"Se você realmente quer aprender, você deve montar a máquina e se acostumar com seus detalhes na prática" (Wilbur Wright).

Adição e Subtração

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Contando em binário e em decimal

Base 10	Base 2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
...	...

Adições

Realizar uma adição em binário (ou em qualquer outra base) segue o mesmo raciocínio da base 10

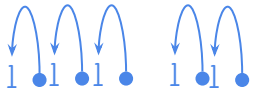
Adições

Realizar uma adição em binário (ou em qualquer outra base) segue o mesmo raciocínio da base 10

```
  00111001
+ 00011011
```

Adições

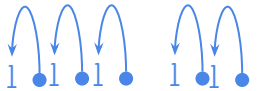
Realizar uma adição em binário (ou em qualquer outra base) segue o mesmo raciocínio da base 10




```
 00111001
+00011011
 01010100
```

Adições

Realizar uma adição em binário (ou em qualquer outra base) segue o mesmo raciocínio da base 10




```
 00111001
+00011011
 01010100
```

Os “vai um” são chamados de **bits de *carry***.

Faça você mesmo

Realize a seguinte soma em binário

```
  01110001
+ 11111000
```

Faça você mesmo

Realize a seguinte soma em binário

```
  1 1 1 1
  01110001
+ 11111000
-----------
101101001
```

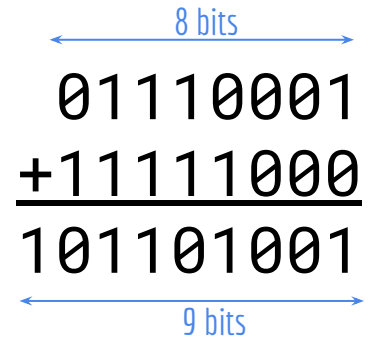
Faça você mesmo

No exemplo os operandos possuem 8 bits, e o resultado possui 9.

Nono bit gerado devido ao carry final

No “lápis e papel” isso não é um problema

Para a **máquina** isso pode ser um **problema sério**



Overflow

Exemplo

Ao armazenar os valores na memória em duas variáveis de 8 bits cada (ex.: um *unsigned char* em C) , e realizar a conta de forma que o resultado seja armazenado em outra variável de 8 bits

O bit extra vai ser desconsiderado por não caber na região de memória

O resultado será incorreto

Ocorreu um **overflow** (transbordo)

Teste você mesmo

```c
#include<stdio.h>
int main(){
      //Em máquinas desktop (x86-64) comumente um char ocupa 8 bits,
      //e um short ocupa 16 bits

      unsigned char var1 = 0b01110001;//113 na base 10
      unsigned char var2 = 0b11111000;//248 na base 10
      unsigned char resultadoChar;
      unsigned short resultadoShort;

      resultadoChar = var1+var2;
      resultadoShort = var1+var2;

      printf("%u %u\n", resultadoChar, resultadoShort);
      return 0;
}
```

Ao executar, o resultado do programa é
105 361

Quanto isso pode custar

Quanto isso pode custar?

https://youtu.be/PK_yguLapgA

Causado por um overflow

O overflow foi causado por uma conversão (cast) de uma variável de 64 bits para uma de 16 bits, mas o princípio é o mesmo que estudamos

Quanto isso pode custar

Quanto isso pode custar?

https://youtu.be/PK_yguLapgA

Causado por um overflow

O overflow foi causado por uma conversão (cast) de uma variável de 64 bits para uma de 16 bits, mas o princípio é o mesmo que estudamos

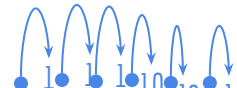
A explosão custou 370 milhões de dólares

Subtração

```
  10001100
-00111110
```

Subtração

O princípio da subtração também é o mesmo usado na base 10



```
 10001100
-00111110
 01001110
```

Outro exemplo

Converta para binário e realize a subtração. Use 8 bits para representar os operandos.

$15_{10} - 42_{10}$

Outro exemplo

Converta para binário e realize a subtração. Use 8 bits para representar os operandos.

$15_{10} - 42_{10}$

```
  00101010
-00001111
-00011011
```

Outro exemplo

Converta para binário e realize a subtração. Use 8 bits para representar os operandos.

$15_{10} - 42_{10}$

```
  00101010
- 00001111
-----------
- 00011011
```

O maior fica na parte superior

Como 15 < 42, o resultado é negativo

Outro exemplo

Converta para binário e realize a subtração. Use 8 bits para representar os operandos.

$15_{10} - 42_{10}$

```
  00101010
- 00001111
-00011011
```

O maior fica na parte superior

Como 15 < 42, o resultado é negativo

Note que o computador só representa zeros e uns, então representar o negativo é um problema. Vamos discutir isso nas próximas aulas. No papel, você pode usar o símbolo - normalmente para indicar um resultado negativo.

Multiplicando e dividindo pela base

Na base 10, quando desejamos multiplicar um número por 10, nós simplesmente “deslocamos” o valor para a esquerda e adicionamos um zero no final

Por exemplo, multiplicar 38 por 10

38 x 10 = 038 x 10 = 380

Desloca uma vez para a esquerda e completa com zero

Multiplicando e dividindo pela base

O mesmo é válido para outras bases. Em uma base β qualquer, ao se deslocar uma vez o número para a esquerda e completar com zero, estamos multiplicando o valor por β

Por exemplo, para multiplicar o número 11_2 por 2 basta:

$11_2 \times 2 = 011_2 \times 2 = 110_2$

Multiplicando e dividindo pela base

Continuando com um exemplo na base 2

Se desejamos multiplicar por 2^n, basta deslocar *n* casas para a esquerda

Mais uma vez, o mesmo raciocínio que usamos para a base 10

Exemplo:

$11_2 \times 8 = 11_2 \times 2^3 = 00011_2 \times 2^3 = 11000_2$

Prova

Considere um número *a* qualquer, cuja representação polinomial é

$a = a_j\beta^j + a_{j-1}\beta^{j-1} + \dots + a_2\beta^2 + a_1\beta^1 + a_0\beta^0$

Ao multiplicar esse número por β^n, onde *n* é um número inteiro, temos que

$\beta^n a = \beta^n(a_j\beta^j + a_{j-1}\beta^{j-1} + \dots + a_2\beta^2 + a_1\beta^1 + a_0\beta^0)$

$\beta^n a = a_j\beta^{j+n} + a_{j-1}\beta^{j-1+n} + \dots + a_2\beta^{2+n} + a_2\beta^{1+n} + a_1\beta^n + 0\beta^{n-1} + 0\beta^{n-2} + \dots + 0\beta^0$

Prova

Considere um número *a* qualquer, cuja representação polinomial é

$$a = a_j\beta^j + a_{j-1}\beta^{j-1} + \dots + a_2\beta^2 + a_1\beta^1 + a_0\beta^0$$

Ao multiplicar esse número por β^n, onde *n* é um número inteiro, temos que

$$\beta^n a = \beta^n(a_j\beta^j + a_{j-1}\beta^{j-1} + \dots + a_2\beta^2 + a_1\beta^1 + a_0\beta^0)$$

$$\beta^n a = a_j\beta^{j+n} + a_{j-1}\beta^{j-1+n} + \dots + a_2\beta^{2+n} + a_2\beta^{1+n} + a_1\beta^n + 0\beta^{n-1} + 0\beta^{n-2} + \dots + 0\beta^0$$

Ao transformar o polinômio para notação posicional, estamos deslocando n casas para a esquerda!

Esses zeros não fazem diferença no polinômio. Só foram adicionados para deixar mais claro que na notação posicional completamos com zeros à direita.

Multiplicando e dividindo pela base

Por um raciocínio análogo, podemos fazer uma divisão inteira por 2^n deslocando o número *n* vezes para a direita, "jogando fora" os bits deslocados

Muito cuidado, pois a divisão é inteira, ou seja, a parte fracionária é jogada fora

Mais uma vez, o mesmo raciocínio da base 10

Exemplo: $1111_2/4$

$1111_2 / 4 = 1111_2 / 2^2 = 11_2$

E de que isso importa?

As divisões e multiplicações que aprendemos **só valem quando vamos multiplicar ou dividir por 2^n** (ou por β^n em uma base β qualquer)

Vamos aprender o caso geral da multiplicação nas próximas aulas

E de que isso importa?

As divisões e multiplicações que aprendemos **só valem quando vamos multiplicar ou dividir por 2^n** (ou por β^n em uma base β qualquer)

Vamos aprender o caso geral da multiplicação nas próximas aulas

Realizar a multiplicação/divisão por 2^n utilizando deslocamentos é **muito** (mas muito) mais fácil do que uma multiplicação convencional para nós **humanos e para a máquina**

A maioria dos hardwares implementa deslocamentos de bits internamente

O processador do seu desktop, notebook, celular, aparelho de microondas, ... sabem fazer isso

E de que isso importa?

Então, ao multiplicar ou dividir por 2^n, sempre opte por deslocamentos de bits

Como acessar isso depende da linguagem de programação (mas lembre-se, a linguagem de programação é só uma forma de instruir o hardware a fazer algo, e quem está fazendo isso é o hardware)

Em C por exemplo, utiliza-se os operadores ›› e ‹‹

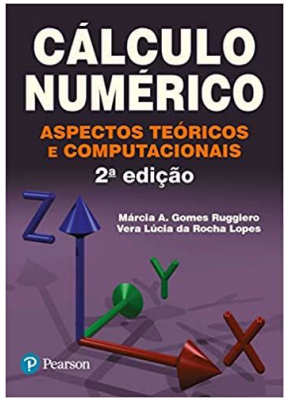
Veja em: *en.wikipedia.org/wiki/Bitwise_operations_in_C#Right_shift_%3E%3E*

Exercícios

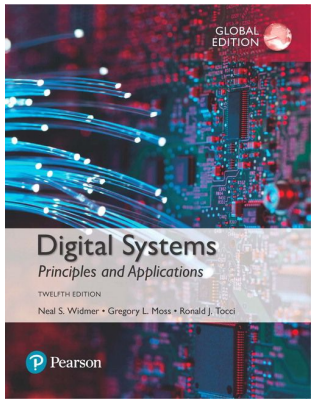
1. Converta os valores a seguir para binário e realize as operações. Para divisões, considere a divisão inteira. Represente os operandos com 8 bits.
    a. 43 + 50
    b. 127 + 128
    c. 255 + 255
    d. 12 - 3
    e. 255 - 127
    f. 128-15
    g. 12 - 128
    h. 14 * 16
    i. 36 / 32
    j. 41 / 64
2. Ao deslocar um valor binário *n* vezes para a esquerda, onde *n* é um número natural, vamos sempre obter um número par? Argumente sobre isso.
3. Ao deslocar um valor binário *n* vezes para a direita, onde *n* é um número natural, vamos sempre obter um número par, ímpar, ou não podemos afirmar nada? Argumente sobre isso.

Referências

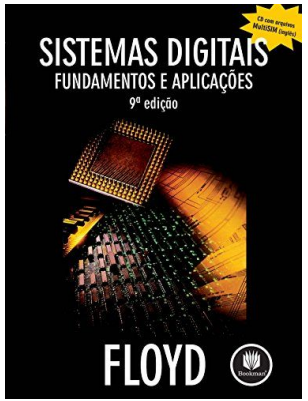
Marcia A. G. Ruggiero, Vera L. R. Lopes. Cálculo numérico aspectos teóricos e computacionais. 1996.



Ronald J. Tocci, Gregory L. Moss, Neal S. Widmer. Sistemas digitais. 10a ed. 2017.



Thomas Floyd. Widmer. Sistemas Digitais: Fundamentos e Aplicações. 2009.



Licença