Estructura de Computadores

Unidad 2: La Unidad Aritmético Lógica (ALU)

Escuela Politécnica Superior - UAM

Índice

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

Introducción

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

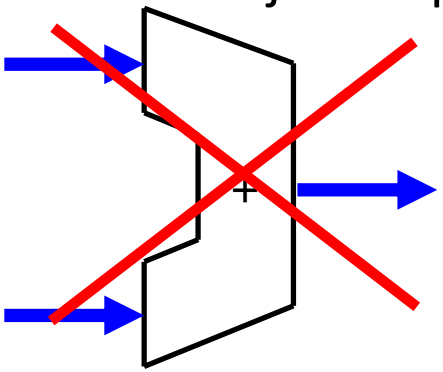
- Bloques que usaremos:
  - ✓ Multiplexores, decodificadores, registros y memorias, circuitos lógicos y aritméticos, etc…

- 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):

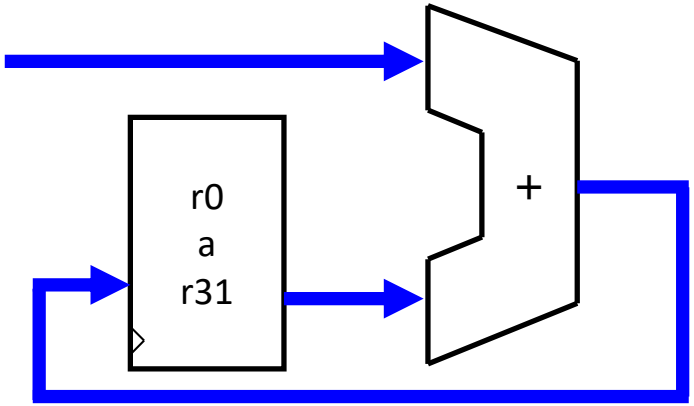
- ✓A + B;
- ✓C + D + E;
- ✓F + G + H + I;
- ✓…



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)



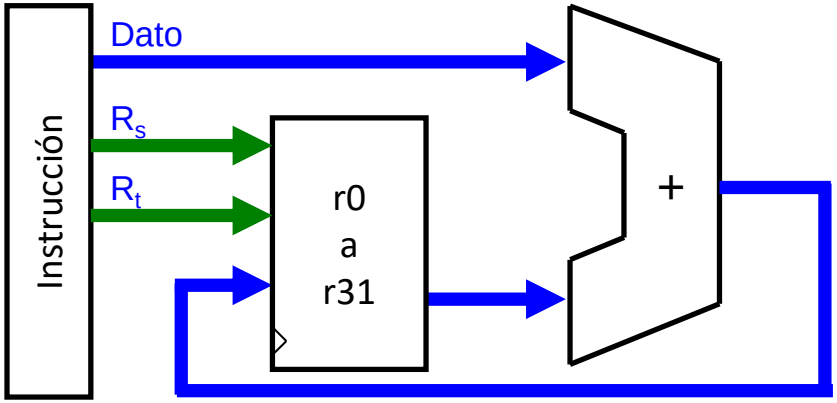
Circuito sumador genérico

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

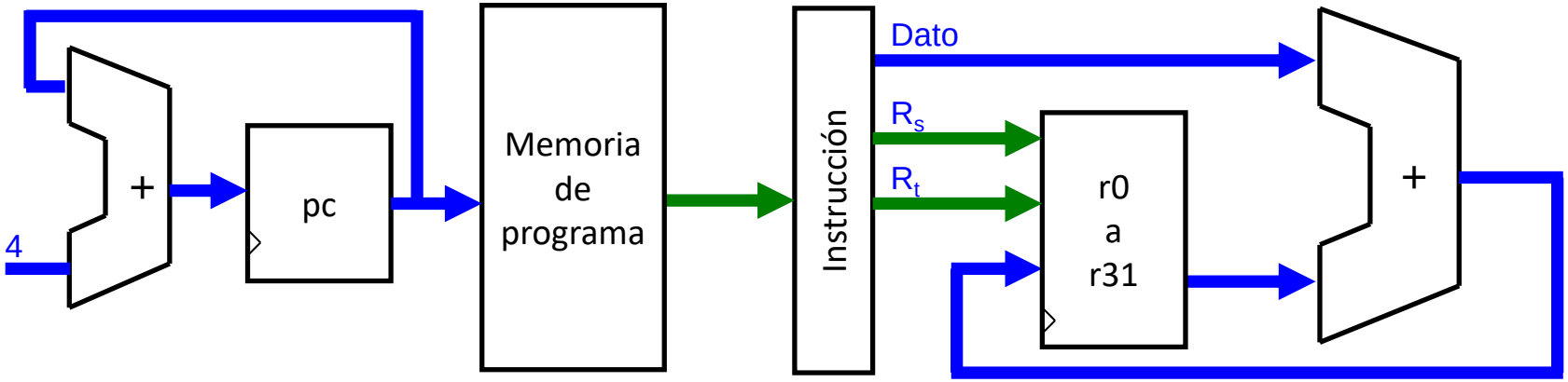
¿Cómo recibe el micro las instrucciones?

- ✓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.

Cada dirección se guarda en un dirección de memoria 4 bytes más adelante



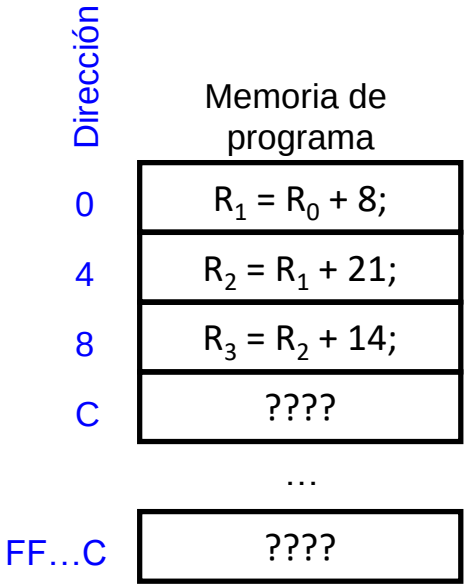
Ejemplo de funcionamiento

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.

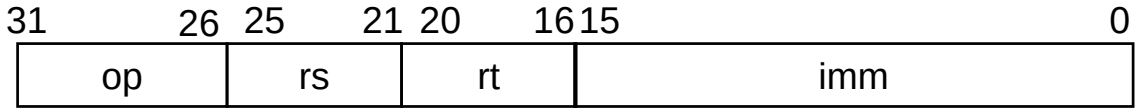
- ✓ Empieza en 0 y sube 4 cada instrucción



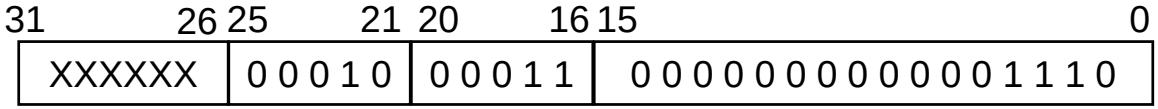
Memoria de programa (instrucciones)

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

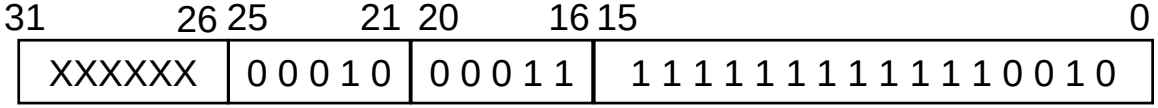
- ✓ 5 bits para el registro destino, R_t
- ✓ 5 bits para el registro fuente, R_s
- ✓ 16 bits para el dato inmediato
- ✓ Resto de bits no se usan. En los micros reales sirven para indicar el código de instrucción, *operation code* (op), ya que hay más de una instrucción de este mismo tipo



Ejemplo: R3 = R2 + 14;



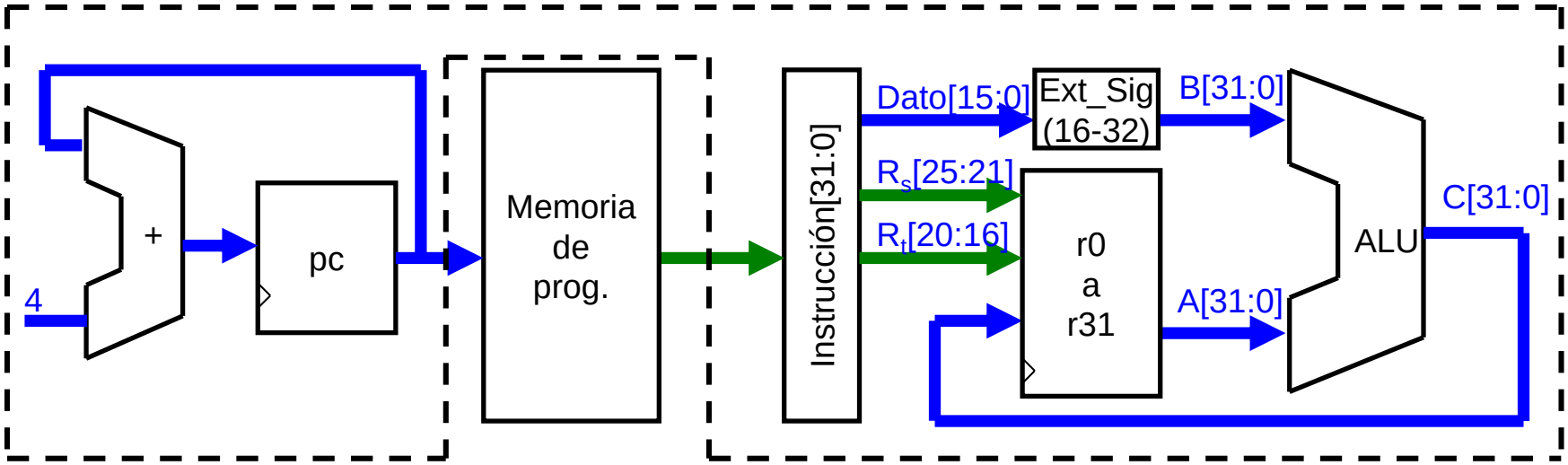
Ejemplo: R3 = R2 - 14;



Ancho de palabra

Nuestro microprocesador utilizará datos de 32 bits:

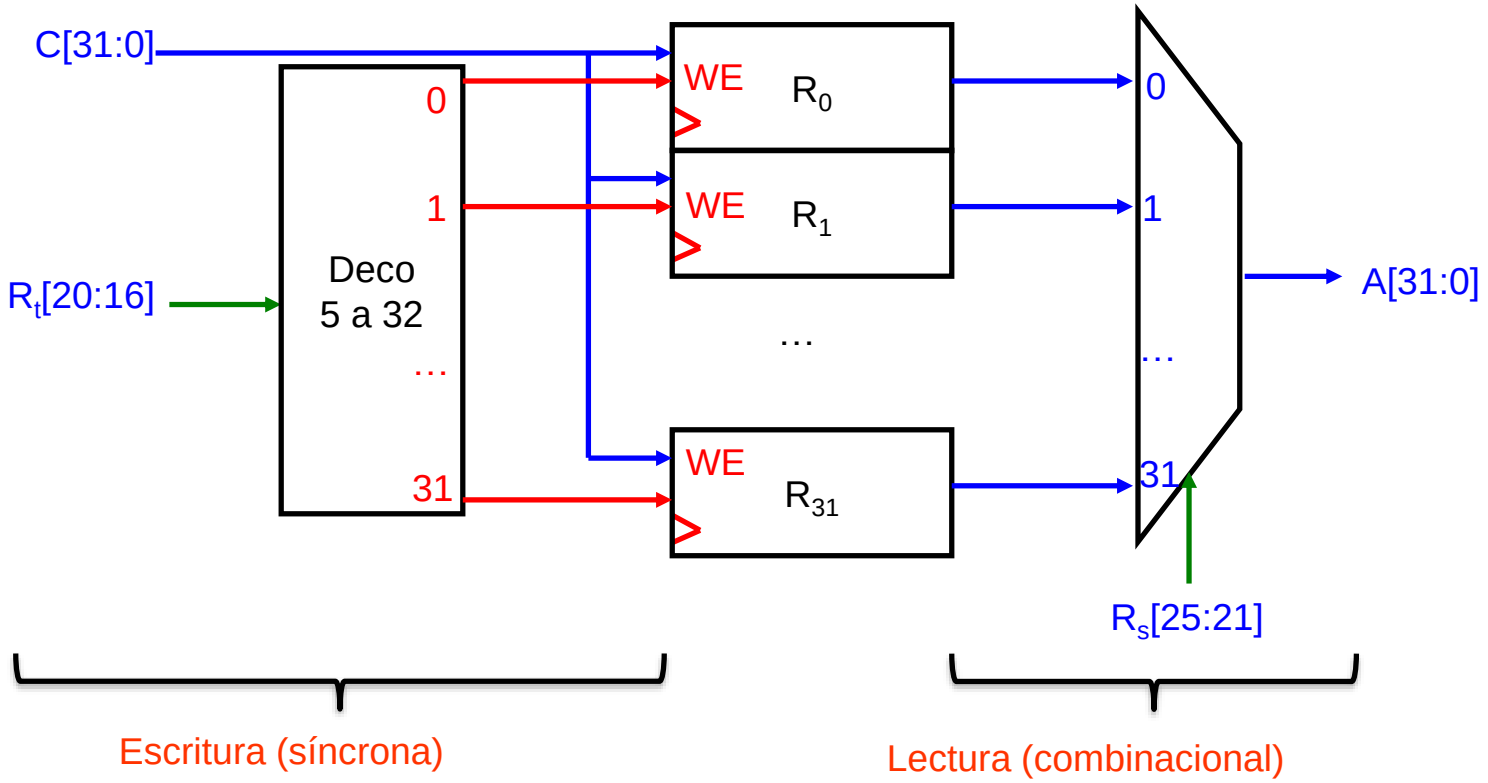
- ✓Las entradas y salida del sumador (ALU) son de 32 bits
- ✓Los registros del banco de registros son de 32 bits
- ✓Como el dato inmediato es de 16 bits, se extiende (con signo) a 32 bits



Banco de Registros (GPR)

¿Cómo se realiza el banco de registros?

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

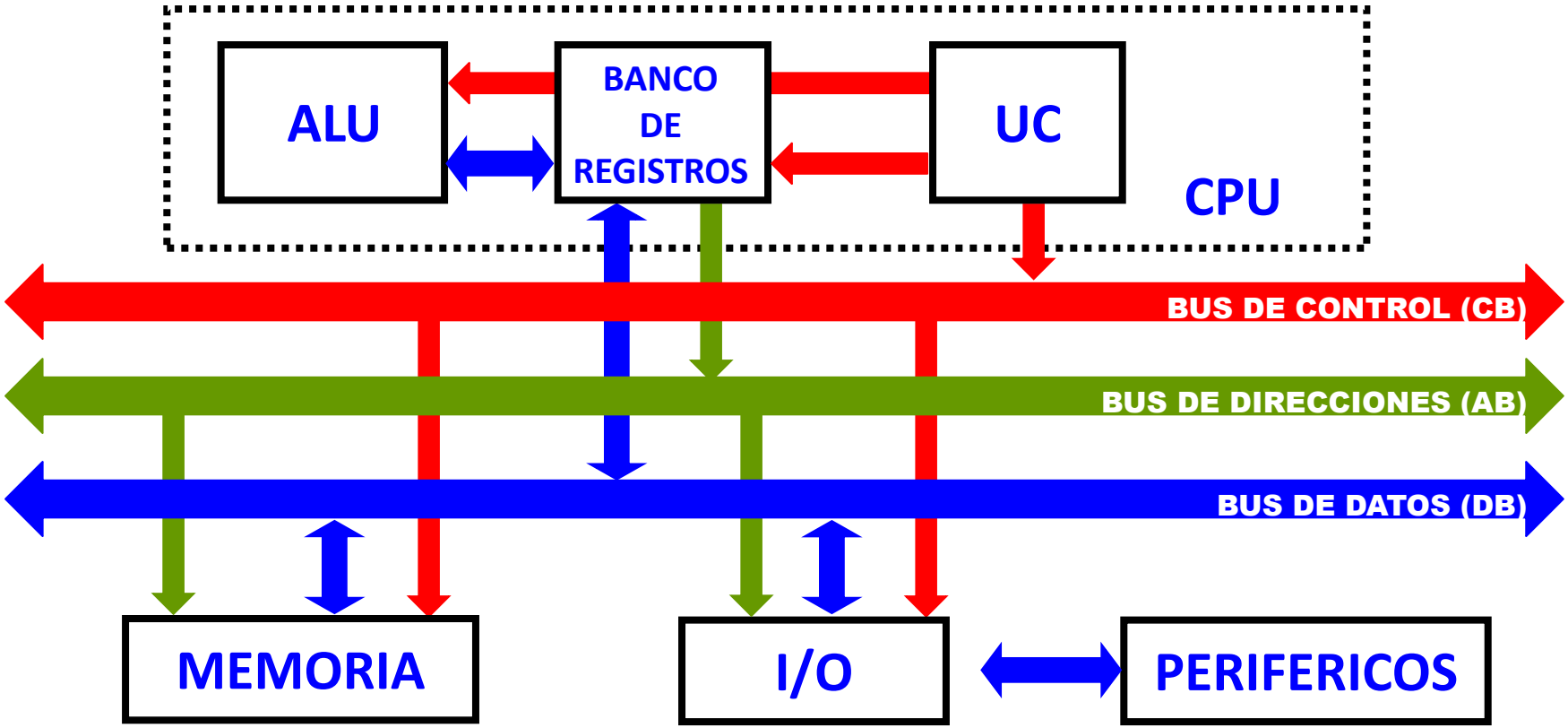


Microprocesador Completo

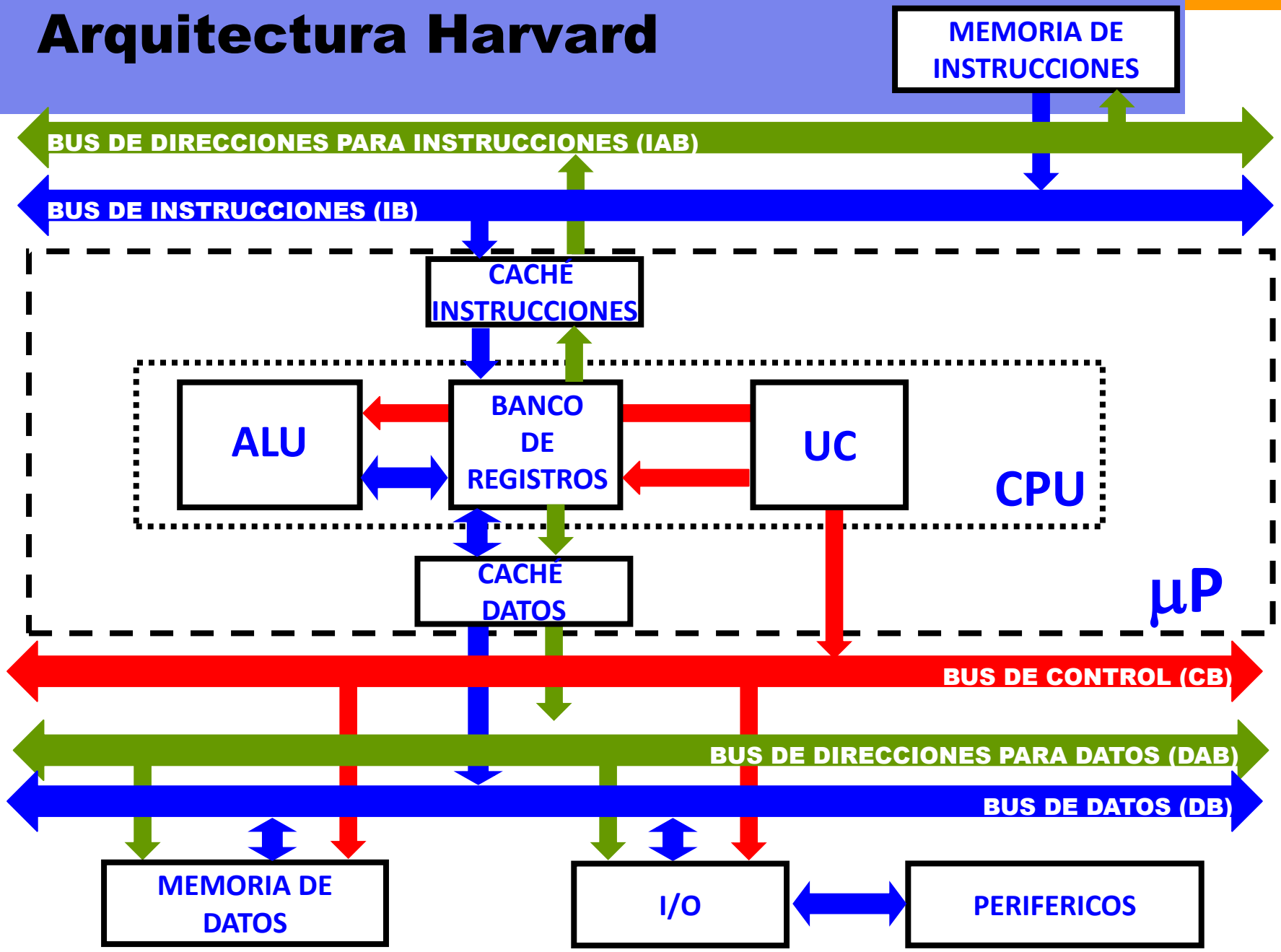
¿Qué le falta para ser un microprocesador completo?

- ✓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

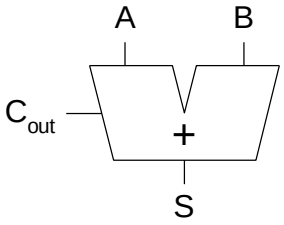


Índice

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

Sumadores de 1 bit

Half Adder

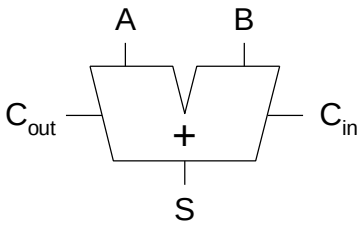


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

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

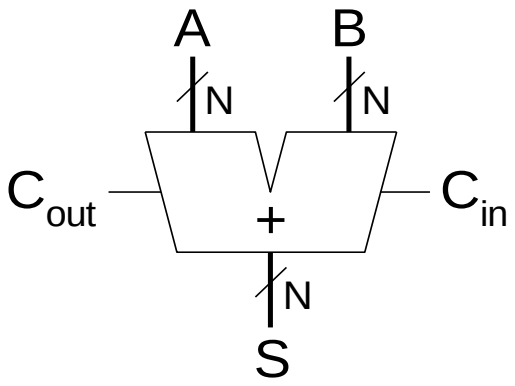
$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + BC_{in} + AC_{in}$

Sumadores multibit

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

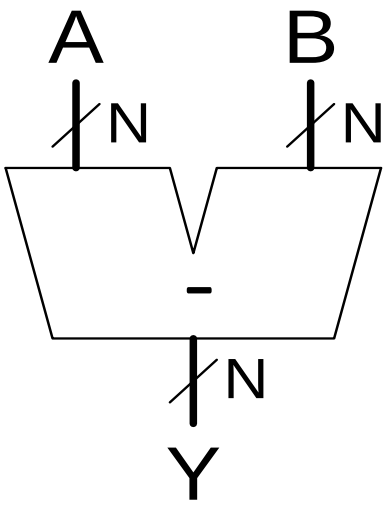
Symbol



Otros operadores

Restador: Y = A - B

Symbol



Recuerda

(Circuitos Electrónicos Digitales)

$Y = A-B = A +(-B)$

¿Cómo calcular el inverso aditivo de un número en complemento a 2? Haciendo la operación **NOT**, y después **sumarle 1**.

Ej: $Y=8_{10}-5_{10}=3_{10}= 01000_2 - 00101_2=?$

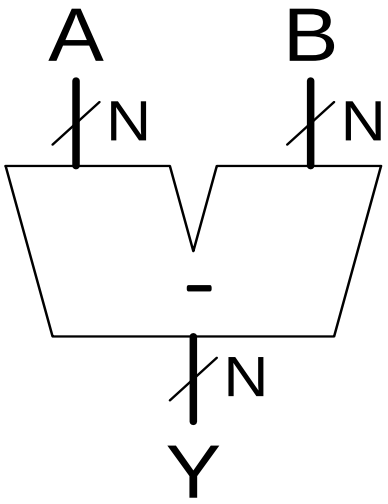
$Inverso(00101_2)$= NOT(00101)+1 =
11010+1 = 11011

$Y=A+(-B) = 01000_2+11011_2 = 00011_2$

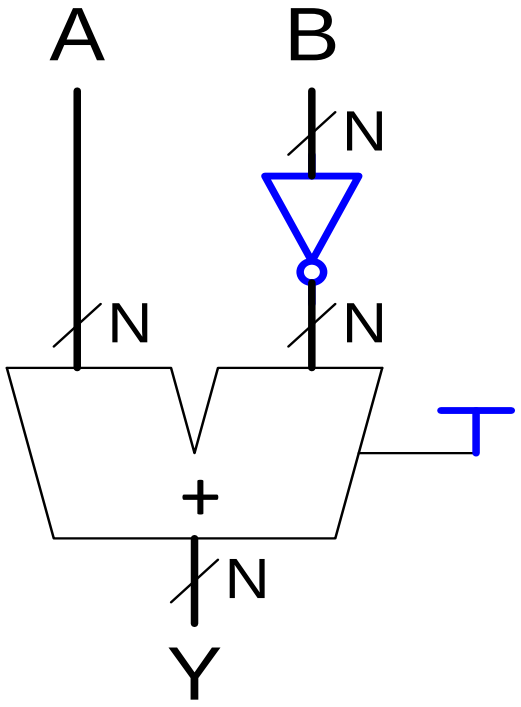
Otros operadores

Restador: Y = A - B

Symbol



Implementation

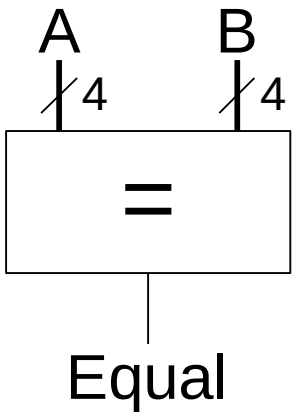


P

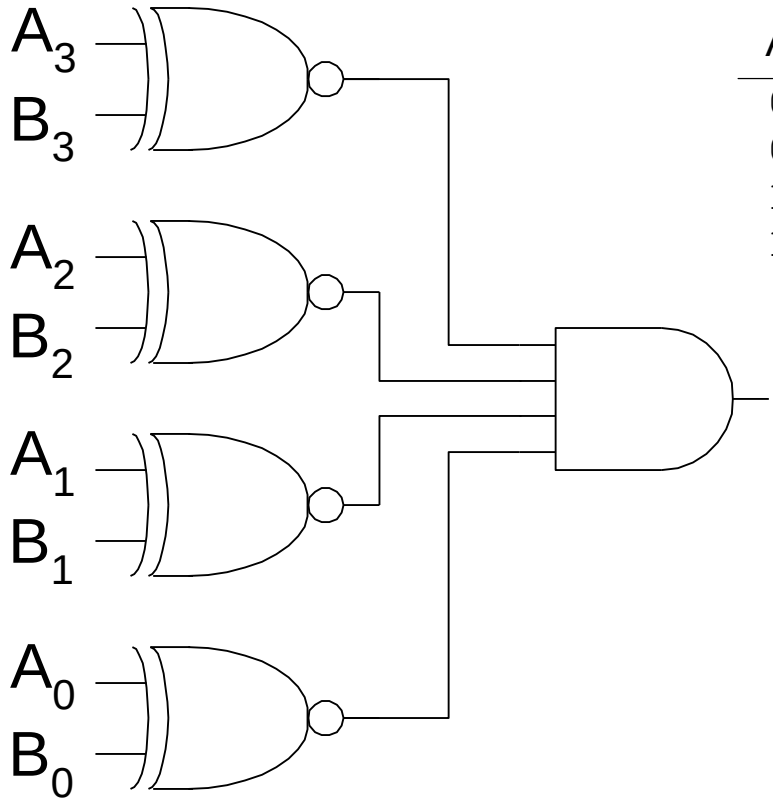
Otros operadores

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

Symbol



Implementation



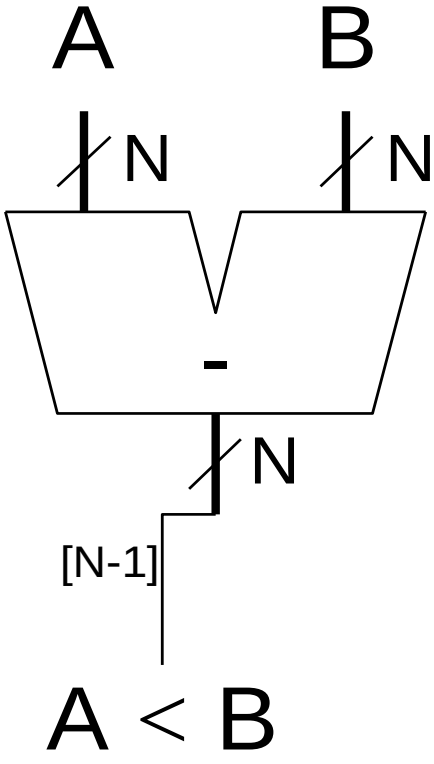
XNOR

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)

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

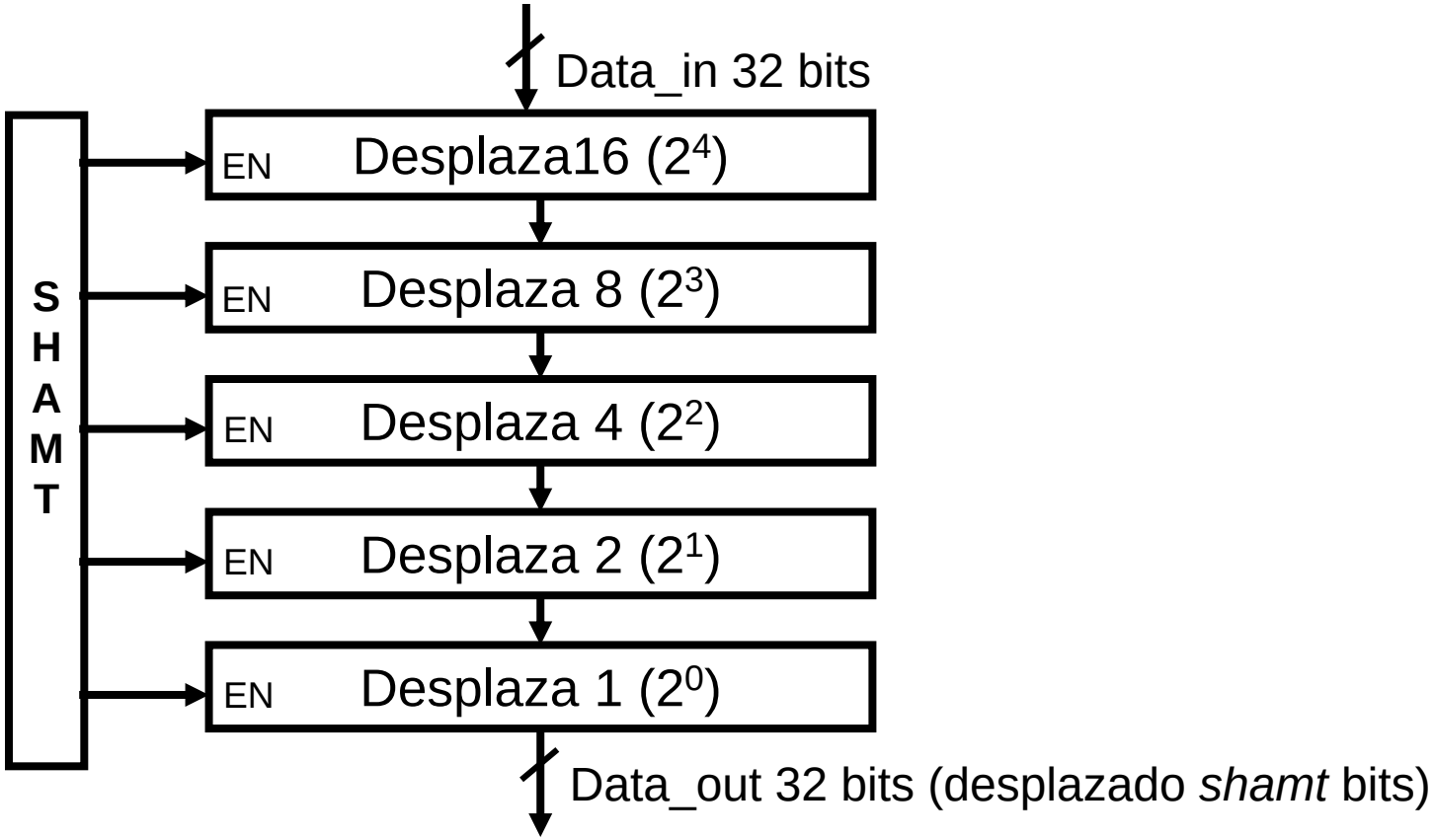
Desplazar para multiplicar o dividir

- 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 $shamt_{4:0}$



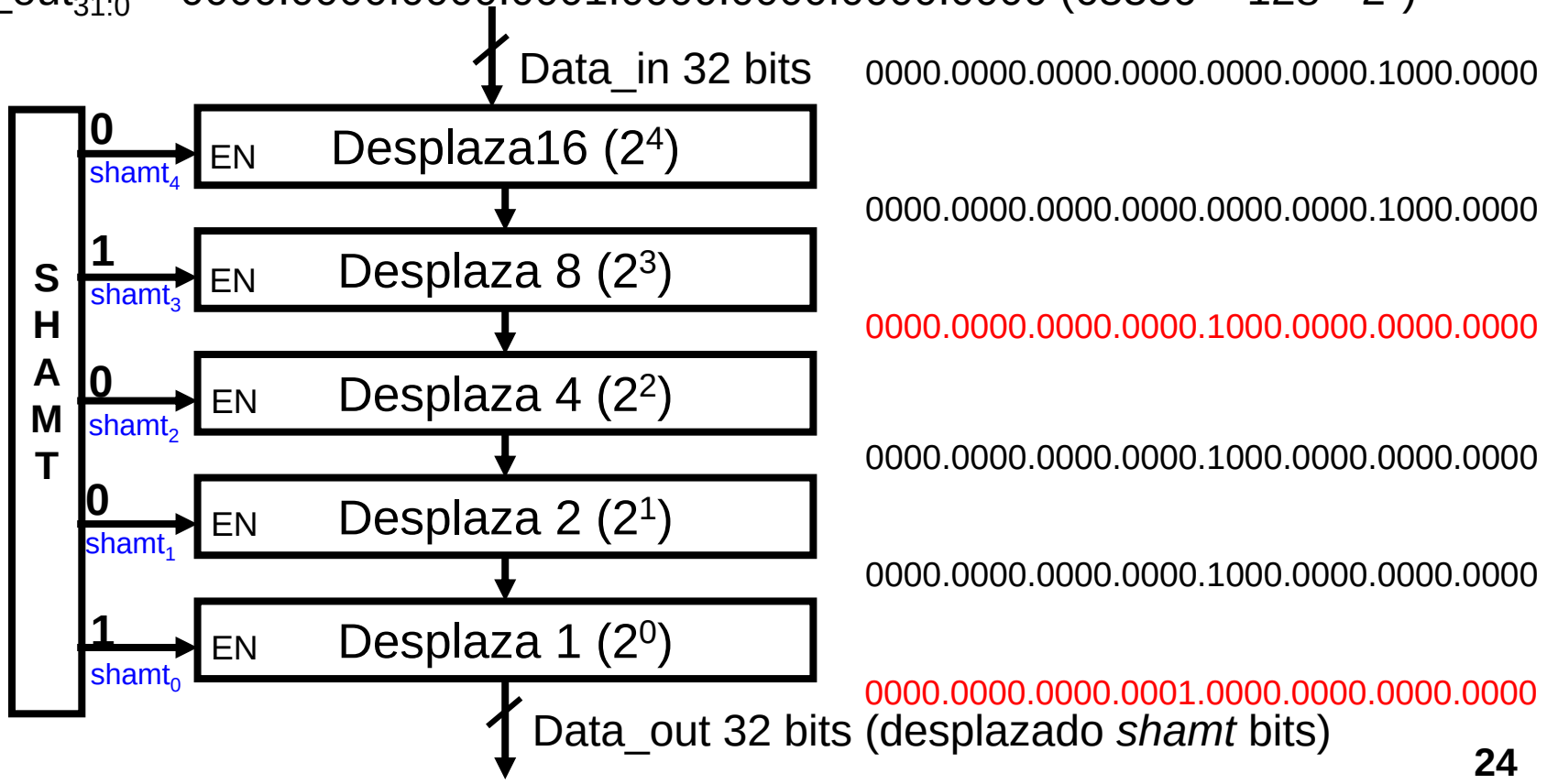
Desplazador en barril (*Barrel Shifter*)

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)



Multiplicadores

- Pasos para multiplicar (en decimal o binario):
  - Los productos parciales se forman multiplicando un dígito del multiplicador por el multiplicando completo
  - Los productos parciales se desplazan y suman para tener el resultado final

Decimal

	230	multiplicand
x	42	multiplier
	460	partial
+	920	products
	9660	result

230 x 42 = 9660

Operación de multiplicar, sin signo

Sin signo

```
    1 1 0 0 0 1      (49)
x   1 1 0 1 0        (26)
-------------------
    0 0 0 0 0 0
  1 1 0 0 0 1
0 0 0 0 0 0
1 1 0 0 0 1
1 1 0 0 0 1
-------------------
1 0 0 1 1 1 1 1 0 1 0   (1274)
```

Sin signo

```
    0 0 1 0 0 1      ( 9)
x       0 1 1 0      (6)
-------------------
    0 0 0 0 0 0
  0 0 1 0 0 1
0 0 1 0 0 1
0 0 0 0 0 0
-------------------
0 0 0 1 1 0 1 1 0      (54)
```

Operación de multiplicar, con signo

- **Aritmética sin signo:**
  - Calcular el signo del resultado:
    - ✓ Pos*Pos = Pos, Pos*Neg = Neg , Neg*Pos = Neg, Neg*Neg = Pos.
  - Si algún operando es negativo:
    - ✓ Hacer su complemento a 2 para hacerlo positivo.
  - Multiplicar igual que sin signo.
  - Si el signo del resultado debe ser negativo:
    - ✓ Hacer el complemento a 2 del resultado obtenido.

Operación de multiplicar, con signo

$0010_2 \times 1100_2 = 2_{10} \times -4_{10}$

P N

Signo resultado: Negativo (PxN)
C2 de $(1100_2) = 0100_2$

P → 0 0 1 0 (2)

P → x 0 1 0 0 (4)

0 0 0 0

0 0 0 0

0 0 1 0

0 0 0 0

0 0 0 1 0 0 0 0 (8)

1 1 1 1 0 0 0 (-8)

Operación de multiplicar, con signo

- **Aritmética con signo**:
  - Hacer las multiplicaciones parciales igual que sin signo, pero extender en signo los resultados.
  - Si el MSB del multiplicador es -1, el multiplicador es negativo, por lo que el resultado parcial último no será el multiplicando, sino el complemento a 2 del mismo.
  - El producto final es la suma de los productos parciales.

Operación de multiplicar, con signo

$0010_2 \times 1100_2 = 2_{10} \times -4_{10}$

	0 0 1 0	(2)
	x 1 1 0 0	(-4)
	0 0 0 0 0 0 0 0	$(2*(0*2^0)$
	0 0 0 0 0 0 0 0	$(2*(0*2^1)$
	0 0 0 0 1 0	$(2*(1*2^2)$
C2(0010) →	1 1 1 1 0	$(2*(-1*2^3)$
	1 1 1 1 1 0 0 0	(-8)

$1011_2 \times 0010_2 = -5_{10} \times 2_{10}$

	1 0 1 1	(-5)
	x 0 0 1 0	(2)
	0 0 0 0 0 0 0 0	$(-5*(0*2^0)$
	1 1 1 1 0 1 1	$(-5*(1*2^1)$
	0 0 0 0 0 0	$(-5*(0*2^2)$
	0 0 0 0 0	$(-5*(0*2^3)$
	1 1 1 1 0 1 1 0	(-10)

Operación de multiplicar, con signo

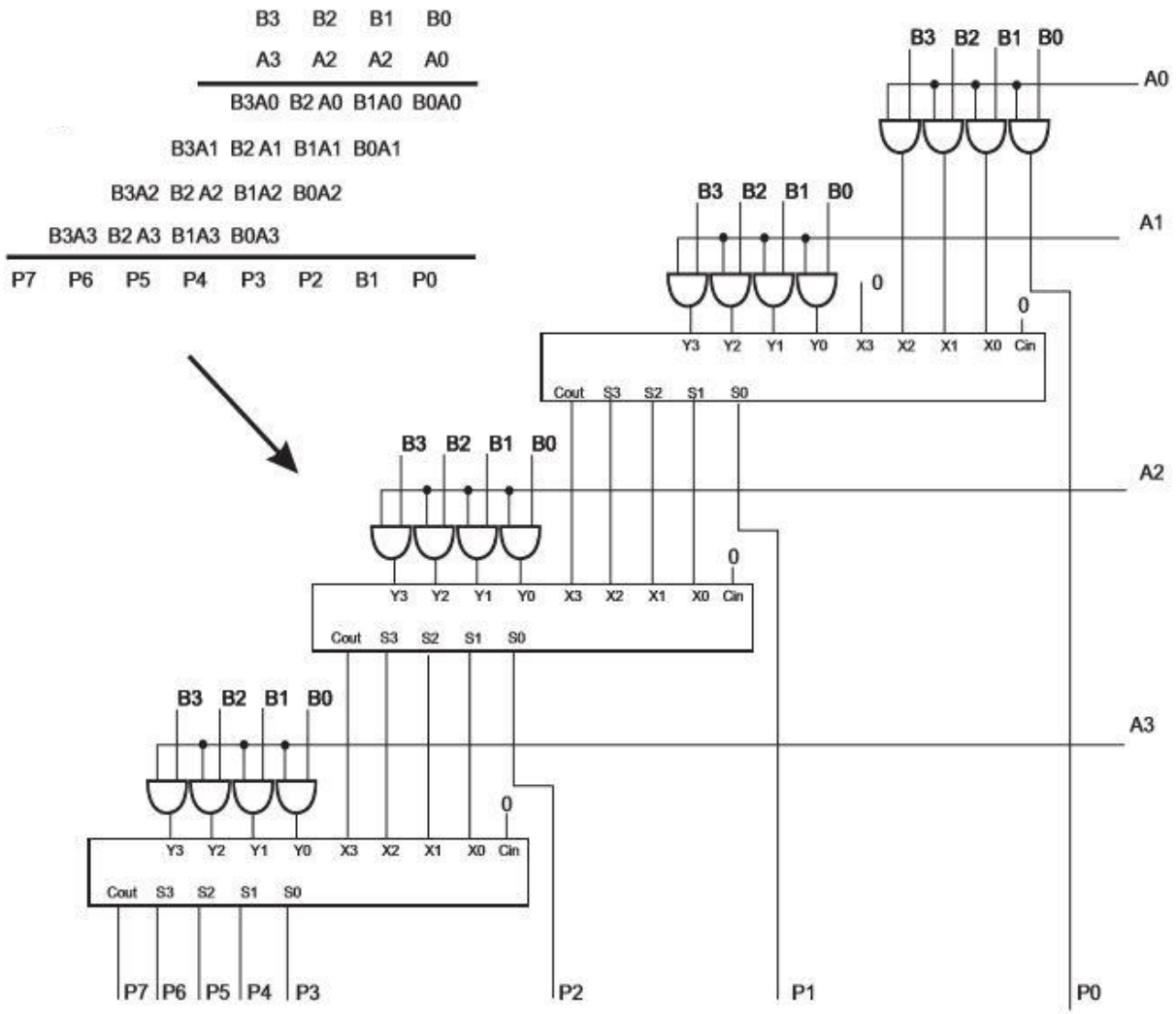
Sin signo

```
      1 1 0 0 0 1      (49)
    x   1 1 0 1 0      (26)
-----------------
      0 0 0 0 0 0
    1 1 0 0 0 1
  0 0 0 0 0 0
1 1 0 0 0 1
1 1 0 0 0 1
-----------------
1 0 0 1 1 1 1 1 0 1 0  (1274)
```

Con signo

```
      1 1 0 0 0 1      (- 15)
    x   1 1 0 1 0      (- 6)
-----------------
0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 1
0 0 0 1 1 1 1
-----------------
0 0 0 0 1 0 1 1 0 1 0  (+ 90)
```

Multiplicador 4 x 4 (sin signo)



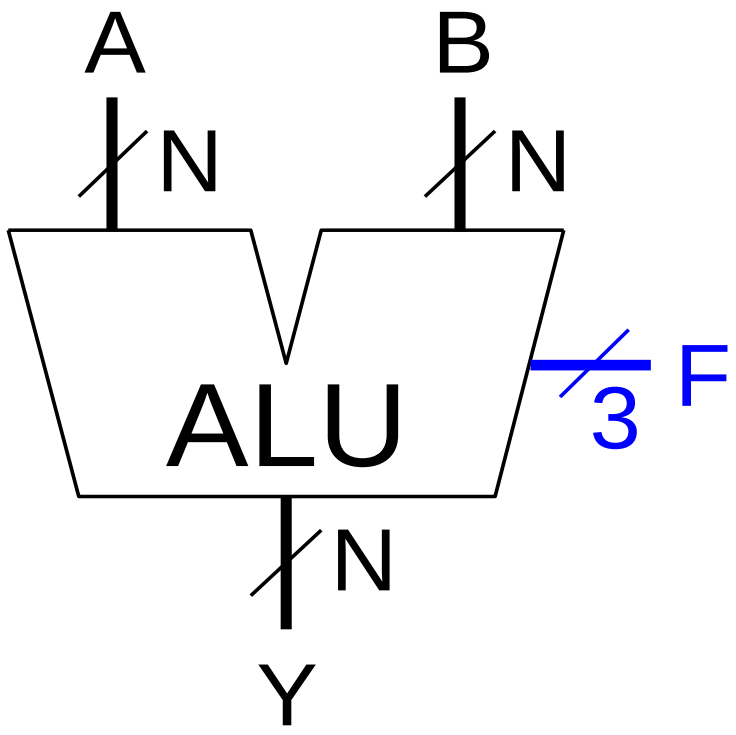
P

Índice

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

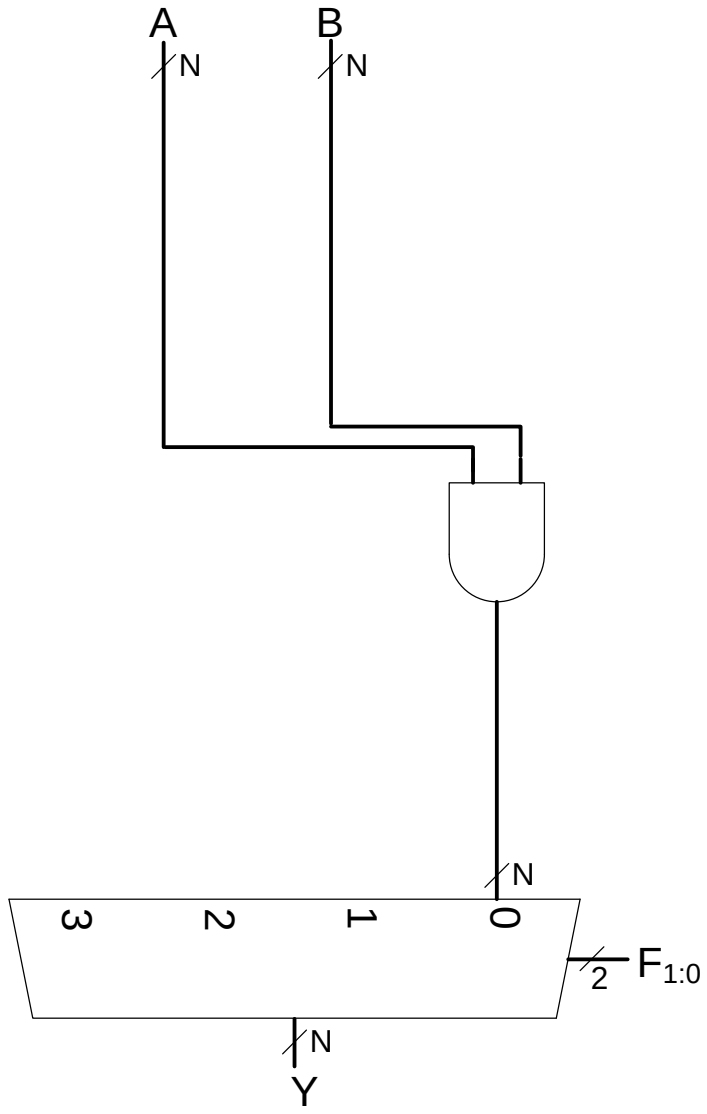
Arithmetic Logic Unit (ALU)

$F_{2:0}$ = Selector de función



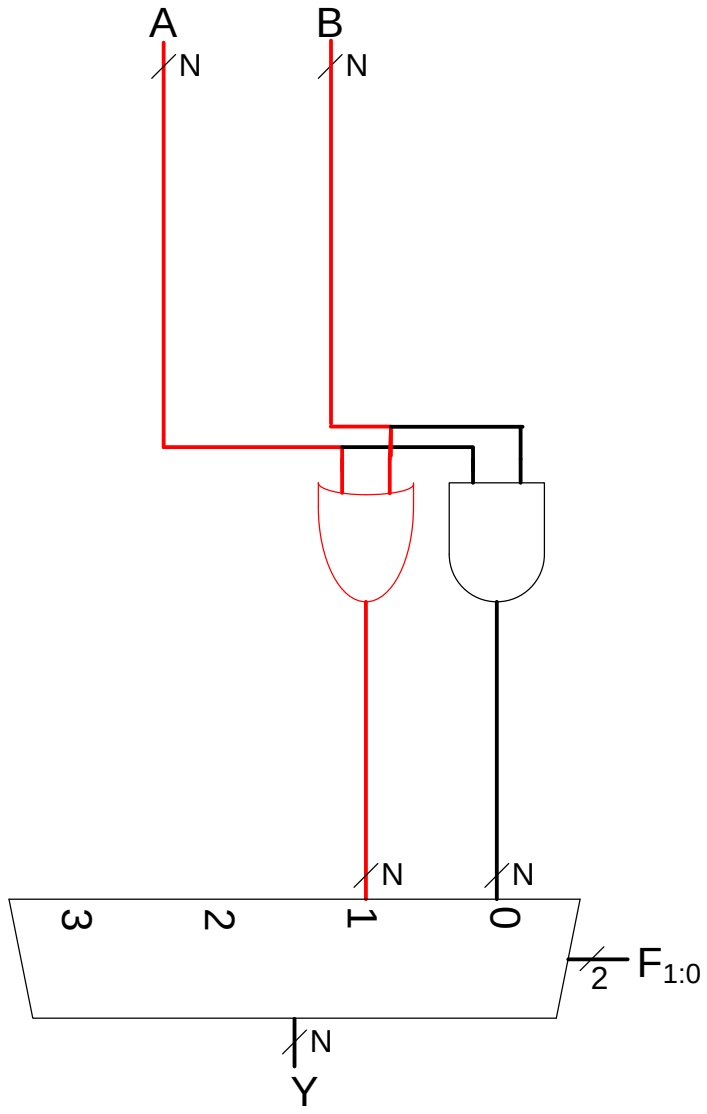
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



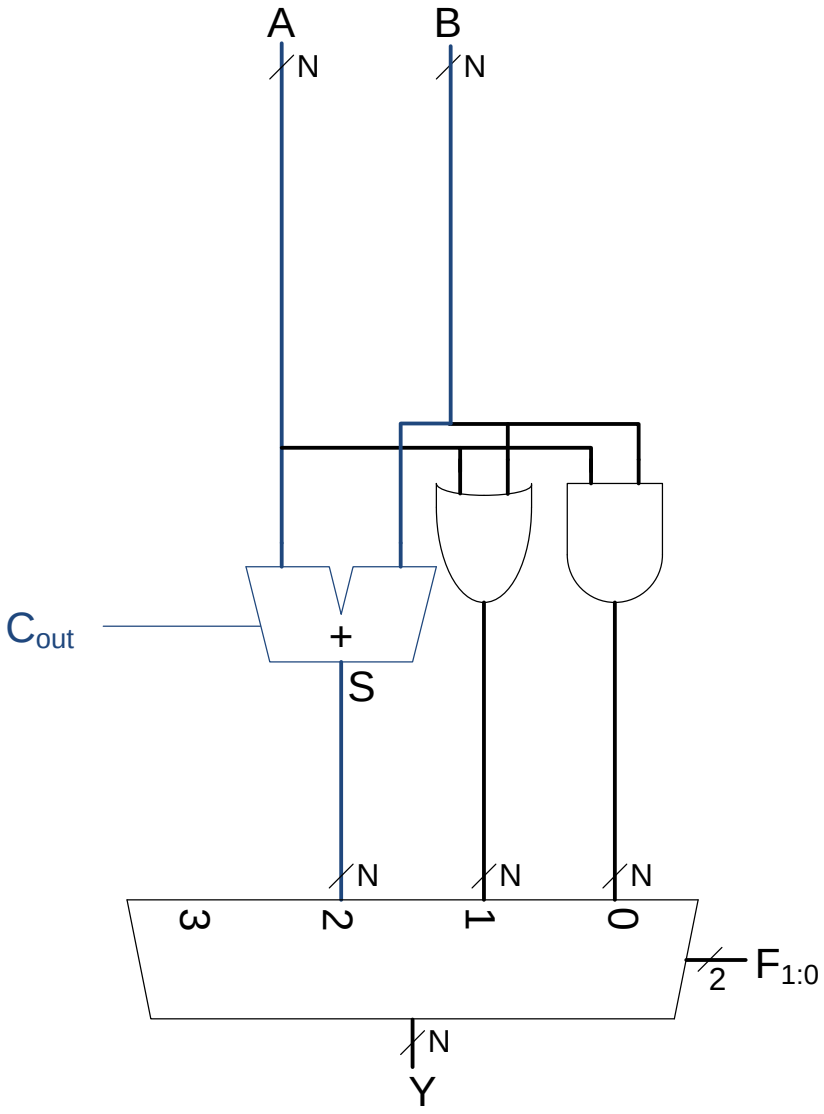
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



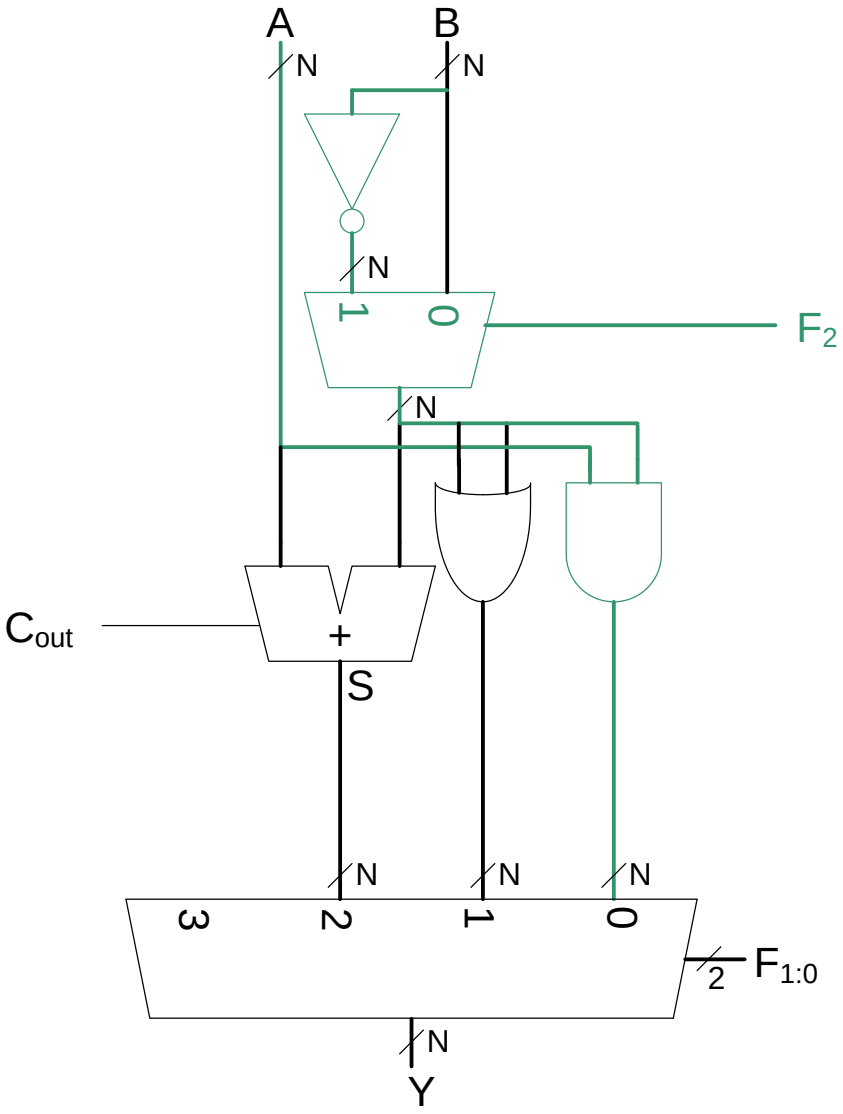
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



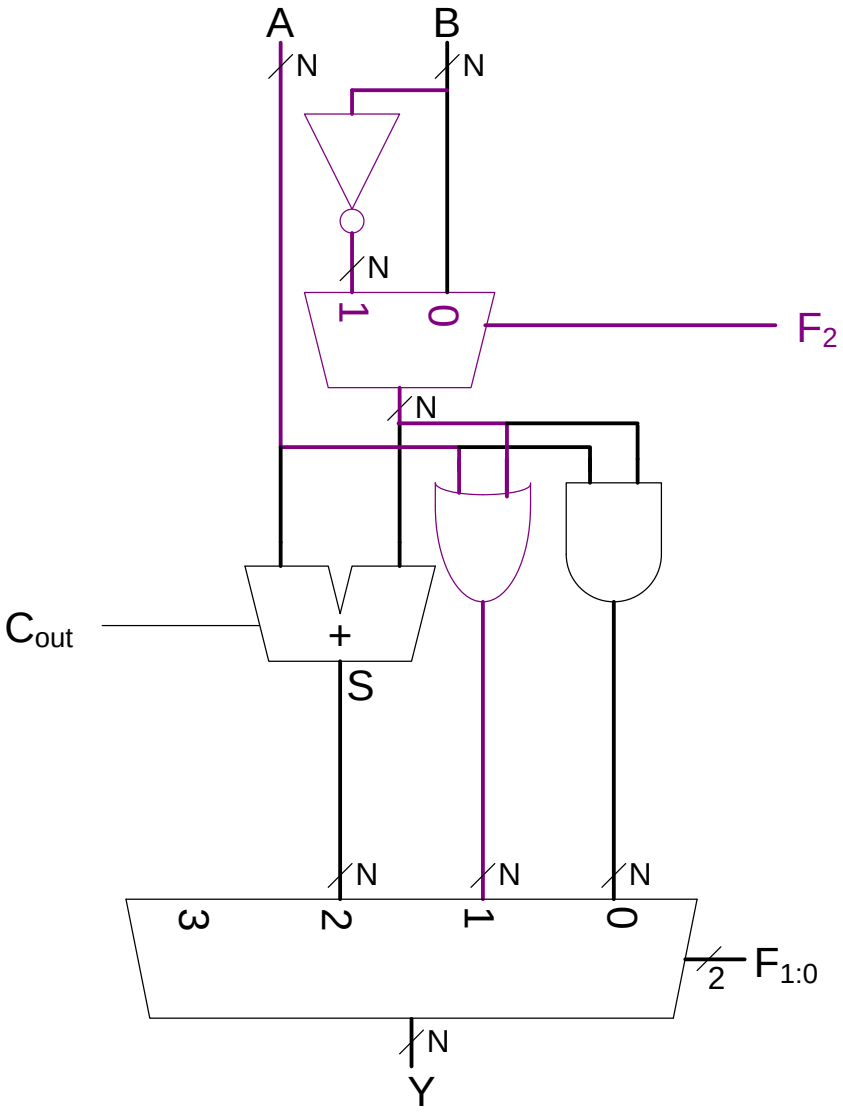
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



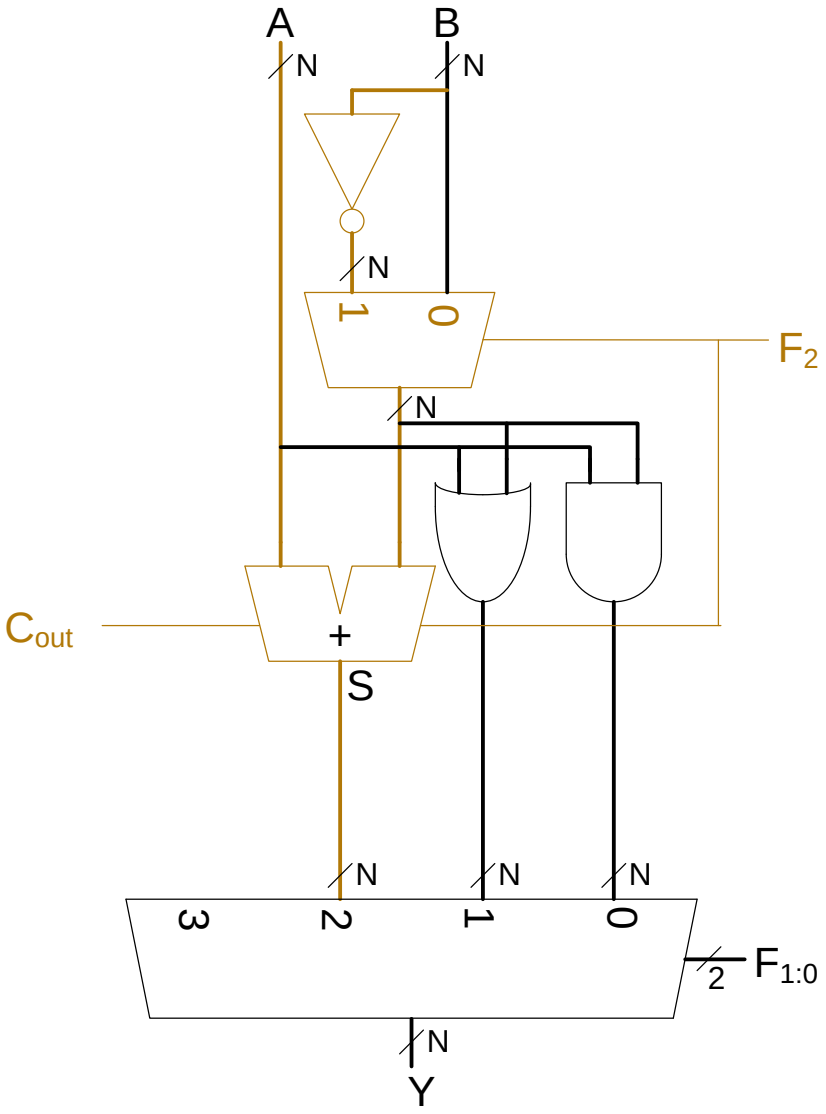
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



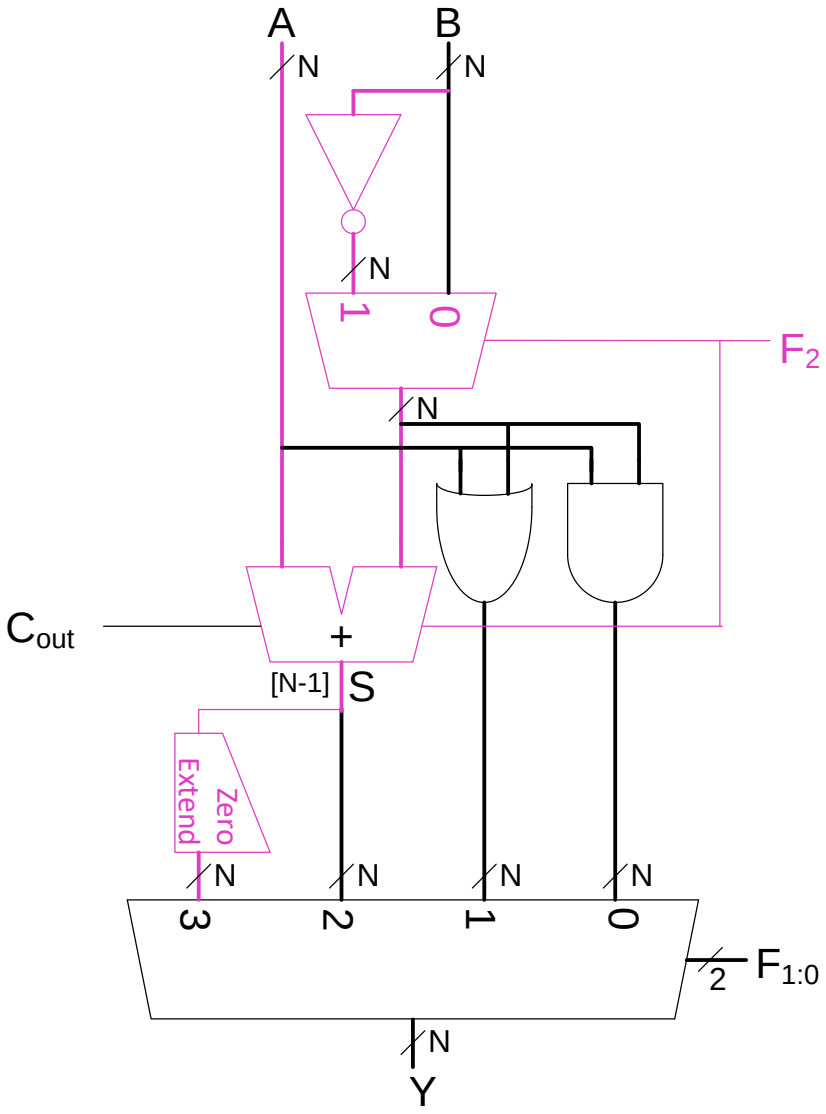
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



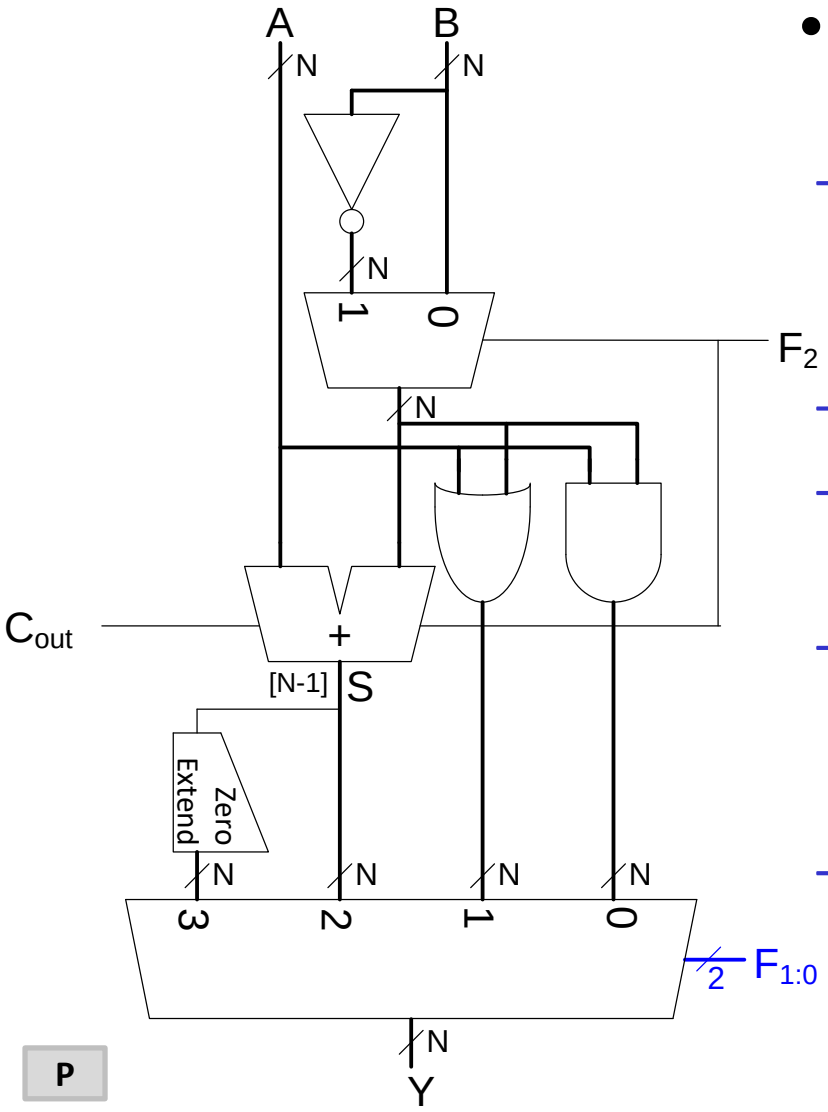
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

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)



- 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.

Unidad 2: La Unidad Aritmético Lógica (ALU)

Escuela Politécnica Superior - UAM

Problemas Unidad 2

2.1.- Demostrar por medio de una tabla de verdad, que la expresión lógica Cn⊕Cn-1 genera una señal denominada bandera de desbordamiento (*overflow*, V), para identificar errores que se generan al sumar números enteros con signo en codificación complemento a 2.

2.2. Además del resultado, la ALU genera un conjunto de bits que pueden ser utilizados por el sistema o por los usuarios para el control de las operaciones aritmético-lógicas desarrolladas. Entre estos los más conocidos son el bit o bandera de signo (N, N='1' si el resultado es negativo), el bit o bandera de cero (Z, Z='1' si el resultado es cero), el bit o bandera de acarreo (C, C='1' si hay acarreo en la operación de suma entre los bits más significativos de ambos operandos), y el bit de desbordamiento o bandera de overflow (V, V='1' si se supera la capacidad de representación del sistema).
Utilizando números binarios de 8 bits con signo y representados en complemento a 2, realice las operaciones señaladas con dos operandos en decimal y compruebe en cada caso, el valor de estos cuatro bits, N, Z, C y V, señale en cada caso su signifcado.

a) 46 + 67 **b)** 112 - 89 **c)** 75 + 95 **d)** -34 – 97

2.3.- Utilizando números binarios de 8 bits con signo y representados en complemento a 2, realice en el orden señalado por los paréntesis las operaciones indicadas. Para cada resultado parcial, compruebe si se produce un desbordamiento aritmético y calcule también la validez del resultado final. Analice los resultados obtenidos en relación con el valor de la bandera V del problema 2.1 y observe que, resultados parciales incorrectos, no suponen necesariamente que el resultado final lo sea.

a) (((((32 +100) + 70) + 24) – 62) – 50). **b) ((((**43 - 12) + 34) + 75) – 47). **c) ((((**15 – 77) – 43) – 38) + 32)

T

Problemas Unidad 2

2.4. Multiplicar las parejas de números con signo en complemento a 2 siguientes:

a) 010111_2 x 110110_2 **b)** 110011_2 x 101100_2 **c)** 0011111_2 x 0011111_2

d) 56_{10} x $(-67)_{10}$ **e)** -43_{10} x $(-113)_{10}$ **f)** $A4_{16}$ x $B8_{16}$.

Nota: Aplicando la propiedad de extensión de signo del complemento a 2, utilizar en todos los casos 8 bits para cada multiplicando

T

Problemas Unidad 2

2.6.- Considerar la ALU y los registros que se muestran en la figura. Responder a las siguientes cuestiones escribiendo la(s) palabra(s) de control adecuada(s). Cada palabra de control debe especificarse de acuerdo con el formato $C_4C_3C_2C_1C_0$. Por ejemplo, la operación "A+B→A" se escribiría 10001.

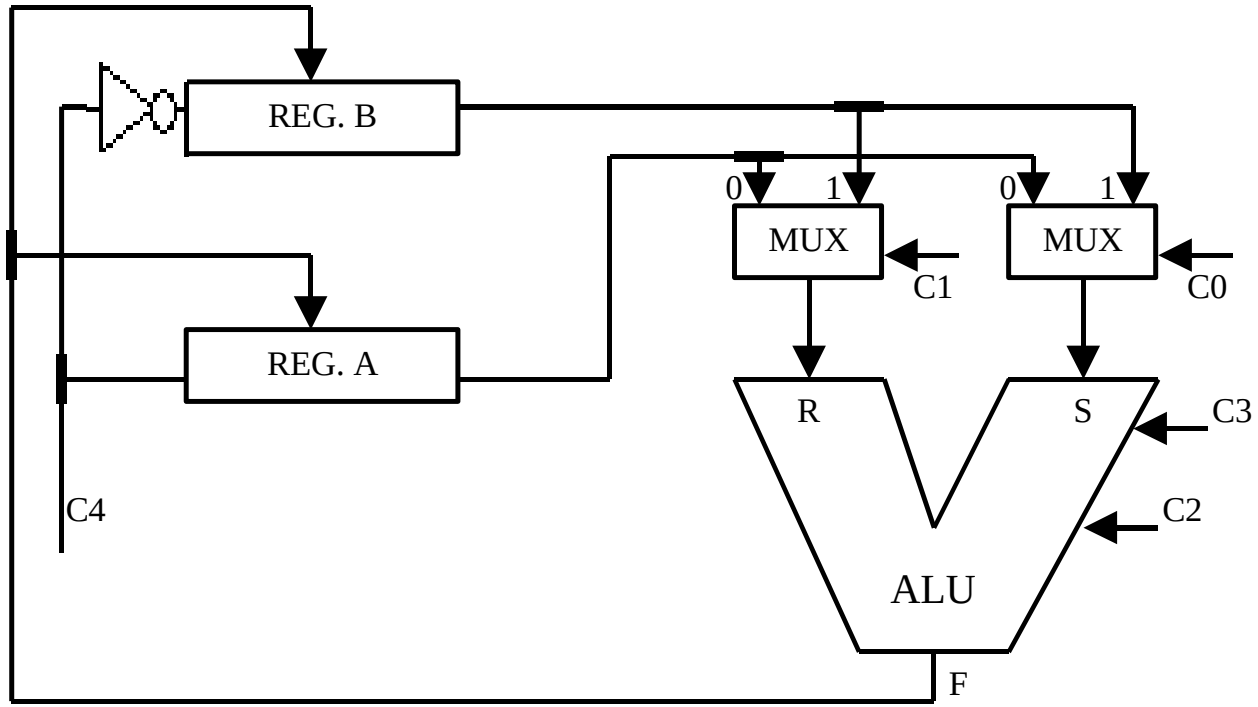
a) ¿Cómo se puede poner el valor 0 en el registro A? (sugerir dos métodos).

b) Sugerir una secuencia de control que intercambie los contenidos de los registros A y B. La interpretación de los distintos puntos de control se resume en las tablas adjuntas.

C_1C_0	→ R	→ S
00	A	A
01	A	B
10	B	A
11	B	B

C_3C_2	F
00	R + S
01	R – S
10	R **AND** S
11	R **XOR** S

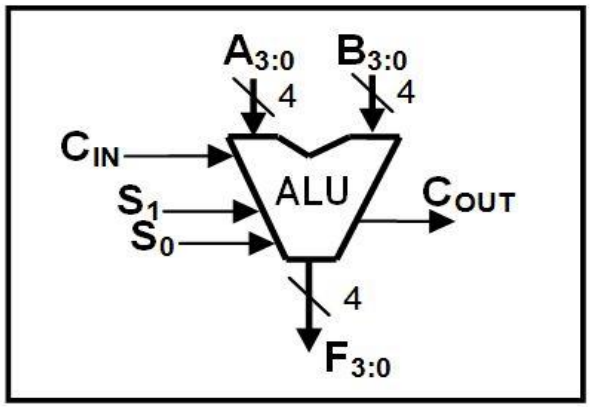
C_4	ACCION
0	F → B
1	F → A



Problemas Unidad 2

2.7. Utilizando la ALU de la figura, indique las operaciones a realizar en la ALU para que las salidas representen el módulo del resultado de la diferencia (X - Y), en donde X e Y son números positivos de 4 bits en complemento a 2.

S_1 S_0	Operación
0 0	F = AND (A,B)
1 0	F = A + B + C_{IN}
0 1	F = /A + C_{IN}
1 1	F= A + /B + C_{IN}



Problemas Unidad 2

2.9. Se tiene la arquitectura de la figura con un banco de dos registros (R0 y R1) de 8 bits y una ALU de 8 bits que realiza cuatro operaciones (ver tabla). La señal de control ALUOp, de 2 bits, sirve para elegir la operación a realizar. Las señales de control Rs1, Rs2 y Rd sirven para indicar cuáles son los 2 registros utilizados en la ALU (Rs1 para Op1 y Rs2 para Op2) y cuál es el registro escrito (Rd para Result). En todas ellas, '0' indica el registro R0 y '1' indica el registro R1.La operación SLT pone la salida a 1 (número entero 1) si Op1 < Op2 y a 0 (número entero 0) en caso contrario, considerando que los operandos tienen signo y están en complemento a 2.

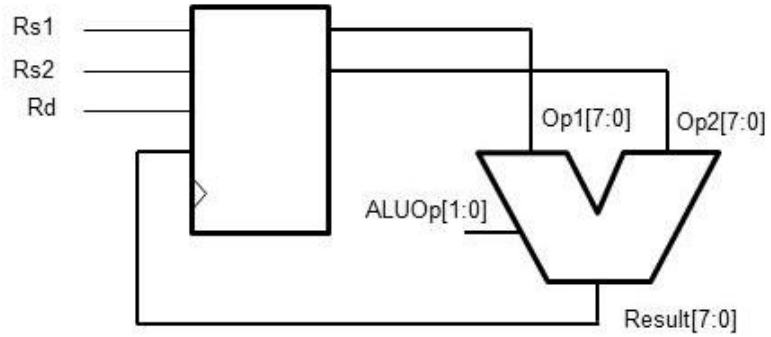
La palabra de control es de 5 bits, (ALUOp,Rs1,Rs2,Rd). Por ejemplo, si la palabra de control es ¨00011¨ se realizará la operación R1 <= R0 + R1;

a) Indique la palabra de control para conseguir que R0 quede a 0

b) Indique la palabra de control para conseguir que R1 reciba NOT(R0)

c) Si inicialmente R0=0x08 y R1=0xFF, indique a qué queda el registro R1 tras realizar la siguiente secuencia de instrucciones: 1. “00010” y 2. “11101”

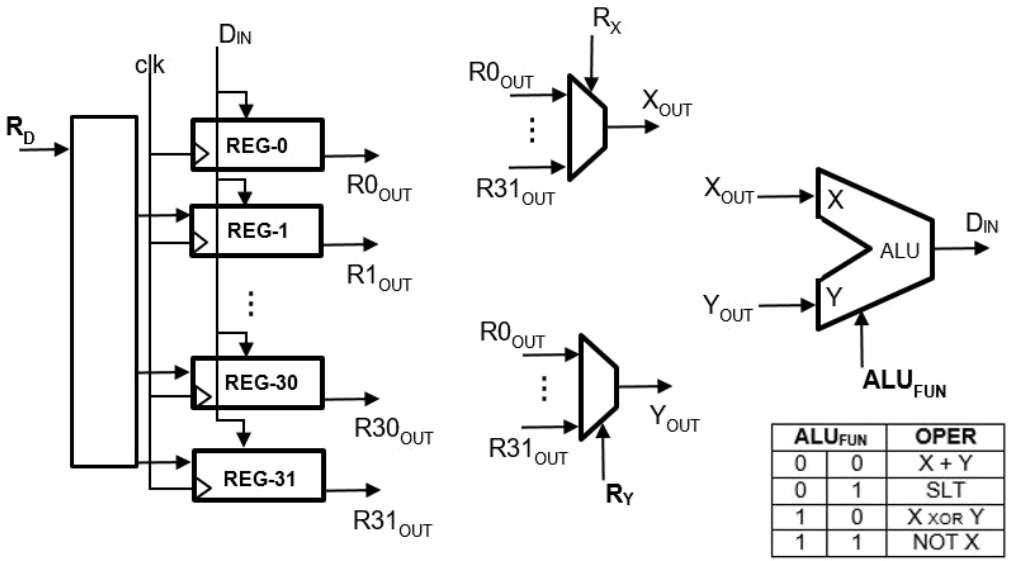
d) Si inicialmente R0=0xFE y R1=0x05, indique a qué queda el registro R1 tras realizar la siguiente secuencia de instrucciones: 1. “11011” y 2. “10011”



ALUOp	Operación
00	Result=Op1+Op2
01	Result=NAND(Op1,Op2)
10	Result=XOR(Op1,Op2)
11	Result=SLT(Op1,Op2)

Problemas Unidad 2

2.10. En el circuito de la figura se muestra la arquitectura de un cierto sistema digital en el que se distinguen elementos combinacionales ya conocidos como multiplexores, decodificadores y una unidad aritmético lógica (ALU), y elementos secuenciales como un conjunto de 32 registros, en el que como es habitual, el registro R0 es de sólo lectura y su valor es siempre 0. Considere que todos los registros son de 32 bits y salvo R_0 con valores desconocidos. En la figura se identifican cuatro señales de control con distinto tamaño en bits, R_D, R_X, R_Y y ALU_{FUN}.



ALU_{FUN}		OPER
0	0	X + Y
0	1	SLT
1	0	X XOR Y
1	1	NOT X

La operación SLT de la ALU, como se ha visto en clase, pone un '1' en el registro destino si el valor de la entrada X es menor que el de la entrada Y.

A la vista del esquema facilitado, se pide:

a) Señale, justificando necesariamente la respuesta, el tamaño en bits y la función de la señal de control "R_D".

b) Señale, justificando necesariamente la respuesta, el tamaño en bits y la función de la señal de control "R_X".

c) Describa un algoritmo como desee y asócielo a la palabra o palabras de control correspondientes para su ejecución para conseguir que R_5 <= 0xFFFFFFFF.

d) Diseñe un algoritmo y escriba la palabra o palabras de control necesarias para su ejecución para calcular la operación R_4 <= Complemento a 2 de R_3.

NOTA: Para considerar válida una palabra de control, debe señalar la secuencia de bits que corresponda en el orden de las variables R_D, R_X, R_Y, ALU_{FUN}. Utilice una coma para separar cada señal.

T