Estouro e Representação em Ponto Flutuante

Cristina Boeres

Insituto de Computação (UFF)

Fundamentos de Arquiteturas de Computadores

Material baseado nos slides de Fernanda Passos

Estouro: Introdução

- Considere a seguinte situação:
  - Computador usa Representação em Complemento a Dois com 8 bits.
  - Deseja-se realizar a soma de dois valores nesta representação: 01101110 e 01011010

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1\ 1\ \ } \\ 01101110 \\ +\ 01011010 \\ \hline 11001000 \end{array}$$

- Algum problema?
  - Aparentemente – uma operação simples

Estouro: Introdução

- Repare no que ocorreu:
  - Ao somar dois números **positivos** (01101110 e 01011010)
  - resultado **negativo** (11001000)!
- O que aconteceu?

- Vamos analisar a soma novamente, mas desta vez em base 10.
  - Primeira parcela: 01101110 em Complemento a Dois representa $110_{(10)}$.
  - Segunda parcela: 01011010 em Complemento a Dois representa $90_{(10)}$.
  - Resultado obtido: 11001000 em Complemento a Dois representa $-56_{(10)}$.
  - Resultado **esperado**: $200_{(10)}$.

Estouro: Introdução

- O resultado da soma é muito **grande** para ser representado com 8 bits em Complemento a Dois
  - Precisaríamos de pelo menos 9
    - ★ 1 bit para sinal
    - ★ 8 bits para magnitude
  - A operação aritmética resulta em um valor **fora da faixa**
    - ★ Outros exemplos com 8 bits: $100_{(10)} + 50_{(10)}$, $80_{(10)} + 80_{(10)}$, ...
- Neste caso, dizemos que houve um **estouro**
  - **overflow**.

Estouro: Formalização

- Estouro: quando número de bits é **insuficiente** para representar o resultado

- Pode ocorrer em qualquer representação
  - somar $101_{(10)}$ e $37_{(10)}$ em Sinal e Magnitude com 8 bits
    - ★ $101_{(10)}$ tem representação 01100101.
    - ★ $37_{(10)}$ tem representação 00100101.
    - ★ Somando as representações, obtemos 10001010, um número negativo
  - somar $-90_{(10)}$ e $-40_{(10)}$ em Complemento a Dois com 8 bits
    - ★ $-90_{(10)}$ tem representação 10100110
    - ★ $-40_{(10)}$ tem representação 11011000
    - ★ Somando as representações (e ignorando bit mais significativo): 01111110, um número positivo

Estouro: Como Detectar?

- Uma maneira de detectar se houver *overflow* é converter os operandos para base 10 e realizar a operação.
  - Se o resultado está fora da faixa de valores da representação, há *overflow*
- Um computador binário não trabalha deste modo
- Detecção tem que ser baseada apenas nos bits manipulados
- Em Complemento a Dois, podemos montar uma tabela
  - Onde c_i representa o vai-um (*carry out*) na posição i

Estouro no Complemento a Dois: Exemplos

- Exemplos com 8 bits:

$$\begin{array}{r} \overset{\text{Overflow}}{\overset{\boxed{0\ 1}\ 1}{00101000}} \\ +\ 01110001 \\ \hline 10011001 \end{array} \qquad \begin{array}{r} \overset{\text{Sem Overflow}}{\overset{\boxed{1\ 1}\ 1}{10101000}} \\ +\ 01110001 \\ \hline 00011001 \end{array} \qquad \begin{array}{r} \overset{\text{Overflow}}{\overset{\boxed{1\ 0}}{10001110}} \\ +\ 11110001 \\ \hline 01111111 \end{array} \qquad \begin{array}{r} \overset{\text{Sem Overflow}}{\overset{\boxed{0\ 0}\ 1}{00101000}} \\ +\ 01001111 \\ \hline 01110111 \end{array}$$

Estouro: Como Detectar?

Sejam n bits, as posições dos bits vão de 0 a $n-1$, sendo e a posição n adicional. A tabela considera todas as combinações:

a_{n-1}	b_{n-1}	c_{n-1}	c_n	Overflow?
0	0	0	0	Não
0	0	1	0	Sim
0	1	0	0	Não
0	1	1	1	Não
1	0	0	0	Não
1	0	1	1	Não
1	1	0	1	Sim
1	1	1	1	Não

Conclusão: *overflow* existe se, e somente se, $c_{n-1} \neq c_n$

Exercícios

- Considere as seguintes somas de valores representados em Complemento a Dois:
  - $001011 + 100001$ com seis bits.
  - $101011 + 111001$ com seis bits.
  - $1011 + 1101$ com quatro bits.
  - $0011 + 0101$ com quatro bits.
  - $100001 + 100001$ com seis bits.
- Sem converter os valores para a base 10, determine quais operações resultam em *overflow*.

Representação em Ponto Flutuante

Ponto Flutuante: Introdução

- Esquemas de representação vistos até aqui lidam com números negativos
- Mas computadores também precisam lidar com números não-inteiros
  - "Números com vírgula"
- Nenhuma das representações vistas até aqui é capaz de fazer isso.
- Precisamos, portanto, de um esquema de representação específico para este tipo de dado

Ponto Flutuante *vs.* Ponto Fixo

Primeira abordagem:Representação por Ponto Fixo

- Se temos n bits para a representação, arbitramos uma vírgula subentendida em algum ponto
- À direita do bit k, para algum $k < n$.

Exemplo, com $n = 8$ e $k = 2$

- $10{,}25_{(10)} = 1010{,}01_{(2)}$
- Representação em Ponto Fixo: 00101001
- Vírgula subentendida depois do segundo bit, da direita para a esquerda

Mesmo valor, com $n = 8$ e $k = 3$

- Representação em Ponto Fixo: 01010010
- Vírgula subentendida depois do terceiro bit, da direita para a esquerda.

Ponto Flutuante *vs.* Ponto Fixo

- Operações de soma e subtração sobre números em Ponto Fixo podem ser executadas como se a vírgula não existisse.
  - Isto é, como se fossem números inteiros.
  - Porque ela está sempre no mesmo lugar para todos os números.
- Valores negativos podem ser representados com um bit de sinal.
  - Como em Sinal e Magnitude
  - Operações de soma e subtração passam a ser realizadas como nesta representação

Ponto Flutuante *vs.* Ponto Fixo

- A representação em Ponto Fixo é efetivamente uma solução para os números não-inteiros
- Mas existe um problema fundamental:
  - Como escolher a posição da vírgula?
- Esta questão é importante por dois motivos:
  - Se o ponto (vírgula) é fixo, ele está no mesmo lugar para todos os números.
  - A posição do ponto impõe um compromisso entre **precisão** e **abrangência**.
    - ★ **Precisão:** o quão exata é a representação de um número
    - ★ **Abrangência:** o quão larga é a faixa de valores representáveis

Ponto Flutuante *vs.* Ponto Fixo

- Considere uma representação em Ponto Fixo com $n = 4$ bits

- **Seja** $k = 1$
  - Apenas um bit à direita da vírgula, três à esquerda
  - Parte inteira pode ser qualquer valor de 0 a 7
  - Parte fracionária é restrita a 0 ou 0,5
  - Não temos **precisão** suficiente para representar 2,3, por exemplo
    - ★ Tem que ser **aproximado** para 2,0 ou 2,5

Ponto Flutuante *vs.* Ponto Fixo

- **Agora seja $k = 2$**

  - Dois bits à direita da vírgula, dois à esquerda
  - Parte inteira pode ser qualquer valor de 0 a 4
  - Parte fracionária é restrita a 0, 0,25, 0,5, ou 0,75
  - Precisão melhorou: 2,3 (por exemplo) pode ser aproximado por 2.25 (erro menor)
  - Mas o **maior número representável** agora é 3,75
    - ★ Perdemos **abrangência**

Ponto Flutuante: Balanceando Abrangência e Precisão

- A decisão sobre a posição da vírgula no Ponto Fixo é difícil
  - Na verdade, em computadores de **propósito geral**, não existe uma decisão perfeita
  - Aplicações diferentes têm requisitos diferentes
- Por este motivo, uma abordagem alternativa pode ser mais interessante
- Chega-se assim à representação de **Ponto Flutuante**
  - Ideia de permitir a "movimentação" da vírgula
  - Para números grandes, vírgula fica mais à direita
  - Para números pequenos, vírgula fica mais à esquerda

Ponto Flutuante: Notação Científica Normalizada

- Um problema desta vírgula móvel é como codificar esta posição em bits
- Uma ideia é armazená-la em um conjunto de bits auxiliar
  - Por exemplo, com 8 bits no total, podemos separar:
    - ★ 5 bits para o número em si
    - ★ 3 bits para a posição da vírgula
  - Exemplos de aplicação:
    - ★ $4{,}75_{(10)}$ seria representado por 10011 e posição da vírgula 010.
    - ★ $8{,}5_{(10)}$ seria representado por 10001 e posição da vírgula 001.
    - ★ $22_{(10)}$ seria representado por 10110 e posição da vírgula 000.

Ponto Flutuante: Notação Científica Normalizada

- Problema desta codificação:
  - um mesmo valor pode ter múltiplas representações
  - *e.g.*, $0{,}5_{(10)}$ tem 5 representações:
    - ★ 00001 e posição 001.
    - ★ 00010 e posição 010.
    - ★ 00100 e posição 011.
    - ★ 01000 e posição 100.
    - ★ 10000 e posição 101.
  - Desperdiça bits $\rightarrow$ reduz precisão e abrangência.

Ponto Flutuante: Notação Científica Normalizada

- Uma solução melhor, embora parecida, é representar valores na **Notação Científica Normalizada**.
- Em notação científica, números são escritos como:

$$\pm\, x,M \times b^E$$

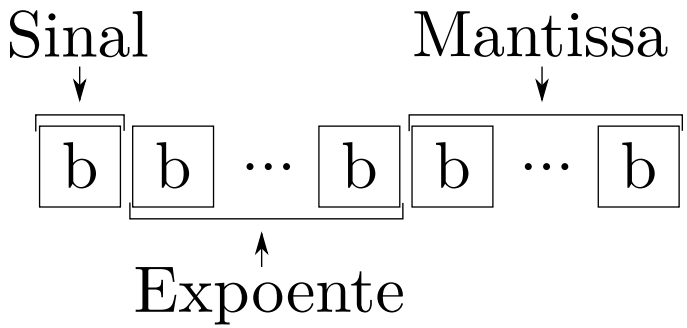
- Onde:
  - b é a base na qual o número está escrito.
  - x é um algarismo na base b antes da vírgula.
  - M é chamado de **mantissa**.
  - E é chamado de **expoente**, ou ordem de grandeza.
- Na notação científica **normalizada**, há uma restrição adicional:
  - Obrigatoriamente, $1 \leq x < b$.

Ponto Flutuante: Notação Científica

- Esta restrição adicional faz com que a representação de um número seja única (em uma dada base)
  - A posição da vírgula é sempre no mesmo lugar.
- Com a notação científica normalizada evita-se o desperdício de bits
- Basicamente, usaremos três conjuntos de bits:
  - A mantissa (M)
  - O expoente (E)
  - O sinal (S)

- Na base 2 normalizada, a algarismo à esquerda da vírgula obrigatoriamente terá **exatamente o bit 1**
  - Expoente é manipulado de acordo

Ponto Flutuante: Notação Científica



- Só um bit é necessário para o sinal.
- O expoente possui $n_{expoente}$ bits.
- A mantissa possui $n_{mantissa}$ bits.

Ponto Flutuante: Exemplo 1

- Representando $5{,}5_{(10)}$ com
  - 1 bit de sinal
  - 4 bits de mantissa
  - 3 bits de expoente.
- temos:
  - Convertendo para a base 2,

$$5{,}5_{(10)} = 101{,}1_{(2)}$$

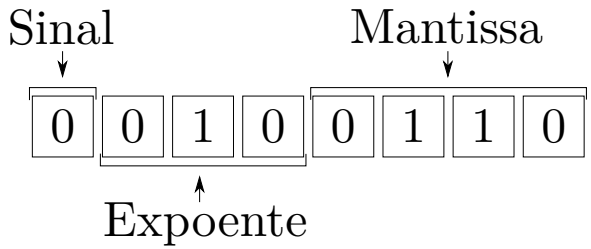
  - Colocando em notação científica normalizada:

$$1{,}011_{(2)} \times 2^2$$

  - Convertendo o expoente para a base 2 com 3 bits:

$$2_{(10)} = 10_{(2)} = 010_{(2)}$$

  - Os conjuntos de bits finais:



Ponto Flutuante: Exemplo 2

- Com 1 bit de sinal, 4 bits de mantissa e 3 bits de expoente, represente $-12{,}875_{(10)}$
  - Convertendo para a base 2,

$$12{,}875_{(10)} = 1100{,}111_{(2)}$$

  - Colocando em notação científica normalizada:

$$1{,}100111_{(2)} \times 2^3$$

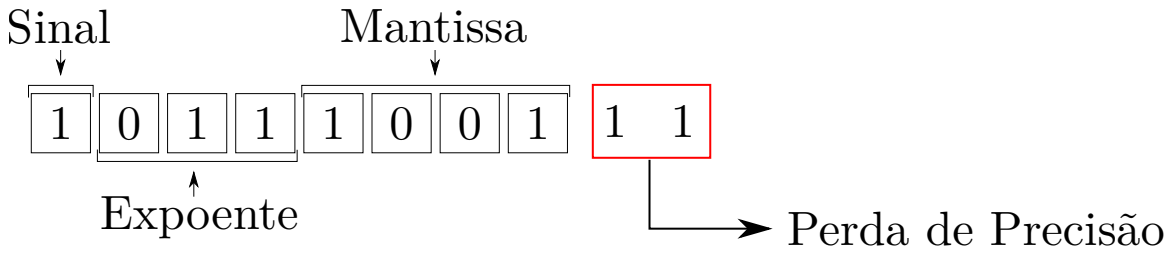
  - Convertendo o expoente para a base 2, com 3 bits:

$$3_{(10)} = 011_{(2)}$$

  .

Ponto Flutuante: Exemplo 2

- Os conjuntos de bits finais:



- Repare que houve uma perda de precisão: precisávamos de mais 2 bits para a mantissa
  - Neste exemplo, optamos por **truncar** o número
- Note também que este exemplo ilustra um número negativo

Ponto Flutuante: Exemplo 3

- Usando 1 bit de sinal, 4 bits de mantissa e 3 bits de expoente, represente $400{,}5_{(10)}$
  - Convertendo para a base 2,

$$400{,}5_{(10)} = 110010000{,}1_{(2)}$$

  - Colocando em notação científica normalizada:

$$1{,}100100001_{(2)} \times 2^{8}$$

  - Convertendo o expoente para a base 2:

$$8_{(10)} = 1000_{(2)}$$

    - ★ **Não é possível representar expoente com apenas 3 bits!**
    - ★ Número é muito grande para esta representação.

Ponto Flutuante: Resumo da Conversão

Da base 10 para base 2:

- 1 bit de sinal S: se positivo, $S = 0$ e se negativo, $S = 1$
- converter o número na base 10 para a base 2
- colocar na notação científica normalizada deslocando a vírgula: $1,M \times 2^E$
- converter o expoente E para a base 2
- juntar os bits na ordem: sinal S, expoente E, mantissa M.
  - Considerando o número de bits do expoente e da mantissa

Da base 2 para base 10:

- Separar os grupos de bits: sinal (S), expoente (E) e mantissa (M).
- Converter E para a base 2
- Colocar na forma: $(-1)^S \times 1,M \times 2^E$.
- Converter a parte $1,M$ para a base 10.
  - Andar com a vírgula da mantissa, se for necessário

Ponto Flutuante: Expoentes Negativos

- Nos exemplos mostrados, o expoente sempre foi positivo
- Mas e números menores que 1 ?
  - $0{,}5 = 5 \times 10^{-}1$
  - $0{,}00614 = 6{,}14 \times 10^{-3}$
- Logo, o expoente precisa ser codificado de forma a permitir números negativos também

Como codificar números negativos

- Sinal e magnitude
- Representação em Excesso
- Complemento a Um
- Complemento a Dois

Ponto Flutuante: Número de Bits por Componente

Quantos bits para cada grupo ?

- O sinal só precisa de um bit
- Mas há um compromisso entre o tamanho da mantissa e do expoente
  - Mantissa maior $\rightarrow$ mais precisão
  - Expoente maior $\rightarrow$ mais abrangência

Ponto Flutuante: Padrão IEEE 754

Para garantir interoperabilidade entre computadores, em 1985, o IEEE estabeleceu formatos padronizados para números em Ponto Flutuante

- Precisão simples: 32 bits no total
- Precisão dupla: 64 bits no total
- Precisão quádrupla: 128 bits no total

IEEE - Institute of Electrical and Electronics Engineers

- A maior associação técnica no mundo
- definindo vários conceitos
- especificando veículos de publicação

Ponto Flutuante: Padrão IEEE 754

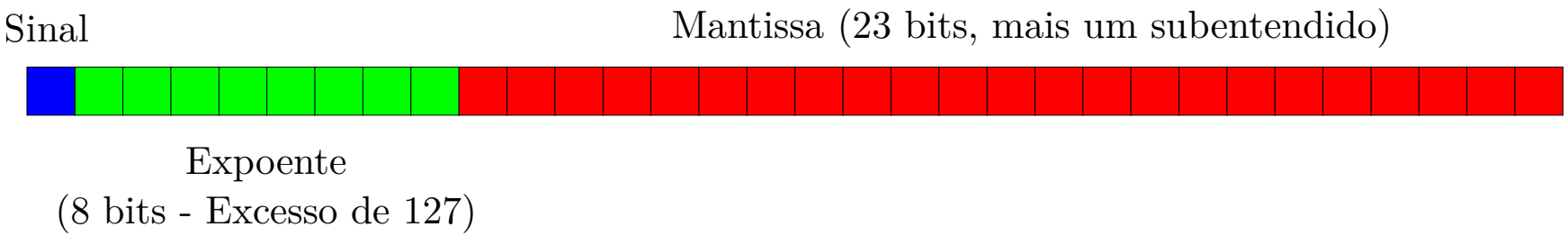
Utilizam Representação em Excesso de k para o expoente

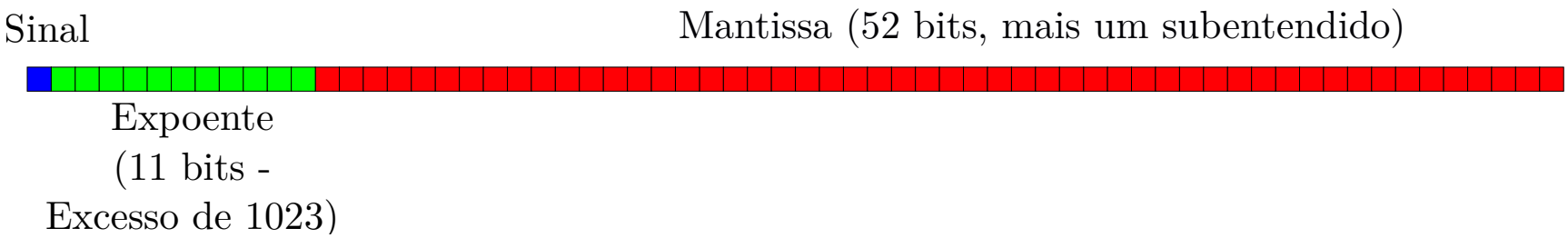
- Na precisão simples, $k = 127$ com 8 bits para o expoente.
  - 23 bits explícitos para a mantissa.
- Na precisão duplas, $k = 1023$ com 11 bits para o expoente.
  - 52 bits explícitos para a mantissa.

Valores de excesso escolhidos para distribuir de forma homogênea os expoentes positivos e negativos

Ponto Flutuante: Padrão IEEE 754 (II)

- Outra decisão importante do IEEE 754 foi em relação à ordem dos componentes.
  - Posicionados, da esquerda para a direita, como sinal, expoente e mantissa.
  - Permitem comparação de números como inteiros.





Padrão IEEE 754: Exemplo 1

- Representar valor $17{,}125_{(10)}$ em precisão simples.
  - Convertendo para a base 2, $17{,}125_{(10)} = 10001{,}001_{(2)}$.
  - Colocando em notação científica normalizada: $1{,}0001001_{(2)} \times 2^4$.
  - Em Excesso de 127, expoente passa a ser $4 + 127 = 131_{(10)}$.
  - Convertendo o expoente para a base 2: $131_{(10)} = 10000011_{(2)}$.
  - Representação final:

$$\boxed{01000001100010010000000000000000}$$

  - 1 bit de sinal, 8 bits de expoente e 23 bits de mantissa

Padrão IEEE 754: Exemplo 2

- Qual o valor correspondente em base 10, sendo a seguinte sequência em ponto flutuante de precisão simples no padrão IEEE 754?

$$\boxed{10101101101101000000000000000000}$$

- Então: 1 bit de sinal, 8 de expoente e 23 de mantissa:
  - Primeiro bit é 1: número negativo
  - Expoente é 01011011.
    - ★ Na base 10, valor corresponde a 91
    - ★ Subtraindo o excesso (que é 128), expoente é $91 - 127 = -36$
  - Mantissa é $1{,}01101_{(2)} = 1 + 0{,}25 + +0{,}125 + 0{,}03125 = 1{,}40625_{(10)}$
  - Valor final (base 10): $-1{,}40625 \times 2^{-36}$

IEEE 754: Casos Especiais

- Exemplo de exceção: o zero
  - Como escrevê-lo em Ponto Flutuante?
  - Não pode ser colocado em notação científica normalizada
    - ★ Qual seria a mantissa?
- Para resolver este problema, padrão propõe uma representação especial para o 0:
  - Sinal 0.
  - Expoente 0.
  - Mantissa 0.

IEEE 754: Casos Especiais (II)

- **Infinito**: representa os valores $+$ *infinito* e $-$ *infinito*.
  - Campo sinal: 0 para $+\infty$ e 1 para $-\infty$.
  - Campo de expoente: todos os bits em 1.
  - Campo de mantissa: todos os bits em 0.
- **Não é um número** (NaN – *Not a Number*):
  - Campo sinal: 0 ou 1.
  - Campo de expoente: todos os bits em 1.
  - Campo de mantissa: qualquer sequência de bits diferente de 0000...0000.
- **Valor não-normalizados** (*denormalized*):
  - Representa valores na faixa de: $0,M \times 2^{-126}$ a $-0,M \times 2^{-126}$.
    - ★ Que não são representáveis pelo caso comum.
  - Campo sinal: 0 ou 1.
  - Campo de expoente: todos os bits em 0.
  - Campo de mantissa: qualquer sequência de bits diferente de 0000...0000.

IEEE 754: Casos Especiais (Resumo)

- Para precisão simples:

Sinal	Expoente	Mantissa	Interpretação
0	0000...0000	0000...0000	$+0$
0	0000...0000	0000...0001 a 1111...1111	$0,M \times 2^{-126}$
0	0000...0001 a 1111...1110	xxxx...xxxx	$1,M \times 2^{(e-127)}$
0	1111...1111	0000...0000	$+\infty$
0	1111...1111	0000...0001 a 1111...1111	NaN
1	0000...0000	0000...0000	-0
1	0000...0000	0000...0001 a 1111...1111	$-0,M \times 2^{-126}$
1	0000...0001 a 1111...1110	xxxx...xxxx	$-1,M \times 2^{(e-127)}$
1	1111...1111	0000...0000	$-\infty$
1	1111...1111	0000...0001 a 1111...1111	NaN

IEEE 754: Limites

Qual é o maior número representável no IEEE 754 (precisão simples)?

- Sinal 0 (positivo)
- Expoente:

$$11111111 \rightarrow (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1) - 128 \rightarrow +127_{(10)}$$

- Mantissa 11111111111111111111111
- Valor final: $1{,}1111\ldots1111_{(2)} \times 2^{127} \approx 3{,}4 \times 10^{38}$

E o menor número (mais negativo)?

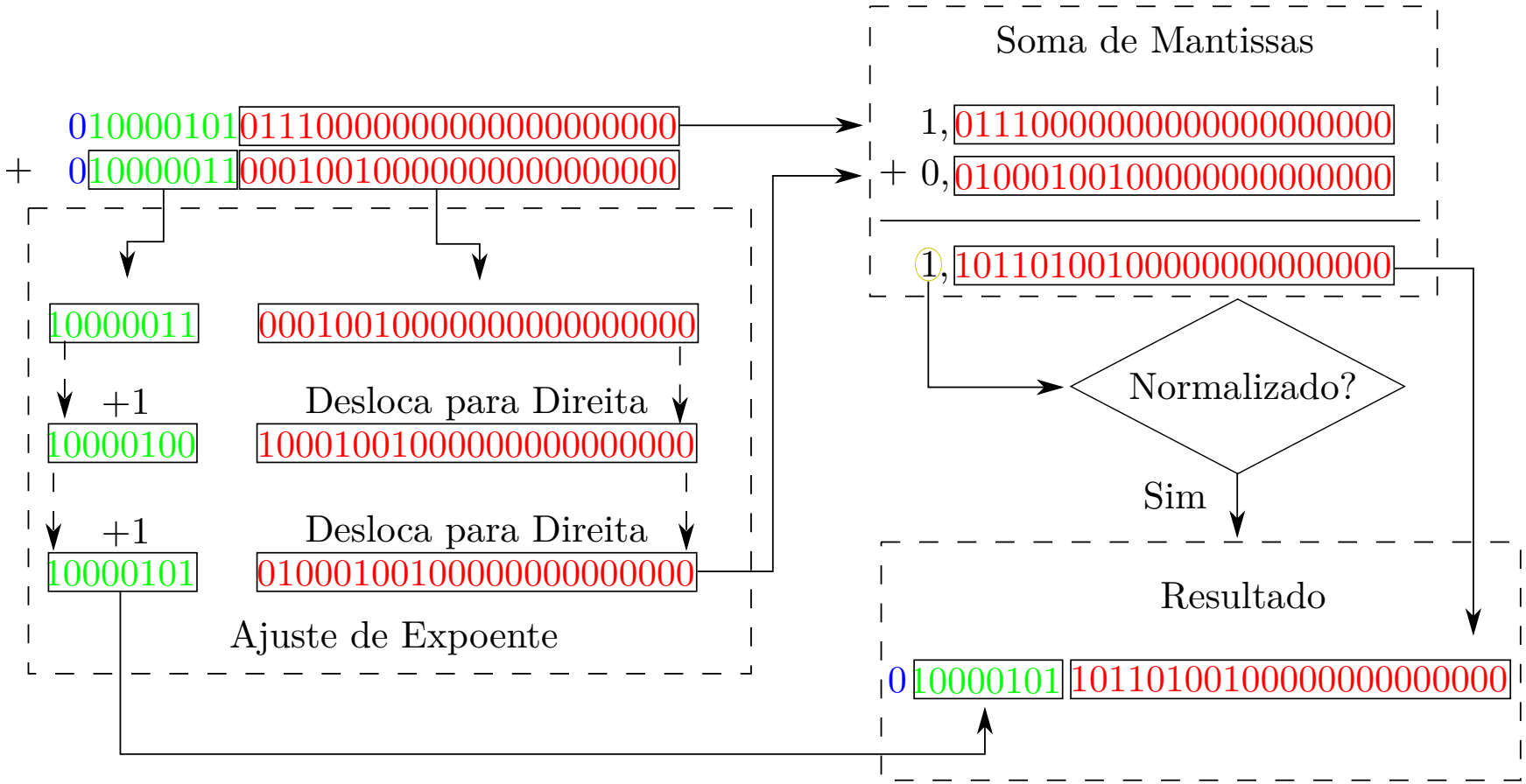
- Basta inverter o sinal do maior
- Valor final: $\approx -3{,}4 \times 10^{38}$

Operações em Ponto Flutuante: Soma

Por exemplo, como somamos dois números em Ponto Flutuante?

- Lembre-se que o número está em notação científica
- Se os expoentes são iguais, podemos somar as mantissas
- Caso contrário, expoente menor é igualado ao maior
  - Mantissa é deslocada para a direita de acordo.
- Ao final, ainda temos que normalizar o resultado

Operações em Ponto Flutuante: Soma (Exemplo)



Operações em Ponto Flutuante: Soma (Detalhes)

- Por que ajustar o menor expoente e não o maior?
- Alterar um expoente resulta em perda de precisão.
  - Bits extremos são perdidos.
- Aumentar o menor expoente resulta em deslocar mantissa para a direita.
  - Bits perdidos são os menos significativos.
- Além disso, aumenta a chance do resultado já estar normalizado.

Operações em Ponto Flutuante: Multiplicação e Divisão

- Na multiplicação, mantissas são multiplicadas e expoentes são somados.
  - Como na soma, precisamos garantir que o resultado final é normalizado.
- Analogamente, na divisão, executamos uma divisão entre as mantissas e uma subtração entre os expoentes.

Operações em Ponto Flutuante: Estouro

- Assim como para as representações de números inteiros, representações em Ponto Flutuante também podem sofrer estouro.
  - Tentativa de representar valores fora da faixa suportada.
  - Em geral, resultantes de operações matemáticas.
- No caso do Ponto Flutuante, o estouro corresponde a um estouro na representação do **expoente**.
- Há dois tipos:
  - *Overflow*: **magnitude** do número é **grande demais**.
    - ★ Exemplo (precisão simples): resultado de $1{,}7 \times 10^{23} \times -4{,}2 \times 10^{30}$.
  - *Underflow*: **magnitude** do número é **pequena demais**.
    - ★ Exemplo (precisão simples): resultado de $1{,}7 \times 10^{-23} \times 4{,}2 \times 10^{-30}$.

Exercícios

- Calcule a representação dos seguintes números em Ponto Flutuante:
  - $10{,}5_{(10)}$, com 4 bits de expoente (Complemento a Dois) e 6 de mantissa.
  - $5{,}125_{(10)}$, com 3 bits de expoente (Excesso de 3) e 8 de mantissa.
- Determine o número representado pelas seguintes sequências de bits em Ponto Flutuante:
  - 10100110000, com 4 bits de expoente (Complemento a Dois) e 6 de mantissa.
  - 010100100000, com 3 bits de expoente (Excesso de 3) e 8 de mantissa.
- Determine o **maior número não normalizado** que pode ser escrito em precisão simples no IEEE 754.