CAPÍTULO 2

SISTEMAS DE NUMERAÇÃO E CÓDIGOS

CONTEÚDO

OBJETIVOS

Após estudar este capítulo, você será capaz de:

- Converter um número de um sistema de numeração (decimal, binário ou hexadecimal) para o equivalente em qualquer outro sistema de numeração.
- Citar vantagens do sistema de numeração hexadecimal.
- Contar em hexadecimal.
- Representar números decimais usando o código BCD; citar os prós e os contras no uso do código BCD.
- Explicar a diferença entre BCD e binário puro.
- Explicar o propósito dos códigos alfanuméricos, como o código ASCII.
- Explicar o método de paridade para detecção de erro.
- Determinar o bit de paridade a ser acrescentado a uma sequência de dados.

INTRODUÇÃO

O sistema de numeração binário é o mais importante em sistemas digitais, mas há outros também importantes. O sistema decimal é importante por ser universalmente usado para representar quantidades fora do sistema digital. Isso significa que há situações em que os valores decimais têm de ser convertidos em binários antes de entrar em um sistema digital. Por exemplo, quando você digita um número decimal em sua calculadora (ou computador), o circuito interno dessas máquinas converte o número decimal em um valor binário.

Da mesma maneira, há casos em que os valores binários das saídas de um sistema digital têm de ser convertidos em decimais para serem apresentados ao mundo externo. Por exemplo, sua calculadora (ou computador) usa números binários para calcular as respostas de um problema e, então, converte-as para valores decimais antes de apresentá-las.

Como veremos, não é fácil apenas olhar para um longo número binário e convertê-lo em seu valor decimal equivalente. É muito cansativo digitar uma longa sequência de 1s e 0s em um teclado numérico ou escrever longos números

binários no papel. É especialmente difícil tentar transmitir uma quantidade binária quando se está falando com alguém. O sistema de numeração de base hexadecimal (base 16) se tornou a maneira padrão de comunicar valores numéricos em sistemas digitais. A grande vantagem é que os números hexadecimais podem ser facilmente convertidos para o sistema binário e vice-versa. Veremos que muitas ferramentas de computador avançadas, que são projetadas para ajudar criadores de softwares a lidar com problemas ou vírus em seus programas, usam o sistema de numeração hexadecimal para inserir números armazenados no computador como binários e exibi-los novamente como hexadecimais.

Outras maneiras de representar quantidades decimais com dígitos codificados em binário foram inventadas. Apesar de não serem realmente sistemas de numeração, facilitam a conversão entre o código binário e o sistema de numeração decimal. Esses códigos costumam ser chamados de decimal codificado em binário (BCD). Quantidades e padrões de bits podem ser representados por quaisquer desses métodos em qualquer sistema dado e em todo o material escrito referente ao sistema digital. Por isso, é muito importante que você seja capaz de interpretar valores em qualquer sistema e efetuar conversões entre quaisquer dessas representações numéricas. Outros códigos que usam 1s e 0s para representar elementos como caracteres alfanuméricos serão estudados, por serem bastante comuns em sistemas digitais.

2.1 CONVERSÕES DE BINÁRIO PARA DECIMAL

Conforme explicado no Capítulo 1, o sistema de numeração binário é um sistema posicional em que cada dígito binário (bit) possui um certo peso, de acordo com a posição relativa ao LSB. Qualquer número binário pode ser convertido em seu decimal equivalente, simplesmente somando os pesos das posições em que o número binário tiver um bit 1. Para ilustrar, vamos converter 11011_2 em seu equivalente decimal.

$$\begin{array}{cccccccccccc} 1 & & 1 & & 0 & & 1 & & 1_2 & & & \\ 2^4 & + & 2^3 & + & 0 & + & 2^1 & + & 2^0 & = & 16 + 8 + 2 + 1 \\ & & & & & & & & & = & 27_{10} \end{array}$$

Vejamos outro exemplo com um número maior de bits:

$$\begin{array}{cccccccccccccc} 1 & & 0 & & 1 & & 1 & & 0 & & 1 & & 0 & 1_2 = \\ 2^7 & + & 0 & + & 2^5 & + & 2^4 & + & 0 & + & 2^2 & + & 0 & + 2^0 = 181_{10} \end{array}$$

Observe que o procedimento é determinar os pesos (isto é, as potências de 2) para cada posição que contenha um bit 1 e, então, somá-los. Observe também que o MSB tem peso de 2^7 ainda que seja o oitavo bit; isto ocorre porque o LSB é o primeiro bit e tem peso de 2^0.

Outro método de conversão de binário para decimal que evita a soma de números grandes e o acompanhamento dos pesos das colunas é chamado de método *double-dabble*. O procedimento é o seguinte:

1. Escreva, de modo inverso, o 1 mais à esquerda no número binário.
2. Dobre-o e some o bit a seguir à direita.
3. Escreva, de modo inverso, o resultado sob o próximo bit.
4. Continue com os passos 2 e 3 até terminar com o número binário.

Vamos usar os mesmos números binários para verificar este método.

$$\begin{array}{lccccc} \text{Dado:} & 1 & 1 & 0 & 1 & 1_2 \\ \text{Resultados:} & 1 \times 2 = 2 & & & & \\ & \underline{+\,1} & & & & \\ & 3 \times 2 = 6 & & & & \\ & & \underline{+\,0} & & & \\ & & 6 \times 2 = 12 & & & \\ & & & \underline{+\,1} & & \\ & & & 13 \times 2 = 26 & & \\ & & & & \underline{+\,1} & \\ & & & & 27_{10} & \end{array}$$

$$\begin{array}{lcccccccc} \text{Dado:} & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1_2 \\ \text{Resultados:} & 1 \rightarrow & 2 \rightarrow & 5 \rightarrow & 11 \rightarrow & 22 \rightarrow & 45 \rightarrow & 90 \rightarrow & \mathbf{181}_{10} \end{array}$$

Questões para revisão

1. Converta o binário 100011011011_2 em seu equivalente decimal somando os produtos dos dígitos e pesos.
2. Qual é o peso do MSB de um número de 16 bits?
3. Repita a conversão na Questão 1 usando o método *double-dabble*.

2.2 CONVERSÕES DE DECIMAL PARA BINÁRIO

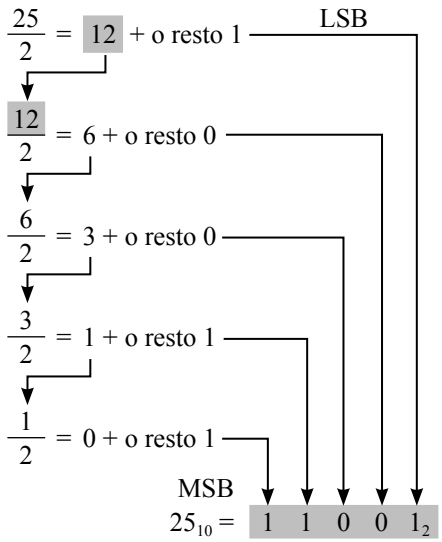
No sistema binário, há duas maneiras de converter um número decimal *inteiro* em seu equivalente. O primeiro método é o processo inverso descrito na Seção 2.1. O número decimal é simplesmente expresso como uma soma de potências de 2, e, então, 1s e 0s são colocados nas posições corretas dos bits. Para ilustrar:

$$\begin{aligned} 45_{10} = 32 + 8 + 4 + 1 &= 2^5 + 0 + 2^3 + 2^2 + 0 + 2^0 \\ &= 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1_2 \end{aligned}$$

Observe que um 0 é colocado nas posições 2^1 e 2^4, visto que todas as posições têm de ser consideradas. Outro exemplo é o seguinte:

$$\begin{aligned} 76_{10} = 64 + 8 + 4 &= 2^6 + 0 + 0 + 2^3 + 2^2 + 0 + 0 \\ &= 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0_2 \end{aligned}$$

Outro método para converter um número decimal inteiro utiliza divisões sucessivas por 2. A conversão, ilustrada a seguir para o número 25_{10}, requer divisões sucessivas pelo número decimal 2 e a escrita, de modo inverso, dos restos de cada divisão, até que um quociente 0 seja obtido. Observe que o resultado binário é alcançado escrevendo-se o primeiro resto na posição do LSB e o último na posição do MSB. Esse processo, representado pelo fluxograma da Figura 2.1, também pode ser usado para converter um número decimal em qualquer outro sistema de numeração, como é possível ver.



Dica para a calculadora

Ao usar uma calculadora para realizar as divisões por 2, é possível saber se o resto é 0 ou 1 caso o resultado tenha ou não parte fracionária. Por exemplo, 25/2 produziria 12,5. Já que há parte fracionária (0,5), o resto é 1. Se não há parte fracionária, como 12/2 = 6, então o resto seria 0. O Exemplo 2.1 ilustra esta situação.

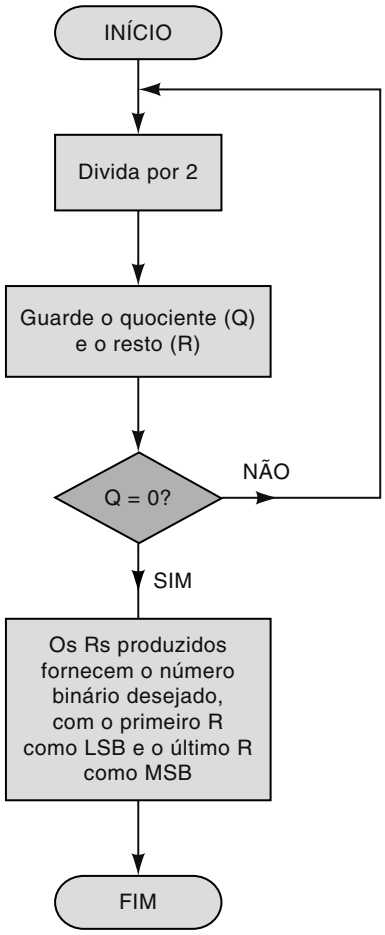


FIGURA 2.1 Fluxograma do método de divisões sucessivas na conversão de decimal (números inteiros) em binário. O mesmo processo pode ser usado para converter um inteiro decimal em qualquer outro sistema de numeração.

Exemplo 2.1

Converta 37_{10} em binário. Tente fazê-lo antes de olhar a solução.

Solução

$$\frac{37}{2} = 18{,}5 \rightarrow \text{o resto } 1 \text{ (LSB)}$$

$$\frac{18}{2} = 9{,}0 \longrightarrow 0$$

$$\frac{9}{2} = 4{,}5 \longrightarrow 1$$

$$\frac{4}{2} = 2{,}0 \longrightarrow 0$$

$$\frac{2}{2} = 1{,}0 \longrightarrow 0$$

$$\frac{1}{2} = 0{,}5 \longrightarrow 1 \text{ (MSB)}$$

Assim, $37^{10} = \mathbf{100101}_2$.

Faixa de contagem

Lembre-se de que usando N bits, podemos contar 2^N diferentes números em decimal (de 0 a $2^N - 1$). Por exemplo, para $N = 4$, podemos contar de 0000_2 a 1111_2, que corresponde a 0_{10} a 15_{10}, em um total de 16 números diferentes. Nesse caso, o valor do maior número decimal é $2^4 - 1 = 15$, e há 2^4 números diferentes.

Portanto, geralmente, podemos dizer:

Usando *N* bits, podemos representar números decimais na faixa de 0 a $2^N - 1$, em um total de 2^N números diferentes.

Exemplo 2.2

(a) Qual é a faixa total de valores decimais que podemos representar com 8 bits?
(b) Quantos bits são necessários para representar valores decimais na faixa de 0 a 12.500?

Solução

(a) Nesse caso, temos $N = 8$. Assim, podemos representar os números decimais na faixa de 0 a $2^8 - 1 = 255$. Podemos comprovar isso verificando que 11111111_2, convertido em decimal, vale 255_{10}.
(b) Usando 13 bits, podemos contar, em decimal, de 0 a $2^{13} - 1 = 8191$. Usando 14 bits, podemos contar de 0 a $2^{14} - 1 = 16.383$. Evidentemente, 13 bits não são suficientes, porém com 14 bits podemos ir além de 12.500. Assim, o número necessário de bits é **14**.

Questões para revisão

1. Converta 83_{10} em binário usando os dois métodos apresentados.
2. Converta 729_{10} em binário usando os dois métodos apresentados. Verifique sua resposta, fazendo a conversão de volta para decimal.
3. Quantos bits são necessários para contar até 1 milhão em decimal?

2.3 SISTEMA DE NUMERAÇÃO HEXADECIMAL

O **sistema de numeração hexadecimal** usa a base 16. Assim, ele tem 16 símbolos possíveis para os dígitos. Utiliza os dígitos de 0 a 9 mais as letras A, B, C, D, E e F como símbolos. As posições dos dígitos recebem pesos como potências de 16, como mostrado a seguir, em vez de usar as potências de 10 como no sistema decimal.

16^4	16^3	16^2	16^1	16^0 ,	16^{-1}	16^{-2}	16^{-3}	16^{-4}

Vírgula hexadecimal

A Tabela 2.1 mostra as relações entre hexadecimal, decimal e binário. Observe que cada dígito hexadecimal é representado por um grupo de quatro dígitos binários. É importante lembrar que os dígitos hexa (abreviação para 'hexadecimal'), de A até F, são equivalentes aos valores decimais de 10 até 15.

TABELA 2.1

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

Conversão de hexa em decimal

Um número hexa pode ser convertido em seu equivalente decimal pelo fato da posição de cada dígito hexa ter um peso que é uma potência de 16. O LSD tem um peso de $16^0 = 1$; o dígito da próxima posição superior tem um peso de $16^1 = 16$; o próximo tem um peso de $16^2 = 256$, e assim por diante. O processo de conversão é demonstrado nos exemplos a seguir.

Dica para a calculadora

Você pode usar a função y^x da calculadora para calcular as potências de 16.

$$\begin{aligned} 356_{16} &= 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0 \\ &= 768 + 80 + 6 \\ &= 854_{10} \end{aligned}$$

$$\begin{aligned} 2AF_{16} &= 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 \\ &= 512 + 160 + 15 \\ &= 687_{10} \end{aligned}$$

Observe que no segundo exemplo o valor 10 foi substituído por A e o valor 15 por F na conversão para decimal. Para praticar, comprove que $1BC2_{16}$ é igual a 7106_{10}.

Conversão de decimal em hexa

Lembre-se de que fizemos a conversão de decimal em binário usando divisões sucessivas por 2. Da mesma maneira, a conversão de decimal em hexa pode ser feita usando divisões sucessivas por 16 (Figura 2.1). O exemplo a seguir apresenta duas ilustrações desta conversão.

Exemplo 2.3

(a) Converta 423_{10} em hexa.

Solução

$$\frac{423}{16} = 26 + \text{o resto } 7$$

$$\frac{26}{16} = 1 + \text{o resto } 10$$

$$\frac{1}{16} = 0 + \text{o resto } 1$$

$$423_{10} = 1\ A\ 7_{16}$$

(b) Converta 214_{10} em hexa.

Solução

$$\frac{214}{16} = 13 + \text{o resto } 6$$

$$\frac{13}{16} = 0 + \text{o resto } 13$$

$$214_{10} = D\ 6_{16}$$

Observe novamente que os restos do processo de divisões sucessivas formam os dígitos do número hexa. Note, também, que quaisquer restos maiores que 9 são representados pelas letras de A até F.

Dica para a calculadora

Se uma calculadora for usada para calcular as divisões no processo de conversão, o resultado incluirá uma fração decimal em vez de um resto. Este pode ser obtido multiplicando-se a fração por 16. Para ilustrar, no Exemplo 2.3(b), a calculadora teria efetuado

$$\frac{214}{16} = 13{,}375$$

O resto é (0,375) × 16 = 6.

Conversão de hexa em binário

O sistema de numeração hexadecimal é usado principalmente como método 'taquigráfico' (compacto) para representar um número binário. A conversão de hexa em binário é relativamente simples. *Cada* dígito hexa é convertido no equivalente binário de 4 bits (Tabela 2.1). Isto está ilustrado a seguir para $9F2_{16}$.

$$\begin{aligned} 9F2_{16} &= \quad 9 \qquad\quad F \qquad\quad 2 \\ &\qquad\quad \downarrow \qquad\quad \downarrow \qquad\quad \downarrow \\ &= 1001 \quad 1111 \quad 0010 \\ &= 100111110010_2 \end{aligned}$$

Para praticar, comprove que $BA6_{16} = 101110100110_2$.

Conversão de binário em hexa

A conversão de binário em hexa consiste, simplesmente, em fazer o inverso do processo anterior. O número binário é disposto em grupos de *quatro* bits, e cada grupo é convertido no dígito hexa equivalente. Os zeros (sombreados a seguir) são acrescentados, quando necessário, para completar um grupo de 4 bits.

$$\begin{aligned} 1110100110_2 &= \underbrace{0011}_{3}\ \underbrace{1010}_{A}\ \underbrace{0110}_{6} \\ &= 3A6_{16} \end{aligned}$$

Para realizar as conversões entre hexa e binário, é necessário conhecer os números binários de 4 bits (0000 a 1111) e seus dígitos hexa equivalentes. Uma vez que essa habilidade é adquirida, as conversões podem ser realizadas rapidamente, sem necessidade de qualquer cálculo. É por isso que o sistema hexa é útil na representação de números binários grandes.

Para praticar, comprove que $101011111_2 = 15F_{16}$.

Contagem em hexadecimal

Quando contamos em hexa, cada dígito pode ser incrementado (acrescido de 1) de 0 a F. Quando o dígito de uma posição chega no valor F, este volta para 0, e o dígito da próxima posição é incrementado. Isto está ilustrado nas seguintes sequências de contagem hexa:

(a) 38, 39, 3A, 3B, 3C, 3D, 3E, 3F, 40, 41, 42

(b) 6F8, 6F9, 6FA, 6FB, 6FC, 6FD, 6FE, 6FF, 700

Observe que, quando o dígito de uma posição é 9, ele torna-se A quando é incrementado.

Com N dígitos hexa podemos contar de 0 até o decimal $16^N - 1$, em um total de 16^N valores diferentes. Por exemplo, com três dígitos hexa podemos contar de 000_{16} a FFF_{16}, que corresponde à faixa de 0_{10} a 4095_{10}, em um total de 4096 = 16^3 valores diferentes.

Vantagens do sistema hexa

O sistema hexa costuma ser usado em sistemas digitais como uma espécie de forma 'compacta' de representar sequências de bits. No trabalho com computadores, sequências binárias de até 64 bits não são incomuns. Elas nem sempre representam valores numéricos, mas, como você descobrirá, podem ser algum tipo de código que representa uma informação não numérica. Quando manipulamos números com uma extensa quantidade de bits, é mais conveniente e menos sujeito a erros escrevê-los em hexa; assim, como já vimos, é relativamente fácil realizar conversões mútuas entre binário

e hexa. Para ilustrar a vantagem da representação em hexa de uma sequência binária, suponha que você tenha uma lista impressa com o conteúdo de 50 posições de memória, tendo cada uma números de 16 bits, e que precise conferi-los de acordo com outra lista. Você preferirá conferir 50 números do tipo 0110111001100111 ou 50 números do tipo 6E67? Em qual dos dois casos você faria, mais provavelmente, uma leitura incorreta? De qualquer modo, é importante ter em mente que os circuitos digitais trabalham com binários. O sistema hexa é usado simplesmente por uma questão de conveniência. Você deve memorizar o padrão binário de 4 bits para cada dígito hexadecimal. Só então você perceberá a utilidade desse recurso para os sistemas digitais.

Exemplo 2.4

Converta o decimal 378 em um número binário de 16 bits, mudando primeiro para hexadecimal.

Solução

$$\frac{378}{16} = 23 + \text{o resto de } 10_{10} = A_{16}$$

$$\frac{23}{16} = 1 + \text{o resto de } 7$$

$$\frac{1}{16} = 0 + \text{o resto de } 1$$

Assim, $378_{10} = 17A_{16}$. Esse valor hexa pode facilmente ser convertido no binário 000101111010. Finalmente, podemos expressar 378_{10} como um número de 16 bits, acrescentando 4 bits zero à esquerda:

$$378_{10} = 0000\ \ 0001\ \ 0111\ \ 1010_2$$

Exemplo 2.5

Converta $B2F_{16}$ em decimal.

Solução

$$\begin{aligned} B2F_{16} &= B \times 16^2 + 2 \times 16^1 + F \times 16^0 \\ &= 11 \times 256 + 2 \times 16 + 15 \\ &= 2863_{10} \end{aligned}$$

Resumo sobre as conversões

Neste momento, você já deve estar pensando em como guardar corretamente as diferentes conversões de um sistema de numeração para outro. Provavelmente, fará com que muitas dessas conversões sejam *automaticamente* efetuadas em sua calculadora apenas pressionando uma tecla, mas é importante dominar essas conversões para compreender o processo. Além disso, o que você fará se a bateria da calculadora estiver descarregada em um momento crucial e você não tiver outra à mão para substituí-la? O resumo a seguir pode ajudá-lo, porém, não substituirá a habilidade obtida com a prática.

1. Quando converter o binário ou hexa em decimal, use o método da soma dos pesos de cada dígito ou siga o procedimento *double-dabble*.
2. Quando converter o decimal em binário ou hexa, use o método de divisões sucessivas por 2 (binário) ou 16 (hexa), reunindo os restos da divisão (Figura 2.1).
3. Quando converter o binário em hexa, agrupe os bits em grupos de quatro e converta cada grupo no dígito hexa equivalente.
4. Quando converter o hexa em binário, converta cada dígito em 4 bits equivalentes.

Questões para revisão

1. Converta $24CE_{16}$ em decimal.
2. Converta 3117_{10} em hexa e, em seguida, em binário.

3. Converta 1001011110110101_2 em hexa.
4. Escreva os próximos quatro números da seguinte contagem hexa: E9A, E9B, E9C, E9D, ___, ___, ___, ___.
5. Converta 3527 em binário$_{16}$.
6. Que faixa de valores decimais pode ser representada por números hexa de quatro dígitos?

2.4 CÓDIGO BCD

Quando números, letras ou palavras são representados por um grupo especial de símbolos, dizemos que estão codificados, sendo denominado *código*. Provavelmente o código mais familiar é o Morse, em que uma série de pontos e traços representa letras do alfabeto.

Vimos que qualquer número decimal pode ser representado por um binário equivalente. Os grupos de 0s e 1s em um número binário podem ser usados como representação codificada de um número decimal. Quando um número decimal é representado por seu número binário equivalente, dizemos que é uma **codificação em binário puro**.

Todos os sistemas digitais usam algum modo de numeração binária em suas operações internas; porém, o mundo externo é naturalmente decimal. Isto significa que conversões entre os sistemas decimal e binário são realizadas frequentemente. Vimos que conversões entre decimal e binário podem se tornar longas e complicadas para números grandes. Por isso, uma maneira de codificar números decimais que combine algumas características dos dois sistemas, binário e decimal, é usada em determinadas situações.

Decimal codificado em binário

Se *cada* dígito de um número decimal for representado por seu equivalente em binário, o resultado será um código denominado **decimal codificado em binário** (daqui em diante abreviado por BCD — *binary-coded-decimal*). Como um dígito decimal pode ter no máximo o valor 9, são necessários 4 bits para codificar cada dígito (o código binário do 9 é 1001).

Para ilustrar o uso do código BCD, pegue um número decimal, por exemplo, 874. Cada *dígito* é convertido no equivalente binário, como mostrado a seguir:

8	7	4	(decimal)
↓	↓	↓	
1000	0111	0100	(BCD)

Exemplificando novamente, vamos converter 943 em código BCD:

9	4	3	(decimal)
↓	↓	↓	
1001	0100	0011	(BCD)

Novamente, cada dígito decimal é convertido no equivalente binário puro. Observe que *sempre* são utilizados 4 bits para cada dígito.

O código BCD representa, então, cada dígito de um número decimal por um número binário de 4 bits. Evidentemente, são usados apenas os números binários de 4 bits, entre 0000 e 1001. O código BCD não usa os números 1010, 1011, 1100, 1101, 1110 e 1111. Em outras palavras, são usados apenas 10 dos 16 possíveis grupos de 4 bits. Se qualquer um desses números de 4 bits 'proibidos' aparecer alguma vez em uma máquina que use o código BCD, é, geralmente, uma indicação de que ocorreu algum erro.

Exemplo 2.6

Converta 0110100000111001 (BCD) em seu equivalente decimal.

Solução
Separe o número BCD em grupos de 4 bits e converta cada grupo em decimal.

$$\underbrace{0110}_{6}\ \underbrace{1000}_{8}\ \underbrace{0011}_{3}\ \underbrace{1001}_{9}$$

Exemplo 2.7

Converta o número BCD 011111000001 em seu equivalente decimal.

Solução

$$\underbrace{0111}_{7}\ \underbrace{1100}_{\downarrow}\ \underbrace{0001}_{1}$$

O grupo referente a um código proibido indica um erro no número BCD.

Comparação entre BCD e binário

É importante perceber que o BCD não é outro sistema de numeração, como os sistemas binário, decimal e hexadecimal. Na realidade, o BCD é um sistema decimal no qual cada dígito é codificado em seu equivalente binário. Além disso, é importante entender que um número BCD *não* é o mesmo que um número binário puro. O código binário puro é obtido a partir do número decimal *completo* que é representado em binário; no código BCD, *cada dígito* decimal é convertido, individualmente, em binário. Para ilustrar, veja como exemplo o número 137, comparando os códigos BCD e binário puro:

$$137_{10} = 10001001_2 \quad \text{(binário)}$$
$$137_{10} = 0001\ 0011\ 0111 \quad \text{(BCD)}$$

O código BCD requer 12 bits; o código binário puro, apenas 8 bits para representar o decimal 137. O código BCD requer mais bits que o binário puro para representar os números decimais maiores que um dígito. Isto acontece porque o código BCD não usa todos os grupos de 4 bits possíveis, conforme demonstrado antes; por isso, é um pouco ineficiente.

A principal vantagem do código BCD é a relativa facilidade de conversão em decimal e vice-versa. Apenas os grupos de 4 bits dos dígitos de 0 a 9 precisam ser memorizados. Essa característica de fácil conversão é especialmente importante do ponto de vista do hardware, porque nos sistemas digitais são os circuitos lógicos que realizam as conversões mútuas entre BCD e decimal.

Exemplo 2.8

Um caixa automático de banco permite que você indique o montante de dinheiro que quer retirar em decimal ao pressionar teclas de dígitos decimais. O computador converte esse número em binário puro ou BCD? Explique.

Solução

O número que representa o saldo (o dinheiro que você tem no banco) está armazenado como um número binário puro. Quando o montante retirado é indicado, ele tem de ser subtraído do saldo. Tendo em vista que a aritmética precisa ser feita nos números, ambos os valores (o saldo e o dinheiro retirado) têm de ser binários puros. Ela converte a entrada decimal em binário puro.

Exemplo 2.9

O telefone celular permite que você tecle/armazene um número de telefone de dígito decimal 10. Ele armazena o número do telefone em binário puro ou BCD? Explique.

Solução

Um número de telefone é uma combinação de muitos dígitos decimais. Não é necessário combinar matematicamente os dígitos (isto é, você nunca soma dois números de telefone juntos). O aparelho só precisa armazená-los na sequência em que foram inseridos e buscá-los quando se pressiona *enviar*. Portanto, serão armazenados como dígitos BCD na memória do computador do celular.

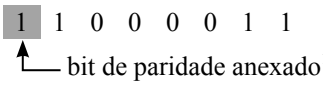
Questões para revisão

1. Represente o valor decimal 178 no equivalente binário puro. Em seguida, codifique o mesmo número decimal usando BCD.
2. Quantos bits são necessários para representar, em BCD, um número decimal de oito dígitos?
3. Qual é a vantagem da codificação em BCD de um número decimal quando comparada com o binário puro? E qual é a desvantagem?

Bit de paridade

Um **bit de paridade** consiste em um bit extra anexado ao conjunto de bits do código a ser transferido de uma localidade para outra. O bit de paridade pode ser 0 ou 1, dependendo do número de 1s contido no conjunto de bits do código. Dois métodos diferentes são usados.

No método que usa *paridade par*, o valor do bit de paridade é determinado para que o número total de 1s no conjunto de bits do código (incluindo o bit de paridade) seja *par*. Por exemplo, suponha que o conjunto de bits seja 1000011. Esse é o código ASCII do caractere 'C'. Esse conjunto de bits tem *três* 1s; portanto, anexamos um bit de paridade par igual a 1 para tornar par o número total de 1s. O *novo* conjunto de bits, *incluindo o bit de paridade*, passa a ser:



Se o grupo de bits do código contiver um número par de 1s, o bit de paridade terá o valor 0. Por exemplo, se o conjunto de bits do código fosse 1000001 (o código ASCII para 'A'), o bit de paridade designado seria o 0, de modo que o novo código, *incluindo o bit de paridade*, seria 01000001.

O método de *paridade ímpar* é usado exatamente da mesma maneira, exceto que o bit de paridade é determinado para que o número total de 1s, incluindo o bit de paridade, seja *ímpar*. Por exemplo, para o grupo de bits 1000001, deve ser 1, e para o grupo de bits 1000011, deve ser 0. Quer a paridade utilizada seja par, quer seja ímpar, o bit de paridade passa a ser parte do código atual da informação. Por exemplo, anexando um bit de paridade ao código ASCII de 7 bits, geramos um código de 8 bits. Assim, o bit de paridade é tratado exatamente como qualquer outro bit do código.

O bit de paridade é gerado para detectar erros de *um só bit* que ocorram durante a transmissão de um código de um local para outro. Por exemplo, suponha que o caractere 'A' seja transmitido e que seja usada a paridade *ímpar*. O código transmitido seria:

1 1 0 0 0 0 0 1

Quando ele chega ao circuito receptor, este verifica se o código contém um número ímpar de 1s (incluindo o bit de paridade). Em caso afirmativo, o receptor considera que o código foi recebido corretamente. No entanto, suponha que, devido a algum ruído ou mau funcionamento do circuito receptor, seja recebido o seguinte código:

1 1 0 0 0 0 0 0

O receptor identificará que o código tem um número *par* de 1s. Isso significa, para o receptor, que há um erro no código, presumindo que transmissor e receptor tenham usado paridade ímpar. Entretanto, não há como o receptor identificar qual bit está errado, visto que ele não sabe qual é o código correto.

É evidente que o método de paridade não funcionará se ocorrer erro em *dois* bits, porque dois bits errados não geram alteração na paridade do código. Na prática, o método de paridade é usado apenas nas situações em que a probabilidade de erro de um único bit é baixa e, em dois bits, é essencialmente zero.

Quando se usa o método de paridade, tem de haver uma concordância entre transmissor e receptor em relação ao tipo de paridade (par ou ímpar) a ser usada. Embora não exista vantagem de um método sobre o outro, a paridade par é mais usada. O transmissor anexa um bit de paridade a cada unidade de informação transmitida. Por exemplo, se o transmissor está enviando um dado codificado em ASCII, ele anexará um bit de paridade a cada conjunto ASCII de 7 bits. Quando o receptor analisar o dado recebido, ele verificará se a quantidade de 1s de cada conjunto de bits (incluindo o bit de paridade) está de acordo com o método de paridade escolhido previamente. Essa operação é frequentemente denominada *verificação de paridade* dos dados. Quando um erro for detectado, o receptor poderá enviar uma mensagem de volta ao transmissor, solicitando a retransmissão do último conjunto de dados. O procedimento a seguir, quando um erro é detectado, depende do tipo de sistema.

Exemplo 2.17

A comunicação entre computadores remotos acontece, muitas vezes, por rede telefônica. Por exemplo, a comunicação pela internet ocorre via rede telefônica. Quando um computador transmite uma mensagem para outro, a informação é, normalmente, codificada em ASCII. Quais seriam as cadeias de caracteres de bits transmitidas por um computador para enviar a mensagem 'HELLO' usando ASCII com paridade par?

1 O bit de paridade pode ser colocado tanto no início quanto no final de um grupo de código; em geral é colocado à esquerda do MSB.

Solução
Primeiro, determine o código ASCII de cada caractere da mensagem. Em seguida, conte o número de 1s de cada código. Se o número de 1s for par, anexe um 0 como o MSB. Caso o número de 1s seja ímpar, anexe um 1. Dessa maneira, os códigos de 8 bits (byte) resultantes terão uma quantidade par de 1s (incluindo o bit de paridade).

		bits de paridade par anexados ↓							
H	=	0	1	0	0	1	0	0	0
E	=	1	1	0	0	0	1	0	1
L	=	1	1	0	0	1	1	0	0
L	=	1	1	0	0	1	1	0	0
O	=	1	1	0	0	1	1	1	1

Correção de erros

A detecção de erros é benéfica, porque o sistema que recebe um dado contendo um erro sabe que recebeu um 'produto danificado'. Não seria ótimo se o receptor pudesse também saber qual bit estava errado? Se um bit binário está errado, então o valor correto é seu complemento. Vários métodos foram desenvolvidos para conseguir isto. Em cada caso, ele exige que vários bits de 'detecção de erro/códigos de correção' sejam aplicados para cada pacote de informação transmitido. À medida que o pacote é recebido, um circuito digital pode detectar se os erros ocorreram (mesmo múltiplos erros) e corrigi-los. Esta tecnologia é usada para transferência maciça de dados em alta velocidade em aplicações como *drives* de discos magnéticos, *flash drives*, CD, DVD, *Blu-Ray Disc*, televisão digital e redes de Internet de banda larga.

Questões para revisão

1. Anexe um bit de paridade ímpar ao código ASCII do símbolo $ e expresse o resultado em hexadecimal.
2. Anexe um bit de paridade par ao código BCD relativo ao decimal 69.
3. Por que o método de paridade não consegue detectar um erro duplo de bit em um dado transmitido?

2.10 APLICAÇÕES

Vejamos algumas aplicações que também servem como revisão de alguns conceitos abordados neste capítulo. Essas aplicações farão você entender como os diversos sistemas de numeração e códigos são usados no mundo digital. Outras aplicações estão presentes nos problemas no final do capítulo.

Aplicação 2.1

Um CD-ROM típico pode armazenar 650 megabytes de dados digitais. Sendo 1 mega = 2^{20}, quantos bits de dados um CD-ROM pode guardar?

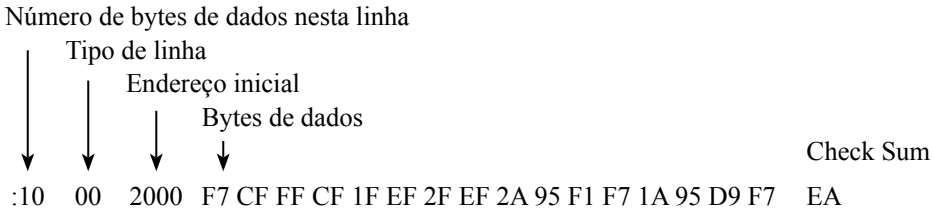
Solução
Lembre-se de que um byte corresponde a 8 bits. Portanto, 650 megabytes equivalem a $650 \times 2^{20} \times 8 =$ **5.452.595.200 bits**.

Aplicação 2.2

Para programar vários microcontroladores, as instruções binárias são armazenadas em um arquivo de um computador pessoal de um modo especial conhecido como formato Intel-Hex. A informação hexadecimal é codificada em caracteres ASCII para ser exibida facilmente na tela do PC, impressa e transmitida (um caractere de cada vez) por uma porta serial COM de um PC padrão. A seguir, você pode ver uma linha de um arquivo em formato Intel-Hex:

:10002000F7CFFFCF1FEF2FEF2A95F1F71A95D9F7EA

Formato Intel-Hex:



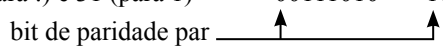
O primeiro caractere enviado é o código ASCII para dois pontos, seguido por um 1. Cada um deles possui um bit de paridade par anexado como o bit mais significativo. Um instrumento de teste verifica o padrão binário ao passar pelo cabo até o microcontrolador.

(a) Qual deve ser a aparência do padrão binário (inclusive a paridade)? (MSB – LSB)
(b) O valor 10, seguindo os dois pontos, representa o número de bytes hexadecimal total, que deve ser carregado na memória do micro. Qual é o número decimal de bytes que está sendo carregado?
(c) O número 0020 é um valor hexa de 4 dígitos, que representa o endereço em que o primeiro byte será armazenado. Qual é o maior endereço possível? Quantos bits seriam necessários para representar esse endereço?
(d) O valor do primeiro byte de dados é F7. Qual é o valor (em binário) do nibble menos significativo desse byte?

FFFF 1111 1111 1111 1111 16 bits

Solução

(a) Os códigos ASCII são 3A (para :) e 31 (para 1) 00111010 10110001



(b) 10 hexa = 1 × 16 + 0 × 1 = 16 bytes decimais.
(c) FFFF é o maior valor possível. Cada dígito hexa tem 4 bits; portanto, precisamos de 16 bits.
(d) O nibble menos significativo (4 bits) é representado pelo hexa 7. Em binário, seria 0111.

Aplicação 2.3

Um pequeno computador de controle de processos usa código hexadecimal para representar seus endereços de memória de 16 bits.

(a) Quantos dígitos hexadecimais são necessários?
(b) Qual é a faixa de endereços em hexadecimal?
(c) Quantas posições de memória existem?

Solução

(a) Visto que 4 bits são convertidos em um único dígito hexadecimal, 16/4 = **4**. Então, quatro dígitos hexadecimais são necessários.
(b) A faixa binária vai de 0000000000000000_2 a 1111111111111111_2. Em hexadecimal, isso se transforma em 0000_{16} a $FFFF_{16}$.
(c) Com quatro dígitos hexadecimais, o número total de endereços é 16^4 = **65.536.**

Aplicação 2.4

Números são fornecidos em BCD para um sistema baseado em microcontrolador e armazenados em binário puro. Como programador, você precisa decidir se precisa de 1 ou 2 bytes de posições de armazenamento.

(a) Quantos bytes serão necessários, se o sistema requerer uma entrada decimal de 2 dígitos?
(b) E se forem necessários 3 dígitos?

Solução

(a) Com 2 dígitos é possível fornecer valores até 99 ($1001\ 1001_{BCD}$). Em binário, esse valor é 01100011, que caberá em uma posição de memória de 8 bits. Dessa forma, você pode usar um único byte.
(b) Três dígitos podem representar valores até 999 (1001 1001 1001). Em binário, esse valor é 1111100111 (10 bits). Ou seja, você não pode usar apenas um byte; precisa de dois.