1	E0	E2	E1					
Entrada x	1	0	0	0	0	0	1	1	1	0	0	1	0	1	0
Salida w	0	1	0	0	1	0	0	0	1	0					



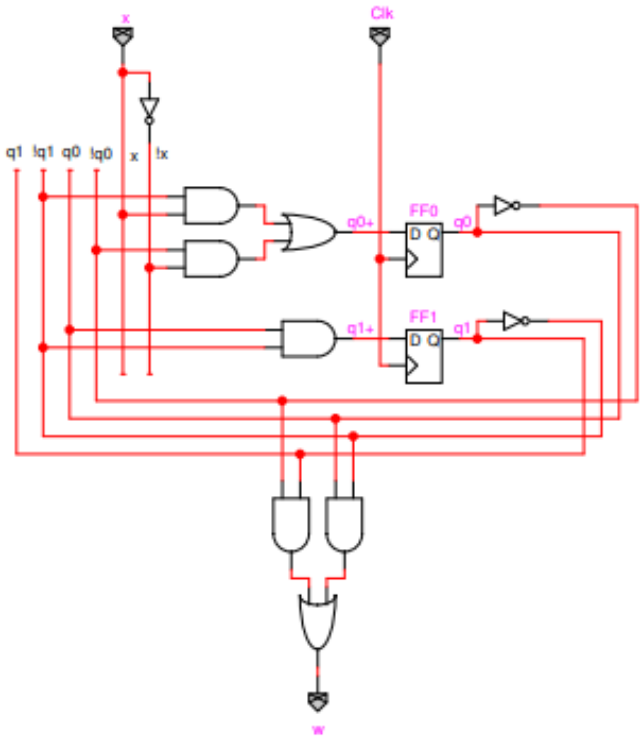
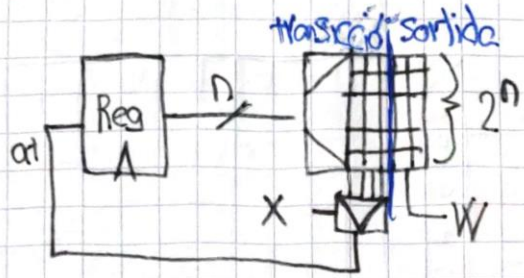
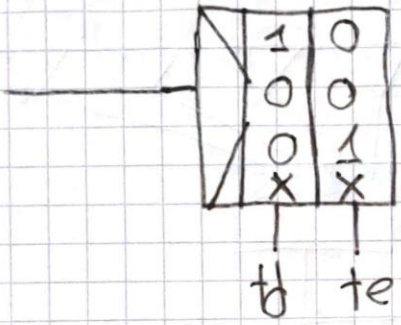
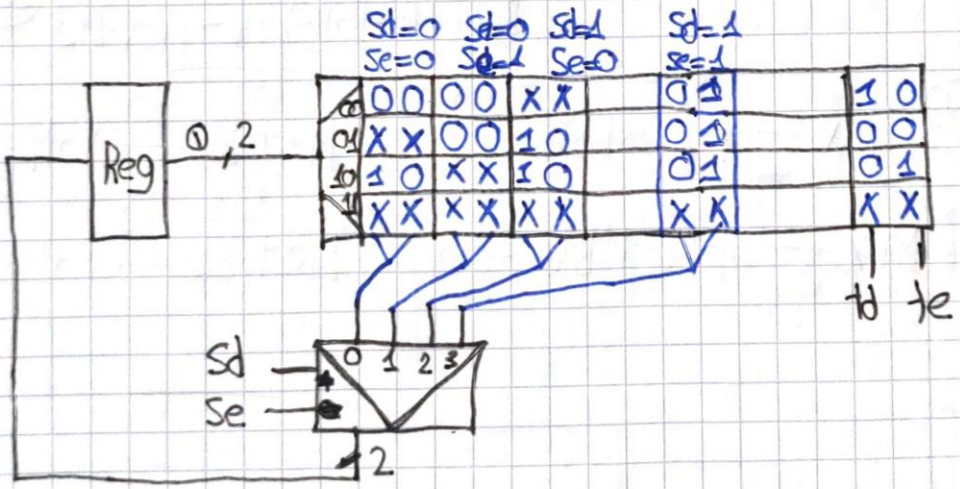


Fig. 6.28 Esquema lógico del circuito secuencial a analizar.

ROM + MUX







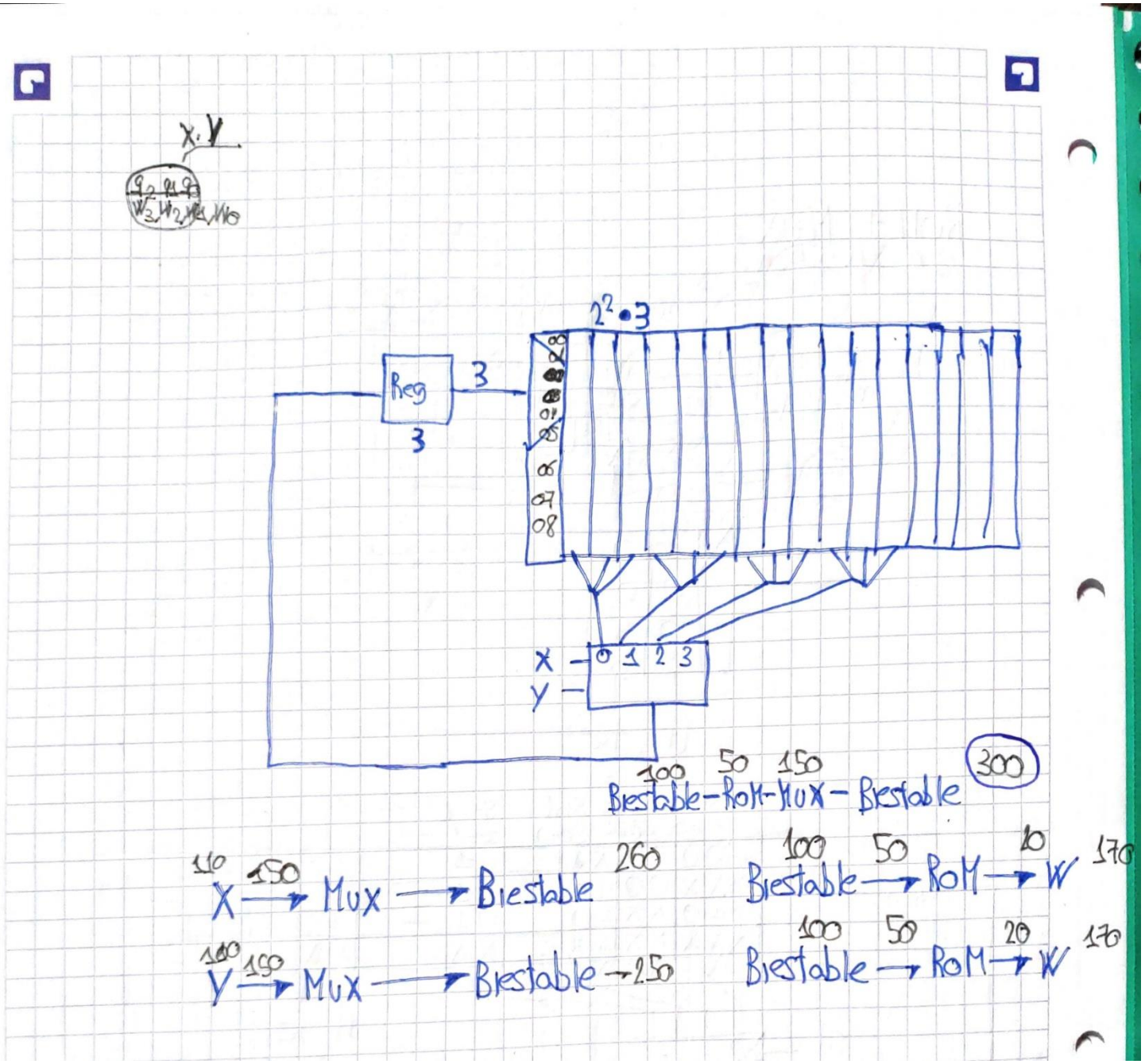
Camí Crític

e – b ^{NO}

b – b ^{NO}

b – s

100 u.t



Con una ROM y un multiplexor de buses. Es posible sintetizar a la vez el circuito del estado siguiente junto con el de las salidas usando una ROM y un multiplexor de buses. El ejemplo anterior se muestra en la figura 6.50. En general, si tenemos un circuito secuencial con n bits de entrada, m de salida y k de estado, la ROM tiene 2^k palabras de $k x 2^n + m$ bits por palabra y el multiplexor de buses tiene n bits de selección para elegir entre 2^n buses de k bits cada bus. Los bits de dirección de la ROM son los bits de estado del circuito y los bits de selección del multiplexor son los bits de entrada del circuito. Hay una palabra de la ROM para cada estado del circuito y cada una tiene la salida del circuito para ese estado y además el estado siguiente para cada uno de los posibles arcos que salen de ese estado (para cada combinación de valores de las entradas). Es un diseño sencillo, pero es fácil olvidarse alguna de las etiquetas de la ROM o del multiplexor o cometer un error al especificar la tabla que representa el contenido de la ROM, lo que haría que el circuito no quedara correctamente especificado.

PPE (Procesador de proposit específic)

De Control

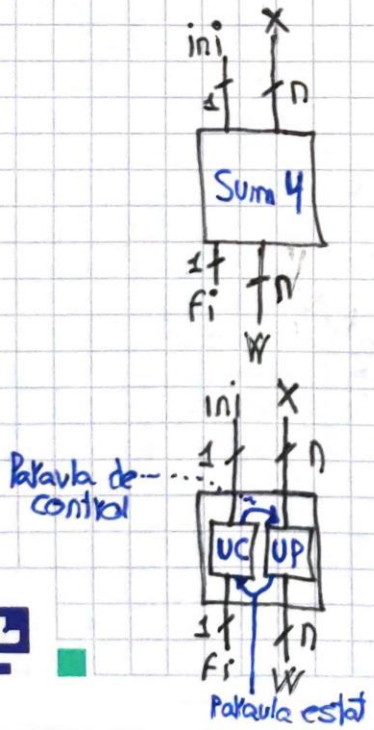
Entrades:

Sortides:

De dades

Entrades:

Sortides:

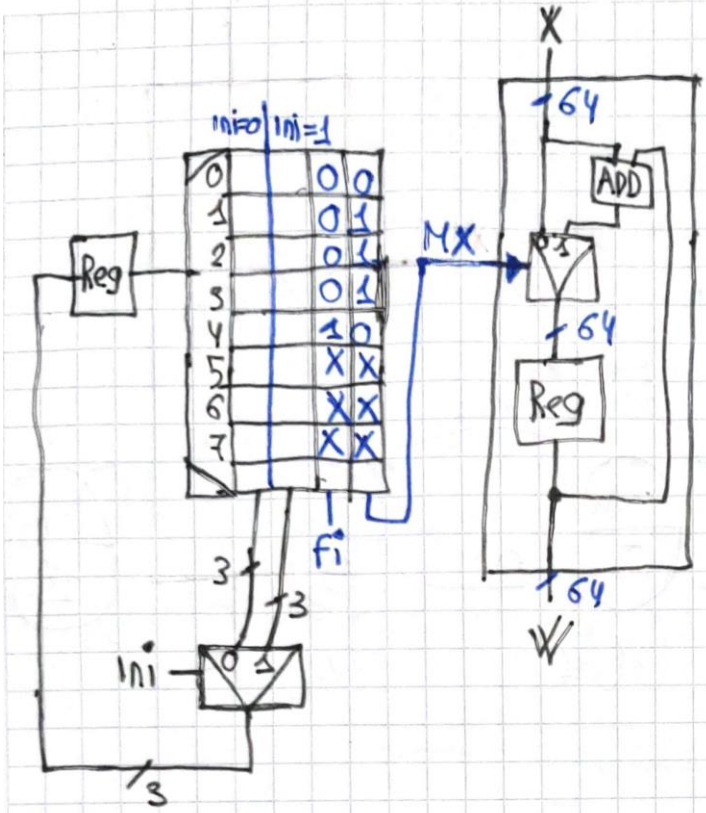


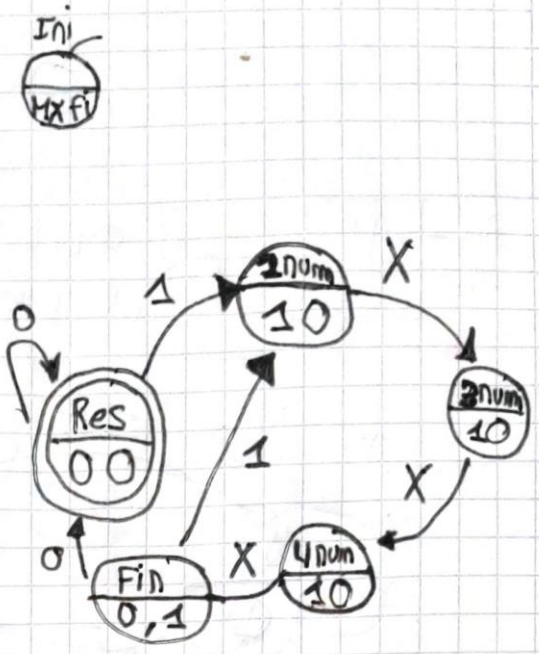
$ini(c) = 1$

$X(c) = 1r\ Valor\ de\ la\ Suma$

$fi(c+n) = 1$

$$W(c+n) = \sum_{i=0}^{3} X(c+i)$$





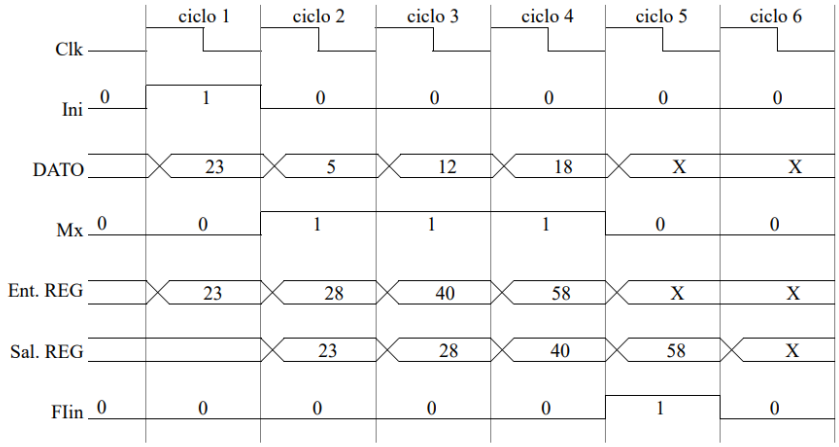


Figura 4. Cronograma del sistema para la suma de cuatro números.

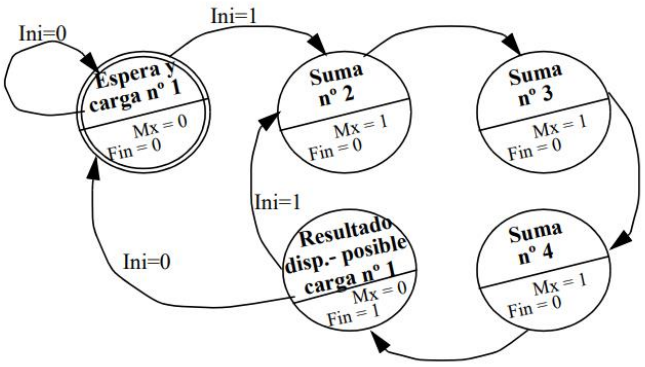
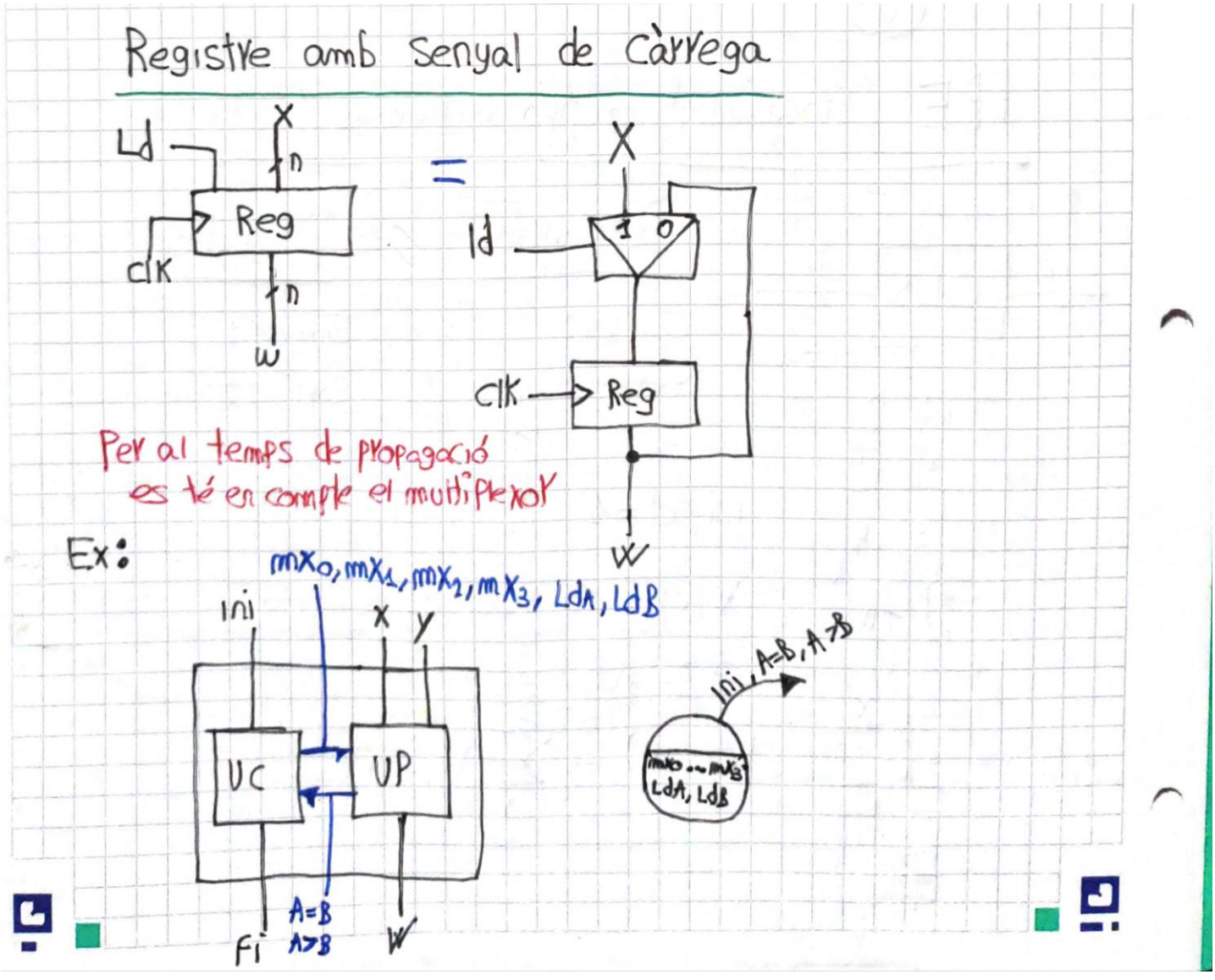
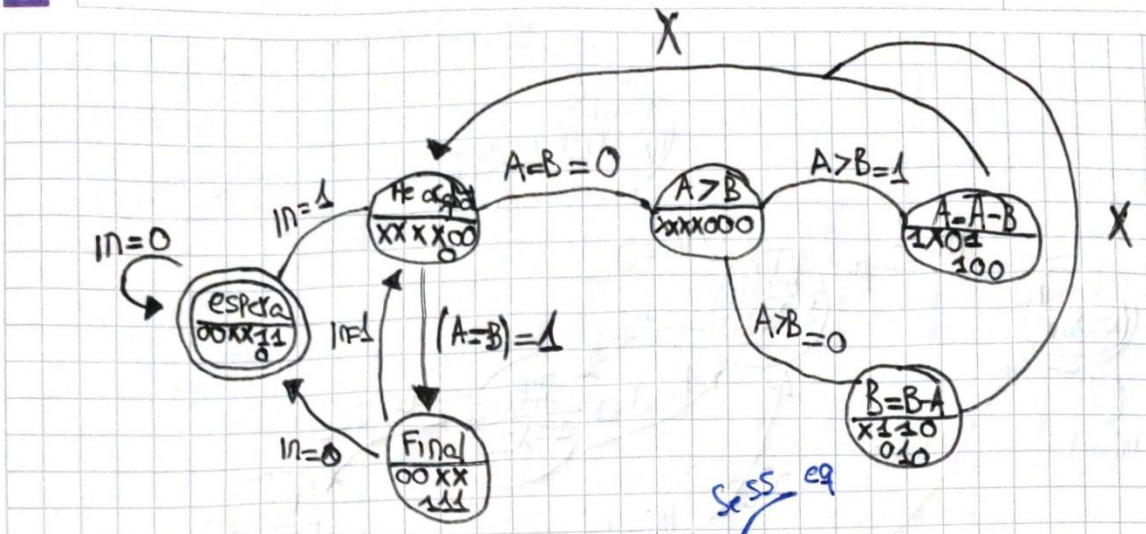
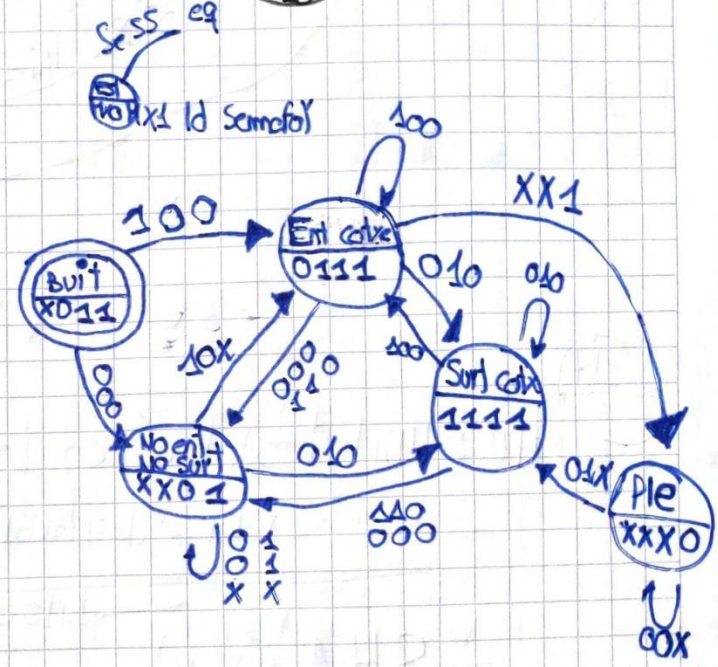
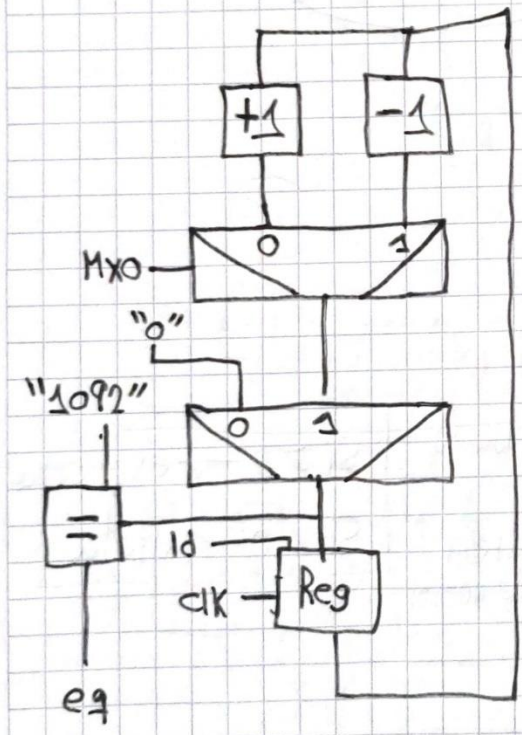


Figura 5. Diagrama de Moore para el sumador de cuatro números.









100 + 400 + 150 + 150 + 500 + 150

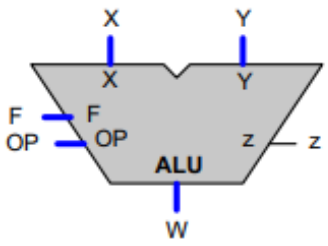
1450 u.t

16 Bit-wide ALU

ALU

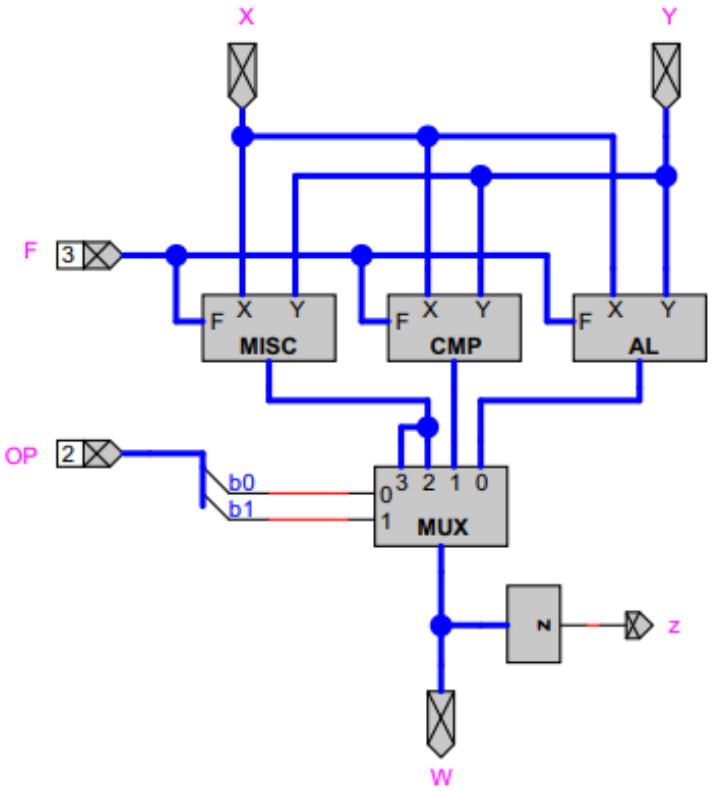
Description:

The 16-bit output W is the result of an arithmetic/logic operation (AL), a comparison (CMP) or an other action (MISC), applied to the 16-bit inputs X and Y. The type of operation is selected by the 2-bit input OP and by the 3-bit input F.



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

If the ALU performs a comparison the result can be true or false. If the result is true then W(b0) = 1 else W(b0) = 0. Moreover, W(bi) = 0 for i = 1 to 15.
The 1-bit output z indicates when the output W is a vector of 16 zeroes.



8.4 Unidad de Proceso General (UPG)

8.4.1 Estructura de la UPG

La figura 8.4 muestra el esquema lógico interno de la Unidad de Proceso General (UPG). Está formada por un banco de registros y una ALU, como los que acabamos de ver, interconectados formando un bucle, para que en cada ciclo de reloj se pueda operar el contenido de dos registros fuente y dejar el resultado, al final del ciclo, en un registro destino (que puede ser el mismo que uno de los registros fuente), como ya se mostró en el esquema simplificado de la figura 8.1.

Además, se ha dispuesto la mínima, pero necesaria, capacidad de comunicación con el exterior: un bus de entrada de datos desde el exterior al banco de registros (RD-IN, *read input*) y otro de salida del banco de registros, en concreto del bus A, al exterior (WR-OUT, *write output*). Hemos añadido un multiplexor

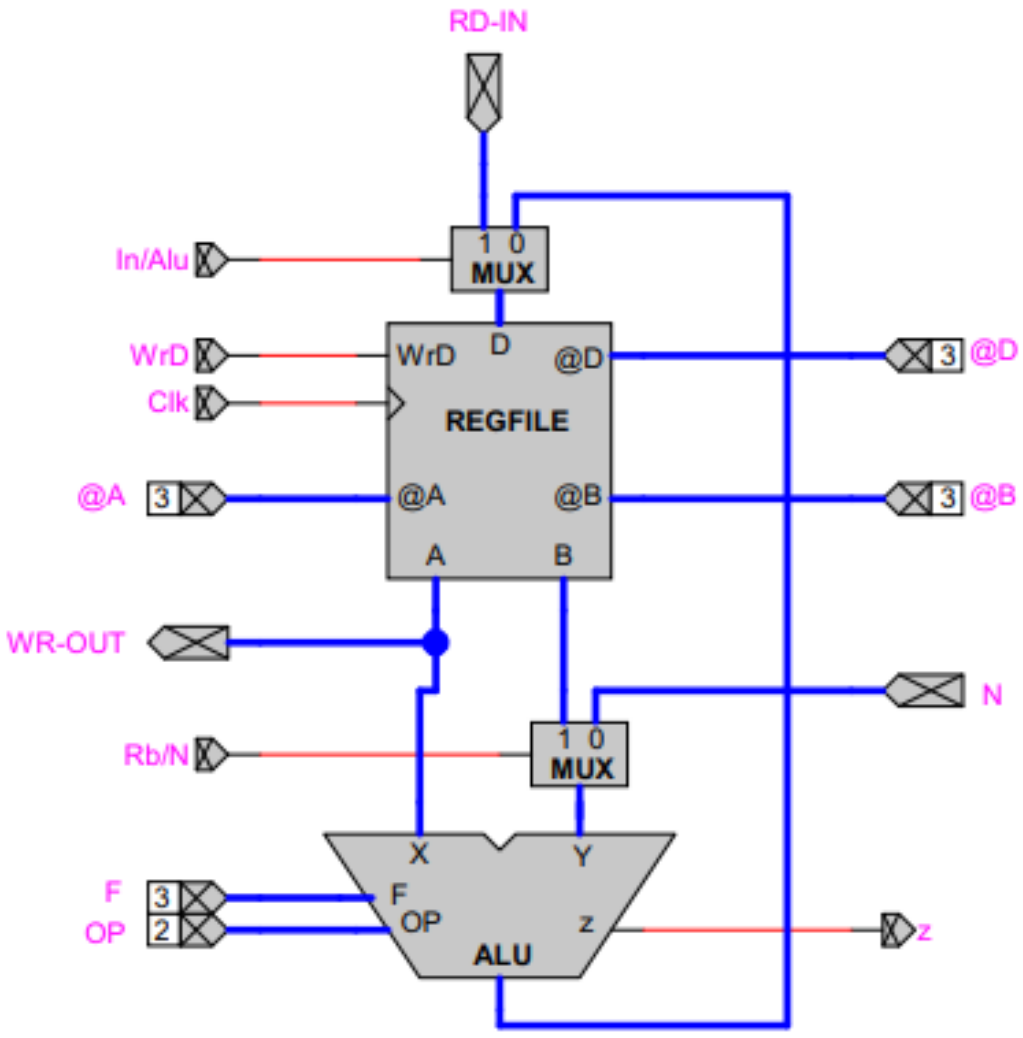
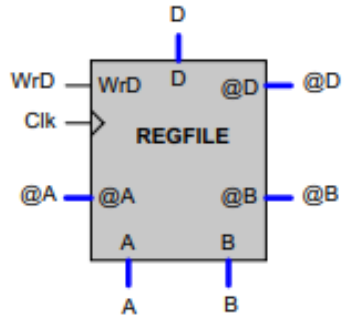


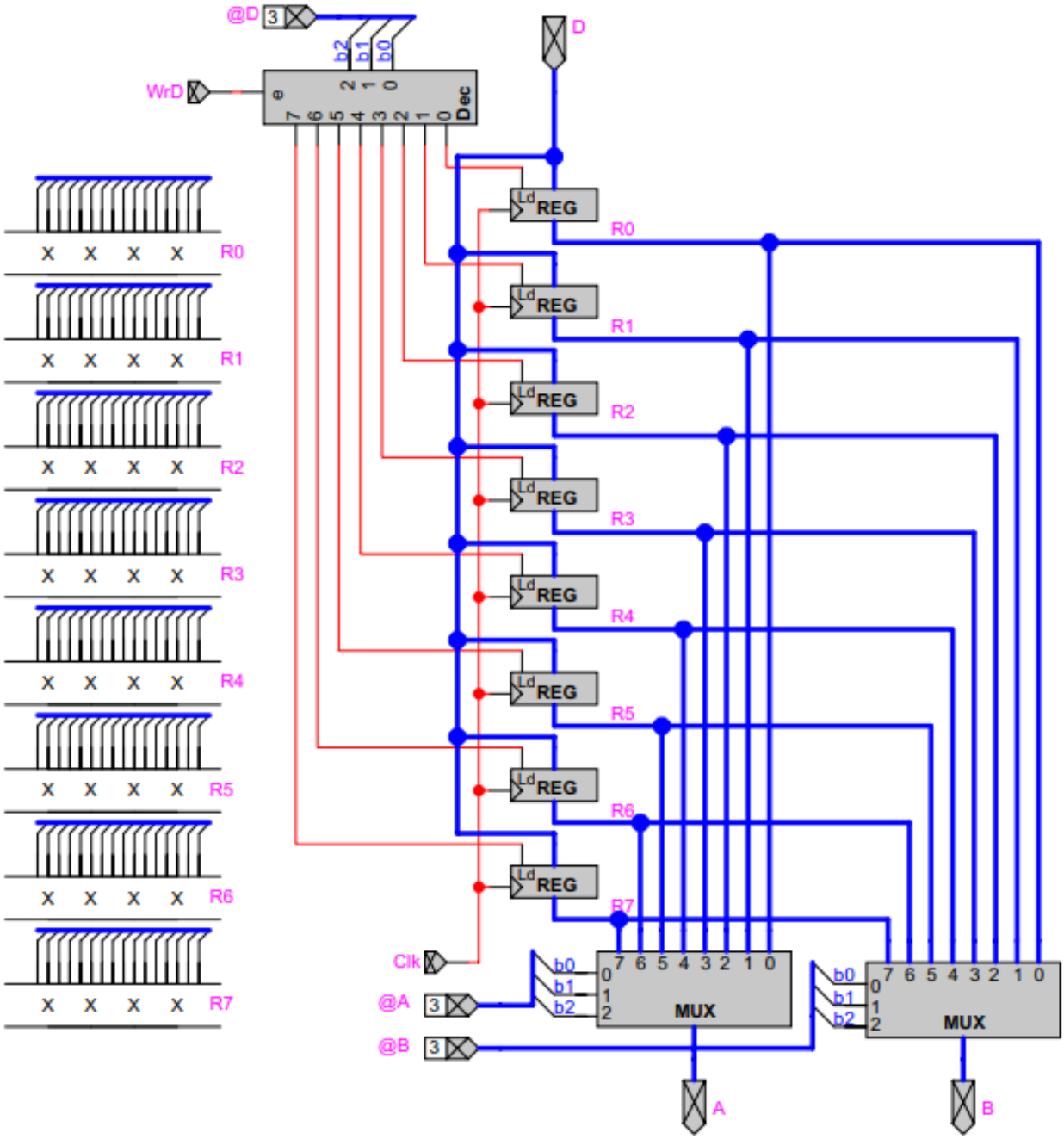
Fig. 8.6 Unidad de Proceso General, UPG

8-by-16 Bit-wide Register File with 1 Write and 2 Read Ports — REGFILE

Description:
Register file with 8 16 Bit-wide registers with 2 read ports, A and B, and 1 write port D. @A, @B and @D are the 3-bit addresses of ports A, B and D respectively. If WrD = 1 the port D is written into Register @D when the Clk binary input changes from 0 to 1.
A and B are the contents of Registers @A and @B.

For simulation purposes, a display of the contents of each register is available.





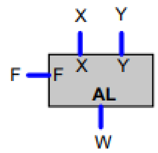
Mnemotécnicos	@A	@B	Rb/N	OP	F	In/Alu	@D	WrD	N (hexa)
	Palabra de Control de 33 bits								
`ADD R6, R3, R5`	0 1 1	1 0 1	1	0 0	1 0 0	0	1 1 0	1	X X X X
`CMPLEU R3, R1, R5`	0 0 1	1 0 1	1	0 1	1 0 1	0	0 1 1	1	X X X X
`ADDI R7, R1, -1`	0 0 1	x x x	0	0 0	1 0 0	0	1 1 1	1	F F F F
`ANDI R2, R3, 0XFF00`	0 1 1	x x x	0	0 0	0 0 0	0	0 1 0	1	F F 0 0
`NOT R4, R2`	0 1 0	x x x	x	0 0	0 1 1	0	1 0 0	1	X X X X
`MOVE R1, R5`	1 0 1	x x x	x	1 0	0 0 0	0	0 0 1	1	X X X X
`MOVEI R3, 0XFA02`	x x x	x x x	0	1 0	0 0 1	0	0 1 1	1	F A 0 2
`IN R2`	x x x	x x x	x	x x	x x x	1	0 1 0	1	X X X X
`OUT R4`	1 0 0	x x x	x	x x	x x x	x	x x x	0	X X X X
`ANDI -, R3, 0x8000`	0 1 1	x x x	0	0 0	0 0 0	x	x x x	0	8 0 0 0
`NOP`	x x x	x x x	x	x x	x x x	x	x x x	0	X X X X
`IN R2 // OUT R4`	1 0 0	x x x	x	x x	x x x	1	0 1 0	1	X X X X

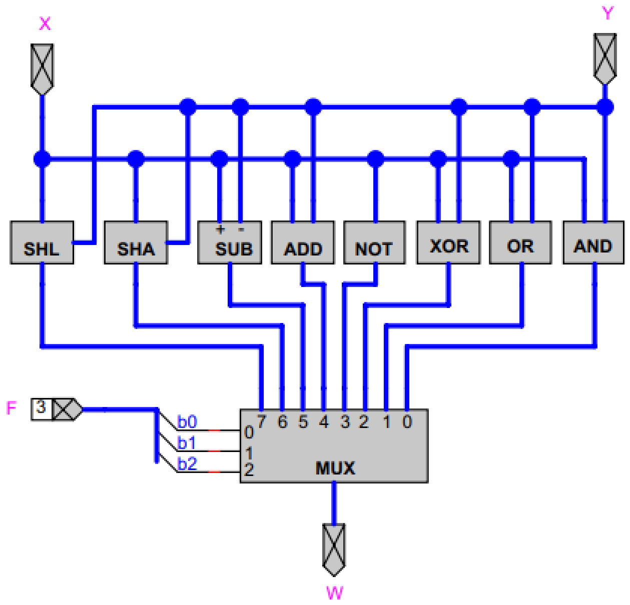
Fig. 8.9 Ejemplos de acciones que se pueden hacer en la UP en un ciclo y las palabras de control de 33 bits asociadas a cada acción

16 Bit-wide Arithmetic and Logic Operators **AL**

Description:
The 16-bit output W is the arithmetic or logic function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y.
The functions are:

F	W
000	AND (X, Y)
001	OR (X, Y)
010	XOR(X, Y)
011	NOT (X)
100	ADD (X, Y)
101	SUB (X, Y)
110	SHA (X, Y)
111	SHL (X, Y)

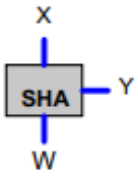




16 Bit-wide Arithmetic Shifter — SHA

Description:

The 16-bit output W is the result of shifting the 16-bit input X the number of bits coded by bits 0 to 4 of the 16-bit input Y. Y(b4:b0) represents a signed integer in two´s-complement.The number of shifting bits is the absolute value of the number represented in Y. If this value is positive, the shifting is to the left and if it is negative the shifting is to the right. When SHA shifts to the left the less significant bits of W are set to 0 and the most significant bits of X are lost. When SHA shifts to the right bit X(15) is sign-extended into the most significant bits of W and the less significant bits of X are lost.

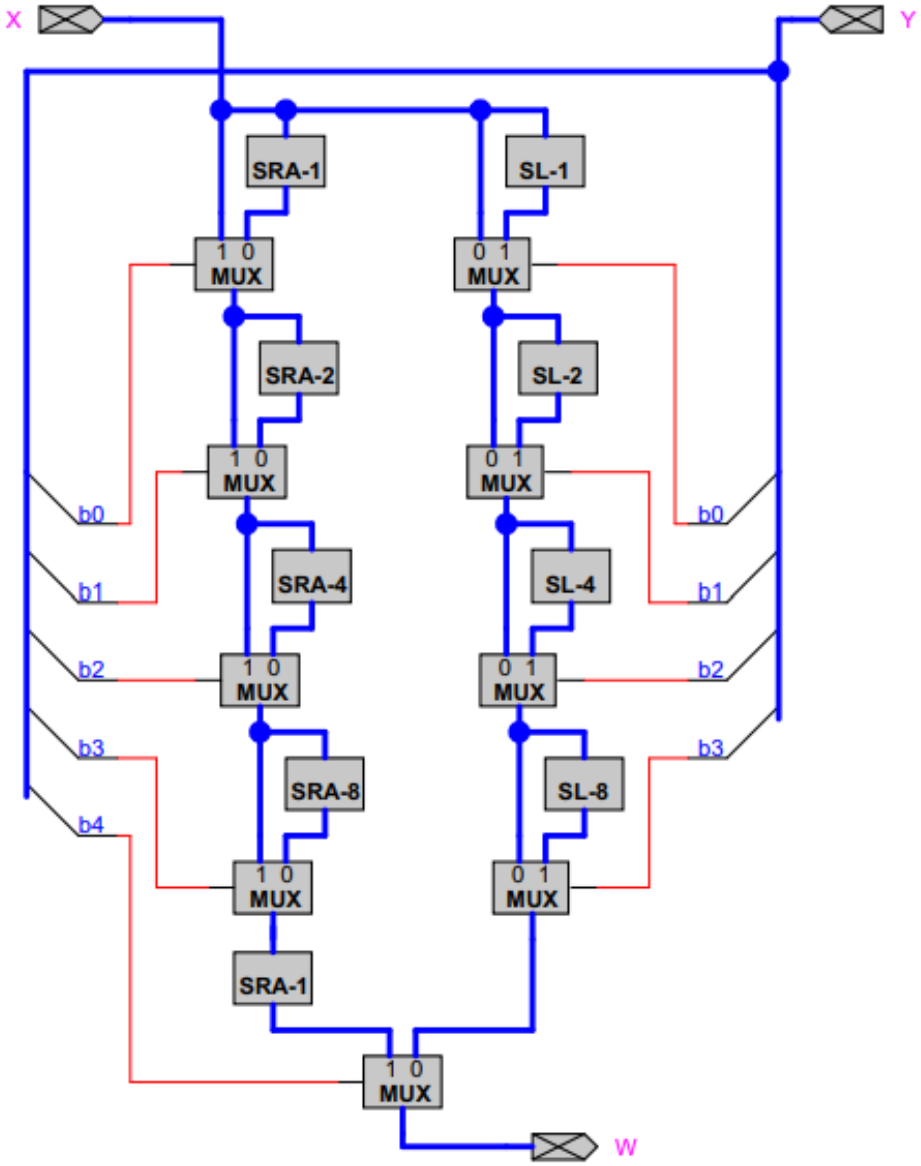


Arithmetic Operation:

Interpreting X, Y(b4:b0) and W as signed integers,
if ((-2**15) <= (Xs * (2**Y(b4:b0)s)) <= (2**15)-1) then Ws = Xs * (2**Y(b4:b0)s)
(The integer división of powers of 2 (when Y(b4:b0)s is negative) is done whith positive remainder)

Bit-level Logical Operation:

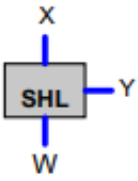
if (Y(b4:b0)s >= 0) then
W(bi) = 0 for i = 0 to Y(b4:b0)s -1
W(bi) = X(bi-Y(b4:b0)s) for i = Y(b4:b0)s to 15
if (Y(b4:b0)s < 0) then
W(bi) = X(bi-Y(b4:b0)s) for i = 0 to -Y(b4:b0)s - 1
W(bi) = X(b15) for i = -Y(b4:b0)s to 15



16 Bit-wide Logic Shifter — SHL

Description:

The 16-bit output W is the result of shifting the 16-bit input X the number of bits coded by bits 0 to 4 of the 16-bit input Y. Y(b0:b4) represents a signed integer in two´s-complement.The number of shifting bits is the absolute value of the number represented in Y. If this value is positive, the shifting is to the left and if it is negative the shifting is to the right. When SHL shifts to the left the less significant bits of W are set to 0 and the most significant bits of X are lost. When SHL shifts to the right the most significant bits of W are set to 0 and the less significant bits of X are lost.



Arithmetic Operation:

Interpreting X and W as unsigned integers and Y as signed integer

if ((Xu * (2**Y(b0:b4)s)) <= (2**16)-1)) then Wu = Xu * (2**Y(b0:b4)s)

Bit-level Logical Operation:

if (Y(b0:b4)s >= 0) then

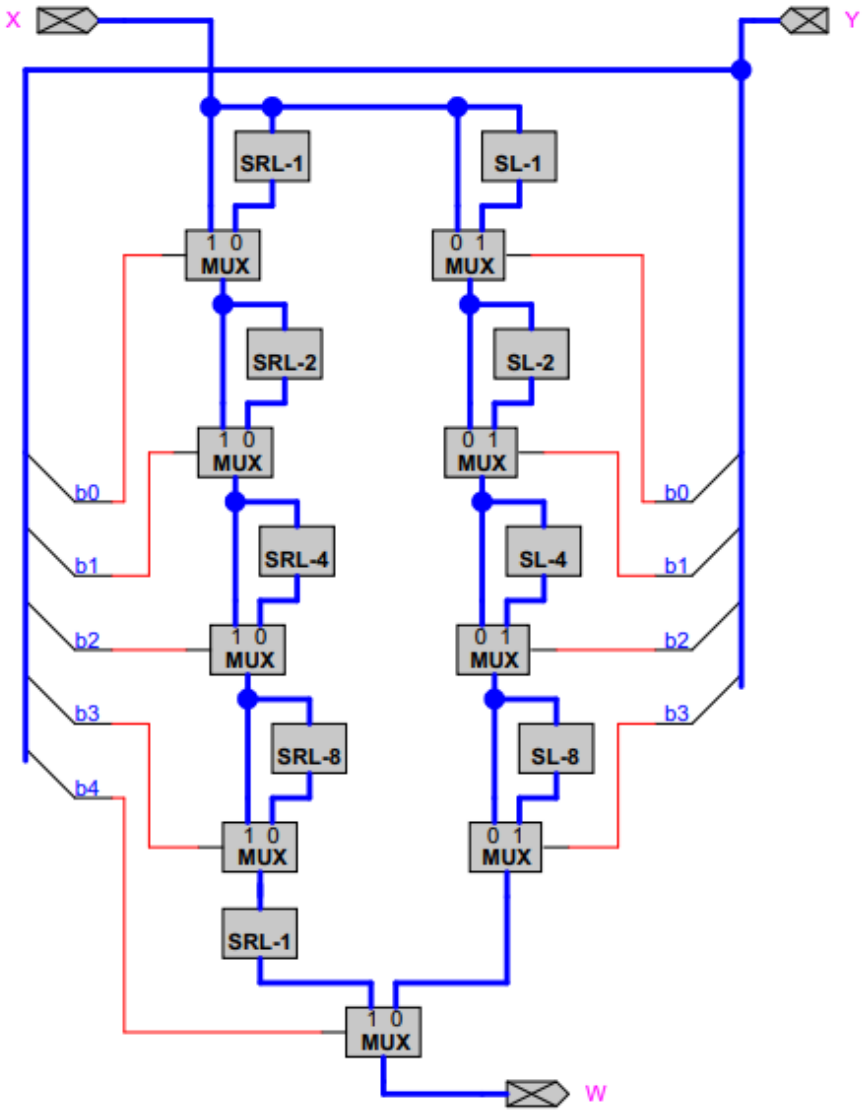
W(bi) = 0 for i = 0 to Y(b0:b4)s -1

W(bi) = X(bi-Y(b0:b4)s) for i = Y(b0:b4)s to 15

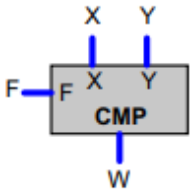
if (Y(b0:b4)s < 0) then

W(bi) = X(bi-Y(b0:b4)s) for i = 0 to -Y(b0:b4)s - 1

W(bi) = 0 for i = -Y(b0:b4)s to 15



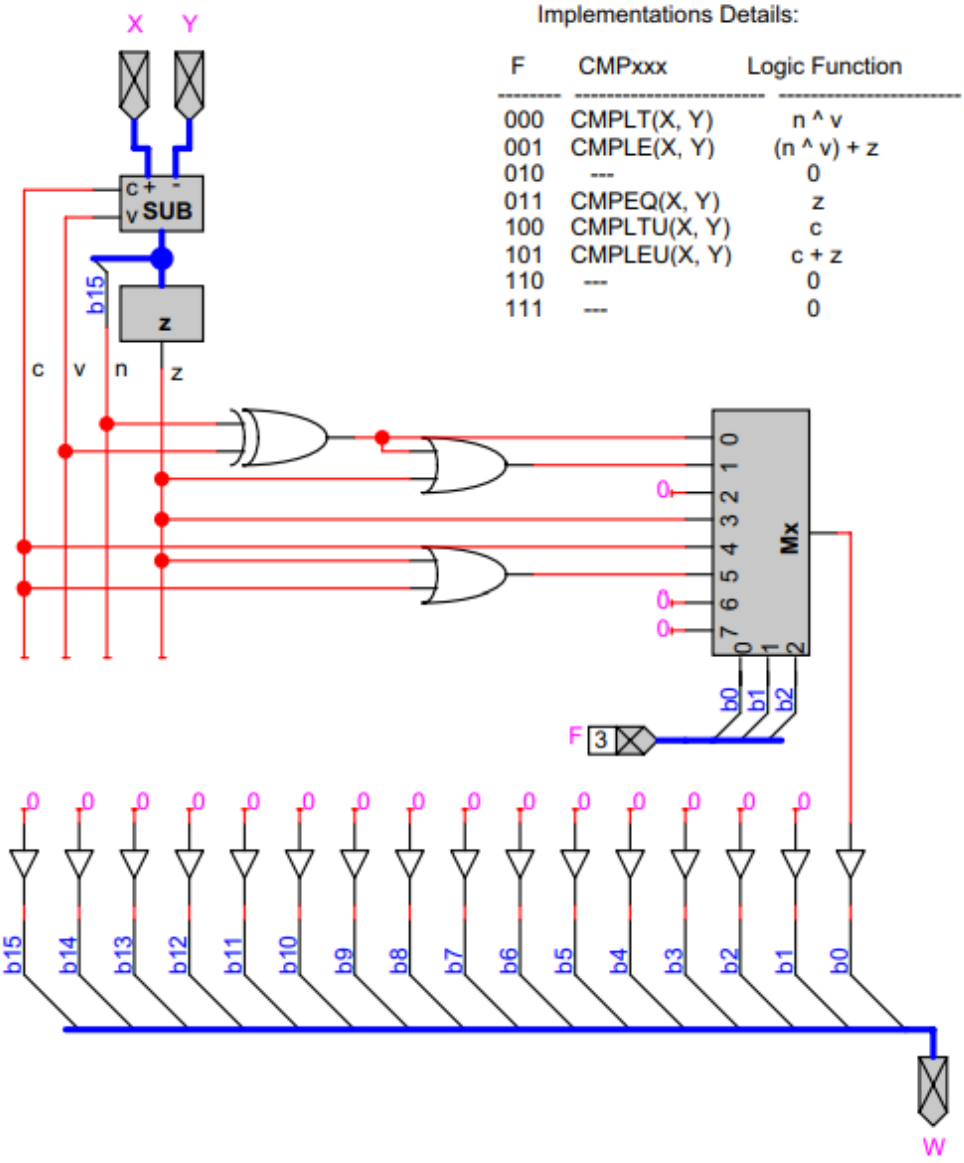
16 Bit-wide Signed and Unsigned Comparator — CMP



Description:

The 16-bit output W has only two posible values as the result of a comparison: true or false. True is coded as W(b0) = 1 and false as W(b0) = 0. For all the cases W(bi) = 0 for i = 1 to 15. The type of comparison and the consideration of the 16-bit inputs X and Y as signed or as unsigned integers is chosen by the 3-bit selection input F according to the next table:

F	W	CMPxx(X, Y)	Name
000	Xs < Ys	CMPLT(X, Y)	Less Than (Signed)
001	Xs <= Ys	CMPLE(X, Y)	Less than or Equal (Signed)
010	---	---	
011	X ==Y	CMPEQ(X, Y)	Equal
100	Xu < Yu	CMPLTU(X, Y)	Less Than Unsigned
101	Xu <= Yu	CMPLEU(X, Y)	Less than or Equal Unsigned
110	---	---	
111	---	---	



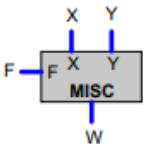
Implementations Details:

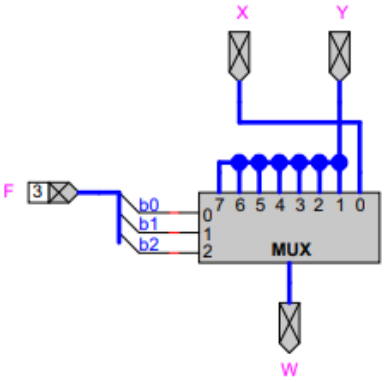
F	CMPxxx	Logic Function
000	CMPLT(X, Y)	n ^ v
001	CMPLE(X, Y)	(n ^ v) + z
010	---	0
011	CMPEQ(X, Y)	z
100	CMPLTU(X, Y)	c
101	CMPLEU(X, Y)	c + z
110	---	0
111	---	0

16 Bit-wide Miscellaneous Operators (Chapter 8) — MISC

Description:
The 16-bit output W is the function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y. The functions are:

F	W
000	X
001	Y
010	---
011	---
100	---
101	---
110	---
111	---

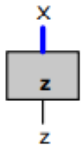


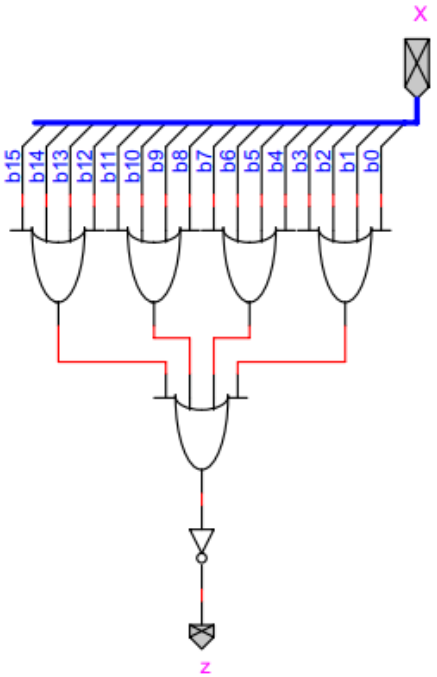


Zero Detector — Zero

Description:
The binary output z takes the value 1 if all the bits of the 16-bit input X are 0, otherwise z takes the value 0.

Bit-level Logical Operation:
if (X(bi) = 0 for i = 0 to 15) then z = 1 else z = 0





```
MCD (R3,R4)
  While (R3!=R4) {
     if  R3>R4 - R3=R3-R4
     else R4=R4-R3
  }
  Out R3
```