Organização de Computadores

Prof. Robson de Souza

Memória Primária

A memória é a parte do computador onde são armazenados programas e dados. Alguns cientistas da computação usam o termo armazém ou armazenagem em vez de memória, na verdade, o termo "armazenagem" está sendo usado cada vez mais para a armazenagem em disco. Sem uma memória da qual os processadores possam ler e na qual possam gravar, ou escrever, informações, não haveria computadores digitais com programas armazenados.

Bits

A unidade básica de memória é o dígito binário, denominado bit. Um bit pode conter um 0 ou um 1. É a unidade mais simples possível. O sistema numérico binário requer a distinção entre apenas dois valores. Por conseguinte, é o método mais confiável para codificar informações digitais.

O sistema binário é o sistema utilizado para os projetos de computadores, embora algumas empresas anunciem que seus computadores têm aritmética decimal, bem como binária, por exemplo, a IBM e seus grandes mainframes.

Na verdade, eles utilizam valores binários para "criar" números decimais, isso é feito usando 4 bits para armazenar um dígito decimal que utiliza um código denominado BCD (Binary Coded Decimal – decimal codificado em binário). Quatro bits oferecem 16 combinações, usadas para os 10 dígitos de 0 a 9, mas seis combinações não são usadas. O número 1.944 é mostrado a seguir codificado em formato decimal e em formato binário puro, usando 16 bits em cada exemplo:

decimal: 0001 1001 0100 0100 binário: 0000011110011000

Embora esse formato de BCD pareça interessante, ele possui uma limitação, pois 16 bits em BCD podem armazenar os números de 0 a 9999, dando somente 10 mil combinações, ao passo que um número binário puro de 16 bits pode armazenar 65.536 combinações diferentes.

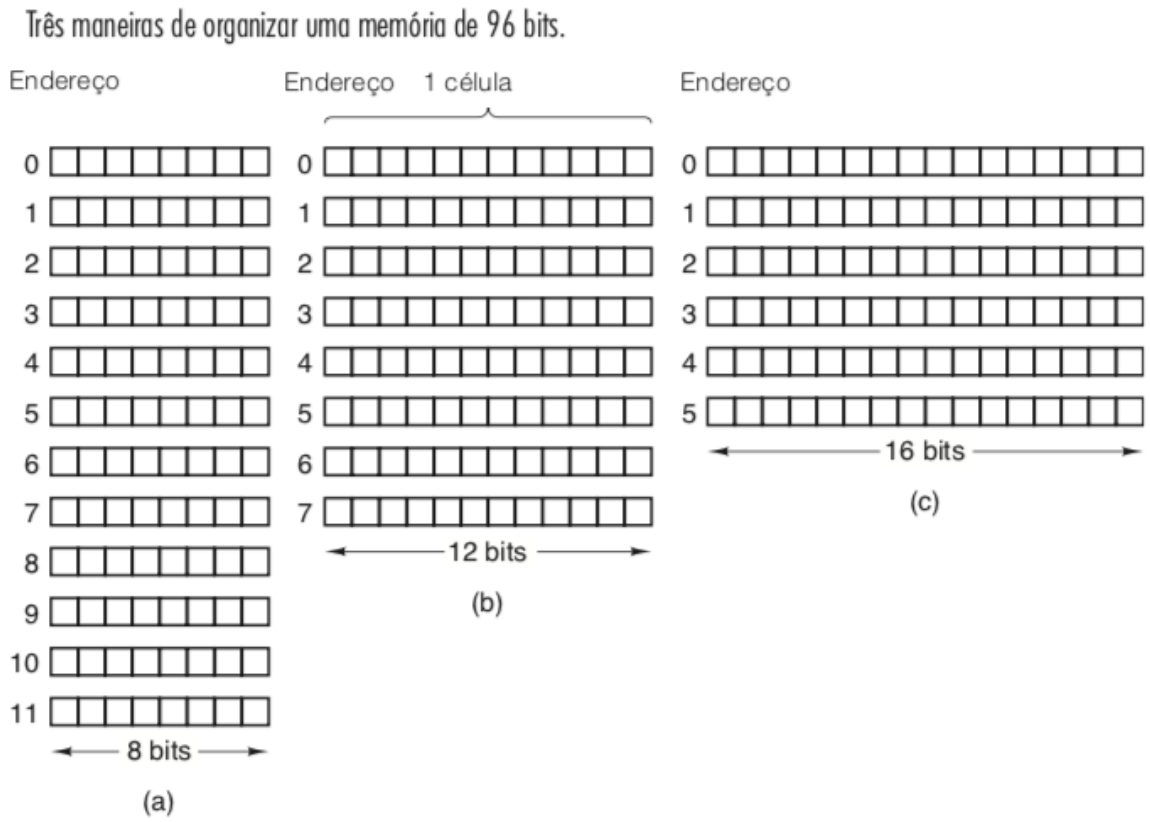
Endereços de memória

Memórias consistem em uma quantidade de células (ou locais), cada uma das quais podendo armazenar uma informação. Cada célula tem um número, denominado seu endereço, pelo qual os programas podem se referir a ela.

Se a memória tiver **n** células, elas terão endereços de **0** a **n – 1**. Todas as células em uma memória contêm o mesmo número de bits. Se uma célula consistir em **k** bits, ela pode conter quaisquer das 2^k diferentes combinações de bits.

Computadores que usam o sistema de números binários (incluindo notação octal ou hexadecimal para números binários) expressam endereços de memória como números binários. Se um endereço tiver **m** bits, o número máximo de células endereçáveis é 2^m.

Como exemplo, observe a figura abaixo que mostra três organizações diferentes para uma memória de 96 bits:



Fonte: (Tanenbaum e Austin, 2013)

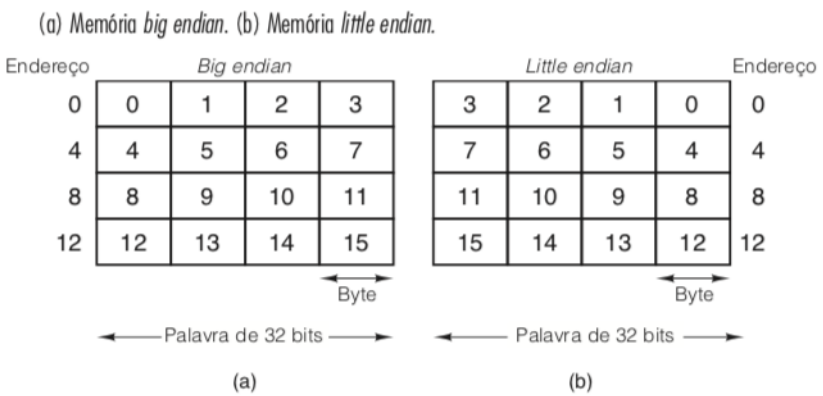
Utilizando a figura como exemplo, um endereço usado para referenciar a memória (a) precisa de no mínimo 4 bits para expressar todos os números de 0 a 11. Contudo, um endereço de 3 bits é suficiente para as memórias (b) e (c). O número de bits no endereço determina o número máximo de células diretamente endereçáveis na memória e é independente do número de bits por célula.

Vale ressaltar que 8 bits representam 1 byte, o termo octeto também é usado. Bytes são agrupados em palavras. Um computador com uma palavra de 32 bits tem 4 bytes/palavra, enquanto um computador com uma palavra de 64 bits tem 8 bytes/palavra. A significância de uma palavra é que grande parte das instruções efetua operações com palavras inteiras, por exemplo, somando duas palavras.

Assim, uma máquina de 32 bits terá registradores de 32 bits e instruções para manipular palavras de 32 bits, enquanto uma máquina de 64 bits terá registradores de 64 bits e instruções para movimentar, somar, subtrair e, em geral, manipular palavras de 64 bits.

Ordenação de bytes

Os bytes em uma palavra podem ser numerados da esquerda para a direita ou da direita para a esquerda. A Figura abaixo (a) retrata parte da memória de um computador de 32 bits cujos bytes são numerados da esquerda para a direita. A parte (b) dá uma representação análoga de um computador de 32 bits que usa uma numeração da direita para a esquerda.



Fonte: (Tanenbaum e Austin, 2013)

Ambas as representações são boas e internamente consistentes. Os problemas surgem quando uma das máquinas tenta enviar um registro à outra, isso pode gerar inconsistências na leitura de determinados dados, dependendo dos valores contidos neles.

Códigos de correção de erro

Memórias de computador podem cometer erros de vez em quando devido a picos de tensão na linha elétrica, raios cósmicos ou outras causas. Para se resguardar contra esses erros, algumas memórias usam códigos de detecção de erros ou códigos de correção de erros.

Nesse caso, bits extras são adicionados a cada palavra de memória de modo especial. Quando uma palavra é lida na memória, os bits extras são verificados para ver se ocorreu um erro.

Supondo que uma palavra possua **m** bits de dados, aos quais serão adicionados **r** bits redundantes, ou de verificação. O comprimento total **n** da palavra seria: n = m + r. Cada unidade de **n** bits costuma ser denominada **palavra de código de n bits**.

Dadas duas palavras de código:

1 0 0 0 1 0 0 1
1 0 1 1 0 0 0 1

Nesse caso é possível determinar quantos bits são diferentes: 3.
O número de posições de bit nos quais as duas palavras de código diferem é denominado **distância de Hamming**. Com isso, se duas palavras de código estiverem separadas por uma distância de Hamming **d,** será preciso **d** erros de um único bit para converter uma na outra.

Com uma palavra de memória de **m** bits, todos os 2^m padrões de bits são válidos, mas, devido ao modo como os bits de verificação são computados, somente 2^m das 2^n palavras de código são válidas. Se uma leitura de memória aparecer com uma palavra de código inválida, o computador sabe que ocorreu um erro de memória.

Dado o algoritmo para calcular os bits de verificação, é possível montar uma lista completa das palavras de código válidas e, por meio dela, achar as duas palavras de código cuja distância de Hamming seja mínima.

As propriedades de detecção e correção de erro de um código dependem de sua distância de Hamming. Para detectar **d** erros de único bit, você precisa de um código de distância **d + 1** porque, com tal código, não existe nenhum modo que permita que **d** erros de único bit mudem uma palavra de código válida para outra.

De modo semelhante, para corrigir erros de único bit, você precisa de um código de distância 2d + 1 (d é o número de erros) porque, desse modo, as palavras de código válidas estão tão distantes uma da outra que, mesmo que d mude, a palavra de código original ainda estará mais perto do que qualquer outra, portanto, ela pode ser unicamente determinada.

Exemplo de um código simples de correção de erros:

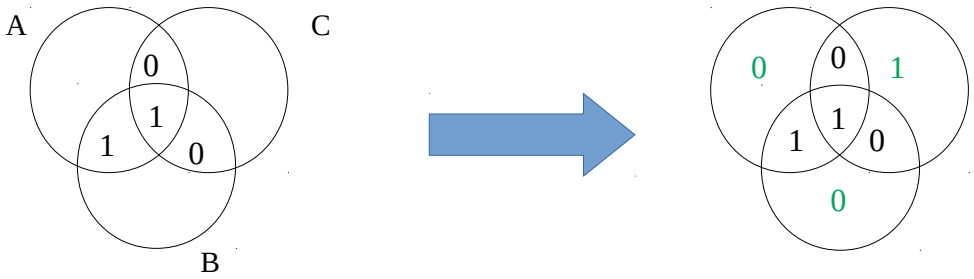
0000000000 0000011111 1111100000 1111111111

Esse código tem uma distância 5, o que significa que pode corrigir erros duplos. Se a palavra de código 0000000111 chegar, o destinatário sabe que a original deve ter sido 0000011111 (se não houver mais do que um duplo erro). Contudo, se um erro triplo mudar 0000000000 para 0000000111, o erro não pode ser corrigido.

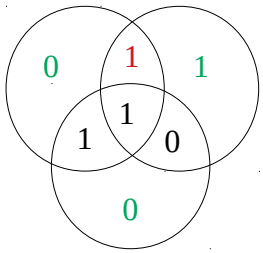
Um código de erro de fácil entendimento para palavras de 4 bits é a utilização de bits de paridade. Essa técnica consiste em “forçar” os bits transmitidos a serem sempre par, desse modo, se ao transmitir um dado juntamente com um bit de paridade, o valor for ímpar, fica claro que ocorreu um erro em algum dos bits.

Essa técnica permite em muitos casos, detectar o bit exato onde ocorreu o erro, com isso, é possível corrigi-lo. Para ficar mais claro como essa técnica funciona, vamos imaginar um número de 4 bits. Para visualizar esse problema, vamos alocar esses bits em regiões A, B e C em um diagrama, onde o primeiro bit vai para a região AB, o segundo bit vai para a região ABC, o terceiro bit vai para a região AC e o quarto bit vai para a região BC.

Após essa inserção dos bits em regiões, o objetivo é adicionar um bit de paridade em cada região, de modo a fazer com que, ao somar os valores das regiões, se obtenha um número par. Por exemplo, vamos utilizar o número 1100 na detecção de erros por paridade, utilizando a lógica que foi explicada, o diagrama ficará assim:



Nessa figura, os bits de paridade estão em verde. Agora, imagine que ocorreu um erro no terceiro bit (região AC), a imagem ficará assim:



Nesse caso, o erro é detectado pois as regiões A e C passam a ter um valor ímpar ao somar os bits. Nesse caso inclusive, a única mudança possível que faz com que os valores voltem ao normal é alterar o bit da região AC para 0, com isso o erro pode ser corrigido.

A verificação de erros por paridade possui um problema, que é o fato de que os bits podem ser alterados e ainda assim gerar valores válidos com relação a paridade.

Referências bibliográficas:

TANENBAUM, Andrew S. Organização Estruturada de Computadores, 2007, 5ª Edição.

TANENBAUM, Andrew S. AUSTIN, Todd; Organização Estruturada de Computadores, 2013, 6ª Edição.