Arquitectura de Computadores I

Vírgula flutuante - standard IEEE754

Miguel Barão

ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Resumo

Vírgula fixa

Vírgula flutuante

Formato binário IEEE754

Precisões simples e dupla

Overflow e underflow

Problemas numéricos no cálculo em vírgula flutuante

Vírgula flutuante em RISC-V

Revisão: Conversão de números fraccionários para binário

$$14.375 = 14 + 0.375 =$$

Parte inteira converte-se com divisões por 2:

14	
7	0
3	1
1	1
0	1

$$14_{\text{dec}} = 1110_{\text{bin}}$$

Revisão: Conversão de números fraccionários para binário

$$14.375 = 14 + 0.375 = 1110.011\bar{0}_{\text{bin}}$$

Parte inteira converte-se com divisões por 2:

14	
7	0
3	1
1	1
0	1

$$14_{\text{dec}} = 1110_{\text{bin}}$$

Parte fraccionária convete-se com multiplicações por 2:

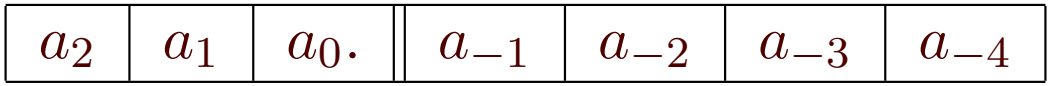
0.	375
0.	75
1.	5
1.	0
0.	0 (rep)

$$0.625_{\text{dec}} = 0.011\bar{0}_{\text{bin}}$$

Vírgula fixa

Vírgula fixa

Vírgula fixa representa números com número fixo de algarismos à esquerda e direita da vírgula



Simples, pode ser implementada recorrendo a números inteiros.

Vírgula fixa

Vírgula fixa representa números com número fixo de algarismos à esquerda e direita da vírgula

a_2	a_1	$a_0.$	a_{-1}	a_{-2}	a_{-3}	a_{-4}

Simples, pode ser implementada recorrendo a números inteiros.

Por exemplo, com 2 casas decimais

$$\begin{aligned} 12.30 \times 6.02 &= (1230 \times 10^{-2}) \times (602 \times 10^{-2}) \\ &= 740460 \times 10^{-4} \\ &= 74.04\cancel{60} \\ &\approx 74.05 \end{aligned}$$

Tipicamente, usa-se vírgula fixa em base decimal para representar valores monetários.

Vírgula flutuante

Vírgula flutuante

Vírgula flutuante consiste na representação de números no formato

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

onde

- base é normalmente 10 ou 2.
- expoente é um número inteiro.
- significando é um número $1 \leq x < \text{base}$.

Vírgula flutuante

Vírgula flutuante consiste na representação de números no formato

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

onde

- base é normalmente 10 ou 2.
- expoente é um número inteiro.
- significando é um número $1 \leq x < \text{base}$.

Exemplos de vírgula flutuante em 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 em binário

- $0.375 =$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} =$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 =$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 =$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 =$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.0001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} =$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.0001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.100110011\ldots_{\text{bin}} \times 2^{-4}$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.0001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.0001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$
- $0.3 = 0.010011001100\ldots_{\text{bin}} = 1.0011001100110\ldots_{\text{bin}} \times 2^{-2}$

A maioria dos números não têm representação exacta!

- Há muitos números com dízima finita em decimal, mas com dízima infinita em binário!

Vírgula flutuante em binário

- $0.375 = 0.011_{\text{bin}} = 1.1_{\text{bin}} \times 2^{-2}$
- $128 = 10000000_{\text{bin}} = 1.0_{\text{bin}} \times 2^{7}$
- $3.25 = 11.01_{\text{bin}} = 1.101_{\text{bin}} \times 2^{1}$
- $0.1 = \underbrace{0.0001100110011\ldots_{\text{bin}}}_{\text{dízima infinita}} = 1.100110011\ldots_{\text{bin}} \times 2^{-4}$
- $0.2 = 0.001100110011\ldots_{\text{bin}} = 1.1001100110011\ldots_{\text{bin}} \times 2^{-3}$
- $0.3 = 0.010011001100\ldots_{\text{bin}} = 1.0011001100110\ldots_{\text{bin}} \times 2^{-2}$

A maioria dos números não têm representação exacta!

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

Formato binário IEEE754

IEEE754

O standard IEEE754 especifica como os números em vírgula flutuante são codificados:

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

IEEE754

O standard IEEE754 especifica como os números em vírgula flutuante são codificados:

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

Adicionalmente são definidos os símbolos:

$\pm$Inf	representa infinito, e.g. devido a overflow.
NaN	not-a-number representa resultados indefinidos.
± 0.0	zeros são números especiais.

Precisões simples e dupla

Precisão simples: float (32 bits)

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

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).

Precisão simples: float (32 bits)

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

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` $= +1.0 \times 2^0$

Precisão simples: float (32 bits)

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

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 (32 bits)

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

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` $= 1.00110011001100110\ldots \times 2^{-2} \approx 0.3$

Precisão simples: float (32 bits)

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

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` $= 1.00110011001100110\ldots \times 2^{-2} \approx 0.3$
`1 10000000 11000000000000000000000` $= -1.11 \times 2^1 = -3.5$

Precisão simples: float (32 bits)

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

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`	$= 1.00110011001100110\ldots \times 2^{-2} \approx 0.3$
`1 10000000 11000000000000000000000`	$= -1.11 \times 2^1 = -3.5$
`0 10000000 10010010000111111011011`	≈ 3.14159265358979

Infinito: Inf

O infinito é gerado em várias situações:

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

Formato:

`0 11111111 00000000000000000000000`	$+\text{Inf}$
`1 11111111 00000000000000000000000`	$-\text{Inf}$

Algumas propriedades:

- $x + \text{Inf} = \text{Inf}$, onde x é um número finito.
- $x/\text{Inf} = 0.0$, se $x \neq \pm\text{Inf}, \text{NaN}$.
- $\exp(-\text{Inf}) = 0.0$
- $1.0/-0.0 = -\text{Inf}$

Not-a-Number: NaN

O Not-a-Number é gerado em várias situações:

- $0.0/0.0 = \text{NaN}$
- $0.0 * \text{Inf} = \text{NaN}$
- $\text{Inf} - \text{Inf} = \text{NaN}$.

Formato:

`0 11111111 10000000000000000000000`	$+\text{NaN}$
`1 11111111 10000000000000000000000`	$-\text{NaN}$

Algumas propriedades:

- Os Not-a-Number tem tendência a propagar-se e contaminar os resultados seguintes.
- $x + \text{NaN} = \text{NaN}$.
- A comparação $\text{NaN} = \text{NaN}$ é *FALSA*.
- A comparação $\text{NaN} \neq \text{NaN}$ é *VERDADEIRA*.
- $1.0^{\text{NaN}} == 1.0$.

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 $0.0 = -0.0$ é *VERDADEIRA*.
- $1.0/0.0 = +\text{Inf}$ enquanto que $1.0/(-0.0) = -\text{Inf}$.
- $0.0^{0.0} = 1.0$.

Precisão dupla: double (64 bits)

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

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).

Precisão dupla: double (64 bits)

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

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 1001001000011111101101010001000100001011010011000
```

Que números são estes?

Overflow e underflow

Overflow e underflow

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 ocorre quando o expoente atinge -127 ou -1023, em precisão simples ou dupla, e o significando ainda não é nulo. Nesta situação, o número deixa de ser normal, passando a ser um número não normalizado ou subnormal. Estes números perdem progressivamente precisão à medida que se aproximam de um underflow.

Problemas numéricos no cálculo em vírgula flutuante

Problemas numéricos

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

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
```

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
```

A adição é associativa?

- Em matemática Sim
- Em vírgula flutuante Não...

Caso notável:

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

Será válido em vírgula flutuante?

Suponhamos que x e y são números próximos. Por exemplo,

$$x = 0.9999, \quad y = 0.9998$$

obtemos

$$x^2 - y^2 \approx 0.0001999700000000 4928$$
$$(x - y)(x + y) \approx 0.0001999699999999 78$$

valor correcto: 0.00019997

Problemas numéricos na vírgula flutuante

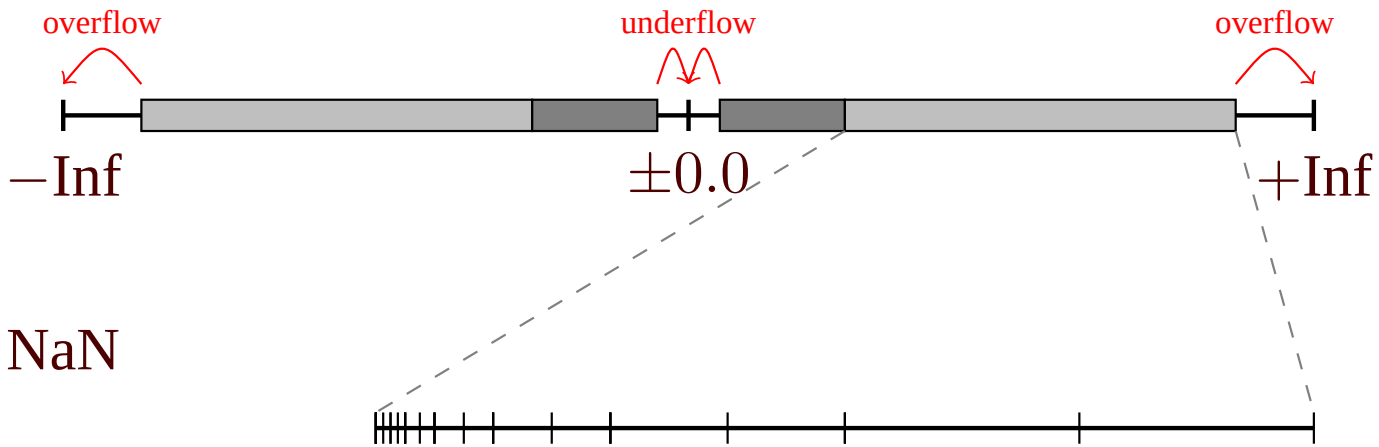
- Operações em decimal e binário dão resultados diferentes.
- Conversão para binário introduz erros.
- Propriedades das operações (e.g. associatividade) não são satisfeitas.
- Erros acumulam-se à medida que se realizam mais operações.
- Não usar vírgula flutuante para valores monetários (se necessário, implementar cálculos decimais em software em vez de usar a vírgula flutuante do cpu.)

Distância entre números

- Inteiros:

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

- Vírgula flutuante:



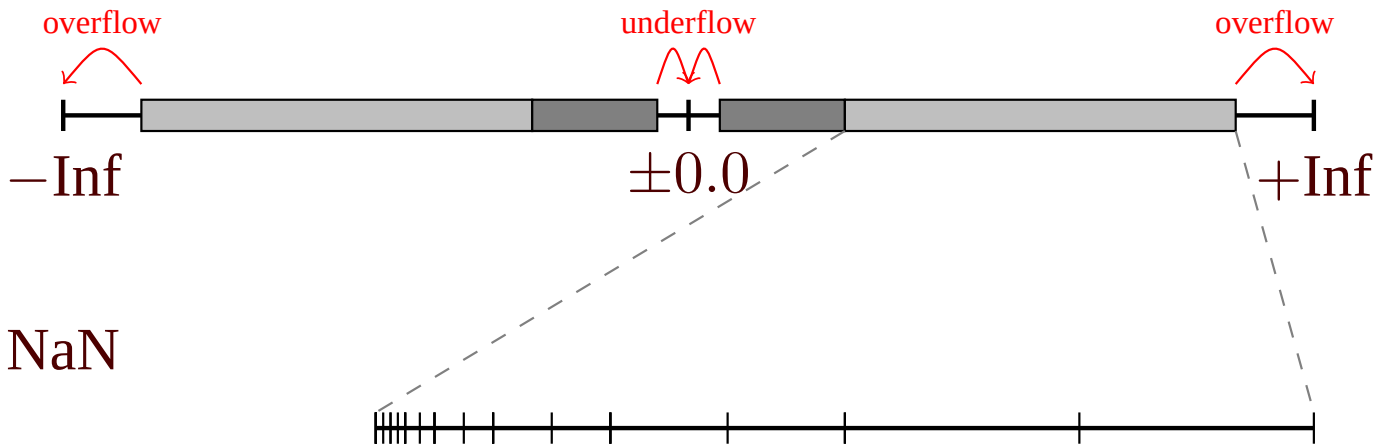
- NaN

Distância entre números

- Inteiros:

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

- Vírgula flutuante:



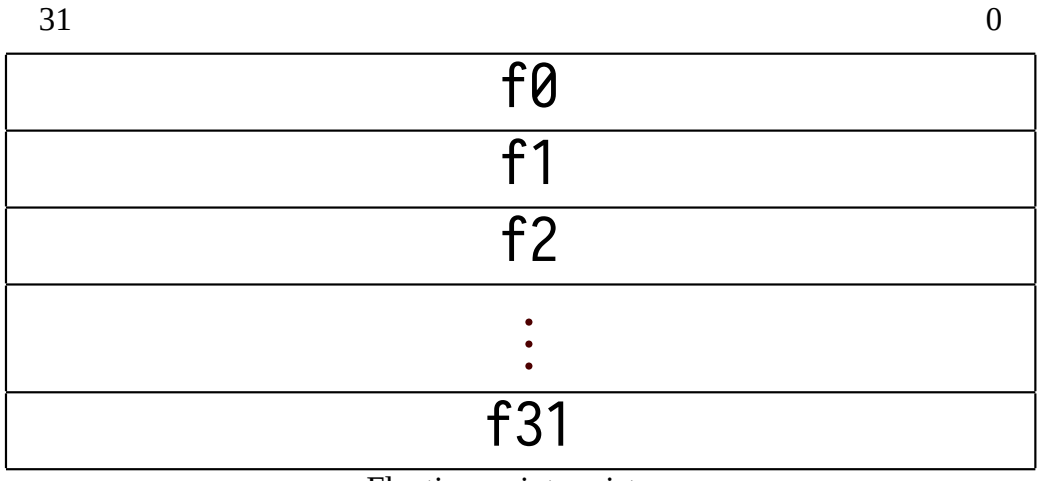
- NaN

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

Vírgula flutuante em RISC-V

Vírgula flutuante na arquitectura RISC-V

- O suporte de vírgula flutuante é opcional na arquitectura RISC-V.
- A extensão F adiciona registos específicos para vírgula flutuante:
  - 32 registos de 32 bits `f0-f31`.
  - 1 registo `fcsr` (*Floating-point Control and Status Register*).



Floating-point registers

Reserved	Rounding Mode	NV	DZ	OF	UF	NX

Floating-point Control and Status Register

Instruções de vírgula flutuante RISC-V

São adicionadas várias instruções novas:

```
flw f0, 0(t0)           # float load word
fsw f0, 4(t0)           # float store word

fadd.s  f2, f0, f1      # soma em vírgula flutuante
fsub.s  f2, f0, f1      # subtracção
fmul.s  f2, f0, f1      # multiplicação
fdiv.s  f2, f0, f1      # divisão
fmin.s  f2, f0, f1      # menor de dois números
fmax.s  f2, f0, f1      # maior de dois números
fsqrt.s f2, f0          # raíz quadrada

fmadd.s f3, f0, f1, f2  # multiply-add f3 = f0*f1+f2
fmsub.s f3, f0, f1, f2  # multiply-sub f3 = f0*f1-f2
```

e mais algumas...