

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 e Silva

Representação de números em Vírgula flutuante

A codificação de quantidades numéricas, que até agora estiveram sempre associadas à representação de números inteiros (com ou sem sinal)

A representação de números reais, por outro lado, é de natureza distinta, em particular no que concerne à **gamma de valores representáveis** e à **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 Virgula flutuante

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

$$-2.345129876 \mid \text{101} \mid \text{-(2 \mid 10^0 + 3 \mid 10^{-1} + 4 \mid 10^{-2} + 5 \mid 10^{-3} + 6 \mid 10^{-4} + 7 \mid 10^{-5} + 8 \mid 10^{-6} + 9 \mid 10^{-7}) \mid \text{101}}$$

$$-0.2345129876 \mid \text{102} \mid \text{-(0 \mid 10^0 + 2 \mid 10^{-1} + 3 \mid 10^{-2} + 4 \mid 10^{-3} + 5 \mid 10^{-4} + 6 \mid 10^{-5} + 7 \mid 10^{-6} + 8 \mid 10^{-7} + 9 \mid 10^{-8}) \mid \text{102}}$$

Se a representação do mesmo valor em que a potência de 10 de ser ponderada, na interpretação numérica da expressão, o valor do expoente de base 10

Esta técnica, em que a virgula pode ser deslocada o valor representado, designa-se também **representação em virgula flutuante**

A representação em virgula flutuante tende a desperdiçar espaço de armazenamento com os zeros à esquerda da quantidade representada

No primeiro exemplo, o número de dígitos à esquerda da virgula é igual a um: diz-se **representação está normalizada**

Representação de números em Virgula flutuante

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

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

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

Em que:

f a parte fraccionária representada por **n** bits

1.f a **mantissa** (1 + parte fraccionária)

Exp o **expoente** da potência de base 2 representado por bits

Representação de números em Virgula flutuante

O problema da divisão do espaço de armazenamento 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**

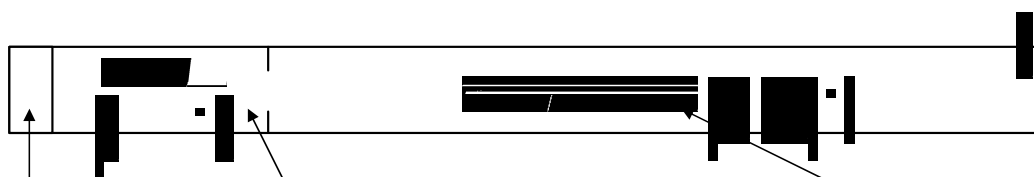
Aumento do número de bits da parte fraccionária \Rightarrow maior precisão

Aumento do número de bits do expoente \Rightarrow maior gama de representação

Um bom *design* implica compromissos adequados!

Representação de números em Virgula flutuante

Norma IEEE 754 (precisão simples)



S: Sinal da quantidade
0 - positivo
1 - negativo

E: Expoente codificado em **excesso 127**

f: Parte fraccionária da mantissa

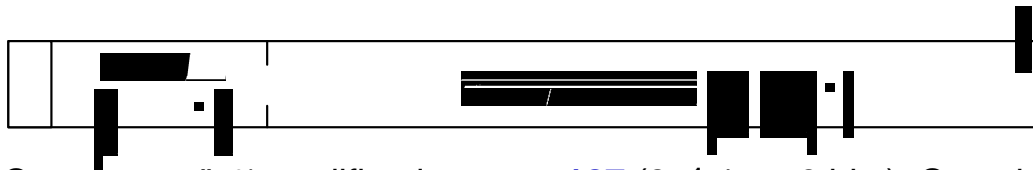
A representação é **normalizada** (o bit à esquerda da vírgula é **sempre 1**). Assim, esse bit é omitido da representação (**hidden bit**)

A Mantissa pode tomar valores compreendidos entre

1.000000000000000000000000 e **1.111111111111111111111111**

Representação de números em Virgula flutuante

Norma IEEE 754 (precisão simples)



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

(i.e. **$Exp = E - 127$**)

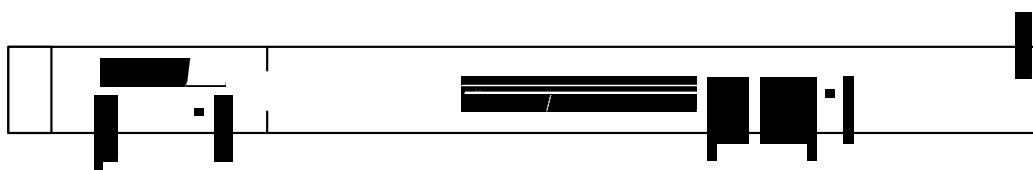
$$N = (-1)^S 1.f | \cdot 2^{Exp} = (-1)^S 1.f | \cdot 2^{E-127}$$

O código 127 representa assim o expoente zero, os maiores do que 127 representam expoentes positivos e os 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 Virgula flutuante

Norma IEEE 754 (precisão simples)



Exemplo: 0 1000010 10100000000000000000000 (0x41500000)

Sinal = 0 (quantidade positiva)

Expoente = [130] offset = 130 - 127 = 3 (Exp = E - offset)

Mantissa = (1 + parte fracionária) = 1.101 = 1.101

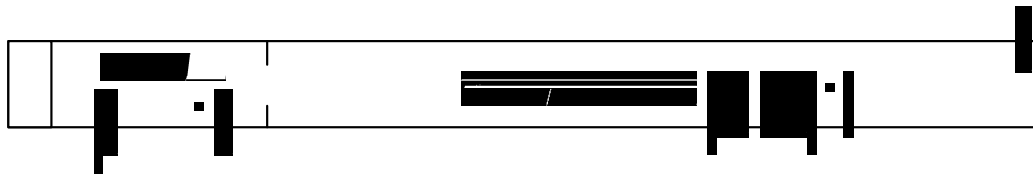
A quantidade representada, Q, será então:

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

$$= 1.625 | \cdot 8 = 13$$

Representação de números em Virgula flutuante

Norma IEEE 754 (precisão simples)



$$N = (-1)^S 1.f |_{2^{Exp}} = (-1)^S 1.f |_{2^{Exp-127}}$$

A gama de representação suportada por este formato portanto:

$[1.00000000000000000000000 |_{2^{126}} \text{ a } 1.11111111111111111111111 |_{2^{127}}]$

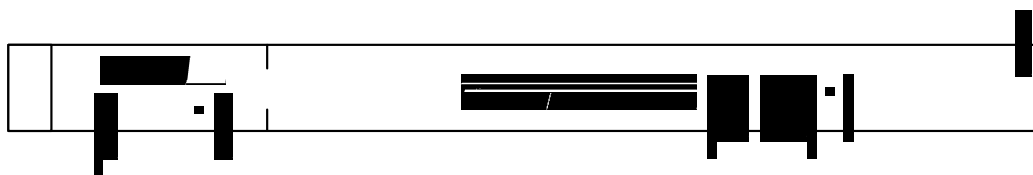
$[1.1754943 |_{10^{-38}} \text{ a } 3.4028235 |_{10^{38}}]$

Nº de bits por 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 Virgula flutuante

Norma IEEE 754 (precisão simples)



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 ($E > 254$)

$$N_{\text{resultado}} > 1.11111111111111111111111 |_{2^{127}}$$

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

$$0 < N_{\text{resultado}} < 1.00000000000000000000000 |_{2^{126}}$$

Representação de números em Virgula flutuante

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

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

Parte fraccionária: $0.59375 = 0.10011_2$

$12.59375_{10} = 1100.10011_2$

Normalização: $1100.10011_2 = 1.10010011_2$

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

1 10000010 100100110000000000000000

0xC1498000

MSB

0.59375
0.18750
0.18750
0.37500
0.37500
0.75000
0.75000
1.50000
0.50000
1.00000

LSB

Representação de números em Virgula flutuante

Operações aritméticas em virgula flutuante (adição/subtração)

Exemplo: $N = 1.11 \times 2^p + 1.00 \times 2^q$

1º Passo Igualar os expoentes ao maior dos expoentes

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

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

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

3º Passo Normalizar o resultado

$N = 10.00 \times 2^p = 1.000 \times 2^{p+1}$

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 os resultados codificados no formato IEEE754, em hexadecimal

$$R1 = A \oplus B$$

$$R2 = B \oplus A$$

$$R3 = A + B$$

$$R4 = B + A$$

Repita o exercício supondo:

$$A = 0xC0C00000$$

$$B = 0x41600000$$

Representação de números em Virgula flutuante

Operações aritméticas em virgula flutuante

Exemplo: $N = (1.11 \mid \cdot 2^0) \mid (1.01 \mid \cdot 2^2)$

1º Passo Somar os expoentes

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

2º Passo Multiplicar as mantissas

$$M_r = 1.11 \mid \cdot 1 \mid 01 = 10.0011$$

3º Passo Normalizar o resultado

$$N = 10.0011 \mid \cdot 2^2 \mid 1.00011 \mid \cdot 2^1 \mid$$

Representação de números em Virgula flutuante

Operações aritméticas em virgula flutuante

Exemplo: $N = (1.001 \cdot 2^0) / (1.1 \cdot 2^2)$

1º Passo Subtrair os expoentes

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

2º Passo Dividir as mantissas

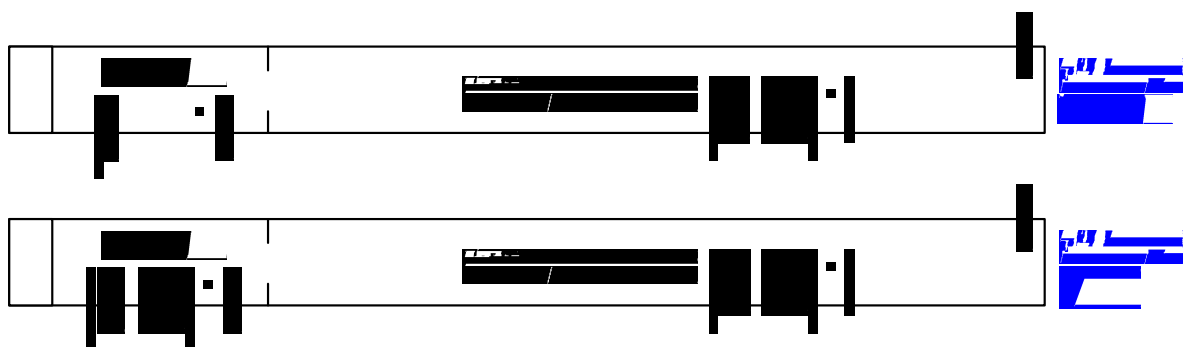
$$M_r = 1.001 / 1.1 = 0.11$$

3º Passo Normalizar o resultado

$$N = 0.11 \cdot 2^2 / 1.1 \cdot 2^1$$

Representação de números em Virgula flutuante

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



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

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

Representação de números em Virgula flutuante

Casos particulares

A norma IEEE 754 suporta ainda a representação de **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 virgula é zero

Representação de números em Virgula flutuante

Representação desnormalizada:

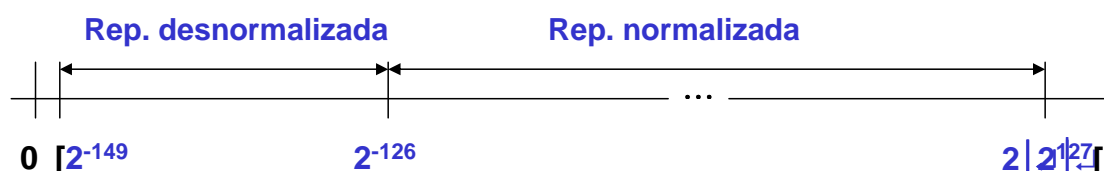
Permite a representação de quantidades cada vez mais pequenas (com cada vez menos precisão) **underflow gradual**

A gama de representação suportada pelo **formato de mantissa desnormalizada** em precisão simples será portanto:

$[0.000000000000000000000001 | 2^{-126}]$ a $0.111111111111111111111111 | 2^{-126}]$

$[1 | 2^{-126}]$ a $1.0 | 2^{-126}]$

$[1,4012985 | 10^{-45}]$ a $1.1754943 | 10^{-38}]$



Representação de números em Virgula flutuante

Casos particulares:

Precisão Simples		Precisão Dupla		Representa
Expoente	Parte Frac.	Expoente	Parte Frac.	
0	0	0	0	0
0	$ \text{0} \text{ } \text{ } $	0	$ \text{0} \text{ } \text{ } $	Número desnormalizado
1 a 254	qualquer	1 a 2046	qualquer	Número em virgula flutuante normalizado
255	0	2047	0	Infinito
255	$ \text{0} \text{ } \text{ } $	2047	$ \text{0} \text{ } \text{ } $	NAN (Not a Number)

Representação de números em Virgula flutuante

Arredondamentos

As operações aritméticas são efectuadas sobre bits da parte fraccionária superior ao disponível no espaço de armazenamento

Desta forma, na conclusão de qualquer operação 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 Virgula 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

$$\text{Erro médio} = (0 + 1/4 + 1/2 + 3/4) / 4 = 3/8$$

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

Representação de números em Virgula 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

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

Mantém-se a parte inteira quando o 1.º dígito decimal for < 0.5 e soma-se 1 à parte inteira quando aquele for ≥ 0.5 (arred(x) = $\text{trunc}(x) + 0.5$)

O erro médio é mais próximo de zero do que na truncatura, mas ligeiramente polarizado do lado positivo

Representação de números em Virgula 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 técnica de arredondamento, mas, neste caso, **x0.10**, em função do primeiro dígito à esquerda da virgula

Erro médio $= -1/8 + 1/8 = 0$

Representação de números em Virgula flutuante

De modo a minimizar o erro introduzido pelo processo de arredondamento, os valores resultantes **de cada fase intermédia** de execução de uma operação aritmética são mantidos em bits suplementares

Estes bits designam-se pelas letras **R** e **S** e destinam-se a:

Guard Bit Bit suplementar que pode ser necessário na pós-normalização da mantissa decorrente das operações de divisão

Round off bit Bit usado na operação de arredondamento

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)

Instruções de cálculo em Virgula flutuante no MIPS

Os registos do coprocessador aritmético são designados em *Assembly* do MIPS, pelas letras **\$fn**, em que o índice **n** toma valores entre 0 e 31

Cada par de registos consecutivos **[\$n, \$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 Virgula flutuante no MIPS

Aritméticas

<code>abs.p</code>	<code>FPdst,FPsrc</code>	# Absolute Value
<code>neg.p</code>	<code>FPdst,FPsrc</code>	# Negate
<code>div.p</code>	<code>FPdst,FPsrc1,FPsrc2</code>	# Divide
<code>mul.p</code>	<code>FPdst,FPsrc1,FPsrc2</code>	# Multiply
<code>add.p</code>	<code>FPdst,FPsrc1,FPsrc2</code>	# Addition
<code>sub.p</code>	<code>FPdst,FPsrc1,FPsrc2</code>	# 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 Virgula flutuante no MI

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 do
resultado

Formato original

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 Virgula flutuante no MI

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 Virgula 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
sdcl	FPsrc,	offset(CPUreg)	# Store double into memory

Instruções da máquina (apenas muda a mnemonica)

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 Virgula flutuante no MIPS

Manipulação de constantes

Nas instruções da FPU do MIPS os operandos são apenas 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 da constante (32 bits para precisão simples ou 64 bits para precisão dupla)
- Carregar essa constante em 1 ou 2 registos do CPU. 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 FPU usando as instruções de acesso à memória vistas anteriormente

Instruções de cálculo em Virgula flutuante no MIPS

Manipulação de constantes

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

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

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

em que `label` representa o endereço onde a constante é armazenada em memória.

A decomposição em instruções nativas descreve (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 Virgula flutuante no MIPS

Instruções de decisão

A tomada de decisões envolvendo quantidades em virgula flutuante realiza-se de forma distinta da utilizada para o mesmo tipo de operações envolvendo quantidades inteiras

Para quantidades em virgula flutuante são necessárias instruções em sequência: uma **comparação** das duas quantidades, seguida da **decisão** (que usa a informação produzida pela comparação):

A instrução de comparação **True or False** produz uma **flag** (1 bit), dependendo de a condição em comparação se **verdadeira** ou **falsa** respectivamente

Em **função do estado da flag**, a instrução de decisão (instrução de salto) pode alterar a sequência de execução

Instruções de comparação em virgula 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 Virgula 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 Virgula flutuante no MIPS

Convenções quanto à utilização de registos:

Registos para **passar parâmetros** para funções:

~~\$12~~ \$12 (\$f13), \$f14 (\$f15)

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

~~\$10~~ \$10 (\$f1), \$f2 (\$f3)

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

~~\$20~~ \$20 (\$f21) ... \$f30 (\$f31)

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

~~\$4~~ \$4 (\$f5) ... \$f10 (\$f11)

~~\$16~~ \$16 (\$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 concretizaçãõ:

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

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

```
.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 concretizaçãõ:

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