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:

  Considere 8 bits, calcule a representação em Q7

  - 0.75 =
  - -0.75 =
  - 0.3 =

  Truncamento? Arredondamento?

  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
  - 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
x 00001010	x10	x2.5	x0.078125
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):**
  - precisão simples: 32 bits: 1+8+23

31	30 29 28 27 26 25 24 23	22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
s	expoente	mantissa
1 bit	8 bits	23 bits

- precisão dupla: 64 bits: 1+11+53

31	30 29 28 27 26 25 24 23 22 21 20	19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
s	expoente	mantissa
1 bit	11 bits	20 bits

mantissa (continuação)
32 bits

-precisão quádrupla: 128 bits: 1+15+112 Ainda pouco utilizado

-meia precisão: 16 bits: 1+5+10 Usada em processamento gráfico

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 127 para precisão simples
    e de 1023 para precisão dupla
  - Formato:

$$(-1)^{\text{sinal}} \times (1 + \text{fração}) \times 2^{(\text{expoente} - \text{offset})}$$

- Exemplo:

Converter o número decimal N = -5,0 para IEE754 precisão simples

Colocar no formato: $N = (-1)^S \times M \times 2^E$ onde $1 \leq M < 2$

$S = 1$

$E = \text{floor}(\log_2(N)) = \text{floor}(\log_2(5{,}0)) = \text{floor}(2{,}3219) = 2$ => expoente=129

$M = N / 2^E = 5{,}0 / 2^2 = 5{,}0/4 = 1{,}25$ => $1{,}01_2$

Assim: $-5{,}0 = (-1)^1 (1{,}01_2) \times 2^{(129-127)}$

precisão simples IEEE: $1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000_2$

0xC0A00000

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

1100 0001 0001 0000 0000 0000 0000 0000

1 10000010 00100000000000000000000

expoente=130 - 127 = 3
Mantissa: 1.001

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

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

- **Soma 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.: Decimal

$5{,}25\times10^{2} + 1{,}5\times10^{-1} = 5{,}25\times10^{2} + 0{,}0015\times10^{2} = 5{,}2515\times10^{2}$

(4 dígitos) $= 5{,}251\times10^{2}$

Ex.: Binário

$1{,}01\times2^{2} + 1{,}1\times2^{-1} = 1{,}01\times2^{2} + 0{,}0011\times2^{2} = 1{,}0111\times2^{2}$

(4 bits) $= 1{,}011\times2^{2}$

Note 5 + 0,75 resulta em 5,5 devido à precisão limitada em 4 bits

Operações em Ponto Flutuante

- Adição e Subtração

Ex.: Em notação Científica Decimal

com limite de representação de 4 dígitos (significado) e 2 dígitos (expoente)

$9{,}999 \times 10^{2} + 1{,}61 \times 10^{-1} = 9{,}999 \times 10^{2} + 0{,}00161 \times 10^{2} =$

$= 9{,}999 \times 10^{2} + 0{,}0016 \times 10^{2} = 10{,}015 \times 10^{2} = 1{,}0015 \times 10^{3} =$

$= 1{,}002 \times 10^{3}$

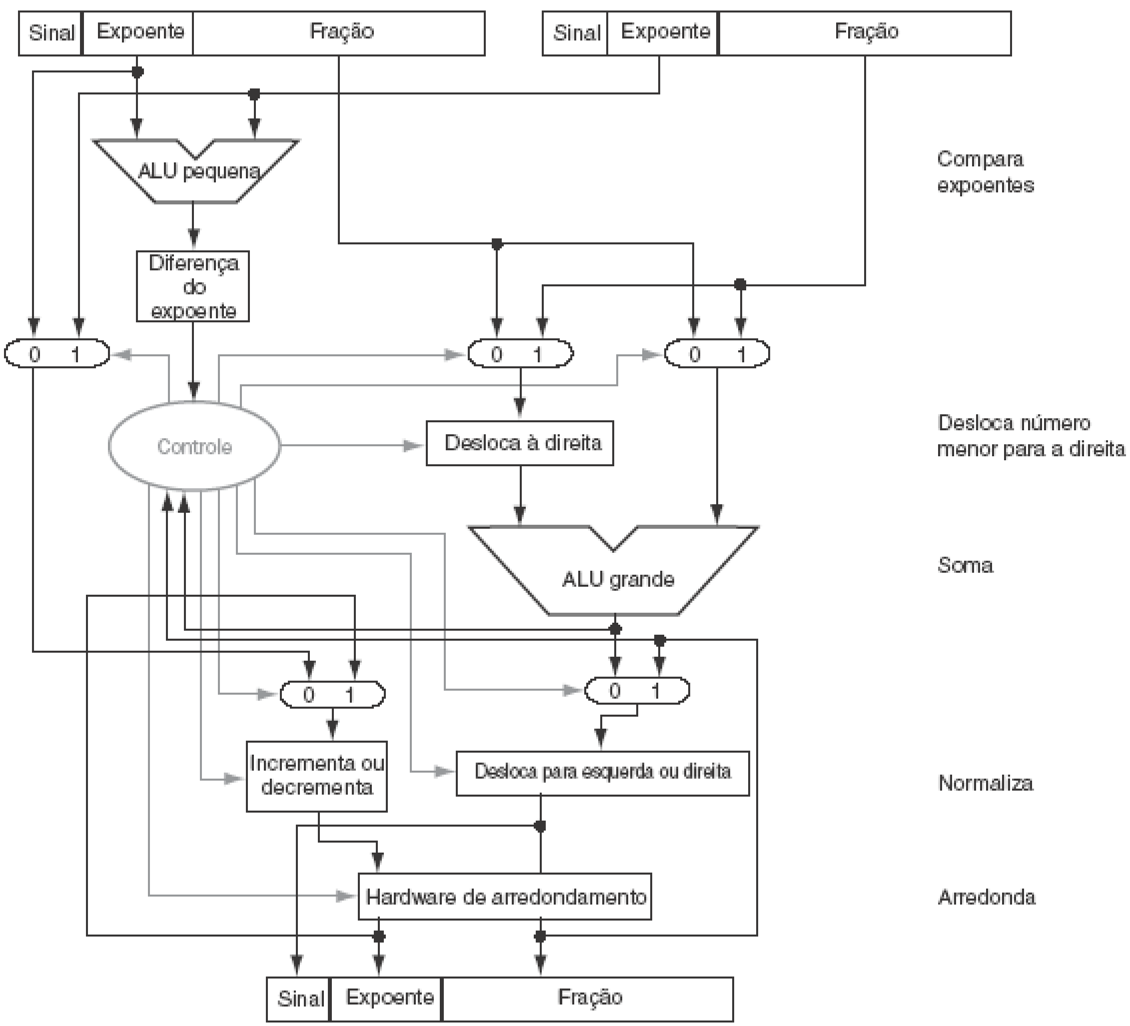
Ex.: Em notação Científica Binária com 4 bits de precisão: 0,5+(-0.4375)

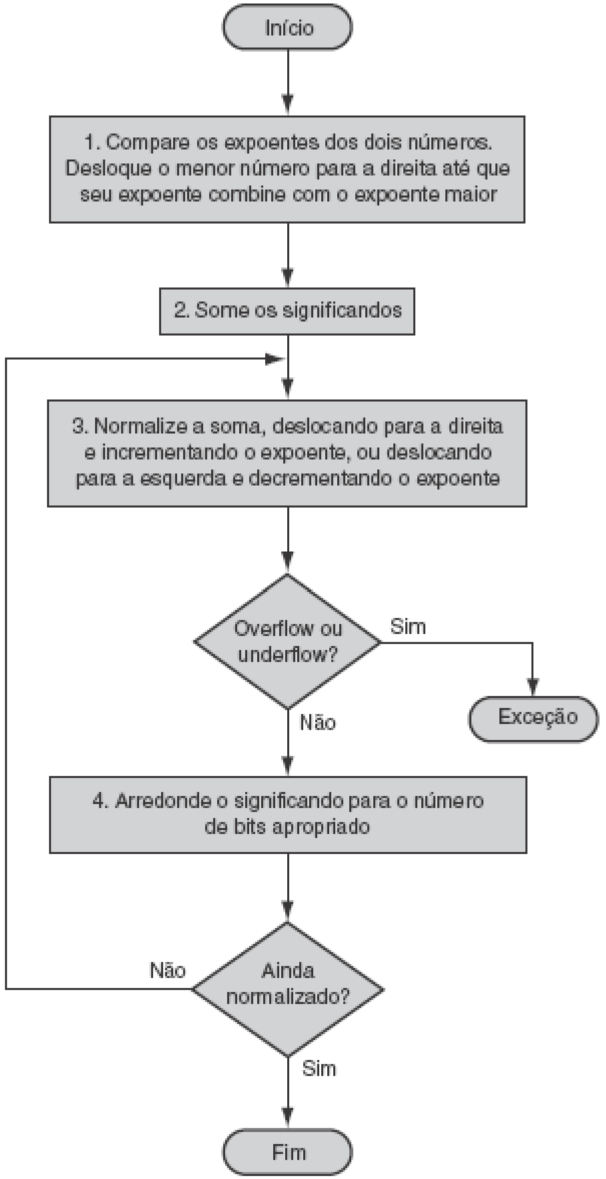
$0.5_{DEC} = 0.1 = 1.0 \times 2^{-1}$

$-0.4375_{DEC} = -0.0111 = -1.11 \times 2^{-2} = -0.111 \times 2^{-1}$

$1.0 \times 2^{-1} + -0.111 \times 2^{-1} = (1.000 - 0.111) \times 2^{-1} = 0.001 \times 2^{-1} = 1.0 \times 2^{-4} = 0.0625_{DEC}$

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

$3,23\times10^{2} \times 3,415\times10^{-1} = 11,03045 \times 10^{1}$

(4 dígitos) $= 1,103\times10^{2}$

Ex.: Binário

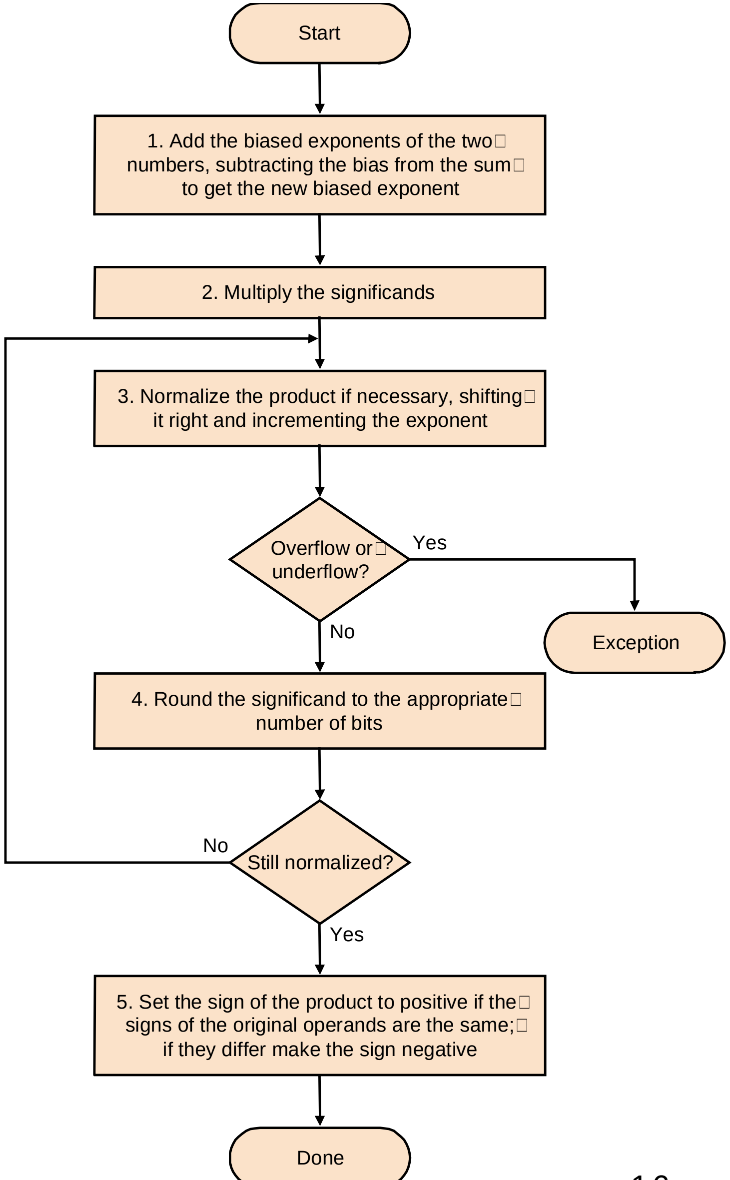
$1,000\times2^{-1} \times (-1,110\times2^{-2}) =- 1,110000\times2^{-3}$

(4 bits) $= -1,110\times2^{-3}$

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 hardware

Ponto Flutuante: Arredondamento

- O IEEE754 permite 4 tipos de arredondamentos
  - Sempre para $+\infty$ (cima, *ceil*): 2.11=>2.2 2.15=> 2.2 2.19=>2.2
  - Sempre para $-\infty$ (baixo, *floor*) : 2.11=>2.1 2.15 => 2.1 2.19=>2.1
  - Truncamento: Despreza os bits menos significativos (*trunc*)

    +1.01101 = 1.40625 => +1.011 = 1.375
  - Ao mais próximo (*round*): 2.11 => 2.1 2.19=>2.2 2.15=>?↑↓?

    Estatisticamente coerente:

    Ao dígito par: 2.15=> 2.2 2.25=> 2.2

Obs.: Em precisão finita $(x+y)+z \neq x+(y+z)$

$x + (y + z) = -1{,}5 \times 10^{38} + (1{,}5 \times 10^{38} + 1{,}0) = 0{,}0$

$(x + y) + z = (-1{,}5 \times 10^{38} + 1{,}5 \times 10^{38}) + 1{,}0 = 1{,}0$

Ponto Flutuante no MIPS32 : Coprocessador1

- 32 Registradores de precisão simples: $f0, $f1,..,$f30,$f31

  ou

- 16 Registradores de precisão dupla: $f0,$f2,$f4, ...,$f30
- Permite operações com precisão simples e dupla
  - Adição: add.s e add.d # add.s $f0,$f1,$f2
  - Subtração: sub.s e sub.d # sub.d $f0,$f2,$f4
  - Multiplicação: mul.s e mul.d # mul.s $f0,$f1,$f2
  - Divisão: div.s e div.d # div.d $f0,$f4,$f8
  - Comparação: c.x.s e c.x.d # c.eq.s 0,$f1,$f2
    onde x pode ser: eq, lt ou le

    Seta um bit do byte de flag como V(1) ou F(0)
  - Desvio se flag V (bc1t) desvio se F (bc1f) # bc1t 0,LABEL
  - Load e Store: lwc1 e swc1, ldc1 e sdc1 # lwc1 $f0,Imm($s2)
  - Move: mfc1, mtc1 # mfc1 $t0, $f0
  - Arredondamentos: ceil, floor, round, trunc # ceil.w.s $f0,$f1
  - cvt.x.y converte de y para x (s,d,w) # cvt.s.w $f2,$f4

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		
FI*	opcode	fmt	rt	fs	0	
	31 26	25 21	20 16	15 11	10 0	

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION		(2) OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--

Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--
Move Single	mov.s	FR	F[fd] = F[fs]		11/10/0/6
Move Double	mov.d	FR	F[fd] = F[fs]		11/11/0/6
Move To C1	mtc1	FI*	F[fs] = R[rt]		11/4/-/0
Move From C1	mfc1	FI*	R[rt] = F[fs]		11/0/-/0
Convert from Y to X	cvt.x.y	FR	$F[fd]_x = F[fs]_y$	(x,y)={S,D,W}	
Square Root	sqrt.s	FR	F[fd] = sqrt(F[fs])		11/10/0/4

Exemplo 1: Conversor de Temperatura

- Ex.:

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

Considerando que o argumento de entrada em $f12 e saída em $f0.

```
#constantes  no segmento de dados
.data
const5:   .float 5.0
const9:   .float 9.0
const32: .float 32.0
.text
fahr:       l.s $f16, const5     # Pseudo Instrução
            l.s $f18, const9
            div.s $f16,$f16,$f18
            l.s $f18, const32
            sub.s $f18, $f12, $f18
            mul.s $f0, $f16, $f18
            jr $ra
```

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
  - quatro 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
  - 80x86 e o bug do Pentium! (07, 09, 11, 12 de 1994)

Conclusões

- A aritmética de computador é restrita por uma precisão limitada
- Os conjuntos de bit não têm um significado inerente mas existem padrões (convenções)
    - 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)