ESCOLA POLITÉCNICA
UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Sistemas Eletrônicos
PSI - EPUSP

REPRESENTAÇÃO DE NÚMEROS EM BINÁRIO E HEXADECIMAL

1. Hexadecimal [A1]

Hexadecimal é o sistema numérico de base 16, denotado utilizando os símbolos 0–9 e A–F (ou a–f). Por exemplo, o número decimal 79 (cuja representação binária é 01001111) pode ser escrito como 4F em hexadecimal (4 = 0100, F = 1111). Cada dígito hexadecimal consegue representar 4 bits. Assim, um byte pode ser representado utilizando 2 dígitos hexadecimais, um WORD (uma variável de 16 bits) pode ser representado utilizando 4 dígitos hexadecimais, e assim por diante.

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tabela 1: Dígitos hexadecimais

Alguns números hexadecimais são indistinguíveis de números decimais. Assim, algumas convenções devem ser usadas para diferenciá-los. Em textos impressos, frequentemente são utilizados sufixos tais como $5A3_{16}$ ou $5A3_{HEX}$.

Em linguagens computacionais, existe uma variedade de formas muito grande para indicá-los. Alguns exemplos:

1. Em C e linguagens com sintaxes semelhantes (tais como C++, C# e Java) prefixam-se os hexadecimais com “0x”, por exemplo “0x5A3”.
2. Alguns *assemblers* indicam hexadecimal com o sufixo “h” (se o primeiro dígito for uma letra, então também se acrescenta um prefixo “0”), por exemplo, “0A3Ch”, “5A3h”.
3. Em *Pascal*, alguns outros *assemblers*, e algumas versões de BASIC utilizam o prefixo “$”, por exemplo, “$5A3”.

2. Representação de Números Inteiros com Sinal [A2]

Matematicamente, os números negativos são representados prefixando-se o número positivo correspondente com o sinal “-”. No computador, existem várias formas de representar números negativos. Nesta seção, veremos quatro métodos para estender o sistema de números binários sem sinal para o sistema com sinal: sinal-e-magnitude, complemento de um, complemento de dois, e excesso de *N*. A seção 3 explica com mais detalhes o sistema complemento de dois, pois os computadores modernos (assim como a placa Lab-PC+) utilizam esse sistema. As outras representações são usadas em circunstâncias especiais e em outros tipos de conversores

analógico-digitais. A tabela 2 mostra os padrões binários com sinal de 4 bits em diferentes representações.

decimal	sem sinal	sinal-e-magn	compl. 1	compl. 2	excesso de 8
+8	1000	-	-	-	-
+7	0111	0111	0111	0111	1111
+6	0110	0110	0110	0110	1110
+5	0101	0101	0101	0101	1101
+4	0100	0100	0100	0100	1100
+3	0011	0011	0011	0011	1011
+2	0010	0010	0010	0010	1010
+1	0001	0001	0001	0001	1001
0	0000	0000	0000	0000	1000
-0	-	1000	1111	-	-
-1	-	1001	1110	1111	0111
-2	-	1010	1101	1110	0110
-3	-	1011	1100	1101	0101
-4	-	1100	1011	1100	0100
-5	-	1101	1010	1011	0011
-6	-	1110	1001	1010	0010
-7	-	1111	1000	1001	0001
-8	-	-	-	1000	0000

Tabela 2: Inteiro de 4 bits em diferentes sistemas de representações.

2.1 Sinal-e-magnitude (*sign-and-magnitude*)

Este método aloca um bit (frequentemente o bit mais significativo) para representar o sinal: atribui 0 a esse bit para representar um número positivo e 1 para representar um número negativo. Os bits restantes indicam a magnitude (ou o valor absoluto) do número. Numa variável com 8 bits, 7 bits são usados para representar a magnitude, indo de 0000000 (0) a 1111111 (127). Assim, os números de -127 a +127 podem ser representados num byte. Neste sistema, existem duas formas para representar o zero, 00000000 (+0) e 10000000 (-0). Alguns computadores binários antigos (por ex., IBM 7090) usavam esta representação. Muitos voltímetros digitais possuem internamente conversores analógico-digitais que utilizam esta representação.

2.2 Complemento de um (one´s complement)

Neste sistema, um número negativo é obtido aplicando a operação NOT bit-a-bit ao número positivo correspondente. Por exemplo, o complemento de um de 00101011 (43) é 11010100 (-43).

Assim como na representação sinal-e-magnitude, o complemento de um possui duas representações diferentes para zero: 00000000 (+0) e 11111111 (–0). Para somar dois números neste sistema, efetua-se primeiro a soma binária convencional (sem sinal), e depois soma-se o vai-um (*carry*). Por exemplo, efetuando a soma binária entre -1 (11111110) e +2 (00000010), obtém-se zero (00000000). Somando o vai-um, obtém-se a resposta correta (00000001). Este sistema era comum em computadores antigos como PDP-1 e a série UNIVAC 1100/2200.

2.3 Complemento de dois (*two´s complement*)

No sistema complemento de dois, as representações múltiplas de zero e a necessidade de tratar especialmente o vai-um são eliminadas. Neste sistema, um número negativo é obtido complementando bit-a-bit o número positivo correspondente, e somando um (sem sinal). Por exemplo, os valores de 8 bits em complemento de 2 estão indicados na tabela 3. A soma de dois inteiros com sinal em complemento de dois é exatamente a mesma soma de dois inteiros sem sinal (exceto pela detecção de *overflow*). Por exemplo, a soma em complemento de dois de 127 e -128 dá o mesmo padrão de bits que a soma sem sinal entre 127 e 128, como pode ser visto na tabela 3. A placa Lab-PC+ no modo bipolar utiliza esta representação. Veja a seção 3 para maiores detalhes.

valor binário	complemento de 2	sem sinal
10000000	-128	128
10000001	-127	129
...	...	...
11111110	-2	254
11111111	-1	255
00000000	0	0

Tabela 3.

2.4 Excesso de *N* (*excess-N* ou *biased representation*)

Excesso de *N* ou a representação deslocada utiliza um número *N* como um valor de deslocamento pré-definido. Um valor *x* é representado pelo número sem sinal *x+N*. Assim, 0 é representado por *N*, e *-N* é representado pelo padrão com todos os bits zeros. Por exemplo,a representação em excesso de 8 (4 bits) está na tabela 2. Esta é uma representação que é atualmente utilizada principalmente em números em ponto flutuante. O padrão IEEE de ponto flutuante define o campo expoente de uma variável de precisão simples (32 bits) como um campo de 8-bit em excesso de 127. O campo expoente de uma variável de precisão dupla (64 bits) é um campo de 11 bits em excesso de 1023. No caso especial em que o deslocamento *N* usado é igual a 2^{n-1} (*n* é o número de bits da variável), a representação em excesso de *N* pode ser convertida para a representação complemento de 2 simplesmente invertendo o bit mais significativo. Observe que as duas últimas colunas da tabela 2 diferem apenas pelos bits mais significativos. Um conversor analógico-digital "bipolar por deslocamento" (*offset bipolar*) usa naturalmente esta representação [A4], e a sua saída pode ser convertida para complemento de 2 invertendo-se o bit mais significativo (é o caso da Lab-PC+ no modo bipolar).

3. Complemento de Dois [A3]

Complemento de dois é o método mais popular para representar os números inteiros com sinal no computador. É também uma operação de negação (converter número positivo para negativo ou vice-versa) em computadores que usam o complemento de dois. O seu uso é muito comum hoje, pois não requer que os circuitos de soma e subtração examinem os sinais dos operandos para determinar se devem efetuar soma ou subtração. Além disso, existe nesse sistema uma única representação de zero, eliminando as sutilezas associadas ao zero negativo.

Nesta representação, o bit mais à esquerda indica o sinal. Se esse bit for 0, o número é interpretado como não-negativo. Se for 1, os bits contêm um número negativo na forma de complemento de dois. Para obter o valor absoluto de um número negativo, todos os bits são invertidos e então 1 é somado ao resultado (sem sinal). Um número de 8 bits em complemento de

2 pode representar qualquer inteiro no intervalo de -128 a +127 (-2^7 a $+2^7-1$). Um número de 16 bits pode representar números de -32768 a +32767 (-2^{15} a $+2^{15}-1$).

3.1 Cálculo do complemento de dois

Para achar o complemento de dois de um número binário, os bits são primeiro invertidos (pela operação NOT bit-a-bit); o valor 1 é então somado ao número resultante.

Por exemplo, o valor +5 decimal tem representação binária:

```
0000 0101 (5)
```

O primeiro bit é zero, assim o numeral representado é positivo. Para convertê-lo em -5 em complemento de dois, primeiro os bits são invertidos (0 torna-se 1 e 1 torna-se 0):

```
1111 1010
```

O numeral obtido é o complemento de um do valor decimal 5. Para obter o complemento de dois, 1 é somado ao resultado (sem sinal):

```
1111 1011 (-5)
```

O resultado representa o valor decimal -5 em complemento de dois. O bit inicial é 1, portanto o numeral é negativo.

O complemento de dois de um número negativo é o número positivo correspondente. Por exemplo, invertendo os bits de -5 (acima) dá:

```
0000 0100
```

Somando um dá o valor final:

```
0000 0101 (5)
```

Note que o complemento de dois do zero é o próprio zero. Além disso, o complemento de dois do número mais negativo possível é ele mesmo (por exemplo, usando 8 bits, o complemento de dois de -128 é ele mesmo, pois não é possível representar +128 usando 8 bits).

3.2 Soma

Somar dois números em complemento de dois não requer processamento especial dos sinais dos operandos, e o sinal do resultado é determinado automaticamente. Por exemplo, somando 15 com -5, usando variáveis de 8 bits:

```
 11111 111    (vai-um)
  0000 1111   (15)
+ 1111 1011   (-5)
```

```
==========

  0000 1010   (10)
```

Resulta na soma 10, aritmeticamente correta. Um vai-um é gerado no bit mais à esquerda mas é ignorado.

Os dois bits mais à esquerda da linha vai-um contém uma informação vital: se a soma gerou um *overflow*, isto é, se o número resultante é excessivamente grande para ser representada pelo número de bits adotado. Um *overflow* acontece quando o vai-um é gerado ou para dentro mas não para fora do bit de sinal mais à esquerda ou para fora mas não para dentro desse bit. Em outras palavras, se os últimos dois bits vai-um forem ambos 1s ou 0s, o resultado é válido; se os últimos bits vai-um forem "1 0" ou "0 1", ocorreu um *overflow*. Uma operação XOR (ou-exclusiva) desses bits pode detectar o *overflow*. Por exemplo, considere a soma 4-bits entre 7 e 3:

```
 0111    (vai-um)
  0111   (7)
+ 0011   (3)
=============
  1010   (-6)  inválido!
```

Neste caso, os dois bits mais significativos de vai-um são "0 1", que significa que ocorreu um *overflow* em complemento de dois. O artigo [A3] traz as descrições das outras operações (subtração e multiplicação) com inteiros em complemento de dois.

3.3 Extensão de sinal

Ao converter um número em complemento de dois com um certo número de bits num número com mais bits (por ex., ao copiar uma variável de 1 byte para uma de 2 bytes), o bit de sinal deve ser repetido em todos os bits extras. Por exemplo:

Decimal	Complemento de 2 com 4 bits	Complemento de 2 com 8 bits
5	0101	0000 0101
-3	1101	1111 1101

A placa Lab-PC+ no modo bipolar utiliza a extensão de sinal para converter a saída do conversor A/D (12 bits) para a entrada do *buffer* FIFO (16 bits).

Referências

[A1] *Hexadecimal*, http://en.wikipedia.org/wiki/Hexadecimal, acessado em 16/02/2006.

[A2] *Signed Numer Representations*,

http://en.wikipedia.org/wiki/Signed_number_representations, acessado em 10/01/2006.

[A3] *Two's Complement*,

http://en.wikipedia.org/wiki/Two's_complement, acessado em 10/01/2006.

[A4] Walter Kester, " The Importance of Data Converter Static Specifications - Don't Lose Sight of the Basics,"

http://www.analog.com/en/content/0,2886,760%255F788%255F91286,00.html, acessado em 27/01/2006.