Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0121 –Arquitetura de Computadores

Aritmética Binária

Prof. Gustavo Girão
girao@imd.ufrn.br

Baseado no material do Prof. Ricardo Weber (UFRGS)

Roteiro

- Números com sinal e sem sinal
- Adição e Subtração
- Multiplicação
- Divisão
- Representação de ponto flutuante.

Introdução

- Objetivos desta aula
  - ***Descrever como os números negativos são representados.***
  - ***Mostrar o que acontece se uma operação cria um número muito maior do que poderia ser representado***
  - ***Mostrar como são implementadas as operações aritméticas***

Números com Sinal e sem Sinal

- Existem algumas abordagens para representar números positivos e negativos em um sistema computacional.
  - Veremos 3 diferentes abordagens:
    - Sinal e Magnitude.
    - Complemento de 1.
    - Complemento de 2.

Números com Sinal e sem Sinal

- Sinal e magnitude.
  - O número binário possui um bit específico para tratamento do sinal.
    - Este bit é o primeiro bit. Quando 1 negativo, quando 0 positivo
  - Os demais bits representam o número.
  - Exemplos:

000 = +0
001 = +1
010 = +2
011 = +3
100 = –0
101 = –1
110 = –2
111 = –3

Números com Sinal e sem Sinal

- Sinal e Magnitude
  - Problemas:
    - 0 positivo e 0 negativo.
    - Implementação em hardware mais complicada.
    - Operações aritméticas se tornam mais complicadas.

Números com Sinal e sem Sinal

- Complemento de 1.
  - Melhoria da representação de sinal e magnitude. Utilizado ainda em alguns sistemas computacionais.
  - Utiliza 1 bit para indicar o sinal
  - Exemplos:

000 = +0
001 = +1
010 = +2
011 = +3
100 = –3
101 = –2
110 = –1
111 = –0

Números com Sinal e sem Sinal

- Como é calculado um número em complemento de 1.
- Para números positivos, procedimento normal.
  - 0000 0111 = 7
- Para números negativos
  - 1000 0111 = - 120.
- Para calcular o número negativo basta **negarmos** a parte sem sinal
- No exemplo acima
  - 1 ***000 0111*** = ***1111 000 = 120***, como o sinal do número é 1, então fica -120.

Números com Sinal e sem Sinal

- Complemento de 1
  - Ainda possui 0 positivo e 0 negativo
  - Também problemático para algumas operações aritméticas

Números com Sinal e sem Sinal

- Complemento de 2
  - Melhoria da representação de complemento de 1, mais utilizada nos sistemas computacionais atuais.
  - Utiliza 1 bit para indicar o sinal
  - Exemplos:

000 = +0
001 = +1
010 = +2
011 = +3
100 = –4
101 = –3
110 = –2
111 = –1

Números com Sinal e sem Sinal

- Como é calculado um número em complemento de 2.
- Para números positivos, procedimento normal.
  - 0000 0111 = 7
- Para números negativos
  - 1000 0111 = - 121.
- Para calcular o número negativo basta negarmos a parte sem sinal (complemento de 1) e somarmos com 1.
- No exemplo acima
  - 1 ***000 0111*** = ***1111 000 + 1 = 121***, como o sinal do número é 1, então fica -121.

Números com Sinal e sem Sinal

- Complemento de 2
  - Só possui 0 (como positivo)
  - Facilita em muito as operações aritméticas (veremos isso nos próximos slides)

Números com Sinal e sem Sinal

- Grande parte dos processadores dão suporte a instruções com sinal ou sem sinal.
- Basicamente, nas instruções sem sinal, o bit de sinal será considerado como bit integrante do número.
  - Todos os números são calculados como se fossem números positivos.
- Muitas das instruções que veremos para o Mips, possuem uma instrução correspondente para fazer a avaliação de números sem levar em conta o sinal.
  - Exemplos
    - sltu, sltiu, addu, subu, etc.

Números com Sinal e sem Sinal

- Adição e Subtração
  - São as operações aritméticas mais simples para os processadores.
  - As operações funciona da mesmo forma que aprendemos na escola, só que agora, com números binários.
  - Exemplos:

0111	0111	0110
+ 0110	− 0110	− 0101

Números com Sinal e sem Sinal

- Adição e Subtração
  - São as operações aritméticas mais simples para os processadores.
  - As operações funciona da mesmo forma que aprendemos na escola, só que agora, com números binários.
  - Exemplos:

0111	0111	0110
+ 0110	− 0110	− 0101
1101	*0001*	*0001*

Números com Sinal e sem Sinal

- Adição e Subtração
  - A notação complemento de 2 facilita estas tarefas.
  - A seguinte subtração 7 – 6 pode ser feita das seguintes formas:
    - ✧ Forma tradicional

$$
\begin{array}{lll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} & = 7_{ten} \\
- & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} & = 6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} & = 1_{ten}
\end{array}
$$

Números com Sinal e sem Sinal

- Adição e Subtração
  - A notação complemento de 2 facilita estas tarefas.
  - A seguinte subtração 7 – 6 pode ser feita das seguintes formas:
    - Complemento de 2 (como uma soma).

$$
\begin{array}{lll}
 & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,0111_{two} & = 7_{ten} \\
+ & 1111\,1111\,1111\,1111\,1111\,1111\,1111\,1010_{two} & = -6_{ten} \\
\hline
= & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,0001_{two} & = 1_{ten}
\end{array}
$$

Números com Sinal e sem Sinal

- Observem que no último caso tivemos um pequeno problema:
  - A soma resultou em um valor que não pode ser representado em 32 bits.
  - Quando isso acontece, dizemos que ocorreu ***overflow.***
  - Para os casos, como do exemplo acima, o overflow não teria maiores problemas, visto que podemos descartar o dígito 33 sem quaisquer problemas.

Aritmética binária avançada

- Detectando Overflow
- Multiplicação
- Divisão
- Ponto flutuante.

Detectando Overflow

- Como vimos, operações aritméticas estão sujeitas a *overflow*.
- O overflow acontece nos seguintes casos:
  - Ao somarmos dois positivos, obtemos um negativo.
  - Ao somarmos dois negativos, obtemos um positivo.
  - Ao subtrairmos um negativo de um positivo e obtemos um negativo.
  - Ao subtrairmos um positivo de um negativo e obtemos um positivo.

Detectando Overflow

- O quadro a seguir ilustram estes casos:

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

- Pode ocorrer overflow se A for 0?

Efeitos do overflow

- Se o bit extra gerado, for necessário, pode acontecer uma interrupção, (exceção):
  - Basicamente, ao detectar a interrupção, o controle saltará para um endereço predefinido para tratamento da interrupção.
  - O endereço interrompido é salvo para uma possível retomada.
    - No caso do MIPS, o endereço de retorno é salvo no registrador EPC (Exception program counter).
    - De novo, veremos estes detalhes na unidade 2.

Efeitos do overflow

- A linguagem de programação, bem como a finalidade da aplicação pode influir para este comportamento.
  - É provável que em um sistema de controle de vôo o software tenha uma preocupação maior com interrupção do que em uma lista de exercícios.
- O MIPS introduz algumas novas instruções (unsigned) para que overflow não seja detectado.
  - addu, addiu, subu.

Multiplicação

- Operação mais complexa do que adição
  - Na verdade ela representa um conjunto de adições.
- Necessita de mais tempo e mais **área de silício** para ser implementada
- Multiplicação quando operando é negativo:
  - Converta e multiplique.

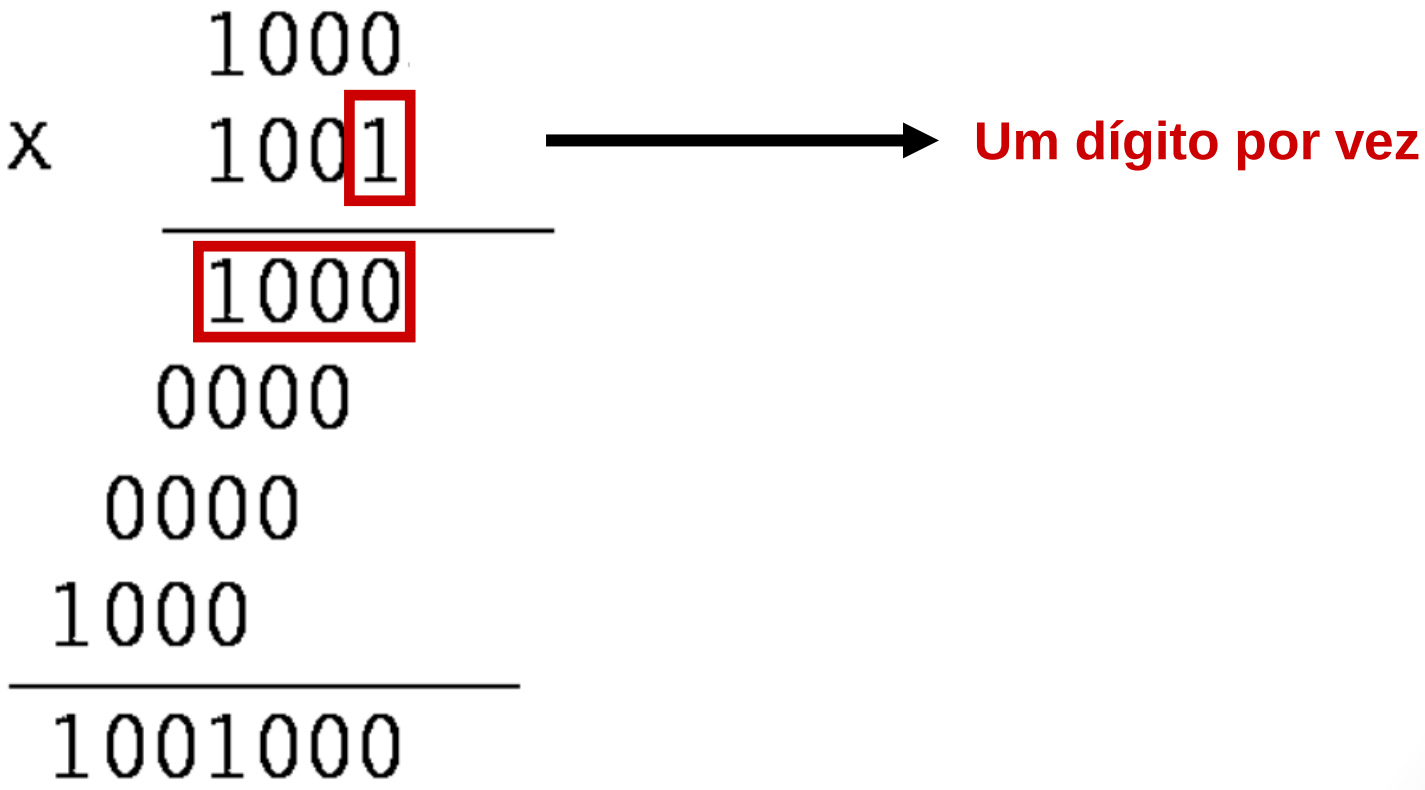
Multiplicação

- De volta ao primeiro grau ☺

```
    1000
x   1001
  ------
    1000
   0000
  0000
 1000
------
1001000
```

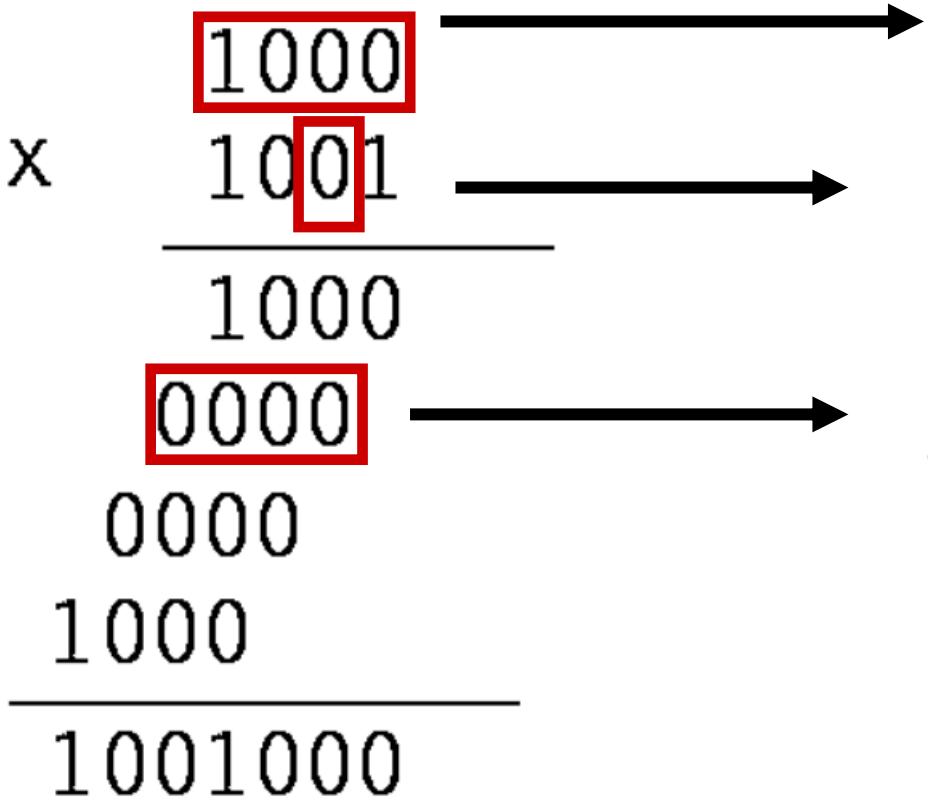
Multiplicação

- De volta ao primeiro grau ☺



Multiplicação

- De volta ao primeiro grau ☺

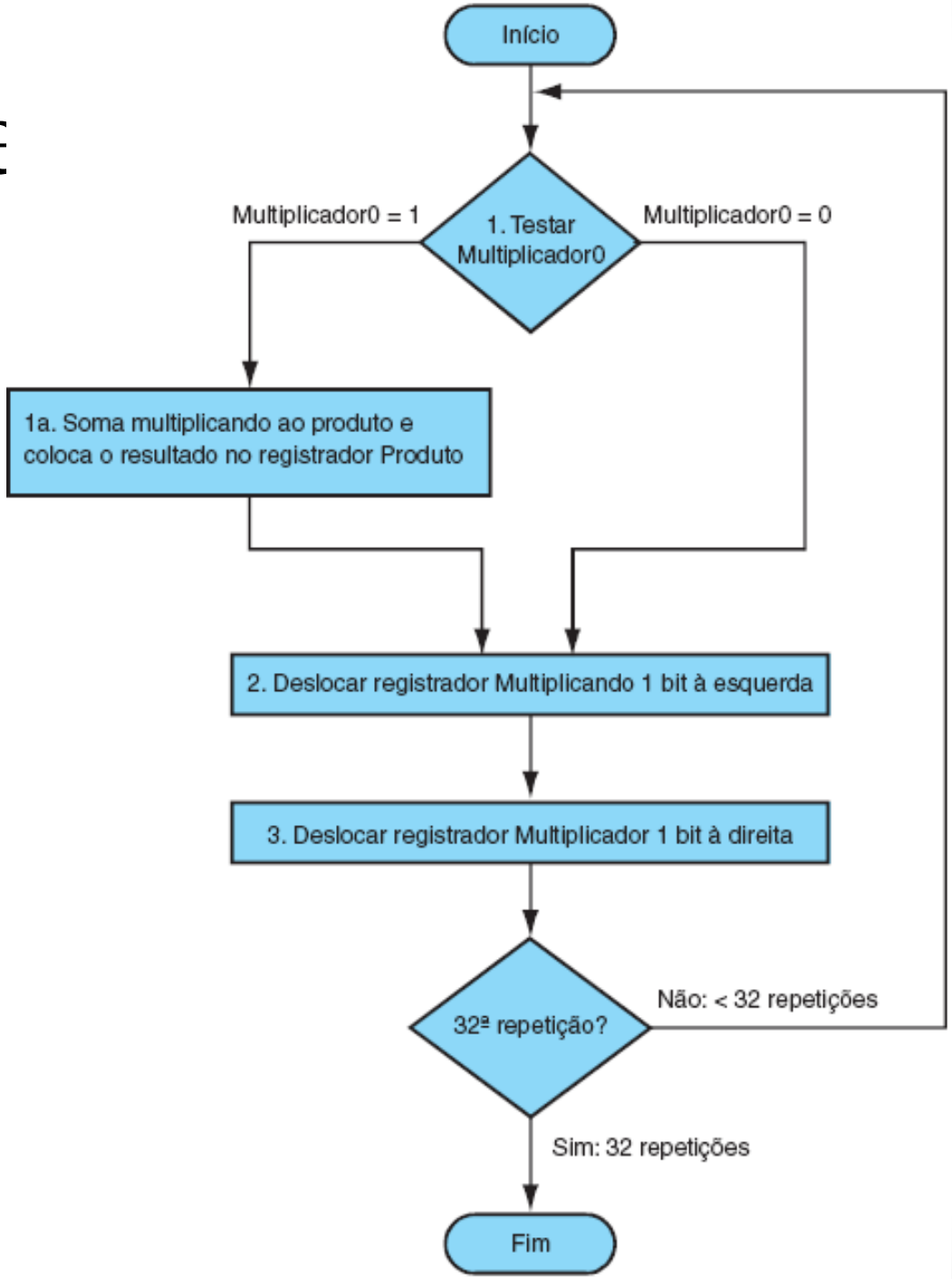


Deslocamento para esquerda

Deslocamento para direita

É como se estívessemos incluído mais um 0 no multiplicando

Controle


Início
1. Testar Multiplicador0
Multiplicador0 = 1
Multiplicador0 = 0
1a. Soma multiplicando ao produto e coloca o resultado no registrador Produto
2. Deslocar registrador Multiplicando 1 bit à esquerda
3. Deslocar registrador Multiplicador 1 bit à direita
32ª repetição?
Não: < 32 repetições
Sim: 32 repetições
Fim

Divisão

- Menos freqüente e mais peculiar que a multiplicação.
- Pode ser efetuada através de sucessivas subtrações e deslocamentos.

Divisão: Como o hardware trabalha

Divisor 1000_{ten} | 1001010_{ten} Dividend

Divisão: Como o hardware trabalha

$$
\begin{array}{rrl}
 & 1 & \\
\text{Divisor } 1000_{ten} & \overline{)\,1001010_{ten}} & \text{Dividend} \\
 & -1000 & \\
 & \overline{1} &
\end{array}
$$

Divisão: Como o hardware trabalha

$$
\begin{array}{rr}
 & 10 \\
\text{Divisor } 1000_{\text{ten}} & \overline{\big|\ 1001010_{\text{ten}}} \qquad \text{Dividend} \\
 & \underline{-1000} \\
 & 10
\end{array}
$$

Divisão: Como o hardware trabalha

$$
\begin{array}{rr}
 & 100 \\
\text{Divisor } 1000_{ten} & \overline{)\ 1001010_{ten}} \qquad \text{Dividend} \\
 & -1000 \\
 & \hline
 & 10 \\
 & 101
\end{array}
$$

Divisão: Como o hardware trabalha

1001_{ten} Quotient

Divisor 1000_{ten} | 1001010_{ten} Dividend

-1000

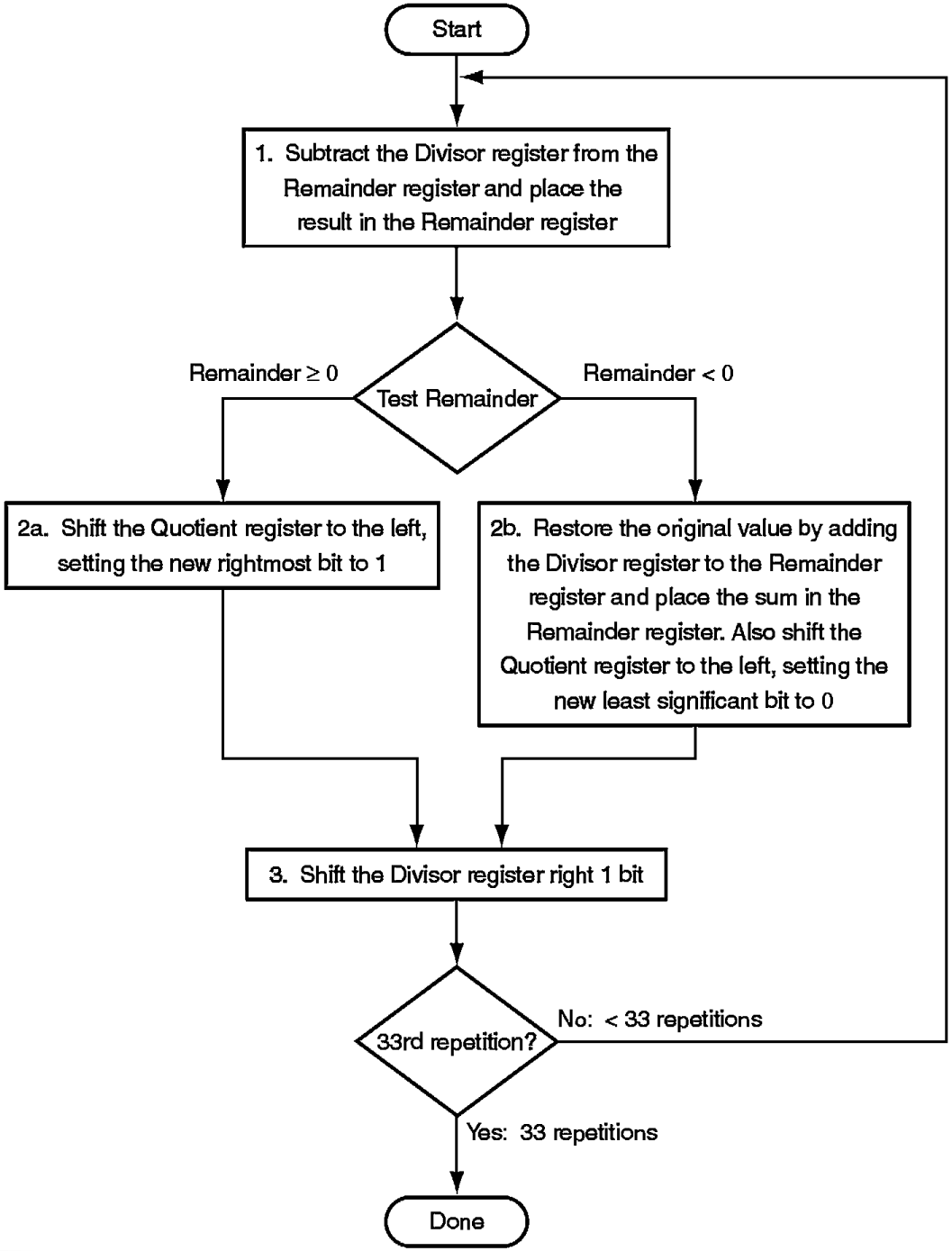
10

101

1010

-1000

10_{ten} Remainder


Start
1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register
Test Remainder
Remainder ≥ 0
Remainder < 0
2a. Shift the Quotient register to the left, setting the new rightmost bit to 1
2b. Restore the original value by adding the Divisor register to the Remainder register and place the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0
3. Shift the Divisor register right 1 bit
33rd repetition?
No: < 33 repetitions
Yes: 33 repetitions
Done

REPRESENTAÇÃO DE PONTO FLUTUANTE

Ponto Flutuante

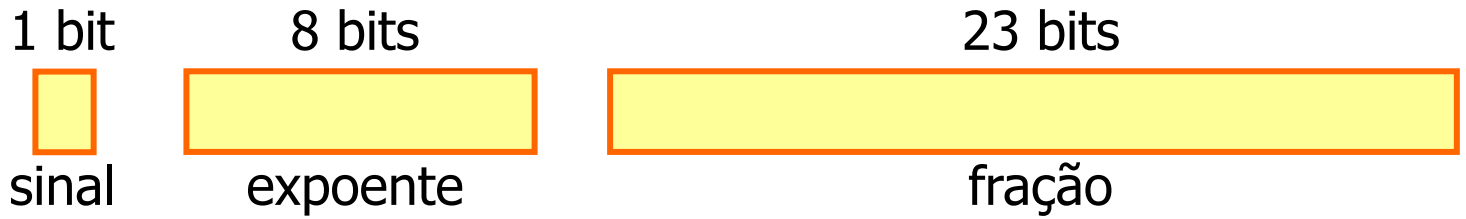
- Precisamos de uma maneira para representar
  - Números com frações, por exemplo, 3,1416
  - Números muito pequenos, por exemplo, 0,00000001
  - Números muito grandes, por exemplo, $3{,}15576 \times 10^9$
- Uma boa representação para os valores expressos anteriormente é a notação científica.
  - $3{,}0 \times 10^0$
- No caso de números binários a notação científica equivalente seria:
  - $1{,}1 \times 2^1$

Ponto Flutuante

- Representação
  - Sinal, expoente, significando
    - $(-1)^{sinal}$ x fração x $2^{expoente}$
    - Mais bits para a fração fornece mais precisão
    - Mais bits para o expoente, aumenta o range de valores.
- Os computadores, em geral, utilizam o padrão de ponto flutuante IEEE 754.
  - Precisão única (float): expoente de 8 bits, fração de 23 bits.
  - Precisão dupla (double): expoente de 11 bits, fração de 52 bits.

Ponto flutuante (Padrão IEEE 754)

- O formato de precisão simples (**`float`**) ocupa **32**



- O formato de precisão dupla (**`double`**) ocupa **64 bits**.

Ponto flutuante (Padrão IEEE 754)

- O bit mais à esquerda guarda o sinal do número:
  - **bit = 0** → número positivo
  - **bit = 1** → número negativo

Ponto flutuante (Padrão IEEE 754)

- A **mantissa** é representada na forma normalizada (base binária):

$$1.b_{-1}b_{-2}b_{-3}\ldots$$

- A **mantissa** é composta por:
  - **Algarismo 1**
  - **Ponto de separação**
  - **Fração**

Ponto flutuante (Padrão IEEE 754)

- O algarismo 1 e o ponto de numeração **não precisam ser armazenados**, pois são os mesmos para todos os números reais representados.

- Caso a fração possua menos bits que o esperado, zeros devem ser colocados **à direita**, pois não têm significância.

fração = 1,110011

23 bits

11001100000000000000000

fração

Ponto flutuante (Padrão IEEE 754)

- Por razões históricas, o co-processador de ponto flutuante Intel não utiliza parte inteira implícita, ou seja, a parte inteira também é representada juntamente com a fração.

- O formato de precisão estendida ocupa 80 bits.

Ponto flutuante (Padrão IEEE 754)

- O expoente é representado na **notação deslocada**, ou excesso de N

- Maior expoente representável: 2^{n-1}
  - Representado por: **11...11**

- Menor expoente representável: $-(2^{n-1} - 1)$
  - Representado por: **00...00**

Ponto flutuante (Padrão IEEE 754)

- **Notação excesso de N**

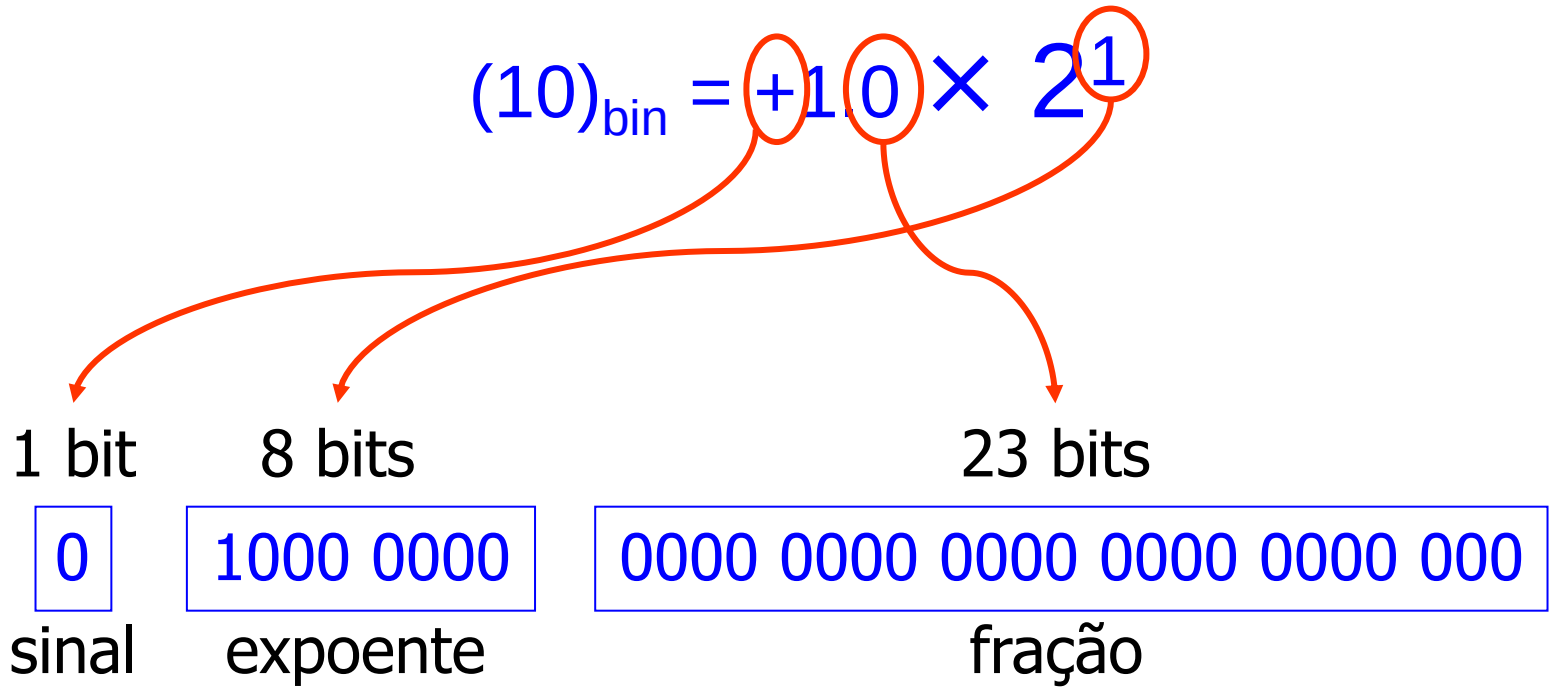
Decimal	Complemento de dois	Notação excesso de N
+4	--	--
+3	011	111
+2	010	110
+1	001	101
0	000	100
-1	111	011
-2	110	010
-3	101	001
-4	100	000

Ponto flutuante (Padrão IEEE 754)

- Notação deslocada
  - Representação do valor **zero**: **01...11**.
  - Representação do valor **um**: **10...00**.
  - Demais valores: **somar ao zero (deslocamento)**.

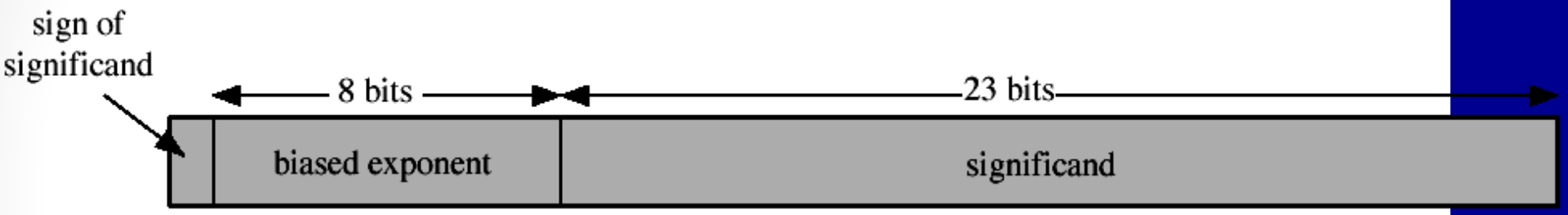
Ponto flutuante (Padrão IEEE 754)

- Exemplo:



Ponto flutuante (Padrão IEEE 754)

- Mais exemplos:



$$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$$
$$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$$
$$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$$
$$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$$

fração em binário

expoente não sinalizado

float

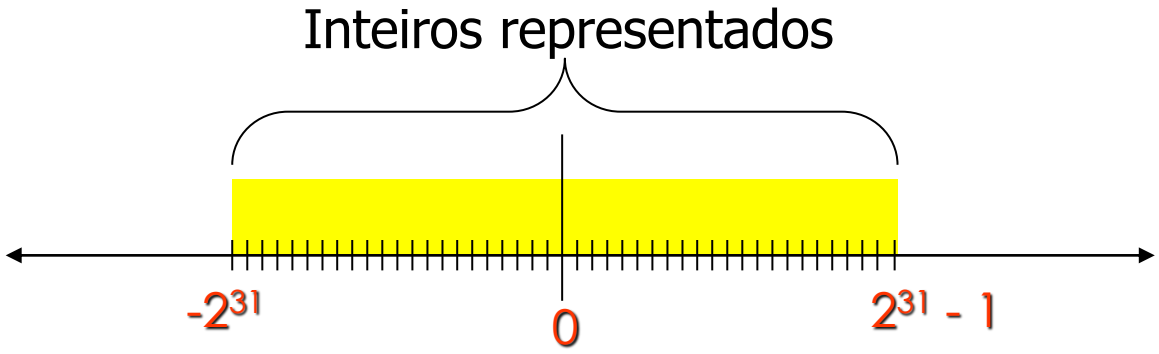
fração em decimal

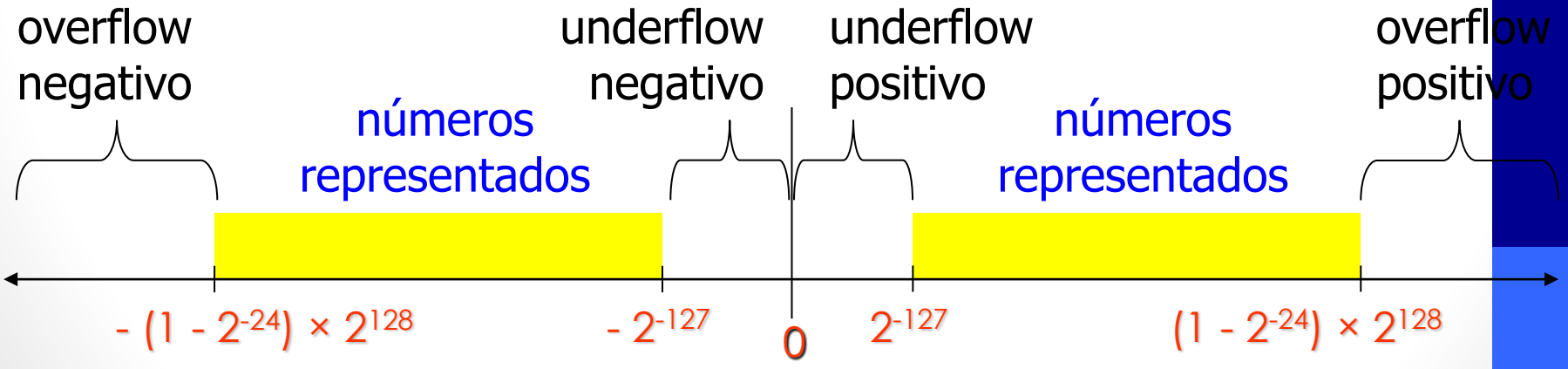
expoente decimal

Ponto flutuante: Operações aritméticas

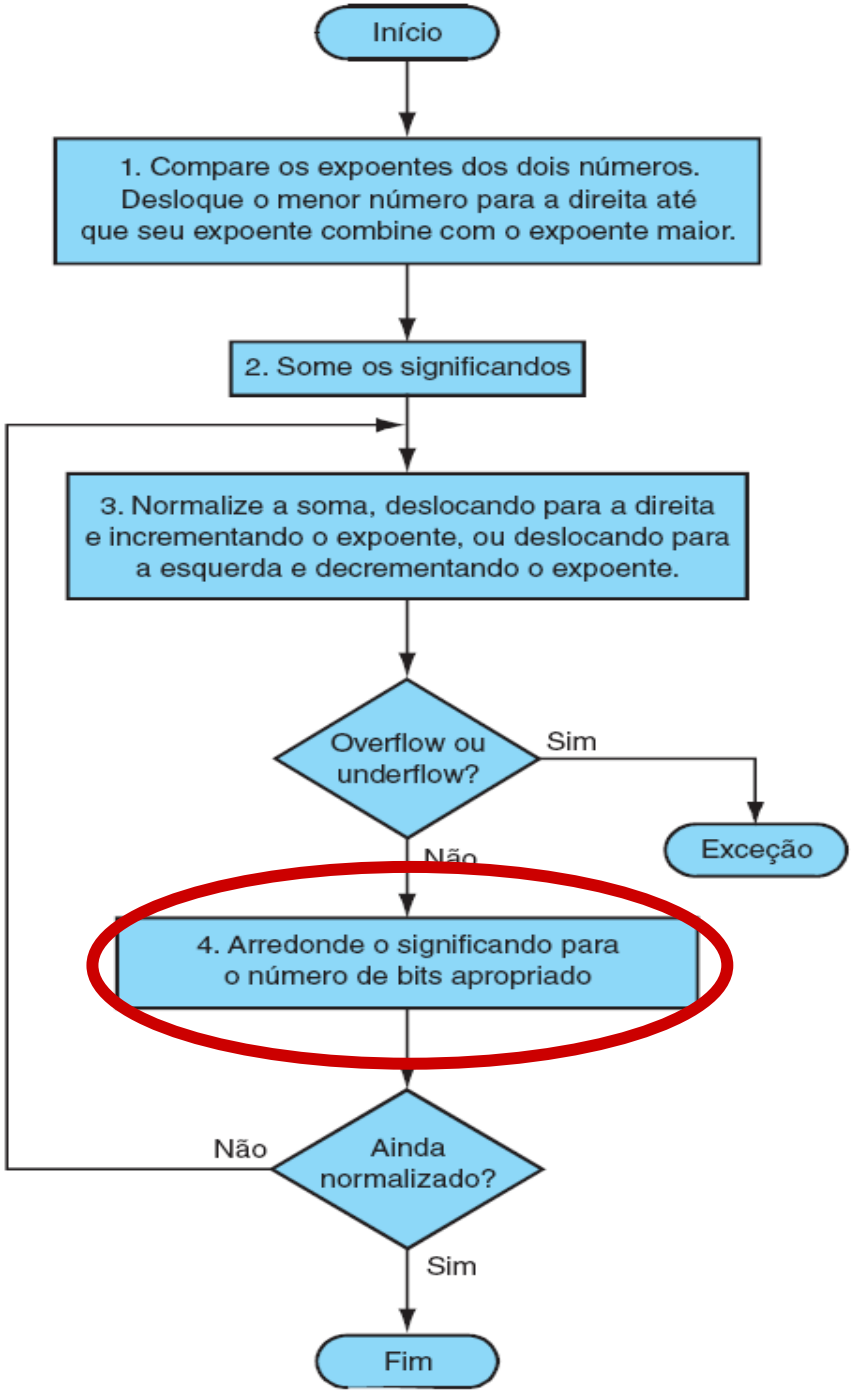
- Operações aritméticas envolvendo ponto flutuantes sofrem do mesmo problema apresentado nas operações inteiras (overflow).
- Além de overflow, operações de ponto flutuante podem também resultar em underflow
  - Underflow é quando o resultado obtido é pequeno demais para ser representado em um número de ponto flutuante.

Ponto flutuante × Ponto fixo

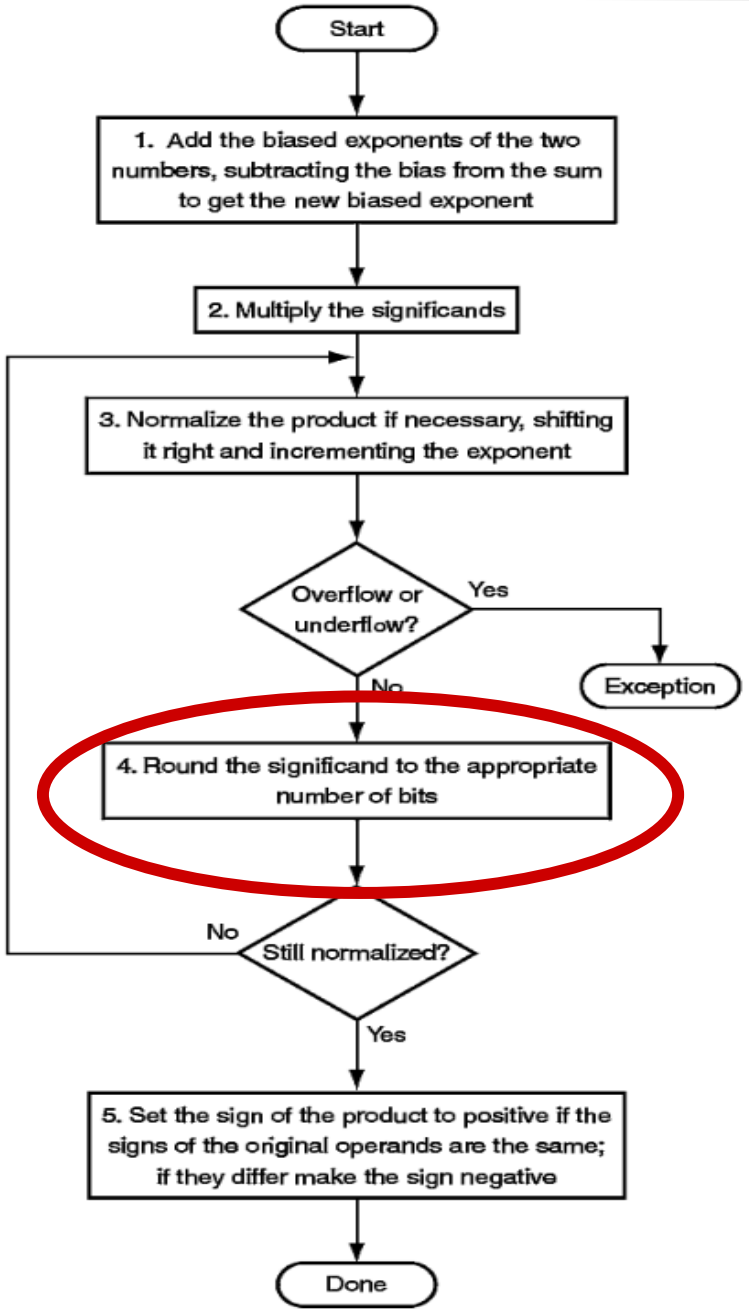




Ponto flutuante
Adição



Ponto Flutuante: Multiplicação



Ponto Flutuante: Arredondamento

- Além do underflow, podemos ter problemas relativo a forma de arredondamento.
- Basicamente existem quatro formas de arredondamento:
  - Sempre arredondar para cima.
  - Sempre arredondar para baixo.
  - Truncamento.
  - Arredondar para o próximo par.

Referências

- **STALLINGS, William. Arquitetura e organização de computadores. 10. ed. São Paulo: Pearson, 2017. 814 p.**
  - **Capítulo 9**

- TOCCI, Ronald J; Widmer, Neal S. Sistemas Digitais: principios e Aplicações. 11. ed. São Paulo SP: Pearson, 2011, 817 p. ISBN 9788576050957
  - Capitulo 1

- PATTERSON, David A; HENNESSY, John L. Organização e projeto de computadores: A interface HARDWARE/SOFTWARE. Rio de Janeiro: Elsevier, 2005, 3ª edição.

Próxima Aula

- Funções lógicas básicas