- mudança de sinal, ou seja, $z = -x$;
- subtração; e
- detecção de condições de resultado: sinal, valor zero e estouro (*overflow*).

Finalmente, apresentaremos as unidades de lógica e aritmética e uma implementação básica da operação de multiplicação.

10.1 MÓDULOS SOMADORES PARA NÚMEROS INTEIROS POSITIVOS

Um **somador binário de *n* bits** (veja a Figura 10.1) é um sistema combinacional que tem duas entradas de n bits $\underline{x} = (x_{n-1}, \ldots, x_0)$ e $\underline{y} = (y_{n-1}, \ldots, y_0)$ que representam os operandos x e y, respectivamente, e uma saída de n bits $\underline{z} = (z_{n-1}, \ldots, z_0)$ que representa o resultado z (a soma). O módulo executa a adição binária dos operandos de entrada. Sinais de entrada e saída adicionais, chamados vem-1 (***carry-in***) c_{in} e vai-1 (***carry-out***) c_{out}, respectivamente, são usados para facilitar a implementação de somadores maiores. O módulo adiciona o valor de vem-1 aos vetores de entrada e faz o sinal de vai-1 ser 1 sempre que o resultado ultrapasse o valor máximo representável com n bits (isto é, um resultado maior do que $2^n - 1$). A expressão aritmética que relaciona entradas e saídas é

$$x + y + c_{in} = 2^n c_{out} + z$$

Uma especificação de alto nível dos somadores binários de n bits é

Entradas: $\underline{x} = (x_{n-1}, \ldots, x_0)$, $x_j \in \{0, 1\}$
$\underline{y} = (y_{n-1}, \ldots, y_0)$, $y_j \in \{0, 1\}$
$c_{in} \in \{0, 1\}$

Saídas: $\underline{z} = (z_{n-1}, \ldots, z_0)$, $z_j \in \{0, 1\}$
$c_{out} \in \{0, 1\}$

Funções: $z = (x + y + c_{in}) \bmod 2^n$

$$c_{out} = \begin{cases} 1 & \textbf{se} \quad (x + y + c_{in}) \geq 2^n \\ 0 & \textbf{caso contrário} \end{cases}$$

Por exemplo, para $n = 5$,

x	y	c_{in}	z	c_{out}
12	14	1	(12 + 14 + 1) mod 32 = 27	0 porque (12 + 14 + 1) < 32
19	14	1	(19 + 14 + 1) mod 32 = 2	1 porque (19 + 14 + 1) > 32

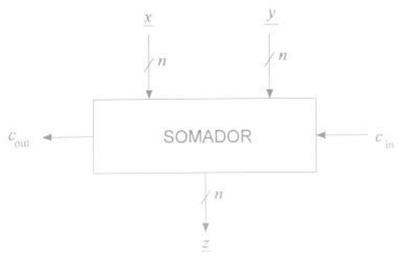

Figura 10.1 Módulo somador.

Implementação do somador de transporte propagado

Módulos somadores podem ser implementados de diversas maneiras, as quais diferem em termos de estrutura, custo e velocidade (retardo). Como acontece com qualquer sistema combinacional, uma implementação potencial de um somador de n bits é uma rede de portas de dois níveis; entretanto, mesmo para um número moderado de bits, essa implementação exige um número grande de portas com um número grande de entradas. Conseqüentemente, outras implementações são necessárias. Vamos estudar primeiro uma que exige o menor número de portas, a qual é conhecida como somador de transporte propagado.

Um **módulo somador de transporte propagado** é uma rede multinível formada pela conexão de somadores de 1 bit, chamados **somadores totais** (*full-adders*) — ST, como ilustra a Figura 10.2. Esta implementação corresponde a uma rede iterativa de módulos somadores totais. Os módulos ST executam a adição de um par de bits de entrada e uma entrada vem-1, produzindo um bit de soma e uma saída vai-1. Ou seja, as entradas para o somador total i são os bits x_i e y_i dos vetores de entrada correspondentes e uma **entrada vem-1** c_i que vem do somador total $i - 1$. Como saída, o somador total i produz o bit de soma z_i e o bit **vai-1** c_{i+1} (o qual é conectado como entrada vem-1 ao somador total $i + 1$).

Uma especificação de alto nível de um somador total é

Entradas: $x_i, y_i, c_i \in \{0, 1\}$
Saídas: $z_i, c_{i+1} \in \{0, 1\}$

Função: $z_i = (x_i + y_i + c_i) \bmod 2$

$$c_{i+1} = \begin{cases} 1 & \textbf{se} \quad (x_i + y_i + c_i) \geq 2 \\ 0 & \textbf{caso contrário} \end{cases}$$

Retardo do somador de transporte propagado

Tratemos do retardo do somador de transporte propagado (a análise funcional deste módulo é deixado para um exercício). O caminho crítico se inicia em c_{in}, atravessa todos os módulos e se encerra ou em z_{n-1} ou c_{out}. Conseqüentemente, o retardo de propagação de pior caso é aproximadamente*

$$t_p(\text{rede}) = (n - 1)t_c + \max(t_z, t_c)$$

em que

$$t_c = \text{Retardo}(c_i \rightarrow c_{i+1})$$
$$t_z = \text{Retardo}(c_i \rightarrow z_i)$$

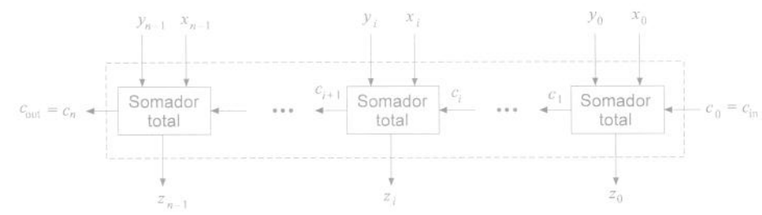

Figura 10.2 Módulo somador de transporte propagado.

Como é patente a partir da expressão, o retardo de pior caso é proporcional a n. Conseqüentemente, o retardo do módulo somador de transporte propagado é elevado para um n grande. Note que o componente mais crítico do retardo de somador total é t_c.

* Uma expressão mais precisa é apresentada depois de mostrarmos a implementação de um somador total.

Implementação do somador total

Consideremos, agora, as implementações do somador total. Da descrição de alto nível, obtemos a seguinte tabela:

x_i	y_i	c_i	c_{i+1}	z_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Uma implementação baseada em multiplexador que usa esta tabela foi apresentada no Capítulo 9. As expressões das somas mínimas de produtos para estas funções são

$$z_i = x_i y_i' c_i' + x_i' y_i c_i' + x_i' y_i' c_i + x_i y_i c_i$$
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

Estas expressões são a base para a implementação de dois níveis mostrada na Figura 10.3*a*.

Uma implementação alternativa baseia-se no seguinte raciocínio:

- A adição mod 2 indica que o bit de soma tem valor 1 sempre que o número de 1's nas entradas (incluindo o vem-1) seja ímpar, ou seja,

$$z_i = x_i \oplus y_i \oplus c_i$$

- O vai-1 é 1 quando $x_i + y_i = 2$ (isto é, $x_i y_i = 1$) ou quando $x_i + y_i = 1$ e $c_i = 1$ (o que correspondente a $(x_i \oplus y_i)c_i = 1$), de forma que

$$c_{i+1} = x_i y_i + (x_i \oplus y_i)c_i$$

Definindo as variáveis intermediárias

Propagar $p_i = x_i \oplus y_i$
Gerar $g_i = x_i \cdot y_i$

as expressões anteriores tornam-se

$$z_i = p_i \oplus c_i$$
$$c_{i+1} = g_i + p_i \cdot c_i$$

Estas expressões são a base para a implementação mostrada na Figura 10.3*b*. O submódulo que produz p_i e g_i é chamado **meio-somador** (*half-adder*) — MS — porque ele executa a adição de dois bits (ao invés de três, como acontece com o somador total), o bit de soma é p_i e o bit vai-1 é g_i, como mostra a tabela seguinte:

x_i	y_i	g_i	p_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

A implementação de um somador total que usa dois meio-somadores e uma porta OR requer um número menor de portas do que a rede de dois níveis; além disso, o retardo do transporte (do vem-1 para o vai-1), o qual é crucial para o retardo do módulo somador de transporte propagado, é menor porque corresponde ao retardo de duas portas de duas entradas.

Uma vez que as portas NAND tem retardos de propagação mais breves do que as portas AND (OR), a implementação mostrada na Figura 10.3*c* é preferível. Incorporando-se esta implementação na estrutura descrita na Figura 10.2, o retardo de propagação de pior caso do somador de transporte propagado é

$$t_p = t_{XOR} + 2(n-1)t_{NAND} + \max(2t_{NAND}, t_{XOR})$$

O primeiro t_{XOR} corresponde a p_0. Note que o retardo de p_i para $i > 0$ não é o caminho crítico, porque todos os p's são computados simultaneamente.

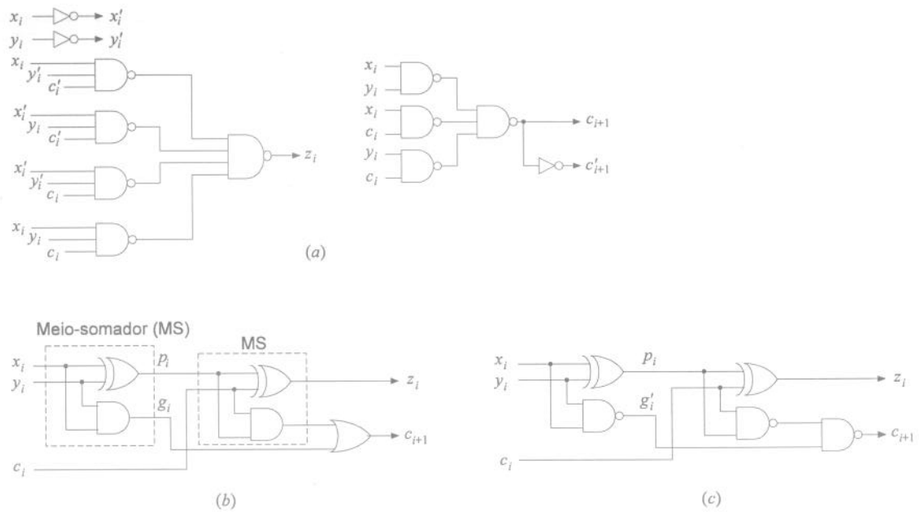

Figura 10.3 Implementações de módulo somador total. (*a*) Rede de portas de dois níveis. (*b*) Implementação multinível com portas XOR, AND e OR. (*c*) Implementação multinível com portas XOR e NAND.

A maioria das famílias de células-padrão inclui um módulo somador total. Por exemplo, na família que estamos usando como exemplo neste livro, o módulo somador total tem as características listadas da Tabela 10.1.

Tabela 10.1 Características do somador total na família de portas CMOS

Entrada	Fator de carga de entrada (cargas-padrão)	De	Para	Retardos de propagação t_{pLH} (ns)	Retardos de propagação t_{pHL} (ns)
c_i	1,3				
x_i	1,1				
y_i	1,3	c_i	z_i	0,43 + 0,03*L*	0,49 + 0,02*L*
Tamanho: 7 (portas equivalentes)		x_i	z_i	0,68 + 0,04*L*	0,74 + 0,02*L*
		y_i	z_i	0,68 + 0,04*L*	0,74 + 0,02*L*
		c_i	c_{i+1}	0,36 + 0,04*L*	0,40 + 0,02*L*
		x_i	c_{i+1}	0,73 + 0,04*L*	0,71 + 0,02*L*
		y_i	c_{i+1}	0,37 + 0,04*L*	0,64 + 0,02*L*

L: carga na saída da porta.

Implementação de somador de transporte antecipado

Uma alternativa mais rápida ao somador de transporte propagado pode ser obtida à custa de mais portas com um número maior de entradas. A redução do retardo do módulo somador exige "quebrar" o caminho de propagação do vem-1. Para estes propósitos, um bom compromisso entre velocidade e custo é obtido executando-se a adição como um processo de dois passos, da seguinte maneira:

- primeiro, determine os valores de todos os vem-1 para os módulos somadores totais;
- depois, compute simultaneamente todos os bits do resultado.

Esta abordagem, que é ilustrada na Figura 10.4, leva a um módulo **somador de transporte antecipado**. O nome é derivado da maneira pela qual o módulo é construído: o primeiro passo é implementado por um **gerador de transporte antecipado — GTA**, o qual determina os valores de todos os transportes (vem-1) intermediários antes que os bits correspondentes da soma sejam computados; o segundo passo usa estes transportes (vem-1) pré-computados para determinar o valor dos bits da soma.

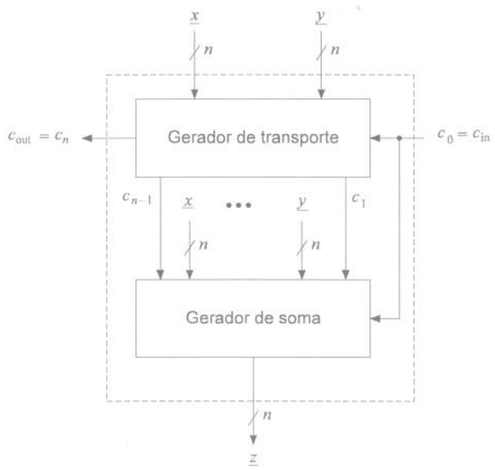

Figura 10.4 Módulo somador de transporte antecipado.

A computação dos transportes (vem-1) intermediários por uma rede de dois níveis ainda exige um grande fanin e um grande número de portas. Para reduzir estas exigências, usamos as variáveis intermediárias p_i e g_i introduzidas na implementação do somador total. Como mostrado na página 252, a computação de transportes (vem-1) é especificada pela seguinte expressão de chaveamento:

$$c_{i+1} = g_i + p_i \cdot c_i$$

em que $g_i = x_i y_i$ e $p_i = x_i \oplus y_i$. Esta expressão indica que o vai-1 de um somador total tem o valor 1 sempre que um transporte seja gerado nessa célula ($g_i = 1$), ou quando um transporte que entra no somador total é propagado por ele ($p_i = 1$). Por substituição, obtemos

$$\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 c_1 \\
&= g_1 + g_0 p_1 + p_0 p_1 c_0 \\
c_3 &= g_2 + p_2 c_2 \\
&= g_2 + g_1 p_2 + g_0 p_1 p_2 + p_0 p_1 p_2 c_0 \\
c_4 &= g_3 + p_3 c_3 \\
&= g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + p_0 p_1 p_2 p_3 c_0
\end{aligned}$$

Uma rede de portas correspondente à expressão acima é mostrada na Figura 10.5*a* (as saídas *P* e *G* serão explicadas brevemente). Além disso, $z_i = x_i \oplus y_i \oplus c_i = p_i \oplus c_i$ de forma que o módulo de transporte antecipado de quatro bits completo é implementado como mostra a Figura 10.5*b*.

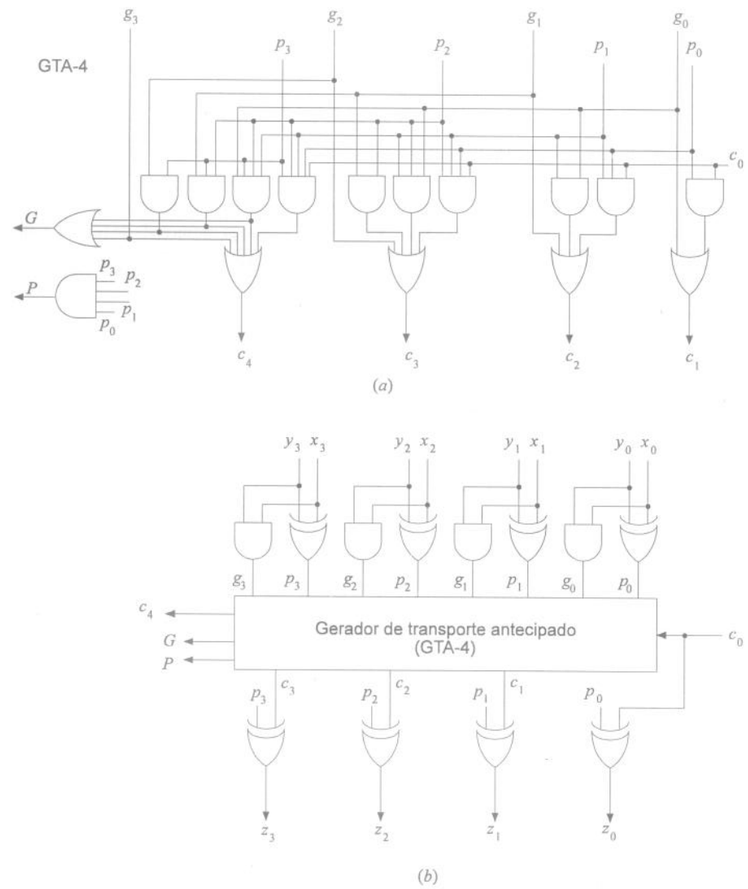

Figura 10.5 Somador de transporte antecipado — STA. (*a*) Gerador de transporte antecipado de quatro bits com saídas *P* e *G* (GTA-4). (*b*) Módulo de quatro bits (STA-4).

EXEMPLO 10.1

As tabelas seguintes mostram exemplos da determinação dos valores de $\underline{p} = (p_3, p_2, p_1, p_0)$, $\underline{g} = (g_3, g_2, g_1, g_0)$, $\underline{c} = (c_4, \ldots, c_1)$ e $\underline{z} = (z_3, z_2, z_1, z_0)$

- $c_{in} = c_0 = 0$

i	4	3	2	1	0
x_i		0	1	1	0
y_i		0	0	1	1
p_i		0	1	0	1
g_i		0	0	1	0
c_i	0	1	1	0	0
z_i		1	0	0	1

- $c_{in} = c_0 = 1$

i	4	3	2	1	0
x_i		0	1	0	1
y_i		1	0	1	0
p_i		1	1	1	1
g_i		0	0	0	0
c_i	1	1	1	1	1
z_i		0	0	0	0

As expressões de vem-1 podem ser generalizadas se as interpretarmos da seguinte maneira: c_3 é 1 se um vem-1 for gerado em ST_2, ou se um vem-1 for gerado em ST_1 e propagado através de ST_2, ou se um vem-1 for gerado em ST_0 e propagado através de ST_1 e ST_2, ou se um vem-1 entrar e for propagado através de ST_0, ST_1 e ST_2. A generalização será então

$$c_{i+1} = g_i + g_{i-1} \cdot p_i + g_{i-2} \cdot p_{i-1} \cdot p_i + \ldots$$
$$+ g_{i-k} \cdot \prod_{j=0}^{k-1} p_{i-j} + \ldots + \prod_{j=0}^{i} p_j \cdot c_0$$

Ou seja, um vem-1 sai do somador total ST_i se esse vem-1 for gerado ou em ST_i, ou for gerado em ST_{i-1} e propagado através de ST_i, ou for gerado em ST_{i-2} e propagado através de ST_{i-1} e ST_i e assim por diante.

Sinais propagar e gerar do módulo

Para facilitar a construção de somadores maiores, o módulo somador de transporte antecipado tem duas saídas adicionais, como mostra a Figura 10.5*a*:

- P, a qual indica que o valor da entrada vem-1 para o módulo é propagado pelo módulo; e
- G, a qual indica que o valor $c_{out} = 1$ é gerado pelo módulo, independentemente do valor da entrada vem-1 correspondente.

A partir das definições, obtemos

$$c_{out} = G + P \cdot c_{in}$$

Uma especificação de alto nível destas saídas adicionais é

$$P = \begin{cases} 1 & \textbf{se} \quad x + y = 2^n - 1 \\ 0 & \textbf{caso contrário} \end{cases}$$

$$G = \begin{cases} 1 & \textbf{se} \quad x + y \geq 2^n \\ 0 & \textbf{caso contrário} \end{cases}$$

Note que P e G não dependem da entrada vem-1 para o módulo.

Para obter uma expressão de chaveamento para P, observe que $x + y = 2^n - 1$ implica que para todos os bits a soma de x_i mais y_i é igual a 1. Conseqüentemente, $p_i = x_i \oplus y_i = 1$ para todos os i e

$$P = \prod p_i \qquad i = 0, \ldots, n-1$$

Similarmente, para G, se $x + y \geq 2^n$, existe algum j para o qual $x_j = y_j = 1$ ($g_j = 1$) e $p_k = x_k \oplus y_k = 1$, para todo $k > j$. Um exemplo disto, para $n = 8$ e $j = 3$ é

i	7	6	5	4	3	2	1	0
x_i	1	0	0	1	1	—	—	—
y_i	0	1	1	0	1	—	—	—

A expressão correspondente será então

$$G = g_{n-1} + g_{n-2} p_{n-1} + g_{n-3} p_{n-1} p_{n-2} + \ldots + g_0 p_{n-1} p_{n-2} \cdots p_1$$

A Figura 10.5*a* mostra uma rede de portas de dois níveis de geração de transporte antecipado (GTA-4) que produz, além de vem-1's, os sinais de módulo propagar P e gerar G para $n = 4$.

O retardo de propagação de um módulo somador de transporte antecipado é:

$$t_p(x_0 \rightarrow c_4) = t_{XOR} + t_{GTA\text{-}4}$$
$$t_p(c_0 \rightarrow c_4) = t_{GTA\text{-}4}$$
$$t_p(x_0 \rightarrow P, G) = t_{XOR} + t_{GTA\text{-}4}$$
$$t_p(x_0 \rightarrow z_3) = t_{XOR} + t_{GTA\text{-}4} + t_{XOR}$$

Esta implementação é mais rápida do que o módulo somador de transporte propagado discutido na seção anterior. Porém, o número de bits de entrada e saída para esta implementação é limitado pelo número de fanin de portas e pelo número de conexões externas; além disso, uma análise exata do retardo de propagação exige considerar os retardos de portas específicos, levando em conta o fanout dos mesmos (veja o Exercício 10.2).

10.2 REDES DE MÓDULOS SOMADORES

Conforme discutimos anteriormente para outros módulos, o número de bits em operandos e o resultado de módulos somadores são limitados devido às limitações no número de entradas e saídas de módulo e ao número de portas e entradas de portas. Vetores de bits maiores exigem a utilização de **redes somadoras**, compostas de módulos somadores.

As redes de módulos somadores são implementadas com as mesmas abordagens usadas para os próprios módulos, ou seja, estruturas iterativas (ou de transporte propagado) e antecipação de transporte. Estas redes resultam de uma extensão dos dois tipos de módulos correspondentes, conforme discutiremos a seguir.

Rede somadora (de transporte propagado) iterativa

Consideremos a implementação de um somador de 16 bits, usando módulos somadores padrão de 4 bits. Para estas finalidades, os vetores de bits de entrada e saída são decompostos em subvetores de 4 bits. Por exemplo, para o vetor de bits $\underline{x}$, a decomposição é a seguinte:

$$\underline{x} = (\underline{x}^{(3)}, \underline{x}^{(2)}, \underline{x}^{(1)}, \underline{x}^{(0)})$$
$$\underline{x}^{(3)} = (x_{15}, x_{14}, x_{13}, x_{12})$$
$$\underline{x}^{(2)} = (x_{11}, x_{10}, x_9, x_8)$$
$$\underline{x}^{(1)} = (x_7, x_6, x_5, x_4)$$
$$\underline{x}^{(0)} = (x_3, x_2, x_1, x_0)$$

Uma vez que os vetores de bits correspondem à representação de base 2 de números inteiros, então

$$x = 2^{12} x^{(3)} + 2^8 x^{(2)} + 2^4 x^{(1)} + x^{(0)}$$

em que $x^{(i)}$ representa o valor inteiro representado por $\underline{x}^{(i)}$. O mesmo tipo de decomposição é aplicado aos vetores de bits $\underline{y}$ e $\underline{z}$.

Baseando-se nesta decomposição, podemos construir uma rede com quatro módulos somadores de quatro bits, como descreve a Figura 10.6. Cada par de subvetores correspondentes é usado como entrada para um dos módulos somadores e cada um destes módulos produz um dos subvetores de saída. Os módulos são interconectados de uma maneira de transporte propagado (iterativa): a saída vai-1 de um módulo é usada como a entrada vem-1 para o módulo vizinho esquerdo. Note que os módulos que formam a estrutura iterativa podem ser ou do tipo de transporte propagado ou do tipo de transporte antecipado.

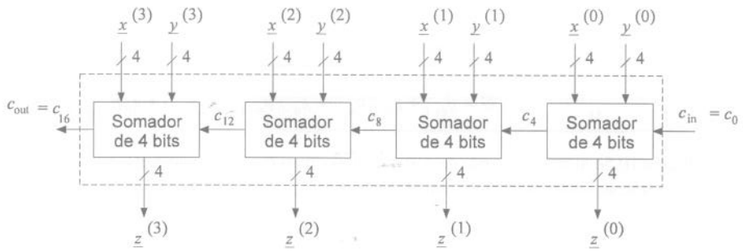

Figura 10.6 Rede somadora de transporte propagado de 16 bits que usa módulos somadores de 4 bits.

A estrutura iterativa pode ser estendida para qualquer número de módulos. Porém, conforme indicamos anteriormente, o retardo desta rede é proporcional ao número de módulos.

Rede somadora de transporte antecipado

Outra abordagem à implementação de somadores grandes é a rede de somadores de transporte antecipado. Esta abordagem opera da mesma maneira que no caso dos módulos somadores de transporte antecipado. Ou seja, todos os transportes que entram nos módulos somadores são computados simultaneamente antes de computarem os bits do resultado, e todos os bits do resultado são computados simultaneamente usando os transportes intermediários obtidos da etapa anterior. A rede resultante usa dois níveis de antecipação: um dentro dos módulos e o outro entre os módulos.

Para este caso, vamos considerar a implementação de um somador de 32 bits usando módulos somadores de transporte antecipado de 4 bits (STA-4). Conseqüentemente, são necessários oito módulos STA-4, como mostra a Figura 10.7. Os vem-1 para estes módulos são computados usando módulos geradores de transporte antecipado (GTAs). Para ser coerente com o tamanho dos módulos somadores, cada módulo gerador de transporte antecipado produz quatro transportes, de forma que são necessários dois módulos GTA-4, como mostra a Figura 10.7.

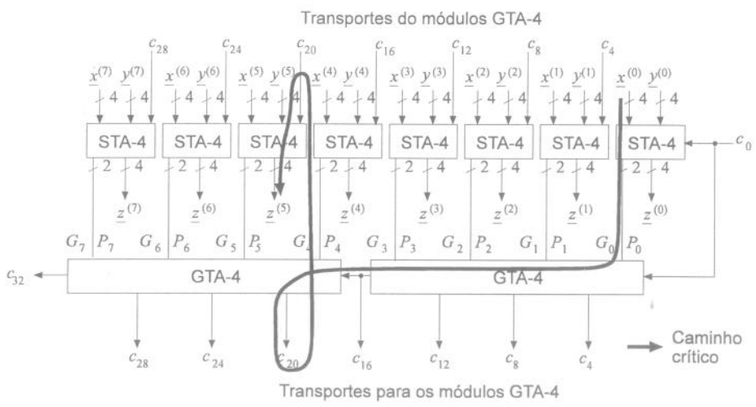

Figura 10.7 Somador de transporte antecipado de 32 bits que usa módulos STA-4 e GTA-4.

O primeiro módulo GTA-4 produz c_4, c_8, c_{12} e c_{16}, da seguinte maneira:

$$
\begin{aligned}
c_4 &= G_0 + P_0c_0 \\
c_8 &= G_1 + P_1G_0 + P_1P_0c_0 \\
c_{12} &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0 \\
c_{16} &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0
\end{aligned}
$$

em que P_i e G_i são os sinais de módulo propagar e gerar definidos na página 256. Note que estas expressões têm a mesma forma que no gerador de transporte antecipado do módulo somador.

O segundo módulo GTA-4 produz os transportes c_{20}, c_{24}, c_{28} e c_{32}. As expressões correspondentes são análogas às do primeiro módulo. Note que a entrada vem-1 para o segundo modulo GTA-4 é c_{16}, a qual é produzida pelo primeiro módulo.

As entradas para os módulos GTA são produzidas por módulos STA; cada um destes módulos recebe entradas de quatro bits, produz saídas P_i e G_i e implementa expressões de chaveamento como estas (como discutimos na seção anterior):

$$
\begin{aligned}
P_0 &= p_3 \cdot p_2 \cdot p_1 \cdot p_0 \\
G_0 &= g_3 + g_2p_3 + g_1p_3p_2 + g_0p_3p_2p_1
\end{aligned}
$$

em que $p_i = x_i \oplus y_i$ e $g_i = x_i \cdot y_i$ como no módulo somador de transporte antecipado.

O retardo de propagação desta rede é obtido seguindo-se o caminho crítico da Figura 10.7, ou seja,

$$t_p(\text{rede}) = t_{PG} + 2t_{GTA\text{-}4} + t_{SOMA}$$

em que t_{PG} é o retardo do módulo STA-4 para produzir as saídas P e G, $t_{GTA\text{-}4}$ é o retardo do módulo gerador de transporte antecipado e t_{SOMA} é o retardo das saídas da soma do módulo somador. Note que t_{SOMA} corresponde ao retardo para gerar os transportes internos, o que é feito usando-se o gerador de antecipação dentro do módulo somador, mais a geração da soma.

Este esquema de antecipação freqüentemente é usado para somadores rápidos; níveis adicionais de antecipação podem ser usados para reduzir ainda mais o retardo em somadores grandes. Uma comparação entre as duas implementações discutidas — redes de somadores com propagação e com antecipação — será deixada como um exercício.

10.3 REPRESENTAÇÃO DE NÚMEROS INTEIROS COM SINAL E OPERAÇÕES BÁSICAS

Consideraremos, agora, a representação de números inteiros com sinal (positivos e negativos) e a implementação das operações de adição e subtração correspondentes.

10.3.1 Representação e detecção de sinal

No Capítulo 2, descrevemos a representação dos números inteiros positivos. Estenderemos, agora, a discussão para a representação de números inteiros com sinal — positivos e negativos. Duas representações são, sem dúvida, as mais comuns: a representação de sinal e magnitude e a representação de complemento de dois. Discutiremos também a representação de complemento de um, menos usada, e consideraremos a implementação da detecção do sinal.

Sistema sinal e magnitude (SM)

Um número inteiro com sinal x é representado no sistema SM pelo par (x_s, x_m), em que x_s é o **sinal** e x_m é a **magnitude**. A convenção usual é que $x_s = 0$ corresponde ao positivo e $x_s = 1$ ao negativo.

Numa representação binária com n bits, um bit é usado para o sinal e $n - 1$ bits para a magnitude. Conseqüentemente, a faixa de números inteiros com sinal é

$$-(2^{n-1} - 1) \le x \le 2^{n-1} - 1$$

Note que o zero tem duas representações: $x_s = 0$, $x_m = 0$ (zero positivo) e $x_s = 1$, $x_m = 0$ (zero negativo).

Uma vez que há um bit específico para representar o sinal, a operação de detecção do sinal é direta. Entretanto, devido ao 0 negativo, a relação $x \geq 0$ exige a verificação do 0 negativo, ou seja

$$x \geq 0 \ \textbf{se} \ (x_s = 0) \ \textbf{ou} \ (x_m = 0)$$

Sistema complemento de dois

No sistema complemento de dois, não é feita nenhuma separação entre a representação do sinal e a representação da magnitude, mas todo o número inteiro com sinal é representado por um número inteiro positivo. Conseqüentemente, esta representação envolve um mapeamento adicional, como indica a Figura 10.8. O número inteiro com sinal x é representado por um número inteiro positivo x_R, o qual, por sua vez, é representado pelo vetor de bits $\underline{x}$. O Mapeamento 1 define a correspondência entre x e x_R, enquanto que o Mapeamento 2 define a representação de x_R por um vetor de bits.

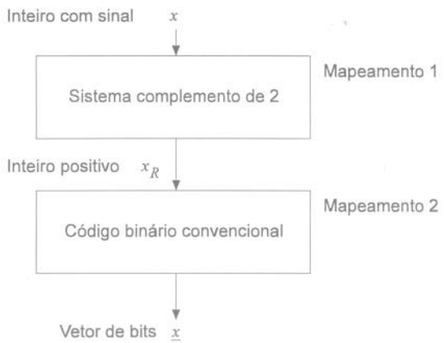

Figura 10.8 Número inteiro com sinal representado por número inteiro positivo.

Consideremos, primeiro, o Mapeamento 2. O número inteiro positivo x_R é representado pelo vetor de n bits $\underline{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ no sistema numérico binário, de forma que

$$x_R = \sum_{i=0}^{n-1} x_i 2^i \qquad 0 \leq x_R \leq 2^{n-1} - 1$$

Então, para o sistema complemento de dois, o Mapeamento 1 é definido por

$$x_R = x \bmod 2^n$$

Pela definição da função mod, para $|x| < 2^n$, isto é equivalente a

$$x_R = \begin{cases} x & \textbf{se} \quad x \geq 0 \\ 2^n - |x| & \textbf{se} \quad x < 0 \end{cases}$$

Conseqüentemente, como ilustra a Tabela 10.2, o mapeamento divide a faixa da representação em duas regiões: a região correspondente aos números inteiros positivos (também chamada **formas verdadeiras** porque nenhuma transformação é executada entre x e x_R), e a região negativa (chamada **formas complementares**, porque a representação é obtida subtraindo-se $|x|$ de 2^n). A descrição circular dada para $n = 4$ na Figura 10.9 mostra a relação entre esta representação e a função mod.

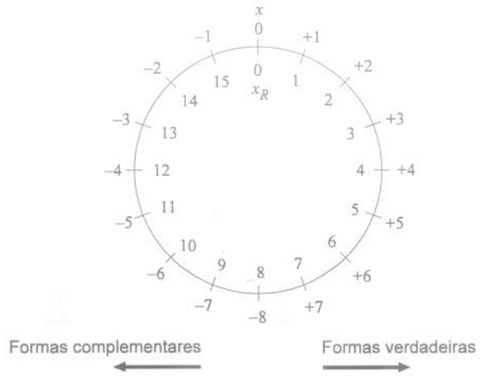

Figura 10.9 Representação em complemento de dois para $n = 4$.

Para se ter uma representação inequívoca, a região $x > 0$ não deve se sobrepor à região correspondente a $x < 0$. Isto exige que

$$|x| \leq 2^{n-1} - 1$$

Porém, esta faixa não inclui $x_R = 2^{n-1}$. Três opções são possíveis para este valor: ele pode ser usado para representar o número inteiro positivo $x = 2^{n-1}$ ou o número inteiro negativo $x = -2^{n-1}$ ou pode não ser usado. Como mostra a Tabela 10.2, a escolha usual é a representação do número inteiro negativo, porque ela simplifica a detecção do sinal e o estouro aritmético (*overflow*) na operação de adição, conforme discutiremos mais tarde.

Tabela 10.2 Mapeamento no sistema complemento de dois

x	x_R	$\underline{x}$	
0	0	00. . . 000	
1	1	00. . . 001	
2	2	00. . . 010	
—	—	—	Formas verdadeiras
—	—	—	(positivas)
—	—	—	$x_R = x$
$2^{n-1} - 1$	$2^{n-1} - 1$	01. . . 111	
-2^{n-1}	2^{n-1}	10. . . 000	
$-(2^{n-1} - 1)$	$2^{n-1} + 1$	10. . . 001	
—	—	—	
—	—	—	Formas complementares
—	—	—	(negativas)
–2	$2^n - 2$	11. . . 110	$x_R = 2^n - \lvert x \rvert$
–1	$2^n - 1$	11. . . 111	

EXEMPLO 10.2

A tabela seguinte apresenta as correspondências para $-4 \leq x \leq 3$ no sistema complemento de dois. Neste caso, $2^n = 8$ e x_R é representado pelo vetor de três bits $\underline{x} = (x_2, x_1, x_0)$.

x	x_R	$\underline{x}$	x	x_R	$\underline{x}$
3	3	011	–1	7	111
2	2	010	–2	6	110
1	1	001	–3	5	101
0	0	000	–4	4	100

Para obter o número inteiro com sinal x dada a representação x_R, o mapeamento inverso é

$$x = \begin{cases} x_R & \textbf{se} \quad x_R \leq 2^{n-1} - 1 \ (x \geq 0) \\ x_R - 2^n & \textbf{se} \quad x_R \geq 2^{n-1} - 1 \ (x < 0) \end{cases}$$

Este mapeamento inverso também pode ser expresso em termos dos bits de x_R. Para isto, consideremos os dois casos do mapeamento, da seguinte maneira:

i. $x = x_R$. Como $x_R < 2^{n-1}$, o bit x_{n-1} é 0. Conseqüentemente,

$$x = x_R = 0 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

ii. $x = x_R - 2^n$. Como $x_R \geq 2^{n-1}$, o bit x_{n-1} é 1. Isto resulta em

$$x = x_R - 2^n = (1 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i) - 2^n = -1 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

Combinando ambos os casos, obtemos

$$x = -x_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

Dois exemplos de oito bits são

$\underline{x}$	x
01000101	0 + 69 = 69
11000101	−128 + 69 = −59

A implementação da detecção de sinal é obtida diretamente do mapeamento inverso, de forma que o sinal é obtido do bit mais significativo de $\underline{x}$, da seguinte maneira:

$$x \geq 0 \quad \textbf{se} \quad x_{n-1} = 0$$
$$x < 0 \quad \textbf{se} \quad x_{n-1} = 1$$

Sistema complemento de um

O sistema de representação de complemento de um pertence à mesma classe que o sistema complemento de dois, a saber, os sistemas verdadeiro e complementar. A representação para o elemento genérico desta classe é descrita por:

$$x_R = x \bmod C$$

em que C é a constante de complementação.

Para o sistema complemento de dois, $C = 2^n$, enquanto que para o sistema complemento de um, $C = 2^n - 1$. Esta representação é ilustrada na Tabela 10.3. Conforme é mostrado,

- o sistema complemento de um é simétrico e a faixa é

$$-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$$

- há duas representações para zero: $x_R = 0$ e $x_R = 2^n - 1$;
- o sinal também é detectado pelo bit mais significativo, mas, como no sistema sinal e magnitude, a relação $x \geq 0$ é implementada como

$$x \geq 0 \ \textbf{se} \ (x_{n-1} = 0) \ \textbf{ou} \ (x_R = 2^n - 1)$$

Tabela 10.3 Mapeamento no sistema complemento de um

x	x_R	$\underline{x}$	
0	0	00. . . 000	
1	1	00. . . 001	
2	2	00. . . 010	
—	—	—	Formas verdadeiras
—	—	—	(positivas)
—	—	—	$x_R = x$
$2^{n-1} - 1$	$2^{n-1} - 1$	01. . . 111	
$-(2^{n-1} - 1)$	2^{n-1}	10. . . 000	
—	—	—	
—	—	—	Formas complementares
−2	$2^n - 3$	11. . . 101	(negativas)
−1	$2^n - 2$	11. . . 110	$x_R = 2^n - 1 - \lvert x \rvert$
0	$2^n - 1$	11. . . 111	

10.3.2 Adição e subtração de números inteiros com sinal

Descreveremos, agora, as operações de adição e subtração de números inteiros com sinal. Especificamente, digamos que x e y sejam números inteiros com sinal representados pelos vetores $\underline{x}$ e $\underline{y}$, respectivamente. A operação de adição ADD produz o vetor de bits $\underline{z}$ que representa o número inteiro com sinal $z = x + y$. Ou seja,

$$\underline{z} = \text{ADD}(\underline{x}, \underline{y})$$

Para a diferença $d = x - y$, como $d = x + (-y)$, basta combinarmos a operação ADD e a operação de troca de sinal CS (*change of sign*). Ou seja,

$$\underline{d} = \text{ADD}(\underline{x}, \text{CS}(\underline{y}))$$

Se a faixa de números inteiros representada por $\underline{z}$ for a mesma que a de $\underline{x}$ e $\underline{y}$ (isto é, se ela tiver o mesmo número de bits), o resultado da adição ou subtração poderá não ser representado por $\underline{z}$. Nesse caso, um sinal de **estouro aritmético** (*overflow*) $v = 1$ indicará esta situação.

A complexidade de implementar operações ADD e CS depende do sistema de representação usado para os números inteiros com sinal. Consideraremos, agora, estas implementações para o sistema complemento de dois, o qual, na prática, é mais comumente usado, e deixaremos os casos de sinal e magnitude e complemento de um como exercícios.

Adição no sistema complemento de dois

No sistema complemento de dois, $z = x + y$ é obtido computando-se

$$z_R = (x_R + y_R) \bmod 2^n$$

Mostraremos, agora, que esta expressão produz o resultado correto se não houver nenhum estouro aritmético, ou seja, se $-2^{n-1} \leq (x + y) \leq 2^{n-1} - 1$. Vamos considerar que

$$(x_R + y_R) \bmod 2^n$$

e mostrar que ele corresponde a z_R, que é a representação da soma z. Pela definição da representação

$$x_R = x \bmod 2^n$$
$$y_R = y \bmod 2^n$$

de forma que

$$(x_R + y_R) \bmod 2^n = (x \bmod 2^n + y \bmod 2^n) \bmod 2^n$$

Entretanto, isto pode ser simplificado, porque $(a \bmod 2^n + b \bmod 2^n) \bmod 2^n = (a + b) \bmod 2^n$; conseqüentemente,

$$(x_R + y_R) \bmod 2^n = (x + y) \bmod 2^n = z \bmod 2^n$$

e, por definição,

$$z \bmod 2^n = z_R$$

Isto significa que, para executar a adição de dois números inteiros com sinal representados no sistema complemento de dois, adicionamos a representação (positiva) e obtemos o resíduo (mod) da soma com respeito a 2^n. Esta operação é feita independentemente das magnitudes relativas dos operandos e seus sinais (isto se contrapõe ao caso do sinal e magnitude; veja o Exercício 10.14). São exemplos de adição para $2^n = 64$ e $-32 \le x, y, z \le 31$

Operandos com sinal		Representação		Adição em complemento de dois	Resultado com sinal
x	y	x_R	y_R	$(x_R + y_R) \bmod 64 = z_R$	z
13	9	13	9	22 mod 64 = 22	22
13	–9	13	55	68 mod 64 = 4	4
–13	9	51	9	60 mod 64 = 60	–4
–13	–9	51	55	106 mod 64 = 42	–22

Conseqüentemente, a operação de adição em complemento de dois consiste em dois passos:

- a adição das representações positivas; e
- a operação mod.

O primeiro passo é executado por um somador para os operandos positivos, conforme discutimos na Seção 10.1 e 10.2; consideraremos, agora, a operação mod. Admitamos que $w_R = x_R + y_R$; como $x_R, y_R < 2^n$, então $w_R < 2 \times 2^n$. Portanto, para este caso, a operação mod corresponde a:

$$z_R = w_R \bmod 2^n = \begin{cases} w_R & \text{se} \quad w_R < 2^n \\ w_R - 2^n & \text{se} \quad 2^n \le w_R < 2 \times 2^n \end{cases}$$

Conseqüentemente, esta operação consiste em determinar se $w_R \ge 2^n$ e, nesse caso, subtrair 2^n dele.

Uma vez que $w_R < 2 \times 2^n$, a representação de w_R no sistema numérico de base 2 é um vetor de bits $\underline{w} = (w_n, w_{n-1}, \ldots, w_0)$ de $n + 1$ bits. A determinação de se $w_R \ge 2^n$ pode ser realizada simplesmente verificando-se o bit mais significativo de $\underline{w}$, de forma que

$$w_R = \begin{cases} < 2^n & \text{se} \quad w_n = 0 \\ \ge 2^n & \text{se} \quad w_n = 1 \end{cases}$$

No primeiro caso, $w_R \bmod 2^n = w_R$ e sua representação é $(w_{n-1}, \ldots, w_0)$. No segundo caso, é necessário subtrair 2^n de w_R; isto é simples de fazer, porque a representação de 2^n é um 1 seguido de n 0's. Conseqüentemente,

$$w_R \bmod 2^n \Leftrightarrow (1, w_{n-1}, \ldots, w_0) - (1, 0, \ldots, 0) = (w_{n-1}, \ldots, w_0)$$

Ou seja, em ambos os casos, a operação mod é executada descartando-se o bit mais significativo do vetor de bits $\underline{w}$. Este bit corresponde ao vai-1 da adição de x_R e y_R para produzir w_R.

Em suma, no sistema complemento de dois, o resultado de adicionar números inteiros com sinal corresponde à saída do somador, descartando-se o vai-1. Descrevemos isto da seguinte maneira:

$$\underline{z} = \text{ADD}(\underline{x}, \underline{y}, 0)$$

em que ADD é a adição de números inteiros positivos e o terceiro operando corresponde ao vem-1. O módulo somador complemento de dois é mostrado na Figura 10.10.

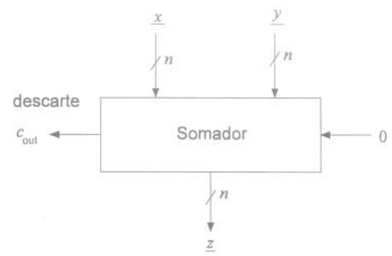

Figura 10.10 Módulo somador complemento de dois.

EXEMPLO 10.3

Dois exemplos de adição de números inteiros com sinal no sistema numérico complemento de dois são:

	Computação no nível de bit	Representação positiva	Valores com sinal
$n = 4$	$\underline{x}$ = 1011 $\underline{y}$ = 0101	x_R = 11 y_R = 5	x = –5 y = 5
	$\underline{w}$ = 10000	w_R = 16	
	$\underline{z}$ = 0000	z_R = 0	z = 0
$n = 8$	$\underline{x}$ = 11011010 $\underline{y}$ = 11110001	x_R = 218 y_R = 241	x = –38 y = –15
	$\underline{w}$ = 111001011	w_R = 459	
	$\underline{z}$ = 11001011	z_R = 203	z = –53

Troca de sinal no sistema complemento de dois

A operação de troca de sinal consiste em obter z_R a partir de x_R de forma que $z = -x$. A operação nesta representação, também chamada de complementação, é

$$z_R = (2^n - x_R) \bmod 2^n$$

Para mostrarmos a correção da operação, consideremos os três casos, $x = 0$, $x \ge 0$ e $x < 0$. Para o caso $x \ne 0$, $x_R > 0$, de forma que $(2^n - x_R) \bmod 2^n = 2^n - x_R$. Conseqüentemente, nesses casos, desprezamos o mod 2^n.

$x = 0$: $-x = 0$, então $z_R = x_R = 0$. Isto confirma a expressão acima em relação à troca de sinal, uma vez que $(2^n - 0) \bmod 2^n = 0$.

$x > 0$: $-x$ é negativo, então $z_R = 2^n - |z| = 2^n - x$. Além disso, x é positivo, de forma que $x_R = x$. Substituindo, obtemos $z_R = 2^n - x_R$.

$x < 0$: $-x$ é positivo, então $z_R = z = -x$. Além disso, x é negativo, de forma que

$$x_R = 2^n - |x| = 2^n + x.$$

Novamente, substituindo, obtemos $z_R = 2^n - x_R$.

Conseqüentemente, a operação de troca de sinal consiste em subtrair x_R de 2^n. A implementação direta exige uma subtração completa, a qual é complexa; uma implementação mais simples é obtida observando-se que $2^n = (2^n - 1) + 1$, de forma

$$z_R = (2^n - 1 - x_R) + 1$$

Uma vez que a representação de $2^n - 1$ é (1, 1, . . ., 1), obtemos $2^n - 1 - x_R$ ao complementarmos cada bit de $\underline{x}$. Por exemplo, para $n = 6$

$$x_R = 17 = 010001$$
$$63 - x_R = 111111 - 010001 = 101110 = 46$$

Conseqüentemente, a operação de troca de sinal consiste em dois passos:

- complemente cada bit para obter $\underline{x}'$; e
- adicione 1.

A adição de 1 pode ser realizada fazendo-se o vem-1 em $c_0 = 1$. A descrição correspondente é

$$\underline{z} = \text{ADD}(\underline{x}', \underline{0}, 1)$$

EXEMPLO 10.4

Para $n = 4$ e $x = -3$, a tabela seguinte descreve a operação de troca de sinal no sistema complemento de dois:

$\underline{x}$	1101	$x = -3$
$\underline{x}'$	0010	
0	0000	
c_0	1	
$\underline{z}$	0011	z = 3

Subtração no sistema complemento de dois

Como já indicamos, a subtração é executada combinando-se troca de sinal e adição, porque $z = x - y = x + (-y)$. Nos termos das operações de adição e troca de sinal já descritas, obtemos

$$z_R = (x_R + (2^n - 1 - y_R) + 1) \bmod 2^n$$

As duas adições desta expressão podem ser combinadas em uma única ao fixarmos em 1 o vem-1 para o somador; a descrição correspondente é

$$\underline{z} = \text{ADD}(\underline{x}, \underline{y}', 1)$$

EXEMPLO 10.5

A tabela seguinte descreve a subtração no sistema complemento de dois:

$\underline{x}$			01100000
$\underline{y}$	00110001	$\underline{y}'$	11001110
			1
$\underline{z}$			00101111

Resumo

Um resumo da implementação das operações de adição, troca de sinal e subtração para o sistema complemento de dois é o seguinte:

Operação	Sistema complemento de dois
$z = x + y$	$\underline{z} = \text{ADD}(\underline{x}, \underline{y}, 0)$
$z = -x$	$\underline{z} = \text{ADD}(\underline{x}', \underline{0}, 1)$
$z = x - y$	$\underline{z} = \text{ADD}(\underline{x}, \underline{y}', 1)$

Detecção de estouro no sistema complemento de dois

Existe um **estouro** (*overflow*) sempre que o resultado da adição ou subtração está fora da faixa dos números inteiros representáveis; nesse caso, o resultado é incorreto, de forma que é necessário detectar esta situação.

Consideramos o caso da adição; a situação é similar para a subtração se o sinal de y for trocado. Na operação de adição, pode ocorrer um estouro somente quando os operandos são do mesmo sinal; quando os sinais são diferentes, o resultado não pode ser mais positivo ou mais negativo do que ambos os operandos. No caso de operandos de mesmo sinal, ocorrerá um estouro se a adição produzir um resultado de sinal oposto (veja a Figura 10.11). Conseqüentemente, no sistema complemento de dois, ocorrerá um estouro ($v = 1$) se

$$v = x'_{n-1} y'_{n-1} z_{n-1} + x_{n-1} y_{n-1} z'_{n-1}$$

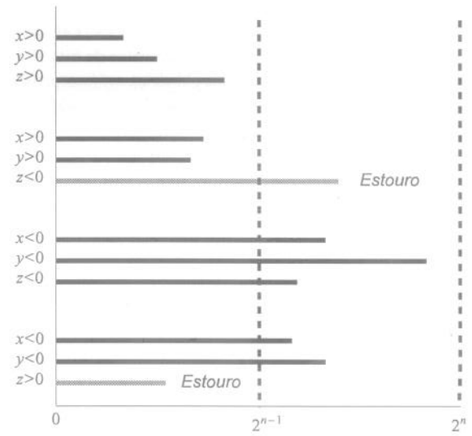

Figura 10.11 Estouro no sistema complemento de dois.

Outra maneira de detectar um estouro aritmético no sistema complemento de dois consiste em verificar os dois transportes mais significativos da adição (veja o Exercício 10.15), da seguinte maneira:

$$v = c_n \oplus c_{n-1}$$

Unidade aritmética em complemento de dois

Descreveremos, agora, uma implementação combinacional de uma unidade aritmética em complemento de dois para operações básicas com números inteiros com sinal. A especificação de alto nível desta unidade é a seguinte:

Entradas: $\underline{x} = (x_{n-1}, \ldots, x_0)$, $x_j \in \{0, 1\}$
$\underline{y} = (y_{n-1}, \ldots, y_0)$, $y_j \in \{0, 1\}$
$c_{in} \in \{0, 1\}$
$F = (f_2, f_1, f_0)$

Saídas: $\underline{z} = (z_{n-1}, \ldots, z_0)$, $z_j \in \{0, 1\}$
c_{out}, *sgn*, *zero*, *ovf* $\in \{0, 1\}$