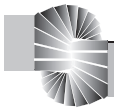
Passando da primeira para a segunda equação, é preciso que os $n - 1$ bits menos significativos não mudem entre as duas representações. Depois, chegamos à penúltima equação, que só é verdadeira se todos os bits nas posições $n - 1$ a $m - 2$ forem 1. Portanto, a regra de extensão de sinal funciona. O leitor poderá achar a regra mais fácil de entender depois de estudar a discussão sobre a negação em complemento de dois, no início da Seção 9.3.

Representação em ponto fixo

Finalmente, mencionamos que as representações discutidas nesta seção às vezes são chamadas de ponto fixo. Isso porque a vírgula (binária) é fixa na posição à direita do bit menos significativo. O programador pode usar a mesma representação para frações binárias, escalando os números de modo que a vírgula binária seja implicitamente posicionado em algum outro local.



9.3 Aritmética com inteiros

Esta seção examina as funções aritméticas comuns sobre número na representação de complemento de dois.

Negação

Na representação sinal-magnitude, a regra para formar a negação de um inteiro é simples: inverta o bit de sinal. Na notação de complemento de dois, a negação de um inteiro pode ser formada com as seguintes regras:

1. Apanhe o complemento booleano de cada bit do inteiro (incluindo o bit de sinal). Ou seja, defina cada 1 como 0, e cada 0 como 1.
2. Tratando o resultado como um inteiro binário sem signal, some 1.

Esse processo em duas etapas é conhecido como a ***operação de complemento de dois***, ou achar o complemento de dois de um inteiro.

$$\begin{aligned} +18 &= 00010010\,(\text{complemento de dois}) \\ \text{complemento bit a bit} &= 11101101 \\ &+ \quad 1 \\ & 11101110 = -18 \end{aligned}$$

Conforme esperado, o negativo do negativo desse número é ele mesmo:

$$\begin{aligned} -18 &= 11101110\,(\text{complemento de dois}) \\ \text{complemento bit a bit} &= 00010001 \\ &+ \quad 1 \\ & 00010010 = +18 \end{aligned}$$

Podemos demonstrar a validade da operação recém-descrita usando a definição da representação de complemento de dois na Equação 9.2. Novamente, interprete uma sequência de n bits de dígitos binários $a_{n-1}a_{n-2} \ldots a_1a_0$ com um inteiro complemento de dois A, de modo que seu valor é

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Agora, forme o complemento booleano bit a bit, $\overline{a}_{n-1}\overline{a}_{n-2} \ldots \overline{a}_0$, e, tratando isso como um inteiro sem sinal, some 1. Finalmente, interprete a sequência resultante de n bits de dígitos binários como um inteiro de complemento de dois B, de modo que seu valor é

$$B = -2^{n-1}\overline{a_{n-1}} + 1 + \sum_{i=0}^{n-2} 2^i \overline{a_i}$$

Agora, queremos $A = -B$, o que significa que $A + B = 0$. Isso é facilmente demonstrado como verdadeiro:

$$\begin{aligned} A + B &= -(a_{n-1} + \overline{a_{n-1}})2^{n-1} + 1 + \left(\sum_{i=0}^{n-2} 2^i(a_i + \overline{a_i})\right) \\ &= -2^{n-1} + 1 + \left(\sum_{i=0}^{n-2} 2^i\right) \\ &= -2^{n-1} + 1 + (2^{n-1} - 1) \\ &= -2^{n-1} + 2^{n-1} = 0 \end{aligned}$$

A derivação anterior considera que primeiro podemos tratar o complemento booleano bit a bit de A como um inteiro sem sinal para a finalidade de somar 1 e depois tratar o resultado como um inteiro em complemento de dois. Existem dois casos especiais a considerar. Primeiro, considere $A = 0$. Nesse caso, para uma representação de 8 bits:

$$\begin{array}{rcl} 0 & = & 00000000 \quad (\text{complemento a dois}) \\ \text{complemento bit a bit} & = & 11111111 \\ & & \underline{+\qquad 1} \\ & & 100000000 = 0 \end{array}$$

Existe um *carry* (vai um) a partir da posição do bit mais significativo, que é ignorado. O resultado é que a negação de 0 é 0, como deveria ser.

O segundo caso especial é um problema maior. Se apanharmos a negação do padrão de bits de 1 seguido por $n - 1$ zeros, voltamos ao mesmo número. Por exemplo, para palavras de 8 bits,

$$\begin{array}{rcl} -128 & = & 10000000 \quad (\text{complemento a dois}) \\ \text{complemento bit a bit} & = & 01111111 \\ & & \underline{+\qquad 1} \\ & & 10000000 = -128 \end{array}$$

Alguma anomalia desse tipo é inevitável. O número de padrões de bits diferentes em uma palavra de n bits é 2^n, que é um número par. Queremos representar inteiros positivos e negativos e 0. Se um número igual de inteiros positivos e negativos forem representados (sinal-magnitude), então existem duas representações para 0. Se houver apenas uma representação de 0 (complemento a dois), então é preciso haver uma quantidade desigual para representar números negativos e positivos. No caso do complemento de dois, para um tamanho de n bits, existe uma representação para -2^{n-1}, mas não para $+2^{n-1}$.

Adição e subtração

A adição em complemento de dois é ilustrada na Figura 9.3. A adição prossegue como se os dois números fossem inteiros sem sinal. Os quatro primeiros exemplos ilustram operações bem sucedidas. Se o resultado da operação for positivo, obtemos um número positivo na forma de complemento de dois, que é a mesma que na forma de inteiro sem sinal. Se o resultado da operação for negativo, obtemos um número negativo na forma de complemento a dois. Observe que, em alguns casos, existe um bit de *carry* além do final da palavra (indicado pelo sombreado), que é ignorado.

Em qualquer adição, o resultado pode ser maior do que pode ser mantido no tamanho da palavra sendo usado. Essa condição é chamada de ***overflow*** (estouro). Quando ocorre *overflow*, a ALU precisa sinalizar esse fato de modo que não haja qualquer tentativa de usar o resultado. Para detectar o *overflow*, a seguinte regra é observada:

Figura 9.3 Adição de números na representação de complemento de dois

1001 = −7 +0101 = 5 1110 = −2 **(a)** (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 **(b)** (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 **(c)** (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 **(d)** (−4) + (−1)
0101 = 5 +0100 = 4 1001 = *Overflow* **(e)** (+5) + (+4)	1001 = −7 +1010 = −6 10011 = *Overflow* **(f)** (−7) + (−6)

Regra do *overflow*: se dois números são somados e ambos são positivos ou ambos negativos, então o *overflow* ocorre se, e somente se, o resultado tiver o sinal oposto.

As Figuras 9.3e e f mostram exemplos de *overflow*. Observe que o *overflow* pode ocorrer havendo ou não um *carry*. A subtração é facilmente tratada com a seguinte regra:

Regra da subtração: para subtrair um número (subtraendo) de outro (minuendo), apanhe o complemento de dois (negação) do subtraendo e some-o ao minuendo.

Assim, a subtração é obtida usando a adição, conforme ilustrado na Figura 9.4. Os dois últimos exemplos demonstram que a regra do *overflow* ainda se aplica.

Figura 9.4 Subtração de números na representação de complemento de dois (M – S)

0010 = 2 +1001 = -7 1011 = -5 **(a)** M = 2 = 0010 S = 7 = 0111 -S = 1001	0101 = 5 +1110 = -2 10011 = 3 **(b)** M = 5 = 0101 S = 2 = 0010 -S = 1110
1011 = -5 +1110 = -2 11001 = -7 **(c)** M = -5 = 1011 S = 2 = 0010 -S = 1110	0101 = 5 +0010 = 2 0111 = 7 **(d)** M = 5 = 0101 S = -2 = 1110 -S = 0010
0111 = 7 +0111 = 7 1110 = *Overflow* **(e)** M = 7 = 0111 S = -7 = 1001 -S = 0111	1010 = -6 +1100 = -4 10110 = *Overflow* **(f)** M = -6 = 1010 S = 4 = 0100 -S = 1100

Uma ideia melhor da adição e subtração em complemento a dois pode ser obtida examinando uma representação geométrica (Benham, 1992[b]), como mostra a Figura 9.5. O círculo na metade superior de cada uma das partes da figura é formado selecionando o segmento apropriado da linha de número e unindo as extremidades. Observe que, quando os números são dispostos em um círculo, o complemento a dois de qualquer número é horizontalmente o oposto desse número (indicado por linhas horizontais tracejadas). Começando em qualquer número no círculo, podemos somar *k* positivo (ou subtrair *k* negativo) a esse número movendo *k* posições em sentido horário, e podemos subtrair *k* positivo (ou somar *k* negativo) desse número movendo *k* posições em sentido anti-horário. Se uma operação aritmética ultrapassar do ponto onde as extremidades são unidas, a resposta estará incorreta (*overflow*).

Todos os exemplos das Figuras 9.3 e 9.4 são facilmente representados no círculo da Figura 9.5.

A Figura 9.6 sugere os caminhos de dados e elementos de hardware necessários para realizar a adição e a subtração. O elemento central é um somador binário, que recebe dois números para adição e produz uma soma e uma indicação de *overflow*. O somador binário trata os dois números como inteiros sem sinal. (Uma implementação de um circuito lógico pelo somador é dada no Capítulo 20.) Para a adição, os dois números são apresentados ao somador a partir de dois registradores, neste caso como registradores *A* e *B*. O resultado pode ser armazenado em um desses registradores ou em um terceiro. A indicação de *overflow* é armazenada em um flag de *overflow* de 1 bit (0 = sem *overflow*; 1 = *overflow*). Para a subtração, o subtraendo (registrador B) é passado por um circuito que calcula o complementador de dois, de modo que seu complemento de dois é passado ao somador. Observe que a Figura 9.6 só mostra os caminhos de dados. Sinais de controle são necessários para controlar se o complementador é usado ou não, dependendo se a operação é de adição ou subtração.

Multiplicação

Em comparação com a adição e a subtração, a multiplicação é uma operação complexa, seja ela realizada no hardware ou pelo software. Diversos algoritmos foram usados em diversos computadores. A finalidade desta

Figura 9.5 Representação geométrica dos inteiros de complemento de dois