



Unidade 3

Aritmética Computacional

Uma breve introdução

Autores: Fernando Gehm Moraes & Ney Laert Vilar Calazans

Porto Alegre, junho de 1998

Última revisão: 30/novembro/2001



Sumário

Introdução



Introdução a Aritmética Computacional

Bibliografia:

- Patterson, D. A. "Computer Architecture: a quantitative approach", Morgan Kaufmann Pubs., 1996. Appendix A.
- Texas Instruments. "TMS320C4x User's Guide", 1996. Capítulo 5. Disponível na Internet (formato PDF):
 - » <http://www.s.ti.com/sc/psheets/spru063b/spru063b.pdf>
 - » <ftp://ftp.inf.pucrs.br/pub/calazans/texas/spru063b.pdf>



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:
 - meio-somador: $s_i = a_i \oplus b_i$, $c_{i+1} = a_i \wedge b_i$;
 - somador completo: $s_i = a_i \oplus b_i \oplus c_i$
 $c_{i+1} = a_i \wedge b_i \vee a_i \wedge c_i \vee b_i \wedge c_i$.



Sumário

Introdução



Aritmética Inteira

- Operações em números sem sinal (naturais)
 - » soma, subtração, multiplicação e divisão
- Representações de números com sinal (inteiros)

Aritmética não Inteira (racionais)

- Representação de racionais
- Operações com o padrão IEEE-754



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;
- Intel perdeu US\$ 300 milhões devido ao "bug" do Pentium (otimizou errado um PLA usado em ÷);
- Aqui, revisão estendida de inteiros e padrão para números racionais (IEEE-754).



Sumário

Introdução



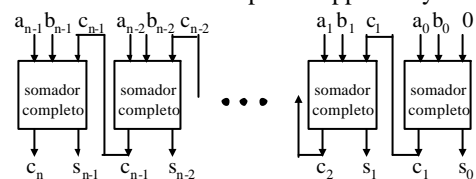
Aritmética Inteira

- Operações em números sem sinal (naturais)
 - » soma, subtração, multiplicação e divisão



Operações sobre naturais - Soma

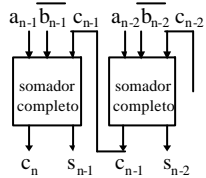
Estruturado somador completo "ripple-carry":



- Problema: atraso de geração do vai-um = $O(n)$, onde n é o número de bits do somador;
- Hardware adicional \rightarrow atraso = $O(\log n)$.

Operações sobre naturais - Subtração

⊗ Estruturado subtrator completo a-b “ripple-carry”:

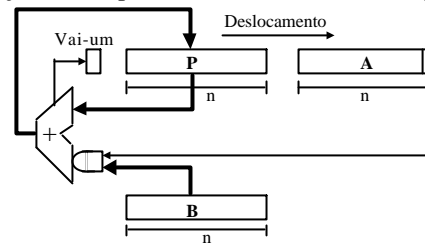


⊗ Mesmo problema de atraso, solução análoga;

⊗ Problema adicional, operação não definida se $b > a$.

Operações sobre naturais - Multiplicação

⊗ Solução natural para $a*b$: somas sucessivas n passos;

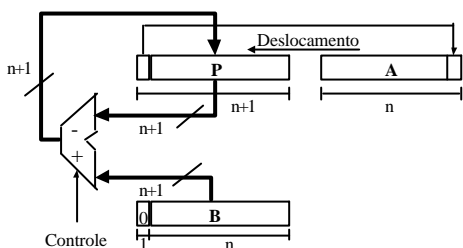


⊗ Inicialmente, $P=0, A=a, B=b$. Cada passo, duas partes:

- soma carregada em P;
- P & A deslocado um bit para a direita.

Operações sobre naturais - Divisão

➔ Solução para a/b : subtrações sucessivas, n passos;



➔ Cada passo, quatro partes: 1) desloca P&A p/ esq 1 bit; 2) $P \leftarrow P-B$; 3) If (passo $2 < 0$), $A_0=0$ else $A_0=1$; 4) If (passo $2 < 0$), restaura P fazendo $P \leftarrow P+B$.

Divisão A/B - Exemplo

$A = 11011$ (27)

$B = 00101$ (5)

A cada passo, mostra-se P e A após as partes

- 1) - primeira linha e
- 4) - segunda linha

1) desloca P&A p/ esq 1 bit;

2) $P \leftarrow P-B$;

3) If ($P < 0$), $A_0=0$ else $A_0=1$;

4) If ($P < 0$), restaura P fazendo $P \leftarrow P+B$

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

Divisão A/B - Exemplo

$A = 11011$ (27)

$B = 00101$ (5)

A cada passo, mostra-se P e A após as partes

- 1) - primeira linha e
- 4) - segunda linha

1) desloca P&A p/ esq 1 bit;

2) $P \leftarrow P-B$;

3) If ($P < 0$), $A_0=0$ else $A_0=1$;

4) If ($P < 0$), restaura P fazendo $P \leftarrow P+B$

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

Divisão A/B - Exemplo

$A = 11011$ (27)

$B = 00101$ (5)

A cada passo, mostra-se P e A após as partes

- 1) - primeira linha e
- 4) - segunda linha

1) desloca P&A p/ esq 1 bit;

2) $P \leftarrow P-B$;

3) If ($P < 0$), $A_0=0$ else $A_0=1$;

4) If ($P < 0$), restaura P fazendo $P \leftarrow P+B$

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

00110 - 00101 = 001

Divisão A/B - Exemplo

$A = 11011$ (27)

$B = 00101$ (5)

A cada passo, mostra-se P e A após as partes

- 1) - primeira linha e
- 4) - segunda linha

1) desloca P&A p/ esq 1 bit;

2) $P \leftarrow P-B$;

3) If ($P < 0$), $A_0=0$ else $A_0=1$;

4) If ($P < 0$), restaura P fazendo $P \leftarrow P+B$

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

Divisão A/B - Exemplo

$A = 11011$ (27)

$B = 00101$ (5)

A cada passo, mostra-se P e A após as partes

- 1) - primeira linha e
- 4) - segunda linha

1) desloca P&A p/ esq 1 bit;

2) $P \leftarrow P-B$;

3) If ($P < 0$), $A_0=0$ else $A_0=1$;

4) If ($P < 0$), restaura P fazendo $P \leftarrow P+B$

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

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 maior que na multiplicação;
- ⊗ Deve-se testar se divisor = 0!
- ⊗ Restauração desnecessária se teste na saída do somador/subtrator, bem como somador.

Sumário

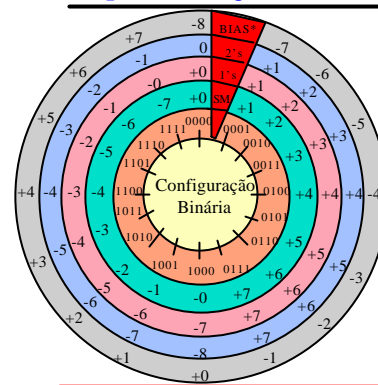
- ✓ Introdução
- ⊗ Aritmética Inteira
 - ✓ Operações em números sem sinal (naturais)
 - ✓ soma, subtração, multiplicação e divisão
 - Representações de números com sinal (inteiros)



Representações de Inteiros

- ⊗ Quatro métodos comuns (convenções) p/ inteiros:
 - 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 = $2^{(n-1)}$
- ⊗ Bias tem distribuição uniforme com relação a binários puros;
- ⊗ 2's facilita soma;
- ⊗ 1's facilita complementação;
- ⊗ SM é fácil de entender e separa sinal de valor;

Transbordo em Inteiros (Overflow)

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

Decimal	Binário	Decimal	Binário	Decimal	Binário
Vai-um: 00111		Vai-um: 11011		Vai-um: 00001	
+ 5	00101	- 5	11011	+ 5	00101
+ 7	00111	- 7	11001	- 7	11001
+12	01100	-12	10100	- 2	11110

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

Transbordo em Inteiros (Overflow)

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

Decimal	Binário	Decimal	Binário	Decimal	Binário
Vai-um: 00111		Vai-um: 11011		Vai-um: 00001	
+ 5	00101	- 5	11011	+ 5	00101
+ 7	00111	- 7	11001	- 7	11001
+12	01100	-12	10100	- 2	11110

Transbordos positivo e negativo

Decimal	Binário	Decimal	Binário
Vai-um: 01000		Vai-um: 10000	
+ 8	01000	- 8	11000
+ 9	01001	- 9	10111
+17	10001	-17	01111

- ▶ Xor de 2 últimos vai-uns identifica transbordo. **Negativo!** **Positivo!**

Sumário

- ✓ Introdução
- ✓ Aritmética Inteira
 - ✓ Operações em números sem sinal (naturais)
 - ✓ soma, subtração, multiplicação e divisão
 - ✓ Representações de números com sinal (inteiros)
- ⊗ Aritmética não Inteira (racionais)
 - Representação de racionais



Aritmética não Inteira (racionais)

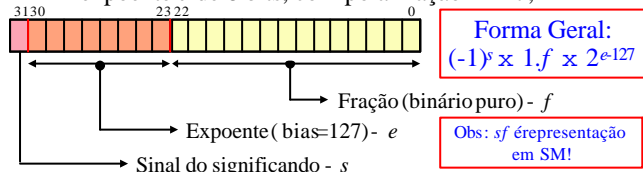
- ⊗ Muitas aplicações requerem números não-inteiros:
 - matemática computacional, engenharia, etc;
- ⊗ Racionais (Q): representados como fração a/b , a e b inteiros; Irracionais (I): têm mantissa infinita sem repetição ($e=2.7218...$, $\pi=3.14...$, por exemplo);
- ⊗ Reais: $Q \cup I$; I e R não representáveis em computadores, porquê?
- ⊗ Aproximação de reais em computadores: Q.

Aritmética não Inteira - representações

- Primeiros computadores - ponto (ou vírgula) fixo;
- Hoje, apenas ponto flutuante; representações possuem um *significandos* (a mantissa do número), um *expoente* e uma *base b*, e o número é dado por: sxb^e ;
- Antes, muitos formatos, hoje, padrão quase universal: definido pela IEEE (IEEE-754-1985), quatro formatos, dois fixos (precisão simples, SP e precisão dupla, DP) e dois variáveis (precisão simples, SE, e dupla, DE, estendidas). Igual ao padrão internacional IEC-559.

Uma Representação de racionais - IEEE-754-1985

- Formato SP ocupa exatamente 32 bits:
 - 1 bit para sinal do significando - s ;
 - significando com 24 bits de precisão, primeiro sempre 1, exceto quando denormalizado, onde é 0 (1º bit implícito);
 - expoente e de 8 bits, com polarização = 127;



IEEE-754-1985 - Valores especiais e denormalização

- Principal novidade do padrão é existência de valores especiais, que permitem tratar exceções:
 - valores infinitos;
 - divisão por zero;
 - operações para os quais os reais não são fechados, como raiz quadrada;
- Denormalização permite situações de "underflow" gradual, tal como: $00000001000000000000000000000000 (=2^{-126})$ dividido por 2^5 , representável se o primeiro dígito for 0 ao invés de 1, por: $00000000000100000000000000000000$.

IEEE-754-1985 - Parâmetros e outros formatos

- Os diferentes formatos e valores para os parâmetros que os definem:

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

Uma Representação de racionais - IEEE-754-1985

- Diferenças para formatos anteriores à padronização:
 - ao arredondar um resultado no meio da faixa, pega valor par mais próximo;
 - inclui valores especiais:
 - » NaN - Not a number (ex: raiz de negativo);
 - » $-\infty$ e $+\infty$ - Mais ou menos infinito (ex: $-1/0$ e $+1/0$);
 - números denormalizados para resultados com valor menor que $+1.0 \times 2^{\text{Emínimo}}$;
 - arredonda para mais próximo (default) mas tem mais três modos de arredondamento;
 - recursos sofisticados para lidar com exceções.

IEEE-754-1985 - Exemplos

- 1) Número positivo, maior que zero

31 30 23 22 0
0 10000001 010000000000000000000000

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

$$((-1)^0 \times 1.01 \times 2^{129-127})_2 = (+1.01 \times 2^2)_2 = (+5)_{10}$$

- 2) Número negativo, menor que zero

31 30 23 22 0
1 01111110 011100000000000000000000

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

IEEE-754-1985 - Valores especiais e denormalização

- Há 5 casos que definem o valor de número em algum formato do padrão IEEE-754 (exemplo para SP):

1) $e=255, f \neq 0$	$v = \text{NaN}$ (not a number)
2) $e=255, f=0$	$v = (-1)^s \infty$
3) $0 < e < 255$	$v = (-1)^s \times (1.f) \times 2^{e-127}$
4) $e=0, f \neq 0$	$v = (-1)^s \times (0.f) \times 2^{-126}$
5) $e=0, f=0$	$v = (-1)^s \times 0$ (zero)

Sumário

- ✓ Introdução
- ✓ Aritmética Inteira
 - ✓ Operações em números sem sinal (naturais)
 - ✓ soma, subtração, multiplicação e divisão
 - ✓ Representações de números com sinal (inteiros)
- Aritmética não Inteira (racionais)
 - ✓ Representações de racionais
 - Operações com o padrão IEEE-754



- ⊗ Operação mais fácil de se implementar em hardware:
 - multiplicação e não soma
 - » adaptação de significandos devido a diferentes expoentes complica soma;
 - » multiplicação é direta, multiplica significandos e soma expoentes, exceto quando resultado é caso especial;
- ⊗ Veremos aqui introdução a ambos, multiplicação e soma.

- ⊗ 3 Passos:
 - multiplicar significandos (não fração, desempacotar o número) usando multiplicação inteira, sem sinal (SM);
 - calcular expoente
 - » lembrar da polarização.
 - arredondamento devido ao aumento da precisão após operação;

- ⊗ Exemplo de multiplicação:


```
1 10000010 000000000000000000000000 = -1x23 = -8
0 10000011 000000000000000000000000 = 1x24 = 16
```

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

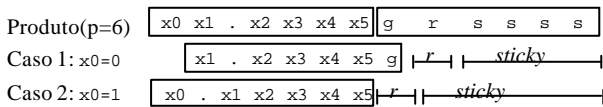

```
1 ???????? 000000000000000000000000
```
 - 2) Expoente - fórmula para cálculo do expoente:

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

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

- 3) arredondamento - precisão é importante:
 - » casos de arredondamento (em decimal, análogo a binário):
 - a) $1.23 \times 6.78 = 8.3394$ $r=9, 9>5$, então arredonda p/8.34
 - b) $2.83 \times 4.47 = 12.6501$ $r=5$, e pelo menos um dígito após diferente de 0, arredonda p/ 1.27x10¹
 - c) $1.28 \times 7.81 = 9.9968$ $r=6, 6>5$, então arredonda p/ 1.00x10¹
 - » em binário, meio da faixa (5 em decimal) é dígito 1!
 - » **negrito** - dígitos significativos; após, dígito arredondador, *r*.

- 3) arredondamento (continuação)
 - » se *r* é menor que 5 - resultado pronto;
 - » se *r* é maior que 5 - soma-se 1 ao número em negrito;
 - » se arredondador exatamente 5 (em binário, 1) - examina 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/ esq; Caso 2 - incrementa expoente.



- 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.

- ⊗ Controlar underflow é complicado, devido aos denormalizados:
 - por exemplo, $1x2^{-64} \times 1x2^{-65} = 1x2^{-129}$, 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 denormalizado, geram-se 0s à esquerda;
- ⊗ Denormalizados causam montes de problemas p/ multiplicação;
- ⊗ Multiplicadores de alto desempenho ignoram denormalizados, gerando exceções para o software cuidar;
- ⊗ Denormalizados não são frequentes, perda em tempo é estatisticamente pequena.

- ⊗ Se um dos multiplicandos é 0, acelera-se multiplicação testando:
 - antes - ambos operandos;
 - depois - neste caso, cuidado com $0 \times \infty$, resultado deve ser NaN;
 - sinal deve ser mantido, $+0$ é diferente de -0 ;
- ⊗ Ao multiplicar pode-se precisar do dobro de bits em inteiros; aplicação define se é aceitável apenas a metade inferior do resultado ou se todo o resultado deve ser usado;
- ⊗ Em linguagens de alto nível, multiplicação inteira sempre usam a primeira opção, mas ponto flutuante é diferente, e as duas são usadas, utilidade é grande na solução de equações lineares;

- Tipicamente, operação em ponto flutuante recebe dois números de mesma precisão e retorna resultado com mesma precisão, p ;
- Algoritmo ideal (erro menor) calcula resultado exato e arredonda;
- Multiplicação funciona assim;
- Para soma, existem procedimentos mais efetivos;
- Exemplo com números de 6 bits: 1.1001_2 e $1.10001_2 \times 2^{-5}$;
- Usando um somador de 6 bits, tem-se:

$$\begin{array}{r} 1.10011 \\ + 0.00001 \\ \hline 1.10100 \end{array}$$

- No exemplo, bit descartado é 1, logo resto deveria ser examinado;
- Novamente, apenas se precisa saber se um destes bits é não-zero, e pode-se assim usar “sticky bits”, como na multiplicação;
- Logo, para somar números de p bits, um somador de p bits chega, desde que se guarde o primeiro bit descartado e o “sticky bit” correspondente;
- No exemplo acima, o sticky é 1, e o resultado final fica 1.10101_2 ;
- Subtração é similar, se se trabalha em complemento de dois;
- A seguir, apresenta-se o algoritmo para somar dois números representados no formato IEEE-754-1985.

Operação de adição de a_1 e a_2 - Algoritmo

- Se $e_1 < e_2$, trocar operandos (para que diferença $e_1 - e_2 \geq 0$). Fazer o expoente do resultado igual a e_1 , temporariamente;
- Se sinais de a_1 e a_2 diferem, substituir s_2 por seu complemento de 2;
- Colocar s_2 em um registrador de p bits e deslocá-lo $d = e_1 - e_2$ 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*.
- ... Este e mais outros quatro passos são complicados como o passo 3, somando 8 passos no algoritmo. E tudo isso implementado em hardware!!!!

Soma

- fazer: $-1.001 \times 2^{-2} + -1.111 \times 2^0$ ($a_1 + a_2$)
- passo 1:
 - se $a_1 < a_2$ swap a_1 e a_2
 - calcula a distância entre expoentes: ($d=2$)
 - expoente igual ao expoente máximo (0)
- passo 2:
 - se sinais diferentes substituir a_2 pelo seu complemento de 2
- passo 3: desloca a_2 d dígitos para a direita, seta g, r, s
 - no exemplo: $0.010\ 01$, logo $g=0, r=1, s=0$

Soma

- passo 4: SOMA
 - $1.111 + 0.010 = 0.001$ e $cout=1$
- passo 5: *muito complicado...*
 - Se mesmo sinal: desloca para a direita, inserindo 1 e atualiza expoente
 - 1.0001 e expoente passa a ser 1
- passo 6: atualiza g, r, s
 - $g=1, r=0, s=1$
- passo 7: arredonda
 - $1.000\ 1\ 0\ 1 \implies 1.0010$ (ou seja: $10.001 = 2,125$)
- passo 8: calcula o sinal *muito complicado...*
 - Se mesmo sinal: não muda

Exercícios

Supor que tenhamos 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 para 12 bits, com as mesmas condições para NaN, 0, $\pm \infty$).

Pergunta-se:

- Qual o valor da polarização e quais os expoentes mínimo e máximo?
- Converta os seguintes números para base decimal:
 - 0 1010 1011000
 - 1 0111 0101000
- Multiplicar os dois números acima, mostrando o procedimento da multiplicação para ponto flutuante, com arredondamento.
- Qual o resultado, em decimal, arredondado, e qual o erro advindo do arredondamento?

Exercícios

Supor que tenhamos 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 para 12 bits, com as mesmas condições para NaN, 0, $\pm \infty$).

Qual o valor da polarização e quais os expoentes mínimo e máximo?

Valor da polarização: 7,
Expoente máximo +7 (1110-7) e exp mínimo -7 (0000-7).

Converta para base decimal: 0 1010 1011000 e 1 0111 0101000
 $+2^{(10-7)} * 1.1011 = +2^3 * 1.1011 = 1101.1 = 13.5$
 $-2^{(7-7)} * 1.0101 = 2^0 * 1.0101 = 1.0101 = -1.3125$

Exercícios

Supor que tenhamos 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 para 12 bits, com as mesmas condições para NaN, 0, $\pm \infty$).

Multiplicar os dois números anteriores (0 1010 1011000 e 1 0111 0101000), mostrando o procedimento da multiplicação para ponto flutuante, com arredondamento.

Resposta: expoente: $10+7-7 = 10$
 partes fracionárias: $1.1011 * 1.0101 = 10.00110111$
 fica $2^{10} * 10.00110111 = 2^{11} * 1.0001101\ 11$

Soma-se uma ou menos significativo pois $round=1$ e $stick=1$

$1\ 1011\ 0001\ \underline{10} = -17.75$

Exercícios

Supor que tenhamos 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 para 12 bits, com as mesmas condições para NaN, 0, $\pm \infty$).

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

Resposta: 17.75, ao invés de 17.71875 ($3.5 * 1.3125$)

Finalmente acabou (esta parte)!



Fim da Unidade 3!