Sequential Logic Design

Arquitectura de Computadores

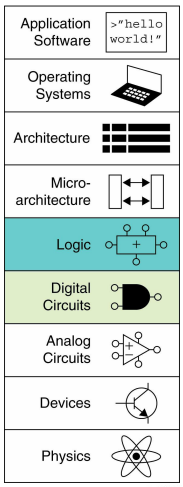
Bloques Digitales Avanzados

Basado en texto: "***Digital Design and Computer Architecture*, 2nd Edition**", David Money Harris and Sarah L. Harris

Sequential Logic Design

Temas

- **Introducción**
- **Latches y Flip-Flops**
- **Diseño Lógico Síncrono**
- **Sumadores**
- **Unidad Logica Aritemetica**
- **Punto Flotante**



Introducción

- La salida de una lógica secuencial depende de la entrada actual y pasada – Tiene ***memoria***.
- Algunas definiciones:
  - **Estado:** Toda la información de un circuito que es necesaria para explicar su comportamiento futuro
  - **Latches y flip-flops:** elementos de estado que almacenan un bit de estado
  - **Circuito secuencial síncrono: lógica** combinacional seguida por un banco de flip-flops

Circuitos Secuenciales

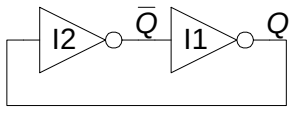
- Dan secuencia a los eventos
- Tiene memoria (de corto plazo)
- Use realimentacion de la salida hacia las entradas para almacenar información

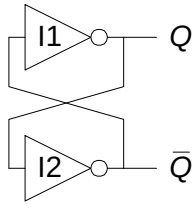
Elementos de Estado

- El estado de un circuito influencia su comportamiento futuro
- Los elementos de estado guardan estado
  - Circuito Biestable
  - Latch SR
  - Latch D
  - Flip-Flop D

Circuito Biestable

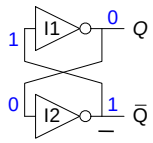
- Bloque fundamental de construcción de los elementos de estado
- Dos salidas: $\overline{Q}$, Q
- No tiene entradas



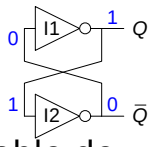


Análisis Circuito Biestable

- Considere los dos posibles casos:
  - ***Q* = 0:**
    luego $\bar{Q} = 1$, $Q = 0$ (consistente)



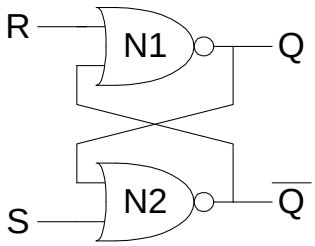
  - ***Q* = 1:**
    luego $\bar{Q} = 0$, $Q = 1$ (consistente)



- Almacena 1 bit de estado en la variable de estado, Q (o $\bar{Q}$)
- Pero **no tiene entradas para controlar el estado**

Latch SR (Set/Reset)

- Latch SR

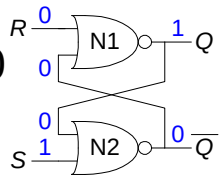


- Considere los cuatro posibles casos:
  - ***S* = 1, *R* = 0**
  - ***S* = 0, *R* = 1**
  - ***S* = 0, *R* = 0**
  - ***S* = 1, *R* = 1**

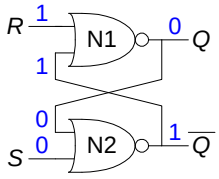
Sequential Logic Design

Análisis Latch SR

- ***S* = 1, *R* = 0:**
  luego $Q = 1$ y $\bar{Q} = 0$



- ***S* = 0, *R* = 1:**
  luego $Q = 1$ y $\bar{Q} = 0$

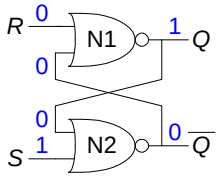


ELSEVIER

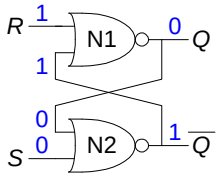
Sequential Logic Design

Análisis Latch SR

- ***S* = 1, *R* = 0:**
  luego $Q = 1$ y $\bar{Q} = 0$
  ***Set* la salida**



- ***S* = 0, *R* = 1:**
  luego $Q = 1$ y $\bar{Q} = 0$
  ***Reset* la salida**

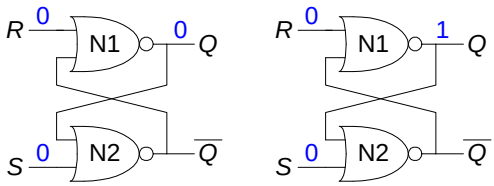


ELSEVIER

Análisis Latch SR

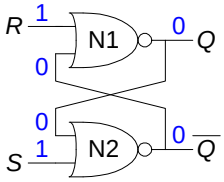
- **$S = 0, R = 0$:** $Q_{prev} = 0$ $Q_{prev} = 1$

  luego $Q = Q_{prev}$



- **$S = 1, R = 1$:**

  luego $Q = 0, \bar{Q} = 0$



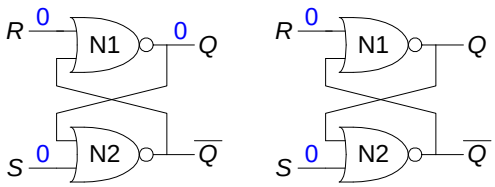
Sequential Logic Design

Análisis Latch SR

- **$S = 0, R = 0$:** $Q_{prev} = 0$ $Q_{prev} = 1$

  luego $Q = Q_{prev}$

  ¡Memoria!

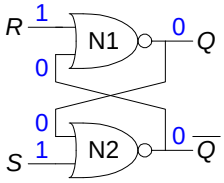


- **$S = 1, R = 1$:**

  luego $Q = 0, \bar{Q} = 0$

  Estado Invalido

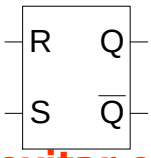
  $\bar{Q} \neq$ NOT Q



Símbolos del Latch SR

- SR significa Latch Set/Reset
  - Almacena un bit de estado (Q)
- Controla que valor se guarda con las entradas S, R
  - **Set:** Hace que la salida sea 1 ($S = 1$, $R = 0$, $Q = \mathbf{1}$)
  - **Reset:** Hace que la salida sea 0 ($S = 0$, $R = 1$, $Q = \mathbf{0}$)

SR Latch Symbol

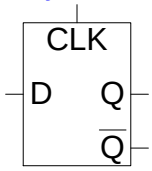


- **Debemos hacer algo para evitar el estado invalido (when $S = R = 1$)**

Latch D

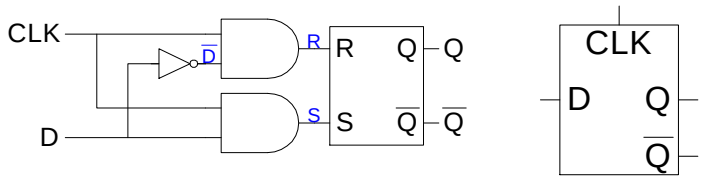
- Dos entradas: *CLK*, *D*
  - ***CLK*:** controla *cuando* la salida cambia
  - ***D*** (el dato de entrada): controla *a que* valor la salida cambia
- Función
  - Cuando ***CLK* = 1**, *D* pasa a *Q* (*transparente*)
  - Cuando ***CLK* = 0**, *Q* mantiene el valor anterior (*opaco*)

D Latch Symbol



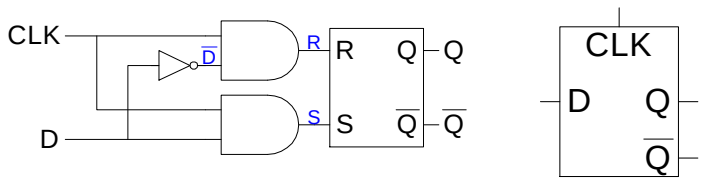
- Evita el caso invalido cuando $\overline{Q} \neq$ NOT Q

Circuito Interno Latch D



CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X					
1	0					
1	1					

Circuito Interno Latch D



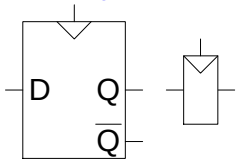
CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	X	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Sequential Logic Design

Flip-Flop D

- **Entradas:** *CLK*, *D*
- **Función**
  - Muestrea *D* en flanco de subida de *CLK*
    - Cuando *CLK* sube desde 0 a 1, *D* pasa a *Q*
    - En caso contrario, *Q* mantiene su valor previo
  - *Q* canbia solo en el flanco de subida de *CLK*
- Se le llama *gatillado por flanco*
- Activado en flanco del reloj

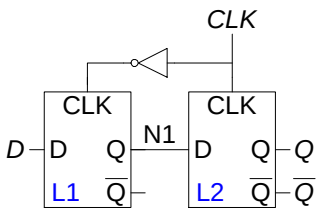
D Flip-Flop Symbols



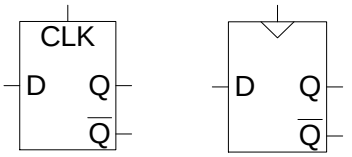
Sequential Logic Design

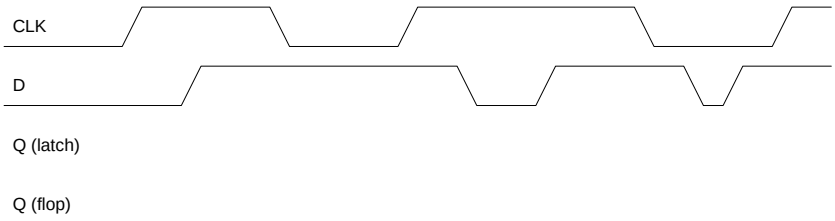
Circuito Interno del Flip-Flop D

- Dos Latches "back-to-back" (L1 y L2) controlado por relojes complementarios
- Cuando ***CLK* = 0**
  - L1 es transparente
  - L2 es opaco
  - *D* pasa a N1
- Cuando ***CLK* = 1**
  - L2 es transparente
  - L1 es opaco
  - N1 pasa a *Q*
- Luego, en el flanco del reloj (cuando ***CLK* sube desde 0 a 1**)
  - *D* pasa a *Q*

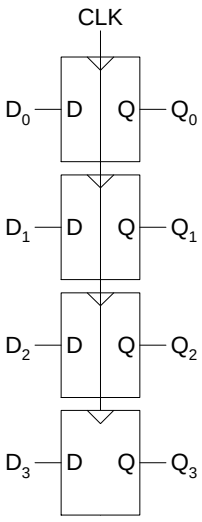


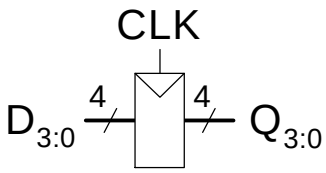
Latch D vs. Flip-Flop D





Uso del Flip-Flop: Registros





¿Como se puede evitar que el valor guardado se cambie en cada ciclo de reloj?

Uso de Flip-Flop

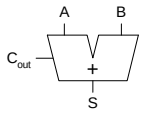
- Contadores
- Cualquier aplicación que requiera "recordar" el pasado
  - Permite implementar Maquinas de Estado Finitas (otro nombre para un **Autómata Finito Determinístico**)
  - Por motivos de tiempo no cubriremos el diseño de circuitos secuenciales y nos centraremos en componentes de logica combinacional:
    - Ejemplo: Sumador

Revisión

- **Bloques de Construcción Digital:**
  - Compuertas, multiplexores, decodificadores, registros, circuitos aritméticos, contadores, arreglos de memoria, Arreglos lógicos
- **La construcción de bloque demuestra jerarquía, modularidad, y regularidad:**
  - Jerarquia de componentes mas simples
  - Interfaces y funciones bien definidas
  - Una estructura regular fácilmente se extiende a tamaños diferentes
- **Usaremos estos bloques para construir en un próximo capitulo un microprocesador**

Sumador de 1 bit

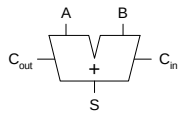
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$
$C_{out} = AB$

Full Adder



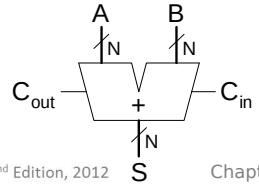
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = A \oplus B \oplus C_{in}$
$C_{out} = AB + AC_{in} + BC_{in}$

Sumadores Multibit (CPAs)

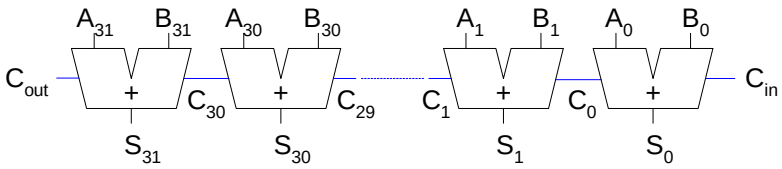
- Tipos de sumadores con propagación de acarreo (Carry Propagate Adders, CPAs):
  - **De acarreo en serie, Ripple-carry (lento)**
  - **De predicción de acarreo, Carry-lookahead (rápido)**
  - **De prefijo, prefix (muy rápido)**
- Sumadores de predicción de acarreo y de prefijo son mas rápidos para grandes sumadores pero requieren mas hardware

Símbolo



Ejemplo: Sumador de Acarreo en Serie

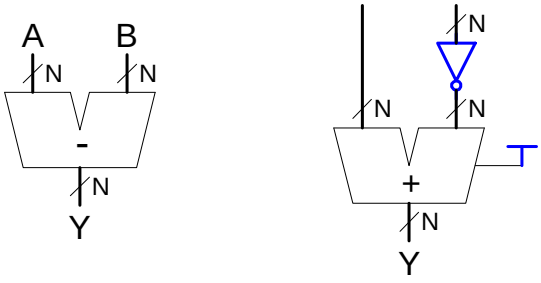
- Cadena de sumadores de 1-bit
- Lleva acarreo a través de toda la cadena
- Ventaja: Muy simple
- Desventaja: **lento**



Sequential Logic Design

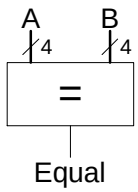
Restador

Symbol **Implementation**

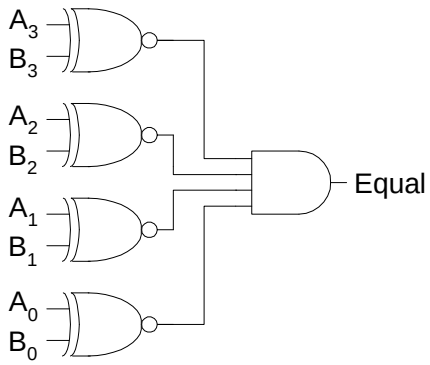


Comparador: Igualdad

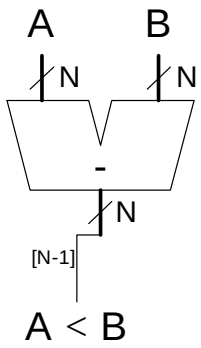
Symbol



Implementation

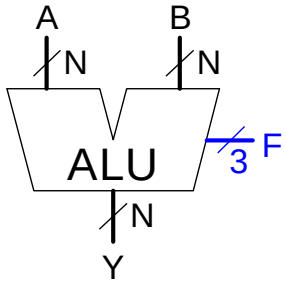


Comparador: Menor que



Sequential Logic Design

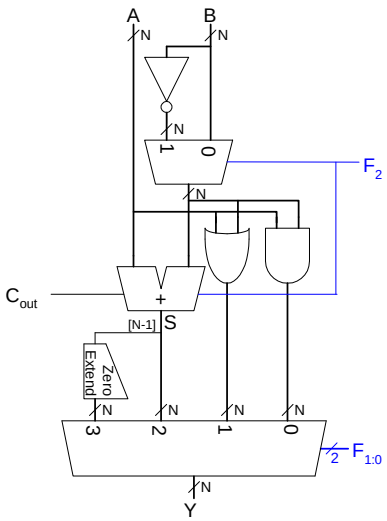
Unidad Logico Aritmética (ALU)



$F_{2:0}$	Función
000	A & B
001	A | B
010	A + B
011	no usado
100	A & ~B
101	A | ~B
110	A - B
111	SLT

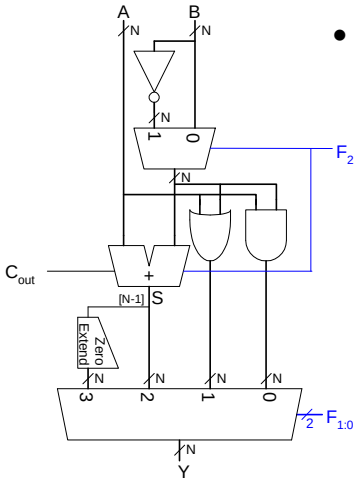
Sequential Logic Design

Diseño ALU



$F_{2:0}$	Función
000	A & B
001	A | B
010	A + B
011	no usado
100	A & ~B
101	A | ~B
110	A - B
111	SLT

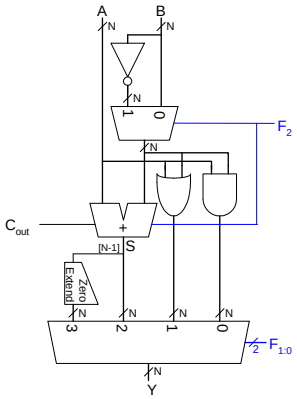
Ejemplo Menor que (Set Less Than - SLT)



- Configure ALU de 32-bit para operación SLT : A = 25 y B = 32

Sequential Logic Design

Ejemplo Menor que (Set Less Than - SLT)



- Configure ALU de 32-bit para operación SLT: A = 25 y B = 32
  - ***A* < *B***, so *Y* deberia ser una representación de 1 de 32-bit (0x00000001)
  - **$F_{2:0}$ = 111**
    - **F_2 = 1** (sumador actúa como restador), luego 25 - 32 = -7
    - -7 tiene 1 en el bit mas significativo (S_{31} = 1)
    - **$F_{1:0}$ = 11** multiplexor selecciona $Y = S_{31}$ (cero extendido) = 0x00000001.

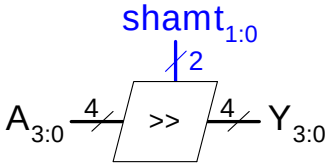
Desplazadores (Shifters)

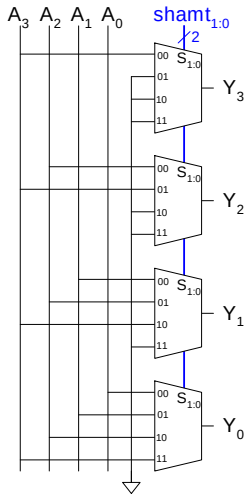
- **Desplazador lógico:** desplaza valor hacia la izquierda o hacia la derecha y llena los espacios vacíos con 0's
  - Ej: 11001 >> 2 =
  - Ej: 11001 << 2 =
- **Desplazador aritmético:** similar al desplazador lógico, pero en el desplazador derecho, llena el espacio vacío con el bit antiguo mas significativo (msb).
  - Ej: 11001 >>> 2 =
  - Ej: 11001 <<< 2 =
- **Rotador:** rota los bits en circulo, tal que los bits desplazados por un extremos son movidos al otro extremo
  - Ej: 11001 ROR 2 =
  - Ej: 11001 ROL 2 =

Desplazadores

- **Desplazador lógico:**
  - Ej: 11001 >> 2 = 00110
  - Ej: 11001 << 2 = 00100
- **Desplazador aritmético:**
  - Ej: 11001 >>> 2 = 11110
  - Ej: 11001 <<< 2 = 00100
- **Rotador:**
  - Ej: 11001 ROR 2 = 01110
  - Ej: 11001 ROL 2 = 00111

Diseño Desplazador





shamt: SHift AMounT

Desplazadores como Multiplicadores, Divisores

- ***A << N = A × 2^N***
  - **Ejemplo:** 00001 << 2 = 00100 ($1 \times 2^2 = 4$)
  - **Ejemplo:** 11101 << 2 = 10100 ($-3 \times 2^2 = -12$)
- ***A >>> N = A ÷ 2^N***
  - **Ejemplo:** 01000 >>> 2 = 00010 ($8 \div 2^2 = 2$)
  - **Ejemplo:** 10000 >>> 2 = 11100 ($-16 \div 2^2 = -4$)

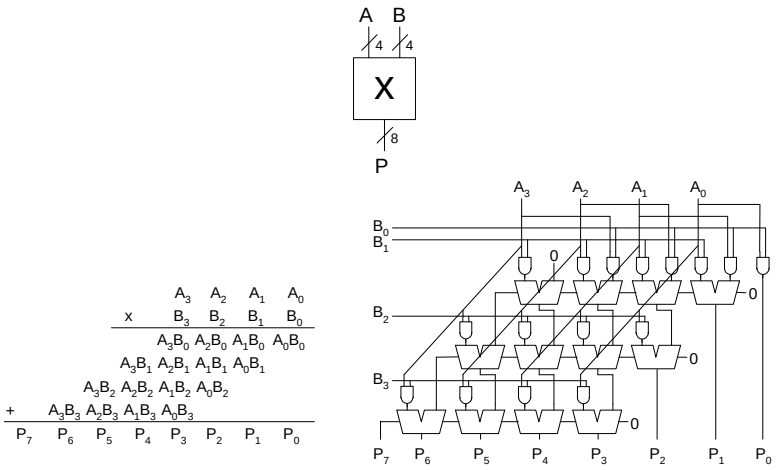
Multiplicador

- **Producto parcial** se forma al multiplicar un simple dígito del multiplicador con el multiplicando.
- **El desplazamiento** de productos parciales **se suman** para formar el resultado

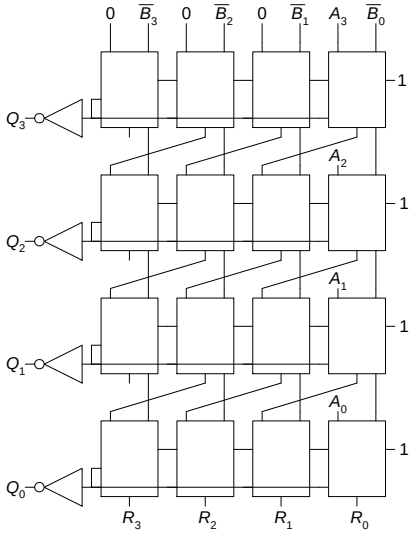
Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

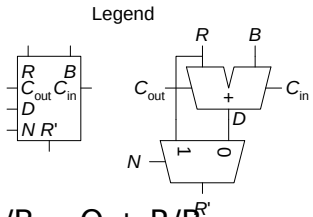
230 x 42 = 9660 5 x 7 = 35

Multiplicador de 4 x 4



Divisor de 4 x 4





A/B = Q + R/B

Algoritmo:

R' = 0
for *i* = N-1 to 0
  R = {R' << 1, A_i}
  D = R - B
  if D < 0, Q_i=0, R'=R
  else Q_i=1, R'=D
R'=R

Sequential Logic Design

Sistemas Numéricos

- Los números pueden ser representados usando representaciones binarias
  - **Números positivos**
    - Binarios sin signo
  - **Números negativos**
    - Complemento de dos
    - Números Signo/magnitud
- ¿Que hay de las **fracciones**?

Numeros con Fracciones

- Hay dos notaciones comunes:
  - **Punto Fijo:** punto del binario es fijo
  - **Punto flotante:** punto del binario flota a la derecha del 1 mas significativo

Números con Punto Fijo

- 6.75 usa 4 bits parte entera y 4 bits parte de fracción:

$$01101100$$

$$0110.1100$$

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$$

El punto binario es implícito

- El numero de bits de la parte entera y fracción deben ser acordadas de antemano

Ejemplo Número Punto Fijo

- Representa 7.5_{10} usando 4 bits parte entera y 4 bits de fracción

Ejemplo Numero Punto Fijo

- Representa 7.5_{10} usando 4 bits parte entera y 4 bits de fracción.

01111000

Numeros Punto Fijo con Signo

- **Representaciones:**
  - Signo/magnitud
  - Complemento de dos
- **Ejemplo:** Represente -7.5_{10} usando 4 bits para la parte entera 4 bits de fracción
  - **Signo/magnitud:**
  - **Complemento de dos:**

Numeros Punto Fijo con Signo

- **Representaciones:**
  - Signo/magnitud
  - Complemento de dos
- **Ejemplo:** Represente -7.5_{10} usando 4 bits de parte entera y 4 bits de fracción
  - **Signo/magnitud:**

    11111000
  - **Complemento de dos:**

    1. +7.5: 1111000
    2. Bits invertidos: 10000111
    3. Sume 1 al lsb: + 1

    10001000

Números Punto Flotante

- Punto binario flota a la derecha del 1 mas significativo
- Similar a la notación científica decimal
- Por ejemplo, escribe 273_{10} en notacion científica:

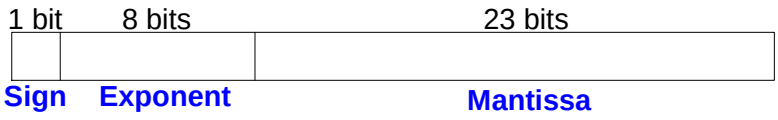
$$273 = 2.73 \times 10^2$$

- En general, un numero en notacion cientifica se escribe como:

$$\pm M \times B^E$$

  - **M** = mantisa
  - **B** = base
  - **E** = exponente
  - En el ejemplo, M = 2.73, B = 10, y E = 2

Números Punto Flotante



- **Ejemplo:** represente el valor 228_{10} usando una representacion en punto flotante de 32-bit

Veremos solo una breve descripcion del estándar **punto flotante IEEE 754**

Sequential Logic Design

Representacion del Punto Flotante 1

1. Convierte decimal a binario (**no invierta pasos 1 & 2!**):

   $228_{10} = 11100100_2$

1. Escriba el numero en “notacion cientifica binaria”:

   $11100100_2 = 1.11001_2 \times 2^7$

1. Complete cada campo del numero punto flotante de 32-bit:
   - El bit de signo positivo (0)
   - El bit 8 del exponente representa el valor 7
   - Los restantes 23 bits son la mantisa

1 bit	8 bits	23 bits
0	00000111	11 1001 0000 0000 0000 0000
Sign	**Exponent**	**Mantissa**

Sequential Logic Design

Representación del Punto Flotante 2

- El primer bit de la mantisa es siempre 1:
  - $228_{10} = 11100100_2 = \mathbf{1}.\mathbf{11001} \times 2^7$
- Luego, no hay necesidad de almacenarlo: *el 1 principal implícito*
- Almacene solo los bits de fraccion en el campo de 23-bit

1 bit	8 bits	23 bits
0	00000111	110 0100 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

Representación del Punto Flotante 3

- *Exponente sesgado*: *sesgo* (bias) = 127 (01111111_2)
  - Exponente sesgado = sesgo + exponente
  - Exponente de 7 es almacenado como:

    $127 + 7 = 134 = 0x10000110_2$
- La representación **punto flotante de 32 bits IEEE 754 de** 228_{10}

1 bit	8 bits	23 bits
0	10000110	110 0100 0000 0000 0000 0000
Sign	**Biased Exponent**	**Fraction**

in hexadecimal: **0x43640000**

Ejemplo Punto Flotante

Escriba -58.25_{10} en punto flotante (IEEE 754)

Ejemplo Punto Flotante

Escriba -58.25_{10} en punto flotante (IEEE 754)

1. Convierta decimal a binario:

   58.25_{10} = **111010.01_2**

1. Escriba en notacion cientifica binaria:

   1.1101001×2^5

3. Complete en los campos:
   Bit signo: **1** (negativo)
   8 bits exponente: (127 + 5) = 132 = **10000100_2**
   23 bits fracción: **110 1001 0000 0000 0000 0000**

1 bit	8 bits	23 bits
1	100 0010 0	110 1001 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

en hexadecimal: **0xC2690000**

Punto Flotante: Casos Especiales

Numero	Signo	Exponente	Fracción
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
- ∞	1	11111111	00000000000000000000000
NaN	X	11111111	no-cero

Precision Punto Flotante

- **Precision Simple:**
  - 32-bit
  - 1 bit signo, 8 bits exponente, 23 bits de fracción
  - bias = 127
- **Doble Precisión:**
  - 64-bit
  - 1 bit signo, 11 bits de exponente, 52 bits de fracción
  - bias = 1023

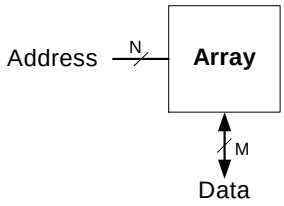
Punto Flotante: Redondeo

- **Overflow:** número demasiado grande para ser representado
- **Underflow:** número demasiado pequeño para ser representado
- **Modos de Redondeo:**
  - Hacia arriba
  - Hacia abajo
  - Hacia cero
  - Al mas cercano
- **Ejemplo:** redondeo 1.100101 (1.578125) a solo 3 bits de fracción
  - Hacia abajo: 1.100
  - Hacia arriba: 1.101
  - Hacia cero: 1.100
  - Al mas cercano: 1.101 (1.625 es mas cercano a 1.578125 que a 1.5)

Sequential Logic Design

Arreglos de Memoria

- Almacenan eficientemente grandes cantidades de datos
- Los 3 tipos mas comunes son:
  - Memoria de Acceso Aleatoria Dinamica (DRAM)
  - Memoria de Acceso Aleatorio Estático (SRAM)
  - Memoria de solo lectura (ROM)
- *Un dato de M*-bit puede ser leído/ escrito en cada direccion unica de N-bit

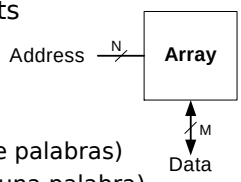


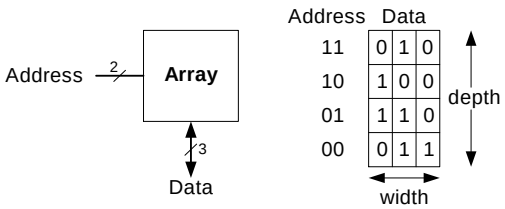
ELSEVIER

Sequential Logic Design

Arreglos de Memoria

- Arreglos bidimendionales de celdas de bits
- Cada celda de bit almacena un bit
- *N bits de direcciones* y *M* bits de datos:
  - 2^N filas y *M* columnas
  - **Profundidad:** numero de filas (numero de palabras)
  - **Ancho:** numero de columnas (tamaño de una palabra)
  - **Tamaño del arreglo:** profundidad × ancho = 2^N × *M*

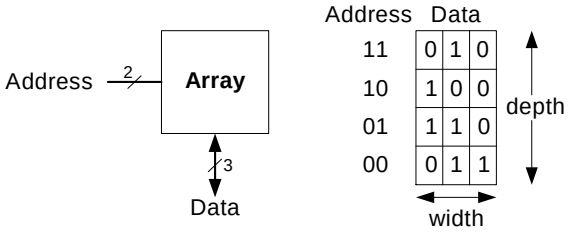




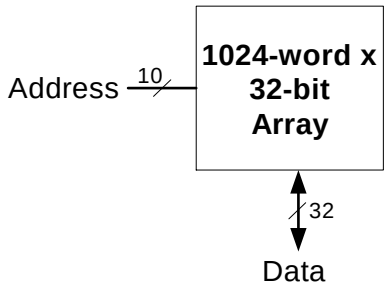
ELSEVIER

Ejemplo de Arreglo de Memoria

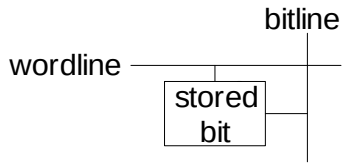
- Arreglo de 2^2 × 3-bit
- Numero de palabras: 4
- Tamaño palabra: 3-bits
- Por ejemplo, la palabra de 3-bit almacenada en la dirección 10 es 100

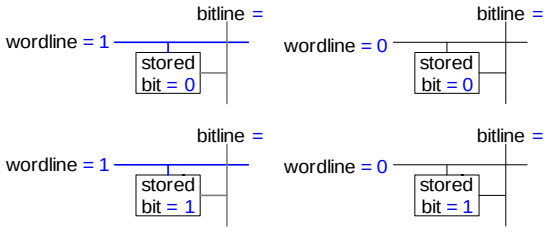


Arreglos de Memoria



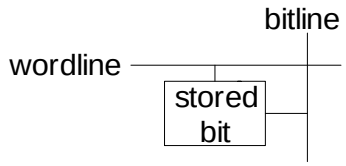
Celdas de bit de un Arreglo de Memoria

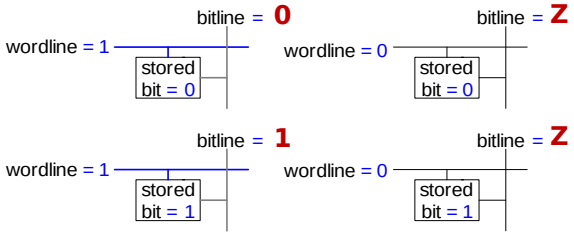




(a) (b)

Celdas de bit de un Arreglo de Memoria

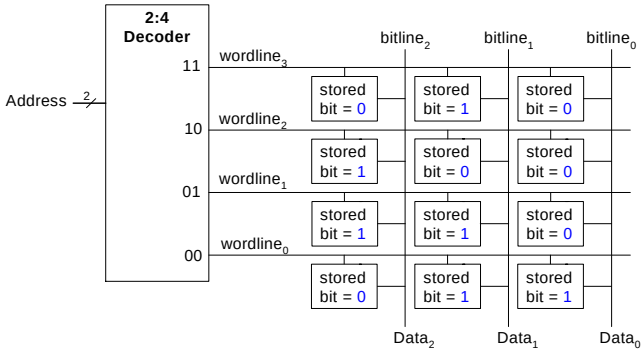




(a) (b)

Arreglo de Memoria

- **Wordline:**
  - Es como un “enable”
  - Es una sola fila del arreglo de memoria que puede ser leída/escrita
  - Posee una única dirección
  - Son una wordline esta en HIGH a la vez



Tipos de Memoria

- Memoria de Acceso Aleatorio (RAM): **volátil**
- Memoria de solo lectura (ROM): **no volátil**

RAM: Memoria de Acceso Aleatorio

- **Volátil:** pierde sus datos cuando es apagada
- Lee y escribe rápidamente
- La memoria principal de tu computadores es una RAM (DRAM)

Historicamente se le llama memorias de acceso aletorio porque cualquier palabra de datos puede ser accedida tan rapido como cualquier otra palabra (a diferencia de las memorias de acceso secuencial como con cintas de grabación)

ROM: Memoria de solo lectura

- **No volátil:** retiene los datos aun cuando se apague
- La lectura es rapida pero la escritura es imposible o lenta
- Las memoria Flash de cámaras, pendrives, y las camaras digitales son todas ROMs

Historicamente se le llama a una memoria de *solo lectura* porque las ROMs eran escritas en durante su proceso de fabricacion o con fusibles de calor. Una vez que la ROM era configurada, no era posible escribirla nuevamente. Ya no es el caso de las memorias Flash y de otros tipos de ROMs.

Tipos de RAM

- **DRAM** (Memoria de Acceso Aleatorio Dinámica)
- **SRAM** (Memoria de Acceso Aleatorio Estático)
- Difieren en como los datos se almacenan:
  - DRAM usa un condensador
  - SRAM usa inversores de acoplamiento cruzado

Arreglos Lógicos

- **PLAs** (Arreglos Lógicos Programables)
  - Arreglo de AND seguido por un arreglo de OR
  - Solo logica Combinacional
  - Conexiones internas fijas
- **FPGAs** (Arreglos de compuertas programables en Campo)
  - Arreglo de elementos lógicos (LEs)
  - Logica Combinacional y secuencial
  - Conexiones internas programables