



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Arquitectura de Computadors

Tema 5

Aritmètica d'enters i coma flotant

Estructura de Computadors (EC)

2023 - 2024 Q2

Adrià Armejach (adria.armejach@upc.edu)



Objectius

- Aritmètica d'enters
 - Condicions de sobreiximent (overflow)
 - Excepcions degudes al overflow
 - Algorismes de multiplicació i divisió
 - Amb propostes hardware senzilles
- Nombres en coma flotant
 - Representació en el format IEEE-754
 - Algorismes de suma/resta i multiplicació/divisió
 - Arrodoniment de resultats i càlcul d'error de precisió



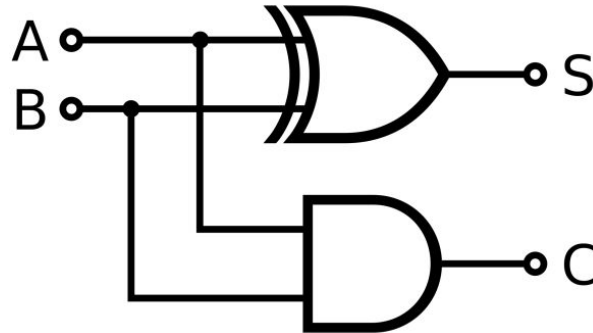
Overflow de suma i resta d'enters



Suma de naturales i enters

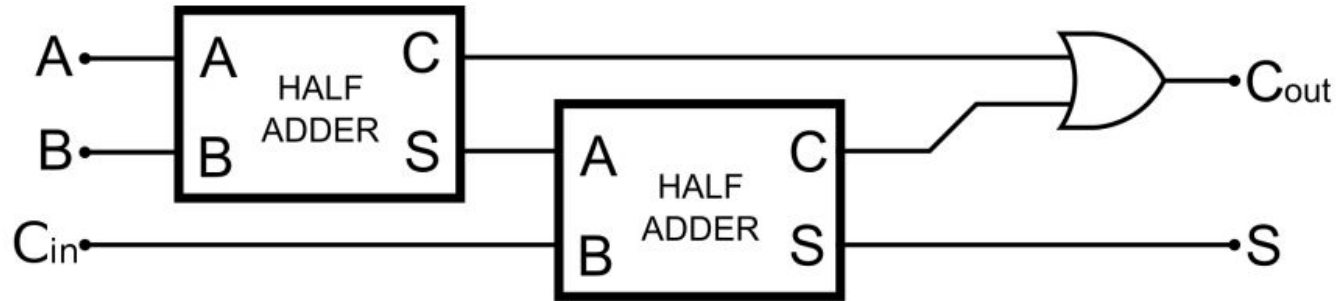
A	B	Suma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Half Adder



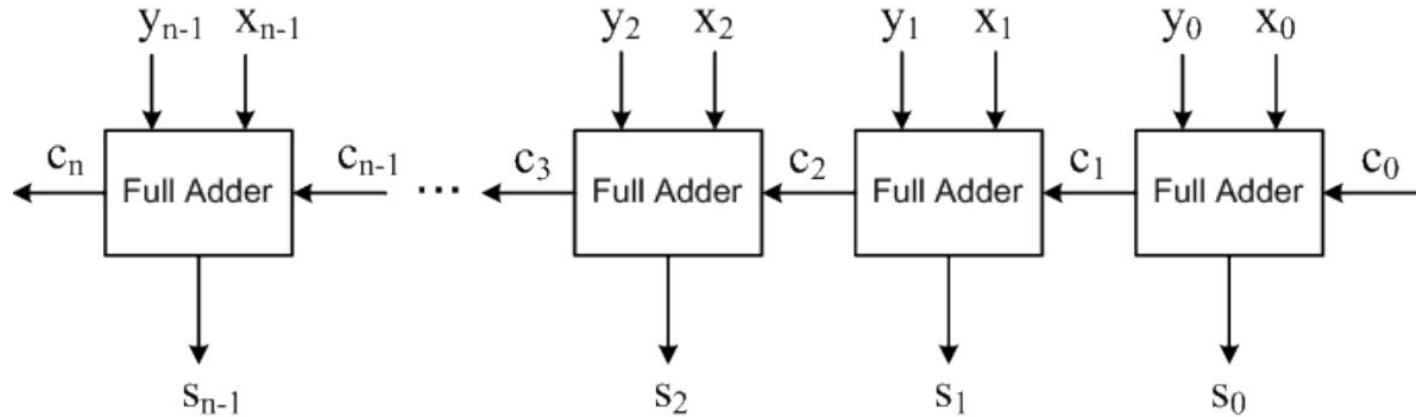
Suma de naturals i enters

- Full Adder



Suma de naturals i enters

- Sumador de n bits amb propagació de carry



- Mateix sumador per naturals i enters en Ca2

Resta de naturals i enters

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

- Es pot utilitzar el sumador de n bits
- Canviar de signe el segon operand i sumar-lo
 - Invertir els bits del segon operand
 - Sumar 1

Overflow

- Rang representació Ca2 per enters de n bits: $[-2^{n-1}, 2^{n-1}-1]$
- Es produeix **overflow** en una operació quan el resultat no pertany al rang
- En cas de **overflow** el resultat calculat amb n bits no és correcte
- Es produeix overflow
 - Suma: si i només si els operands són del mateix signe i el resultat és de signe diferent
 - Resta: si i només si la diferencia (d) és del mateix signe que el substraend (b) però de signe diferent que el minuend (a).
 - $d = a - b$
 - $a = b + d$


Detecció de l'overflow

- Per naturals, overflow si c_n és igual a 1
- Per Ca2, overflow si $c_{n-1} \neq c_n$
 - $\text{overflow} = c_{n-1} \oplus c_n$
- MIPS fa distinció en el tractament d'overflows per naturals i enters
- Suma d'enters
 - add, addi, sub
 - Produeixen una excepció en cas d'overflow
- Suma de naturals
 - addu, addiu, subu
 - Ignoren els overflows
- C estipula que els overflows son ignorats


Detecció de l'overflow

- MIPS no inclou instruccions específiques per consultar si s'ha produït overflow
- Es pot calcular per software, suposant $s = a + b$:
 - $\text{overflow} = \overline{(a_{31} \oplus b_{31})} \wedge (a_{31} \oplus s_{31})$
- Executem `addu $t2, $t0, $t1` - calcula en `$t3` la condició d'overflow

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



Multiplicació entera de 32
bits i resultat de 64 bits



Multiplicació de naturals

- Números decimals (base 10)

$$\begin{array}{r} 348 \\ \times 951 \\ \hline 348 \\ 1740 \\ + 3132 \\ \hline 330948 \end{array}$$

multiplicand
multiplicador

$$\begin{aligned} &= 348 \times 1 \\ &= 3480 \times 5 \\ &= 34800 \times 9 \end{aligned}$$

- Obtenir tants productes parcials com dígits té el multiplicador i sumar

Multiplicació de naturals

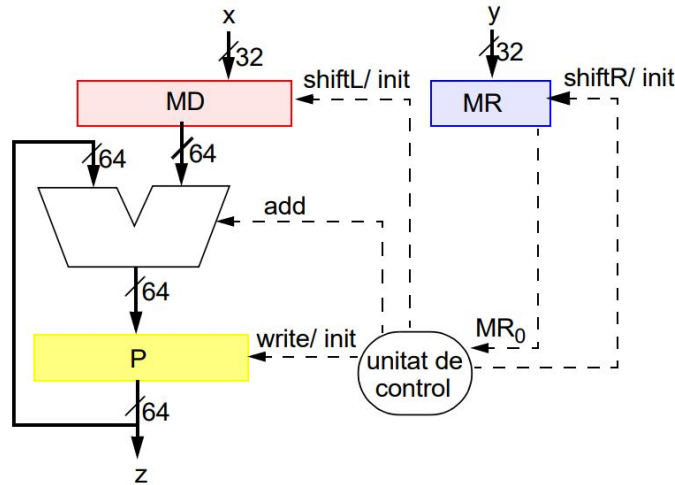
- Números binaris (base 2)

$\begin{array}{r} 10 \\ \times 13 \\ \hline \end{array}$	$\begin{array}{r} 1010 \\ \times 1101 \\ \hline 1010 \\ 0000 \\ 1010 \\ + 1010 \\ \hline 10000010 \end{array}$	$\begin{array}{l} \text{multiplicand} \\ \text{multiplicador} \\ = 1010 \times 1 \\ = 10100 \times 0 \\ = 101000 \times 1 \\ = 1010000 \times 1 \end{array}$
$= 130$		

- Productes parcials es redueixen a una decisió binària

Circuit multiplicador seqüencial

Multiplicador seqüencial de naturals: $z = x * y$



Pseudocodi

```
// Inicialització
MD0:31 = x; MD32:63 = 0;
P = 0;
MR = y;

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

- Naturals de 32 bits amb resultat de 64 bits
 - Tarda 33 cicles a obtenir el producte ...
 - ... assumint que una suma de 64 bits tarda 1 cicle

Exemple de multiplicació: 1010 x 1101

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

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	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

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	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	1	1	1

Exemple de multiplicació: 1010 x 1101

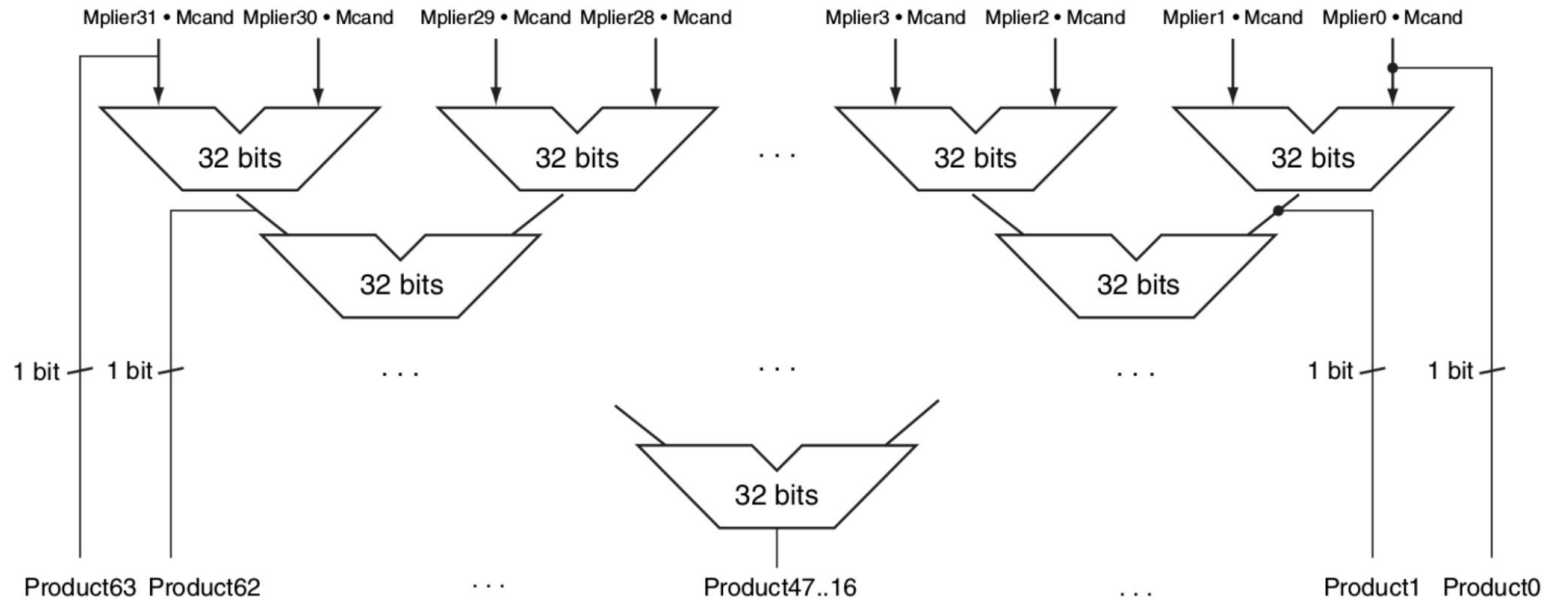
iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1
1	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	0	0	1	1	0
2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1
3	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	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

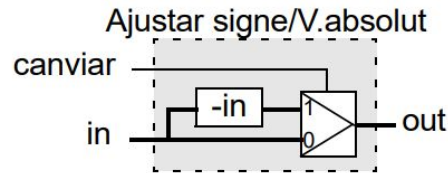
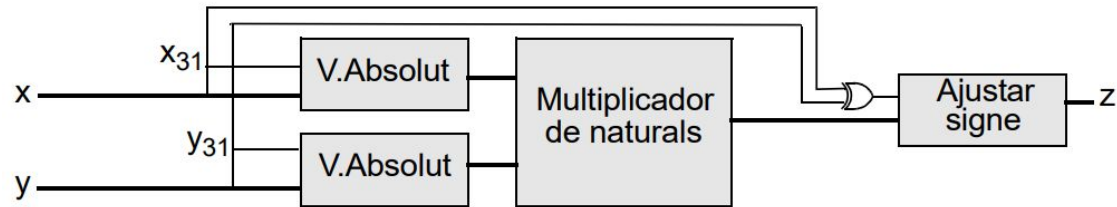
Multiplicador més ràpid

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



Multiplicació d'enters

1. Calcular els valors absoluts
2. Multiplicar els valors absoluts
3. Canviar el signe del resultat si els operands tenen signes diferents



Multiplicació en MIPS

- En MIPS:
 - `mult rs, rt` `# $hi:$lo <- rs * rt (enters)`
 - `multu rs, rt` `# $hi:$lo <- rs * rt (naturals)`
- `$hi` i `$lo` són registres especials
- Per moure el resultat a registres de propòsit general
 - `mflo rd` `# rd <- $lo`
 - `mfhi rd` `# rd <- $hi`
- Overflow
 - Naturals: `$hi` és diferent de zero
 - Enters: `$hi` no és l'extensió de signe de `$lo`
- En C, la multiplicació entera de 32 bits retorna un resultat de 32 bits
 - Els bits de la part alta (`$hi`) s'ignoren



Divisió entera de 32 bits
amb càlcul de residu



Divisió de naturals

- Naturals en base 10

		421	013	
Prova 1:	\nless	013		--> Hi cap a 0 , multiplicar: $0 \times 013 = 000$
(restar 0)	-	000		
		421		
Prova 2:	\geq	013		--> Hi cap a 3 , multiplicar: $3 \times 013 = 039$
(restar)	-	039		
		031		
Prova 3:	\geq	013		--> Hi cap a 2 , multiplicar: $2 \times 013 = 013$
(restar)	-	026		
		005		--> Residu = 005, Quocient = 032

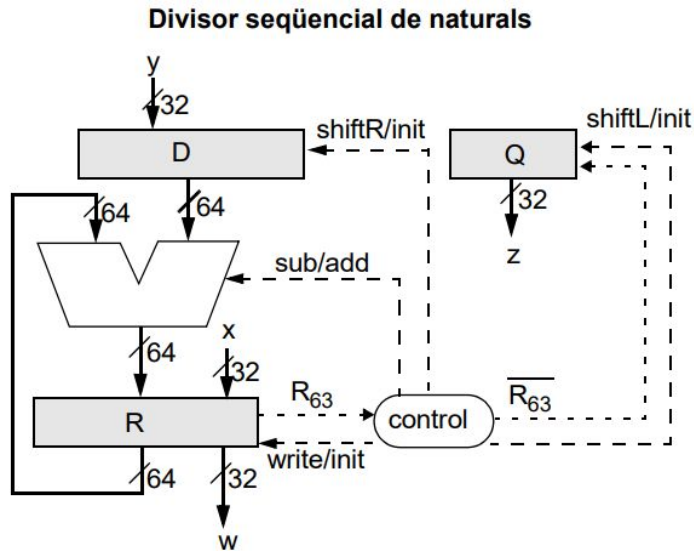
Divisió de naturals

- Naturals en base 2

Dividend= 1011 **0010** **= Divisor**
prova 1: \nless 0010 --> 0 1 0 1 **= Quocient**
 (no canvia) 1011
prova 2: \geq 0010 -----
 (restar) - 0010
 0011
prova 3: \nless 0010 -----
 (no canvia) 0011
prova 4: \geq 0010 -----
 (restar) - 0010
 0001 **= Residu**

Circuit seqüencial per a la divisió de naturals de 32 bits

- Divisió de naturals de 32 bits “amb restauració”
 - Quocient: $z = x / y$
 - Residu: $w = x \% y$



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;
    }
}
```

Exemple divisió: 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

Exemple divisió: 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

Exemple divisió: 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

Exemple divisió: 1011 / 0010

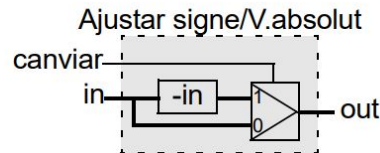
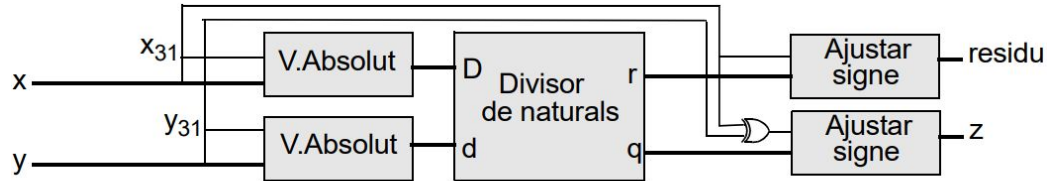
iter.	R (Dividend/Residu)								D (Divisor)								Q (Quocient)			
init	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	1	1
3	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0

Exemple divisió: 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	1	1
3	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1

Divisió d'enters

1. Calcular els valors absoluts
2. Dividir els valors absoluts (divisió de naturals) amb càlcul de quocient i residu
3. Canviar el signe del quocient si els operands tenen signes diferents, i el del residu, si el dividend és negatiu



Divisió en MIPS

- `div rs, rt`
 - Divisió d'enters
 - `$lo <- rs/rt` (quocient)
 - `$hi <- rs%rt` (residu)
 - Overflow? Únic cas: dividend = -2^{31} , divisor = -1 , resultat = 2^{31}
- `divu rs, rt`
 - Divisió de naturals
 - `$lo <- rs/rt` (quocient)
 - `$hi <- rs%rt` (residu)
 - No pot donar overflow
- Si el divisor és $0 \rightarrow$ resultat indefinit

Divisió per potències de 2

- Per números naturals
 - `sr1` calcula el mateix quocient que `divu`
- Per enters
 - Si el dividend és positiu, `sra` i `div` donen el mateix quocient
 - Si el dividend és negatiu i la divisió no és exacta, els resultats de `sra` i `div` són diferents
- Per traduir les operacions de divisió (/) i mòdul (%), sempre usarem les instruccions `div` i `divu`