Tema 5. Aritmètica d'enters i coma flotant

Joan Manuel Parcerisa

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Aritmètica d'enters

- Suma i resta
- Multiplicació
- Divisió

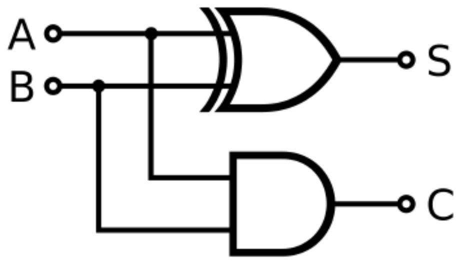
Suma de naturals i enters

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

Suma de naturals i enters

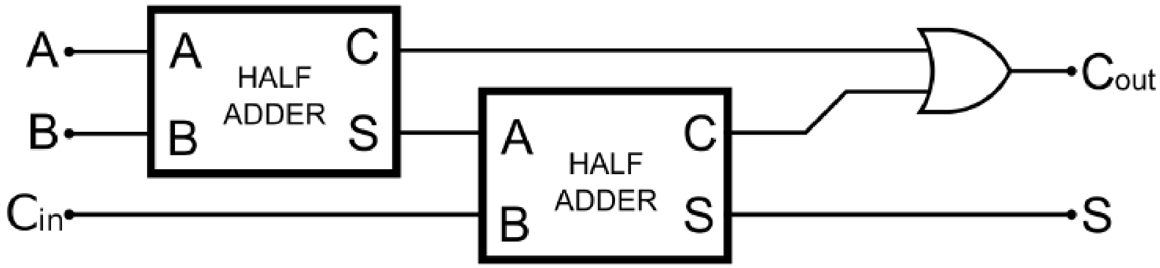
A	B	Suma	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Half Adder



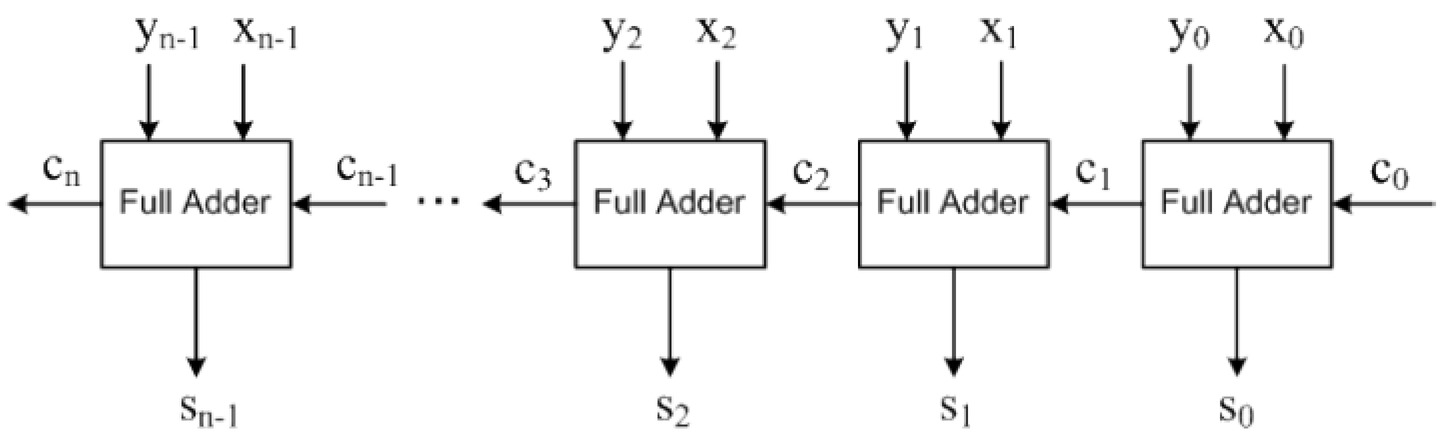
Suma de naturals i enters

- Full Adder



Suma de naturals i enters

- Sumador de *n* bits amb propagació de carry



- El mateix sumador per a naturals i enters en Ca2

Resta d'enters

- La resta

$$D = A - B$$

Equival a

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

Resta d'enters

- La resta

  $D = A - B$

  Equival a

  $D = A + (-B)$

- Podem usar el sumador de n bits, canviant el signe de B
  - Invertir els bits de B (amb una porta NOT a cada bit)
  - Sumar 1 (fent el carry inicial c_0=1)

Overflow de suma i resta

- Rang d'enters en Ca2 amb *n* bits: $[-2^{n-1}, 2^{n-1}-1]$
  - Overflow: si el resultat no pertany al rang
    → El resultat amb n bits NO és correcte

Overflow de suma i resta

- Rang d'enters en Ca2 amb *n* bits: $[-2^{n-1}, 2^{n-1}-1]$
  - Overflow: si el resultat no pertany al rang
    → El resultat amb n bits NO és correcte
- Overflow de la suma
  - Si operands del mateix signe però resultat de signe contrari

Overflow de suma i resta

- Rang d'enters en Ca2 amb *n* bits: $[-2^{n-1}, 2^{n-1}-1]$
  - Overflow: si el resultat no pertany al rang
    → El resultat amb n bits NO és correcte
- Overflow de la suma
  - Si operands del mateix signe però resultat de signe contrari
- Formulem la resta com una suma

$$\textit{diferència} = \textit{minuend} - \textit{substraend}$$

$$\textit{minuend} = \textit{substraend} + \textit{diferència}$$

Overflow de suma i resta

- Rang d'enters en Ca2 amb *n* bits: $[-2^{n-1}, 2^{n-1}-1]$
  - Overflow: si el resultat no pertany al rang
    → El resultat amb n bits NO és correcte
- Overflow de la suma
  - Si operands del mateix signe però resultat de signe contrari
- Formulem la resta com una suma

$$diferència = minuend - substraend$$

$$minuend = substraend + diferència$$

- Overflow de la resta
  - Si substraend i diferència són del mateix signe però el minuend és de signe contrari

Detecció d'overflow

- Com detecta el hardware que hi ha overflow?
  - Naturals: si $c_n = 1$ ($overflow = c_n$)
  - Enters: si $c_{n-1} \neq c_n$ ($overflow = c_{n-1} \oplus c_n$)

Detecció d'overflow

- Com detecta el hardware que hi ha overflow?
  - Naturals: si $c_n = 1$ ($overflow = c_n$)
  - Enters: si $c_{n-1} \neq c_n$ ($overflow = c_{n-1} \oplus c_n$)
- Instruccions MIPS
  - `add`, `addi`, `sub` l'overflow d'enters causa excepció
  - `addu`, `addiu`, `subu` l'overflow s'ignora

Detecció d'overflow

- Com detecta el hardware que hi ha overflow?
  - Naturals: si $c_n = 1$ ($overflow = c_n$)
  - Enters: si $c_{n-1} \neq c_n$ ($overflow = c_{n-1} \oplus c_n$)
- Instruccions MIPS
  - `add`, `addi`, `sub` — l'overflow d'enters causa excepció
  - `addu`, `addiu`, `subu` — l'overflow s'ignora
- En alguns llenguatges d'alt nivell, l'overflow d'enters ha de causar excepció
  - `add`, `addi`, `sub` — per a enters
  - `addu`, `addiu`, `subu` — per a naturals

Detecció d'overflow

- Com detecta el hardware que hi ha overflow?
  - Naturals: si $c_n = 1$ (*overflow* = c_n)
  - Enters: si $c_{n-1} \neq c_n$ (*overflow* = $c_{n-1} \oplus c_n$)
- Instruccions MIPS
  - `add`, `addi`, `sub` l'overflow d'enters causa excepció
  - `addu`, `addiu`, `subu` l'overflow s'ignora
- En alguns llenguatges d'alt nivell, l'overflow d'enters ha de causar excepció
  - `add`, `addi`, `sub` per a enters
  - `addu`, `addiu`, `subu` per a naturals
- En C, s'ignoren els overflows
  - Usarem `addu`, `addiu`, `subu` tant per a enters com naturals

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

- Però es pot calcular: suposem la suma $s = a+b$

$$overflow = (\overline{a_{31} \oplus b_{31}}) \cdot (a_{31} \oplus s_{31})$$

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

- Però es pot calcular: suposem la suma $\boxed{s = a+b}$

$$overflow = (\overline{a_{31} \oplus b_{31}}) \cdot (a_{31} \oplus s_{31})$$

- En assemblador:

```
addu  $t2, $t0, $t1   # a + b
```

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

- Però es pot calcular: suposem la suma $s = a+b$

$$overflow = \boxed{\overline{(a_{31} \oplus b_{31})}} \cdot (a_{31} \oplus s_{31})$$

- En assemblador:

```
addu   $t2, $t0, $t1   # a + b
xor    $t3, $t0, $t1   # a xor b
nor    $t3, $t3, $zero
```

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

- Però es pot calcular: suposem la suma *s* = *a*+*b*

$$overflow = (\overline{a_{31} \oplus b_{31}}) \cdot \boxed{(a_{31} \oplus s_{31})}$$

- En assemblador:

```
addu  $t2, $t0, $t1   # a + b
xor   $t3, $t0, $t1   # a xor b
nor   $t3, $t3, $zero
xor   $t4, $t0, $t2   # a xor s
```

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

- Però es pot calcular: suposem la suma $s = a+b$

$$overflow = \boxed{\left(\overline{a_{31} \oplus b_{31}}\right) \cdot \left(a_{31} \oplus s_{31}\right)}$$

- En assemblador:

```
addu   $t2, $t0, $t1   # a + b
xor    $t3, $t0, $t1   # a xor b
nor    $t3, $t3, $zero
xor    $t4, $t0, $t2   # a xor s
and    $t3, $t3, $t4
```

Detecció d'overflow

- Com pot detectar el software que hi ha overflow?
  - MIPS no té instruccions específiques

- Però es pot calcular: suposem la suma $s = a+b$

$$overflow = (\overline{a_{31} \oplus b_{31}}) \cdot (a_{31} \oplus s_{31})$$

- En assemblador:

```
addu  $t2, $t0, $t1   # a + b
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    # mou el bit 31 a posició 0
```

Aritmètica d'enters

- Suma i resta
- Multiplicació
- Divisió

Multiplicació de naturals

- En base 10

	348	multiplicand
x	951	multiplicador
	348	= 348 x 1
	1740	= 348 x 50
+	3132	= 348 x 900
	330948	

Multiplicació de naturals

- En base 2

	1010	multiplicand
x	1101	multiplicador
	1010	= 1010 x 1
	0000	= 1010 x 00
	1010	= 1010 x 100
+	1010	= 1010 x 1000
	10000010	

- Problema: hem d'emmagatzemar els 4 productes parcials per fer la suma final

Multiplicació de naturals

- Solució: acumular els productes parcials
  - Sols hem de guardar el valor acumulat (P)

Inicialment, P=0: 0 0 0 0 0 0 0 0 $= P_0$

Multiplicació de naturals

- Solució: acumular els productes parcials
  - Sols hem de guardar el valor acumulat (P)

Acumulem el primer producte

$$
\begin{array}{lrl}
 & 0\,0\,0\,0\,0\,0\,0\,0 & = P_0 \\
1010 \times 1 = & 1\,0\,1\,0\ + & \\
\hline
 & 0\,0\,0\,0\,1\,0\,1\,0 & = P_1
\end{array}
$$

Multiplicació de naturals

- Solució: acumular els productes parcials
  - Sols hem de guardar el valor acumulat (P)

Acumulem el segon producte

$$
\begin{array}{rrl}
 & 0\,0\,0\,0\,0\,0\,0\,0 & = P_0 \\
1010 \times 1 = & 1\,0\,1\,0\; + & \\
\hline
 & 0\,0\,0\,0\,1\,0\,1\,0 & = P_1 \\
1010 \times 00 = & 0\; + & \\
\hline
 & 0\,0\,0\,0\,1\,0\,1\,0 & = P_2
\end{array}
$$

Multiplicació de naturals

- Solució: acumular els productes parcials
  - Sols hem de guardar el valor acumulat (P)

Acumulem el tercer producte

$$
\begin{array}{lrl}
 & 0\,0\,0\,0\,0\,0\,0\,0 & = P_0 \\
1010 \times 1 = & 1\,0\,1\,0 + & \\
\hline
 & 0\,0\,0\,0\,1\,0\,1\,0 & = P_1 \\
1010 \times 00 = & 0 + & \\
\hline
 & 0\,0\,0\,0\,1\,0\,1\,0 & = P_2 \\
1010 \times 100 = & 1\,0\,1\,0\,0\,0 + & \\
\hline
 & 0\,0\,1\,1\,0\,0\,1\,0 & = P_3
\end{array}
$$

Multiplicació de naturals

- Solució: acumular els productes parcials
  - Sols hem de guardar el valor acumulat (P)

Acumulem el quart producte

	0 0 0 0 0 0 0 0		$= P_0$
1010 x 1 =	1 0 1 0	+	
	0 0 0 0 1 0 1 0		$= P_1$
1010 x 00 =	0	+	
	0 0 0 0 1 0 1 0		$= P_2$
1010 x 100 =	1 0 1 0 0 0	+	
	0 0 1 1 0 0 1 0		$= P_3$
1010 x 1000 =	1 0 1 0 0 0 0	+	
	1 0 0 0 0 0 1 0		$= P_4$

Multiplicació de naturals

	0 0 0 0 0 0 0 0	= P_0
1010 x 1 =	1 0 1 0 +	
	0 0 0 0 1 0 1 0	= P_1
1010 x 00 =	0 +	
	0 0 0 0 1 0 1 0	= P_2
1010 x 100 =	1 0 1 0 0 0 +	
	0 0 1 1 0 0 1 0	= P_3
1010 x 1000 =	1 0 1 0 0 0 0 +	
	1 0 0 0 0 0 1 0	= P_4

- Hardware necessari per al multiplicador seqüencial:
  - Registre Multiplicand (**MD**, 8 bits), que es desplaça a l'esquerra

Multiplicació de naturals

	0 0 0 0 0 0 0 0	= P_0
1010 x 1 =	1 0 1 0 +	
	0 0 0 0 1 0 1 0	= P_1
1010 x 00 =	0 +	
	0 0 0 0 1 0 1 0	= P_2
1010 x 100 =	1 0 1 0 0 0 +	
	0 0 1 1 0 0 1 0	= P_3
1010 x 1000 =	1 0 1 0 0 0 0 +	
	1 0 0 0 0 0 1 0	= P_4

- Hardware necessari per al multiplicador seqüencial:
  - Registre Multiplicand (MD, 8 bits), que es desplaça a l'esquerra
  - Registre Multiplicador (**MR**, 4 bits)

Multiplicació de naturals

	$\boxed{0\,0\,0\,0\,0\,0\,0\,0}$	$= P_0$
1010 x 1 =	1 0 1 0 +	
	$\boxed{0\,0\,0\,0\,1\,0\,1\,0}$	$= P_1$
1010 x 00 =	0 +	
	$\boxed{0\,0\,0\,0\,1\,0\,1\,0}$	$= P_2$
1010 x 100 =	1 0 1 0 0 0 +	
	$\boxed{0\,0\,1\,1\,0\,0\,1\,0}$	$= P_3$
1010 x 1000 =	1 0 1 0 0 0 0 +	
	$\boxed{1\,0\,0\,0\,0\,0\,1\,0}$	$= P_4$

- Hardware necessari per al multiplicador seqüencial:
  - Registre Multiplicand (MD, 8 bits), que es desplaça a l'esquerra
  - Registre Multiplicador (MR, 4 bits)
  - Registre Producte on acumular els parcials (**P**, 8 bits)

Multiplicació de naturals

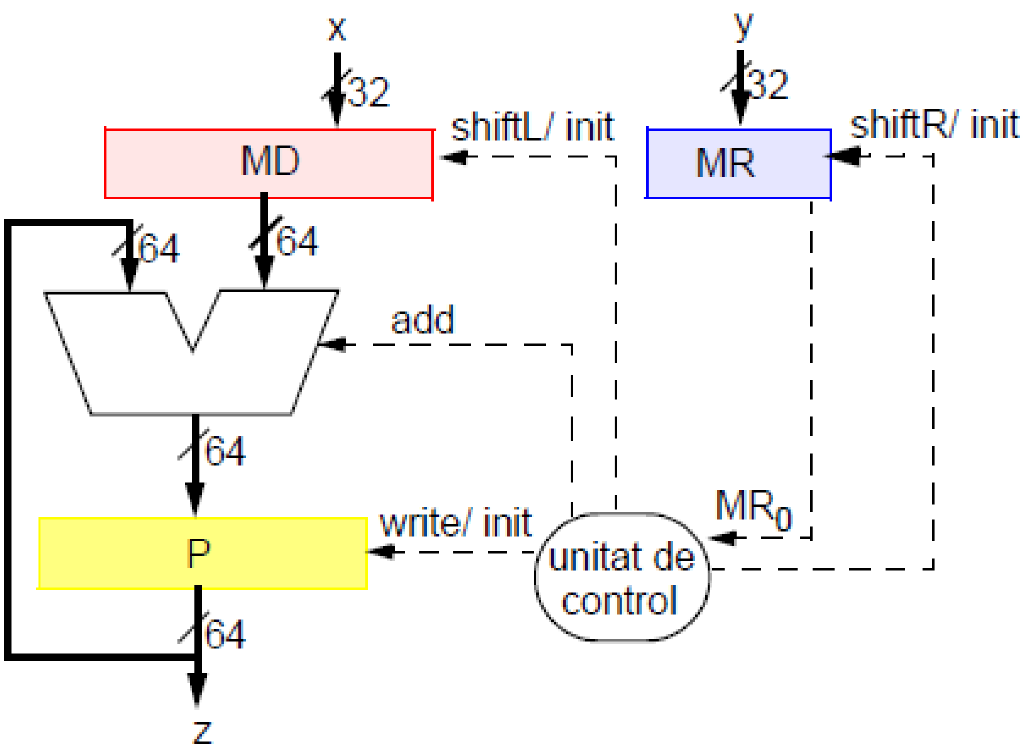
	0 0 0 0 0 0 0 0	$= P_0$
1010 x 1 =	1 0 1 0 +	
	0 0 0 0 1 0 1 0	$= P_1$
1010 x 00 =	0 +	
	0 0 0 0 1 0 1 0	$= P_2$
1010 x 100 =	1 0 1 0 0 0 +	
	0 0 1 1 0 0 1 0	$= P_3$
1010 x 1000 =	1 0 1 0 0 0 0 +	
	1 0 0 0 0 0 1 0	$= P_4$

- Hardware necessari per al multiplicador seqüencial:
  - Registre Multiplicand (MD, 8 bits), que es desplaça a l'esquerra
  - Registre Multiplicador (MR, 4 bits)
  - Registre Producte on acumular els parcials (P, 8 bits)
  - **Sumador** de 8 bits
  - **Desplaçar** MR a la dreta i consultar sempre el bit de menys pes

Multiplicació de naturals: circuit

- Naturals de 32 bits amb resultat de 64 bits
  - Tarda 33 cicles... (suposant que una suma de 64 bits tarda 1 cicle)

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;
```

Multiplicació de naturals

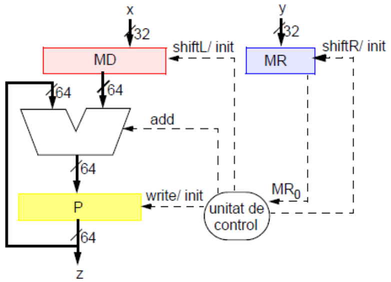
- Exemple. Multipliquem x × y: 1010 × 1101 (amb 4 bits)

Init: Inicialitzem $MD_{0:3}$ = x, MR = y, i P=0

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**

```
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;
```



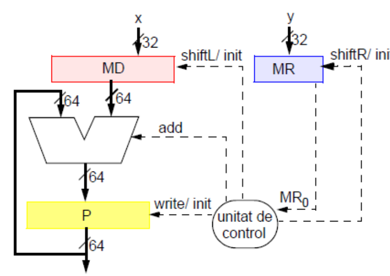
Multiplicació de naturals

Iter. 1: Com que **MR_0=1**, sumem P = P + MD

P	0	0	0	0	**0**	**0**	**0**	**0**
MD	0	0	0	0	**1**	**0**	**1**	**0**
P'		0	0	0	**0**	**1**	**0**	**1**	**0**

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1													0	0	0	**0**	**1**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```

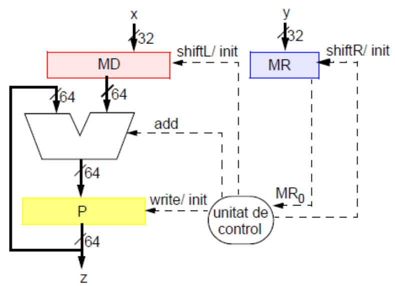


Multiplicació de naturals

Iter. 1: Com que MR_0=1, sumem P = P + MD
Desplacem **MD** a esquerra i **MR** a dreta

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```

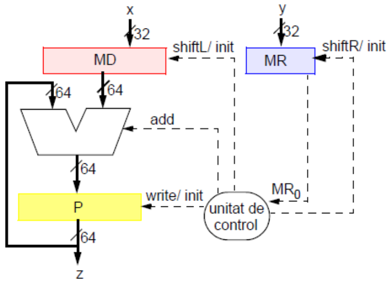


Multiplicació de naturals

Iter. 2: Com que **MR_0=0**, P no es modifica

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**
2													0	0	**0**	**0**	**1**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```

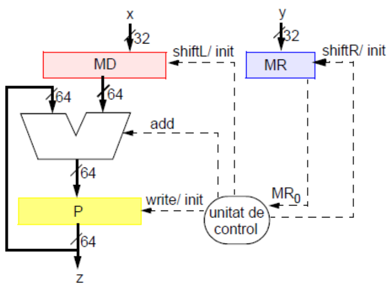


Multiplicació de naturals

Iter. 2: Com que MR_0=0, P no es modifica
Desplacem **MD** a esquerra i **MR** a dreta

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**
2	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**	0	0	**0**	**0**	**1**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```



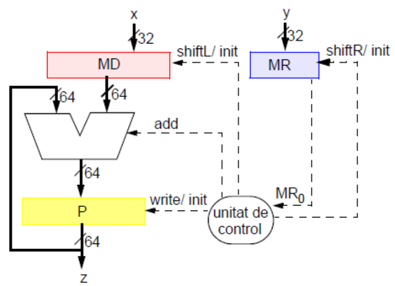
Multiplicació de naturals

Iter. 3: Com que **MR_0=1**, sumem P = P + MD

P	0	0	**0**	**0**	**1**	**0**	**1**	**0**
MD	0	0	**1**	**0**	**1**	**0**	0	0
P'	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**
2	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**	0	0	**0**	**0**	**1**	**0**	**1**	**0**
3													0	**0**	**1**	**1**	**0**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```

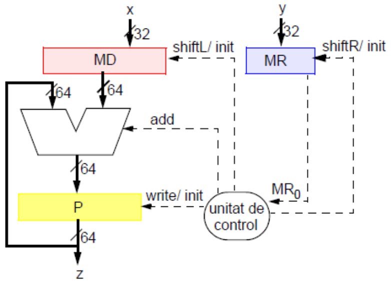


Multiplicació de naturals

Iter. 3: Com que MR_0=1, sumem P = P + MD
Desplacem **MD** a esquerra i **MR** a dreta

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**
2	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**	0	0	**0**	**0**	**1**	**0**	**1**	**0**
3	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```



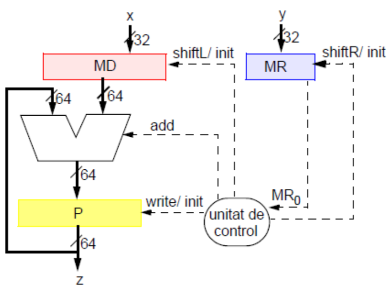
Multiplicació de naturals

Iter. 4: Com que **MR_0=1**, sumem P = P + MD

P	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**
MD	0	**1**	**0**	**1**	**0**	0	0	0
P'	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**
2	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**	0	0	**0**	**0**	**1**	**0**	**1**	**0**
3	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**
4													**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```

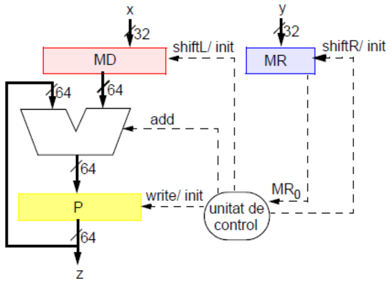


Multiplicació de naturals

Iter. 4: Com que MR_0=1, sumem P = P + MD
Desplacem **MD** a esquerra i **MR** a dreta

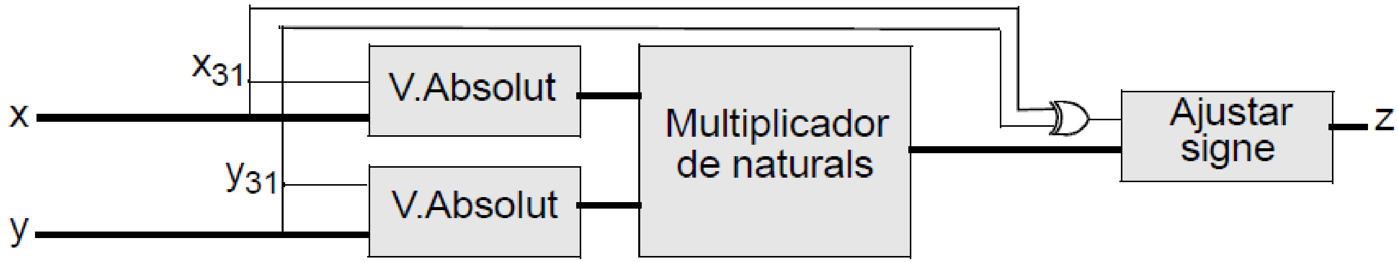
Iter.	MD (Multiplicand)								MR (Multiplicador)				P (Producte)							
Init	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**	0	0	0	**0**	**1**	**0**	**1**	**0**
2	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**	0	0	**0**	**0**	**1**	**0**	**1**	**0**
3	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**
4	**1**	**0**	**1**	**0**	0	0	0	0	0	0	0	0	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

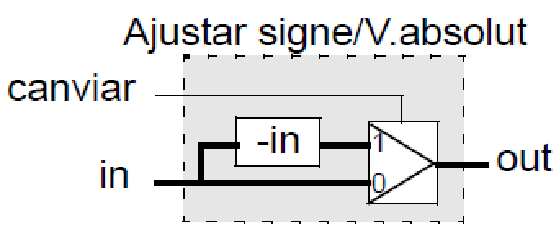
```
if (MR0 == 1)
        P = P + MD;
MD = MD << 1;
MR = MR >> 1;
```



Multiplicació d'enters

1. Calcular els valors absoluts
2. Multiplicar valors absoluts (producte de naturals)
3. Canviar el signe del resultat si els operands tenen signe diferent





Multiplicació d'enters

- Instruccions MIPS

```
mult    rs, rt       # $hi:$lo ← rs * rt (enters)
multu   rs, rt       # $hi:$lo ← rs * rt (naturals)
```

Multiplicació d'enters

- Instruccions MIPS

```
mult    rs, rt        # $hi:$lo ← rs * rt (enters)
multu   rs, rt        # $hi:$lo ← rs * rt (naturals)
```

- `$hi` i `$lo` són registres especials
  - No es poden usar a la resta d'instruccions estudiades fins ara

Multiplicació d'enters

- Instruccions MIPS

```
mult     rs, rt        # $hi:$lo ← rs * rt (enters)
multu    rs, rt        # $hi:$lo ← rs * rt (naturals)
```

- `$hi` i `$lo` són registres especials
  - No es poden usar a la resta d'instruccions estudiades fins ara
- Per moure el resultat a registres de propòsit general:

```
mflo     rd            # rd ← $lo
mfhi     rd            # rd ← $hi
```

Multiplicació d'enters

- Instruccions MIPS

```
mult    rs, rt      # $hi:$lo ← rs * rt (enters)
multu   rs, rt      # $hi:$lo ← rs * rt (naturals)
```

- `$hi` i `$lo` són registres especials
  - No es poden usar a la resta d'instruccions estudiades fins ara
- Per moure el resultat a registres de propòsit general:

```
mflo    rd          # rd ← $lo
mfhi    rd          # rd ← $hi
```

- Overflow
  - Naturals: si `$hi` ≠ 0
  - Enters: si `$hi:$lo` no és l'extensió de signe de `$lo`

Aritmètica d'enters

- Suma i resta
- Multiplicació
- Divisió

Divisió de naturals

- En base 10 (4 dígits):   0421 / 0013

Dividend → 0 4 2 1 | 0 0 1 3 ← Divisor

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

Dividend →	0 0 0 0 0 **4 2 1**	0 0 **1 3**	← Divisor

Inicialment extenem el dividend 0421 a 8 dígits...

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

Dividend → 0 0 0 0 0 4 2 1 | 0 0 1 3 ← Divisor

0 0 **1 3** 0 0 0 0

... i ubiquem el divisor 0013 a sota, alineat a l'esquerra

Divisió de naturals

- En base 10 (4 dígits):   0421 / 0013

```
0 0 0 0 0 4 2 1 | 0 0 1 3
0 0 0 1 3 0 0 0
```

Pas 1.- Desplacem 0013 un lloc a la dreta...

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

0	0	0	0	0	4	2	1		0	0	1	3
0	0	0	1	3	0	0	0		0			

... i comparem amb el dividend:

$421 - 13000 < 0$

No hi cap:

$\rightarrow$ 0 al quocient

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

```
0 0 0 0 0 4 2 1 | 0 0 1 3
0 0 0 0 1 3 0 0   0
```

Pas 2.- Desplacem 0013 un lloc més...

Divisió de naturals

- En base 10 (4 dígits):   0421 / 0013

0	0	0	0	0	4	2	1		0	0	1	3
0	0	0	0	1	3	0	0		0	**0**		

i comparem:
421 - 1300 < 0

No hi cap:

→ 0 al quocient

Divisió de naturals

- En base 10 (4 dígits):   0421 / 0013

0	0	0	0	0	4	2	1		0	0	1	3
0	0	0	**0**	**0**	**1**	**3**	0		0	0		

Pas 3.- Desplacem 0013 un lloc més...

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

0	0	0	0	0	4	2	1		0	0	1	3
0	0	0	**0**	**0**	**1**	**3**	0		0	0		

i comparem:
$421 - 130 \geq 0$

Hi cap

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

0	0	0	0	0	4	2	1		0	0	1	3
0	0	0	0	0	**3**	**9**	0		0	0	**3**	

Hi cap a 3:

$3 \times 130 = 390$

$\rightarrow$ 3 al quocient

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

```
  0 0 0 0 0 4 2 1 | 0 0 1 3
- 0 0 0 0 0 3 9 0   0 0 3
  0 0 0 0 0 0 3 1
```

I restem:

421 – 390 = 31

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

```
0 0 0 0 0 4 2 1 | 0 0 1 3
0 0 0 0 0 3 9 0   0 0 3
0 0 0 0 0 0 3 1
0 0 0 0 0 0 1 3
```

Pas 4.- Desplacem 0013 un lloc més...

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

```
0 0 0 0 0 4 2 1 | 0 0 1 3
                 ---------
0 0 0 0 0 3 9 0   0 0 3
          -----
0 0 0 0 0 0 3 1
0 0 0 0 0 0 1 3
```

i comparem:
31 – 13 >= 0

Hi cap

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

0	0	0	0	0	4	2	1		0	0	1	3
0	0	0	0	0	3	9	0		0	0	3	**2**
0	0	0	0	0	0	3	1					
0	0	0	0	0	0	**2**	**6**					

Hi cap a 2:

$2 \times 13 = 26$

$\rightarrow$ 2 al quocient

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

```
  0 0 0 0 0 4 2 1 | 0 0 1 3
                  +--------
  0 0 0 0 0 3 9 0   0 0 3 2
            -----
  0 0 0 0 0 0 3 1
- 0 0 0 0 0 0 2 6
              ---
  0 0 0 0 0 0 0 5
```

i restem:

31 -26 = 5

Divisió de naturals

- En base 10 (4 dígits): 0421 / 0013

	0	0	0	0	0	4	2	1	0	0	1	3		
	0	0	0	0	0	3	9	0	**0**	**0**	**3**	**2**		← quocient
	0	0	0	0	0	0	3	1						
–	0	0	0	0	0	0	2	6						
	0	0	0	0	**0**	**0**	**0**	**5**	← residu					

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

Dividend → 1 0 1 1 | 0 0 1 0 ← Divisor

Divisió de naturals

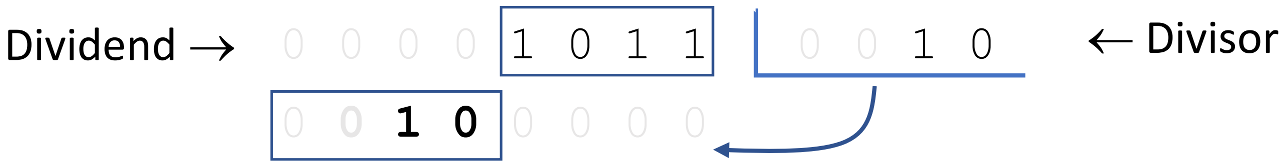
- En base 2 (4 dígits):   1011 / 0010

Dividend → 0 0 0 0 **1 0 1 1** | 0 0 **1 0** ← Divisor

Inicialment extenem el dividend 1011 a 8 dígits...

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010



... i ubiquem el divisor 0010 a sota, alineat a l'esquerra

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

```
0 0 0 0 1 0 1 1 | 0 0 1 0
0 0 0 1 0 0 0 0
```

Pas 1.- Desplacem 0010 un lloc a la dreta...

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

	0	0	0	0	1	0	1	1		0	0	1	0
−	0	0	0	1	0	0	0	0		0			
							?						

... i comparem (restem) :
1011 – 10000 < 0

No hi cap:

→ 0 al quocient

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

```
0 0 0 0 1 0 1 1 | 0 0 1 0
0 0 0 0 1 0 0 0   0
```

Pas 2.- Desplacem 0010 un lloc més...

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

	0	0	0	0	1	0	1	1		0	0	1	0
–	0	0	**0**	**0**	1	0	0	0		0	**1**		
	0	0	0	0	**0**	**0**	**1**	**1**					

... i comparem (restem):
1011 – 1000 ≥ 0

Hi cap

→ 1 al quocient

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

	0	0	0	0	1	0	1	1		0	0	1	0
−	0	0	0	0	1	0	0	0		0	1		
	0	0	0	0	0	0	1	1					
	0	0	0	0	0	**1**	**0**	0					

Pas 3.- Desplacem 0010 un lloc més...

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

0 0 0 0 1 0 1 1 | 0 0 1 0

0 0 0 0 1 0 0 0   0 1 **0**

0 0 0 0 0 0 1 1

− 0 0 0 0 0 1 0 0

?

... i comparem (restem):
11 – 100 < 0

No hi cap

→ 0 al quocient

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

```
0 0 0 0 1 0 1 1 | 0 0 1 0
0 0 0 0 1 0 0 0   0 1 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 0
```

Pas 4.- Desplacem 0010 un lloc més...

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

	0	0	0	0	1	0	1	1	0	0	1	0
	0	0	0	0	1	0	0	0	0	1	0	**1**
	0	0	0	0	0	0	1	1				
–	0	0	0	0	0	0	1	0				
	0	0	0	0	**0**	**0**	**0**	**1**				

... i comparem (restem):
$11 - 10 \geq 0$

Hi cap

$\rightarrow$ 1 al quocient

Divisió de naturals

- En base 2 (4 dígits): 1011 / 0010

	0	0	0	0	1	0	1	1		0	0	1	0		
	0	0	0	0	1	0	0	0		**0**	**1**	**0**	**1**		← quocient
	0	0	0	0	0	0	1	1							
–	0	0	0	0	0	0	1	0							
	0	0	0	0	**0**	**0**	**0**	**1**	← residu						

Hardware per a un divisior de 4 bits

- Un registre D, de 8 bits per al divisor
  - Inicialment hi ubicarem el divisor alineat a l'esquerra
  - I en cada pas successiu el desplaçarem a la dreta 1 bit

Hardware per a un divisior de 4 bits

- Un registre D, de 8 bits per al divisor
  - Inicialment hi ubicarem el divisor alineat a l'esquerra
  - I en cada pas successiu el desplaçarem a la dreta 1 bit
- Un registre R, de 8 bits per al dividend
  - Inicialment extendrem el dividend amb zeros a l'esquerra
  - I en cada pas li restarem els successius divisors: R = R – D
  - Al final contindrà el Residu

Hardware per a un divisior de 4 bits

- Un registre D, de 8 bits per al divisor
  - Inicialment hi ubicarem el divisor alineat a l'esquerra
  - I en cada pas successiu el desplaçarem a la dreta 1 bit
- Un registre R, de 8 bits per al dividend
  - Inicialment extendrem el dividend amb zeros a l'esquerra
  - I en cada pas li restarem els successius divisors: R = R – D
  - Al final contindrà el Residu
- Un registre Q, de 4 bits per al quocient
  - Inicialment a zero
  - En cada pas el desplaçarem a l'esquerra insertant-li un 1 o un 0

Hardware per a un divisior de 4 bits

- Un registre D, de 8 bits per al divisor
  - Inicialment hi ubicarem el divisor alineat a l'esquerra
  - I en cada pas successiu el desplaçarem a la dreta 1 bit
- Un registre R, de 8 bits per al dividend
  - Inicialment extendrem el dividend amb zeros a l'esquerra
  - I en cada pas li restarem els successius divisors: R = R – D
  - Al final contindrà el Residu
- Un registre Q, de 4 bits per al quocient
  - Inicialment a zero
  - En cada pas el desplaçarem a l'esquerra insertant-li un 1 o un 0
- Un registre temporal T de 8 bits
  - Hi escrivim el resultat de la comparació (resta): T = R – D
  - Si T ≥ 0, es copia T en R, altrament es descarta

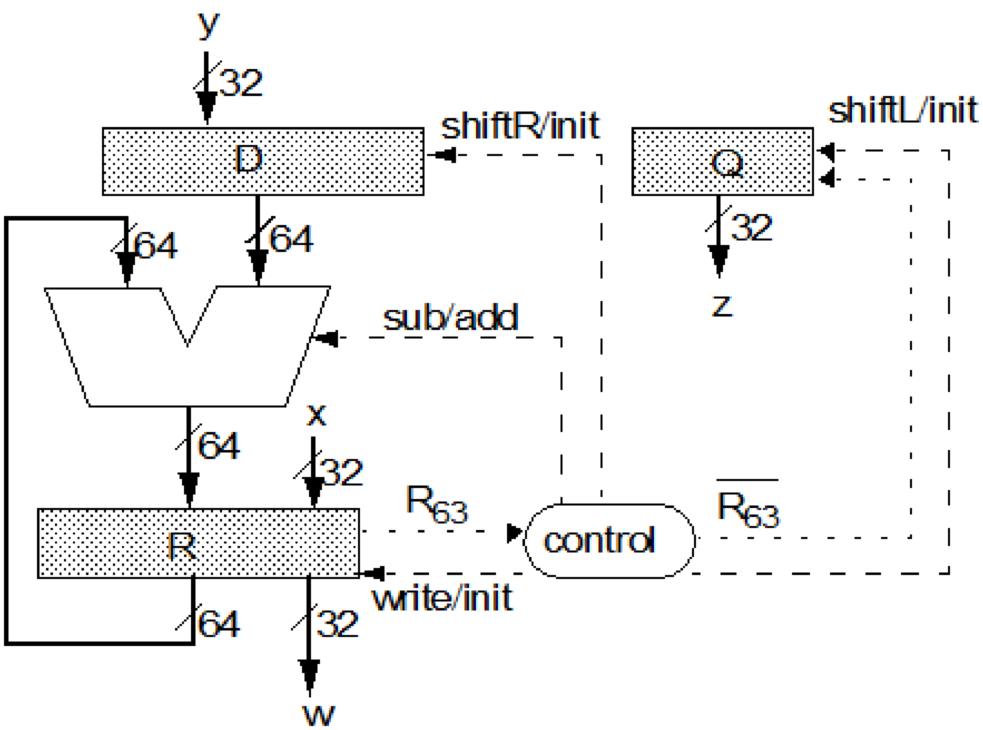
Hardware per a un divisior de 4 bits

- Un registre D, de 8 bits per al divisor
  - Inicialment hi ubicarem el divisor alineat a l'esquerra
  - I en cada pas successiu el desplaçarem a la dreta 1 bit
- Un registre R, de 8 bits per al dividend
  - Inicialment extendrem el dividend amb zeros a l'esquerra
  - I en cada pas li restarem els successius divisors: R = R – D
  - Al final contindrà el Residu
- Un registre Q, de 4 bits per al quocient
  - Inicialment a zero
  - En cada pas el desplaçarem a l'esquerra insertant-li un 1 o un 0
- Un registre temporal T de 8 bits
  - Hi escrivim el resultat de la comparació (resta): T = R – D
  - Si T ≥ 0, es copia T en R, altrament es descarta
- Algorisme de divisió "amb restauració": estalvia el registre temporal
  - Escrivim en R el resultat de la resta: R = R – D
  - Si R < 0, "restaurem" el seu valor anterior: R = R + D

Divisió de naturals: circuit

- Divisió de naturals de 32 bits "amb restauració"
  - Quocient: *z = x / y*
  - Residu: *w = x % y*

Divisor seqüencial de naturals

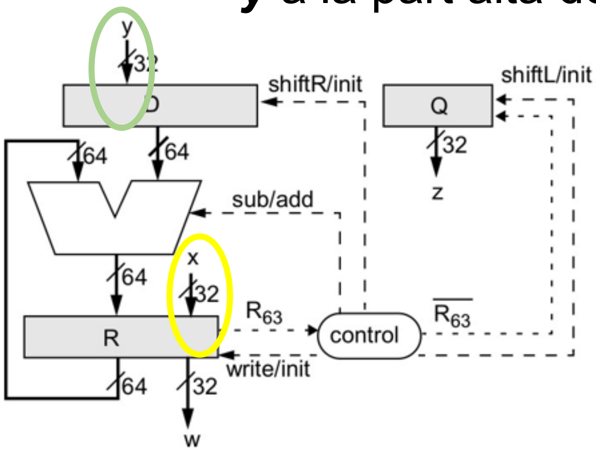


Pseudocodi

```
// Inicialització
R0:31 = x; R32:63 = 0;
D0:31 = 0; D32:63 = y;
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;
      }
}
z = Q;
w = R0:31
```

Exemple divisió x/y: 1011 / 0010

Init: ***x*** a la part baixa del Dividend/Residu (**R**)
y a la part alta del Divisor (**D**), i zeros a la resta



// Inicialització

$R_{0:31}$ = x; $R_{32:63}$ = 0;

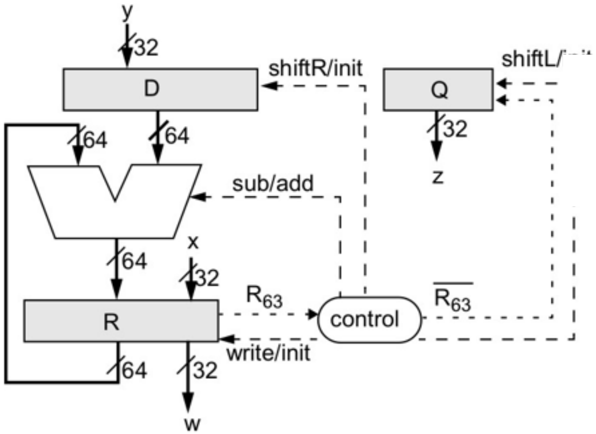
$D_{0:31}$ = 0; $D_{32:63}$ = y;

Q = 0;

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 x/y: 1011 / 0010

Iter 1: Desplacem **D** a dreta

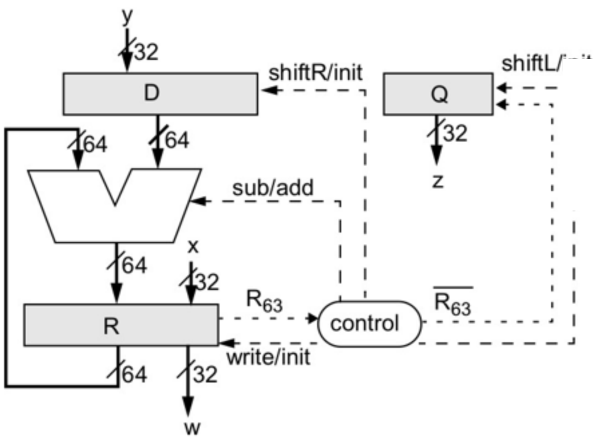



```
D = D >> 1;
R = R - D;
if (R63 == 0)
```

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**	**1**	**0**	0	0	0				

Exemple divisió x/y: 1011 / 0010

Iter 1: Restem **R=R-D** per comparar-los. Comprovem **R < 0**!



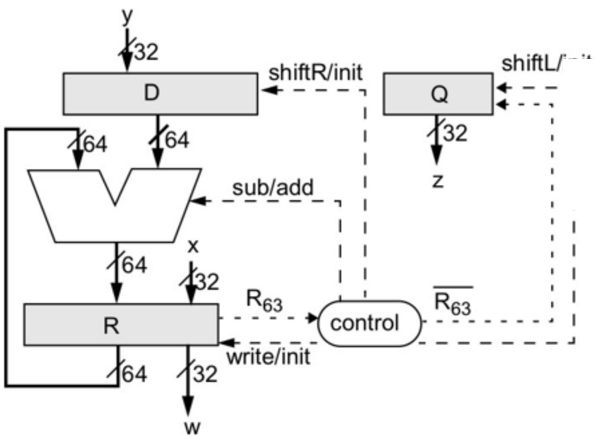

```
D = D >> 1;
R = R - D;
if (R63 == 0)
```

R	0	0	0	0	**1**	**0**	**1**	**1**
– D	0	**0**	**0**	**1**	**0**	0	0	0
R'	**1**	1	1	1	1	0	1	1

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	1	1	1	1	1	0	1	1	0	**0**	**0**	**1**	**0**	0	0	0				

Exemple divisió x/y: 1011 / 0010

Iter 1: **Restaurem R** sumant **R=R+D**



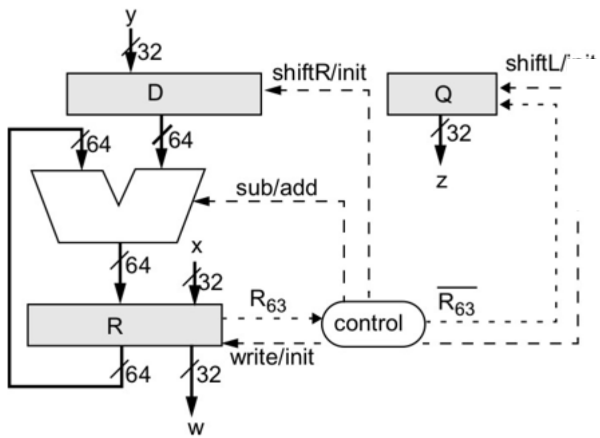

```
D = D >> 1;
R = R - D;
if (R63 == 0)

else {
        R = R + D;
        Q = Q << 1;
}
```

R	1	1	1	1	1	0	1	1
+ D	0	**0**	**0**	**1**	**0**	0	0	0
R'	0	0	0	0	**1**	**0**	**1**	**1**

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				

Exemple divisió x/y: 1011 / 0010

Iter 1: Insertem **Q_0=0** desplaçant Q a l'esquerra

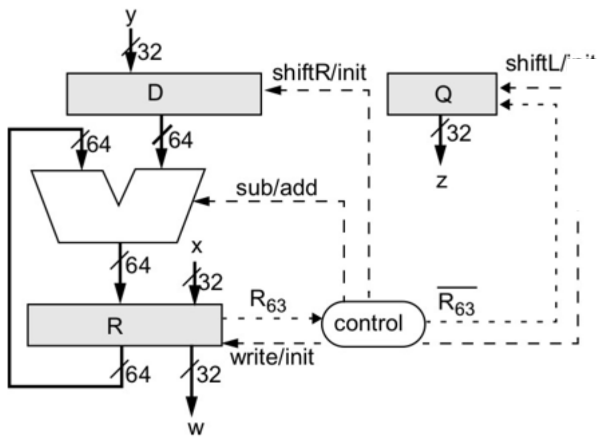



```
D = D >> 1;
R = R - D;
if (R63 == 0)

else {
        R = R + D;
        Q = Q << 1;
}
```

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 x/y: 1011 / 0010

Iter 2: Desplacem **D** a la dreta

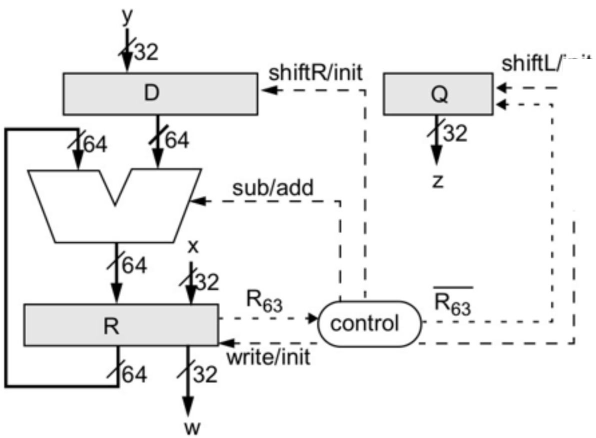



```
D = D >> 1;
R = R - D;
if (R63 == 0)
  Q = (Q << 1) | 1;
```

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**	**1**	**0**	0	0				

Exemple divisió x/y: 1011 / 0010

Iter 2: Restem **R=R-D** i comprovem que **R ≥ 0**



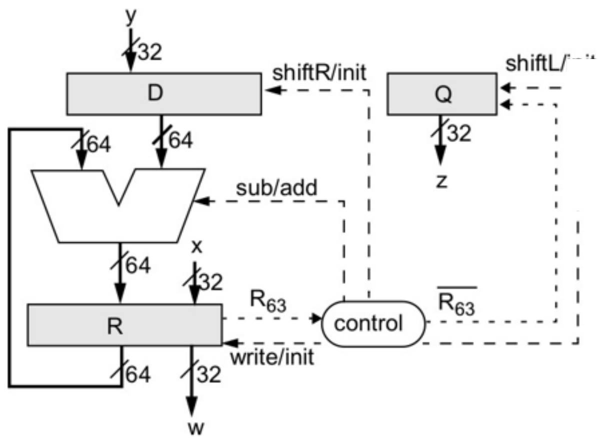

```
D = D >> 1;
R = R - D;
if (R63 == 0)
   Q = (Q << 1) | 1;
```

R	0	0	0	0	**1**	**0**	**1**	**1**
– D	0	0	**0**	**0**	**1**	**0**	0	0
	0	0	0	0	**0**	**0**	**1**	**1**

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				

Exemple divisió x/y: 1011 / 0010

Iter 2: Insertem Q_0=0 desplaçant Q a l'esquerra

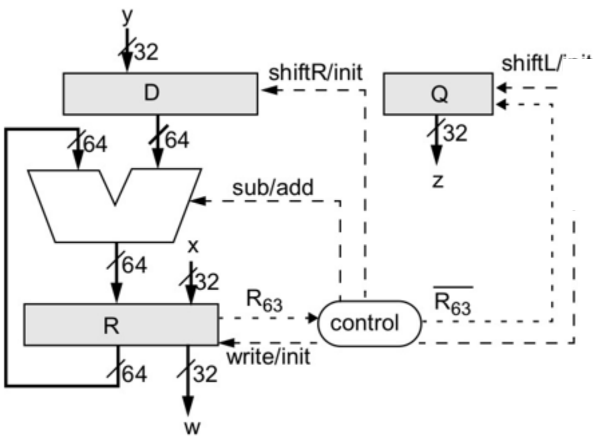



```
D = D >> 1;
R = R - D;
if (R63 == 0)
  Q = (Q << 1) | 1;
```

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 x/y: 1011 / 0010

Iter 3: Desplacem **D** a la dreta

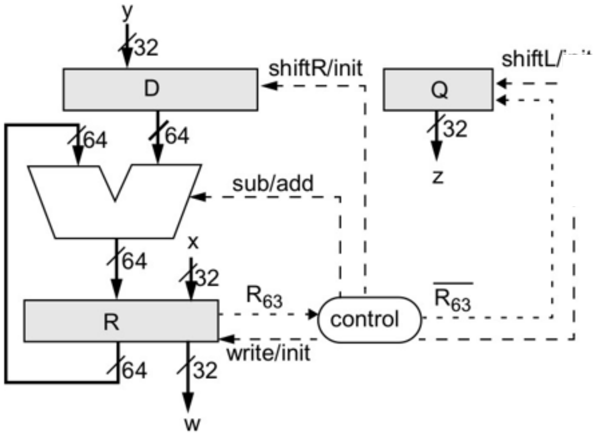



```
D = D >> 1;
R = R - D;
if (R63 == 0)
```

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**	**1**	**0**	0				

Exemple divisió x/y: 1011 / 0010

Iter 3: Restem **R=R-D** per comparar-los. Comprovem **R < 0**!



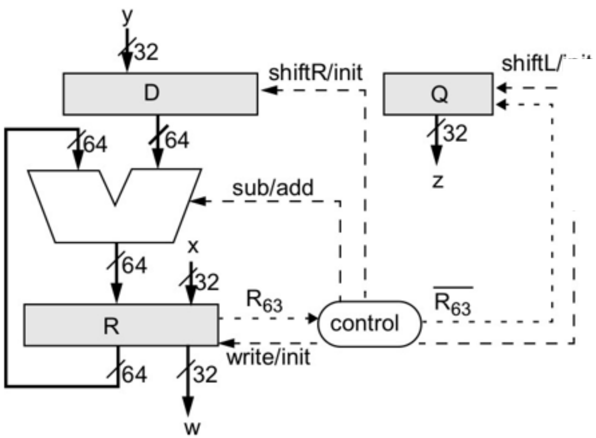

```
D = D >> 1;
R = R - D;
if (R63 == 0)
```

R	0	0	0	0	**0**	**0**	**1**	**1**
– D	0	0	0	**0**	**0**	**1**	**0**	0
	1	1	1	1	1	1	1	1

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	1	1	1	1	1	1	1	1	0	0	0	**0**	**0**	**1**	**0**	0				

Exemple divisió x/y: 1011 / 0010

Iter 3: **Restaurem R** sumant **R=R+D**



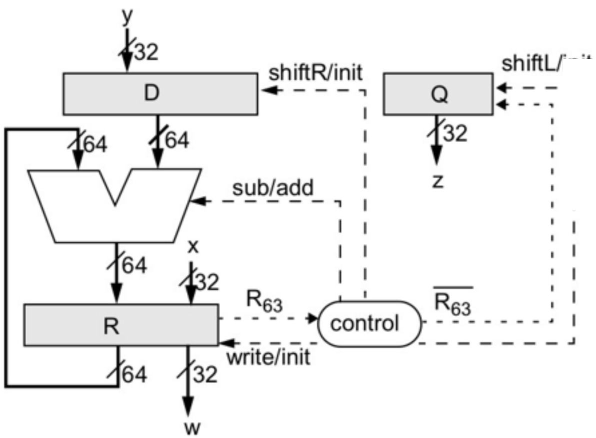

```
D = D >> 1;
R = R - D;
if (R63 == 0)

else {
        R = R + D;
        Q = Q << 1;
}
```

	R	1	1	1	1	1	1	1	1
+	D	0	0	0	**0**	**0**	**1**	**0**	0
		0	0	0	0	**0**	**0**	**1**	**1**

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				

Exemple divisió x/y: 1011 / 0010

Iter 3: Insertem **Q_0=0** desplaçant Q a l'esquerra

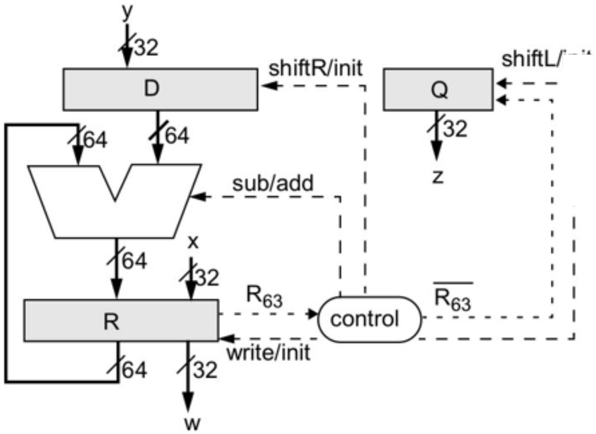



```
D = D >> 1;
R = R - D;
if (R63 == 0)

else {
        R = R + D;
        Q = Q << 1;
}
```

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**

Exemple divisió x/y: 1011 / 0010

Iter 4: Desplacem **D** a la dreta

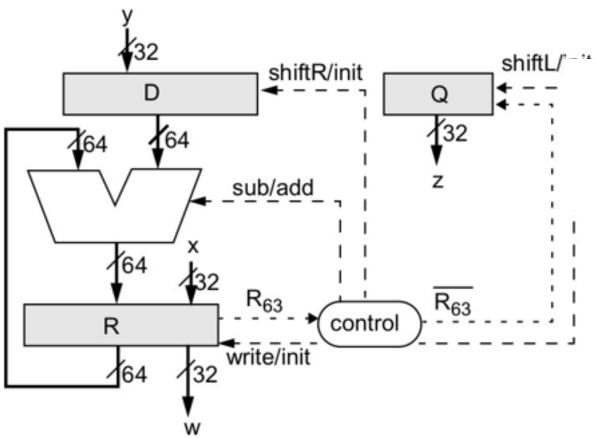



```
D = D >> 1;
R = R - D;
if (R63 == 0)
  Q = (Q << 1) | 1;
```

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**	**1**	**0**				

Exemple divisió x/y: 1011 / 0010

Iter 4: Restem **R=R-D** i comprovem que **R ≥ 0**



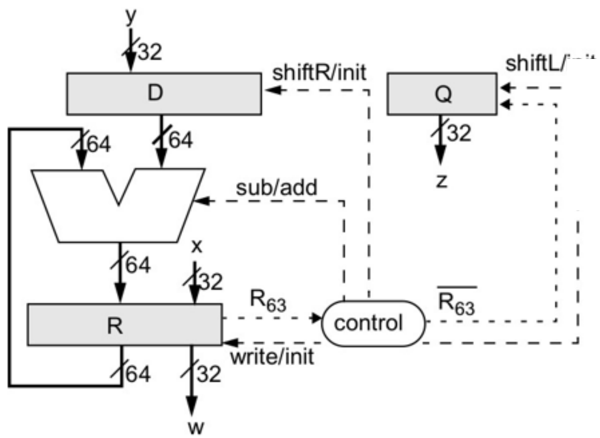

```
D = D >> 1;
R = R - D;
if (R63 == 0)
    Q = (Q << 1) | 1;
```

R	0	0	0	0	**0**	**0**	**1**	**1**
– D	0	0	0	0	**0**	**0**	**1**	**0**
	0	0	0	0	**0**	**0**	**0**	**1**

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**				

Exemple divisió x/y: 1011 / 0010

Iter 4: Insertem **Q_0=1** desplaçant Q a l'esquerra




```
D = D >> 1;
R = R - D;
if (R63 == 0)
    Q = (Q << 1) | 1;
```

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

Divisió d'enters: *x/y*

Si y ≠ 0:

1. Calcular valors absoluts del operands: $|x|$, $|y|$

Divisió d'enters: *x/y*

Si $y \neq 0$:

1. Calcular valors absoluts del operands: $|x|$, $|y|$
2. Dividir els valors absoluts (divisió de naturals)
   - Obtenim quocient i residu

Divisió d'enters: *x/y*

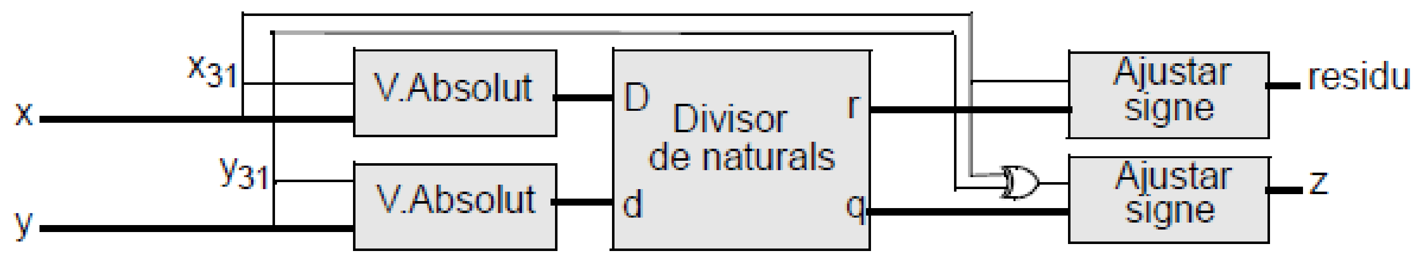
Si $y \neq 0$:

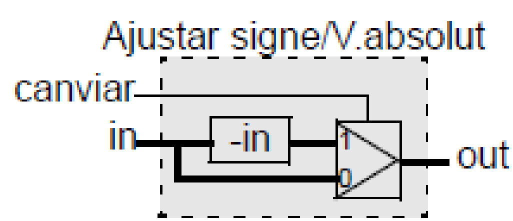
1. Calcular valors absoluts del operands: $|x|$, $|y|$
2. Dividir els valors absoluts (divisió de naturals)
   - Obtenim quocient i residu
3. Ajustar signes
   - Canviar signe del quocient si x, y de diferent signe
   - Canviar signe del residu si el dividend és negatiu

Divisió d'enters: *x/y*

Si y ≠ 0:

1. Calcular valors absoluts del operands: |*x*|, |*y*|
2. Dividir els valors absoluts (divisió de naturals)
   - Obtenim quocient i residu
3. Ajustar signes
   - Canviar signe del quocient si x, y de diferent signe
   - Canviar signe del residu si el dividend és negatiu





Divisió d'enters i naturals

- Instruccions MIPS

```
divu    rs, rt        # Naturals
div     rs, rt        # Enters
```

Divisió d'enters i naturals

- Instruccions MIPS

```
divu   rs, rt        # Naturals
div    rs, rt        # Enters
```

- Operació

  $\$lo \leftarrow rs / rt$ (quocient)

  $\$hi \leftarrow rs \% rt$ (residu)

Divisió d'enters i naturals

- Instruccions MIPS

```
divu    rs, rt          # Naturals
div     rs, rt          # Enters
```

- Operació

  $\$lo \leftarrow rs\ /\ rt$ (quocient)

  $\$hi \leftarrow rs\ \%\ rt$ (residu)

- Si el divisor és 0
  - El resultat és *indefinit*

Divisió d'enters i naturals

- Instruccions MIPS

```
divu    rs, rt          # Naturals
div     rs, rt          # Enters
```

- Operació

  $lo ← rs / rt (quocient)

  $hi ← rs % rt (residu)

- Si el divisor és 0
  - El resultat és *indefinit*
- Overflow
  - Naturals: no n'hi ha
  - Enters: només hi ha un cas. Quin és?

Divisió d'enters i naturals

- Instruccions MIPS

```
divu    rs, rt          # Naturals
div     rs, rt          # Enters
```

- Operació

  $\$lo \leftarrow rs\ /\ rt$ (quocient)

  $\$hi \leftarrow rs\ \%\ rt$ (residu)

- Si el divisor és 0
  - El resultat és *indefinit*

- Overflow
  - Naturals: no n'hi ha
  - Enters: només hi ha un cas. Quin és?

$$\frac{-2^{31}}{-1} = 2^{31} \quad \rightarrow \text{ No representable en Ca2!}$$

Divisió per potències de 2

- **Algunes** divisions per potències de 2
  - Es poden traduir per un shift, molt més ràpid que una divisió

Divisió per potències de 2

- **Algunes** divisions per potències de 2
  - Es poden traduir per un shift, molt més ràpid que una divisió
- Per a naturals
  - `srl` i `divu` calculen el mateix quocient

Divisió per potències de 2

- **Algunes** divisions per potències de 2
  - Es poden traduir per un shift, molt més ràpid que una divisió
- Per a naturals
  - `srl` i `divu` calculen el mateix quocient
- Per a enters
  - Si el dividend és positiu, `sra` i `div` calculen el mateix quocient
  - Però atenció! Si el dividend és negatiu i la divisió no és exacta, `sra` i `div` donen resultats diferents

Divisió per potències de 2

- **Algunes** divisions per potències de 2
  - Es poden traduir per un shift, molt més ràpid que una divisió
- Per a naturals
  - `srl` i `divu` calculen el mateix quocient
- Per a enters
  - Si el dividend és positiu, `sra` i `div` calculen el mateix quocient
  - Però atenció! Si el dividend és negatiu i la divisió no és exacta, `sra` i `div` donen resultats diferents
- En conclusió
  - Traduirem operadors C de divisió (/) i mòdul (%) amb `div` i `divu`
  - Només optimitzarem amb `sra` si el resultat és equivalent a `div`

Aritmètica de coma flotant

- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

Coma fixa

- Com representarem nombres fraccionaris?
  - Necessaris per a la física, l'enginyeria, etc.

Coma fixa

- Com representarem nombres fraccionaris?
  - Necessaris per a la física, l'enginyeria, etc.

- En *coma fixa*
  - Alguns bits per a la part entera, i alguns per a la part fraccionària
  - Exemple amb 8 bits:

    `eeeeefff` → part entera i part fraccionària

Coma fixa

- Com representarem nombres fraccionaris?
  - Necessaris per a la física, l'enginyeria, etc.

- En *coma fixa*
  - Alguns bits per a la part entera, i alguns per a la part fraccionària
  - Exemple amb 8 bits:

    `eeeeefff` → part entera i part fraccionària

    `10101,110 =`

$= 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3}$

$= 21,75_{10}$

Coma fixa

- Com representarem nombres fraccionaris?
  - Necessaris per a la física, l'enginyeria, etc.

- En *coma fixa*
  - Alguns bits per a la part entera, i alguns per a la part fraccionària
  - Exemple amb 8 bits:

    `eeeeefff` → part entera i part fraccionària

    $10101{,}110 =$

    $= 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3}$

    $= 21{,}75_{10}$

  - El rang és bastant limitat
  - Els números representables són equidistants

Coma flotant

- En coma flotant (base 10)
  - També coneguda com *notació exponencial o científica*

    $v = \pm\, m \times 10^{e}$ *m = mantissa, e = exponent*

Coma flotant

- En coma flotant (base 10)
  - També coneguda com *notació exponencial o científica*

    $v = \pm\, m \times 10^{e}$ *m = mantissa, e = exponent*
  - Rang molt més gran que la coma fixa
  - Però números representables no-equidistants

Coma flotant

- En coma flotant (base 10)
  - També coneguda com *notació exponencial o científica*

    $v = \pm\, m \times 10^e$ *m = mantissa, e = exponent*
  - Rang molt més gran que la coma fixa
  - Però números representables no-equidistants

- Notació científica *normalitzada* (base 10)

    $v = \pm\, m \times 10^e$ tal que $1 \leq m < 10$

  - És a dir: la part entera de m ha de tenir 1 sol dígit, i ha de ser no-nul

Coma flotant

- En coma flotant (base 10)
  - També coneguda com *notació exponencial o científica*

    $v = \pm\, m \times 10^{e}$ *m = mantissa, e = exponent*
  - Rang molt més gran que la coma fixa
  - Però números representables no-equidistants

- Notació científica *normalitzada* (base 10)

  $v = \pm\, m \times 10^{e}$ tal que $1 \leq m < 10$

  - És a dir: la part entera de m ha de tenir 1 sol dígit, i ha de ser no-nul
  - Exemples

    2.34×10^{6} → Normalitzat

    0.0234×10^{8} → No normalitzat!

Coma flotant en base 2

- Coma flotant en base 2

$$\boxed{v = \underbrace{\pm}_{\text{signe (S)}} \underbrace{\overset{\text{Part entera}}{1},\overbrace{fff \ldots f}^{\text{fracció (F)}}}_{\text{mantissa}} \times 2^{\underbrace{eee \ldots e}_{\text{exponent (E)}}}}$$

  - En forma més compacta: $v = (-1)^{S} \times (1 + 0,F) \times 2^{E}$

Coma flotant en base 2

- Coma flotant en base 2

$$v = \underset{\text{signe (S)}}{\pm} \underbrace{\overset{\text{Part entera}}{1},\overbrace{fff \ldots f}^{\text{fracció (F)}}}_{\text{mantissa}} \times 2^{\underbrace{eee \ldots e}_{\text{exponent (E)}}}$$

  - En forma més compacta: $v = (-1)^{S} \times (1 + 0,F) \times 2^{E}$

- Format: **s**igne, **e**xponent, **f**racció

$$\boxed{\texttt{s eeeeeeee fffffffffffffffffffffff}}$$

  - Signe: 0=positiu, 1=negatiu
  - Exponent: enter representat “en excés”

Coma flotant en base 2

- Coma flotant en base 2

$$v = \pm \underbrace{\overset{\text{bit ocult}}{1},\overbrace{\text{fff} \ldots \text{f}}^{\text{fracció (F)}}}_{\text{mantissa}} \times 2^{\overbrace{\text{eee} \ldots \text{e}}_{\text{exponent (E)}}}$$

signe (S)

  - En forma més compacta: $v = (-1)^{S} \times (1 + 0{,}F) \times 2^{E}$

- Format: **s**igne, **e**xponent, **f**racció

```
s eeeeeeee fffffffffffffffffffffff
```

  - Signe: 0=positiu, 1=negatiu
  - Exponent: enter representat “en excés”
  - Mantissa: normalitzada
    → La part entera val 1, és implícita i no es codifica (“bit ocult”)
    → Sols es codifica la fracció

Coma flotant en base 2

bit ocult

$$v = \pm \underbrace{1,\overbrace{fff \ldots f}^{\text{fracció (F)}}}_{\text{mantissa}} \times 2^{\underbrace{eee \ldots e}_{\text{exponent (E)}}}$$

signe (S)

- El format és un compromís

```
s eeeeeeee fffffffffffffffffffffff
```

  - Si dedica + bits a exponent → major rang
  - Si dedica + bits a fracció → major precisió

Aritmètica de coma flotant

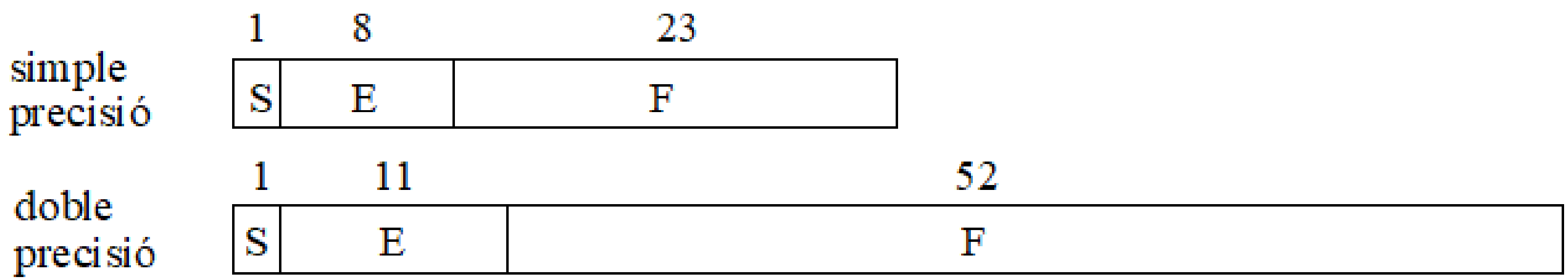
- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

Estàndard IEEE-754 (1985-2019)

- El IEEE-754
  - Estàndard de coma flotant nascut el 1985
  - Necessitat d'intercanviar dades entre diferents sistemes (abans cada fabricant tenia el seu format)
  - Regula representació, operacions, arrodoniment i excepcions
  - Es va renovant cada cert temps, darrerament el 2008 i 2019

Estàndard IEEE-754 (1985-2019)

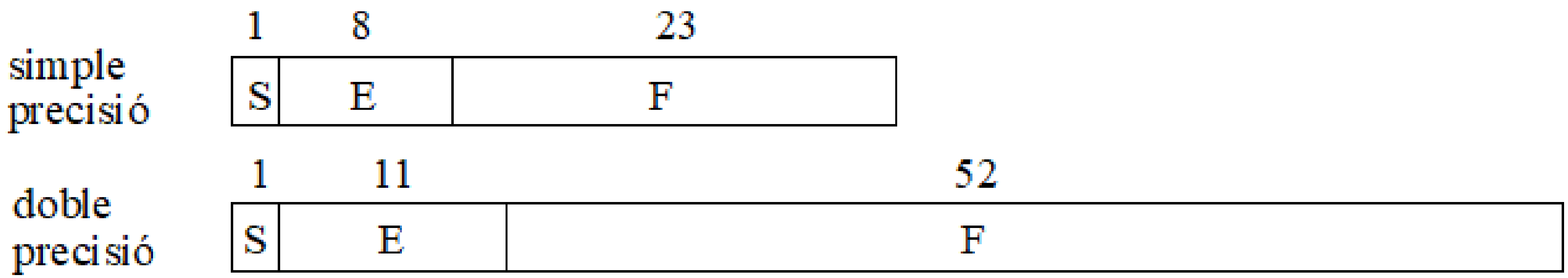
- Dos formats



- En C

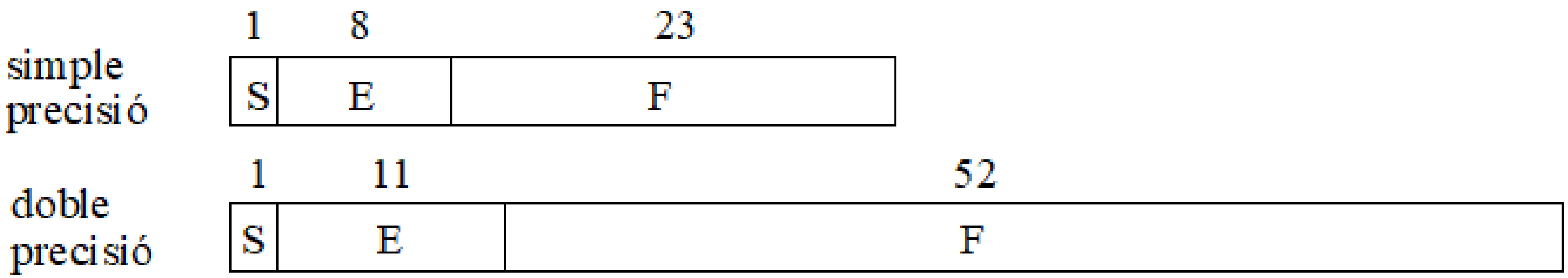
```c
float x;        // simple precisió (4 bytes)
double y;       // doble precisió (8 bytes)
```

Estàndard IEEE-754 (1985-2019)



- Signe: 1 bit
- Exponent: 8 bits / 11 bits
  - Codificat en excés a 127 / 1023
  - Permet comparar magnituds amb un comparador de naturals

Estàndard IEEE-754 (1985-2019)



- Signe: 1 bit
- Exponent: 8 bits / 11 bits
  - Codificat en excés a 127 / 1023
  - Permet comparar magnituds amb un comparador de naturals
- Fracció: 23 bits / 52 bits
  - Part fraccionària de la mantissa
- Part entera = 1
  - *bit ocult* implícit, no es representa

Aritmètica de coma flotant

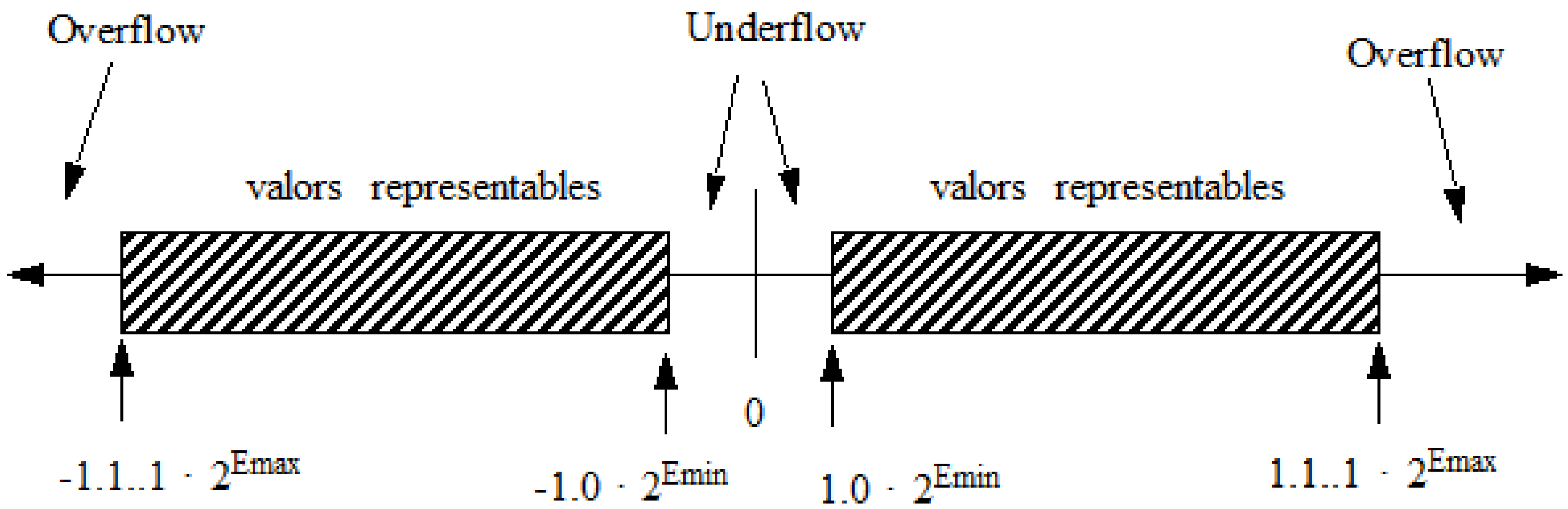
- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

IEEE-754: Valors representables

- Mantissa: `1,000...0` $\leq$ mantissa $\leq$ `1,111...1`

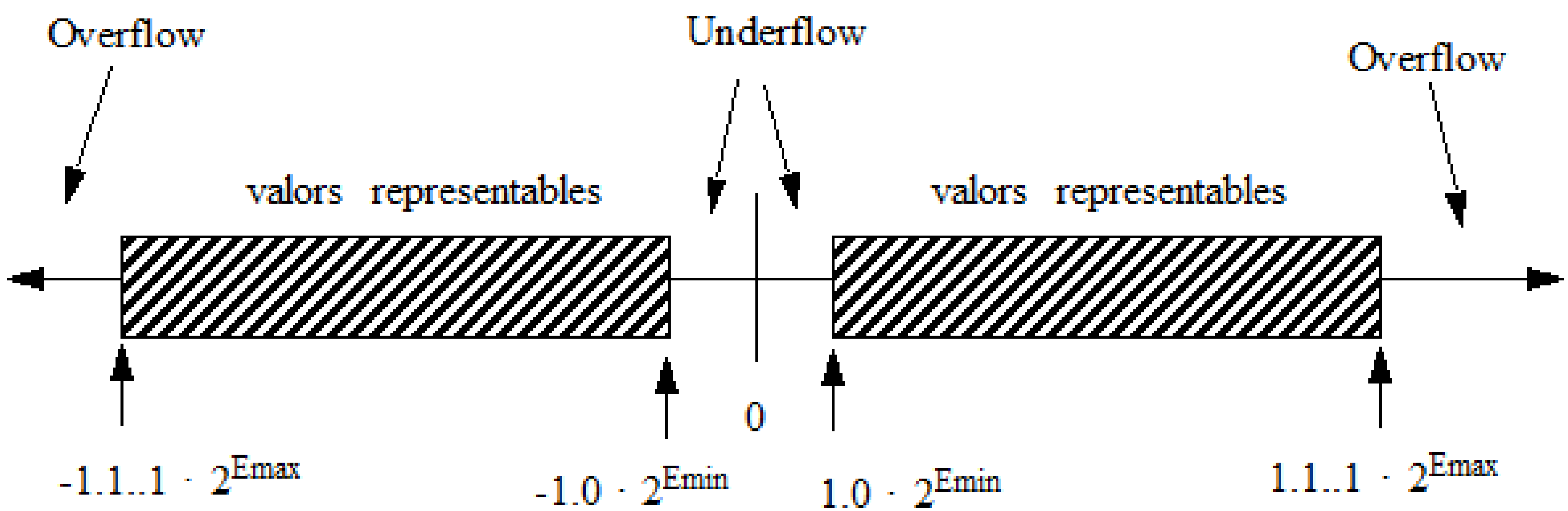
IEEE-754: Valors representables

- Mantissa: `1,000...0` $\leq$ mantissa $\leq$ `1,111...1`
- Exponent: Emin ≤ E ≤ Emax
- Resultats fora de rang (Overflow): E > Emax



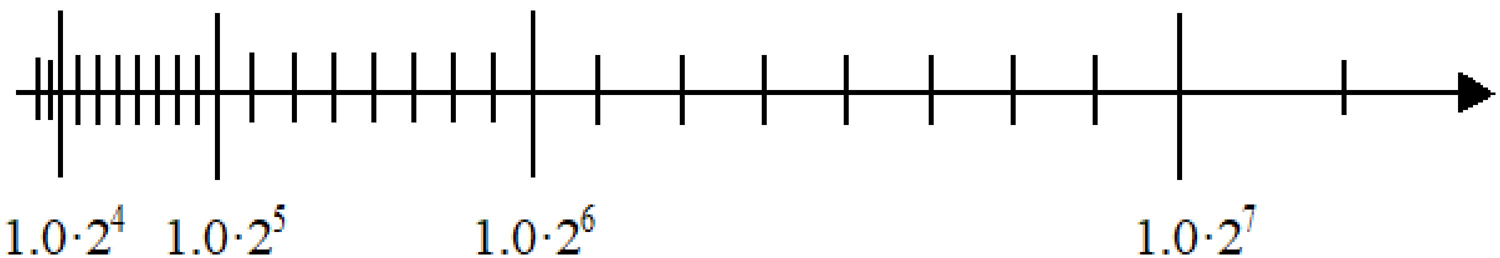
IEEE-754: Valors representables

- Mantissa: `1,000...0` $\leq$ mantissa $\leq$ `1,111...1`
- Exponent: Emin ≤ E ≤ Emax
- Resultats fora de rang (Overflow): E > Emax
- Resultats "no fiables" (Underflow): *magnitud* < $1{,}0 \cdot 2^{Emin}$

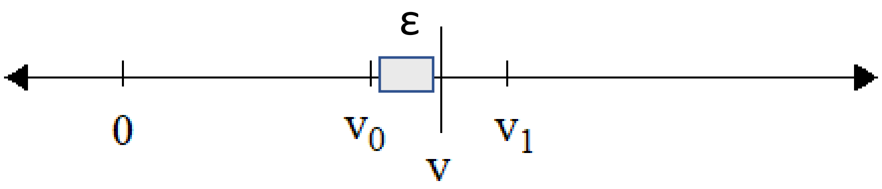


IEEE-754: Valors representables

- Valors representables no equidistants
  - Amb 32 bits es poden representar 2^{32} números, igual que amb coma fixa, però no són equidistants:

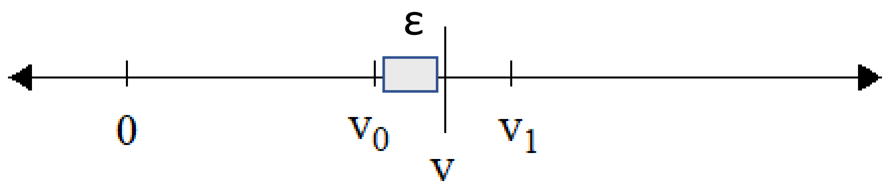


Error de precisió per arrodoniment



- Resultat exacte v està entre 2 valors representables: v_0 i v_1
- Si l'arrodonim a v_0 l'error de precisió és $\varepsilon = |v - v_0|$

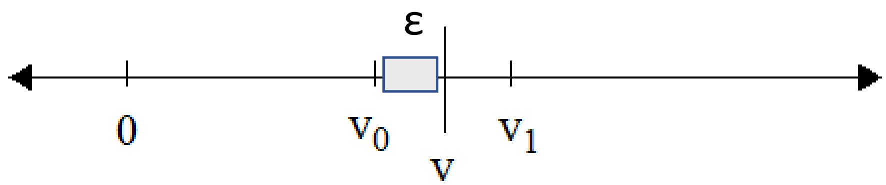
Error de precisió per arrodoniment



- Resultat exacte v està entre 2 valors representables: v_0 i v_1
- Si l'arrodonim a v_0 l'error de precisió és $\varepsilon = |v - v_0|$
- Exemple: representar el racional $\frac{1}{10}$ en simple precisió
  - És el número amb fracció periòdica:

$$v = \underbrace{1{,}10011001100110011001100}_{\text{23 bits de fracció}}110011001100110011\ldots \times 2^{-4}$$

Error de precisió per arrodoniment



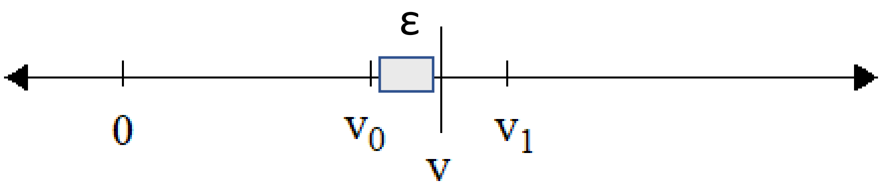
- Resultat exacte v està entre 2 valors representables: $\mathtt{v}_0$ i $\mathtt{v}_1$
- Si l'arrodonim a $\mathtt{v}_0$ l'error de precisió és $\varepsilon = |\mathtt{v} - \mathtt{v}_0|$
- Exemple: representar el racional $\frac{1}{10}$ en simple precisió
  - És el número amb fracció periòdica:

$$\mathbf{v} = \underbrace{1,10011001100110011001100}_{\text{23 bits de fracció}}\cancel{110011001100110011}\ldots \times 2^{-4}$$

  - Si l'arrodonim a $\mathtt{v}_0$ (eliminant bits)

$$\mathbf{v}_0 = 1,10011001100110011001100 \times 2^{-4}$$

Error de precisió per arrodoniment



- Resultat exacte v està entre 2 valors representables: v_0 i v_1
- Si l'arrodonim a v_0 l'error de precisió és $\varepsilon = |v - v_0|$
- Exemple: representar el racional $\frac{1}{10}$ en simple precisió
  - És el número amb fracció periòdica:

    $\mathbf{v} = \underbrace{1{,}10011001100110011001100}_{\text{23 bits de fracció}}110011001100110011\ldots \times 2^{-4}$

  - Si l'arrodonim a v_0 (eliminant bits)

    $\mathbf{v_0} = 1{,}10011001100110011001100 \times 2^{-4}$

  - Error de precisió ($\varepsilon = |v - v_0|$)

    $\boldsymbol{\varepsilon} = 0{,}00000000000000000000000110011001100110011\ldots \times 2^{-4}$

Error de precisió per arrodoniment

- Fita de l'error absolut en l'interval (v_0, v_1)

  $\varepsilon_{max} = |v_1 - v_0|$

Error de precisió per arrodoniment

- Fita de l'error absolut en l'interval (v_0, v_1)

  $\varepsilon_{max} = |v_1 – v_0|$

- Fita de l'error absolut en simple precisió
  - $v_0 = m \cdot 2^E$
  - $v_1 = (m + 2^{-23}) \cdot 2^E$

  $\varepsilon_{max} = (m + 2^{-23} – m) \cdot 2^E = 2^{E-23}$

Error de precisió per arrodoniment

- L'error relatiu pot ser més significatiu que l'absolut
  - Un error ε = 1m és petit per a la distància Terra-Sol...
    ... però inacceptable per a la longitud d'un pont

$$\eta = \frac{\varepsilon}{|v|}$$

Error de precisió per arrodoniment

- L'error relatiu pot ser més significatiu que l'absolut
  - Un error ε = 1m és petit per a la distància Terra-Sol...

    ... però inacceptable per a la longitud d'un pont

$$\eta = \frac{\varepsilon}{|v|}$$

- Fita de l'error relatiu en simple precisió

$$\eta_{max} < \frac{\varepsilon_{max}}{|v_0|} = \frac{2^{E-23}}{m \cdot 2^E} = \frac{2^{-23}}{m}$$

Error de precisió per arrodoniment

- L'error relatiu pot ser més significatiu que l'absolut
  - Un error ε = 1m és petit per a la distància Terra-Sol...
    ... però inacceptable per a la longitud d'un pont

$$\eta = \frac{\varepsilon}{|v|}$$

- Fita de l'error relatiu en simple precisió

$$\eta_{max} < \frac{\varepsilon_{max}}{|v_0|} = \frac{2^{E-23}}{m \cdot 2^E} = \frac{2^{-23}}{m}$$

  - I com que m ≥ 1,0, queda

$$\eta_{max} < 2^{-23} = 1 \text{ ULP}$$ (“Unit in the Last Place”)

Error de precisió i "Underflow"

- Hi ha un cas especial, quan un resultat és $|v| < 2^{Emin}$
  - $|v|$ pertany a l'interval $(v_0, v_1) = (0, 2^{Emin})$

Error de precisió i "Underflow"

- Hi ha un cas especial, quan un resultat és $|v| < 2^{Emin}$
  - $|v|$ pertany a l'interval $(v_0, v_1) = (0, 2^{Emin})$
  - Si l'arrodonim a $v_0=0$, l'error absolut és

    $\varepsilon = |v - 0| = v$
  - I l'error relatiu és

    $$\eta = \frac{\varepsilon}{|v|} = 1$$

Error de precisió i "Underflow"

- Hi ha un cas especial, quan un resultat és $|v| < 2^{Emin}$
  - $|v|$ pertany a l'interval $(v_0, v_1) = (0, 2^{Emin})$
  - Si l'arrodonim a $v_0=0$, l'error absolut és

    $\varepsilon = |v - 0| = v$
  - I l'error relatiu és

    $$\eta = \frac{\varepsilon}{|v|} = 1$$
- L'error és tan gran com el resultat!
  - No és fiable
  - L'estàndard determina que es produeix un "underflow"

Error de precisió i "Underflow"

- Hi ha un cas especial, quan un resultat és $|v| < 2^{Emin}$
  - $|v|$ pertany a l'interval $(v_0, v_1) = (0, 2^{Emin})$
  - Si l'arrodonim a v_0=0, l'error absolut és

    $\varepsilon = |v - 0| = v$
  - I l'error relatiu és

    $$\eta = \frac{\varepsilon}{|v|} = 1$$
- L'error és tan gran com el resultat!
  - No és fiable
  - L'estàndard determina que es produeix un "underflow"
  - El tractament és configurable: arrodonir a zero, excepció, etc.

IEEE-754: 4 modes d'arrodoniment

1. Truncament ("cap al zero") $\rightarrow$ `v = v`$_0$
   - El més simple d'implementar, sols cal eliminar els bits extra

IEEE-754: 4 modes d'arrodoniment

1. Truncament ("cap al zero") $\rightarrow$ $v = v_0$
   - El més simple d'implementar, sols cal eliminar els bits extra
   - Fita superior d'error, en simple precisió

     $\varepsilon_{max} < |v_1 - v_0|$

     $\varepsilon_{max} < 2^{E-23}$ $\eta_{max} < 2^{-23}$ = 1 ULP

IEEE-754: 4 modes d'arrodoniment

1. Truncament ("cap al zero") $\rightarrow$ $v = v_0$
   - El més simple d'implementar, sols cal eliminar els bits extra
   - Fita superior d'error, en simple precisió

     $\varepsilon_{max} < |v_1 - v_0|$

     $\varepsilon_{max} < 2^{E-23}$ $\eta_{max} < 2^{-23}$ = 1 ULP

2. Cap al $+\infty$ $\rightarrow$ $v = max(v_0, v_1)$

3. Cap al $-\infty$ $\rightarrow$ $v = min(v_0, v_1)$
   - Usats en "aritmètica d'intervals"

IEEE-754: 4 modes d'arrodoniment

1. Truncament ("cap al zero") $\rightarrow$ `v = v`$_0$
   - El més simple d'implementar, sols cal eliminar els bits extra
   - Fita superior d'error, en simple precisió

     $\varepsilon_{max} < |v_1 - v_0|$

     $\varepsilon_{max} < 2^{E-23}$ $\qquad\qquad$ $\eta_{max} < 2^{-23}$ = 1 ULP

2. Cap al +∞ $\rightarrow$ $v = \max(v_0, v_1)$

3. Cap al - ∞ $\rightarrow$ $v = \min(v_0, v_1)$
   - Usats en "aritmètica d'intervals"

4. Cap al més pròxim (o al valor "parell", si equidistant)
   - Mètode usat per defecte, perquè dona el menor error possible

IEEE-754: 4 modes d'arrodoniment

1. Truncament ("cap al zero") $\rightarrow$ $v = v_0$
   - El més simple d'implementar, sols cal eliminar els bits extra
   - Fita superior d'error, en simple precisió

     $\varepsilon_{max} < |v_1 - v_0|$

     $\varepsilon_{max} < 2^{E-23}$ $\qquad\qquad$ $\eta_{max} < 2^{-23}$ = 1 ULP

2. Cap al $+\infty$ $\rightarrow$ $v = max(v_0, v_1)$

3. Cap al $-\infty$ $\rightarrow$ $v = min(v_0, v_1)$
   - Usats en "aritmètica d'intervals"

4. Cap al més pròxim (o al valor "parell", si equidistant)
   - Mètode usat per defecte, perquè dona el menor error possible
   - Fita superior d'error: quan v és equidistant de v_0 i v_1

     $\varepsilon_{max} < |v_1 - v_0|/2$

     $\varepsilon_{max} < 2^{E-24}$ $\qquad\qquad$ $\eta_{max} < 2^{-24}$ = 0,5 ULP

IEEE-754: Arrodoniment al més pròxim

- Regla pràctica per arrodonir al més pròxim
  - Calculem el resultat amb alguns bits extra de precisió:

$$v = 1,\underbrace{\boxed{\texttt{xxxxxxx...0011}}}_{\text{bits del format}}\ \underbrace{\texttt{0000101...}}_{\text{bits extra}} \times 2^{E}$$

IEEE-754: Arrodoniment al més pròxim

- Regla pràctica per arrodonir al més pròxim
  - Calculem el resultat amb alguns bits extra de precisió
  - Examinem el primer bit extra de la mantissa en diversos casos:

    a) El bit és **0**: arrodonim a **l'anterior** (v_0)

$$v = 1{,}xxxxxx...0011\ \boxed{0}000101...$$

$$\mathbf{v_0 = 1{,}xxxxxx...0011}$$

IEEE-754: Arrodoniment al més pròxim

- Regla pràctica per arrodonir al més pròxim
  - Calculem el resultat amb alguns bits extra de precisió
  - Examinem el primer bit extra de la mantissa en diversos casos:

a) El bit és **0**: arrodonim a **l'anterior** (v_0)

```
v  = 1,xxxxxx...0011 0000101...
v0 = 1,xxxxxx...0011
```

b) El bit és **1**, i la resta NO són tots zeros: arrodonim al **següent** (v_1)

```
v  = 1,xxxxxx...0011 1010001...
   + 0,000000...0001
----------------------
v1 = 1,xxxxxx...0100
```

IEEE-754: Arrodoniment al més pròxim

- Regla pràctica per arrodonir al més pròxim
  - Calculem el resultat amb alguns bits extra de precisió
  - Examinem el primer bit extra de la mantissa en diversos casos:

a) El bit és **0**: arrodonim a **l'anterior** (v_0)

```
v  = 1,xxxxxx...0011 0000101...
v0 = 1,xxxxxx...0011
```

b) El bit és **1**, i la resta NO són tots zeros: arrodonim al **següent** (v_1)

```
v  = 1,xxxxxx...0011 1010001...
   + 0,000000...0001
v1 = 1,xxxxxx...0100
```

c) El bit és **1**, i la resta TOTS zeros

```
v  = 1,xxxxxx...0011 1000000...
```

v és **equidistant** de v_0 i v_1

```
v0 = 1,xxxxxx...0011
v1 = 1,xxxxxx...0100
```

IEEE-754: Arrodoniment al més pròxim

- Regla pràctica per arrodonir al més pròxim
  - Calculem el resultat amb alguns bits extra de precisió
  - Examinem el primer bit extra de la mantissa en diversos casos:

a) El bit és **0**: arrodonim a **l'anterior** (v_0)

```
v  = 1,xxxxxx...0011 0000101...
v0 = 1,xxxxxx...0011
```

b) El bit és **1**, i la resta NO són tots zeros: arrodonim al **següent** (v_1)

```
v  = 1,xxxxxx...0011 1010001...
   + 0,000000...0001
v1 = 1,xxxxxx...0100
```

c) El bit és **1**, i la resta TOTS zeros

```
v  = 1,xxxxxx...0011 1000000...
```

v és **equidistant** de v_0 i v_1: arrodonim al que sigui **parell**

```
v0 = 1,xxxxxx...0011
v1 = 1,xxxxxx...0100
```

Aritmètica de coma flotant

- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

IEE-754: Codificacions especials

- Es reserven dos exponents per a casos especials
  - E = 000 . . . 0, i E = 111 . . . 1
- Per tant
  - E_{min} = 000 . . . 01 i E_{max} = 111 . . . 10
  - El rang d'exponents és E ∈ [-126,127] o [-1022, 1023]
- Casos especials
  - Zero
  - Infinit
  - Not a Number
  - Denormals

IEE-754: Codificacions especials

- Zero
  - No hi ha cap combinació de Fracció i Exponent que doni zero!

$$(-1)^s \cdot (1 + 0{,}F) \cdot 2^E = 0 \qquad\qquad ???$$

IEE-754: Codificacions especials

- Zero
  - No hi ha cap combinació de Fracció i Exponent que doni zero!

$$(-1)^s \cdot (1 + 0{,}F) \cdot 2^E = 0 \qquad ???$$

  - Codificació especial del **zero**

$$E = 000\,.\,.\,.\,0 \qquad\qquad F = 000\,.\,.\,.\,0$$

IEE-754: Codificacions especials

- Zero
  - No hi ha cap combinació de Fracció i Exponent que doni zero!

$$(-1)^s \cdot (1 + 0{,}F) \cdot 2^E = 0 \qquad ???$$

  - Codificació especial del **zero**

$$E = 000\ .\ .\ .\ 0 \qquad\qquad F = 000\ .\ .\ .\ 0$$

  - De fet, amb el signe, tenim 2 codificacions del zero: +0 i -0
  - En simple precisió:

s 00000000 00000000000000000000000

IEE-754: Codificacions especials

- Infinit ("Inf")
  - Segueix algunes regles bàsiques d'operació. Per exemple

    $\frac{1}{0} = +\infty$, $\quad \frac{1}{\infty} = 0$, $\quad x + \infty = \infty$, etc.

IEE-754: Codificacions especials

- Infinit ("Inf")
  - Segueix algunes regles bàsiques d'operació. Per exemple

    $\frac{1}{0} = +\infty,$ $\qquad \frac{1}{\infty} = 0,$ $\qquad x + \infty = \infty,$ etc.
  - Concepte útil que permet evitar overflows en algunes expressions

    $$y = \frac{1}{1 + \frac{100}{x}}$$

    Normalment, $\frac{100}{x}$ causaria overflow per a $x \to 0$

IEE-754: Codificacions especials

- Infinit ("Inf")
  - Segueix algunes regles bàsiques d'operació. Per exemple

    $\frac{1}{0} = +\infty$, $\frac{1}{\infty} = 0$, $x + \infty = \infty$, etc.
  - Concepte útil que permet evitar overflows en algunes expressions

$$y = \frac{1}{1 + \frac{100}{x}}$$

Normalment, $\frac{100}{x}$ causaria overflow per a $x \to 0$

Si usem l'infinit, $\frac{100}{x} = \infty$ i resulta $y = 0$

IEE-754: Codificacions especials

- Infinit ("Inf")
  - Segueix algunes regles bàsiques d'operació. Per exemple

    $\frac{1}{0} = +\infty$, $\frac{1}{\infty} = 0$, $x + \infty = \infty$, etc.
  - Concepte útil que permet evitar overflows en algunes expressions

$$y = \frac{1}{1 + \frac{100}{x}}$$

    Normalment, $\frac{100}{x}$ causaria overflow per a $x \to 0$

    Si usem l'infinit, $\frac{100}{x} = \infty$ i resulta $y = 0$
  - Codificació especial de **Inf** (en realitat tenim +Inf i –Inf)

    E = 111 . . . 1 F = 000 . . . 0

IEE-754: Codificacions especials

- Infinit ("Inf")
  - Segueix algunes regles bàsiques d'operació. Per exemple

    $\frac{1}{0} = +\infty,$ $\frac{1}{\infty} = 0,$ $x + \infty = \infty,$ etc.
  - Concepte útil que permet evitar overflows en algunes expressions

    $$y = \frac{1}{1 + \frac{100}{x}}$$

    Normalment, $\frac{100}{x}$ causaria overflow per a $x \to 0$

    Si usem l'infinit, $\frac{100}{x} = \infty$ i resulta $y = 0$
  - Codificació especial de **Inf** (en realitat tenim +Inf i –Inf)

    E = 111 . . . 1 F = 000 . . . 0
  - En simple precisió:

    s 11111111 00000000000000000000000

IEE-754: Codificacions especials

- Not a Number ("NaN")
  - Representa *resultat invàlid,* en algunes operacions:

$$\sqrt{-1} = \text{NaN} \qquad \log(-1) = \text{NaN}$$

$$\infty - \infty = \text{NaN} \qquad 0 \times \infty = \text{NaN}$$

$$\frac{\infty}{\infty} = \text{NaN}$$

  - El programa pot comprovar si un resultat és vàlid o no

IEE-754: Codificacions especials

- Not a Number ("NaN")
  - Representa *resultat invàlid,* en algunes operacions:

$$\sqrt{-1} = \text{NaN} \qquad \log(-1) = \text{NaN}$$

$$\infty - \infty = \text{NaN} \qquad 0 \times \infty = \text{NaN}$$

$$\frac{\infty}{\infty} = \text{NaN}$$

  - El programa pot comprovar si un resultat és vàlid o no
  - En general, qualsevol operació amb un NaN dona resultat = NaN

    → En una cadena de càlculs, podem diferir la comprovació fins al final

IEE-754: Codificacions especials

- Not a Number ("NaN")
  - Representa *resultat invàlid,* en algunes operacions:

$$\sqrt{-1} = \text{NaN} \qquad \log(-1) = \text{NaN}$$

$$\infty - \infty = \text{NaN} \qquad 0 \times \infty = \text{NaN}$$

$$\frac{\infty}{\infty} = \text{NaN}$$

  - El programa pot comprovar si un resultat és vàlid o no
  - En general, qualsevol operació amb un NaN dona resultat = NaN

    → En una cadena de càlculs, podem diferir la comprovació fins al final
  - Codificació de NaN

E = 111 . . . 1 F ≠ 000 . . . 0

IEE-754: Codificacions especials

- Not a Number ("NaN")
  - Representa *resultat invàlid,* en algunes operacions:

$$\sqrt{-1} = \text{NaN} \qquad \log(-1) = \text{NaN}$$

$$\infty - \infty = \text{NaN} \qquad 0 \times \infty = \text{NaN}$$

$$\frac{\infty}{\infty} = \text{NaN}$$

  - El programa pot comprovar si un resultat és vàlid o no
  - En general, qualsevol operació amb un NaN dona resultat = NaN

    → En una cadena de càlculs, podem diferir la comprovació fins al final
  - Codificació de NaN

    E = 111 . . . 1 F ≠ 000 . . . 0
  - En simple precisió:

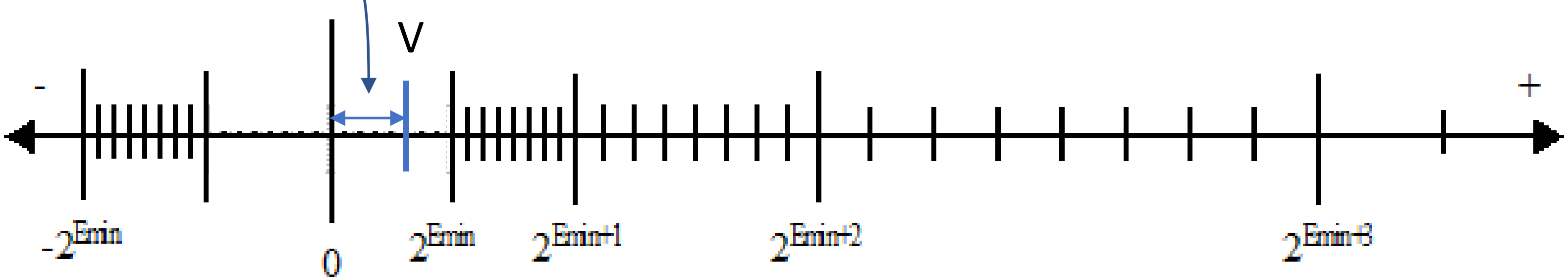
    s 11111111 xxxxxxxxxxxxxxxxxxxxxxx (algun x ≠ 0)

IEE-754: Codificacions especials

- Denormals
  - Un resultat molt petit, del tipus $|v| < 2^{Emin}$ no és fiable, ja que si l'arrodonim a 0, l'error de precisió és enorme

$\varepsilon = |v - 0| = |v|$

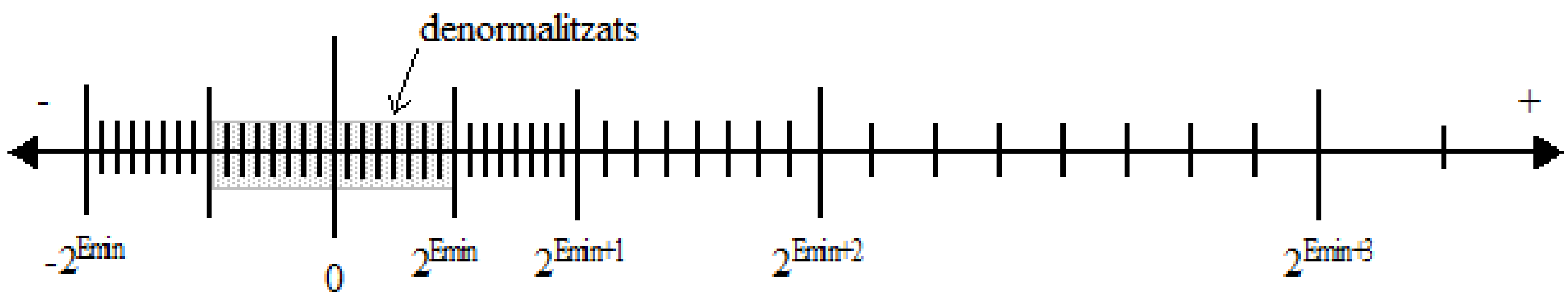
$$\eta = \frac{|v-0|}{|v|} = 1$$



IEE-754: Codificacions especials

- Denormals
  - Un resultat molt petit, del tipus $|v| < 2^{Emin}$ no és fiable, ja que si l'arrodonim a 0, l'error de precisió és enorme

$$\varepsilon = |v - 0| = |v| \qquad \eta = \frac{|v-0|}{|v|} = 1$$



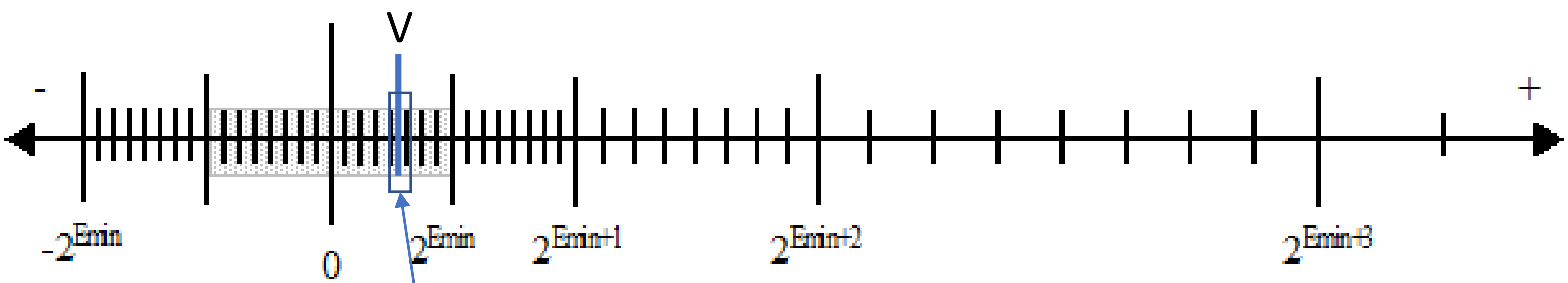
  - En aquests casos podem millorar la precisió admetent nombres no-normalitzats o *denormals* en el rang $(0, 2^{Emin})$, de la forma:

$$v = 0{,}xxx\ \ldots\ x * 2^{Emin}$$

IEE-754: Codificacions especials

- Denormals
  - Un resultat molt petit, del tipus $|v| < 2^{Emin}$ no és fiable, ja que si l'arrodonim a 0, l'error de precisió és enorme

$$\varepsilon = |v - 0| = |v| \qquad \eta = \frac{|v-0|}{|v|} = 1$$



  - En aquests casos podem millorar la precisió admetent nombres no-normalitzats o *denormals* en el rang $(0, 2^{Emin})$, de la forma:

$$v = 0{,}xxx\ \ldots\ x * 2^{Emin}$$

  - En simple precisió, la fita superior d'error absolut és

$$\varepsilon_{max} = |v_1 - v_0| = 2^{Emin-23}$$

IEE-754: Codificacions especials

- Denormals
  - Codificació

$$E = 000\ldots0 \qquad\qquad F \neq 000\ldots0$$

IEE-754: Codificacions especials

- Denormals
  - Codificació

    $E = 000 \dots 0$ $\qquad$ $F \neq 000 \dots 0$

  - En simple precisió:

    s 00000000 xxxxxxxxxxxxxxxxxxxxxxx (algun $x \neq 0$)

IEE-754: Codificacions especials

- Denormals
  - Codificació

    $E = 000\,.\,.\,.\,0 \qquad\qquad F \neq 000\,.\,.\,.\,0$

  - En simple precisió:

    s 00000000 xxxxxxxxxxxxxxxxxxxxxxx (algun $x \neq 0$)

  - Alerta! Si els hem d'operar, tinguem en compte que
    - El bit ocult implícit és 0
    - L'exponent implícit és E_{min}

IEE-754: resum dels formats

<table>
<tr><th rowspan="2">Mantissa (F)</th><th></th><th colspan="3">Exponent (E)</th></tr>
<tr><th></th><th>Tot 0</th><th>Altres</th><th>Tot 1</th></tr>
<tr><td></td><td>Tot 0</td><td>Zero</td><td rowspan="2">Normalitzat</td><td>Infinit</td></tr>
<tr><td></td><td>Altres</td><td>Denormal</td><td>NaN</td></tr>
</table>

Aritmètica de coma flotant

- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

Exemple. Representar `v=-1029,68`

1. Convertir la part entera, per divisions successives

   `1029 = 10000000101` (té `11` bits)

Exemple. Representar `v=-1029,68`

1. Convertir la part entera, per divisions successives

   `1029 = 10000000101` (té `11` bits)

2. Convertir la fracció, per multiplicacions successives

   ```
   0,68 × 2 = 1,36   → 1   (1er bit de fracció)
   0,36 × 2 = 0,72   → 0   (2on bit de fracció)
   ...
   ```
   etc.

   - Quants bits de fracció calculem?

Exemple. Representar `v=-1029,68`

1. Convertir la part entera, per divisions successives

   `1029 = 10000000101`   (té `11` bits)

2. Convertir la fracció, per multiplicacions successives

   `0,68 × 2 = `**`1`**`,36  → 1  (1er bit de fracció)`
   `0,36 × 2 = `**`0`**`,72  → 0  (2on bit de fracció)`
   `...` etc.

   - Quants bits de fracció calculem?

     → 13 bits, per totalitzar 24 bits de mantissa

     `0,68 = 0,`**`1010111000010`**   (té `13` bits)

Exemple. Representar `v=-1029,68`

1. Convertir la part entera, per divisions successives

   `1029 = 10000000101` (té `11` bits)

2. Convertir la fracció, per multiplicacions successives

   `0,68 × 2 = `**`1`**`,36 → 1 (1er bit de fracció)`
   `0,36 × 2 = `**`0`**`,72 → 0 (2on bit de fracció)`
   `...` etc.

   - Quants bits de fracció calculem?

     → 13 bits, per totalitzar 24 bits de mantissa

     `0,68 = 0,`**`1010111000010`** (té `13` bits)

     → i alguns "bits extra", per decidir com arrodonir al més pròxim

     `0,68 = 0,1010111000010`**`10001`**`...` (té `18` bits)

Exemple. Representar `v=-1029,68`

1. Convertir la part entera, per divisions successives

   `1029 = 10000000101` (té `11` bits)

2. Convertir la fracció, per multiplicacions successives

   ```
   0,68 × 2 = 1,36   → 1   (1er bit de fracció)
   0,36 × 2 = 0,72   → 0   (2on bit de fracció)
   ... etc.
   ```

   - Quants bits de fracció calculem?

     → 13 bits, per totalitzar 24 bits de mantissa

     `0,68 = 0,1010111000010` (té `13` bits)

     → i alguns "bits extra", per decidir com arrodonir al més pròxim

     `0,68 = 0,101011100001010001...` (té `18` bits)

3. Ajuntar part entera i fracció (`11+18=29` bits)

   `1029,68 = 10000000101,101011100001010001...`

Exemple. Representar `v=-1029,68`

1. Convertir la part entera, per divisions successives

   `1029 = 10000000101` (té `11` bits)

2. Convertir la fracció, per multiplicacions successives

   `0,68 × 2 = 1,36 → 1 (1er bit de fracció)`
   `0,36 × 2 = 0,72 → 0 (2on bit de fracció)`
   `...` etc.

   - Quants bits de fracció calculem?

     → 13 bits, per totalitzar 24 bits de mantissa

     `0,68 = 0,`**`1010111000010`** (té `13` bits)

     → i alguns "bits extra", per decidir com arrodonir al més pròxim

     `0,68 = 0,1010111000010`**`10001`**`...` (té `18` bits)

3. Ajuntar part entera i fracció (`11+18=29` bits)

   `1029,68 = 10000000101,1010111000010`**`10001`**`...`

4. Normalitzar (moure la coma 10 llocs i ajustar l'exponent)

   `1029,68 = 1,00000001011010111000010`**`10001`**`...` $\times 2^{10}$

Exemple (cont.)

5. Arrodonir (al més pròxim), usant els “bits extra”

   $1029,68 = 1,00000001011010111000010\boxed{\mathbf{10001...}} \times 2^{10}$

   $\rightarrow$ Arrodonim “al següent”

Exemple (cont.)

5. Arrodonir (al més pròxim), usant els “bits extra”

   1029,68 = 1,00000001011010111000010**10001...** × 2^{10}

   → Arrodonim “al següent”

   1029,68 = 1,00000001011010111000010 × 2^{10}

   \+ **1**

   = 1,0000000101101011100001**1** × 2^{10}

Exemple (cont.)

5. Arrodonir (al més pròxim), usant els “bits extra”

   `1029,68 = 1,00000001011010111000010`**`10001...`**` × 2`^{10}

   → Arrodonim “al següent”

   `1029,68 = 1,00000001011010111000010 × 2`^{10}

   `+ `**`1`**

   `= 1,0000000101101011100001`**`1`**` × 2`^{10}

6. Codificar l’exponent en excés a `127`

   `E = 10 + 127 = 137 = 10001001`

Exemple (cont.)

5. Arrodonir (al més pròxim), usant els "bits extra"

$$1029{,}68 = 1{,}00000001011010111000010\mathbf{10001...} \times 2^{10}$$

→ Arrodonim "al següent"

$$1029{,}68 = 1{,}00000001011010111000010 \times 2^{10}$$

$$+\ \mathbf{1}$$

$$= 1{,}0000000101101011100001\mathbf{1} \times 2^{10}$$

6. Codificar l'exponent en excés a `127`

$$E = 10 + 127 = 137 = 10001001$$

7. Ajuntar signe, exponent i fracció
   - La part entera (1) no s'escriu: és el bit ocult!

Exemple (cont.)

5. Arrodonir (al més pròxim), usant els “bits extra”

   `1029,68 = 1,00000001011010111000010`**`10001...`**` × 2`^{10}

   → Arrodonim “al següent”

   `1029,68 = 1,00000001011010111000010 × 2`^{10}

   `+ `**`1`**

   `= 1,0000000101101011100001`**`1`**` × 2`^{10}

6. Codificar l’exponent en excés a `127`

   `E = 10 + 127 = 137 = 10001001`

7. Ajuntar signe, exponent i fracció
   - La part entera (1) no s’escriu: és el bit ocult!

   `-1029,68 = 1 10001001 00000001011010111000011`

Exemple (cont.)

5. Arrodonir (al més pròxim), usant els "bits extra"

   `1029,68 = 1,00000001011010111000010`**`10001...`** `× 2`^{10}

   → Arrodonim "al següent"

   `1029,68 = 1,00000001011010111000010 × 2`^{10}

   `+ `**`1`**

   `= 1,0000000101101011100001`**`1`**` × 2`^{10}

6. Codificar l'exponent en excés a `127`

   `E = 10 + 127 = 137 = 10001001`

7. Ajuntar signe, exponent i fracció

   - La part entera (1) no s'escriu: és el bit ocult!

   `-1029,68 = 1 10001001 00000001011010111000011`

8. Expressar en hexadecimal

   `-1029,68 = `**`0xC480B5C3`**

Exemple (cont.)

- Calcular l'error de precisió ($\varepsilon = |\mathtt{v}-\mathtt{v}_0|$)
  - Restant el valor arrodonit i el valor "exacte"

$$
\begin{array}{rlr}
\varepsilon = & (\ 1{,}00000001011010111000011 & \times\ 2^{10} \\
 & -\ 1{,}00000001011010111000010\mathbf{10001}\ldots & \times\ 2^{10}) \\
\hline
= & 0{,}0000000000000000000000001111 & \times\ 2^{10}
\end{array}
$$

Exemple (cont.)

- Calcular l'error de precisió ($\varepsilon = |v-v_0|$)
  - Restant el valor arrodonit i el valor "exacte"

$$
\begin{aligned}
\varepsilon = \ & (\ \mathtt{1,00000001011010111000011} && \times 2^{10} \\
& -\ \mathtt{1,00000001011010111000010}\mathbf{\color{red}{10001}}\mathtt{...} && \times 2^{10}) \\
= \ & \ \ \mathtt{0,0000000000000000000000001111} && \times 2^{10}
\end{aligned}
$$

  - Normalitzem: movem la coma 25 posicions a la dreta …

$$\varepsilon = \mathtt{0000000000000000000000001,111} \times 2^{10-25}$$

$$= \mathbf{1,111 \times 2^{-15}}$$

Exemple (cont.)

- Calcular l'error de precisió ($\varepsilon = |v-v_0|$)
  - Restant el valor arrodonit i el valor "exacte"

$$\varepsilon = (\ 1{,}00000001011010111000011 \times 2^{10}$$
$$-\ 1{,}00000001011010111000010\mathbf{10001}... \times 2^{10})$$
$$= 0{,}0000000000000000000000001111 \times 2^{10}$$

  - Normalitzem: movem la coma 25 posicions a la dreta …

$$\varepsilon = 0000000000000000000000001{,}111 \times 2^{10-25}$$
$$= \mathbf{1{,}111 \times 2^{-15}}$$

  - Convertim a decimal (no es demanarà sense calculadora) …

$$\varepsilon = 1{,}875 \times 2^{-15}$$
$$= 1{,}875 / 2^{15}$$
$$= 1{,}875/32768$$
$$= \mathbf{5{,}722 \times 10^{-5}}$$

Exemple: convertir `v=0x45814140` a base `10`

1. L’escrivim en binari
   `v = 0100 0101 1000 0001 0100 0001 0100 0000`

Exemple: convertir `v=0x45814140` a base `10`

1. L'escrivim en binari
   `v = 0100 0101 1000 0001 0100 0001 0100 0000`
2. Identifiquem els 3 camps: signe, exponent, fracció
   `v = 0 10001011 00000010100000101000000`

Exemple: convertir `v=0x45814140` a base `10`

1. L'escrivim en binari
   `v = 0100 0101 1000 0001 0100 0001 0100 0000`
2. Identifiquem els 3 camps: signe, exponent, fracció
   `v = 0 10001011 00000010100000101000000`
3. Convertim l'exponent a decimal i li restem l'excés `127`
   `10001011 = 139`
   `E = 139 - 127 = 12`

Exemple: convertir `v=0x45814140` a base `10`

1. L'escrivim en binari
   `v = 0100 0101 1000 0001 0100 0001 0100 0000`
2. Identifiquem els 3 camps: signe, exponent, fracció
   `v = 0 10001011 00000010100000101000000`
3. Convertim l'exponent a decimal i li restem l'excés `127`
   `10001011 = 139`
   `E = 139 - 127 = 12`
4. En coma flotant: signe, bit ocult, fracció i exponent
   $v = +\mathbf{1},00000010100000101000000 \times 2^{12}$

Exemple: convertir `v=0x45814140` a base `10`

1. L'escrivim en binari
   `v = 0100 0101 1000 0001 0100 0001 0100 0000`
2. Identifiquem els 3 camps: signe, exponent, fracció
   `v = 0 10001011 00000010100000101000000`
3. Convertim l'exponent a decimal i li restem l'excés `127`
   `10001011 = 139`
   `E = 139 - 127 = 12`
4. En coma flotant: signe, bit ocult, fracció i exponent
   v = +**1**,00000010100000101000000 × 2^{12}
5. En coma fixa: moure la coma `12` llocs a la dreta i eliminar zeros finals
   v = +**1**000000101000,00101~~000000~~

Exemple: convertir `v=0x45814140` a base `10`

6. Convertir la part entera a base 10 (suma ponderada)

$$\mathbf{1}000000\mathbf{1}0\mathbf{1}000 = \mathbf{1}\cdot 2^{12} + \mathbf{1}\cdot 2^{5} + \mathbf{1}\cdot 2^{3} = 4136$$

Exemple: convertir `v=0x45814140` a base `10`

6. Convertir la part entera a base 10 (suma ponderada)

   $1000000101000 = 1 \cdot 2^{12} + 1 \cdot 2^{5} + 1 \cdot 2^{3} = 4136$

7. Convertir la fracció a base 10 (movent la coma a dreta)

   $0{,}00101 = 101 \times 2^{-5} = 5/32 = 0{,}15625$

Exemple: convertir `v=0x45814140` a base `10`

6. Convertir la part entera a base 10 (suma ponderada)

   $1000000101000 = 1 \cdot 2^{12} + 1 \cdot 2^{5} + 1 \cdot 2^{3} = 4136$

7. Convertir la fracció a base 10 (movent la coma a dreta)

   $0{,}00101 = 101 \times 2^{-5} = 5/32 = 0{,}15625$

8. Ajuntar part entera i fracció

v = 4136,15625

Aritmètica de coma flotant

- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

Suma (resta) en coma flotant

- Suposem un cas senzill que coneixem bé (base `10`):
  - Format: mantissa normalitzada amb 4 dígits: $x,xxx \times 10^{xx}$
  - Sumar: $9,999 \times 10^{1} + 1,680 \times 10^{-1}$

Suma (resta) en coma flotant

- Suposem un cas senzill que coneixem bé (base 10):
  - Format: mantissa normalitzada amb 4 dígits: $x,xxx \times 10^{xx}$
  - Sumar: $9,999 \times 10^{1} + 1,680 \times 10^{-1}$
  - Així?

$$
\begin{array}{rrl}
 & 9,999 & \times\ 10^{1} \\
+ & 1,680 & \times\ 10^{-1} \\
\hline
= & 11,679 & \times\ 10^{?}
\end{array}
$$

Suma (resta) en coma flotant

- Suposem un cas senzill que coneixem bé (base 10):
  - Format: mantissa normalitzada amb 4 dígits: $x,xxx \times 10^{xx}$
  - Sumar: $9,999 \times 10^{1} + 1,680 \times 10^{-1}$
  - Així? No!

$$
\begin{array}{rrl}
 & 9,999 & \times 10^{1} \\
+ & 1,680 & \times 10^{-1} \\
\hline
= & 11,679 & \times 10^{?}
\end{array}
$$

Suma (resta) en coma flotant

- Suposem un cas senzill que coneixem bé (base `10`):
  - Format: mantissa normalitzada amb 4 dígits: $x,xxx \times 10^{xx}$
  - Sumar: $9,999 \times 10^{1} + 1,680 \times 10^{-1}$
  - Així? No!

$$\cancel{\begin{array}{rrl} & 9,999 & \times 10^{1} \\ + & 1,680 & \times 10^{-1} \\ \hline = & 11,679 & \times 10^{?} \end{array}}$$

  - Igualar exponents (al major = 1), alinear mantisses i sumar:

$$\begin{array}{rrl} & 9,999 & \times 10^{1} \\ + & \mathbf{0,0}1680 & \times 10^{\mathbf{1}} \\ \hline = & 10,01580 & \times 10^{1} \end{array}$$

Suma (resta) en coma flotant

- Suposem un cas senzill que coneixem bé (base `10`):
  - Format: mantissa normalitzada amb 4 dígits: $x{,}xxx \times 10^{xx}$
  - Sumar: $9{,}999 \times 10^{1} + 1{,}680 \times 10^{-1}$
  - Així? No!

$$\cancel{\begin{array}{rrl} & 9{,}999 & \times 10^{1} \\ + & 1{,}680 & \times 10^{-1} \\ \hline = & 11{,}679 & \times 10^{?} \end{array}}$$

  - Igualar exponents (al major = 1), alinear mantisses i sumar:

$$\begin{array}{rrl} & 9{,}999 & \times 10^{1} \\ + & \mathbf{0{,}0}1680 & \times 10^{\mathbf{1}} \\ \hline = & 10{,}01580 & \times 10^{1} \end{array}$$

  - Normalitzar:

$$1{,}001580 \times 10^{\mathbf{2}}$$

Suma (resta) en coma flotant

- Suposem un cas senzill que coneixem bé (base `10`):
  - Format: mantissa normalitzada amb 4 dígits: $x,xxx \times 10^{xx}$
  - Sumar: $9,999 \times 10^{1} + 1,680 \times 10^{-1}$
  - Així? No!

$$\cancel{\begin{array}{rrl} & 9,999 & \times\ 10^{1} \\ + & 1,680 & \times\ 10^{-1} \\ \hline = & 11,679 & \times\ 10^{?} \end{array}}$$

  - Igualar exponents (al major = 1), alinear mantisses i sumar:

$$\begin{array}{rrl} & 9,999 & \times\ 10^{1} \\ + & \mathbf{0,0}1680 & \times\ 10^{\mathbf{1}} \\ \hline = & 10,01580 & \times\ 10^{1} \end{array}$$

  - Normalitzar:

$$1,001580 \times 10^{\mathbf{2}}$$

  - Arrodonir a 4 dígits:

$$1,001\boxed{\mathbf{580}} \times 10^{2}$$

$$1,00\mathbf{2} \times 10^{2}$$

Suma (resta) en coma flotant

1. Igualar els exponents, al major dels dos
   - I alinear les mantisses, desplaçant a l'esquerra la coma d'aquella amb menor exponent

Suma (resta) en coma flotant

1. Igualar els exponents, al major dels dos
   - I alinear les mantisses, desplaçant a l'esquerra la coma d'aquella amb menor exponent
2. Sumar les magnituds (valors absoluts)
   - Signes iguals $\rightarrow$ sumar magnituds
   - Signes diferents $\rightarrow$ restar la magnitud major menys la menor, i assignar al resultat el signe de la major

Suma (resta) en coma flotant

1. Igualar els exponents, al major dels dos
   - I alinear les mantisses, desplaçant a l'esquerra la coma d'aquella amb menor exponent
2. Sumar les magnituds (valors absoluts)
   - Signes iguals $\rightarrow$ sumar magnituds
   - Signes diferents $\rightarrow$ restar la magnitud major menys la menor, i assignar al resultat el signe de la major
3. Normalitzar el resultat
   - Movent la coma per obtenir un dígit no-nul a la part entera

Suma (resta) en coma flotant

1. Igualar els exponents, al major dels dos
   - I alinear les mantisses, desplaçant a l'esquerra la coma d'aquella amb menor exponent
2. Sumar les magnituds (valors absoluts)
   - Signes iguals → sumar magnituds
   - Signes diferents → restar la magnitud major menys la menor, i assignar al resultat el signe de la major
3. Normalitzar el resultat
   - Movent la coma per obtenir un dígit no-nul a la part entera
4. Arrodonir la mantissa
   - Al valor representable més pròxim
   - Pot requerir haver de normalitzar i arrodonir per segon cop

Suma (resta) en coma flotant

1. Igualar els exponents, al major dels dos
   - I alinear les mantisses, desplaçant a l'esquerra la coma d'aquella amb menor exponent
2. Sumar les magnituds (valors absoluts)
   - Signes iguals → sumar magnituds
   - Signes diferents → restar la magnitud major menys la menor, i assignar al resultat el signe de la major
3. Normalitzar el resultat
   - Movent la coma per obtenir un dígit no-nul a la part entera
4. Arrodonir la mantissa
   - Al valor representable més pròxim
   - Pot requerir haver de normalitzar i arrodonir per segon cop
5. Codificar el resultat
   - Signe, exponent (en excés) i mantissa (sense el bit ocult)

Exemple: sumar z = x + y

Suposem `x=0x3F40000D`, `y=0xC0800004`

1. Els escrivim en binari
```
x = 0011 1111 0100 0000 0000 0000 0000 1101
y = 1100 0000 1000 0000 0000 0000 0000 0100
```

Exemple: sumar z = x + y

Suposem `x=0x3F40000D`, `y=0xC0800004`

1. Els escrivim en binari
   ```
   x = 0011 1111 0100 0000 0000 0000 0000 1101
   y = 1100 0000 1000 0000 0000 0000 0000 0100
   ```
2. Identifiquem els camps: signe, exponent, fracció
   ```
   x = 0 01111110 10000000000000000001101
   y = 1 10000001 00000000000000000000100
   ```

Exemple: sumar z = x + y

Suposem `x=0x3F40000D`, `y=0xC0800004`

1. Els escrivim en binari
```
x = 0011 1111 0100 0000 0000 0000 0000 1101
y = 1100 0000 1000 0000 0000 0000 0000 0100
```
2. Identifiquem els camps: signe, exponent, fracció
```
x = 0 01111110 10000000000000000001101
y = 1 10000001 00000000000000000000100
```
3. Convertim exponents a base 10 (restant l'excés)

   `01111110 = 126` $\rightarrow$ E = `126-127 = -1`

   `10000001 = 129` $\rightarrow$ E = `129-127 = 2`

Exemple: sumar z = x + y

Suposem `x=0x3F40000D`, `y=0xC0800004`

1. Els escrivim en binari
   `x = 0011 1111 0100 0000 0000 0000 0000 1101`
   `y = 1100 0000 1000 0000 0000 0000 0000 0100`
2. Identifiquem els camps: signe, exponent, fracció
   `x = 0 01111110 10000000000000000001101`
   `y = 1 10000001 00000000000000000000100`
3. Convertim exponents a base 10 (restant l'excés)

   `01111110 = 126` → E = `126-127 = -1`

   `10000001 = 129` → E = `129-127 = 2`
4. Expressem *x* i *y* en coma flotant, afegint el bit ocult i el signe
   $x = \mathbf{+1,}10000000000000000001101 \times 2^{-1}$
   $y = \mathbf{-1,}00000000000000000000100 \times 2^{2}$

Exemple: sumar z = x + y

Suposem `x=0x3F40000D`, `y=0xC0800004`

1. Els escrivim en binari
   ```
   x = 0011 1111 0100 0000 0000 0000 0000 1101
   y = 1100 0000 1000 0000 0000 0000 0000 0100
   ```
2. Identifiquem els camps: signe, exponent, fracció
   ```
   x = 0 01111110 10000000000000000001101
   y = 1 10000001 00000000000000000000100
   ```
3. Convertim exponents a base 10 (restant l'excés)

   $01111110 = 126 \rightarrow E = 126-127 = -1$

   $10000001 = 129 \rightarrow E = 129-127 = 2$

4. Expressem x i y en coma flotant, afegint el bit ocult i el signe

   $x = \mathbf{+1,}10000000000000000001101 \times 2^{-1}$

   $y = \mathbf{-1,}00000000000000000000100 \times 2^{2}$

5. Igualem exponents al major (=2), movent la coma 3 bits a l'esquerra

   $x = \mathbf{+}0,\underbrace{00\mathbf{1}10000000000000000001}_{\text{Fracció de 23 bits}}\underbrace{\boxed{\mathbf{101}}}_{\text{bits extra}} \times 2^{\mathbf{2}}$

Exemple: sumar z = x + y

```
x   = +0,00110000000000000000001101 × 2^2
y   = -1,00000000000000000000100    × 2^2
```

4. Signes diferents → restar la magnitud major (y) menys la menor (x)...

```
 |y| =  1,00000000000000000000100    × 2^2
-|x| =  0,00110000000000000000001101 × 2^2
-----------------------------------------
 |z| =  0,11010000000000000000010011 × 2^2
```

Exemple: sumar z = x + y

$x = +0{,}00\mathbf{1}1000000000000000000001\mathbf{101} \times 2^{\mathbf{2}}$

$y = \mathbf{-1}{,}00000000000000000000100 \times 2^2$

4. Signes diferents → restar la magnitud major (y) menys la menor (x)...

$$
\begin{array}{rl}
|y| = & 1{,}00000000000000000000100 \times 2^2 \\
-|x| = & 0{,}0011000000000000000000001\mathbf{101} \times 2^{\mathbf{2}} \\
\hline
|z| = & 0{,}11010000000000000000010\mathbf{011} \times 2^2
\end{array}
$$

...i assignar el signe del que té major valor absolut: és *y* (negatiu)

$z = -0{,}11010000000000000000010\mathbf{011} \times 2^2$

Exemple: sumar z = x + y

$$z = -0{,}1101000000000000000001001\mathbf{1}\mathbf{1} \times 2^{2}$$

5. Normalitzar mantissa (desplaçant els bits a l’esquerra)

$$|z| = 1{,}1010000000000000000010011 \times 2^{1}$$

Exemple: sumar z = x + y

$$z = -0{,}11010000000000000000010\mathbf{011} \times 2^{2}$$

5. Normalitzar mantissa (desplaçant els bits a l'esquerra)

$$|z| = 1{,}10100000000000000000100\mathbf{11} \times 2^{1}$$

6. Arrodonir mantissa (amunt)

$$|z| = 1{,}10100000000000000000100 \times 2^{1}$$

$$+\ 1$$

$$|z| = 1{,}10100000000000000000101 \times 2^{1}$$

Exemple: sumar z = x + y

$$z = -0{,}11010000000000000000010\mathbf{\color{red}{011}} \times 2^{2}$$

5. Normalitzar mantissa (desplaçant els bits a l'esquerra)

$$|z| = 1{,}10100000000000000000100\mathbf{\color{red}{11}} \times 2^{1}$$

6. Arrodonir mantissa (amunt)

$$|z| = 1{,}10100000000000000000100 \times 2^{1}$$

$$+\ 1$$

$$|z| = 1{,}\color{blue}{10100000000000000000101} \times 2^{1}$$

7. Codificar exponent (en excés)

$$E \rightarrow 1+127 = 128 = \color{red}{10000000}$$

Exemple: sumar z = x + y

$z \quad = -0{,}11010000000000000000010\mathbf{\color{red}{011}} \times 2^2$

5. Normalitzar mantissa (desplaçant els bits a l'esquerra)

$|z| = 1{,}10100000000000000000100\mathbf{\color{red}{11}} \times 2^1$

6. Arrodonir mantissa (amunt)

$|z| = 1{,}10100000000000000000100 \times 2^1$

$+ 1$

$|z| = 1{,}\color{blue}{10100000000000000000101} \times 2^1$

7. Codificar exponent (en excés)

E $\rightarrow$ $1+127 = 128 = \color{red}{10000000}$

8. Ajuntar signe (negatiu=1), exponent i fracció (sense el bit ocult)

$z \quad = 1 \; \color{red}{10000000} \; \color{blue}{10100000000000000000101}$

$= 1100\ 0000\ 0101\ 0000\ 0000\ 0000\ 0000\ 0101$

$=$ **0xC0500005**

Bits de guarda

- Quan igualem els exponents al major, ¿quants bits es desplaça la mantissa en el pitjor cas?

Bits de guarda

- Quan igualem els exponents al major, ¿quants bits es desplaça la mantissa en el pitjor cas?
  - Per exemple, restem $x = 1{,}0 \times 2^{127}$ menys $y = 1{,}0 \times 2^{-126}$

Bits de guarda

- Quan igualem els exponents al major, ¿quants bits es desplaça la mantissa en el pitjor cas?
  - Per exemple, restem $x = 1{,}0 \times 2^{127}$ menys $y = 1{,}0 \times 2^{-126}$
  - Igualem els exponents al major = 127, movent la coma a l'esquerra 126 + 127 = 253 posicions

$$x = 1{,}\underbrace{\boxed{000...0}}_{\text{23 bits}} \times 2^{127}$$

$$y = 0{,}\underbrace{\boxed{00000000000000...001}}_{\text{253 posicions a l'esquerra}}\ \underbrace{\boxed{000...0}}_{\text{23 bits}} \times 2^{127}$$

Bits de guarda

- Quan igualem els exponents al major, ¿quants bits es desplaça la mantissa en el pitjor cas?
  - Per exemple, restem $x = 1{,}0 \times 2^{127}$ menys $y = 1{,}0 \times 2^{-126}$
  - Igualem els exponents al major = 127, movent la coma a l'esquerra 126 + 127 = 253 posicions

$$x = 1{,}\underbrace{000...0}_{\text{23 bits}} \times 2^{127}$$

$$y = 0{,}0000000\underbrace{\boxed{0000000...001\ 000...0}}_{\text{més de 200 bits "extra"}} \times 2^{127}$$

- Per no perdre precisió ens cal un sumador amb més de 200 bits bits de guarda!

Bits de guarda

- Podem aconseguir el mateix resultat amb sols 3 bits de guarda
  - Guard (**G**): bit 24 de la mantissa
  - Round (**R**): bit 25 de la mantissa
  - Sticky (**S**): OR lògica de tots els bits a la dreta del bit 25

Bits de guarda

- Podem aconseguir el mateix resultat amb sols 3 bits de guarda
  - Guard (**G**): bit 24 de la mantissa
  - Round (**R**): bit 25 de la mantissa
  - Sticky (**S**): OR lògica de tots els bits a la dreta del bit 25
- Exemple:

$$x = 1{,}00000000000000000000000 \text{ x } 2^{5}$$

$$-y = 1{,}00110011001100110011011 \text{ x } 2^{-3}$$

Bits de guarda

- Podem aconseguir el mateix resultat amb sols 3 bits de guarda
  - Guard (**G**): bit 24 de la mantissa
  - Round (**R**): bit 25 de la mantissa
  - Sticky (**S**): OR lògica de tots els bits a la dreta del bit 25
- Exemple:

$$x = 1{,}00000000000000000000000 \text{ x } 2^{5}$$

$$-y = 1{,}00110011001100110011011 \text{ x } 2^{-3}$$

- Igualant exponents i alineant mantisses

$$x = 1{,}00000000000000000000000 \qquad\qquad \times\ 2^{5}$$

$$-y = 0{,}0000000100110011001100110011011 \times 2^{5}$$

Bits de guarda

- Podem aconseguir el mateix resultat amb sols 3 bits de guarda
    - Guard (**G**): bit 24 de la mantissa
    - Round (**R**): bit 25 de la mantissa
    - Sticky (**S**): OR lògica de tots els bits a la dreta del bit 25
- Exemple:

$$x = 1,00000000000000000000000 \text{ x } 2^{5}$$

$$-y = 1,00110011001100110011011 \text{ x } 2^{-3}$$

- Igualant exponents i alineant mantisses

$$x = 1,00000000000000000000000 \times 2^{5}$$

$$-y = 0,00000001001100110011001\boxed{1}\boxed{0}\underbrace{\boxed{011011}}_{\text{OR}} \times 2^{5}$$

- Operem amb 3 bits de guarda: G, R, S

$$x = 1,00000000000000000000000 \times 2^{5}$$

$$-y = 0,00000001001100110011001\underbrace{\mathbf{101}}_{\text{GRS}} \times 2^{5}$$

Bits de guarda

- Podem aconseguir el mateix resultat amb sols 3 bits de guarda
  - Guard (**G**): bit 24 de la mantissa
  - Round (**R**): bit 25 de la mantissa
  - Sticky (**S**): OR lògica de tots els bits a la dreta del bit 25
- Exemple:

```
 x = 1,00000000000000000000000 x 2^5
-y = 1,00110011001100110011011 x 2^-3
```

- Igualant exponents i alineant mantisses

```
 x = 1,00000000000000000000000            × 2^5
-y = 0,0000000100110011001100110011011    × 2^5
```

- Operem amb 3 bits de guarda: G, R, S

OR

```
 x = 1,00000000000000000000000            × 2^5
-y = 0,0000000100110011001100110 1        × 2^5
```

GRS

- **I obtindrem idèntic resultat que amb infinits bits!**

Multiplicació (divisió) en coma flotant

- Siguin *x*, *y*

$$x = m_x \times 2^{ex}$$

$$y = m_y \times 2^{ey}$$

- El producte i el quocient són

$$x \times y = m_x \times 2^{ex} \times m_y \times 2^{ey} = (m_x \times m_y) \times 2^{(ex+ey)}$$

$$x / y = m_x \times 2^{ex} / m_y \times 2^{ey} = (m_x / m_y) \times 2^{(ex-ey)}$$

Multiplicació (divisió) en coma flotant

- Siguin *x*, *y*

$$x = m_x \times 2^{ex}$$

$$y = m_y \times 2^{ey}$$

- El producte i el quocient són

$$x \times y = m_x \times 2^{ex} \times m_y \times 2^{ey} = (m_x \times m_y) \times 2^{(ex+ey)}$$

$$x / y = m_x \times 2^{ex} / m_y \times 2^{ey} = (m_x / m_y) \times 2^{(ex-ey)}$$

- Algorisme
  - Multiplicar (o dividir) les mantisses (igual que amb els naturals)

Multiplicació (divisió) en coma flotant

- Siguin *x*, *y*

$$x = m_x \times 2^{ex}$$

$$y = m_y \times 2^{ey}$$

- El producte i el quocient són

$$x \times y = m_x \times 2^{ex} \times m_y \times 2^{ey} = (m_x \times m_y) \times 2^{(ex+ey)}$$

$$x / y = m_x \times 2^{ex} / m_y \times 2^{ey} = (m_x / m_y) \times 2^{(ex-ey)}$$

- Algorisme
  - Multiplicar (o dividir) les mantisses (igual que amb els naturals)
  - Sumar (o restar) exponents

Multiplicació (divisió) en coma flotant

- Siguin *x*, *y*

$$x = m_x \times 2^{ex}$$

$$y = m_y \times 2^{ey}$$

- El producte i el quocient són

$$x \times y = m_x \times 2^{ex} \times m_y \times 2^{ey} = (m_x \times m_y) \times 2^{(ex+ey)}$$

$$x / y = m_x \times 2^{ex} / m_y \times 2^{ey} = (m_x / m_y) \times 2^{(ex-ey)}$$

- Algorisme
  - Multiplicar (o dividir) les mantisses (igual que amb els naturals)
  - Sumar (o restar) exponents
  - Ajustar el signe: positiu si els signes són iguals, negatiu altrament

Exemple en base 10

- Suposem un cas senzill que coneixem bé (base `10`)
  - Format: mantissa normalitzada de 4 dígits: $x,xxx \times 10^{xx}$
  - Multiplicar: $(-1,110 \times 10^{10}) \times (9,200 \times 10^{-5})$

Exemple en base 10

- Suposem un cas senzill que coneixem bé (base `10`)
  - Format: mantissa normalitzada de 4 dígits: $x,xxx \times 10^{xx}$
  - Multiplicar: $(-1,110 \times 10^{10}) \times (9,200 \times 10^{-5})$
  - Producte de mantisses:

```
        1,1 1 0
      × 9,2 0 0
---------------
        0 0 0 0
      0 0 0 0
    2 2 2 0
+ 9 9 9 0
---------------
1 0,2 1 2 0 0 0
```

Exemple en base 10

- Suposem un cas senzill que coneixem bé (base `10`)
  - Format: mantissa normalitzada de 4 dígits: $x,xxx \times 10^{xx}$
  - Multiplicar: $(-1,110 \times 10^{10}) \times (9,200 \times 10^{-5})$
  - Producte de mantisses:

```
          1,1 1 0
        × 9,2 0 0
  ---------------
          0 0 0 0
        0 0 0 0
      2 2 2 0
  + 9 9 9 0
  ---------------
  1 0,2 1 2 0 0 0
```

  - Suma d'exponents: 10 + (-5) = 5

$$1\,0,2\,1\,2\,0\,0\,0 \times 10^{5}$$

Exemple en base 10

- Suposem un cas senzill que coneixem bé (base `10`)
  - Format: mantissa normalitzada de 4 dígits: $x,xxx \times 10^{xx}$
  - Multiplicar: $(-1,110 \times 10^{10}) \times (9,200 \times 10^{-5})$
  - Producte de mantisses:

```
            1,1 1 0
          × 9,2 0 0
    ---------------
            0 0 0 0
          0 0 0 0
        2 2 2 0
    + 9 9 9 0
    ---------------
    1 0,2 1 2 0 0 0
```

  - Suma d'exponents: 10 + (-5) = 5

$$1\,0,2\,1\,2\,0\,0\,0 \times 10^{5}$$

  - Normalitzar:

$$1,0\,2\,1\,2\,0\,0\,0 \times 10^{6}$$

Exemple en base 10

- Suposem un cas senzill que coneixem bé (base 10)
  - Format: mantissa normalitzada de 4 dígits: $x,xxx \times 10^{xx}$
  - Multiplicar: $(-1,110 \times 10^{10}) \times (9,200 \times 10^{-5})$
  - Producte de mantisses:

$$\begin{array}{r} 1,110 \\ \times\ 9,200 \\ \hline 0000 \\ 0000 \\ 2220 \\ +\ 9990 \\ \hline 10,212000 \end{array}$$

  - Suma d'exponents: 10 + (-5) = 5

$$10,212000 \times 10^5$$

  - Normalitzar:

$$1,021\boxed{\color{red}{2000}} \times 10^6$$

  - Arrodonir a 4 dígits (avall):

$$1,02\mathbf{1} \times 10^6$$

Exemple en base 10

- Suposem un cas senzill que coneixem bé (base `10`)
  - Format: mantissa normalitzada de 4 dígits: $x,xxx \times 10^{xx}$
  - Multiplicar: $(-1,110 \times 10^{10}) \times (9,200 \times 10^{-5})$
  - Producte de mantisses:

```
          1,1 1 0
        × 9,2 0 0
  ---------------
          0 0 0 0
        0 0 0 0
      2 2 2 0
  + 9 9 9 0
  ---------------
  1 0,2 1 2 0 0 0
```

  - Suma d'exponents: 10 + (-5) = 5

    $10,212000 \times 10^5$

  - Normalitzar:

    $1,0212000 \times 10^6$

  - Arrodonir a 4 dígits (avall):

    $1,021 \times 10^6$

  - Afegir signe (negatiu): $\mathbf{-1,021 \times 10^6}$

Exemple en base 2: $z = x \times y$

Suposem `x=0x3F600000`, `y=0xBED00002`

1. Els escrivim en binari

```
x = 0011 1111 0110 0000 0000 0000 0000 0000
y = 1011 1110 1101 0000 0000 0000 0000 0010
```

Exemple en base 2: $z = x \times y$

Suposem `x=0x3F600000`, `y=0xBED00002`

1. Els escrivim en binari

   x = `0011 1111 0110 0000 0000 0000 0000 0000`
   y = `1011 1110 1101 0000 0000 0000 0000 0010`

2. Separem els camps: signe, exponent, fracció

   x = `0 01111110 11000000000000000000000`
   y = `1 01111101 10100000000000000000010`

Exemple en base 2: $z = x \times y$

Suposem `x=0x3F600000`, `y=0xBED00002`

1. Els escrivim en binari

   x = `0011 1111 0110 0000 0000 0000 0000 0000`
   y = `1011 1110 1101 0000 0000 0000 0000 0010`

2. Separem els camps: signe, exponent, fracció

   x = `0 01111110 11000000000000000000000`
   y = `1 01111101 10100000000000000000010`

3. Convertim exponents a decimal, afegim bit ocult, i el signe

   x = **+1**,`11000000000000000000000` $\times$ 2^{-1}
   y = **-1**,`10100000000000000000010` $\times$ 2^{-2}

Exemple en base 2: $z = x \times y$

Suposem `x=0x3F600000`, `y=0xBED00002`

1. Els escrivim en binari

```
x = 0011 1111 0110 0000 0000 0000 0000 0000
y = 1011 1110 1101 0000 0000 0000 0000 0010
```

2. Separem els camps: signe, exponent, fracció

```
x = 0 01111110 11000000000000000000000
y = 1 01111101 10100000000000000000010
```

3. Convertim exponents a decimal, afegim bit ocult, i el signe

```
x = +1,11000000000000000000000 × 2^-1
y = -1,10100000000000000000010 × 2^-2
```

4. Producte de mantisses (ignorant zeros finals, part entera en negreta)

```
       111 ...(+20 zeros)
  ×    110100000000000000000010
  ------------------------------
                             111
        111
       111
      111
  ------------------------------
  1011011000000000000000001110...(+20 zeros)
```

Exemple (cont.)

5. Suma d'exponents: $-1+(-2) = -3$

$$|z| = \mathbf{10},1101100000000000000001110\ldots \times 2^{-3}$$

Exemple (cont.)

5. Suma d'exponents: $-1+(-2) = -3$

$$|z| = \mathbf{10},1101100000000000000001110... \times 2^{-3}$$

6. Normalitzar

$$|z| = \mathbf{1},01101100000000000000001110... \times 2^{-2}$$

Exemple (cont.)

5. Suma d'exponents: $-1+(-2) = -3$

$$|z| = \mathbf{10},1101100000000000000001110\ldots \times 2^{-3}$$

6. Normalitzar

$$|z| = \mathbf{1},01101100000000000000001110\ldots \times 2^{-2}$$

7. Arrodonir (amunt)

$$\mathbf{1},01101100000000000000001\boxed{\mathbf{110}\ldots} \times 2^{-2}$$

$$+\ 1$$

$$|z| = \mathbf{1},011011000000000000000\mathbf{10} \times 2^{-2}$$

Exemple (cont.)

5. Suma d'exponents: $-1+(-2) = -3$

$$|z| = \mathbf{10},110110000000000000000001110... \times 2^{-3}$$

6. Normalitzar

$$|z| = \mathbf{1},0110110000000000000000001110... \times 2^{-2}$$

7. Arrodonir (amunt)

$$\begin{array}{r} \mathbf{1},01101100000000000000001\mathbf{110}... \times 2^{-2} \\ +\ 1 \\ \hline |z| = \mathbf{1},01101100000000000000010 \times 2^{-2} \end{array}$$

8. Codificar l'exponent (en excés a 127)

$$-2 + 127 = 125 = 01111101$$

Exemple (cont.)

5. Suma d'exponents: $-1+(-2) = -3$

$$|z| = \mathbf{10},1101100000000000000001110... \times 2^{-3}$$

6. Normalitzar

$$|z| = \mathbf{1},01101100000000000000001110... \times 2^{-2}$$

7. Arrodonir (amunt)

$$\begin{array}{rl} & \mathbf{1},01101100000000000000001\mathbf{110...} \times 2^{-2} \\ & \underline{\qquad\qquad\qquad\qquad\qquad + 1} \\ |z| = & \mathbf{1},01101100000000000000010 \qquad \times 2^{-2} \end{array}$$

8. Codificar l'exponent (en excés a 127)

$$-2 + 127 = 125 = 01111101$$

9. Ajuntar signe (negatiu), exponent i mantissa (sense el bit ocult!)

$$\begin{aligned} z &= 1\ 01111101\ 01101100000000000000010 \\ &= 1011\ 1110\ 1011\ 0110\ 0000\ 0000\ 0000\ 0010 \\ &= \mathbf{0xBEB60002} \end{aligned}$$

Aritmètica de coma flotant

- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

Coma flotant en el MIPS

- Coprocesador de coma flotant
  - Històricament, els processadors tenien la unitat de coma flotant (FPU) en un xip opcional separat de la CPU
  - La ISA de MIPS conserva aquesta distinció, encara que actualment les FPUs són part de la CPU: la FPU de MIPS rep el nom de CP1 (co-processador 1)

Coma flotant en el MIPS

- Coprocesador de coma flotant
  - Històricament, els processadors tenien la unitat de coma flotant (FPU) en un xip opcional separat de la CPU
  - La ISA de MIPS conserva aquesta distinció, encara que actualment les FPUs són part de la CPU: la FPU de MIPS rep el nom de CP1 (co-processador 1)
  - Banc de registres propi: 32 registres de 32 bits: `$f0,... $f31`
  - Cada registre pot contenir un "float"
  - Per operar "doubles", només s'usen registres parells: `$f0,$f2,...`

Coma flotant en el MIPS

- Coprocesador de coma flotant
  - Històricament, els processadors tenien la unitat de coma flotant (FPU) en un xip opcional separat de la CPU
  - La ISA de MIPS conserva aquesta distinció, encara que actualment les FPUs són part de la CPU: la FPU de MIPS rep el nom de CP1 (co-processador 1)
  - Banc de registres propi: 32 registres de 32 bits: `$f0,... $f31`
  - Cada registre pot contenir un "float"
  - Per operar "doubles", només s'usen registres parells: `$f0,$f2,...`
  - Hi ha un registre de control addicional per reportar excepcions, configurar els modes d'arrodoniment, etc.

Instruccions MIPS

- Accés a memòria

Simple precisió	Doble precisió
`lwc1 ft, offset(rs)`	`ldc1 ft, offset(rs)`
`swc1 ft, offset(rs)`	`sdc1 ft, offset(rs)`

Instruccions MIPS

- Accés a memòria

Simple precisió	Doble precisió
`lwc1 ft, offset(rs)`	`ldc1 ft, offset(rs)`
`swc1 ft, offset(rs)`	`sdc1 ft, offset(rs)`

- Aritmètiques

Simple precisió	Doble precisió
`add.s  fd,fs,ft`	`add.d  fd,fs,ft`
`sub.s  fd,fs,ft`	`sub.d  fd,fs,ft`
`mul.s  fd,fs,ft`	`mul.d  fd,fs,ft`
`div.s  fd,fs,ft`	`div.d  fd,fs,ft`

Instruccions de coma flotant

- Còpia entre registres

`mfc1   rt, fs`	→	copia de `fs` a `rt`
`mtc1   rt, fs`	→	copia de `rt` a `fs`
`mov.s  fd, fs`	→	copia de `fs` a `fd`

Instruccions de coma flotant

- Còpia entre registres

`mfc1   rt, fs`	→ copia de `fs` a `rt`
`mtc1   rt, fs`	→ copia de `rt` a `fs`
`mov.s  fd, fs`	→ copia de `fs` a `fd`

- Comparació

Simple precisió	Doble precisió
`c.xx.s fs,ft`	`c.xx.d fs,ft`

  - on `xx` ∈ {`eq`,`lt`,`le`}
  - Escriu el resultat al *bit de condició* (és un registre intern)

Instruccions de coma flotant

- Còpia entre registres

`mfc1  rt, fs`	→ copia de `fs` a `rt`
`mtc1  rt, fs`	→ copia de `rt` a `fs`
`mov.s fd, fs`	→ copia de `fs` a `fd`

- Comparació

Simple precisió	Doble precisió
`c.xx.s fs,ft`	`c.xx.d fs,ft`

  - on `xx` ∈ {`eq`,`lt`,`le`}
  - Escriu el resultat al *bit de condició* (és un registre intern)

- Salt

`bc1t  etiqueta`	→ salta si el *bit de condició* = `TRUE`
`bc1f  etiqueta`	→ salta si el *bit de condició* = `FALSE`

Declaracions

- Declaració de variables globals de coma flotant
  - En C

```
float v[2] = {3.1416, -3.5E2};
double x = 3E350, y;
```

  - En MIPS

```
        .data
v:      .float 3.1416, -3.5E2
x:      .double 3E350
y:      .double 0.0
```

  - Alineen a adreces múltiples de 4 (`.float`) o de 8(`.double`)

Subrutines

- Pas de paràmetres i resultats a subrutines
  - Nota: La mescla de paràmetres de coma flotant amb altres enters segueix en MIPS unes regles complexes, que no estudiarem en EC. Sols estudiarem un cas
    - Quan tenim sols **1 o 2 paràmetres de tipus "float"**

Subrutines

- Pas de paràmetres i resultats a subrutines
  - Nota: La mescla de paràmetres de coma flotant amb altres enters segueix en MIPS unes regles complexes, que no estudiarem en EC. Sols estudiarem un cas
    - Quan tenim sols **1 o 2 paràmetres de tipus "float"**
  - Paràmetres: en `$f12` i `$f14`
  - Resultat: en `$f0`
  - Registres "segurs": del `$f20` al `$f31`

Exemple: traduir la funció `func()`

```
float func (float x)
{
  if (x < 1.0)
   return x * x;
  else
   return 2.0 - x;
}
```

```
            .data
const1:     .float 1.0

            .text
            ...
func:
```

Guardem la constant 1.0 en memòria

Exemple: traduir la funció `func()`

```
float func (float x)
{
  if (x < 1.0)
   return x * x;
  else
   return 2.0 - x;
}
```

```
            .data
const1:     .float 1.0

            .text
            ...
func:
    la      $t0, const1
    lwc1    $f16, 0($t0)        # $f16 = 1.0
```

Carreguem la constant 1.0 en $f16

Exemple: traduir la funció `func()`

```
float func (float x)
{
  if (x < 1.0)
   return x * x;
  else
   return 2.0 - x;
}
```

```
             .data
const1:      .float 1.0

             .text
             ...
func:
    la       $t0, const1
    lwc1     $f16, 0($t0)        # $f16 = 1.0
    c.lt.s   $f12, $f16          # x < 1.0?
    bc1f     else                # br. if false
```

Si x < 1.0 és fals, saltem a else

Exemple: traduir la funció `func()`

```
float func (float x)
{
  if (x < 1.0)
   return x * x;
  else
   return 2.0 - x;
}
```

```
            .data
const1:     .float 1.0

            .text
            ...
func:
    la      $t0, const1
    lwc1    $f16, 0($t0)          # $f16 = 1.0
    c.lt.s  $f12, $f16            # x < 1.0?
    bc1f    else                  # br. if false
    mul.s   $f0, $f12, $f12       # x * x
    b       fisi
```

El paràmetre x està en $f12

Exemple: traduir la funció `func()`

```
float func (float x)
{
  if (x < 1.0)
   return x * x;
  else
   return 2.0 - x;
}
```

```
             .data
const1:      .float 1.0

             .text
             ...
func:
    la       $t0, const1
    lwc1     $f16, 0($t0)        # $f16 = 1.0
    c.lt.s   $f12, $f16          # x < 1.0?
    bc1f     else                # br. if false
    mul.s    $f0, $f12, $f12     # x * x
    b        fisi
else:
    add.s    $f16, $f16, $f16 # 1.0 + 1.0
```

Calculem la constant 2.0

Exemple: traduir la funció `func()`

```
float func (float x)
{
  if (x < 1.0)
   return x * x;
  else
   return 2.0 - x;
}
```

```
             .data
const1:      .float 1.0

             .text
             ...
func:
    la       $t0, const1
    lwc1     $f16, 0($t0)       # $f16 = 1.0
    c.lt.s   $f12, $f16         # x < 1.0?
    bc1f     else               # br. if false
    mul.s    $f0, $f12, $f12    # x * x
    b        fisi
else:
    add.s    $f16, $f16, $f16   # 1.0 + 1.0
    sub.s    $f0, $f16, $f12    # 2.0 - x
fisi:
    jr       $ra
```

Aritmètica de coma flotante

- Introducció
- Estàndard IEEE-754: Format
- Rang, precisió i arrodoniment
- Codificacions especials, underflow i nombres denormals
- Conversions entre base 10 i base 2
- Operacions: suma, resta, bits de guarda, multiplicació i divisió
- Coma flotant en MIPS
- Associativitat de la suma

Associativitat de la suma

- La suma de números en coma flotant **no** té la propietat associativa

$$\boxed{x + (y + z) \neq (x + y) + z}$$

Associativitat de la suma

- La suma de números en coma flotant **no** té la propietat associativa

$$\boxed{x + (y + z) \neq (x + y) + z}$$

- Suposem $x = -1{,}5 \times 10^{38}$, $y = 1{,}5 \times 10^{38}$, $z = 1{,}0$

$$\begin{aligned} x + (y + z) &= -1{,}5 \times 10^{38} + (1{,}5 \times 10^{38} + 1{,}0) \\ &= -1{,}5 \times 10^{38} + 1{,}5 \times 10^{38} \\ &= 0{,}0 \end{aligned}$$

Associativitat de la suma

- La suma de números en coma flotant **no** té la propietat associativa

$$\boxed{x + (y + z) \neq (x + y) + z}$$

- Suposem $x = -1{,}5 \times 10^{38}$, $y = 1{,}5 \times 10^{38}$, $z = 1{,}0$

$$\begin{aligned}
x + (y + z) &= -1{,}5 \times 10^{38} + (1{,}5 \times 10^{38} + 1{,}0) \\
&= -1{,}5 \times 10^{38} + 1{,}5 \times 10^{38} \\
&= 0{,}0
\end{aligned}$$

$$\begin{aligned}
(x + y) + z &= (-1{,}5 \times 10^{38} + 1{,}5 \times 10^{38}) + 1{,}0 \\
&= 0{,}0 + 1{,}0 \\
&= 1{,}0
\end{aligned}$$

Exercici: sumar $z = x + y$

Suposem x=`3DC00046`, y=`0xC0800004`

1. Els escrivim en binari

```
x = 0011 1101 1100 0000 0000 0000 0100 0110
y = 1100 0000 1000 0000 0000 0000 0000 0100
```

Exercici: sumar $z = x + y$

Suposem x=`3DC00046`, y=`0xC0800004`

1. Els escrivim en binari

   x = `0011 1101 1100 0000 0000 0000 0100 0110`

   y = `1100 0000 1000 0000 0000 0000 0000 0100`

2. Separem els camps: signe, exponent, fracció

   x = `0 01111011 10000000000000001000110`

   y = `1 10000001 00000000000000000000100`

Exercici: sumar $z = x + y$

Suposem x=3DC00046, y=0xC0800004

1. Els escrivim en binari
   x = 0011 1101 1100 0000 0000 0000 0100 0110
   y = 1100 0000 1000 0000 0000 0000 0000 0100
2. Separem els camps: signe, exponent, fracció
   x = 0 01111011 10000000000000001000110
   y = 1 10000001 00000000000000000000100
3. Convertim exponents a decimal, afegim bit ocult, i el signe

   01111011 = 123 → E = 123 - 127 = -4
   10000001 = 129 → E = 129 - 127 = 2

Exercici: sumar $z = x + y$

Suposem x=3DC00046, y=0xC0800004

1. Els escrivim en binari

   x = 0011 1101 1100 0000 0000 0000 0100 0110
   y = 1100 0000 1000 0000 0000 0000 0000 0100

2. Separem els camps: signe, exponent, fracció

   x = 0 01111011 10000000000000001000110
   y = 1 10000001 00000000000000000000100

3. Convertim exponents a decimal, afegim bit ocult, i el signe

   01111011 = 123 → E = 123 − 127 = -4
   10000001 = 129 → E = 129 − 127 = 2

   x = **+1**,10000000000000001000110 × 2^{-4}
   y = **-1**,00000000000000000000100 × 2^{2}

Exercici: sumar $z = x + y$

Suposem x=3DC00046, y=0xC0800004

1. Els escrivim en binari

   x = 0011 1101 1100 0000 0000 0000 0100 0110
   y = 1100 0000 1000 0000 0000 0000 0000 0100

2. Separem els camps: signe, exponent, fracció

   x = 0 01111011 10000000000000001000110
   y = 1 10000001 00000000000000000000100

3. Convertim exponents a decimal, afegim bit ocult, i el signe

   01111011 = 123 → E = 123 − 127 = -4
   10000001 = 129 → E = 129 − 127 = 2

   x = **+1**,10000000000000001000110 × 2^{-4}
   y = **-1**,00000000000000000000100 × 2^{2}

4. Igualem al major (2) l'exponent de x, desplaçant la coma 6 llocs

   $x = \mathbf{+}0{,}00000\mathbf{1}10000000000000001\boxed{\mathbf{000110}} \times 2^{\mathbf{2}}$

   bits extra

Exercici: sumar $z = x + y$

Suposem x=3DC00046, y=0xC0800004

1. Els escrivim en binari

   x = 0011 1101 1100 0000 0000 0000 0100 0110
   y = 1100 0000 1000 0000 0000 0000 0000 0100

2. Separem els camps: signe, exponent, fracció

   x = 0 01111011 10000000000000001000110
   y = 1 10000001 00000000000000000000100

3. Convertim exponents a decimal, afegim bit ocult, i el signe

   01111011 = 123 → E = 123 − 127 = −4
   10000001 = 129 → E = 129 − 127 = 2

   x = **+1**,10000000000000001000110 × 2^{-4}
   y = **−1**,00000000000000000000100 × 2^{2}

4. Igualem al major (2) l'exponent de x, desplaçant la coma 6 llocs

   x = **+**0,00000**1**10000000000000001 **0** **0** **0110** × 2^{2}

5. Determinem els bits de guarda G, R, S

   OR

   x = **+**0,00000**1**10000000000000001**001** × 2^{2}
   GRS

Exercici: sumar $z = x + y$

6. Sumar magnituds: signes diferents ⇒ restar de la major: |y| – |x|

```
                                               GRS
  |y|   =   1,00000000000000000000100 000  ×  2^2
- |x|   =   0,00000110000000000000001 001  ×  2^2
  |z|   =   0,11111010000000000000010 111  ×  2^2
```

Exercici: sumar $z = x + y$

6. Sumar magnituds: signes diferents ⇒ restar de la major: |y| – |x|

$$
\begin{array}{rcll}
 & & & \text{GRS} \\
|y| & = & 1{,}00000000000000000000100 & \boxed{000} \times 2^{2} \\
-\,|x| & = & 0{,}00000110000000000000001 & \boxed{001} \times 2^{2} \\
\hline
|z| & = & 0{,}11111010000000000000010 & \boxed{111} \times 2^{2}
\end{array}
$$

7. Normalitzar

$$
|z| = \mathbf{1}{,}11110100000000000000101\,\boxed{11} \times 2^{1}
$$

Exercici: sumar $z = x + y$

6. Sumar magnituds: signes diferents ⇒ restar de la major: |y| – |x|

$$
\begin{array}{rcll}
 & & \hspace{20em}\mathbf{GRS} & \\
|y| & = & 1,00000000000000000000100\,\boxed{000} & \times\ 2^{2} \\
-\,|x| & = & 0,00000110000000000000001\,\boxed{\mathbf{001}} & \times\ 2^{2} \\
\hline
|z| & = & 0,11111010000000000000010\,\boxed{\mathbf{111}} & \times\ 2^{2}
\end{array}
$$

7. Normalitzar

$$
|z| = \mathbf{1},11110100000000000000101\,\boxed{\mathbf{11}} \times\ 2^{1}
$$

8. Arrodonir (amunt)

$$
\begin{array}{rcll}
 & & \mathbf{1},11110100000000000000101 & \times\ 2^{1} \\
 & & +\quad 1 & \\
\hline
|z| & = & \mathbf{1},111101000000000000001\mathbf{10} & \times\ 2^{1}
\end{array}
$$

Exercici: sumar $z = x + y$

6. Sumar magnituds: signes diferents ⇒ restar de la major: |y| – |x|

$$
\begin{array}{rcll}
 & & & \textbf{GRS} \\
|y| & = & 1{,}00000000000000000000100 & \boxed{000} \times 2^2 \\
-\,|x| & = & 0{,}00000110000000000000001 & \boxed{\mathbf{001}} \times 2^2 \\
\hline
|z| & = & 0{,}11111010000000000000010 & \boxed{\mathbf{111}} \times 2^2
\end{array}
$$

7. Normalitzar

$$|z| \;=\; \mathbf{1}{,}11110100000000000000101\boxed{\mathbf{11}} \times 2^1$$

8. Arrodonir (amunt)

$$
\begin{array}{rcl}
 & & \mathbf{1}{,}11110100000000000000101 \times 2^1 \\
 & & +\quad 1 \\
\hline
|z| & = & \mathbf{1}{,}111101000000000000001\mathbf{10} \times 2^1
\end{array}
$$

9. Codificar exponent (en excés)

$$E \rightarrow 1{+}127 = 128 = 10000000$$

Exercici: sumar *z* = *x* + *y*

6. Sumar magnituds: signes diferents ⇒ restar de la major: |y| – |x|

```
                                               GRS
 |y|   =  1,00000000000000000000100 000  × 2^2
-|x|   =  0,00000110000000000000001 001  × 2^2
 |z|   =  0,11111010000000000000010 111  × 2^2
```

7. Normalitzar

```
 |z|   =  1,11110100000000000000101 11   × 2^1
```

8. Arrodonir (amunt)

```
          1,11110100000000000000101  × 2^1
                                + 1
 |z|   =  1,11110100000000000000110  × 2^1
```

9. Codificar exponent (en excés)

E → 1+127 = 128 = 10000000

10. Ajuntar signe (negatiu=1), exponent i fracció (sense el bit ocult)

```
z  =  1 10000000 11110100000000000000110
   =  1100 0000 0111 1010 0000 0000 0000 0110
   =  0xC07A0006
```