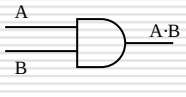
TEMA 6
UNIDAD ARITMÉTICO LÓGICA

1. **OPERADORES LÓGICOS.**
2. **PROPIEDADES DE LA UAL.**
3. **OPERADORES DE DESPLAZAMIENTO.**
   - Desplazamientos lógicos.
   - Desplazamientos circulares.
   - Desplazamientos aritméticos.
4. **OPERACIONES LÓGICAS.**
5. **OPERACIONES DE SUMA Y RESTA.**
   1. Sumador elemental.
   2. Sumador con acarreo. Generador rápido de arrastres.
6. **OPERACIÓN DE MULTIPLICACIÓN.**
   - Multiplicador por suma-desplazamiento.
   - Algoritmo de Booth.
7. **OPERACIÓN DE DIVISIÓN.**
   - Algoritmo de división con restauración.
   - Algoritmo de división sin restauración.

1. OPERADORES LÓGICOS.

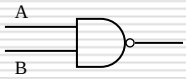
Puertas Lógicas

AND



	0	1
0	0	0
1	0	1

NAND

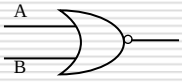


	0	1
0	1	1
1	1	0

OR

	0	1
0	0	1
1	1	1

NOR

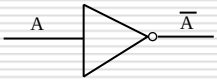


	0	1
0	1	0
1	0	0

XOR

	0	1
0	0	1
1	1	0

NOT



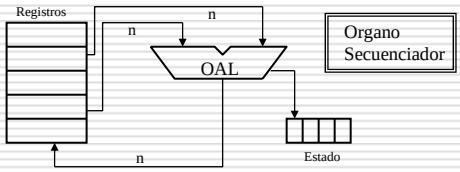
Leyes de Morgan

$$\overline{(a+b)} = \overline{a} \cdot \overline{b}$$

$$\overline{(a \cdot b)} = \overline{a} + \overline{b}$$

2. PROPIEDADES DE LA UAL.

- Elemento encargado de realizar cualquier tipo de operación.
- Mayor parte de las operaciones se basan en un simple sumador-restador (operaciones elementales).
- Operadores generales y especializados
- En función del número de dígitos con los que pueden operar a la vez se denominan operadores serie o paralelo.
- Estructura de una unidad aritmética:



3. OPERADORES DE DESPLAZAMIENTO

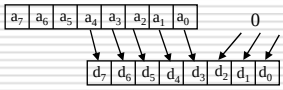
Consisten en mover todos los bits de una palabra hacia la izquierda o hacia la derecha.

$A = a_0a_1a_2\ldots d_n$ (Operando origen)

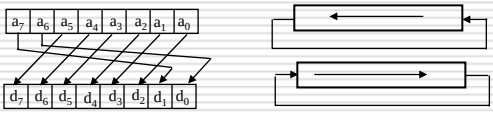
$D = d_0d_1d_2\ldots d_n$ (Operando destino)

$d_{i+k} = a_i \qquad i = 0, 1, \ldots N$

- **Desplazamientos lógicos**
  - Los valores extremos se completan con 'ceros'

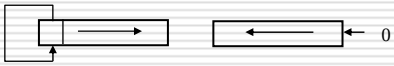


- **Desplazamientos circulares**

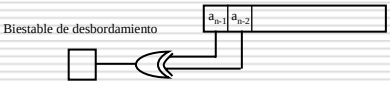


3. OPERADORES DE DESPLAZAMIENTO

- **Desplazamientos aritméticos**
  - Multiplicación o División por potencias de 2
    - Se debe conservar el signo del operando



- Detección del desbordamiento en multiplicación



4. OPERACIONES LÓGICAS.

- Operaciones:
  - NOT (!)
  - OR (|)
  - AND (&)
  - XOR (^) .....
- Se ejecutan bit a bit, según las tablas de la diapositiva 2.
- Ejemplo:

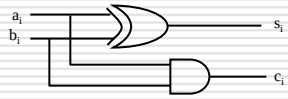
```
          0110 1011
XOR       1010 1001
_____________________

          1100 0010
```

5. OPERACIONES DE SUMA Y RESTA.

- Son dos operaciones muy típicas.
- Hay que tener en cuenta el bit resultante de la operación y el acarreo para sumas siguientes.
- La entrada a una operación de suma serán los dos bits sobre los que se realiza la operación y el acarreo previo, y a la salida debe devolver el resultado y el acarreo para la etapa siguiente.

Entradas			Salidas	
a_i	b_i	c_{i-1}	s_i	c_i
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

- CIRCUITO SUMADOR ELEMENTAL.



$s_i = a_i \text{ XOR } b_i$

$c_i = a_i \cdot b_i$

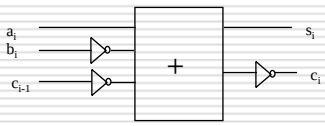
- CIRCUITO SUMADOR CON ACARREO.

$s_i = a_i \text{ XOR } b_i \text{ XOR } c_{i-1}$

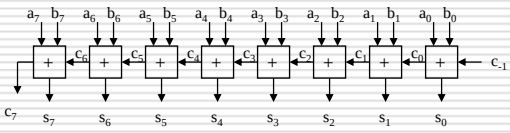
$c_i = a_i \cdot b_i + a_i \cdot c_{i-1} + b_i \cdot c_{i-1}$

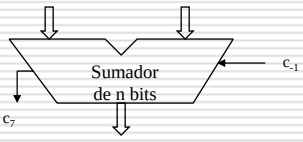
5. OPERACIONES DE SUMA Y RESTA.

- La resta se consigue con una pequeña modificación sobre la suma.



- Para sumar un número de bits determinada, hará falta colocar en serie varios sumadores elementales para ir sumando cada pareja de bits y para tener en cuenta el acarreo siguiente



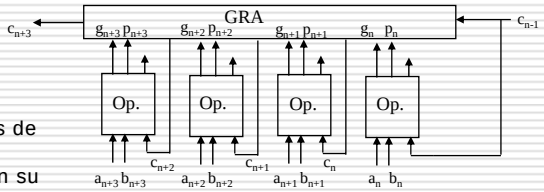


- El tiempo necesario para realizar una suma de n bits sería de n veces el tiempo necesario para una suma elemental

5. OPERACIONES DE SUMA Y RESTA.

- **Sumadores Rápidos: Sumador de Anticipador de Acarreo**
  - Genera simultáneamente todos los acarreos sin esperar a su propagación

- Funciones lógicas ***g*** y ***p***
- ***g***: Generador de acarreo
  $g_i = a_i \cdot b_i$
- ***p***: Propagador de acarreo
  $p_i = a_i$ ***XOR*** b_i
- Se forman directamente con los bits de cada etapa
- Requiere un solo nivel de puertas en su utilización



$c_n = g_n + p_n \cdot c_{n-1}$
$c_{n+1} = g_{n+1} + p_{n+1} \cdot c_n = g_{n+1} + p_{n+1} \cdot g_n + p_{n+1} p_n \cdot c_{n-1}$
$c_{n+2} = g_{n+2} + p_{n+2} \cdot c_{n+1} = g_{n+2} + p_{n+2} \cdot g_{n+1} + p_{n+2} \cdot p_{n+1} \cdot g_n + p_{n+2} \cdot p_{n+1} \cdot p_n \cdot c_{n-1}$
$c_{n+3} = g_{n+3} + p_{n+3} \cdot c_{n+2} = g_{n+3} + p_{n+3} \cdot g_{n+2} + p_{n+3} \cdot p_{n+2} \cdot g_{n+1} + p_{n+3} \cdot p_{n+2} \cdot p_{n+1} \cdot g_n + p_{n+3} \cdot p_{n+2} \cdot p_{n+1} \cdot p_n \cdot c_{n-1}$

- Todos se calculan simultaneamente con lo que se tienen los acarreos necesarios
- Se pueden conectar estos bloques para realizar estructuras más complejas
- Los GRA actúan bajo otros GRA con sus propias señales g* y p*

6. OPERACIONES DE MULTIPLICACIÓN.

Se pueden realizar de dos formas:
- **Circuito combinacional**
  - Sólo máquinas muy potentes.
- **Algoritmo secuencial**
  - En base a algoritmos.

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

- **Multiplicador por Suma-Desplazamiento**
  - Reproduce fielmente el método de multiplicar empleado manualmente.
  - Se multiplica usando únicamente sumas y desplazamientos.

1. Se observa el bit menos significativo de Q. Si es 1, se copia M y se suma. Si es 0, se suman todos ceros.
2. Se desplazan los registros A y Q a la derecha.
3. Se repiten los pasos 1 y 2 tantas veces como bits tenga el multiplicador (Q).
4. El resultado se encuentra en A y Q.

```
 M        A          Q
0 0 1 0   0 0 0 0    0 1 0 1
          0 0 1 0
          -------
          0 0 1 0    0 1 0 1
          0 0 0 1    0 0 1 0 →
          0 0 0 0
          -------
          0 0 0 1    0 0 1 0
          0 0 0 0    1 0 0 1 →
          0 0 1 0
          -------
          0 0 1 0    1 0 0 1
          0 0 0 1    0 1 0 0 →
          0 0 0 0
          -------
          0 0 0 1    0 1 0 0
          0 0 0 0    1 0 1 0 →
```

6. OPERACIONES DE MULTIPLICACIÓN.

- **Algoritmo de Booth:**
  - Permite operar números positivos y negativos en formato C2.
  - En el multiplicador (Q) se añade un cero por la derecha.
  - En lugar de desplazamientos lógicos, aritméticos (se preserva el bit de signo).
  - En lugar de ver el bit menos significativo se observan los dos menos significativos:
    - 10 : Restar y desplazar
    - 01 : Sumar y desplazar
    - 00 : Desplazar
    - 11 : Desplazar
  - El algoritmo acaba cuando se terminan de desplazar todos los bits del multiplicador

M	A	Q	
1011	0000	00100	→
	0000	00010	
	− 1011		
	0101	00010	
	0010	10001	→
	+ 1011		
	1101	10001	
	1110	11000	→
	1111	01100	→

- 1011 x 0010 = 1110110 (-5*2 = -10)
- Con el anterior algoritmo hubiera dado 00001010

7. OPERACIONES DE DIVISIÓN

7.1. Método de División con Restauración

- M = divisor, A = 0, Q = dividendo.
- Se añade a M un cero por la izquierda.
- nº bits de A = nº bits de M.
- Se resta el divisor (M) de A.
- Si el resultado cabe, se desplazan A y Q a la izquierda, introduciendo un 1.
- Si no cabe, se restaura A, y se desplazan A y Q a la izquierda introduciendo un 0.
- Se finaliza cuando se ha realizado una operación de desplazamiento más que el número de bits de Q.
- En A y Q estará el resto y el cociente de la división.
- 100110 : 101 = 111 (38:5 = 7 + 3/5)

M	A	Q	
0101	0000	100110	
	0101 -		
	1011 +		
	0101		
	0000	100110	←
	0001	001100	
	0101 -		
	1100		
	0101 +		
	0001	001100	←
	0010	011000	
	0101 -		
	1101		
	0101 +		
	0010	011000	←
	0100	110000	
	0101 -		
	1111		
	0101 +		
	0100	110000	←
	1001	100000	
	0101 -		
	0100	100000	←
	1001	000001	
	0101 -		
	0100	000001	←
	1000	000011	
	0101 -		
	0011	000011	←
	0110	000111	

Resto

Cociente

7. OPERACIONES DE DIVISIÓN

7.2. Método de División sin Restauración

- M = divisor, A = 0, Q = dividendo, como en el caso anterior.
- Se resta M de A.
- Se observa el bit mas significativo del resultado.
  - Si es '1', se desplaza insertando un 0 y se suma.
  - Si es '0', se desplaza insertando un 1 y se resta.
- Se finaliza cuando se ha realizado una operación de desplazamiento más que el número de bits de Q.
- En A y Q estará el resto y el cociente de la división.
- Si la última operación es una resta el resto obtenido es válido. Si es una suma habría que restaurar el contenido de los registros.
- Algoritmo más rápido que el anterior puesto que se avanza en todas las operaciones, no es necesario hacer operaciones que después se deshacen.

M	A	Q	
0101	0000	100110	
	0101 -		
	1011	100110	←
	0111	001100	
	0101 +		
	1100	001100	←
	1000	011000	
	0101 +		
	1101	011000	←
	1010	110000	
	0101 +		
	1111	110000	←
	1111	100000	
	0101 +		
	0100	100000	←
	1001	000001	
	0101 -		
	0100	000001	←
	1000	000011	
	0101 -		
	0011	000011	←
	0110	000111	