Capítulo 7

Aritmética Binária

As operações aritméticas básicas de um sistema digital são adição e subtração de dois números (binários). Nós veremos nesta e na próxima aula que elas podem ser implementadas com as portas lógicas AND, OR, NOT e XOR. As operações de multiplicação e de divisão são comparativamente mais complexas. Essas podem, porém, ser implementadas como uma sequência de adição, multiplicação e deslocamento entre os dígitos.

7.1 Representações de Números

A **base**, ou **raiz**, de um sistema de numeração é definida como o número de dígitos diferentes que podem ocorrer em cada posição num sistema de numeração.

Exemplo 7.1 *Um sistema binário tem 2 dígitos, 0 e 1; quinário tem 5 dígitos (usado pelos esquimós e índios da América do Norte); octal tem 8 dígitos; decimal tem 10 dígitos (é o mais conhecido e os dígitos são 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9); duodecimal tem 12 dígitos e hexadecimal tem 16 dígitos.*

A partir dos dígitos, $0 \leq p_i \leq b-1$, de uma base b, podemos representar qualquer número na base b, $(N)_b$, utilizando a **notação posicional**:

$$N_b = (p_n p_{n-1} \cdots p_1 p_0, p_{-1} p_{-2} \cdots p_{-m})_b = \sum_{i=-m}^{n} p_i b^i,$$

onde $p_n p_{n-1} \cdots p_1 p_0$ e $p_{-1} p_{-2} \cdots p_{-m}$ correspondem, respectivamente a parte inteira e a parte fracionária do número.

Uma pergunta que se pode surgir ao ver esta variedade nas bases de representação é como se pode converter a representação de uma base para a outra. E, particularmente, de uma base qualquer para a base de um sistema digital, que é binária.

7.1.1 Conversão de Bases

Um método para converter os números representados em qualquer base b para a base decimal é o **método polinomial**, no qual expressa-se um número $(N)_b$ na base b como um polinômio $(N)_b = \sum_{i=0}^{k} = p_i b^i$, onde p_i são os algarismos do número.

Exemplo 7.2

$$\begin{aligned}(1011,101)_2 &= 1\times(2)^3 + 0\times(2)^2 + 1\times(2)^1 + 1\times(2)^0 + 1\times(2)^{-1} + \\ &\quad +1\times(2)^{-2} + 1\times(2)^{-3} = (11,625)_{10}\end{aligned}$$

A conversão de uma representação na base decimal para uma base b pode ser efetuado através do **método iterativo**, envolvendo iterações em multiplicação e divisão. Este método trata a parte inteira e a parte fracionária separadamente.

Para a parte inteira divide-se iterativamente N por b e vai acumulando os restos $\cdots r_2 r_1 r_0$ sucessivamente até que o quociente da divisão fique nula.

Exemplo 7.3 *Considere a conversão dos seguintes números na base decimal para outras bases:*

$(25)_{10} = (11001)_2$	*2*	*25*	*Resto*
	2	*12*	*1*
	2	*6*	*0*
	2	*3*	*0*
	2	*1*	*1*
		0	*1*
$(410)_{10} = (3120)_5$	*5*	*410*	*Resto*
	5	*82*	*0*
	5	*16*	*2*
	5	*3*	*1*
		0	*3*
$(34)_{10} = (42)_8$	*8*	*34*	*Resto*
	8	*4*	*2*
		0	*4*
$(128)_{10} = (152)_9$	*9*	*128*	*Resto*
	9	*14*	*2*
	9	*1*	*5*
		0	*1*
$(30)_{10} = (1E)_{16}$	*16*	*30*	*Resto*
	16	*1*	*14 (E)*
		0	*1*

Para a parte fracionária o procedimento consiste em multiplicar esta parte por b. Se o produto s for menor que 1, então o primeiro algarismo depois da vírgula é 0; senão ele é a parte inteira do s. O próximo algarismo é a parte inteira do resultado obtido pela multiplicação da parte fracionária de s por b. E assim sucessivamente até que a parte fracionária fique nula ou então até uma determinada precisão.

Exemplo 7.4 *Considere a conversão do número fracionário* $0,9375$ *na base decimal para outras bases:*

$(0,9375)_{10} = (0,1111)_2$	*0,9375*	×	*2*	=	*1,888*	*0,1*
	0,888	×	*2*	=	*1,75*	*0,11*
	0,75	×	*2*	=	*1,5*	*0,111*
	0,5	×	*2*	=	*1,0*	*0,1111*
$(0,9375)_{10} = (0,33)_4$	*0,9375*	×	*4*	=	*3,75*	*0,3*
	0,75	×	*4*	=	*3,0*	*0,33*
$(0,9375)_{10} = (0,74)_8$	*0,9375*	×	*8*	=	*7,50*	*0,7*
	0,5	×	*8*	=	*4,0*	*0,74*
$(0,9375)_{10} = (0,F)_{16}$	*0,9375*	×	*16*	=	*15,0*	*0,F*

Vale observar aqui que a conversão **exata** da base 2 para a base 4, 8 e 16 é direta, agrupando de 2 em 2, de 3 em 3 e de 4 em 4 algarismos, respectivamente.

Exemplo 7.5

$$(11101,11111)_2 = (131,332)_4 = (35,76)_8 = (1D,F1)_{16}$$

Quando se trata de uma conversão de um número N de base b_1 para outra base qualquer b_2, é usual converter primeiro $(N)_b$ para a base decimal e depois, da base decimal para b_2, como ilustra o seguinte exemplo.

Exemplo 7.6

$$(1110) = (30)_{10} = (1010)_3 = (33)_9.$$

7.1.2 Números Negativos

Três formas mais conhecidas para distinguir os números positivos dos números negativos são:

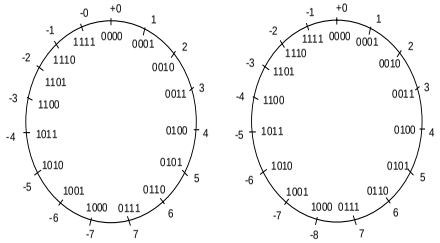
- sinal-e-magnitude;
- complemento de $b-1$, onde b é a base do sistema.
- complemento de b, onde b é a base do sistema;

Na representação sinal-e-magnitude reserva-se um dígito para distinguir os números negativos (usualmente, atribui-se 1 ao dígito) dos positivos (usualmente, atribui-se 0 ao dígito). E o resto dos dígitos é utilizado para representar o valor absoluto do número. **Nas representações em complemento, definimos a-priori o "universo" dos números possíveis de serem**

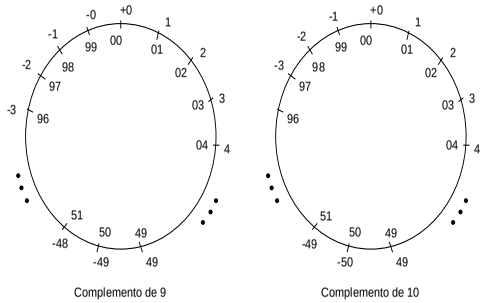
representados, através da especificação da quantidade máxima n de dígitos que serão utilizados para denotar um número. Para obter o complemento de $b-1$ (o correspondente negativo) de um número positivo N subtrai-se do b^n-1 a magnitude de N. E, na representação por complemento de b, o negativo de N corresponde à subtração $b^n - N$.

Note-se que

- há dois tipos de complementos para cada sistema de numeração de base b: complemento de b e complemento de $b-1$.
- num sistema de base b com n dígitos, consegue-se representar distintamente b^n números. Nem sempre é possível representar uma quantidade igual de números positivos e números negativos para uma dada combinação de b e n.
- no sistema binário, com n dígitos, podemos representar no complemento de 2 $\frac{2^n}{2}-1$ números positivos e $\frac{2^n}{2}$ números negativos e no complemento de 1, podemos representar $\frac{2^n}{2}-1$ números positivos e $\frac{2^n}{2}-1$ números negativos. Neste caso, os números positivos e os números negativos podem ser diferenciados através do valor do dígito mais significativo (0 para números positivos e 1 para números negativos).



- num sistema b, diferente de 2, se quisermos representar a máxima quantidade de números positivos e negativos, temos que associar mais dígitos, além de 0 e 1, ao sinal positivo e ao sinal negativo. A interpretação pode ser mais complexa.



Exemplo 7.7 *A tabela seguinte apresenta as três formas de representar um número na base decimal com uso de 2 dígitos:*

Representação	*Valor Representado*		
	Sinal-e-magnitude	*Complemento de 9*	*Complemento de 10*
09	*+9*	*+9*	*+9*
08	*+8*	*+8*	*+8*
07	*+7*	*+7*	*+7*
06	*+6*	*+6*	*+6*
05	*+5*	*+5*	*+5*
04	*+4*	*+4*	*+4*
03	*+3*	*+3*	*+3*
02	*+2*	*+2*	*+2*
01	*+1*	*+1*	*+1*
00	*+0*	*+0*	*+0*
90	*-0*	*-9*	*-10*
91	*-1*	*-8*	*-9*
92	*-2*	*-7*	*-8*
93	*-3*	*-6*	*-7*
94	*-4*	*-5*	*-6*
95	*-5*	*-4*	*-5*
96	*-6*	*-3*	*-4*
97	*-7*	*-2*	*-3*
98	*-8*	*-1*	*-2*
99	*-9*	*-0*	*-1*

Exemplo 7.8 *A tabela seguinte apresenta as três formas de representar*

um número na base binária com uso de 4 dígitos:

Representação	*Valor Representado*		
	Sinal-e-magnitude	*Complemento de 1*	*Complemento de 2*
0111	*+7*	*+7*	*+7*
0110	*+6*	*+6*	*+6*
0101	*+5*	*+5*	*+5*
0100	*+4*	*+4*	*+4*
0011	*+3*	*+3*	*+3*
0010	*+2*	*+2*	*+2*
0001	*+1*	*+1*	*+1*
0000	*+0*	*+0*	*+0*
1000	*-0*	*-7*	*-(8)*
1001	*-1*	*-6*	*-7*
1010	*-2*	*-5*	*-6*
1011	*-3*	*-4*	*-5*
1100	*-4*	*-3*	*-4*
1101	*-5*	*-2*	*-3*
1110	*-6*	*-1*	*-2*
1111	*-7*	*-0*	*-1*

Note-se que no complemento de $b-1$, o negativo de um número é obtido complementando-o, dígito a dígito; e no complemento de b, o negativo de um número é o seu complemento de $b-1$ incrementado de 1.

7.2 Adição e Subtração em Complemento

Ao deparar com as representações em complemento, a primeira pergunta que se pode surgir é: qual é a utilidade das representações em complemento (de b ou de $b-1$)? Veremos que elas permitem uniformizar algoritmicamente as operações de adição e de subtração.

No caso do complemento de b numa representação com n dígitos, consideremos $\mathcal{A}-\mathcal{B}$. Se $\mathcal{A}-\mathcal{B}>0$, então

$$(|\mathcal{A}-\mathcal{B}|)mod(b^n) = (\mathcal{A}-\mathcal{B}+b^n)mod(b^n) = (\mathcal{A}+(b^n-\mathcal{B}))mod(b^n) > 0.$$

Ou seja, **a subtração de duas magnitudes pode ser expressa como a soma das suas representações em complemento de** b.

E se $\mathcal{A}-\mathcal{B}<0$, então

$$(|\mathcal{A}-\mathcal{B}|)mod(b^n) = (-\mathcal{A}+\mathcal{B}+b^n)mod(b^n) = (b^n-(\mathcal{A}-\mathcal{B})))mod(b^n) > 0.$$

Ou seja, **o resultado negativo é dado em complemento de** b.

Exemplo 7.9 *Sejam as seguintes adições em complemento 10 com 3 dígitos (consideremos aqui que o número é negativo quando o dígito mais significativo é 9 e positivo, quando ele é 0):*

045	*(+45)*	*045*	*(+45)*	*970*	*(-30)*	*930*	*(-70)*
020	*(+20)*	*980*	*(-20)*	*960*	*(-40)*	*042*	*(+42)*
065	*(+65)*	*(1) 025*	*(+25)*	*(1) 930*	*(-70)*	*972*	*(-28)*

Em relação ao complemento de $b-1$, a análise é análoga, com a ressalva de que nesta representação os números negativos correspondem a complementos de b^n-1 e não de b^n, porisso, se o resultado da soma

$$(\mathcal{A}+((b^n-1)-\mathcal{B}))mod(b^n-1) = (\mathcal{A}-\mathcal{B})mod(b^n-1)$$

for positivo, ele deve ser adicionado do "vai-um" para obtermos o resultado correto.

Exemplo 7.10 *Sejam as seguintes adições em complemento 9 com 3 dígitos:*

		045	*(+45)*	*969*	*(-30)*		
045	*(+45)*	*979*	*(-20)*	*959*	*(-40)*	*929*	*(-70)*
020	*(+20)*	*(1) 024*		*(1) 928*		*042*	*(+42)*
065	*(+65)*	*1*		*01*		*971*	*(-28)*
		025	*(+25)*	*(1) 929*	*(-70)*		

Vale observar aqui que sendo o universo dos números representáveis estabelecidos a priori nas representações em complemento, a adição de dois números de mesmo sinal pode gerar valores muito grandes (números positivos) ou valores muito pequenos (números negativos) que caiam fora o "universo" definido. Neste caso dizemos que ocorreu **overflow** nas operações e o resultado obtido não é mais válido.

Exemplo 7.11 *Considere que o universo dos números válidos seja* $\{-49,-48,\cdots,+48,+49\}$. *A adição dos seguintes números com 2 dígitos representados em complemento de 9 resultará em números que não pertencem ao universo especificado.*

45	*(+45)*		*48*	*(+48)*			*78*	*(-21)*		*54*	*(-45)*
35	*(+35)*		*48*	*(+48)*			*51*	*(-48)*		*54*	*(-45)*
80	*(?)*		*96*	*(?)*		*(1)*	*29*		*(1)*	*08*	
							1			*1*	
							30	*(?)*		*09*	*(?)*

Uma regra prática para decidir a ocorrência de *overflow* **num sistema binário**, para os quais os números positivos e negativos se diferem no dígito mais significativo (1 para negativos e 0 para positivos), é testar o dígito de sinal dos operandos e do resultado. Se os operandos tiverem o mesmo sinal, o resultado deve ter o mesmo sinal dos operandos; senão, o resultado não é válido.

7.3 Operações Binárias

As operações entre os números em qualquer base são efetuadas de forma similar às operações entre os números na base decimal.

Em se tratando de números no sistema binário, as tabelas-verdade das operações são muito simples, pois envolvem somente dois valores, 0 e 1.

1. Adição

Soma	Transporte
0+0 = 0	0
0+1 = 1	0
1+0 = 1	0
1+1 = 0	1

2. Subtração

Soma	Empréstimo
0-0 = 0	0
0-1 = 1	1
1-0 = 1	0
1-1 = 0	0

3. Multiplicação

Multiplicação
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

4. Divisão

Divisão
0 / 1 = 0
1 / 1 = 1

7.3.1 Adição e Subtração

Vimos que com representações em complemento de 1 ou de 2 podemos tratar uma subtração como uma adição.

No caso da adição, alinha-se os dígitos em relação à vírgula e efetua-se a soma, dígito a dígito, como ilustra a seguinte adição binária

Exemplo 7.12 *Considere a seguinte adição binária:*

Transporte		*1*	*0*	*0*	*1*	*1*		*1*	*1*
			1	*0*	*0*	*1*	*,*	*0*	*1*	*1*
	+		*1*	*1*	*0*	*1*	*,*	*1*	*0*	*1*
		1	*0*	*1*	*1*	*1*	*,*	*0*	*0*	*0*

Note-se que o transporte ("vai-um") de cada estágio é adicionado aos dígitos do próximo estágio.

7.3.2 Multiplicação

O processo de multiplicação feito a lápis é muito similar à multiplicação na base decimal.

Exemplo 7.13 *Considere a seguinte multiplicação binária:*

		1	*1*	*0*	*,1*	*0*	*Multiplicando*
				1	*0*	*,1*	*Multiplicador*
		1	*1*	*0*	*1*	*0*	*Produto Parcial*
	0	*0*	*0*	*0*	*0*		*Produto Parcial*
	0	*1*	*1*	*0*	*1*	*0*	*Primeira Soma Parcial*
1	*1*	*0*	*1*	*0*			*Produto Parcial*
1	*0*	*0*	*0*	*0*	*,0*	*1*	*0*	*Segunda Soma Parcial = Resultado*

Note-se que as operações básicas na multiplicação entre os valores absolutos de dois números consistem em:

1. **deslocar** o multiplicando e
2. **adicionar** o multiplicando à soma parcial dos estágios anteriores, quando o dígito do multiplicador for igual a 1.

O sinal do produto é determinado pelo sinal dos operandos. Se forem diferentes, o resultado é negativo; e se forem iguais, o resultado é positivo.

Vale, porém, ressaltar aqui que existem outros esquemas de multiplicação binária mais rápida, como o algoritmo de Booth, que faz uso de representações em complemento.

7.3.3 Divisão

Dois esquemas mais conhecidos para a divisão binária são:

- com restauração e
- sem restauração.

No primeiro esquema, o divisor é subtraído do dividendo com os seus dígitos mais esquerdos alinhados. Se o resultado for positivo, "1" entra como um dígito no lado direito do quociente; senão, "0" entra como um dígito no lado direito do quociente e restauramos o valor do dividendo. A seguir, agregamos o próximo dígito do dividendo para formar o novo dividendo parcial. E assim sucessivamente até esgotar todos os dígitos do dividendo.

Exemplo 7.14 *Consideremos a divisão de* $(1000)_2$ *por* $(11)_2$ *pelo esquema com restauração:*

Quociente		*0*	*1*	*0*	,	*1*
Dividendo	*1*	*0*	*0*	*0*		
Divisor alinhado	- *1*	*1*				
Resultado Negativo	- *0*	*1*				
Restaura	+ *1*	*1*				
Dividendo Parcial	*1*	*0*	*0*			
	- *0*	*1*	*1*			
Resultado Positivo	*0*	*0*	*1*			
Dividendo Parcial	*0*	*0*	*1*	*0*		
	- *0*	*0*	*1*	*1*		
Resultado Negativo	- *0*	*0*	*0*	*1*		
Restaura	+ *0*	*0*	*1*	*1*		
Dividendo Parcial	*0*	*0*	*1*	*0*		*0*
	- *0*	*0*	*0*	*1*		*1*
Resultado Positivo=Resto	*0*	*0*	*0*			*1*

No esquema sem restauração, substitui-se as etapas (restaura o divisor Y e subtrai o divisor deslocado) pela adição do divisor deslocado, pois $X+Y-\frac{1}{2}Y = X+\frac{1}{2}Y$.

Exemplo 7.15 *Consideremos a divisão de* $(1000)_2$ *por* $(11)_2$ *pelo esquema sem restauração:*

Quociente		*0*	*1*	*0*	,	*1*
Dividendo	*1*	*0*	*0*	*0*		
Divisor alinhado	- *1*	*1*				
Resultado Negativo	- *0*	*1*				
Agrega	- *0*	*1*	*0*			
Divisor deslocado	+ *0*	*1*	*1*			
Resultado Positivo	*0*	*0*	*1*			
Dividendo Parcial	*0*	*0*	*1*	*0*		
	- *0*	*0*	*1*	*1*		
Resultado Negativo	- *0*	*0*	*0*	*1*		
Agrega	- *0*	*0*	*0*	*1*		*0*
Divisor deslocado	+ *0*	*0*	*0*	*1*		*1*
Resultado Positivo=Resto	*0*	*0*	*0*			*1*

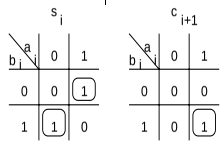
A partir dos dois esquemas, pode-se concluir que uma divisão pode ser realizada através de uma sequência de subtrações, adição e deslocamente.

Ressaltamos, novamente, que existem outros esquemas mais sofisticados e rápidos para efetuar a divisão binária, que fogem do escopo desta disciplina.

7.4 Projeto de Somadores de Um Dígito

Para projetar um circuito **somador** de dois dígitos binários a_i e b_i, considerando o transporte $c_i = 0$ (que é o caso da soma dos bits menos significativos), basta definirmos a seguinte tabela onde s_i e c_i correspondem, respectivamente, à soma e ao transporte:

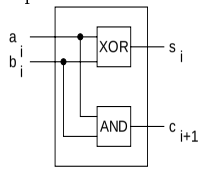
a_i	b_i	s_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



A partir do mapa de Karnaugh é fácil derivar a saída s_i e c_i em função de a_i e b_i

$$\begin{aligned} s_i &= a_i \oplus b_i \\ c_{i+1} &= a_i \cdot b_i \end{aligned}$$

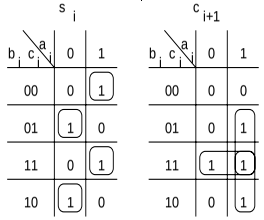
O diagrama lógico correspondente é



A este circuito denominamos **semi-somador**, porque ele não considera o transporte do estágio anterior (bit menos significativo).

Um **somador completo**, por sua vez, tem uma terceira entrada para o transporte do "estágio anterior", ou seja, a sua tabela verdade tem três colunas de entrada

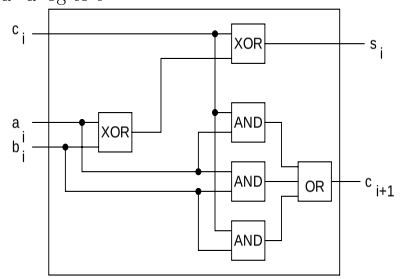
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



A partir do mapa de Karnaugh é fácil derivar a saída s_i e c_i em função de a_i e b_i

$$\begin{aligned} s_i &= a_i b_i' c_i' + a_i' b_i' c_i + a_i b_i c_i + a_i' b_i c_i' \\ &= c_i(a_i' b_i' + a_i b_i) + c_i'(a_i b_i' + a_i' b_i) \\ &= c_i(a_i \oplus b_i)' + c_i'(a_i \oplus b_i) \\ &= c_i \oplus (a_i \oplus b_i) \\ c_{i+1} &= a_i c_i + b_i c_i + a_i b_i \end{aligned}$$

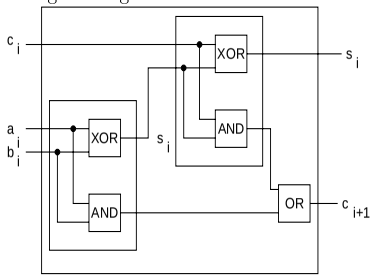
cujo diagrama lógico é



Note-se que podemos ainda expressar c_{i+1} em termos de operações AND e XOR

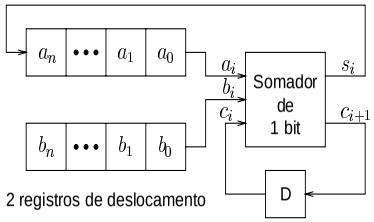
$$c_{i+1} = a_i b_i \not c_i + a_i \not b_i c_i + a_i b_i = c_i(a_i \oplus b_i) + a_i b_i$$

e realizar um somador completo com uso de dois meio-somadores, como ilustra o seguinte diagrama lógico



7.5 Projeto de Somadores de n Dígitos

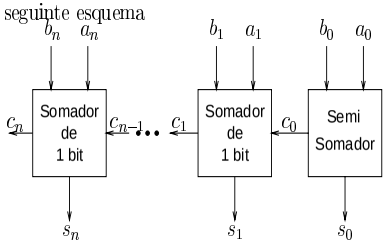
Podemos efetuar adição de dois números com $n+1$ dígitos (binários) com uso de um único somador (completo). O algoritmo é similar ao processo que utilizamos para efeturar manualmente as adições. Adicionamos primeiro os dígitos menos significativos, depois os da segunda coluna e o transporte e assim sucessivamente até chegar nos dígitos mais significativos. Um somador que segue este princípio é conhecido como **somador série**, cujo esquema lógico é



Observe o elemento atrasador (*flip-flop* D) no diagrama. O que aconteceria com o valor do s_i, se o eliminarmos do diagrama?

Embora este somador seja simples em termos de portas lógicas, é fácil ver pelo esquema que para somar n dígitos (binários), n pulsos de relógio são necessários. Ou seja, o tempo de uma adição que envolve uma quantidade de bits muito grande pode ser proibitivamente alto.

Uma outra forma de efetuar a adição binária é utilizar um **somador paralelo**, no qual os n dígitos são adicionado "em paralelo". Isso pode ser conseguido ao ligarmos n somadores completos e 1 semi-somador em cascata, como ilustra o seguinte esquema



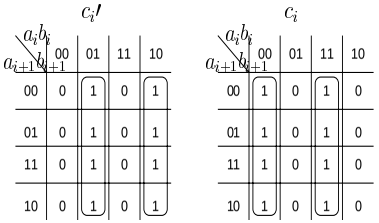
Somadores paralelos que tem este esquema de organização são conhecidos como **somadores paralelos ripple**. Note-se que o transporte c_i é propagado ao longo da cascata e, no pior caso ($1111\cdots1 + 0000\cdots1$), o tempo de propagação do transporte do *bit* menos significativo até o *bit* mais significativo é nt, onde t é o tempo de atraso inerente a cada somador. Portanto, supondo que o tempo de propagação entre os somadores seja desprezível e que o tempo de atraso no semi-somador seja igual ao tempo de atraso no somador completo, no mínimo $(n+1)t$ é necessário para garantir que a saída $s_n \cdots s_2 s_1 s_0$ seja uma soma correta.

Um somador mais rápido pode ser obtido derivando a saída s_i, s_{i+1} e c_{i+2} diretamente das entradas c_i, a_i, b_i, a_{i+1}, b_{i+1} e c_i, cuja tabela-verdade é

a_i	b_i	a_{i+1}	b_{i+1}	c_i=0			c_i=1		
				s_i	s_{i+1}	c_{i+2}	s_i	s_{i+1}	c_{i+2}
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0
0	1	0	0	1	0	0	0	1	0
1	1	0	0	0	1	0	1	1	0
0	0	1	0	0	1	0	1	1	0
1	0	1	0	1	1	0	0	0	1
0	1	1	0	1	1	0	0	0	1
1	1	1	0	0	0	1	1	0	1
0	0	0	1	0	1	0	1	1	0
1	0	0	1	1	1	0	0	0	1
0	1	0	1	1	1	0	0	0	1
1	1	0	1	0	0	1	1	0	1
0	0	1	1	0	0	1	1	0	1
1	0	1	1	1	0	1	0	1	1
0	1	1	1	1	0	1	0	1	1
1	1	1	1	0	1	1	1	1	1

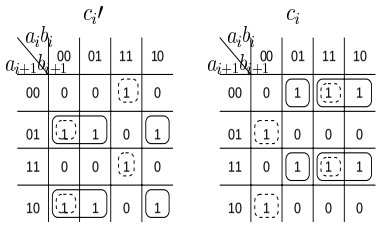
Através do mapa de Karnaugh, podemos derivar as expressões de s_i, s_{i+1} e c_{i+2} em termos de soma de produtos de a_i, b_i, a_{i+1}, b_{i+1} e c_i.

1. $s_i = (a_i\prime b_i + a_i b_i\prime)c_i\prime + (a_i b_i + a_i\prime b_i\prime)c_i$

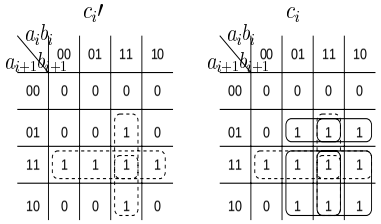


2.

$$\begin{aligned} s_{i+1} &= c_i\prime(a_i\prime a_{i+1}\prime b_{i+1} + a_i\prime a_{i+1} b_{i+1}\prime + a_i b_i\prime a_{i+1}\prime b_{i+1} + a_i b_i\prime a_{i+1} b_{i+1}\prime) + \\ &+ c_i(a_i a_{i+1}\prime b_{i+1}\prime + a_i a_{i+1} b_{i+1} + a_i\prime b_i a_{i+1}\prime b_{i+1}\prime + a_i\prime b_i a_{i+1} b_{i+1}) + \\ &+ a_i b_i a_{i+1}\prime b_{i+1}\prime + a_i\prime b_i\prime a_{i+1}\prime b_{i+1} + a_i b_i a_{i+1} b_{i+1} + a_i\prime b_i\prime a_{i+1} b_{i+1}\prime \end{aligned}$$



3. $c_{i+2} = a_{i+1}b_{i+1} + a_i b_i a_{i+1} + a_i b_i b_{i+1} + b_i a_{i+1}\prime b_{i+1} c_i + a_i a_{i+1}\prime b_{i+1} + a_i a_{i+1} c_i + b_i a_{i+1} c_i$



Embora a partir destas expressões possamos facilmente implementar um circuito com dois níveis lógicos (AND e OR), a quantidade de termos em cada produto e na soma cresce exponencialmente com o número de entradas. Portanto, esta alternativa se limita a somadores com um número pequeno de entradas.

Uma outra alternativa para acelerar a soma entre os dois números é o uso do **transporte antecipado** (*carry look-ahead*). A idéia básica da determinação do transporte "antecipadamente" se baseia no fato de que

- para uma particular combinação de entradas a_i e b_i, o estágio i **gera** um transporte se ele produz um transporte independentemente dos estágios anteriores, ou seja, $a_i b_i = 1$, e
- para uma particular combinação de entradas a_i e b_i, o estágio i **propaga** um transporte se uma das entradas for igual a 1 e os estágios anteriores produzirem um transporte igual a 1.

Assim, para cada estágio $i+1$, temos um transporte

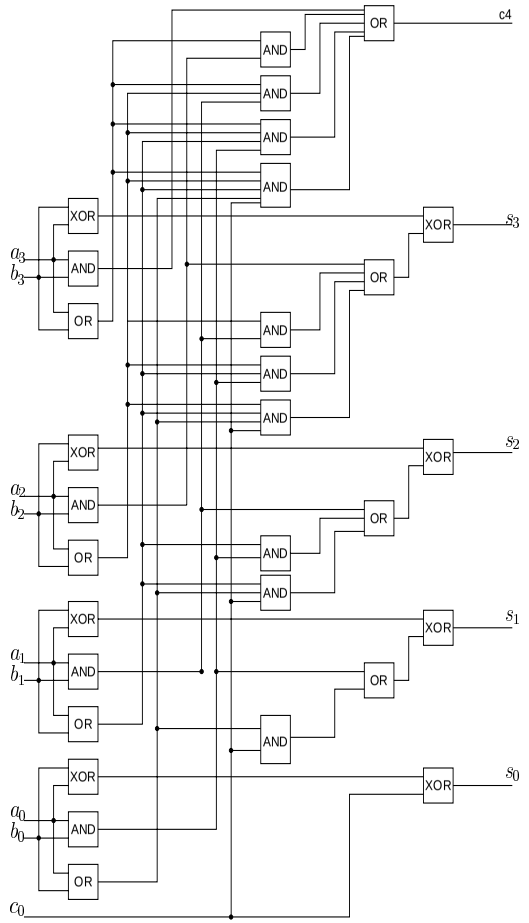
$$c_{i+1} = g_i + p_i \cdot c_i = (a_i \cdot b_i) + ((a_i + b_i) \cdot c_i),$$

que pode ser obtido de forma recursiva. Cada equação só envolve três níveis lógicos: g_i (transporte gerado) ou p_i (transporte propagado), produto dos termos e soma dos produtos.

Exemplo 7.16 *Seja um somador com 4 dígitos de entrada, os transportes* c_4, c_3, $_2$ *e* c_1 *podem ser obtidos de forma recorrente:*

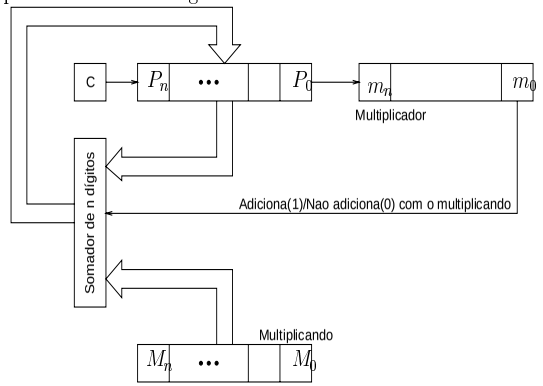
$$\begin{aligned}
c_1 &= g_0 + p_0 \cdot c_0 \\
c_2 &= g_1 + p_1 \cdot c_1 \\
&= g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0) \\
&= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0 \\
c_3 &= g_2 + p_2 \cdot c_2 \\
&= g_2 + p_2 \cdot (g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0) \\
&= g_2 + p_2 \cdot g_1 + p_2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

Uma implementação deste circuito com transporte antecipado seria



7.6 Projeto de Multiplicadores e Divisores Binários

Por completude, apresentearemos nesta seção um esquema de um circuito de multiplicador com uso de registros de deslocamento e somador.



O circuito de um divisor é mais complexo, pois além dos registros de deslocamento e do somador/subtrador para obter os dividendos "parciais", como ilustra na seguinte figura, uma lógica adicional é necessária (1) para decidir a partir do sinal do resultado de uma subtração o dígito a ser acumulado no quociente e (2) para controlar o deslocamento dos dígitos no registro do resultado parcial (A) e do dividendo/quociente.