Estructura de Computadores

Tema 5. Aritmética de enteros y coma flotante

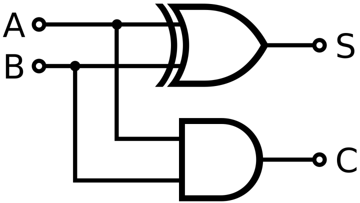
Suma de naturales y enteros

A	B	Suma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Suma de naturales y enteros

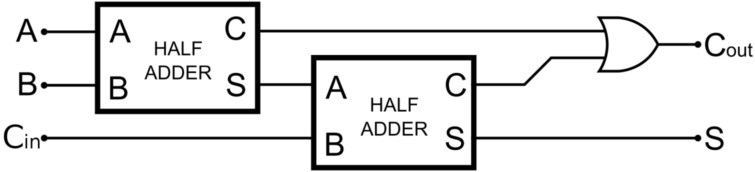
A	B	Suma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Half Adder



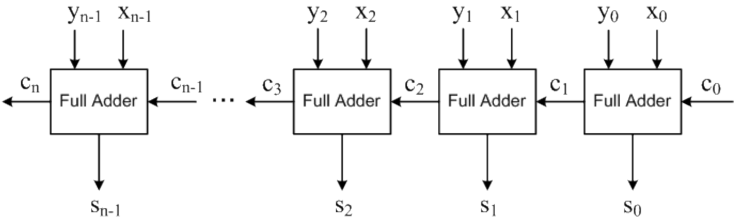
Suma de naturales y enteros

- Full Adder



Suma de naturales y enteros

- Sumador de *n* bits con propagación de acarreo



- Mismo sumador para naturales y enteros en Ca2

Resta de naturales y enteros

A	B	Resta	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Resta de naturales y enteros

A	B	Resta	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- Se puede usar el sumador de *n* bits
- Sumar el valor de signo opuesto del segundo operando
  - Invertir los bits del segundo operando
  - Sumar 1

Overflow

- Rango para Ca2 con n bits: $[-2^{n-1}, 2^{n-1}-1]$

Overflow

- Rango para Ca2 con n bits: $[-2^{n-1}, 2^{n-1}-1]$
- Se produce overflow en una operación cuando el resultado no pertenece al rango
- En caso de overflow, el resultado en n bits **NO** es correcto

Overflow

- Rango para Ca2 con n bits: $[-2^{n-1}, 2^{n-1} - 1]$
- Se produce overflow en una operación cuando el resultado no pertenece al rango
- En caso de overflow, el resultado en n bits **NO** es correcto
- Se produce overflow en la suma cuando los operandos son del mismo signo y el resultado es de signo opuesto

Overflow

- Rango para Ca2 con n bits: $[-2^{n-1}, 2^{n-1} - 1]$
- Se produce overflow en una operación cuando el resultado no pertenece al rango
- En caso de overflow, el resultado en n bits **NO** es correcto
- Se produce overflow en la suma cuando los operandos son del mismo signo y el resultado es de signo opuesto
- Se produce overflow en la resta cuando la *diferencia* tiene el mismo signo que el *sustraendo*, pero tiene distinto signo que el *minuendo*
  - $diferencia = minuendo - sustraendo$
  - $minuendo = sustraendo + diferencia$

Detección del overflow

- Para naturales, overflow si c_n es igual a 1

Detección del overflow

- Para naturales, overflow si c_n es igual a 1
- En Ca2, overflow si $c_{n-1} \neq c_n$
  - $\mathit{overflow} = c_{n-1} \oplus c_n$

Detección del overflow

- Para naturales, overflow si c_n es igual a 1
- En Ca2, overflow si $c_{n-1} \neq c_n$
  - $overflow = c_{n-1} \oplus c_n$
- Suma de enteros y naturales en MIPS se diferencia en cómo gestionan los overflows

Detección del overflow

- Para naturales, overflow si c_n es igual a 1
- En Ca2, overflow si $c_{n-1} \neq c_n$
  - $overflow = c_{n-1} \oplus c_n$
- Suma de enteros y naturales en MIPS se diferencia en cómo gestionan los overflows
- Suma de enteros
  - `add`, `addi`, `sub`
  - Producen una excepción en caso de overflow

Detección del overflow

- Para naturales, overflow si c_n es igual a 1
- En Ca2, overflow si $c_{n-1} \neq c_n$
  - $overflow = c_{n-1} \oplus c_n$
- Suma de enteros y naturales en MIPS se diferencia en cómo gestionan los overflows
- Suma de enteros
  - `add`, `addi`, `sub`
  - Producen una excepción en caso de overflow
- Suma de naturales
  - `addu`, `addiu`, `subu`
  - Ignoran los overflows

Detección del overflow

- MIPS no incluye instrucciones específicas para consultar si se ha producido un overflow
- Se puede calcular por software
  - $\mathit{overflow} = \overline{(a_{31} \oplus b_{31})} \cdot (a_{31} \oplus s_{31})$

```
xor $t3, $t0, $t1    # a xor b
nor $t3, $t3, $zero
xor $t4, $t0, $t2
and $t3, $t3, $t4
srl $t3, $t3, 31     # Traslada el bit 31
                     # a la posicion 0
```

Multiplicación de naturales

- Números decimales (base 10)

```
       348     multiplicando
x      951     multiplicador
__________
       348    = 348 x 1
     1740     = 348 x 50
+  3132       = 348 x 900
__________
   330948
```

Multiplicación de naturales

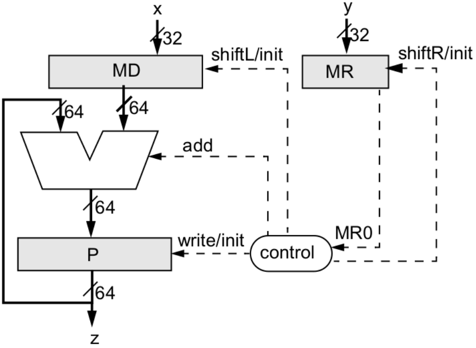
- Números binarios (base 2)

```
        1010   multiplicando
x       1101   multiplicador
_____________
        1010   = 1010 x 1
       0000    = 1010 x 00
      1010     = 1010 x 100
+    1010      = 1010 x 1000
_____________
    10000010
```

Circuito para multiplicación secuencial

- Naturales de 32 bits con resultado de 64 bits
  - Tarda 33 ciclos en completar el producto...
  - ...asumiendo que una suma de 64 bits tarda 1 ciclo

Multiplicador seqüencial de naturals



Pseudocodi

```
// Inicialització
MD_{63:32} = 0; MD_{31:0} = x;
P  = 0;
MR = y;

for (i=1; i<=32; i++)
{
     if (MR0 == 1)
           P = P + MD;
     MD = MD << 1;
     MR = MR >> 1;
}
```

Ejemplo multiplicación: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
init	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**

Ejemplo multiplicación: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
init	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	1	1	**0**

Ejemplo multiplicación: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
init	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	1	1	**0**
2	0	0	**0**	**0**	**1**	**0**	**1**	**0**	0	0	**1**	**0**	**1**	**0**	0	0	0	0	1	**1**

Ejemplo multiplicación: 1010 x 1101

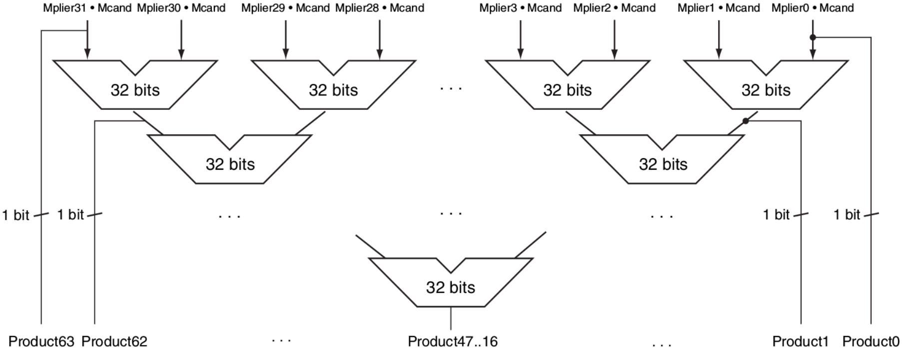
iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
init	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	1	1	**0**
2	0	0	**0**	**0**	**1**	**0**	**1**	**0**	0	0	**1**	**0**	**1**	**0**	0	0	0	0	1	**1**
3	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**

Ejemplo multiplicación: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
init	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	1	1	**0**
2	0	0	**0**	**0**	**1**	**0**	**1**	**0**	0	0	**1**	**0**	**1**	**0**	0	0	0	0	1	**1**
3	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**
4	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	0	0	0	0	0	0	0	0

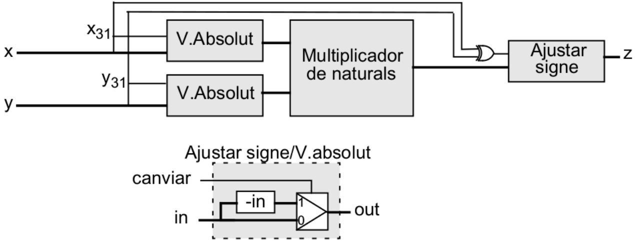
Multiplicador más rápido

- Patterson, David A., and Hennessy, John L., Computer Organization and Design: The Hardware/Software Interface, 5th. edition , Ed. Morgan Kaufmann , 2013.



Multiplicación de enteros

1. Calcular los valores absolutos
2. Multiplicar los valores absolutos (producto de naturales)
3. Cambiar el signo del resultado si los operandos tienen distinto signo



Multiplicación en MIPS

- En MIPS:
  - `mult rs, rt      # $hi:$lo <- rs * rt (enteros)`
  - `multu rs, rt     # $hi:$lo <- rs * rt (naturales)`
- `$hi` y `$lo` son dos registros especiales
  - No se pueden utilizar en el resto de instrucciones estudiadas hasta ahora
- Para mover el resultado a registros de propósito general:
  - `mflo rd     # rd <- $lo`
  - `mfhi rd     # rd <- $hi`
- Overflow
  - Naturales: `$hi` es diferente de cero
  - Enteros: `$hi` no es la extensión de signo de `$lo`

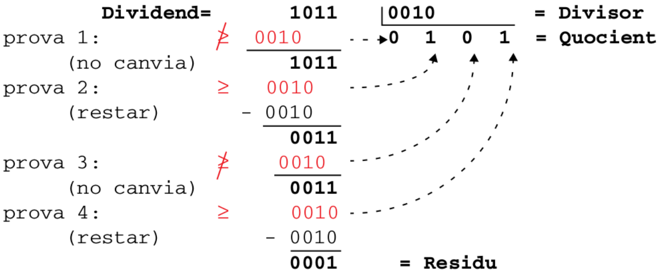
División de naturales

- Naturales en base 10

```
                               421    |013
Prova 1:             ≱       013              --> Hi cap a 0, multiplicar: 0X013=000
  (restar 0)         -       000
                               421
Prova 2:             ≥        013             --> Hi cap a 3, multiplicar: 3X013=039
  (restar)           -        039
                               031
Prova 3:             ≥         013            --> Hi cap a 2, multiplicar: 2X013=013
  (restar)           -         026
                               005            --> Residu = 005, Quocient = 032
```

División de naturales

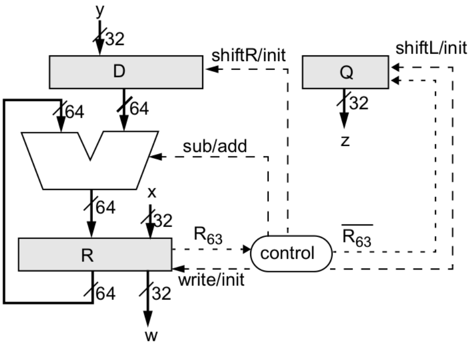
- Naturales en base 2



Circuito para división de naturales

- División de naturales de 32 bits "con restauración"
  - Cociente: $z = x/y$
  - Resto: $w = x\%y$

Divisor seqüencial de naturals



Pseudocodi

```
// Inicialització
R63:32 = 0; R31:0 = x;
D63:32 = y; D31:0 = 0;
Q  = 0;
for (i=1; i<=32; i++) {
    D = D >> 1;
    R = R - D;
    if (R63 == 0)
        Q = (Q << 1) | 1;
    else {
        R = R + D;
        Q = Q << 1;
    }
}
```

Ejemplo división: 1011 / 0010

iter.	R (Dividend/Residu)								D (Divisor)								Q (Quocient)			
init	0	0	0	0	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	0	0	0	0	0	0	0

Ejemplo división: 1011 / 0010

iter.	R (Dividend/Residu)								D (Divisor)								Q (Quocient)			
init	0	0	0	0	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	0	0	0	0	0	0	0
1	0	0	0	0	**1**	**0**	**1**	**1**	0	**0**	**0**	**1**	**0**	0	0	0	0	0	0	**0**

Ejemplo división: 1011 / 0010

iter.	R (Dividend/Residu)								D (Divisor)								Q (Quocient)			
init	0	0	0	0	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	0	0	0	0	0	0	0
1	0	0	0	0	**1**	**0**	**1**	**1**	0	**0**	**0**	**1**	**0**	0	0	0	0	0	0	**0**
2	0	0	0	0	**0**	**0**	**1**	**1**	0	0	**0**	**0**	**1**	**0**	0	0	0	0	**0**	**1**

Ejemplo división: 1011 / 0010

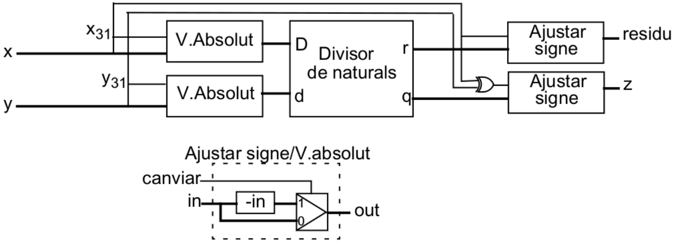
iter.	R (Dividend/Residu)								D (Divisor)								Q (Quocient)			
init	0	0	0	0	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	0	0	0	0	0	0	0
1	0	0	0	0	**1**	**0**	**1**	**1**	0	**0**	**0**	**1**	**0**	0	0	0	0	0	0	**0**
2	0	0	0	0	**0**	**0**	**1**	**1**	0	0	**0**	**0**	**1**	**0**	0	0	0	0	**0**	**1**
3	0	0	0	0	**0**	**0**	**1**	**1**	0	0	0	**0**	**0**	**1**	**0**	0	0	**0**	**1**	**0**

Ejemplo división: 1011 / 0010

iter.	R (Dividend/Residu)								D (Divisor)								Q (Quocient)			
init	0	0	0	0	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	0	0	0	0	0	0	0
1	0	0	0	0	**1**	**0**	**1**	**1**	0	**0**	**0**	**1**	**0**	0	0	0	0	0	0	**0**
2	0	0	0	0	**0**	**0**	**1**	**1**	0	0	**0**	**0**	**1**	**0**	0	0	0	0	**0**	**1**
3	0	0	0	0	**0**	**0**	**1**	**1**	0	0	0	**0**	**0**	**1**	**0**	0	0	**0**	**1**	**0**
4	0	0	0	0	**0**	**0**	**0**	**1**	0	0	0	0	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**

División de enteros

- Calcular los valores absolutos
- Dividir los valores absolutos (naturales) calculando el cociente y el residuo
- Cambiar el signo del cociente si los operandos tienen signo diferente, y el del residuo si el dividendo es negativo



División en MIPS

- `div rs, rt`
  - División de enteros
  - `$lo <- rs/rt`
  - `$hi <- rs%rt`
  - ¿Overflow?
- `divu rs, rt`
  - División de naturales
  - `$lo <- rs/rt`
  - `$hi <- rs%rt`
  - No puede dar overflow
- Si el divisor es 0, el resultado es indefinido

División por potencias de 2

- Para números naturales, `srl` calcula el mismo cociente que `divu`
- Para enteros:
  - Si el dividendo es positivo, `sra` y `div` calculan el mismo cociente
  - Si el dividendo es negativo y la división no es exacta, los resultados de `sra` y `div` son distintos
- Para traducir los operadores de división (/) y módulo (%), siempre usaremos las instrucciones `div` y `divu`