# Fundamentos de Microprocesadores

## *Unidad 2: La Unidad Aritmético Lógica (ALU)*

Escuela Politécnica Superior - UAM

# Índice

- **Estructura básica de un ordenador (sumador)**
- Circuitos lógicos y aritméticos
  - ✓ Sumadores y Restadores
  - ✓ Desplazadores y Multiplicadores
  - ✓ Otros operadores
- Diseño de una ALU optimizada

# Introducción

- Para diseñar un microprocesador, inicialmente se hace un diseño jerárquico reutilizando bloques.

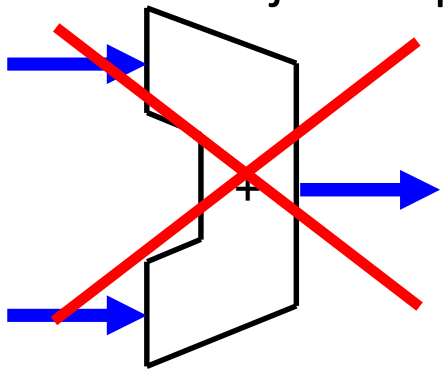
- Bloques que usaremos:
  - ✓ Multiplexores, decodificadores, registros y memorias, circuitos lógicos y aritméticos, etc…

- Nos planteamos realizar un circuito ordenador muy simple para sumar un número cualquiera de operandos.

# Circuito sumador genérico

Nos planteamos realizar un circuito para sumar un número cualquiera de operandos de 32 bits (un sumador de un número fijo de operandos no sirve):

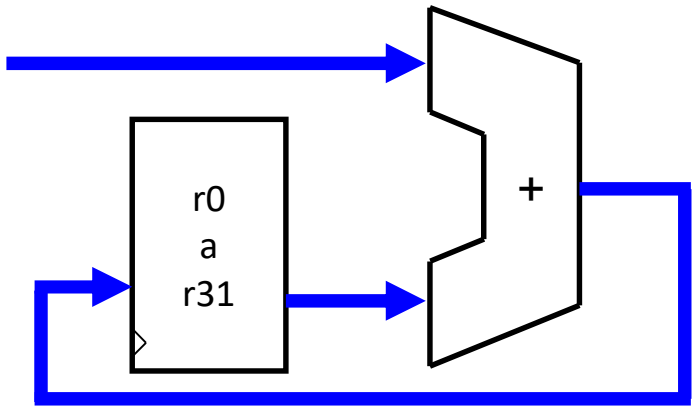
- ✓A + B;
- ✓C + D + E;
- ✓F + G + H + I;
- ✓…




Sirve un sumador de dos entradas si almacenamos resultados parciales en registros => banco de registros

- ✓$R_1 = R_0 + F$;
- ✓$R_2 = R_1 + G$;
- ✓$R_3 = R_2 + H$;
- ✓$R_4 = R_3 + I$;

Equivalente a:
$R_4 = F + G + H + I$;
(el reg. $R_0$ es siempre 0)



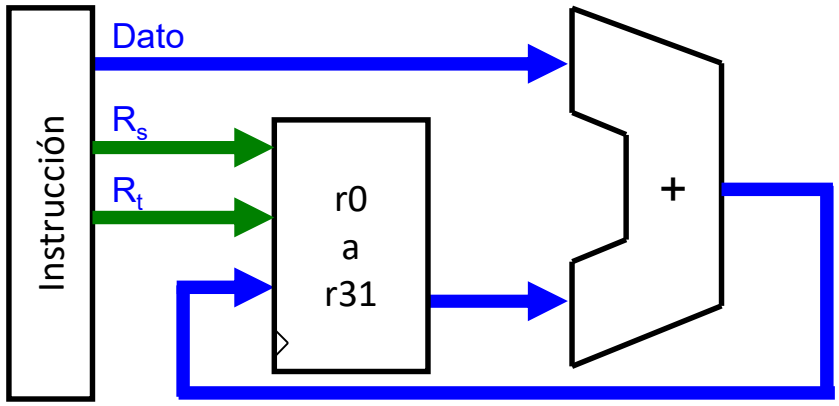
# Circuito sumador genérico

Por tanto, necesitamos un circuito capaz de realizar la siguiente “instrucción” (es un microprocesador muy simplificado):

- ✓ $R_t$ <= $R_s$ + Dato;

En cada “instrucción”, necesitamos darle la siguiente información:

- ✓ Número del registro destino ($R_t$), del 0 al 31 => 5 bits
- ✓ Número del registro fuente ($R_s$), del 0 al 31 => 5 bits
- ✓ Dato (dato inmediato) => 16 bits



# Circuito sumador genérico

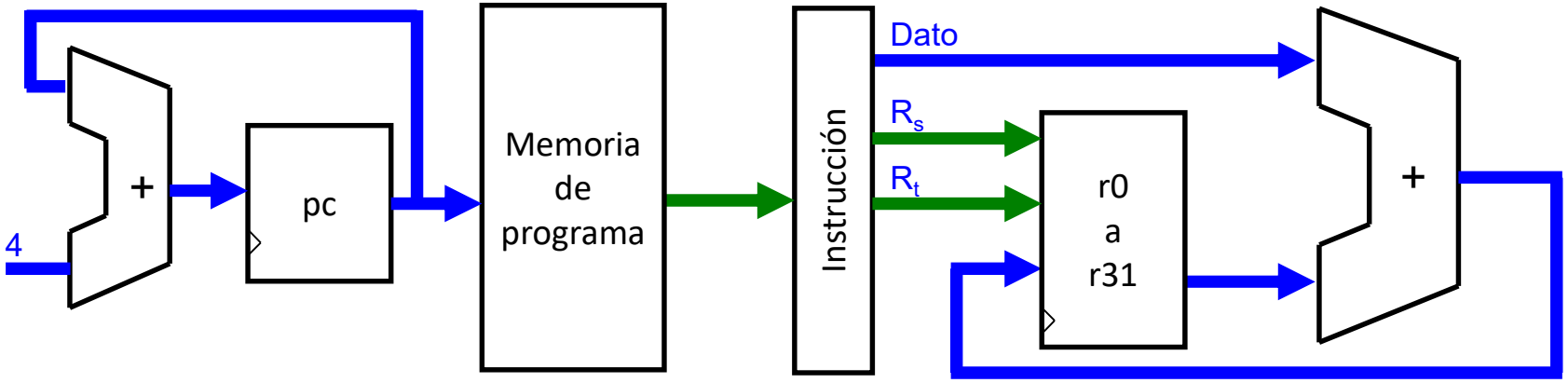
¿Cómo recibe el micro las instrucciones?

- ✓Se almacenan en una memoria (de programa)
- ✓El micro tiene que ser capaz de ir leyendo la memoria, instrucción tras instrucción

Cada instrucción necesita como mínimo 5+5+16 = 26 bits

- ✓Se ajusta a 32 bits (potencia de 2), por una decisión de diseño.
- ✓32 bits => 4 bytes.

**Una instrucción se almacena en una dirección de memoria 4 bytes más adelante que la anterior**



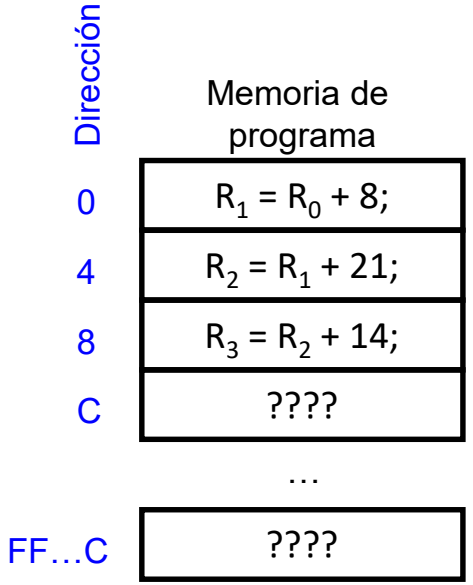
# Ejemplo de funcionamiento

Sumar 8 + 21 + 14. Se hace con un programa que tiene tres instrucciones:

- ✓$R_1 = R_0 + 8;$
- ✓$R_2 = R_1 + 21;$
- ✓$R_3 = R_2 + 14;$

El contador de programa (*program counter*, pc) indica la dirección de memoria de la instrucción actual.

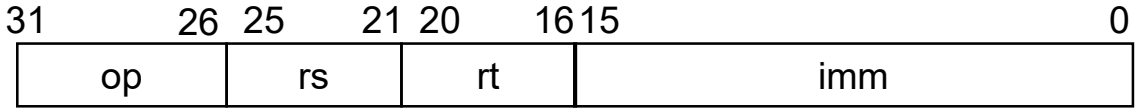
- ✓Empieza en 0 y sube 4 cada instrucción



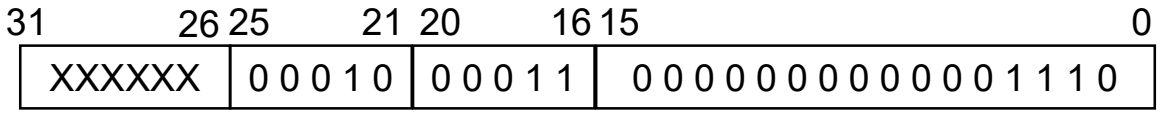
# Memoria de programa (instrucciones)

En cada posición de memoria se almacena una instrucción de 32 bits:

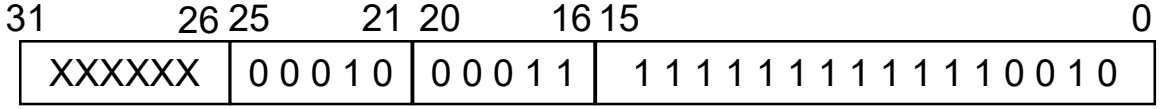
- ✓ 5 bits para el registro destino, $R_t$
- ✓ 5 bits para el registro fuente, $R_s$
- ✓ 16 bits para el dato inmediato
- ✓ Resto de bits no se usan. En los micros reales sirven para indicar el código de instrucción, *operation code* (op), ya que hay más de una instrucción de este mismo tipo




Ejemplo: R3 = R2 + 14;



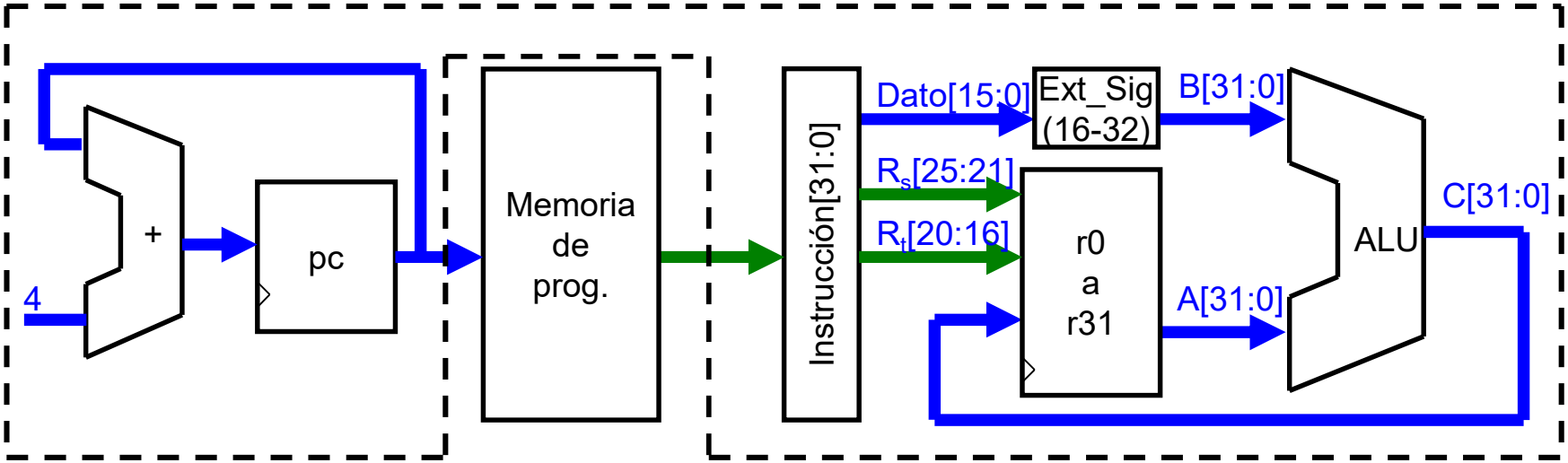

Ejemplo: R3 = R2 - 14;



# Ancho de palabra

Nuestro microprocesador utilizará datos de 32 bits:

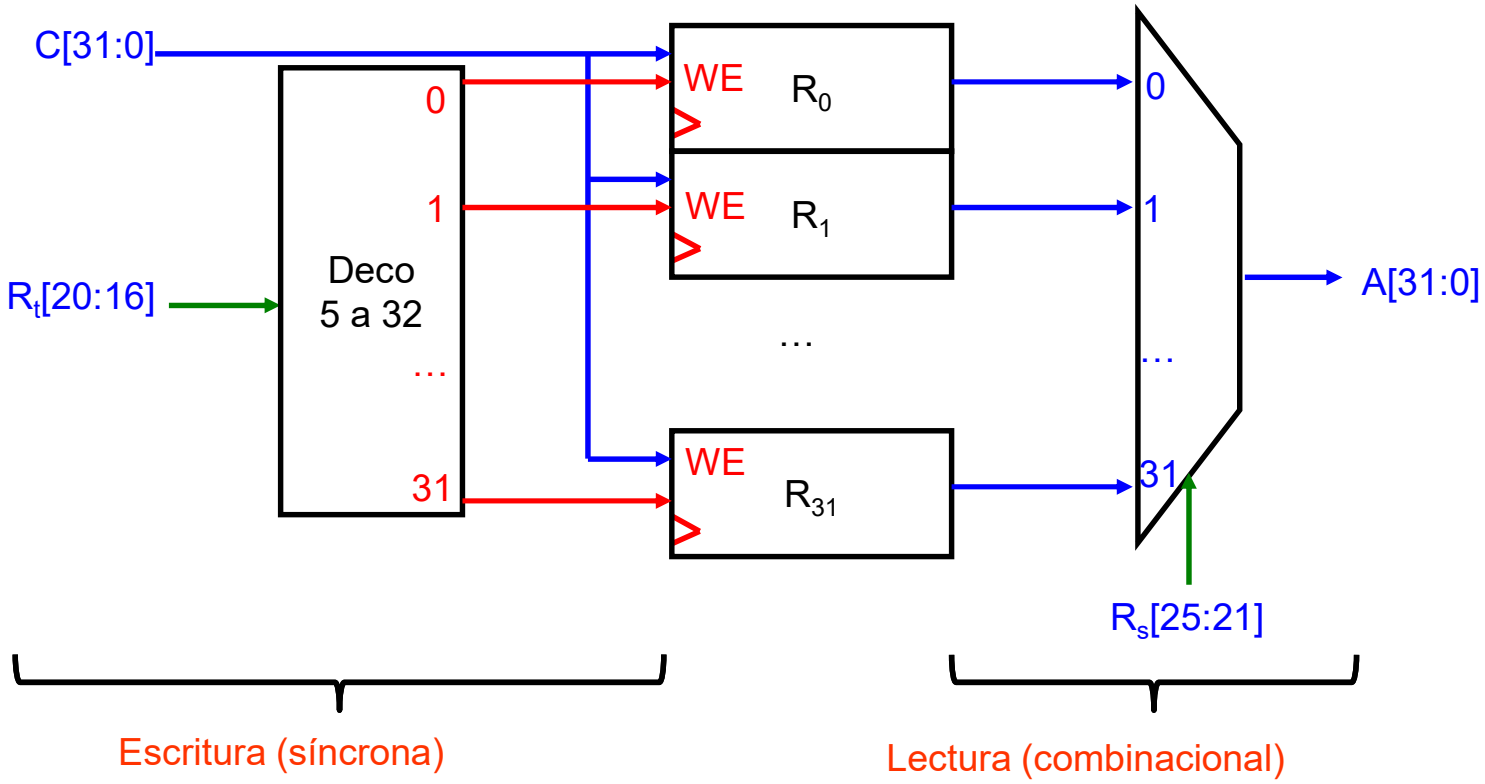
- ✓Las entradas y salida del sumador (ALU) son de 32 bits
- ✓Los registros del banco de registros son de 32 bits
- ✓Como el dato inmediato es de 16 bits, se extiende (con signo) a 32 bits



# Banco de Registros (GPR)

¿Cómo se realiza el banco de registros?

- ✓Básicamente, multiplexando 32 registros (cada uno de 32 bits)

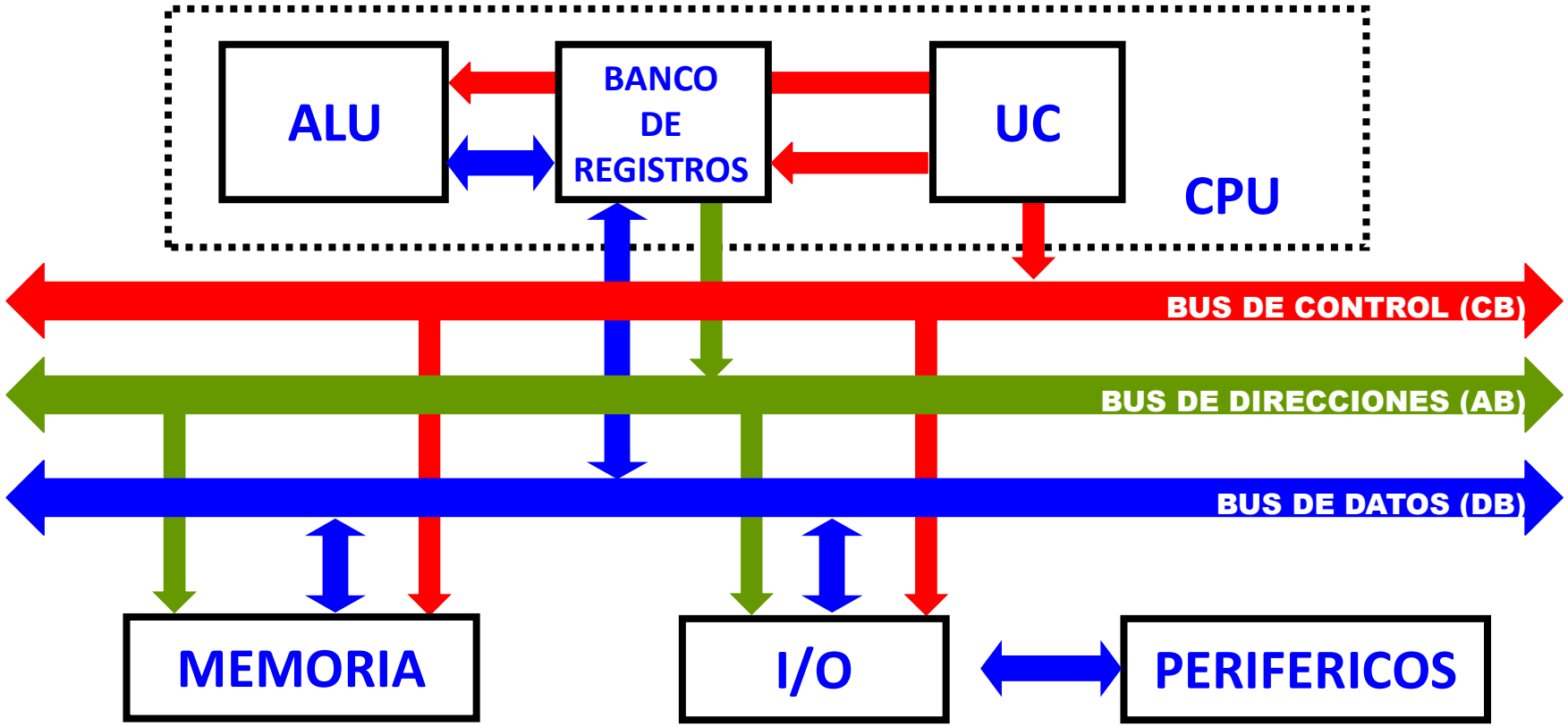


# Microprocesador Completo

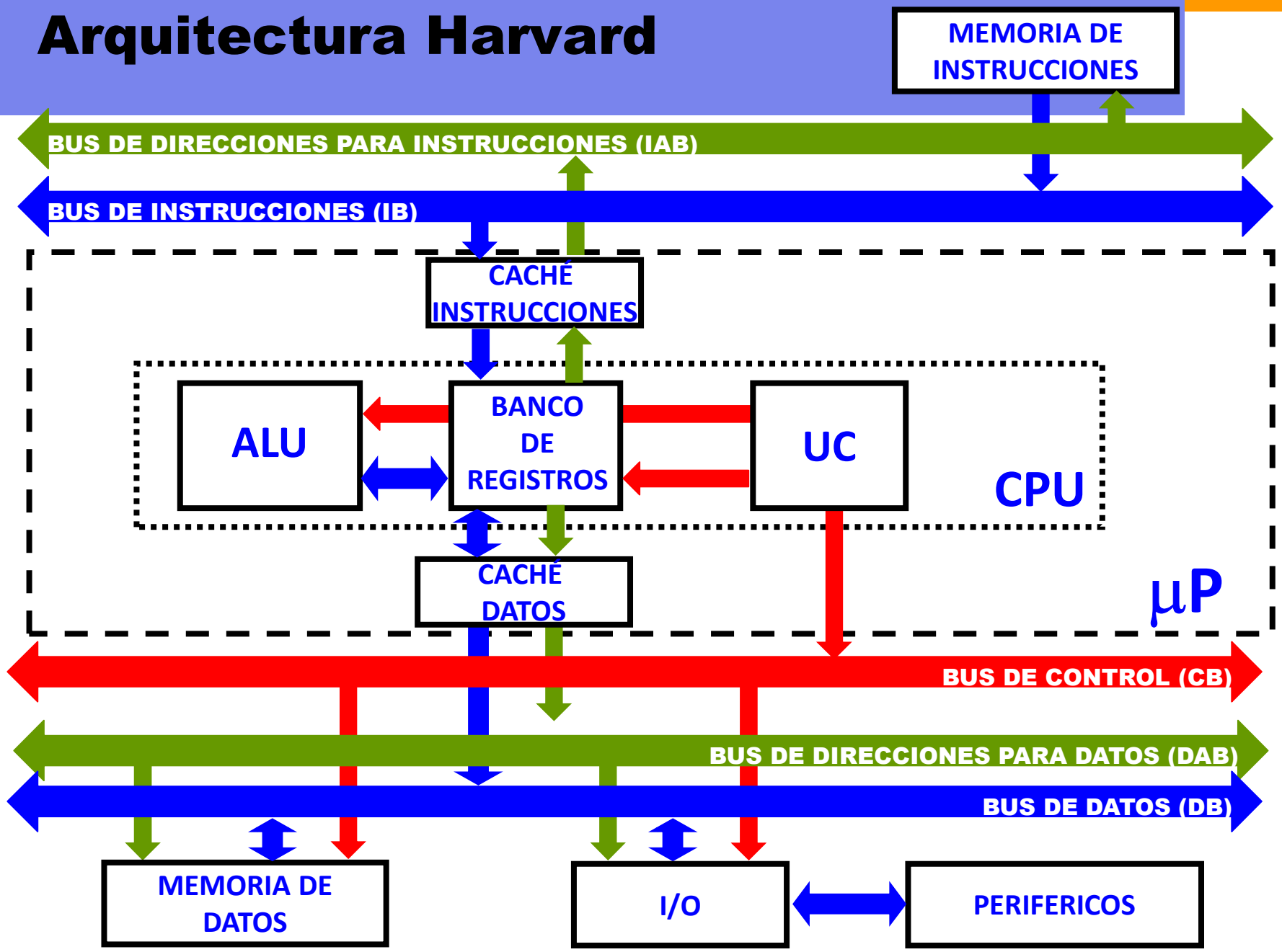
¿Qué le falta para ser un microprocesador completo?

- ✓Poder realizar otras operaciones (instrucciones aritmético-lógicas)
- ✓Poder usar más de 32 datos, y para ello se añade la memoria de datos (instrucciones con acceso a memoria de datos)
- ✓Poder variar la secuencia de ejecución para realizar bucles o control de flujo, como for, if, etc… (saltos condicionales e incondicionales)

# Arquitectura clásica Von Neumann



# Arquitectura Harvard

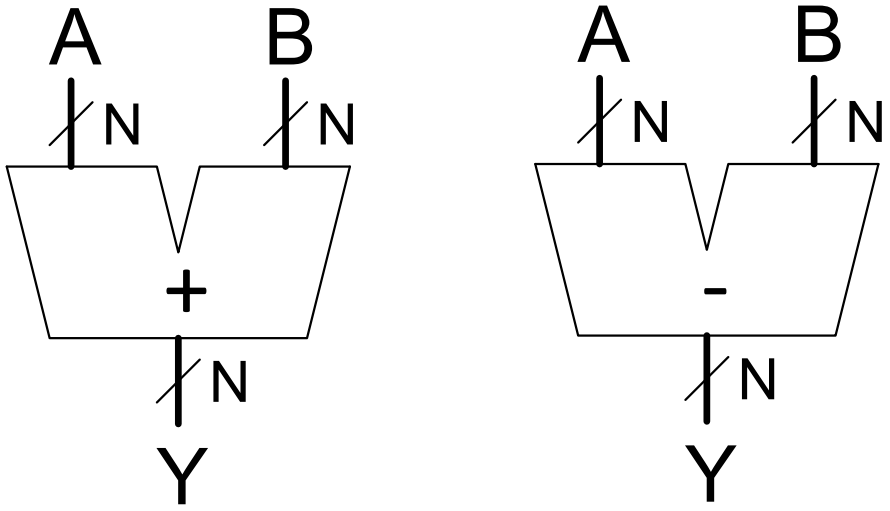


# Índice

- Estructura básica de un ordenador (sumador)
- **Circuitos lógicos y aritméticos**
  - ✓ Sumadores y Restadores
  - ✓ Desplazadores y Multiplicadores
  - ✓ Otros operadores
- Diseño de una ALU optimizada

# Sumadores/restadores multibit

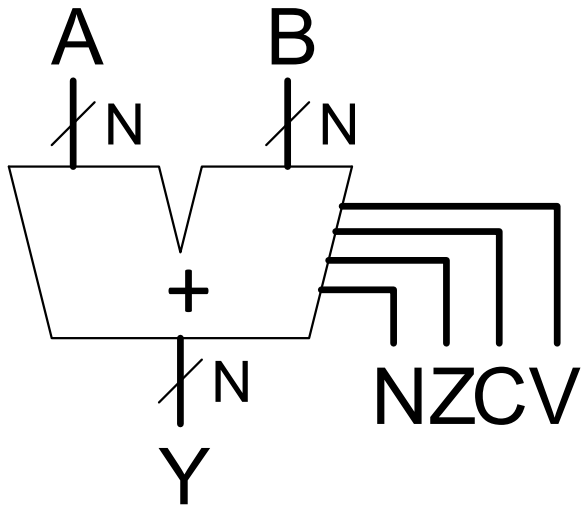
- Sumadores/restadores multibit por propagación del acarreo (*carry propagate, CPA*) de tres tipos:
  - Sumadores/restadores ripple-carry, RCA (lento)
  - Sumadores carry-lookahead, CLA (rápido)
  - Sumadores prefijo-paralelo, PPA (+ rápido)
- Los sumadores/restadores CLA y PPA son más rápidos, pero requieren más hardware.



# Banderas de la ALU

Señales de un bit que indican información adicional sobre el resultado:

- Negative (N): Activa si el resultado es negativo
- Zero (Z): Activa si el resultado es 0
- Carry (C): Activa si el resultado ha generado un acarreo.
- Overflow (V): Activa si el resultado ha generado un desbordamiento




2.1, 2.2, 2.3

# Desplazadores

- **Desplazador lógico (logical shifter):** desplaza el valor a izquierda o derecha y rellena con 0's.
  - ✓ Ex: 11001 >> 2 = 00110
  - ✓ Ex: 11001 << 2 = 00100
- **Desplazador aritmético (arithmetic shifter):** igual que el lógico, salvo que hacia la derecha rellena con el bit de signo (msb).
  - ✓ Ex: 11001 >>> 2 = **11**110
  - ✓ Ex: 11001 <<< 2 = 00100
- **Rotador (rotator):** rota a izquierda o derecha los bits en círculo, lo que sale por un lado entra por el otro.
  - ✓ Ex: 11001 ROR 2 = 01110
  - ✓ Ex: 11001 ROL 2 = 00111

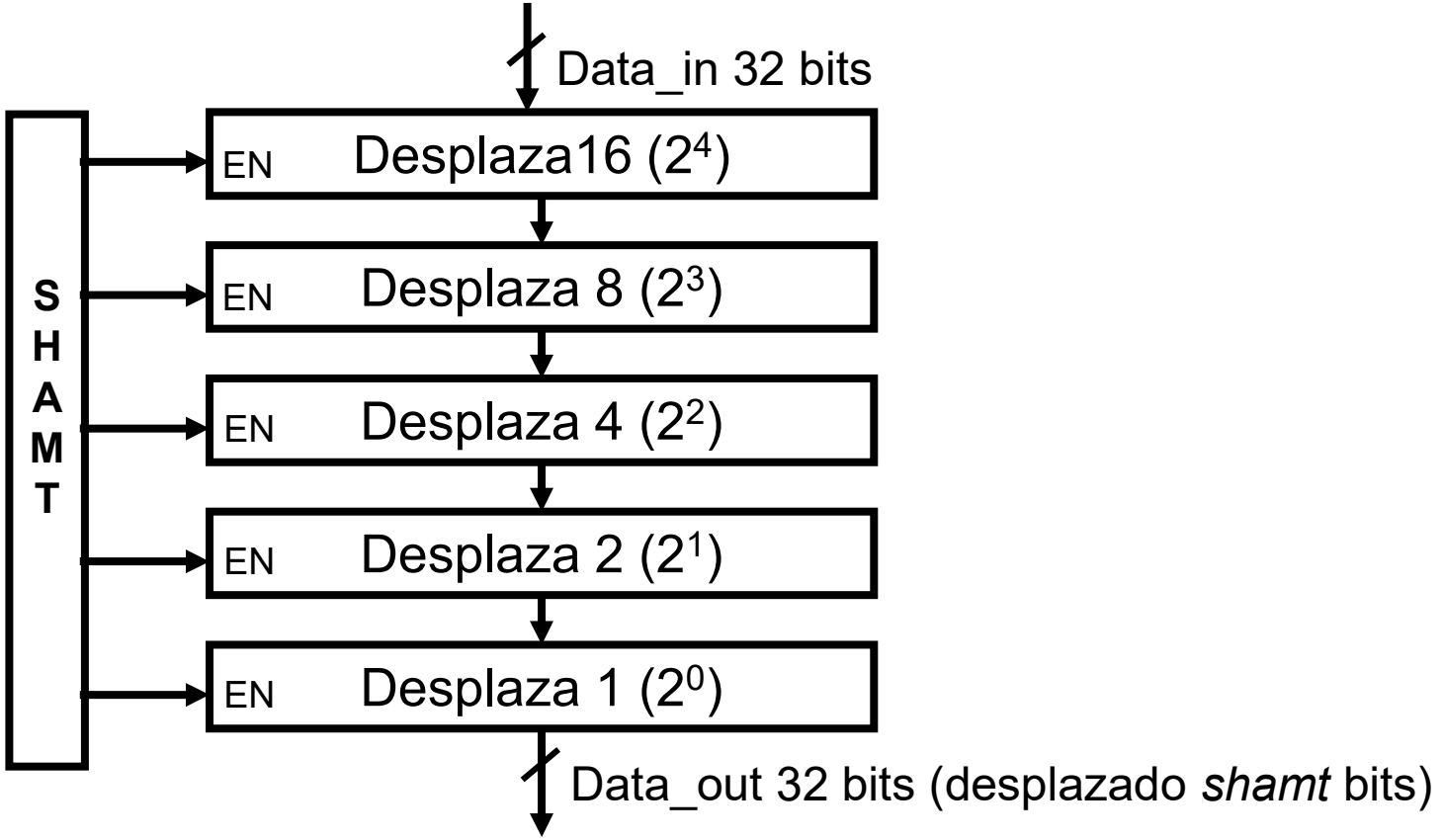
# Desplazar para multiplicar o dividir

- Un desplazamiento a izquierda de *N* bits equivale a multiplicar por $2^N$
  - ✓Ex: 00001 << 2 = 00100 $(1 \times 2^2 = 4)$
  - ✓Ex: 11101 << 2 = 10100 $(-3 \times 2^2 = -12)$

- Un desplazamiento aritmético a derecha de *N* bits equivale a dividir entre $2^N$
  - ✓Ex: 01000 >>> 2 = 00010 $(8 \div 2^2 = 2)$
  - ✓Ex: 10000 >>> 2 = 11100 $(-16 \div 2^2 = -4)$

# Desplazador en barril ( *Barrel Shifter* )

- ¿Cómo desplazar un número variable de posiciones, de 0 a 31?
  - ✓ En MIPS, dicha cantidad se codifica en $shamt_{4:0}$
- Usando desplazadores fijos $2^N$ en cadena
  - ✓ Cada desplazador se activa o no dependiendo de un bit en $shamt_{4:0}$



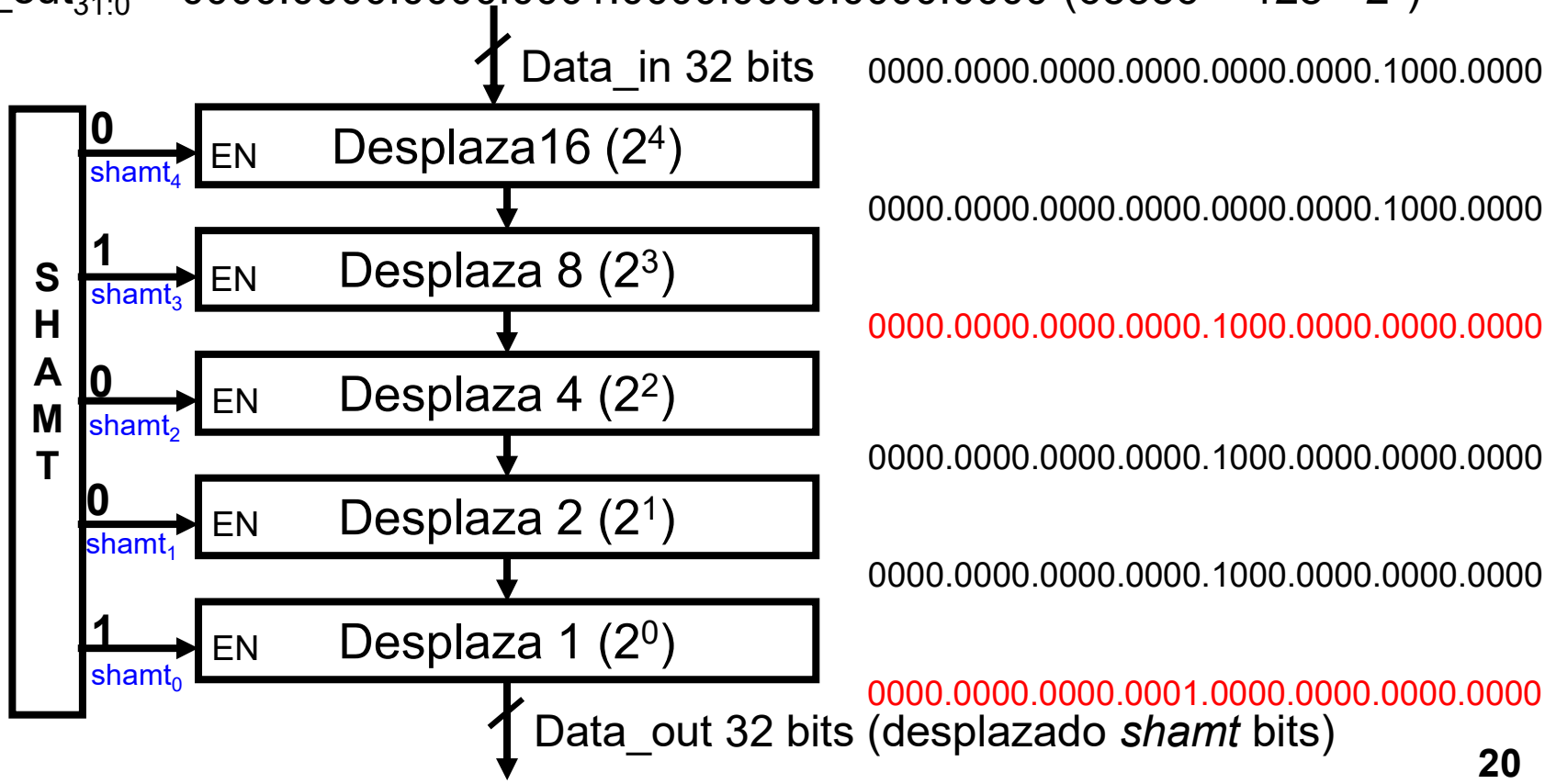
# Desplazador en barril ( *Barrel Shifter* )

Ejemplo:

$shamt_{4:0}$ = 01001 (9)

$Data\_in_{31:0}$ = 0000.0000.0000.0000.0000.0000.1000.0000 (128)

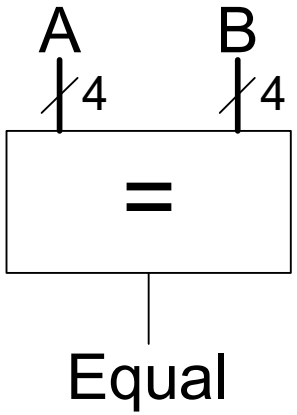
$Data\_out_{31:0}$ = 0000.0000.0000.0001.0000.0000.0000.0000 (65536 = 128 * $2^9$)



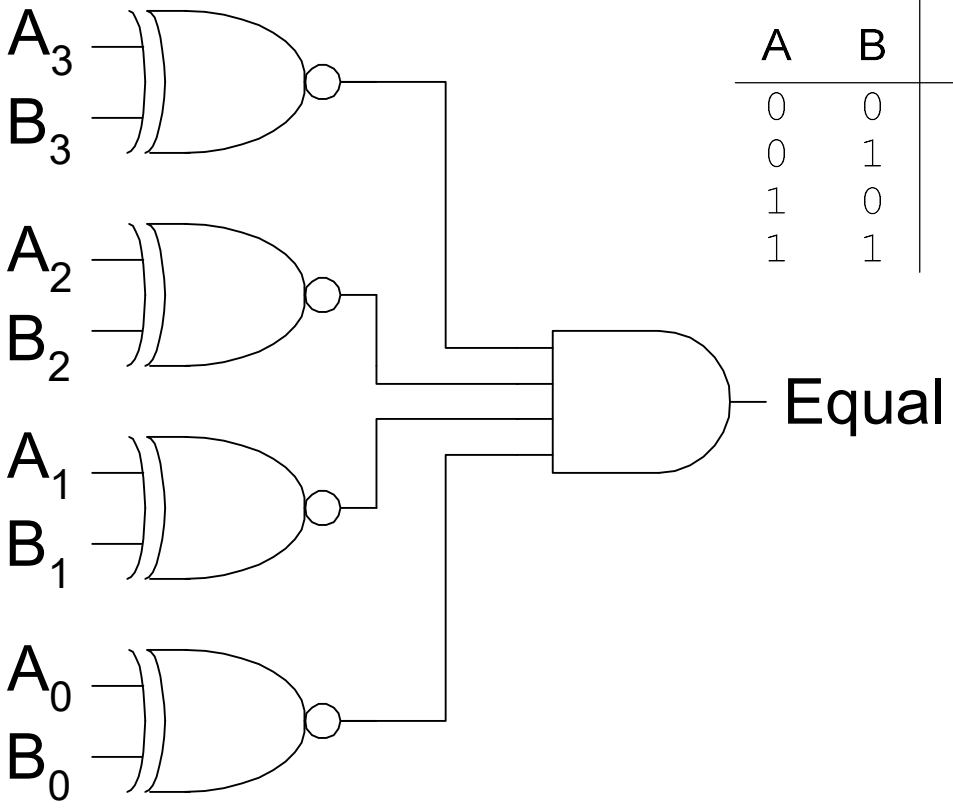
# Otros operadores

## Comparador igualdad (*Equal*): ¿ A = B ?

**Symbol**




**Implementation**



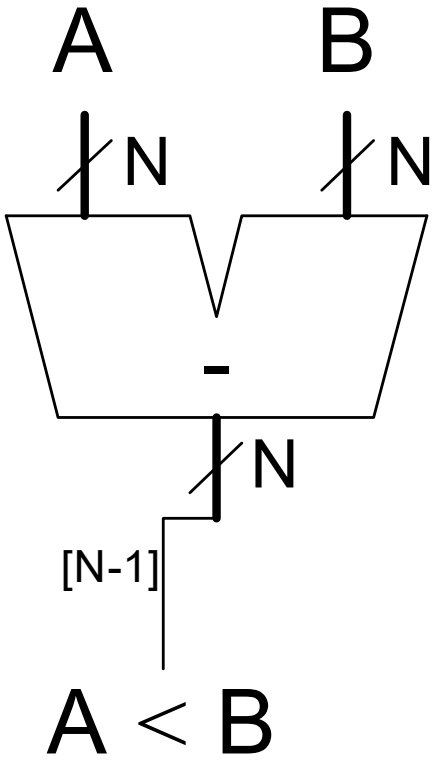

XNOR

| A | B | S |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Dos números son iguales cuando todos sus bits son iguales

# Otros operadores

**Comparador menor que (*Less Than*): ¿ A < B ?**



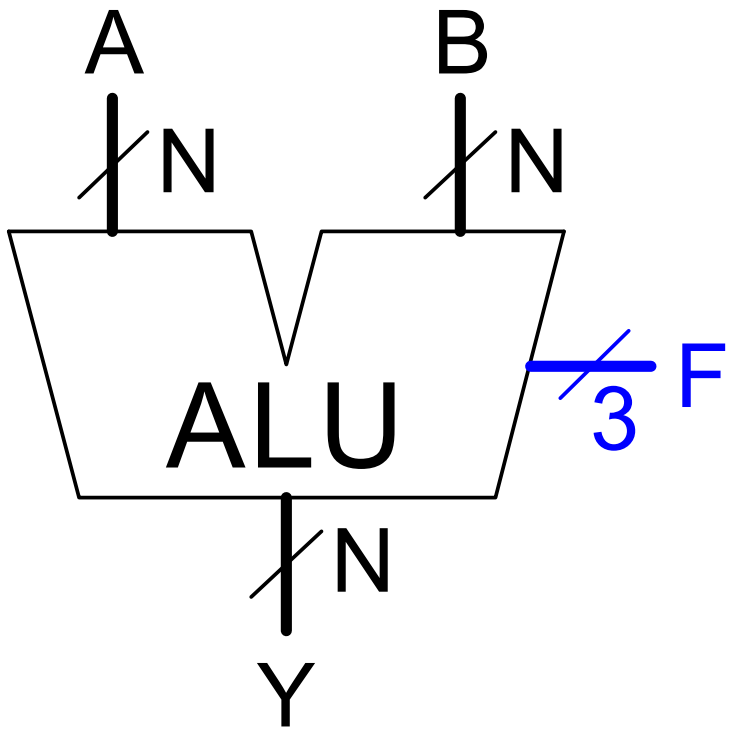

Para comprobar si un número es mayor/menor que otro, sólo hace falta restarlos y comprobar el signo del resultado (MSB)

# Índice

- Estructura básica de un ordenador (sumador)
- Circuitos lógicos y aritméticos
  - ✓ Sumadores y Restadores
  - ✓ Desplazadores y Multiplicadores
  - ✓ Otros operadores
- **Diseño de una ALU optimizada**

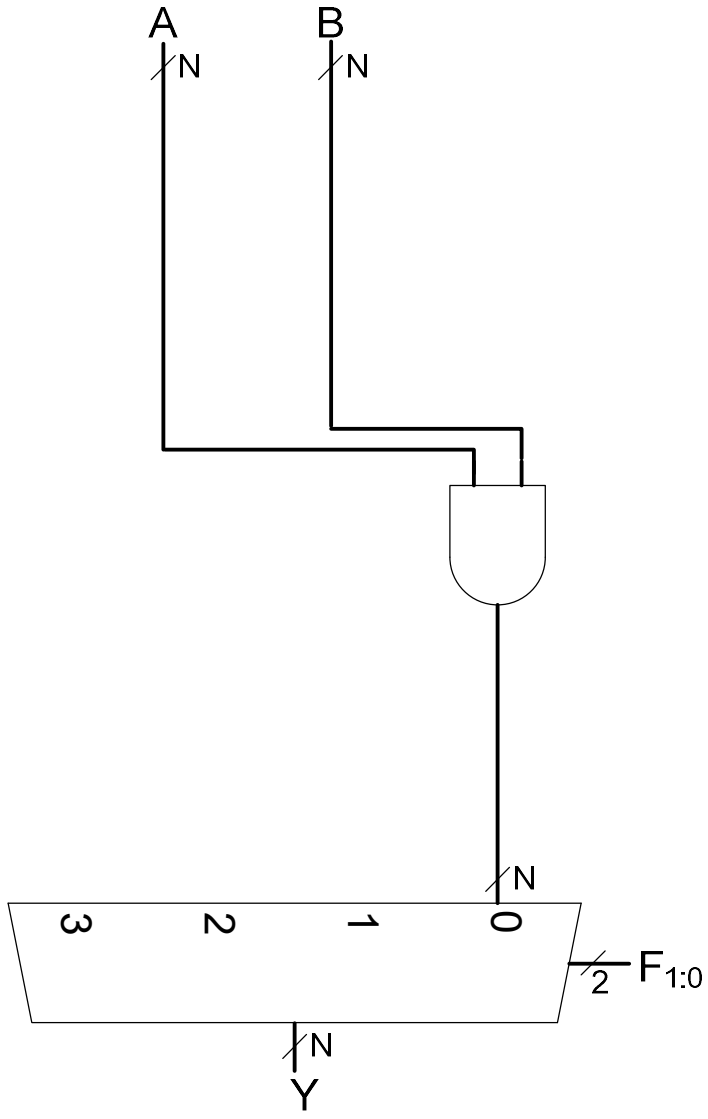
# Arithmetic Logic Unit (ALU)

$F_{2:0}$ = Selector de función



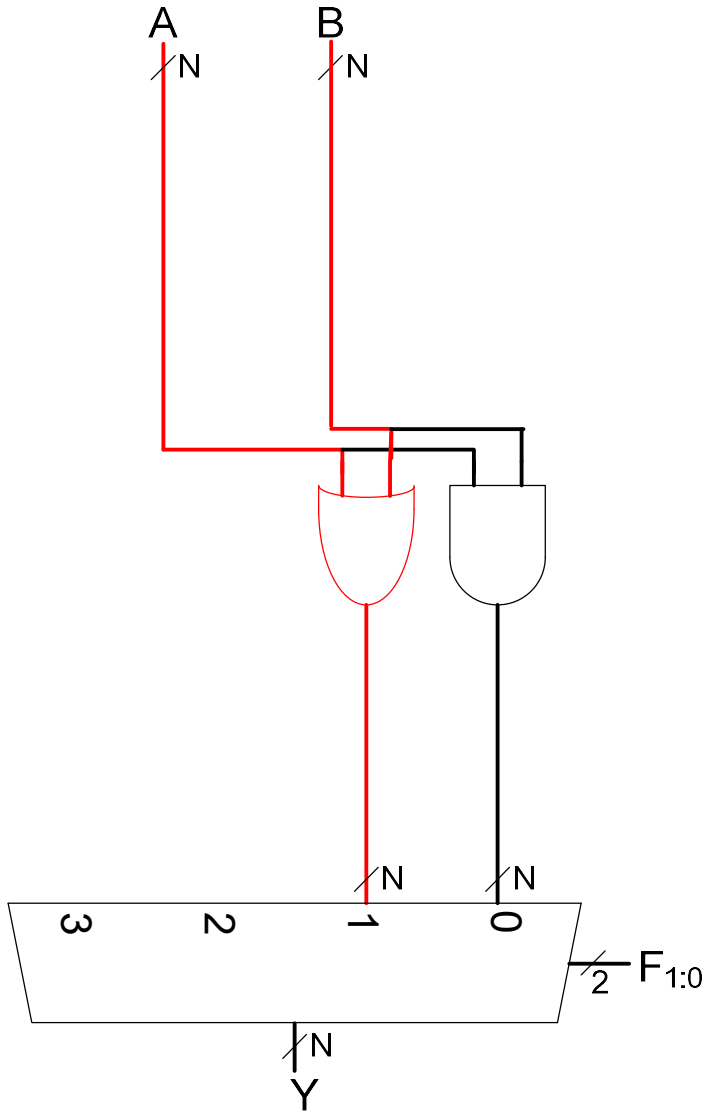

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU



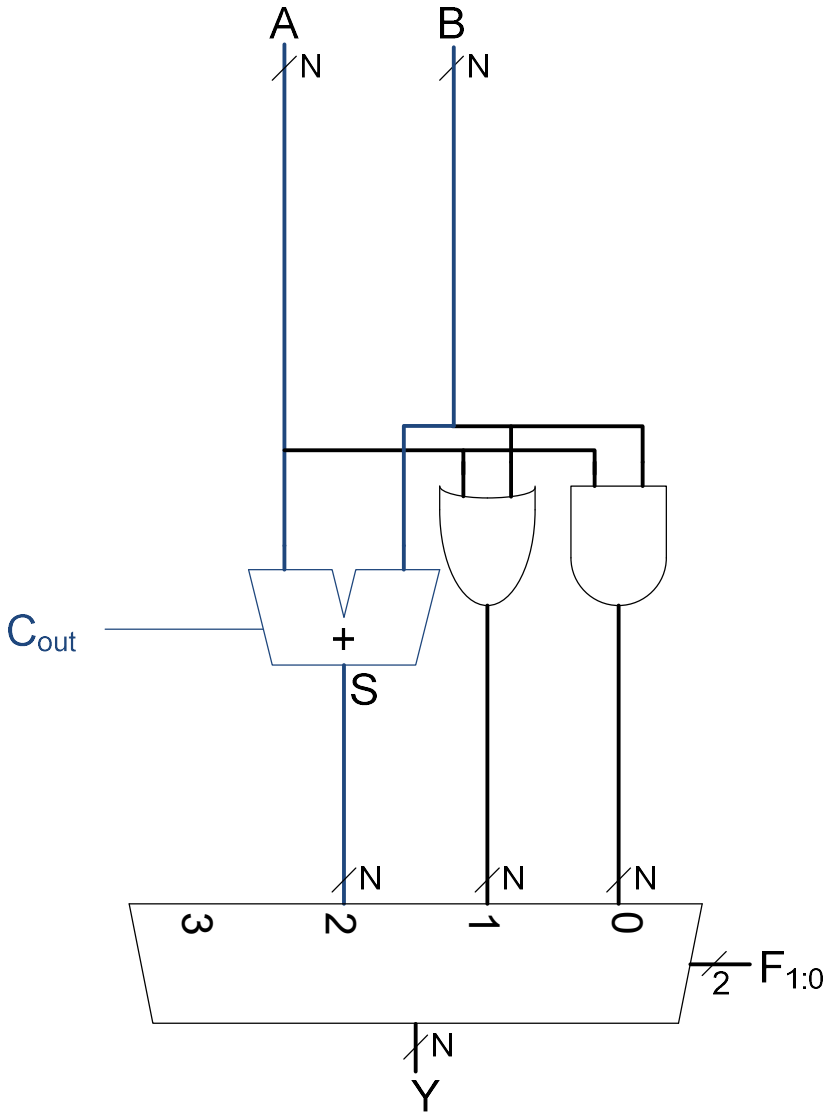

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU



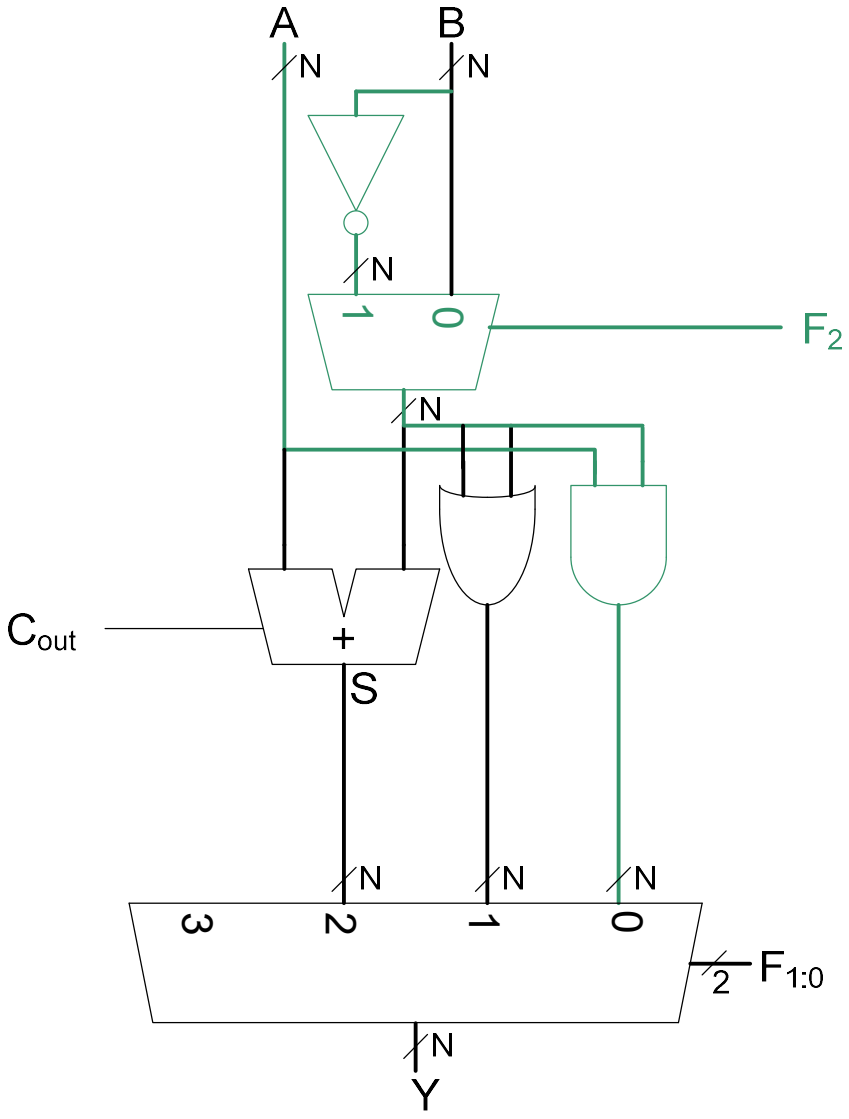

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU



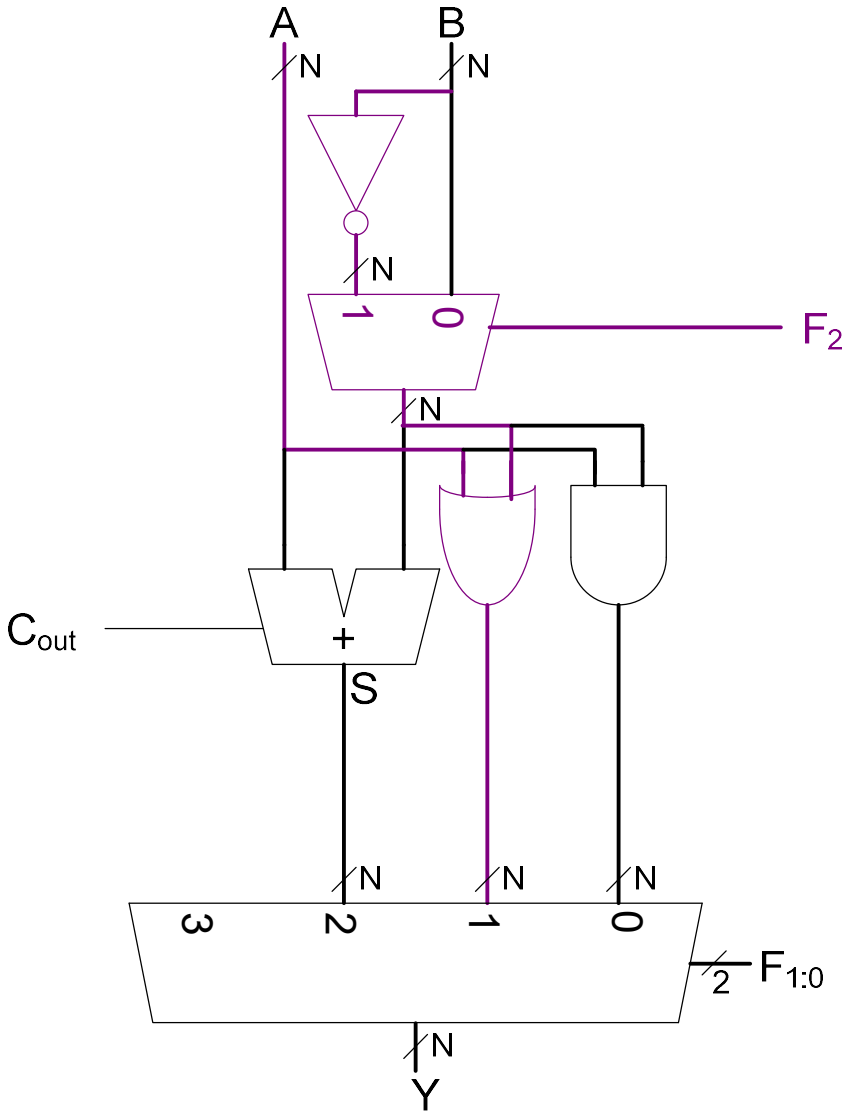

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU



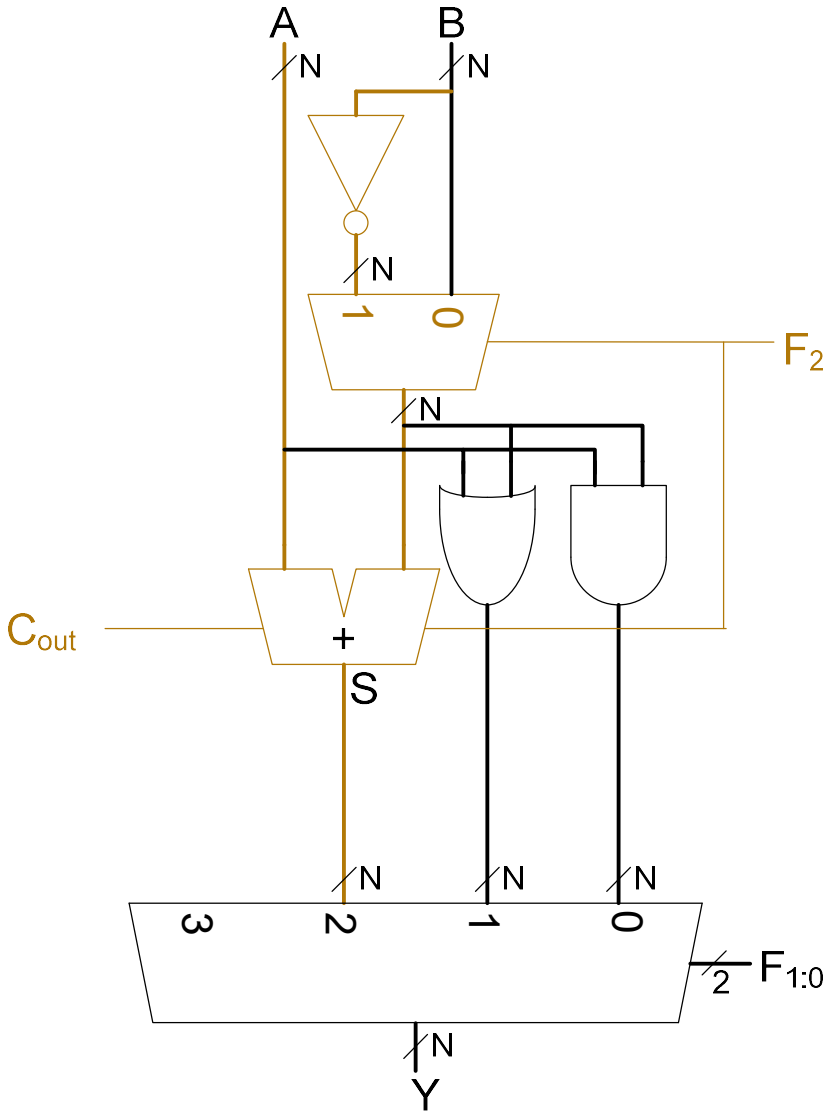

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU



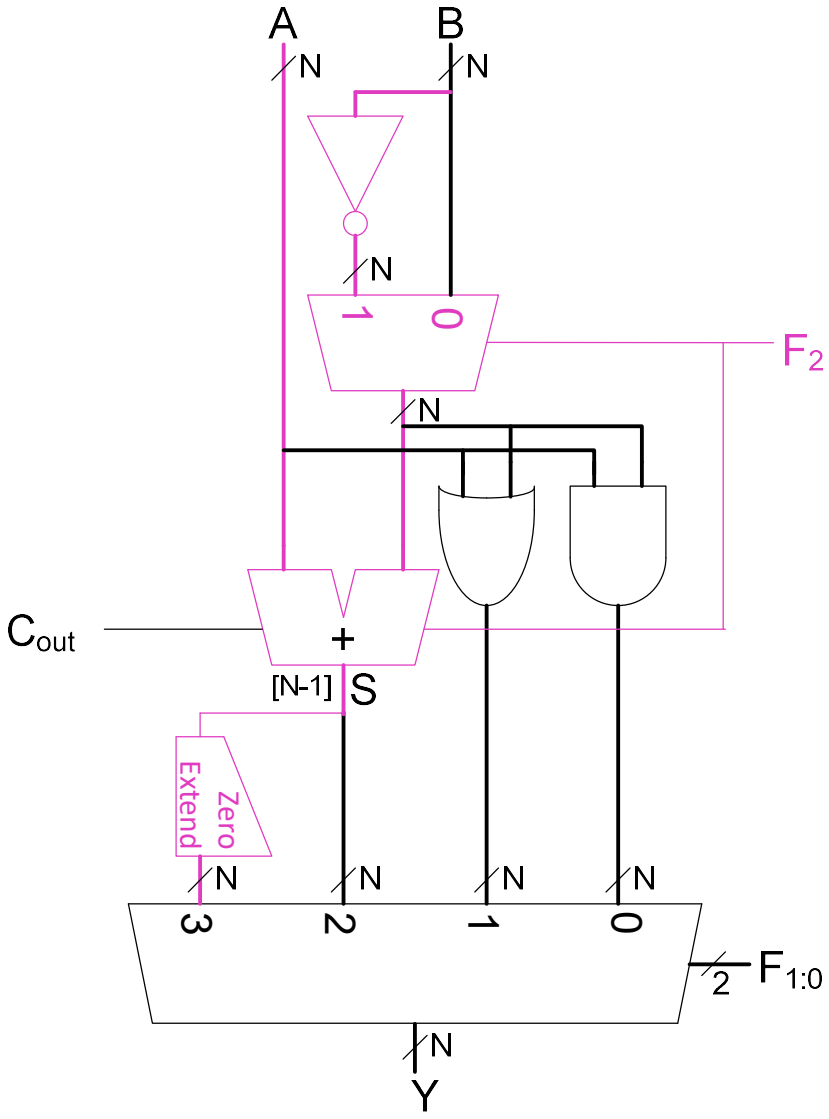

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU



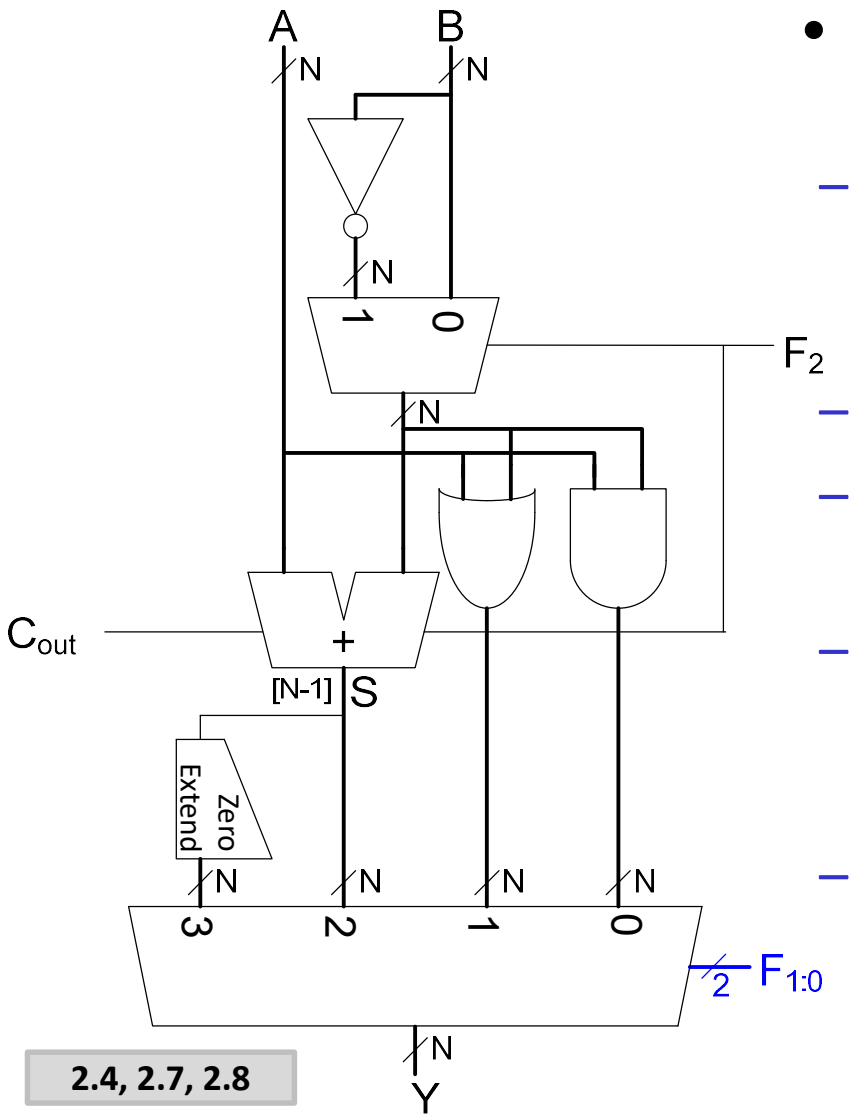

| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Diseño de la ALU




| $F_{2:0}$ | Función |
|---|---|
| 000 | A and B |
| 001 | A or B |
| 010 | A + B |
| 011 | Sin usar |
| 100 | A and /B |
| 101 | A or /B |
| 110 | A – B |
| 111 | SLT |

# Ejemplo de “Set Less Than” (SLT)




2.4, 2.7, 2.8

- El resultado es 1 si A<B y 0 en caso contrario. Suponemos $A$ = 25 y $B$ = 32.
  - A es menor que B, así que esperamos que Y sea la representación en 32 bits de 1 (0x00000001).
  - Para SLT, $F_{2:0}$ = 111.
  - $F_2$ = 1 hace que el sumador haga la resta. Así que 25 - 32 = -7.
  - La representación en C2 de -7 tiene un 1 en el “most significant bit”, así que $S_{31}$ = 1.
  - Los bits $F_{1:0}$ = 11, así que el mux elige $Y = S_{31}$ (zero extended) = 0x00000001.

# Fundamentos de Microprocesadores

## *Unidad 2: La Unidad Aritmético Lógica (ALU)*

Escuela Politécnica Superior - UAM