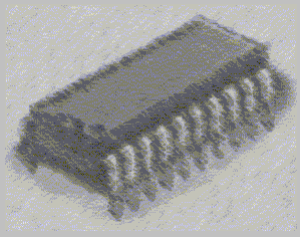
# Tema 3. La Unidad Aritmético-Lógica



***Arquitectura de Computadores I***

I. T. Informática de Sistemas

Curso 2009-2010



## Índice

# Estructura e implementación de la ALU (I)

- **Unida aritmético-lógica (ALU)**: es el conjunto de operadores disponibles en un computador
- Formada por:
  - Operadores:aritméticos, lógicos y de desplazamiento
  - Registros para almacenar datos temporales
  - Registro de estado: conjunto de *flags* que indican situaciones ocurridas al operar
  - Registro contador de programa
  - Registro de direcciones de interrupción
- Tipos de ALU:
  - Coma fija
  - Coma flotante





# Estructura e implementación de la ALU (y II)

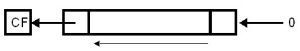
**Clasificación de los operadores:**

| | |
|---|---|
| Ámbito de aplicación | General<br>Especializado |
| Realización | Combinacional<br>Secuencial |
| Número de operandos | Monádico<br>Diádico |
| Paralelismo | Serie o de dígito<br>Paralelo o de vector |
| Operación | De desplazamiento<br>Lógico<br>Aritmético |
| Tecnología empleada | MOS<br>Bipolar |

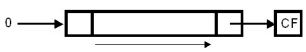
# Circuitos y algoritmos para la ALU (I)
# Operaciones de desplazamiento (I)

- **Desplazamientos lógicos**:
  - Independientemente del sistema de representación de los operandos se introducen ceros por la derecha o por la izquierda según se trate de un desplazamiento a la izquierda o a la derecha, respectivamente
  - El bit o los bits que salen suelen copiarse en el indicador de acarreo (el último que ha salido es el que queda)

Desplazamientos lógicos a la izquierda




Desplazamientos lógicos a la derecha



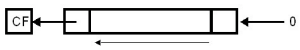




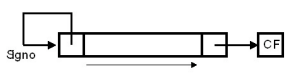

# Circuitos y algoritmos para la ALU (II)
# Operaciones de desplazamiento (II)

- **Desplazamientos aritméticos**:
  - Equivalen a multiplicaciones y divisiones por dos, según sean hacia la izquierda o a la derecha, respectivamente.
  - El sistema de representación de los operandos debe tenerse en cuenta si los operandos tienen signo
  - El bit o los bits que salen suelen copiarse en el indicador de acarreo (el último que ha salido es el que queda)

Desplazamientos aritméticos a la izquierda en C2



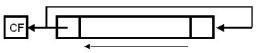

Desplazamientos aritméticos a la derecha en C2



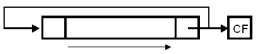
# Circuitos y algoritmos para la ALU (III)
# Operaciones de desplazamiento (III)

- **Desplazamientos circulares**:
  - Los bits que salen por un extremo entran por el otro
  - El bit o los bits que salen suelen copiarse en el indicador de acarreo (el último que ha salido es el que queda)

Desplazamiento concatenado a la izquierda




Desplazamiento concatenado a la derecha



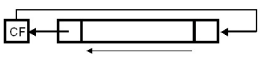




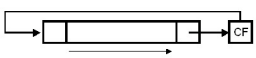

# Circuitos y algoritmos para la ALU (IV)
# Operaciones de desplazamiento (y IV)

- **Desplazamientos circulares a través del flag de acarreo**:
  - Los bits que salen por un extremo entran por el otro
  - El bit o los bits que salen suelen copiarse en el indicador de acarreo

Desplazamiento concatenado a la izquierda a través del acarreo



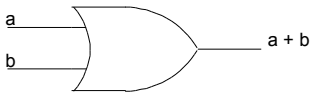

Desplazamiento concatenado a la derecha a través del acarreo



# Circuitos y algoritmos para la ALU (V)
# Operaciones Lógicas (I)

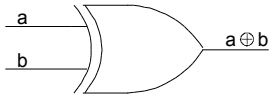
- Las operaciones lógicas realizan la operación sobre cada uno de los bits del operando o de los operandos

OR




| a | b | a OR b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

XOR



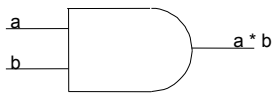

| a | b | a XOR b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |





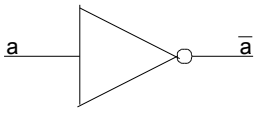
# Circuitos y algoritmos para la ALU (VI)
# Operaciones Lógicas (y II)

AND




| a | b | a AND b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

NOT

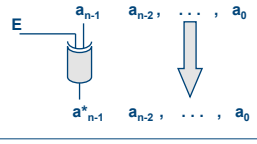



| a | $\overline{a}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

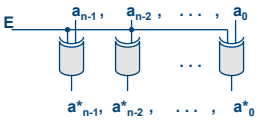
# Circuitos y algoritmos para la ALU (VII)
# Operaciones Aritméticas (I)

- **Operaciones sobre el signo. Cambio de signo**

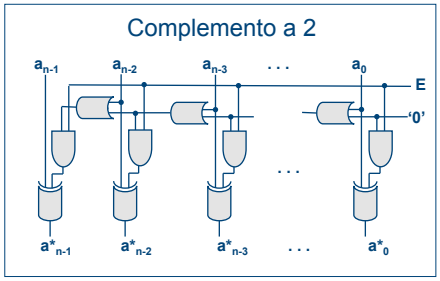
Signo Magnitud




Complemento a 1




Complemento a 2





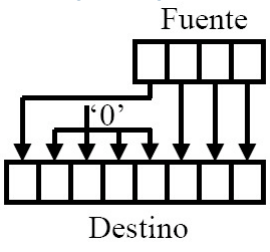



# Circuitos y algoritmos para la ALU (VIII)
# Operaciones Aritméticas (II)

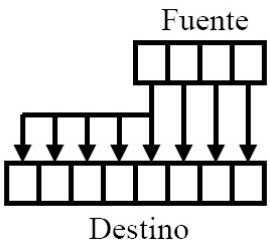
- **Operaciones sobre el signo. Extensión de signo.**

Es cuando se pasa de un operando de *n* bits a otro de *m* bits con *m* > *n*. Consiste en ver como "rellenar" los bits que sobran conservando el signo

Signo Magnitud




Complemento a 1 y a 2

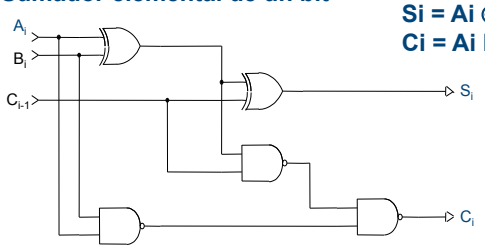


# Circuitos y algoritmos para la ALU (IX) Operaciones Aritméticas (III)

- **Suma:** la suma es las operación más importante de todas, ya que:
  - Se emplea para el cálculo de la dirección de la siguiente instrucción
  - Se utiliza para el cálculo de las direcciones a los operandos
  - Otras operaciones la emplean: multiplicación, división

**Sumador elemental de un bit**

**Si = Ai ⊕ Bi ⊕ Ci-1**
**Ci = Ai Bi + Bi Ci-1 + Ai Ci-1**

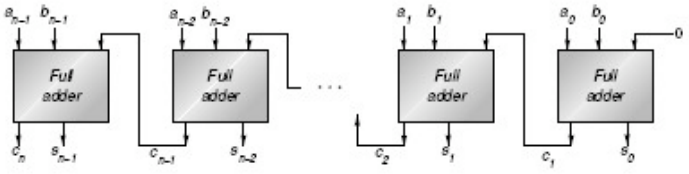







# Circuitos y algoritmos para la ALU (X) Operaciones Aritméticas (IV)

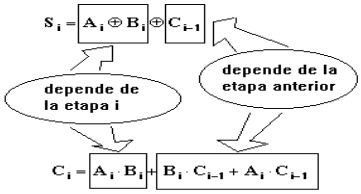
- **Sumador con propagación de acarreo de *n* bits empleando sumadores completos (*full adder*) de 1 bit**

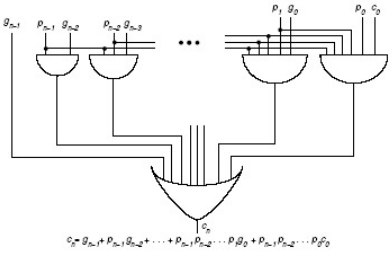



- **Problemas:**
  - Muy lento porque debe propagarse el acarreo desde el primer sumador al segundo, del segundo al tercero y así hasta que llegue al último sumador
  - El retardo es $2 \cdot n \cdot r$ ya que es el máximo retardo de niveles de puertas que tiene que pasar para obtener el acarreo es de dos, $n$ es el número de sumadores y $r$ el retardo de las puertas

# Circuitos y algoritmos para la ALU (XI)
# Operaciones Aritméticas (V)

- **Aceleración de la suma entera:**
  - Debido a la importancia de la suma se han diseñado circuitos que permiten acelerar el proceso Se intenta identificar qué es lo que depende de los bits de la suma y del acarreo para mediante circuitos adicionales anticipar su cálculo



Función generación ($g_i = a_i \cdot b_i$)
Función propagación ($p_i = a_i \oplus b_i$)
Acarreo siguiente ($c_{i+1} = g_i + p_i \cdot c_i$)

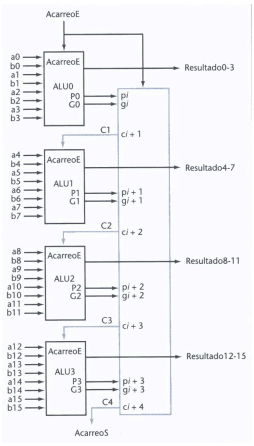
- **Problema:** *fan-in* variable





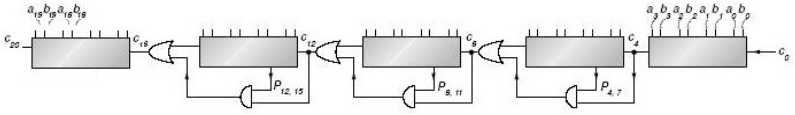
# Circuitos y algoritmos para la ALU (XII)
# Operaciones Aritméticas (VI)

- **Aceleración de la suma entera:**
  - Se intenta homogeneizar la *fan-in* a partir de sumadores con propagación de acarreo de 4 bits como bloque básico y se definen las fuciones de Generación y Propagación a nivel de bloque para obtener, por ejemplo un sumador de 16 bits con anticipación
- **Funciones a nivel de bloque**
  - $P_0 = p_3 \cdot p_2 \cdot p_1 \cdot p_0$
  - $G_0 = g_3 + (g_2 p_3) + (g_1 p_3 p_2) + (g_0 p_3 p_2 p_1)$
  - $C_1 = G_0 + P_0 \cdot c_e$
  - $C_2 = G_1 + (G_0 \cdot P_1) + (c_e \cdot P_1 \cdot P_0)$



# Circuitos y algoritmos para la ALU (XIII)
# Operaciones Aritméticas (VII)

- **Aceleración de la suma. Salto de acarreo (CSK)**
  - Es un sumador intermedio entre uno de propagación y uno de anticipación.
  - Se basa en calcular las Pi que son de cálculo más sencillo.
- **Para un sumador de salto de acarreo de K bits:**
  - Primer propagador: *2k +1 (el primer sumador un nivel más)*
  - Número de puertas: *2(n/k-2)*
  - Último propagador: *2k*
  - Total: número niveles = *4k + 2n/k -3*
  - En nuestro ejemplo *n = 20, K = 4* → num. Niveles = 21

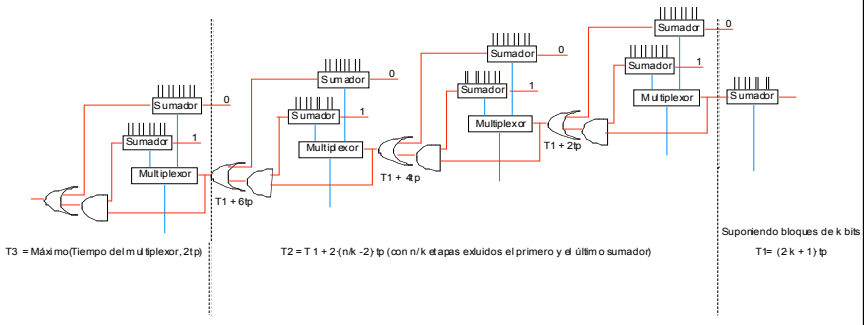






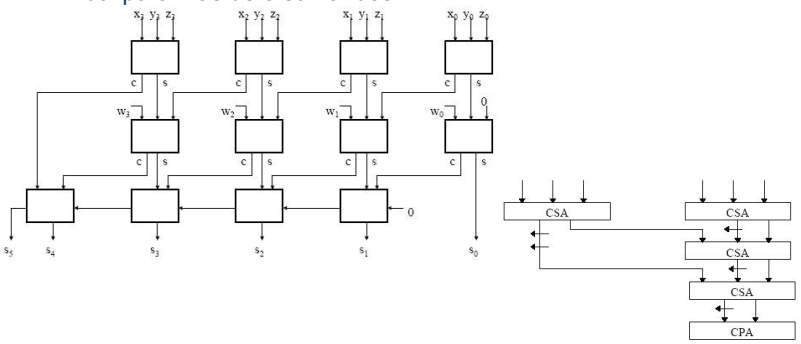
# Circuitos y algoritmos para la ALU (XIV)
# Operaciones Aritméticas (VIII)

- **Aceleración de la suma. Selección de acarreo**
  - Se basa en duplicar el hardware. Se suman los bits suponiendo acarreo cero y acarreo 1 y mediante multiplexores se elige el resultado correcto una vez conocido el acarreo



# Circuitos y algoritmos para la ALU (XV) Operaciones Aritméticas (IX)

- **Aceleración de la suma: sumador sin propagación (CSA)**
  - Reduce el cálculo del tiempo casi a la mitad pero solamente es útil para más de 3 sumandos





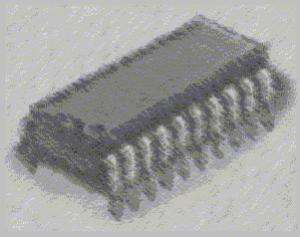



# Circuitos y algoritmos para la ALU (XVI) Operaciones Aritméticas (X)

- **Tiempo de proceso y espacio requerido por cada uno de los distintos tipos de sumadores**

| Sumador | Tiempo | Espacio |
|---|---|---|
| Propagación | $O(n)$ | $O(n)$ |
| Anticipación | $O(\log n)$ | $O(n \log n)$ |
| Salto | $O(\sqrt{n})$ | $O(n)$ |
| Selección | $O(\sqrt{n})$ | $O(n)$ |

# Tema 3. La Unidad Aritmético-Lógica



***Arquitectura de Computadores I***

I. T. Informática de Sistemas

Curso 2008-2009

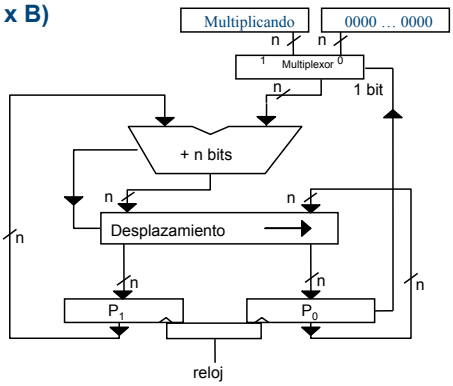


## Índice

- Estructura e implementación de la ALU
- Circuitos y algoritmos para la ALU:
  - Operadores lógicos y de desplazamiento
  - Operaciones sobre el signo
  - Suma en coma fija
  - Suma en coma flotante
  - Dígitos de guarda
  - Técnicas de redondeo
  - Multiplicación y división en coma fija
  - Multiplicación y división en coma flotante
- La ALU en la arquitectura von Neuman: camino de datos y unidades funcionales
- La unidad de ejecución del MC60888
- Bibliografía

# Circuitos y algoritmos para la ALU (XVII)
# Operaciones Aritméticas (XI)

- **Multiplicación. Algoritmo de suma desplazamiento (A x B)**
- Inicialmente $P_0$ = B
- Sólo números sin signo

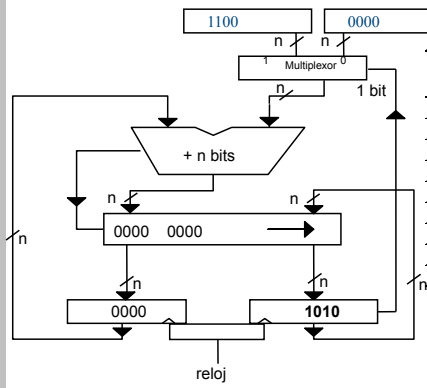







# Circuitos y algoritmos para la ALU (XVIII)
# Operaciones Aritméticas (XII)

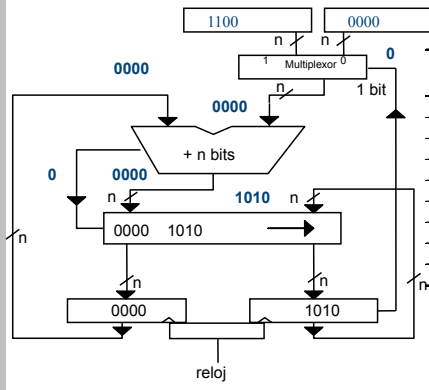
- **Ejemplo A = 1100 y B = 1010**




| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Circuitos y algoritmos para la ALU (XIX)
# Operaciones Aritméticas (XIII)

- **Ejemplo A = 1100 y B = 1010**



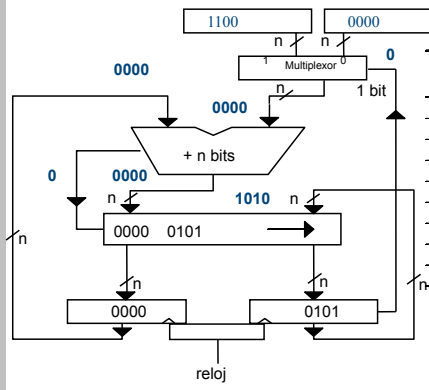

| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| **0000 1010** | **0000** | **1010** | **Suma** |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |





# Circuitos y algoritmos para la ALU (XX)
# Operaciones Aritméticas (XIV)

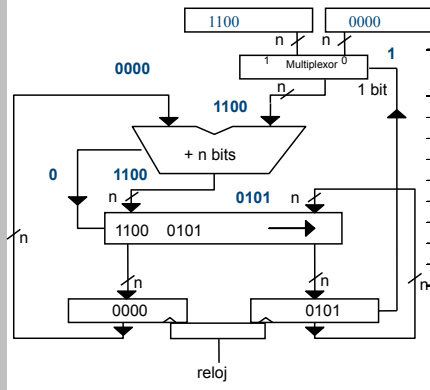
- **Ejemplo A = 1100 y B = 1010**




| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| **0000 0101** | **0000** | **0101** | **Desplazamiento** |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Circuitos y algoritmos para la ALU (XXI)
# Operaciones Aritméticas (XV)

- **Ejemplo A = 1100 y B = 1010**



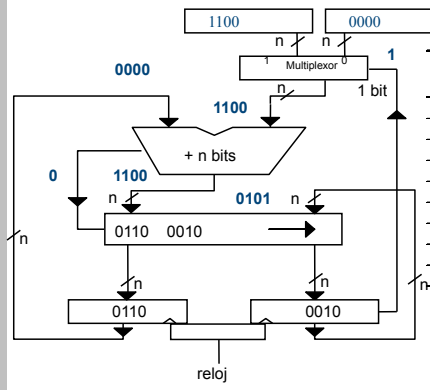

| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| 0000 0101 | 0000 | 0101 | Desplazamiento |
| **1100 0101** | **0000** | **0101** | **Suma** |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |





# Circuitos y algoritmos para la ALU (XXII)
# Operaciones Aritméticas (XVI)

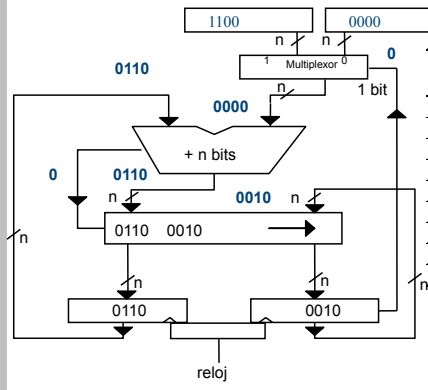
- **Ejemplo A = 1100 y B = 1010**




| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| 0000 0101 | 0000 | 0101 | Desplazamiento |
| 1100 0101 | 0000 | 0101 | Suma |
| **0110 0010** | **0110** | **0010** | **Desplazamiento** |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Circuitos y algoritmos para la ALU (XXIII)
# Operaciones Aritméticas (XVII)

- **Ejemplo A = 1100 y B = 1010**



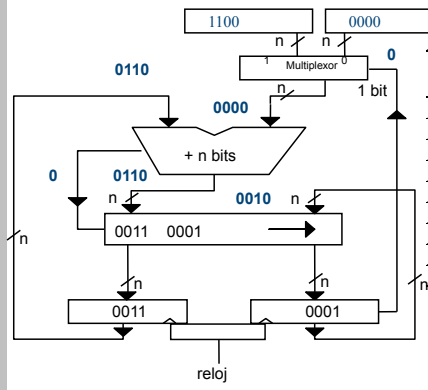

| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| 0000 0101 | 0000 | 0101 | Desplazamiento |
| 1100 0101 | 0000 | 0101 | Suma |
| 0110 0010 | 0110 | 0010 | Desplazamiento |
| **0110 0010** | **0110** | **0010** | **Suma** |
| | | | |
| | | | |
| | | | |





# Circuitos y algoritmos para la ALU (XXIV)
# Operaciones Aritméticas (XVIII)

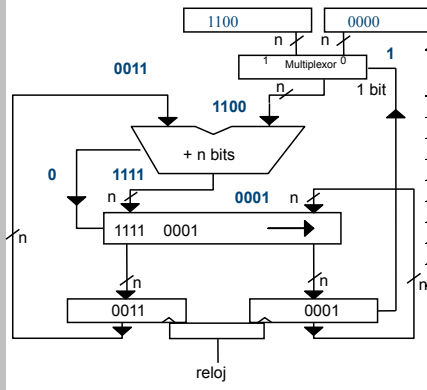
- **Ejemplo A = 1100 y B = 1010**




| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| 0000 0101 | 0000 | 0101 | Desplazamiento |
| 1100 0101 | 0000 | 0101 | Suma |
| 0110 0010 | 0110 | 0010 | Desplazamiento |
| 0110 0010 | 0110 | 0010 | Suma |
| **0011 0001** | **0011** | **0001** | **Desplazamiento** |
| | | | |
| | | | |
| | | | |

# Circuitos y algoritmos para la ALU (XXV)
# Operaciones Aritméticas (XIX)

- **Ejemplo A = 1100 y B = 1010**



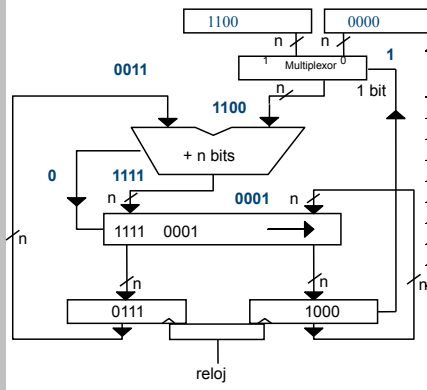

| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| 0000 0101 | 0000 | 0101 | Desplazamiento |
| 1100 0101 | 0000 | 0101 | Suma |
| 0110 0010 | 0110 | 0010 | Desplazamiento |
| 0110 0010 | 0110 | 0010 | Suma |
| 0011 0001 | 0011 | 0001 | Desplazamiento |
| **1111 0001** | **0011** | **0001** | **Suma** |





# Circuitos y algoritmos para la ALU (XXVI)
# Operaciones Aritméticas (XX)

- **Ejemplo A = 1100 y B = 1010**




| Registro de desplazamiento | P | | Operación |
|---|---|---|---|
| | $P_1$ | $P_0$ | |
| 0000 0000 | 0000 | 1010 | Estado inicial |
| 0000 1010 | 0000 | 1010 | Suma |
| 0000 0101 | 0000 | 0101 | Desplazamiento |
| 1100 0101 | 0000 | 0101 | Suma |
| 0110 0010 | 0110 | 0010 | Desplazamiento |
| 0110 0010 | 0110 | 0010 | Suma |
| 0011 0001 | 0011 | 0001 | Desplazamiento |
| 1111 0001 | 0011 | 0001 | Suma |
| **0111 1000** | **0111** | **1000** | **Desplazamiento** |

# Circuitos y algoritmos para la ALU (XXVII)
# Operaciones Aritméticas (XXI)

- **Ajustes para operar con números representados en complemento a 1 y complemento a 2 (con signo)**
  - **Complemento a 2**: si el multiplicador es negativo, cuando llegue el 1 del bit más significativo al multiplexor, entonces, restar A
  - **Complemento a 1**: si el multiplicador es negativo, cuando llegue el 1 del bit más significativo al multiplexor, entonces, restar A. Y además, en la fase de inicialización P1 y el registro de desplazamiento toman el valor inicial del multiplicando





# Circuitos y algoritmos para la ALU (XXVIII)
# Operaciones Aritméticas (XXII)

- **Multiplicación. Algoritmo de Booth (C2)**
  - Se trata de evitar las sumas de cero que consumen ciclos de máquina y no aportan nada al cálculo puesto que solamente se desplazaría luego el resultado.
  - Emplea la recodificación del multiplicador buscando cadenas de 1's y afectándoles del peso de ese 1 ó del 0

- **A = 1011, B = 0110**
- **A x B = +Ax$2^3$ –Ax$2^1$**

  0001010
  1011000 +
  **1100010**

| $x_i$ | $x_{i-1}$ | significado | acción asociada | $y_i$ |
|---|---|---|---|---|
| 0 | 0 | cadena de 0s | desplazar | 0 |
| 1 | 1 | cadena de 1s | desplazar | 0 |
| 1 | 0 | principio de cadena de 1s | restar y desplazar | -1 |
| 0 | 1 | fin de cadena de 1s | sumar y desplazar | 1 |

# Circuitos y algoritmos para la ALU (XXIX)
# Operaciones Aritméticas (XXIII)

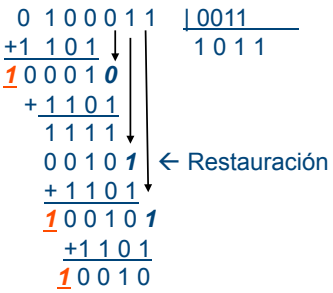
- **División. Algoritmo de división con restauración (sin signo)**
  - **Dividendo parcial inicial**: tomar tantos bits del dividendo como tenga el divisor.
  - Garantizar que tanto el dividendo como el divisor son positivos (si empiezan por 1 añadirle un cero a la izquierda)
  - Sumar al dividendo el complemento a 2 del divisor
    - Si el resultado es positivo:
      - Bajar un nuevo bit del dividendo
      - Añadir 1 al cociente
    - Si el resultado es negativo:
      - Sumar de nuevo el divisor
      - Bajar un nuevo bit del dividendo
      - Añadir un 0 al cociente
  - Repetir hasta que no queden más bits para bajar del dividendo





# Circuitos y algoritmos para la ALU (XXX)
# Operaciones Aritméticas (XXIV)

- **División. Algoritmo de división con restauración (sin signo)**
  - A = 0100011, B = 0011, Cociente = 1011, Resto = 0010



# Circuitos y algoritmos para la ALU (XXXI)
# Operaciones Aritméticas (XXV)

- **División. Algoritmo de división sin restauración (sin signo)**
  - **Dividendo parcial inicial**: tomar tantos bits del dividendo como tenga el divisor.
  - Garantizar que tanto el dividendo como el divisor son positivos (si empiezan por 1 añadirle un cero a la izquierda)
  - Sumar al dividendo el complemento a 2 del divisor
    - Si el resultado es positivo:
      - Bajar un nuevo bit del dividendo
      - Añadir 1 al cociente
    - Si el resultado es negativo:
      - Sumar de nuevo el divisor en vez del complemento al bajar el siguiente bit
      - Bajar un nuevo bit del dividendo
      - Añadir un 0 al cociente
  - Repetir hasta que no queden más bits para bajar del dividendo





# Circuitos y algoritmos para la ALU (XXXII)
# Operaciones Aritméticas (XXVI)

- **División. Algoritmo de división sin restauración (sin signo)**
  - A = 0100011, B = 0011, Cociente = 1011, Resto = 0010

```
 0 1 0 0 0 1 1   | 0011
+1 1 0 1         ------
 0 0 0 1 0         1 0 1 1
 + 1 1 0 1
   1 1 1 1 1
   + 0 0 1 1
   1 0 0 1 0 1
     + 1 1 0 1
     1 0 0 1 0
```

# Circuitos y algoritmos para la ALU (XXXIII) Operaciones Aritméticas (y XXVII)

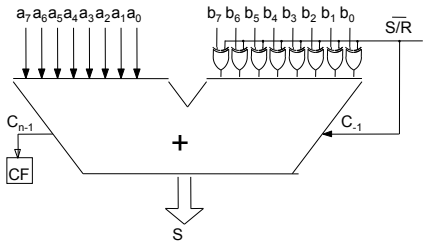
- **Multiplicación y división en coma flotante**
  - Se pueden adaptar los algoritmos vistos antes
  - Los algoritmos se aplican a las mantisas
  - Los exponentes se suman (multiplicación) o se restan (división)
  - El resultado deberá ser normalizado y redondeado
  - Se emplean bits de guarda para mejorar la precisión





# Circuitos y algoritmos para la ALU (XXXIV) Circuitos (I)

- **Sumador-restador en binario puro**



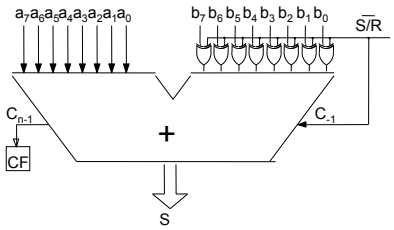

$Desbordamiento = c_{n-1} \oplus \bar{S}/R$

- **Sumador-restador en signo-magnitud**

Se realiza tomando un sumador-restador en binario puro, y se le añade el circuito para tratar los signos de los operandos

# Circuitos y algoritmos para la ALU (XXXV)
# Circuitos (II)

- **Sumador-restador en complemento a 2**



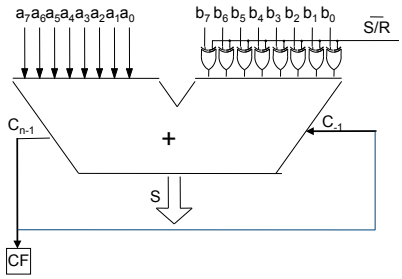

*Desbordamiento* $= c_{n-1} \oplus c_{n-2}$





# Circuitos y algoritmos para la ALU (XXXVI)
# Circuitos (III)

- **Sumador-restador en complemento a 1**




*Desbordamiento* $= c_{n-1} \oplus c_{n-2}$

# Circuitos y algoritmos para la ALU (XXXVII)
# Circuitos (IV)

- **Sumador-restador en exceso**

Habrá que corregir el resultado:

- Si la operación es una suma → restar M
- Si la operación es una resta → sumar M

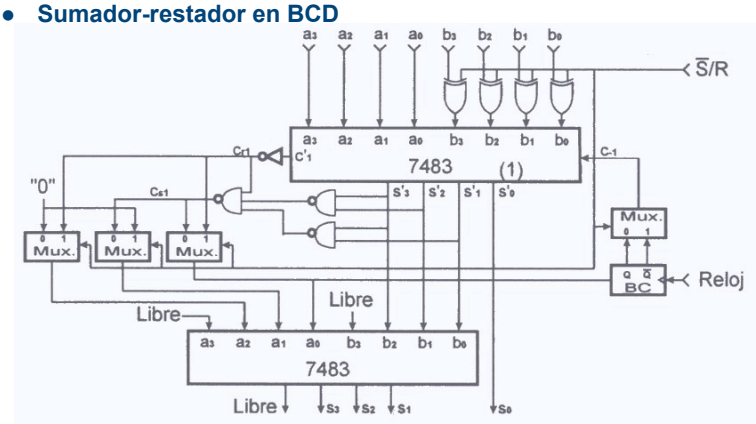
- Si **M = $2^{n-1}$** bastará con:
  1º Invertir el bit más significativo de cada operando
  2º Operar como si fueran representaciones en C2
  3º Invertir el bit más significativo del resultado





# Circuitos y algoritmos para la ALU (XXXVIII)
# Circuitos (y V)

- **Sumador-restador en BCD**



# Circuitos y algoritmos para la ALU (XXXIX)
# Suma-resta en coma flotante (I)

- **Para sumar o restar números en coma flotante se debe:**
  1. Separar las mantisas de los exponentes
  2. Comparar los exponentes y:
     - Guardar el exponente mayor que será el del resultado salvo que el número salga desnormalizado
     - Restar del exponente mayor el menor. Dicho número será el número de veces que se tendrá que desplazar a la derecha la mantisa menor
  3. Desplazar la mantisa menor a la derecha para alinear las mantisas
  4. Realizar la suma o la resta
  5. Comprobar si el número está normalizado y en caso de que no lo esté, normalizarlo
  6. Realizar el redondeo si es preciso





# Circuitos y algoritmos para la ALU (XXXX)
# Suma-resta en coma flotante (y II)

- **Ejemplo de suma en coma flotante:**
  - Sean A y B dos números expresados en coma flotante. Exponente en exceso $2^{n-1}$ con 8 bits y mantisa también con 8 bits, expresada en complemento a 2, normalizada y sin bit implícito
  - EA = 1000 0100  MA = 0100 0100
  - EB = 1000 0011  MB = 0111 0000
  - Comparamos exponentes EA(4) > EB(3). Exponente resultado EA
  - Desplazamos MB EA-EB veces, es decir, 4 – 3 = 1
  - 0100 0100
    0011 1000 +
    0111 1100
  - ER = 1000 0100  MR = 0111 1100

# Circuitos y algoritmos para la ALU (XXXXI)
# Dígitos de guarda

- Los bits de guarda se añaden y se emplean únicamente dentro de la Unidad Aritmético-Lógica.
- Se emplean para aumentar la precisión de los resultados y permitir el redondeo y la normalización de manera correcta.
- Normalmente se emplean 2 bits de guarda y un bit retenedor que se añaden al final del número:

$$b_8b_7b_6b_5b_4b_3b_2b_1b_0\ b_{g1}\ b_{g2}\ b_r$$

- Los bits $b_8$ a $b_0$ son un dato de 8 bits
- $b_{g1}$es el primer bit de guarda que se emplea para la normalización
- $b_{g2}$ es el segundo bit de guarda que se emplea para el redondeo
- $b_r$ es el bit retendor que se emplea para no perder la precisión en la operación de resta y que se mantiene a 1 al pasar un 1 por él





# Circuitos y algoritmos para la ALU (XXXXII)
# Técnicas de redondeo

- Al pasar de la ALU a los registros o la memoria, los bits de guarda y el bit retenedor, tienen que eliminarse por lo que se deberán tener en cuanta para mejorar la precisión del resultado
- Existen varias técnicas de redondeo entre las que destacan:
  - **Truncación**: se eliminan los bits de guarda y el bit retenedor
  - **Forzar a 1 el LSB**: se pone a 1 el bit menos significativo del resultado y se truncan los bits de guarda
  - **Redondeo al más próximo**: es la más difícil de implementar pero la que mejor resultado proporciona. Si supera la mitad del valor de los bits de guarda se suma uno al resultado. Si no llega a la mitad, se trunca. En el caso de que sea la mitad, se fuerza a par.
  - Para 3 bits de guarda las combinaciones 000, 001, 010, 011 truncaría. Con 101, 110 y 111 se sumaría 1 al resultado. La combinación 100 sumará 1 si el LSB es 1 ó truncará si el LSB es 0

# ALU y Arquitectura Von Neumann (I) Camino de datos

- El órgano aritmético de la arquitectura von Neumann se denomina camino de datos o *datapath*
- Consta de: unidades aritmético-lógicas, desplazadores, registros y caminos de comunicación entre ellas
- El camino de datos contiene el estado del computador
- También contiene el registro contador de programa CP y el registro de direcciones de interrupción
- El camino de datos influye en el coste del procesador, aproximadamente necesita la mitad de los transistores y la mitad del área de silicio del procesador
- La duración del ciclo de reloj está determinada por los circuitos más lentos por lo que en el procesador es el camino de datos el que hace de cuello de botella
- Los pasos clave para diseñar el camino de datos son:
  - Elegir el número de puertos del banco de registros
  - Seleccionar el tipo de ALU o de ALUs

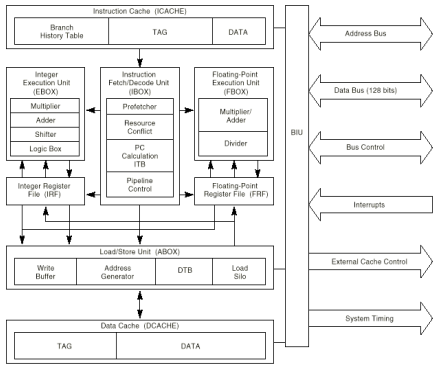




# ALU y Arquitectura Von Neumann (II) Camino de datos. Registro de estado

- Al realizar operaciones aritméticas, existen una serie de biestables de estado asociados a ellas y que se suelen recoger en un registro de estado
- El objetivo es guardar constancia de algunas características del resultado de las operaciones realizadas
- Los biestables de estado más frecuentes son:
  - Cero, signo, desbordamiento y acarreo
  - Paridad del resultado, acarreo BCD
- Condiciones de excepción. Algunas de ellas se basan en el contenido de los biestables de estado: desbordamiento.

# Ejemplo de hardware real (I)
# Alpha 21064. Aceleración de la suma (I)




**Unidad de cálculo entero (Ebox )**

Contiene un camino de datos de 64 bits

- Adder
- Logic box
- Barrel shifter
- Bypassers
- Integer multiplier
- Archivo de registros con 32 registros de 64 bits, 4 ports de lectura y 2 de escritura.

**Unidad de cálculo de Punto Flotante (Fbox )**

Contiene:

- Multiplicador/Sumador
- Divisor
- Un archivo de registros de 32 entradas, con registros de punto flotante de 64 bits
- Un registro de control accesible por el usuario (FPCR) que contine:
  - Controles para modos de redondeo dinámico
  - Información del flag de excepciones

Acepta una instrucción por ciclo, con excepción la instrucción de división.





# Ejemplo de hardware real (II)
# Alpha 21064. Aceleración de la suma (y II)

- El sumador de las unidades de enteros y de coma flotante del microprocesador Alpha 21064 es de 64bits y realiza 2 sumas por ciclo de reloj ya que es segmentado de 2 etapas
- Se combinan tres métodos de suma binaria:
  - Propagación de acarreo en bloques de 8 bits
  - Selección de acarreo en 2 bloques de 32 bits
  - Anticipación de acarreo entre ambos bloques de 32 bits
- En todos los caminos de datos se han colocado latches de manera que el operador cuenta con dos etapas aisladas que permiten realizar dos operaciones por ciclo

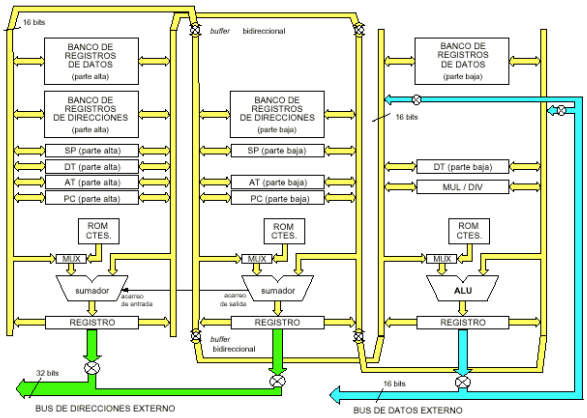
# Ejemplo de hardware real (III)
# La unidad de ejecución del MC68000 (I)


- 8 registros de direcciones de 32 bits divididos en parte alta y parte baja en 2 bancos separados
- Operaciones sobre:
  - Direcciones pueden ser de 16 o de 32 bits
  - Datos de 8, 16 ó 32 bits.
- El conjunto de los registros esta organizado en 3 secciones de modo que se puedan realizar operaciones a la vez en cada una de las secciones.
- La UE posee una unidad que realiza operaciones especiales





# Ejemplo de hardware real (y IV)
# La unidad de ejecución del MC68000 (y II)



# Bibliografía


- Estructura y diseño de computadores
  David A. Patterson y John L. Hennessy. Reverté, 2000
  Capítulo 4

- Arquitectura de computadores. Un enfoque cuantitativo
  John L. Hennessy y David A. Patterson. Mc Graw Hill, 3ª ed, 2002
  Apéndices A: aritmética de computadores

- Organización y arquitectura de computadores
  William Stallings. Prentice Hall. 1996
  Capítulo 8