ARITMETICA PARA COMPUTADORAS

1.- INTRODUCCION

- Objetivo
    - Estudio de uno de los Componentes Clásicos de una Computadora
        - Unidad Aritmético Lógica (ALU)
- Partes a estudiar
    - Representación de los números negativos
    - Operaciónes Aritméticas y/o lógicas
    - Una ALU sencilla
    - Operaciones complejas con enteros (* y /)
    - Operaciones en Punto Flotante

2.- NÚMEROS NEGATIVOS

- Sistema de numeración
  - Definición
  - Concepto de base
  - Sistemas de numeración más usuales
    - Decimal : base 10; Símbolos 0, 1, ..., 9
    - Hexadecimal: " 16; " 0, 1, ...,9, A, ...F
    - Octal " 8 " 0, 1, ..., 7
    - Binario " 2 " 0, 1

- Códigos de Numeración
  - Aplicación concreta entre combinaciones de símbolos y cantidades
  - Algunos Códigos:
    - Código binario natural
    - Código Gray .....

- Código Binario Natural
  - Similitud con el decimal usado normalmente
  - Valor de cada bit ➔ depende de su posición

    peso = $b_i * 2^i$ (i= 0,….,n-1)
  - Ejemplo

$$\mathbf{a_{n-1}, a_{n-2}, a_{n-3}, ..., a_1, a_o}$$
$$\mathbf{N = a_{n-1}*2^{n-1} + a_{n-2}*2^{n-2} + ... + a_1*2^1 + a_0*2^0}$$

  - Numeración de los bits (derecha a izquierda)

- Números binarios en MIPS

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dos} = 0_{diez}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{dos} = 1_{diez}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dos} = 2_{diez}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dos} = 4.294.967.293_{diez}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dos} = 4.294.967.294_{diez}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dos} = 4.294.967.295_{diez}$

- Forma de representar los números negativos

  - Signo Magnitud
    - bit de signo separado de la magnitud (bsm)
      - positivo: bsm = 0 y la magnitud
      - negativo: bsm = 1 y la magnitud
    - existen dos ceros: + 0 y - 0
    - rango:

      $$-2^{n-1}-1 \leftrightarrow +2^{n-1}-1$$

  - Complemento a 1
    - Definición:

      $$\mathbf{l\,a = 2^{n-1} - 1 - a}$$ (n = nº de bits)
    - Signo
      - positivo: bit de signo = 0 más número en valor absoluto
      - negativo: bit de signo = 1 más Compl. a 1 del número
    - rango:

      $$-2^{n-1} \leftrightarrow +2^{n-1}-1$$

- Exceso a M
  - Valor binario = Valor Decimal + M
  - Signo
    - positivo: bsm = 1
    - negativo: bsm = 0
  - Rango:

    $$-2^{n-1} \leftrightarrow +2^{n-1} - 1$$

- Complemento a 2
  - Def:

    $$2\ a = 2^{n} - a$$ (n = nº de bits)
  - Signo
    - positivo: bsm = 0 y el número en valor absoluto
    - negativo: bsm = 1 y el número en Complemento a 2
  - rango:

    $$2^{n-1} \leftrightarrow +2^{n-1} - 1$$

- ejemplo

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dos} = 0_{diez}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{dos} = 1_{diez}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dos} = 2_{diez}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dos} = 2.147.483.645_{diez}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dos} = 2.147.483.646_{diez}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dos} = 2.147.483.647_{diez}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{dos} = -2.147.483.648_{diez}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{dos} = -2.147.483.647_{diez}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{dos} = -2.147.483.646_{diez}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{dos} = -3_{diez}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{dos} = -2_{diez}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{dos} = -1_{diez}$

- Ventajas:
  - Es la representación más coherente
  - Facilita las operaciones de suma y resta
  - No mantiene el orden

- Instrucciones con los números enteros con o son signo:
  - Hay valores que nunca pueden ser negativos (p. ej. direcciones)
  - Inicializar menor que sin signo (sltu)
  - Inicializar menor que (inmediato) sin signo (sltiu)
  - Ejemplo:

    $16 = FFFFFFFC (-4 ó 4.294.964.295)
    $17 = 00000001 (1 ó 1)

    slt $8, $16, $17 # con signo 1⇒ $8
    sltu $8, $16, $17 # sin signo 0⇒ $8

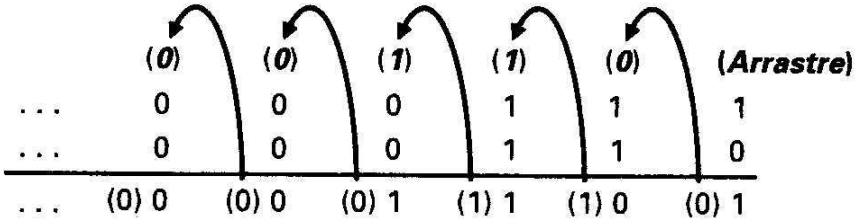
- Arquitectura MIPS revelada

Lenguaje ensamblador MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	sumar	add $1,$2,$3	$1=$2+$3	3 operandos; datos en reg.
	restar	sub $1,$2,$3	$1=$2-$3	3 operandos; datos en reg.
	sumar inmediato	addi $1,$2,100	$1 = $2 + 100	Usado para sumar constantes
Transfe-rencia de datos	cargar palabra	lw $1,100($2)	$1=Memoria[$2+100]	Dato de memoria a registro
	almacenar palabra	sw $1,100($2)	Memoria[$2+100]=$1	Dato de registro a memoria
	cargar superior inmediato	lui $1,100	$1=100 × 2^{16}	Carga constantes en 16 bits superiores
Salto condicional	saltar sobre igual	beq $1,$2,100	si ($1==$2) ir a PC+4+100	Test igual; salto relativo al PC
	saltar sobre no igual	bne $1,$2,100	si ($1!=$2) ir a PC+4+100	Test no igual; relativo al PC
	inicializa sobre menor que	slt $1,$2,$3	si ($2 < $3)$1 = 1; si no $1 = 0	Compara menor que; **comp. a 2**
	inicializa menor que inm.	slti $1,$2,100	si ($2 < 100)$1 = 1; si no $1 = 0	Compara constante menor que; **comp. a 2**
	inicializa sobre menor que sin signo	**sltu $1,$2,$3**	**si ($2 < $3) $1 = 1; si no $1 = 0**	**Compara menor que; números sin signo**
	inicializa menor que inm. sin signo	**sltiu $1,$2,100**	**si ($2 < 100)$1 = 1; si no $1 = 0**	**Compara menor que constante; números sin signo**
Bifurcación incondicional	bifurcar	j 10000	ir a 10000	Bifurca a dirección destino
	bif. registro	jr $31	ir a $31	Para cambiar, vuelta de procedimiento
	bif. y enlazar	jal 10000	$31 = PC + 4; ir a 10000	Para llamada a procedimiento

3.- SUMA Y RESTA

- Suma Binaria similar a la suma decimal
  - Concepto de acarreo parcial y final



- Funcionamiento con números enteros con signo
  - Transformación de sumas en restas

$$\mathbf{a - b = a + 2_b = 2^n + a - b}$$

a > b ➔ Solución a - b; sobra 2^n
a = b ➔ " 0 ; " "

a < b ➔ " Compl. a dos de (a - b)

- Problema del desbordamiento (overflow)
  - Inversión del bit de signo

Operación	Operando A	Operando B	Resultado
A + B	0	0	<0
A + B	<0	<0	0
A − B	0	<0	<0
A − B	<0	0	0

- MIPS detecta el overflow y genera una Excepción

  - Excepción: llamada planificada a un Procedimiento
    - Dirección de Retorno en EPC (Exception Program Control)
    - Uso de Instrucciones específicas:
      - mfco (move from system control)

- Jr (para el retorno)
- Necesidad de restaurar Registros
  - Uso de $26, $27 que no se restauran en las excepciones

- Instrucciones que detectan la posibilidad de overflow o no
  - add, addi y sub ⇒ producen overflow
  - addu, addiu y subu ⇒ no producen overflow

- Arquitectura MIPS revelada

Lenguaje ensamblador MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	sumar	add $1,$2,$3	$1 = $2 + $3	3 operandos; **excepción posible**
	restar	sub $1,$2,$3	$1 = $2 − $3	3 operandos; **excepción posible**
	sumar inmediato	addi $1,$2,100	$1 = $2 + 100	+ constante; **excepción posible**
	sumar sin signo	**addu $1,$2,$3**	**$1 = $2 + $3**	**3 operandos; no excepciones**
	restar sin signo	**subu $1,$2,$3**	**$1 = $2 − $3**	**3 operandos; no excepciones**
	sumar inmediato sin signo	**addiu $1,$2,100**	**$1 = $2 + 100**	**+ constante; no excepciones**
	transferir desde reg. coprocesador	**mfc0 $1,$epc**	**$1 = $epc**	**Usado para conseguir excepción PC**
Transfe-rencia de datos	cargar palabra	lw $1,100($2)	$1 = Memoria[$2+100]	Dato de memoria a registro
	almacenar palabra	sw $1,100($2)	Memoria[$2+100]=$1	Dato de registro a memoria
	cargar superior inmediato	lui $1,100	$1 = 100 × 2^{16}	Carga ctes. en 16 bits superiores
Salto condicional	saltar sobre igual	beq $1,$2,100	si ($1 = =$2) ir a PC + 4 + 100	Test igual; salto relativo al PC
	saltar sobre no igual	bne $1,$2,100	si ($1!=$2) ir a PC + 4 + 100	Test no igual; relativo al PC
	inicializa sobre menor que	slt $1,$2,$3	si ($2 < $3)$1 = 1; si no $1 = 0	Compara menor que; comp. a 2
	inicializa menor que inmediato	slti $1,$2,100	si ($2 < 100)$1 = 1; si no $1 = 0	Compara constante menor que; comp. a 2
	inicializa menor que sin signo	**sltu $1,$2,$3**	**si ($2 < $3) $1 = 1; si no $1 = 0**	**Compara menor que; no natural**
	inicializa menor que inmediato sin signo	**sltiu $1,$2,100**	**si ($2 < 100)$1 = 1; si no $1 = 0**	**Compara < constante; natural**
Bifurcación incondicional	bifurcar	j 10000	ir a 10000	Bifurca a dirección destino
	bif. registro	jr $31	ir a $31	Para cambiar, vuelta de procedimiento
	bif. y enlazar	jal 10000	$31 = PC + 4; ir a 10000	Para llamada a procedimiento

4.-OPERACIONES LOGICAS

- Necesidad de operar sobre campos de bits (1 o varios) de una palabra
  - p. ej : carácteres (8 bits) que tiene una palabra
  - Necesidad de nuevas instrucciones para enpaquetamiento y desempaquetamiento de bits
  - Desplazamientos (Shifts)
    - Desplazamiento lógico a derecha (Shift right logical)
    - Desplazamiento lógico a izquierda (Shift left logical)

sll $10, $16, 8 # $10 = $16 despl. 8 veces

  - En lenguaje maquina de la

op	rs	rt	rd	shamt	funct
0	0	16	10	8	0

  - Shamt (Shift amount) ≡ nº de desplazamiento

- Instrucciones por aislar o colocar campos de bits
  - Instrucción
    - **and** y **or**

and $8, $9, $10 # $8 = $9 AND $10

- Operaciones lógicas de C y sus correspondientes en ensamblador

Operaciones lógicas	Operadores C	Instrucciones MIPS
Desplazamiento a la izquierda	<<	sll
Desplazamiento a la derecha	>>	srl
AND	&	and, andi
OR	|	or, ori

- Los operandos inmediatos se expanden con CEROS

• Arquitectura MIPS revelada

Operandos MIPS

Nombre	Ejemplo	Comentarios
32 registros	$0, $1, $2, ..., $31	Posiciones rápidas para datos. En MIPS los datos deben estar en registros para realizar la aritmética. El registro $0 de MIPS siempre es igual a 0. El registro $1 se reserva para que el ensamblador maneje las pseudoinstrucciones y grandes constantes.
2^{30} palabras de memoria	Memoria[0], Memoria[4], ..., Memoria[4294967292]	Accedidas sólo por instrucciones de transferencia de datos. MIPS utiliza sólo direcciones de bytes, así las palabras secuenciales difieren en 4. La memoria contiene estructuras de datos, tales como arrays, y registros derramados, como los salvados en las llamadas a los procedimientos

Lenguaje ensamblador MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	sumar	add $1,$2,$3	$1 = $2 + $3	3 operandos; excepción posible
	restar	sub $1,$2,$3	$1 = $2 − $3	3 operandos; excepción posible
	sumar inmediato	addi $1,$2,100	$1 = $2 + 100	+ constante; excepción posible
	sumar sin signo	addu $1,$2,$3	$1 = $2 + $3	3 operandos; no excepciones
	restar sin signo	subu $1,$2,$3	$1 = $2 − $3	3 operandos; no excepciones
	sumar inmediato sin signo	addiu $1,$2,100	$1 = $2 + 100	+ constante; no excepciones
	transferir desde reg. coprocesador	mfc0 $1,$epc	$1 = $epc	Usado para conseguir excepción PC
Lógicas	**and**	**and $1,$2,$3**	**$1 = $2 & $3**	**3 reg. operandos; AND Logica**
	or	**or $1,$2,$3**	**$1 = $2|$3**	**3 reg. operandos; OR Logica**
	and inmediato	**and $1,$2,100**	**$1 − $2&100**	**reg. Logica AND, constante**
	or inmediato	**or $1,$2,$100**	**$1 = $2|100**	**reg. Logica OR, constante**
	desplazamiento lógico a la derecha	**sll $1,$2,$10**	**$1 = $2 << 10**	**Desplaza izquierda por constante**
	desplazamiento lógico a la izquierda	**srl $1,$2,$10**	**$1 = $2 >> 10**	**Desplaza derecha por constante**
Transferencia de datos	cargar palabra	lw $1,100($2)	$1 = Memoria[$2+100]	Dato de memoria a registro
	almacenar palabra	sw $1,100($2)	Memoria[$2+100]=$1	Dato de registro a memoria
	cargar superior inmediato	lui $1,100	$1 = 100 × 2^{16}	Carga ctes. en 16 bits superiores
Salto condicional	saltar sobre igual	beq $1,$2,100	si ($1 = =$2) ir a PC + 4 + 100	Test igual; salto relativo al PC
	saltar sobre no igual	bne $1,$2,1000	si ($1!=$2) ir a PC + 4 + 100	Test no igual; relativo al PC
	inicializa sobre menor que	slt $1,$2,$3	si ($2 < $3)$1 = 1; si no $1 = 0	Compara menor que; comp. a 2
	inicializa menor que inmediato	slti $1,$2,100	si ($2 < 100)$1 = 1; si no $1 = 0	Compara constante menor que; comp. a 2
	inicializa menor que sin signo	**sltu $1,$2,$3**	**si ($2 < $3) $1 = 1; si no $1 = 0**	**Compara menor que; número natural**
	inicializa menor que inmediato sin signo	**sltiu $1,$2.100**	**si ($2 < 100)$1 = 1; si no $1 = 0**	**Compara < constante, número natural**
Bifurcación incondicional	bifurcar	j 10000	ir a 10000	Bifurca a dirección destino
	bif. registro	jr $31	ir a $31	Para cambiar, vuelta de procedimiento
	bif. y enlazar	jal 10000	$31 = PC + 4; ir a 10000	Para llamada a procedimiento

5.- CONSTRUCCION DE UNA UNIDAD ARITMETICO LOGICA

- ALU: Bloque que realiza las operaciones
    - Aritméticas y Lógicas
    - Necesidad de una ALU de 32bits
        - Diseñamos una de 1 bit
        - 32 ALUs de 1 bits

- Elementos básicos y compartimientos

1. Puerta and ($c = a \cdot b$)

a	b	$c = a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

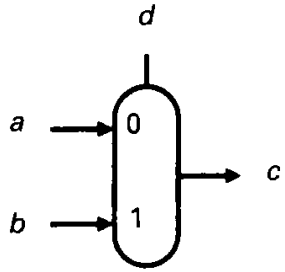
2. Puerta or ($c = a + b$)

a	b	$c = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

3. Inversor ($c = \bar{a}$)

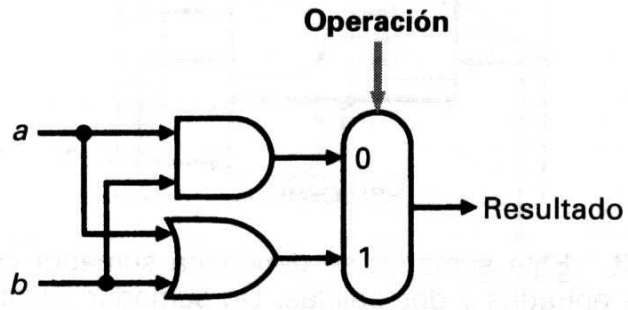
a	$c = \bar{a}$
0	1
1	0

4. Multiplexor
(si $d = = 0$ $c = a$;
sino $c = b$)

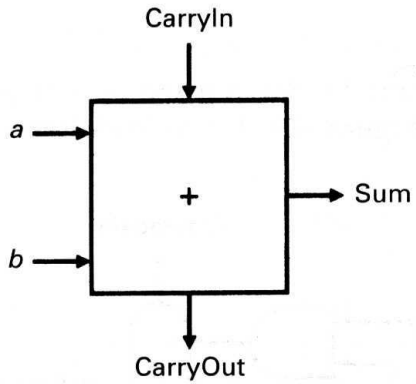


d	c
0	*a*
1	*b*

- ALU de 1 bit.
  - Bloque Lógico AND y OR



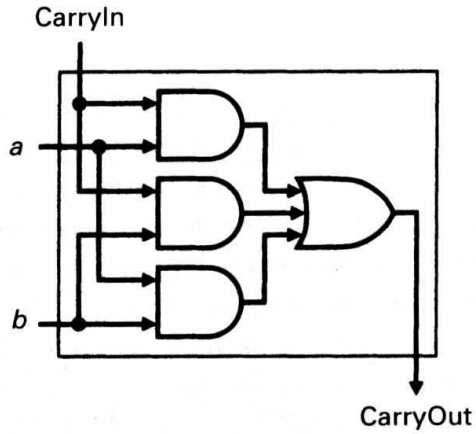
- Bloque Aritmético (sumador completo)



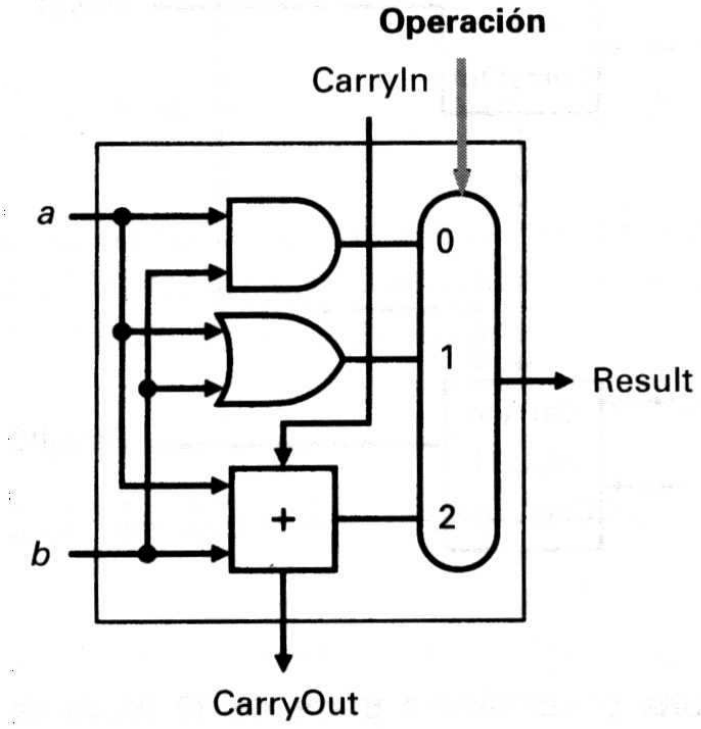
- Tabla del sumador

Entradas			Salidas		Comentarios
a	*b*	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{dos}$
0	0	1	0	1	$0 + 0 + 1 = 01_{dos}$
0	1	0	0	1	$0 + 1 + 0 = 01_{dos}$
0	1	1	1	0	$0 + 1 + 1 = 10_{dos}$
1	0	0	0	1	$1 + 0 + 0 = 01_{dos}$
1	0	1	1	0	$1 + 0 + 1 = 10_{dos}$
1	1	0	1	0	$1 + 1 + 0 = 10_{dos}$
1	1	1	1	1	$1 + 1 + 1 = 11_{dos}$

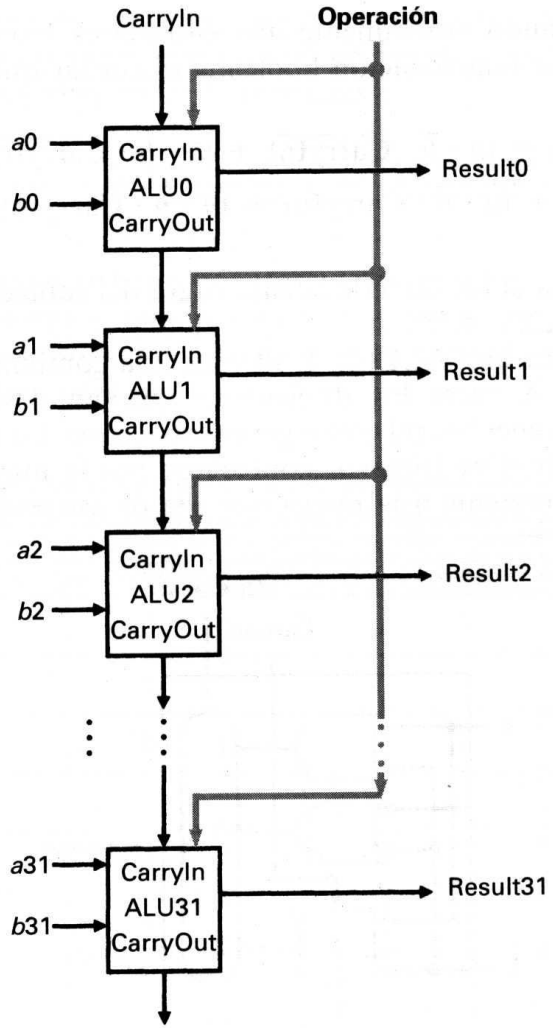
- Bloque sumador



- ALU completa

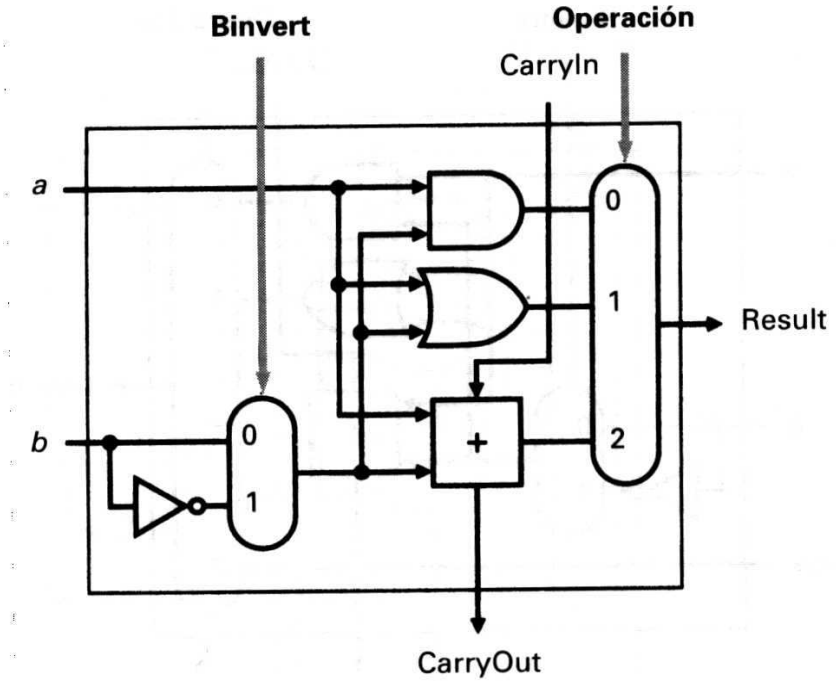


- ALU de 32 bits



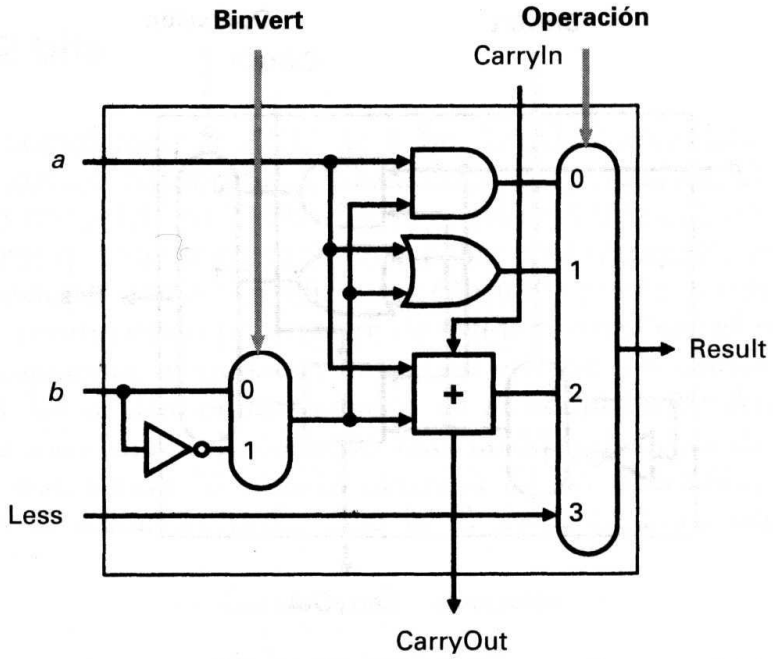
- ALU de 1 bit con las Operaciónes

a AND b; a OR b; a + b ; a + b'



- Confección de la ALU de 32 bits para MIPS

  - Necesidad de incluir hardware para la operación:
    Inicializar sobre menor que ⇒ set-on-less-than
  - Para ello se expande el multiplexor.



- Como se realiza: Resta Rt de Rs

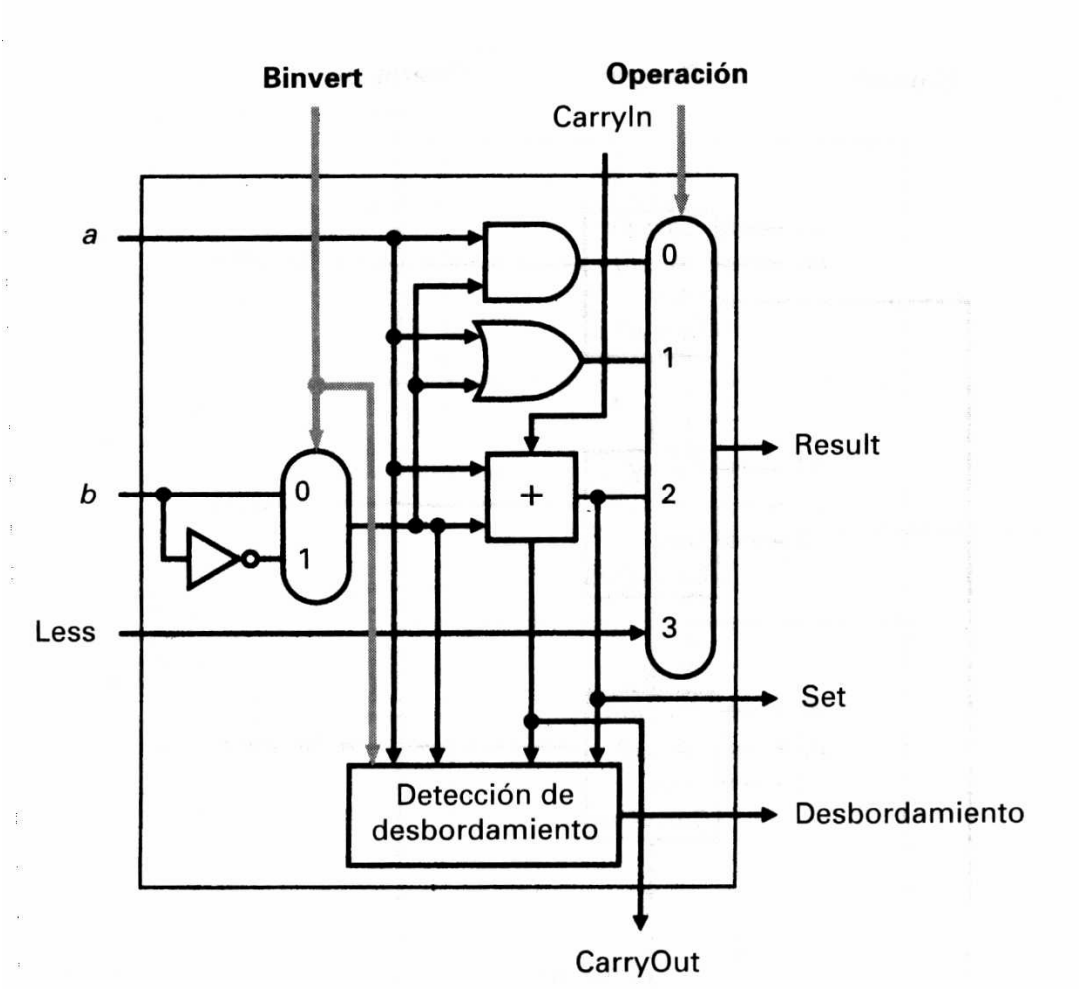
$(Rs - Rt) < 0 \Rightarrow ((Rs - Rt) + Rt) < (0+Rt) \Rightarrow Rs < Rt$

  - Si la diferencia es negativa pone **lsb** a 1

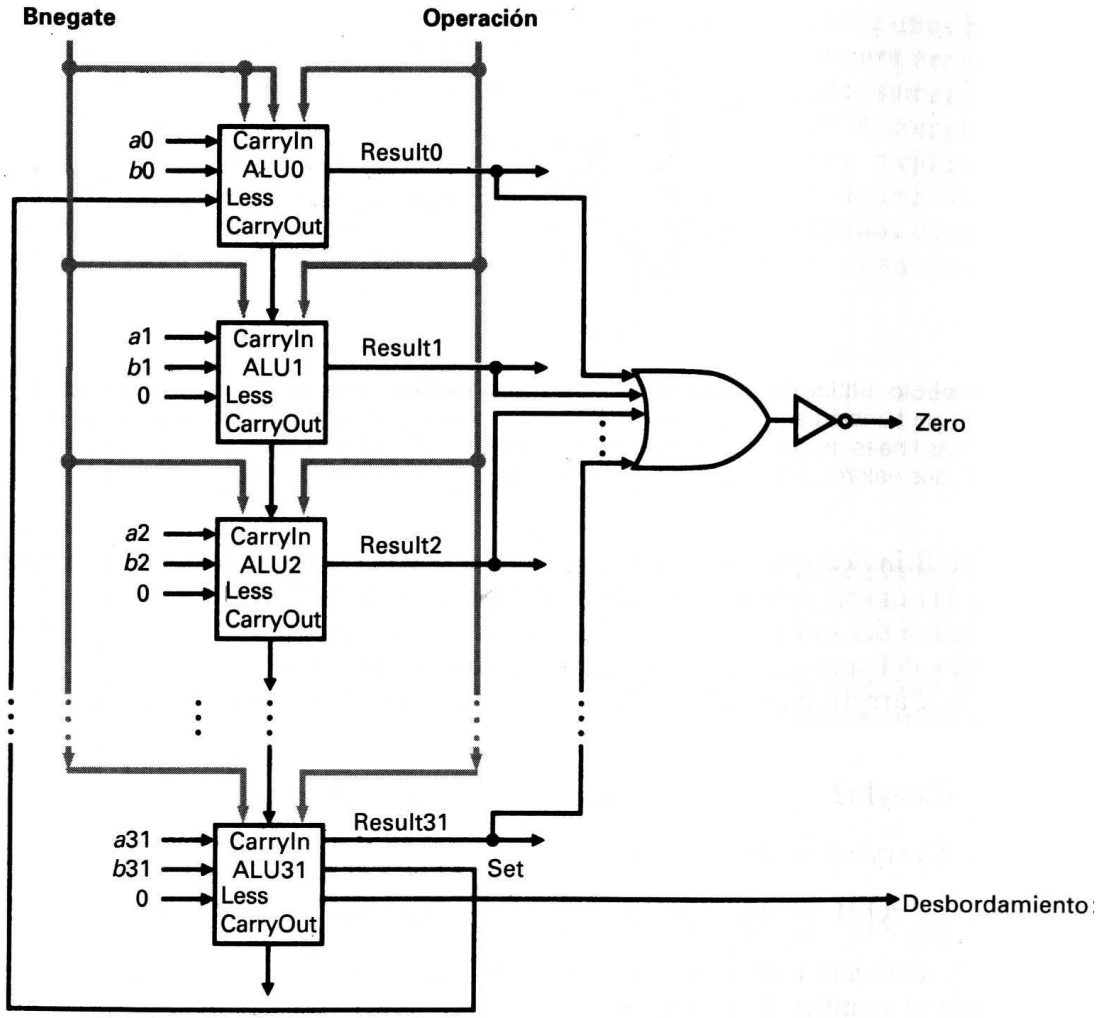
- El bit de sigo es el **msb**
  - Conectar el **msb** de salida del sumador al **lsb**.
  - La salida del msb para esa comparación no es la salida del sumador

- ALU especial para el bit más significativo
  - Incluye el desbordamiento

- Necesidad de añadir hardware por los saltos condicionados.
  - Funcion Z = NOR de los bits de salida de la ALU

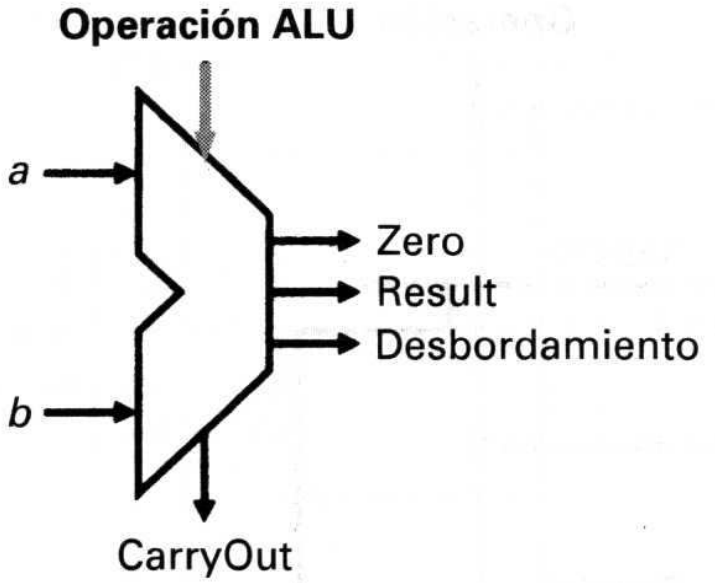


- ALU completa de 32 bits con la función Z



- Bloque ALU y función que realiza.

Líneas de control de la ALU	Función
000	And
001	Or
010	Sumar
110	Restar
111	Inicializar sobre menor que



- Anticipación de acarreo
  - El resultado de la suma no es valido hasta la propagación de acarreo
    - Lentitud (proporcional al nº de bits)
    - Forma de obtener el acarreo

$$g_i = a_i\, b_i \quad \text{(generador de acareo)}$$
$$p_i = a_i + b_i \quad \text{(propagador de acarreo)}$$

$$
\begin{aligned}
c1 &= g0 + (p0 \cdot c0) \\
c2 &= g1 + (p1 \cdot g0) + (p1 \cdot p0 \cdot c0) \\
c3 &= g2 + (p2 \cdot g1) + (p2 \cdot p1 \cdot g0) + (p2 \cdot p1 \cdot p0 \cdot c0) \\
c4 &= g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) \\
&\quad + (p3 \cdot p2 \cdot p1 \cdot p0 \cdot c0)
\end{aligned}
$$

- Todos los acarreos se generan a partir de los bits de los sumandos y el acarreo inicial
- Acarreo final Cd con mayor rapidez

- Obtención del acarreo a partir del acarreo de sumadores de 4 bits

$$
\begin{aligned}
P0 &= p3 \cdot p2 \cdot p1 \cdot p0 \\
P1 &= p7 \cdot p6 \cdot p5 \cdot p4 \\
P2 &= p11 \cdot p10 \cdot p9 \cdot p8 \\
P3 &= p15 \cdot p14 \cdot p13 \cdot p12
\end{aligned}
$$

$$
\begin{aligned}
G0 &= g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) \\
G1 &= g7 + (p7 \cdot g6) + (p7 \cdot p6 \cdot g5) + (p7 \cdot p6 \cdot p5 \cdot g4) \\
G2 &= g11 + (p11 \cdot g10) + (p11 \cdot p10 \cdot g9) + (p11 \cdot p10 \cdot p9 \cdot p8) \\
G3 &= g15 + (p15 \cdot g14) + (p15 \cdot p14 \cdot g13) + (p15 \cdot p14 \cdot p13 \cdot g12)
\end{aligned}
$$

- Entonces la ecuación del acarreo anticipado sera:

$$
\begin{aligned}
C1 &= G0 + (P0 \cdot c0) \\
C2 &= G1 + (P1 \cdot G0) + (P1 \cdot P0 \cdot c0) \\
C3 &= G2 + (P2 \cdot G1) + (P2 \cdot P1 \cdot G0) + (P2 \cdot P1 \cdot P0 \cdot c0) \\
C4 &= G3 + (P3 \cdot G2) + (P3 \cdot P2 \cdot G1) + (P3 \cdot P2 \cdot P1 \cdot G0) \\
&\quad + (P3 \cdot P2 \cdot P1 \cdot P0 \cdot c0)
\end{aligned}
$$

- Otras cuestiones sobre la ALU
  - Implementación del sumador con puertas OR . EXCLUSIVA

    Sum = a ⊕ b ⊕ C_{in}

  - El desplazamiento no se incluye en la ALU
  - Circuitos desplazador (barrel shifter)

6.- MULTIPLICACION.

- Ejemplo de producto de dos números binarios

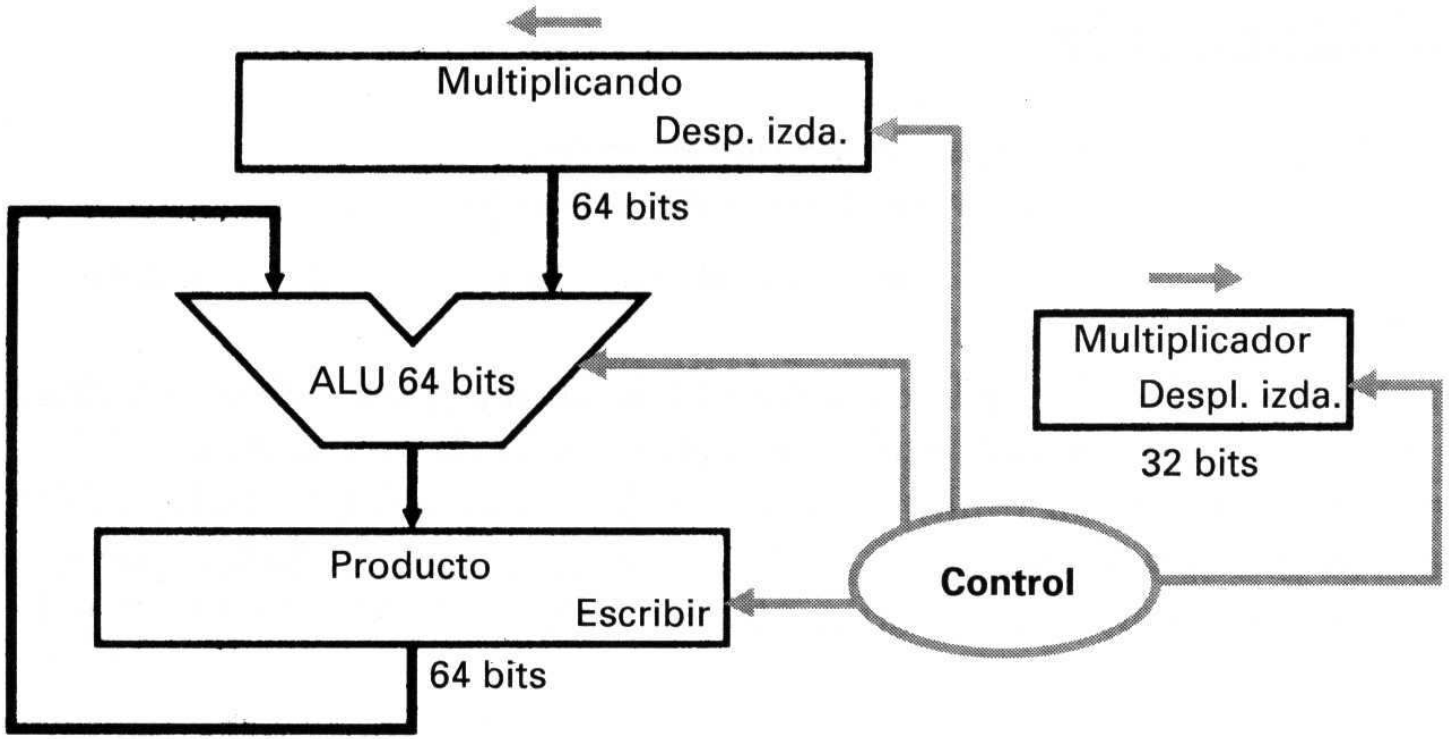
```
    1000
    1001
    1000
   0000
  0000
 1000
01001000
```

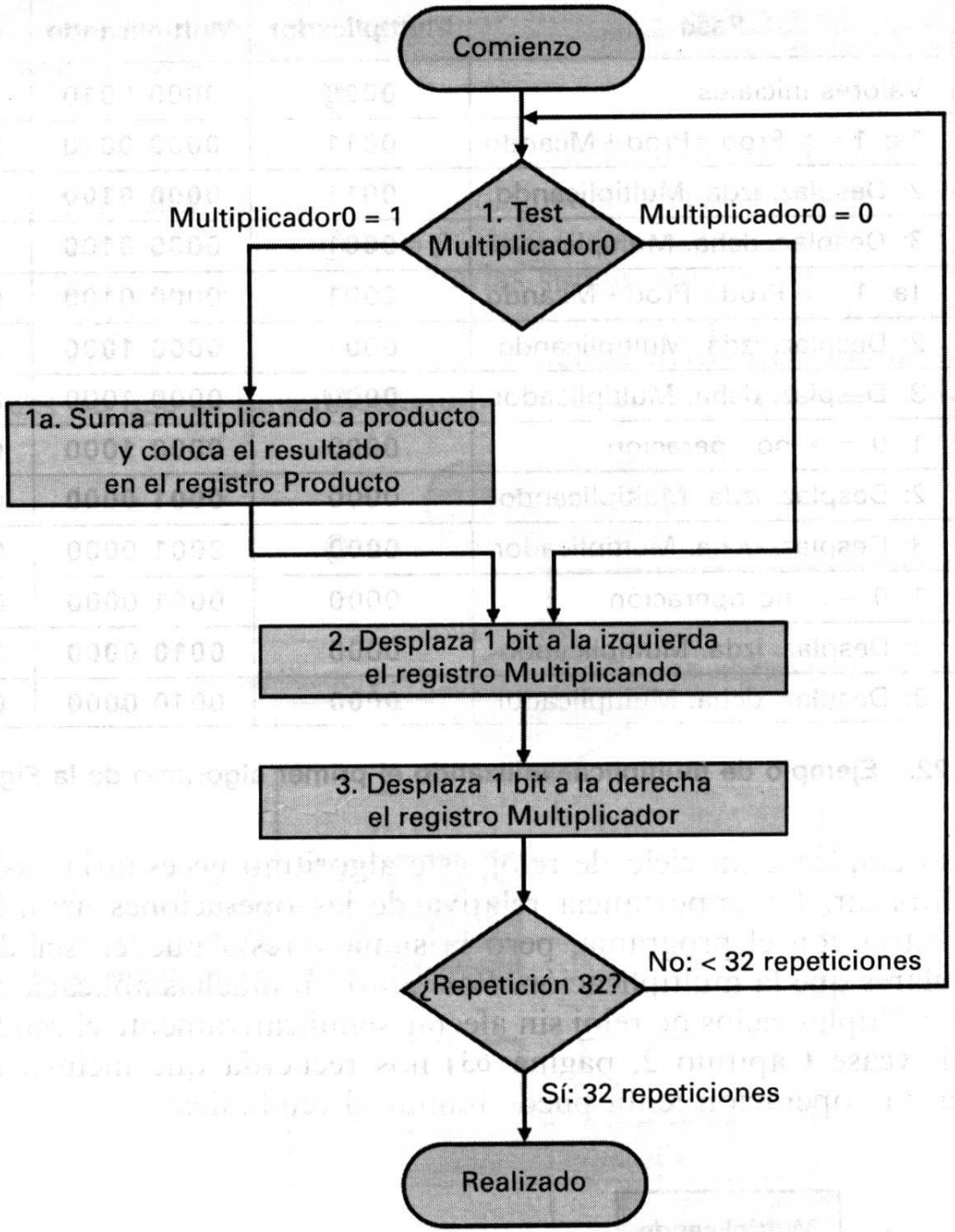
  - En general el producto de dos nº de n bit produce un resultado de 2n bit

  - En binario si el bit **i** del multiplicando es
    - 1: se suma el multiplicando desplazando i veces al resultado parcial
    - 0: no se le suma nada

  - En ambos caso se desplaza el multiplicando cada vez que se avanza en el análisis de los bits del multiplicador

- Primera iteración del hardware y el algoritmo de multiplicación
  - Hardware y Algoritmo asociado

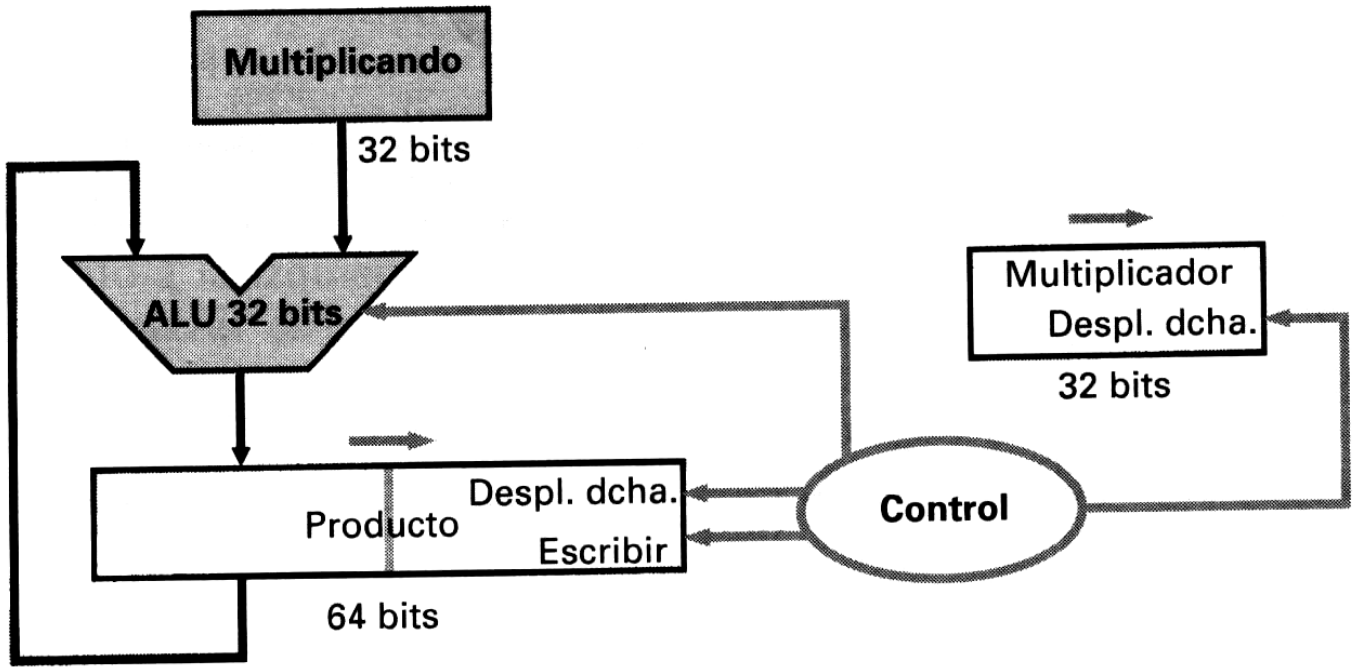


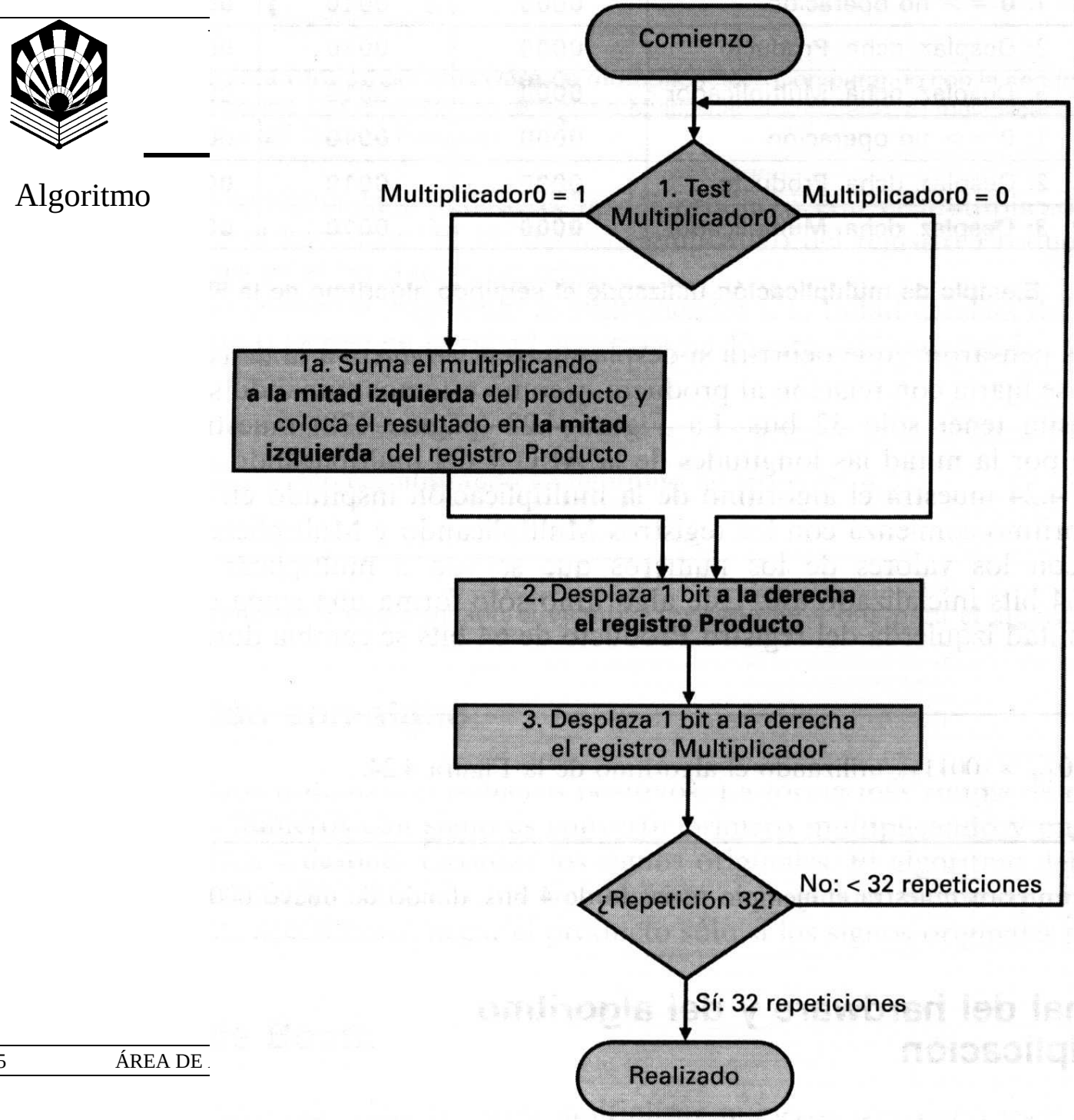

Comienzo
1. Test Multiplicador0
Multiplicador0 = 1
Multiplicador0 = 0
1a. Suma multiplicando a producto y coloca el resultado en el registro Producto
2. Desplaza 1 bit a la izquierda el registro Multiplicando
3. Desplaza 1 bit a la derecha el registro Multiplicador
¿Repetición 32?
No: < 32 repeticiones
Sí: 32 repeticiones
Realizado

- Ejemplo:

Iteración	Paso	Multiplicador	Multiplicando	Producto
0	Valores iniciales	**0010**	0000 0010	0000 0000
1	1a: 1 = > Prod = Prod + Mcando	0011	0000 0010	**0000 0010**
	2: Desplaz. izda. Multiplicando	0011	**0000 0100**	0000 0010
	3: Desplaz. dcha. Multiplicador	**0011**	0000 0100	0000 0010
2	1a: 1 = > Prod = Prod + Mcando	0001	0000 0100	**0000 0110**
	2: Desplaz. izda. Multiplicando	0001	**0000 1000**	0000 0110
	3: Desplaz. dcha. Multiplicador	**0010**	0000 1000	0000 0110
3	1: 0 = > no operación	0000	0000 1000	0000 0110
	2: Desplaz. izda. Multiplicando	0000	**0001 0000**	0000 0110
	3: Desplaz. dcha. Multiplicador	**0001**	0001 0000	0000 0110
4	1: 0 = > no operación	0000	0001 0000	0000 0110
	2: Desplaz. izda. Multiplicando	0000	**0010 0000**	0000 0110
	3: Desplaz. dcha. Multiplicador	**0000**	0010 0000	0000 0110

- Segunda iteración del hardware y del Algoritmo de multiplicación.
  - Reducir la suma a n bits (p.ej. 32 bits) y de algunos registros
  - Hardware





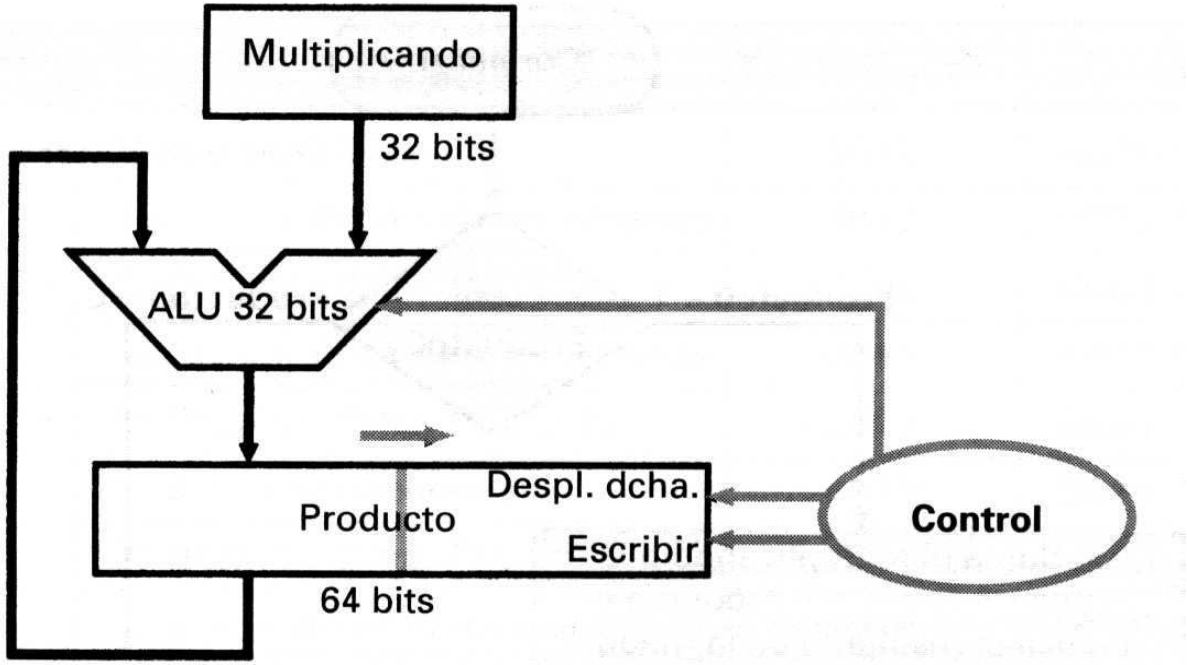
- Algoritmo

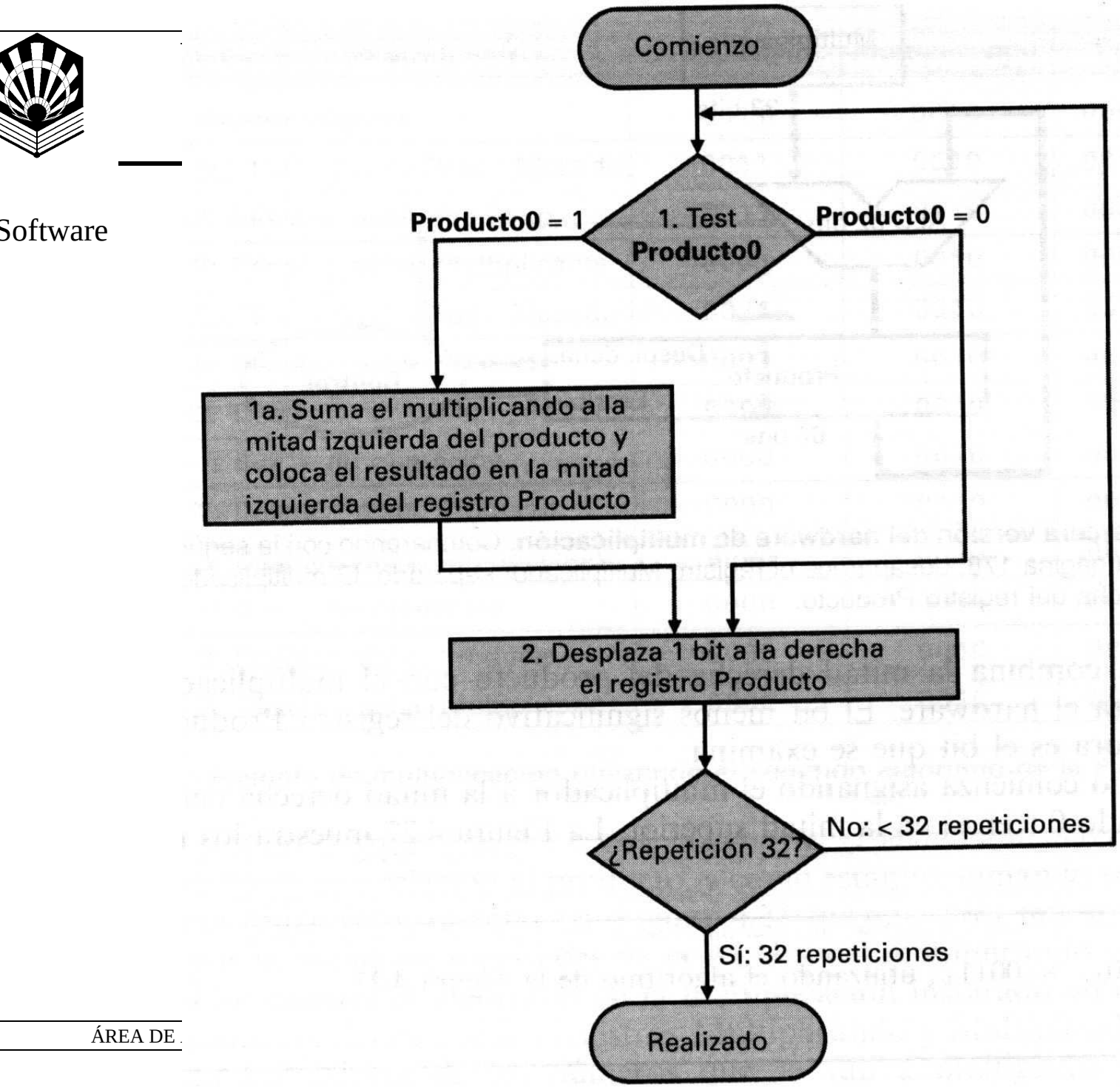
Comienzo
1. Test Multiplicador0
Multiplicador0 = 1
Multiplicador0 = 0
1a. Suma el multiplicando a la mitad izquierda del producto y coloca el resultado en la mitad izquierda del registro Producto
2. Desplaza 1 bit a la derecha el registro Producto
3. Desplaza 1 bit a la derecha el registro Multiplicador
¿Repetición 32?
No: < 32 repeticiones
Sí: 32 repeticiones
Realizado

- Ejemplo

Iteración	Paso	Multiplicador	Multiplicando	Producto
0	Valores iniciales	001①	0010	0000 0000
1	1a: 1 => Prod = Prod + Mcando	0011	0010	**0010** 0000
	2: Desplaz. dcha. Producto	0011	0010	**0001 0000**
	3: Desplaz. dcha. Multiplicador	**000①**	0010	0001 0000
2	1a: 1 => Prod = Prod + Mcando	0001	0010	**0011** 0000
	2: Desplaz. dcha. Producto	0001	0010	**0001 1000**
	3: Desplaz. dcha. Multiplicador	**000⓪**	0010	0001 1000
3	1: 0 => no operación	0000	0010	0001 1000
	2: Desplaz. dcha. Producto	0000	0010	**0000 1100**
	3: Desplaz. dcha. Multiplicador	**000⓪**	0010	0000 1100
4	1: 0 => no operación	0000	0010	0000 1100
	2: Desplaz. dcha. Producto	0000	0010	**0000 0110**
	3: Desplaz. dcha. Multiplicador	**0000**	0010	0000 0110

- Tercera iteración del hardware y del Algoritmo de multiplicación
  - Disminuye el nº de registros.
  - Hardware




Comienzo
1. Test Producto0
Producto0 = 1
Producto0 = 0
1a. Suma el multiplicando a la mitad izquierda del producto y coloca el resultado en la mitad izquierda del registro Producto
2. Desplaza 1 bit a la derecha el registro Producto
¿Repetición 32?
No: < 32 repeticiones
Sí: 32 repeticiones
Realizado

- Software

- Ejemplo

Iteración	Paso	Multiplicando	Producto
0	Valores iniciales	0010	0000 001①
1	1a: 1 => Prod = Prod + Mcando	0010	**0010** 0011
	2: Desplaz. dcha. Producto	0010	**0001 000①**
2	1a: 1 => Prod = Prod + Mcando	0010	**0011** 0001
	2: Desplaz. dcha. Producto	0010	**0001 100⓪**
3	1: 0 => no operación	0010	**0001** 1000
	2: Desplaz. dcha. Producto	0010	**0000 110⓪**
4	1: 0 => no operación	0010	**0000** 1100
	2: Desplaz. dcha. producto	0010	**0000 0110**

- Multiplicación con signo

    - Primera solución:
        - Convertir los números a positivos
            - Hacer el producto sin signo (31 bit)
            - De acuerdo con los signos y la representación cambia el resultado final o no

    - Segunda solución:
        - Multiplicar independiente de los signos: Algoritmo de Booth.

- Algoritmo de BOOTH

Dada una cadena de UNOS seguidos, su aportación al resultado de la multiplicación se reduce a una resta y una suma

- Ejemplo:
    - Sea el producto de M por el número 00111000

M * (0011100) = M (0100000 – 0000100) = M 2^5 – M 2^2

- restar el multiplicando desplazado 2 veces (i) veces

- Sumar el multiplicando desplazado 5 veces (i +1) veces

- El nuevo algoritmo realiza la secuencia siguiente
  - Analiza los dos bits consecutivos del multiplicando
    - 00 no realiza suma y desplaza.
    - 11 no realiza suma y desplaza.
    - 10 realiza resta y desplaza.
    - 01 realiza suma y desplaza.
  - Inicialmente se supone que tiene un bit a ∅ de antes

- Comparación de Algoritmos

Itera-ción	Multipli-cando	Algoritmo original		Algoritmo de Booth	
		Paso	Producto	Paso	Producto
0	0010	Valores iniciales	0000 011(0)	Valores iniciales	0000 011[0 0]
1	0010	1: 0 = > no operación	0000 0110	1a: 00 = > no operación	0000 0110 0
	0010	2: Despl. dcha. Producto	**0000 001(1)**	2: Despl. dcha. Producto	**0000 001[1 0]**
2	0010	1a: 1 = > Prod = Prod + Mcando	**0010** 0011	1c: 10 = > Prod = Prod − Mcando	**1110** 0011 0
	0010	2: Despl. dcha. Producto	**0000 000(1)**	2: Despl. dcha. Producto	**1111 000[1 1]**
3	0010	1a: 1 = > Prod = Prod + Mcando	**0011** 0001	1d: 11 = > no operación	1111 0001 1
	0010	2: Despl. dcha. Producto	**0001 100(0)**	2: Despl. dcha. Producto	**1111 000[0 1]**
4	0010	1: 0 = > no operación	0001 1000	1b: 01 = > Prod = Prod + Mcando	**0001** 1000 1
	0010	2: Despl. dcha. Producto	**0000 1100**	2: Despl. dcha. Producto	**0000 1100 0**

- Ojo: Operaciones de desplazamientos aritméticos a derecha

- Validez para los números negativos (Complemento a dos)

Iteración	Paso	Multiplicando	Producto
0	Valores iniciales	0010	0000 110[1 0]
1	1c: 10 = > Prod = Prod − Mcando	0010	**1110** 1101 0
	2: Desplaz. dcha. Producto	0010	**1111 011[0 1]**
2	1b: 01 = > Prod = Prod + Mcando	0010	**0001** 0110 1
	2: Desplaz. dcha. Producto	0010	**1111 101[1 0]**
3	1c: 10 = > Prod = Prod − Mcando	0010	**1110** 1011 0
	2: Desplaz. dcha. Producto	0010	**1111 010[1 1]**
4	1d: 11 = > no operación	0010	1111 0101 1
	2: Desplaz. dcha. producto	0010	**1111 1010 1**

- Multiplicación por constantes en el programa
    - Algunos compiladores lo reduce a desplazamientos, sumas y restas.
        - Equivalente a aplicar el algoritmo de Booth.
        - Lo suelen hacer con constantes pequeñas
- Multiplicaciones reales en MIPS
    - Tiene dos instrucciones:
        - **mult**: multiplicación con signo
        - **multu**: multiplicación sin signo
    - Usa un par de registros (32 +32 bit) **Hi** y **Lo** para el resultado
    - Instrucciones para recuperar valor de Hi y Lo
        - **mflo** (move from lo)
        - **mfhi** (move from hi)
            - Ambas instrucciones ignoran el desbordamiento, comprobarlo por software

7.-DIVISION.

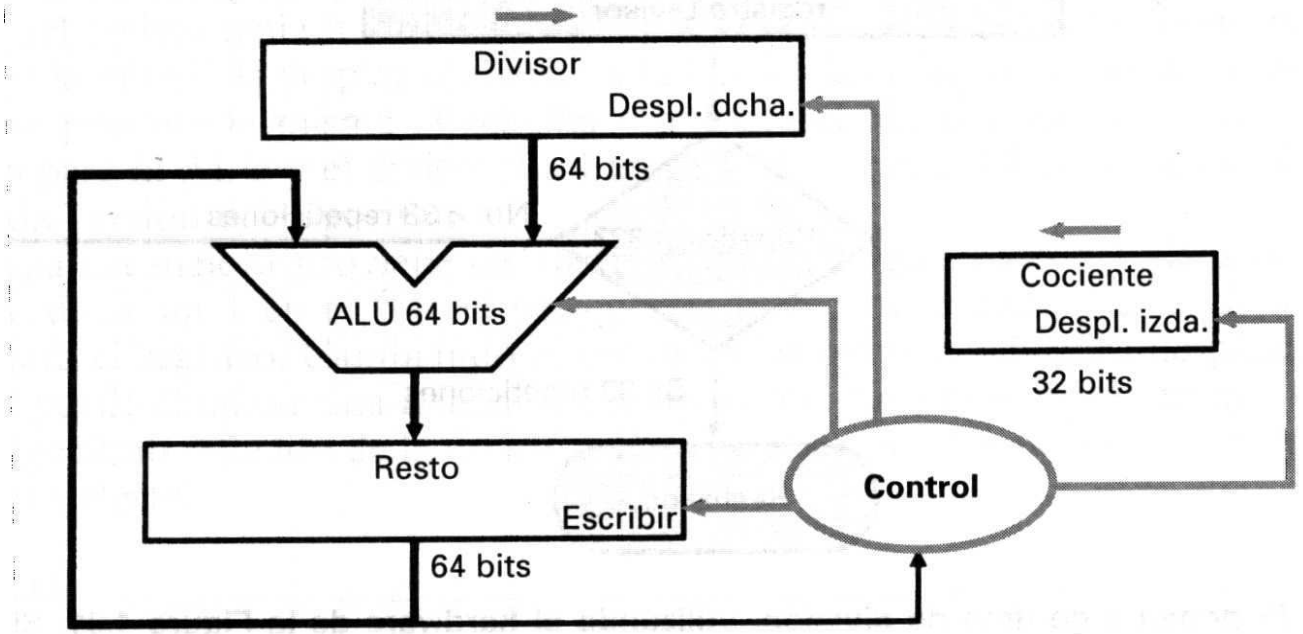
- Operación: Dividendo = Cociente * Divisor + Resto

  - Ejemplo: división números binarios

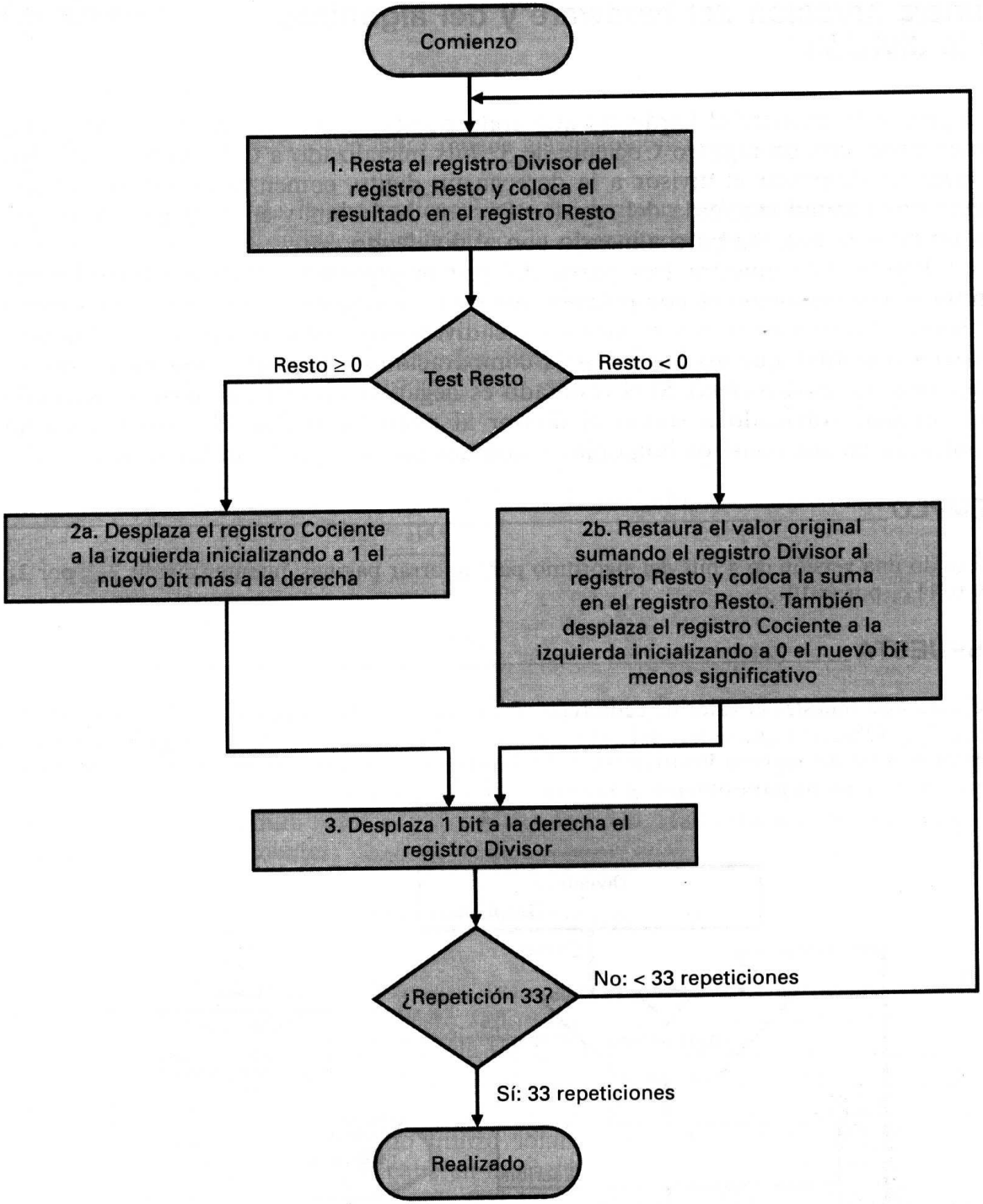
Divisor 1000

1001010 Dividendo Cociente 1001
-1000
00010
101
1010
-1000
Resto 10

- Primera iteración del hardware y del algoritmo de división
  - Hardware necesario y Algoritmo



Cociente: 32 bits (iniciar a 0); Dividiendo-Resto: 64 bit (iniciar dividendo)
Divisor : 64 bit (iniciar divisor * 2^{32})

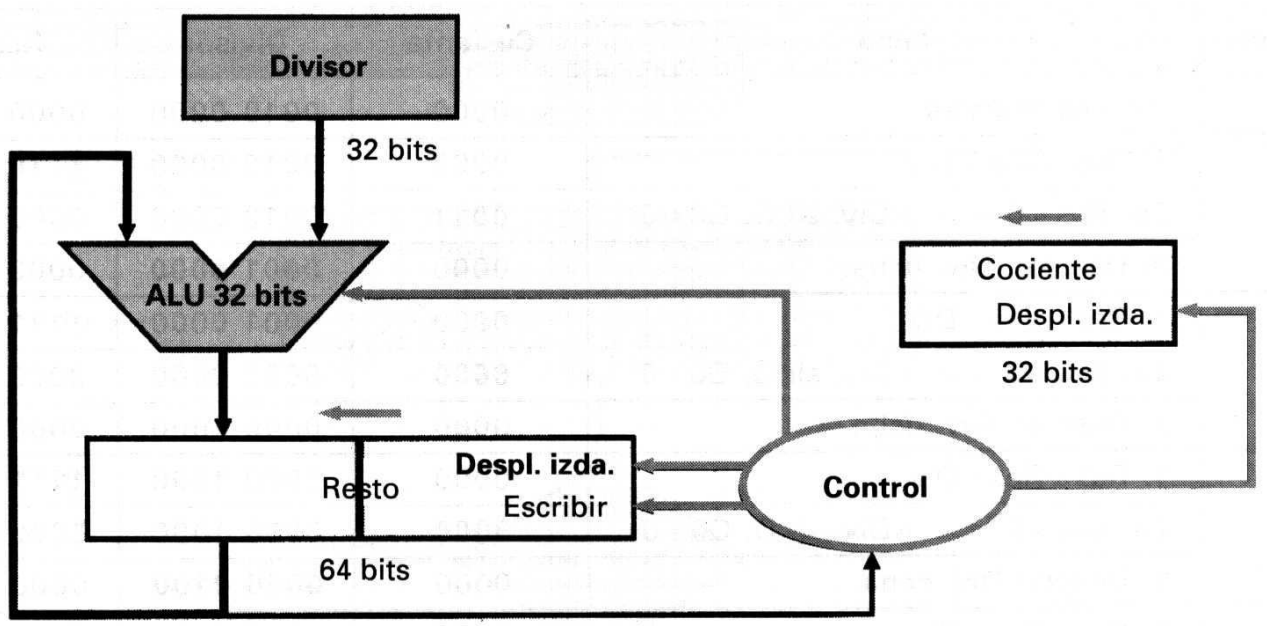

Comienzo
1. Resta el registro Divisor del registro Resto y coloca el resultado en el registro Resto
Test Resto
Resto ≥ 0
Resto < 0
2a. Desplaza el registro Cociente a la izquierda inicializando a 1 el nuevo bit más a la derecha
2b. Restaura el valor original sumando el registro Divisor al registro Resto y coloca la suma en el registro Resto. También desplaza el registro Cociente a la izquierda inicializando a 0 el nuevo bit menos significativo
3. Desplaza 1 bit a la derecha el registro Divisor
¿Repetición 33?
No: < 33 repeticiones
Sí: 33 repeticiones
Realizado

- El por qué del método de restauración
- Ejemplo: Contenido de los registros

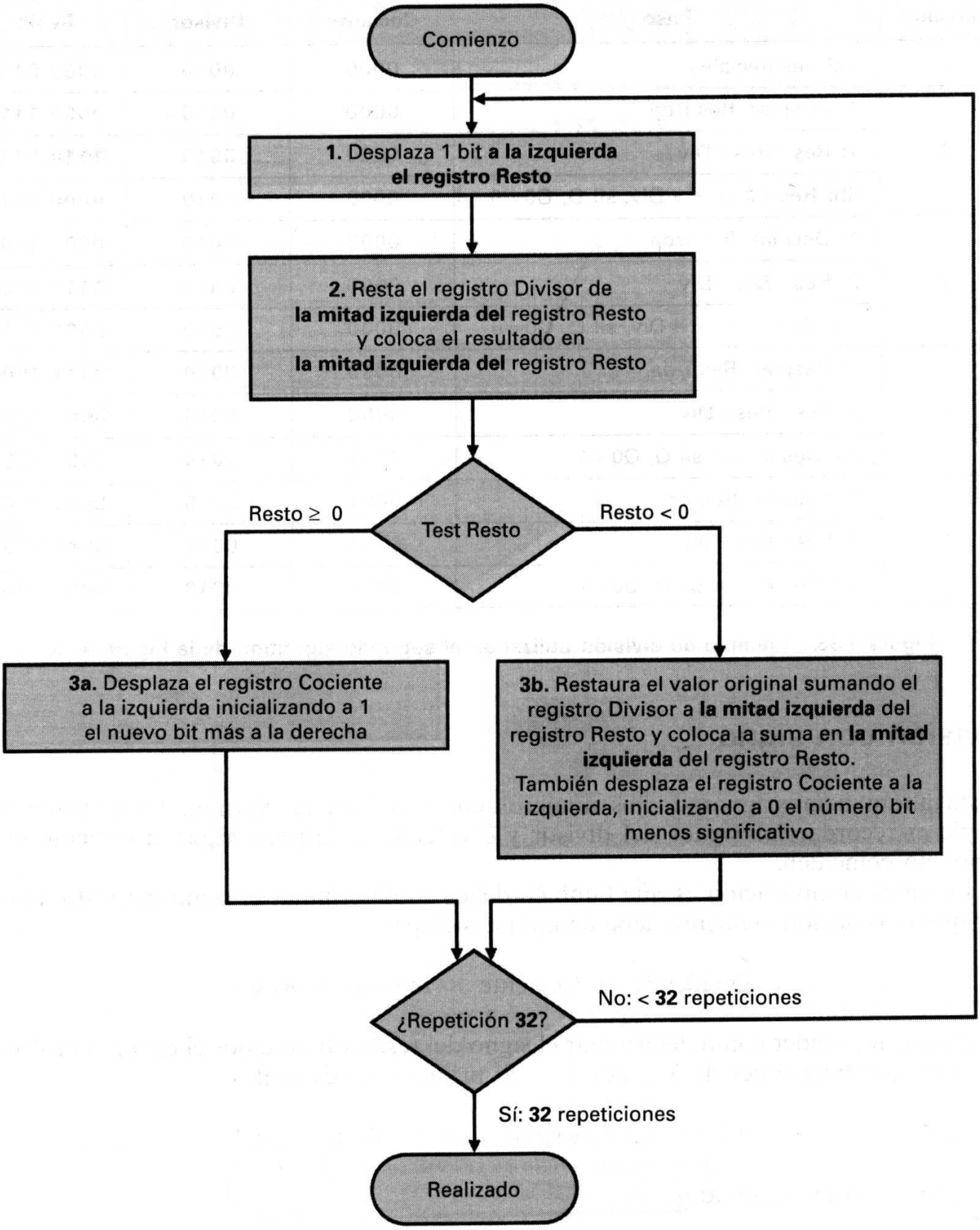
Iteración	Paso	Cociente	Divisor	Resto
0	Valores iniciales	0000	0010 0000	0000 0111
1	1: Res = Res − Div	0000	0010 0000	**①110 0111**
	2b: Res < 0 => +Div, sll Q, Q0 = 0	**0011**	0010 0000	**0000 0111**
	3: Desplaz. Div. dcha.	0000	**0001 0000**	0000 0111
2	1: Res = Res − Div	0000	0001 0000	**①111 0111**
	2b: Res < 0 => +Div, sll Q, Q0 = 0	**0000**	0001 0000	**0000 0111**
	3: Desplaz. Div. dcha.	0000	0000 1000	0000 0111
3	1: Res = Res − Div	0000	0000 1000	**①111 1111**
	2b: Res < 0 => +Div, sll Q, Q0 = 0	**0000**	0000 1000	**0000 0111**
	3: Desplaz Div. dcha.	0000	**0000 0100**	0000 0111
4	1: Res = Res − Div	0000	0000 0100	**⓪000 0011**
	2a: Res < 0 => sll Q, Q0 = 1	**0001**	0000 0100	0000 0011
	3: Desplaz. Div. dcha.	0001	**0000 0010**	0000 0011
5	1: Res = Res − Div	0001	0000 0010	**⓪000 0001**
	2a: Res < 0 => sll Q, Q0 = 1	**0011**	0000 0010	0000 0001
	3: Desplaz. Div. dcha.	0011	**0000 0001**	0000 0001

- Segunda versión del Algoritmo de división y hardware

  - Se realiza inicialmente un desplazamiento (sobreflujo)
  - La mitad del divisor no tiene información valida
  - Desplazar el dividiendo a la izquierda ←→ al divisor a la derecha.
  - La ALU solo de 32 bits.
  - Nuevo hardware necesario



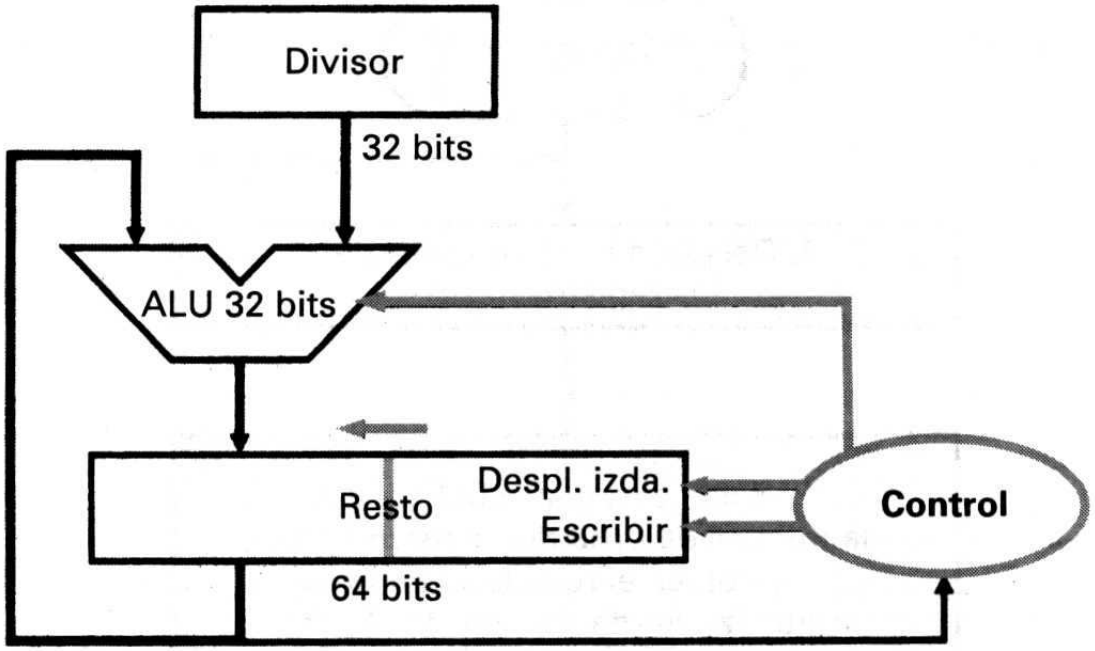
- Algoritmo Correspondiente



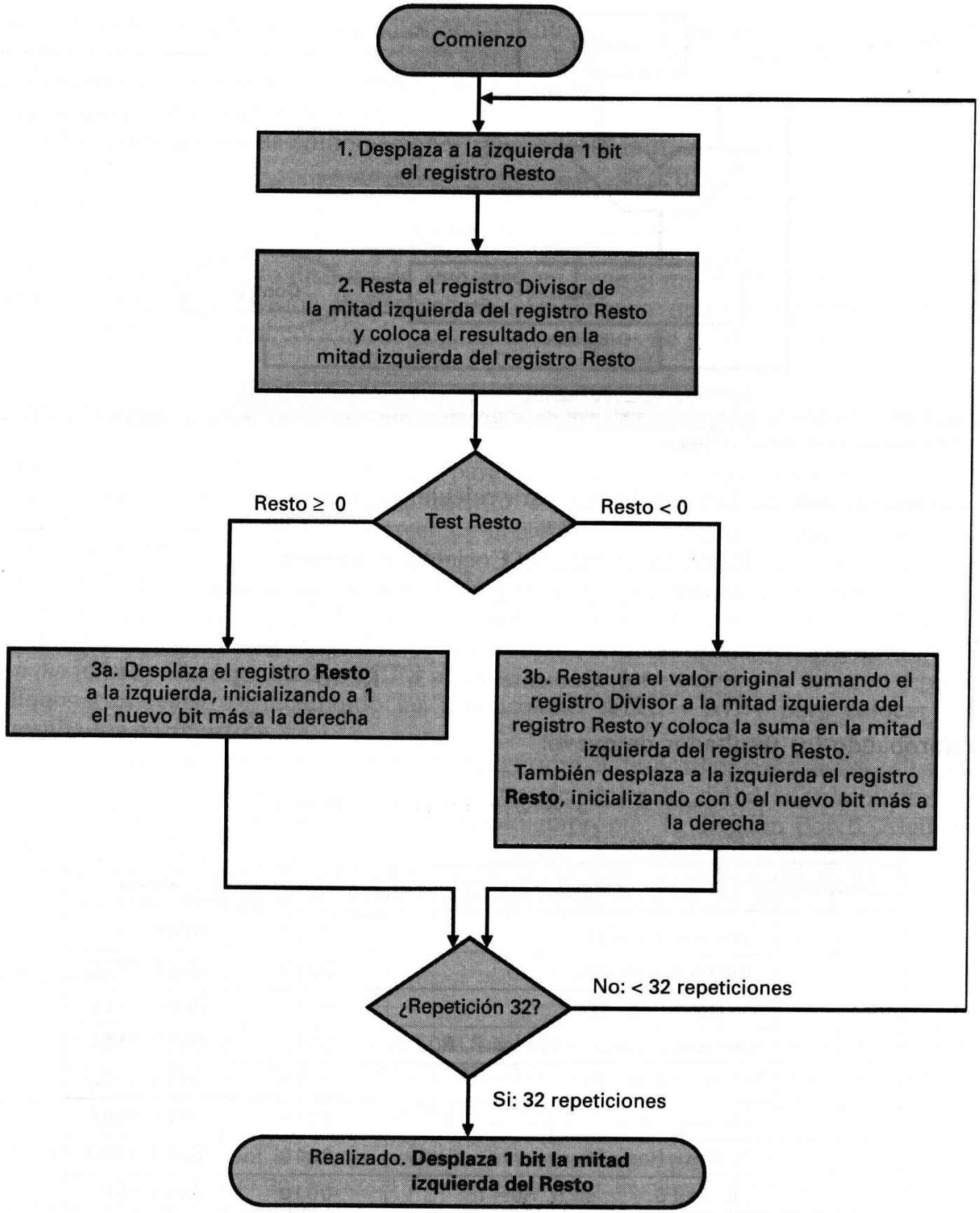
- Ejemplo:

Iteración	Paso	Cociente	Divisor	Resto
0	Valores iniciales	0000	0010	0000 0111
1	1: Desplaz. Res izda.	0000	0010	**0000 1110**
	2: Res = Res − Div	0000	0010	**①110** 1110
	3b: Res < 0 => +Div, sll Q, Q0 = 0	0000	0010	**0000** 1110
2	1: Desplaz. Res izda.	0000	0010	0001 **1100**
	2: Res = Res − Div	0000	0010	**①111** 1100
	3b: Res < 0 => +Div, sll Q, Q0 = 0	0000	0010	**0001** 1100
3	1: Desplaz. Res izda.	0000	0010	**0011 1000**
	2: Res = Res − Div	0000	0010	**⓪001** 1000
	3a: Res 0 => sll Q, Q0 = 1	0001	0010	0001 1000
4	1: Desplaz. Res izda.	0001	0010	**0011 0000**
	2: Res = Res − Div	0001	0010	**⓪001** 0000
	3a: Res 0 => sll Q, Q0 = 1	0011	0010	0001 0000

- Versión final del hardware y del algoritmo de la división
  - Aprovecha parte del dividendo /resto para introducir el cociente.
  - Nuevo hardware necesario.



- Algoritmo correspondiente.



- Ejemplo:

Iteración	Paso	Divisor	Resto
0	Valores iniciales	0010	0000 0111
	Desplaz. Res izda. 1	0010	**0000 1110**
1	1: Res = Res − Div	0010	**0**001 1110
	2b: Res < 0 => +Div, sll R, R0 = 0	0010	**0001 1100**
2	1: Res = Res − Div	0010	**1**111 1100
	2b: Res < 0 => +Div, sll R, R0 = 0	0010	**0011 1000**
3	1: Res = Res − Div	0010	**0**001 1000
	2a: Res 0 => sll R, R0 = 1	0010	**0011 0001**
4	1: Res = Res − Div	0010	**0**001 0001
	2a: Res 0 => sll R, R0 = 1	0010	**0010 0011**
	Mitad izda. de Res 1 a la dcha.	0010	**0001** 0011

- División con signo.
    - La división de las magnitudes es identica
        - El signo: + si son iguales y - si son diferentes
        - Dividendo y resto del mismo signo
- Solución MIPS
    - Necesidad de un registro de 64 bits (Reg. Hi y Lo)
        - **Hi** → Resto
        - **Lo** → Cociente
    - Aparición de instrucciónes que operan sin o con signo en división
        - **divu**
        - **div**
    - El ensamblador MIP permite especificar otros registros
        - Auxiliado por la instruciones **mflo mhhi**
    - Ignoran desbordamiento y división por cero
        - Solución tipo software

- Arquitectura MIPS revelada.

Operandos MIPS

Nombre	Ejemplo	Comentarios
32 registros	$0, $1, $2, ..., $31 **Hi, Lo**	Posiciones rápidas para datos. En MIPS los datos deben estar en registros para realizar la aritmética. El registro $0 de MIPS siempre es igual a 0. El registro $1 se reserva para que el ensamblador maneje las pseudoinstrucciones y grandes constantes. **Hi y Lo son registros de 32 bits que contienen el resultado de la multiplicación y división.**
2^{30} palabras de memoria	Memoria[0], Memoria[4], ..., Memoria[4294967292]	Accedidas sólo por instrucciones de transferencia de datos. MIPS utiliza sólo direcciones de bytes, así las palabras secuenciales difieren en 4. La memoria contiene estructuras de datos, tales como arrays, y registros derramados, como los salvados en las llamadas a los procedimientos

Lenguaje ensamblador MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	sumar	add $1,$2,$3	$1 = $2 + $3	3 operandos; excepción posible
	restar	sub $1,$2,$3	$1 = $2 − $3	3 operandos; excepción posible
	sumar inmediato	addi $1,$2,100	$1 = $2 + 100	+ constante; excepción posible
	sumar sin signo	**addu $1,$2,$3**	**$1 = $2 + $3**	3 operandos; no excepciones
	restar sin signo	**subu $1,$2,$3**	**$1 = $2 − $3**	3 operandos; no excepciones
	sumar inmediato sin signo	**addiu $1,$2,100**	**$1 = $2 + 100**	+ constante; no excepciones
	transf. reg. cop. NO	**mfc0 $1,$epc**	**$1 = $epc**	Usado para conseguir excep. PC
	multiplicación	**mult $2,$3**	**Hi, Lo = $2 × $3**	**prod. con signo de 64 bits en Hi, Lo**
	multiplicación sin signo	**multu $2,$3**	**Hi, Lo = $2 × $3**	**prod. sin signo de 64 bits en Hi, Lo**
	división	**div $2,$3**	**Lo = $2 $3, Hi = $2 mod $3**	**Lo = cociente, Hi = resto**
	división sin signo	**divu $2,$3**	**Lo = $2 $3, Hi = $2 mod $3**	**cociente y res. sin signo**
	Transferir de Hi	**mfhi $1**	**$1 = Hi**	**usado para obtener copia de Hi**
	Transferir de Lo	**mflo 1**	**$1 = Lo**	**usado para obtener copia de Lo**
Logicas	and	and $1,$2,$3	$1 = $2 & $3	operandos 3reg; AND lógica
	or	or $1,$2,$3	$1 = $2|$3	operandos 3reg; OR lógica
	and inmediato	and $1,$2,100	$1 = $2&100	AND lógica AND reg. constante
	or inmediato	or $1,$2,100	$1 = $2|100	OR lógica reg. constante
	desplaz. lógico izda.	sll $1,$2,10	$1 = $2 << 10	Desplaz. izquierda por constante
	desplaz. lógico dcha.	srl $1,$2,10	$1 = $2 >> 10	Desplaz. derecha por constante
Transfe-rencia de datos	cargar palabra	lw $1,100($2)	$1 = Memoria[$2+100]	Dato de memoria a registro
	almacenar palabra	sw $1,100($2)	Memoria[$2+100]=$1	Dato de registro a memoria
	cargar inmediato sup.	lui $1,100	$1 = 100 × 2^{16}	Cargar ctes. en 16 bits superiores
Salto condicional	saltar sobre igual	beq $1,$2,100	si ($1 = =$2) ir a PC + 4 + 100	Test igualdad; salto rela. PC
	saltar sobre no igual	bne $1,$2,1000	si ($1!=$2) ir a PC + 4 + 100	Test no igualdad; salto rela. PC
	inic. sobre menor que	slt $1,$2,$3	si ($2 < $3)$1 = 1; si no $1 = 0	Compara menor que; comp. a 2
	inic. sobre menor que inm.	slti $1,$2,100	si ($2 < 100)$1 = 1; si no $1 = 0	Compara < cte.; comp. a 2
	inic. menor que sin signo	**sltu $1,$2,$3**	**si ($2 < $3) $1 = 1; si no $1 = 0**	Compara menor que; no natural
	inic. menor que inm. sin signo	**sltiu $1,$2,100**	**si ($2 < 100)$1 = 1; si no $1 = 0**	Compara < cte., natural
Bifurcación incondicional	bifurcar	j 10000	ir a 10000	Bifurca a dirección destino
	bif. registro	jr $31	ir a $31	Para cambiar; vuelta de proced.
	bif. y enlazar	jal 10000	$31 = PC + 4; ir a 10000	Para llamada a procedimiento

8.- PUNTO FLOTANTE

- Necesidad de representación en el computador de númro Reales
  - Notación científica:

    **M * b ****e

    - M → mantisa
      - entera, fraciona o mixta
      - debe incluir signo
    - b → base del sistema de numeración (diez, dos,...)
    - e → exponente incluyendo signo

  - Notación binaria:
    - M → combinación binaria (signo magnitud)
    - b → la base, que es 2
    - e → combinación binaria exceso E

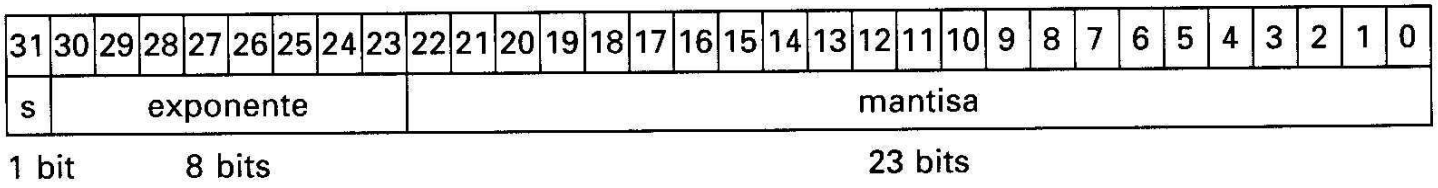
Punto flotante normalizado

$$\mathbf{0.1XXXX.\text{-}X_2 * 2^{YYYY}}$$

- Ventaja de la normalización
  - Simplifica el intercambio de datos
  - Incrementa la precisión

- Formato IEEE 754
  - Simple precisión: 32 bits.
    - 1 bit signo
    - 8 bit exponente (exceso 127)
    - 23 bit mantisa (magnitud)

$$\mathbf{N = (-1)^s * 1.M * 2^{exp - 127}}$$



- Doble precisión : 64 bits
  - 1 bit signo
  - 11 bit exponente (exceso 1023)
  - 52 bit mantisa

$$N = (-1)^s * 1.M * 2^{exp - 1023}$$

- Representación de números singulares

Simple precisión		Doble precisión		Objeto representado
Exponente	Mantisa	Exponente	Mantisa	
0	0	0	0	0
0	0	0	0	Dernormalizado
1 a 254	Cualquiera	1 a 2046	Cualquiera	Número en punto flotante
255	0	2047	0	Infinito
255	0	2047	0	NaN (No un Número)

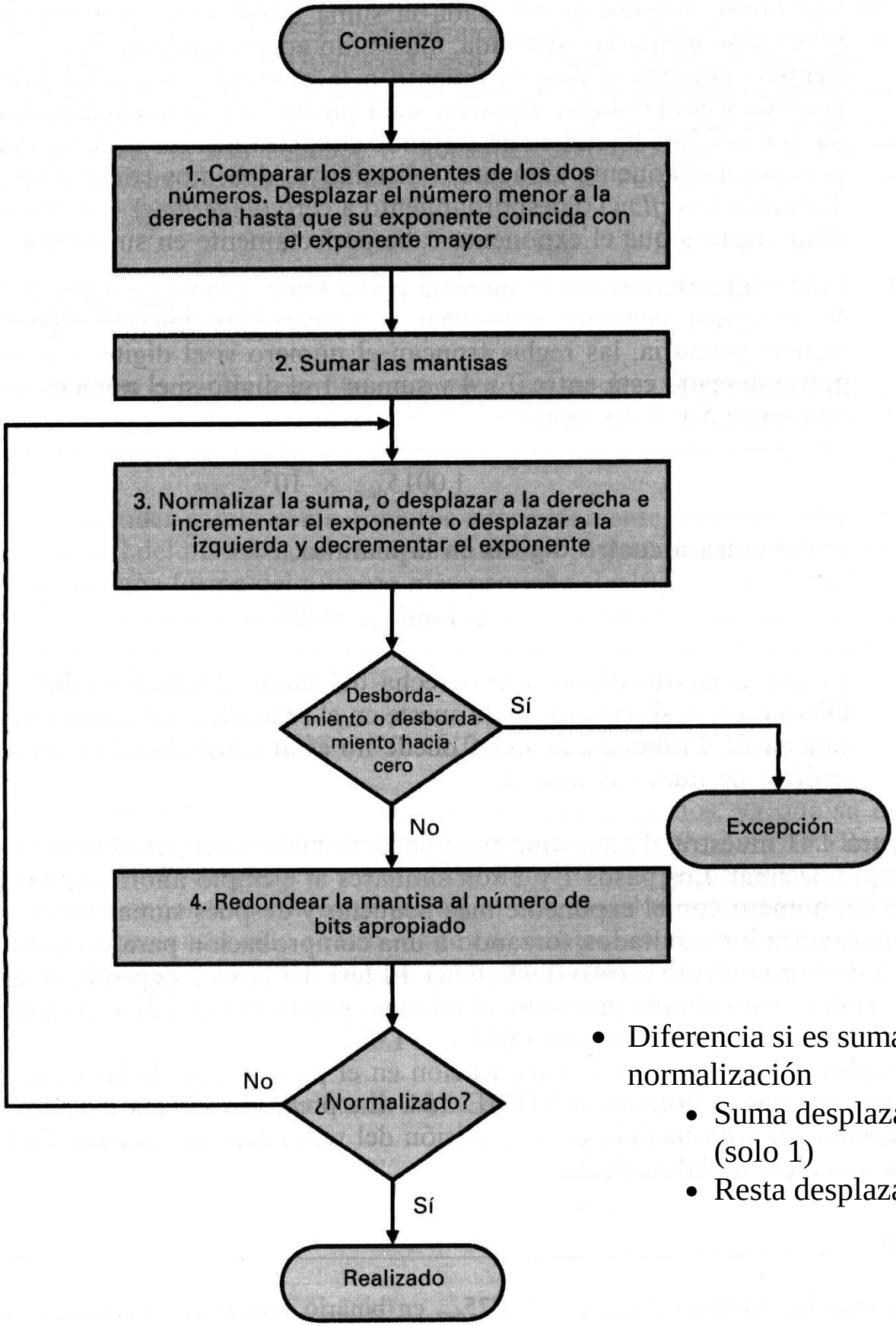
- Suma en punto flotante: Etapas
  - Alineación de mantisa (necesidad de exponentes iguales iguales)
    - Acercar exponentes de menor a mayor
  - Suma de mantisas
  - Normalización y redondeo
    - Truncación solución de redondeo
  - Desbordamiento
    - Overflow
    - Underflow
  - Ejemplo:

$$9'999_{10} * 10^{1} + 1'610 * 10^{-1} = 9'999_{10} * 10^{1} + 0'016 * 10^{1} =$$

$$= 10'015 * 10^{1} = 1'0015 * 10^{2} \approx 1'002 * 10^{2}$$

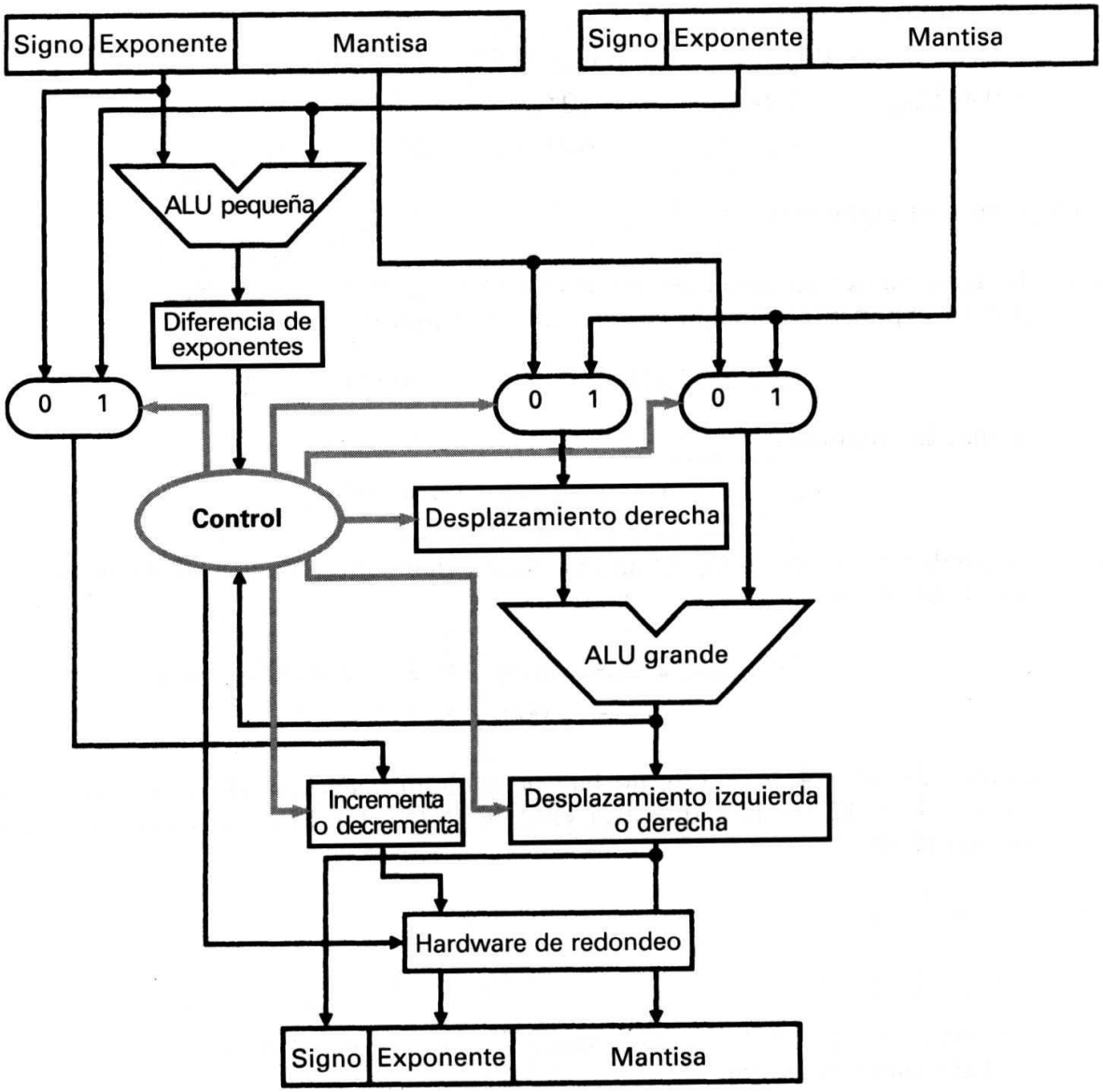
    - Solo tres dígitos de mantisa

- Algoritmo de suma en punto flotante



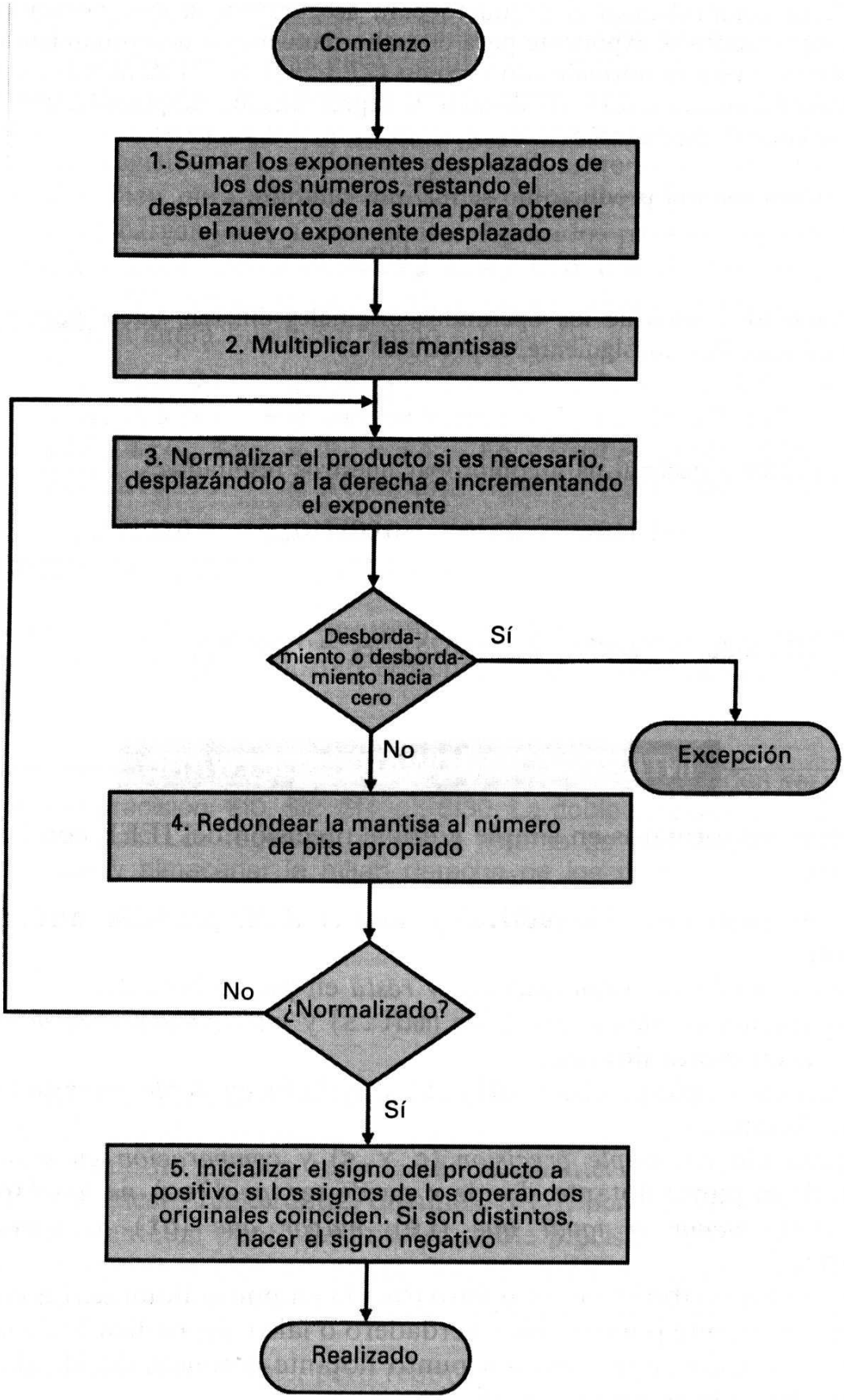
- Diferencia si es suma o resta en la normalización
  - Suma desplazamiento a derecha (solo 1)
  - Resta desplazamiento a izquierda

- Hardware para la suma en punto flotante.



- Multiplicación en punto flotante: Etapas
    - Suma de exponentes
        - Correción del exceso que se produce
    - Multiplicación de mantisa
        - Solución de doble longitud
    - Normalización y desbordamiento
        - Como mucho hay un desplazamiento.
        - Desbordamiento y desbordamiento hacia 0 (operaciones pequeñas)
    - Redondeo
    - Calculo del signo

- Algoritmo para multiplicación en punto flotante



- Ejemplo:

$$\mathbf{(1'110_{10} * 10^{10}) * (9' 200_{10} * 10^{-5})}$$

  - exponente = 10 + (-5) = 5
    - exp. desplaz. = (10+127) + (-5 + 127) = 259 = 132 + 127
    - corrección = 259 – 127 = 132 = 5 + 127
  - mantisa = 1'110 * 9'200 = 10'212000 * 10^5
  - producto = 10'0212 * 10^5
  - normalización = 1'021 * 10^6
  - signo = +

- Soporte MIPS para punto flotante (Instrucciones)

  - Posee las siguientes instruciones (simple y doble precisión)

    - **add.s/ add.d :** suma en P.F
    - **sub.s/ sub.d :** resta en P.F.
    - **mul.s/ mul.d :** multiplicación en P.F.

- **div.s/ div.d :** división en P.F.
- **c.x.s/ c.x.d :** comparación en P.F.
  - x puede ser :
    - eq : igual que
    - neq: diferente que
    - lt: menor que
    - le: menor o igual que
    - gt: mayor que
    - ge: mayor o igual que
- **bclt/ bclf :** salto verdadero/ salto falso
  - decisión posterior a una instrucción de comparación

- Soporte MIPS para punto flotante (Registros)

  - Posible solución: usar los mismos que para enteros
  - Mejor solución: usar Registros específicos de P.F.
    - $f0, $f1, ... $f31 de 32 bits
    - para simple precisión solo 16 → $f0, $f2, $f4,...$f30
    - para doble precisión se agrupan de dos en dos
      - $f0f1, $f2f3,.....,$f30f31

- Instrucciones referidas a registros en P.F.
  - Las básicas (+ , - , * y /)
  - De carga y almacenamiento de simple (s) y de doble (d) precisión
    - l.s, l.d, s.s, s.d

```
l.s     $f4, x($29)        # carga nº P.F en f4 de SP
add.s   $f2, $f4, $f6      # suma f2 = f4 + f6 en SP
s.s     $f2, 2($29)        # almacena nº de f2 de SP
```

- Precisión aritmetica
  - Comparación con los enteros
    - Los enteros representan una cantidad fija exactamente
    - Los Nº en P.F son una aproximación a un nº que no puede representar
      - Hay infinitos valores entre el valor máximo y minimo.
  - IEEE 754 ofrece métodos de redondeo
    - Que sea lo más aproximado
      - Necesidad de bit adicionado para obtener valores buenos.
    - El truncamiento impide el redondeo
    - IEEE conserva 2 bit extra a la derecha
      - Bit de Guarda
      - Bit de Redondeo

- Ejemplo (suma):

Sin dígitos adiccionales **$2'56 * 10^0 + 2'34 * 10^2 =$**
$= 0'02 * 10^2 + 2'34 * 10^2 = 2'36 * 10^2$
Con dígitos adiccionales **$2'56 * 10^0 + 2'34 * 10^2 =$**
$= 0'0256 * 10^2 + 2'34 * 10^2 = 2'3656 * 10^2$
$= 2.37 * 10^2$

- La precisión se suele dar como nº de bits de error en el bit menos significativos de la mantisa. (nº de unidades en la ultimas posición ulp) IEE garantiza que se tomaran el nº que esta en ½ ulp
- Diferentes metodos de redondeos
  - Redondeo siempre hacia arriba
  - Redondeo siempre hacia abajo
  - Redondeo siempre por truncación
  - Redondeo siempre hacia el más próximo
    - Problema con el valor del medio
      - una vez arriba y otra abajo
- bit adicional (bit retenedor → shaki)
  - Se pone a 1 si hay algún bit a la derecha a 1
  - Se pone a 0 en caso contrario

- Arquitectura MIPS reflejada

Lenguaje ensamblador MIPS en punto flotante

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	suma FP simple precisión	add.s $f2,$f4,$f6	$f2=$f4 + $f6	Suma punto flotante (simple prec.)
	resta FP simple precisión	sub.s $f2,$f4,$f6	$f2=$f4 − $f6	Resta punto flotante (simple prec.)
	multiplicación FP s. prec.	mul.s $f2,$f4,$f6	$f2=$f4 × $f6	Multiplica. punto flotante (s. prec.)
	división FP simple precisión	div.s $f2,$f4,$f6	$f2=$f4 $f6	División punto flotante (s. prec.)
	suma FP doble precisión	add.d $f2,$f4,$f6	$f2=$f4 + $f6	Suma punto flotante (doble prec.)
	resta FP doble precisión	sub.d $f2,$f4,$f6	$f2=$f4 − $f6	Resta punto flotante (doble prec.)
	multiplicación FP d. prec.	mul.d $f2,$f4,$f6	$f2=$f4 × $f6	Multiplica. punto flotante (d. prec.)
	división FP doble precisión	div.d $f2,$f4,$f6	$f2=$f4 $f6	División punto flotante (d. prec.)
Transfe-rencia de datos	carga palabra copr. 1	lwc1 $f1,100($2)	$f1=Memoria[$2 + 100]	Dato de 32 bits a registro FP
	almacena palabra copr. 1	swc1 $f1,100($2)	Memoria[$2 + 100] = $f1	Dato de 32 bits a memoria
Salto condicional	salta sobre verdad FP	bc1t 100	si (cond==1) ir a PC + 4 + 100	Salto relativo PC si FP cond.
	salta sobre falso FP	bc1f 100	si (cond==0) ir a PC + 4 + 100	Salto relativo PC si no cond.
	Comp. FP simple precisión (eq,ne,lt,le,gt,ge)	c.lt.s $f2,$f4	si ($f2 < $f4) cond=1; si no cond=0	Compara menor que punto flotante simple precisión
	Comp. FP doble precisión (eq,ne,lt,le,gt,ge)	c.lt.d $f2,$f4	si ($f2 < $f4) cond=1; si no cond=0	Compara menor que punto flotante doble precisión

Lenguaje máquina MIPS en punto flotante

Nombre	Formato	Ejemplo						Comentarios
add.s	R	17	16	6	4	2	0	add.s $f2,$f4,$f6
sub.s	R	17	16	6	4	2	1	sub.s $f2,$f4,$f6
mul.s	R	17	16	6	4	2	2	mul.s $f2,$f4,$f6
div.s	R	17	16	6	4	2	3	div.s $f2,$f4,$f6
add.d	R	17	17	6	4	2	0	add.d $f2,$f4,$f6
sub.d	R	17	17	6	4	2	1	sub.d $f2,$f4,$f6
mul.d	R	17	17	6	4	2	2	mul.d $f2,$f4,$f6
div.d	R	17	17	6	4	2	3	div.d $f2,$f4,$f6
lwc1	I	49	2	1	100			lwc1 $f1,100($2)
swc1	I	57	2	1	100			swc1 $f1,100($2)
bc1t	I	17	8	1	100			bc1t 100
bc1f	I	17	8	0	100			bc1f 100
c.lt.s	R	17	16	4	2	0	60	c.lt.s $f2, $f4
c.lt.d	R	17	17	4	2	0	60	c.lt.d $f2, $f4
Tamaño de campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas las instrucciones MIPS de 32 bits

9.- FALACIAS Y PIFIAS

- Olvidar que la suma en punto flotante no es conmutativa

$$x + (y + z) \neq (x + y) + z$$

$$-1'5 * 10^{38} + (1'5 * 10^{38} + 1'0) = 0'0 * 10^{0}$$
$$(-1'5*10^{38} + 1'5 * 10^{38}) + 1{,}0 = 1'0 * 10^{0}$$

Es debido a que se suma nº de valores muy diferentes

- Igual que n desplazamiento a la izquierda es multiplicar por 2^n a derecha es dividir entre 2^n
  - Para enteros sin signos si es correcto
  - para enteros con signos no.
    - Desplazamientos aritméticos

- Arquitectura MIPS revelada

Instrucciones MIPS fundamentales	Nombre	Formato	Instrucciones de multiplicación y división enteras + punto flotante	Nombre	Formato
suma	add	R	multiplicación	mult	R
suma inmediata	addi	I	multiplicación sin signo	multu	R
suma sin signo	addu	R	división	div	R
suma inmediata sin signo	addiu	I	división sin signo	divu	R
resta	sub	R	transfiere desde hi	mfhi	R
resta sin signo	subu	R	transfiere desde lo	mflo	R
and	and	R	transf. desde control del sist. (EPC)	mfc0	R
and inmediato	andi	I	suma punto flotante simple prec.	add.s	R
or	or	R	suma punto flotante doble precisión	add.d	R
or inmediato	ori	I	resta punto flotante simple prec.	sub.s	R
desplaz. lógico a la izda.	sll	R	resta punto flotante doble precisión	sub.d	R
desplaz. lógico a la dcha.	srl	R	mult. punto flotante simple prec.	mul.s	R
carga superior inmediato	lui	I	mult. punto flotante doble precisión	mul.d	R
carga palabra	lw	I	división punto flotante simple prec.	div.s	R
alamcena palabra	sw	I	división punto flotante doble prec.	div.d	R
salta sobre igual	beq	I	carga pal. en punto flotante s. prec.	l.s	I
salta sobre no igual	bne	I	carga pal. en punto flotante d. prec.	l.d	I
bifurcación	j	J	almacena pal. en p. flotante s. prec.	s.s	I
bifurcación y enlace	jal	J	almacena pal. en p. flotante s. prec.	s.d	I
bifurcación a registro	jr	R	salta sobre punto flotante verdadero	bc1t	I
inicializa menor que	slt	R	salta sobre punto flotante falso	bc1f	I
inicial. menor que inmediato	slti	I	compara en punto flotante s. prec.	c.x.s	R
inicializa menor que sin signo	sltu	R	(*x*=eq, neq, lt, le, gt, ge)		
inicial. menor que inm. sin signo	sltiu	I	compara en punto flotante d. prec. (*x*=eq, neq, lt, le, gt, ge)	c.x.d	R

- Frecuencia de instrucción MIPS para gcc y spice

Núcleo de MIPS	Nombre	gcc	spice	Multiplicación y división + FP	Nombre	gcc	spice
sumar	add	0%	0%	multiplicación	mult	0%	0%
sumar inmediato	addi	0%	0%	multiplicación sin signo	mul	0%	0%
sumar sin signo	addu	8%	13%	división	div	0%	0%
sumar inmediato sin signo	addiu	16%	6%	división sin signo	divu	0%	0%
restar	sub	0%	0%	transfiere desde hi	mfhi	0%	0%
restar sin signo	subu	1%	1%	transfiere desde lo	mflo	0%	0%
and	and	2%	1%	trans. desde control del sist. (EPC)	mfc0	0%	0%
and inmediato	andi	2%	1%	suma punto flotante s. prec.	add.s	0%	0%
or	or	2%	0%	suma punto flotante doble prec.	add.d	0%	5%
or inmediato	ori	0%	1%	resta punto flotante s. prec.	sub.s	0%	0%
desplaz. lógico a la izda.	sll	8%	6%	resta punto flotante doble prec.	sub.d	0%	3%
desplaz. lógico a la dcha.	srl	2%	1%	mult. punto flotante s. prec.	mul.s	0%	0%
cargar superior inmediato	lui	2%	0%	mult. punto flotante doble prec.	mul.d	0%	6%
cargar palabra	lw	22%	11%	división punto flotante s. prec.	div.s	0%	0%
almacenar palabra	sw	11%	5%	división punto flotante d. prec.	div.d	0%	3%
saltar sobre igual	beq	8%	5%	carga pal. en p. flotante s. prec.	l.s	0%	0%
saltar sobre no igual	bne	8%	1%	carga pal. en p. flotante d. prec.	l.d	0%	15%
bifurcar	j	0%	0%	almacena pal. en p. f. s. prec.	s.s	0%	3%
bifurcación y enlace	jal	1%	1%	almacena pal. en p. f. d. prec.	s.d	%	6%
bifurcación a registro	jr	1%	1%	salta sobre p. flotante verdadero	bc1t	0%	1%
inicializa menor que	slt	3%	1%	salta sobre punto flotante falso	bc1f	0%	1%
inicial. menor que inm.	slti	1%	0%	compara en p. flotante s. prec.	c.x.s	0%	0%
inic. menor que sin signo	sltu	1%	0%	(x=eq, neq, lt, le, gt, ge)			
inic. < que inm. sin signo	sltiu	1%	0%	compara en p. flotante d. prec.	c.x.d	0%	2%
Total columna		100%	55%	Total columna		0%	45%