Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Escola Politécnica

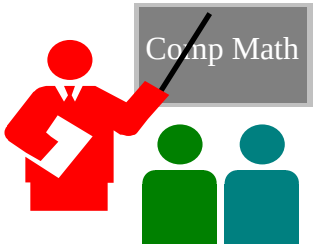
Aritmética Computacional
Uma Breve Introdução

Autores: Fernando Gehm Moraes & Ney Laert Vilar Calazans

Porto Alegre, junho de 1998

Última revisão: 28/outubro/2022

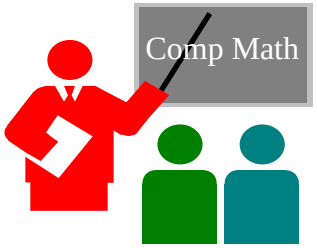
Sumário



- Introdução
- Aritmética Natural/Inteira
  - Operações sobre números sem sinal (naturais - N)
    - » Soma, subtração, multiplicação e divisão
  - Operações sobre números com sinal (inteiros - Z)
    - » Soma, subtração, multiplicação e divisão
- Aritmética não-Inteira (racionais - Q)
  - Representação de racionais - Q
  - Operações aritméticas usando IEEE-754

Sumário

- Introdução



Introdução a Aritmética Computacional

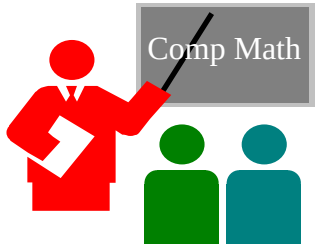
- Uma parte especializada do projeto de computadores
- Contudo, uma parte muito, muito importante
  - gráficos, comunicações, transações bancárias, matemática computacional, cálculo de estrutura, solução de equações, entre tantas aplicações
- Exemplo da importância: Intel perdeu US$ 300milhões devido ao “bug” do Pentium (otimizou errado um PLA usado em ÷)
- Aqui, revisão curta da aritmética em hardware para N e Z, e estudo do padrão para números racionais (IEEE-754) e operações sobre esta representação

Introdução a Aritmética Computacional

- Bibliografia:
  - John L. Hennessy and David A. Patterson. “Computer Architecture - a quantitative approach”, Morgan Kaufmann Pubs., Secon Edition, 1996. Appendix A
  - Texas Instruments. “TMS320C4x User’s Guide”, 1996. Capítulo 5. Disponível na Internet (formato PDF). (Usada para princípios do padrão IEEE-754 – 1985 e funcionamento de operações aritméticas)
    - https://www.ti.com/lit/ug/spru063c/spru063c.pdf?ts=1633696568594

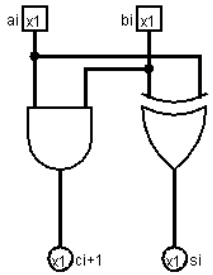
Sumário

- √ Introdução
- ➡ Aritmética Natural e Inteira
  - Operações sobre números sem sinal (naturais - N)
    - » soma, subtração, multiplicação e divisão

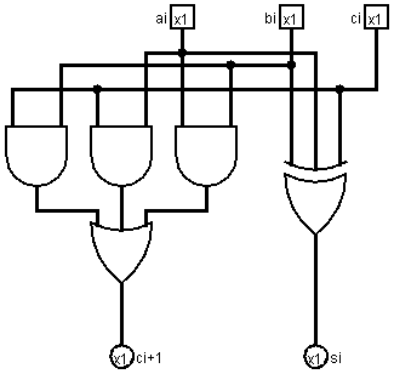


Operações sobre Naturais - Soma

- Hardware mais simples - baseado em *ripple-carry* (propagação em onda do vai-um) e componentes simples (meio somador e somador completo)
- Equações dos componentes ($\oplus$ = XOR, $\wedge$ = AND, $\vee$ = OR)
  - meio-somador (*half-adder* ou HA)
    - $s_i = a_i \oplus b_i$
    - $c_{i+1} = a_i \wedge b_i$

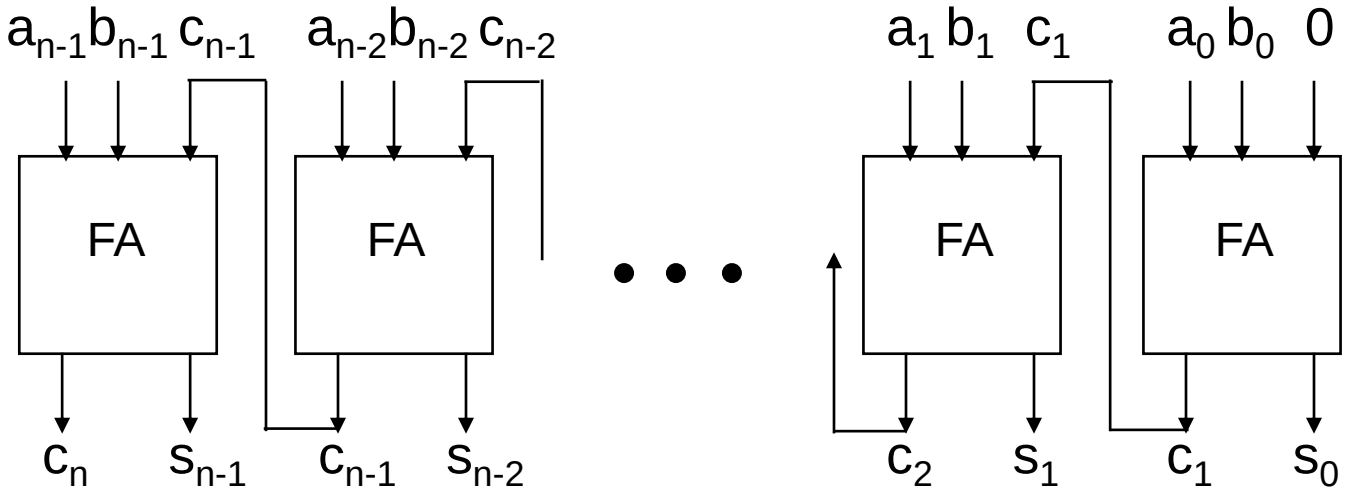


  - somador completo (*full-adder* ou FA)
    - $s_i = a_i \oplus b_i \oplus c_i$
    - $c_{i+1} = a_i \wedge b_i \vee a_i \wedge c_i \vee b_i \wedge c_i$



Operações sobre Naturais - Soma

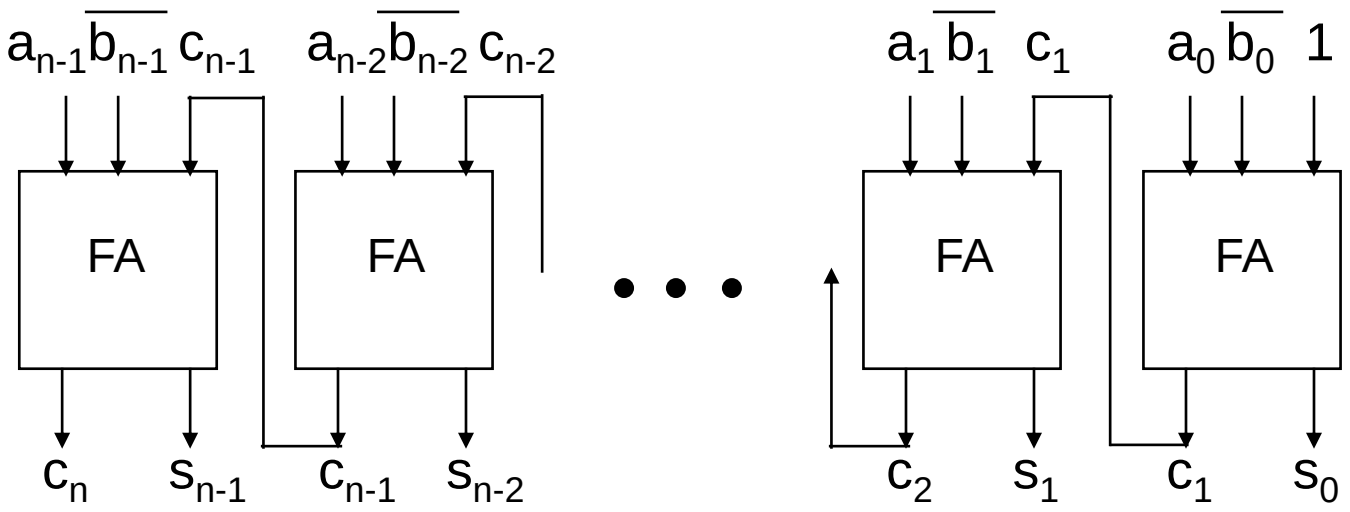
- Estrutura do somador completo "*ripple-carry*"



- Problema: atraso de geração do vai-um = *O(n)*, onde *n* é o número de bits do somador
- Hardware adicional → atraso = *O(log n)*

Operações sobre Naturais - Subtração

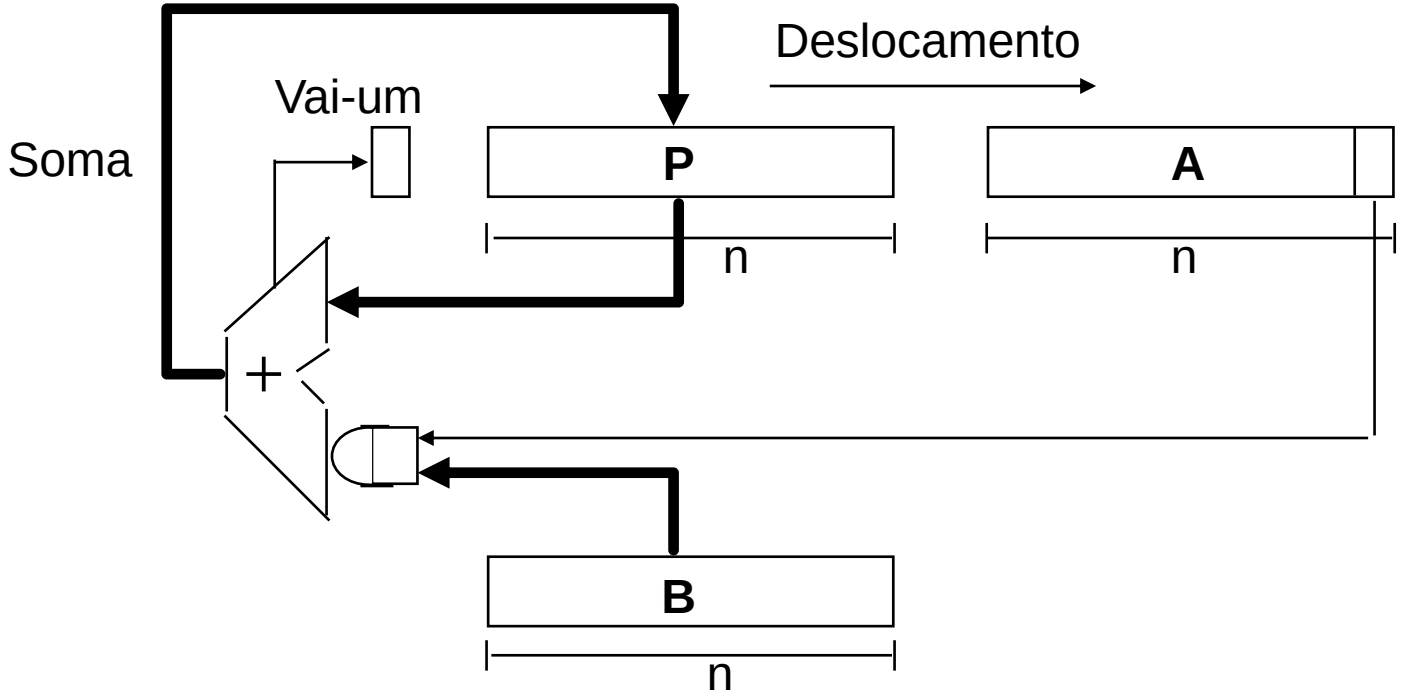
- Estrutura do subtrator completo a-b *ripple-carry*



- Mesmo problema de atraso, solução análoga
- Problema adicional, operação não definida (em N) se b>a

Operações sobre Naturais - Multiplicação

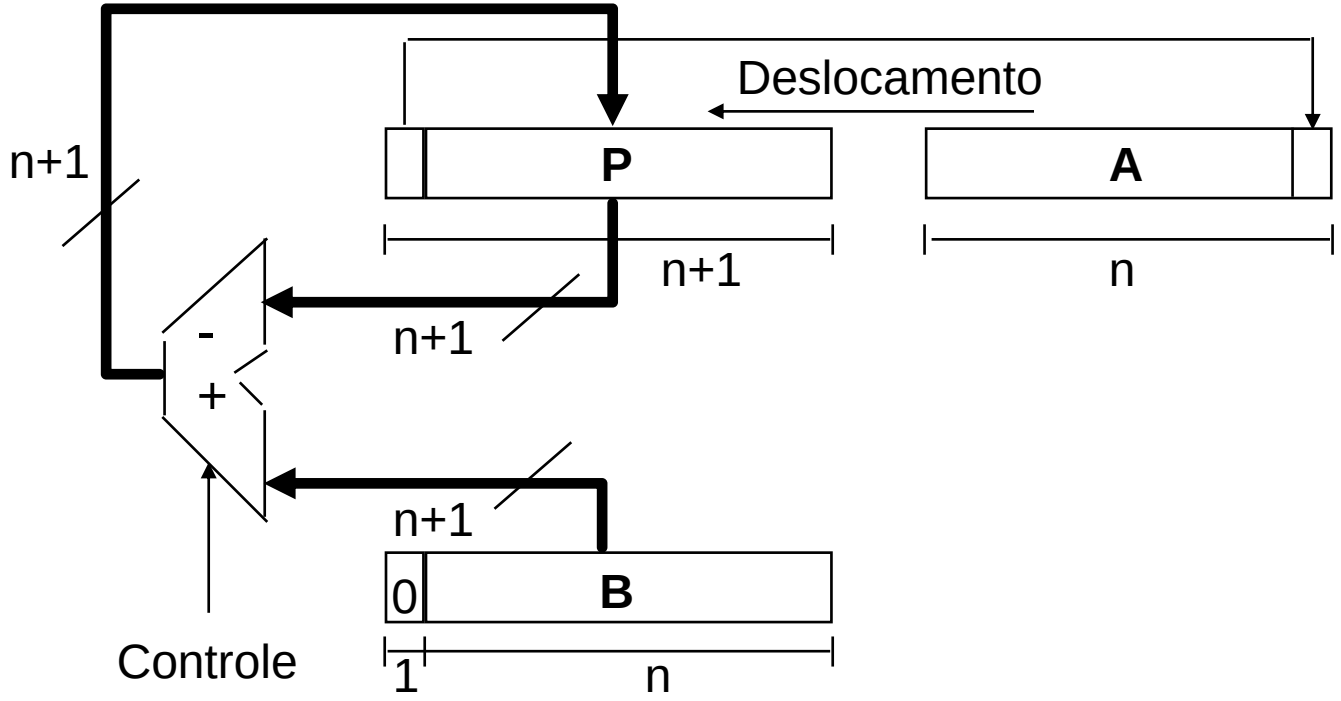
- Solução em N para b*a: somas sucessivas, n passos



- Inicialmente, P=0, A=a, B=b. Cada passo, duas partes
  - Soma carregada em P
  - Vai-um & P & A deslocado um bit para a direita

Operações sobre Naturais - Divisão

- Solução para a/b: subtrações sucessivas, n passos



- Quatro passos
  - 1) Desloca P&A p/ a esquerda de 1 bit
  - 2) P ← P-B
  - 3) Se (passo 2<0), A_0=0 else A_0 =1
  - 4) Se (passo 2<0), restaura P fazendo P ← P+B

Divisão em Naturais A/B - Exemplo

A = 11011 (27)

B = 00101 (5)

A cada volta, Tabela mostra P e A após passo executado

Passo 1) Na primeira linha

Passo 4) Na segunda linha

Algoritmo

1) Desloca P&A p/ esq 1 bit
2) P ← P-B
3) Se (P<0), A_0=0 else A_0 =1
4) Se (P<0), restaura P fazendo P ← P+B

	P (conterá o resto)						A (conterá a divisão)				
volta	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2											
3											
4											
5											

Divisão em Naturais A/B - Exemplo

A = 11011 (27)

B = 00101 (5)

A cada volta, Tabela mostra P e A após passo executado

Passo 1) Na primeira linha

Passo 4) Na segunda linha

Algoritmo

1) Desloca P&A p/ esq 1 bit
2) P ← P-B
3) Se (P<0), A_0=0 else A_0 =1
4) Se (P<0), restaura P fazendo P ← P+B

	P (conterá o resto)						A (conterá a divisão)				
volta	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3											
4											
5											

Divisão em Naturais A/B - Exemplo

A = 11011 (27)

B = 00101 (5)

A cada volta, Tabela mostra P e A após passo executado

Passo 1) Na primeira linha

Passo 4) Na segunda linha

Algoritmo

1) Desloca P&A p/ esq 1 bit
2) P ← P-B
3) Se (P<0), A_0=0 else A_0 =1
4) Se (P<0), restaura P fazendo P ← P+B

	P (conterá o resto)						A (conterá a divisão)				
volta	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3	0	0	0	1	1	0	1	1	0	0	0
	0	0	0	0	0	1	1	1	0	0	1
4											
5											

00110 - 00101 = 001

Divisão em Naturais A/B - Exemplo

A = 11011 (27)

B = 00101 (5)

A cada volta, Tabela mostra P e A após passo executado

Passo 1) Na primeira linha

Passo 4) Na segunda linha

Algoritmo

1) Desloca P&A p/ esq 1 bit
2) P ← P-B
3) Se (P<0), A_0=0 else A_0 =1
4) Se (P<0), restaura P fazendo P ← P+B

	P (conterá o resto)						A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3	0	0	0	1	1	0	1	1	0	0	0
	0	0	0	0	0	1	1	1	0	0	1
4	0	0	0	0	1	1	1	0	0	1	0
	0	0	0	0	1	1	1	0	0	1	0
5											

Divisão em Naturais A/B - Exemplo

A = 11011 (27)

B = 00101 (5)

A cada volta, Tabela mostra P e A após passo executado

Passo 1) Na primeira linha

Passo 4) Na segunda linha

Algoritmo

1) Desloca P&A p/ esq 1 bit
2) P ← P-B
3) Se (P<0), A_0=0 else A_0 =1
4) Se (P<0), restaura P fazendo P ← P+B

	P (conterá o resto)						A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3	0	0	0	1	1	0	1	1	0	0	0
	0	0	0	0	0	1	1	1	0	0	1
4	0	0	0	0	1	1	1	0	0	1	0
	0	0	0	0	1	1	1	0	0	1	0
5	0	0	0	1	1	1	0	0	1	0	0
	0	0	0	0	1	0	0	0	1	0	1

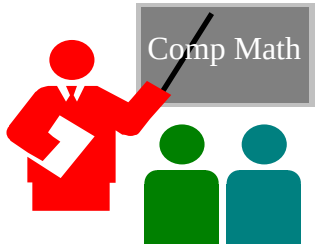
Resto = 2 **Quociente = 5**

Operações sobre Naturais - Divisão

- Algoritmo: versão binária - procedimento lápis e papel
- Existe versão sem restauração de P (ver H & P)
- n passos, somador 1 bit maior que na multiplicação
- Deve-se testar se divisor =0!
- Restauração desnecessária se teste feito na saída do somador/subtrator, bem como somador

Sumário

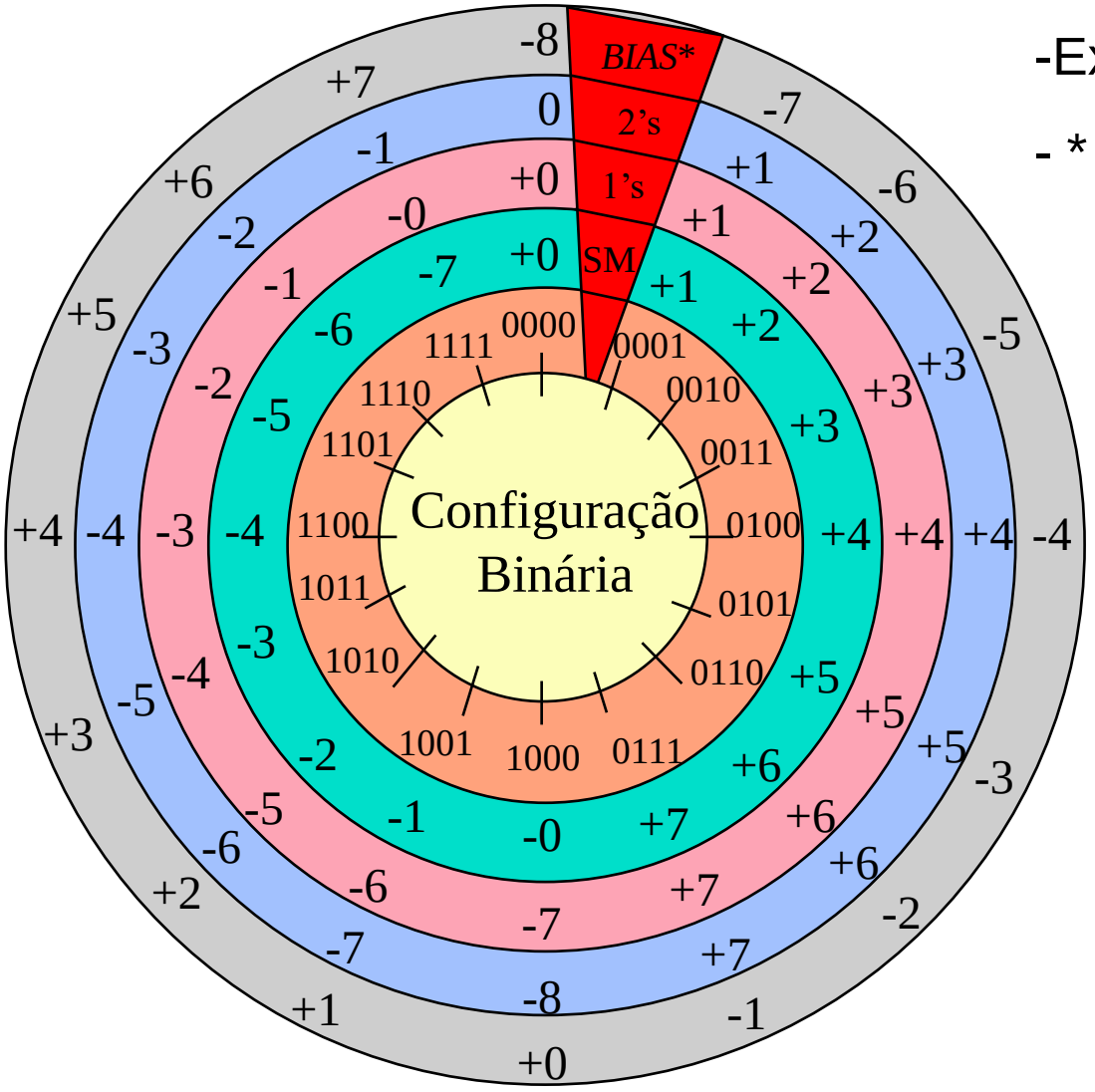
- √ Introdução
- ➡ Aritmética Natural/Inteira
  - √ Operações sobre números sem sinal (naturais - N)
    - √ soma, subtração, multiplicação e divisão
  - – Operações sobre números com sinal (inteiros - Z)
    - » Soma, subtração, multiplicação e divisão



Representações de Inteiros

- Quatro formas distintas de representar elementos de Z em binário
  - sinal-magnitude (SM) - bit mais significativo é sinal, restante é o valor absoluto equivalente a binário sem sinal
  - complemento de 1 (1's) - positivos, idem a SM, negativos são positivos com valor invertido
  - complemento de 2 (2's) - positivos, idem a SM, negativos obtidos adicionando 1 a 1's
  - polarização (bias) - representação é a do positivo binário obtido a partir da soma de um valor k

Representações de Inteiros - *4 formas*



-Exemplo para 4 bits

- * polarização (=*Bias*) = $2^{(n-1)}$ =8

- *Bias* tem distribuição uniforme com relação a binários puros
- 2's facilita soma/subtração
- 1's facilita complementação
- SM é fácil de entender e separa sinal de valor

Transbordo em Complemento de 2

- Casos de transbordo em complemento de 2 (2's)
  - Para 5 bits, faixa representável é -16 a +15

Decimal	Binário
Vai-um:	00111
+ 5	00101
+ 7	00111
+12	01100

Decimal	Binário
Vai-um:	11011
- 5	11011
- 7	11001
-12	10100

Decimal	Binário
Vai-um:	00001
+ 5	00101
- 7	11001
- 2	11110

Decimal	Binário
Vai-um:	11111
- 5	11011
+ 7	00111
+ 2	00010

Transbordo em Complemento de 2

- Casos de transbordo em complemento de 2 (2's)
  - Para 5 bits, faixa representável é -16 a +15

Decimal	Binário
Vai-uns:	00111
+ 5	00101
+ 7	00111
+12	01100

Decimal	Binário
Vai-uns:	11011
- 5	11011
- 7	11001
-12	10100

Decimal	Binário
Vai-uns:	00001
+ 5	00101
- 7	11001
- 2	11110

Decimal	Binário
Vai-uns:	11111
- 5	11011
+ 7	00111
+ 2	00010

Transbordos (*overflow*) positivo e negativo

Decimal	Binário
Vai-uns:	01000
+ 8	01000
+ 9	01001
+17	~~10001~~

Decimal	Binário
Vai-uns:	10000
- 8	11000
- 9	10111
-17	~~01111~~

- XOR dos 2 últimos vai-uns identifica transbordo

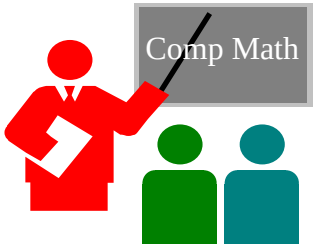
Negativo!

Positivo!

Multiplicação e Divisão em Z?

- Não veremos detalhadamente aqui → existem muitos esquemas para
  - Multiplicação de Inteiros
    - Existe um algoritmo consagrado como eficiente para implementações de Hw
      - Algoritmo de Booth (https://pt.wikipedia.org/wiki/Algoritmo_de_multiplica%C3%A7%C3%A3o_de_Booth)
      - Muitos outros algoritmos podem ser encontrados na literatura especializada
  - Divisão de Inteiros
    - A literatura aqui é bem escassa, mas há pelo menos uma boa descrição de como implementar um divisor em Hw no link abaixo (dissertação de mestrado de um aluno da *Ohio State University* - OSU, de 2019)
      - https://core.ac.uk/download/pdf/289095991.pdf
      - 41 páginas → Representações e Funcionalidade
      - Divisor de inteiros em complemento de 2
      - Implementação em Verilog

Sumário



- √ Introdução
- √ Aritmética Natural/Inteira
  - √ Operações sobre números sem sinal (naturais - N)
    - √ soma, subtração, multiplicação e divisão
  - √ Operações sobre números com sinal (inteiros - Z)
    - √ soma, subtração, multiplicação e divisão
- ➡ Aritmética Não-Inteira (racionais - Q)
  - Representação de racionais

Aritmética não Inteira – Racionais Q

- Muitas aplicações → números não-inteiros
  - matemática computacional, engenharia etc
- Racionais (Q)
  - representados como fração *a/b* → *a* e *b* em Z, b diferente de 0
- Irracionais (I)
  - têm mantissa infinita, sem repetição (e.g. *e*=2.7218... e π=3.14...)
- Reais (R)
  - R= Q $\cup$ I → I e R não representáveis em computadores...
  - Porquê?
- Aproximação de reais em computadores → subconjunto de Q

Aritmética não Inteira - Representações

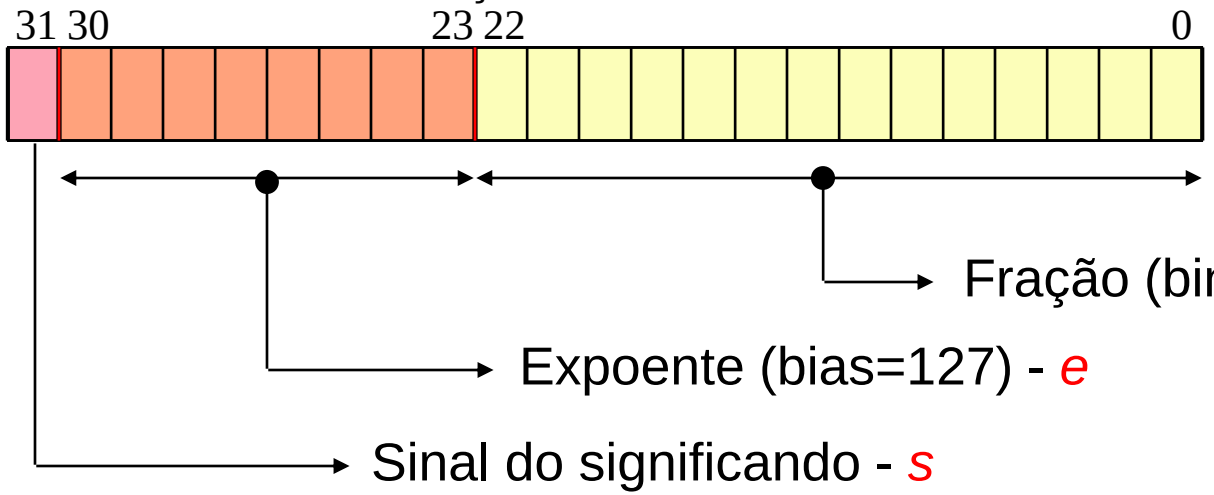
- Primeiros computadores → ponto fixo (ou vírgula fixa)
- Hoje → ponto flutuante muito usado
  - Representações com um *significando* *s* (a *mantissa* do número), um *expoente* *e* de uma *base* *b*
  - Número representado a partir da forma $s * b^e$
- Antes → muitos formatos proprietários (IBM, DEC etc.)
- Hoje → padrão universal
  - Definido pela IEEE (IEEE-754-1985), revisado em 2008 e 2019
  - Norma original (de 1985) → 4 formatos
    - 2 fixos → precisão simples de 32 bits (SP) e precisão dupla (DP) de 64 bits
    - 2 variáveis → precisão simples estendida (SE) e dupla estendida (DE)
  - Norma de 2019 → 5 formatos (3 binários, de 32/64/128 bits; 2 decimais, de 64/128)

Números Racionais - IEEE-754-1985

- Diferenças IEEE-754 *versus* formatos pré-padronização
  - inclui valores especiais
    - » NaN - *Not a number* (e.g. raiz quadrada de número negativo)
    - » $-\infty$ e $+\infty$ Mais ou menos infinito (e.g. -1/0 e +1/0)
  - números desnormalizados para resultados com valor menor que $+1.0 \times 2^{e\_mínimo}$
    - » arredonda para mais próximo (*default*) mas tem mais três modos de arredondamento
    - » ao arredondar um resultado no meio da faixa, pega valor par mais próximo
  - recursos sofisticados → exceções

Números Racionais - IEEE-754-1985

- Formato precisão simples (SP) ocupa exatamente 32 bits
  - Usado em programação e.g. ao declarar número como ***`float`***
  - 1 bit para sinal do significando - *s*
  - significando *f* com 24 bits
    - » Precedido de 1., exceto quando denormalizado, onde é 0. (1° bit implícito)
  - expoente *e* de 8 bits
    - » Polarização = 127



Forma Geral:
$(-1)^s \times 1.f \times 2^{e-127}$

Obs: *sf* é representação em sinal-magnitude (SM)!

IEEE-754-1985 - Exemplos

1) Um número positivo, maior que zero

Forma Geral:
$(-1)^{s} \times 1.f \times 2^{e-127}$

```
31 30        23 22                                  0
0 10000001 01000000000000000000000 – em hexa 0x40A00000, certo?
```

$((-1)^{0} \times 1.01 \times 2^{129-127})_2 = (+1.01 \times 2^{2})_2 = (+5.0)_{10}$

2) Um número negativo, menor que zero

```
31 30        23 22                                  0
1 01111110 01110000000000000000000
```

$((-1)^{1} \times 1.0111 \times 2^{126-127})_2 = (-1.0111 * 2^{-1})_2 = (-0.10111)_2 = (-0.71875)_{10}$

Valores Especiais e Desnormalização

- 5 formas de interpretar vetores de 32 bits IEEE-754 (SP)
  - 1 caso geral → Linha 3) na Tabela abaixo
  - 4 casos especiais → Linhas 1), 2), 4) e 5)

1) e=255, f≠0 (**NaN**)	v = NaN (**not a number**)
2) e=255, f=0 (**+/- infinito**)	$v = (-1)^s = \infty$
3) 0<e<255 (**forma geral**)	$v = (-1)^s \times (1.f) \times 2^{e-127}$
4) e=0, f≠0 (**desnormalizados**)	$v = (-1)^s \times (0.f)\ 2^{-126}$
5) e=0, f=0 (**+0 ou -0**)	$v = (-1)^s \times (0) =$ (zero)

Valores Especiais e Denormalização

- Principal novidade do padrão IEEE-754 → existência de casos especiais, que permitem representar/tratar exceções
  - Valores infinitos → $+\infty$ e $-\infty$
  - Aproximação de 0 pela esquerda (-0) ou pela direita (+0)
  - Divisão por zero → $+\infty$ e $-\infty$
  - Operações onde racionais não são fechados, como raiz quadrada → NaN
- Desnormalização → permite situações de "*underflow*" gradual e.g.

`0 00000001 00000000000000000000000`

- ($=2^{-126}$) dividido por 2^5
- representável se o primeiro dígito da representação for 0 ao invés de 1

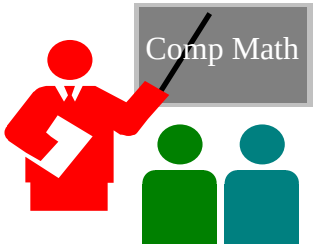
`0 00000000 00010000000000000000000`

Parâmetros e outros Formatos

- Diferentes formatos e valores → IEEE-754-1985

formato → parâmetros ↓	precisão simples (SP)	precisão simples estendida (SE)	precisão dupla (DP)	precisão dupla estendida (DE)
bits de precisão (p)	24	≥32	53	≥64
Emax	127	≥1023	1023	≥16383
Emín	-126	≤-1022	-1022	≤-16382
Polarização (bias)	127	depende	1023	depende
Total de bits	32 exatamente	variável, ≥43, <64	64 exatamente	variável, ≥79

Sumário



- √ Introdução
- √ Aritmética Natural/Inteira
  - √ Operações sobre números sem sinal (naturais - N)
    - √ soma, subtração, multiplicação e divisão
  - √ Operações sobre números com sinal (inteiros - Z)
    - √ soma, subtração, multiplicação e divisão
- ➡ Aritmética não-Inteira (racionais - Q)
  - √ Representação de racionais - Q
  - – Operações aritméticas usando IEEE-754

Operações com o Padrão IEEE-754

- Operação mais fácil de se implementar em hardware → Multiplicação
  - Multiplicação → direta, multiplica significandos e soma expoentes
  - Exceções → se operando ou resultado recai em caso especial
  - Soma é mais complexa, pois
    - » Requer adaptação de significandos
    - » Números com diferentes expoentes → complicam a soma
- Aqui → Breve introdução a ambas, multiplicação e soma

Multiplicação no Padrão IEEE-754

- 3 Passos
  - Multiplicar significandos
    - » Não fração, desempacotar o número usando multiplicação inteira, sem sinal (SM)
  - Calcular expoente
    - » Lembrar → representação com polarização
  - Arredondamento → pode ser necessário
    - » Devido ao aumento da precisão após operação

Multiplicação no Padrão IEEE-754

- Exemplo muito simples de multiplicação (sem arredondamento)

1 10000010 00000000000000000000000 = $-1 \times 2^3 = -8$

0 10000011 00000000000000000000000 = $1 \times 2^4 = 16$

  - 1) Desempacotando -> 1.0 x 1.0 = 1.0
    - » logo, resultado tem a forma

1 ???????? 00000000000000000000000

  - 2) Expoente - fórmula para cálculo do expoente

$(\text{exp polarizado}(e_1+e_2))_{2's}$ = (exp polarizado(e_1) + (exp polarizado(e_2) + (-polarização)$_{2's}$, ou seja,

```
  10000010 = 130
  10000011 = 131
+10000001 =-127
  10000110 = 134=e    -> E=134-127=7!
```

Multiplicação no Padrão IEEE-754

- 3) Arredondamento - precisão é importante
  - » Em binário, meio da faixa (5 em decimal) é dígito 1!
  - » Negrito - dígitos significativos
    - Após, dígito arredondador, *r*
  - » Casos de arredondamento (em decimal, binário é análogo)

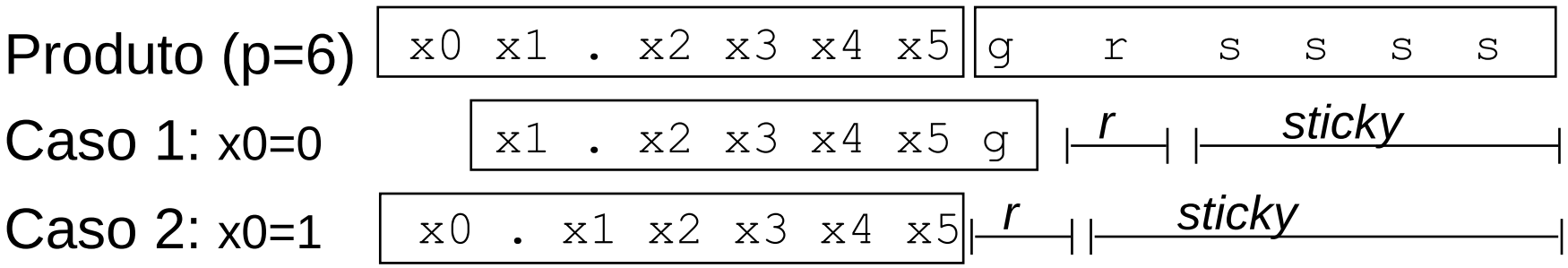
a) 1.23 r=9, 9>5, então arredonda p/8.34
x 6.78
8.3394

b) 2.83 r=5, e pelo menos um dígito após é
x 4.47 diferente de 0, arredonda p/ 1.27×10^1
12.6501

c) 1.28 r=6, 6>5, então arredonda p/ 1.00×10^1
x 7.81
0**9.99**68

Multiplicação no Padrão IEEE-754

- 3) Arredondamento (continuação)
  - Se *r* é menor que 5 - resultado pronto
  - Se *r* é maior que 5 - soma-se 1 ao número em negrito mais à direita
  - Se arredondador exatamente 5 (em binário, 1) - examinar bits seguintes, até achar um diferente de 0 ou chegar ao fim
    - Técnica - usa o "bit grudento" (*sticky bit*), durante a multiplicação, o OU lógico de todos os bits a partir do bit *r*
    - Caso 1 - desloca um bit p/ esquerda
    - Caso 2 - incrementa expoente



Multiplicação no Padrão IEEE-754

- 3) Arredondamento (continuação)
  - Após acertar expoente e resultado, pode-se finalmente arredondar
    - Se r=0, resultado correto
    - Se r=1 e s=1, soma P+1 para obter o produto dos significandos
    - Se r=1 e s=0, exatamente no meio da faixa - IEEE-754 possui quatro modos possíveis, cujo comportamento depende do sinal do resultado

Multiplicação e Desnormalização

- Controlar *underflow* é complicado, devido aos desnormalizados
  - por exemplo, $1x2^{-64}$ X $1x2^{-65}$ é $1x2^{-129}$, expoente não representável no formato normalizado, mas representável como $0.125x2^{-126}$
  - se o expoente não polarizado for menor que -126, produto deve ser deslocado bit a bit e expoente incrementado até -126. Se todo o significando se anular, aí sim, houve *underflow*
  - Quando um dos operandos é desnormalizado, geram-se 0s à esquerda
  - Desnormalizados causam montes de problemas na multiplicação
  - Multiplicadores de alto desempenho ignoram desnormalizados → geram exceções para o software cuidar
  - Desnormalizados não são frequentes → perda em tempo é estatisticamente pequena

Multiplicação, 0 e Precisão

- Se um dos multiplicandos é 0, acelera-se multiplicação testando
  - Antes - ambos operandos
  - Depois - neste caso, cuidado com 0 x ∞, resultado deve ser NaN
  - Sinal deve ser mantido, +0 é diferente de -0 – regra do padrão
- Ao multiplicar, pode-se precisar do dobro de bits em inteiros
  1. Aplicação define se é aceitável apenas a metade inferior do resultado ou
  2. Todo o resultado deve ser usado
- Em linguagens de alto nível
  - Multiplicação inteira sempre usa a opção 1. acima
  - Ponto flutuante é diferente → 1. e 2. são usadas
  - Utilidade grande na solução numérica de equações lineares

Exercício Resolvido

Supor que se tenha a seguinte representação para ponto flutuante: 1 bit para sinal, 4 para expoente e 7 para a parte fracionária (ou seja, uma norma IEEE-754*-like* para 12 bits, com as mesmas condições para NaN, 0, $\pm\infty$)

Pergunta-se:

1. Qual o valor da polarização e quais os expoentes mínimo e máximo?
2. Converta os seguintes números para base decimal

   `0 1010 1011000`

   `1 0111 0101000`
3. Multiplicar os dois números acima, mostrando o procedimento da multiplicação para ponto flutuante, com arredondamento
4. Qual o resultado, em decimal, arredondado, e qual o erro advindo do arredondamento?

Exercício Resolvido

Supor que se tenha a seguinte representação para ponto flutuante: 1 bit para sinal, 4 para expoente e 7 para a parte fracionária (ou seja, uma norma IEEE-754*-like* para 12 bits, com as mesmas condições para NaN, 0, $\pm\infty$)

Respostas:

1. Qual o valor da polarização e quais os expoentes mínimo e máximo?
   Valor da polarização: 7
   Expoente máximo +7, pois $(1110)_2$= 14, e 14-7=7
   Expoente mínimo -6, pois $(0001)_2$=1, e 1-7=-6

2. Converta para base decimal: **0 *1010* 1011000 e 1 *0111* 01010000**
   $+1.1011*2^{(10-7)} = +1.1011*2^3 = 1101.1 = 13.5$
   $-1.0101*2^{(7-7)} = -1.0101*2^0 = -1.0101 = -1.3125$

Exercício Resolvido

Supor que se tenha a seguinte representação para ponto flutuante: 1 bit para sinal, 4 para expoente e 7 para a parte fracionária (ou seja, uma norma IEEE-754-*like* para 12 bits, com as mesmas condições para NaN, 0, $\pm\infty$)

Respostas (continuação):

3. Multiplicar os dois números anteriores (**0 1010 1011000 e 1 0111 01010000**), mostrando o procedimento da multiplicação para ponto flutuante, com arredondamento

Cálculo do expoente: 10+7-7 = 10 (este é o expoente já polarizado!)
Produto das partes fracionárias: 1.1011 * 1.0101 = 10.00110111
10.00110111 deve ser normalizado para 1.0001101 11 * 2^1. Como o expoente da multiplacação estava em 10 ele vai para 11 (de novo, já polarizado)
Soma-se 1 ao bit menos significativo representável pois r=1 e o sticky bit s=1. No final a representação fica
1 1011 000111 = - 17.75
s e f

Exercício Resolvido

Supor que se tenha a seguinte representação para ponto flutuante: 1 bit para sinal, 4 para expoente e 7 para a parte fracionária (ou seja, uma norma IEEE-754*-like* para 12 bits, com as mesmas condições para NaN, 0, $\pm\infty$).

Respostas (continuação):

4. Qual o resultado, em decimal, arredondado, e qual o erro advindo do arredondamento?

Resposta: 17.75, ao invés de 17.71875 (este último é o resultado exato de multiplicar 13.5 * 1.3125). O erro é 17.75-17.71875 = 0,03125

Adição no Padrão IEEE-754

- Tipicamente, operação em ponto flutuante recebe dois números de mesma precisão (p bits) e retorna resultado com mesma precisão, *p bits*
- Algoritmo ideal (erro menor) calcula resultado exato e depois arredonda
- Multiplicação funciona assim
- Para soma, existem procedimentos mais efetivos
- Exemplo com números de 6 bits: 1.10011_2 e $1.10001_2 \times 2^{-5}$
- Usando um somador de 6 bits, tem-se

```
  1.10011
+ 0.00001
---------
  1.10100
```

Adição no Padrão IEEE-754

- No exemplo, bit descartado (r) é 1. Logo examina-se o resto
- Novamente, apenas se precisa saber se um destes bits é não-zero, e pode-se assim usar o “sticky bit”, como na multiplicação para saber se é melhor arredondar ou não
- Logo, para somar números de *p* bits, um somador de *p* bits chega, desde que se guarde o primeiro bit descartado (r) e o “sticky bit” correspondente
- No exemplo, o sticky é 1, e o resultado final fica 1.10101_2
- Subtração é similar, se se trabalha em complemento de dois

Adição no Padrão IEEE-754

- Outro exemplo com precisão de 6 bits

```
  1.11011
+0.0101001
```

```
   1.11011
+  0.0101001
  10.0010101
```

- Devido ao vai-um à esquerda, (r) não é descartado. Como o sticky bit é 1, o resultado correto é 10.0011_2
- Outro exemplo com precisão de 6 bits, usando complemento de 2

```
  1.00000
-0.0000101111
+1.1111010001
```

```
   1.00000
+  1.11111
  0.1111101 0001
```

Adição no Padrão IEEE-754

- Devido ao bit de guarda ter de ser acrescentado (pois o bit mais significativo foi zerado → 0.111110_2 resultado inicial;
- O bit (r) é 1 e o sticky bit é 1 (vem de 0001)
- Logo, o resultado final é 0.111111_2
- A seguir, apresenta-se esboço de algoritmo
  - Somar dois números representados no formato IEEE-754- 1985

Adição no Padrão IEEE-754

Sejam 2 números (a_1 e a_2) a somar. Expoentes e significandos são denotados por e_i e s_i, respectivamente. São 8 Passos!!

1) Se $e_1<e_2$, troque operandos (para que diferença $d=e_1-e_2 >=0$). Faça o expoente do resultado igual a e_1, temporariamente

2) Se sinais de a_1 e a_2 diferem, substitua s_2 por seu complemento de 2

3) Coloque s_2 em um registrador de *p* bits e desloque-o $d= e_1-e_2$ posições para a direita (entrando com 1's se s_2 foi complementado no passo 2). Dos bits deslocados para fora do registrador, guarde o último em um flip-flop *g*, o penúltimo em um flip-flop *r* e armazene o OU de todos os restantes como *sticky bit*

Adição no Padrão IEEE-754

4) Compute o significando preliminar $S=s_1 + s_2$, somando s_1 ao registrador de p bits contendo s_2. Se os sinais de a_1 e a_2 são diferentes, o bit mais significativo de S é 1 e não há vai-um, então S é negativo. Substitua S pelo seu complemento de 2. Isto só pode ocorrer quando d=0

5) Desloque S da seguinte maneira

5.1) Se os sinais de a_1 e a_2 são iguais e houve vai-um no passo 4, desloque S para a direita 1 bit, preenchendo a posição de mais alta ordem com 1 (o vai-um)

5.2) Senão, desloque S para a esquerda até normalizá-lo. Ao deslocar, no primeiro bit preencha a posição de ordem inferior com o bit g. Após, insira zeros. Ajuste o expoente do resultado de acordo

Adição no Padrão IEEE-754

6) Ajuste r e s (*round* e *sticky bit*). Se S foi deslocado à direita no passo 5, faça r ← bit de mais baixa ordem de S antes de deslocar e s ← g OR r OR s. Se não houve deslocamento, faça r ← g, s ← r OR s. Se houve um único deslocamento à esquerda, não mude r nem s. Se houve 2 ou mais deslocamentos, faça r ← 0 e s ← 0. (no último caso, 2 ou mais deslocamentos só podem acontecer quando a_1 e a_2 possuem sinais opostos e o mesmo expoente, em cujo caso a computação $s_1 + s_2$ será exata)

7) Arredonde S usando as regras de arredondamento. Se o arredondamento causar vai-um, desloque S à direita e ajuste o expoente. Isto é o significando do resultado

Adição no Padrão IEEE-754

8) Compute o sinal do resultado. Se a_1 e a_2 têm o mesmo sinal, este é o sinal do resultado. Se a_1 e a_2 possuem sinais diferentes, então o sinal depende de qual dentre a_1 e a_2 é negativo, se houve troca no passo 1 e se S foi substituído pelo seu complemento de 2 no passo 4. A Tabela abaixo resume os diferentes casos quando a_1 e a_2 possuem sinais diferentes

Troca p1	Complemento	Sinal(a1)	Sinal(a2)	Sinal(resultado)
Sim	-	+	-	-
Sim	-	-	+	+
Não	Não	+	-	+
Não	Não	-	+	-
Não	Sim	+	-	-
Não	Sim	-	+	+

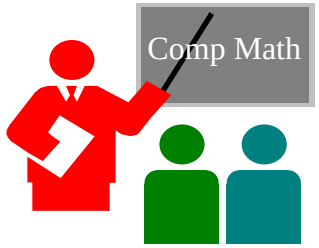
Adição no Padrão IEEE-754 – Exemplo

- Fazer: $(+1.001 * 2^{-2}) + (+1.111 * 2^{0})$ (a1 + a2)
- Passo 1
  - $e_1 < e_2$, logo troca
  - Calcula a distância entre expoentes: (d=2)
  - Expoente inicial igual ao expoente máximo (exp=0)
- Passo 2
  - Sinais iguais, logo não nega s_2
- Passo 3
  - Desloca s_2 (1.001 após a troca) à direita dois bits gerando s_2=0.010, g=0, r=1, s=0

Adição no Padrão IEEE-754

- Passo 4
  - 1.111 + 0.010 = (1)0.001 S=0.0001 com vai-um=1
- Passo 5
  - Como houve vai-um desloca S à direita, S=1.000, exp=exp+1, exp=1
- Passo 6 → atualiza g,r,s
  - r(bit de mais baixa ordem da soma)=1, s=g OR r OR s=0 OR 1 OR 0 =1
- Passo 7 → arredonda
  - r=1e s=1, então arredonda para cima S=S+1, S=1.001
- Passo 8: calcula o sinal
  - Ambos sinais positivos, então resultado é positivo

Sumário → That´s All Folks!!



- √ Introdução
- √ Aritmética Natural/Inteira
  - √ Operações sobre números sem sinal (naturais - N)
    - √ soma, subtração, multiplicação e divisão
  - √ Operações sobre números com sinal (inteiros - Z)
    - √ soma, subtração, multiplicação e divisão
- ➡ Aritmética não-Inteira (racionais - Q)
  - √ Representação de racionais – Q
  - √ Operações aritméticas usando IEEE-754