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
  - ✓ Operadores lógicos
  - ✓ Sumadores y Restadores
  - ✓ Desplazadores y Multiplicadores
- Diseño de una ALU

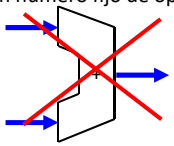
Introducción

- 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):

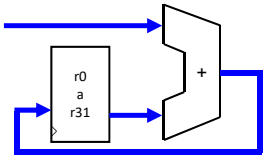
- ✓A + B;
- ✓C + D + E;
- ✓F + G + H + I;
- ✓…



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)



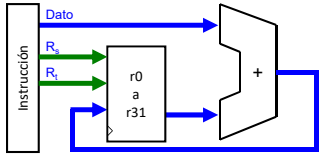
Circuito sumador genérico

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



5

Circuito sumador genérico

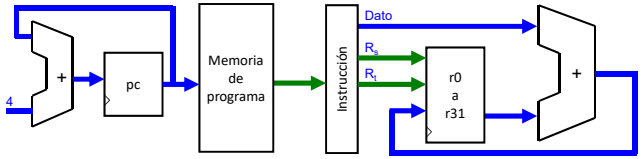
¿Cómo recibe el micro las instrucciones?

- ✓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



6

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

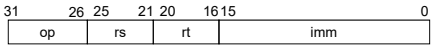
Dirección	Memoria de programa
0	$R_1 = R_0 + 8$;
4	$R_2 = R_1 + 21$;
8	$R_3 = R_2 + 14$;
C	????
	…
FF…C	????

7

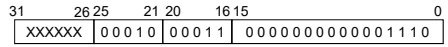
Memoria de programa (instrucciones)

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

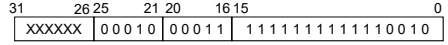
- ✓ 5 bits para el registro destino, R_t
- ✓ 5 bits para el registro fuente, R_s
- ✓ 16 bits para el dato inmediato
- ✓ Resto de bits no se usan. En los micros reales sirven para indicar el código de instrucción, *operation code* (op), ya que hay más de una instrucción de este mismo tipo



Ejemplo: R3 = R2 + 14;



Ejemplo: R3 = R2 - 14;

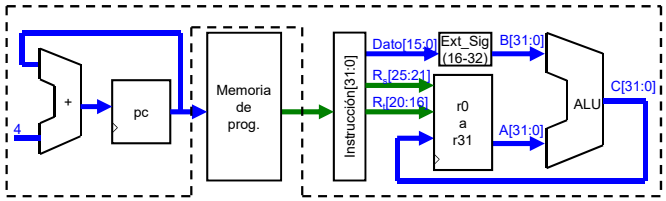


8

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

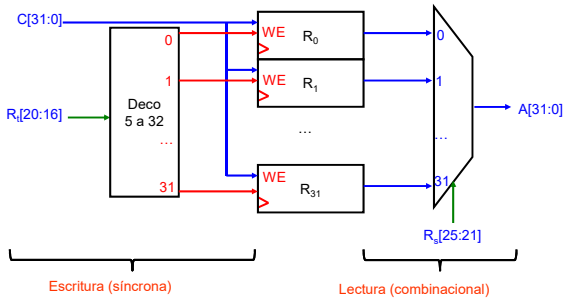


9

Banco de Registros (GPR)

¿Cómo se realiza el banco de registros?

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



10

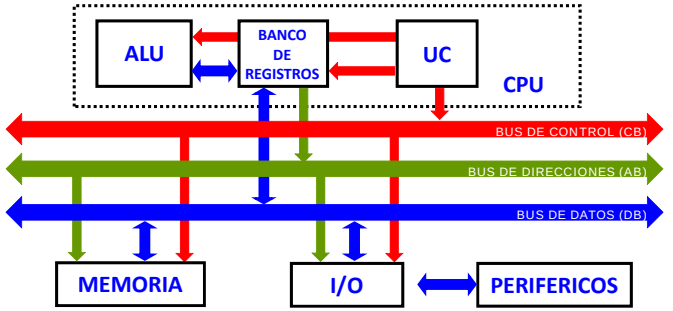
Microprocesador Completo

¿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)

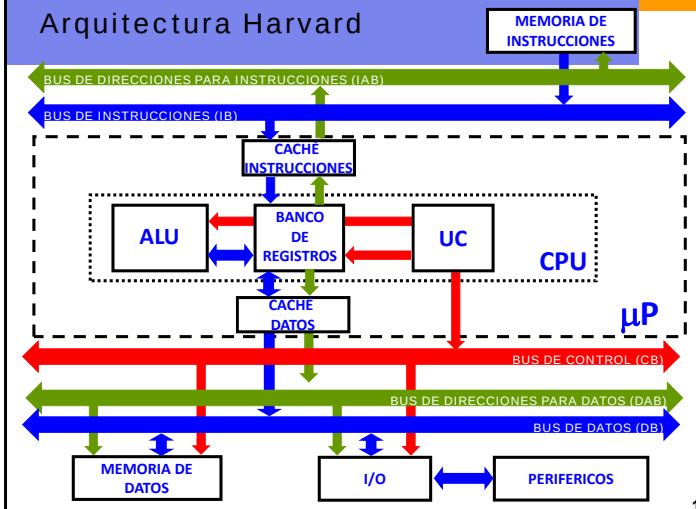
11

Arquitectura clásica Von Neumann



12

Arquitectura Harvard

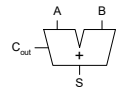


Índice

- 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

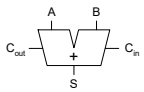


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

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

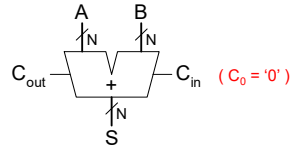
$S = A \oplus B \oplus C_{in}$

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

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.

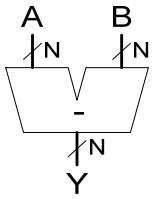
Symbol



Otros operadores

Restador: Y = A - B

Symbol



Recuerda
(Fundamentos de Computadores)

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

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

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

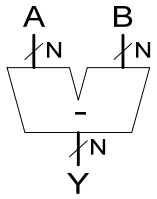
$Inverso(00101_2) = NOT(00101)+1 =$
$11010+1 = 11011$

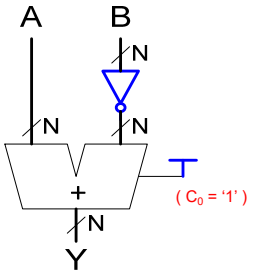
$Y=A + (-B) = 01000_2 + 11011_2 = 00011_2$

Otros operadores

Restador: Y = A - B

Symbol **Implementation**



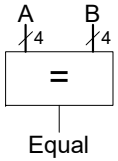


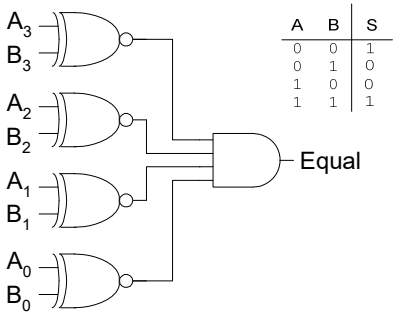
P

Otros operadores

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

Symbol **Implementation**





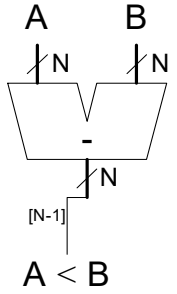
XNOR

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

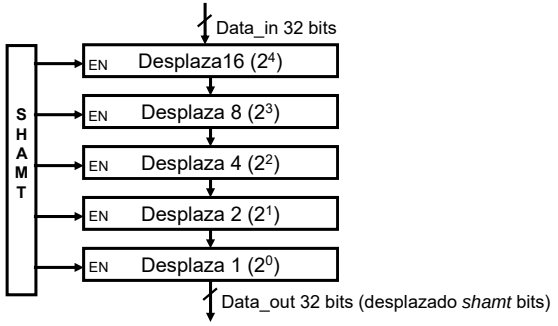
- **Desplazador lógico (logical shifter):** desplaza el valor a izquierda o derecha y rellena con 0's.
  - ✓ Ex: 11001 >> 2 = 00110
  - ✓ Ex: 11001 << 2 = 00100
- **Desplazador aritmético (arithmetic shifter):** igual que el lógico, salvo que hacia la derecha rellena con el bit de signo (msb).
  - ✓ 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 $shamt_{4:0}$



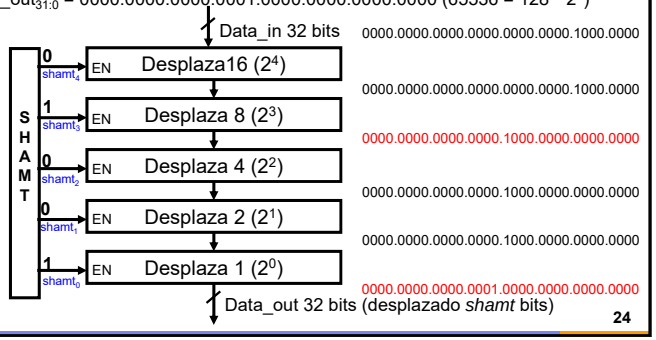
Desplazador en barril (*Barrel Shifter*)

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

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

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

P N

```
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     (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.

29

Operación de multiplicar, con signo

$0010_2 \times 1100_2 = 2_{10} \times -4_{10}$ $\quad 1011_2 \times 0010_2 = -5_{10} \times 2_{10}$

	0 0 1 0	(2)	1 0 1 1	(-5)
	x 1 1 0 0	(-4)	x 0 0 1 0	(2)
	0 0 0 0 0 0 0 0	($2*(0*2^0)$)	0 0 0 0 0 0 0 0	($-5*(0*2^0)$)
	0 0 0 0 0 0 0	($2*(0*2^1)$)	1 1 1 1 0 1 1	($-5*(1*2^1)$)
	0 0 0 0 1 0	($2*(1*2^2)$)	0 0 0 0 0 0	($-5*(0*2^2)$)
C2(0010) →	1 1 1 1 0	($2*(-1*2^3)$)	0 0 0 0 0	($-5*(0*2^3)$)
	1 1 1 1 1 0 0 0	(-8)	1 1 1 1 0 1 1 0	(-10)

30

Operación de multiplicar, con signo

Sin signo

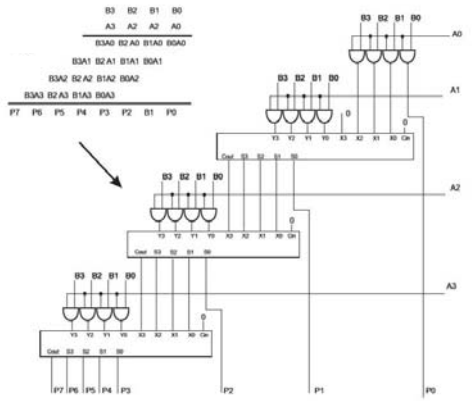
1 1 0 0 0 1	(49)
x 1 1 0 1 0	(26)
0 0 0 0 0 0	
1 1 0 0 0 1	
0 0 0 0 0 0	
1 1 0 0 0 1	
1 1 0 0 0 1	
1 0 0 1 1 1 1 1 1 0 1 0	(1274)

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 0	
1 1 1 1 1 1 1 0 0 0 1	
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)

31

Multiplicador 4 x 4 (sin signo)



P

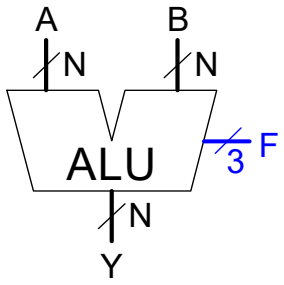
32

Índice

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

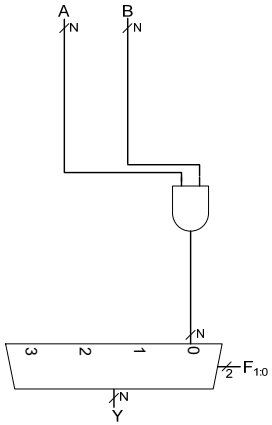
Arithmetic Logic Unit (ALU)

$F_{2:0}$ = Selector de función



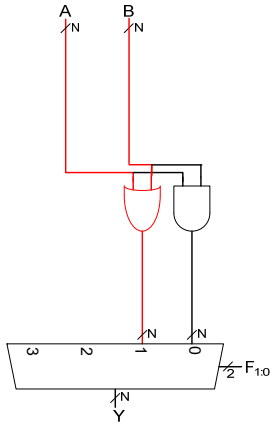
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

Diseño de la ALU



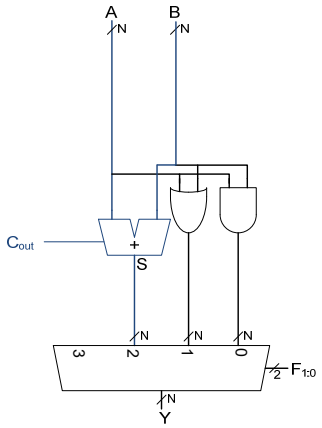
$F_{2:0}$	Función
000	A and B
001	A or B
010	A + B
011	Sin usar
100	A and /B
101	A or /B
110	A – B
111	SLT

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

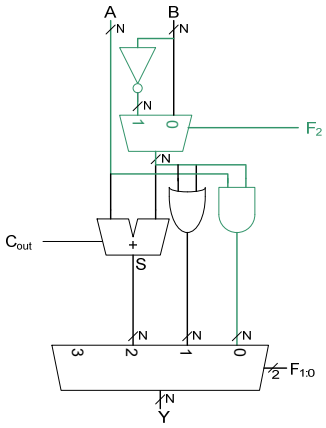
Diseño de la ALU



$F_{2:0}$	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

37

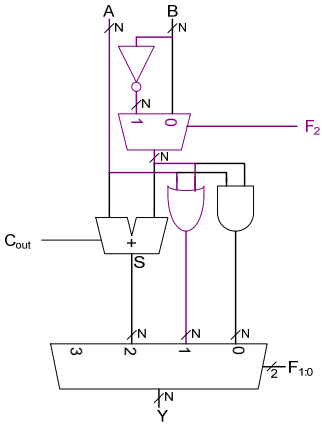
Diseño de la ALU



$F_{2:0}$	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

38

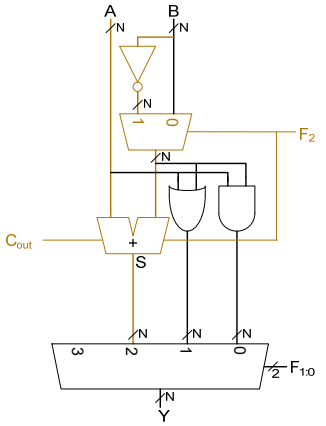
Diseño de la ALU



$F_{2:0}$	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

39

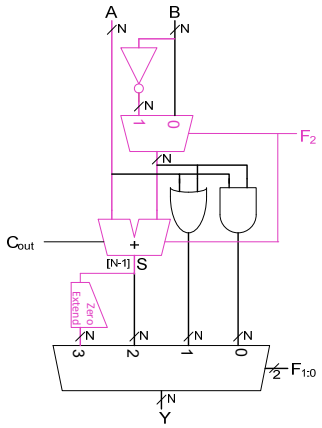
Diseño de la ALU



$F_{2:0}$	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

40

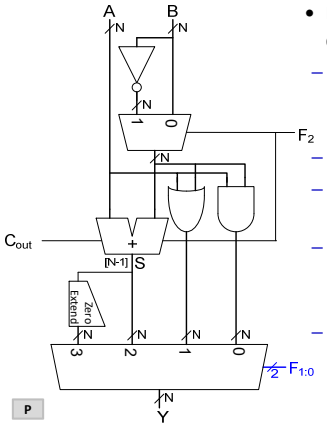
Diseño de la ALU



$F_{2:0}$	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

41

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.

P

42