# Quadern de laboratori
# Estructura de Computadors

Emilio Castillo
José María Cela
Montse Fernández
David López
Joan Manuel Parcerisa
Angel Toribio
Rubèn Tous
Jordi Tubella
Gladys Utrera

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Quadrimestre de Primavera - Curs 2014/15

# Licencia Creative Commons

# Sessió 4: Codificació en coma flotant

**Objectiu:** Estudiar la codificació de nombres en coma flotant, i practicar la manipulació de bits (desplaçaments i operacions lògiques bit a bit), i la codificació de subrutines.

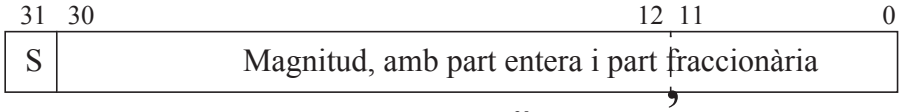
## Lectura prèvia

En aquesta sessió posarem en pràctica els coneixements sobre subrutines, sobre operacions amb camps de bits, i sobre el format de coma flotant per crear un programa en assemblador que resolgui el següent problema: sigui `A` un número real codificat en un cert format de coma fixa en base 2, es demana convertir-lo en el número real R equivalent, codificat en el format de coma flotant de simple precisió.

### El format de coma fixa (format inicial)

El format de coma fixa del número A és el següent: té 32 bits, el bit 31 conté el *signe* (0 per als positius) i la resta de bits del 30 al 0 representen la *magnitud*, codificada en el sistema posicional de base 2, amb 19 bits a l'esquerra de la coma (part entera) i 12 bits a la dreta (part fraccionària):

**A =**

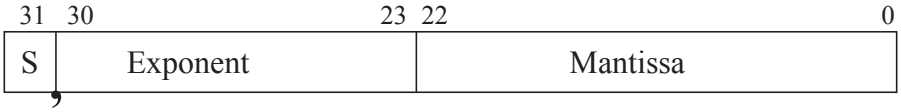



$$valorimplicit(A) = (-1)^{A_{31}} \times \sum_{i=0}^{30} A_i \times 2^{i-12}$$

### El format de coma flotant IEEE-754 de simple precisió (format final)

El format de coma flotant del número R, de simple precisió definit a l'estàndard IEEE-754 també té 32 bits: el bit 31 conté el signe (0 per als positius); els 23 bits de menys pes contenen la mantissa, en base 2, fraccionària, normalitzada, amb bit ocult i coma a la dreta d'aquest bit; i els restants 8 bits (del 23 al 30) contenen l'exponent, un enter codificat en excés a 127:

**R =**




$$valorimplicit(R) = (-1)^{R_{31}} \times \left(1 + \sum_{i=0}^{22} R_i \times 2^{i-23}\right) \times 2^{\left(-127 + \sum_{i=23}^{30} R_i \times 2^{i-23}\right)}$$

# Enunciats de la sessió

Aquesta sessió es dedicarà a escriure el següent programa de conversió de coma fixa a coma flotant, seguint les instruccions del text. Els formats de coma fixa i de coma flotant en simple precisió estan explicats a la Lectura Prèvia.

```
int signe;
int exponent;
int mantissa;
int cfixa = 0x87D18A00;          // número inicial, en coma fixa
float cflotant;                  // resultat final, en coma flotant

main() {
     descompon(cfixa, &signe, &exponent, &mantissa);
     cflotant = compon(signe, exponent, mantissa);
}
```

**Figura 4.1:** Programa en alt nivell que converteix un número en coma fixa a coma flotant

El programa de conversió (figura 4.1), s'executa en dos passos:

- Funció `descompon`. Descompon el número en coma fixa en tres parts: signe, mantissa normalitzada i exponent en excés a 127.
- Funció `compon`. Compon les tres parts, signe, mantissa i exponent, en un únic número en coma flotant en simple precisió.

Per emmagatzemar les tres parts obtingudes al primer pas definirem tres enters de mida word: `signe`, `exponent` i `mantissa`. En les activitats següents es detalla el funcionament de les dues subrutines.

## Activitat 4.A: Funció `descompon`

La funció `descompon` (veure figura 4.2) rep com a paràmetres un nombre en coma fixa `cf` i tres enters passats per referència. La funció extrau de `cf` els següents valors: el signe (0 o 1); mantissa normalitzada i sense bit ocult (23 bits); i exponent en excés a 127 (8 bits), i els guarda als enters `signe`, `mantissa` i `exponent`.

```
void descompon(int cf, int *s, int *e, int *m) {
int exp;
     *s = (cf < 0);                      // si és negatiu val 1, sinó 0
     cf = cf << 1;                       // elimina el signe
     if (cf == 0)
            exp = 0;                     // el número és un +0 o un -0
     else {
            exp = 18;
            while (cf >= 0) {            // normalitza i calcula exponent
                   cf = cf << 1;
                   exp--;
            }
            cf = (cf >> 8) & 0x7FFFFF;   // alinear i eliminar bit ocult
            exp = exp + 127;             // codificar en excés a 127
     }
     *e =   exp;                         // guarda exponent resultant
     *m =   cf;                          // guarda mantissa resultant
}
```

**Figura 4.2:** Codi en alt nivell de la subrutina `descompon`

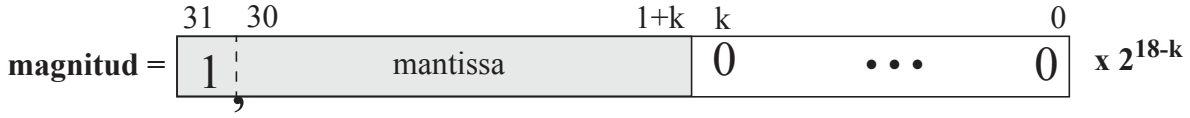
Recordem que el nombre en coma fixa es compon de signe (1 bit) i magnitud (31 bits). Per tant, la primera operació de la subrutina consisteix a separar el signe de la magnitud: codifica el signe en l'enter `signe` (0 si és positiu, 1 si és negatiu) i l'elimina del nombre `cf` desplaçant tots els bits 1 posició a l'esquerra. Fixa't que la coma de la magnitud ara s'ha desplaçat també 1 posició i està situada a la dreta del bit 13:


A continuació es comprova si la magnitud val zero (pot ser un +0 o un -0, ja que hem eliminat el signe). Si la magnitud és zero, codificarem la mantissa `cf` i l'exponent `exp` com zero.

Si no és zero, per obtenir la mantissa normalitzada i l'exponent corresponent, desplacem la coma 18 posicions a l'esquerra fins a situar-la a la dreta del bit 31. Compte! que els bits no

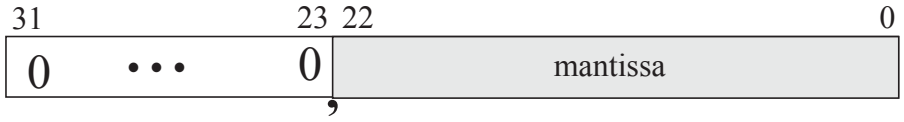
es desplacen, tan sols hem d'ajustar el valor inicial de l'exponent `exp`, que en lloc de 0 serà 18:


Per normalitzar la mantissa caldrà eliminar els zeros inicials que pugui tenir, desplaçant la magnitud repetidament cap a l'esquerra fins que el primer bit no nul ocupi la posició 31. Naturalment, aquest desplaçament de la magnitud sense moure la coma equival a un desplaçament a la dreta de la coma, i en conseqüència necessita un ajust de l'exponent, restant-li la mateixa quantitat. L'algorisme (figura 4.2) consisteix en un bucle on a cada iteració es desplaça `cf` 1 posició a l'esquerra i es decrementa l'exponent una unitat, fins que el bit més significatiu és 1. En aquest punt, podem dir que la mantissa ja està normalitzada i l'exponent calculat. Per exemple, si suposem que aquest bucle itera `k` vegades, `cf` i `exp` queden així:




A continuació, la mantissa normalitzada s'ha de desplaçar a la dreta perquè ocupi els 24 bits de menor pes de `cf`. Per tal que el bit 31 passi a ocupar la posició 23 cal fer un desplaçament a la dreta de 8 posicions.

Després hem de deixar en `cf` una mantissa de sols 23 bits, posant a zero el bit ocult, que ara ocupa la posiciò 23, així com els bits del 23 al 31, fent una *and* lògica amb la màscara `0x7FFFFF`:




Després, a l'exponent `exp` li sumem 127 per codificar-lo en excés a 127.

Finalment, guardem els resultats de `cf` i `exp` als enters `mantissa` i `exponent`.

Completa l'exercici 4.1 abans de continuar:

**Exercici 4.1:** Tradueix a assemblador MIPS la subrutina `descompon`.

```
  descompon:
descompon:
        addiu   $sp, $sp,
-4              # EPILOG
        sw      $ra, 0($sp)


        slt     $t0, $a0,
$zero           # If negative the val is 1,
else 0
        sw      $t0, 0($a1)
        sll     $a0 $a0,
1               # remove sign

        bne     $a0, $zero, else  # if cf ==
0
        li      $t1,
0                       # $t1 <--> exp,
exp = 0
        b       end_else

        else:
        li      $t1,
18                      # else {...}
        do:
#
do {...}
        sll     $a0, $a0, 1
        addiu   $t1, $t1, 1
        bge     $a0, $zero, do          #
while (cf >= 0)

        sra     $a0, $a0, 8
        li      $t2, 0x7FFFFF
        and     $a0, $a0,
$t2             # align and remove
hidden bit
        addiu   $t1, $t1,
127             # apply excess-127

        end_else:

        sw      $t1,
0($a2)          # assign final result
        sw      $a0, 0($a3)

        lw      $ra,
0($sp)          # PROLEG
        addiu   $sp, $sp, 4
        jr      $ra
```

Al fitxer **s4a.s** estan ja programats el programa principal (`main`) i les declaracions de variables globals en assemblador MIPS. Afegeix-hi el codi de l'exercici 4.1. A continuació, carrega'l al simulador, assembla'l i executa'l.

Comprova que al final de l'execució del programa `signe = 1`, que `exponent= 0x0000008D` (`141` en decimal), i que `matissa = 0x007A3140`.

## Activitat 4.B: Funció `compon`

```
float compon(int signe, int exponent, int mantissa)
{
     return (signe << 31) | (exponent << 23) | mantissa;
}
```

**Figura 4.3:** Codi en alt nivell de la subrutina `compon`

La subrutina `compon` (figura 4.3) rep com a paràmetres els enters `signe`, `exponent` i `mantissa` passats per valor i retorna el número en coma flotant de simple precisió equivalent. El signe s'ha de desplaçar a la posició 31, i l'exponent a la posició 23. Els tres camps de bits resultants es poden combinar amb operacions `or` lògiques, ja que sabem que cada camp ocupa sols els bits que li són propis, essent la resta zeros.

Completa l'exercici 4.2 abans de continuar:

**Exercici 4.2:** Programa en assemblador la subrutina `compon`.

```
compon:
 compon:
        addiu   $sp, $sp,
-4              # PROLEG
        sw      $ra, 0($sp)

        sll     $a0, $a0,
31              # Make shifts to registers
        sll     $a1, $a1, 23

        or      $t0, $a0,
$a1             # assemble the final result
        or      $t0, $t0, $a2
```

```
        mtc1    $t0, $f0        # move result
to return register in Coprocessor 1

        lw      $ra,
0($sp)          # PROLEG
        addiu   $sp, $sp, 4
        jr      $ra
```

Afegeix el codi de l'exercici 4.2 al fitxer **s4a.s**. Veuràs que el fitxer conté una subrutina `compon` provisional amb 1 sola instrucció, escrita per permetre fer proves a l'activitat 4.A: esborra-la. A continuació, assembla'l i executa'l.

Comprova que el número resultant, en coma flotant, val `cflotant=0xC6FA3140`. També pots comprovar el seu valor en decimal en el simulador. De quina manera ho consultaràs?

Dins el registre $f0 de la pestanya del coprocessador 1 al simulador MARS

Quin és el valor final de `cflotant`, en decimal, que dóna Mars?

Valor decimal de `cflotant` = -32024,625

Completa el següent exercici abans de continuar:

**Exercici 4.3:** Codifica en coma fixa i en coma flotant (en hexadecimal) els següents números:

| Decimal | cfixa (valor inicial) | cflotant (valor final) |
|---|---|---|
| 0.0 | 0x 00000000 | 0x 00000000 |
| - 0.0 | 0x80000000 | 0x80000000 |
| 12.75 | 0x0000CC00 | 0x 414C0000 |

Verifica que el teu programa també converteix correctament els números de l'exercici 4.3, inicialitzant `cfixa` amb els valors de la segona columna i observant al simulador si el resultat en `cflotant` és el de la tercera columna.

## Activitat 4.C: Errors de precisió en la conversió

Un cop llegida la pràctica i resolts els anteriors exercicis hauries de ser capaç de reflexionar sobre els errors de precisió que es poden cometre fent la conversió proposada, i respondre les preguntes del següent exercici:

**Exercici 4.4:** Contesta les següents preguntes

1) Quina condició ha de complir el valor inicial de `cfixa` perquè es produeixi pèrdua de precisió en la conversió que proposa aquesta pràctica?

   Si el valor en coma fixa te més bits significatius que el permes a la mantissa del format en coma flotant

2) Indica un valor de `cfixa` per al qual es produiria pèrdua de precisió al convertir-lo, i el corresponent valor en coma flotant:

| cfixa | cflotant |
|---|---|
| 0x 00FFFFFF | 0x 457FFFFF |

3) En quina sentència concreta del programa en alt nivell es pot produir la pèrdua de precisió?

   cf = (cf >> 8) & 0x7FFFFF

4) Quin dels 4 modes d'arrodoniment que coneixes està portant a la pràctica aquest programa de conversió?

   El de truncament, descarta els bits sobrants

5) El format de coma fixa explicat en aquesta pràctica permet codificar un rang de valors bastant limitat. Indica un número positiu que estigui DINS el rang del format de coma flotant de simple precisió (en decimal) però que estigui FORA del rang del format de coma fixa. Indica també quin és el MENOR número potència de 2 que compleixi aquesta condició.

   2^(24) esta dins del rang de la representacio en coma flotant, però fora del rang de la representacio en coma fixa. El número mes petit que compleix aquesta condició es 2^(19).