Universidade de Brasília
Departamento de Ciência da Computação

Aula 10
Aritmética Computacional
Aritmética Fracionária

Ponto Fixo

Representação de casas decimais em complemento de 2:

Ex.: 8 bits

Q3: $2^4\ 2^3\ 2^2\ 2^1\ 2^0\ ,\ 2^{-1}\ 2^{-2}\ 2^{-3}$

Menor valor: 10000000 $-2^4 = -16$
Maior valor: 01111111 $2^3+ 2^2+ 2^1+ 2^0+2^{-1}+ 2^{-2} + 2^{-3} = 15{,}875$

Q1: $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0,\ 2^{-1}$

Menor valor: 10000000 $-2^6 = -64$
Maior valor: 01111111 $2^5+ 2^4+ 2^3+ 2^2+2^1+ 2^0 + 2^{-1} = 63{,}5$

Q7: $2^0,\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ 2^{-5}\ 2^{-6}\ 2^{-7}$

Menor valor: 10000000 $-2^0 = -1$
Maior valor: 01111111 $2^{-1}+ 2^{-2}+ 2^{-3}+ 2^{-4}+2^{-5}+ 2^{-6} + 2^{-7} = 0{,}9921875$

- Exemplo:

Considerando 8 bits, calcule a representação em Q7

- 0.75 =
- -0.75 =
- 0.3 =

Considerando 16 bits, calcule a representação em Q15

- 0.75 =
- -0.75 =
- 0.3 =

Ponto Fixo

- Operações Matemáticas: da mesma forma que inteiros usando mesmo Q.
  - Soma
  - Subtração
  - Multiplicação (algoritmos clássicos apenas para positivos!)
  - Divisão (ex.: 5/8)

  Ex.: 8 bits

	Q0	Q2	Q7
01010001	81	20.25	0.6328125
+ 10111001	-71	-17.75	-0.5546875
00001010	10	2.50	0.0781250
00000110	6	1.5	0.046875
× 00001010	× 10	× 2.5	× 0.078125
00000000 00111100	60	3.75	0.003662109375

Ponto Fixo

- Vantagens:
  - Aritmética é simples e rápida
    (Processador menor, mais rápido e mais barato)
- Problemas:
  - Pequena faixa dinâmica
  - Precisão depende da faixa dinâmica

Ponto flutuante

- **Precisamos de uma maneira de representar grande faixa dinâmica**
  - números muito pequenos: 0,000000000000001182721226716
  - números muito grandes: 1672838763512000000000000000000

- Notação Científica (base 10): Mantissa ou Significando
  Característica ou Expoente

  - Ex.: $1.182721226716 \times 10^{-15}$ e $1.672838763512 \times 10^{29}$

  - Sempre normalizado, isto é, apenas 1 dígito não decimal (diferente de zero).

- Notação Científica (base 2)

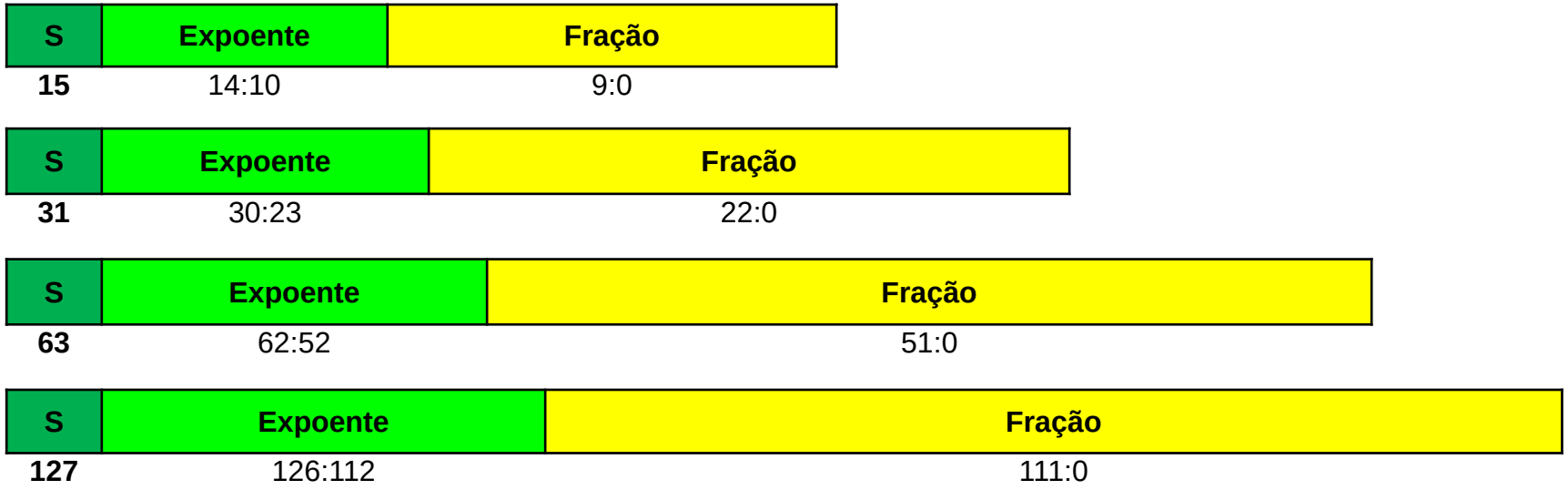
  - Ex.: $1.010 \times 2^{-2} = 0.01010 = 0.3125 = (1+0.25) \times 2^{-2}$

Ponto flutuante

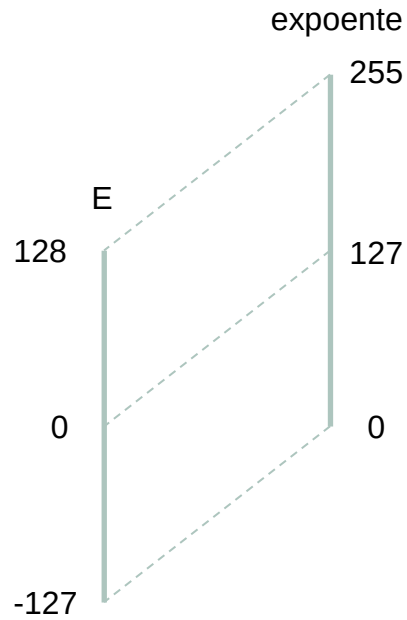
- Representação:- sinal, expoente, significando: $(-1)^S \times M \times 2^E$
  - mais bits para o significando fornece mais precisão
  - mais bits para o expoente aumenta a faixa

- **Padrão de ponto flutuante IEEE 754 (2008):**

- meia precisão: 16 bits: 1+5+10 Usada em processamento gráfico (GPU)
- precisão simples: 32 bits: 1+8+23 tipo (float)
- precisão dupla: 64 bits: 1+11+52 tipo (double)
- precisão quádrupla: 128 bits: 1+15+112 Ainda pouco utilizado



Padrão de ponto flutuante IEEE 754



- O bit “1” inicial do significando está implícito (aumenta a precisão)

- O expoente possui um off-set para facilitar a ordenação
  - Off-set de 15 para meia precisão 127 para precisão simples
    1023 para precisão dupla 16383 para precisão quádrupla

  - Formato:

$$(-1)^{sinal} \times (1 + Fração) \times 2^{(Expoente - offset)}$$

- Exemplo:
  Converter o número decimal N = -5,0 para IEEE754 precisão simples
  Colocar no formato: $N = (-1)^S \times M \times 2^E$ onde $1 \leq M < 2$
  $S = 1$
  $E = floor(\log_2(|N|)) = floor(\log_2(5,0)) = floor(2,3219) = 2 \Rightarrow$ Expoente=129
  $M = |N| / 2^E = 5,0 / 2^2 = 5,0/4 = 1,25 \Rightarrow 1,01_2$

Assim: $-5,0 = (-1)^1 \times (1,01_2) \times 2^{(129-127)}$
precisão simples IEEE: $1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000_2$
0xC0A00000

Padrão de ponto flutuante IEEE 754

- Dado o número em FP IEEE754: 0xC1100000
  qual o número decimal representado?

1100 0001 0001 0000 0000 0000 0000 0000

1 10000010 00100000000000000000000

E = 130 - 127 = 3
M = 1.001

Logo: $(-1)^1 \times (1.001) \times 2^3 = -(1001.0) = -9{,}0$

Padrão de ponto flutuante IEEE 754

- Como representar 0?

Precisão Simples		Precisão Dupla		Objeto
Expoente	Fração	Expoente	Fração	
0	0	0	0	0
0	≠0	0	≠0	±Número desnormalizado
1-254	∀	1-2046	∀	±Número Ponto Flutuante
255	0	2047	0	±∞
255	≠0	2047	≠0	NaN

Obs.: Número desnormalizado: Considera 0 inicial na mantissa

Qual a faixa dinâmica dos números representáveis em precisão simples e dupla sem overflow ou underflow?

Operações em Ponto Flutuante

- **Adição e Subtração em IEEE 754:**

  Procedimento idêntico às operações em Notação Científica base 10.

- Converte-se o número com menor expoente para igualar ao expoente do maior e soma-se (subtrai-se) as mantissas

Ex.: Em notação científica Decimal

com limite de representação de 4 dígitos fracionários na mantissa

$9.9999\times10^{2} + 1.7100\times10^{-1} = 9.9999\times10^{2} + 0.00171\times10^{2} =$

$= 9.9999\times10^{2} + 0.0017\times10^{2} = 10.0016\times10^{2} = 1.00016\times10^{3} =$

$= 1.0002\times10^{3}$

$999.99 + 0.171 = 1000.161 \rightarrow 1000.2$

Ex.: Em notação científica Binária

com limite de representação de 4 bits fracionários na mantissa

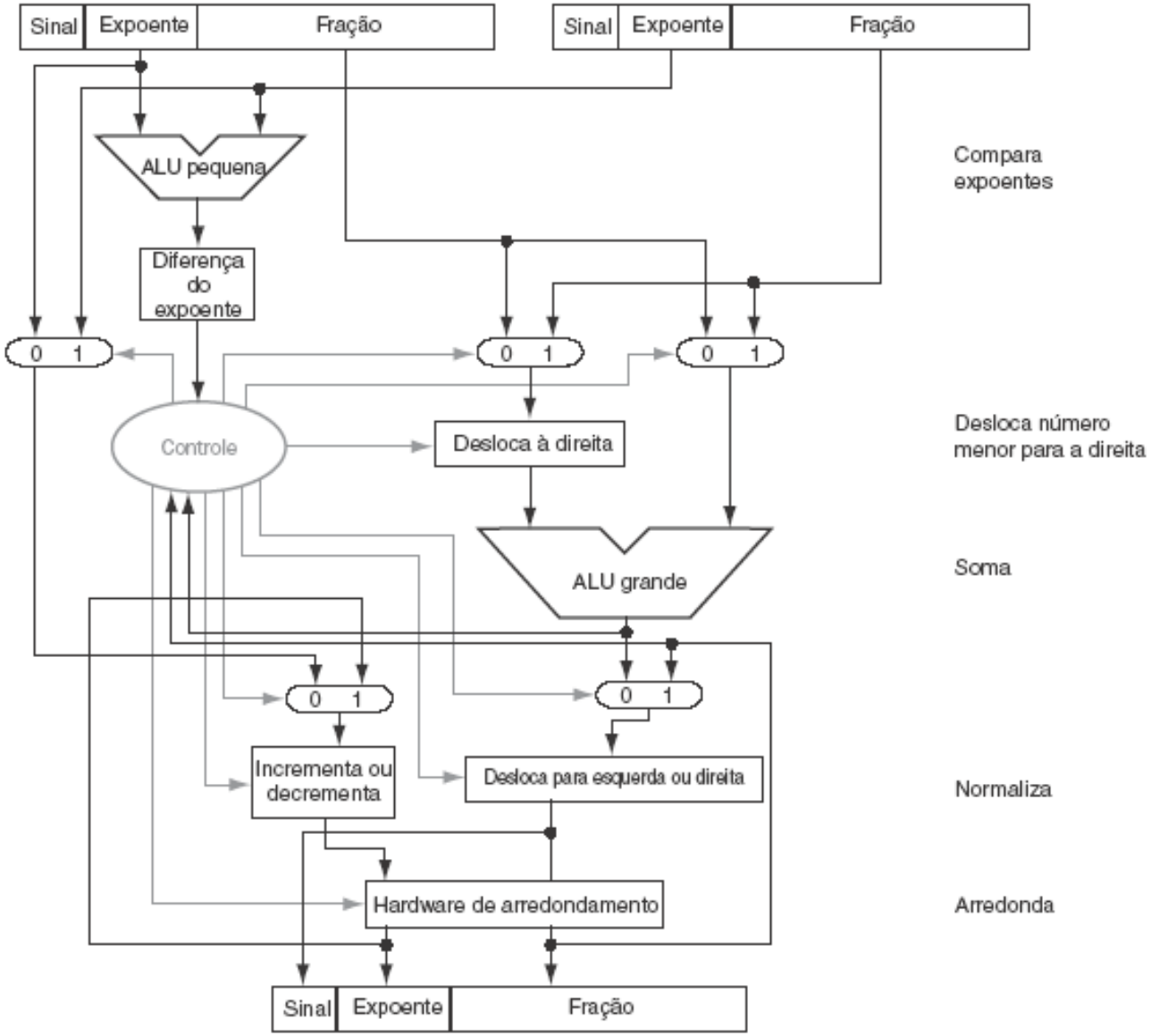
$1.1110\times2^{2} + 1.1100\times2^{-1} = 1.1110\times2^{2} + 0.00111\times2^{2}$

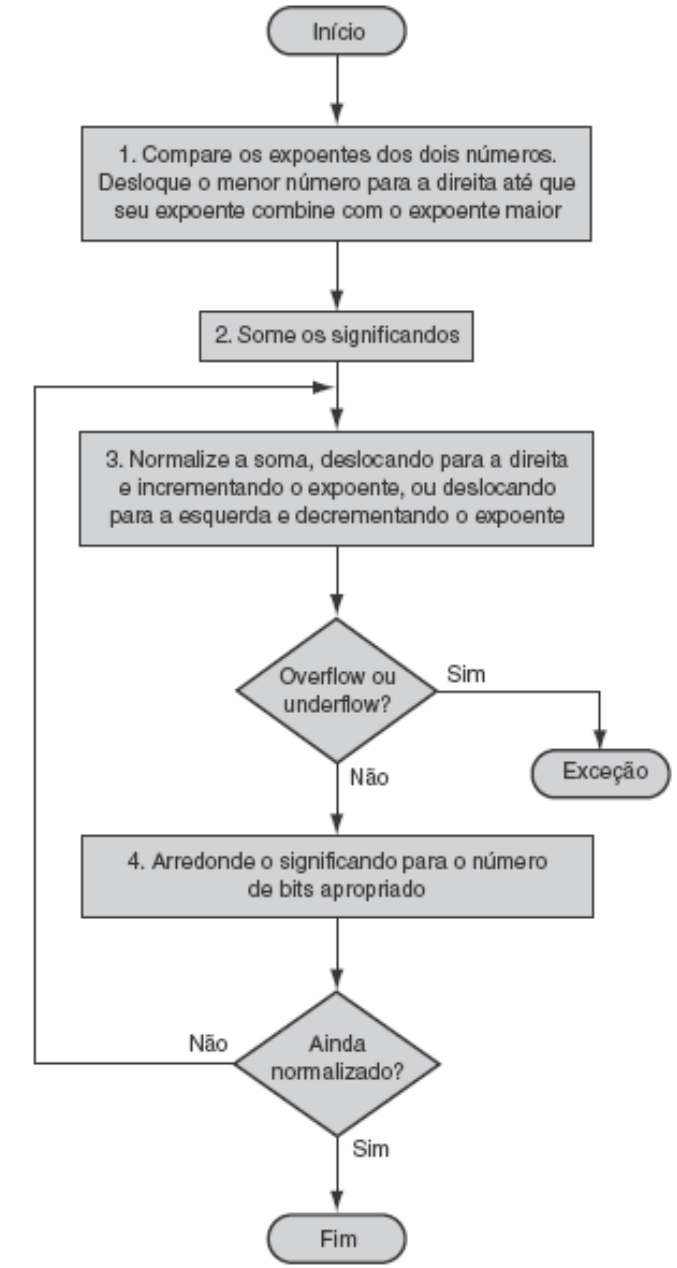
$= 1.1110\times2^{2} + 0.0011\times2^{2} = 10.0001\times2^{2} = 1.00001\times2^{3}$

$= 1.0000\times2^{3}$

$7.5 + 0.875 = 8.375 \rightarrow 8.000$

Adição de ponto flutuante





Operações em Ponto Flutuante

- **Multiplicação e Divisão em IEEE 754:**

  Procedimento idêntico às operações em Notação Científica base 10:

- Multiplica-se as mantissas e soma-se os expoentes ou
- Divide-se as mantissas e subtrai-se os expoentes

Ex.: Decimal

$3{,}2300 \times 10^{2} \times 3{,}4150 \times 10^{-1} = 11{,}03045 \times 10^{1}$

(4 dígitos) $= 1{,}1030 \times 10^{2}$

Ex.: Binário

$1{,}1000 \times 2^{-1} \times (-1{,}1101 \times 2^{-2}) = -10{,}10111 \times 2^{-3}$

(4 bits) $= -1{,}0101 \times 2^{-2}$

Overflow: |resultado| > MAX : |resultado|=infinito

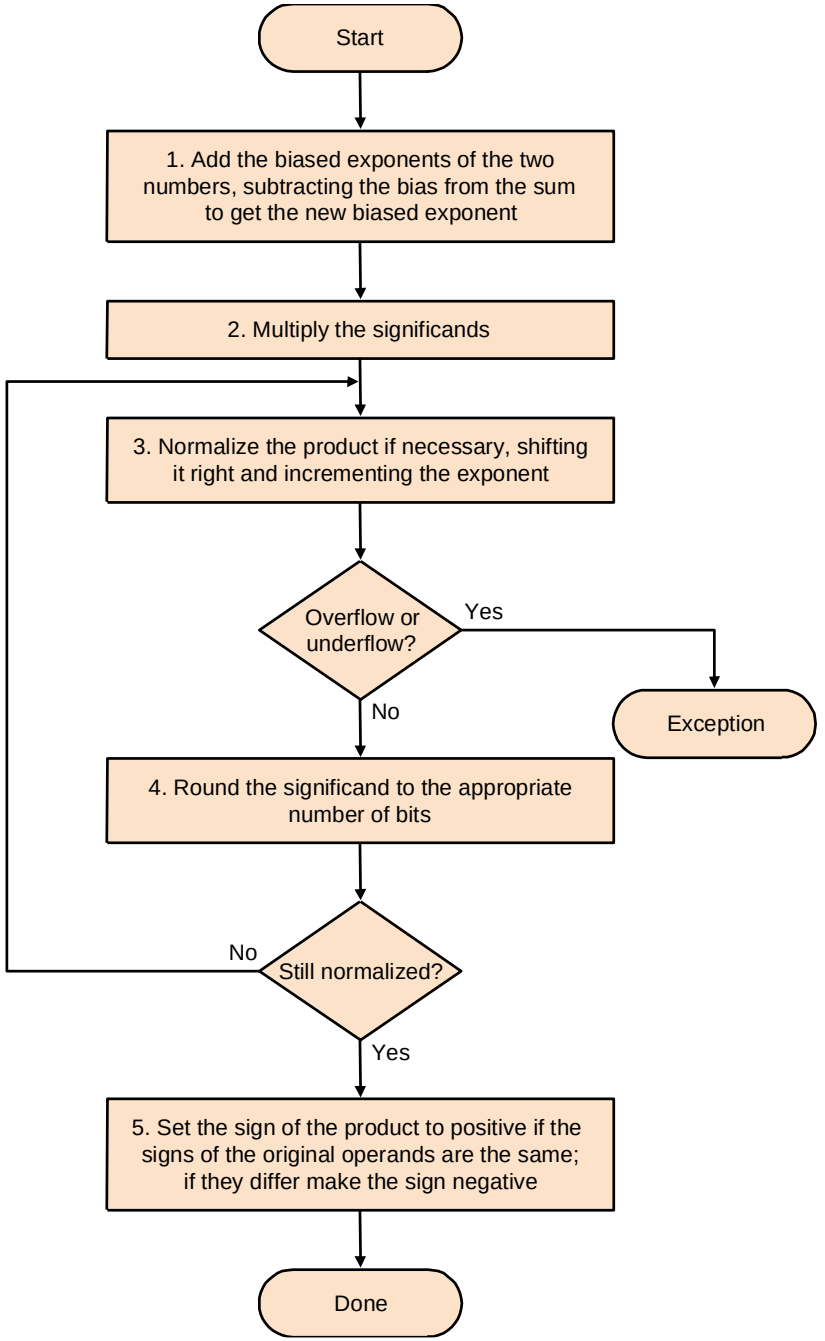
Underflow: |resultado| < MIN : |resultado|=0,0

Multiplicação de ponto flutuante

Obs.: IEEE 754

Expoentes com offset, logo diminuir 1 offset da soma dos expoentes!

Obs.: Projetar o hardware!



Ponto Flutuante: Arredondamento

- O IEEE754 permite 5 tipos de arredondamentos

  - Sempre para +∞ (cima, *ceil*): 2.1 → 3 2.5 → 3 2.9 → 3
  - Sempre para -∞ (baixo, *floor*) : 2.1 → 2 2.5 → 2 2.9 → 2
  - Ao mais próximo (*round*): 2.1 → 2 2.9 → 3 2.5 → ? ↑ ↓ ?

    Ao zero: 2.5 → 2 ↓ -2.5 → -2 ↑

    À maior magnitude: 2.5 → 3 ↑ -2.5 → -3 ↓

    Ao dígito par: 2.5 → 2 ↓ 3.5 → 4 ↑

Os arredondamentos se aplicam também ao último bit de precisão da mantissa.

Obs.: Em precisão finita operações lineares passam a ser não-lineares!

Ex.: $(x+y)+z \neq x+(y+z)$

$x + (y + z) = -1{,}5\times10^{38} + (1{,}5\times10^{38} + 1{,}0) = 0{,}0$

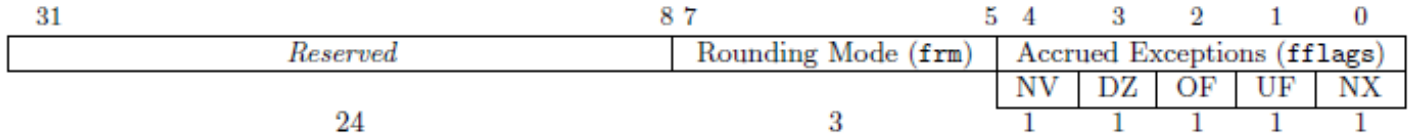
$(x + y) + z = (-1{,}5\times10^{38} + 1{,}5\times10^{38}) + 1{,}0 = 1{,}0$

Ponto Flutuante - ISA RV32IM**FD**

- F: Single precision
- D: Double precision
- Q: Quad precision

- 32 Registradores de 64 bits:
  f0, f1,..., f30, f31 com convenção →

- *Float-point Control and Status Register* (fcsr)



Modos de Arredondamento:
000: ao número par (round)
001: para o zero (round)
010: para baixo (floor)
011: para cima (ceil)
100: para a maior magnitude (round)

Flags de exceção
NV = Operação Inválida
DZ = Divisão por zero
OF = Overflow
UF = Underflow
NX = Inexato

Control and Status

Registers | Floating Point

Name	Number	Value
ft0	0	0x0000000000000000
ft1	1	0x0000000000000000
ft2	2	0x0000000000000000
ft3	3	0x0000000000000000
ft4	4	0x0000000000000000
ft5	5	0x0000000000000000
ft6	6	0x0000000000000000
ft7	7	0x0000000000000000
fs0	8	0x0000000000000000
fs1	9	0x0000000000000000
fa0	10	0x0000000000000000
fa1	11	0x0000000000000000
fa2	12	0x0000000000000000
fa3	13	0x0000000000000000
fa4	14	0x0000000000000000
fa5	15	0x0000000000000000
fa6	16	0x0000000000000000
fa7	17	0x0000000000000000
fs2	18	0x0000000000000000
fs3	19	0x0000000000000000
fs4	20	0x0000000000000000
fs5	21	0x0000000000000000
fs6	22	0x0000000000000000
fs7	23	0x0000000000000000
fs8	24	0x0000000000000000
fs9	25	0x0000000000000000
fs10	26	0x0000000000000000
fs11	27	0x0000000000000000
ft8	28	0x0000000000000000
ft9	29	0x0000000000000000
ft10	30	0x0000000000000000
ft11	31	0x0000000000000000

Registers | Floating Point | Control and Status

Name	Number	Value
ustatus	0	0x00000000
fflags	1	0x00000000
frm	2	0x00000000
fcsr	3	0x00000000
uie	4	0x00000000
utvec	5	0x00000000
uscratch	64	0x00000000
uepc	65	0x00000000
ucause	66	0x00000000
utval	67	0x00000000
uip	68	0x00000000
misa	769	0x40001128
cycle	3072	0x00000000
time	3073	0x00000000
instret	3074	0x00000000
cycleh	3200	0x00000000
timeh	3201	0x00000000
instreth	3202	0x00000000

Ponto Flutuante - ISA RV32IM**F** – Float-point Single

- Operações com precisão simples
  - Adição: fadd.s f0, f1, f2 # f0 = f1 + f2
  - Subtração: fsub.s f0, f1, f2 # f0 = f1 - f2
  - Multiplicação: fmul.s f0, f1, f2 # f0 = f1 × f2
  - Divisão: fdiv.s f0, f1, f2 # f0 = f1 ÷ f2
  - Raiz Quadrada: fsqrt.s f0, f1 # $f0 = \sqrt{f1}$
  - Comparação:

```
feq.s t1, f1, f2      # se f1 = f2 então t1=1 senão t1=0
fle.s  t1, f1, f2     # se f1 ≤ f2 então t1=1 senão t1=0
flt.s   t1, f1, f2    # se f1 < f2 então t1=1 senão t1=0:

fmax.s f0, f1, f2     #  f0 = max(f1,f2)
fmin.s  f0, f1, f2    #  f0 = min(f1,f2)
```

Ponto Flutuante - ISA RV32IM**F** – Float-point Single

- Transferências de dados
  - Load: flw f1, -100(t0) # f1 = Mem[t0 - 100]
  - Store: fsw f1, -100(t0) # Mem[t0 - 100] = f1
  - Movimentação sem conversão:

```
fmv.x.s t0, f1      # transfere os bits  t0 = f1
fmv.s.x f1, t0      # transfere os bits  f1 = t0
```

  - Movimentação com conversão:

```
fcvt.s.w   f1, t0   # converte inteiro → float
fcvt.s.wu f1, t0    # converte inteiro sem sinal → float
fcvt.w.s   t0, f1   # converte float → inteiro
fcvt.wu.s t0, f1    # converte float → inteiro sem sinal
```

Exemplo: Conversor de Temperatura

```
float fahr2c(float fahr)
{
   return 5.0*(fahr-32.0) / 9.0;
}
```

constantes no segmento de dados ou em imediatos (o que é + eficiente?)

```
.data
const5:   .float 5.0
const9:   .float 9.0

.text
fahr2c:       la t0,const5
       flw ft0,0(t0)          # ft0=5.0
       flw ft1,4(t0)          # ft1=9.0
       li t0,32
       fcvt.s.w ft2,t0        # Não requer acesso à memória de dados!!!
       fsub.s fa0,fa0,ft2
       fmul.s fa0,fa0,ft0
       fdiv.s fa0,fa0,ft1
       ret
```

Complexidades do ponto flutuante

- As operações aritméticas são mais complexas
- Além do *overflow* podemos ter *underflow*
- A precisão pode ser um grande problema
  - O IEEE 754 mantém dois bits extras, guarda e arredondamento
  - cinco modos de arredondamento
  - positivo dividido por zero produz infinito
  - zero dividido por zero produz um não-número (NaN)
  - outras complexidades...
- Implementar o padrão pode ser arriscado
- Não usar o padrão pode ser ainda pior
  - x86 e o bug da instrução `fdiv` do Pentium! (jul, set, nov, dez de 1994)

Conclusões

- A aritmética do computador é restrita por uma precisão limitada
- Os conjuntos de bit não têm um significado inerente mas existem padrões (convenções)
  - sem sinal
  - complemento de dois
  - ponto fixo
  - ponto flutuante IEEE 754
- As **instruções** determinam o “significado” dos conjuntos de bit
- O desempenho e a precisão são importantes; portanto, existem muitas complexidades nas máquinas reais
- A escolha do algoritmo é importante e pode levar a otimizações de hardware para espaço e tempo (por exemplo, multiplicação)