Tema 5. Coma flotant
Problemes

Curs 2019-20 Primavera

Grup 30

Joan Manuel Parcerisa

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Problema 5.22

5.22. Suposant que *f* i *g* són variables de tipus float (coma flotant en simple precisió), i estan emmagatzemades als registres $f12 i $f14 del coprocessador aritmètic CP1, tradueix a assemblador les següents sentències escrites en C. Tingues en compte que el llenguatge assemblador no admet nombres decimals fraccionaris en qualsevol context. Comprova que el teu codi es pot compilar sense errors i que dóna el resultat correcte en el simulador MARS:

a) `f = 3.14;`

```
        .data
pi:     .float 3.14

        .text
        la   $t0, pi
        lwc1 $f12, 0($t0)
```

Problema 5.22

5.22. Suposant que *f* i *g* són variables de tipus float (coma flotant en simple precisió), i estan emmagatzemades als registres $f12 i $f14 del coprocessador aritmètic CP1, tradueix a assemblador les següents sentències escrites en C. Tingues en compte que el llenguatge assemblador no admet nombres decimals fraccionaris en qualsevol context. Comprova que el teu codi es pot compilar sense errors i que dóna el resultat correcte en el simulador MARS:

b)
```
f = g;
```
```
mov.s    $f12, $f14
```

Problema 5.22

5.22. Suposant que *f* i *g* són variables de tipus float (coma flotant en simple precisió), i estan emmagatzemades als registres $f12 i $f14 del coprocessador aritmètic CP1, tradueix a assemblador les següents sentències escrites en C. Tingues en compte que el llenguatge assemblador no admet nombres decimals fraccionaris en qualsevol context. Comprova que el teu codi es pot compilar sense errors i que dóna el resultat correcte en el simulador MARS:

b)
```
f = g;
        mov.s    $f12, $f14
```

c)
```
f = g + 3.14
        .data
    pi: .float 3.14

        .text
        la      $t0, pi
        lwc1    $f10, 0($t0)
        add.s   $f12, $f10, $f14
```

Problema 5.23

5.23. Converteix els següents nombres decimals al format de coma flotant de simple precisió, expressant el resultat en hexadecimal. Determina en cada cas si es comet error per pèrdua de precisió en la seva representació i, en cas afirmatiu, calcula aquest error expressant-lo en decimal.

a) **340.0** = 101010100 = **1**,01010100 * 2^8

(Exponent = 8 + 127 = 10000111)

340 = 0 10000111 01010100000000000000000

= 0100 0011 1010 1010 0000 0000 0000 0000

= 0x43AA0000

No hi ha arrodoniment → No hi ha error

Problema 5.23

5.23. Converteix els següents nombres decimals al format de coma flotant de simple precisió, expressant el resultat en hexadecimal. Determina en cada cas si es comet error per pèrdua de precisió en la seva representació i, en cas afirmatiu, calcula aquest error expressant-lo en decimal.

b) 44,4

44 = 101100

0,4 =

0,4 * 2 = **0**,8
0,8 * 2 = **1**,6
0,6 * 2 = **1**,2
0,2 * 2 = **0**,4

período

0,4 * 2 = **0**,8
...

0,4 = 0,0110 0110 0110 0110 0110 01 …

44,4 = 101100,01100110011001100**1**1001 …

Problema 5.23

44,4 = 101100,01100110011001100111001 … =

= **1**,011 0001 1001 1001 1001 1001 1001 … * 2^5

Bits extra: arrodonim amunt!

Exponent = 5 + 127 = 10000100

44,4 = 0 10000100 01100011001100110011**10**

exponent fracció (23 bits)

= 0x4231999A

Problema 5.23

44,4 = 101100,01100110011001100 … =

= 1,011 0001 1001 1001 1001 1001 1001 … * 2^5

Bits extra: arrodonim amunt!

Exponent = 5 + 127 = 10000100

44,4 = 0 10000100 01100011001100110011**10**

exponent fracció (23 bits)

= 0x4231999A

Error = 1,01100011001100110011**10** * 2^5 (arrodonit)

- 1,01100011001100110011**01**1001 * 2^5 ("exacte")

= 0,00000000000000000000000111 * 2^5 = 111 * 2^{-27} * 2^5

= 7 * 2^{-22} (acceptable sense calculadora)

= 1,67 * 10^{-6} (amb calculadora)

Problema 5.24

Donats els números decimals A=−1609,5 i B=−938,8125

a) Converteix-los al format de coma flotant de simple precisió, expressant el resultat en hexadecimal, i calcula l'error comès en la conversió, per pèrdua de precisió, expressat en decimal.

A = -1609,5 = -11001001001,1 = -**1**,10010010011 * 2^{10}

Exponent = 10+127 = 10001001

A = 1 10001001 10010010011000000000000 = 0xC4C93000

exponent fracció (23 bits)

No hi ha arrodoniment → no hi ha error de precisió

Problema 5.24

Donats els números decimals A=−1609,5 i B=−938,8125

a) Converteix-los al format de coma flotant de simple precisió, expressant el resultat en hexadecimal, i calcula l'error comès en la conversió, per pèrdua de precisió, expressat en decimal.

A = -1609,5 = -11001001001,1 = -1,10010010011 * 2^{10}

Exponent = 10+127 = 10001001

A = 1 10001001 10010010011000000000000 = 0xC4C93000

exponent fracció (23 bits)

No hi ha arrodoniment → no hi ha error de precisió

B = -938,8125 = -1110101010,1101 = -**1**,1101010101101 * 2^{9}

Exponent = 9+127 = 10001000

B = 1 10001000 11010101011010000000000 = 0xC46AB400

exponent fracció (23 bits)

No hi ha arrodoniment → no hi ha error de precisió

Problema 5.24

b) Converteix-los al format de coma flotant de doble precisió, expressant el resultat en hexadecimal, i calcula l'error comès en la conversió, per pèrdua de precisió, expressat en decimal.

$A = -1609,5 = 11001001001,1 = \mathbf{1},10010010011 * 2^{10}$

Exponent = 10+1023 = 10000001001

= 1 10000001001 1001001001100000000000000 … 0

Exponent (11) fracció (52)

A = 0xC09926000000000

No hi ha arrodoniment → no hi ha error de precisió

Problema 5.24

b) Converteix-los al format de coma flotant de doble precisió, expressant el resultat en hexadecimal, i calcula l'error comès en la conversió, per pèrdua de precisió, expressat en decimal.

A = -1609,5 = 11001001001,1 = 1,10010010011 * 2^{10}

Exponent = 10+1023 = 10000001001

= 1 10000001001 1001001001100000000000000 … 0

Exponent (11) fracció (52)

A = 0xC09926000000000

No hi ha arrodoniment → no hi ha error de precisió

B = -938,8125 = 1110101010,1101 = **1**,1101010101101 * 2^{9}

Exponent = 9+1023 = 10000001000

= 1 10000001000 1101010101101000000000000 … 0

Exponent (11) fracció (52)

B = 0xC08D56800000000

No hi ha arrodoniment → no hi ha error de precisió

Problema 5.28

5.28. Suposem que `$f2=0xC076c000`, `$f4=0x3eca8000`, i que executem la instrucció: `add.s $f6,$f2,$f4`. Suposant que el sumador té 1 bit de guarda, un d'arrodoniment i un de "sticky", i que arrodoneix al més pròxim (al parell en el cas equidistant):

a) Calcular a mà, seguint l'algorisme de suma de nombres en coma flotant, el valor final de `$f6` en hexadecimal ?

`$f2` = 0xC076C000 = 1100 0000 0111 0110 1100 0000 0000 0000

(**Exponent** = 128 – 127 = +1)

= **-1**,111 0110 1100 0000 0000 0000 * 2^{1}

`$f4` = 0x3ECA8000 = 0011 1110 1100 1010 1000 0000 0000 0000

(**Exponent** = 125 – 127 = -2)

= **1**,100 1010 1000 0000 0000 0000 * 2^{-2}

Igualar els exponents al major (el de $f2, que és +1):

`$f4` = 0,00**1** 1001 0101 0000 0000 0000 000 * 2^{1}

Problema 5.28

Signes diferents → **restar valors absoluts**:

|$f2| = 1,111 0110 1100 0000 0000 0000 * 2^1

|$f4| = - 0,00**1** 1001 0101 0000 0000 0000 000 * 2^1

|$f6| = 1,101 1101 0111 0000 0000 0000 000 * 2^1

Problema 5.28

Signes diferents → restar valors absoluts:

|$f2| = 1,111 0110 1100 0000 0000 0000 $* 2^1$

|$f4| = - 0,00**1** 1001 0101 0000 0000 0000 000 $* 2^1$

|$f6| = 1,101 1101 0111 0000 0000 0000 000 $* 2^1$

Normalitzar (ja ho està)

Arrodonir

= 1,101 1101 0111 0000 0000 0000 $* 2^1$

(**Signe**: el del major ($f2) = negatiu)

$f6 = - **1**,101 1101 0111 0000 0000 0000 $* 2^1$

(**Exponent** = 1 + 127 = 128 = 10000000)

$f6 = 1 10000000 10111010111000000000000

= 1100 0000 0101 1101 0111 0000 0000 0000

= 0xC05D7000

Problema 5.28

b) Es produeix algun error de precisió en el resultat ?

No hi ha arrodoniment → No hi ha error

c) Converteix a decimal el valor final de `$f6`.

$f6 = - 1,101 1101 0111 0000 0000 0000 * 2^{1}

Fem la mantissa entera (movent la coma i ajustant l'exponent)

= - 1101 1101 0111 * 2^{-10}

= -3543 / 2^{10} = -3543 / 1024 = **-3,46**

Problema 5.29

5.29. Suposem que `$f1=0x42000000` i `$f2=0x3d800000`, i que executem la instrucció: `mul.s $f6, $f2, $f4`. Suposant que el sumador té 1 bit de guarda, un d'arrodoniment i un de "sticky", i que arrodoneix al més pròxim (al parell en el cas equidistant) ¿quin és el valor final de `$f6` en hexadecimal?

`$f1` = 0x42000000 = 0100 0010 0000 0000 0000 0000 0000 0000

(**Exponent** = 10000100 = 132 – 127 = +5)

= **1**,0 * 2^5

`$f2` = 0x3d800000 = 0011 1101 1000 0000 0000 0000 0000 0000

(**Exponent** = 01111011 = 123 – 127 = -4)

= **1**,0 *2^{-4}

Multiplicació de mantisses: 1,0 * 1,0 = 1,0

Suma d'exponents: +5 + (-4) = +1

`$f6` = **1**,0 * 2^1

(**Exponent** = 1 + 127 = 10000000)

= 0 10000000 00000000000000000000

= 0100 0000 0000 0000 0000 0000 0000 0000

= 0x40000000

Problema 5.31

5.31. Donades les següents declaracions de funcions:

```
float mitjana (float p[]);

float variancia (float vec[])
{
      int i;
      float m, q;
      float vquadrats[100];

      for (i=0; i<100; i++)
            vquadrats[i] = vec[i] * vec[i];

      q = mitjana(vquadrats);
      m = mitjana(vec);
      return q - m * m;
}
```

a) Quines variables locals cal guardar al B.A. i quines en registres?

A la pila: `vquadrats`

En registres: `i, m, q`

Quines variables, paràmetres i/o valors intermedis han d'ocupar **registres segurs**, i quines han d'ocupar registres temporals?

Registres segurs: `vec` (cal copiar-lo a `$s0`) i `q` (en `$f20`)

Registres temporals: `i` (en `$t0`) i `m` (en `$f0`)

Problema 5.31

b) Tradueix a assemblador MIPS la funció *variancia*.

```
variancia:
  addiu   $sp, $sp, -412
  swc1    $f20, 400($sp)
  sw      $s0, 404($sp)
  sw      $ra, 408($sp)
```

```
float variancia (float vec[])
{
      int i;
      float m, q;
      float vquadrats[100];

      for (i=0; i<100; i++)
            vquadrats[i] = vec[i] * vec[i];

      q = mitjana(vquadrats);
      m = mitjana(vec);
      return q - m * m;
}
```

```
  lwc1    $f20, 400($sp)
  lw      $s0, 404($sp)
  lw      $ra, 408($sp)
  addiu   $sp, $sp, 412
  jr      $ra
```

Problema 5.31

b) Tradueix a assemblador MIPS la funció *variancia*.

```
variancia:
  addiu   $sp, $sp, -412
  swc1    $f20, 400($sp)
  sw      $s0, 404($sp)
  sw      $ra, 408($sp)

  li      $t0, 0          # i=0
  li      $t1, 100
for:
  bge     $t0, $t1, fifor
  sll     $t2, $t0, 2     # i*4
  addu    $t3, $a0, $t2   # @vec+i*4
  lwc1    $f1, 0($t3)     # vec[i]
  mul.s   $f1, $f1, $f1   # vec[i]*vec[i]
  addu    $t3, $t2, $sp   # @vquadrats+i*4
  swc1    $f1, 0($t3)     # vquadrats[i]
  addiu   $t0, $t0, 1     # i++
  b       for
fifor:
```

```
float variancia (float vec[])
{
      int i;
      float m, q;
      float vquadrats[100];

      for (i=0; i<100; i++)
            vquadrats[i] = vec[i] * vec[i];

      q = mitjana(vquadrats);
      m = mitjana(vec);
      return q - m * m;
}
```

```
  lwc1    $f20, 400($sp)
  lw      $s0, 404($sp)
  lw      $ra, 408($sp)
  addiu   $sp, $sp, 412
  jr      $ra
```

Problema 5.31

b) Tradueix a assemblador MIPS la funció *variancia*.

```
variancia:
  addiu   $sp, $sp, -412
  swc1    $f20, 400($sp)
  sw      $s0, 404($sp)
  sw      $ra, 408($sp)

  li      $t0, 0          # i=0
  li      $t1, 100
for:
  bge     $t0, $t1, fifor
  sll     $t2, $t0, 2     # i*4
  addu    $t3, $a0, $t2   # @vec+i*4
  lwc1    $f1, 0($t3)     # vec[i]
  mul.s   $f1, $f1, $f1   # vec[i]*vec[i]
  addu    $t3, $t2, $sp   # @vquadrats+i*4
  swc1    $f1, 0($t3)     # vquadrats[i]
  addiu   $t0, $t0, 1     # i++
  b       for
fifor:
  move    $s0, $a0        # copia vec a $s0
  move    $a0, $sp        # passa vquadrats
  jal     mitjana
```

```
float variancia (float vec[])
{
      int i;
      float m, q;
      float vquadrats[100];

      for (i=0; i<100; i++)
            vquadrats[i] = vec[i] * vec[i];

      q = mitjana(vquadrats);
      m = mitjana(vec);
      return q - m * m;
}
```

```
  lwc1    $f20, 400($sp)
  lw      $s0, 404($sp)
  lw      $ra, 408($sp)
  addiu   $sp, $sp, 412
  jr      $ra
```

Problema 5.31

b) Tradueix a assemblador MIPS la funció *variancia*.

```
variancia:
  addiu   $sp, $sp, -512
  swc1    $f20, 400($sp)
  sw      $s0, 404($sp)
  sw      $ra, 408($sp)

  li      $t0, 0          # i=0
  li      $t1, 100
for:
  bge     $t0, $t1, fifor
  sll     $t2, $t0, 2     # i*4
  addu    $t3, $a0, $t2   # @vec+i*4
  lwc1    $f1, 0($t3)     # vec[i]
  mul.s   $f1, $f1, $f1   # vec[i]*vec[i]
  addu    $t3, $t2, $sp   # @vquadrats+i*4
  swc1    $f1, 0($t3)     # vquadrats[i]
  addiu   $t0, $t0, 1     # i++
  b       for
fifor:
  move    $s0, $a0        # copia vec a $s0
  move    $a0, $sp        # passa vquadrats
  jal     mitjana
```

```
float variancia (float vec[])
{
      int i;
      float m, q;
      float vquadrats[100];

      for (i=0; i<100; i++)
            vquadrats[i] = vec[i] * vec[i];

      q = mitjana(vquadrats);
      m = mitjana(vec);
      return q - m * m;
}
```

```
  mov.s   $f20, $f0       # copia q a $f20
  move    $a0, $s0        # passa vec
  jal     mitjana
```

```
  lwc1    $f20, 400($sp)
  lw      $s0, 404($sp)
  lw      $ra, 408($sp)
  addiu   $sp, $sp, 512
  jr      $ra
```

Problema 5.31

b) Tradueix a assemblador MIPS la funció *variancia*.

```
variancia:
  addiu   $sp, $sp, -412
  swc1    $f20, 400($sp)
  sw      $s0, 404($sp)
  sw      $ra, 408($sp)

  li      $t0, 0          # i=0
  li      $t1, 100
for:
  bge     $t0, $t1, fifor
  sll     $t2, $t0, 2     # i*4
  addu    $t3, $a0, $t2   # @vec+i*4
  lwc1    $f1, 0($t3)     # vec[i]
  mul.s   $f1, $f1, $f1   # vec[i]*vec[i]
  addu    $t3, $t2, $sp   # @vquadrats+i*4
  swc1    $f1, 0($t3)     # vquadrats[i]
  addiu   $t0, $t0, 1     # i++
  b       for
fifor:
  move    $s0, $a0        # copia vec a $s0
  move    $a0, $sp        # passa vquadrats
  jal     mitjana
```

```
float variancia (float vec[])
{
      int i;
      float m, q;
      float vquadrats[100];

      for (i=0; i<100; i++)
            vquadrats[i] = vec[i] * vec[i];

      q = mitjana(vquadrats);
      m = mitjana(vec);
      return q - m * m;
}
```

```
  mov.s   $f20, $f0       # copia q a $f20
  move    $0a, $s0        # passa vec
  jal     mitjana
  mul.s   $f0, $f0, $f0   # m*m
  sub.s   $f0, $f20, $f0  # q - m*m
```

```
  lwc1    $f20, 400($sp)
  lw      $s0, 404($sp)
  lw      $ra, 408($sp)
  addiu   $sp, $sp, 412
  jr      $ra
```