

# Tema 5. Aritmètica d'enters i coma flotant

## Estructura de Computadors (EC)

Rubèn Tous

rtous@ac.upc.edu  
Computer Architecture Department  
Universitat Politècnica de Catalunya



UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH

# Índex

- 1 5.1 Overflow de suma i resta d'enters
- 2 5.2 Multiplicació entera de 32 bits amb resultat de 64 bits
- 3 5.3 Divisió entera de 32 bits amb càlcul del residu

# Índex

- 1 5.1 Overflow de suma i resta d'enters
- 2 5.2 Multiplicació entera de 32 bits amb resultat de 64 bits
- 3 5.3 Divisió entera de 32 bits amb càlcul del residu

# Overflow de suma i resta de naturals i enters

Si realitzem la següent suma de bits:

$$\begin{array}{r}
 0000\ 0001 \\
 +\ 1111\ 1111 \\
 \hline
 (1)0000\ 0000
 \end{array}$$

Hi ha carry? Sí. Overflow? Depèn. Overflow si són naturals (255+1=256).

# Overflow de suma i resta de naturals i enters

Operació	carry/borrow	ovf. nat.	overflow enters
$a + b = c$	$c < a$ (nat.)	=carry	$(\text{sign\_a} == \text{sign\_b}) \ \&\& \ (\text{sign\_a} != \text{sign\_c})$
$a - b = c$	$a < b$ (nat.)	=borrow	$(\text{sign\_a} != \text{sign\_b}) \ \&\& \ (\text{sign\_a} != \text{sign\_c})$

# Overflow de suma i resta de naturals i enters

Exemple càlcul carry i overflow suma naturals:

```
# $t3 = carry = overflow suma naturals  
# $t2 = t0 + $t1  
addu $t2, $t0, $t1  
sltu $t3, $t2, $t0
```

# Overflow de suma i resta de naturals i enters

Exemple càlcul overflow suma enters:

```
# $t3 = overflow suma entera
# $t2 = t0 + $t1
addu $t2, $t0, $t1
xor  $t3, $t0, $t1    # a xor b
nor  $t3, $t3, $zero  # signe_a == signe_b
xor  $t4, $t0, $t2    # a xor s
and  $t3, $t3, $t4    # (signe_a == signe_b) && (sig
srl  $t3, $t3, 31     # trasllada el bit 31 a la pos
```

# Overflow de suma i resta de naturals i enters

- add, addi i sub generen una excepció en cas d'overflow d'enters (ús: Fortran).
- addu, addiu i subu no generen cap excepció en cas d'overflow (ni de naturals ni d'enters) (ús: C).



# Overflow de suma i resta de naturals i enters

NOTA: En cas de naturals, C especifica que l'overflow ha de fer 'wrapping', és a dir, modul  $2^n$ .

```
#include <stdio.h>
```

```
int main() {  
    register unsigned char a;  
    a = 255;  
    a = a + 1;  
    if (a == 0)  
        printf("Overflow wrapping OK");  
}
```

# Overflow de suma i resta de naturals i enters

En MIPS, en cas de variables de menys de 32 bits, després d'una suma cal afegir una màscara per assegurar que és així:

```
...  
li      $t0, 255  
addiu   $t0, $t0, 1  
andi    $t0, $t0, 0x00ff  
bne     $t0, $zero, fi_if  
...
```

# Índex

- 1 5.1 Overflow de suma i resta d'enters
- 2 5.2 Multiplicació entera de 32 bits amb resultat de 64 bits
- 3 5.3 Divisió entera de 32 bits amb càlcul del residu

# Multiplicació naturals. Algorisme paper i llapis"

Exemple multiplicació naturals:

$$11 * 13 = 143$$

$$1011 = 11$$

$$1101 = 13$$

-----

$$1011$$

$$0000$$

$$1011$$

$$1011$$

-----

$$10001111 = 143$$

# Multiplicació enters

- Multiplicació entera = Canvi de signe + Multiplicació naturals + canvi de signe (si signes diferents)

# Instruccions

mult/multu Ra, Rb

mflo Rd # 32 bits menor pes

mfhi Rd # 32 bits major pes

No usarem (no permet tractar overflow):

mul Rd, Ra, Rb

# Overflow multiplicació natural/entera

Sobreeiximent (no excepció):

- El resultat de la multiplicació pot requerir fins a 64 bits.
- Sobreeiximent = més de 32 bits.

# Overflow multiplicació natural/entera

Sobreeiximent:

- En naturals si algú dels bits 63..32 és diferent de zero
- En enters si algú dels bits 63..32 és diferent del bit 31 (el signe)



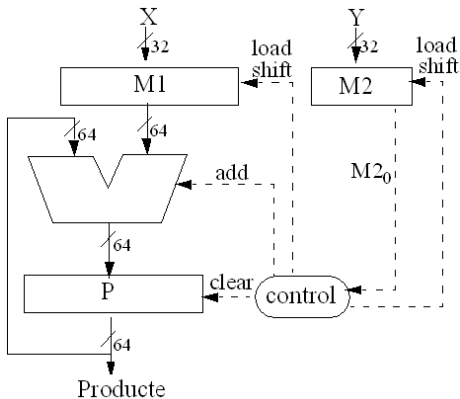
# Overflow multiplicació natural/entera

Algorisme càlcul overflow enters (no provoca excepció):

```
mult $t1, $t2
mflo $t3
mfhi $t4
li    $t0, 31
sra   $t0, $t3, $t0
bne   $t4, $t0
```

# Hardware

$$Z = X * Y$$



```

M163:32 = 0      ;
M131:0 = X       ;
M2 = Y           ;
P = 0            ;

for (i=1; i<=32; i++)
{
    if(M20==1) P=P+M1;
    M1=M1<<1;
    M2=M2>>1;
}
Producte = P;

```

# Hardware

## Exemple (amb 4 bits):

1011 x 1101 (11 x 13 = 143)

-----  
 INI: |P= 00000000|M2=1101|  
      |M1=00001011|

-----  
 IT1: |P= 00001101|M2=0110| s'ha sumat  
      |M1=00010110|

-----  
 IT2: |P= 00001011|M2=0011| NO s'ha sumat  
      |M1=00101100|

-----  
 IT3: |P= 00110111|M2=0001| s'ha sumat  
M1=01011000

# Hardware

- Si cada pas necessita un cicle de rellotge, en total caldran gairebé 100 cicles per fer una multiplicació.
- Un compilador traduirà sempre una multiplicació per  $M$ , essent  $M$  una potència de 2 per un  $\text{sll}$  de  $\log_2 M$ .

# Índex

- 1 5.1 Overflow de suma i resta d'enters
- 2 5.2 Multiplicació entera de 32 bits amb resultat de 64 bits
- 3 5.3 Divisió entera de 32 bits amb càlcul del residu

## Divisió naturals. Algorisme paper i llapis"

Algorisme divisió natural (més fàcil en binari que en el sistema decimal):

Dividend = quocient \* divisor + residu (remainder)

25/2 = 12, residu = 1

```

    25 : 2 = 12
-2*10 -----^^
-----|
      5      |
-2*2  -----|
-----|
      1 (residu)
```

## Divisió naturals. Algorisme paper i llapis"

- Dividir equival a comptar quantes vegades li podem restar  $Y$  a  $X$ . Però fer-ho un a un seria lent.
- Busquem un dígit  $q$  i una potència  $n$  tals que  $Y * q * 10^n$  sigui el més gran possible i menor que  $X$ .
- $q * 10^n$  passa al quocient però només escrivim el dígit  $q$ , no els zeros, com si anèssim sumant.
- En binari  $q$  només pot ser 1 o 0. Busquem  $Y$  multiplicat per la potència de 2 (desplaçat a l'esquerra) més gran que sigui més petita que  $X$ .

# Divisió naturals. Algorisme paper i llapis"

Exemple 25/2:

```

0001 1001 : 10 = 1100
-1 0000-----^^
-----|
 0 1001      |
- 1000-----|
-----
    0001 (residu)

```



# Divisió entera

- Divisió entera = Canvi de signe + Divisió naturals + canvi de signe (si signes diferents)
- Ajustar el signe del residu de  $a/b$  per que sigui el mateix que el del dividend  $a$ .

# Divisió entera

## Sobre el residu:

- La divisió entera en C arrodoneix a 0. Això es podria haver definit d'una altra manera.
- Per aquest motiu el residu ha de tenir signe del dividend (e.g.  $-5/2=-2$  i  $R=-1$  però  $5/-2=-2$  i  $R=1$ ).
- En canvi, una divisió feta amb sra arrodoneix a -infinit. Això exigeix un residu de signe sempre positiu ( $1011 \gg 1 = 1101 = -3$ . Per tant residu =  $-5-(2*-3) = 1$ ).
- Divergència quan dividim un negatiu imparell per una potència de dos.
- L'algorisme "paper i llapis" (el que implementa el hardware) arrodoneix a 0. S'ha d'ajustar el signe del residu per que sigui el mateix que el del divisor.

# Instruccions

div/divu Ra, Rb

mflo Rd #Quocient de la divisió

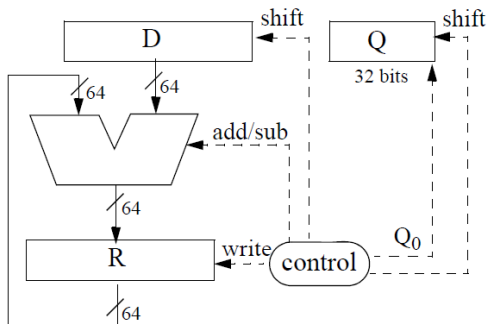
mfhi Rd #Residu de la divisió

# Overflow

- Sobreeiximent (no excepció): En naturals mai. En enters quan dividim  $-2^{31}$  (el menor número enter) per  $-1 = 2^{31}$  (fora de rang)
- No excepció si divisió per 0 en MIPS.

# Hardware

$$Q = X \text{ div } Y; R = X \text{ mod } Y$$



```

D63:32 = Y ;
D31:0 = 0 ;
Q = 0 ;
R63:32 = 0 ;
R31:0 = X ;
for (i=1; i<=32; i++)
{
    D = D>>1;
    R = R-D;
    if (R>=0)
        Q = (Q<<1) | 0x1;
    else
    {
        R = R+D;
        Q = Q<<1;
    }
}
    
```

# Hardware

Exemple:  $1101 / 10 = 110$  ( $13 / 2 = 6$ )

```
-----
INI:   |R= 00001101|Q=0000 |
      |D= 00100000|         |
-----
```

```
IT1:   |R= 00001101|Q=0000 | D>>1
      |D= 00010000|         |
-----
```

```
IT1:   |R=  00000101|Q=0001 | D>>1 + R=R-D
      |D= 00001000|         |
-----
```

```
IT3:   |R= 00000001|Q=0011 | D>>1 + R=R-D
      |D= 00000100|         |
-----
```

```
IT4:   |R= 00000001|Q=0110 | D>>1
      |D= 00000010|         |
-----
```