Curso Básico de Hardware

Sistemas de Numeração

Versão 1/2004

Profs. M.Sc. Lucio M. Duarte e Ph.D. Avelino Zorzo
Faculdade de Informática - PUCRS

REPRESENTAÇÃO DE DADOS E SISTEMAS DE NUMERAÇÃO

Profs. M.Sc. Lucio M. Duarte e Ph.D. Avelino Zorzo[1]
Faculdade de Informática - PUCRS

1 REPRESENTAÇÃO DE DADOS

Acredita-se que a criação de números veio com a necessidade de contar, seja o número de animais, alimentos, ou coisas do gênero. Como a evolução nos legou algumas características, como os cinco dedos em cada mão e cinco dedos em cada pé, seria muito natural que os primeiros sistemas de numeração fizessem uso das bases 10 (*decimal*) e 20 (*vigesimal*). O número 80 em francês, por exemplo, escrito como *quatre-vingt* (ou, quatro vezes o vinte), é remanescente de um sistema vigesimal.

Computadores modernos, por outro lado, usam "chaves elétricas" para representar números e caracteres. Cada chave pode estar ligada ou desligada e a combinação dos estados de um conjunto destas chaves representa algo (número ou caracter). Visto que o "cérebro" de um computador é simplesmente um conjunto de chaves elétricas, onde cada chave possui apenas dois estados possíveis (ligada/desligada), computadores "pensam" usando apenas 2 dígitos: 0 e 1 (0 para desligada e 1 para ligada). Portanto, computadores se utilizam de uma forma de representação de dados para descrever números e caracteres na forma de um conjunto de 0s e 1s.

Linguagens humanas usam palavras que contêm um número variável de caracteres. Computadores não possuem a capacidade de trabalhar com palavras de tamanho variável. Por isso, suas "palavras" (representação de caracteres e números) têm um número predeterminado de caracteres, que, na linguagem binária, são chamados de *bits* (*binary digits*). Os primeiros computadores pessoais que se tornaram populares usavam 8 bits (1 *byte- binary term*) para representar uma "palavra". Assim, o computador sabia onde começava uma palavra e onde ela acabava apenas contando o número de bits. A partir da evolução dos computadores, as "palavras" evoluíram para 16 bits (PC 286), 32 bits (PC 386-Pentium) e 64 bits (maioria dos computadores de hoje). Dessa forma, uma "palavra" do computador passou a não ser mais composta apenas por um byte, mas por 2, 4 e agora 8 bytes. Essa evolução permitiu que cada vez mais coisas pudessem ser representadas através das palavras do computador, aumentando o número de instruções inteligíveis por ele.

2 SISTEMAS DE NUMERAÇÃO

Desde tempos remotos o homem utiliza a escrita para registrar e transmitir informação. A escrita vai do antigo hieróglifo egípcio até o alfabeto latino atual (ou "alfabeto" chinês/japonês). O alfabeto, como conjunto de símbolos, se desenvolveu, originalmente na Grécia e, posteriormente, em Roma e constitui a origem de nosso alfabeto atual.

Uma das primeiras tentativas de registro de quantidades sob a forma escrita foi o sistema de numeração indo-arábico, do qual é derivado o atual sistema de numeração

[1] Material originalmente elaborado pelo Prof. Lúcio, adaptado e complementado pelo Prof. Avelino. Revisado pelo Prof. PhD Luis Fernando Pereira – Faculdade de Engenharia/PUCRS

decimal. Um *sistema de numeração* é formado por um conjunto de símbolos utilizados para representação de quantidades (alfabeto) e as regras que definem a forma de representação.

Quando falamos em sistema decimal, estamos estabelecendo que a nossa base de contagem é o número 10, pois o sistema decimal possui um alfabeto de 10 símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Este conjunto de símbolos do alfabeto define o que é chamado de *base* do sistema de numeração. Assim, se temos 10 símbolos, estamos trabalhando sobre a base 10. Um sistema de numeração é determinado fundamentalmente pela sua base.

2.1 TIPOS DE SISTEMAS DE NUMERAÇÃO

Sistemas de numeração podem ser divididos em 2 grupos: os sistemas não-posicionais e os sistemas posicionais.

2.1.1 SISTEMAS NÃO-POSICIONAIS

São aqueles em que o valor atribuído a um símbolo não se altera, independentemente da posição em que ele se encontre no conjunto de símbolos que está representando um número. Um exemplo de sistema não-posicional é o sistema de numeração romano. Neste sistema temos os símbolos I, V, X, L, C, D e M. Em qualquer posição dentro de um conjunto destes símbolos, eles não alteram seus valores (I → 1, V → 5, X → 10, L → 50, C → 100 e M → 1000).

No sistema de numeração romano antigo por exemplo (onde o número 4 era representado por IIII), a posição do símbolo tinha sempre o mesmo significado, ou seja o número 1.469 era representado como MCCCCLXVIIII. Esta forma era somente por conveniência pois ele também poderia ser representado como CMCCLCIIXVII.

Já no sistema romano "moderno", o que se altera é a sua utilização para a definição da quantidade representada (porém individualmente eles continuam representando a mesma quantidade), a partir das regras definidas pelo sistema:

- Cada símbolo colocado à direita de um maior é adicionado a este.
  Ex.: XI → 10 + 1 = 11;
- Cada símbolo colocado à esquerda de um maior tem o seu valor subtraído deste.
  Ex.: IX → 10 – 1 = 9;

Assim, o número XXI representa 21 em decimal (10 + 10 + 1), enquanto que XIX representa 19 (10 + 10 – 1).

2.1.2 SISTEMAS POSICIONAIS

São aqueles em que o valor atribuído a um símbolo depende da posição em que ele se encontra no conjunto de símbolos que está representando um número. O exemplo típico de sistema posicional é o sistema de numeração decimal. Neste sistema, por exemplo, o símbolo 5 pode representar o valor 5, o valor 50, como em 57 (50 + 7), o valor 500, como em 503 (500 + 3), e assim por diante. Isto é, a regra válida para o sistema decimal é que quanto mais à esquerda do número o símbolo está, mais ele vale. Na verdade, a cada posição mais à esquerda, o símbolo vale 10 vezes mais.

2.2 REPRESENTAÇÃO NUMÉRICA

A representação de quantidade no computador se baseia na representação sistemas numéricos tradicionalmente conhecidos. Estes sistemas numéricos são posicionais, isto é, cada quantidade é representada em uma única forma, mediante uma certa combinação de símbolos, que têm um significado distinto, segundo sua posição.

No sistema decimal, como já comentado, cada posição tem um valor intrínseco que equivale a dez vezes o valor da posição que está imediatamente a sua direita. Supondo que a cada posição designamos uma casa, o valor das casas vai aumentando para a esquerda de 10 em 10 vezes e os dígitos ou símbolos que podemos colocar nelas são: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, os quais possuem um valor intrínseco distinto para cada um.

Se representarmos o número 245 assinalando um símbolo a cada casa, indicando o valor de cada casa, teremos:

Valor da casa	**1000**	**100**	**10**	**1**	**0,1**	**0,01**
Dígitos	0	2	4	5	0	0

O significado de cada dígito em determinada posição é o valor da casa multiplicado pelo valor do dígito e a quantidade representada é a soma de todos os produtos. A partir disto podemos dizer que o valor de um número **X** é

$$\mathbf{X = a_n B^n + a_{n-1} B^{n-1} + ... + a_0 B^0}$$

onde $\mathbf{a_n > 0}$, cada $\mathbf{a_i}$ é um inteiro não negativo e **n** é um valor que representa a posição mais à esquerda do número, ou posição mais significativa do número. Este valor é contado atribuindo-se o valor zero à posição mais à direita (no caso de um valor inteiro) e somando-se 1 até chegar à última posição do número. Esta representação de **X** é única e é chamada de **representação de X na base B**, representada como $(X)_B$. Assim, temos que o número 3547, por exemplo pode ser representado da seguinte forma

$$\mathbf{3.10^3 + 5.\ 10^2 + 4.10^1 + 7.\ 10^0 = 3000 + 500 + 40 + 7 = 3547}$$

2.3 SISTEMA BINÁRIO

No sistema binário, cada número é representado de uma forma única, mediante uma combinação de símbolos 0 e 1, que, em nosso caso, será uma combinação de "estados 1" e "estados 0" dos bits que formam um conjunto ordenado. Designaremos por $\mathbf{b_i}$ cada bit deste conjunto ordenado, no qual o sub-índice **i** corresponde ao número da casa que o bit está ocupando. Seguindo a lógica de que cada posição em número decimal vale 10 vezes mais que a posição imediatamente a sua direita e 10 vezes menos que a posição imediatamente a sua esquerda, no sistema binário cada casa vale 2 vezes mais que aquela que está imediatamente a sua direita e 2 vezes menos que a que está a sua esquerda. Desta forma, teremos que, se o valor da primeira casa da direita for 2^0, a segunda valerá $2^0 \times 2 = 2^1$, e assim consecutivamente para a esquerda. Os valores das casas ficam claros no seguinte esquema:

...	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	...

Se b_0, b_1, b_2, etc., são os bits que se coloca em cada posição, a quantidade representada valerá:

$$\mathbf{\ldots + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + \ ...}$$

Para evitar a representação mediante o somatório, adota-se a convenção de separar mediante vírgulas as casas 2^0 e 2^{-1}, de tal modo que a representação fique:

$$\mathbf{... \ b_4 \ b_3 \ b_2 \ b_1 \ b_0, \ b_{-1} \ b_{-2} \ \ldots}$$

Em que $\mathbf{b_i} = 0$ ou 1.

Exemplo: o número binário 10011,01 representa a quantidade:

$$\mathbf{1.2^4 + 0.2^3 + 0.2^2 + 1.2^1 + 1.2^0 + 0.2^{-1} + 0.2^{-2}}$$

A partir do conhecimento sobre a base, podemos saber quantos números ou caracteres podem ser representados de acordo com o número de bits utilizados. Sabe-se, por exemplo, que com um bit podemos representar dois valores diferentes: 0 e 1. Se tivermos 2 bits, então poderemos representar 4 diferentes valores com as combinações dos valores possíveis de cada bit. Isto é, chamando-se o primeiro bit de $\mathbf{b_1}$ e o segundo de $\mathbf{b_2}$, podemos ter todos valores possíveis da combinação de valores $\mathbf{b_1 b_2}$. Como tanto $\mathbf{b_1}$ quanto $\mathbf{b_2}$ podem assumir o valor 0 ou 1, teremos as seguintes possíveis combinações:

Bit b_1	**Bit b_2**	**Valor b_1 b_2**
0	0	00
0	1	01
1	0	10
1	1	11

Da mesma forma, se tivermos 3 bits, poderemos combinar os valores destes três bits e obteremos 8 diferentes valores: 000, 001, 010, 011, 100, 101, 110 e 111. Disso, podemos concluir que, quando temos 1 bit conseguimos representar 2 valores distintos (2^1 valores); quando temos 2 bits, conseguimos representar 4 valores distintos (2^2 valores); e quando temos 3 bits, podemos representar 8 valores diferentes (2^3 valores). Logo, para um número **n** de bits, poderemos representar $\mathbf{2^n}$ valores distintos. Com isso, com 8 bits poderemos representar $2^8 = 256$ valores distintos.

2.4 SISTEMA HEXADECIMAL

Como já visto, computadores usam o sistema binário para representar seus dados. Mas é difícil e trabalhoso para um programador descrever com conjuntos de 0 e 1 os números a serem postos na memória do computador por um dado programa. Por isso, usa-se uma forma mais compacta de representação em que os bits são agrupados de 4 em 4. Assim, cada grupo de 4 bits é transformado em um único símbolo. Como o maior valor representado por um conjunto de 4 bits é 1111, o qual representa o valor 15 em decimal, ou

seja é possível representar 16 números (0 até 15), este sistema é chamado de hexadecimal ou sistema de base 16.

No sistema hexadecimal, cada casa vale 16 vezes a que está a sua direita, e os símbolos utilizados são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F. O símbolo A equivale a dez, o B equivale a onze e assim consecutivamente até F que equivale a quinze, no sistema decimal.

Exemplo o número hexadecimal A17,B9 representa a quantidade:

$$10.16^2 + 1.16^2 + 7.16^0 + 11.16^{-1} + 9.16^{-2}$$

A representação do conjunto dos símbolos deste sistema mediante grupos de quatro bits, em que cada símbolo se faz corresponder com sua representação binária, é apresentada a seguir:

(0000)……0	(1000)……8
(0001)……1	(1001)……9
(0010)……2	(1010)……A
(0011)……3	(1011)……B
(0100)……4	(1100)……C
(0101)……5	(1101)……D
(0110)……6	(1110)……E
(0111)……7	(1111)……F

Desta forma, o nosso número A17, B9 ficaria em binário:

101000010111,10111001

É possível considerar, do conjunto de símbolos hexadecimais representado em binário, aqueles que correspondem a um sistema decimal codificado em binário. Em tal caso, a expressão binária dos dígitos A, B, C, D, E, F não teria significado.

3 TRANSFORMAÇÕES NUMÉRICAS

Conforme visto nos sistemas de numeração apresentados, cada um deles possui uma certa proximidade com os outros. Isto torna possível que valores representados em um dado sistema possam ser convertidos para outro sistema. Isto é necessário porque nós humanos trabalhamos no sistema decimal e o computador, no sistema binário. Por isso, temos de poder converter valores decimais em valores binários. Da mesma forma, devemos poder entender os dados gerados pelo computador, o que nos leva a converter valores binários em decimais. Como, muitas vezes valores hexadecimais são usados para representar dados a serem armazenados na memória, de forma a facilitar o trabalho do programador, também devemos saber converter valores decimais para hexadecimais e valores hexadecimais para valores binários, que são os valores realmente armazenados na memória. O caminho contrário é igualmente importante.

Converter valores de um sistema de numeração para outro é um processo chamado de *conversão de bases*. Os conceitos vinculados à conversão de base, cálculos aritméticos em diferentes bases e complementos (que veremos mais adiante), são necessários à construção dos computadores. Basicamente estes conceitos são usados para a construção de circuitos

digitais (usados no *hardware*), que, juntamente com as chamadas portas lógicas (estudadas em estruturas algébricas), são combinados formando o coração do processador, a ULA (Unidade Aritmética e Lógica), e todas as outras partes que o formam. Através de uma série de circuitos que fazem somas, subtrações, comparações, entre outros, os bits podem ser interpretados e arranjados da forma que quisermos. Estes conhecimentos são essenciais ainda para o estudo de circuitos digitais, lógica para computação, arquitetura de computadores, entre outras áreas.

3.1 BINÁRIO EM DECIMAL

A conversão de binário em decimal corresponde simplesmente a utilizar a idéia vista anteriormente de associação de valores a cada posição (casa) do número a partir da base 2, de forma que cada posição mais à esquerda vale duas vezes mais que a anterior. O valor de cada posição é multiplicado pelo valor do bit da posição.

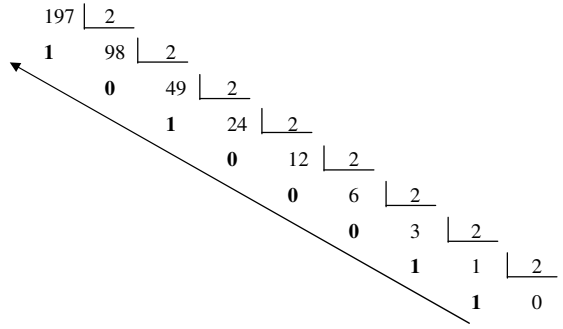
Exemplo: $\mathbf{(11001)_2}$

$1.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0 = 16 + 8 + 0 + 0 + 1 = \mathbf{(25)_{10}}$

3.2 DECIMAL EM BINÁRIO

Mediante divisões inteiras sucessivas por dois, tomando-se os restos das divisões no sentido ascendente.

Exemplo: 197



$\mathbf{(197)_{10} = (11000101)_2}$

3.3 BINÁRIO EM HEXADECIMAL

Divide-se o número em grupos de quatro bits, a partir da direita, substituindo-se tais grupos pelos símbolos hexadecimais correspondentes. Quando o número for fracionário, deve-se começar a divisão em grupos de quatro, a partir da vírgula, em ambas as direções.

Exemplo: $(11110001)_2$ 1111 0001
(F 1)$_{16}$ = $(F1)_{16}$

$(10010011)_2$ 1001 0011
(9 3)$_{16}$ = $(93)_{16}$

Se a divisão em grupos de quatro deixar o grupo extremo com menos de quatro dígitos, completá-lo com zeros.

$(101011)_2$ ⟶ 10 1011 ⟶ 0010 1011
(2 B)$_{16}$

$(101011)_2 = (2B)_{16}$

3.4 HEXADECIMAL EM BINÁRIO

Usa-se o processo inverso ao anterior.

Exemplo: $(A56B)_{16}$

A	5	6	B
1010	0101	0110	1011

$(A56B)_{16} = (1010010101101011)_2$

3.5 HEXADECIMAL EM DECIMAL

Usa-se o mesmo sistema para transformar binário em decimal, com a diferença de se usar a base 16.

Exemplo: $(A6B)_{16}$

A 6 B
$10.16^2 + 6.16^1 + 11.16^0 = 2560 + 96 + 11 = (2667)_{10}$

$(A6B)_{16} = (2667)_{10}$

3.6 DECIMAL EM HEXADECIMAL

Semelhante às transformações de decimal para binário, com divisões sucessivas pelo número 16.

Exemplo: 2736

2736 | 16
0 171 | 16
11 10 | 16
10 **0**

mas, $(10)_{10} = (A)_{16}$
$(11)_{10} = (B)_{16}$

então: $(2736)_{10} = (AB0)_{16}$

3.7 OUTRAS BASES

O processo de conversão de uma base qualquer para a base decimal tem o mesmo formato do processo executado nas seções anteriores para a base binária e para a base hexadecimal (veja Seção 2.2).

Para converter um número da base decimal para uma base qualquer, novamente, utiliza-se o mesmo procedimento executado para converter para a base binária ou hexadecimal, trocando-se o divisor, ou seja se quiser converter para a base 5 usa-se como divisor o número 5 em vez de 2 ou 16.

4 OPERAÇÕES NUMÉRICAS EM BINÁRIO

4.1 ADIÇÃO

Como se sabe, no sistema decimal, quando se quer somar 9 com 1, o resultado é sempre 0 e vai 1, ou seja, é igual a 10. No sistema binário, ocorre o mesmo quando se soma 1 com 1. O resultado é 0 e vai 1, ou seja 10. As regras para a adição binária são as seguintes:

0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	0 e vai 1 (escreve-se 10, mas diz-se “um zero”)

Exemplos:

a) 1010	+ 111
binário	decimal
1010	10
+ 0111	+ 7
10001	17

b) 1010 + 101

binário	decimal
1010	10
+ 0101	+ 5
1111	15

4.2 SUBTRAÇÃO

Regras:

0	-	0	=	0
1	-	1	=	0
1	-	0	=	1
0	-	1	=	1 (com empréstimo de 1).

Exemplos:

1001	11110
- 110	-11011
0011	00011

5 REPRESENTAÇÃO DE NÚMEROS INTEIROS

Até este momento vimos como representar números em diversas bases. Porém os números que representamos até agora foram números Naturais, ou seja positivos. Neste capítulo veremos como representar números Inteiros, ou seja números positivos e negativos.

Os computadores digitais utilizam principalmente quatro métodos para representar números inteiros:

- módulo de sinal, ou dígito de sinal (MS);
- complemento de 1 (C1);
- complemento de 2 (C2);
- excesso de 2 elevado a N-1.

Nessas representações de números utilizam-se do sistema binário e considera-se que temos um número limitado de dígitos para cada dado numérico, ou seja quando estivermos representando um número em binário deveremos especificar quantos dígitos estarão sendo utilizados para representar este número. Esse número de dígitos disponível é representado por N.

5.1 MÓDULO DE SINAL (MS)

Neste sistema de representação o bit que está situado mais a esquerda representa o sinal, e o seu valor será:

- 0 para o sinal +; e
- 1 para o sinal -.

Os bits restantes representam o módulo do número.

Exemplo: Representar 10 e –10
Limitação de 8 bits (N = 8)

10	0	0001010		-10	1	0001010
↓	↓	↓		↓	↓	↓
nº	sinal	módulo		nº	sinal	módulo

Denomina-se **AMPLITUDE ou FAIXA** de representação num determinado método o conjunto de números que podem ser nele representados.

Para o sistema módulo e sinal, a faixa de representação para N dígitos é de:

$$-2^{N-1} + 1 <= X <= +2^{N-1} - 1$$

Para 8 bits a faixa é: - 127 <= X <= +127

Para 16 bits a faixa é: - 32767 <= X <= + 32767

Para 32 bits a faixa é: - 2147483647 <= X <= + 2147483647

Vantagem: possuir faixa simétrica.
Inconveniência: 2 representações para o número 0.
Para 8 bits o 0 tem as seguintes representações:

00000000 (+0)
10000000 (-0)

5.2 COMPLEMENTO DE 1 (C1)

Este sistema de representação também utiliza o bit mais à esquerda para o sinal, correspondendo o 0 ao sinal + e o 1 ao sinal -. Para os números positivos, os N-1 bits da direita representam o módulo. O simétrico de um número positivo é obtido pelo complemento de todos os seus dígitos (trocando 0 por 1 e vice-versa) incluindo o bit de sinal.

Exemplo: Representar 10 e –10
Limitação de 8 bits (N = 8)

10 → nº
0 → sinal
0001010 → módulo

Número -10 é o complemento do seu simétrico

-10 → nº
1 → sinal
1110101 → módulo

Neste caso, a faixa de representação é

$- 2^{N-1} + 1 <= X <= + 2^{N-1} - 1$

Para 8 bits a faixa é: - 127 <= X <= +127

Para 16 bits a faixa é: - 32767 <= X <= + 32767

Para 32 bits a faixa é: - 2147483647 <= X <= + 2147483647

Vantagem: possuir faixa simétrica.e facilitar operações aritméticas

Inconveniência: 2 representações para o número 0.
Para 8 bits o 0 tem as seguintes representações:
00000000 (+0)
11111111 (-0)

5.3 COMPLEMENTO DE 2 (C2)

Este sistema de representação utiliza o bit mais à esquerda para o sinal, correspondendo o 0 ao sinal + e o 1 ao sinal -. Para os números positivos, os N-1 bits da direita representam o módulo, igual ao que ocorre em MS e C1.

O simétrico de um número é obtido em dois passos:

1º passo – obtém-se o complemento de todos os bits do número positivo (trocando 0 por 1 e vice-versa) incluindo o bit de sinal; isto é, executa-se o complemento de 1;

2º passo – ao resultado obtido no primeiro passo com o complemento de 1, soma-se 1 (em binário), desprezando o último transporte, se houver.

Exemplo: Complemento de 2 dos números 10 e –10
Limitação de 8 bits (N = 8)

Número 10:

10 → nº
0 → sinal
0001010 → módulo

Número -10:

1º passo: complemento de 1

-10 → nº
1 → sinal
1110101 → módulo

2º passo:

```
 1110101
+      1
--------
 1110110
```

Neste caso a faixa de representação é

$- 2^{N-1} <= X <= + 2^{N-1} - 1$

Para 8 bits a faixa é: - 128 <= X <= +127

Para 16 bits a faixa é: - 32768 <= X <= + 32767

Para 32 bits a faixa é: - 2147483648 <= X <= + 2147483647

Vantagem: uma única representação para o 0 e facilitar operações aritméticas.
Para 8 bits teremos:

```
Nº  0                 00000000 (+0)
Nº -0   passo 1       11111111 (-0)
        Passo 2              1
                     ---------
                     100000000
                    ↙
             estouro desprezado
```

Logo 0 e –0 têm a mesma representação.

Incoveniência: Assimetria → existem mais valores negativos que positivos.

5.4 EXCESSO DE 2 ELEVADO A N-1

O método de representação em excesso não utiliza nenhum bit para o sinal, de modo que todos os bits representam um módulo ou valor. Esse valor corresponde ao número representado mais um excesso, que para N bits é igual a 2 elevado a N–1.

Exemplo: para 8 bits o excesso é 128 ($2^7 = 128$), logo
o número 10 é representado por 10 + 128 = 138; e
o número –10 é representado por – 10 + 128 = 118.

O número 10 é 10001010

O número –10 é 01110110

Neste método o **número 0 tem uma única representação**, que para 8 bits corresponde a: número 0 (0 + 128) = 10000000

Neste caso a faixa de representação é **assimétrica** (o que é inconveniente) e é dada por: $- 2^{N-1} <= X <= + 2^{N-1} - 1$

Para 8 bits a faixa é: - 128 <= X <= +127

Para 16 bits a faixa é: - 32768 <= X <= + 32767

Para 32 bits a faixa é: - 2147483648 <= X <= + 2147483647

É interessante observar que todo o número representado em excesso é igual a sua correspondente representação em complemento de 2, mas com o 1º dígito da esquerda trocado.

5.5 CONSIDERAÇÕES

Cabe salientar que nesta seção mostramos como representar números positivos e negativos em binário e as formas de representação MS, C1 e C2 tem sempre a mesma representação para números positivos, diferenciando somente na representação de números negativos.

6 ARITMÉTICA COMPLEMENTAR

Uma das grandes vantagens de se representar números em complemento de 1 ou complemento de 2 em relação a digito de sinal é a facilidade de realizar operações de soma

e subtração. Nesta seção mostraremos como realizar soma e subtração[2] de números binários em complemento de 1 e complemento de 2.

6.1 SOMA EM COMPLEMENTO DE 1

Na aritmética de complemento de 1, dois números são somados da mesma forma que na representação binária. Com a diferença que deve ser somado o vai-um ao resultado (se o vai-um for zero então não é necessário realizar esta soma).

Exemplo: Somar os valores 10 e -3 em complemento de 1, para 8 bits

10 em complemento de 1 é 00001010
- 3 em complemento de 1 é 11111100

somando, resulta em 1 00000110

```
  00000110
+        1
----------
  00000111 (7) que é o resultado da operação.
```

6.2 SOMA EM COMPLEMENTO DE 2

Na aritmética em complemento de 2, o processo é idêntico ao de complemento de 1, desprezando-se o vai-um (*carry)*, se houver.

Exemplo: somar os valores 10 e -3 em complemento de 2, para 8 bits

```
10 em complemento de 2 é       00001010   (10)
- 3 em complemento de 2 é      11111101   (-3)
                             ------------
somando                      1 00000111   (7)
                            /
                       carry
```

Observe que houve *carry*. Este *carry* deve ser desprezado.

[2] Subtração é realizada na realidade pela soma de um número positivo com um número negativo, ou seja 5 – 3 é na verdade realizado como 5 + (-3).