Aritmética de Ponto Flutuante

Autores	
Odilon de Oliveira Dutra	Unifei

Histórico de Revisões		
1 de março de 2025	1.0	Primeira versão do documento.

Tópicos

1. Introdução
2. Normatização
3. Operações com Ponto Flutuante
4. Valores Especiais
5. Hands-on
   - Atividade 1: Somador IEEE754
   - Atividade 2: Multiplicador IEEE754

Introdução

O que é o ponto flutuante?

- Representação de números que pode acomodar uma ampla faixa de valores (alto Dynamic Range).
- Similar ao que fazemos com notação científica.
- Formato IEEE 754 de precisão simples e dupla.

O que é o ponto flutuante?

- Definido pelo Instituto de Engenheiros Eletricistas e Eletrônicos;
- Foi adotado em 1985 e desde então passou por algumas modificações;
- Define algumas regras de normalização a serem seguidas nas operações e representações de números binários com ponto flutuante;
- Antes dele, cada fabricante de computadores e outros dispositivos, possuía um formato de representação diferente.

Formato IEEE 754 - Precisão Simples

- Precisão Simples: 32 bits (1 bit de sinal, 8 bits de expoente, 23 bits de mantissa)

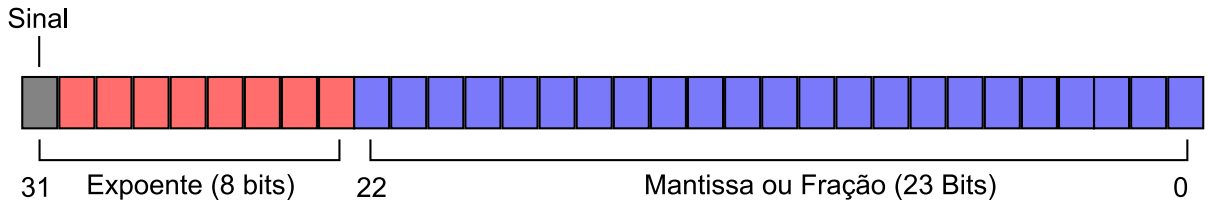


Figura 1: IEE754 Precisão Simples

Formato IEEE 754 - Precisão Dupla

- Precisão Dupla: 64 bits (1 bit de sinal, 11 bits de expoente, 52 bits de mantissa)

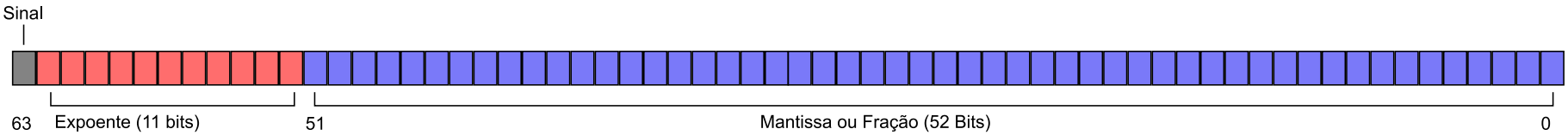


Figura 2: IEEE 754 Precisão Dupla

IEEE 754 - Normatização

Para que o número esteja de acordo com as normas, deve obedecer a seguinte configuração:

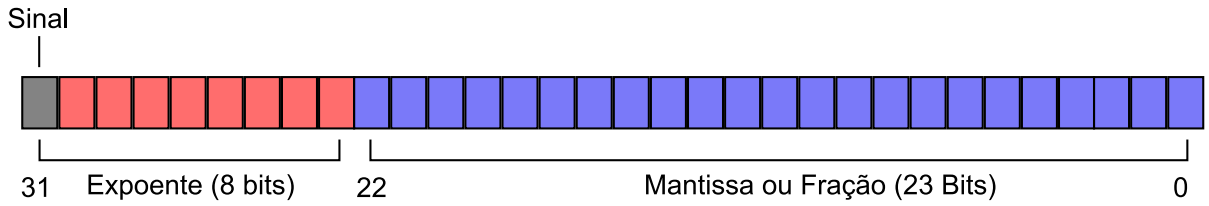
$$S \quad M * 2^{E} \tag{1}$$

Onde:

- S é o sinal (0 significa positivo e 1, negativo);
- M é a mantissa (parte fracionária);
- 2 é a base (binário);
- E é o expoente.

Normatização

IEEE 754 - 32 bits (Precisão Simples)

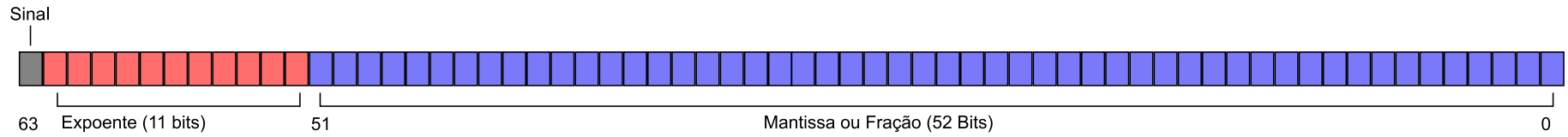


Exemplo:

1. $9,5$ para binário $\implies$ $1001,1$
2. Deslocamos a vírgula $\implies$ $1,0011 * 2^3$
3. Agora que temos o expoente 3, devemos normalizá-lo $\implies$ $3 + 127 = 130$ em binário temos $3 = 11$ e $127 = 1111111$, somando os dois temos 10000010

Resultado: $\underbrace{0}_{Sinal}$ $\underbrace{10000010}_{Expoente}$ $\underbrace{00110000000000000000000}_{Mantissa}$

IEEE 754 - 64 bits (Precisão Dupla)



Exemplo:

1. $9,5$ para binário $\implies$ $1001,1$
2. Deslocamos a vírgula $\implies$ $1,0011 * 2^3$
3. Agora que temos o expoente 3, devemos normalizá-lo $\implies$ $3 + 1023 = 1026$ em binário temos $3 = 11$ e $1023 = 1111111111$, somando os dois temos 10000000010

Resultado: $\underbrace{0}_{Sinal}$ $\underbrace{10000000010}_{Expoente}$ $\underbrace{0011000000000000000000}_{Mantissa}$

Operações com Ponto Flutuante

Soma de 4,75 e 2,125

- Representação em IEEE 754 (precisão simples):
  - 4,750 = 0 10000001 00110000000000000000000
  - 2,125 = 0 10000000 00010000000000000000000
- Reposicionar o 1 Implícito nas mantissas:
  - **1**00110000000000000000000 e **1**00010000000000000000000
- Alinhar os expoentes e deslocar mantissas de acordo:
  - 4,750 = 0 10000001 **1**00110000000000000000000
  - 2,125 = 0 10000001 0**1**00010000000000000000000 (ajustado)
- Somar as mantissas ajustadas:
  - **1**00110000000000000000000 + 0**1**00010000000000000000000
- Reposicionar a Mantissa no Padrão IEEE754 se Necessário
- Ajustar o Expoente caso a Mantissa do Resultado tenha sido reposicionada
- Resultado= 5.8125
  0 10000001 10111000000000000000000

Subtração de 4,75 e 2,125

- Representação em IEEE 754 (precisão simples):
  - 4,750 = 0 10000001 00110000000000000000000
  - 2,125 = 0 10000000 00010000000000000000000
- Reposicionar o 1 Implícito nas mantissas:
  - **1**00110000000000000000000 e **1**00010000000000000000000
- Alinhar os expoentes e deslocar mantissas de acordo:
  - 4,750 = 0 10000001 **1**00110000000000000000000
  - 2,125 = 0 10000001 0**1**00010000000000000000000 (ajustado)
- Subtrair as mantissas ajustadas:
  - **1**00110000000000000000000 - 0**1**00010000000000000000000
- Reposicionar a Mantissa no Padrão IEEE754 se Necessário
- Ajustar o Expoente caso a Mantissa do Resultado tenha sido reposicionada
- Resultado = 3.6875
  0 10000001 01110100000000000000000

Multiplicação de 4,75 e 2,125

- Representação em IEEE 754 (precisão simples):
  - 4,750 = 0 10000001 00110000000000000000000
  - 2,125 = 0 10000000 00010000000000000000000
- Reposicionar o 1 Implícito e Multiplicar as mantissas:
  - **1**00110000000000000000000 * **1**00010000000000000000000
- Soma dos expoentes:
  - $2 + 1 = 3 \rightarrow$ 10000001 + 10000000
- Reposicionar a Mantissa no Padrão IEEE754 se Necessário
- Ajustar o Expoente caso a Mantissa tenha sido reposicionada
- Renormalizar o Expoente:
  - $3 + 127 = 130 \rightarrow$ 00000011 + 11111111 = 10000010
- Resultado = 10.09375
  0 10000010 01000011000000000000000

Divisão de 4,75 e 2,125

- Representação em IEEE 754 (precisão simples):
  - 4,750 = 0 10000001 00110000000000000000000
  - 2,125 = 0 10000000 00010000000000000000000
- Reposicionar o 1 Implícito e Dividir as mantissas:
  - **1**00110000000000000000000 / **1**00010000000000000000000
- Subtração dos expoentes:
  - $2 - 1 = 1 \rightarrow$ 10000001 - 10000000
- Reposicionar a Mantissa no Padrão IEEE754 se Necessário
- Ajustar o Expoente caso a Mantissa tenha sido reposicionada
- Renormalizar o Expoente:
  - $1 + 127 = 128 \rightarrow$ 00000001 + 11111111 = 10000000
- Resultado = 2.23529411765
  0 10000000 00011110000111100001111

Valores Especiais

IEEE 754 - Valores Especiais

Representam valores específicos.

Valor	Sinal	Expoente	Mantissa
Zero	0	0s	0s
+ Infinito	0	1s	0s
- Infinito	1	1s	0s
NaN	0	1s	Diferente de 0s

Hands-on

Hands-On

Atividades:

1. Projetar um Somador em Verilog, que opere operandos em IEEE754 de precisão simples Parta do principio que apenas números positivos serão somados.
   - Simular seu funcionamento através de testbench e verificar a corretude da operação para diferentes entradas binárias.
2. Projetar um Multiplicador em Verilog, que opere operandos em IEEE754 de precisão simples.
   - Simular seu funcionamento através de testbench e verificar a corretude da operação para diferentes entradas binárias.
3. Exploração

Atividade 1: Somador Simples em IEEE754

```
1  module ieee754_adder(
2      input [31:0] a,
3      input [31:0] b,
4      output [31:0] result
5  );
6
7      reg [23:0] mantissa_a, mantissa_b;
8      reg [24:0] mantissa_sum;
9      reg [7:0] exp_diff;
10     reg [7:0] expoente_maior, expoente_final;
11     reg [4:0] shift; // Para armazenar o numero maximo de deslocamentos possiveis (5 bits sao
           suficientes para 24 posicoes)
12
13     always @(*) begin
14         // Extrai as mantissas e adiciona bit implicito
15         mantissa_a = {1'b1, a[22:0]};
16         mantissa_b = {1'b1, b[22:0]};
17
18         // Determina qual numero tem maior expoente
19         if (a[30:23] >= b[30:23]) begin
20             exp_diff = a[30:23] - b[30:23];
21             expoente_maior = a[30:23];
22             mantissa_b = mantissa_b >> exp_diff; // Desloca a mantissa do numero menor
23         end else begin
24             exp_diff = b[30:23] - a[30:23];
25             expoente_maior = b[30:23];
26             mantissa_a = mantissa_a >> exp_diff; // Desloca a mantissa do numero menor
27         end
```

- O código Verilog ajusta os expoentes e as mantissas para somar números em ponto flutuante IEEE 754.

Atividade 1: Continuação

```
28
29          // Soma das mantissas
30          mantissa_sum = mantissa_a + mantissa_b;
31
32          // Normalizacao
33          if (mantissa_sum[24]) begin
34              // Se houve carry (bit extra), deslocamos para a direita e incrementamos o expoente
35              mantissa_sum = mantissa_sum >> 1;
36              expoente_final = expoente_maior + 1;
37          end else begin
38              // Se nao houve carry, encontramos o primeiro bit 1 mais significativo
39              shift = 0;
40              while (mantissa_sum[23 - shift] == 0 && shift < 23) begin
41                  shift = shift + 1;
42              end
43              mantissa_sum = mantissa_sum << shift;
44              expoente_final = expoente_maior - shift;
45          end
46      end
47
48      assign result[31] = 0; // Assumindo que ambos os numeros sao positivos
49      assign result[30:23] = expoente_final;
50      assign result[22:0] = mantissa_sum[22:0]; // Pegamos apenas os 23 bits menos significativos
51
52  endmodule
```

- O código Verilog ajusta os expoentes e as mantissas para somar números em ponto flutuante IEEE 754.

Atividade 1: Testbench Somador Simples em IEEE754

```
1  module ieee754_adder_tb;
2      reg [31:0] a, b;
3      wire [31:0] result;
4
5      ieee754_adder uut (
6          .a(a),
7          .b(b),
8          .result(result)
9      );
10
11     initial begin
12         // Teste de soma: 4,75 + 2,125 = 6,875 => 0_10000001_10111000000000000000000
13         a = 32'b0_10000001_00110000000000000000000; // 4,75
14         b = 32'b0_10000000_00010000000000000000000; // 2,125
15         //add_sub = 0; // Soma
16         #10;
17         $display("Soma: --> Sinal: %b -- Expoente: %b ---- Mantissa: %b", result[31],
               result[30:23], result[22:0]);
18
19         // Teste de soma: 9,5 + 3,75 = 13,25 => 0_10000010_10101000000000000000000
20         a = 32'b0_10000010_00110000000000000000000; // 9,5
21         b = 32'b0_10000000_11100000000000000000000; // 3,75
22         #10;
23         $display("Soma: --> Sinal: %b -- Expoente: %b ---- Mantissa: %b", result[31],
               result[30:23], result[22:0]);
24
25         $finish;
26     end
27 endmodule
```

- O testbench testa a soma dos operandos utilizados nos exemplos de operações aritméticas com IEEE754 apresentados na seção 3.

Atividade 2: Multiplicador Simples em IEEE754

```
1  module ieee754_multiplier (
2      input [31:0] a,
3      input [31:0] b,
4      output reg [31:0] result
5  );
6      wire sign_a, sign_b;
7      wire [7:0] exp_a, exp_b;
8      wire [23:0] mantissa_a, mantissa_b;
9      reg [47:0] mantissa_mul;
10     reg [8:0] exp_result;  // Usar 9 bits para o expoente
11     reg sign_result;
12     reg [23:0] mantissa_result;
13     integer shift;
14
15     // Separar os campos dos numeros de entrada
16     assign sign_a = a[31];
17     assign exp_a = a[30:23];
18     assign mantissa_a = (exp_a == 0) ? 24'd0 : {1'b1, a[22:0]}; // Adicionar bit implicito
19     assign sign_b = b[31];
20     assign exp_b = b[30:23];
21     assign mantissa_b = (exp_b == 0) ? 24'd0 : {1'b1, b[22:0]}; // Adicionar bit implicito
```

- O código Verilog soma os expoentes e multiplica as mantissas para multiplicar números em ponto flutuante IEEE 754.
- A descrição da multiplicação é comportamental. Assim, delega à ferramenta de síntese a implementação do circuito em si. Isso facilita o projeto mas pode gerar problemas em atender constraints de performance. Em geral, a linha 32 do código, onde a multiplicação das mantissas é realmente feita, seria substituída por um multiplicador previamente implementado e o operador de multiplicação (*) não seria utilizado..

Atividade 2: Continuação

```
22
23      always @(*) begin
24          // Tratamento de multiplicacao por zero
25          if ((exp_a == 0 && a[22:0] == 0) || (exp_b == 0 && b[22:0] == 0)) begin
26              result = 32'b0;
27          end else begin
28              // Definir o sinal do resultado
29              sign_result = sign_a ^ sign_b;
30
31              // Multiplicacao das mantissas
32              mantissa_mul = mantissa_a * mantissa_b;
33
34              // Calcular novo expoente (soma dos expoentes com correcao do bias)
35              exp_result = (exp_a - 8'd127) + (exp_b - 8'd127) + 7'd127; // Remover e adicionar o
                    bias de 127
36
37              // Normalizacao: encontrar o primeiro bit 1 mais significativo
38              shift = 0;
39              while (mantissa_mul[47 - shift] == 0 && shift < 48) begin
40                  shift = shift + 1;
41              end
42
43              mantissa_result = mantissa_mul[47:24] << shift; // Apenas os 23 bits mais
                    significativos
44
45              // Ajustar o expoente pela normalizacao
46              exp_result = exp_result - shift + 1; // Subtrair o numero de deslocamentos ate apos
                    o 1 mais significativo
47
48              // Construir o resultado final
49              result = {sign_result, exp_result[7:0], mantissa_result[22:0]};
50          end
51      end
52  endmodule
```

Atividade 2: Testbench Multiplicador Simples

```
1  module ieee754_multiplier_tb;
2      reg [31:0] a, b;
3      //reg mul_div;
4      wire [31:0] result;
5
6      ieee754_multiplier uut (
7          .a(a),
8          .b(b),
9          .result(result)
10     );
11
12     initial begin
13         // Teste de multiplicacao: 4,75 * 2,125 = 10,09375 => 0_10000010_01000110000000000000000
14         a = 32'b0_10000001_00110000000000000000000; // 4,75
15         b = 32'b0_10000000_00010000000000000000000; // 2,125
16         #10;
17         $display("Multiplicacao: --> Sinal: %b -- Expoente: %b ---- Mantissa: %b", result[31],
               result[30:23], result[22:0]);
18
19         // Teste de multiplicacao: 9,5 * 3,75 = 35,625 => 0_10000100_00011101000000000000000
20         a = 32'b0_10000010_00110000000000000000000; // 9,5
21         b = 32'b0_10000000_11100000000000000000000; // 3,75
22         #10;
23         $display("Multiplicacao: --> Sinal: %b -- Expoente: %b ---- Mantissa: %b", result[31],
               result[30:23], result[22:0]);
24
25         $finish;
26     end
27 endmodule
```

- O testbench testa a multiplicação dos operandos utilizados nos exemplos de operações aritméticas com IEEE754 apresentados na seção 3.

Atividade 3: Exploração

1. Utilizando os códigos fornecidos, teste outros operandos para as operações de Soma e Multiplicação. Verifique os resultados.
2. Utilizando os códigos fornecidos na subseção 1, tente gerar um subtrator simples (pode utilizar a mesma premissa dos operandos positivos).
3. Utilizando os códigos fornecidos na subseção 2, tente gerar um divisor simples.

Conclusão

- Aritmética de ponto fixo e ponto flutuante são fundamentais para a representação e manipulação de números em sistemas digitais.
- Ponto flutuante (IEEE 754) oferece uma ampla faixa de valores e é ideal para cálculos científicos e de engenharia.
- Compreender as operações de soma, subtração, multiplicação e divisão é essencial para o desenvolvimento de algoritmos e hardware eficientes.
- Implementações em Verilog permitem a realização de operações aritméticas em hardware, proporcionando desempenho elevado em aplicações práticas.