Aulas 13 & 14

- Representação de números em vírgula flutuante
- A norma IEEE 754
  - Operações aritméticas em vírgula flutuante
  - Precisão simples e precisão dupla
  - Arredondamentos
- Unidade de vírgula flutuante do MIPS
  - Instruções da FPU do MIPS
- Exemplo de codificação utilizando as instruções da FPU do MIPS

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva

Representação de números em Vírgula flutuante

- A codificação de quantidades numéricas com que trabalhamos até agora estiveram sempre associadas à representação de números inteiros (com ou sem sinal)
- A representação de números reais, por outro lado, coloca desafios de natureza distinta, em particular no que concerne à **gama de valores representáveis** e à sua **precisão** (número de algarismos representativos das partes inteira e fraccionária, respectivamente)

Exemplo: -23.45129876 (Representação em vírgula fixa)

Quantos dígitos devem ser reservados para a parte **inteira** e quantos para a parte **fraccionária**, sabendo nós que o espaço de armazenamento é limitado?

Representação de números em Vírgula flutuante

A quantidade **–23.45129876** pode também ser representada recorrendo à notação científica:

$-2.345129876 \times \mathbf{10^1}$ — $\boxed{\mathbf{-(2\times10^0+3\times10^{-1}+4\times10^{-2}+5\times10^{-3}+\ldots+6\times10^{-9}) \times 10^1}}$

$-0.2345129876 \times \mathbf{10^2}$ — $\boxed{\mathbf{-(0\times10^0+2\times10^{-1}+3\times10^{-2}+4\times10^{-3}+\ldots+6\times10^{-10}) \times 10^2}}$

- São representações do mesmo valor em que a posição da vírgula tem de ser ponderada, na interpretação numérica da quantidade, pelo valor do expoente de base 10
- Esta técnica, em que a vírgula pode ser deslocada sem alterar o valor representado, designa-se também de **representação em vírgula flutuante**
- A representação em vírgula flutuante tem a vantagem de não desperdiçar espaço de armazenamento com os zeros à esquerda da quantidade representada
- No primeiro exemplo, o número de dígitos diferentes de zero à esquerda da vírgula é igual a um: diz-se que a **representação está normalizada**

Representação de números em Vírgula flutuante

- A representação de quantidades em vírgula flutuante, em sistemas computacionais digitais, faz-se recorrendo à estratégia descrita nos slides anteriores, mas usando agora a base dois:

$$\boxed{\mathbf{N = (+/-)\ 1.f \times 2^{Exp}}}$$

(representação **normalizada** de uma quantidade binária em vírgula flutuante)

Em que:

f – parte **fraccionária** representada por ***n*** bits
1.f – **mantissa** (1 + parte fraccionária)
Exp – **expoente** da potência de base 2 representado por ***m*** bits

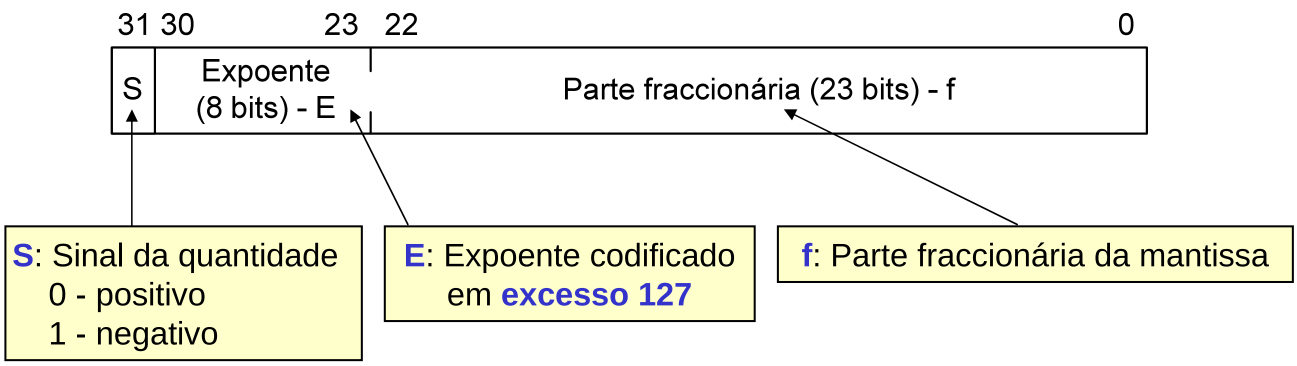
Representação de números em Vírgula flutuante

- O problema da divisão do espaço de armazenamento coloca-se também neste caso, mas agora na determinação do **número de bits** ocupados pela **parte fraccionária** e pelo **expoente**
- Essa divisão é um **compromisso** entre **gama de representação** e **precisão**:
  - Aumento do número de bits da parte fraccionária ⇒ maior precisão
  - Aumento do número de bits do expoente ⇒ maior gama de representação

Um bom *design* implica compromissos adequados!

Representação de números em Vírgula flutuante

Norma IEEE 754 (precisão simples)



- A representação é **normalizada** (o bit à esquerda da vírgula é **sempre 1**). Assim, esse bit é omitido da representação (*hidden bit*)
- A Mantissa pode tomar valores compreendidos entre **1.00000000000000000000000** e **1.11111111111111111111111**

Representação de números em Vírgula flutuante

Norma IEEE 754 (precisão simples)

31	30 … 23	22 … 0
S	Expoente (8 bits) - E	Parte fraccionária (23 bits) - f

- O expoente é codificado em **excesso 127** ($2^{n-1}-1$, n=8 bits). Ou seja, é somado ao expoente verdadeiro (Exp) o valor 127 para obter o código de representação (i.e. ***Exp = E – 127***)

$$N = (-1)^S\ 1.f \times 2^{Exp} = (-1)^S\ 1.f \times 2^{E-127}$$

- O código 127 representa assim o expoente zero, códigos maiores do que 127 representam expoentes positivos e códigos menores que 127 representam expoentes negativos
- O expoente pode, desta forma, tomar valores entre **-126** e **+127** [códigos 1 a 254]. **Os códigos 0 e 255 são reservados**

Representação de números em Vírgula flutuante

Norma IEEE 754 (precisão simples)

31	30 … 23	22 … 0
S	Expoente (8 bits) - E	Parte fraccionária (23 bits) - f

Exemplo: **0 10000010 10100000000000000000000 (0x41500000)**

Sinal = 0 (quantidade positiva)

Expoente = [130] – *offset* = 130 – 127 = 3 ⇔ (Exp = E – offset)

Mantissa = (1 + parte fraccionária) = **1** + **.101** = **1.101**

A quantidade representada, Q, será então:

$$Q = +1.101 \times 2^3 = (1 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}) \times 2^3$$

$$= 1.625 \times 8 = 13 \times 10^0 = 13$$

Representação de números em Vírgula flutuante

Norma IEEE 754 (precisão simples)

31	30 … 23	22 … 0
S	Expoente (8 bits) - E	Parte fraccionária (23 bits) - f

$$N = (-1)^S\ 1.f \times 2^{\,Exp} = (-1)^S\ 1.f \times 2^{\,E-127}$$

A gama de representação suportada por este formato será portanto:

$$[\mathbf{1.00000000000000000000000 \times 2^{-126}}\ \text{a}\ \mathbf{1.11111111111111111111111 \times 2^{+127}}]$$

$$[\mathbf{1.1754943 \times 10^{-38}}\ \text{a}\ \mathbf{3.4028235 \times 10^{+38}}]$$

Nº de bits por casa decimal = $\log_2(10) \cong 3.3$

A **representação em decimal** deve comportar, assim, 23 / 3.3 = **7 casas decimais**

Representação de números em Vírgula flutuante

Norma IEEE 754 (precisão simples)

31	30 … 23	22 … 0
S	Expoente (8 bits) - E	Parte fraccionária (23 bits) - f

Nas operações com quantidades representadas neste formato podem ocorrer situações de ***overflow*** e de ***underflow***:

- **Overflow**: quando o expoente do resultado não cabe no espaço que lhe está destinado → **E > 254**)

$$N_{resultado} > \mathbf{1.11111111111111111111111 \times 2^{+127}}$$

- **Underflow**: caso em que o expoente é tão pequeno que também não é representável → **E < 1**)

$$\mathbf{0} < N_{resultado} < \mathbf{1.00000000000000000000000 \times 2^{-126}}$$

Representação de números em Vírgula flutuante

Exemplo: codificar no formato vírgula flutuante IEEE 754 precisão simples, o valor **-12.59375_{10}**

Parte inteira: $12_{10} = 1100_2$

Parte fraccionária: $0.59375_{10} = 0.10011_2$

$12.59375_{10} = 1100.10011_2 \times 2^0$

Normalização: $1100.10011_2 \times 2^0 = 1.10010011_2 \times 2^3$

Expoente: $+3 + 127 = 130_{10} = 10000010_2$

1 **10000010** **10010011000000000000000**

0xC1498000

```
      0.59375
          × 2
MSB   1.18750
      0.18750
          × 2
      0.37500
      0.37500
          × 2
      0.75000
      0.75000
          × 2
      1.50000
      0.50000
          × 2
LSB   1.00000
```


Representação de números em Vírgula flutuante

Operações aritméticas em vírgula flutuante – **ADIÇÃO (subtracção)**

Exemplo: **$N = 1.11 \times 2^0 + 1.00 \times 2^{-2}$**

1º Passo: Igualar os expoentes ao **maior** dos expoentes

$N = 1.11 \times 2^0 + 0.01 \times 2^0$

2º Passo: Somar (subtrair) as mantissas mantendo os expoentes

$N = 1.11 \times 2^0 + 0.01 \times 2^0 = 10.00 \times 2^0$

3º Passo: **Normalizar** o resultado

$N = 10.00 \times 2^0 = 1.000 \times 2^1$

- **Exercício**: A e B representam, em hexadecimal, a codificação no formato IEEE754 de duas quantidades reais:
    - A = 0x41600000
    - B = 0xC0C00000
  - Realize as seguintes operações, apresentando o resultado codificado no formato IEEE754, em hexadecimal
    - R1 = A – B
    - R2 = B – A
    - R3 = A + B
    - R4 = B + A
  - Repita o exercício supondo:
    - A = 0xC0C00000
    - B = 0x41600000

Representação de números em Vírgula flutuante

Operações aritméticas em vírgula flutuante – **MULTIPLICAÇÃO**

Exemplo: $\mathbf{N = (1.11 \times 2^{0}) \times (1.01 \times 2^{-2})}$

1º Passo: Somar os expoentes

Expr = 0+(-2) = [0+127]+[-2+127] = [127+125]**-127** = [125] = -2

2º Passo: Multiplicar as mantissas

Mr = $1.11 \times 1.01 = 10.0011$

3º Passo: **Normalizar** o resultado

$N = 10.0011 \times 2^{-2} = \mathbf{1.00011 \times 2^{-1}}$

Representação de números em Vírgula flutuante

Operações aritméticas em vírgula flutuante – **DIVISÃO**

Exemplo: **N = (1.001×2^{0}) / (1.1×2^{-2})**

1º Passo: Subtrair os expoentes

Expr = 0-(-2) = [0+127]-[-2+127] = [127-125]**+127** = [129] = 2

2º Passo: Dividir as mantissas

Mr = 1.001 / 1.1 = 0.11

3º Passo: **Normalizar** o resultado

$N = 0.11 \times 2^{2} = 1.1 \times 2^{1}$

Representação de números em Vírgula flutuante

A norma IEEE 754 suporta a representação de quantidades em **precisão simples (32 bits)** e em **precisão dupla (64 bits)**

31	30 … 23	22 … 0	
S	Expoente (8 bits) - E	Parte fraccionária (23 bits) - f	**Precisão Simples**

63	62 … 52	51 … 0	
S	Expoente (11 bits) - E	Parte fraccionária (52 bits) - f	**Precisão Dupla**

$N = (-1)^{S}\ 1.f \times 2^{(E-127)}$ (Precisão simples - tipo **float**)

$N = (-1)^{S}\ 1.f \times 2^{(E-1023)}$ (Precisão dupla - tipo **double**)

Representação de números em Vírgula flutuante

Casos particulares

A norma IEEE 754 suporta ainda a representação de alguns **casos particulares**:

1. **A quantidade zero**; essa quantidade não seria representável de acordo com o formato descrito até aqui
2. **+/-infinito**
3. Resultados não numéricos (***NAN – Not a Number***). Um exemplo possível é o resultado de uma divisão de zero por zero
4. A fim de aumentar a resolução (menor quantidade representável) é ainda possível adoptar um formato de **mantissa desnormalizada** no qual o bit à esquerda da vírgula é zero

Representação de números em Vírgula flutuante

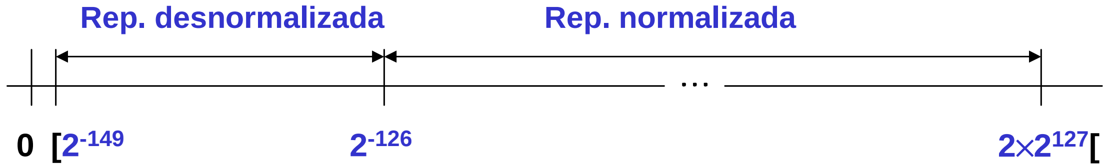
Representação desnormalizada:

- Permite a representação de quantidades cada vez mais pequenas (com cada vez menos precisão - *underflow* gradual)
- A gama de representação suportada pelo **formato de mantissa desnormalizada** em precisão simples será portanto:

$$[0.00000000000000000000001 \times 2^{-126} \text{ a } 0.11111111111111111111111 \times 2^{-126}]$$

$$[1 \times 2^{-23} \times 2^{-126} \text{ a } 1.0 \times 2^{-126}[$$

$$[1{,}4012985 \times 10^{-45} \text{ a } 1.1754943 \times 10^{-38}[$$



Representação de números em Vírgula flutuante

Casos particulares:

Precisão Simples		Precisão Dupla		Representa
Expoente	Parte Frac.	Expoente	Parte Frac.	
0	0	0	0	0
0	$\neq 0$	0	$\neq 0$	Número desnormalizado
1 a 254	*qualquer*	*1 a 2046*	*qualquer*	*Nº em vírgula flutuante normalizado*
255	0	2047	0	Infinito
255	$\neq 0$	2047	$\neq 0$	NAN (Not a Number)

Representação de números em Vírgula flutuante

Arredondamentos

- As operações aritméticas são efectuadas com um número de bits da parte fraccionária superior ao disponível no espaço de armazenamento
- Desta forma, na conclusão de qualquer operação aritmética é necessário proceder ao arredondamento do resultado por forma a assegurar a sua adequação ao espaço que lhe está destinado
- As técnicas mais comuns no processo de **arredondamento do resultado** (o qual introduz um erro) são:
    - Truncatura
    - Arredondamento
    - Arredondamento para o par (ímpar) mais próximo

Representação de números em Vírgula flutuante

Técnicas de arredondamento do resultado :

Truncatura (exemplo com 2 dígitos na parte fraccionária: d=2)

Número	Trunc(x)	Erro
x.00	x	0
x.01	x	-1/4
x.10	x	-1/2
x.11	x	-3/4

Erro médio = (0 – 1/4 – 1/2 – 3/4) / 4

= **–3/8**

- Mantém-se a parte inteira, desprezando qualquer informação que exista à direita da vírgula

Representação de números em Vírgula flutuante

Técnicas de arredondamento do resultado :

Arredondamento (exemplo com 2 dígitos na parte fraccionária: d=2)

Número	Arred(x)	Erro
x.00	x	0
x.01	x	-1/4
x.10	x + 1	+1/2
x.11	x + 1	+1/4

Erro médio = (0 – 1/4 + 1/2 + 1/4) / 4

= **+1/8**

- Mantém-se a parte inteira quando o 1º dígito decimal for 0 ou soma-se “1” à parte inteira quando aquele for “1” (arred(x) = trunc(x + 0.5))
- O erro médio é mais próximo de zero do que no caso da truncatura, mas ligeiramente polarizado do lado positivo

Representação de números em Vírgula flutuante

Técnicas de arredondamento do resultado :

Arredondamento para o par mais próximo (exemplo com d=2)

Número	Arred(x)	Erro	Número	Arred(x)	Erro
x0.00	x0	0	x1.00	x1	0
x0.01	x0	-1/4	x1.01	x1	-1/4
x0.10	**x0**	**-1/2**	**x1.10**	**x1 + 1**	**+1/2**
x0.11	x1	+1/4	x1.11	x1 + 1	+1/4

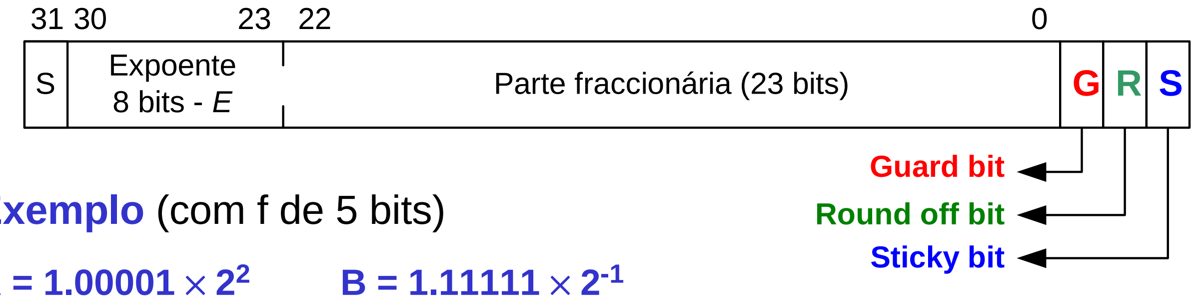
- Semelhante à técnica de arredondamento, mas decidindo, para o caso "**xx.10**", em função do primeiro dígito à esquerda da vírgula
- **Erro médio** = -1/8 + 1/8 = **0**

Representação de números em Vírgula flutuante

- De modo a minimizar o erro introduzido pelo processo de arredondamento, os valores resultantes **de cada fase intermédia** da execução de uma operação aritmética são armazenados com três bits suplementares
- Estes bits designam-se pelas letras **G**, **R** e **S** e destinam-se a:
  - **G – *Guard Bit*** – Bit suplementar que pode ser necessário na pós-normalização da mantissa decorrente das operações de divisão
  - **R – *Round off bit*** – Bit usado na operação de arredondamento
  - **S – *Sticky bit*** – Bit que resulta da soma lógica de todos os bits à direita do bit R. Este bit é usado, juntamente com o bit R, na implementação de um esquema de arredondamento para o par mais próximo (no caso de haver pós-normalização)

Representação de números em Vírgula flutuante

Arredondamentos



Exemplo (com f de 5 bits)

A = 1.00001 $\times$ 2^2 **B = 1.11111 $\times$ 2^{-1}**

Mant(A/B) = 1.00001 / 1.11111 Expoente(A/B) = 2 – (-1) = 3

= **0.10000 1100001** **G=1**, **R=1**, **S = OR(00001) = 1**

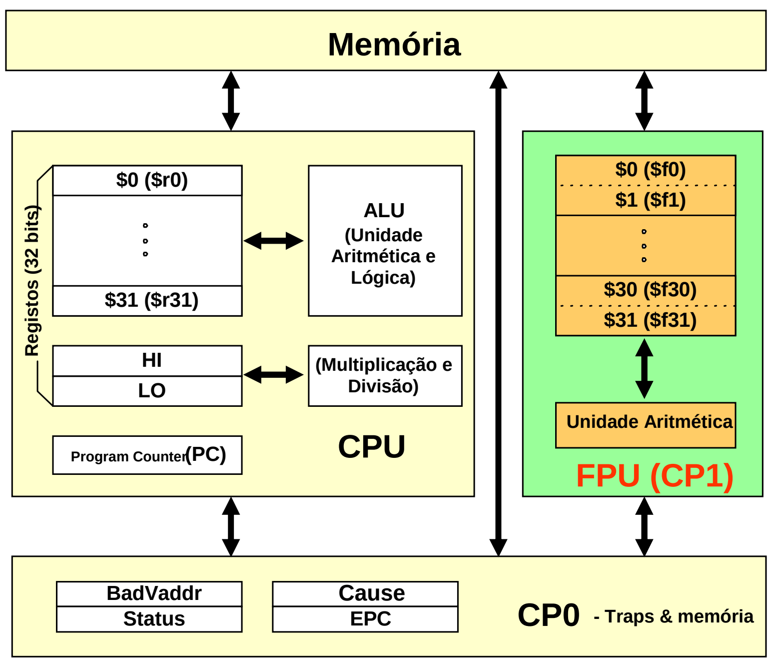
= **0.10000 111**

Mant(A/B) $_{norm}$ = **1.0000$\boxed{1\ 11}$** Arred(1,11$_2$) = 10$_2$

Arredondamento $\Rightarrow$ Mant(A/B) = 1.00010

A/B = 1.00010 $\times$ 2^2

Instruções de cálculo em Vírgula flutuante no MIPS



- O MIPS inclui um coprocessador aritmético (Coprocessador 1) capaz de efectuar operações aritméticas em vírgula flutuante
- Esse coprocessador tem o seu próprio espaço de armazenamento composto por um conjunto de 32 registos de 32 bits cada, e o seu próprio *set* de instruções

Instruções de cálculo em Vírgula flutuante no MIPS

- Os registos do coprocessador aritmético são designados, no *Assembly* do MIPS, pelas letras **$fn**, em que o indíce **n** toma valores entre 0 e 31
- Cada par de registos consecutivos [**$fn**,**$fn+1**] (**com n par**) pode funcionar como um registo de 64 bits para armazenar valores em **precisão dupla**. Em *Assembly* a referência ao par de registos faz-se indicando como operando o **registo par**
- **Apenas os registos de índice par** podem ser usados no contexto das instruções

Instruções de cálculo em Vírgula flutuante no MIPS

Aritméticas

```
abs.p   FPdst,FPsrc               # Absolute Value
neg.p   FPdst,FPsrc               # Negate
div.p   FPdst,FPsrc1, FPsrc2      # Divide
mul.p   FPdst,FPsrc1, FPsrc2      # Multiply
add.p   FPdst,FPsrc1,FPsrc2       # Addition
sub.p   FPdst,FPsrc1, FPsrc2      # Subtract
```

O sufixo **.p** representa a **precisão** com que é efectuada a operação (simples ou dupla). Deverá, na instrução, ser substituído pelas letras **.s** ou **.d** respectivamente.

Instruções de cálculo em Vírgula flutuante no MIPS

Conversão entre tipos

```
cvt.d.s FPdst,FPsrc       # Convert Single to Double
cvt.d.w FPdst,FPsrc       # Convert Integer to Double
cvt.s.d FPdst,FPsrc       # Convert Double to Single
cvt.s.w FPdst,FPsrc       # Convert Integer to Single
cvt.w.d FPdst,FPsrc       # Convert Double to Integer
cvt.w.s FPdst,FPsrc       # Convert Single to Integer
```

Formato original

Formato do resultado

As **conversões** entre tipos de representação **são efectuadas pela FPU** pelo que apenas podem ter como operandos/destinos registos da FPU

Instruções de cálculo em Vírgula flutuante no MIPS

Transferência de informação (entre registos do CPU e da FPU e entre registos da FPU)

Registo do **CPU** Registo da **FPU**

```
mtc1  CPUSrc,FPdst   # Move to Coprocessor 1
mfc1  CPUdst,FPsrc   # Move from Coprocessor 1
mov.s FPdst, FPsrc   # Move from FPsrc to FPdst (single)
mov.d FPdst, FPsrc   # Move from FPsrc to FPdst (double)
```

Estas instruções copiam o conteúdo integral do registo fonte para o registo destino.

Não efectuam qualquer tipo de conversão entre tipos de informação.

Instruções de cálculo em Vírgula flutuante no MIPS

Transferência de informação (entre registos da FPU e a memória)

Registo da FPU | Endereço de memória

```
lwc1   FPdst, offset(CPUreg) # Load single from memory
swc1   FPsrc, offset(CPUreg) # Store single into memory
ldc1   FPdst, offset(CPUreg) # Load double from memory
sdc1   FPsrc, offset(CPUreg) # Store double into memory
```

Instruções da máquina virtual (apenas muda a mnemónica):

```
l.s    FPdst, offset(CPUreg) # Load single from memory
s.s    FPsrc, offset(CPUreg) # Store single into memory
l.d    FPdst, offset(CPUreg) # Load double from memory
s.d    FPsrc, offset(CPUreg) # Store double into memory
```


Instruções de cálculo em Vírgula flutuante no MIPS

Manipulação de constantes

- Nas instruções da FPU do MIPS os operandos têm que residir em registos internos, o que significa que **não há suporte para a manipulação directa de constantes**. Como lidar então com operandos que são constantes?
- **Método 1:**
  - Determinar, em tempo de compilação, o valor que codifica a constante (32 bits para precisão simples ou 64 bits para precisão dupla)
  - Carregar essa constante em 1 ou 2 registos do CPU e copiar o(s) seu(s) valor(es) para o(s) registo(s) da FPU
- **Método 2:**
  - Usar as directivas "`.float`" ou "`.double`" para definir em memória o valor da constante: 32 bits (`.float`) ou 64 bits (`.double`)
  - Ler o valor da constante da memória para um registo da FPU usando as instruções de acesso à memória vistas anteriormente (`l.s` ou `l.d`)

Instruções de cálculo em Vírgula flutuante no MIPS

Manipulação de constantes

- O MARS disponibiliza duas instruções virtuais que permitem usar o método 2 (i.e., definição da constante em memória) de forma simplificada. Essas instruções têm o seguinte formato:

```
l.s  $FPdst,label
l.d  $FPdst,label
```

  em que "label" representa o endereço onde a constante está armazenada em memória.

- A decomposição em instruções nativas destas instruções é (admitindo, por exemplo, que a constante K1 está armazenada no endereço 0x1001000C):

```
l.s  $f0,k1
    lui   $1,0x1001
    lwc1 $f0,0x000C($1)
```

```
l.d  $f0,k1
    lui   $1,0x1001
    ldc1 $f0,0x000C($1)
```


Instruções de cálculo em Vírgula flutuante no MIPS

Instruções de decisão

- A tomada de decisões envolvendo quantidades em vírgula flutuante realiza-se de forma distinta da utilizada para o mesmo tipo de operação envolvendo quantidades inteiras
- Para quantidades em vírgula flutuante são necessárias duas instruções em sequência: uma **comparação** das duas quantidades, seguida da **decisão** (que usa a informação produzida pela comparação):
  - A instrução de comparação coloca a **True** ou **False** uma *flag* (1 bit), dependendo de a condição em comparação ser verdadeira ou falsa, respectivamente
  - Em **função do estado dessa *flag*** a instrução de decisão (instrução de salto) pode alterar a sequência de execução

Instruções de comparação em vírgula flutuante - exemplo

```
float a, b;
...

if( a > b)
      a = a + b;
   else
      a = a - b;
```

```
# $f0 ← a
# $f2 ← b
...
if:    c.le.s $f0, $f2          # if(a > b)
       bc1t   else              # {
       add.s  $f0, $f0, $f2     #     a = a + b;
       j      endif             # }
                                # else
else: sub.s  $f0, $f0, $f2     #     a = a – b;
endif:...
```


Instruções de cálculo em Vírgula flutuante no MIPS

- **Instruções de comparação:**

c.x.s	FPUreg1, FPUreg2	# compare single
c.x.d	FPUreg1, FPUreg2	# compare double

Em que **x** pode ser uma das seguintes condições:

EQ	- equal
LT	- less than
LE	- less or equal

- **Instruções de salto:**

bc1t	# branch if true
bc1f	# branch if false

Instruções de cálculo em Vírgula flutuante no MIPS

Convenções quanto à utilização de registos:

Registos para **passar parâmetros** para funções:

- **$f12 ($f13), $f14 ($f15)**

Registos para **devolução de resultados** das funções:

- **$f0 ($f1), $f2 ($f3)**

Registos que **não podem** ser alterados pelas funções:

- **$f20 ($f21) ... $f30 ($f31)**

Registos que **podem** ser alterados pelas funções:

- **$f4 ($f5) ... $f10 ($f11)**
- **$f16 ($f17), $f18 ($f19)**

Exemplo de concretização:

```
float func(float, int);

void main(void)
{
      float res;

      res = func( 12.5E-2, 2 );
      printFloat( res );      // syscall 2
}

float func(float a, int k)
{
      float val;
      if( a >= -5.6)
            val = (float)k * (a - 32.0);
      else
            val = 0.0;
      return val;
}
```

Conversão entre tipos (inteiro para float, neste caso)

Exemplo de concretização:

```
void main(void)
{
        float  res;

        res = func( 12.5E-2, 2 );
        printFloat( res );  // syscall 2
}
```

```
float func(float a, int k)
```

```
        .data
k1:     .float 12.5E-2
k2:     .float -5.6
k3:     .float 32.0
k4:     .float 0.0
        .text
        .globl main
main:   ...                   # void main(void) {
        l.s    $f12, k1       #
        li     $a0, 2         #
        jal    func           #
        mov.s  $f12, $f0      #    res = func(12.5E-2, 2)
        li     $v0, 2         #
        syscall               #    print_float(res)
        ...
        jr     $ra            # }
```


Exemplo de concretização:

```
float func(float a, int k)
{
        float val;
        if( a >= -5.6)
              val = (float)k * (a - 32.0);
        else
              val = 0.0;
        return val;
}
```

```
# $f4 ← val                           # float func(float, int){
func:  l.s      $f4, k2                #   $f4 = -5.6
       c.lt.s   $f12, $f4             #   if( a >= -5.6 )
       bc1t     else                  #   {
       mtc1     $a0, $f0              #     $f0 = k
       cvt.s.w  $f0, $f0              #     $f0 = (float)k
       l.s      $f4, k3               #     val = 32.0
       sub.s    $f4, $f12, $f4        #     val = a - 32.0
       mul.s    $f4, $f0, $f4         #     val = (float)k * val
       j        endif                 #   } else
else:  l.s      $f4, k4               #     val = 0.0
endif: mov.s    $f0, $f4              #   return val;
       jr       $ra                   # }
```