Introdução aos números inteiros

Nesta seção examinaremos a representação de *números inteiros* no computador.

1. Sequência de *bits* (e *bytes*) para representar um inteiro

Toda informação no computador digital é composta por *bits*, em particular todo símbolo ou **caractere** é representado por um número fixo de *bits*, sendo que geralmente usa-se uma quantidade fixa de *bits* para representar qualquer caractere.

Por outro lado, podemos interpretar qualquer sequência de *bits* por seu correspondente em decimal usando a notação de valor posicional. Assim, considerando $b_7b_6b_5b_4b_3b_2b_1b_0$, para cada b_i sendo `0` ou `1`, esse será um valor numérico binário com 8 *bits*, cujo correspondente em decimal pode ser calculado via o polinômio $b_7*2^7+b_6*2^6+b_5*2^5+b_4*2^4+b_3*2^3+b_2*2^2+b_1*2^1+b_0*2^0$.

Por exemplo, o binário `00000111` corresponde ao decimal $0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+1*2^1+1*2^0 = 4+2+1 = 7$ (última linha da tabela 1).

Tab. 1. Exemplo de números binários e seu correspondente decimal

Número binário	decimal correspondente
00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
00000101	5
00000110	6
00000111	7

Um dos primeiros padrões para codificar caracteres foi o padrão American Standard Code for Information Interchange (ASCII) que utilizava apenas 8 *bits*, que convencionou-se a chamar de byte. Isso é discutido no texto introdutório sobre caracteres. Por exemplo, o número binário `01000001`, em *ASCII* corresponde ao caractere `A` ('a' maiúsculo). Mas o binário `01000001` é correspondente ao número decimal $2^6+2^0 = 64+1 = 65$, assim o caractere `A` está associado ao número `65` (vide explicação sobre código ASCII).

2. Quantos inteiros conseguimos com 8 *bits*

Assim, o número de *bits* utilizados para representar os inteiros define o intervalo de inteiro que podemos conseguir. Por exemplo, se dispomos de 8 *bits*, que é denominado um **bytes**, podemos conseguir até 2^8 diferentes valores, pois podemos usar desde `00000000` até o binário `11111111`.

Portanto, se o computador tivesse apenas 8 *bits* para representar inteiros positivos, poderíamos ter desde o `00000000` (correspondendo ao decimal *0*) até o binário `11111111`, que correspondente ao decimal $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 1+2+4+8+16+32+64+128 = 256-1 = 255$.

Mas se precisarmos dos inteiros negativos, poderíamos usar o primeiro *bit* para o sinal (*1* indicando negativo) ou fazer o complemento binário ("invertendo" os *bits*). No primeiro caso, com os mesmos *8 bits*, poderíamos ir desde `-127` ($-2^6-2^5-\ldots-2^1-2^0$) até `127`, mas perderíamos uma entrada (tanto `10000000`, quanto `00000000` poderiam ser associadas ao *0*). Essa associação, que não é utilizada na prática, está ilustrada na terceira coluna da tabela 2.

A outra opção, denominada **complemento de dois**, podemos variar de `-128` até `127`, como indicado na segunda coluna da tabela 2. Nessa representação para obter o negativo de um número deve-se aplicar dois passos:

1. Inverter os *bits* do número e depois somar um.
   Exemplo 1. Para o *127* cujo binário é `01111111`: inverte `10000000` e soma-se um, resultando `10000001` (*-127*).
   Exemplo 2. Para o *126* cujo binário é `01111110`: inverte `10000001` e soma-se um, resultando `10000010` (*-126*).
   Exemplo 3. Para o *1* cujo binário é `00000001`: inverte `11111110` e soma-se um, resultando `11111111` (*-1*).

Assim, nessa representação existe um negativo a mais, no caso o *-128*, que é `10000000`, pois ao somar `10000000` com `01111111` (que é o decimal *127*) obtém-se `11111111` que é o decimal *-1* (exemplo 3), ou seja, *-128 + 127 = -1*.

```
  01111111
         +
  10000000
  --------
  11111111
```

Na tabela 2 apresentamos os binários entre *00000000* e *11111111*, mostrando seu correspondente decimal usando a técnica de *complemento de dois* (coluna do meio) e a conversão usual para decimal (coluna da direita). Lembrando citação anterior, para converter um valor em *base binária* para *base decimal* basta considerar o valor posicional do *bit* (*digito*) multiplicado por sua potência. Assim, o primeiro binário *00000000* é *0*, pois é 0×2^k *para todo natural k, por outro lado o binário 00010011 é 19, pois é* $1 \times 2^4 + 1 \times 2^1 + 1 \times 2^0 = 16 + 2 + 1 = 19$.

Tab. 2. Exemplo de números binários e seu correspondente decimal usando complemento de dois e decimal usual

Número binário	decimal com sinal (complemento)	decimal sem sinal
00000000	0	0
00000001	1	1
00000010	2	2

00000011	3	3
00000100	4	4
00000101	5	5
...	...	...
01111110	126	126
01111111	127	127
10000000	-128	128
10000001	-127	129
10000010	-126	130
10000011	-125	131
...	...	...
11111101	-3	253
11111110	-2	254
11111111	-1	255

3. Inteiros com dois bytes (ou 16 bits)

*Como citado anteriormente, um computador que tivesse apenas 8 bits (que equivale a um **byte**) permitiria a existência de apenas 256 caracteres distintos (incluindo digítos, letras e caracteres especiais). Em lingua inglesa isso poderia ser suficiente, mas em Português (e outras), devido aos acentos, 256 possibilidades não é o suficiente. Assim, se for usado o dobro de bits, passando a usar 16 bits (ou dois bytes), o número de símbolos possíveis crescem muito: com 16 bits é possível representar 2^{16}=65536 símbolos distintos, que restrito aos valores naturais possibilita desde o `0` até o `65535`. Considerando inteiros negativos, usando a representação com complemento de dois, conseguimos representar desde o -32768 até o 32767.*

*Uma vez que o ASCII mostrou-se muito restritivo, surgiram vários outros padrões, como o **ISO-8859-1** e o padrão atual dominante **UTF-8** que permite codificar símbolos em qualquer das linguas correntes. Entretanto, tanto o ISO-8859-1 quanto o UTF-8 tem como códigos iniciais os mesmos 128 primeiros do ASCII (portanto, em ambos a letra 'A' continua com o código 65).*

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações