Arquitectura de Sistemas e Computadores I

Semana #9

Miguel Barão
mjsb@di.uevora.pt

ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Resumo

- Vírgula flutuante e standard IEEE754
- Precisão simples e dupla (float e double)
- +Inf, -Inf
- +0.0, -0.0?
- NaN, sNaN
- Representação de números decimais...
- Overflow, Underflow e underflow progressivo
- Alguns problemas numéricos
- Vírgula flutuante no processador MIPS: Coprocessador 1

Vírgula flutuante consiste na representação de números no formato

$$\text{significando} \times \text{base}^{\text{expoente}}$$

onde

- significando é um número $1 \leq x < 10$.
- expoente é um número inteiro.
- base é tipicamente 10, 2 ou 16.

Vírgula flutuante consiste na representação de números no formato

$$\text{significando} \times \text{base}^{\text{expoente}}$$

onde

- significando é um número $1 \leq x < 10$.
- expoente é um número inteiro.
- base é tipicamente 10, 2 ou 16.

Exemplos de números representados em vírgula flutuante usando a base decimal:

- $3.14159265359 \longrightarrow 3.14159265359 \times 10^0$
- $-0.0012345 \longrightarrow -1.2345 \times 10^{-3}$
- $1024 \longrightarrow 1.024 \times 10^3$

Vírgula flutuante

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\mathsf{bin}} = 1.0_{\mathsf{bin}} \times 2^{-2}$

Vírgula flutuante

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\mathsf{bin}} = 1.0_{\mathsf{bin}} \times 2^{-2}$
- $0.125 =$

Vírgula flutuante

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} =$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 =$

Vírgula flutuante

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} =$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 =$

Vírgula flutuante

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 =$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 =$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.00011001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} =$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\mathrm{bin}} = 1.0_{\mathrm{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\mathrm{bin}} = 1.0_{\mathrm{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\mathrm{bin}} = 1.1_{\mathrm{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\mathrm{bin}} = 1.0_{\mathrm{bin}} \times 2^{10}$
- $3.25 = 11.01_{\mathrm{bin}} = 1.101_{\mathrm{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.00011001100110011\ldots_{\mathrm{bin}}}_{\text{dízima infinita}} = 1.1001100110011\ldots_{\mathrm{bin}} \times 2^{-4}$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.00011001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.0011001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.0001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.0011001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$
- $0.3 = 0.0100110011001100\ldots_{\text{bin}} = 1.0011001100110\ldots_{\text{bin}} \times 2^{-2}$

$\Rightarrow$ Não é possível representar números decimais de forma exacta!

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.00011001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.0011001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$
- $0.3 = 0.0100110011001100\ldots_{\text{bin}} = 1.0011001100110\ldots_{\text{bin}} \times 2^{-2}$

$\Rightarrow$ Não é possível representar números decimais de forma exacta!

- Há (muitos) números com dízima finita em decimal que têm dízima infinita em binário!

Vírgula flutuante usando base binária:

- $0.25 = 0.01_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-2}$
- $0.125 = 0.001_{\text{bin}} = 1.0_{\text{bin}} \times 2^{-3}$
- $0.75 = 0.11_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-1}$
- $1024 = 10000000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{10}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.00011001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.0011001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$
- $0.3 = 0.0100110011001100\ldots_{\text{bin}} = 1.0011001100110\ldots_{\text{bin}} \times 2^{-2}$

$\Rightarrow$ Não é possível representar números decimais de forma exacta!

- Há (muitos) números com dízima finita em decimal que têm dízima infinita em binário!
- Apenas os números que são somas de potências de 2 têm dízima finita em binário.

O standard IEEE754 especifica o formato com que os números em vírgula flutuante são codificados:

- **float** precisão simples: 32 bits
- **double** precisão dupla: 64 bits

O standard IEEE754 especifica o formato com que os números em vírgula flutuante são codificados:

- **float** precisão simples: 32 bits
- **double** precisão dupla: 64 bits

Além dos números em vírgula flutuante são ainda definidos os símbolos:

- $\pm$**Inf** para representar infinito (e.g. consequência de overflow).
- **NaN** not-a-number para representar resultados indefinidos.
- $\pm$**0.0** zero é um número especial, como se irá ver.

Precisão simples: float

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (8 bits)	significando (23 bits)

32 bits

Expoente:

- entre 00000001_{bin} e 11111110_{bin}, i.e. de 1 a 254 decimal.
- expoente é biased: 2^0 é codificado com $127 = 01111111_{\text{bin}}$.

Significando:

- Na realidade são 24 bits (um dos bits está escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (8 bits)	significando (23 bits)

32 bits

Expoente:

- entre 00000001_{bin} e 11111110_{bin}, i.e. de 1 a 254 decimal.
- expoente é biased: 2^0 é codificado com $127 = 01111111_{\text{bin}}$.

Significando:

- Na realidade são 24 bits (um dos bits está escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

0 01111111 00000000000000000000000 $\quad = +1.0 \times 2^0$

Precisão simples: float

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (8 bits)	significando (23 bits)

32 bits

Expoente:

- entre 00000001_{bin} e 11111110_{bin}, i.e. de 1 a 254 decimal.
- expoente é biased: 2^0 é codificado com $127 = 01111111_{\text{bin}}$.

Significando:

- Na realidade são 24 bits (um dos bits está escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

---

0 01111111 00000000000000000000000 $= +1.0 \times 2^0$
1 01111111 01000000000000000000000 $= -1.01 \times 2^0$

Precisão simples: float

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (8 bits)	significando (23 bits)

32 bits

Expoente:

- entre 00000001_{bin} e 11111110_{bin}, i.e. de 1 a 254 decimal.
- expoente é biased: 2^0 é codificado com $127 = 01111111_{\text{bin}}$.

Significando:

- Na realidade são 24 bits (um dos bits está escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

---

0 01111111 00000000000000000000000	$= +1.0 \times 2^0$
1 01111111 01000000000000000000000	$= -1.01 \times 2^0$
0 01111101 00110011001100110011001	$= 1.00110011001100110011001 \times 2^{-2}$
	≈ 0.3

Precisão simples: float

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (8 bits)	significando (23 bits)

32 bits

Expoente:

- entre 00000001_{bin} e 11111110_{bin}, i.e. de 1 a 254 decimal.
- expoente é biased: 2^0 é codificado com $127 = 01111111_{\text{bin}}$.

Significando:

- Na realidade são 24 bits (um dos bits está escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

0 01111111 00000000000000000000000	$= +1.0 \times 2^0$
1 01111111 01000000000000000000000	$= -1.01 \times 2^0$
0 01111101 00110011001100110011001	$= 1.00110011001100110011001 \times 2^{-2}$
	≈ 0.3
1 10000000 11000000000000000000000	$= -1.11 \times 2^1 = -3.5$

Precisão simples: float

Codificação de números em vírgula flutuante:

± (1 bit)	expoente (8 bits)	significando (23 bits)

32 bits

Expoente:

- entre 00000001_{bin} e 11111110_{bin}, i.e. de 1 a 254 decimal.
- expoente é biased: 2^0 é codificado com $127 = 01111111_{\text{bin}}$.

Significando:

- Na realidade são 24 bits (um dos bits está escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

0 01111111 00000000000000000000000	$= +1.0 \times 2^0$
1 01111111 01000000000000000000000	$= -1.01 \times 2^0$
0 01111101 00110011001100110011001	$= 1.00110011001100110011001 \times 2^{-2}$
	≈ 0.3
1 10000000 11000000000000000000000	$= -1.11 \times 2^1 = -3.5$
0 10000000 10010010000111111011011	≈ 3.14159265358979

O infinito ocorre em várias situações. Exemplos:

- Uma operação resulta em overflow.
- Divisão por zero: $1.0/0.0 = \mathit{Inf}$ ou $-1.0/0.0 = -\mathit{Inf}$.
- Funções podem definir esse resultado: $\log(0.0) = -\mathit{Inf}$.

Formato:

0 11111111 00000000000000000000000	$+\mathit{Inf}$
1 11111111 00000000000000000000000	$-\mathit{Inf}$

Algumas propriedades:

- $x + \mathit{Inf} = \mathit{Inf}$, onde x é um número finito.
- $x/\mathit{Inf} = 0.0$, se $x \neq \pm\infty, \mathit{NaN}$.
- $\exp(-\mathit{Inf}) = 0.0$
- $1/-0.0 = -\mathit{Inf}$

Not-a-Number: NaN

O Not-a-Number ocorre em várias situações. Exemplos:

- $0.0/0.0 = NaN$
- $0.0 * Inf = NaN$
- $Inf - Inf = NaN$.

Formato:

0	11111111	10000000000000000000000	$+NaN$
1	11111111	10000000000000000000000	$-NaN$

Algumas propriedades:

- Os Not-a-Number tem tendência a propagar-se e contaminar os resultados seguintes.
- $x + NaN = NaN$.
- A comparação $NaN = NaN$ é FALSA.
- A comparação $NaN \neq NaN$ é VERDADEIRA.
- $1^{NaN} == 1$.

Zero: 0.0

Existem dois zeros:

- $+0.0$
- -0.0

Formato:

0	00000000	00000000000000000000000	$+0.0$
1	00000000	00000000000000000000000	-0.0

Algumas propriedades:

- O zero não é um número em vírgula flutuante regular.
- Não se considera o bit escondido no significando.
- A comparação $0 = -0$ é VERDADEIRA.
- $1/0 = +\mathit{Inf}$ enquanto que $1/(-0) = -\mathit{Inf}$.
- $0^0 = 1$.

Precisão dupla: double

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (11 bits)	significando (52 bits)

64 bits

Expoente:

- entre 00000000001_{bin} e 11111111110_{bin}, i.e. de 1 a 2046.
- expoente é *biased*: 2^0 é codificado com $1023 = 01111111111_{\text{bin}}$.

Significando:

- Na realidade são 53 bits (há 1 bit escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

Codificação de números em vírgula flutuante:

$\pm$ (1 bit)	expoente (11 bits)	significando (52 bits)

64 bits

Expoente:

- entre 00000000001_{bin} e 11111111110_{bin}, i.e. de 1 a 2046.
- expoente é *biased*: 2^0 é codificado com $1023 = 01111111111_{\text{bin}}$.

Significando:

- Na realidade são 53 bits (há 1 bit escondido).
- O primeiro bit é sempre 1 e não é representado (é o bit escondido).

0	01111111111	0000000000000000000000000000000000000000000000000000
0	01111111111	0100000000000000000000000000000000000000000000000000
0	01111111101	0011001100110011001100110011001100110011001100110011
1	10000000000	1100000000000000000000000000000000000000000000000000
0	10000000000	1001001000011111101101010100010001000010110100011000

Que números são estes?

Overflow ocorre quando o expoente atinge 128 em precisão simples, ou 1024 em precisão dupla. Nesta situação deixa de ser possível representar o número em vírgula flutuante, passando o resultado a ser representado por $\pm Inf$.

Underflow ocorre quando o expoente atinge -127 em precisão simples, ou -1023 em precisão dupla, e o significando é nulo. Nesta situação deixa de ser possível representar o número em vírgula flutuante, passando o resultado a ser representado por ± 0.0.

Underflow progressivo é uma condição que ocorre quando o expoente atinge -127 em precisão simples ou -1023 em precisão dupla, e o significando ainda não é nulo. Nesta situação o número deixa de ser um número de vírgula flutuante normalizado, passando a ser um número não normalizado. Estes números não têm o bit escondido 1 à esquerda da vírgula. Perdem progressivamente precisão à medida que se aproximam de um underflow.

Problemas numéricos

1. Calcule $(0.1 + 0.2) + 0.3$
2. Calcule $0.1 + (0.2 + 0.3)$

1. Calcule $(0.1 + 0.2) + 0.3$
2. Calcule $0.1 + (0.2 + 0.3)$

Resultado em precisão dupla:

0 01111111110 0011001100110011001100110011001100110011001100110100
0 01111111110 0011001100110011001100110011001100110011001100110011

(processador Intel Core2 Duo)

- Em matemática, a adição é associativa?

1. Calcule $(0.1 + 0.2) + 0.3$
2. Calcule $0.1 + (0.2 + 0.3)$

Resultado em precisão dupla:

0 01111111110 0011001100110011001100110011001100110011001100110100
0 01111111110 0011001100110011001100110011001100110011001100110011

(processador Intel Core2 Duo)

- Em matemática, a adição é associativa? Sim
- Em vírgula flutuante, a adição é associativa?

Problemas numéricos

1. Calcule $(0.1 + 0.2) + 0.3$
2. Calcule $0.1 + (0.2 + 0.3)$

Resultado em precisão dupla:

0 01111111110 0011001100110011001100110011001100110011001100110100
0 01111111110 0011001100110011001100110011001100110011001100110011

(processador Intel Core2 Duo)

- Em matemática, a adição é associativa? Sim
- Em vírgula flutuante, a adição é associativa? Não $\rightarrow$ *Trouble*...

Sabemos que

$$x^2 - y^2 = (x+y)(x-y)$$

Mas... será que ainda é verdade num computador?

Suponhamos que x e y são números próximos, sendo diferentes apenas nos algarismos menos significativos.

Por exemplo, $x \approx \pi, \quad y \approx \pi - 2^{-50}$.

0 10000000000 1001001000011111101101010100010001000010110100011000
0 10000000000 1001001000011111101101010100010001000010110100010110

Fazendo as contas no processador Intel Core2 Duo:

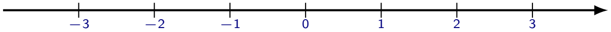
$x^2 - y^2 \approx 5.32907051820075e - 15$
0 01111001111 1000000000000000000000000000000000000000000000000000
$(x-y)(x+y) \approx 5.58058959681381e - 15$
0 01111001111 1001001000011111101101010100010001000010110100010111

Apenas 3 bits correctos em 52 bits do significando !!!

Consequências

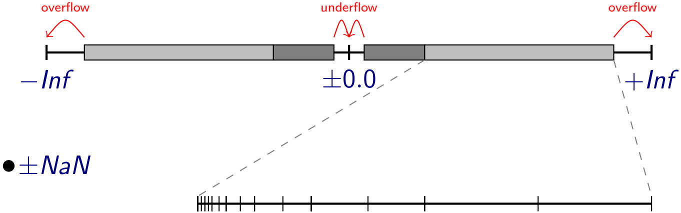
- Se um humano faz as contas em decimal, e o processador faz em binário obtêm-se, em geral, resultados diferentes.
- Crítico por exemplo em cálculo financeiro onde ocorrem milhões de operações todos os dias. Resultado tem de ser consistente com contas feitas com lápis e papel: não usar vírgula flutuante neste contexto.
- Em computação científica o factor humano não é crítico, mas a precisão do cálculo sim. Não se pode “ir a Marte” cometendo erros como os anteriores. É necessário desenvolver e implementar algoritmos que sejam pouco sensíveis a erros.
- Em certas situações, podem implementar-se funções para efectuar cálculos em base decimal.

- Inteiros:



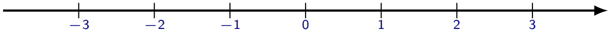
A distância entre os números inteiros é constante em toda a escala.

- Vírgula flutuante:



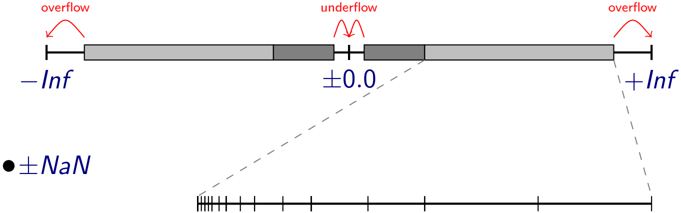
Distância entre números

- Inteiros:



A distância entre os números inteiros é constante em toda a escala.

- Vírgula flutuante:



A resolução de um número em vírgula flutuante varia com o expoente.

Vírgula flutuante no processador MIPS

- O processdor MIPS inclui, opcionalmente, um coprocessador especializado em cálculo de vírgula flutuante: CP1.

Vírgula flutuante no processador MIPS

- O processdor MIPS inclui, opcionalmente, um coprocessador especializado em cálculo de vírgula flutuante: CP1.
- Consiste num conjunto adicional de registos `$f0`–`$f31` de 32 bits e num conjunto adicional de instruções.

Vírgula flutuante no processador MIPS

- O processdor MIPS inclui, opcionalmente, um coprocessador especializado em cálculo de vírgula flutuante: CP1.
- Consiste num conjunto adicional de registos `$f0`–`$f31` de 32 bits e num conjunto adicional de instruções.
- Os números em precisão simples são armazenados nestes registos.

Vírgula flutuante no processador MIPS

- O processdor MIPS inclui, opcionalmente, um coprocessador especializado em cálculo de vírgula flutuante: CP1.
- Consiste num conjunto adicional de registos `$f0`–`$f31` de 32 bits e num conjunto adicional de instruções.
- Os números em precisão simples são armazenados nestes registos.
- Os números em precisão dupla (64 bits) são armazenados em pares consecutivos de registos: `$f0-$f1, $f2-$f3, $f4-$f5`, etc.

Vírgula flutuante no processador MIPS

- O processdor MIPS inclui, opcionalmente, um coprocessador especializado em cálculo de vírgula flutuante: CP1.
- Consiste num conjunto adicional de registos `$f0-$f31` de 32 bits e num conjunto adicional de instruções.
- Os números em precisão simples são armazenados nestes registos.
- Os números em precisão dupla (64 bits) são armazenados em pares consecutivos de registos: `$f0-$f1`, `$f2-$f3`, `$f4-$f5`, etc.
- Exemplos de instruções:
  - `add.d $f2, $f4, $f6 # soma em precisão dupla`
  - `add.s $f0, $f1, $f2 # soma em precisão simples`
  - Outras instruções:
    `sub.s`, `sub.d`, `mul.s`, `mul.d`, `div.s`, `div.d`, `sqrt.s`, `sqrt.d`, `abs.s`, `abs.d`, `round.w.s`, `round.w.s`, etc.

Exemplo:

```
li $t0, 0x40000000
li $t1, 0x40400000

mtc1 $t0,$f0
mtc1 $t1,$f1

mul.s $f2, $f0, $f1
```

Qual o resultado?