

1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Aulas 14 & 15

Representação de números em vírgula flutuante

Unidade de vírgula flutuante do MIPS

- A norma IEEE 754 (precisão simples)
- Operações aritméticas em vírgula flutuante
- Precisão simples e precisão dupla
- Arredondamentos
- Instruções da FPU do MIPS

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 inteiras e fraccionárias 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:

$$\begin{aligned} & -2.345129876 \times 10^1 \\ & -0.2345129876 \times 10^2 \end{aligned}$$

- São representações da mesma quantidade em que a posição da vírgula tem de ser ponderada, na interpretação numérica da quantidade, pelo valor do expoente de base 10.
- No primeiro exemplo, em que o **número de dígitos diferentes de zero à esquerda da vírgula é igual a um**, diz-se que a representação está **normalizada**.
- Pelo facto de a vírgula poder ser deslocada sem alterar o valor representado chama-se tb. a esta técnica **representação em vírgula flutuante**. Apresenta a vantagem de não desperdiçar espaço de armazenamento com os zeros à esquerda da quantidade representada.

Representação de números em Vírgula flutuante

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

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

(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

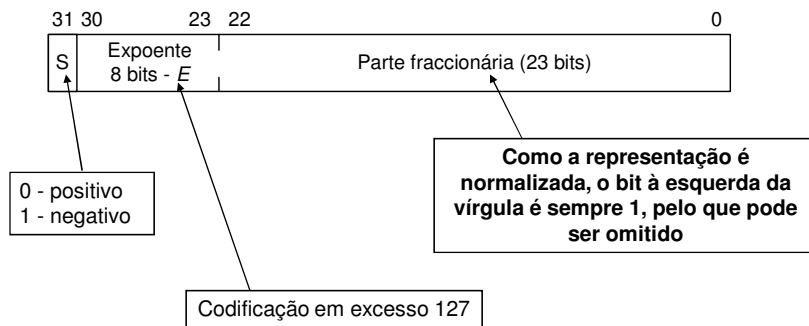
Representação de números em Vírgula flutuante

- O problema da divisão do espaço de armazenamento coloca-se também neste caso, embora se aplique à 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 \Rightarrow maior precisão
 - Aumento do número de bits do expoente \Rightarrow maior gama de representação

Um bom design implica compromissos adequados!

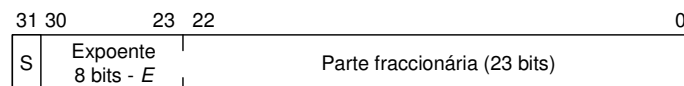
Representação de números em Vírgula flutuante

Norma IEEE 754 (adoptada pelo MIPS)



Representação de números em Vírgula flutuante

Norma IEEE 754 (adoptada pelo MIPS)



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

- O expoente é codificado em excesso 127. Ou seja, é somado ao expoente verdadeiro (Exp) o *offset* 127 para obter o código de representação (i.e. $\text{Exp} = E - 127$)
- 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.**

Arquitectura de Computadores I

2007/08

Representação de números em Vírgula flutuante

Norma IEEE 754 (adoptada pelo MIPS)



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

- A Mantissa pode tomar valores compreendidos entre 1.00000000000000000000000 e 1.11111111111111111111111.
- O bit à esquerda da vírgula é sempre 1 (representação normalizada) pelo que pode ser omitido (*hidden bit*).

Universidade de Aveiro

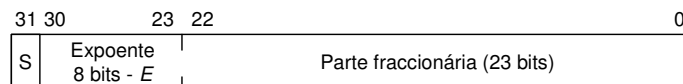
Slide 14&15 - 9

Arquitectura de Computadores I

2007/08

Representação de números em Vírgula flutuante

Norma IEEE 754 (adoptada pelo MIPS)

**Exemplo:** 0 10000010 10100000000000000000000

Sinal = 0 (quantidade positiva)

Expoente = $[130] - \text{offset} = 130 - 127 = 3$ Mantissa = $(1 + \text{parte fraccionária}) = 1 + .101 = 1.101$

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

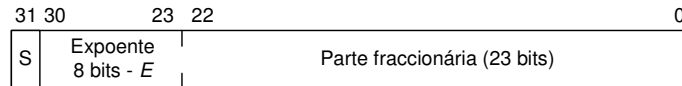
$$= 1.625 \times 8 = 13 \times 10^0 = 13$$

Universidade de Aveiro

Slide 14&15 - 10

Representação de números em Vírgula flutuante

Norma IEEE 754 (adoptada pelo MIPS)



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

A gama de representação suportada por este formato será portanto:

$$[1.00000000000000000000000000000000 \times 2^{-126} \text{ a } 1.11111111111111111111111111111111 \times 2^{+127}]$$

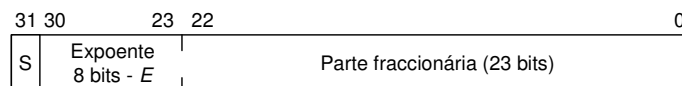
$$[1.1754943 \times 10^{-38} \text{ a } 3.4028235 \times 10^{+38}]$$

Nº de bits / casa decimal = $\log_2(10) \approx 3.3$

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

Representação de números em Vírgula flutuante

Norma IEEE 754 (adoptada pelo MIPS)



- Note-se que, nas operações com números em vírgula flutuante, não só pode existir **overflow** (caso em que o expoente do resultado não cabe no espaço que lhe está destinado $\rightarrow E > 254$), como pode haver igualmente **underflow** (se o expoente for tão pequeno que tb. não é representável $\rightarrow E < 1$)

Overflow se: $N_{\text{resultado}} > 1.11111111111111111111111111111111 \times 2^{+127}$

Underflow se: $0 < N_{\text{resultado}} < 1.00000000000000000000000000000000 \times 2^{-126}$

Arquitectura de Computadores I

2007/08

Representação de números em Vírgula flutuante

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

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 100100110000000000000000

0xC1498000

| | |
|-----|------------|
| | 0.59375 |
| | $\times 2$ |
| MSB | 1.18750 |
| | 0.18750 |
| | $\times 2$ |
| | 0.37500 |
| | 0.37500 |
| | $\times 2$ |
| | 0.75000 |
| | 0.75000 |
| | $\times 2$ |
| | 1.50000 |
| | 0.50000 |
| | $\times 2$ |
| LSB | 1.00000 |

Universidade de Aveiro

Slide 14&15 - 13

Arquitectura de Computadores I

2007/08

Representação de números em Vírgula flutuante

Operações aritméticas em vírgula flutuante – **Adição**

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 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

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

Universidade de Aveiro

Slide 14&15 - 14

Representação de números em Vírgula flutuanteOperações aritméticas em vírgula flutuante – **Multiplificação**Exemplo: $N = 1.11 \times 2^0 * 1.01 \times 2^{-2}$ **1º Passo:** Somar os expoentes

$$\text{Expr} = 0 + (-2) = [0 + 127] + [-2 + 127] = 127 + 125 - 127 = [125] = -2$$

2º Passo: Multiplicar as mantissas

$$M_r = 1.11 * 1.01 = 10.0011$$

3º Passo: Normalizar o resultado

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

Representação de números em Vírgula flutuanteOperações aritméticas em vírgula flutuante – **Divisão**Exemplo: $N = 1.001 \times 2^0 / 1.1 \times 2^{-2}$ **1º Passo:** Subtrair os expoentes

$$\text{Expr} = 0 - (-2) = [0 + 127] - [-2 + 127] = 127 - 125 + 127 = [129] = 2$$

2º Passo: Dividir as mantissas

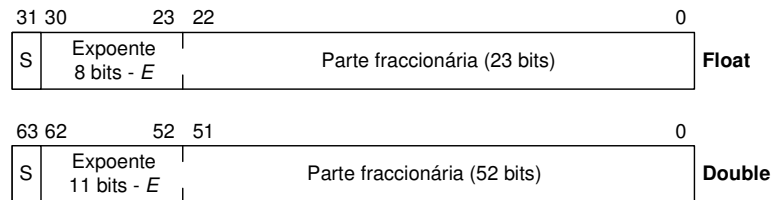
$$M_r = 1.001 / 1.1 = 0.11$$

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)



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

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

Representação de números em Vírgula flutuante

Casos particulares

- A norma IEEE 754 suporta ainda a representação de alguns casos particulares, como sejam:
 - A quantidade zero; essa quantidade não seria representável de acordo com o formato descrito até aqui;
 - +/-infinito;
 - Resultados não numéricos (*NAN – Not a Number*). Um exemplo possível é o resultado de uma divisão de zero por zero;
 - 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

Casos particulares:

| Precisão Simples | | Precisão Dupla | | Representa |
|------------------|-----------------|-----------------|-----------------|--------------------------------|
| Expoente | Frac. | Expoente | Frac. | |
| 0 | 0 | 0 | 0 | 0 |
| 0 | ≤ 0 | 0 | ≤ 0 | Número desnormalizado |
| 1 a 254 | qualquer | 1 a 2046 | qualquer | Nº em vírgula flutuante |
| 255 | 0 | 2047 | 0 | Infinito |
| 255 | ≤ 0 | 2047 | ≤ 0 | NAN (Not a Number) |

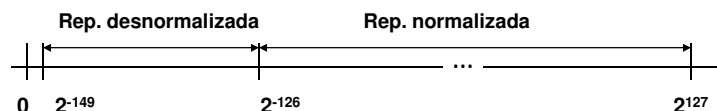
Representação de números em Vírgula flutuante

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:

[illegible]

[1x2⁻²³ *2⁻¹²⁶ a 1.0 x 2⁻¹²⁶]

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

Representação de números em Vírgula flutuante**Arredondamentos**

- As operações aritméticas são efectuadas com um número de dígitos significativos 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 c/ 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 |

$$\begin{aligned}\text{Erro médio} &= (0 - 1/4 - 1/2 - 3/4) / 4 \\ &= -3/8\end{aligned}$$

- 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 c/ 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 |

$$\text{Erro médio} = (0 - 1/4 + 1/2 + 1/4) / 4 \\ = +1/8$$

- Mantém-se a parte inteira, quando o 1º dígito decimal for 0 ou soma-se “1” à parte inteira quando aquele seja “1”.
- 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 p/ o par mais próximo (exemplo c/ 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 |

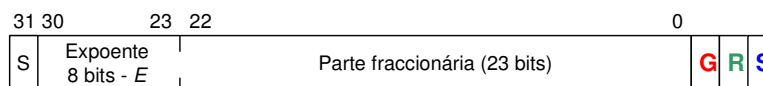
- Semelhante ao arredondamento, mas decidindo, para o caso “**xx.10**” em função do primeiro dígito à esquerda da vírgula.
- Erro médio = $-1/2 + 1/2 = 0$

Representação de números em Vírgula flutuante

- A fim de assegurar a minimização do 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 designados pelas letras G, R e S. Estes bits destinam-se respectivamente a:
 - G – Guard Bit** – Servir de espaço armazenamento para um 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 sua direita. 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)

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

$$\text{Mant}(A/B) = 1.00001 / 1.11111 \quad \text{Exp}(A/B) = 2 - (-1) = 3$$

$$= 0.10000 \mathbf{1}100001 \quad \mathbf{G=1}, \mathbf{R=1}, \mathbf{S = OR(00001) = 1}$$

$$= 0.10000 \mathbf{111}$$

$$\text{Mant}(A/B)_{\text{norm}} = 1.0000 \mathbf{11}$$

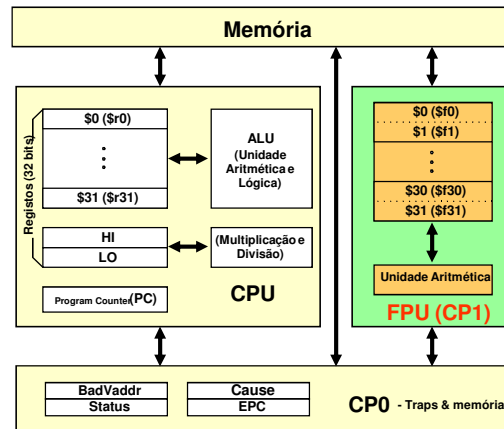
Arredondamento para o par mais próximo $\Rightarrow \text{Mant}(A/B) = 1.00010$

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

Arquitectura de Computadores I

2007/08

Instruções de cálculo em Vírgula flutuante



- 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.

Universidade de Aveiro

Slide 14&15 - 27

Arquitectura de Computadores I

2007/08

Instruções de cálculo em Vírgula flutuante

- Os registos do coprocessador aritmético são designados, no *Assembly* do MIPS, pelas letras **\$fn**, em que o índice **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.

Universidade de Aveiro

Slide 14&15 - 28

Instruções de cálculo em Vírgula flutuante

• Instruções aritméticas

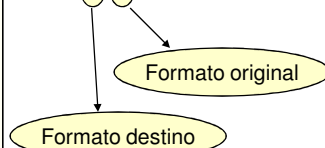
| | | |
|-------|---------------------|------------------|
| abs.p | FPdst,FPsrc | # Absolute Value |
| add.p | FPdst,FPsrc1,FPsrc2 | # Addition |
| div.p | FPdst,FPsrc1,FPsrc2 | # Divide |
| mul.p | FPdst,FPsrc1,FPsrc2 | # Multiply |
| neg.p | FPdst,FPsrc | # Negate |
| sub.p | FPdst,FPsrc1,FPsrc2 |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

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

Instruções de cálculo em Vírgula flutuante

• Instruções de 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 |



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

Arquitectura de Computadores I

2007/08

Instruções de cálculo em Vírgula flutuante

• Instruções de transferência de informação entre CPU e FPU



```

mtc1  CPUSrc, FPdst  # Move to Coprocessor 1
mfcl  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.

Universidade de Aveiro


Slide 14&15 - 31

Arquitectura de Computadores I

2007/08

Instruções de cálculo em Vírgula flutuante

• Instruções de transferência de informação entre FPU e 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:

```

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

```

Universidade de Aveiro

Slide 14&15 - 32

Instruções de cálculo em Vírgula flutuante

Instruções de comparação

- 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.
- A tomada de decisões realiza-se através de duas instruções em sequência: uma instrução de comparação das duas quantidades, seguida de uma instrução de salto que decide em função de informação produzida pela primeira.
- A instrução de comparação coloca a **True** ou **False** uma *flag* (1 bit), dependendo da condição de comparação ser verdadeira ou falsa, respectivamente. A instrução de salto decide, em função dessa *flag*, se deve ou não 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)
      bclt  else              # {
      add.s $f0, $f0, $f2      #     a = a + b;
      j     endif             # }
else: sub.s $f0, $f0, $f2      # else
endif: ...                    #     a = a - b;
```

Arquitectura de Computadores I

2007/08

Instruções de cálculo em Vírgula flutuante

- 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

Universidade de Aveiro

Slide 14&15 - 35

Arquitectura de Computadores I

2007/08

Instruções de cálculo em Vírgula flutuante**Convenções quanto à utilização de registos:**

Registos para passar parâmetros para funções:

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

Registos para devolução de parâmetros para 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)

Universidade de Aveiro

Slide 14&15 - 36

Exemplo de utilização:

```
float func(float, int);

void main(void)
{
    float res;

    res = func( 12.5, 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;
}
```

Exemplo de utilização:

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

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

```
# $f12 ← res
#
main:                                # void main(void) {
    subu    $sp, $sp, 4             #     reserve space in stack
    sw      $ra, 0($sp)             #     push( $ra )
    li.s    $f12, 12.5              #
    li      $a0, 2                  #
    jal     func                    #
    mov.s   $f12, $f0               #     res = func(12.5, 2)
    li      $v0, 2                  #
    syscall                               #     printFloat(res)
    lw      $ra, 0($sp)             #     pop( $ra )
    addiu   $sp, $sp, 4             #     free stack
    jr      $ra                    # }
```

Exemplo de utilização:

```
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: li.s \$f4, -5.6 | # { |
| c.lt.s \$f12, \$f4 | # if(a >= -5.6) |
| bclt else | # { |
| mtcl \$a0, \$f0 | # \$f0 = k |
| cvt.s.w \$f0, \$f0 | # \$f0 = (float)k |
| li.s \$f4, 32.0 | # 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: li.s \$f4, 0.0 | # val = 0.0 |
| endif: mov.s \$f0, \$f4 | # return val; |
| jr \$ra | # } |