Tema 5. Aritmètica d'enters i coma flotant

Estructura de Computadors (EC)

Rubèn Tous

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

Índex

Índex

5.5.1 Suma (i resta)

- Estem treballant amb nombres $mantissa * 2^{exponent}$.
- Per poder sumar $x * b^n + y * b^m$ cal que tinguin els mateixos exponents ($n = m$).
- Aleshores podem fer $(x + y) * b^n$.

5.5.1 Suma (i resta)

Algorisme de suma/resta de nombres IEEE 754:

1. Alinear exponents (cap a l'exponent major per facilitar la normalització).
2. Sumar o restar mantisses. Encara que l'operació sigui una suma si els dos operands tenen signes diferents caldrà realitzar una resta (el més petit al més gran, en valor absolut).
3. Normalitzar si cal.
4. Arrodonir al més proper o parell.
5. Ajustar el signe del resultat.

5.5.1 Suma (i resta)

Exemple:

- Suposem $f2=0x4116000 i $f4=0x3F400000.
- Fem add.s $f0, $f2, $f4.
- PAS 1: Passar a binari els nombres:

```
$f2 = 0|100 0001 0|001 0110 0000 0000 0000 0000

Positiu.
Exponent = 130 - 127 = 3

$f4 = 0|011 1111 0|100 0000 0000 0000 0000 0000

Positiu.
Exponent = 126 - 127 = -1
```

5.5.1 Suma (i resta)

- PAS 2: Determinar l'operació que cal realitzar i la posició dels operands.
- Donat que tots dos són positius i l'operació és una suma farem una suma. La posició dels operands no serà rellevant.

5.5.1 Suma (i resta)

- PAS 3: Alinear exponents (cap a l'exponent major per facilitar la normalització).
- Aliniem $f4 ja que té l'exponent més petit:

  `1,100 0000 0000 0000 0000 0000 * 2^-1`
- Per que tingui el mateix exponent que $f2 (3), desplacem la coma quatre posicions a l'esquerra (-1 + 4 = 3):

  `0,000 1100 0000 0000 0000 0000 0000 * 2^3`

5.5.1 Suma (i resta)

PAS 4: Sumem (o restem):

```
 1,001 0110 0000 0000 0000 0000 0000 * 2^3
+0,000 1100 0000 0000 0000 0000 0000 * 2^3
------------------------------------------
 1,010 0010 0000 0000 0000 0000 0000 * 2^3
```

5.5.1 Suma (i resta)

PAS 5: Normalitzem si cal (no cal en aquest cas).

5.5.1 Suma (i resta)

PAS 6: Arrodonim al més proper o parell si cal (no cal en aquest cas):

5.5.1 Suma (i resta)

PAS 7: Codifiquem el resultat (ajustant el signe si cal):

```
1,010 0010 0000 0000 0000 0000 0000 * 2^3

Exponent = 3 + 127 = 130
130 = 128 + 2 = 100 0001 0

0|100 0001 0|010 0010 0000 0000 0000 0000

= 0x4122 0000
```

5.5.1 Suma (i resta)

Exemple (problema 5.27). Suma simple però cal normalitzar.

- Suposem $f2=0x417ac000, $f4=0x3f140000.
- Fem add.s $f0,$f2,$f4.
- PAS 1: Passar a binari els nombres:

```
$f2 = 0|100 0001 0|111 1010 1100 0000 0000 0000

Positiu.
Exponent = 130 - 127 = 3

$f4 = 0|011 1111 0|001 0100 0000 0000 0000 0000

Positiu.
Exponent = 126 - 127 = -1
```

5.5.1 Suma (i resta)

- PAS 2: Determinar l'operació que cal realitzar i la posició dels operands.
- Donat que tots dos són positius i l'operació és una suma farem una suma. La posició dels operands no serà rellevant.

5.5.1 Suma (i resta)

- PAS 3: Alinear exponents (cap a l'exponent major per facilitar la normalització).
- Aliniem $f4 ja que té l'exponent més petit:

  `1,001 0100 0000 0000 0000 0000 * 2^-1`
- Per que tingui el mateix exponent que $f2 (3), desplacem la coma quatre posicions a l'esquerra (-1 + 4 = 3):

  `0,000 1001 0100 0000 0000 0000 0000 * 2^3`

5.5.1 Suma (i resta)

PAS 4: Sumem (o restem):

```
 1,111 1010 1100 0000 0000 0000 0000 * 2^3
+0,000 1001 0100 0000 0000 0000 0000 * 2^3
------------------------------------------
10,000 0100 0000 0000 0000 0000 0000 * 2^3
```

5.5.1 Suma (i resta)

PAS 5: Normalitzem si cal (sí en aquest cas):

```
10,000 0100 0000 0000 0000 0000 0000 * 2^3
=
1,000 0010 0000 0000 0000 0000 0000 * 2^4
```

5.5.1 Suma (i resta)

PAS 6: Arrodonim al més proper o parell si cal (no cal en aquest cas):

5.5.1 Suma (i resta)

PAS 7: Codifiquem el resultat (ajustant el signe si cal):

```
1,000 0010 0000 0000 0000 0000 0000 * 2^4

Exponent = 4 + 127 = 131
131 = 128 + 3 = 100 0001 1

0|100 0001 1|000 0010 0000 0000 0000 0000

= 0x4182 0000
```

5.5.1 Suma (i resta)

Exemple (problema 5.28). Una suma que en realitat és una resta.

- Suposem $f2=0xC076C000, $f4=0x3ECA8000.
- Fem add.s $f0,$f2,$f4.
- PAS 1: Passar a binari els nombres:

```
$f2 = 1|100 0000 0|111 0110 1100 0000 0000 0000

Positiu.
Exponent = 128 - 127 = 1

$f4 = 0|011 1110 1|100 1010 1000 0000 0000 0000

Positiu.
Exponent = 125 - 127 = -2
```

5.5.1 Suma (i resta)

- PAS 2: Determinar l'operació que cal realitzar i la posició dels operands.
- Donat que $f2 té signe negatiu caldrà restar.
- Donat que $f2 és més gran (té exponent més gran) en valor absolut, l'ordre dels operands serà:

```
 |$f2|
-|$f4|
------
 |$f0|
```

- El resultat ($f0) tindrà signe negatiu.

5.5.1 Suma (i resta)

- PAS 3: Alinear exponents (cap a l'exponent major per facilitar la normalització).
- Aliniem $f4 ja que té l'exponent més petit:

  `1,100 1010 1000 0000 0000 0000 * 2^-2`
- Per que tingui el mateix exponent que $f2 (-1), desplacem la coma 3 posicions a l'esquerra (-2 + 3 = 1):

  `0,001 1001 0101 0000 0000 0000 0000 * 2^1`

5.5.1 Suma (i resta)

PAS 4: Sumem (o restem):

```
 1,111 0110 1100 0000 0000 0000 0000 * 2^1
-0,001 1001 0101 0000 0000 0000 0000 * 2^1
------------------------------------------
 1,101 1101 0111 0000 0000 0000 0000 * 2^1
```

5.5.1 Suma (i resta)

PAS 5: Normalitzem si cal (no cal en aquest cas).

5.5.1 Suma (i resta)

PAS 6: Arrodonim al més proper o parell si cal (no cal en aquest cas):

5.5.1 Suma (i resta)

PAS 7: Codifiquem el resultat (ajustant el signe si cal):

```
1,101 1101 0111 0000 0000 0000 0000 * 2^1

Exponent = 1 + 127 = 128
= 100 0000 0

Signe negatiu!!!

1|100 0000 0|101 1101 0111 0000 0000 0000 0000

= 0xC05D 7000
```

Índex

5.5.2 Bits GUARD, ROUND i STICKY

Bits ROUND i STICKY:

- L'alineament d'un dels operands pot requerir molts bits.
- Els bits ROUND i STICKY s'afegeixen al final dels dos operands i del resultat per calcular l'arrodoniment.
- El bit ROUND és senzillament el primer que queda fora.
- El bit STICKY és la OR de tots els bits que venen després.

5.5.2 Bits GUARD, ROUND i STICKY

Exemple:

- Si tinguèssim infinits bits, la següent suma donaria:

```
 1,000 0000 0000 0000 0000 0000 |
+0,000 0000 0000 0000 0000 0000 |1000 0001
------------------------------------------
 1,000 0000 0000 0000 0000 0000 |1000 0001

 Arrodonim:

 1,000 0000 0000 0000 0000 0001
```

5.5.2 Bits GUARD, ROUND i STICKY

- Afegim els bits ROUND (1) i STICKY (0 OR 0 OR 0 OR 0 OR 0 OR 0 OR 1).

```
                                      RS
 1,000 0000 0000 0000 0000 0000 |  00
+0,000 0000 0000 0000 0000 0000 |  11
---------------------------------------------
 1,000 0000 0000 0000 0000 0000 |  11

Arrodonim:

1,000 0000 0000 0000 0000 0001
```

5.5.2 Bits GUARD, ROUND i STICKY

Bit GUARD:

- En cas que el resultat d'una suma o resta s'hagi de normalitzar ens farà falta un bit més.
- S'afegeix un bit GUARD abans dels bits ROUND i STICKY.

5.5.2 Bits GUARD, ROUND i STICKY

Exemple necessitat del bit GUARD:

- Si tinguèssim infinits bits, la següent suma donaria:

```
 1,000 0000 0000 0000 0000 0000 |
-0,000 0000 0000 0000 0000 0000 | 1001
-----------------------------------------
 0,111 1111 1111 1111 1111 1111 | 0111

Normalitzem:

 1,11 1111 1111 1111 1111 1111 0 | 111

Arrodonim:

 1,11 1111 1111 1111 1111 1111 1
```

5.5.2 Bits GUARD, ROUND i STICKY

- Ara només amb els bits ROUND i STICKY:

```
 1,000 0000 0000 0000 0000 0000 |
-0,000 0000 0000 0000 0000 0000 | 11
----------------------------------------------
 0,111 1111 1111 1111 1111 1111 | 01

Normalitzem:

 1,11 1111 1111 1111 1111 1111 0 | 1

 Perdem el bit round!

Arrodonim:

 1,11 1111 1111 1111 1111 1111 0 :-(
```

5.5.2 Bits GUARD, ROUND i STICKY

- Ara amb els bits GUARD, ROUND i STICKY:

```
                                          GRS
 1,000 0000 0000 0000 0000 0000 |
-0,000 0000 0000 0000 0000 0000 | 101
---------------------------------------------
 0,111 1111 1111 1111 1111 1111 | 011

Normalitzem:

 1,11 1111 1111 1111 1111 1111 0 | 11

Arrodonim:

 1,11 1111 1111 1111 1111 1111 1 :-)
```

5.5.2 Bits GUARD, ROUND i STICKY

- Els bits GUARD, ROUND i STICKY garanteixen el mateix arrodoniment que amb infinits bits.
- Però si no arrodoníssim el resultat no seria el mateix.
- Per calcular l'error d'una operació haurem de comparar el resultat codificat amb el resultat amb infinits bits.

Índex

5.5.3 Exemple suma (Examen Final 2011-2012 Q2)

Suposem \$f2=0x40800000 i \$f4=0xBE800009 i s'executa add.s \$f0, \$f2, \$f4.

- PAS 1: Passem a binari i calculem exponents.

```
$f2 = 0|100 0000 1|000 0000 0000 0000 0000 0000
(exponent = 129 - 127 = 2)

$f4 = 1|011 1110 1|000 0000 0000 0000 0000 1001
(exponent = 125 - 127 = -2)
```

5.5.3 Exemple suma (Examen Final 2011-2012 Q2)

- PAS 2: Mirem signes i magnituds per veure quina operació cal fer:

```
Cal restar ja que $f4 és negatiu.

Com el valor absolut de $f2 és més gran farem:

 |$f2|
-|$f4|
------
 |$f0|

El resultat serà positiu.
```

5.5.3 Exemple suma (Examen Final 2011-2012 Q2)

- PAS 3: Aliniem exponent $f4 ja que és el més petit:

  `0,000 1000 0000 0000 0000 0000 101 * 2^2`

5.5.3 Exemple suma (Examen Final 2011-2012 Q2)

- PAS 4: Restem:

```
                                           GRS
 1,000 0000 0000 0000 0000 0000 |          * 2^2
-0,000 1000 0000 0000 0000 0000 | 101 * 2^2
---------------------------------------
 0,111 0111 1111 1111 1111 1111 | 011 * 2^2
```

5.5.3 Exemple suma (Examen Final 2011-2012 Q2)

- PAS 5: Normalitzem el resultat (compte amb l'exponent!).

```
                                            GRS
1,110 1111 1111 1111 1111 1110 | 11   * 2^1
```

5.5.3 Exemple suma (Examen Final 2011-2012 Q2)

- PAS 6: Arrodonim al més proper o parell

  `1,110 1111 1111 1111 1111 1111 * 2^1`

- PAS 7: Ajustem signe (positiu) i codifiquem.

  `0|100 0000 0|110 0111 1111 1111 1111 1111`

  `= 0x406FFFFF`