Introducción a los Computadores

Transparencias de la asignatura

Juan Segarra y Alejandro Valero

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Curso 2021–2022

Resumen de Licencia Creative Commons

Atribución-CompartirIgual 4.0 Internacional (CC BY-SA 4.0)

Este es un resumen legible por humanos (y no un sustituto) de la licencia.

Usted es libre de:

Compartir — copiar y redistribuir el material en cualquier medio o formato.

Adaptar — remezclar, transformar y construir a partir del material para cualquier propósito, incluso comercialmente.

El licenciante no puede revocar estas libertades en tanto usted siga los términos de la licencia

Bajo los siguientes términos:

Atribución — Usted debe dar crédito de manera adecuada, proporcionar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo de cualquier forma razonable, pero no de forma que sugiera que usted o su uso tienen el apoyo del licenciante.

CompartirIgual — Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

No hay restricciones adicionales — No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia.

Avisos:

No tiene que cumplir con la licencia para elementos del material en el dominio público o cuando su uso esté permitido por una excepción o limitación aplicable. No se dan garantías. La licencia podría no darle todos los permisos que necesita para el uso que tenga previsto. Por ejemplo, otros derechos como publicidad, privacidad, o derechos morales pueden limitar la forma en que utilice el material.

Introducción a los computadores

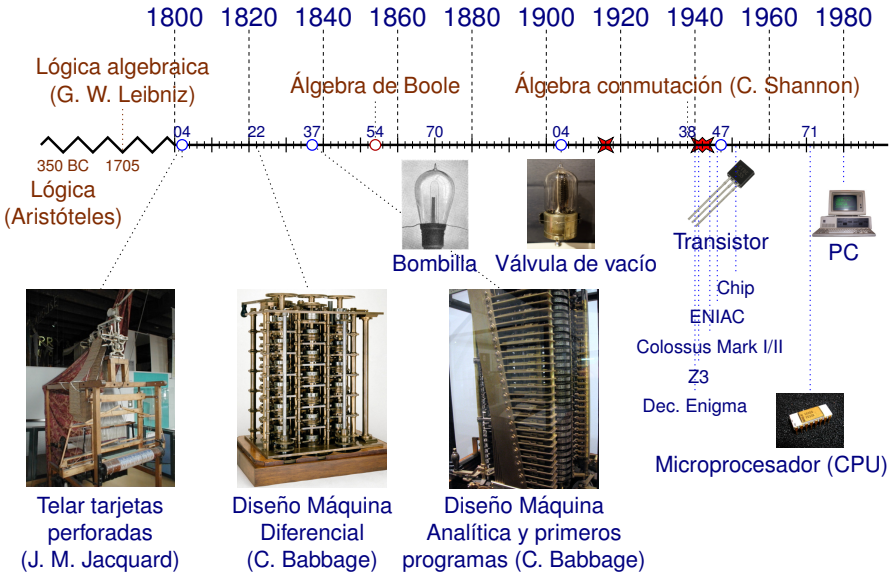
Tema 1 – Álgebra de Boole

Juan Segarra y Alejandro Valero

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción



1 Introducción (II)

Computador: dispositivo capaz de ejecutar secuencias de instrucciones aritmético-lógicas previamente programadas

Los computadores actuales operan mediante circuitos lógicos

- Base matemática: *Álgebra de Boole*
- Descripción de comportamientos por medio de *funciones lógicas* o *booleanas*
- Implementación de operaciones algebraicas mediante *puertas lógicas*

Teoría de autómatas: Posibilidad de actuar en base a la entrada actual (*sistema combinacional*), a la secuencia de entradas hasta el momento (*sistema secuencial*), o a la secuencia de entradas y un almacén de datos (*Máquina de Turing*)

2 Álgebra de Boole

Estructura algebraica definida por un conjunto de elementos $B = \{0, 1\}$ (falso, verdadero), dos operaciones binarias $\{+ \equiv \vee, \cdot \equiv \wedge\}$ y una operación unaria $\{^- \equiv \neg\}$, que satisface los siguientes axiomas/propiedades:

- Existencia de elementos
  - $\exists\, x, y \in B,\ x \neq y$
- Clausura
  - $\forall\, x, y \in B,\ x + y \in B$
  - $\forall\, x, y \in B,\ x \cdot y \in B$
- Propiedad asociativa
  - $\forall\, x, y, z \in B,$ $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
  - $\forall\, x, y, z \in B,$ $x + (y + z) = (x + y) + z$
- Propiedad conmutativa
  - $\forall\, x, y \in B,\ x + y = y + x$
  - $\forall\, x, y \in B,\ x \cdot y = y \cdot x$
- Propiedad distributiva
  - $\forall\, x, y, z \in B,$ $x \cdot (y + z) = x \cdot y + x \cdot z$
  - $\forall\, x, y, z \in B,$ $x + (y \cdot z) = (x + y) \cdot (x + z)$
- Involución
  - $\forall\, x \in B,\ \overline{(\overline{x})} = x$

2 Álgebra de Boole (II)

- Elementos de identidad
  - $\forall\, x \in B,\ x + 0 = x$
  - $\forall\, x \in B,\ x \cdot 1 = x$
- Complemento
  - $\forall\, x \in B,\ x + \overline{x} = 1$
  - $\forall\, x \in B,\ x \cdot \overline{x} = 0$
- La ley de De Morgan
  - $\forall\, x, y, z, \ldots, w \in B,\ \overline{x + y + z + \ldots + w} = \overline{x} \cdot \overline{y} \cdot \overline{z} \cdot \ldots \cdot \overline{w}$
  - $\forall\, x, y, z, \ldots, w \in B,\ \overline{x \cdot y \cdot z \cdot \ldots \cdot w} = \overline{x} + \overline{y} + \overline{z} + \ldots + \overline{w}$
- Idempotencia
  - $\forall\, x \in B,\ x + x = x$
  - $\forall\, x \in B,\ x \cdot x = x$
- Elementos dominantes
  - $\forall\, x \in B,\ x + 1 = 1$
  - $\forall\, x \in B,\ x \cdot 0 = 0$

Ley De Morgan

$$x + y = \overline{\overline{x} \cdot \overline{y}}$$
$$\overline{\overline{x} + \overline{y}} = x \cdot y$$

3 Funciones lógicas

- Una *función lógica* o booleana es una aplicación

$$f(e_1, e_2, \ldots, e_n) \to s$$

siendo $e_1, e_2, \ldots, e_n \in \{0, 1\}$ las variables lógicas de entrada y $s \in \{0, 1\}$ la salida de la función. E.g. $f(a, b) = a + (\overline{a} \cdot \overline{b})$

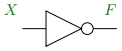
- Su *tabla de verdad* muestra todas las posibles combinaciones de entradas y su salida correspondiente

a	b	f
0	0	1
0	1	0
1	0	1
1	1	1

4 Puertas lógicas

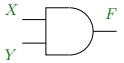
ANSI/IEEE Standard 91-1984

NOT



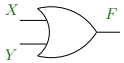
X	F
0	1
1	0

AND



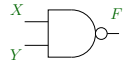
X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1

OR



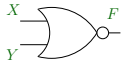
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

NAND



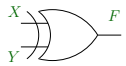
X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR



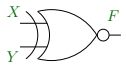
X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0

XOR ($\oplus$)



X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

XNOR ($\overline{X \oplus Y}$)



X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

4.1 Conjunto funcionalmente completo

- ¿Podemos construir unas puertas con otras?
- Un conjunto de puertas es funcionalmente completo si cualquier función lógica puede realizarse con un circuito que utilice sólo esos tipos de puertas
- La puerta/función NAND es funcionalmente completa y la puerta/función NOR también

✍ Hallar expresiones equivalentes usando sólo NANDs/NORs:

$\overline{A} =$ $=$

$A \cdot B =$ $=$

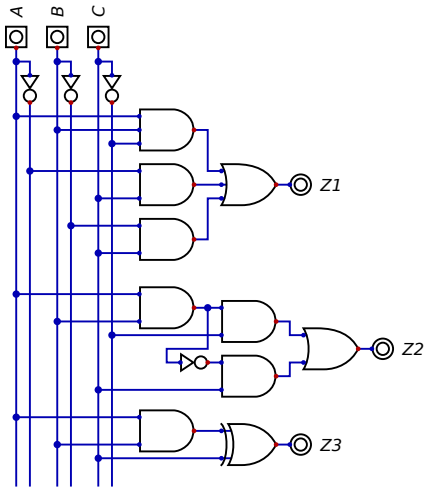
$A + B =$ $=$

$\overline{A \cdot B} =$ $\overline{A \cdot B}$ $=$

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

$A \oplus B =$ $=$

4.2 Diversidad de implementaciones



Dos niveles (inversores no cuentan):
$Z_1 = (A \cdot B \cdot \overline{C}) + (\overline{A} \cdot C) + (\overline{B} \cdot C)$

Multinivel; las líneas no atraviesan el mismo nº de puertas:
$Z_2 = ((A \cdot B) \cdot \overline{C}) + (\overline{(A \cdot B)} \cdot C)$

Puertas complejas (menos puertas):
$Z_3 = (A \cdot B) \oplus C$

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

5 Formas canónicas

- La tabla de verdad es única para cada función booleana
- Existen muchas expresiones de una misma tabla de verdad
- *Formas canónicas*: formas estándar de expresiones booleanas
  1. Forma *suma de minitérminos*, forma canónica de suma de productos o forma normal disyuntiva
  2. Forma *producto de maxitérminos*, forma canónica de producto de sumas o forma normal conjuntiva
- Las formas canónicas son *únicas* para cada función
- Las formas canónicas en general *no* son expresiones minimizadas

5.1 1ª FC: Suma de Minitérminos

A	B	C	F	Minitérmino
0	0	0	0	$\overline{A} \cdot \overline{B} \cdot \overline{C} = m_0$
0	0	1	0	$\overline{A} \cdot \overline{B} \cdot C = m_1$
0	1	0	0	$\overline{A} \cdot B \cdot \overline{C} = m_2$
0	1	1	***1***	$\overline{A} \cdot B \cdot C = m_3$
1	0	0	***1***	$A \cdot \overline{B} \cdot \overline{C} = m_4$
1	0	1	***1***	$A \cdot \overline{B} \cdot C = m_5$
1	1	0	***1***	$A \cdot B \cdot \overline{C} = m_6$
1	1	1	***1***	$A \cdot B \cdot C = m_7$

Minitérmino: Producto de literales en el que cada variable aparece *exactamente* una vez, *complementada si es 0* o sin complementar si es 1

Forma canónica: Suma de los minitérminos con salida *1*

$$F(A,B,C) = \sum m(3,4,5,6,7) = m_3 + m_4 + m_5 + m_6 + m_7$$
$$= \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

5.2 2ª FC: Producto de Maxitérminos

A	B	C	F	Maxitérmino
0	0	0	**0**	$A+B+C=M_0$
0	0	1	**0**	$A+B+\overline{C}=M_1$
0	1	0	**0**	$A+\overline{B}+C=M_2$
0	1	1	1	$A+\overline{B}+\overline{C}=M_3$
1	0	0	1	$\overline{A}+B+C=M_4$
1	0	1	1	$\overline{A}+B+\overline{C}=M_5$
1	1	0	1	$\overline{A}+\overline{B}+C=M_6$
1	1	1	1	$\overline{A}+\overline{B}+\overline{C}=M_7$

Maxitérmino: Suma de literales en los que cada variable aparece *exactamente* una vez, *complementada si es 1* o sin complementar si es 0

Forma canónica: Producto de maxitérminos con salida *0*

$$F(A,B,C) = \prod M(0,1,2) = M_0 \cdot M_1 \cdot M_2$$
$$= (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+C)$$

6 Func. incompletamente especificadas

Con n entradas hay 2^n posibles combinaciones de entradas, pero puede que algunas de ellas sean imposibles

- Su salida no importa
- En tabla de verdad, X en la salida de la fila que no importa
- En formas canónicas, d/D (*don't care*)
  - $F = \sum m(\ldots) + \sum d(\ldots)$
  - $F = \prod M(\ldots) \cdot \prod D(\ldots)$
- Se puede aprovechar para simplificar el circuito asumiendo lo que más convenga (1/0) para cada X

7 Mapas de Karnaugh

1. Transformar la tabla de verdad en un *mapa de Karnaugh*[1] como si fuera un tablero de coordenadas

A	B	F
0	0	**1**
0	1	**0**
1	0	**1**
1	1	**0**

→

B \ A	$\overline{A}$ (0)	A (1)
$\overline{B}$ (0)	**1**	**1**
B (1)	**0**	**0**

→ $F = \overline{B}$

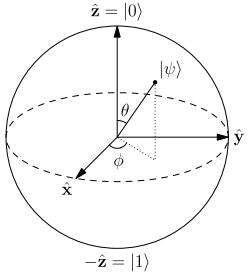
2. Seleccionar conjuntos *lo más grandes posible* de 2^n elementos iguales
3. Ver qué términos tienen en común sus elementos
4. Conjuntos de 1s representados como *sumas de productos* y de 0s como *productos de sumas con términos negados*

[1] M. Karnaugh (1953), E. Veitch (1952), A. Marquand (1881)

7 Mapas de Karnaugh (II)

- Teniendo dos variables por fila/columna la secuencia de valores en la tabla debe ser *obligatoriamente* 00, 01, 11, 10
- Los conjuntos se pueden solapar
- Se pueden hacer conjuntos por los bordes
- Cada X que aparezca puede valer lo que más convenga
- Si los conjuntos no son lo más grandes posible, la función resultante NO estará minimizada
- Cuando hay 5 (o más) variables podemos hacer mapas de 4 variables fijando el valor del resto de variables V para cada uno de los mapas
  - Podemos hacer conjuntos entre los dos mapas si están exactamente en la misma posición del mapa, en cuyo caso no aparecerá la variable V en la expresión del conjunto
  - En los conjuntos que sólo aparezcan en uno de los mapas, aparecerá la variable V con el valor de dicho mapa

Curiosidad: bit cuántico (qubit)



$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, con $\alpha, \beta \in \mathbb{C}$

- Un qubit está en todos sus estados posibles (∞) a la vez
- Los qubits están almacenados en ubicaciones fijas y se les "inducen" funciones/puertas
- Las puertas para qubits son completamente distintas a las del Álgebra de Boole
- Tras aplicar puertas, se puede obtener la probabilidad de que el qubit se encuentre en cierto estado (e.g. 0,1)

Créditos de material reutilizado

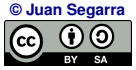
Imagen «Máquina diferencial» (p. 3): CC BY SA Science Museum London (bordes eliminados)
Imagen «Máquina analítica» (p. 3): CC BY Marcin Wichary
Imagen «Bombilla de Edison» (p. 3): PD
Imagen «Válvula de vacío» (p. 3): Daredot
Imagen «Transistor» (p. 3): CC BY SA Aminba1376
Imagen «Intel C4004» (p. 3): PD Peter1912
Imagen «IBM PC 5150» (p. 3): CC BY SA Boffy b
Imagen «Esfera de Bloch» (p. 17): CC BY SA Glosser.ca

Introducción a los computadores

Tema 2 – Representación numérica

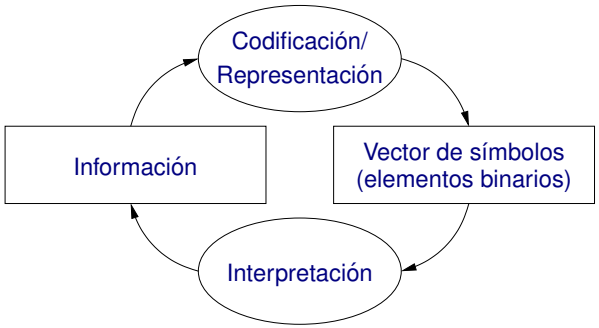
Juan Segarra y Alejandro Valero

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza



Índice

1 Introducción



- Una misma información se puede codificar/representar de distintas formas
  - Un cuatro se puede representar como 4 o como IV
- Un mismo símbolo se puede interpretar de distintas formas
  - "$+$" puede ser un operador aritmético (suma) o lógico (OR)

1 Introducción (II)

Limitaciones del computador:

- Sólo puede almacenar 0s y 1s
  - Cualquier información se debe codificar en 0s y 1s
  - Ayuda: Uso de base 2 (dígitos 0/1) en vez de base 10
- Tiene un espacio de almacenamiento limitado
  - Limitación en el número de bits a usar $\rightarrow$ rangos de representación ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$) y precisión

Longitudes usuales de datos en un computador:

- Bit (bit): binary digit (1/0)
- Byte (B): vector de 8 bits

2 Sistema de numeración posicional

$$X = \sum x_i \cdot b^i$$

siendo X el valor a representar, x_i el dígito i de la representación y b la base numérica

- Base 10: $b = 10$ y $x_i \in \{0, 9\}$; base 2: $b = 2$ y $x_i \in \{0, 1\}$

E.g.: $423{,}5_{10} = 4 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 5 \cdot 10^{-1}$
$1101{,}1_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} = 13{,}5_{10}$

- Con n dígitos se pueden representar b^n valores
- El dígito x_{n-1} (izquierda) es el de mayor peso o más significativo (MSB), y el de la derecha el de menor peso o menos significativo (LSB)

2 Sistema de numeración posicional (II)

Binario $\rightarrow$ decimal: Desglosar en dígitos multiplicados por la base con su exponente y realizar los cálculos

Parte natural decimal $\rightarrow$ binario: Dividir sucesivamente el número entre 2 hasta que el cociente sea 0 ó 1 y tomar ese cociente y los restos de forma ascendente

Parte fraccionaria decimal $\rightarrow$ binario: Multiplicar sucesivamente la parte fraccionaria del número por 2 hasta que desaparezca y tomar las partes enteras en orden descendente

Base 8 y 16: cada dígito *octal* (0-7) corresponde a tres binarios y cada *hexadecimal* a cuatro (letras A-F para dígitos 10-15)

Decimal	Binario	Octal	Hexadecimal
5	101	5 (05)	5 (0x5)
9	1001	11 (011)	9 (0x9)
10	1010	12 (012)	A (0xA)
31	11111	37 (037)	1F (0x1F)

3 Representación de enteros (n bits)

- Signo-Magnitud
  - El bit más significativo (MSB) indica el signo (0=$+$, 1=$-$) y los bits restantes indican la magnitud ($\mathbb{N}$) en binario
  - *Los positivos se representan como en binario* $\mathbb{N}$
  - Rango de representación: $[-(2^{n-1}-1), +2^{n-1}-1]$
  - Dos posibles representaciones del 0: $+0$ y -0
- Complemento a 1 (Ca1, C1)
  - Se utiliza la función $fCa1(x) = 2^n - 1 - x$ para obtener la representación con signo cambiado de $x \in \mathbb{Z}$
  - *Los positivos se representan como en binario* $\mathbb{N}$
  - En la práctica, $fCa1(x)$ se calcula intercambiando $0 \leftrightarrow 1$
  - El bit más significativo (MSB) indica el signo (0=$+$, 1=$-$)
  - Rango de representación: $[-(2^{n-1}-1), +2^{n-1}-1]$
  - Dos posibles representaciones del 0: $+0$ y -0

3 Representación de enteros (n bits) (II)

- Complemento a 2 (Ca2, C2)
  - Se utiliza la función $fCa2(x) = 2^n - x$ para obtener la representación con signo cambiado de $x \in \mathbb{Z}$
  - *Los positivos se representan como en binario* $\mathbb{N}$
  - En la práctica, se calcula como $fCa2(x) = fCa1(x) + 1$
  - El bit más significativo (MSB) indica el signo (0=$+$, 1=$-$)
  - Rango de representación: $[-2^{n-1}, +2^{n-1}-1]$
  - Una única representación posible del 0
- Dado un vector de bits, ¿qué número entero representa?
  1. Mirar el bit más significativo
  2. Si es 0, el número es positivo y se interpreta como binario $\mathbb{N}$
  3. Si es 1, el número es negativo y se interpreta según la codificación

4 Operaciones aritméticas básicas

$$\begin{array}{r} A \\ +B \\ \hline CS \end{array}$$

$$\begin{array}{r} 0+0=00 \\ 0+1=01 \\ 1+1=10 \\ 1+1+1=11 \end{array}$$

$$\begin{array}{rl} C_3C_2C_1C_0 & \leftarrow AC\text{arreos} \\ A_3A_2A_1A_0 & \leftarrow A \\ +\,B_3B_2B_1B_0 & \leftarrow B \\ \hline S_3\,S_2\,S_1\,S_0 & \leftarrow S\text{uma} \end{array}$$

- Diferenciar la «suma» lógica OR ($1+1=1$) y la suma aritmética ($1+1=2_{10}=10_2$)
- Restar es sumar cambiando el signo: $A-B=A+(-B)$
  - En Ca1, $-B=fCa1(B)$; en Ca2, $-B=fCa2(B)$
- El resultado se representa con el mismo número de bits que los operandos
- Si el resultado NO está dentro del *rango de representación* será incorrecto (desbordamiento/*overflow*)

4 Operaciones aritméticas básicas (II)

- Suma en binario $\mathbb{N}$ (n bits)
  1. Sumar $A+B$ descartando el acarreo final C_n
  2. El resultado es correcto si el acarreo generado por el bit de mayor peso es 0; hay desbordamiento si es 1: $D=C_n$
- Suma en Ca1 (n bits)
  1. Sumar $A+B$ para obtener su $acarreoFinal$
  2. Sumar recirculando acarreo: $A+B+acarreoFinal$
     - ¡Si $acarreoFinal=1$ es *imprescindible* volver a sumar!
  3. El resultado es correcto si los acarreos recibido y generado por el bit de mayor peso son iguales; hay desbordamiento si no lo son: $D=C_n\oplus C_{n-1}$
- Suma en Ca2 (n bits)
  1. Sumar $A+B$ descartando el acarreo final
  2. Comprobar si el resultado es correcto igual que en Ca1

5 Otros códigos numéricos

- Exceso-n
  - Desplaza el número en n unidades
  - Mueve el rango de representación, por ejemplo para que el 0 quede en el medio
  - E.g. exceso-7: $Representacion(x)=Valor(x)+7$
  - Facilita comparaciones ($<$, $>$)
- BCD (Decimal codificado en binario)
  - Codifica cada dígito decimal en binario con 4 bits
  - E.g. $27_{10}\rightarrow \underbrace{0010}_{2}\underbrace{0111}_{7}$
  - Facilita visualizar dígitos decimales (e.g. 27)

6.1 Tabla de interpretación (4 bits)

4 bits	Bin. $\mathbb{N}$	S-M	Ca1	Ca2	XS-7	BCD	Oct.	Hex.
0000	0	0	0	0	-7	0	0	0
0001	1	+1	+1	+1	-6	1	1	1
0010	2	+2	+2	+2	-5	2	2	2
0011	3	+3	+3	+3	-4	3	3	3
0100	4	+4	+4	+4	-3	4	4	4
0101	5	+5	+5	+5	-2	5	5	5
0110	6	+6	+6	+6	-1	6	6	6
0111	7	+7	+7	+7	0	7	7	7
1000	8	0	-7	-8	+1	8	10	8
1001	9	-1	-6	-7	+2	9	11	9
1010	10	-2	-5	-6	+3	—	12	A
1011	11	-3	-4	-5	+4	—	13	B
1100	12	-4	-3	-4	+5	—	14	C
1101	13	-5	-2	-3	+6	—	15	D
1110	14	-6	-1	-2	+7	—	16	E
1111	15	-7	0	-1	+8	—	17	F

6.2 Tabla de representación (4 bits)

Dec	Bin. $\mathbb{N}$	S-M	Ca1	Ca2	XS-7	BCD
+8	1000	—	—	—	1111	1000
+7	0111	0111	0111	0111	1110	0111
+6	0110	0110	0110	0110	1101	0110
+5	0101	0101	0101	0101	1100	0101
+4	0100	0100	0100	0100	1011	0100
+3	0011	0011	0011	0011	1010	0011
+2	0010	0010	0010	0010	1001	0010
+1	0001	0001	0001	0001	1000	0001
0	0000	0000/1000	0000/1111	0000	0111	0000
-1	—	1001	1110	1111	0110	—
-2	—	1010	1101	1110	0101	—
-3	—	1011	1100	1101	0100	—
-4	—	1100	1011	1100	0011	—
-5	—	1101	1010	1011	0010	—
-6	—	1110	1001	1010	0001	—
-7	—	1111	1000	1001	0000	—
-8	—	—	—	1000	—	—

7 Representación de números reales

- Coma fija (*fixed point*)
  - k bits parte entera, $n-k$ bits parte fraccionaria
  - E.g. ($n=16$, $k=8$) $11{,}40625_{10} = 1011{,}01101_2$:

    $\underbrace{\boxed{00001011}}_{\text{parte entera}}\ \underbrace{\boxed{01101000}}_{\text{parte frac.}}$
  - No se pueden representar números muy grandes ni muy precisos $\rightarrow$ pocos lenguajes lo implementan
- Coma flotante (*floating point*): estándar IEEE 754
  - Basado en notación científica $x = \pm m \cdot b^e$
  - Con 32 bits: 1 bit signo, 8 bits exponente (exceso-127), 23 bits mantisa más 1 bit implícito
  - E.g. $1011{,}01101 = +1{,}01101101 \cdot 2^3$:

    $\underbrace{\boxed{0}}_{+}\ \underbrace{\boxed{10000010}}_{\text{expon.: } 3+127}\ \underbrace{\boxed{01101101000000000000000}}_{\text{mantisa con bit implícito: } (1,)01101101}$
  - Códigos específicos para 0, ∞, NaN (Not a Number), etc.

Introducción a los computadores

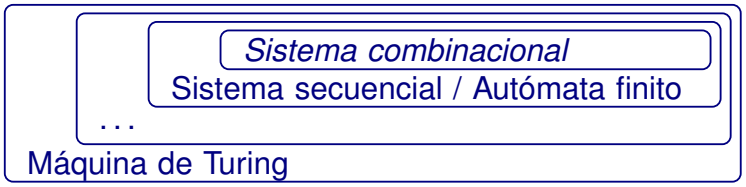
Tema 3 – Sistemas combinacionales

Juan Segarra y Alejandro Valero

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción



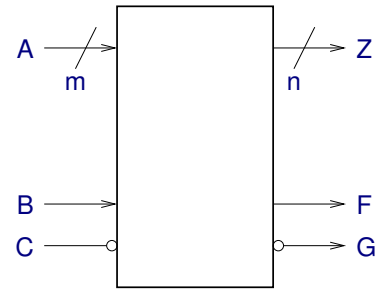
En un *circuito combinacional* el valor de la(s) salida(s) depende *solo* del valor de la(s) entrada(s)

- E.g. Un sistema que tenga como entrada un número entero y nos diga si es mayor que 0 o no
  - El resultado depende únicamente del número de la entrada
- Los sistemas combinacionales *no tienen memoria*
- Análisis: Obtener función/tabla de verdad a partir de circuito
- Síntesis: Obtener circuito a partir de función/tabla de verdad

1.1 Diseño de un sistema combinacional

1. Extraer *información* a partir de la descripción del problema: ver qué datos de entrada tenemos y qué datos debemos generar en la salida
2. Codificar dichos datos (si no lo están) en *variables lógicas*
3. Obtener la *tabla de verdad* (salidas para cada una de las posibles combinaciones de entradas)
4. Obtener una *función minimizada* para cada una de sus variables de salida a partir de su tabla de verdad
5. Dibujar el *circuito* correspondiente a cada una de sus variables de salida

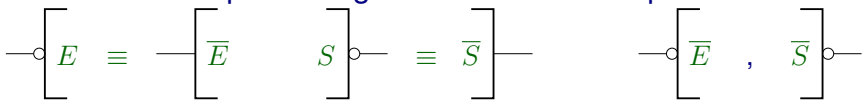
1.2 Convenios gráficos para bloques



Etiquetas

- m entradas (A)
- n salidas (Z)
- B: permiso/*enable*
- C: Inhibición (permiso nivel bajo)
- F: Salida activa
- G: Salida activa nivel bajo

Etiquetas negadas dentro del bloque

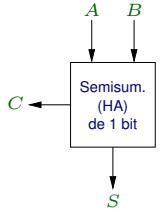


Correcto Correcto Incorrecto

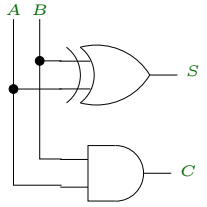
2 Sumadores

- Semisumador (*Half Adder*, HA)

$$\begin{array}{r} A \\ +B \\ \hline CS \end{array}$$

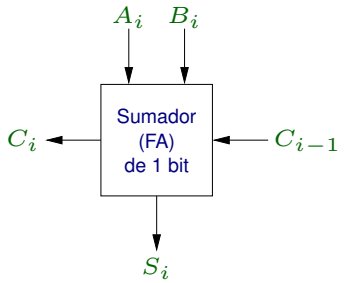


A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



- Sumador completo (*Full Adder*, FA)

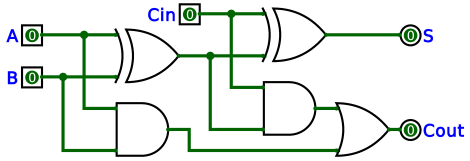
$$\begin{array}{r} C_{i-1} \\ A_i \\ +\ B_i \\ \hline C_i\ S_i \end{array}$$



A_i	B_i	C_{i-1}	C_i	S_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

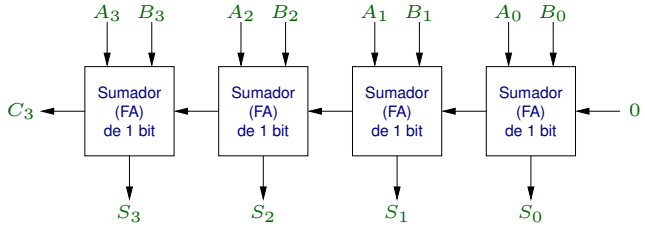
2 Sumadores (II)

- Sumador completo a partir de semisumadores



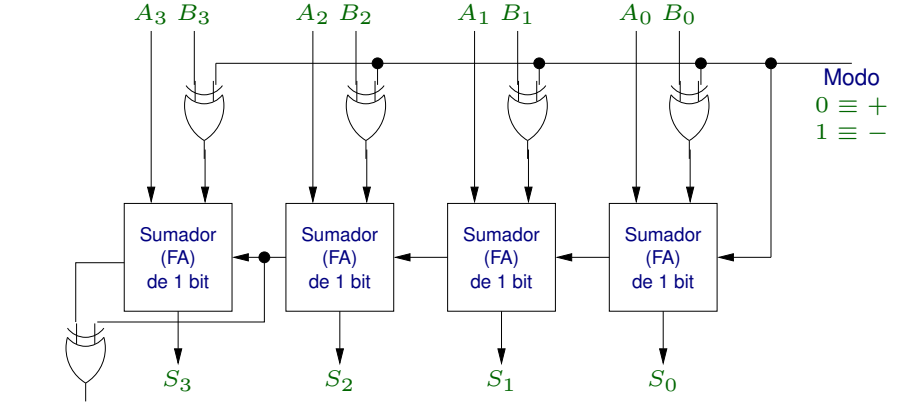
$$C_i = A_i B_i + (A_i \oplus B_i) C_{i-1}$$
$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$

- Sumador con propagación de acarreo (*Ripple Carry Adder*, RCA)



2.1 Sumador/Restador en Ca2

- Modo 0: $S = A + B + 0 = \boxed{A + B}$
- Modo 1: $S = A + fCa1(B) + 1 = A + fCa2(B) =$
  $= A + (-B) = \boxed{A - B}$

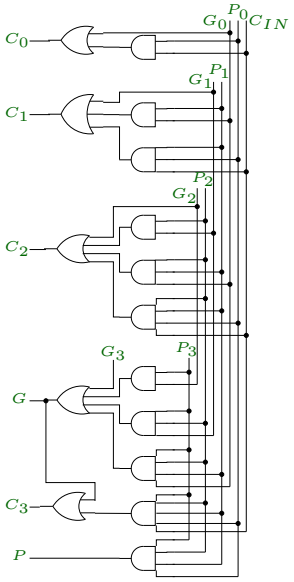


2.2 Sumador con acarreo anticipado

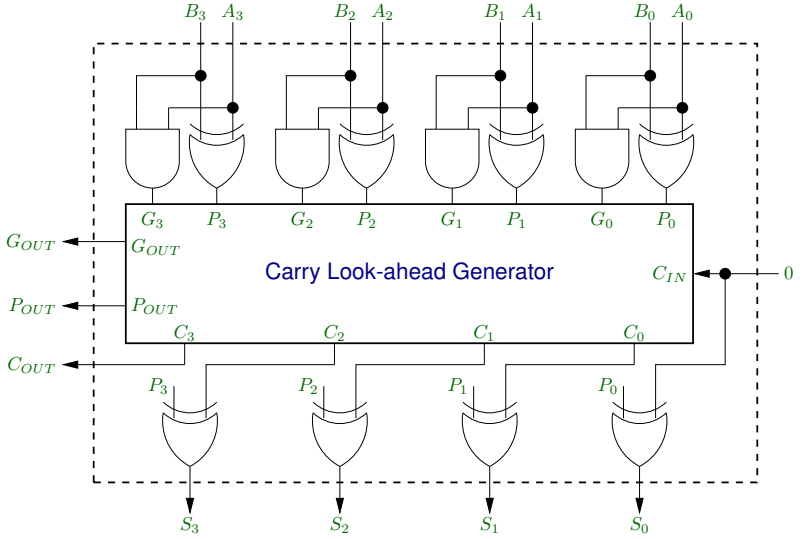
Carry Look-ahead Generator

$C_i = \underbrace{A_i B_i}_{G_i} + \underbrace{(A_i \oplus B_i)}_{P_i} C_{i-1}$

- $C_0 = G_0 + P_0 C_{IN}$
- $C_1 = G_1 + P_1 C_0 = G_1 + P_1(G_0 + P_0 C_{IN}) = G_1 + P_1 G_0 + P_1 P_0 C_{IN}$
- $C_2 = G_2 + P_2 C_1 = G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 C_{IN}) = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{IN}$
- $C_3 = G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{IN}$

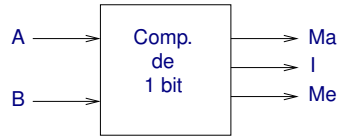


2.2 Sumador con acarreo anticipado (II)



$S_i = P_i \oplus C_{i-1}$

3 Comparadores

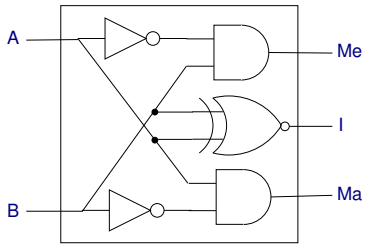


A	B	Ma(yor)	I(gual)	Me(nor)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

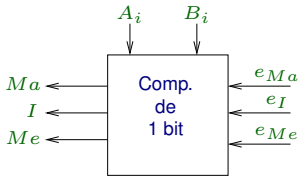
Ma $= A\overline{B}$

Me $= \overline{A}B$

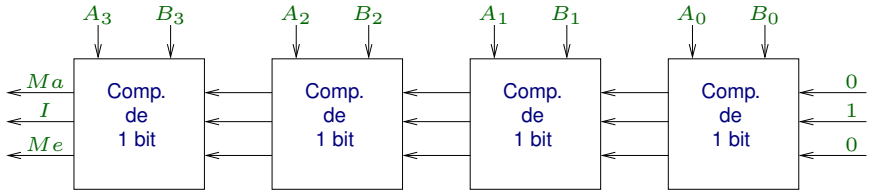
I $= \overline{A}\,\overline{B} + AB = \overline{A \oplus B} = \overline{A\overline{B} + \overline{A}B}$



3.1 Comparador encadenable



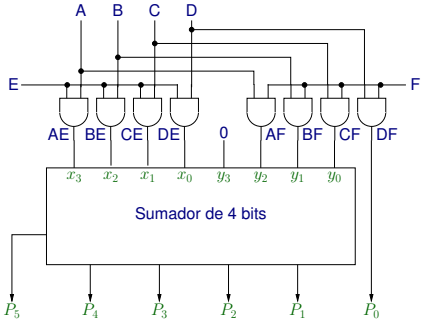
A_i	B_i	e_{Ma}	e_I	e_{Me}	Ma	I	Me
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	1	X	X	X	0	0	1
1	0	X	X	X	1	0	0
1	1	0	0	1	0	0	1
1	1	0	1	0	0	1	0
1	1	1	0	0	1	0	0



4 Multiplicador 4 bits $\times$ 2 bits

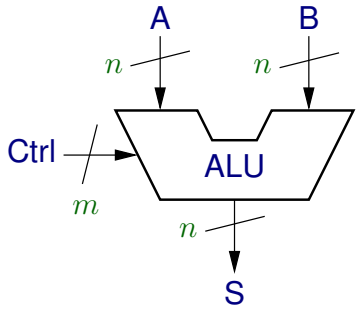
- Multiplicaciones de una cifra $\rightarrow$ puertas AND
- La suma se puede hacer con un sumador en el que uno de los números de entrada está desplazado

		A	B	C	D
		$\times$		E	F
		AF	BF	CF	DF
	AE	BE	CE	DE	
P_5	P_4	P_3	P_2	P_1	P_0



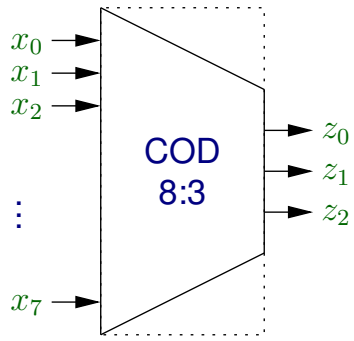
5 ALU (Unidad Aritmético-Lógica)

- Una ALU es un circuito capaz de realizar varias operaciones (especificadas mediante un código de control/operación) sobre sus operandos (Ejemplo circuito L4C381)



Ctrl	S	
000	Clear	(todo ceros)
001	$B - A$	
010	$A - B$	
011	$A + B$	(aritmética)
100	$A \oplus B$	
101	$A + B$	(lógica)
110	$A \cdot B$	(lógica)
111	Preset	(todo unos)

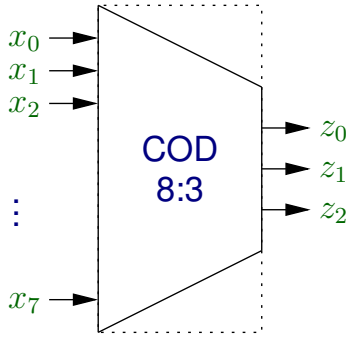
6 Codificador



- Codifica en binario natural en las líneas de salida el número i de la línea de entrada x_i por la que recibe un 1

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	$z_2 z_1 z_0$
0	0	0	0	0	0	0	1	0 0 0
0	0	0	0	0	0	1	0	0 0 1
0	0	0	0	0	1	0	0	0 1 0
0	0	0	0	1	0	0	0	0 1 1
0	0	0	1	0	0	0	0	1 0 0
0	0	1	0	0	0	0	0	1 0 1
0	1	0	0	0	0	0	0	1 1 0
1	0	0	0	0	0	0	0	1 1 1

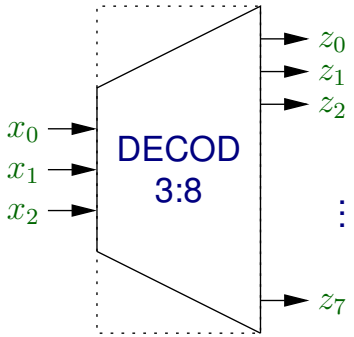
6.1 Codificador con prioridad



- Codifica en binario natural en la salida el número i de la línea de entrada x_i de más peso que recibe un 1

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	$z_2 z_1 z_0$
0	0	0	0	0	0	0	1	0 0 0
0	0	0	0	0	0	1	X	0 0 1
0	0	0	0	0	1	X	X	0 1 0
0	0	0	0	1	X	X	X	0 1 1
0	0	0	1	X	X	X	X	1 0 0
0	0	1	X	X	X	X	X	1 0 1
0	1	X	X	X	X	X	X	1 1 0
1	X	X	X	X	X	X	X	1 1 1

7 Decodificador



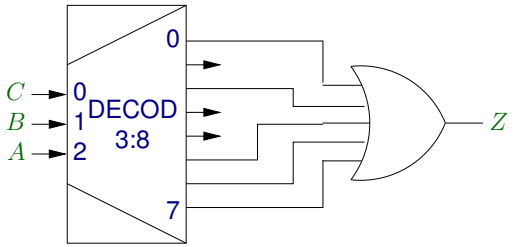
- Pone a 1 la línea de salida z_i igual al número i codificado en binario natural en la entrada $(x_2x_1x_0)$ y a 0 las demás

$x_2x_1x_0$	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
0 0 0	0	0	0	0	0	0	0	1
0 0 1	0	0	0	0	0	0	1	0
0 1 0	0	0	0	0	0	1	0	0
0 1 1	0	0	0	0	1	0	0	0
1 0 0	0	0	0	1	0	0	0	0
1 0 1	0	0	1	0	0	0	0	0
1 1 0	0	1	0	0	0	0	0	0
1 1 1	1	0	0	0	0	0	0	0

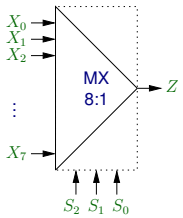
7 Decodificador (II)

- Modos de implementar funciones con un decodificador
  - Conectar las líneas de salida con 1s en la tabla mediante una puerta OR (ejemplo)
  - Conectar las líneas de salida con 0s en la tabla mediante una puerta NOR

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



8 Multiplexor



- Da como salida el valor de la línea X_i, siendo i el valor codificado en binario natural en S $(S_2S_1S_0)$
- Si el número de entradas "selectoras" es n, el número de entradas "seleccionables" es 2^n

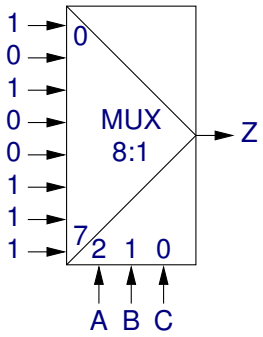
S_2	S_1	S_0	Z
0	0	0	X_0
0	0	1	X_1
0	1	0	X_2
0	1	1	X_3
1	0	0	X_4
1	0	1	X_5
1	1	0	X_6
1	1	1	X_7

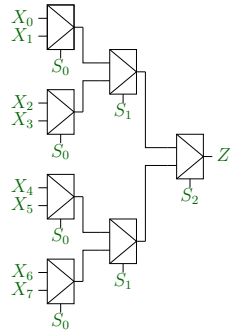
$$Z = \overline{S_2}\cdot\overline{S_1}\cdot\overline{S_0}\cdot X_0 + \overline{S_2}\cdot\overline{S_1}\cdot S_0\cdot X_1 + \overline{S_2}\cdot S_1\cdot\overline{S_0}\cdot X_2 + \overline{S_2}\cdot S_1\cdot S_0\cdot X_3 + S_2\cdot\overline{S_1}\cdot\overline{S_0}\cdot X_4 + S_2\cdot\overline{S_1}\cdot S_0\cdot X_5 + S_2\cdot S_1\cdot\overline{S_0}\cdot X_6 + S_2\cdot S_1\cdot S_0\cdot X_7$$

8 Multiplexor (II)

- Implementación de funciones → Se pone la columna de salida de la tabla como entradas del multiplexor y las variables como entradas selectoras (ejemplo 1)
- Se pueden construir multiplexores grandes a partir de pequeños encadenándolos "en árbol" (ejemplo 2)

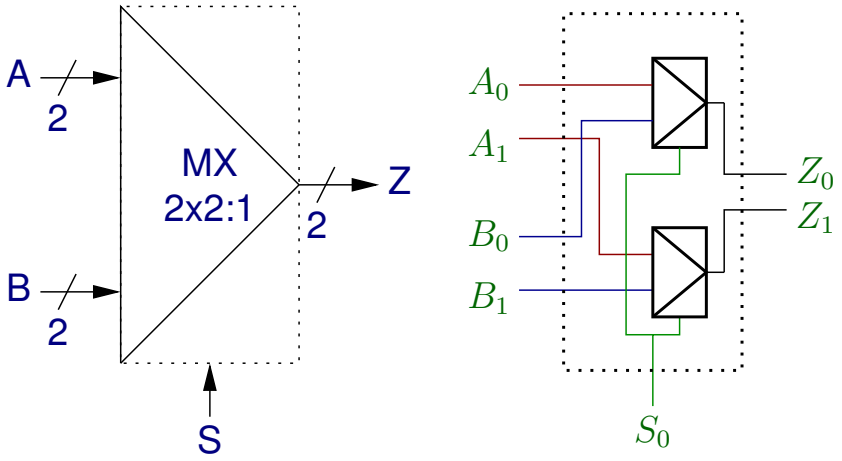
A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



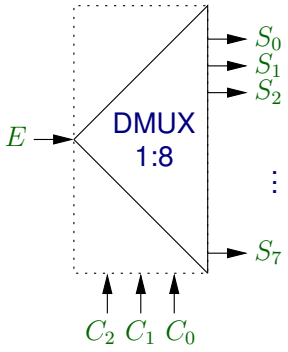


8 Multiplexor (III)

Multiplexor de varios canales/vectores



9 Demultiplexor

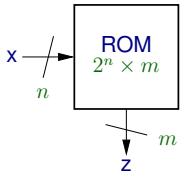


- Pone el valor de E en la línea S_i y 0 en el resto, siendo i el valor codificado en binario natural en C $(C_2C_1C_0)$
- Si el número de entradas "selectoras" es n, el número de salidas es 2^n

$C_2C_1C_0$	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0 0 0	0	0	0	0	0	0	0	E
0 0 1	0	0	0	0	0	0	E	0
0 1 0	0	0	0	0	0	E	0	0
0 1 1	0	0	0	0	E	0	0	0
1 0 0	0	0	0	E	0	0	0	0
1 0 1	0	0	E	0	0	0	0	0
1 1 0	0	E	0	0	0	0	0	0
1 1 1	E	0	0	0	0	0	0	0

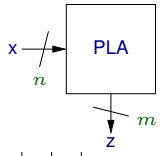
✍ Si E fuera un *enable*, ¿qué sería el demultiplexor?

10 ROM (Read-Only Memory)

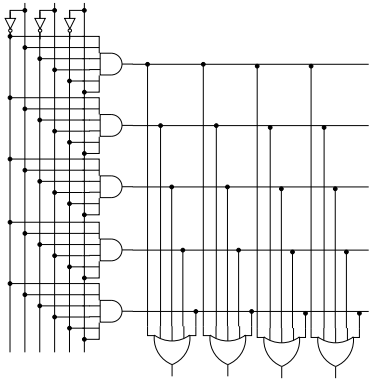


- Una ROM realiza m funciones (predefinidas) de n variables cada una
- Las variables n suelen verse como una *dirección* de la memoria ROM y las m salidas como el *contenido* de la dirección n
- Al "diseñar" una ROM hay que especificar su contenido (tabla de verdad) al fabricante, no cómo construirla
- Una ROM típica no permite modificar su contenido (read-only)
- Otros tipos de ROM: PROM/PLA (*programmable*), EPROM (*erasable* por luz ultravioleta), EEPROM/E2PROM (*electrically erasable*), Flash (EEPROM programable por bloques grandes)

10.1 PLA (Programmable Logic Array)



- Es una ROM especialmente diseñada para ser programada con facilidad

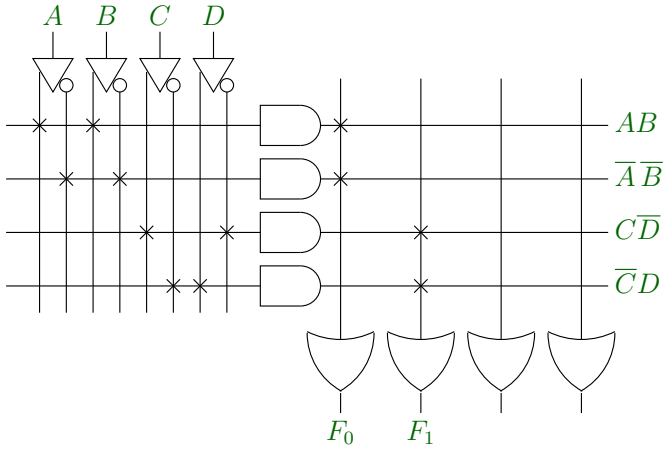


- Un PLA sale "de fábrica" con todas las conexiones hechas y programarlo implica romper las que no se usan

10.1 PLA (Programmable Logic Array) (II)

- Notación abreviada de PLA
- Ejemplo de conexiones:

$F_0 = AB + \overline{A}\,\overline{B} \qquad F_1 = C\overline{D} + \overline{C}D$



Introducción a los computadores

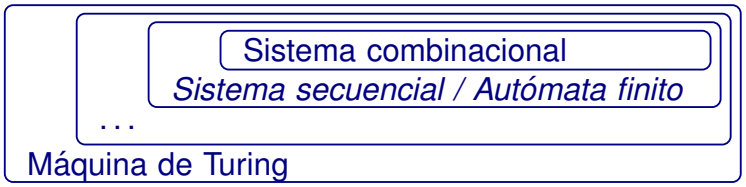
Tema 4 – Sistemas secuenciales

Juan Segarra y Alejandro Valero

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

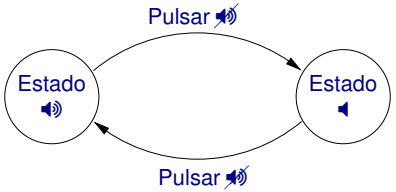
Índice

1 Introducción



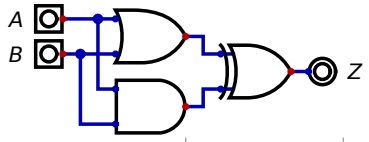
En un *circuito secuencial* el valor de la(s) salida(s) depende de la *secuencia* de entradas hasta el momento

- *Estados* para «resumir» secuencias de entrada
- E.g. Estado «mudo» del televisor

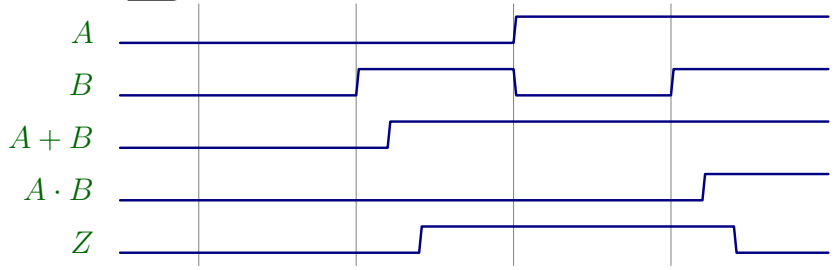


2 La señal de reloj

- Las puertas lógicas generan la salida con cierto retardo
- Pueden aparecer pulsos cortos «incorrectos» (*glitches*)
- Cronograma: Diagrama con los valores respecto al tiempo



A	B	$A+B$	$A \cdot B$	Z
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

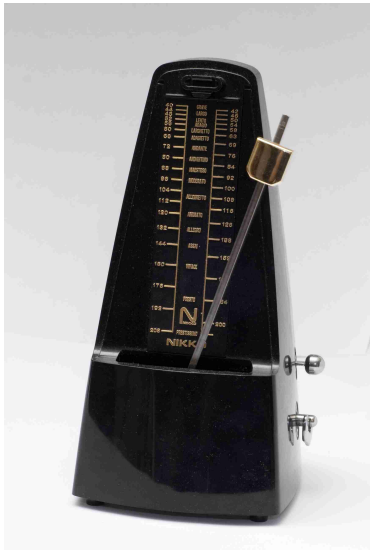


2 La señal de reloj (II)

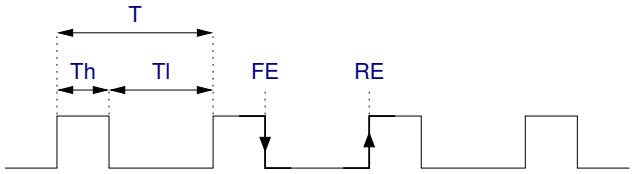
- El estado sólo debe cambiar cuando las señales son *estables*

Reloj: Señal periódica distribuida por un circuito que marca cuándo pueden cambiar de estado los elementos de memoria

- Se mide en *hertzs* o *hercios*: 1 Hz=1 ciclo/s
- Los sistemas con reloj se llaman *síncronos*



2 La señal de reloj (III)



T = periodo T = Thigh + Tlow duty-cycle = Thigh/T
FE = Falling Edge (Flanco desc.) RE = Rising Edge (F. asc.)

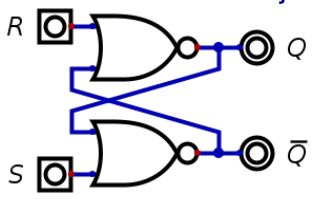
Si tengo una CPU a $2{,}8$ GHz...

- su reloj interno está a nivel alto/bajo $2{,}8 \times 10^9$ veces cada segundo y su periodo es de $\frac{1}{2{,}8\times10^9} = 3{,}57 \times 10^{-10}$ s
- el retardo máximo de las puertas entre dos elementos que cambien de estado debe ser menor que 357 ps para que el sistema funcione correctamente

3 Elementos de memoria

- Estado (codificado) = conjunto de bits ($Q_{n...0}$)
- Para guardar cada bit se necesita un sistema capaz de almacenar dos valores (0/1) de forma estable $\rightarrow$ *Biestable*

Ejemplo biestable R-S

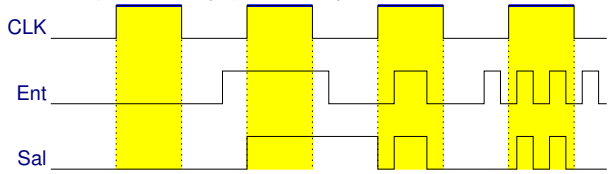


R	S	Q	$\overline{Q}$	
1	0	0	1	salida Q=0 ($\overline{Q}$=1)
0	1	1	0	salida Q=1 ($\overline{Q}$=0)
0	0	Q	$\overline{Q}$	salida Q=memoria
1	1	0	0	incorrecto (Q=$\overline{Q}$)

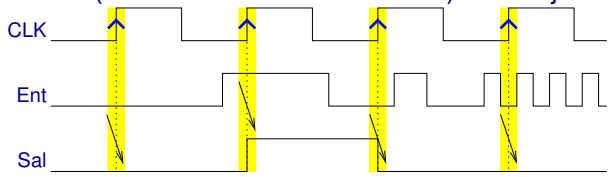
- Poniendo ciertos valores en sus entradas, permite almacenar/generar en Q un valor concreto conocido (0/1) o el valor anterior, incluso sin conocer cuál era

3.1 Clasificación de biestables

1. Sin reloj (ejemplo: biestable R-S anterior)
2. Con reloj: sincr. por nivel (*latch*) o por flanco (*flip-flop*)
   - Latch: Capta entradas y genera salidas en uno de los niveles (alto o bajo) de reloj



   - Flip-flop: Capta entradas y genera salidas en uno de los flancos (ascendente o descendente) de reloj



3.2 Especificación de flip-flops

Tres formas equivalentes de especificar su funcionamiento:

Tabla de transición: Especifica la salida (estado) futura generada por las entradas y el estado actual

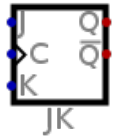
Ecuación característica: Tabla de transición expresada como función lógica

Tabla de excitación: Especifica las entradas necesarias para pasar de un estado al siguiente

Ejemplo flip-flop J-K

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

T. Transición



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

E. Característica

$Q \to Q^+$	J	K
0 0	0	X
0 1	1	X
1 0	X	1
1 1	X	0

T. Excitación

3.2 Especificación de flip-flops (II)

Flip-flop *D*

$Q^+ = D$

D	Q	Q^+
0	X	0
1	X	1

$Q \to Q^+$	D
0 0	0
0 1	1
1 0	0
1 1	1

Flip-flop *T*

$Q^+ = T \oplus Q$

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

$Q \to Q^+$	T
0 0	0
0 1	1
1 0	1
1 1	0

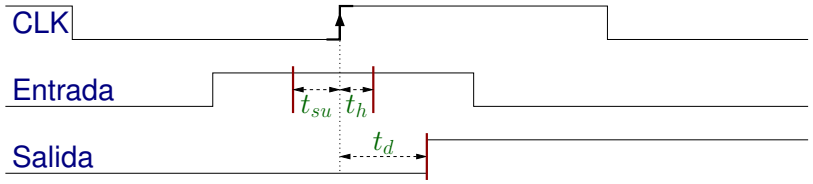
Flip-flop *R-S*

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

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

$Q \to Q^+$	S	R
0 0	0	X
0 1	1	0
1 0	0	1
1 1	X	0

3.3 T. característicos de biestables



Tiempo de *setup* t_{su}: tiempo mínimo en el que las entradas deben mantenerse estables antes del flanco

Tiempo de *hold* t_h: tiempo mínimo en el que las entradas deben mantenerse estables después del flanco

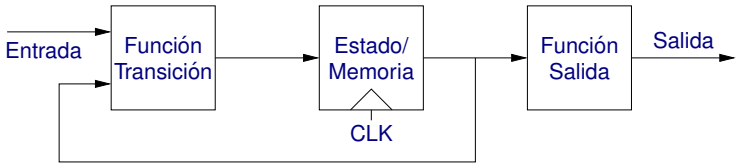
tiempo de *delay*/retardo t_d: tiempo que tarda el biestable en actuar y propagar una salida correcta

- Si la entrada cambia en el rango $[-t_{su}, t_h]$ la salida del flip-flop será incorrecta
- Entradas *asíncronas* son aquellas que «no esperan» al flanco de reloj (e.g. reset)

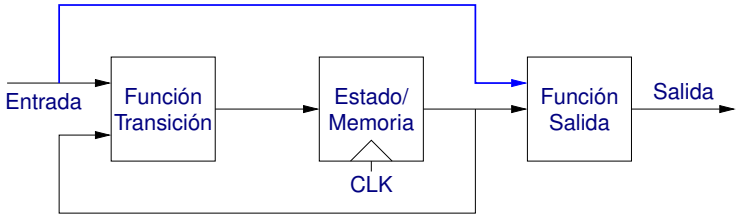
4 Circuitos secuenciales síncronos

- El estado sólo cambia en los instantes marcados por el reloj
- Dos formas para especificar máquinas de estados finitos:

Moore:



Mealy:



4 Circuitos secuenciales síncronos (II)

- Moore: $Z(t) = f(S(t)) = f(X(-\infty, \ldots, t^-))$
  - La salida sólo depende del estado actual (que depende de las entradas anteriores)
  - La salida sólo puede cambiar en los flancos de reloj
  - Un mismo estado siempre genera una misma salida
  - $\sim$ Diagramas con más estados
  - $\sim$ Funciones de salida más simples
- Mealy: $Z(t) = f(S(t), X(t)) = f(X(-\infty, \ldots, t))$
  - La salida depende del estado actual y de la entrada actual
  - La salida puede cambiar en los flancos de reloj y ante cambios en la entrada
  - Un mismo estado puede generar distintas salidas
  - $\sim$ Diagramas con menos estados
  - $\sim$ Funciones de salida más complejas

4 Circuitos secuenciales síncronos (III)

Diagrama de estados: Grafo dirigido que representa las funciones de transición y de salida

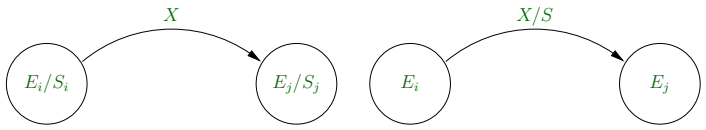


Diagrama Moore Diagrama Mealy

$E_1 \ldots E_k$: Estados $1 \ldots k$, identificados por valores concretos de n variables de estado $Q_{n-1\ldots0}$ (memoria), con $k \leq 2^n$

S: Salida, en Moore asociada a un estado y en Mealy a una transición (un estado con una entrada)

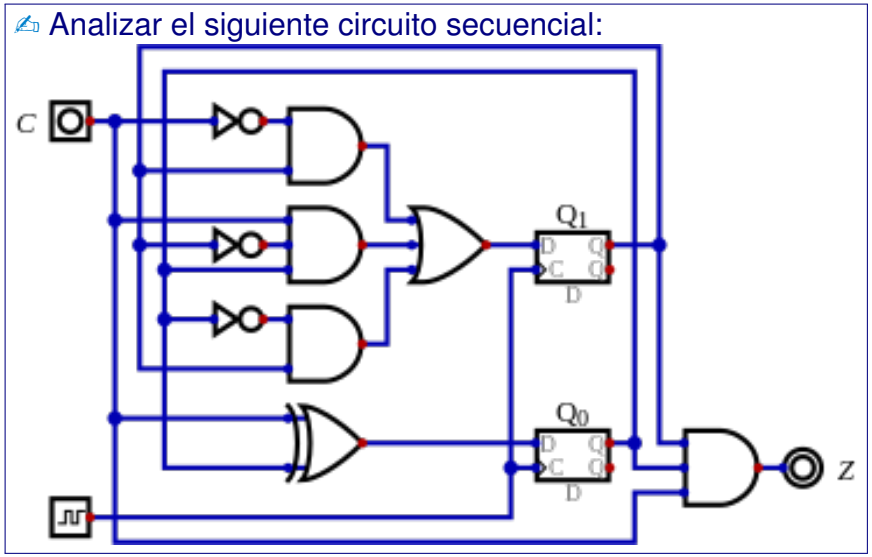
X: Valor de la variable (o variables) de entrada que provoca la transición de estados

4.1 Análisis de circuitos secuenciales

1. Obtener *funciones de entrada de flip-flops* y *funciones de salida* del circuito
   - Si las salidas están en función de las entradas del circuito, el circuito es tipo Mealy, si no es Moore
2. Obtener las *funciones de transición* (estado futuro en función de entradas y estado actual) de los flip-flops a partir de las ecuaciones características y las funciones de entrada de los flip-flops obtenidas en el paso 1
3. Dibujar la tabla de verdad, especificando estados futuros y salidas *actuales* a partir de estados y entradas actuales
4. Obtener diagrama de estados a partir de tabla de verdad
5. Seguir el diagrama para ver qué hace

4.1.1 Ejercicio de análisis secuencial

✍ Analizar el siguiente circuito secuencial:

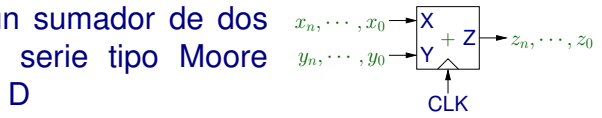


4.2 Diseño de circuitos secuenciales

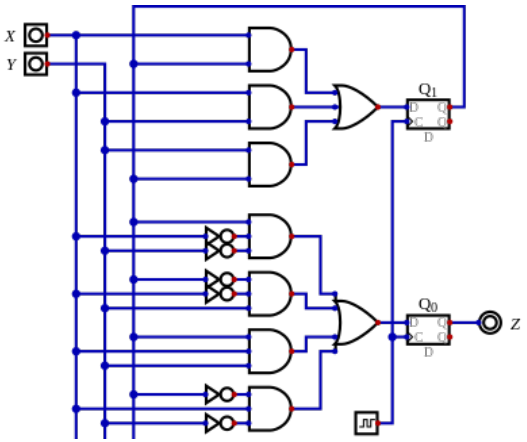
1. Trasladar las especificaciones del problema al tipo de diagrama más adecuado (Moore o Mealy)
2. Codificar las entradas, salidas y estados
3. Dibujar la tabla de verdad, especificando estados futuros y salidas *actuales* a partir de estados y entradas actuales
4. Elegir el tipo de flip-flops a utilizar y añadir a la tabla las entradas de los flip-flops elegidos (usar tabla de excitación)
5. Minimizar las funciones de entrada de los flip-flops y las salidas del circuito
6. Dibujar el circuito resultante

4.2.1 Ejercicio: sumador serie Moore

✍ Diseñar un sumador de dos números en serie tipo Moore con flip-flops D

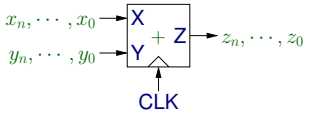


Solución:

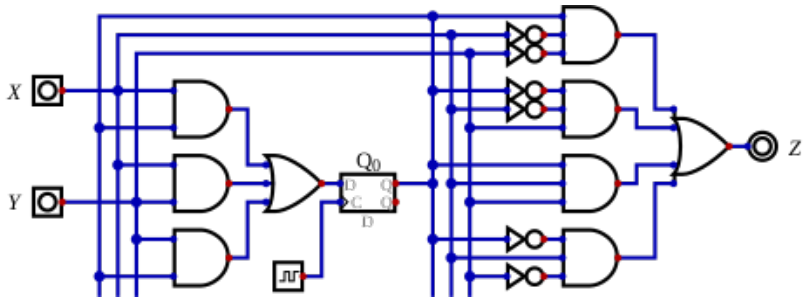


4.2.2 Ejercicio: sumador serie Mealy

✍ Diseñar un sumador de dos números en serie tipo Mealy con flip-flops D

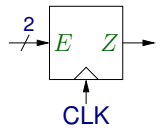


Solución:

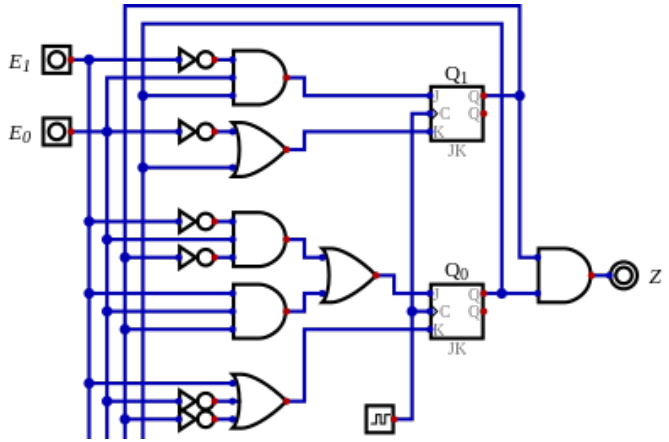


4.2.3 Ejercicio: reconocedor 113 Moore

✍ Diseñar un reconocedor ($Z = 1$) de la secuencia de entradas 113 ($E = \{0, 1, 2, 3\}$) tipo Moore con flip-flops JK

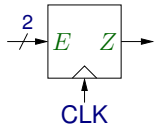


Solución:

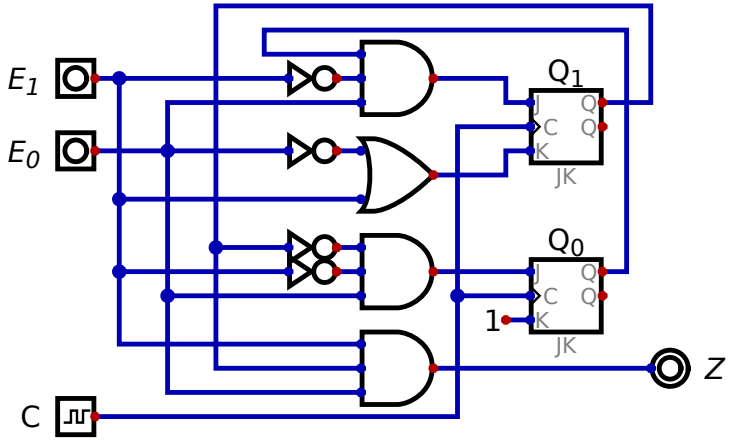


4.2.4 Ejercicio: reconocedor 113 Mealy

✍ Diseñar un reconocedor ($Z = 1$) de la secuencia de entradas 113 ($E = \{0, 1, 2, 3\}$) tipo Mealy con flip-flops JK

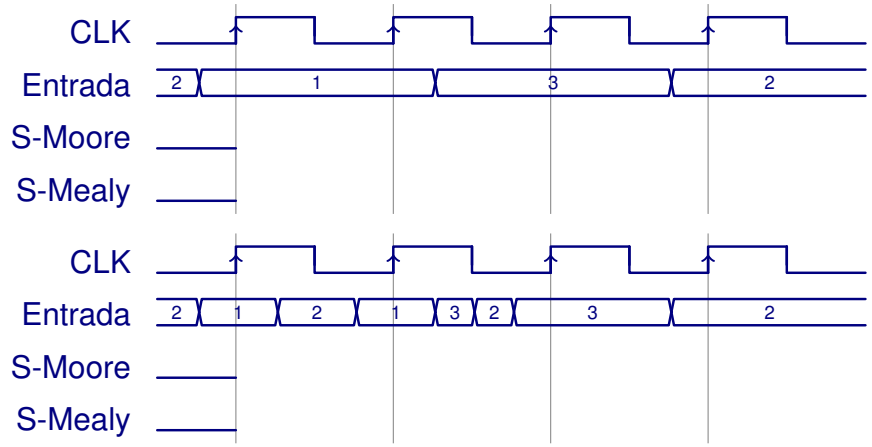


Solución:



4.2.5 Ejercicio: cronograma

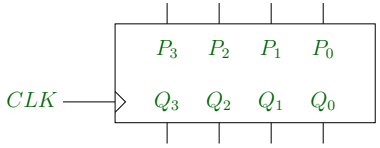
✍ Completar los siguientes cronogramas del reconocedor 113



5 Bloques secuenciales

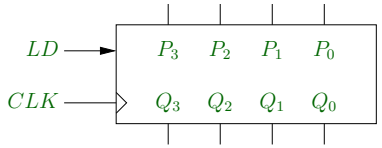
Registro: Circuito secuencial síncrono capaz de almacenar información temporalmente

- Registro básico de 4 bits



CLK	Q_3^+	Q_2^+	Q_1^+	Q_0^+
$\uparrow$	P_3	P_2	P_1	P_0

- Registro de 4 bits con carga paralela

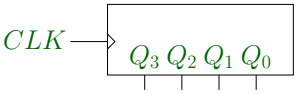


CLK	LD	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
$\uparrow$	0	Q_3	Q_2	Q_1	Q_0	No cambia
$\uparrow$	1	P_3	P_2	P_1	P_0	Carga paralela

5 Bloques secuenciales (II)

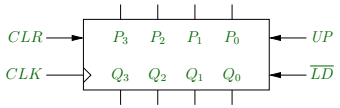
Contador: Circuito secuencial cuya salida representa el número de impulsos que han aparecido en la entrada

- Contador de 4 bits (módulo 16) en binario natural con entrada CLK



CLK	Q_3^+	Q_2^+	Q_1^+	Q_0^+
$\uparrow$	$(Q+1)$ mód 16			

- Contador *up/down* (UP) de 4 bits (módulo 16) en binario natural con carga (*load* $\overline{LD}$) paralela síncrona y clear asíncrono CLR

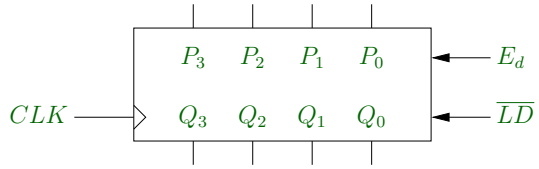


CLK	CLR	LD	UP	Q^+
X	1	X	X	0
$\uparrow$	0	0	X	P
$\uparrow$	0	1	1	$(Q+1)$ mód 16
$\uparrow$	0	1	0	$(Q-1)$ mód 16

5 Bloques secuenciales (III)

Registro de desplazamiento (*shift register*): Circuito secuencial síncrono capaz de efectuar conversiones serie/paralelo en la transmisión de datos y de almacenar información temporalmente

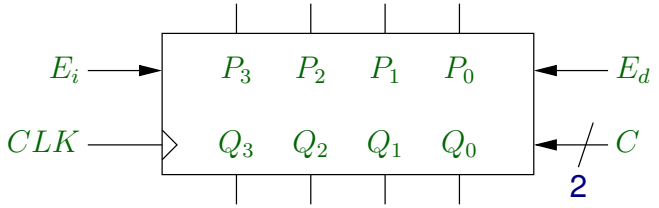
- Registro de desplazamiento a izquierda con carga paralela ($\overline{LD}$) de datos P



CLK	LD	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
$\uparrow$	0	P_3	P_2	P_1	P_0	Carga paralela
$\uparrow$	1	Q_2	Q_1	Q_0	E_d	Despl. izquierda

5 Bloques secuenciales (IV)

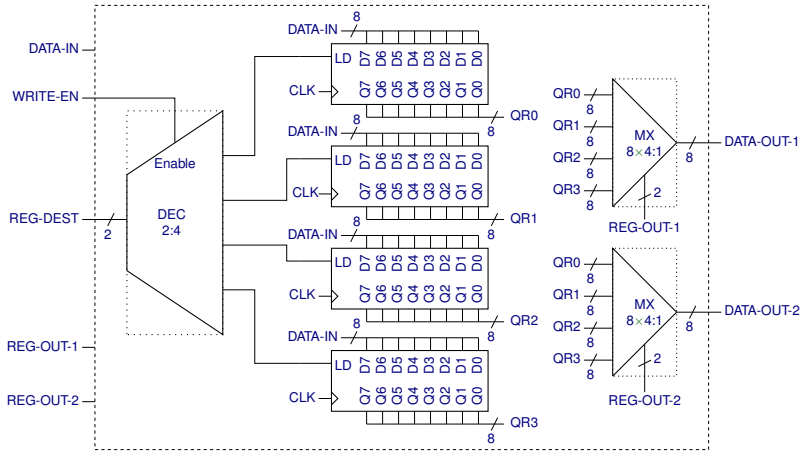
- Registro de desplazamiento a ambos lados con carga paralela de datos P



CLK	C_1	C_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
$\uparrow$	0	0	Q_3	Q_2	Q_1	Q_0	No cambia
$\uparrow$	0	1	Q_2	Q_1	Q_0	E_d	Despl. izquierda
$\uparrow$	1	0	E_i	Q_3	Q_2	Q_1	Despl. derecha
$\uparrow$	1	1	P_3	P_2	P_1	P_0	Carga paralela

5 Bloques secuenciales (V)

- Banco de registros



Ejemplo: banco de 4 registros de 8 bits con un puerto de escritura DATA-IN y dos puertos de lectura DATA-OUT

6 Memorias RAM

Permiten el acceso directo (no secuencial) a cualquier posición de la memoria (*Random Access Memory*)

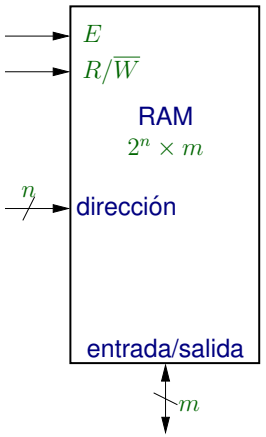
ROM (coloquialmente no suele incluirse en las RAM)

Memoria de lectura/escritura

- Volátil: pierde su contenido al apagarla
  - SRAM (*Static RAM*): almacena bits en biestables
    - Rápida y cara
    - Usada en caches
  - DRAM (*Dynamic RAM*): almacena bits en condensadores
    - Necesita realimentación periódica
    - Barata pero lenta
    - Usada en memoria principal
- No volátil: mantiene el contenido incluso apagada
  - Suele usar tecnología ROM (e.g. memoria *flash*)
  - Coloquialmente no suele incluirse en las RAM

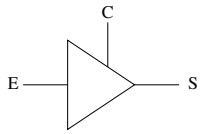
6 Memorias RAM (II)

- Entrada: dirección de n bits (selecciona la información a consultar)
- Control:
  - E: Permiso (puede tener otros nombres: *Chip Select*, *Output Enable*, etc.)
  - $R/\overline{W}$ ($L/\overline{E}$): Especifica si queremos o leer o escribir
- Entrada/Salida: m bits de datos



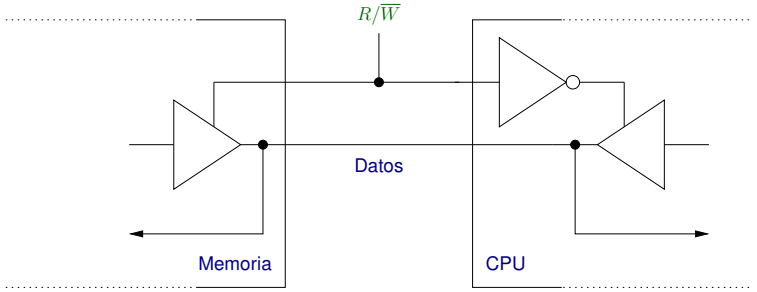
6 Memorias RAM (III)

- Puertas *triestado*: Permiten "desconectar" la línea



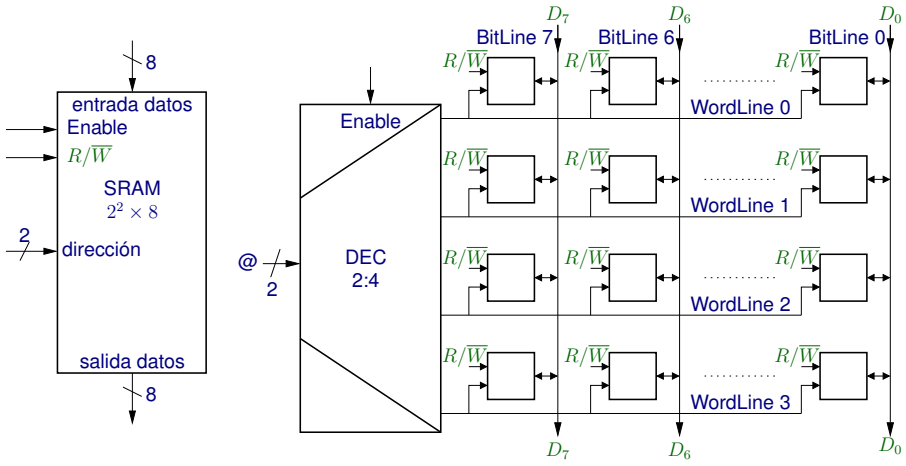
C	E	S
0	X	Alta impedancia (Z)
1	0	0
1	1	1

- Ejemplo de líneas entrada/salida con puertas triestado



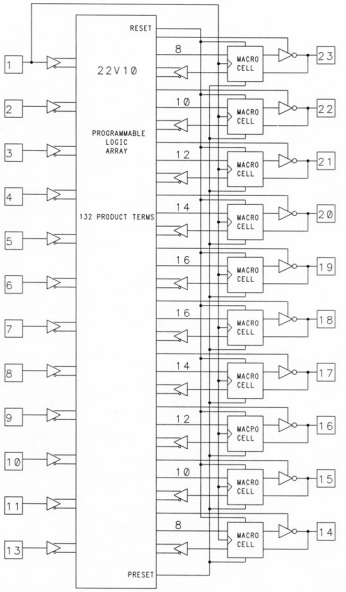
6 Memorias RAM (IV)

- Ejemplo SRAM

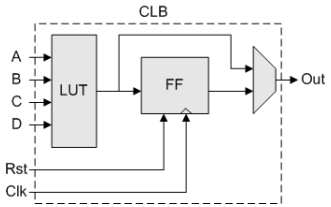


7 PLDs (Programmable Logic Device)

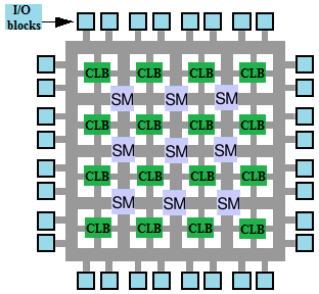
- PAL (*Programmable Array Logic*): PLA con flip-flops
- Permite sintetizar sistemas secuenciales
- Pueden incluir todo lo visto anteriormente (tecnología de reprogramación, líneas de entrada/salida con puertas triestado, etc.)
- Figura: AMD PAL 22V10



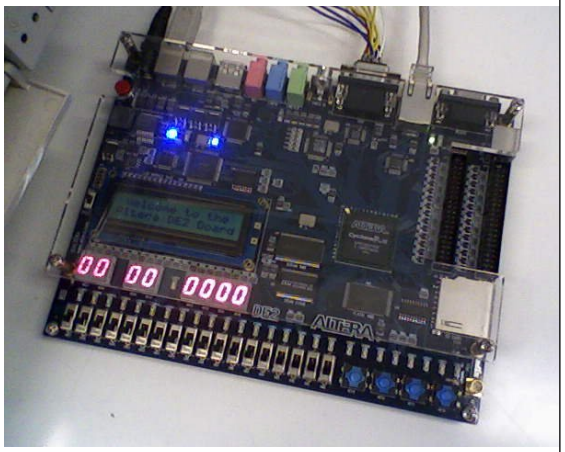
7.1 FPGAs (Field Program. Gate Array)



Celda FPGA



FPGA



Placa FPGA

Créditos de material reutilizado

Imagen «Metrónomo Nikko» (p. 5): Vincent Quach (Invincible)
Imagen «AMD PAL 22V10» (p. 32): sin restricciones, en:User:Swtpc6800/Hoenny
Imagen «Celda FPGA» (p. 33): Ldvbin at English Wikibooks
Imagen «FPGA» (p. 33): Vorobev.2017
Imagen «ALTERA DE2 FPGA Board» (p. 33): Thotypous (Paulo Matias)

Introducción a los computadores

Tema 5 – La Máquina Sencilla

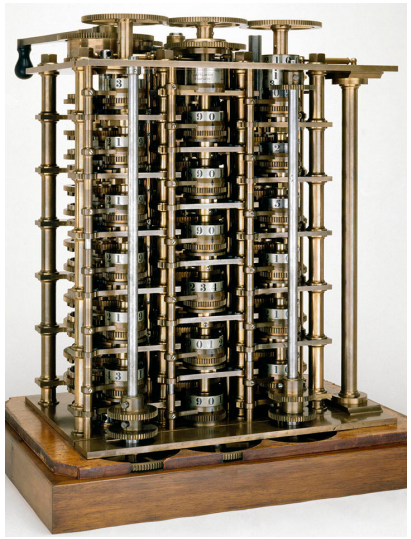
Juan Segarra y Alejandro Valero

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

- Primeros cálculos: manuales
  - E.g. Trigonometría en el antiguo Egipto y Babilonia (pre-antigua Grecia)
  - Radio terrestre calculado sobre el año 1000 ($\ll$1492)
  - Cálculos complejos: Tablas matemáticas + interpolación (e.g. logaritmos)
- Problema: errores
  - Necesidad de automatizar
  - E.g. Pascalina (B. Pascal, 1642), Máquina diferencial (C. Babbage, 1822)
  - Una máquina por algoritmo



Parte de máquina diferencial

1 Introducción (II)

- Siguiente paso: considerar el algoritmo como una entrada
  - Automatiza la resolución de cualquier algoritmo
  - Primer diseño (Máquina analítica) y primeros programas: C. Babbage, 1837
  - Válvulas de vacío (1904-8)
  - Teoría de computación: Máquina de Turing (1936)
  - Primeros computadores (electrónicos): 1940-1945

Parte de máquina analítica

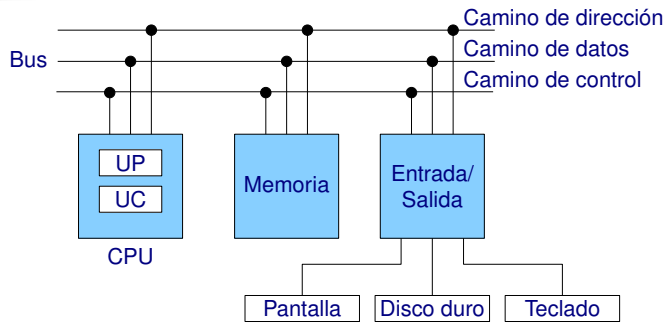
1 Introducción (III)

Computador: *Sistema secuencial* capaz de ejecutar un conjunto de *instrucciones* ubicadas en *memoria*

Partes de un computador:

- Unidad central de proceso (CPU) o procesador
  - Unidad de proceso (UP, ruta de datos, *datapath*)
  - Unidad de control (UC)
- Memoria
- Dispositivos de entrada/salida
- Bus (caminos de dirección, datos y control)

1.1 Estructura



- Programa: Secuencia de instrucciones que, aplicada a unos datos, resuelve un problema
- Las instrucciones y datos se almacenan en memoria (arquitectura von Neumann)
- La información en la memoria se referencia mediante una numeración consecutiva (*dirección de memoria*)

1.2 Repertorio de instrucciones

Repertorio de instrucciones: Conjunto de operaciones que un computador es capaz de realizar (e.g. x86: Intel 32 bits)

Repertorio de instrucciones de la Máquina Sencilla

Mnemónico	Descripción
ADD F, D	Suma el contenido de la dirección de memoria F con el contenido de la dirección de memoria D y lo guarda en la dirección de memoria D. Guarda en FZ si el resultado de la suma ha sido 0 o no
MOV F, D	Mueve el contenido de la dirección de memoria F (fuente) a la dirección de memoria D (destino). Guarda en FZ si el valor movido ha sido 0 o no
CMP F, D	Guarda en FZ si el contenido de la dirección de memoria F es igual al contenido de la dirección D
BEQ D	Si FZ es verdadero, la próxima instrucción a ejecutar será la almacenada en la dirección de memoria D

1.3 Compilación

- Un *compilador* traduce un programa de *alto nivel* al *repertorio de instrucciones* de la máquina destino

Programa de alto nivel

```
begin
c ← 0
i ← 0
while i ≠ b do
  c ← c + a
  i ← i + 1
end while
end
```

⇒

Programa en ensamblador (compilado)

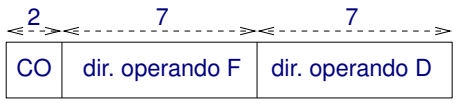
```
begin: MOV cero, c      ; c ← cero
       MOV cero, i      ; i ← cero
while: CMP i, b         ; ¿i = b?
       BEQ end          ; si cierto, ir a end
       ADD a, c         ; c ← c + a
       ADD uno, i       ; i ← i + uno
       CMP cero, cero   ; ¿cero = cero?
       BEQ while        ; si cierto, ir a while
end:
```

1.4 Formato de instrucción

- El formato de instrucción define la representación de las instrucciones mediante código máquina (unos y ceros)
- Las instrucciones constan del identificador de operación (*código de operación*) junto con referencias a los operandos necesarios
- Los operandos, de 16 bits, están en memoria
- La instrucción debe contener las direcciones de memoria, de 7 bits, donde están los operandos (¡¡¡no los propios operandos!!!)

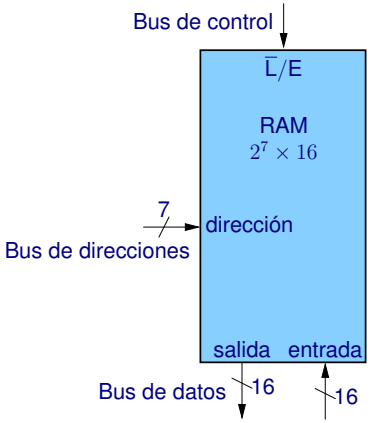
Operación	CO1	CO0
ADD	0	0
CMP	0	1
MOV	1	0
BEQ	1	1

Formato de instrucción



1.5 Memoria

- Memoria de 128 palabras de 16 bits
- Instrucciones y datos en memoria



Ejemplo de ubicación

begin:	→	@0
while:	→	@2
end:	→	@8
a	→	@100
b	→	@101
c	→	@102
i	→	@103
uno	→	@104
cero	→	@105

1.6 Programa en memoria

Programa en ensamblador

```
begin:  MOV cero, c      ; c ← cero
        MOV cero, i      ; i ← cero
while:  CMP i, b         ; ¿i = b?
        BEQ end          ; si cierto, ir a end
        ADD a, c         ; c ← c + a
        ADD uno, i       ; i ← i + 1
        CMP cero, cero   ; ¿cero = cero?
        BEQ while        ; si cierto, ir a while
end:
```

Ejemplo: MOV cero, c

10 (2)	1101001 (105)	1100110 (102)
CO1-CO0	dir. de cero	dir. de c

Programa en memoria

@0	2	105	102
@1	2	105	103
@2	1	103	101
@3	3	X	8
@4	0	100	102
@5	0	104	103
@6	1	105	105
@7	3	X	2

...

@100	? (a)
@101	? (b)
@102	? (c)
@103	? (i)
@104	1 (uno)
@105	0 (cero)

2 Unidad de Proceso

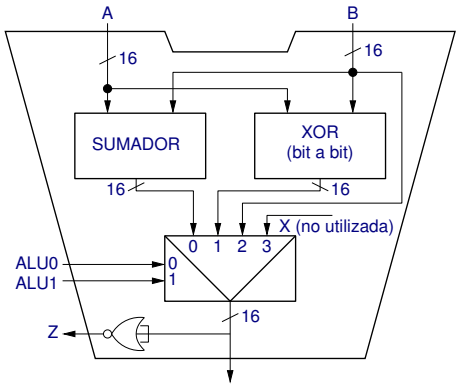
Componentes:

- Unidad aritmetico-lógica (ALU)
- Registros
  - Registros para operandos (A, B) de 16 bits
  - Registro de instrucción (IR) de 16 bits
  - Contador de programa (PC) de 7 bits
  - Indicador (*flag*) de cero (FZ) de 1 bit
- Cableado de direccionamiento a memoria

2.1 Unidad aritmetico-lógica

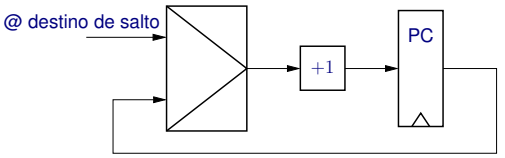
- La ALU debe generar los resultados necesarios para cada una de las instrucciones del repertorio: resultado de la operación e indicador de cero

ALU_{10}	Operación
0 0	A+B
0 1	A⊕B (bit a bit)
1 0	B
1 1	X



2.2 Registros

- Los registros A y B guardan los operandos Destino y Fuente
- El registro de instrucción (IR) guarda la instrucción en curso (con su correspondiente *formato*)
- El contador de programa (PC) contiene la dirección de la siguiente instrucción a ejecutar (bien la situada justo detrás de la actual o bien la de detrás del destino de un salto)



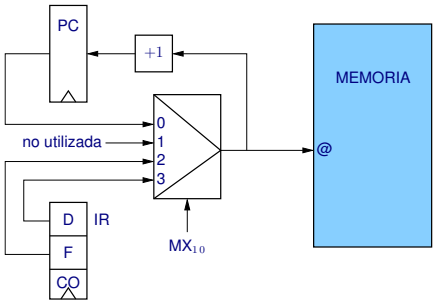
- El indicador de cero (FZ) guarda si el resultado de la última operación ha sido cero

2.3 Direccionamiento de la memoria

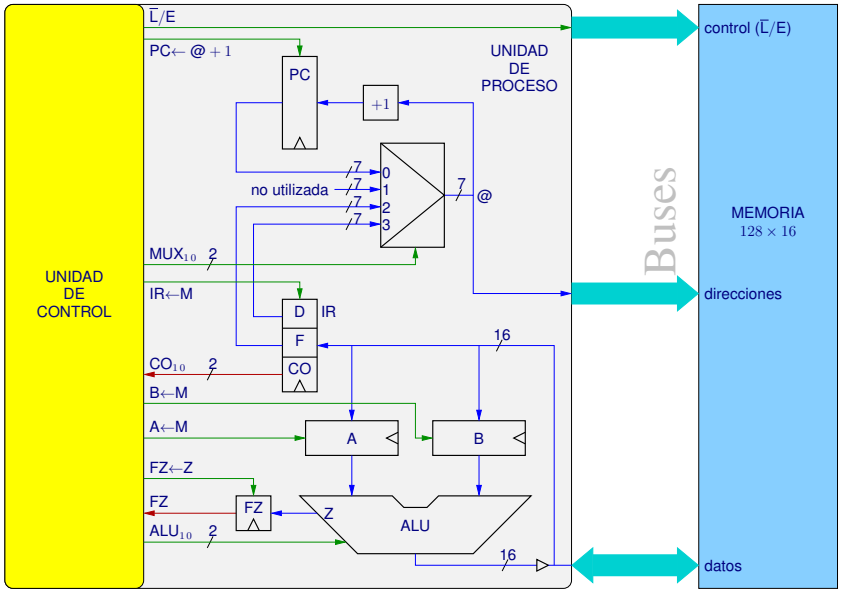
Necesitamos acceder a memoria para:

- leer el operando fuente (su dirección está en IR, en F)
- leer el operando destino (su dirección está en IR, en D)
- escribir el resultado en destino (su dir. está en IR, en D)
- leer la siguiente instrucción (dir. en PC o en IR, en D)

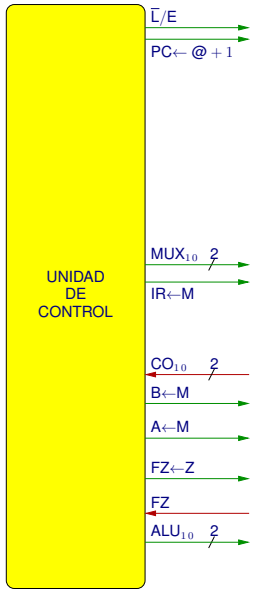
MX_1	MX_0	dirección
0	0	PC
0	1	no utilizada
1	0	F
1	1	D



2.4 UP de la Máquina Sencilla



3 Unidad de Control



La UC debe activar las señales hacia la UP en la secuencia apropiada para realizar cada instrucción

Fases de ejecución de una instrucción:

1. (*Fetch*) *Búsqueda* en memoria *de la instrucción* a ejecutar y almacenamiento de la misma en IR; Incremento de PC
2. *Decodificación* de la instrucción
3. *Búsqueda de operandos* en memoria (ADD, MOV, CMP) o evaluación de FZ (BEQ)
4. *Ejecución* de la instrucción (ALU) y *almacenamiento del resultado* en memoria

3.1 Fases de ejecución

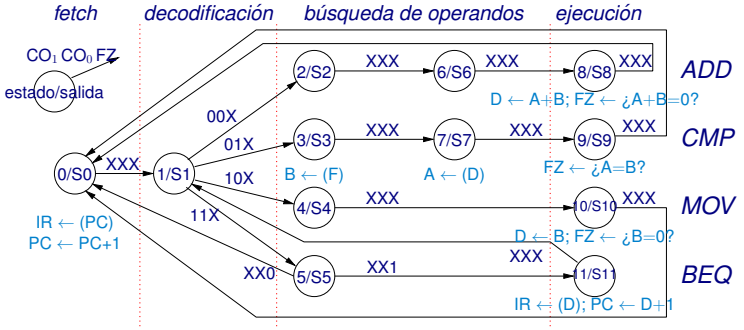
Notación: acciones RTL (*Register Transfer Level*)

Z←X: X: qué hago, Z: dónde guardo (registro/dir. mem.)

(X): Paréntesis: contenido de la dirección X de memoria

Fase	Instrucción	Detalle
1	ADD CMP MOV BEQ	IR ← (PC); PC ← PC+1
2	ADD CMP MOV BEQ	evaluación CO_{10}
3	ADD CMP MOV	B ← (F)
3bis	ADD CMP	A ← (D)
3	BEQ	evaluación FZ
4	ADD	D ← A+B; FZ ← ¿A+B=0?
4	CMP	FZ ← ¿A=B?
4	MOV	D ← B; FZ ← ¿B=0?
4 (1)	BEQ	Si FZ=1: IR ← (D); PC ← D+1

3.2 Diagrama de estados



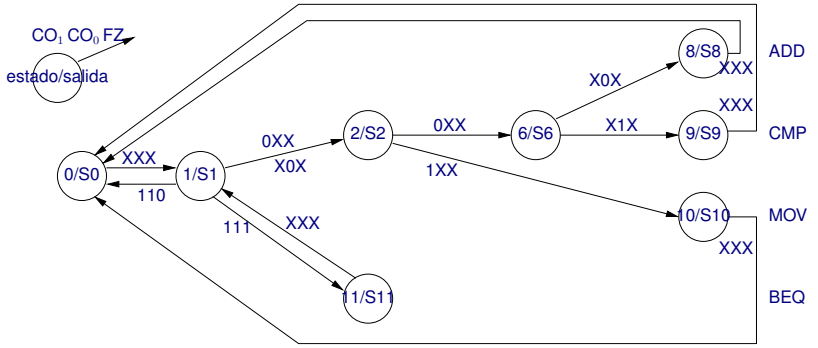
Salida	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
MX_{10}	00	XX	10	10	10	XX	11	11	11	XX	11	11
ALU_{10}	XX	XX	XX	XX	XX	XX	XX	XX	00	01	10	XX
L/E	0	0	0	0	0	0	0	0	1	0	1	0
PC ← @ +1	1	0	0	0	0	0	0	0	0	0	0	1
IR ← M	1	0	0	0	0	0	0	0	0	0	0	1
A ← M	0	0	0	0	0	0	1	1	0	0	0	0
B ← M	0	0	1	1	1	0	0	0	0	0	0	0
FZ ← Z	0	0	0	0	0	0	0	0	1	1	1	0

3.2 Diagrama de estados (II)

Optimizaciones:

1. Búsqueda del 1º operando: igual para ADD, MOV y CMP
2. Búsqueda del 2º operando: igual para ADD y CMP
3. La consulta de FZ se puede realizar en el estado 1

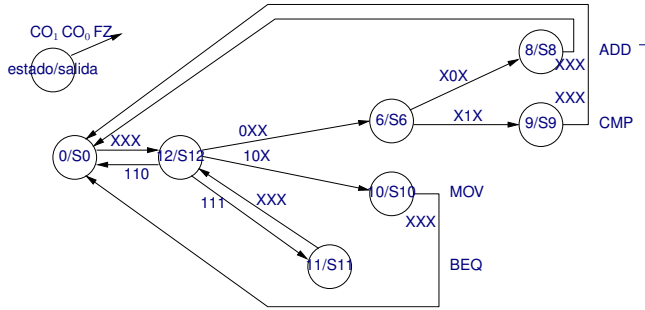
La fase de decodificación queda distribuida en varios estados



3.2 Diagrama de estados (III)

Optimizaciones:

4. La primera fase de decodificación se unifica con la búsqueda del primer operando

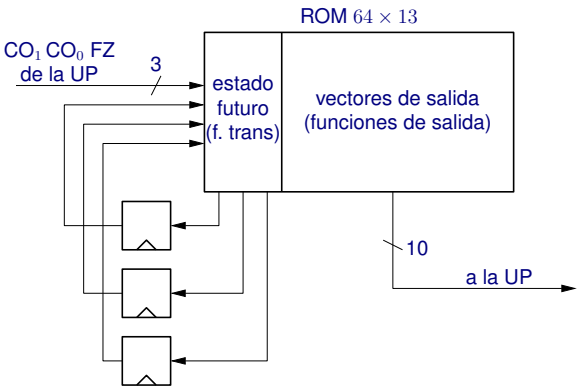


Salidas	S12
MX_{10}	10
ALU_{10}	XX
$\overline{L}/E$	0
PC $\leftarrow$ @ $+1$	0
IR $\leftarrow$ M	0
A $\leftarrow$ M	0
B $\leftarrow$ M	1
FZ $\leftarrow$ Z	0

- ¿Cuántos ciclos se ahorran en cada optimización?
- ¿Cómo afecta al consumo energético?

3.3 Diseño en ROM

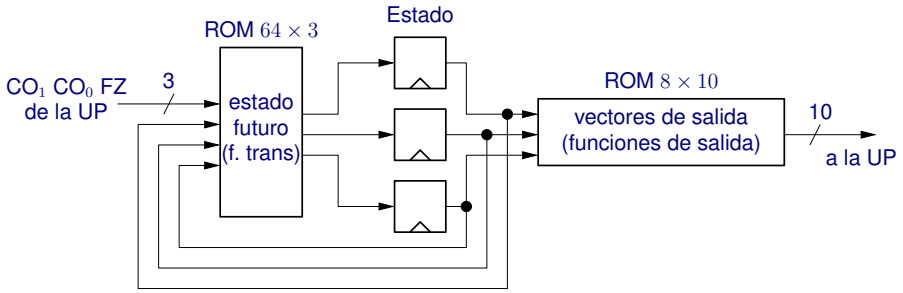
- En lugar de diseñar con puertas podemos usar una ROM



- Sistema Moore $\rightarrow$ las salidas dependen sólo del estado
  - 8 estados $\rightarrow$ 8 salidas (de las 64 en ROM)

3.3 Diseño en ROM (II)

- Diseño optimizado con dos ROMs
  ($8 \times 10 + 64 \times 3 \ll 64 \times 13$)

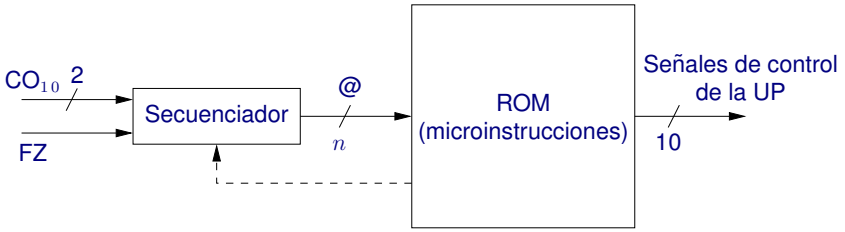


4 Microprogramación

- El diseño y la modificación de una UC *cableada* (hecha con puertas lógicas) es costoso
- La implementación en ROM facilita los cambios, pero el proceso de diseño es igual que para la cableada
- La *microprogramación* es una alternativa de diseño que realiza cada instrucción mediante una secuencia de *microinstrucciones* (microprograma)
  - Facilita el diseño: cada microinstrucción es independiente
  - El coste de implementación es similar al de la UC basada en ROM
  - Puede limitar la velocidad de ejecución: cada microinstrucción requerirá al menos un ciclo para ejecutarse

4.1 Secuenciador

- Si cada instrucción es un microprograma necesitamos un mecanismo para ejecutar las microinstrucciones en la secuencia correcta
  - En la UP el contador de programa (PC) indica la instrucción que toca ejecutar
  - En la UC microprogramada usaremos un *secuenciador* que nos indicará la microinstrucción que toca ejecutar

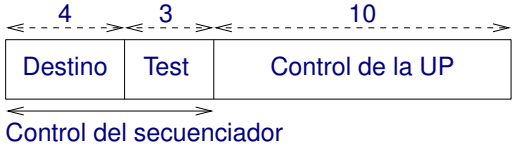


4.2 Esquema de microprograma

Microinstrucciones	
@=PC, PC=@+1, $\overline{L}$/E=0, IR=M	;fetch
$\overline{L}$/E=0	;decodificación
@=F, $\overline{L}$/E=0, B=M	;ADD
@=D, $\overline{L}$/E=0, A=M	
@=D, ALU= +, $\overline{L}$/E=1, FZ=Z	
@=F, $\overline{L}$/E=0, B=M	;CMP
@=D, $\overline{L}$/E=0, A=M	
@=X, ALU= $\oplus$, $\overline{L}$/E=0, FZ=Z	
@=F, $\overline{L}$/E=0, B=M	;MOV
@=D, ALU=B, $\overline{L}$/E=1, FZ=Z	
@=D, PC=@+1, $\overline{L}$/E=0, IR=M	;BEQ

4.3 Formato de microinstrucción

- Cada microinstrucción debe contener las señales de control de la UP
- Hay que especificar la secuencia de microinstr. de cada instrucción
- Habrá muchos saltos → en vez de usar una microinstr. de salto, hacer que todas las microinstr. puedan saltar
- Hay que permitir saltar siempre, nunca o en función de las señales de entrada (campo *Test*)



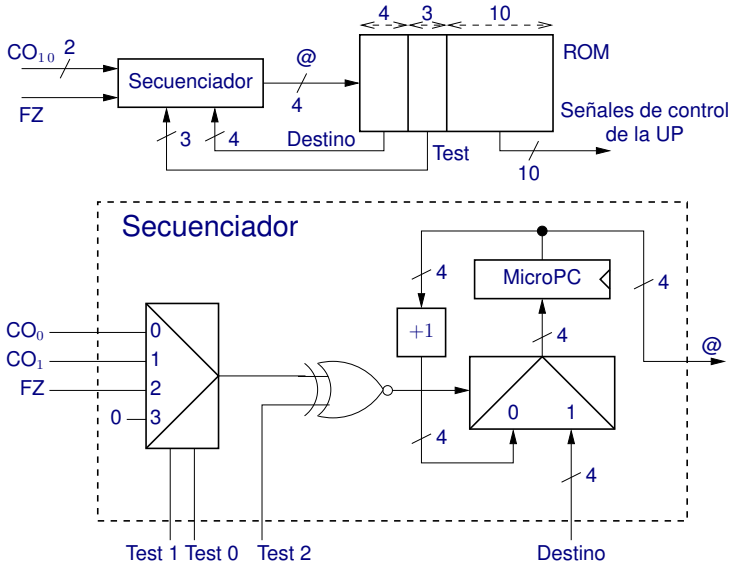
$Test_1$	$Test_0$	Señal
0	0	CO_0
0	1	CO_1
1	0	FZ
1	1	0

- Saltar a *Destino* si $Test_2$ = Señal($Test_1$ $Test_0$)
- E.g. 100 → ¿1=CO_0?, 001 → ¿0=CO_1?, 111 → ¿1=0?

4.4 Microprograma en micromemoria

	Dest.	Test	MX	ALU	$\overline{L}$/E	PC	IR	A	B	FZ	
0	XXXX	111	00	XX	0	1	1	0	0	0	*fetch*
1	1000	100	XX	XX	0	0	0	0	0	0	*decod*: si CO_0 =1 ir a 8
2	0110	101	XX	XX	0	0	0	0	0	0	*decod*: si CO_1 =1 ir a 6
3	XXXX	111	10	XX	0	0	0	0	1	0	*ADD* (CO=00): búsq. op. F
4	XXXX	111	11	XX	0	0	0	1	0	0	búsqueda op. D
5	0000	011	11	00	1	0	0	0	0	1	suma; flag Z; ir a 0
6	XXXX	111	10	XX	0	0	0	0	1	0	*MOV* (CO=10): búsq. op. F
7	0000	011	11	10	1	0	0	0	0	1	mueve; flag Z; ir a 0
8	1100	101	XX	XX	0	0	0	0	0	0	*decod*: si CO_1 =1 ir a 12
9	XXXX	111	10	XX	0	0	0	0	1	0	*CMP* (CO=01): búsq. op. F
10	XXXX	111	11	XX	0	0	0	1	0	0	búsqueda op. D
11	0000	011	XX	01	0	0	0	0	0	1	compara; flag Z; ir a 0
12	0000	010	XX	XX	0	0	0	0	0	0	*BEQ* (CO=11): si FZ=0 ir a 0
13	0001	011	11	XX	0	1	1	0	0	0	ir a 1; (*fetch* destino de salto)

4.5 Secuenciador detallado



5 Modificaciones

Deseamos realizar una operación que multiplique por 4 un operando:

CUAD F, D D $\leftarrow 4 \times$ (F)
FZ $\leftarrow$ ¿(F) $= 0$?

- Modificación por *software*: No modificamos la máquina sencilla, sino que realizamos la operación CUAD usando las instrucciones disponibles

  CUAD A,B $\equiv$ MOV A,B ; ADD B,B ; ADD B,B

  E.g: "Ejecutar" (emular) en un PC (x86-64/amd64) un código de CDROM (ROM) de PlayStation (MIPS R3000)
- Modificación por *hardware*: Modificamos tanto la UP como la UC. La ALU deber hacer la operación de multiplicar por 4 (e.g. ALU_{10} =1 1). La UC debe tener la sucesión de estados que lleven a cabo la instrucción

5 Modificaciones: CUAD F, D

- Modificación por *firmware*: Modificamos sólo la UC. Sólo necesitamos sustituir el bloque ROM (microprogramado o no) por otro que contenga la sucesión de estados para llevar a cabo CUAD *usando la UP original*
  - Si en vez de en ROM está en mem. flash sólo hay que actualizar su contenido (e.g. *flashear* el móvil, cámara de fotos, *router*, TV, etc.) para cambiar su funcionamiento

Valoración:

- Por *software* es más barato (no hacemos cambios de hardware) pero más lento en tiempo de ejecución de la nueva instrucción
- Por *hardware* es más caro pero más rápido
- Por *firmware* tiene coste y velocidad intermedios

5.1 Modif.: CLEAR, MOVD, ACUM

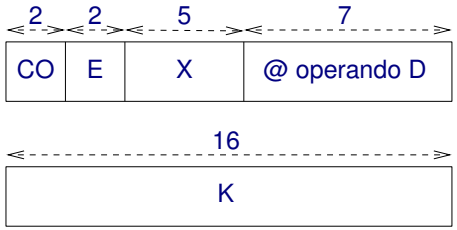
Deseamos *añadir* las siguientes instrucciones a la máquina sencilla:

Mnemónico	Detalle	Descripción
CLEAR D	D $\leftarrow 0$	Puesta a 0 sin alterar FZ
MOVD K, D	D $\leftarrow$ K FZ $\leftarrow$ ¿K $= 0$?	Mueve la constante K a D
ACUM K, D	D $\leftarrow$ (D) $+$ K FZ $\leftarrow$ ¿(D) $+$ K $= 0$?	Acumula la constante K en D

- Como tenemos más instrucciones necesitamos más bits de código de operación
  - Debemos cambiar el formato de instrucción (lo mínimo posible: usando bits no usados hasta ahora)

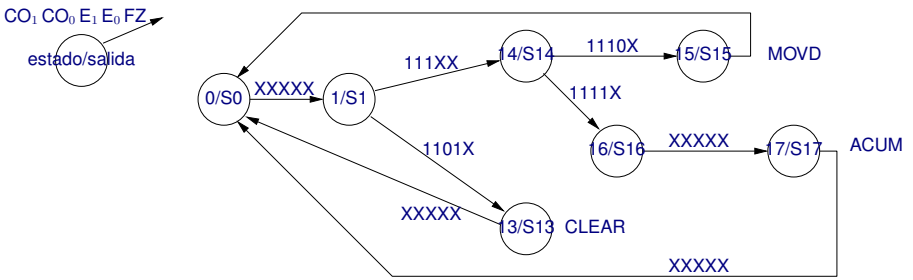
5.1 Modif.: CLEAR, MOVD, ACUM (II)

- Podemos incluir los nuevos códigos de operación (E) en los bits no usados de la instrucción BEQ
- Debemos incluir la constante K (16 bits) en la própia instrucción para las instrucciones MOVD y ACUM
  - Necesitamos una palabra adicional para codificar K



Instr.	CO_1	CO_0	E_1	E_0
ADD	0	0		
CMP	0	1		
MOV	1	0		
BEQ	1	1	0	0
CLEAR	1	1	0	1
MOVD	1	1	1	0
ACUM	1	1	1	1

5.1 Modif.: CLEAR, MOVD, ACUM (III)



Salidas	S13	S14	S15	S16	S17
MX_{10}	11	00	11	11	11
ALU_{10}	11	XX	10	XX	00
$\overline{L}$/E	1	0	1	0	1
PC $\leftarrow$ @ $+1$	0	1	0	0	0
IR $\leftarrow$ M	0	0	0	0	0
A $\leftarrow$ M	0	0	0	1	0
B $\leftarrow$ M	0	1	0	0	0
FZ $\leftarrow$ Z	0	0	1	0	1

Créditos de material reutilizado

Imagen «Máquina diferencial» (p. 3): CC BY NC SA Science Museum London (bordes eliminados)

Imagen «Máquina analítica» (p. 4): CC BY Marcin Wichary