Organização de Computadores I
DCC006

Aula 5 – Aritmética Computacional
Ponto Flutuante

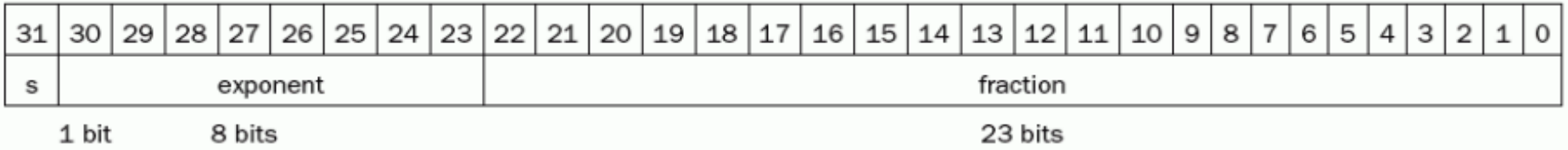
Prof. Omar Paranaiba Vilela Neto

Ponto Flutuante

- Nós necessitamos de uma maneira para representar
  - Números com frações, ex., 3.1416
  - Números muito pequenos, ex., .000000001
  - Números muito grandes, ex., 3.15576×10^9
- Representação:
  - Sinal, expoente e fração: $(-1)^{Sinal}$ x fração $\times$ $2^{expoente}$
  - Quanto mais bits a fração tiver, maior a precisão
  - Quanto mais bits o expoente tiver, maior é a faixa representável
- Compromisso entre:
  - Tamanho da fração;
  - Tamanho do expoente.

Ponto Flutuante

- Padrão de ponto flutuante IEEE 754 :
  - Precisão simples: 8 bit expoente, 23 bit fração

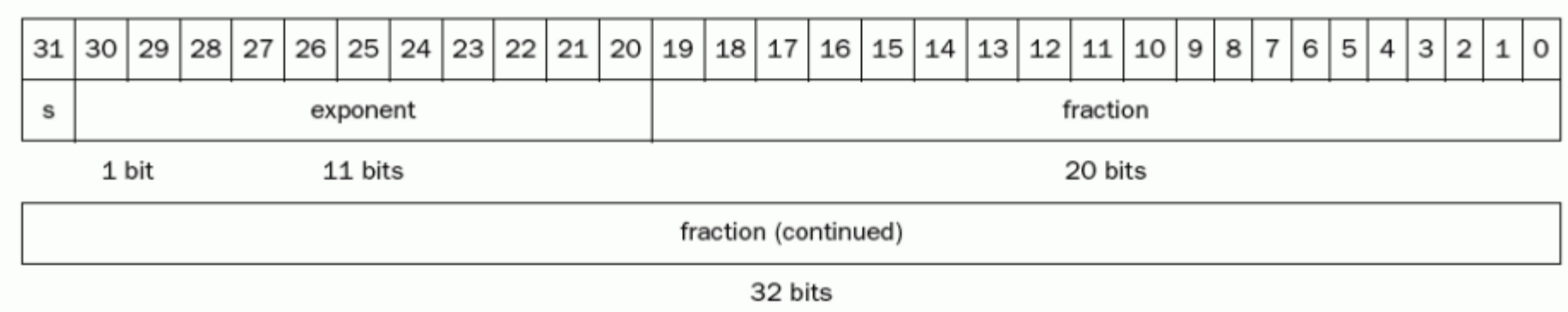


- Formato: $(-1)^s \times F \times 2^E$
- Pode ocorrer overflow em ponto flutuante?
- Novidade: underflow.

Ponto Flutuante

Como **minimizar** o **overflow** e o **underflow**?

Precisão Dupla



Precisão dupla: 11 bit expoente, 52 bit fração

Padrão Ponto Flutuante IEEE 754

- **Bit à esquerda do ponto binário é implícito, vale "1"**

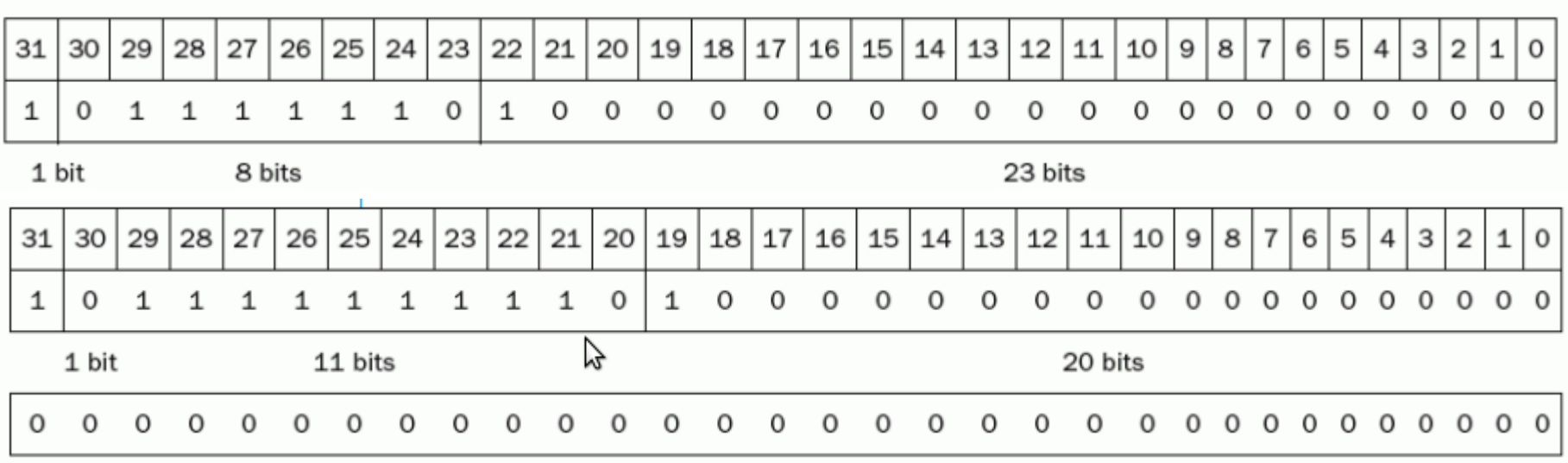
- **Expoente**
  - **Todo 0s é o menor expoente, todo 1s é o maior**
  - **Peso (excesso) de 127 para precisão simples e 1023 para precisão dupla**
  - **sumário: $(-1)^{sinal} \times (1 + \text{fração}) \times 2^{expoente - peso}$**

- **Exemplo:**

  - **Mostre a representação binária para o número decimal: -0.75 em precisão simples e dupla.**

Padrão Ponto Flutuante IEEE 754

- O número -0,75 também é:
  - $-3/4_{dec}$ ou $-3/2^2{}_{dec}$
- A representação binária é:
  - $-11_{bin}/2^2{}_{dec}$ ou $-0{,}11_{bin}$
- A notação científica é:
  - $-0{,}11_{bin} \times 2^0$
- A notação científica normalizada
  - $-1{,}1_{bin} \times 2^{-1}$
- A representação geral em precisão simples é:
  - $(-1)^s \times (1+\text{fração}) \times 2^{(\text{Expoente} - 127)}$



Padrão Ponto Flutuante IEEE 754

Exercício

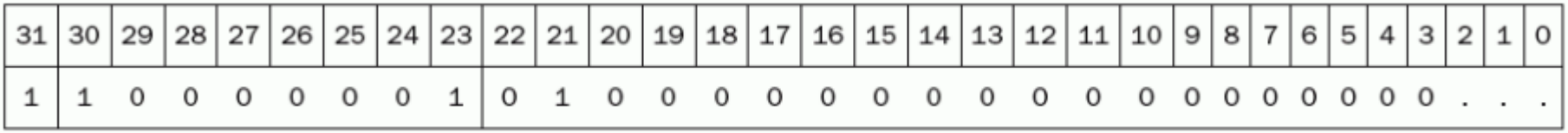
- **Qual o número decimal representado por este float em precisão simples?**

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
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

Padrão Ponto Flutuante IEEE 754

Exercício

- **Qual o número decimal representado por este float em precisão simples?**



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
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

11000000101000…00

- S = 1
- Fraction = $01000\ldots00_2$
- Fxponent = $10000001_2 = 129$

- $x = (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

Adição de Ponto Flutuante

Vamos somar os números $0,5_{dec}$ e $-0,4375_{dec}$ em binário.

Vamos transformá-los em binários na notação científica normalizada, supondo que mantemos 4 bits de precisão.

$$0,5_{dec} = 1/2_{dec} = 1/2^1{}_{dec} = 0,1_{bin} \times 2^0 = 1,0_{bin} \times 2^{-1}$$

$$-0,4375_{dec} = -7/16_{dec} = -7/2^4{}_{dec} = -0,0111_{bin} \times 2^0 = -1,110_{bin} \times 2^{-2}$$

Adição de Ponto Flutuante

Vamos somar os números $0{,}5_{dec}$ e $-0{,}4375_{dec}$ em binário.

1º – O significado do número com menor expoente ($-1{,}110_{bin} \times 2^{-2}$) é deslocado para a direita até seu expoente combinar com o número menor:

$$-1{,}110_{bin} \times 2^{-2} = -0{,}111_{bin} \times 2^{-1}$$

2º – Some os significados:

$$1{,}000_{bin} \times 2^{-1} + -0{,}111_{bin} \times 2^{-1} = 0{,}001_{bin} \times 2^{-1}$$

Adição de Ponto Flutuante

Vamos somar os números $0,5_{dec}$ e $-0,4375_{dec}$ em binário.

3º – Normalize a soma e verifique overflow ou underflow:

$$0,001_{bin} \times 2^{-1} = 1,000_{bin} \times 2^{-4}$$

Como $127 > 4 > -126$, o número está ok.

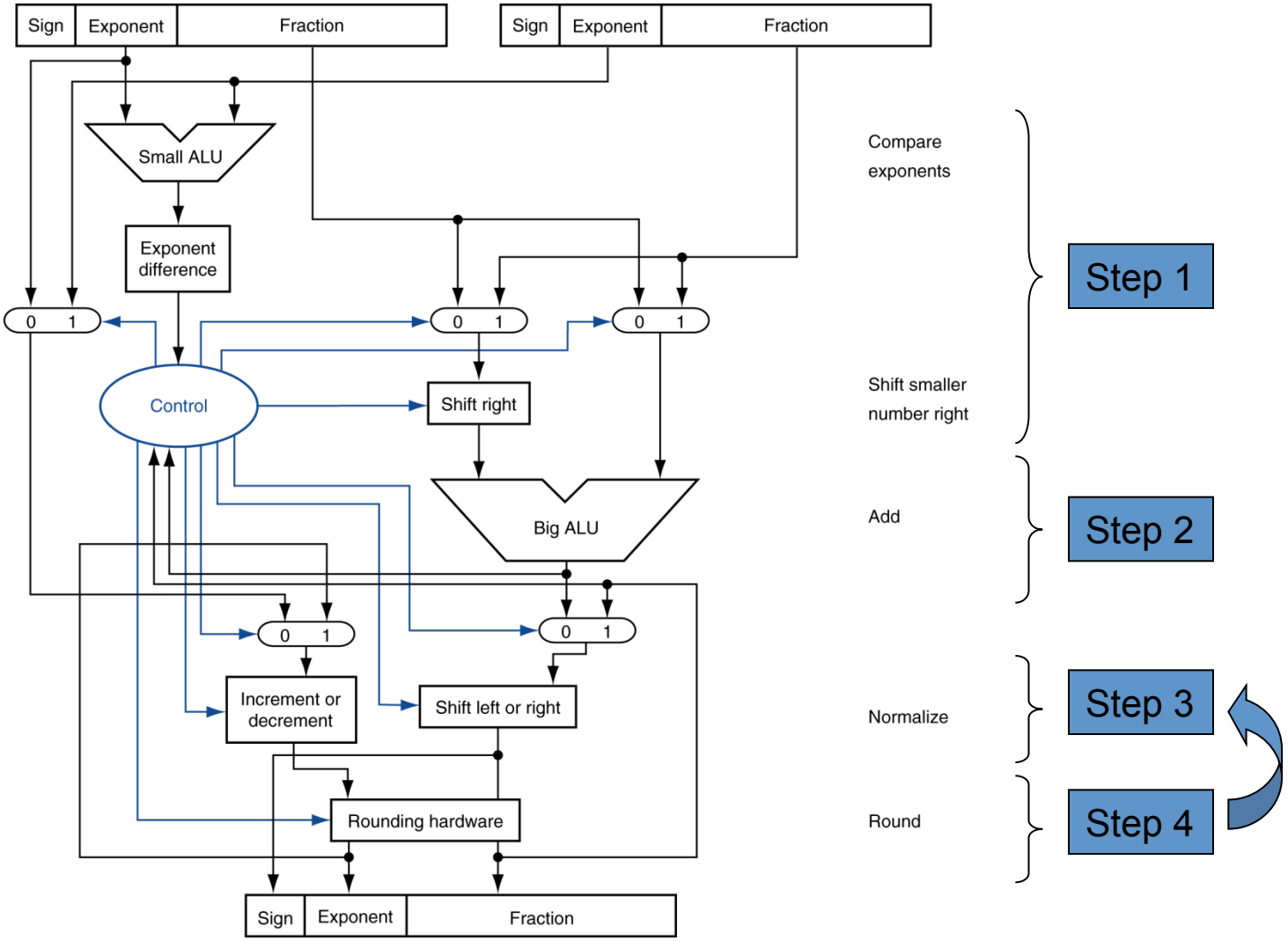
4º – Arredonde a soma:

$$1,000_{bin} \times 2^{-4}$$

Verificando

$$1,000_{bin} \times 2^{-4} = 0,0001_{bin} = 1/2^{4}_{dec} = 1/16_{dec} = 0,0625_{dec}$$

Adição de Ponto Flutuante



Multiplicação de Ponto Flutuante

Vamos multiplicar os números $0{,}5_{dec}$ e $-0{,}4375_{dec}$ em binário.

Vamos transformá-los em binários na notação científica normalizada, supondo que mantemos 4 bits de precisão.

$$0{,}5_{dec} = 1/2_{dec} = 1/2^1{}_{dec} = 0{,}1_{bin} \times 2^0 = 1{,}0_{bin} \times 2^{-1}$$

$$-0{,}4375_{dec} = -7/16_{dec} = -7/2^4{}_{dec} = -0{,}0111_{bin} \times 2^0 = -1{,}110_{bin} \times 2^{-2}$$

Multiplicação de Ponto Flutuante

Vamos multiplicar os números $0{,}5_{dec}$ e $-0{,}4375_{dec}$ em binário.

1º – Somando os expoentes sem bias

$$-1+(-2) = -3$$

2º – Multiplicando os significados:

$$\begin{array}{rr} & 1.000_{two} \\ \times & 1.110_{two} \\ \hline & 0000 \\ & 1000 \\ & 1000 \\ & 1000 \\ \hline & 1110000_{two} \end{array}$$

Multiplicação de Ponto Flutuante

Vamos multiplicar os números $0{,}5_{dec}$ e $-0{,}4375_{dec}$ em binário.

3º – Normalize a multiplicação e verifique overflow ou underflow:

$$1{,}110_{bin} \times 2^{-3}$$

Como 127 > -3 > -126, o número está ok.

4º – Arredonde a soma:

$$-1{,}110_{bin} \times 2^{-3}$$

Verificando

$$-1{,}110_{bin} \times 2^{-3} = -0{,}00111_{bin} = -7/2^{5}{}_{dec} = -7/32_{dec} = -0{,}21875_{dec}$$

Instruções de Ponto Flutuante no RISC-V

- Separate FP registers: f0, …, f31
  - double-precision
  - single-precision values stored in the lower 32 bits
- FP instructions operate only on FP registers
  - Programs generally don’t do integer ops on FP data, or vice versa
  - More registers with minimal code-size impact
- FP load and store instructions
  - `flw, fld`
  - `fsw, fsd`

Instruções de Ponto Flutuante no RISC-V

- Single-precision arithmetic
  - `fadd.s, fsub.s, fmul.s, fdiv.s, fsqrt.s`
    - e.g., `fadds.s f2, f4, f6`
- Double-precision arithmetic
  - `fadd.d, fsub.d, fmul.d, fdiv.d, fsqrt.d`
    - e.g., `fadd.d f2, f4, f6`
- Single- and double-precision comparison
  - `feq.s, flt.s, fle.s`
  - `feq.d, flt.d, fle.d`
  - Result is 0 or 1 in integer destination register
    - Use beq, bne to branch on comparison result
- Branch on FP condition code true or false
  - `B.cond`

Exemplo

- C code:

```
float f2c (float fahr) {
  return ((5.0/9.0)*(fahr - 32.0));
}
```

  - `fahr` in f10, result in f10, literals in global memory space

- Compiled RISC-V code:

```
f2c:
  flw    f0,const5(x3)  // f0 = 5.0f
  flw    f1,const9(x3)  // f1 = 9.0f
  fdiv.s f0, f0, f1  // f0 = 5.0f / 9.0f
  flw    f1,const32(x3) // f1 = 32.0f
  fsub.s f10,f10,f1  // f10 = fahr - 32.0
  fmul.s f10,f0,f10  // f10 = (5.0f/9.0f) * (fahr-32.0f)
  jalr   x0,0(x1)    // return
```

Sumário

- Operações são mais complexas
- Além de overflow podemos ter "underflow"
- Precisão pode ser um grande problema
  - IEEE 754 mantém 2 bits extra, guarda e arredondamento
  - Quatro modos de arredondamento
  - Positivo dividido por zero produz "infinito"
  - Zero dividido por zero produz "não é um número"
  - Outras complexidades
- Implementar o padrão pode ser macetoso
- Não usar o padrão pode ser pior ainda
  - Veja o texto da descrição do 80x86 e do bug do Pentium!

Sumário

- A aritmética do computador é restringida pela precisão limitada.
- Padrões de bits não têm significado inerente, mas há normas
  - Complemento de dois
  - Ponto flutuante IEEE 754.
- As instruções do computador determinam o sentido dos padrões de bits.
- Desempenho e precisão são importantes e portanto há muitas complexidades nas máquinas reais (algoritmos e implementações).
- Estamos prontos para seguir (e implementar um processador)