Informações

PONTIFÍCIA UNIVERSISADE CATÓLICA DE MINAS GERAIS

Instituto de Ciências Exatas e Informática – ICEI

Autor : Vinícius Miranda de Araújo

Matéria : Arquitetura de Computadores I

Ano : 2024

Estrutura do Documento

Sistemas de Numeração

Conversão Entre Bases

Parte Inteira

- Converter `decimal` para `binário` :

```
 Sistema decimal :
163(10) = 1x10^2 + 6x10^1 + 3x10^0  - na forma canônica

Para converter um valor decimal ( base = 10 ) para binário ( base = 2 ), usar divisões sucessivas por 2 e tomar
os restos na ordem inversa em que forem calculados:

operação         quociente      resto
163 / 2     =       81      +     1   ( último )
81  / 2     =       40      +     1
40  / 2     =       20      +     0
20  / 2     =       10      +     0
10  / 2     =        5      +     0
5   / 2     =        2      +     1
2   / 2     =        1      +     0
1   / 2     =        0      +     1   ( primeiro )

Sistema binário :
1010 0011(2) - número na base 2
```

- Converter `binário` para `decimal` :

```
 Para converter um valor binário ( base = 2 ) para decimal ( base = 10 ), usar a soma dos produtos de cada
algarismo pela potência da base equivalente à posição:

Sistema binário :
1010 0011(2) - número na base 2

Sistema decimal :
1x2^7 + 0x2^6 + 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 1x2^0  - forma canônica
128   +  0   +  32   +  0   +  0  +  0    +   2   +  1     = 163(10)
```

- Converter `decimal` para `base 4` (quaternário) :

```
 Para converter um valor decimal para a base 4 ( quaternário ):

operação      quociente    resto
163 / 4     =   40      +     3   ( último )
40  / 4     =   10      +     0
10  / 4     =    2      +     2
2   / 4     =    0      +     2   ( primeiro )

Sistema quaternário :
2203(4) - número na base 4
```

- Converter `decimal` para `base 8` (octal) :

```
 Para converter um valor decimal para a base 8 ( octal ):

operação      quociente    resto
163 / 8     =   20      +     3   ( último )
20  / 8     =    2      +     4
2   / 8     =    0      +     2   ( primeiro )

Sistema octal :
243(8) - número na base 8
```

- Converter `decimal` para `base 16` (hexadecimal) :

```
 Para converter um valor decimal para a base 16 ( hexadecimal ):

operação      quociente    resto
163 / 16   =    10      +     3   ( último )
10  / 16   =     0      +    10   ( primeiro, substituindo pelo algarismo A=10 )

Sistema hexadecimal :
A3(16) - número na base 16
```

- Converter da `base 4` para `decimal` :

```
 Sistema quaternário :
2203(2) =  2x4^3 + 2x4^2 + 0x4^1 + 3x4^0  - número na base 4 na forma canônica
        =    128 + 32    +   0   + 3      = 163(10)
```

- Converter da `base 8` para `decimal` :

```
 Sistema octal :
243(8) = 2x8^2 + 4x8^1 + 3x8^0  - número na base 8 na forma canônica
       = 128   + 32    +   3    = 163(10)
```

- Converter da `base 16` para `decimal` :

```
 Sistema hexadecimal :
A3(16) = (A=10)x16^1 + 3x16^0  - número na base 16 forma canônica
       =        160  +  3      = 163(10)
```

> [!TIP]
>
> **Converter entre bases potências múltiplas sem passar para decimal:**
>
> As bases que são potências múltiplas de outra compartilham propriedades especiais, como a possibilidade de conversões entre elas, sem passar pela base decimal:
>
> Obs: Caso necessário, completar com zeros para formar os grupos.
>
> - Sistema `binário` (base = 2) para `quaternário` (base = 4 = 2^2) :
>
> ```
>  Agrupar de 2 em 2 e substituir pelos dígitos equivalentes :
>
> 1010 0011(2) = [10] [10] [00] [11] (4) = 2203(4)
> ```
>
> - Sistema `binário` (base = 2) para `octal` (base = 8= 2^3):
>
> ```
>  Agrupar de 3 em 3 e substituir pelos dígitos equivalentes :
>
> 1010 0011(2) = [010] [100] [011] (8) = 243(8)
> ```

> - Sistema `binário` (base = 2) para `hexadecimal` (base = 16 =2^4):
>
>   ```
>   Agrupar de 4 em 4 e substituir pelos dígitos equivalentes :
>
>   1010 0011(2) = [1010] [0011] (16) = A3(16) e A(16)=10
>   ```

Parte Fracionária

- Sistema decimal :

  ```
   0,6875(10) = 6x1^(0-1) + 8x1^(0-2) + 7x1^(0-3) + 5x1^(0-4) - na forma canônica

  Para converter a parte fracionária de um valor decimal (base=10) para binário (base=2), usar multiplicações
  sucessivas por 2 e tomar as partes inteiras na mesma ordem em que forem calculados, prosseguindo com a parte
  fracionária restante.

  operação       produto     parte inteira    parte fracionária   binário
  0,6875 * 2  =  1,3750    =       1            ,3750               0,1       ( primeiro )
  0,3750 * 2  =  0,7500    =       0            ,7500               0,10
  0,7500 * 2  =  1,5000    =       1            ,5000               0,101
  0,5000 * 2  =  1,0000    =       1            ,0000               0,1011    ( último )

  Parar, se a parte fracionária se tornar igual a zero.
  ```

- Sistema binário :

  ```
   0,1011(2) - número na base 2

  ou

      2^-1     2^-2      2^-3       2^-4   - potências negativas da base 2
       0,5     0,25     0,125     0,0625   - valor decimal da potência na base 10
  0,     1        0         1          1   - coeficientes

  Caso a parte fracionária não se tornar igual a zero dentro de certo número de operações, parar quando for
  alcançada a precisão desejada ou se esgotar a quantidade de casas disponíveis. Também podem surgir dízimas,
  periódicas ou não.

  operação    produto    parte inteira      parte fracionária     binário
  0,69     *     2     =     1,38        =       1     ,38          0,1            ( primeiro )
  0,38     *     2     =     0,76        =       0     ,76          0,10
  0,76     *     2     =     1,52        =       1     ,52          0,101
  0,52     *     2     =     1,04        =       1     ,04          0,1011
  0,04     *     2     =     0,08        =       0     ,08          0,10110
  0,08     *     2     =     0,16        =       0     ,16          0,101100
  0,16     *     2     =     0,32        =       0     ,32          0,1011000
  0,32     *     2     =     0,64        =       0     ,64          0,10110000
  0,64     *     2     =     1,28        =       1     ,28          0,101100001
  0,28     *     2     =     0,56        =       0     ,56          0,1011000010
  0,56     *     2     =     1,02        =       1     ,02          0,10110000101
  0,02     *     2     =     0,04        =       0     ,04          0,101100000010 ( dízima )
  ```

- Para converter um valor `decimal` para a `base 4` (quaternário) :

  ```
   operação       produto     parte inteira    parte fracionária   quaternário
  0,6875 * 4  =  2,7500    =        2          ,7500                  0,2          ( primeiro )
  0,7500 * 4  =  3,0000    =        3          ,0000                  0,23         ( último )
  ```

- Sistema quaternário :

  ```
   0,23(4) - número na base 4

  Por agrupamento do binário equivalente  e substituição do valor binário por dígitos dessa base:

  0,1011(2) = 0, [10] [11] (4) = 0,23(4) - agrupar de 2 em 2 para a direita
  ```

- Para converter um valor `decimal` para a `base 8` (octal) :

  ```
   operação       produto    parte inteira    parte fracionária   octal
  0,6875 * 8  =  2,7500    =        5          ,5000                     0,5             (primeiro)
  0,5000 * 8  =  4,0000    =        4          ,0000                     0,4             (último)
  ```

- Sistema octal :

```
0,54(8) - número na base 8

Por agrupamento do binário equivalente e completando com zeros (0), se necessário, e substituição do valor
binário por dígitos dessa base:

0,1011(2) = 0, [101] [100] (8) = 0,54(8) - agrupar de 3 em 3 para a direita
```

- Para converter um valor `decimal` para a `base 16` (hexadecimal):

```
operação      produto     parte inteira    parte fracionária   hexadecimal
0,6875 * 16 = 2,7500    =        11        ,0000                  0,B         ( primeiro e último )
```

- Sistema hexadecimal :

```
0,B(16) - número na base 16

Por agrupamento do binário equivalente e substituição do valor binário por dígitos dessa base:

0,1011(2) = 0, [1011] (16) = 0,B(16) - agrupar de 4 em 4 para a direita
```

Para converter um valor fracionário em `binário` (base = 2) para `decimal` (base = 10), usar a soma dos produtos de cada algarismo pela potência negativa da base equivalente à posição:

- Sistema binário :

```
0,1011(2)      - número na base 2

Sistema decimal :

1x2^-1 + 0x2^-2 + 1x2^-3 + 1x2^-4 - forma canônica
1/21  +   0  + 1/23 + 1/24
1/2   +   0  + 1/8  + 1/16    = (8+2+1)/16
0,5   +   0 + 0,125 + 0,0625  = 0,6875(10)

Para converter um valor da base 4 (quaternário) para decimal:
```

- Sistema quaternário :

```
0,23(4) - número na base 4

Sistema decimal :

2x4-1 + 3x4-2+ 0x4-3 + 0x4-4 - forma canônica
2/41   + 3/42   + 0/43   + 0/44
2/4    + 3/16 + 0/64 + 0/256   = (8+3)/16
0,5    + 0,1875 + 0  + 0          = 0,6875(10)

Para converter um valor da base 8 (octal) para decimal:
```

- Sistema octal :

```
0,54(8) - número na base 8

Sistema decimal :

5x8^-1 + 4x8^-2 + 0x8^-3 + 0x8^-4 - forma canônica
5/81   + 4/82   + 0/83   + 0/84
5/8    + 4/64   + 0/512  + 0/4096 = (40+4)/64
0,625  + 0,0625 +  0     + 0      = 0,6875(10)

Para converter um valor da base 16 (hexadecimal) para decimal:
```

- Sistema hexadecimal :

```
0,B(16) - número na base 16

Sistema decimal :

11x16^-1 + 0x16^-2 + 0x16^-3 + 0x16^-4 - forma canônica
11/161  + 0/162    + 0/163   + 0/164
11/16   + 0/256    + 0/4096  + 0/65536 = (11)/16
0,6875  +    0     +    0    + 0       = 0,6875(10)
```

Operações Aritméticas

Adição

- Sistema binário :

```
 Relações fundamentais:

0(2) + 0(2) =   0(2)
0(2) + 1(2) =   1(2)
1(2) + 0(2) =   1(2)
1(2) + 1(2) =  10(2)   ( zero e "vai-um" para a próxima potência )

Aplicação:

    1111      ← "vai-um"
   101101(2)  ← operando 1
+     111(2)  ← operando 2
   _________
   110100(2)  ← resultado
```

- Sistema quaternário :

```
 Aplicação:

   1111               11      ← "vai-um"   ( excessos de 4 )
  101101(2)           231(4)  ← operando 1
+    111(2)         +  13(4)  ← operando 2
  _________         _______
  110100(2)           310(4)  ← resultado
```

- Sistema octal :

```
 Aplicação:

   1111               1      ← "vai-um"   ( excessos de 8 )
  101101(2)           55(8)  ← operando 1
+    111(2)         +  7(8)  ← operando 2
  _________           _____
  110100(2)           64(8)  ← resultado
```

- Sistema hexadecimal :

```
 Aplicação:

    1111               1       ← "vai-um"   ( excessos de 16 )
   101101(2)           2D(16)  ← operando 1
+     111(2)        +   7(16)  ← operando 2
 _________          ______
   110100(2)           34(16)  ← resultado
```

Subtração

```
 Relações fundamentais:

  0(2) - 0(2) =   0(2)
  0(2) - 1(2) = ???
  1(2) - 0(2) =   1(2)
  1(2) - 1(2) =   0(2)
 10(2) - 1(2) =  01(2) ( zero e "vem-um" para a potência considerada )
100(2) - 1(2) = 011(2) ( zero e "vem-um" para as potências necessitadas )

Aplicação:
                   1               (10)                               1      ← "vem-um"
 101101(2)        101001(2)        1010 (0)1(2)        100 (10) 01(2)       101101(2)  ← operando 1
-   111(2)       -   111(2)       -   1  1 1(2)       -      1  11(2)       -  111(2)  ← operando 2
 _________        _________        ____________        _______________      _________
      0(2)             0(2)               1 0(2)        100   1  10(2)      100110(2)  ← resultado
```

> [!NOTE]
>
> Quando se "toma emprestado" na potência seguinte, um valor unitário é debitado na potência que "empresta", é “creditado” na potência que o recebe, compensada a diferença entre essas potências.

Multiplicação

- Sistema binário

```
 Relações fundamentais:

0(2) * 0(2) = 0(2)
0(2) * 1(2) = 0(2)
1(2) * 0(2) = 0(2)
1(2) * 1(2) = 1(2)

Aplicação:

   101101(2)   ← operando 1
*     101(2)   ← operando 2
 ___________
    1111
     101101
+   000000-
   101101--
 ___________
   11100001(2) ← resultado
```

Divisão

- Sistema binário :

```
 Aplicação:

  11100001(2) ÷ 101 (2)                 11100001(2) ÷ 101 (2)
- 101         =   1(2)                - 101         =   10(2)
  ___                                    ____
  010                                    0100


  11100001(2) ÷ 101 (2)                 11100001(2) ÷ 101 (2)
- 101         = 101(2)                - 101         = 1011(2)
  _____                                 _____
  01000                                  01000
-   101                                  - 101
  _____                                 ______
  00011                                 000110
                                         - 101
                                        ______
                                        0000010


  11100001(2)  ÷ 101 (2)                 11100001(2)  ÷ 101 (2)
- 101          = 10110(2)              - 101          = 101101(2)
  _____                                  _____
  01000                                  01000
- 101                                  -   101
  ______                                 ______
  000110                                 000110
-    101                               -    101
  ________                               ________
  00000101                               00000101
-      101
  ________
  00000000
```

Representação de Dados

A representação binária depende da quantidade de bits disponíveis e dos formatos escolhidos.

Para os valores inteiros, por exemplo, pode-se utilizar o formato em que o primeiro bit, à esquerda, para o sinal e o restante para a amplitude, responsável pela magnitude (grandeza) do valor representado.

Exemplo:

```
  5(10)  =  101(2)

+5(10) = 0101(2)

-5(10) = 1101(2)
```

Essa represesentação, contudo, não é conveniente para realizar operações, pois ao adicionar ambos, obtém-se:

```
 +5(10)     =     0101(2)
-5(10)     =     1101(2)
________         _______
0(10)      = (1) 0010(2)
```

O que ultrapassa a quantidade de bits originalmente escolhida e, obviamente, não é igual a zero em sua amplitude.

Complemento de 1

Uma das possíveis representações para valores negativos pode ser aquela onde se invertem os valores individuais de cada bit.

Exemplo:

```
  5(10) =  101(2)

+5(10) = 0101(2)

-5(10) = 1010(2)   (complemento de 1)
```

Essa represesentação, contudo, também não é conveniente para realizar operações, pois ao adicionar ambos, obtém-se:

```
  +5(10)    =      0101(2)
 -5(10)    =      1010(2) +
_______          ________
- 0(10)    =      1111(2) → +0(10) = 0000(2)
```

O que mantém a quantidade de bits originalmente escolhida, mas gera duas representações para zero (-0) e (+0), o que requer ajustes adicionais nas operações.

Complemento de 2

Outra das possíveis representações para valores negativos pode ser aquela onde se invertem os valores individuais de cada bit, e acrescenta-se mais uma unidade ao valor encontrado, buscando completar o que falta para atingir a próxima potència da base.

Exemplo:

```
  5(10) =  101(2)

+5(10) = 0101(2)

-5(10) = 1010(2)   ( complemento de 1, ou C1(5) )

-5(10) = 1011(2)   ( complemento de 2, ou C2(5) )
```

Essa represesentação é bem mais conveniente para realizar operações, pois ao adicionar ambos, obtém-se:

```
 +5(10)    =     0101(2)
-5(10)    =     1011(2) +
________        _______
 0(10)    = (1) 0000(2)
```

Com uma única representação para zero, mas com um excesso (1) que não é comportado pela quantidade de bits originalmente escolhida. Porém, se desprezado esse excesso, o valor poderá ser considerado correto, com a ressalva de que a quantidade de bits deverá ser rigorosamente observada (ou haverá risco de transbordamento – OVERFLOW)

Para efeitos práticos, o tamanho da representação deverá ser sempre indicado, e as operações deverão ajustar os operandos para a mesma quantidade de bits (de preferência, a maior possível).

Exemplo:

```
 5(10) =  101(2)

+5(10) = 0101(2)

-5(10) = 1010(2)   ( complemento de 1, com 4 bits ou C14 (5) )

-5(10) = 1011(2)   ( complemento de 2, com 4 bits ou C24 (5) )

logo,

C1,5 (+5) = C1 ( 00101(2) )    = 11010(2)

C2,5 (+5) = C2 ( 00101(2) )    = 11011(2)

C1,8 (+5) = C1 ( 00000101(2) ) = 11111010(2)

C2,8 (+5) = C2 ( 00000101(2) ) = 11111011(2)
```

De modo inverso, dado um valor em complemento de 2, se desejado conhecer o equivalente positivo, basta retirar uma unidade e substituir os valores individuais de cada dígito binário.

Exemplo:

```
 1011(2)   ( complemento de 2, com 4 bits )

1011(2) - 1 = 1010(2) e invertendo ( 0101(2) ) = +5(10)

logo, 1011(2) = - 5(10)

Portanto, para diferentes quantidades de bits:

11011(2) = 11010(2) = 00101(2) = 5(10)

11111011(2) = 11111010(2) = 00000101(2) = -5(10)
```

Fim