MICRO COMPUTADOR

CURSO BASICO

Sumário

VOLUME 1

Chips & bytes

Conexões

Fundamentos

Hardware

Os precursores

Perspectivas

VOLUME 1

Por dentro do hardware

Programação Basic

Software

Sumário

VOLUME 2

Chips & bytes

Conexões

Fundamentos

Hardware

Os precursores

Perspectivas

Por dentro do hardware

VOLUME 2

Programação BASIC

Software

Som e luz

Bits e bytes

O computador só entende de números, e tudo o que podemos fazer com dez dígitos ele faz com apenas dois — 0 e 1.

As palavras bits e bytes são sempre empregadas quando se escreve sobre computadores. Referem-se ao modo pelo qual os computadores armazenam e usam os números — um jeito absolutamente diferente daquele empregado pelas pessoas.

Representamos os números com dez símbolos diferentes (de 0 a 9) e os manipulamos em múltiplos de 10 (o que é chamado sistema de base 10). Já os computadores, para toda a sua prestidigitação matemática, precisam de apenas dois símbolos: 0 e 1. Para representar suas combinações é que se recorre aos bits e bytes. Um bit é a menor unidade de informação que um computador pode usar. É com um bit que o computador representa os dois dígitos — 0 e 1.

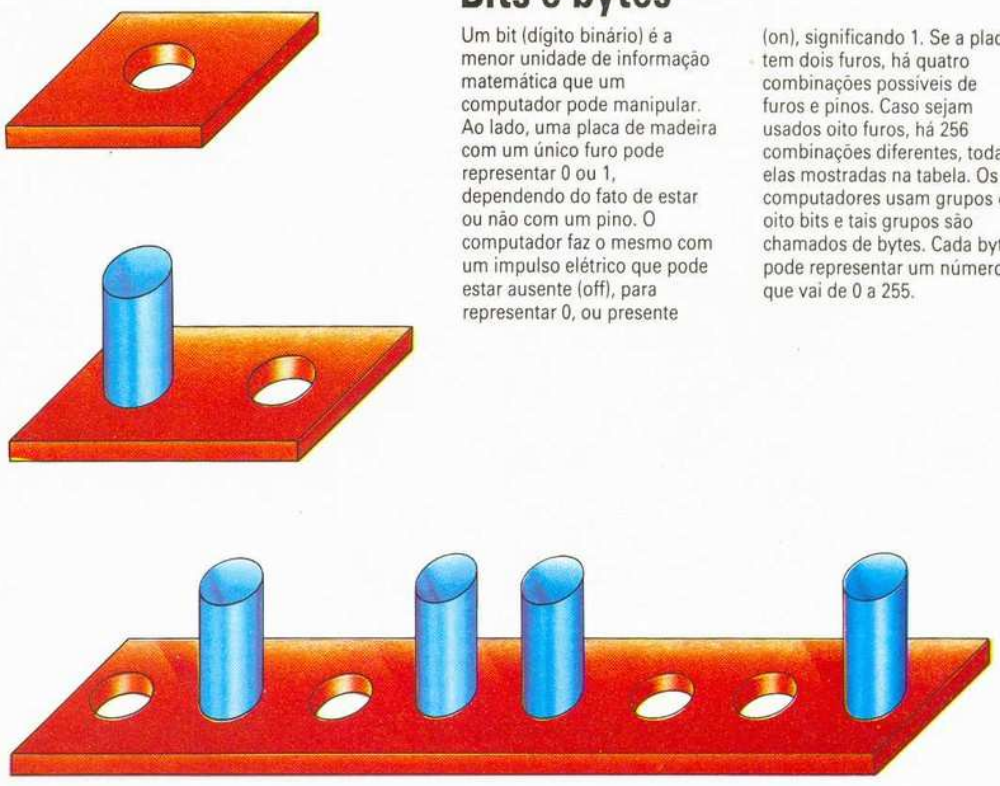
Em primeiro lugar, vamos ver o que são os bits e a razão por que são assim chamados. Computadores são instrumentos eletrônicos; tudo o que realizam é feito com sinais ou impulsos elétricos. Um único impulso pode estar "presente" (on) ou "ausente" (off): este é o princípio que permite representar números com impulsos.

A ilustração mostra uma placa de madeira com um furo em que se pode colocar um pino. Embora se trate de um único furo, pode representar dois números e serve de excelente analogia à maneira de funcionar do computador. Ou o furo está sem o pino e nesse caso representa 0, ou com o pino e representa 1. Uma única placa pode, então, simbolizar 0 ou 1.

Num computador, obtém-se o mesmo resultado com um impulso elétrico. Quando este está ausente (off), significa 0; caso contrário (on), representa 1. Um único dispositivo eletrônico, ou o pedaço de madeira com o furo, pode então ser usado para representar dois estados: sem pino ou com pino; ausente ou presente; off ou on; 0 ou 1.

A menor unidade de informação chama-se bit. A palavra, em inglês, já sugere algo de tamanho reduzido e representa dois estados possíveis. A palavra deriva de **BI**nary digi**T** (dígito binário). Pensado de outra forma, um bit é capaz de contar, mas apenas de 0 a 1.

Uma placa com dois furos pode indicar quatro di-



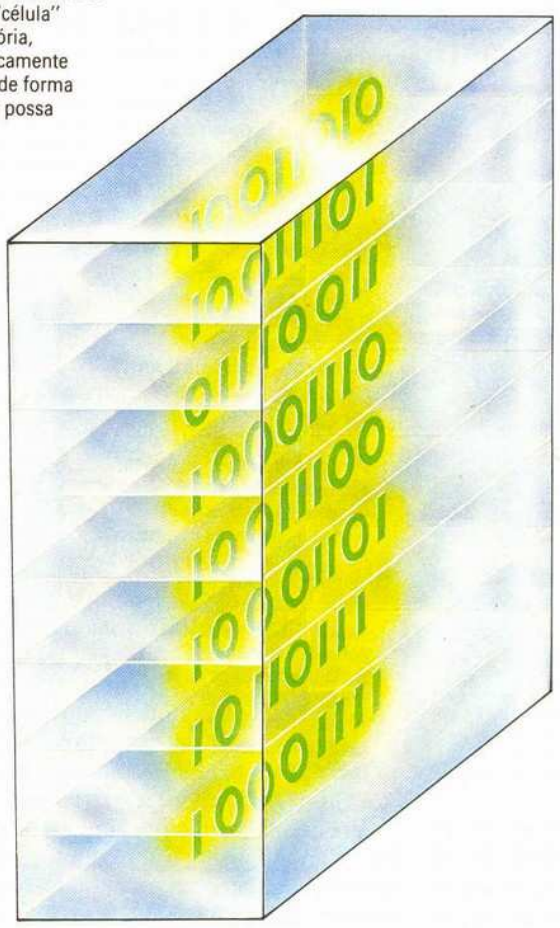
Bits e bytes

Um bit (dígito binário) é a menor unidade de informação matemática que um computador pode manipular. Ao lado, uma placa de madeira com um único furo pode representar 0 ou 1, dependendo do fato de estar ou não com um pino. O computador faz o mesmo com um impulso elétrico que pode estar ausente (off), para representar 0, ou presente (on), significando 1. Se a placa tem dois furos, há quatro combinações possíveis de furos e pinos. Caso sejam usados oito furos, há 256 combinações diferentes, todas elas mostradas na tabela. Os computadores usam grupos de oito bits e tais grupos são chamados de bytes. Cada byte pode representar um número que vai de 0 a 255.

0	00000000	128	10000000
1	00000001	129	10000001
2	00000010	130	10000010
3	00000011	131	10000011
4	00000100	132	10000100
5	00000101	133	10000101
6	00000110	134	10000110
7	00000111	135	10000111
8	00001000	136	10001000
9	00001001	137	10001001
10	00001010	138	10001010
11	00001011	139	10001011
12	00001100	140	10001100
13	00001101	141	10001101
14	00001110	142	10001110
15	00001111	143	10001111
16	00010000	144	10010000
17	00010001	145	10010001
18	00010010	146	10010010
19	00010011	147	10010011
20	00010100	148	10010100
21	00010101	149	10010101
22	00010110	150	10010110
23	00010111	151	10010111
24	00011000	152	10011000
25	00011001	153	10011001
26	00011010	154	10011010
27	00011011	155	10011011
28	00011100	156	10011100
29	00011101	157	10011101
30	00011110	158	10011110
31	00011111	159	10011111
32	00100000	160	10100000
33	00100001	161	10100001
34	00100010	162	10100010
35	00100011	163	10100011
36	00100100	164	10100100
37	00100101	165	10100101
38	00100110	166	10100110
39	00100111	167	10100111
40	00101000	168	10101000
41	00101001	169	10101001
42	00101010	170	10101010
43	00101011	171	10101011
44	00101100	172	10101100
45	00101101	173	10101101
46	00101110	174	10101110
47	00101111	175	10101111
48	00110000	176	10110000
49	00110001	177	10110001
50	00110010	178	10110010
51	00110011	179	10110011
52	00110100	180	10110100
53	00110101	181	10110101
54	00110110	182	10110110
55	00110111	183	10110111
56	00111000	184	10111000
57	00111001	185	10111001
58	00111010	186	10111010
59	00111011	187	10111011
60	00111100	188	10111100
61	00111101	189	10111101
62	00111110	190	10111110
63	00111111	191	10111111
64	01000000	192	11000000
65	01000001	193	11000001
66	01000010	194	11000010
67	01000011	195	11000011
68	01000100	196	11000100
69	01000101	197	11000101
70	01000110	198	11000110
71	01000111	199	11000111
72	01001000	200	11001000
73	01001001	201	11001001
74	01001010	202	11001010
75	01001011	203	11001011
76	01001100	204	11001100
77	01001101	205	11001101
78	01001110	206	11001110
79	01001111	207	11001111
80	01010000	208	11010000
81	01010001	209	11010001
82	01010010	210	11010010
83	01010011	211	11010011
84	01010100	212	11010100
85	01010101	213	11010101
86	01010110	214	11010110
87	01010111	215	11010111
88	01011000	216	11011000
89	01011001	217	11011001
90	01011010	218	11011010
91	01011011	219	11011011
92	01011100	220	11011100
93	01011101	221	11011101
94	01011110	222	11011110
95	01011111	223	11011111
96	01100000	224	11100000
97	01100001	225	11100001
98	01100010	226	11100010
99	01100011	227	11100011
100	01100100	228	11100100
101	01100101	229	11100101
102	01100110	230	11100110
103	01100111	231	11100111
104	01101000	232	11101000
105	01101001	233	11101001
106	01101010	234	11101010
107	01101011	235	11101011
108	01101100	236	11101100
109	01101101	237	11101101
110	01101110	238	11101110
111	01101111	239	11101111
112	01110000	240	11110000
113	01110001	241	11110001
114	01110010	242	11110010
115	01110011	243	11110011
116	01110100	244	11110100
117	01110101	245	11110101
118	01110110	246	11110110
119	01110111	247	11110111
120	01111000	248	11111000
121	01111001	249	11111001
122	01111010	250	11111010
123	01111011	251	11111011
124	01111100	252	11111100
125	01111101	253	11111101
126	01111110	254	11111110
127	01111111	255	11111111

Bytes na memória

Bytes são grupos de oito dígitos binários (bits). Cada byte é usado pelo computador para armazenar números que podem ir de 0 a 255. Além disso, todo byte é armazenado numa espécie de "célula" separada da memória, disposta sistematicamente junto com outras, de forma que o computador possa localizar o byte de que precisa. Para isso, ele tem de saber onde está o compartimento de memória que contém o byte.



ferentes estados, ou contar de 0 a 3. Os furos podem estar vazios; o furo da direita pode estar com um pino; o da esquerda pode estar com um pino; ou ambos estarem com pinos. A ilustração mostra uma placa com oito furos. Há, portanto, 256 combinações possíveis de pinos e furos — todas mostradas na tabela ao lado com números 1 representando pinos e zeros representando furos.

Um conjunto como esse, de oito dígitos binários (bits), é chamado de byte. Um único byte, portanto, pode representar 256 diferentes estados (ou pode contar de 0 a 255).

Quando se diz que um computador "armazena" um byte, significa que um número (de 0 a 255) está na memória do computador para ser utilizado no momento que for necessário. Cada byte tem seu próprio compartimento e tais compartimentos podem ser dispostos em seqüência (a ilustração acima mostra os bytes como se estivessem empilhados). Se o computador quer recuperar um número, tudo que precisa saber é em qual compartimento o byte (que contém esse número) está armazenado.

Todos os números de 0 a 255 podem ser representados mediante combinações de 1 e 0 (veja tabela). Os bits são armazenados e usados por computadores em grupos de 8. Oito bits juntos formam um byte.

Memória infalível

O computador armazena no seu interior um grande número de informações. Esses dados devem ser organizados cuidadosamente para que possam ser recuperados a qualquer momento.

No ser humano, a memória é um arquivo da mente; um lugar em que detalhes de experiências vividas são armazenados para uso posterior. Em computação, a palavra "memória" significa quase a mesma coisa; mas a memória do computador limita-se a determinadas funções.

Para um ser humano, a memória fraca pode causar momentos embaraçosos. Para um computador, a falta de memória é algo desastroso. Sem a memória, o computador não teria base para funcionar, já que faz uso dela para armazenar os programas que o dirigem.

Em ambos os casos, o termo memória refere-se a duas coisas: armazenagem e lembrança. Armazenar informações sem poder retirá-las seria inútil; e tentar lembrar-se de informações que não foram armazenadas obviamente não tem sentido.

Ambas as memórias são similares também sob outro aspecto. A memória humana parece atuar a curto e a longo prazo. Um homem atravessando a rua, por exemplo: certamente ele "se lembrará" de esperar que o último carro passe. Entretanto, quando já estiver do outro lado da rua, o homem não se lembrará mais do veículo. Sua memória é, neste caso, de curto prazo.

No entanto, se dentro do mesmo carro ele visse dois homens mascarados no banco traseiro, ameaçando sua esposa, com certeza se lembraria do modelo e da cor do carro e talvez até da placa do veículo. A memória é, neste caso, de longo prazo.

O computador possui esses dois tipos de memória. A memória a longo prazo ou permanente contém programas e informações que o usuário deseja manter. São armazenados na forma de gravações magnéticas em fitas cassete, discos flexíveis ou cartuchos ROM.

O tipo de memória a curto prazo ou transitória é chamado RAM, um chip que fica dentro do próprio computador. Seu uso restringe-se ao período em que o computador está em funcionamento. Se o computador não receber energia, mesmo que seja por uma fração de segundo, todo o conteúdo memorizado desaparece instantaneamente.

Entretanto, a analogia feita com a memória humana não é inteiramente exata. Para que o computador trabalhe, os programas e dados devem ser transferidos da memória a longo prazo para a memória a curto prazo; só assim o computador poderá ter acesso instantâneo às informações. E o modo como são armazenados e transferidos os dados num computador é completamente diferente do mecanismo da memória humana.

É ainda um mistério o mecanismo da memória humana, já que as lembranças não são depositadas em partes identificáveis do cérebro. Não precisamos descobrir a localização de determinada informação para trazê-la à lembrança. E quando uma lembrança não é mais necessária não a depositamos novamente numa pequena divisão de nosso cérebro.

Caos organizado

Na memória do computador, a localização de cada item é vital. O computador deve encontrar um byte de informação, saber se ela é parte do programa ou dos dados do programa e saber, finalmente, onde depositar a informação.

A memória humana se parece com uma caixa desorganizada, abarrotada de informações. As informações são depositadas aparentemente ao acaso e se confundem com outras lembranças, à medida que novas experiências ocorrem. Mesmo assim, o cérebro é capaz de conseguir a informação correta, no momento certo.

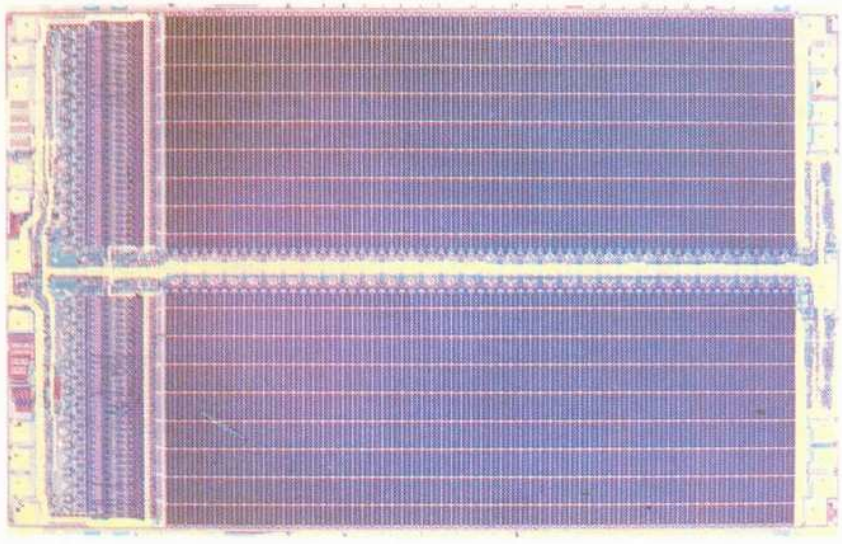
A memória do computador é como um enorme pombal em que cada parte é totalmente separada da outra. Tudo é muito organizado: cada parte do pombal tem um número (conhecido como seu "endereço") e contém apenas um byte, nem mais nem menos. O computador encontra a informação pelo número do compartimento do pombal e não pelo que nele está armazenado.

O computador não tem inteligência, portanto não

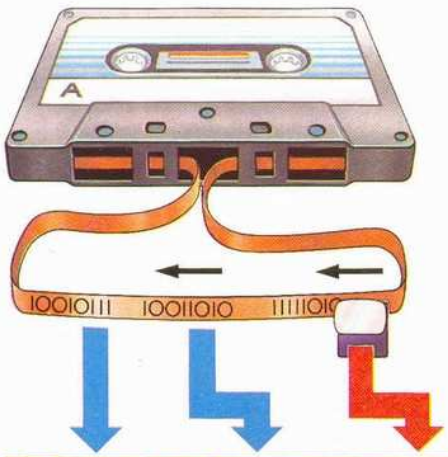
Memória RAM
O chip RAM (abaixo) constitui um dos mais recentes avanços no que se refere à tecnologia de computadores. RAM (Random Access Memory) é uma das variedades de memória totalmente eletrônica, categoria que também abrange ROM (Read Only Memory). As fitas cassete e os discos magnéticos flexíveis são exemplos de outro gênero importante, a memória eletromagnética.

A memória RAM é fabricada com silício e utiliza processo fotográfico e gravação química, a fim de criar milhares de minúsculos transistores. Cada bit de memória requer pelo menos um transistor.

O tempo que se leva para "escrever" um simples bit em qualquer das 16.384 células de memória é de cerca de 200 nanossegundos, ou seja, 5 milionésimos de segundo.



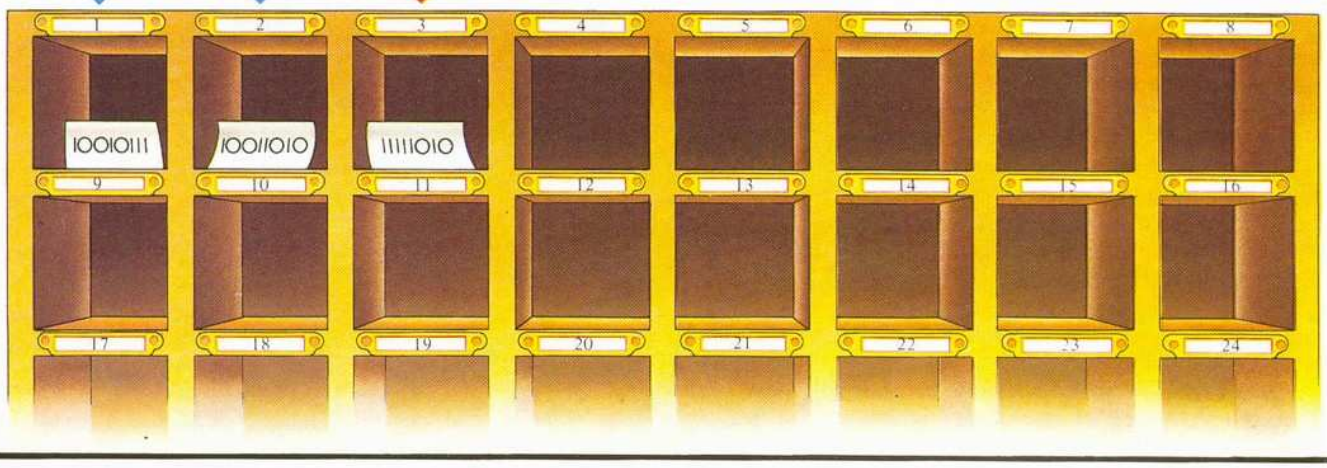
Memória



Os programas registrados em fita são armazenados em seqüência, cada bit de cada byte registrado em ordem. Quando a fita é utilizada novamente, o computador faz a leitura de cada bit, mas armazena-os em grupos de oito (bytes) em cada célula de memória. O primeiro byte da fita é colocado na primeira célula disponível; o segundo byte, na segunda, e assim por diante. Quando o computador for executar o programa, só precisa saber o endereço de início. O computador transfere o conteúdo de cada célula de memória para a CPU (Unidade Central de Processamento) e estes bytes fazem com que se "realizem" as ações requisitadas pelo programa.

Parte da memória do computador é ocupada por programas de "controle interno", responsáveis pelos aspectos fundamentais de sua operação. Eles verificam qual das teclas foi pressionada, expõem caracteres na tela etc. Esse software interno pode também incluir a linguagem BASIC de programação. Esses programas ocupam lugar na memória e deixam menos espaço para a armazenagem de programas feitos pelo usuário ou adquiridos comercialmente. Algumas versões BASIC, por exemplo, ocupam 16 Kbytes de memória. Quando um programa é carregado de fita cassete, a primeira locação de memória disponível não será a primeira locação em RAM. É uma das tarefas dos programas de controle interno saber a primeira locação de memória disponível para o usuário. Depois de o programa ter sido carregado na memória RAM do computador, os programas de controle dizem: "Comece olhando para a locação x de memória, depois continue examinando cada locação sucessiva de memória, registrando o conteúdo da locação na CPU". A ordem original em que o programa foi registrado no teclado pelo programador é a mesma ordem registrada na fita.

Quando o programa passa da fita para a memória do computador, é colocado nas células de memória na mesma ordem em que estava, como se tivesse sido digitado no teclado.



consegue organizar sua memória sozinho. O que leva o computador a armazenar dados é o fato de que alguém preencheu a parte certa do pombal, na hora e na ordem exatas. E como isso acontece com o sistema de computador típico?

Quando você liga seu microcomputador, em geral uma mensagem aparece na tela para indicar que ele está funcionando. Na maioria das vezes, ele também informa que você pode começar a escrever o programa. A mensagem e as condições que lhe permitem a programação estão armazenadas na parte interior da memória do computador e devem ter sido armazenadas na memória a longo prazo (em geral, a Read Only Memory ou chip ROM).

Esta parte da memória do computador contém programas que verificam se as teclas foram pressionadas, imprimem os caracteres (letras, números, símbolos) na tela e executam outras tarefas essenciais a seu funcionamento. Contém ainda um programa especial que traduz comandos geralmente escritos em BASIC na mais simples linguagem binária de uns e zeros, compreendida pelo computador.

Quando o computador é ligado, a mensagem na tela diz: "x bytes livres" — onde x é algo como 15, 797, ou qualquer outro número estranho. Isso indica a quantidade de compartimentos do pombal que está disponível. À medida que as teclas são pressionadas, os compartimentos livres vão sendo preenchidos; e aqui chegamos a outro fator importante em relação à memória do computador: a ordem de armazenagem das informações.

A pressão de uma tecla envia um byte que representa a letra pressionada (ver p. 3), para que seja armazenado na memória. Pressionando-se o D, por exemplo, essa letra ocupa espaço em um compartimento de memória na forma binária.

Mas que compartimento do pombal será ocupado pela letra D? Ela vai para o primeiro espaço vazio da memória não permanente do computador. Se pensarmos na seqüência de compartimentos de um pombal, a letra D iria para o primeiro espaço à esquerda.

Pressione outra tecla, digamos I, e a configuração apropriada de bits irá para o segundo espaço vazio, à direita do D. Pressione uma terceira tecla, o A, e irá para o terceiro espaço vazio, ao lado do I. Olhando para o pombal completo, veremos os códigos da palavra DIA aparecerem na primeira fila.

O computador tem um contador interno para avaliar que parte do pombal foi atingida. Ele sabe onde começar porque um programa interno de controle avisa onde começa a próxima área livre de memória. À medida que cada letra é armazenada, o contador é acrescido de um, a fim de dizer qual é o compartimento seguinte do pombal para a próxima letra digitada.

Verdadeiro ou falso?

Os computadores ainda não podem "pensar" como você, mas são capazes de seguir facilmente todas as leis da lógica.

A CPU (Unidade Central de Processamento) é comumente considerada o coração do computador. É o lugar onde ocorrem todos os cálculos e as tomadas de decisão lógica. Mas de que modo são feitos esses cálculos e tomadas essas decisões?

Para compreender isso é necessário conhecer os fundamentos da aritmética binária e estar familiarizado com as portas lógicas. Essas portas são circuitos elétricos capazes de tomar decisões lógicas e fazer comparações. Isso parece mais complicado do que é na realidade e os princípios podem ser facilmente ilustrados com exemplos da vida diária.

Existem três tipos fundamentais de portas — a AND (E), a OR (OU) e a NOT (NÃO).

Conexões lógicas

A porta AND é um circuito que dá saída "verdadeira", se todas as entradas forem "verdadeiras". Vamos ver o que isso significa. Suponha que você queira fazer uma viagem para o campo. Se tem carro E (AND) gasolina, pode fazer a viagem. Se tem gasolina, mas não tem carro, não pode viajar. De modo semelhante, o carro sem a gasolina significa que não haverá viagem.

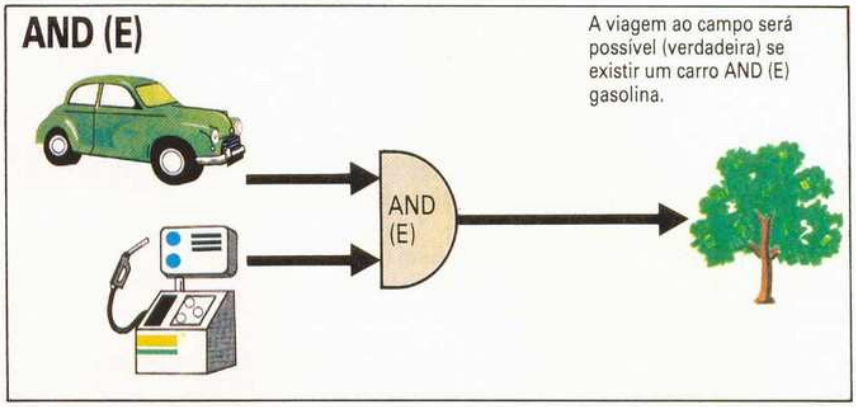
Neste "circuito" AND existem duas condições de entrada e ambas precisam ser "verdadeiras". Para poder viajar (a "saída") precisa haver um carro E (AND) precisa haver gasolina, e só neste caso a saída se torna "verdadeira". Posteriormente, veremos de que modo este diagrama lógico pode ser apresentado na forma de uma equação lógica, além de poder ser representado em uma "tabela de validação".

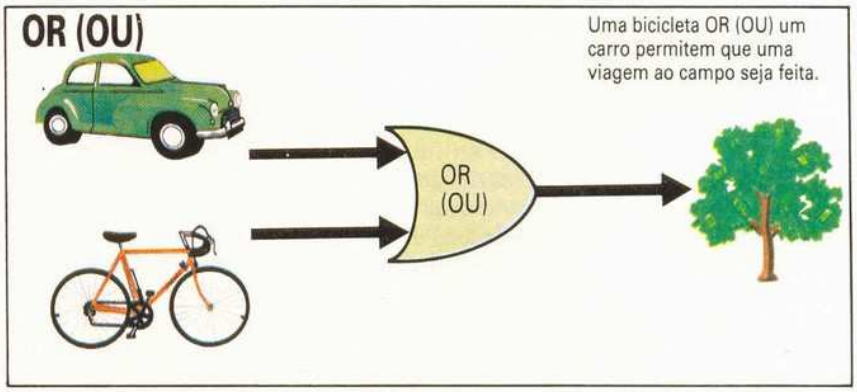
Imagine agora uma situação ligeiramente diferente. Alguém deseja viajar para o campo. A viagem será possível se a pessoa tiver carro OU (OR) bicicleta (vamos supor que o carro esteja com gasolina). Se a pessoa tiver carro poderá viajar. Se tiver bicicleta, também. Contudo, se nenhuma das condições de entrada for verdadeira, a viagem se tornará impossível — no jargão do computador, a saída se torna falsa (i.e., não é verdade que a pessoa vai viajar para o campo).

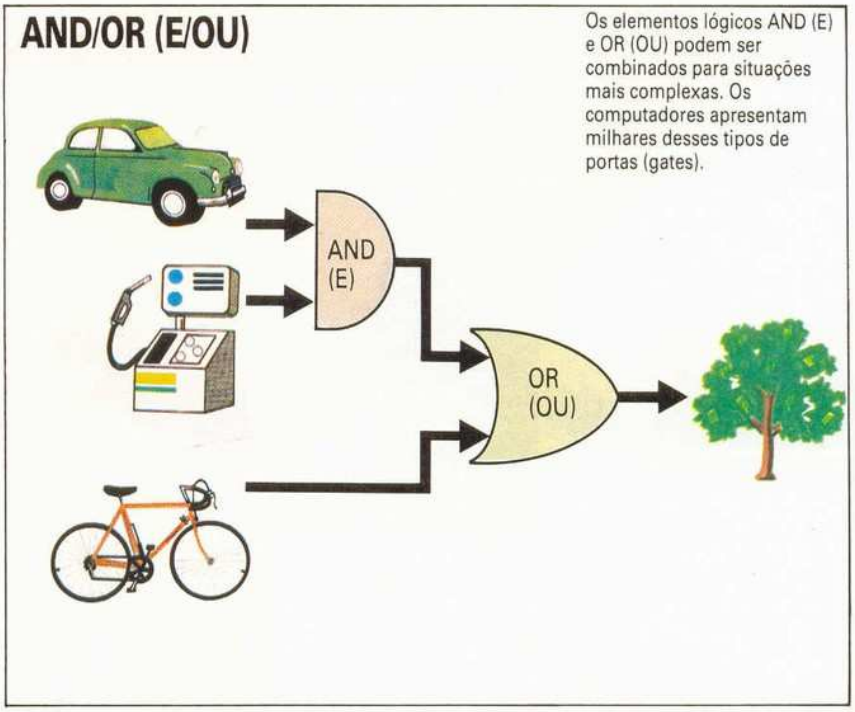
Há mais uma porta lógica fundamental, a porta NOT. Esta porta simplesmente fornece uma saída oposta à entrada. Se a entrada for verdadeira, a saída será falsa. Se a entrada for falsa, a saída será verdadeira. Ampliando nossa metáfora, a viagem ao campo de carro ou bicicleta será falsa se houver um pneu furado.

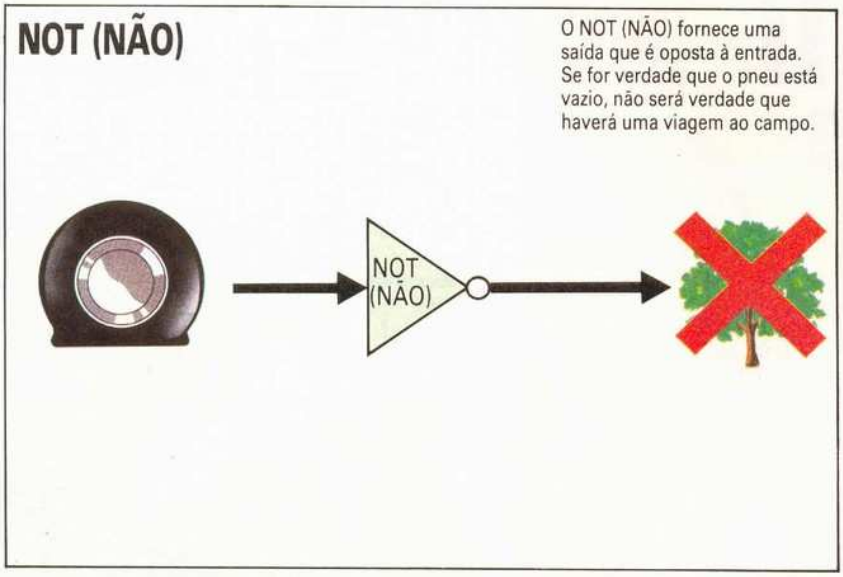
Se a entrada (pneu furado) é verdadeira, então a saída (a viagem) será falsa.

Esses elementos lógicos podem ser combinados e ilustramos isso com o exemplo da viagem ao campo.









As combinações de AND, OR e NOT permitem que todas as decisões baseadas na lógica convencional sejam feitas.

Experimente solucionar o que seria necessário para se fazer um churrasco no jardim, por exemplo. Fica bem complicado. Para termos um churrasco no jardim (a saída verdadeira), precisaríamos de várias condições de entrada: dinheiro OU (OR) cheque OU (OR) cartão de crédito (para comprar comida e bebida) E (AND) um dia livre E (AND) um tempo agradável E (AND) uma churrasqueira E (AND) carvão E (AND) fósforo E (AND) espetos.

Tabelas de validação

Os símbolos que utilizamos na ilustração são os mesmos dos diagramas de circuito do computador. A fim de sabermos como as decisões lógicas podem ser tomadas empregando circuitos elétricos, vamos analisar a ''tabela de validação'' para o AND. Se utilizarmos a letra (c) para representar a condição de entrada ''possuir um carro'' e a letra (g) para ''ter gasolina'', podemos representar a condição de saída ''fazer uma viagem para o campo'' com a letra (v). Podemos também empregar V para verdadeiro e F para falso. A tabela de validação apresenta todas as combinações possíveis de condições de entrada e o efeito da utilização de AND na saída. Assim temos:

CARRO	F	V	F	V	(c)
GASOLINA	F	F	V	V	(g)
VIAGEM	F	F	F	V	(v)

A tabela de validação para duas entradas AND

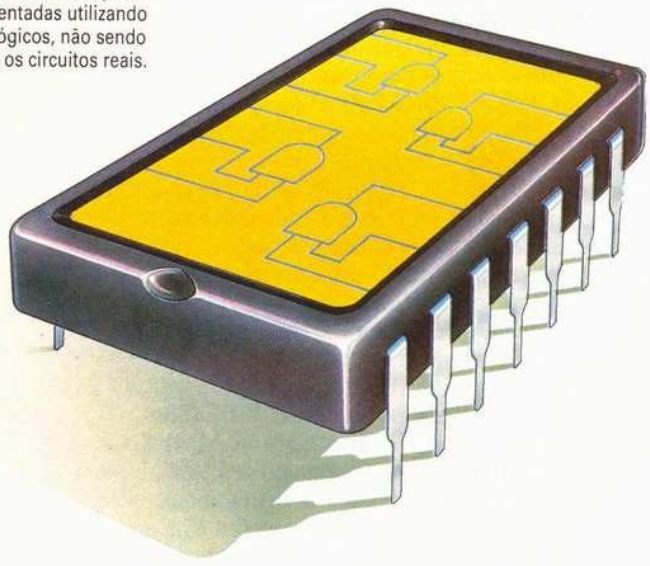
CARRO	0	1	0	1	(c)
GASOLINA	0	0	1	1	(g)
VIAGEM	0	0	0	1	(v)

A mesma tabela usando 0 e 1 para Falso e Verdadeiro

Nos computadores empregamos 0 e 1 para significar falso e verdadeiro. O computador interpreta a voltagem positiva como 1 e a voltagem zero como 0. E o circuito AND pode ser feito, utilizando transistores, de modo que, se ambas as entradas são voltagens positivas, a saída é também uma voltagem positiva. Se uma ou ambas as entradas forem de voltagem zero, a saída do circuito também será zero.

Um circuito eletrônico OR fornece uma saída de voltagem positiva, se uma ou ambas as entradas forem positivas. Se ambas as entradas forem 0, a saída também será 0. Em um circuito NOT, a entrada é simplesmente invertida: se a entrada for positiva, a saída será 0; se a entrada for 0, a saída será positiva.

O chip 7408
Os grandes chips de silício normalmente contêm milhares de portas (gates) para executar a lógica de AND, OR e NOT. O pequeno chip 7408 da ilustração contém todos os transistores e circuitos para quatro portas AND. As portas são representadas utilizando símbolos lógicos, não sendo mostrados os circuitos reais.



Caixa-forte

Os dígitos binários 0 e 1 podem ser facilmente somados quando você utiliza as lógicas do AND, OR e NOT.

Em artigo anterior (ver p. 68) vimos como era relativamente simples o uso de circuitos transistorizados para realizar decisões lógicas, como AND, OR e NOT. É surpreendente como essas mesmas "portas lógicas" (logic gates) são igualmente a base para a execução de procedimentos aritméticos no interior do computador.

Pela lógica, as entradas das portas (gates) são 0 volt, para representar o valor "falso", ou uma voltagem positiva para o valor "verdadeiro". A ausência de voltagem é geralmente representada por 0, e a voltagem positiva, por 1. Quando as portas lógicas são usadas para realizar procedimentos aritméticos, os mesmos dígitos 0 e 1 são usados; desta vez, porém, representando exatamente os algarismos 0 e 1 que estiveram sendo somados.

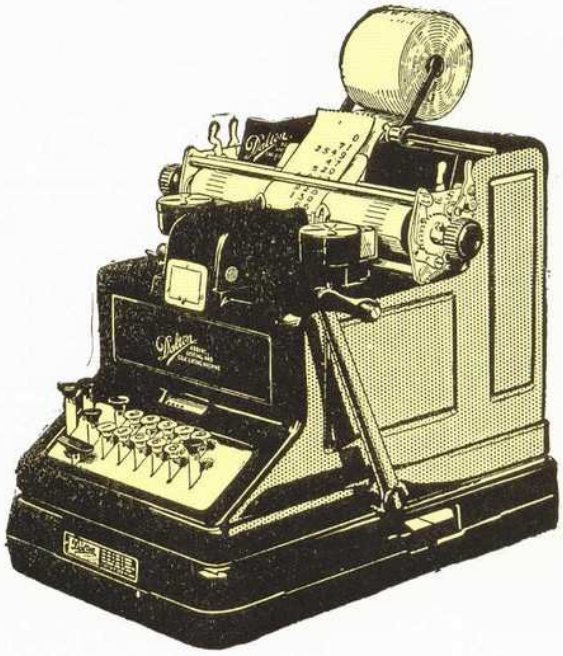
Se quisermos somar dois dígitos binários, haverá apenas duas entradas para o circuito de soma e pode haver apenas quatro combinações de entrada: 0 + 0, 0 + 1, 1 + 0 e 1 + 1. Na matéria anterior sobre aritmética binária, vimos que 0 mais 0 é igual a 0 (como na aritmética do sistema decimal). Também vimos que 0 mais 1 (ou 1 mais 0) é igual a 1 (novamente como na aritmética do sistema decimal). A diferença entre a aritmética que aprendemos na escola e o sistema binário é que, neste, 1 mais 1 é igual a 0, e um dígito é transportado para a coluna da esquerda. A representação aritmética dessas quatro adições é a seguinte:

X		Y		Z
0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	10

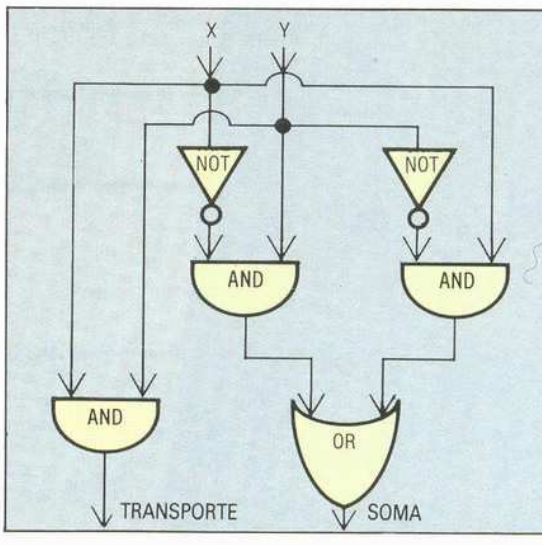
Usando uma porta OR para realizar a adição, se ambas as entradas tiverem valor falso (0), obteremos uma saída de valor falso (0); se uma das entradas tiver o valor verdadeiro (0 + 1 ou 1 + 0), teremos uma saída com valor verdadeiro (1).

Até este ponto, o recurso a uma única porta OR pode parecer perfeitamente adequado para a soma de dois dígitos binários. Porém, cuidado: se ambas as entradas forem verdadeiras, a saída de uma porta OR simples terá também valor verdadeiro, mas esta seria uma resposta errada em aritmética binária. A resposta deveria ser um dígito 0 e um outro dígito 1, de transporte. Uma simples porta OR dará respostas corretas para três das quatro combinações, porém três em quatro ainda não é uma resposta satisfatória.

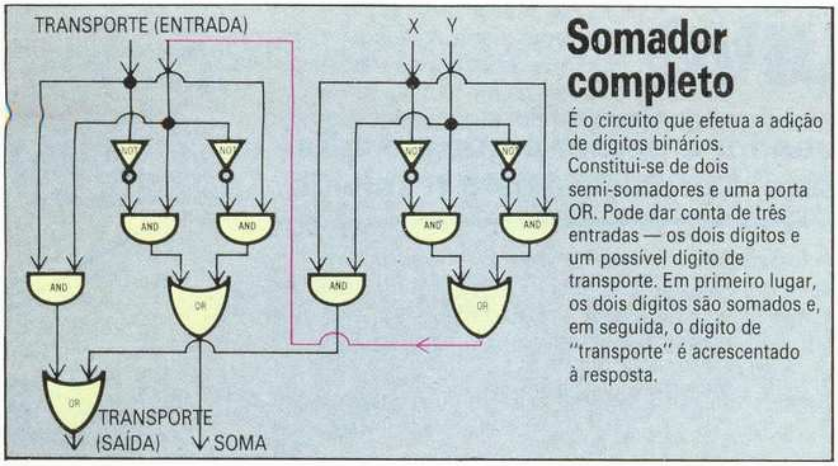
O que é necessário é um circuito que dê 0 como resposta, se ambas as entradas forem 0; 1, se uma delas for 1 e a outra 0; e ainda 0, se ambas forem 1 e mais 1 de transporte. Isto não é tão difícil quanto parece. Se tivermos duas portas AND, com as duas entradas conectadas passando em ambas as portas, mas com uma entrada sendo invertida através de uma porta NOT em uma das portas AND, e se a outra entrada for invertida através de outra porta NOT e na outra porta AND (ver ilustração), nós teremos uma situação em que 0 em ambas as entradas resultará em uma saída de valor falso em ambas as portas AND, e um dígito 1, em ambas as entradas, dará analogamente uma saída com valor falso, em ambas as por-



A máquina de somar
Até a invenção recente da calculadora eletrônica, a máquina de somar mecânica (ou caixa registradora) era uma presença comum em lojas e escritórios. Com exceção de algum aperfeiçoamento, quase não sofreu mudanças por 300 anos, funcionando com uma combinação de rodas dentadas e engrenagens. O potencial comercial da calculadora foi logo percebido. Pascal inventou a primeira máquina de somar para uso, por seu pai, no recolhimento de impostos. Com a criação, por Leibniz, dos procedimentos para cálculo de divisão e multiplicação, a calculadora estava preparada para ser utilizada no comércio.



Semi-somador
É um dispositivo para soma de dois números binários que, para tal, usa uma combinação de portas lógicas. É chamado semi-somador porque não pode dar conta do dígito de transporte que geralmente resulta da realização de somas. Lembre-se de que uma porta NOT converte o 1 em 0 e o 0 em 1. Para uma saída 1, as duas entradas de uma porta AND devem ser 1. A saída de uma porta OR será 1, se uma ou ambas as entradas forem 1. A saída será 0, somente se ambas as entradas forem 0.



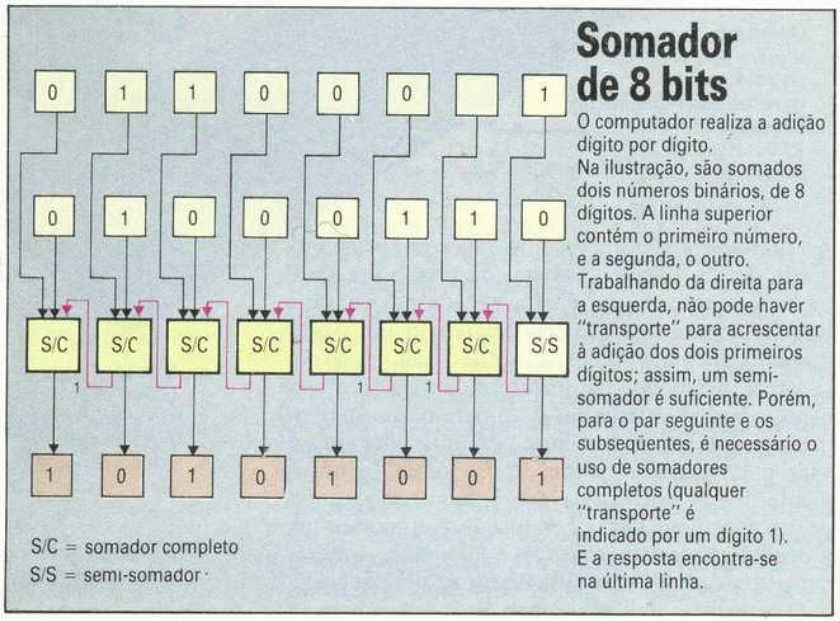
Somador completo

É o circuito que efetua a adição de dígitos binários. Constitui-se de dois semi-somadores e uma porta OR. Pode dar conta de três entradas — os dois dígitos e um possível dígito de transporte. Em primeiro lugar, os dois dígitos são somados e, em seguida, o dígito de "transporte" é acrescentado à resposta.

tas. Por outro lado, 0 em uma entrada e 1 na outra darão duas saídas de valor "verdadeiro" em uma das portas AND. Desse modo, uma dessas portas produzirá uma saída de valor verdadeiro. Se as duas portas AND tiverem suas saídas ligadas a uma porta OR, a saída da porta OR somente terá o valor verdadeiro se uma e apenas uma das entradas for verdadeira.

Agora, já temos quase o circuito desejado: o único problema é que uma entrada de dois dígitos 1, embora resultando em uma "soma" correta 0, não produz o sinal de transporte. Todavia, o acréscimo de uma porta AND, ligada paralelamente às duas entradas, produzirá esse sinal, quando e somente quando ambas as entradas tiverem valor verdadeiro. A tabela de validação do circuito, na ilustração, chamada semi-somador, é a seguinte:

X (entrada 1)	Y (entrada 2)	T (saída "transporte")	S (saída "soma")
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Somador de 8 bits

O computador realiza a adição dígito por dígito. Na ilustração, são somados dois números binários, de 8 dígitos. A linha superior contém o primeiro número, e a segunda, o outro. Trabalhando da direita para a esquerda, não pode haver "transporte" para acrescentar à adição dos dois primeiros dígitos; assim, um semi-somador é suficiente. Porém, para o par seguinte e os subseqüentes, é necessário o uso de somadores completos (qualquer "transporte" é indicado por um dígito 1). E a resposta encontra-se na última linha.

Tem o nome de semi-somador porque, de certo modo, é adequado apenas parcialmente. Se pretendêssemos apenas a soma de uma coluna única de dois dígitos binários, seria suficiente. Entretanto, geralmente queremos somar 2 bytes de dados e cada byte contém 8 bits. O somador que controla a coluna da extrema direita necessitaria, na verdade, de apenas um semi-somador. Todavia, todos os somadores à esquerda dessa coluna têm de estar em condições de admitir três entradas: os dois dígitos de "sua" coluna e eventuais dígitos de transporte da coluna à direita. Observemos esta adição:

```
  011
+ 111
 1010
```

Ao somar a coluna "um", dizemos 1 e 1 é 0, transportamos o 1 e escrevemos 0 sob a coluna "um". Ao somar a coluna "dois", dizemos 1 e 1 é 0, transportamos o 1, que somado ao transporte da coluna "um" dá 1, restando 1, de transporte. Escrevemos 1 sob a coluna "dois" e transportamos o número 1 para a coluna "quatro". Aqui, dizemos 0 e 1 é 1, que, somado ao número de transporte da coluna "dois", é 0, restando 1, de transporte. Escrevemos o 0 na coluna "quatro", restando 1 de transporte, que podemos escrever sob a coluna "oito". Ou seja, a tabela de validação para um "somador completo", capaz de operar tanto transportes como dois dígitos binários, seria a seguinte:

X	Y	T (entrada)	T (saída)	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

O somador completo pode ser obtido pelo uso de dois semi-somadores, além de uma porta OR suplementar. Para cada somador completo, a "saída do transporte" está diretamente ligada à "entrada do transporte" do somador à sua esquerda; desse modo, podem ser encadeados tantos somadores completos quantos necessários.

Nos microcomputadores modernos, a maior parte das somas e outras operações aritméticas é realizada com um grande número de circuitos de somadores, idênticos em seu princípio de funcionamento aos descritos acima. O interior de uma CPU comum consiste em um simples chip de silício que incorpora pequenas áreas de memória RAM e ROM, um número muito grande de circuitos interruptores e a ALU (Unidade Aritmética e Lógica), que é a parte que contém todas as portas lógicas e somadores necessários ao computador, para realizar cálculos e decisões lógicas.

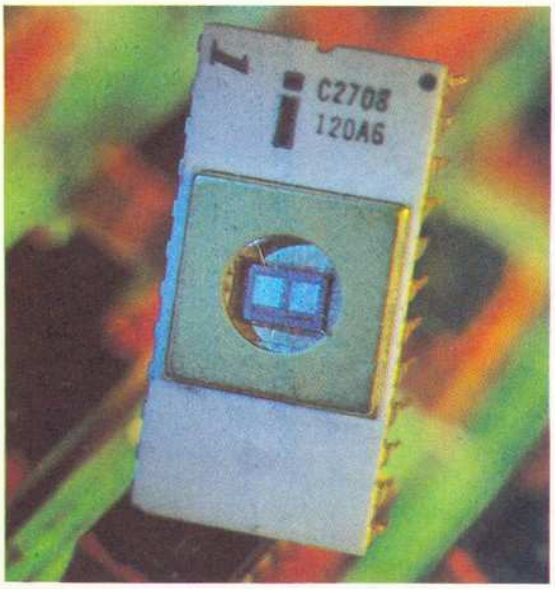
Lógica misteriosa

O computador armazena na memória milhares de bytes de informação. E consegue lembrar onde está cada uma delas!

Uma das formas de referência à memória do computador é o armazenamento a curto e a longo prazo. Este segundo tipo não perde a informação, podendo retê-la por longos períodos de tempo, até mesmo quando a energia é desligada. A fita magnética e o disco flexível enquadram-se nessa categoria. Mas os computadores também necessitam de memória para armazenamento temporário dos programas e resultados.

Outra forma de referência à memória do computador é considerá-la como interna ou externa. A memória interna, localizada no interior do computador, é totalmente "eletrônica"; a externa, como o próprio nome diz, é periférica ou fora do computador. A externa quase sempre é parcialmente mecânica, envolvendo mecanismos como fita cassete, unidades de disco, ou fitas perfuradas e leitoras.

A memória interna eletrônica é geralmente chamada de memória principal, enquanto a externa é mencionada como memória secundária, memória



EPROM
O inconveniente das ROMs comuns é que o conteúdo da memória é "embutido" e não pode ser alterado. A EPROM (Erasable Programmable Read Only Memory) é bem mais flexível. Uma vez programada, pode ser reprogramada apagando-se o conteúdo e "escrevendo" um novo programa. A EPROM possui uma "janela" de sílica que permite que raios ultravioleta atravessem o interior, fazendo com que os capacitores que armazenam os bits na EPROM se descarreguem. Na ausência da luz ultravioleta, os capacitores mantêm a carga indefinidamente, e o conteúdo da memória é retido.

auxiliar ou memória de reserva. Atualmente, há dois tipos de memória interna: RAM e ROM. Ambas são dispositivos inteiramente eletrônicos, na forma de chips de silício, colocados em caixas retangulares de plástico, com conjuntos de terminais paralelos de estanho ou banhados com prata. Há outras semelhanças, na forma como são selecionadas e "endereçadas" pela CPU do computador, e falaremos delas mais adiante.

A principal diferença funcional é que os chips da memória ROM são utilizados para armazenar programas de forma permanente. O padrão de zeros e uns em cada local da memória é fixo e ajustado na fabricação, e não pode, conseqüentemente, ser alterado. As ROMs podem ser consideradas verdadeiras "bibliotecas de referência" do computador. Este consulta o conteúdo ROM mas não é capaz de "escrever" qualquer coisa nela.

ROM significa Read Only Memory (Memória de Leitura Somente). Ler é o termo utilizado para indicar o que o computador faz ao "acessá-la" ou recuperar as informações da memória. As ROMs podem ser de tipos ligeiramente diferentes e algumas têm programas que podem ser removidos ou apagados, e depois reprogramados. Uma ROM típica é a 2364 da Intel. É um chip descrito como sendo uma ROM de 65.536 bits, organizados em 8 Kbytes de 8 bits. O que significa que 64 Kbits são agrupados em 8 bits, e cada local "endereçável" acessa ou lê um byte completo. Em matemática, 1 K = 2^{10} (2 elevado à décima potência) ou 1.024; assim, 64 K = 64 × 1.024 ou 65.536.

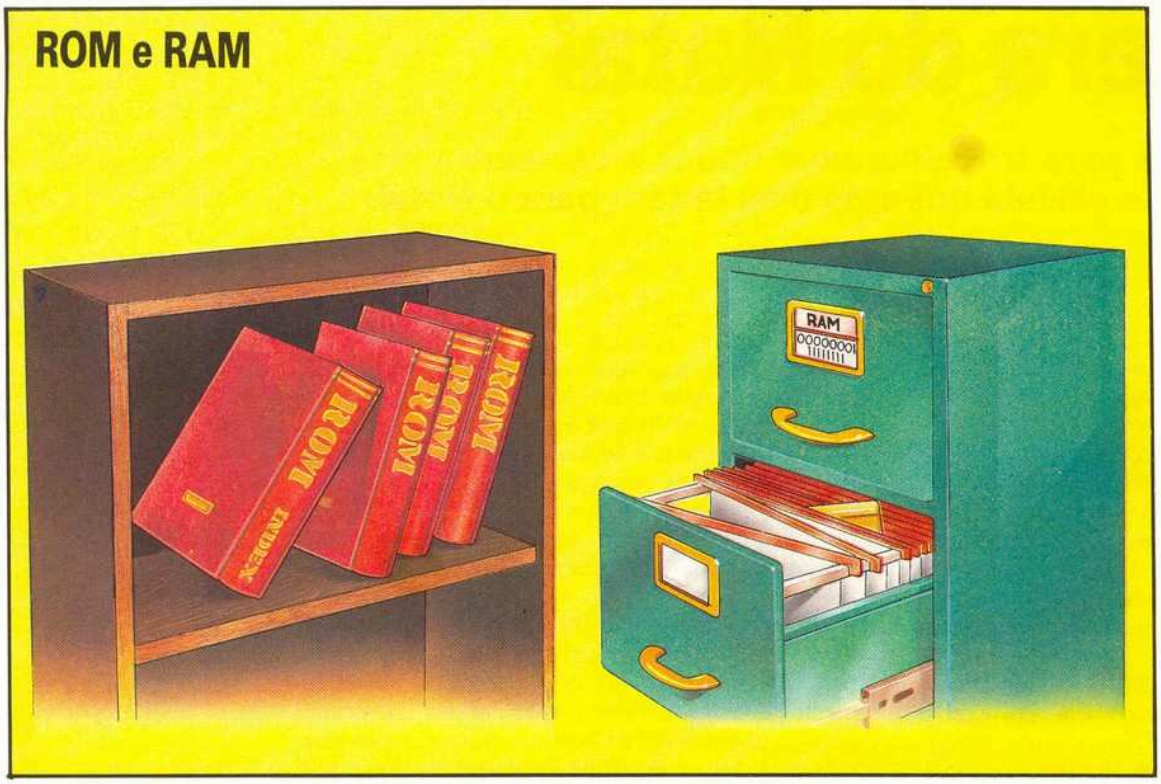
Portanto, o computador é capaz de selecionar qualquer um dos 8.192 (8 K) endereços. Um exame mais minucioso das especificações do chip 2364 revela que possui 28 pinos, com um reservado para fornecimento de energia de +5 volts, e um para a conexão do fio terra, o que dá o total de 26 pinos. Cada byte contém 8 bits, de modo que, quando 1 byte é lido do chip, os 8 bits deste byte devem ser conduzidos por meio de fios do chip à CPU. Conseqüentemente, há oito fios para transportar os bits no byte que está sendo lido pela CPU. Estes fios são chamados de "data bus" (bus de dados). Oito dos pinos no chip são dedicados a essa tarefa, um para cada bit no byte.

Isto nos deixa dezoito pinos. Um pino não é necessário e não está conectado. Ele existe, pois é mais fácil fabricar chips com um número par de pinos. Quatro pinos são utilizados para "selecionar" o chip de diversas maneiras. Estes são o pino de "capacidade de saída", o de "capacidade de chip" e dois de "seleção de chip". Estes pinos recebem os sinais do computador e possibilitam ao chip "saber" quando está sendo requerido.

Os pinos restantes são os de "endereço". Cada pino é conectado a um fio de "address bus" (bus de endereço) e estes transportam o endereço do byte necessário, codificado na forma binária. Treze dígitos binários podem fornecer 2^{13} ou 8.192 combinações únicas dos algarismos 1 e 0, de modo que as treze linhas de endereço são suficientes para selecionar cada um e todos os 8.192 bytes armazenados na ROM.

As RAMs são os quadros-negros do computador. Ou seja, programas e dados são armazena-

ROM e RAM



Memória de longo prazo
A ROM (Read Only Memory, Memória de Leitura Somente) é análoga a um livro em que estão "armazenadas" as informações permanentemente. Não se pode alterar ou remover os dados, nem alterar as palavras impressas nas páginas.

Memória de curto prazo
A RAM (Random Access Memory, Memória de Acesso Aleatório) mais se parece com um sistema de arquivo, uma vez que as informações podem ser alteradas, e os dados não são permanentes. A RAM fica limpa quando o computador é desligado.

dos neles temporariamente, enquanto o computador está trabalhando, e os resultados e outros dados são também "escritos" temporariamente. A memória RAM, em geral, é mais complexa internamente que a ROM, pois cada bit em cada byte da RAM deve ser capaz de alterações, se assim for "escrito".

Um chip típico RAM é o Intel 2114. Cada chip RAM 2114 contém 4.096 bits de memória, e estes são organizados como 1.024 "nibbles" (meios bytes) de 4 bits. Isto significa que cada endereço dará saída a 4 bits de dados. Dois destes chips serão necessários para produzir 1 Kbyte total de dados.

Cada chip 2114 possui apenas dezoito pinos, dois dos quais são para fio terra e fornecimento de energia. Quatro são usados para linhas de dados de entrada/saída. Um é utilizado para sinal de seleção de chip (o sinal diz ao chip que este é solicitado ou "selecionado") e outro diz ao chip quando deve ser escrito ou lido. Os dez pinos restantes são empregados para os address bus.

Dez linhas de endereço podem identificar unicamente 2^{10} locais, ou 1.024. Se o computador possuir 64 Kbytes de RAM, e tiverem sido utilizados chips 2114 Intel, um total de 128 chips RAM seriam necessários, já que 2 chips formam um byte completo. Atualmente, é mais comum empregar chips RAM de maior densidade, que contêm mais memória no mesmo espaço. Com os chips RAM modernos, tal como o 4164, é possível obter 64 Kbytes de RAM com apenas 8 chips.

Os chips RAM e ROM têm-se tornado cada vez mais compactos e de menor custo e, agora, existem 128 Kbits em um único chip. Contudo, o acondicionamento de densidades mais altas em chips únicos não poderá evoluir indefinidamente. O circuito no silício tem ficado tão diminuto que as técnicas ópticas empregadas para "gravar" os circuitos estão chegando ao seu limite.

Os chips de memória de alta densidade do futuro serão provavelmente fabricados utilizando-se feixes de elétron ou raios X.

De modo mais abrangente, há dois tipos de memória RAM em uso: a RAM estática e a RAM dinâmica. Há vantagens e desvantagens em ambas, mas a RAM dinâmica é atualmente mais utilizada. Ambos os tipos apagam o conteúdo da memória, logo que o fornecimento de energia é interrompido, mas a memória dinâmica precisa ter seu conteúdo "restaurado" em intervalos de alguns milissegundos.

Cada bit na memória precisa ser restaurado ou reescrito sem diminuir a velocidade com que a CPU acessa os dados ali contidos. Isto significa que um circuito de controle de tempo muito especial deve ser projetado, o que torna o trabalho do projetista do circuito mais difícil.

A memória dinâmica oferece duas vantagens em relação à memória estática. A dinâmica requer apenas um transistor por bit, ao passo que três transistores são normalmente necessários para cada bit na memória estática. Isso permite que maior quantidade de memória seja colocada em chips menores. A maioria dos chips RAM dinâmicos possui apenas dezesseis pinos. A outra vantagem da RAM dinâmica é utilizar menos energia, comparada a sua equivalente estática. Portanto, gera menos calor e tem baixo consumo de força.

A vantagem da RAM estática reside na simplicidade do projeto do circuito. Escritos os conteúdos da memória, ali permanecem, sem a necessidade de ser restaurados. Cada célula de memória de 1 bit requer três transistores; assim sendo, é difícil conseguir as altas densidades que a RAM dinâmica permite. A RAM estática consome energia, o calor extra produzido complica o sistema de resfriamento do computador e pode demandar o uso de um ventilador para resfriamento, tornando o modelo mais dispendioso.

Diálogo digital

A Unidade Central de Processamento controla todos os procedimentos de entrada e saída do computador.

Analógico para digital
No mundo real, poucas informações vêm em unidades digitais discretas. Pelo contrário, elas são muito variáveis como os níveis de ruído, ou as marés.

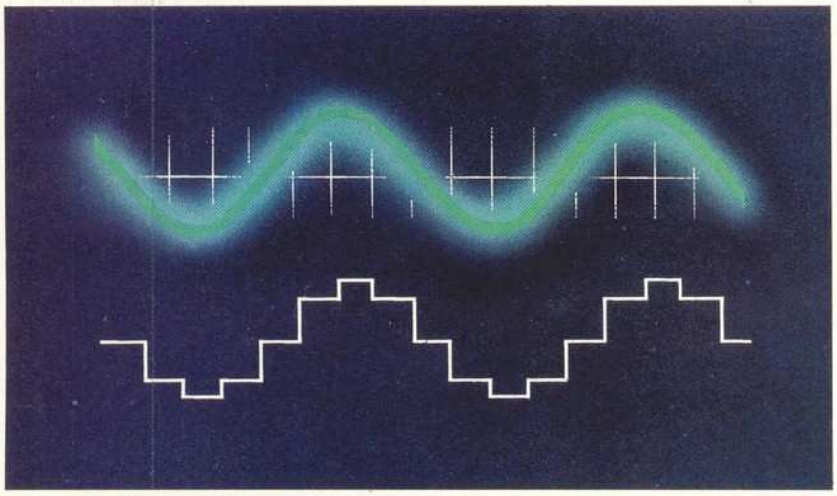
Para que esses dados sejam compreendidos pelo computador, o sinal deve primeiramente ser digitalizado. O conversor de analógico para digital (A/D) recebe amostras de uma fonte de sinais numa taxa constante, talvez 100 por segundo. Cada uma dessas amostras é depositada num local separado de memória como um valor digital, permitindo que cálculos de variação sejam feitos e as condições fora do limite sejam conhecidas.

Os conversores de digital para analógico (D/A) trabalham de modo similar, mas ao contrário, usando técnicas estatísticas para encaixar os picos numa curva regular.

Input/Output, ou I/O (abreviação comum), são os termos usados para indicar a transferência de informação entre a CPU (a Unidade Central de Processamento que forma o centro do computador) e o "mundo exterior". Aqui, "mundo exterior" significa qualquer aparelho ligado ao computador. Nesse caso não se incluem as memórias ROM e RAM, consideradas partes integrantes do computador. A distinção entre o que faz parte ou não do computador é arbitrária. Mas todos os circuitos lógicos (ver p. 92), que trabalham em estrita conexão com a CPU e a memória central, são considerados parte integrante do computador.

Os equipamentos externos que têm I/O para a comunicação com o computador incluem uma variedade de periféricos: teclado, unidades de discos flexíveis, joysticks, impressora e unidades de vídeo.

Quando a CPU quer acessar dados da memória, a primeira coisa que faz é "endereçar" o local onde o byte de dados é depositado. Do mesmo modo, caso a CPU queira depositar um byte de dados para uso posterior, ela deve primeiramente estabelecer o local onde será depositado. Este processo recebe o nome de "endereçamento de memória", e pressupõe que a CPU coloque os dígitos binários de acordo com a posição de memória desejada num conjunto de dezesseis fios ligados aos "pinos de endereço". Estes fios recebem o nome de "address bus". Um circuito especial na seção de memória é capaz de decodificar estes dezesseis dígitos binários para que selecionem a posição de memória correta. (Dezesseis dígitos binários possibilitam 65.536 combinações dos algarismos 0 e 1 e podem endereçá-las às diferentes locações de memória.)



Caso o computador queira estabelecer comunicação com um equipamento externo, ele terá de "localizá-lo" da mesma forma. Somente oito linhas de endereço estão disponíveis; isso permite que o número total de locações I/O separadas que podem ser selecionadas seja de 256. Este número é pequeno, comparado ao poder de dezesseis linhas de endereço; mas, na prática, 256 é uma quantidade adequada. Não há necessidade de ser conectada ao computador uma quantidade maior de equipamentos.

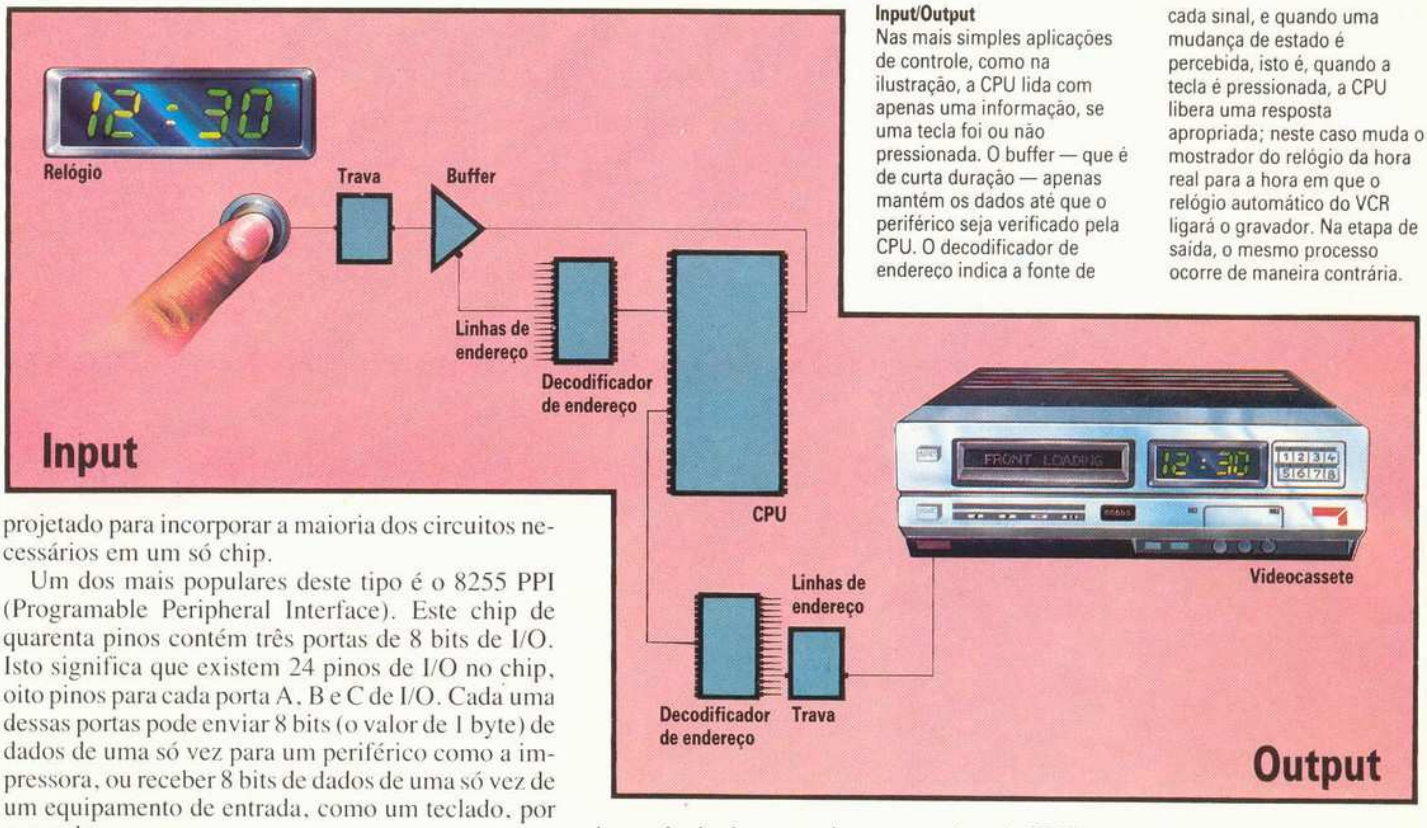
A escolha dos equipamentos

Para descobrir como, na verdade, o computador seleciona um equipamento externo e envia dados para ele, consideremos um dos mais simples equipamentos de output — um LED (Diodo de Emissão de Luz) colocado no teclado do computador para mostrar quando a tecla de maiúscula é pressionada.

Para o computador, o LED é apenas outro dispositivo externo a que envia dados. No caso de um único LED, o dado será um único 1 para ligar o LED; ou um único 0 para desligá-lo. Apesar de ser apenas um LED requerendo um bit de dados, ele precisa ter endereço e posição. A CPU não pode gastar todo seu tempo endereçando um LED. Ela seleciona o LED para que possa dizer quando ele deve ser ligado e depois desligado. Admitamos que o LED tenha um endereço de I/O de 32. A fim de selecioná-lo, as linhas de endereço são estabelecidas pela CPU através do binário equivalente a 32, ou seja, 00100000. O LED terá um circuito "decodificador" especial que irá ignorar todas as outras combinações de bits nas linhas de endereço. Quando a linha de endereço torna-se 00100000, este circuito decodificador a reconhece, produz uma alta voltagem e, conseqüentemente, um output "verdadeiro".

A parte seguinte do circuito que precisa ser ligada no LED é um pequeno chip chamado trava de dados ("data latch"). Este prende o dado enviado, para que o LED fique ligado ou desligado até o próximo endereçamento e novos dados sejam enviados a ele. Este processo é chamado "toggling".

A maioria dos equipamentos externos com que o computador se comunica são mais complexos que um simples LED. A impressora é considerada um periférico típico e, toda vez que o computador se comunica com ela, os dados transmitidos irão representar o código de um caractere a ser impresso. Geralmente, quando uma grande quantidade de dados deve ser transferida, como para a impressora, por exemplo, um chip especial de interface I/O é utilizado. Tais chips simplificam a tarefa do engenheiro de computador, já que o circuito interface é



Input/Output
Nas mais simples aplicações de controle, como na ilustração, a CPU lida com apenas uma informação, se uma tecla foi ou não pressionada. O buffer — que é de curta duração — apenas mantém os dados até que o periférico seja verificado pela CPU. O decodificador de endereço indica a fonte de cada sinal, e quando uma mudança de estado é percebida, isto é, quando a tecla é pressionada, a CPU libera uma resposta apropriada; neste caso muda o mostrador do relógio da hora real para a hora em que o relógio automático do VCR ligará o gravador. Na etapa de saída, o mesmo processo ocorre de maneira contrária.

projetado para incorporar a maioria dos circuitos necessários em um só chip.

Um dos mais populares deste tipo é o 8255 PPI (Programable Peripheral Interface). Este chip de quarenta pinos contém três portas de 8 bits de I/O. Isto significa que existem 24 pinos de I/O no chip, oito pinos para cada porta A, B e C de I/O. Cada uma dessas portas pode enviar 8 bits (o valor de 1 byte) de dados de uma só vez para um periférico como a impressora, ou receber 8 bits de dados de uma só vez de um equipamento de entrada, como um teclado, por exemplo.

A fim de enviar 8 bits de dados para uma impressora, a CPU primeiro terá de endereçar o PPI; só então enviará a ele os 8 bits de dados através do "data bus". Esses dados serão depositados temporariamente numa célula de memória de 1 byte, dentro do chip, chamada registrador. A seguir, o PPI fará com que esse dado esteja disponível a um grupo apropriado de pinos de I/O.

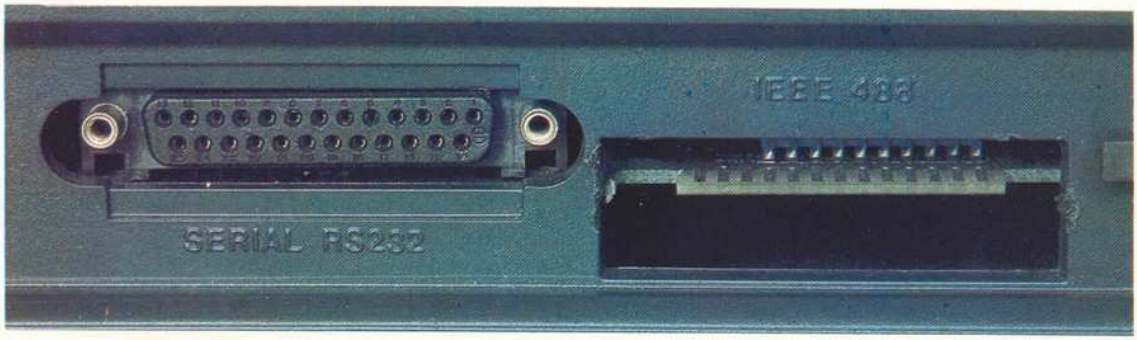
Um princípio similar, mas que trabalha ao contrário, permite que dados de periféricos de entrada sejam armazenados num registrador no chip, e depois colocados no "data bus", quando a CPU lhes enviar o sinal apropriado.

Como se nota acima, os periféricos não podem ser liberados para colocar continuamente seus dados no "data bus" do computador; é necessário que se permutem dados da memória e de outros equipamentos de I/O. O chip de I/O armazena os dados temporariamente, e só coloca os dados no "data bus" (para serem colhidos pela CPU) quando a CPU ordena que isso seja feito.

Como a CPU sabe se um periférico está tentando enviar dados para o computador? Existem duas técnicas principais que podem ser usadas. A CPU interrompe periodicamente a execução do programa e verifica todas as portas de entrada. Se ela encontra dados à espera, instrui a porta para que os coloque no "data bus". O processo de inquirição dos periféricos de entrada é conhecido como "polling".

A outra técnica utiliza "interrupções". O equipamento que está aguardando envia um sinal de interrupção diretamente à CPU, o que força o programa que está sendo executado a parar, enquanto a porta de entrada envia seus dados. As vantagens e desvantagens de ambos os métodos serão discutidas no curso, mais adiante. O I/O descrito até agora é chamado "I/O paralelo", porque os dados entram ou saem de 1 byte por vez, usando oito fios ou linhas de I/O (8 bits paralelos).

Uma outra técnica recebe o nome de "I/O serial". Neste caso, a informação é alimentada em 1 bit de cada vez, 1 bit após o outro. Algumas impressoras utilizam interfaces seriais, e a saída de modems (ver p. 108) também é serial. A vantagem principal é que, essencialmente, a comunicação serial permite que um único par de fios seja usado, em vez de oito ou mais.

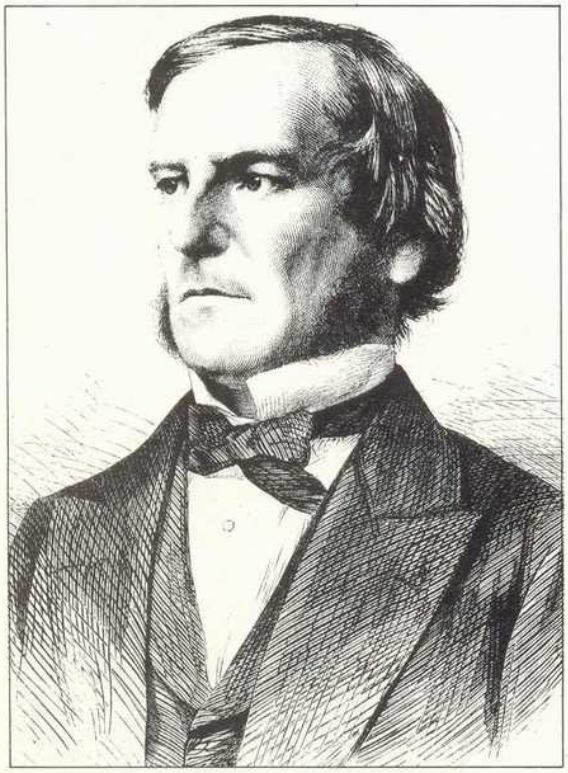


Portas paralelas e seriais
A maioria dos microcomputadores modernos oferece tanto as portas paralelas como as seriais. As portas seriais transmitem os dados em 1 bit de cada vez, e as paralelas em bytes completos. O tipo mais comum de porta serial é conhecido como RS232 C, usa um conector do tipo "D", um exemplo de 25 pinos (ilustração à esquerda), ou, mais raramente, um plugue DIN semelhante aos utilizados em aparelhos de som.

A porta paralela (à direita) segue a convenção IEEE 488, desenvolvida por Hewlett Packard e adotada como modelo industrial pelo Instituto de Engenharia Elétrica e Eletrônica, dos Estados Unidos.

Leis do pensamento

Um século antes da invenção dos modernos equipamentos eletrônicos, Boole expunha suas idéias sobre lógica matemática.



George Boole, 1815-1864
George Boole nasceu em uma época em que não era possível imaginar os computadores eletrônicos; ainda assim, ele é um dos fundadores da lógica matemática usada nos computadores de hoje. De família modesta, estudou matemática sozinho, nas horas vagas. Estava convicto de que os processos de pensamento de que nos valemos cotidianamente estão fundados na razão e que esta poderia ser depurada até alcançar a forma da lógica matemática. Publicou suas idéias em 1847 e tornou-se famoso da noite para o dia, sendo convidado para primeiro professor de matemática da nova Universidade de Cork, na Irlanda.

Em 1815, ano da derrota de Napoleão na batalha de Waterloo, outro acontecimento importante teve lugar na História da humanidade.

Nesse ano nasceu George Boole, em Lincoln, Inglaterra. Filho de um sapateiro, estava destinado a tornar-se um dos gênios que possibilitaram, mais tarde, a invenção do computador. Morreu em 1864, um século antes da revolução do microcomputador; entretanto, sem suas idéias o moderno computador não teria as características que tem hoje.

Boole sabia que os nossos processos de raciocínio na vida cotidiana podem ser representados em termos de lógica formal, desenvolvida primeiramente pelos gregos. Ele acreditava que, a rigor, poderíamos mesmo chegar a exprimir o raciocínio humano em termos matemáticos. Boole dispôs-se a fazer exatamente isso; estudou matemática sozinho e iniciou sua pesquisa sobre a lógica dos processos de pensamento.

Conjuntos de informação

Suponha que uma determinada pessoa vai a uma festa. Ela quer dançar e, na sala de dança, procura um parceiro.

Há pessoas ali que estão dançando ou não — porque não podem estar fazendo ambas as coisas ao mesmo tempo. O par procurado será uma mulher — ou um homem.

Boole examinou essas alternativas de outro modo. Para ele, o salão contém "conjuntos" de pessoas, o conjunto de homens e o conjunto de mulheres, que podem ser simbolizados por H e M. Boole também levou em conta o conjunto das pessoas que estão dançando (D) e o conjunto das que estão esperando (E) para dançar.

O parceiro de dança (de um homem) teria de satisfazer duas condições: ser mulher **e** também estar esperando para dançar. Boole percebeu a importância do conetivo **e** ("and") que liga as duas condições e atribuiu-lhe um símbolo: a letra U invertida. Com isto, tornou-se possível simbolizar o conjunto de eventuais pares como sendo $M \cap E$.

Entretanto, se não quisesse dançar, mas apenas conversar com uma pessoa amiga, o homem poderia escolher uma pessoa dos conjuntos H **ou** M, porque estes dois incluem qualquer pessoa na sala. Aqui, novamente, Boole notou a importância do conetivo **ou** ("or") e atribuiu-lhe o símbolo U. Assim, em sua lógica algébrica, H U M inclui todos os homens e todas as mulheres presentes na sala.

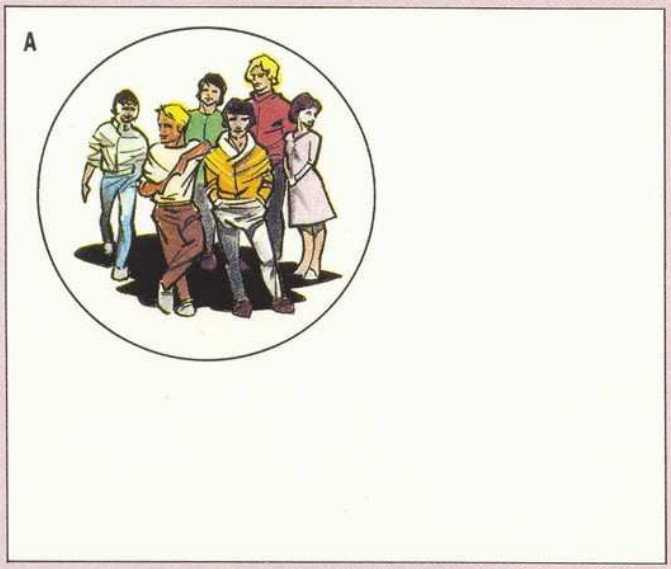
As portas lógicas encontradas no computador são representadas de acordo com símbolos criados por Boole, tais como AND e OR. Na seção de programação em linguagem BASIC, examinaremos as funções desses dois úteis conetivos AND e OR. Porém, antes de prosseguir, há uma interpretação interessante da lógica booleana — feita pelos matemáticos John Venn (1834-1923) e Charles Dodgson (1832-1892), mais conhecido como o escritor Lewis Carroll.

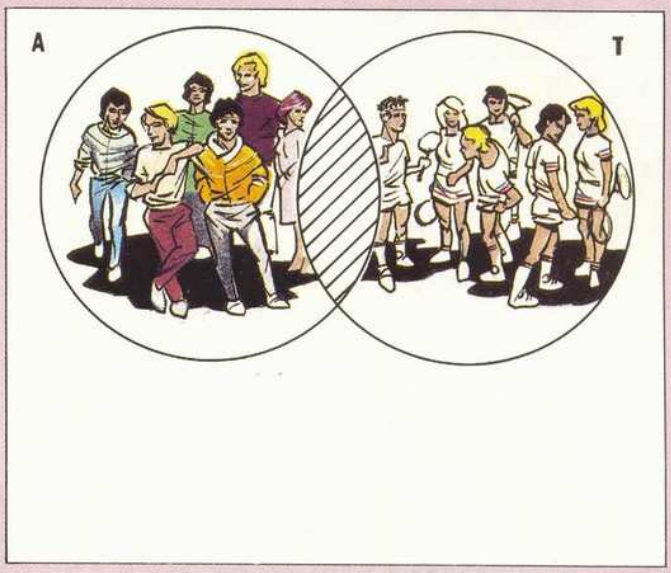
Examinemos um problema prático. Vamos supor que você tenha armazenado uma lista de pessoas conhecidas na memória do computador. Junto a cada nome estão relacionados outros dados, como nú-

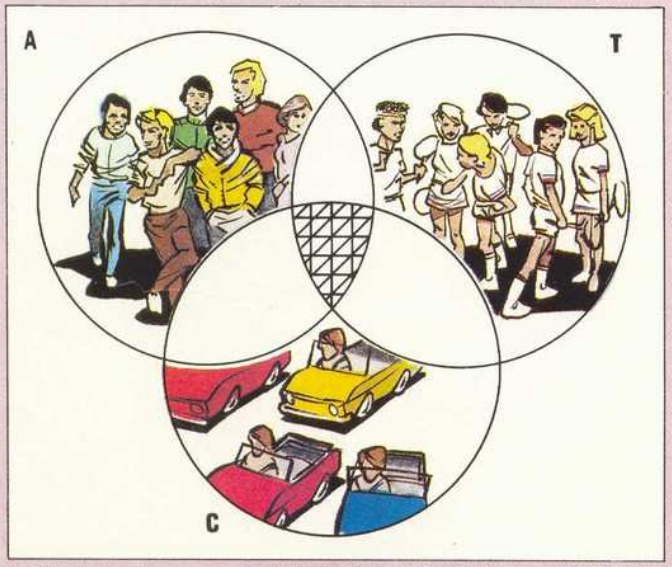
O programa Amigos

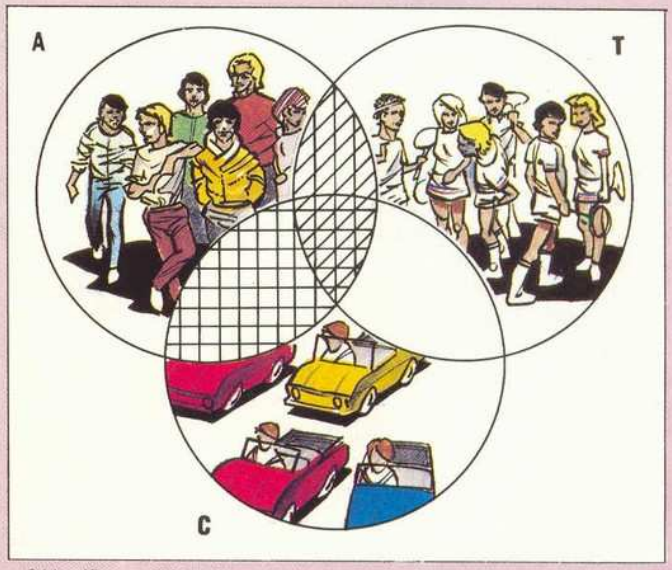
```
10 DIM N$(10), D$(10), A$(10), T$(10), C$(10)
15 REM NOME, NO. TEL., AMIGO?,TENIS?, CARRO?
17 PRINT "FORNECER DETALHES NO FORMATO:"
18 PRINT "NOME, TELEFONE, SIM/NAO, SIM/NAO, SIM/NAO"
20 FOR K = 1 TO 10
30 INPUT N$(K), D$(K), A$(K), T$(K), C$(K)
40 NEXT K
45 REM ROTINA DE PESQUISA
50 FOR J = 1 TO 10
60 IF A$(J)="SIM" AND T$(J)="SIM" AND C$(J)="SIM" THEN GOSUB 100
70 NEXT J
80 END
100 PRINT N$(J), D$(J)
110 RETURN
```

Diagramas de Venn









O quadro representa o conjunto de todas as pessoas cujos nomes estão armazenados no computador e é chamado, nos diagramas de Venn, de conjunto Universo. Os círculos no interior do quadro representam conjuntos individuais. Os amigos estão no conjunto A. Nem todas as pessoas jogam tênis, mas as que o fazem estão no conjunto T. Uma vez que há pessoas que tanto jogam tênis como são amigas, os dois círculos se sobrepõem (A ∩ T). O conjunto C identifica as pessoas que têm carro. Este conjunto se sobrepõe aos outros dois e as pessoas que satisfazem as três condições estão incluídas na região em que os três círculos se interseccionam (A ∩ T ∩ C). O último diagrama representa os mesmos conjuntos (A,T,C), mas as condições que devem ser satisfeitas são outras. Você resolveu jogar tênis ou passear de carro. O conjunto de conhecidos que são amigos e jogam tênis está riscado com linhas diagonais. O conjunto de amigos que têm carro está riscado com linhas horizontais. A combinação dessas duas áreas representa os amigos com que você poderá jogar tênis ou passear de carro.

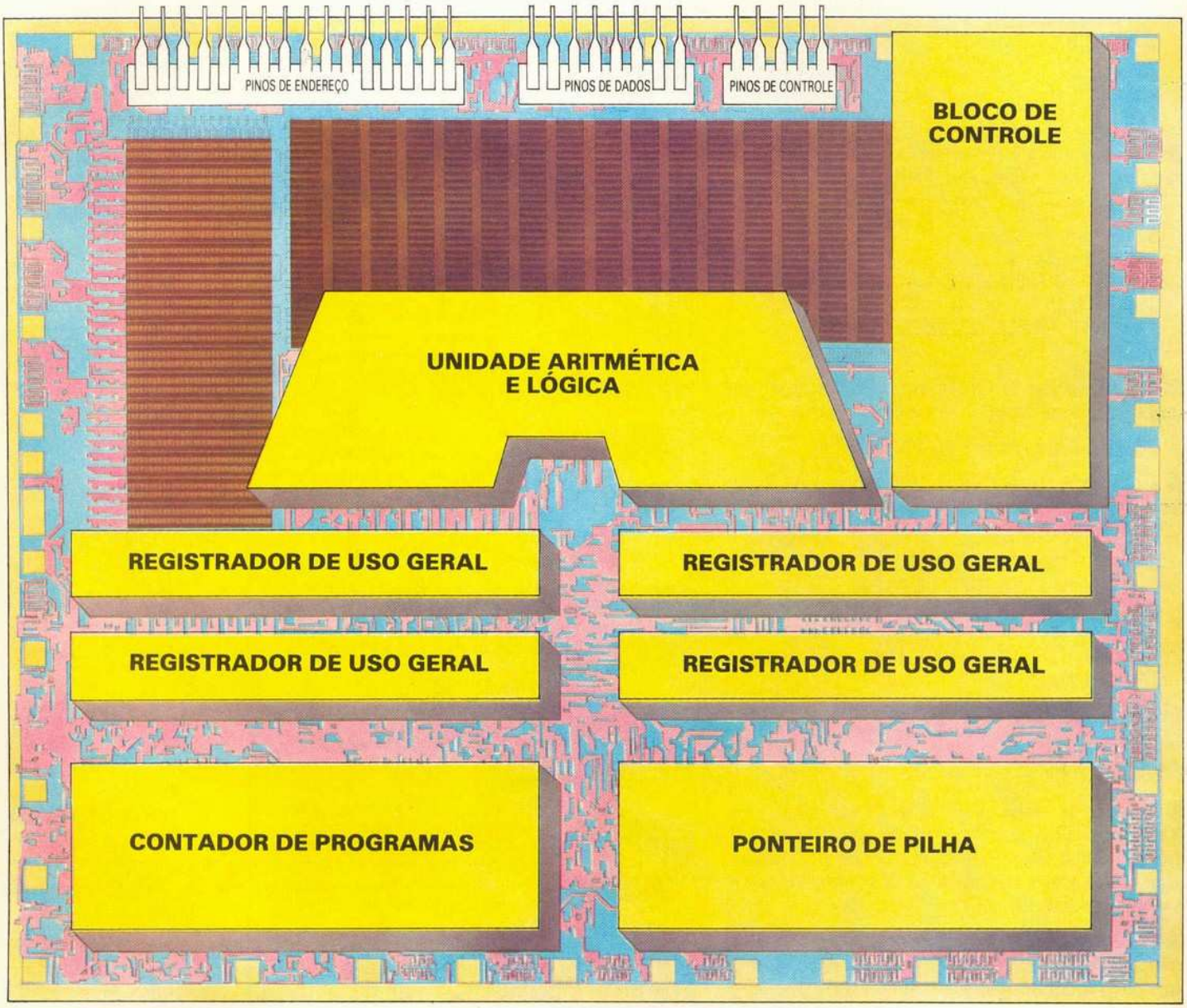
mero de telefone, hobbies etc. Uma tarde, você decide jogar tênis em um clube do outro lado da cidade. Você necessita de um amigo (não de um simples conhecido) que jogue tênis e tenha carro. Seu computador recebe a instrução para selecionar o nome e o telefone de todas as pessoas que satisfaçam as três condições: jogar tênis E (AND) ter carro E (AND) ser amigo.

O programa (na página anterior) primeiramente busca dados sobre todos os conhecidos: são amigos, têm carro, jogam tênis? O programa prevê um total de dez conhecidos, mas você pode mudar esse número para quantos quiser (desde que não se esqueça de mudar o número 10 entre parênteses, na instrução DIM, linha 10). A lista é então examinada pelo uso da instrução IF-THEN, na qual foi introduzida uma condição múltipla. A maioria dos programas em linguagem BASIC admite que o comando IF-THEN opere com condições constituídas de subcondições conectadas por AND e OR. Finalmente, são apresentados o nome e o número telefônico dos conhecidos que satisfazem todas as condições — são amigos, jogam tênis e têm carro.

Em alguns programas, encontram-se combinações muito complexas de funções lógicas. Só na era do computador é que a álgebra de Boole, que em sua época era considerada uma curiosidade, veio a ter o merecido reconhecimento.

O centro nervoso

Todas as atividades de computação se dirigem para a Unidade Central de Processamento ou dela provêm.



As atividades do computador são controladas pela CPU (Central Processing Unit), Unidade Central de Processamento. Os termos significam: Unidade — auto-suficiente; Central — o coração do equipamento; Processamento — execução do trabalho. Um computador muito simples (ver ilustração) poderia consistir em apenas uma CPU, alguma capacidade de memória e um sistema de circuitos I/O (entrada e saída).

O computador necessita do circuito I/O para sua comunicação com o exterior. Um exemplo bem simples: o computador utilizado em uma máquina de lavar roupa precisa de um circuito I/O para fazer funcionar o motor e os aquecedores. A memória é necessária para o armazenamento de instruções e de dados para processamento pela CPU. Os dados processados pela CPU podem incluir números e códigos binários que representam caracteres (letras, dígitos e sinais, como "/" e o sinal "!").

Se algumas posições da memória contêm instruções para a CPU e outros dados, como pode ela saber quais são as instruções e quais os dados? Para responder a esta questão, devemos examinar o interior de um microcomputador.

As CPUs de microcomputadores de 8 bits (a maioria dos microcomputadores) estão geralmente contidas em um único chip com quarenta pinos, vinte de cada lado, nas laterais mais longas. Cada um deles (com exceção dos pinos de abastecimento de energia de 0 e de +5V) transporta sinais para dentro e para fora da CPU procedentes de outros dispositivos, como a memória e os circuitos I/O.

As CPUs comuns de 8 bits têm dezesseis pinos de "endereço", que são conectados ao "address bus". Cada um desses pinos transporta um sinal de saída, que representa um dígito 1 ou um dígito 0. Pela combinação dos dois, pode-se chegar a 65.536 combinações, usadas para selecionar posições específicas na memória.

Há igualmente oito pinos de "dados", ligados ao "data bus". Os pinos de dados transportam os dados da memória ou dos circuitos I/O para a CPU e vice-versa.

Uma série de outros pinos transporta sinais de "controle". Alguns desses sinais são saídas da CPU e outros são entradas. Examinaremos resumidamente como os sinais de controle são usados.

No interior da CPU, existem algumas células de memória, de 1 ou 2 bytes, chamadas registradores, algumas delas reservadas para finalidades especiais. As demais são usadas para armazenamento temporário de informações e são denominadas registradores de finalidades gerais. Há dois outros importantes "blocos" de função na CPU, a ALU e o "bloco de controle".

A sigla ALU (Arithmetic and Logic Unit) representa a Unidade Aritmética e Lógica. Esta parte da CPU efetua operações lógicas e aritméticas, inclusive (mas não só) a soma, as operações com os conetivos AND e OR, e a mudança dos bits no interior do byte, para a direita ou para a esquerda.

O bloco de controle é um circuito especial projetado para fazer com que o comportamento da CPU esteja de acordo com a instrução recebida da memória. Podemos tomar um exemplo específico, usando os códigos de instrução da conhecida CPU Z80. Se receber da memória a instrução em código 11000110, a CPU somará o conteúdo do byte seguinte da memória ao conteúdo de um dos registradores no interior da CPU. Se quisermos, então, armazenar o resultado dessa adição em uma posição específica da memória, a instrução que a CPU deverá receber a seguir será 00110010, seguida de 2 bytes que especificam a posição efetiva na memória, na qual desejamos armazenar o resultado.

Admitamos que o resultado da soma seja 37 (pela notação decimal) e que os 2 bytes que seguem a instrução especifiquem a posição de endereço 33126 (novamente usando a notação decimal). O código de instruções fará com que o bloco de controle ajuste os pinos de endereço para seu equivalente binário 33126 (ou seja, 1000000101100110). Os pinos de controle enviarão sinais indicando à memória que deve aguardar dados e armazená-los. Em seguida, os pinos de dados assumem a forma do dígito binário equivalente a 37 (00100101). Esses dados passam, a seguir, através do "data bus", para a memória e são armazenados na posição especificada pelo "address bus". Se, em um estágio posterior, a CPU necessitar desses dados para processá-los de alguma outra forma (mostrá-los na tela ou imprimi-los, por exemplo), uma instrução diferente poderá ser enviada à CPU. O bloco de controle interpretará este procedimento como "acessar o endereço de memória 33126, obter seu byte e armazená-lo temporariamente em um dos registradores internos".

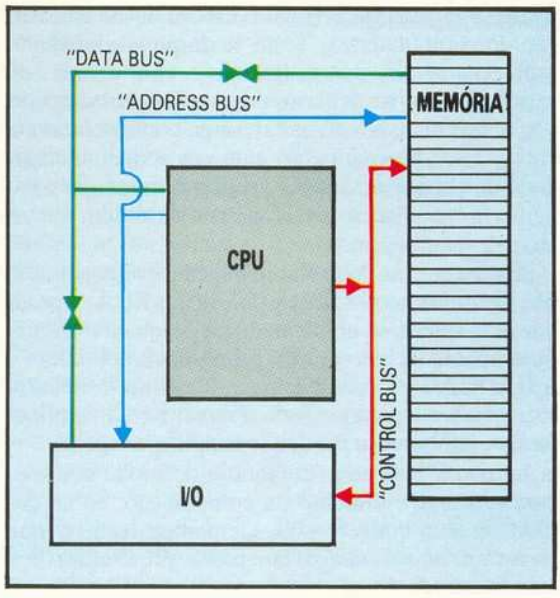
O total de registradores ou de células de memória temporária no interior da CPU depende da CPU. Podem ser registradores de 8 bits (1 byte) ou de 16 bits (2 bytes). Os registradores especializados geralmente recebem nomes especiais, tais como ponteiro de pilha ("stack pointer"), contador de programas ou acumulador. Os registradores gerais são geralmente denominados por "o registrador X", "o registrador Y", e assim sucessivamente.

Um dos registradores de 16 bits — um dos mais importantes — será o registrador "contador de programas". É uma célula de memória interna que sempre contém o endereço (em código binário) da instrução da memória a ser executada a seguir. Quando chega o momento de obter a instrução seguinte para a CPU, o conteúdo do contador de programas será colocado no "address bus" e o byte nessa posição será transmitido (através do "data bus") à CPU.

O mais importante dos registradores de 8 bits é o "acumulador". É o registrador que geralmente armazena (temporariamente) o resultado das operações efetuadas pela ALU, os bytes trazidos da memória ou dos circuitos I/O, ou ainda o lugar em que os bytes são temporariamente armazenados, imediatamente antes de serem enviados para a memória ou para os circuitos I/O.

Esta apresentação da CPU foi muito genérica; veremos pontos específicos, em detalhe, mais tarde. O objetivo aqui foi demonstrar que a leitura de instruções especiais que estão na memória induz a CPU a efetuar operações específicas e a ajustar os pinos de endereço para acessar determinadas posições da memória. Os dados são recolhidos dessas locações ou a elas enviados através de um "data bus". As instruções igualmente fazem com que os sinais de controle do bus informem à memória ou ao circuito I/O se os dados devem ser "lidos" ou "escritos".

Sob controle
A ilustração mostra uma CPU, com "registradores" de memória, uma Unidade Aritmética e Lógica que inclui centenas de portas lógicas (que efetuam operações como a soma e a complementação de dígitos binários) e, ainda, um bloco de controle. Esse bloco admite a instrução codificada (em binário), interpreta-a e induz todas as outras partes da CPU a atuar de modo adequado. Por exemplo, se uma instrução indica que o conteúdo do acumulador deve ser armazenado em determinada posição da memória, o bloco de controle coloca o endereço nos pinos de endereço, envia sinais de controle, instruindo a memória a armazenar dados, e coloca o conteúdo do acumulador no "data bus", para transmissão à memória.



Seguindo ordens
Um computador simples pode consistir apenas em uma CPU, alguma capacidade de memória e um circuito de I/O. A memória armazenará as instruções que permitirão à CPU executar atividades específicas. Também armazenará dados para a CPU processar, de acordo com as instruções. A CPU necessitará do circuito I/O para comunicação com o exterior. Se o computador estiver controlando uma máquina de lavar roupa, o circuito I/O receberá os sinais provenientes dos botões do painel e enviará sinais para ligar e desligar os motores e aquecedores.
Cada modelo específico de CPU tem seu próprio conjunto de códigos de instrução.

O endereço certo

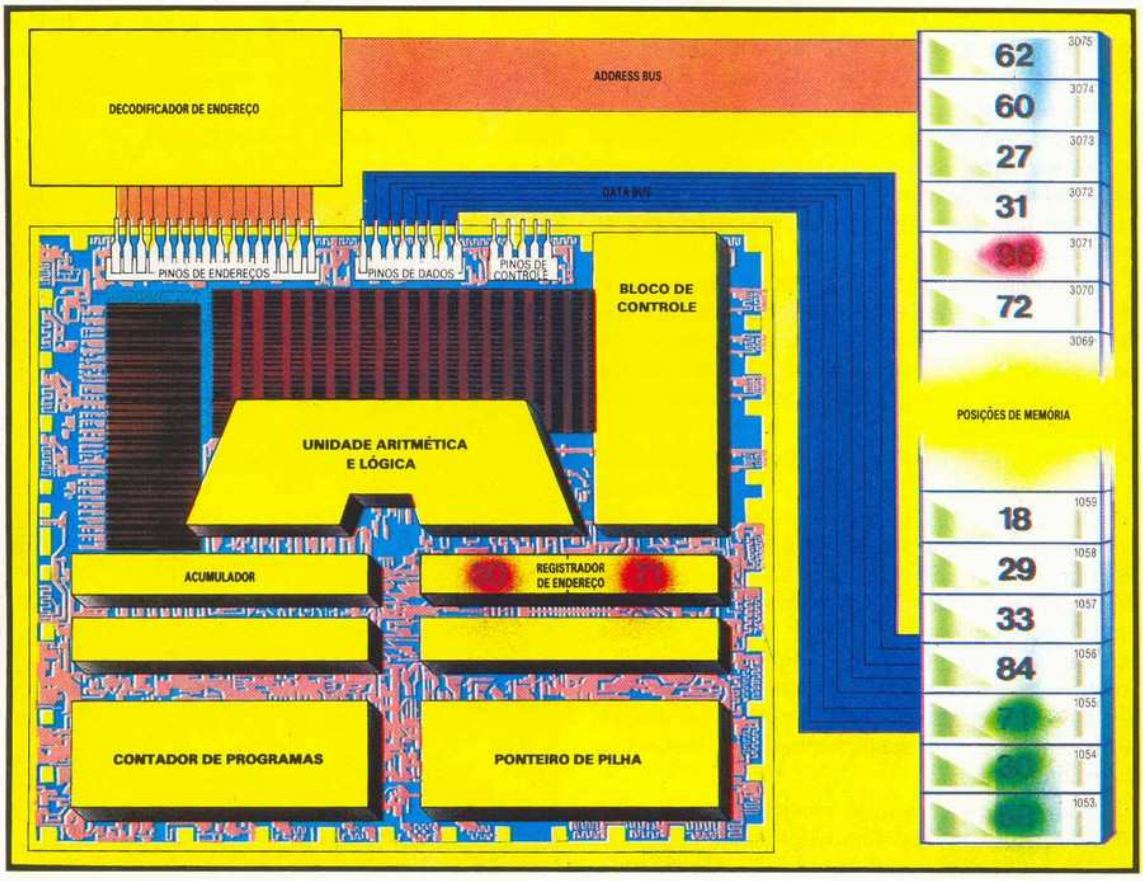
A função da CPU é localizar instruções e dados armazenados na memória. Você sabe o que acontece no seu interior quando as instruções do programa são executadas?

A CPU recebe suas instruções e dados das posições na memória do computador, atribuindo aos pinos de endereço o código binário adequado, permitindo à memória localizar e ler o conteúdo da posição no interior da CPU, através do data bus. Entretanto, essa operação é um pouco mais complexa. O problema está em que os bytes (códigos binários de 8 bits) em qualquer dos milhares de células da memória do computador podem ser instruções, informando à CPU que operações devem ser realizadas, ou dados que a CPU deve, de alguma forma, manipular. O que permite à CPU identificar quais dos bytes representam instruções e quais representam dados?

Reconhecimento de códigos

Primeiramente, verifiquemos o que é uma instrução. É um código em sistema binário que induz à realização de uma seqüência específica de operações no interior da CPU. Desse modo, o código 00111010, quando identificado pela CPU como uma instrução, em vez de simplesmente uma unidade de dados, fará com que a CPU enderece os 2 bytes seguintes da memória, interprete os dados neles contidos, coloque esses dados em um "registrador de endereços", ajuste os pinos de endereço para o mesmo número, dirija-se à posição de memória agora endereçada, obtenha os conteúdos dessa posição no data bus e carregue o acumulador da CPU com esses conteúdos.

Apresentado assim, pode parecer confuso, mas o que acabamos de descrever é um dos métodos de endereçamento da memória usado na CPU do conhecido Z80. Todo o processo de obtenção de 1 byte de dado no interior da CPU é apresentado no quadro abaixo. Suponha que a CPU saiba previamente que o byte a ser acessado em seguida, na memória, será uma instrução (e não dados) e que esse byte se situa na posição de memória 1053 (todos os números utilizados no quadro estão em notação decimal). Esse endereço, 1053, será colocado no address bus. Em código binário, esse endereço seria escrito como 0000010000011101. Os dezesseis pinos de endereço estão "ligados" ou "desligados" de modo a corresponder a esse número. Quando recebe esse endereço através do address bus, o "decodificador de



Seqüência de eventos
São muitos os estágios envolvidos, mesmo na mais simples operação da CPU. As instruções, também chamadas "códigos de operação", são lidas no interior do computador. Essas instruções são decodificadas pelo bloco de controle, e levam à realização de operações específicas. Neste exemplo, a instrução 58 é lida da posição de memória 1053. Essa instrução específica induz à realização da seguinte seqüência de procedimentos:
o byte na posição seguinte da memória (1054) será lido e armazenado em uma das metades do registrador de endereço de 16 bits da CPU.
O byte da posição seguinte (1055) será lido e armazenado na outra metade. Esses 2 bytes agora representam o endereço (algum outro lugar da memória) em que alguns dados estão armazenados.
O conteúdo do registrador de endereço é agora colocado no address bus de modo que a próxima posição de memória acessada seja 3071.
O conteúdo desse endereço é colocado no data bus e lido pela CPU. Este byte (96, em nosso exemplo) é então colocado no acumulador da CPU, onde permanecerá até ser utilizado por alguma outra instrução. O address bus retornará então a seu endereço anterior +1, de modo que passará a endereçar agora a posição 1056. A CPU sabe que o que estiver contido nessa posição será uma instrução e repetirá uma seqüência análoga de operações. Neste exemplo, 84, a instrução que vem a seguir, é interpretada pelo bloco de controle como sendo para inverter os bits no acumulador. Uma vez que a instrução 84 consiste em "1 byte", a CPU sabe que o byte na próxima posição de memória, 1057, também será uma instrução.

endereços'' o ''decodifica'' e liga uma e apenas uma das linhas de saída — e esta será a linha correspondente à posição de memória 1053.

No próximo estágio, o conteúdo desse endereço, que corresponde a 58, ou 00111010, em binário, é colocado no data bus e carregado no interior da CPU. Neste ponto, como a CPU está aguardando uma instrução, o byte será interpretado pelo bloco de controle e induzirá à execução de uma seqüência muito precisa de operações.

Essa instrução em particular determina que os próximos bytes da memória conterão 16 bits a serem utilizados como posição de memória, e especifica também que o seu conteúdo deverá ser carregado no acumulador da CPU. Após identificar essa instrução, a CPU prevê que os próximos 2 bytes da memória serão especificações do endereço e que o conteúdo desse endereço deve ser carregado no acumulador. Conseqüentemente, a CPU pode prever que não receberá outra instrução da memória, até que essas operações tenham sido efetuadas; saberá também que a próxima instrução se situará na posição 1056.

A instrução que estamos utilizando como exemplo faz com que o address bus aumente em uma unidade, de modo que a próxima posição de memória endereçada será 1054. O conteúdo dessa posição é então colocado no data bus e carregado na CPU. Entretanto, desta vez, é colocado em uma metade do registrador de endereços. Tendo realizado isto, a CPU incrementa novamente o address bus para endereçar a posição 1055. O conteúdo dessa posição segue para o data bus e é carregado de modo análogo na CPU, sendo armazenado na outra metade do registrador de endereços.

Transferência de números

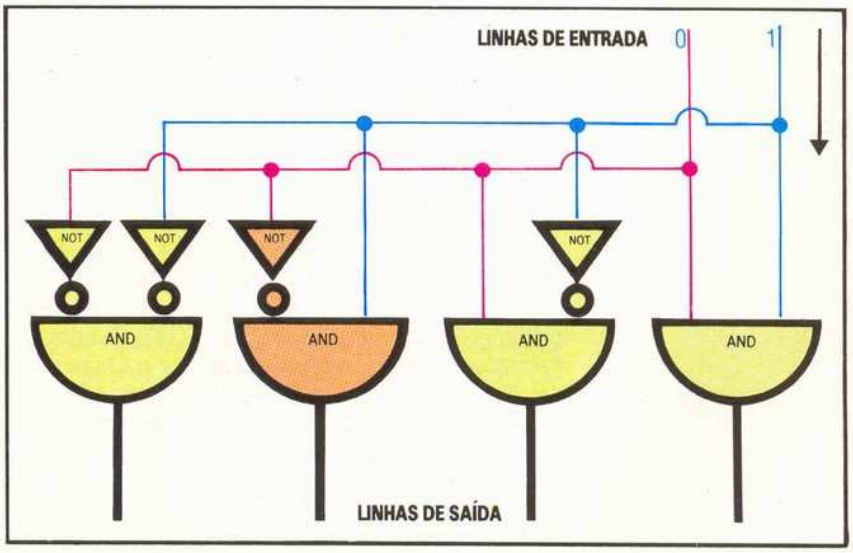
O estágio seguinte — e lembre-se de que todos estes procedimentos ocorrem automaticamente, como resultado da instrução original — é a transferência dos números do registrador de endereços para o address bus, neste caso 3071. Portanto, é a posição de memória 3071 que agora está sendo endereçada. Esse endereço (0000101111111111, pelo sistema binário) é decodificado pelo decodificador de endereços, que seleciona a célula de memória 3071. O conteúdo dessa posição, 96 (01100000, pelo sistema binário), é colocado no data bus e carregado na CPU. Desta vez, porém, os dados serão colocados no acumulador da CPU. Após isto, o address bus estará ajustado para 1056 e a CPU preparada para aí encontrar outra instrução.

Agora que a CPU tem uma unidade de dados em seu acumulador, que espécie de instrução pode-se encontrar a seguir? Poderia ser praticamente qualquer coisa — as CPUs são capazes de identificar de poucas dezenas a algumas centenas de instruções, dependendo da sua capacidade —, porém, vamos supor que queiramos inverter os dados no acumulador. Inverter significa converter cada dígito 0 em um dígito 1 e vice-versa. A instrução que permite realizar esse procedimento será encontrada no endereço 1056. Em nossa imaginária CPU, o código para essa instrução é 84. Quando este número for recebido pela CPU, os dados do acumulador serão invertidos. O número no acumulador era 96 (01100000, pelo sistema binário). Esta instrução ocasionará sua inversão para o número 10011111, em binário. A instrução para inversão de números no acumulador é de 1 byte; desse modo, a CPU prevê novamente que o conteúdo da próxima posição de memória, 1057, deverá ser uma instrução e não dados.

Esse método de endereçar uma posição de memória para recuperar uma unidade de dados é apenas um entre os diversos métodos que o programador pode utilizar. Os bytes de instruções utilizados no exemplo citado (58 para carregar o acumulador e 84 para inverter seu conteúdo) são as instruções de nossa CPU hipotética, mas o mesmo princípio se aplica a todos os outros chips microprocessadores. A única diferença é que códigos diferentes são utilizados para as várias instruções, e cada um dos tipos de CPU possui ''conjuntos de instruções'' próprios.

As posições I/O devem igualmente ter um endereço específico, mas os procedimentos pelos quais a CPU os endereça são os mesmos. Em geral, em microprocessadores de 8 bits, apenas oito das linhas de endereço estão disponíveis para endereçamento de posições I/O. Desse modo, o número máximo de endereços de I/O é 256 — mais do que suficiente para a maioria das aplicações de microcomputadores, mesmo se utilizados comercialmente.

Decodificação de endereços As dezesseis linhas que constituem o address bus podem identificar qualquer uma das 65.536 posições isoladas da memória. A combinação de dígitos 1 e 0 no address bus é decodificada por decodificadores de endereço. Parte da decodificação é realizada por decodificadores de endereço constituídos de simples portas lógicas em chips montados no circuito; boa parte do procedimento de decodificação é realizada por circuitos equivalentes no interior dos próprios chips de memória. O quadro mostra o processo pelo qual duas linhas de endereço podem ser decodificadas de modo que selecionem um e apenas um entre quatro chips.



A decodificação de endereços é sempre necessária e faz com que o dispositivo escolhido pela CPU (seja ele uma posição de memória, seja uma posição de I/O) mantenha-se ativo enquanto todas as outras posições de memória ou I/O permanecem inativas. Esse processo é chamado ''enabling''. Quando há pequena quantidade de linhas de endereço, é possível utilizar chips com portas lógicas simples a fim de realizar a decodificação. O princípio do decodificador de linhas dois-para-quatro é apresentado no quadro.

É comum utilizar esse tipo de decodificação simples para seleção de dispositivos de I/O. Entretanto, à medida que aumenta o total de endereços, também aumenta a complexidade do circuito de decodificação. Quando há 65.536 posições de memória a serem individual e exclusivamente selecionadas, é comum a maior parte da decodificação de endereço realizar-se no interior dos chips de memória.

Números hexadecimais

Os computadores utilizam o sistema binário, pois admitem apenas dois estados. Por que, então, os programadores às vezes recorrem ao sistema hexadecimal, de base 16?

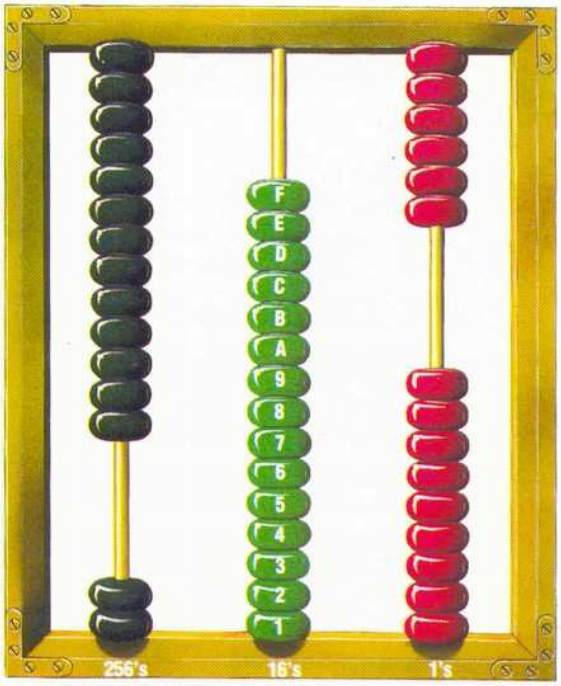
É fácil perceber por que se usa o sistema binário nas operações internas do computador — a representação dos números que utiliza unicamente os dígitos 0 e 1 presta-se bem aos sinais elétricos ligado/desligado com os quais o computador funciona. É igualmente fácil entender por que o sistema decimal é usado quase universalmente pelas sociedades humanas — o fato de termos dez dedos faz dele o nosso número base.

Que utilidade, porém, pode ter o sistema hexadecimal (de base 16)? A resposta está no fato de os números hexadecimais serem convertidos em números binários, e vice-versa, muito mais facilmente do que os decimais. Uma vez que os computadores "compreendem" os números binários, por que os programadores não fazem seus programas em dígitos binários? Há duas razões para isso. Primeiramente, os números binários são muito mais extensos do que seus correspondentes na base 10 ou 16. Por exemplo, o número 356 em decimal é 0000000101100100 no sistema binário. Em segundo lugar, é fácil cometer erros quando se usam apenas os dígitos 0 e 1.

O sistema hexadecimal utiliza 16 dígitos diferentes, inclusive o zero, de modo que o maior número que pode ser representado na coluna das unidades corresponde ao decimal 15. Os matemáticos poderiam ter adotado seis novos símbolos para os números entre 10 e 15, mas convencionou-se utilizar as seis primeiras letras do alfabeto. Ao realizar uma contagem pelo acréscimo de dígitos 1, recorremos a dígitos numéricos padrão juntamente com os dígitos alfabéticos até a letra F — o hexadecimal equivalente a 15. Uma vez esgotados os símbolos, temos de passar para uma outra coluna, a coluna dos "dezesseis". Aqui, igualmente, utilizamos os mesmos símbolos disponíveis até F. Após o preenchimento também desta segunda coluna, o acréscimo de mais um dígito significa iniciar uma terceira coluna, a dos "duzentos e cinqüenta e seis" (256). O quadro apresenta alguns números decimais com seus equivalentes binários e hexadecimais.

O número 65.535 é representado por FFFF no sistema hexadecimal, e isto evidencia uma das vantagens do sistema hexadecimal: um número que no sistema decimal utiliza cinco dígitos, e no sistema binário precisa de dezesseis, pelo sistema hexadecimal necessita de apenas quatro dígitos. Além dessa economia de espaço, há outra vantagem mais importante. Quatro dígitos binários podem ser representados por apenas um dígito hexadecimal. Isto torna a conversão do sistema binário em hexadecimal, e vice-versa, uma operação relativamente simples.

A conversão de números binários em seus equivalentes hexadecimais consiste em dividir o número binário em grupos de quatro dígitos, iniciando pela direita, e em seguida realizar a conversão, um grupo de cada vez. Eis aqui alguns exemplos:



O ábaco hexadecimal
Um ábaco feito para contagem pelo sistema hexadecimal tem 15 contas em cada coluna. Na primeira coluna à direita, as contas são puxadas, uma por vez, até alcançar F, a última. O sistema hexadecimal é um sistema de numeração de base 16 que utiliza letras do alfabeto para números maiores que 9. Como mostra a coluna do meio, a letra A corresponde ao decimal 10, B ao 11, C ao 12, D ao 13, E ao 14 e F ao 15. Após o "número" F, os dígitos se esgotam e temos de passar para a outra coluna, a dos "dezesseis".

A coluna das unidades, no quadro, mostra uma contagem de 9 contas, a coluna dos "dezesseis", uma contagem de F e a coluna dos "duzentos e cinqüenta e seis", uma contagem de 2. Este número é lido como "2F9 hex" e equivale ao número decimal 761 (256 × 2 + 16 × 15 + 1 × 9).

		1110	1001	binário
		=E	9	hex
		=233		decimal
	1111	1000	1100	binário
	=F	8	C	hex
	=3.980			decimal
0111	1110	0010	1101	binário
=7	E	2	D	hex
=32.301				decimal

Quando seria realmente interessante utilizar a notação hexadecimal? Embora linguagens de alto nível como o BASIC não exijam a utilização do sistema hexadecimal (ou mesmo do binário), as linguagens de baixo nível, como código de máquina e assembly, o fazem.

Costuma-se colocar o símbolo H após o número hexadecimal para distingui-lo dos números decimais. Desse modo, o número 256 (decimal) seria escrito da seguinte forma: 100H e sua leitura seria "um, zero, zero, hex".

Não teremos necessidade prática dos números hexadecimais em nosso curso. Entretanto, poderão ser mencionados em algum apêndice do manual de instruções de seu computador e, com estes esclarecimentos, você pode ao menos calcular os equivalentes no sistema binário ou decimal.

Binário	Decimal	Hex
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
00000101	5	5
00000110	6	6
00000111	7	7
00001000	8	8
00001001	9	9
00001010	10	A
00001011	11	B
00001100	12	C
00001101	13	D
00001110	14	E
00001111	15	F
00010000	16	10
00010001	17	11
00010010	18	12
00010011	19	13
00010100	20	14
00010101	21	15

Peek e poke

Esses dois comandos são empregados toda vez que se quer programar algo não previsto na linguagem BASIC. Eles vão ser utilizados de modo diferente pelos diversos equipamentos.

Usando o POKE

A instrução POKE deve ser usada com cuidado, pois modifica os conteúdos das posições de memória, o que pode afetar o funcionamento do computador. Disto não advirá nenhum dano ao equipamento, mas o programa poderá perder-se. Eis algumas instruções POKE que são "seguras", para você experimentar.

No Atari 400 ou 800, o armazenamento do número 1 pela instrução POKE na posição 751 desligará o cursor na tela; experimente a posição 751,1.

Nos computadores da linha Apple, experimente a posição 1024,1; o número 1024 é o endereço da primeira posição na tela. Nos computadores compatíveis com o Sinclair (TK83, TK85, CP 200 etc.), as instruções PEEK e POKE são utilizadas como se segue:

```
POKE 17300,57
PRINT PEEK 17300
```

Com estas instruções, o resultado será o número 57 que foi atribuído ao endereço 17300 pela instrução POKE.

PEEK e POKE são duas "instruções" da linguagem BASIC usadas em programação de nível mais adiantado, quando bits e bytes, isoladamente, devem ser manipulados na memória. A instrução PEEK é utilizada para examinar os conteúdos de um endereço específico na memória; a instrução POKE é usada para armazenar números (que variam de 0 a 255) em determinadas posições da memória.

As instruções PEEK e POKE permitem ao programador que utiliza a linguagem BASIC o acesso às operações internas do computador, o que de outro modo seria impossível. Geralmente, a linguagem BASIC incorporada em seu computador se encarrega das posições efetivas em que se encontram armazenadas as variáveis e os dados que definem os caracteres a serem apresentados na tela. Embora não precisemos nos preocupar com os locais da memória em que estão esses elementos, ocasionalmente teremos de localizá-los. É a instrução PEEK que nos permite realizar isso.

Escreve-se desse modo um pequeno programa para verificar qualquer posição na memória:

```
10 REM EXAMINAR POSICOES DE MEMORIA
20 PRINT "FORNECER A POSICAO DA MEMORIA EM
   NUMEROS DECIMAIS"
30 INPUT M
40 P = PEEK(M)
50 PRINT "OS CONTEUDOS DA POSICAO "; M;"
   SAO ";P
60 GOTO 20
70 END
```

Este programa imprimirá o conteúdo do endereço especificado, através de um número decimal (evidentemente, o computador o armazena em código binário). Se você quiser verificar os equivalentes dos conteúdos em termos de caracteres "imprimíveis", a linguagem BASIC inclui uma função que permite converter números decimais em seus caracteres correspondentes. Essa função é a CHR$ e a alteração da linha 50 imprimirá, em seu lugar, caracteres equivalentes às posições da memória:

```
50 PRINT "OS CONTEUDOS DA POSICAO ";M;"
   SAO "; CHR$(P)
```

Para examinar a memória por completo, acrescenta-se um loop FOR-NEXT, suprimindo a linha 30 e alterando a linha 20 para FOR X = 0 TO 65535; substitui-se, ainda, a linha 60 pela instrução NEXT X.

Para dar tempo suficiente ao exame de cada caractere enquanto é impresso, você acrescenta um loop de retardo após a instrução PRINT e antes da instrução NEXT X. Note que o limite superior do loop FOR-NEXT supõe que você tenha um equipamento com 64 Kbytes de memória. Este número pode ser alterado no caso de memórias de menor capacidade: para memórias de 16 Kbytes é necessário o número 16383, em notação decimal; para memórias de 32 Kbytes, 32767; e para as de 48 Kbytes, 49151. A listagem completa deste programa é a seguinte:

```
10 REM VERIFICACAO E IMPRESSAO DE TODAS AS
   POSICOES DE MEMORIA
20 FOR X = 0 TO 65535
30 LET Y = PEEK(X)
40 PRINT "POSICAO ";X;"=";Y;"=";
50 PRINT CHR$(Y)
60 FOR D = 1 TO 200
70 NEXT D
80 NEXT X
90 END
```

Embora a função CHR$ converta números decimais em seus caracteres equivalentes, os caracteres que podem ser impressos representam-se pelos números 32 a 127. A maioria dos computadores utiliza números entre 128 e 255 (o maior número que pode ser representado através de um único byte) para caracteres gráficos especiais. Muitos dos números entre 0 e 31 têm funções especiais de controle de tela. Quando o programa, ao ser processado, encontra esses números na memória, a função CHR$ converte esses números em interessantes efeitos de tela.

A instrução POKE é, basicamente, o procedimento oposto à instrução PEEK: permite que você "escreva" 1 byte de dados (qualquer número entre 0 e 255) em qualquer posição de memória. Esta instrução deve ser utilizada com cuidado: se, pela instrução POKE, você fizer o armazenamento de um número na posição errada da memória, poderá danificar uma parte de um programa essencial. O único modo de corrigir isso será desligar e ligar o computador novamente, a menos que tenha um botão RESET. De qualquer modo, você vai perder o que está na memória naquele momento. Assim, antes de utilizar a instrução POKE, verifique em seu manual qual é a área, no mapa da memória, designada como "área para o usuário".

A maioria dos microcomputadores possui memória de vídeo (a memória utilizada para armazenamento dos caracteres que serão apresentados na tela) acessível ao usuário. Normalmente, o computador obtém o formato desses caracteres da memória ROM especial, chamada gerador de caracteres, que armazena os padrões de pontos para cada caractere. Mas é igualmente possível utilizar a memória RAM. Quando os códigos de padrão para os caracteres estão armazenados na memória RAM, novos padrões podem ser armazenados pelo comando POKE, na posição correspondente da memória RAM e utilizados para definição de novos caracteres.

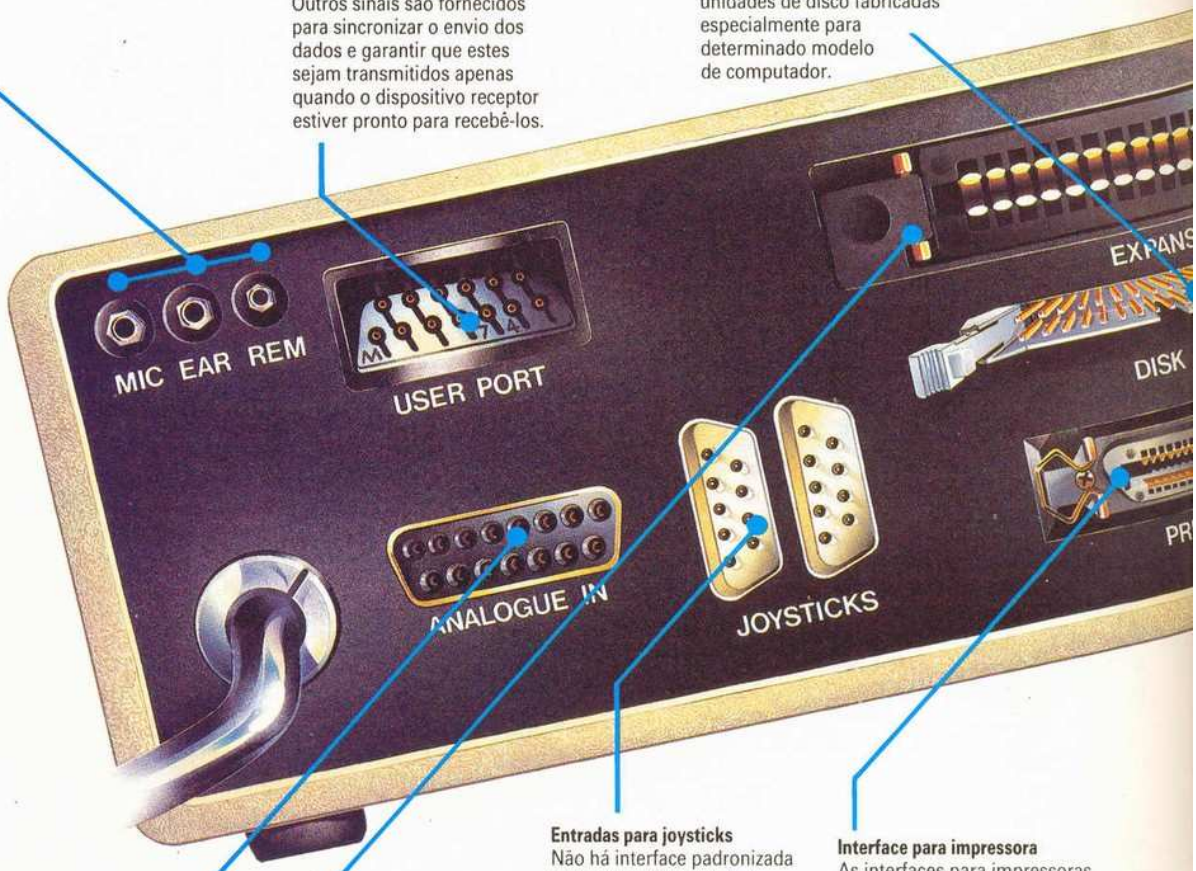
Entradas e saídas

Você não consegue encaixar uma peça quadrada em um orifício redondo. Do mesmo modo, dois dispositivos de computador não podem ser ligados, a menos que tenham interfaces compatíveis.

Interface para cassete
A entrada para cassete, ou interface, existente na maioria dos microcomputadores é efetivamente um tipo de interface serial. Uma vez que os dados têm de ser gravados em fita cassete comum, não são possíveis altas velocidades de transferência de dados. O sistema de circuitos toma da memória os bytes de dados a serem gravados e os converte em um conjunto de bits. Os bits 0 e 1 transformam-se em dois tons diferentes de som. Carregadas as fitas na memória, os tons são decodificados pelos circuitos da interface e os bits 1 e 0 resultantes, combinados em bytes de 8 bits para armazenamento na memória. As interfaces cassete da maioria dos microcomputadores são universais, ou seja, pode-se utilizar qualquer gravador comum. O conector empregado não é padrão, mas as tomadas DIN e os mini-jacks são os mais comuns.

Porta paralela
É uma interface paralela para finalidades diversas, que liga o micro a dispositivos periféricos. Os 8 bits de cada byte transmitido são enviados juntamente (em transmissão paralela), através de oito fios. Outros sinais são fornecidos para sincronizar o envio dos dados e garantir que estes sejam transmitidos apenas quando o dispositivo receptor estiver pronto para recebê-los.

Interface para unidade de disco
As unidades de disco são geralmente ligadas ao computador através de uma interface paralela. Não têm padronização e, com freqüência, só é possível ligar unidades de disco fabricadas especialmente para determinado modelo de computador.



Entrada analógica
Encontrada apenas nos computadores mais caros e de finalidade educacional, as entradas analógicas ligam o computador a dispositivos de laboratório, como medidores elétricos de temperatura ou fotossensores. A interface apenas apresentará uma ou mais linhas que poderão receber e ler a voltagem dentro de um nível específico. Cabe ao usuário assegurar-se de que o computador não seja ligado a uma voltagem maior que seu limite, porque danificaria seriamente o equipamento.

Porta de expansão da memória
Conecta a maioria das linhas, senão todas, que vêm diretamente do microprocessador, isto é, os busses de endereço, de dados e de controle. É aí que a memória suplementar será conectada e, em alguns computadores, até mesmo os periféricos produzidos pelo fabricante do equipamento. O conector é só um terminal PCB, embora, em alguns casos, seja uma tomada que admite um conector terminal, como os cartuchos de jogos (na verdade, uma forma de complemento da ROM).

Entradas para joysticks
Não há interface padronizada para os joysticks. Muitos fabricantes adotam o padrão da Atari, para não produzir seus próprios joysticks. A maioria dessas interfaces tem apenas cinco linhas ativas: as provenientes de cada interruptor, nas quatro extremidades do movimento do joystick, e uma para o botão detonador. Os joysticks analógicos, todavia, exigem uma interface que admita uma série completa de voltagens, indicadoras da posição exata do comando. A maioria dos computadores tem mais de uma porta para joystick; às vezes, a mesma tomada é utilizada por mais de um dispositivo.

Interface para impressora
As interfaces para impressoras são bastante padronizadas, de acordo com o sistema desenvolvido pela Centronics Corporation; assim, há pouca dificuldade em obter uma impressora com interface Centronics que possa ser conjugada à interface impressora da maioria dos computadores. Os níveis de sinais, bem como as funções de sinais, também são padronizados: 0 e 5 volts para os binários 0 e 1, respectivamente. Os conectores utilizados e a atribuição dos sinais aos diferentes pinos não são padronizados; pode ser que você tenha de montar um fio conector especial para ligar a impressora ao computador.

Interface para vídeo (RF)
Todos os microcomputadores são projetados visando à sua conexão com uma unidade de vídeo, e, para a maioria deles, isto corresponde a um aparelho de televisão. Se for usada a tomada comum para antena, o sinal de vídeo deverá estar em RF (Radio Frequency) modulada, de modo a assemelhar-se a sinais de emissão para televisão. Este sinal é em seguida demodulado pelo sintonizador do televisor.

Interface RGB
Resultados de imagem colorida ainda melhores serão possíveis se os elementos necessários ao monitor do vídeo forem mantidos separados. A interface RGB possibilita sinais de vídeo vermelhos, verdes e azuis separados, além dos sinais de sincronização horizontal e vertical.

Interface para vídeo (composto)
Alguns televisores e muitos monitores de vídeo incorporam uma entrada para sinais compostos ou "vídeo" que evita o estágio de demodulação e possibilita a produção de imagens de melhor qualidade. Todos os elementos do sinal padrão de vídeo estão presentes na saída do computador, mas este sinal "composto" não necessita de modulação complementar para processamento pelo sintonizador de televisão.

Interface serial
Ao contrário de muitas outras, a interface serial RS232 é teoricamente definida de modo preciso pela padronização da Electrical Industries Association. Entretanto, poucos fabricantes se atêm ao padrão, o que dificulta o funcionamento de dispositivos seriais em certos computadores. Dos 25 pinos presentes na interface padronizada, geralmente apenas três são utilizados: o pino 2, para transmissão de dados seriais entre o computador e o dispositivo periférico; o pino 3, para recepção de dados provenientes do dispositivo periférico; e o pino 7, para o sinal terra. Tanto os dispositivos de emissão como os de recepção necessitam ser ajustados de modo que a velocidade da transmissão e o formato dos dados transmitidos sejam os mesmos.

Interface IEEE
É uma interface paralela universal, baseada no Hewlett-Packard Interface Bus, agora adotado como padrão pelo Institute of Electrical and Electronics Engineers. O padrão é bem definido, tanto física como eletronicamente. Ao contrário de outras interfaces de dados (como a interface serial RS232 e a paralela Centronics), que são conectadas a um único dispositivo periférico por vez, o bus IEEE pode ser ligado a quinze instrumentos (inclusive o computador) ao mesmo tempo. Dispositivos que incorporam interfaces IEEE abrangem impressoras, unidades de discos flexíveis, plotters, geradores de sinais, voltímetros e outros equipamentos para testes. Por se prestar tão bem ao uso em equipamentos de teste e medição, o bus IEEE é preferido em laboratórios e estabelecimentos industriais. Atualmente, apenas alguns microcomputadores apresentam interface IEEE.

A palavra interface é utilizada livremente, com o significado de porta de entrada ou tomada em que dispositivos externos são ligados ao computador. Mais precisamente, o termo se refere ao sistema de circuitos e ao software complementar, que tornam possível a conexão entre dois dispositivos quaisquer, vinculados ao computador.

Internamente, o computador comunica-se pelo envio de dados através de "busses" — conjuntos de condutores paralelos, que levam, cada um, um sinal binário único. Na maioria dos computadores, há três busses internos: um bus de dados de 8 bits, um bus de endereço de 16 bits e um bus de controle, geralmente com sinais de 5 a 12 bits, que indicam o estado efetivo da CPU. Alguns dos sinais de controle informam à memória e aos dispositivos periféricos se a CPU vai recuperar dados (ler) ou depositar dados (escrever). Outros transmitem à CPU dados provenientes de fora do equipamento, informando-a, por exemplo, que um dispositivo periférico tem dados para entrada.

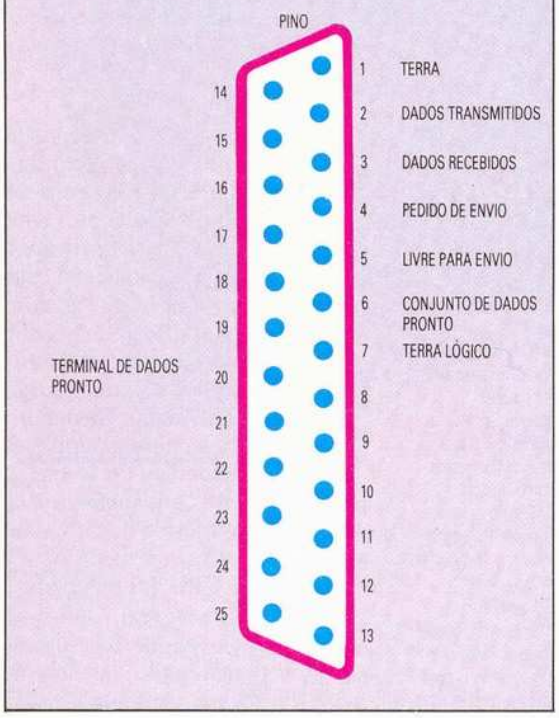
Ainda internamente, o computador manipula informação que consiste, em geral, em 8 ou 16 bits de cada vez. Assim, se for recuperar os dados na posição de memória 65535 (ou FFFF, pelo sistema hexadecimal), a CPU atribuirá o valor 1 a todos os dezesseis fios do bus de endereço, de modo a identificar essa posição. Se os conteúdos dessa posição de memória forem, por exemplo, 182 (B6, pelo sistema hexadecimal), esses dados serão colocados no bus de dados como oito dígitos binários: 10110110.

Quando se transferem os dados desse modo, 8 ou 16 bits por vez, a transferência chama-se paralela. Muitos periféricos são projetados para transmissão ou recepção paralelas de dados. As interfaces providas de dispositivos periféricos deste tipo denominam-se interfaces paralelas e a maioria dos computadores apresenta pelo menos uma tomada ou conector especiais para dispositivos paralelos.

Nem todos os dispositivos periféricos são projetados para recepção ou transmissão paralelas de dados. Alguns têm um fio único para comunicação com o computador, 1 bit por vez. Internamente, utilizam ainda os dados como bits de 8 ou 16 bytes, mas os bits de cada byte serão transmitidos ou recebidos, um de cada vez, a partir do "último bit significativo" do byte, concluindo com o bit "mais significativo". Cada byte é dividido em uma série de bits, enviados um após o outro, e reconstituídos em 1 byte na outra extremidade, mediante circuitos especiais de conversão de paralelo para serial e de serial para paralelo.

Tanto as interfaces seriais como as paralelas podem ser induzidas a transmitir dados, quer para fora quer para dentro do computador. Os computadores, de modo geral, incluem outras interfaces, que ou sempre enviam dados para fora (como o circuito de saída da televisão) ou sempre aceitam dados que chegam (por exemplo, portas para joysticks).

Funções padrão dos pinos para a interface RS232



Interface serial

Existe uma interface serial padrão chamada RS232, para a qual os detalhes dos níveis de sinal e do ajuste de pinos são definidos. Mesmo o tipo de conector é também especificado. Infelizmente, não se costuma adotar esse padrão de modo rigoroso, e a execução de trabalho por conexões seriais pode tornar-se difícil. A interface RS232 é bidirecional serial, com um pino (terminal) para transmissão e outro para recepção de dados do computador.

Os dados são enviados, 1 bit de cada vez, pelo terminal de "dados transmitidos" e, em sentido contrário, através do terminal de dados recebidos. O conjunto de bits pode exigir uma série de formatos standard, mas sem distinguir qual está sendo usado, desde que tanto o dispositivo emissor como o receptor utilizem o mesmo formato.

Uma vez que cada byte de informação é enviado como um conjunto em série de bits, o software que controla a interface deve ter um modo de indicar quando o primeiro bit de dados se inicia e quando o último se encerra. A convenção mais comumente usada tem um único "bit de início" (0, em lógica booleana), seguido de 8 bits de dados, e com um único "bit de encerramento" (dígito lógico 1).

A velocidade com que os dados são transmitidos deve ser especificada antecipadamente; caso contrário, o padrão de impulsos que representa os 8 bits de dados em dígitos 0 e 1 poderá ser interpretado de maneira errônea. Essa velocidade de transmissão de dados é conhecida como velocidade baud, nome que faz referência ao inventor francês Baudot. A velocidade baud varia de 75 a 9.600 bauds. Esses números correspondem aos 75 e 9.600 bits transmitidos por segundo, e como há, geralmente, 10 bits para cada caractere, a transmissão do caractere corresponde a um décimo da velocidade baud.

Interfaces paralelas

A interface paralela transmite ou recebe dados, 1 byte de cada vez, mas, além das oito "linhas de dados", será necessário fornecer outros sinais, de modo que o computador e o equipamento periférico tenham a indicação de quando é possível, ou não, a transmissão de dados. O tipo mais comum de interface paralela é a Centronics (da fabricante de impressoras Centronics Corporation, dos Estados Unidos). O conector utilizado e a atribuição de sinais a pinos específicos diferem bastante de um fabricante para outro. A maioria das interfaces Centronics possibilita pelo menos os seguintes sinais:

DADOS 0 a DADOS 7	Oito fios para transportar os 8 bits do byte que está sendo transmitido.
ADK	Sinal de entrada para o computador; indica que o dispositivo receptor está preparado para receber dados.
GND	O fio terra que fornece uma referência comum de 0 volt, tanto para o computador como para o dispositivo periférico.
BUSY	Sinal do dispositivo periférico para o computador; indica que o periférico não pode aceitar dados.
STROBE	Sinal de saída do computador; indica ao periférico que os dados estão prontos para leitura.

Muitos outros dispositivos, além das impressoras, adotam a interface paralela Centronics, e a conexão com o computador pode exigir apenas a aquisição de um fio especial ou a instalação elétrica feita pelo usuário. De modo geral, não haverá alterações no software necessário para "controle" dos periféricos.

Sala de espera

O computador transfere informação com tal rapidez que um dispositivo como a impressora não pode acompanhá-lo. A solução é dada pelo buffer, ou memória intermediária.

Os pára-choques (buffers) dos vagões ferroviários são projetados para reduzir um eventual impacto, graças à absorção de energia pelas molas e pistões amortecedores. Os computadores também possuem os chamados buffers, que, de certo modo, funcionam como os amortecedores dos trens, ou seja, como isolantes, possibilitando que as diferentes partes do sistema do computador funcionem juntas.

Esse termo estranho é usado um tanto livremente no mundo dos computadores e abrange dois aspectos distintos. Para o programador, buffer significa um uso especializado da memória do computador; já para o projetista do circuito, quer dizer um tipo de amplificador de um sinal elétrico. O segundo tipo, a que chamaremos buffers de sinal, é operado no próprio painel.

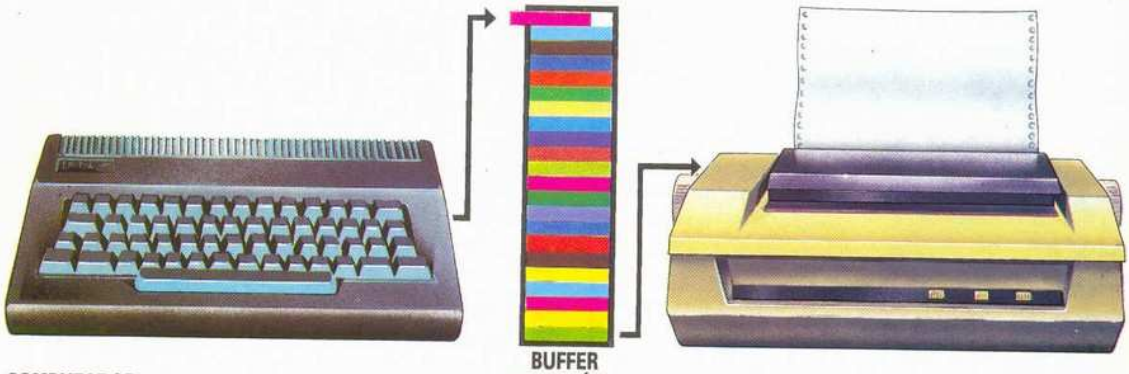
Buffers de memória

Imagine um programa processador de palavras que, entre outras coisas, pode transpor um bloco de texto de uma parte do "documento" na memória do computador para outra parte. O texto consiste em caracteres imprimíveis e espaços, e caracteres não imprimíveis, como RETURN, por exemplo. Todos estes são representados na memória do computador em código binário ASCII. Um byte de memória é necessário para cada caractere. A fim de que os blocos sejam transferidos de sua posição na memória para novas localizações, uma outra parte da memória do computador tem de ser reservada ao armazenamento temporário do texto. Esta área de memória deixada propositadamente de lado para uma tarefa específica recebe o nome de buffer.

Consideremos, por exemplo, a tarefa de imprimir um "documento" ou texto redigido num processador de palavras. O documento consiste, digamos, em 15.000 caracteres, e é claro que eles não podem ser colocados na impressora de uma só vez — a maioria das impressoras não funciona com velocidade superior a 80 caracteres por segundo. Para contornar esse problema, uma parte da memória do computador é reservada e mantida sob o controle do software processador de palavra, funcionando como um "amortecedor" de impressão. Primeiramente, o software alimenta este buffer com os caracteres a serem impressos; a seguir, envia-os numa velocidade adequada à impressora.

O buffer de impressão pode não ser muito grande, talvez apenas 128 ou 256 bytes de capacidade, mas o princípio é o mesmo, não importando seu tamanho. Primeiro um "bloco" de caracteres em ASCII é escrito nele, depois os caracteres são enviados novamente, 1 byte de cada vez. O primeiro byte a ser escrito no buffer será também o primeiro a ser lido (é óbvio que nosso objetivo é ter os caracteres impressos do mesmo modo em que foram digitados). Este tipo de buffer é chamado "Primeiro a Entrar, Primeiro a Sair", ou buffer FIFO (First In First Out). Quando todos os caracteres do buffer tiverem sido lidos, o software fará novo preenchimento com os blocos seguintes de caracteres destinados à impressora.

Os buffers FIFO deste tipo constituem um recurso extremamente comum na maioria dos softwares. São utilizados todas as vezes em que há incompatibilidade de velocidade, não só entre computador e impressora mas também entre computador e unidade de disco flexível, e entre computador e teclado. Apesar de a extraordinária velocidade de processamento dos computadores permitir que eles localizem instantaneamente as teclas pressionadas, há momentos em que o computador não pode identificá-las nem apresentar os caracteres correspondentes tão rapidamente. Isso acontece quando o computador está, em dado momento, ocupado em outra função (acessando um disco, por exemplo). Neste caso, é comum a incorporação de um "buffer de digitação" ao sistema de operação do computador. O buffer "lembra" quais teclas foram pressionadas e o computador apresenta os caracteres tão logo seja possível. Normalmente, o usuário não percebe isso,



Parada temporária
Um dos usos mais comuns do buffer é entre o computador e uma impressora, porque esta não pode liberar caracteres com a mesma velocidade que o computador os envia. Os caracteres são, portanto, armazenados na memória temporária até que o buffer esteja completo; um sinal de "ocupado" é enviado ao computador para que este pare de transmitir. O conteúdo do buffer é então enviado à impressora na mesma ordem em que foi recebido, mas num andamento bem mais lento. Quando essa parte termina, o processo recomeça, até que todo o texto seja impresso.

Controle de fluxo

Se derramarmos a água de um balde num tanque, como o que aparece acima, perceberemos que a torneira continua com a saída de água com vazão constante. Do mesmo modo, o computador "derrama" dados no buffer, que os libera numa velocidade bem mais lenta, adequada aos diversos periféricos.

mas em certos sistemas operacionais de disco, ou em certos tipos de software de aplicação (que envolvem muito processamento de informação), pode haver um pequeno atraso entre o momento em que a tecla é pressionada e o aparecimento do sinal na tela. Alguns sistemas operacionais permitem que o buffer de digitação seja ligado e desligado, e que sua capacidade seja alterada pelo usuário.

A maneira exata com que o software é organizado para manejar buffers varia, dependendo do que o buffer deve fazer; mas, geralmente, é necessário deixar de lado aguns bytes, que serão usados como contadores e sinalizadores (flags). É preciso saber quantos bytes foram lidos de um buffer completo antes que mais bytes sejam escritos; caso contrário, dados importantes podem ser destruídos antes de serem usados.

Hardware de buffers de memória

Quem possui uma impressora deve ter notado que ela parece ser lenta, principalmente quando imprime uma listagem ou texto longo. De modo geral, os sistemas operacionais do computador não conseguem desempenhar nenhuma outra atividade enquanto a impressora está em funcionamento; assim, se ela demora muito para terminar a impressão, o usuário tem de esperar a conclusão do processo. Atualmente, muitos fabricantes oferecem buffers de impressão como acessório, em geral na forma de uma caixa que liga o computador à impressora. Essa caixa acelera o desempenho da impressora mas só em relação ao computador. Apesar de, na verdade, a impressora não estar realmente imprimindo com maior rapidez, a caixa contém memória extra, específica (às vezes de até 16 Kbytes), com seu próprio software incorporado. Para o computador, parece que a impressora está trabalhando com maior velocidade. Quando o computador tem de imprimir um arquivo de dados, ele envia bytes para a impressora até receber um sinal de "ocupado" — isso significa que a impressora não pode aceitar mais bytes. O computador, então, espera que o sinal "ocupado" se torne "falso", indicando que a impressora está livre novamente para aceitar dados. As impressoras têm, geralmente, um pequeno buffer de memória incorporado, mas isso não passa de 2 Kbytes e não permite que o computador envie mais dados até que o buffer esteja vazio. O hardware acessório de buffers contém maior quantidade de memória; assim sendo, pode aceitar mais dados antes de enviar um sinal de "ocupado" ao computador. Se o buffer for suficientemente grande, será capaz de reter todos os dados a serem impressos em uma única vez, e o computador poderá prosseguir com outras tarefas, enquanto o buffer envia os dados em menor velocidade para a impressora.

A memória é mais freqüentemente utilizada como um "pombal" que armazena programas e dados, mas ela pode também ser organizada em pilhas ou buffers. Pilha significa "Último a Entrar, Primeiro a Sair" ou estrutura LIFO (Last In First Out), ao passo que os buffers são estruturas FIFO. Pode-se fazer uma analogia com uma pilha de pratos: o último prato empilhado será o primeiro a ser retirado. Como os buffers, as pilhas são áreas de memória temporária e diferem dos primeiros apenas na ordem em que os dados entram e são recuperados. As pilhas são usadas "internamente" em linguagens de alto nível (em interpretadores BASIC, por exemplo) quando a informação precisa ser armazenada temporariamente e recuperada mais tarde. Considere este fragmento de programa BASIC:

```
FOR X = 1 TO 10
PRINT "X = ";X
FOR Y = 1 TO 10
GOSUB SCAN
NEXT Y
PRINT "C$ = ";C$
NEXT X
```

Este é um exemplo de ninhos de loops FOR-NEXT. Quando o interpretador BASIC chega à segunda instrução FOR, ele deve lembrar-se da variável utilizada no primeiro FOR (X, neste caso), e assim "empurra" a informação com o primeiro FOR para cima da pilha. Quando o loop interior está completo, ele toma a informação do alto da pilha e sabe que o FOR em questão usa a variável X. Já que os loops FOR-NEXT podem ser muito bem guardados, ele talvez precise empurrar informações para vários FOR sobre a pilha. Ao tirar a informação da pilha, o interpretador BASIC precisa obviamente ter a informação na ordem inversa da que ele empurrou.

Os buffers, entretanto, organizam a memória de tal modo que a primeira informação que entrou é a primeira a sair. Os buffers são freqüentemente utilizados em rotinas de entrada/saída e também como "interfaces" entre as rotinas e os dispositivos que funcionam em diversas unidades ou em diferentes velocidades. Por exemplo, uma rotina de entrada em BASIC poderá trabalhar em unidades de linhas, terminadas por um RETURN <CR>, mas o interpretador trabalharia nas linhas em unidades de um caractere. Os buffers geralmente necessitam de um "indicador" para apontar onde o próximo caractere deve ser escrito. O indicador seria um byte ou vários bytes que contenham o endereço desse caractere. O endereço seria incrementado após o armazenamento de cada caractere.

Sinais poderosos

A lógica interna de seu computador funciona em nível TTL (Transistor-Transistor-Logic). Esta lógica significa o binário 1 com 5 volts, e o 0 com zero volt. Entretanto, apesar de dispositivos como a CPU serem capazes de produzir essas voltagens, não podem gerar corrente suficiente para levar todos os outros chips que devem ser ligados a cada pino. Os buffers de sinal são, assim, ligados às linhas de output da CPU, que aumenta a quantidade de corrente existente. Os buffers de sinal são, em si, pequenos chips, cada um tipicamente capaz de agir como um buffer para seis sinais.

O visual dos caracteres

O "gerador de caracteres" é a parte da memória do computador que define a forma dos caracteres na tela. Em alguns sistemas, é possível ao usuário desenhar seus próprios símbolos.

Já vimos no curso de programação BASIC (ver p. 214) que todos os caracteres alfanuméricos — e os símbolos gráficos, caso seu computador os tenha — são armazenados na memória RAM na forma de códigos de 8 bits (geralmente em ASCII); assim sendo, cada caractere ocupa 1 byte.

Quando a informação é impressa na tela, os códigos para cada caractere são depositados numa área reservada da memória chamada RAM de vídeo. Se, por exemplo, a letra A é impressa no alto, no canto esquerdo da tela, o primeiro byte da RAM de vídeo irá conter o código 65 (ASCII para A). Se um C é impresso embaixo do A, e o computador tem uma tela de quarenta colunas, o valor 67 será encontrado na 41.ª locação da RAM de vídeo, e assim por diante. Como o computador converte o valor 65 ao padrão de pontos que formam o caractere A na tela? Isto fica a cargo do dispositivo chamado gerador de caracteres.

Um gerador de caracteres nada mais é do que uma coleção de configurações ou padrões depositados como bits na memória. Os microcomputadores têm o gerador de caracteres armazenado em ROM, permitindo uma aparição imediata de caracteres assim que a máquina é ligada. O gerador de caracteres pode ser incorporado à ROM que contenha o interpretador BASIC e o sistema operacional, ou estar no próprio chip da ROM. Quando isto ocorre, é comum encontrar fornecedores independentes que oferecem a recolocação de ROMs que produzirão um grupo de caracteres de língua estrangeira, por exemplo, ou uma série de símbolos especializados para, digamos, engenharia ou matemática. Entretanto, um número crescente de computadores permite que o gerador de caracteres seja transformado em RAM — admitindo que o programador projete seus próprios caracteres e símbolos.

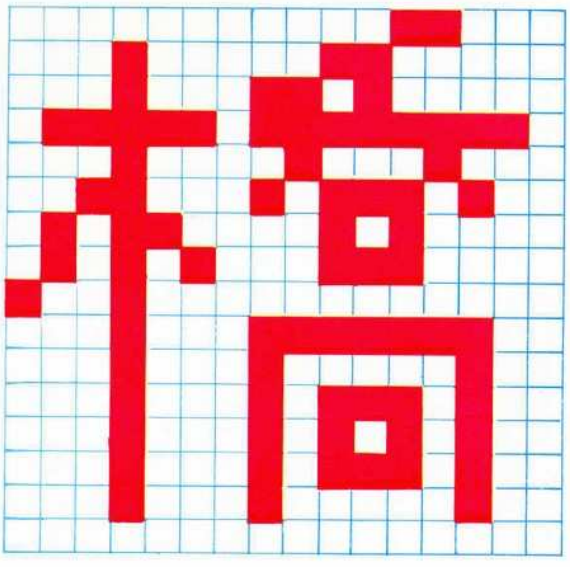
Todos os caracteres são construídos numa matriz de pontos, que num microcomputador é de 8 x 8, embora uma matriz maior resulte em melhor legibilidade e em maior variedade de caracteres a ser mostrada. Os caracteres são formados com o preenchimento de quadriculados na retícula. O quadriculado preenchido representa o dígito 1, e o quadriculado em branco o dígito 0, dando o total de 64 bits, ou 8 bytes para cada caractere.

O primeiro byte no gerador de caracteres representaria o padrão de bit para a linha do alto do primeiro caractere na tabela. Se o computador só apresenta caracteres em ASCII, com códigos de 0 a 127, então o gerador de caracteres vai requerer 128 x 8 bytes (1 Kbyte de memória).

A dificuldade para o computador é que, quando o varredor na tela de televisão está gerando a linha mais ao alto do vídeo, ele precisa produzir a linha mais ao alto de pontos para o caractere posicionado no alto da tela, à esquerda; em seguida, produz a linha mais ao alto do caractere posicionado a sua direita, e assim por diante, varrendo toda a tela. Dessa maneira, quando o varredor começa sua segunda passagem, precisa encontrar e mostrar a segunda fileira de pontos para cada um dos caracteres da linha anterior.

O circuito de vídeo atinge isso por meio de dois contadores independentes. Um contador mantém a pista da posição de memória de vídeo correspondente ao ponto por onde o varredor está passando. O outro conta as linhas do varredor, começando do zero para a primeira linha e atingindo o sete para a oitava, recomeçando do zero para a nona, e assim por diante. Dessa forma, o computador consulta o ASCII ou código de display na memória de vídeo, multiplica-o por 8 e adiciona ao valor do contador de linha naquele momento. Isto fornece um endereço do gerador de caracteres para um padrão de 8 bits que corresponde à linha correta do caractere que está sendo examinado.

Tomemos o exemplo da geração do caractere A, que tem um código ASCII de 65. Podemos calcular que a primeira linha do caractere será armazenada no byte de número 520 (65 × 8 + 0); a segunda linha no byte 521 (65 × 8 + 1); a terceira linha no byte 522 (65 × 8 + 2) etc. Resta apenas para o circuito de vídeo converter aqueles 8 bits numa seqüência de voltagens que ligará ou desligará o raio de elétrons varredor para mostrar os caracteres na tela.



Caracteres complexos
Os caracteres japoneses, para quem não os conhece, são bastante complicados, e a matriz normal 8 × 8 não mostra detalhes suficientes para que eles sejam lidos. O caractere para "ponte" (ao lado) só pode ser lido em matriz de 16 × 16. Obtém-se maior clareza com matriz de pontos de 24 × 24.

Questão de segurança

A "paridade par" garante que o número de bits de valor 1 em 1 byte seja sempre par. Assim, detectam-se erros de transmissão.

Uma das principais vantagens dos computadores digitais em relação aos dispositivos analógicos é que as falhas e imprecisões comuns nos circuitos elétricos não se acumulam à medida que os sinais passam por muitos circuitos (ver p. 239). Todavia, quando os dados são transmitidos a qualquer distância — seja por uma interface serial e um par de fios, seja por uma linha telefônica —, o "ruído" elétrico de fundo na linha pode às vezes ser suficiente para alterar um único bit de 0 para 1, e vice-versa. Normalmente, o equipamento receptor não tem meios de distinguir se isto aconteceu e aceitará os dados errôneos como corretos.

Observe o que ocorre se um bit no código ASCII para a letra Q for adulterado:

[ ] 1010001 (Código ASCII transmitido para Q)
[ ] 1000001 (Código ASCII recebido para A)

Um erro como esse na transmissão dos dados será, no mínimo, desagradável e, potencialmente, uma catástrofe. Entretanto, você se recorda de que os códigos ASCII são atribuídos apenas para valores até o total de 127, o que requer a utilização de apenas sete bits (numerados de 0 a 6). O bit mais significativo (bit 7) é, portanto, freqüentemente utilizado como bit "de paridade", para detectar a ocorrência de erros.

Há duas convenções para a utilização de bits de paridade: "paridade par" e "paridade ímpar". Examinaremos a primeira. A expressão "paridade par" significa que o bit de paridade (o bit 7, em código ASCII) é fixado de modo que o número total de bits de valor 1 no byte seja sempre um número par. Eis como as letras A e Q seriam representadas com paridade par:

[0] 1000001
(o código ASCII para A, pela paridade par)
[1] 1010001
(o código ASCII para Q, pela paridade par)

Há dois bits de valor 1 no código ASCII para A; assim, o bit de paridade assume o valor 0, de modo que o total de todos os oito bits seja par. No código ASCII para Q há três bits de valor 1; assim, o bit de paridade é constituído como um dígito 1. Isto eleva para 4 (que é par) o número total de bits de valor 1.

Agora examinemos o que acontecerá se o bit 4 em nossa letra Q, em código ASCII, for adulterado, como no exemplo acima.

[1] 1000001 (Q em ASCII adulterado)

Quando a paridade do byte for verificada (seja pelo software ou por hardware especial), será observado que a letra Q correta tem um número par de valores 1 (inclusive o bit de paridade). A letra Q adulterada, entretanto, teve o bit 4 acidentalmente alterado de 1 para 0, mas o bit original de paridade — bit 7 — permanece com o valor 1. Ao ser conferida a paridade deste byte adulterado, será verificado que ele possui um número ímpar de bits 1 e, desse modo, será detectada a adulteração do byte, que poderá então ser rejeitado. Se você refletir sobre isso, notará o seguinte: mesmo que o próprio bit de paridade venha a se adulterar na transmissão, a ocorrência do erro será detectada pelo processo de verificação de paridade, e o byte será rejeitado.

Ao observar o código ASCII utilizado por seu próprio computador, você notará que o bit 7 (o bit mais significativo, ou MSB, Most Significant Bit) é efetivamente utilizado, mas não como um bit de paridade. Isso ocorre a fim de possibilitar ao computador a recepção de um conjunto de caracteres adicionais (geralmente um conjunto de caracteres gráficos), e também porque os erros de transmissão de dados no *interior* do computador são muito raros. A paridade em geral é empregada apenas na transmissão de dados através de longas distâncias ou no registro de dados em superfícies magnéticas de gravação (como fita ou disco), que também é suscetível de "erros de bit".

A verificação de paridade é excelente para indicar a transmissão incorreta de um byte, mas não identifica qual dos bits foi incorretamente transmitido; assim, o erro não pode ser corrigido pelo equipamento receptor. Pior ainda, se dois bits em um byte forem adulterados, o byte incorretamente transmitido poderá ser tomado como correto.

Nos casos, porém, em que o erro é detectado pelo dispositivo receptor, este pode enviar uma mensagem de erro ao equipamento emissor e o software fará com que o byte seja transmitido novamente de modo correto. Processos mais sofisticados de detecção e correção de erros foram criados; localizam com precisão o bit ou bits incorretos, fazendo a correção automática. Os códigos de correção são um assunto a ser examinado posteriormente no curso.

Apenas para verificação
O último dígito do Padrão Internacional de Codificação de Livros (International Standard Book Number, ISBN) consiste em um dígito de verificação, equivalente à paridade em um computador. Multiplique o primeiro dígito (aqui, 0) por 10, o segundo (5), por 9, e assim sucessivamente, e a seguir some os resultados. Você notará que o dígito de verificação foi determinado de modo que, somado ao total, o resultado seja exatamente divisível por 11.

Trabalho de detetive

Quando a informação passa de um computador para outro, há o risco de ficar deturpada. Os códigos de Hamming podem detectar e corrigir o erro.

Quase todos nós já ouvimos histórias de erros terríveis cometidos por computadores, como o que resultou na remessa por uma empresa de quinhentas cópias de um folheto à mesma pessoa. A verdade é que nem sempre a máquina tem culpa: o engano provém de falha humana, talvez um simples erro de digitação. Ocasionalmente, surge um erro porque o programa aplicativo não foi escrito para prever todas as eventualidades, como no caso de uma emissão pelo computador do último aviso de cobrança de uma conta de luz no valor de Cr$ 0,00.

Entretanto, às vezes os computadores cometem erros que não podem ser atribuídos à intervenção humana e com freqüência se manifestam em forma de "erros de bits". Um erro de bit ocorre quando em uma seção de dados um bit é transposto de 1 para 0 ou vice-versa. Isto pode acontecer quando surge uma falha num componente do hardware — um chip de RAM, por exemplo. É por essa razão que os microcomputadores passam por uma rotina de software que procura diagnosticar erros para serem "diagnosticados", sempre que a força é ligada.

A maioria dos erros de bit são "erros de soft" — os bits ficam "invertidos", apesar de toda a RAM ter passado pelo teste do diagnóstico. Os microcomputadores são projetados para operar em condições ambientais normais, mas, durante uma onda de calor no verão, é possível que a temperatura exceda o limite de tolerância estabelecido para os componentes. É provável que o dano assim causado não seja permanente, mas os erros de bit talvez resultem na súbita mudança de um caractere na tela — um "A" mudar para "B", por exemplo, ou, caso o bit faça parte de um indicador importante, ele pode fazer o programa sofrer um "colapso", havendo necessidade de reprogramar o computador.

Os erros de bit também podem surgir durante períodos de intensa atividade solar, quando partículas subatômicas penetram na atmosfera e interferem no fluxo de elétrons em um circuito miniaturizado. Em aplicações tais como sistemas militares, controle industrial, experiências científicas ou serviços bancários internacionais, os erros podem trazer conseqüências desastrosas, e por isso vários métodos foram adotados para detectá-los.

O método mais simples é a verificação de paridade (ver p. 253). Um método alternativo é o "checksum" (teste de soma), muito usado na gravação de dados em disco ou fita magnética. Os dados são sempre manipulados em blocos de 128 bytes, e o último a ser lido ou escrito será o byte do teste de soma. Este byte répresenta o resto da divisão da soma de todos os outros bytes (cada um com valor de 0 a 255) por 256. Eis um exemplo:

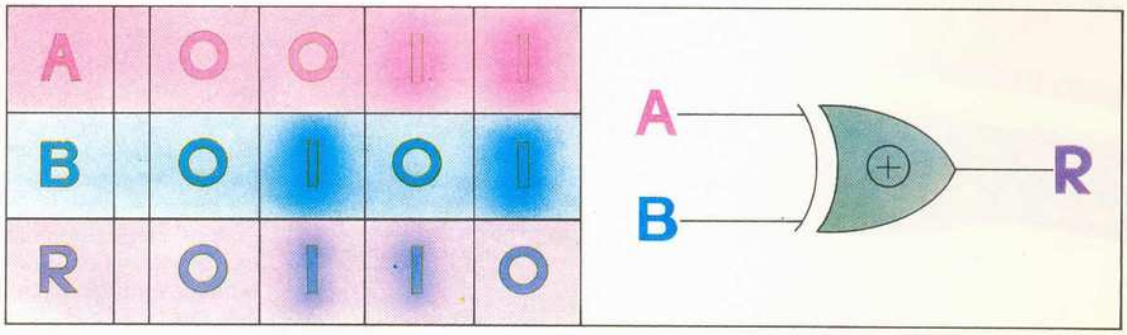
Dados: 114,67,83...(121 outros valores)... 36,154,198
Total dos 127 bytes = 16.673
Total dividido por 256 = 65, resto 33
Portanto, teste de soma = 33

O total dos bytes (16.673) é igual a 65 grupos de 256, mais um resto de 33 — este valor é inscrito no 128.º byte como um teste de soma. Quando o computador lê o bloco novamente, faz seus próprios cálculos de teste da soma com os dados e, se o valor diferir de 33, ele saberá que houve um erro de bit durante o processo de gravação.

Tanto com o método de paridade como no teste de soma, o computador não tem meios de saber qual bit foi alterado. Caso o erro tenha ocorrido na transmissão, o computador receptor pode solicitar que um determinado byte ou bloco de bytes sejam transmitidos de novo; no caso de um erro de gravação, talvez não haja maneira de recuperar os dados não alterados.

Quando os erros são absolutamente inaceitáveis, deve-se usar um sistema para detectá-los e corrigi-los. Os códigos de Hamming, que levam o nome de seu inventor, R. W. Hamming, dos Laboratórios da Bell de Telefone, desempenham esta função.

Todos os sistemas de correção de erros funcionam segundo o princípio da redundância. As linguagens humanas apresentam um alto grau de redundância: se ocorre um erro de datilografia em um texto, ou um ruído no telefone faz com que se perca alguma palavra da conversação, às vezes é possível recons-



Exclusive-Or (Exclusivo-Ou)
Uma porta Exclusive-Or simples tem dois inputs e um output. Se os dois inputs estiverem em 0 lógico, o output será 0. Se cada input for 1, o output também será 1. Entretanto, se os dois inputs forem 1, o output será 0. Esta última condição diferencia a porta Or da Ex-Or (abreviação). A operação pode ser representada por uma tabela de validação. Onde uma Ex-Or tem mais de dois inputs, o output será 1 se houver um número ímpar de dígitos 1 no input. Estes dispositivos são os meios para criar os bits de paridade e os verificadores de erros.

tituir a palavra considerando-se o contexto da frase. Outras vezes, fazemos uma redundância extra, usada em ambientes "barulhentos": é o recurso de *a* de "amor", *b* de "bola" e *c* de "casa", em radiotelefonia, por exemplo.

Suponha que enviamos pelo nosso computador uma palavra que tem x bits de extensão, consistindo em y bits de dados reais e z bits redundantes (isto é, $x = y + z$). Em nossa explicação de paridade, tínhamos y igual a 7 e z igual a 1. Para os códigos de Hamming, z deverá ser proporcionalmente maior. Agora admitamos que um erro de bit pode ocorrer em qualquer um dos x bits (os z bits redundantes são tão passíveis de erro quanto os y bits de dados). Se a possibilidade de ocorrer um erro de bit numa palavra é de, digamos, uma em um milhão, então a probabilidade de dois erros numa palavra é de uma em um trilhão, e portanto podemos ignorar esta possibilidade.

Quando os dados são recebidos na outra extremidade, haverá x + 1 eventualidades: poderá não haver erros, ou o primeiro bit dos dados estará errado, e assim por diante até o x bit. Agora, com z bits redundantes podemos representar 2^z situações, de modo que a palavra seja à prova de um erro de bit:

$$2^z \geq y + z + 1$$

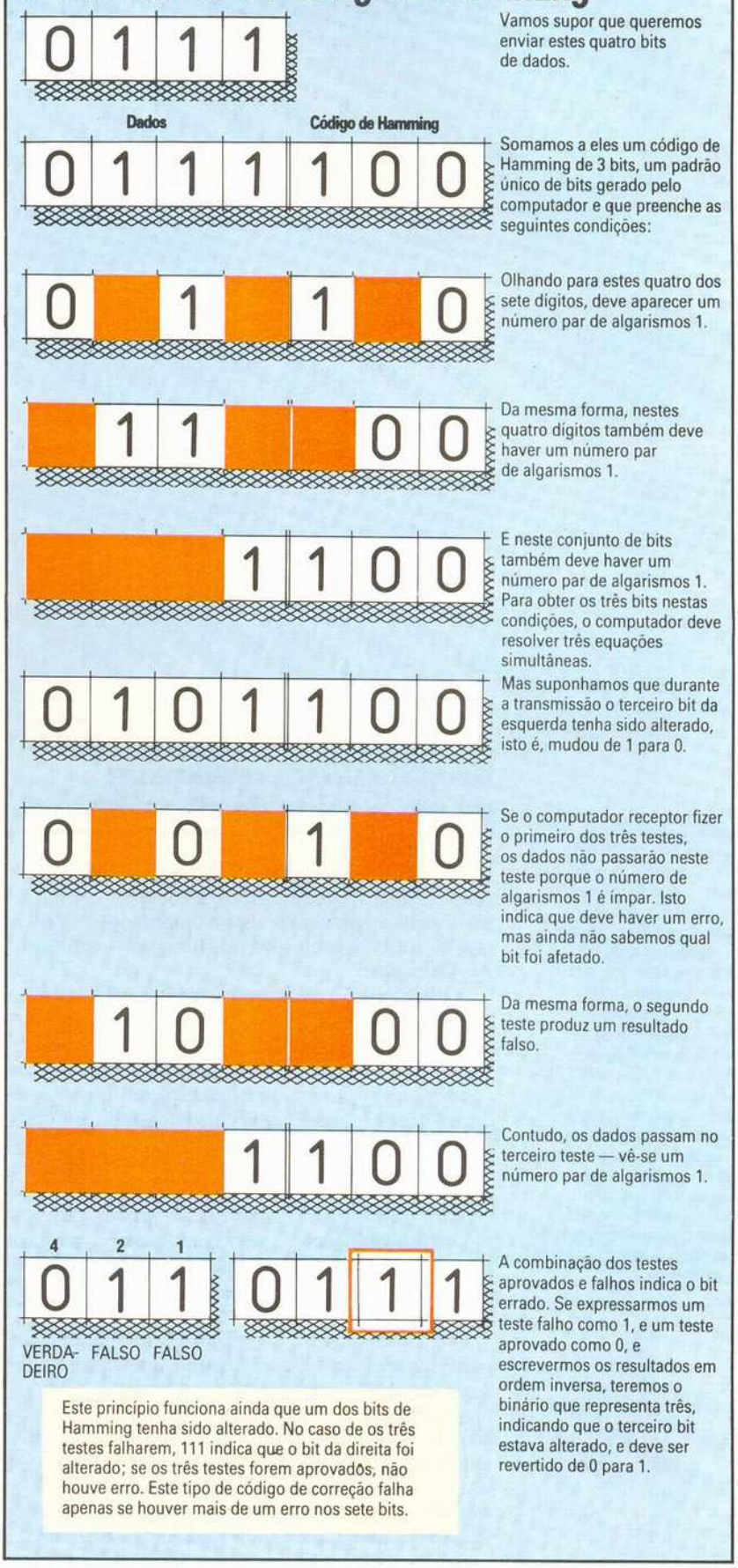
Se y é igual a 7 (pelos códigos ASCII), z deve ser igual a 4. Se y é igual a 4 (como no nosso exemplo do quadro), z deve ser igual a 3. Entretanto, se y é igual a 16, z aumenta apenas para 5. Segue-se que os códigos de Hamming são bem mais eficientes com palavras mais longas do que com palavras curtas.

Em um código de Hamming, cada um dos bits redundantes atua como uma verificação de paridade-par de uma combinação diferente de bits em uma palavra. Se algum bit ficar invertido na transmissão, um ou mais bits de checagem estarão errados e a combinação destes bits mostrará o bit errado na palavra (ver exemplos). O software do computador receptor pode então invertê-lo novamente, fazendo-o voltar à forma certa.

O ponto-chave do funcionamento dos códigos de Hamming são as diferentes combinações de bits sobre as quais cada bit de Hamming atua como verificador de paridade. O número total de bits é dividido em vários conjuntos diferentes, mas superpostos — planejados de modo que dois bits nunca apareçam na mesma combinação de conjuntos. O computador receptor desempenha funções de verificador de paridade nos mesmos conjuntos em que o dispositivo emissor já as desempenhou para criar o código de Hamming. Se algum dos bits, inclusive os de Hamming, foi invertido na transmissão, então um ou mais desses conjuntos não passarão no teste de paridade. A combinação dos testes não aprovados indica um bit único.

Alguns computadores empregam os códigos de Hamming até em suas operações internas de memória. Quando isto acontece, é possível remover um chip inteiro de RAM e constatar que o computador continua funcionando! Alguns computadores de uso militar levam o princípio de redundância ao extremo de duplicar todos os componentes do computador e comparar os resultados das duas metades após cada operação.

Como funciona o código de Hamming



Este princípio funciona ainda que um dos bits de Hamming tenha sido alterado. No caso de os três testes falharem, 111 indica que o bit da direita foi alterado; se os três testes forem aprovados, não houve erro. Este tipo de código de correção falha apenas se houver mais de um erro nos sete bits.

Controle editorial

A maioria dos micros permite editar programas na tela, poupando tempo e trabalho.

Todos cometem erros quando usam o teclado de um computador, e por essa simples razão é necessário ter recursos de edição. Existem diversas situações em que podemos querer mudar os dados de uma tela: corrigir um erro de digitação, alterar algum texto incorreto ou atualizar informações que mudaram depois de introduzidas.

Muitos programas aplicativos incluem alguma forma de "editor" especializado. As funções de edição de um processador de palavras, por exemplo, são planejadas para fazer as alterações necessárias nas diversas etapas de preparação de cartas e relatórios. Entre elas incluem-se a capacidade de eliminar frases, mudar parágrafos inteiros de lugar no texto e trocar todas as ocorrências de um nome ou frase por outros novos.

Entretanto, quase todos os microcomputadores possuem algum tipo de editor — embutido no sistema operacional em ROM — para preparar listagens de programas. Estes são extremamente vulneráveis a erros. Em qualquer programa ocorrem erros de sintaxe com grande regularidade e em sua operação certamente haverá "bugs" (erros) que deverão ser eliminados, sem falar nos ajustes que podem vir a ser necessários. Os recursos do editor do seu computador podem fazer muita diferença no desenvolvimento de um programa longo. Devemos ressaltar, no entanto, que um bom programador passa um tempo considerável testando a operação de seu programa no papel antes de digitar a listagem no computador. Considera-se um mau procedimento de programação digitar a primeira solução que vem à cabeça e passar 90% do tempo de desenvolvimento do programa corrigindo-o.

Há dois tipos de editor: "editor de tela" e "editor de linha". O primeiro é mais flexível e fácil de usar, mas o segundo é o mais comum nos microcomputadores. O editor de linha data da época em que toda computação era executada através de teletipos ou terminais de um computador remoto. Os teletipos tinham uma memória buffer para apenas uma linha de 80 caracteres (80 bytes). O programador podia obter uma lista impressa do programa inteiro, bastando para isso digitar LIST, mas se fosse preciso fazer uma correção na linha 120, por exemplo, ele teria de digitar toda a linha novamente. Em alguns sistemas, digitando-se EDIT 120, aquela linha específica seria impressa, e então as alterações ou anulações poderiam ser feitas usando-se as teclas "backspace" (retrocesso) e "rubout" (apagar), porque ainda não era possível fazer inserções. Outros comandos, como DELETE (para uma quantidade específica de números de linhas), foram acrescentados, mas a limitação de ter de chamar e modificar uma linha inteira continuava presente.

Os editores de muitos microcomputadores ainda se comportam como se tivessem somente um buffer de uma única linha, quando, na realidade, a tela inteira é mapeada na memória — cada localização de caractere corresponde a 1 byte da memória.

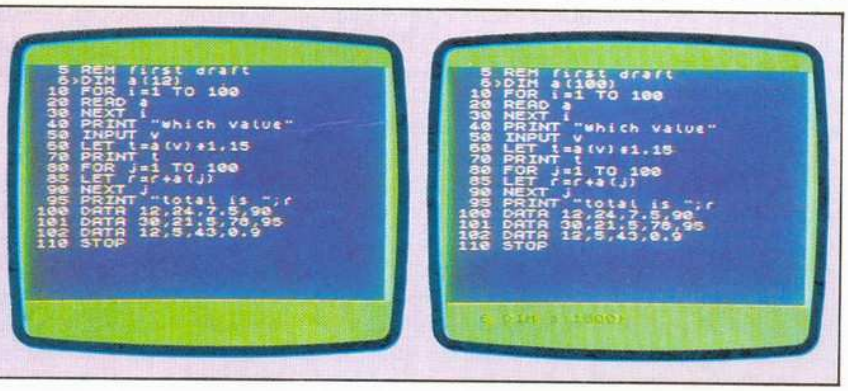
Um editor de tela é bem mais eficiente. Permite que você movimente textos e gráficos por toda a tela com facilidade. Toda vez que o usuário aperta RETURN, o editor lê a linha que está sendo indicada pelo cursor para o interpretador, onde ela é executada (se for um comando) ou introduzida no programa (se começar com um número de linha). Usando as quatro teclas com setas, o usuário pode movimentar o cursor para qualquer ponto do programa na tela e então inserir, retirar ou substituir caracteres à vontade.

Um editor de tela deve ser escrito em linguagem de máquina para conseguir a velocidade necessária, e pode apresentar recursos extremamente úteis. Os melhores editores de tela permitem a movimentação da listagem para cima e para baixo, bem como a inserção ou retirada de linhas inteiras ou caracteres individuais. Alguns até apresentam comandos semelhantes aos dos processadores de palavras que localizam e alteram todas as ocorrências de um determinado conjunto de caracteres.

Os editores estão ficando mais sofisticados e mais fáceis de usar a cada nova geração de computadores. Com a introdução do mouse (ver p. 296) e do software, que imita os processos manuais de cortar e colar pedaços de texto, o tempo despendido para editar um documento ou uma listagem até sua forma final vem diminuindo gradativamente.

Linha por linha

Os serviços de edição do micro inglês Spectrum são consideravelmente melhores que os dos demais, embora estejam longe de ser tão fáceis de usar como um editor de tela completa. Para trocar determinada linha de um programa, o marcador de posição (>) que aparece entre o número da linha e a linha propriamente dita deve ser movido.



Registro de trilhas

A função do Sistema Operacional de Discos (DOS, em inglês) é controlar o lugar em que as informações estão registradas nos discos. Sem o DOS, a programação seria uma tarefa árdua.

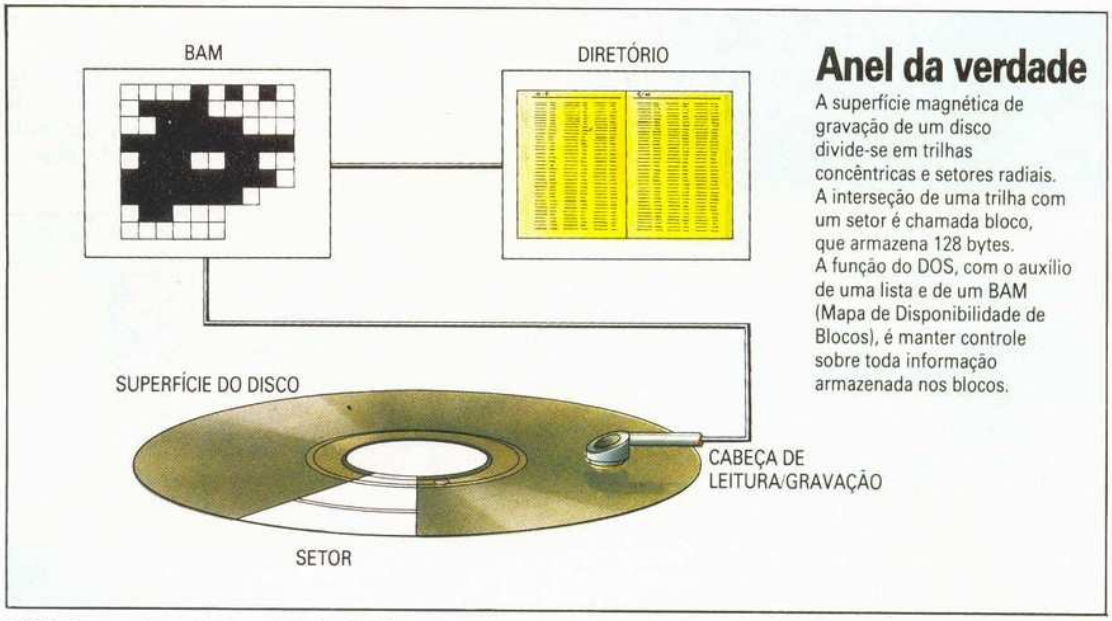
Antes que um computador possa executar qualquer tipo de programa aplicativo, precisa ter seu próprio conjunto de programas para dirigir as diversas partes do sistema e entender as instruções contidas no programa do usuário. Esse conjunto de programas internos é chamado Sistema Operacional, e na maioria dos microcomputadores fica permanentemente dentro do computador, em forma de memória ROM. Em geral não se percebe que o Sistema Operacional está funcionando, e por esta razão dizemos que ele é "transparente quando em operação".

Se o seu micro inclui uma unidade de disco, então uma grande parte do Sistema Operacional estará envolvida com as diversas operações relacionadas com os discos. Este conjunto de operações rotineiras é chamado Sistema Operacional de Discos (Disk Operating System, DOS). Estas três letras são encontradas nas marcas de muitos produtos do ramo — o sistema operacional da Microsoft, por exemplo, chama-se MSDOS. Um DOS pode vir de três formas diferentes. Primeiramente, ele pode fazer parte da ROM dentro do computador, como no micro inglês Sinclair Spectrum, que tem comandos para operar o microdrive embutido (que não consiste de fato num disco, mas é semelhante em operação).

Um outro tipo de sistema armazena o DOS em ROM dentro da própria unidade de disco. Este processo só é aplicável se o disco for um dispositivo "inteligente" (como o Commodore Disk Unit), ou seja, se tiver incorporado seu próprio microprocessador ROM e RAM. Este tipo é de fabricação mais dispendiosa, mas oferece vantagens consideráveis sobre unidades de disco "não inteligentes". Por exemplo, ele não ocupa a memória principal do computador e pode executar uma operação de disco complexa, enquanto o computador propriamente dito continua com o programa aplicativo.

Na terceira forma, o DOS fica dentro do computador em RAM. Esta técnica vem tendo grande aceitação nos sistemas de aplicação comercial, nos quais as unidades de disco vêm embutidas no computador, e há bastante RAM disponível (digamos, mais de 128 Kbytes, como medida padrão). Para o fabricante, este sistema oferece a vantagem de eliminar a necessidade de criar um conjunto totalmente novo de ROMs sempre que houver a mínima modificação no DOS, e para o usuário há a possibilidade de escolha entre uma variedade de marcas de sistemas operacionais que podem ser usados no mesmo hardware.

Ao ser ligado o sistema, imediatamente surge uma pergunta: como o DOS entra em primeiro lugar na RAM? O DOS tem de ser transferido do disco para a RAM, mas se não há DOS no computador para dizer-lhe como controlar o disco, como é que ele pode colocar algo na RAM? Um programa não pode colocar-se na RAM de forma espontânea; por isso, o computador tem de ter um programa minúsculo em ROM, executado sempre que a máquina é ligada. Este programa chamado "bootstrap" é uma forma muito simples de DOS. A tarefa do "bootstrap" é simplesmente achar o DOS principal no disco, transferindo-o, byte por byte, para a RAM, depois do que o DOS pode assumir a direção e de-



Anel da verdade

A superfície magnética de gravação de um disco divide-se em trilhas concêntricas e setores radiais. A interseção de uma trilha com um setor é chamada bloco, que armazena 128 bytes. A função do DOS, com o auxílio de uma lista e de um BAM (Mapa de Disponibilidade de Blocos), é manter controle sobre toda informação armazenada nos blocos.

sempenhar funções bem mais sofisticadas. Este processo de ligar o computador e esperar o DOS assumir a direção chama-se "booting-up". Quando ele termina, aparece um aviso na tela que indica que o computador está pronto para receber ordens do usuário.

Qualquer que seja a forma tomada pelo DOS no sistema, sua função principal é controlar o lugar em que as informações estão contidas no disco. Lembre-se de que um disco (ver p. 114) é dividido em anéis concêntricos, chamados trilhas, que, por sua vez, são divididas em setores; a interseção de uma trilha com um setor denomina-se bloco. Um bloco contém 128 bytes de informação, e é a menor unidade que o disco pode ler ou gravar por vez. Uma das principais razões para a existência do DOS é que ele faz o computador lembrar o local exato de tudo o que está contido no disco, e esta tarefa é mais impressionante do que parece. Suponhamos que uma unidade de disco tenha uma capacidade de 320 Kbytes — suficiente para armazenar vinte programas de 16 Kbytes cada. Uma vez que cada bloco contém 128 bytes, para carregar um destes programas sem se utilizar do DOS seria necessário que você especificasse 128 blocos diferentes, cada um com sua própria trilha e seu próprio número de setor!

Para poder desempenhar esta função, o DOS possui um diretório (uma espécie de lista de endereços), normalmente localizado na trilha central do disco,

Diretório local

Arquivo	Tipo	Local (trilha-setor)
Invaders	Prog	20-1,20-7,20-2...
Temperat	Prog	25-11,26-5,26-12...
Budget	Prog	23-12,24-3,24-9...
Budgetdat	Dados	27-1,27-7,27-2...

O diretório sempre ocupa a trilha central de um disco. Contém os nomes e os tipos de arquivos (programa, dados e talvez outras categorias), e números da trilha e do setor em que o arquivo está armazenado.

porque tendo de ser consultado com freqüência isto minimiza a distância que a cabeça de leitura/gravação precisa percorrer. A velocidade de operação de um disco depende mais do tempo despendido para mover a cabeça, de uma trilha para outra, do que da velocidade de rotação do disco.

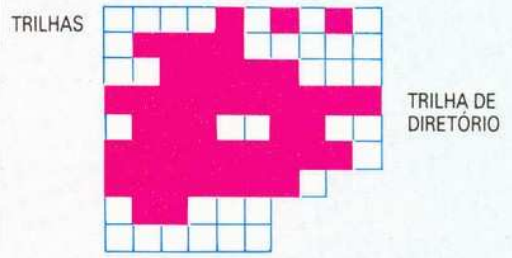
O diretório é uma lista que contém tudo o que está arquivado no disco (tanto programas quanto arquivos de dados), com detalhes como nome e tipo de arquivo e uma lista dos blocos (todos com especificação de trilha e setor) onde ele está armazenado. Pode haver outras entradas, tais como os dados de uma cópia back-up (de reserva) que acabou de ser feita, ou uma lista dos usuários que têm acesso a determinado arquivo.

Quando um novo arquivo é armazenado, o DOS deve consultar uma espécie de lista de endereços, chamada Block Availability Map (BAM) ou Free Sector List, que tem um bit correspondente a cada bloco do disco e, quando o bloco é usado, o valor de seu bit muda de 0 para 1.

Alguns computadores de uso pessoal com unidades de disco possuem um programa utilitário que apresenta o BAM na tela, e pode-se ver as entradas sendo feitas enquanto você grava um programa. Quando um arquivo é apagado, o DOS não se ocupa em limpar todos os blocos usados — ele muda as entradas no BAM, para indicar que o conteúdo daqueles blocos passou a ser desnecessário.

Outro aspecto deste sistema é que os arquivos não

Espaço de sobra



Antes que o DOS possa armazenar um arquivo novo e fazer uma entrada no diretório, ele deve consultar o Block Availability Map (BAM) ou Free Sector List. Esta é uma parte da memória onde cada bit corresponde a um bloco do disco. O binário 1 indica que o bloco está ocupado e o 0 indica que está livre (representados por quadrados cheios e vazios). Note-se que as trilhas que ficam mais próximas do centro do disco por serem mais curtas, têm menos setores.

são armazenados em blocos vizinhos consecutivos, como se poderia supor. Admitamos, por exemplo, que uma trilha consista em doze setores, numerados de 1 a 12, no sentido horário. O primeiro conjunto de 128 bytes de um programa encontra-se no setor 1; o segundo, no setor 7; o terceiro, no setor 2, e assim por diante. Isso acontece porque, enquanto o conteúdo de um bloco é transferido para o buffer (memória intermediária), usado para gravar o bloco, transcorre um pequeno lapso de tempo. Se o DOS precisasse gravar setores consecutivos, teria de esperar uma rotação completa do disco entre cada gravação, o que retardaria o sistema. Além do mais, um

Teste de QI
Algumas unidades de disco contêm seu próprio microprocessador e sua RAM. Estas são chamadas unidades "inteligentes", onde o DOS está incorporado em forma de ROM. Quando são usadas unidades "não inteligentes", o DOS fica armazenado dentro do computador.

disco que tem certo tempo de uso, com arquivos que mudam de tamanho todo dia, acaba com um BAM parecendo um pedaço de queijo suíço, e assim os novos arquivos têm de ser encaixados nos buracos.

Um Sistema Operacional de Discos possui muitas outras funções, inclusive a de formatar novos discos (marcar trilhas e setores em discos virgens e criar diretórios vazios), fazendo cópias back-up e "arrumando" discos cheios. Versões mais sofisticadas incluem uma variedade de estruturas de manipulação de dados (ver p. 204).

Passo a passo

Na medição de um movimento linear ou angular por meio de um sensor óptico, a codificação binária não produz bons resultados — por isso foi estruturado o código Gray.

Sistemas diferentes
Abaixo, os decimais de 0 a 15, nos códigos binário e Gray.

Decimal	Binário	Código Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Existem muitas tarefas em que a posição física de um objeto móvel deve ser determinada com precisão e passada para o computador. Em robótica, por exemplo, o computador precisa ter conhecimento de todas as posições e orientações dos membros de um robô e, na fabricação de máquinas com controle computadorizado, a posição da mesa fresadora deve ser estabelecida com precisão. Mas, como é que uma posição pode ser convertida em valor binário, para ser processada pelo computador?

Um dos métodos inclui o uso de um sistema analógico. Consiste na ligação da peça móvel a uma resistência variável, e a voltagem resultante é passada para um conversor analógico-digital (ou diretamente para a porta analógica, caso seu computador tenha uma). Entretanto, este sistema não oferece muita precisão e as partes mecânicas estão sujeitas a desgaste.

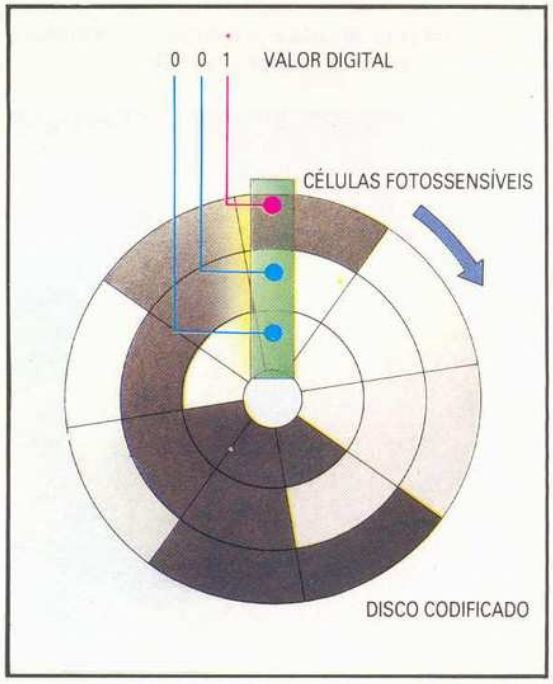
A alternativa seria gravar um código binário na peça móvel, passando-o diretamente para o computador. Em geral, o código é gravado em forma de arranjos de blocos brancos e pretos, na parte superior da peça, e lido por um foco de luz que incide sobre um desses arranjos, em conjunto com uma linha de células fotossensíveis, cada uma responsável por um dos dígitos no arranjo binário. À medida que a peça se move, os diferentes arranjos vão passando sob as células luminosas, e isto produz um resultado binário, que define a posição do objeto. Além dos arranjos lineares, são também empregados padrões radiais, que codificam movimentos angulares, como o da articulação do braço de um robô.

Entretanto, quando a peça se movimenta de um código binário para outro, podem surgir alguns problemas, principalmente se a peça parar a meio caminho entre dois códigos. A precisão da gravação tem uma tolerância finita e, quando a peça pára numa junção entre dois códigos, as células luminosas fazem a leitura de qualquer um dos dois. Se a peça parar no ponto em que as células luminosas estiverem incidindo sobre a junção da posição binária 11 (1011) com a 12 (1100), por exemplo, então só o bit mais significativo (isto é, o 1 da esquerda) é que pode ser levado em consideração para fornecer a posição correta, enquanto as outras três células luminosas encontrarão valores conflitantes para ler. Em determinadas situações, todos os bits mudam, como no caso da junção do binário 7 (0111) com o 8 (1000); portanto, a menor falta de precisão na gravação pode produzir leituras incorretas em todas as células. O resultado seria um valor totalmente falso para a posição, e o computador não teria meios de saber que isso estava acontecendo.

Portanto, é preciso que haja um sistema de contagem alternativo, onde apenas 1 bit mude em cada movimento. Isto significa que só pode haver dúvidas sobre 1 bit em cada junção e o resultado pode apresentar um erro de, no máximo, uma posição. Este sistema alternativo é chamado código Gray, determinado pelas seguintes condições: passando de um valor para o seguinte, um único bit se modifica, e este deve ser sempre o que fica mais à direita, resultando na formação de um padrão único. Desta forma, se começarmos com 0000, como no sistema binário, o número 1 será representado por 0001. No entanto, para representar o número 2, devemos mudar o segundo bit da direita, obtendo 0011. Para o número 3, já é possível mudar o primeiro da direita e obter 0010. Observe como esta seqüência difere da binária para esses mesmos números: 0000, 0001, 0010, 0011.

O quadro mostra este processo ampliado até o equivalente ao número 15 do sistema decimal, e os números vêm acompanhados de seus respectivos equivalentes no sistema binário para simples referência. Como exercício, você pode calcular os valores do código Gray além deste ponto.

Computadores poderiam ser projetados para fazer cálculos matemáticos e funcionar internamente pelo código Gray, mas isto seria ineficiente e desnecessário. Assim, deve-se usar um meio de converter o código Gray no binário, o que pode ser feito com hardware ou software.



Ângulo de visão
A posição angular de uma peça pode ser lida por um computador através de um disco com código gravado. Uma luz incide sobre a peça e uma linha de células fotossensíveis detecta o arranjo do código. O sinal digital que é produzido simultaneamente muda à medida que a peça se move. O inconveniente de se usar o sistema binário como código é que, se o disco parar na junção de dois valores, o resultado produzido pode não fazer sentido. O código Gray evita este problema.

O mapa da mina

Usando uma linguagem de alto nível você não precisa saber o que acontece na memória do computador. Mas, para usar linguagem de máquina, precisa saber o que se passa lá dentro.

A unidade central de processamento de um computador tem uma capacidade de endereçamento que determina o número máximo de posições de memória que pode manipular; na maioria dos micros, essa capacidade é de 64 Kbytes.

Nesse espaço devem caber os programas em ROM e RAM que vêm com a máquina, os eventualmente acrescentados e mais todas as portas e chips de interfaces, que também são considerados posições de memória. Assim, um dos aspectos mais importantes na arquitetura de um computador é seu mapeamento de memória — a lista ou o diagrama que especifica a parcela de memória destinada a cada uma das funções da máquina.

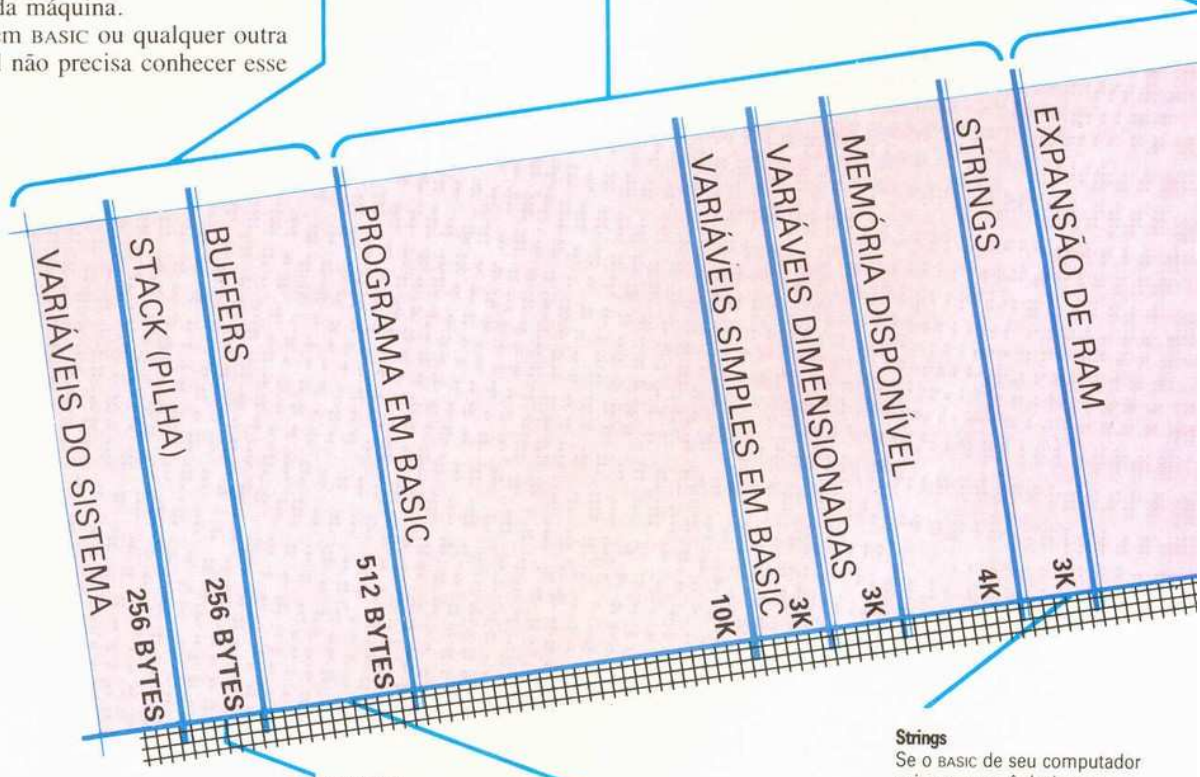
Quem só programa em BASIC ou qualquer outra linguagem de alto nível não precisa conhecer esse diagrama em detalhes; mas para quem quer utilizar linguagem de máquina ou construir acessórios de hardware esse conhecimento se mostra imprescindível.

Nestas duas páginas mostramos um típico mapa de memória, mais semelhante a um sistema baseado no microprocessador 6502 do que no Z80, embora quase todas as características sejam comuns aos dois Alguns fabricantes de microcomputadores incluem, no manual do proprietário, um completo mapa da memória. Mas há outros que fazem disso um segredo, por razões industriais. No entanto, grupos de usuários conseguiram, com a experiência, determinar todo o mapeamento.

Overhead do sistema
Um computador com 4 Kbytes de memória RAM pode ter apenas 3 Kbytes disponíveis para os programas do usuário. O restante fica para o overhead do sistema, parte da memória reservada pelo sistema operacional sempre que a máquina está ligada. Parte dessa área é ocupada com variáveis do próprio sistema, valores intermediários de cálculo e outros elementos

Área do usuário (RAM)
Talvez seja o ponto mais importante a ser avaliado na compra de um computador para uso pessoal: seu tamanho determina o grau de sofisticação dos programas que você pode usar

Vazio
O mapa de memória deve reservar espaço para expansão da RAM. Alguns sistemas permitem o acréscimo de mais do que 64 K, e um circuito especial seleciona os trechos que interessam ao trabalho, inserindo-os e removendo-os da memória principal



Stack (pilha)
Esta região está reservada para o uso da CPU e é organizada na forma de uma estrutura de dados acumulados (LIFO — Last In/First Out), onde só o que está no topo da pilha (justamente o último que entrou) pode ser manipulado. Quando uma instrução GOSUB é executada em BASIC, por exemplo, a CPU armazena no topo da pilha o endereço de memória para o qual deverá retornar (RETURN) depois de cumprir a sub-rotina

Buffers
A memória deve ter uma área que funcione como buffer de teclado, para que os caracteres não se percam caso entrem mais rápido do que podem ser processados. Deve haver também um buffer para o cassete, recebendo os lotes de dados que o sistema envia para serem gravados

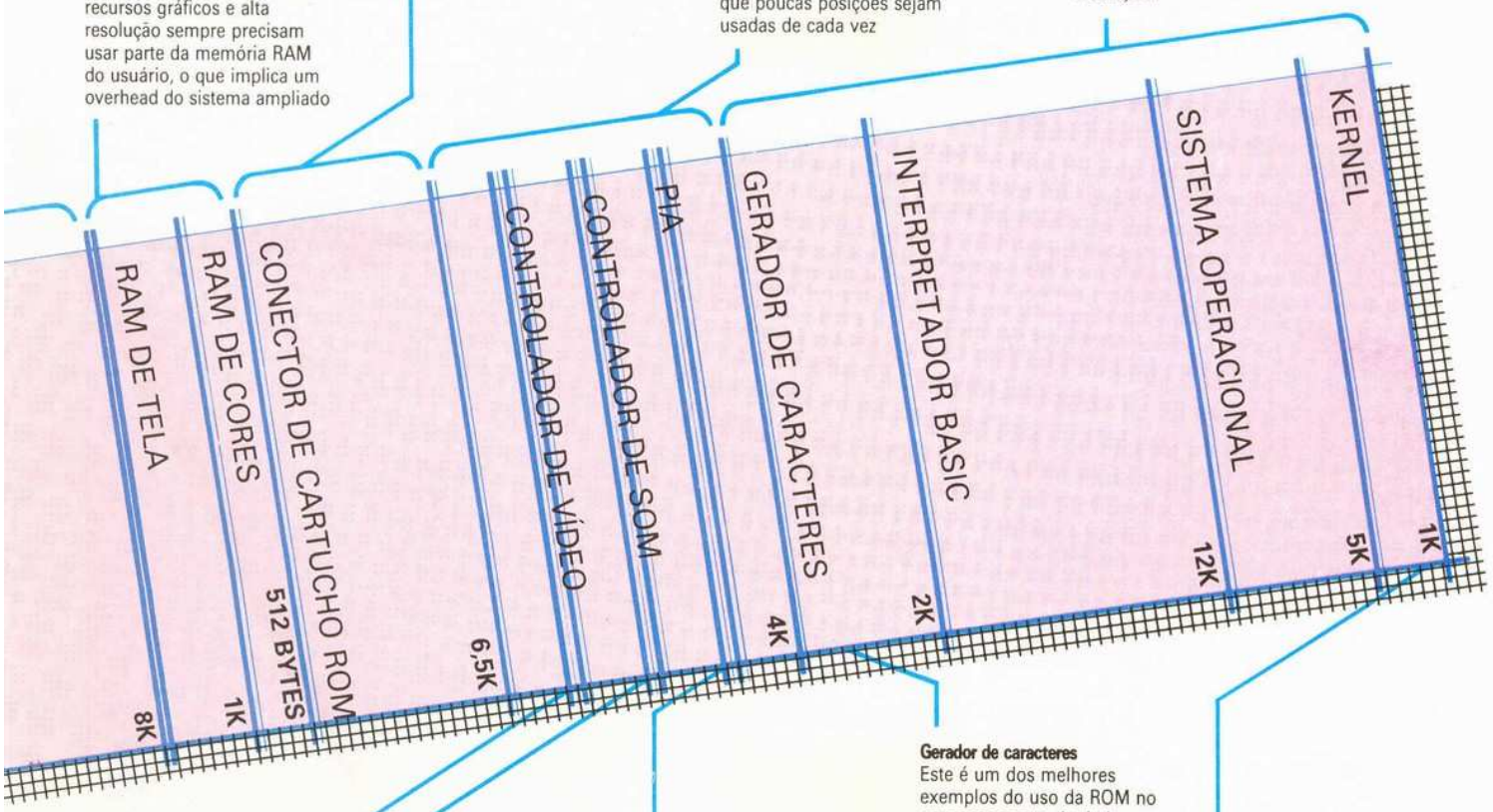
Strings
Se o BASIC de seu computador exige que você declare os tamanhos de todas as strings (variáveis alfanuméricas), elas são armazenadas numa tabela, como variáveis dimensionadas. Caso contrário, esses dados ficam numa região da memória que está sempre mudando de tamanho. Periodicamente, o sistema operacional faz uma "coleta de lixo" na área das strings, removendo os dados considerados desnecessários ao programa

RAM do sistema
Certos computadores dispõem de um trecho de memória para o sistema que não é discriminado como parte da RAM do usuário. Em geral essa região é usada como memória de vídeo (cada byte corresponde a uma posição na tela) e de registro de cores (cada byte corresponde à cor de um caractere e à cor de fundo de cada posição). Computadores com grandes recursos gráficos e alta resolução sempre precisam usar parte da memória RAM do usuário, o que implica um overhead do sistema ampliado

Vazio
Quando se introduz um programa por cartucho, ele é reconhecido no mapa de memória como uma expansão. Algumas máquinas têm conectores para cartuchos com linguagens diferentes do BASIC. O espaço para elas também está reservado no mapa de memória

Chips de entrada e saída
A CPU só se comunica com dispositivos que reconheça como posições da memória. Assim, todas as portas de interfaces e de outros circuitos devem estar no mapeamento. Isso inclui as interfaces de teclado, cassete, impressora e controlador de vídeo. Como a CPU faz endereçamento em lotes de 4 Kbytes, deve ser reservado um espaço de mesmo tamanho para os chips de entrada/saída, ainda que poucas posições sejam usadas de cada vez

ROM do sistema
Num computador, a ROM armazena informações que nunca se modificam. As mais importantes são as que compõem o sistema operacional — conjunto de programas que toma conta do funcionamento do computador. Esses programas cumprem tarefas como verificar se alguma tecla foi apertada e armazenar ou recuperar informação de um cassete. Na ROM fica também o interpretador BASIC, que transforma programas em instruções



Controlador de vídeo
Os recursos gráficos mais sofisticados têm sido baseados mais em hardware do que em software. No mapa de memória, o controlador de vídeo aparecerá como uma dúzia ou pouco mais de registradores de um byte, que determinam desde a cor de fundo para cada elemento a ser impresso até sua exata posição no vídeo

Controlador de som
Efeitos sonoros simples podem ser conseguidos com software. Mas os computadores com recursos mais sofisticados têm até controle de som, que alimenta um pequeno amplificador

PIA
Peripheral Interface Adaptors (adaptadores de interface de periféricos) são usados na comunicação do computador com periféricos simples como teclado, cassete, joystick e impressora. Os melhores (como o 6522 Versatile Interface Adaptor) convertem dados paralelos em seriais e têm cronômetros embutidos, usados em programação para controlar a velocidade de transferência dos dados

Gerador de caracteres
Este é um dos melhores exemplos do uso da ROM no armazenamento de dados ao invés de programas: ela guarda os bits que definem como cada caractere aparecerá no vídeo. Em certas máquinas, todos os caracteres ou parte deles são copiados na RAM; isso permite que o usuário defina seus próprios caracteres

Kernel
Constitui a alma do sistema operacional. Embora tenha um nome diferente em cada máquina, é o primeiro programa a ser executado pela CPU. Entre outras coisas, ele verifica o tamanho da memória disponível e se há programa em cartucho alimentando a máquina. Também manipula as formas mais elementares de entrada e saída de dados

Autor original

Pode-se desenvolver programas de computação que geram outros ou corrigem erros de codificação humana.

"Se os computadores são tão espertos, por que seres humanos devem programá-los?" Programadores experientes tendem a descartar questões como essa, formulada por um principiante cético. A dúvida, porém, não é tola como parece. Hoje se faz muita pesquisa de desenvolvimento de programas que podem dar origem a outros programas e de sistemas operacionais que objetivam corrigir erros em codificações escritas por seres humanos.

"ERRO SINTÁTICO?" é uma mensagem encontrada com freqüência por usuários de microcomputador. Pode ser irritante, porque fornece pouca informação. O compilador de um computador de grande porte fornecerá, em geral, muito mais informação quanto às características do erro encontrado. Por exemplo, a mensagem de erro pode se apresentar desta forma:

```
1090 LET A=(C*2+F$)*((FG-C)*TH+1))
ERROS: 1) CONCORDANCIA — VARIAVEL ALFANU-
          MERICA F$
          NAO ADMISSIVEL
       2) ULTIMO PARENTESE NAO ESPERADO
```

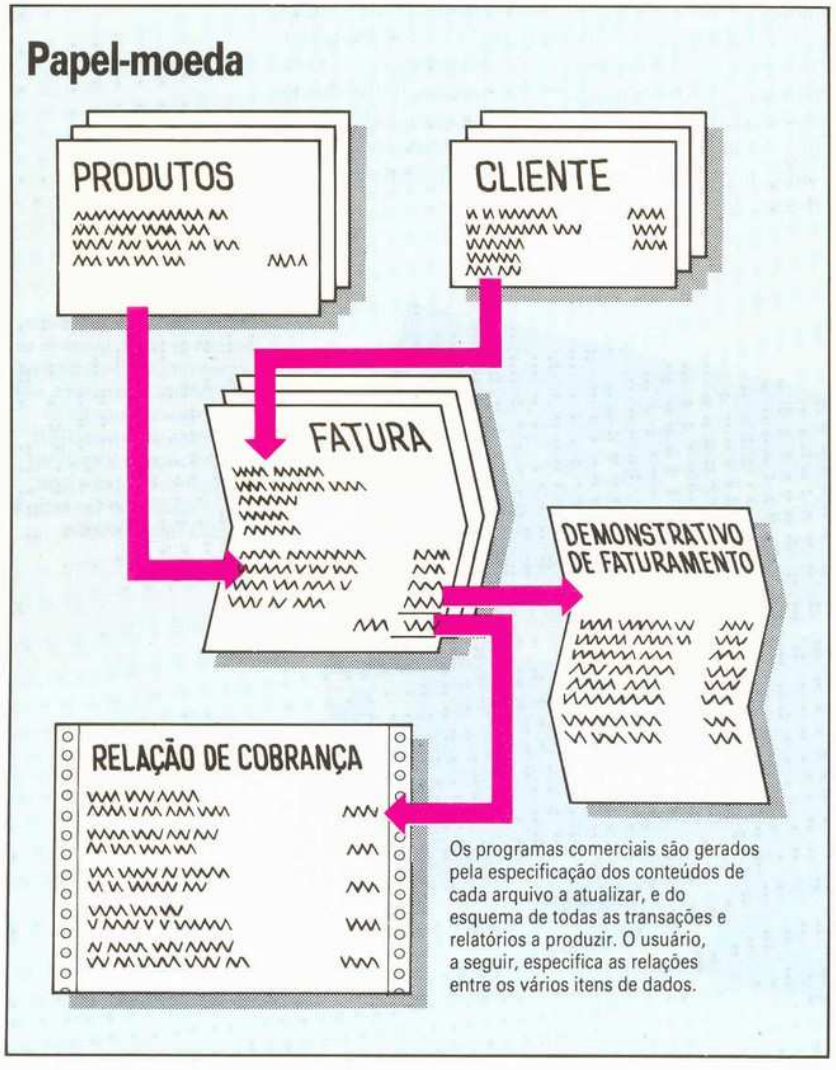
Não há razão séria para que tais técnicas não sejam empregadas num interpretador de micro — o custo da memória ROM adicional necessária para o armazenamento das rotinas seria mínimo. Mas poucos microcomputadores utilizam, pelo menos, procedimentos superficiais de supervisão de erro: a maioria nem chega a conferir a sintaxe do código à medida que é fornecido. Pode-se, contudo, comprar chips adicionais de memória ROM ou cartuchos de software acopláveis que ampliarão o alcance dos comandos disponíveis, sobretudo os relacionados com desenvolvimento e depuração de erros dos programas. Esses comandos BASIC abrangem:

HELP — Imprime a linha do programa e destaca a posição exata do caractere onde terminou a execução do programa. Isto, de modo geral, mas nem sempre, indicará a fonte do erro de sintaxe.

DUMP — Imprime uma lista de todos os nomes de variáveis, com seus conteúdos, que estiverem em uso no programa. É muito útil para se saber quanto o programa avançou em sua tarefa, antes da ocorrência do erro.

TRACE — Apresenta numa janela no canto da tela o número (ou números) da linha que está sendo executada, à medida que o programa se processa. Isso ajuda o usuário a traçar o fluxo do programa e assegura, entre outras coisas, que as sub-rotinas sejam executadas na ordem desejada.

O desenvolvimento de programas que permitam ao computador corrigir a codificação humana não é, em geral, tarefa simples. Porém, no caso de alguns erros, mostra-se muito fácil. Sabemos, por exemplo, que todas as linhas de programas devem começar por uma palavra-chave em BASIC (embora algumas máquinas admitam a não utilização da palavra LET). Assim, se uma linha se inicia pelo comando PRUNT ou PRONT, é fácil fazer com que diga PRINT. Neste curso de programação em BASIC, examinamos o conceito de combinação embaralhada (algoritmos para escolha da combinação mais próxima de qualquer frase) e isso também pode ser utilizado em palavras-chaves do programa. Da mesma forma, o interpretador pode apenas incluir uma lista de tipos de erros mais freqüentes de digitação com seus equivalentes corretos. Para maior segurança, seria con-



Os programas comerciais são gerados pela especificação dos conteúdos de cada arquivo a atualizar, e do esquema de todas as transações e relatórios a produzir. O usuário, a seguir, especifica as relações entre os vários itens de dados.

veniente que o computador conferisse com o operador quaisquer alterações feitas.

Mas, afora esses procedimentos elementares, a correção automática se torna bem mais difícil. No exemplo dado, F$ corresponde a um erro de impressão de F ou FS ou F4? Ou de algo inteiramente diferente? Se você mostrasse a listagem completa a outro programador competente, ele seria capaz de identificar as falhas e fazer as correções. Para tanto usaria dois critérios na tomada de decisão: o contexto no qual a linha do programa apareceu e sua própria experiência.

De modo bastante estranho, essa técnica tem sido mais utilizada na correção do texto em português ou inglês do que na verificação do código do programa. Um pacote de verificação de erros de grafia, por exemplo, trabalha o texto e destaca as palavras que não corresponderem às entradas em seu dicionário de cerca de 50.000 palavras, retidas em disco. A maioria desses pacotes dispõe de recursos para aprender novas palavras (como os nomes de empresas ou de pessoas) e as incluem em seus dicionários. Os pacotes mais elaborados até sugerem a maneira correta de soletrá-las, se uma correspondência aproximada for detectada. Processadores experimentais de palavras empregam os mesmos processos para correções gramaticais e de estilo literário indicando erros ou imprecisões de pontuação, exagerada repetição de palavras no mesmo parágrafo e adjetivos ou advérbios inadequados. Esses pacotes operam pelo exame do contexto de cada sentença e pela referência a uma biblioteca de exemplos.

Um empenho maior, contudo, foi dedicado ao desenvolvimento de sistemas que *criam* programas em vez de apenas *corrigir* os já existentes. Em 1981, surgiu no mercado o software habilmente denominado The Last One (O último). Pretendia ser um programa capaz de desenvolver qualquer outro programa que se desejasse. Em outras palavras, seria o último programa que se precisaria adquirir. Tal pretensão mostrou-se exagerada, embora The Last One fosse um auxiliar muito útil ao desenvolvimento de certos tipos de software, sobretudo para aplicações comerciais. Existem hoje vários produtos desse tipo no mercado, rodando tanto em micros comerciais quanto em alguns de uso doméstico — todos eles genericamente denominados ''geradores de programas''.

Examinemos agora o princípio básico que fundamenta um programa que pode desenvolver outros programas. Observe este exemplo bem comum:

```
10 PRINT "O QUE VOCE DESEJA QUE O
   PROGRAMA APRESENTE NA TELA?"
20 INPUT A$
30 PRINT "O PROGRAMA E:"
40 PRINT "10 PRINT ";CHR$(34);A$;CHR$(34)
```

Se responder OLÁ à pergunta, o programa (que deve rodar na maioria dos microcomputadores) imprimirá a seguinte linha:

```
O PROGRAMA E
10 PRINT "OLA"
```

Caso se utilize a mesma técnica nas fases de entrada de dados, de cálculo e de saída para a aplicação que se tem em mente, pode-se desenvolver um gerador bem simples de programas. Se todas as questões que ele solicita estiverem formuladas de modo claro, será fácil, mesmo para quem não tem experiência anterior, desenvolver um programa simples com o auxílio desse software.

Ferramentas do ofício

"Kits de ferramentas para o programador" podem ser adquiridos para muitos microcomputadores: são chips de ROM ou cartuchos adaptáveis. Ampliam o alcance dos comandos em BASIC, especialmente no desenvolvimento de programas e na eliminação de erros.

Os geradores de programas disponíveis no mercado utilizam as mesmas técnicas. A maioria das aplicações comerciais consiste numa combinação de cinco processos distintos: entrada de dados, saída para a tela ou impressora, armazenamento em arquivo, recuperação e cálculo. O gerador precisa dispor de sub-rotinas padronizadas e muito flexíveis para cada um desses procedimentos. Ao lhe pedir que especifique a estrutura exata dos dados que você estará usando, os cálculos feitos com esses dados e os formatos de saída de que você necessitará na tela e na impressora, o gerador vai alterar os valores de certas variáveis de algumas sub-rotinas e depois fazer a conexão para criar um programa.

Embora estejam se tornando mais sofisticados, é improvável que os geradores de programas venham a substituir o programador humano em futuro próximo, pois apresentam sérias limitações. A técnica descrita é adequada a aplicações comerciais em bases transacionais, como contabilidade ou controle de estoque, mas em geral não pode ser empregada no desenvolvimento de processadores de palavras ou programas de jogos. Em segundo lugar, como o gerador de programa recorre a sub-rotinas padronizadas, a listagem resultante nem se aproxima da eficiência (tanto em velocidade quanto em memória utilizada) que teria se fosse desenvolvida por um programador. Além disso, os programas produzidos por geradores não são tão *user-friendly* (fáceis de usar) como os produzidos pelo programador, quando este é um ser humano. Poucas vezes utilizam, por exemplo, os recursos gráficos oferecidos pelas máquinas mais modernas.

Enfim, os geradores de programas disponíveis no mercado conseguem de fato substituir apenas o último estágio da programação — o desenvolvimento do código. O usuário ainda tem de elaborar a forma exata da saída e entrada de dados de que precisa. Em geral, os estágios que antecedem a programação são os mais difíceis e exigem habilidade específica, diferente daquela necessária à programação. A maior parte das grandes empresas emprega especialistas, os analistas de sistemas, para especificar os programas necessários; e programadores convertem essas especificações em códigos.

Fim específico

Uncommitted Logic Arrays (ULAs) podem controlar as funções do microcomputador, independentemente de CPU, ROM e RAM.

Entre os muitos avanços da eletrônica, resultantes do rápido desenvolvimento do microcomputador, um dos mais importantes foi o aperfeiçoamento de um tipo de chip chamado Uncommitted Logic Array (matriz lógica sem funções determinadas) ou ULA. Embora distante do conhecimento público, essa revolução silenciosa vem acontecendo e está tão avançada que tornou possível a construção de computadores mais sofisticados e de outros dispositivos com apenas quatro componentes principais: a CPU, um pouco de RAM, um pouco de ROM e — para unir as três — uma ULA.

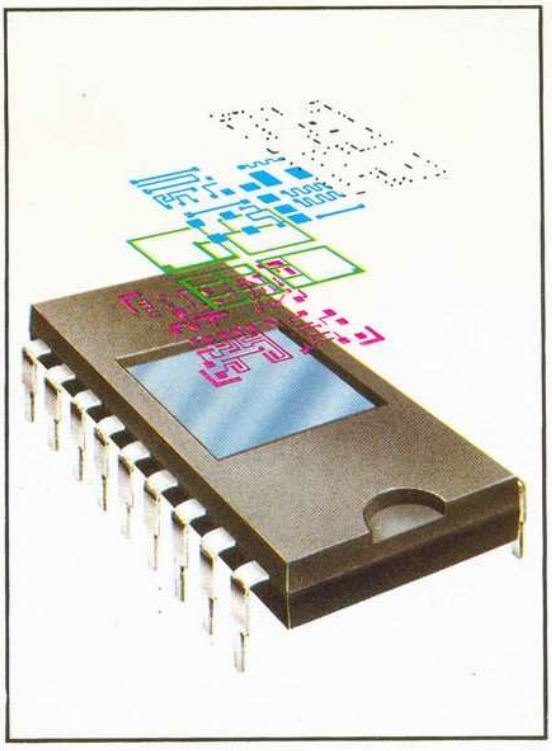
Como o próprio nome sugere, a ULA corresponde a um grande número (uma matriz) de portas lógicas que, de início, não têm funções específicas, mas podem ser modificadas para executar praticamente qualquer operação que o designer desejar. A ULA é considerada um desenvolvimento da ROM, uma vez que o conteúdo de ambas só se vê especificado pelo fabricante do chip, e não pelo usuário.

Antes de ser "programada", uma ROM ou ULA consiste apenas em grande número de circuitos ou células eletrônicas simples, que não estão ligadas e portanto não executam nenhuma ação. Todos os chips são constituídos de camadas sobrepostas de materiais semicondutores (ver p. 122). A camada final compõe-se de material condutor e forma conexões entre as diversas células. A ampla variedade de interligações possíveis dá flexibilidade à ULA; e, embora as células sejam bastante simples, consistindo num par de transistores, ou uma simples resistência, elas podem ser todas interligadas através da última camada, para montar circuitos complexos, tais como flip/flops (ver p. 305).

Constróem-se esses circuitos, chamados módulos, até com menos de meia dúzia de células, e já que uma ULA grande tem milhares delas, os módulos propriamente ditos podem se interligar para montar circuitos complexos, tais como máquinas registradoras, calculadoras e circuitos de tempo. As funções desempenhadas por esses circuitos são, com freqüência, executadas num microcomputador por uma coleção de chips lógicos de uso geral.

Uma ULA pode ser programada para desempenhar variedade muito diversificada de atividades. É possível fazer qualquer ULA sintetizar sons, controlar a exposição, o foco e o motor de uma câmara fotográfica ou fazer a maior parte do trabalho num termômetro digital. Além da ULA, quase nenhum circuito externo se faz necessário — apenas a bateria, o interruptor e alguns sensores ou botões de controle.

Os computadores são muitas vezes empregados para executar o projeto da camada que interliga as células da ULA. O minicomputador como um DEC PCP11/23, executando programas do tipo CAD — Computer Aided Design —, monta de início um diagrama codificado da lógica desejada. Então, o sistema traça e codifica um mapa do layout planejado. Tudo isso ocorre num terminal gráfico, e uma cópia impressa do projeto pode ser produzida num plotter, se houver conveniência.



Plano mais elevado
Todos os chips semicondutores são formados por camadas de materiais semicondutores, gravados um a um para criar os elementos do circuito. A camada final determina a conexão entre os elementos. A ULA consiste numa matriz de elementos lógicos combinados para formar um circuito lógico complexo.

Máscara óptica

Concluído, o design é trasmitido para um computador maior, que verifica se o plano é aceitável, compara-o com o projeto lógico original e certifica-se de que não contém erros graves. Então, ele é submetido a outro programa que simula o circuito resultante, usando um programa de teste preparado pelo usuário. Quando o projeto estiver terminado, o computador produz o trabalho de acabamento para a máscara óptica usada na execução da última camada.

As possibilidades da ULA são variadas. A idéia de colocar grande número de circuitos simples em silício e deixar o usuário decidir como eles devem atuar uns sobre os outros mostra-se tão fascinante que poderia resultar na execução de um número maior de circuitos. Contudo, no atual nível da tecnologia, as ULAs só são econômicas quando há necessidade de no mínimo alguns milhares de circuitos elétricos idênticos. PROM (Programmable Read Only Memory), EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM) e EAROM (Electrically Alterable ROM) são, todas elas, alternativas para a ROM, programável pelo usuário com equipamento adequado. Talvez, em breve, também apareçam alternativas para a ULA programáveis pelo usuário.

Código de ordenação

Quando se trabalha com matrizes extensas, a classificação Shell é mais eficiente que as classificações bolha ou por inserção. A Shell opera pela divisão dos dados em séries.

Examinamos, na página 286, dois métodos de classificação de matrizes — as classificações bolha e por inserção. Em geral, a bolha revela-se mais fácil de ser desenvolvida, mas a classificação por inserção opera com maior rapidez. A experiência com ambos mostra que boa parte do tempo se despende com a troca de cartas a curtas distâncias.

```
7999 REM**************
8000 REM* SHELL *
8001 REM
8025 PRINT "CLASSIFICACAO SHELL - INICIAR ! ! ! ! ! "
8050 LET LK=LT
8100 FOR Z=0 TO 1 STEP 0
8150 LET LK=INT (LK/11)
8200 FOR LB=1 TO LK
8250 LET LL=LB+LK
8300 FOR P=LL TO STEP LK
8350 LET D=R(P)
8400 FOR Q=P TO LL STEP-LK
8450 LET R(Q)=R(Q-LK)
8500 IF D<=R(Q) THEN LET R(Q)=D:LET Q=LL
8550 NEXT Q
8600 IF D>R(LB) THEN LET R(LB)=D
8650 NEXT P
8700 NEXT LB
8750 IF LK=1 THEN LET Z=1
8800 NEXT Z
8850 PRINT "CLASSIFICACAO SHELL - ENCERRAR ! ! ! ! ! "
8900 RETURN
```

Para incluir essa rotina no programa do procedimento de classificação, na página 287, altere a linha 350 para:

```
350 LET I=1:LET O=0:LET II=2:LET TH=3
```

e a linha 900 para:

```
900 ON SR GOSUB 6000,7000,8000
```

Um método mais eficaz que esses dois é a "classificação Shell". Esse método garante que a desordem na matriz seja reduzida no começo da classificação (de modo que os itens não fiquem muito distantes de suas posições efetivas) e possibilita a realização das trocas a distâncias longas. Um método para essa classificação inclui cinco procedimentos:

1) Disponha todas as cartas de um naipe em ordem decrescente, de modo que o rei seja a carta da extrema esquerda, e o ás, a da extrema direita. Conte as cartas, divida o número resultante (no caso, 13) por 2, deixando de lado a que restar, e registre o resultado (6) num pedaço de papel assinalado "Elo".

2) Coloque uma moeda pequena sob a carta da extrema esquerda (posição Um) e uma moeda maior na posição Elo (isto é, posição Seis, na primeira vez). Todas as cartas, da primeira até a posição Elo, serão as cartas da extremidade esquerda numa série de "correntes" de cartas. O número de correntes coincide com o valor do Elo naquele momento. Forma-se cada corrente a partir da carta da extremidade a que pertence, somando-se o Elo ao número de posição da carta da extremidade. Consegue-se desse modo a posição da carta seguinte, à qual se acrescenta o Elo para se obter a posição da próxima carta e assim sucessivamente, até que o final da matriz seja alcançado ou ultrapassado. A primeira corrente, desse modo, abrange as cartas nas posições Um, Sete e Treze; a segunda corresponde às cartas nas posições Dois e Oito e a terceira equivale às cartas nas posições Três e Nove. A última coincide com as cartas que ocupam as posições Seis (o valor do Elo no momento) e Doze.

3) Agora, tendo assinalado os limites com as moedas, desloque para fora da tabela as cartas incluídas na primeira corrente, de modo que possam ser vistas isoladamente, e proceda a sua ordenação, utilizando a classificação bolha ou por inserção, como descrito na página 286 (a listagem, nesse artigo, utiliza o método de inserção).

4) Desloque a corrente ordenada de volta a suas posições na matriz e repita o procedimento acima com a corrente seguinte, e assim por diante, até que todas as correntes cujas cartas da extrema esquerda caiam entre as moedas estejam ordenadas.

5) Quando as correntes estiverem ordenadas, divida o Elo por 2, deixando de lado a que restar. Se o Elo for agora menor que 1, a matriz estará classificada. Caso contrário, repita a operação a partir do procedimento 2 com o novo valor do Elo.

Painel da classificação Shell

Posição n.º	Valor de Elo	Comentários
1 2 3 4 5 6 7 8 9		
2 8 9 3 T 5 K 6 7	(9/2)=>4	Início da passagem
* + @ $ * + @ $ *		Formação de correntes
T 7 2		Corrente de classificação 1
8 5		Corrente de classificação 2
K 9		Corrente de classificação 3
6 3		Corrente de classificação 4
T 8 K 6 7 5 9 3 2		Início da passagem
T 8 K 6 7 5 9 3 2	(4/2)=>2	Fim da passagem
* + * + * + * + *		Formação de correntes
K T 9 7 2		Corrente de classificação 1
8 6 5 3		Corrente de classificação 2
K 8 T 6 9 5 7 3 2		Fim da passagem
K 8 T 6 9 5 7 3 2	(2/2)=>1	Início da passagem
* * * * * * * * *		Formação da corrente 1
K T 9 8 7 6 5 3 2		Encerramento da passagem

CHAVE	
*	Membro da corrente 1
+	Membro da corrente 2
@	Membro da corrente 3
$	Membro da corrente 4

Classificação Shell
O exemplo da classificação Shell com reduzida mão de cartas apresentado no painel demonstra seu excepcional método de divisão da matriz numa série de correntes (com espacejamentos no número do Elo em determinado momento). Essas cadeias são ordenadas uma a uma, no caso, pelo método de inserção, antes da passagem.

Esta listagem de programa para classificação Shell deve ser usada com o programa-teste da página 287.
Quando o testamos, houve considerável melhora com relação a outros métodos de classificação, sempre que o número de itens a serem ordenados excedia a quarenta.

Máquina abstrata

A máquina de Turing é um dispositivo puramente teórico, usado para decidir se um problema é computável ou não.

Até agora, no MICROCOMPUTADOR — CURSO BÁSICO, procuramos enfatizar assuntos práticos e intens com os quais você pudesse lidar em seu microcomputador. Neste artigo, contudo, vamos examinar o lado teórico dos computadores, o campo da "ciência da computação". Ele está para a informática como a matemática pura está para a engenharia — um assunto altamente abstrato, mas do qual derivam quase todas as idéias práticas.

A máquina de Turing, por exemplo, é uma idéia teórica desenvolvida para auxiliar no estudo de algoritmos e computação. Ela constitui um computador tão elementar quanto possível. Assim, se estudiosos provarem que determinado problema não pode ser resolvido pela máquina de Turing, consideram a questão "não computável".

Turing decidiu que esse computador mínimo precisaria de três funções: um armazenamento externo para gravar e arquivar as entradas e saídas de informação; um meio de ler e gravar esse arquivo; e uma unidade de controle para determinar as atitudes a ser tomadas.

Inclui-se na constituição da máquina de Turing, portanto, uma fita — para ficar mais claro, imagine uma fita magnética — de comprimento infinito (quer dizer: qualquer que seja a quantidade de fita necessária para a solução de um problema, sempre haverá o suficiente). Ela é dividida em quadrados, que podem estar em branco ou conter símbolos. O mecanismo da cabeça da fita, que pode ler ou gravar os símbolos nos quadrados, anda junto com ela, recebendo instruções de uma unidade de controle que determina quais os símbolos a ser gravados e que direção ela deve tomar em seguida.

Cinco quíntuplos

A unidade de controle contém um programa de execução e, nesse aspecto, pode-se considerar que a máquina de Turing foi "construída" para executar uma única aplicação, pois não há nenhuma cláusula em sua especificação para carregar ou alterar um programa. Usamos o termo "construída" entre aspas porque máquinas de Turing concretas só foram construídas com fins educativos. No entanto, é relativamente simples escrever um programa em BASIC que simule a operação de uma máquina de Turing num microcomputador.

O programa de controle de uma máquina de Turing é formado por uma coleção de "quíntuplos", ou frases compostas por cinco elementos. A decisão de qual quíntuplo deve ser executado em cada fase depende de dois fatores: o símbolo que está no quadrado embaixo da cabeça da fita, e o "estado" ou "condição" da máquina. O estado é uma qualidade arbitrária: pode-se especificar que a máquina começou no estado S_A, e, quando alcançar o estado especial H, ela pára (*halt*, parada), encerrando a computação. Nesse meio tempo, o estado mudará muitas vezes, de acordo com as instruções dos quíntuplos. Ele apenas reflete o que aconteceu até o momento na computação e serve para selecionar o quíntuplo que será executado em seguida. (Para ficar mais claro, imagine que seja uma variável de flag na programação BASIC.)

Os cinco elementos de cada quíntuplo são:

1) o estado da máquina;
2) o símbolo no quadrado da fita que está embaixo da cabeça;
3) o símbolo a ser inscrito no quadrado (é o mesmo que 2, se não houver qualquer mudança na informação);
4) o estado para o qual a máquina deverá ir em seguida; e
5) a direção em que a cabeça da fita deve se movimentar — para a esquerda (L) ou para a direita (R).

O quíntuplo (S_A, 5, 3, S_B, R), por exemplo, é executado sempre que a máquina estiver no estado S_A e a cabeça da fita ler 5. O 5 será então substituído por 3, a máquina passará do estado S_A para o estado S_B e a cabeça da fita andará um quadrado para a direita (R, de *right*, direita).

O projeto de uma máquina de Turing teórica para executar determinada tarefa implica especificar o formato do input da informação que será introduzido em forma de fita na máquina, o formato do output da informação em forma de fita quando a computação estiver terminada (isto é, quando a máquina estiver no estado H) e o conjunto de quíntuplos necessários para executar o algoritmo.

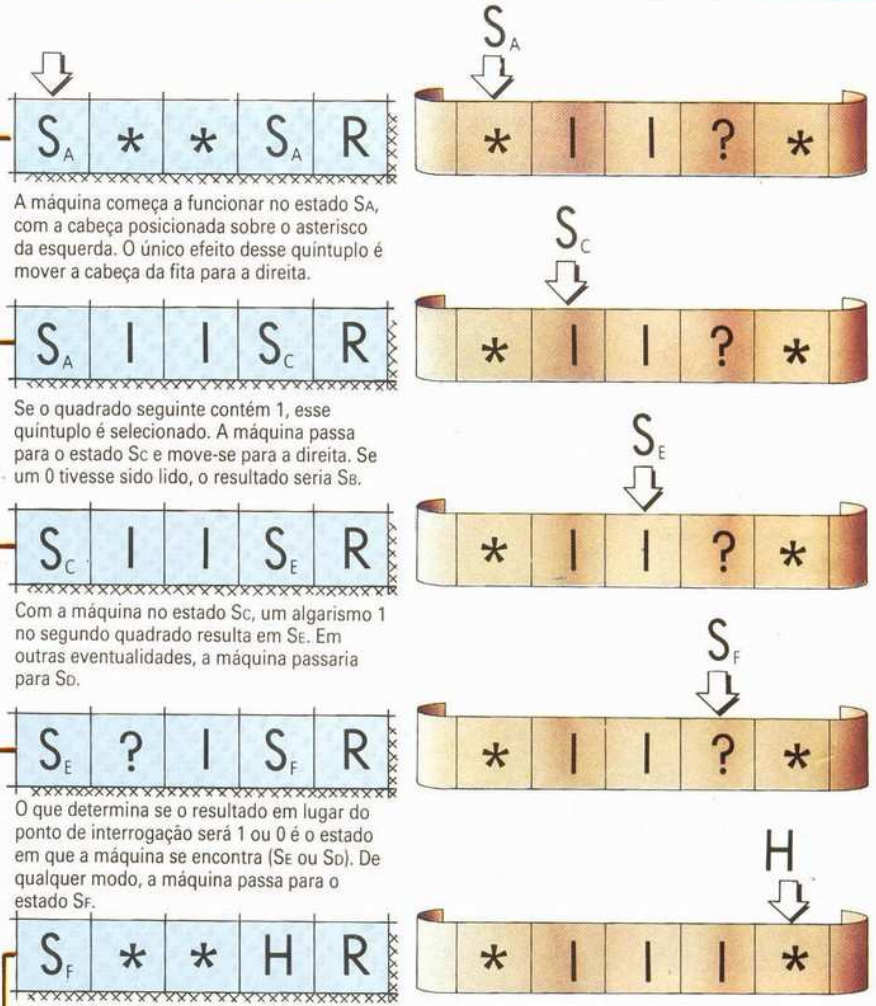
A ilustração mostra uma máquina de Turing para "desempenhar" a função AND. Colocamos os dois bits do input (1 ou 0) em quadrados adjacentes, seguidos de símbolo representado por um ponto de interrogação, que deve ser substituído pela resposta (de novo com os algarismos 1 ou 0, dependendo dos dois inputs). Por uma questão de ordem, acrescentamos um asterisco a cada ponta do segmento de informação, e ligamos a máquina no estado S_A, no asterisco da esquerda, terminando no da direita.

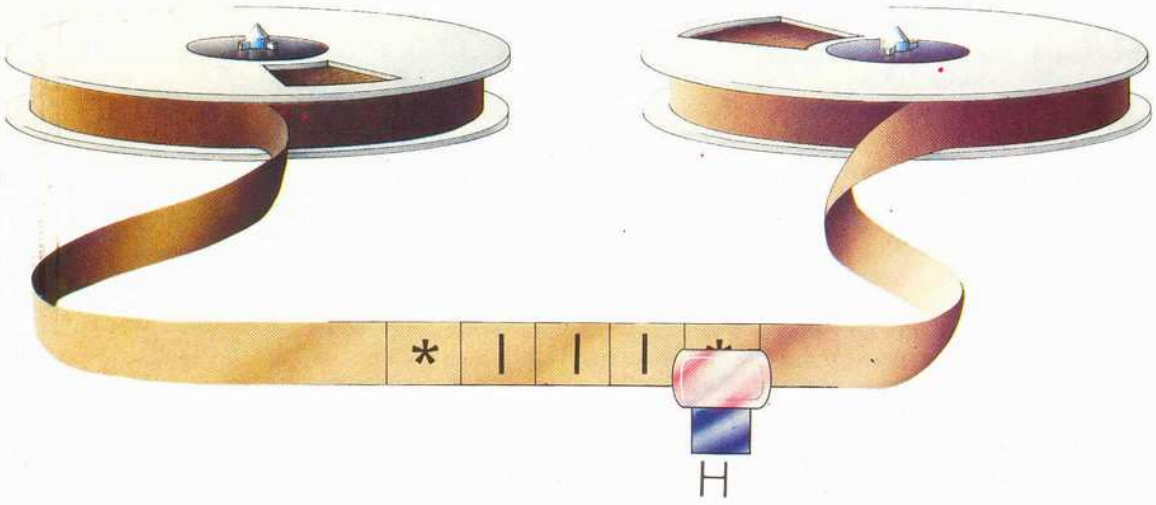
É necessário um total de dez quíntuplos para especificar essa máquina, embora, como você pode constatar no exemplo executado (1 AND 1 = 1), somente cinco são usados para qualquer execução. Se você experimentar a mesma máquina para, digamos, 0 AND 1, verá que um conjunto diferente de quíntuplos será selecionado entre os dez.

Máquina de Turing

Ilustração da máquina de Turing para desempenhar a função AND. Os dois bits de input são colocados em quadrados adjacentes, seguidos de um ponto de interrogação que será substituído pelo resultado. Dois asteriscos são postos nas extremidades do segmento de informação para agir como delimitadores. Os dez quíntuplos abaixo especificam a operação da máquina, embora, para cada exemplo executado (neste caso, 1 AND 1), apenas cinco dos dez sejam usados.

S_A	*	*	S_A	R
S_A	0	0	S_B	R
S_A	1	1	S_C	R
S_B	0	0	S_D	R
S_B	1	1	S_D	R
S_C	0	0	S_D	R
S_C	1	1	S_E	R
S_D	?	0	S_F	R
S_E	?	1	S_F	R
S_F	*	*	H	R





Novilíngua

O mundo dos computadores criou um jargão próprio, uma linguagem imaginativa constituída pelas "buzzwords".

Toda atividade humana possui seu jargão próprio (palavras, códigos e expressões que são utilizados sobretudo por pessoas relacionadas com a área). O pessoal ligado à computação, por sua vez, emprega até mesmo um neologismo inglês para designar os termos novos, chamando-os de "buzzwords" ("palavras zumbidoras" ou "computês").

BUZZWORDS

O termo **BUZZWORD** surgiu pela primeira vez na década de 60, quando o departamento de publicidade da Honeywell lançou um passatempo chamado "gerador de buzzwords". A base do jogo era constituída por três colunas de dez palavras cada, numeradas de 0 a 9. Duas colunas continham substantivos e a terceira, adjetivos. O usuário devia escolher aleatoriamente um número de três algarismos, procurar as palavras correspondentes e obter uma expressão sem significado, como "módulo situacional interativo". Essas expressões poderiam depois ser usadas em conversas com colegas, deixando-os impressionados.

BOOT

BOOT provém da contração de bootstrap: como na expressão "to pull oneself up by one's bootstraps" (vencer por seu próprio esforço). O carregador bootstrap ou autocarregador constitui uma rotina que se processa automaticamente toda vez que o computador é ligado. Em máquinas que não têm um sistema operacional em ROM, a rotina boot deve conter instruções que chamem o sistema a partir do disco.

BOMBAS LÓGICAS
CAVALO DE TRÓIA

Os crimes de computação constituem solo particularmente fértil para a produção de buzzwords. **BOMBAS LÓGICAS** e **CAVALO DE TRÓIA** são dois métodos empregados com finalidades criminosas. O primeiro descreve um conjunto de códigos, desenvolvido num programa aplicativo que permanece inativo até que seja processado um número suficiente de vezes para que o crime (por exemplo, a transferência de dinheiro de uma conta para outra) passe despercebido. O Cavalo de Tróia, a julgar pelo nome, constitui um programa disfarçado em outro, de modo a permitir acesso ao sistema.

TURNKEY

Esse termo designa uma modalidade de contrato. Muitas organizações comerciais e empresas de consultoria de computação que instalam hardware e software possibilitam ao cliente assumir o controle do equipamento já preparado para funcionamento. É a operação **TURNKEY**, na qual tudo o que o cliente tem a fazer é girar a chave — "turn the key" — e dar a partida.

BASIC

Essa buzzword significa "Beginners All-purpose Symbolic Instruction Code" (Código de Instruções Simbólicas de Uso Geral para Principiantes).

HARDWARE
SOFTWARE

HARDWARE e **SOFTWARE** são buzzwords: *hard* significa "duro" e é empregado como "tangível", e *soft*, o oposto. Há mais dois outros tipos de "ware": **FIRMWARE** significa que o software está protegido por hardware (como na ROM e na EPROM) e **PEOPLEWARE** refere-se a todas as pessoas que trabalham com computadores.

BAUD

A velocidade em que os dados são transmitidos recebeu o nome de **BAUD** em homenagem a Émile Baudot, inventor de um código telegráfico, que rivalizou com o de Samuel Morse.

BYTE

Até o aparecimento do microprocessador de 8 bits, 1 byte correspondia a bits suficientes para codificar um caractere individual — ora 6, ora 8. Nessa época, os computadores raramente utilizavam palavras com menos de 24 bits; e alguns equipamentos, sobretudo os projetados com fins científicos, chegavam a 64 bits. A grafia excepcional de byte (próxima da do termo inglês *bite*, "mordida") levou à criação do termo **NYBBLE** (que lembra a grafia do termo inglês *nibble*, "mordiscada"). Nybble corresponde à metade de 1 byte.

BIT

Embora a maioria dos dicionários informe que **BIT** corresponde a uma contração de BInary digiT — dígito binário —, é possível que essa buzzword provenha de "pequeno pedaço". Na gíria americana o termo bit corresponde à oitava parte de 1 dólar e é sempre precedido de "dois": "2 bits", por exemplo, correspondem a um quarto de dólar.

Bit freqüentemente aparece como um prefixo. É o caso de *bit-slicing*, termo utilizado para explicar como certos microprocessadores podem ser montados a partir de "unidades menores" com capacidade para 2, 4 ou 8 bits, que resultam em dispositivos com capacidade de até 32 bits. Estudiosos da computação especulam que programas não utilizados por muito tempo desenvolvem a "deterioração bit": geram-se *bugs* maiores e insolúveis.

BOMBA-RELÓGIO

O termo **TIME BOMB (BOMBA-RELÓGIO)** refere-se a uma engenhosa técnica para proteção de software comercial contra pirataria. É um conjunto de códigos no interior do pacote que seria desativado quando o sistema fosse instalado por um vendedor autorizado. Na cópia pirateada, contudo, a Bomba-Relógio espera algum tempo — em geral, até que o usuário esteja bastante dependente do pacote. No dia seguinte à "explosão", os arquivos do usuário estarão inutilizados e a cópia do programa, destruída (a menos que o disco tenha sido protegido contra gravação superposta).

GARBAGE

É um termo ("sucata", em inglês) que aparece em várias expressões de glossários de jargões para usuários de computador. A sigla **GIGO**, por exemplo, representa "Garbage In, Garbage Out" ("Sucata Entrando, Sucata Saindo"). Isso serve para lembrar que computadores apenas processam dados e, desse modo, você não pode esperar resultados confiáveis se não fornecer dados corretos.

GARBAGE COLLECTION

É o nome dado a um processo interno que pode ser utilizado em seu microcomputador, se este usar uma versão do BASIC que admita variáveis alfanuméricas dinâmicas (isto é, que podem mudar de comprimento durante o programa). Cada vez que uma dessas variáveis aumenta de comprimento, faz-se nova cópia completa em RAM. Assim, se houver muitas instruções na forma `LET A$=A$+"*"` (sobretudo no interior de loops), não demorará muito tempo para que a memória seja completamente preenchida. Nesse ponto, a execução do programa se interrompe por algum tempo e uma rotina em ROM, denominada "coletor de sucata", limpa a área das variáveis alfanuméricas e elimina todas as partes dessas variáveis que foram deixadas de lado em manipulações anteriores. Embora o programa passe a ser muito mais eficiente após a "limpeza" do coletor de sucata, o processo pode demorar vários segundos ou até minutos, durante os quais o computador interrompe o processamento do programa.

HANDSHAKE

Muitas buzzwords originam-se de analogias. Quando se chega a um acordo comercial, os participantes de fato podem dar as mãos: assim, em termos de computação, **HANDSHAKE** ("aperto de mão") é o nome dado ao sinal eletrônico que significa que uma troca de dados se completou.

0. Integrado/a	0. de Banco de Dados	0. Rede
1. Interativo/a	1. Situacional	1. Capacidade
2. de Memória	2. Randômico/a	2. Sistema
3. Digitado/a	3. de Diagnóstico	3. Algoritmo
4. Estocástico/a	4. de Endereçamento	4. Processador
5. Periférico/a	5. Linear	5. Matriz
6. Heurístico/a	6. Gráfico/a	6. Módulo
7. Relacional	7. Alfanumérico/a	7. Recurso
8. Homeostático/a	8. de Imagem	8. Hierarquia
9. Programável	9. Esquemático/a	9. Gerador

Capacidade randômica periférica

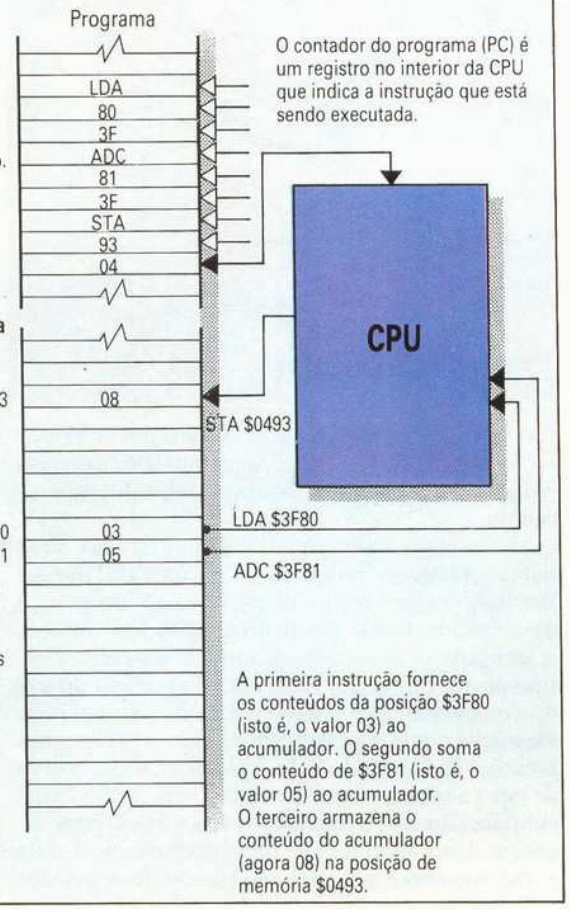
Você pode criar seu gerador de buzzwords com três colunas de dez palavras cada, como fizemos aqui. A escolha de um número aleatório de três dígitos origina uma expressão aparentemente tecnológica, impressionante e sem sentido, como a do título.

Código de máquina

A passagem do BASIC para o código de máquina exige considerável salto conceitual que possibilita, porém, maior velocidade e eficiência.

Um passo por vez

O programa em linguagem de máquina é armazenado numa área da memória, embora os dados que ele opera possam estar em alguma outra posição. Observe que os operandos (por exemplo, $3F80) estão armazenados em 2 bytes, com o byte menor ($80) situado antes do maior ($3F).



Todos os programas em código de máquina são constituídos por operações simples que transportam bytes de memória para os registros internos da CPU, realizam seu processamento e, a seguir, os remetem de novo a uma posição da memória. Esse diagrama mostra o programa necessário para somar os conteúdos de duas posições e armazenar o resultado numa terceira.

Até este ponto, todo o nosso trabalho de programação centrou-se na linguagem BASIC, versátil e de uso fácil. No entanto, à medida que sua experiência aumenta e os projetos de programação com que você lida se tornam mais audaciosos, evidenciam-se as limitações dessa linguagem. Você descobrirá que os gráficos não podem ser deslocados por toda a tela tão depressa quanto deseja e que é necessário apelar com freqüência para os complexos comandos PEEK e POKE a fim de aproveitar melhor os recursos de sua máquina.

Em contrapartida, a programação em código de máquina impõe pouquíssimas restrições e, em comparação com o BASIC, dá impressão de velocidade enorme. Mas poucos usuários de microcomputador optam por esse recurso, em parte porque exige um processo de programação muito laborioso e também por ser conceitualmente diferente do BASIC ou de outra linguagem de alto nível. Todavia, não se pode deixar de ter uma idéia do funcionamento do código de máquina e neste artigo, o primeiro de uma série de dois, examinaremos os procedimentos fundamentais necessários a sua utilização.

O código de máquina, conforme já vimos antes, é a linguagem entendida pelo microprocessador (a CPU), que constitui o cerne do computador e pode apenas executar funções muito simples (adiciona dois dígitos a um número, por exemplo, mas não os multiplica). Faz isso, no entanto, a velocidades muito altas. Cada operação do microprocessador é especificada de acordo com o número de "ciclos de relógio" empregados. Se a CPU em seu computador funcionar a 1 MHz, o ciclo de relógio será de 1 microssegundo, e uma operação que requer quatro "ciclos de relógio" será realizada em 4 milionésimos de segundo.

Como conseqüência, um programa desenvolvido em código de máquina vai requerer grande quantidade de instruções e qualquer função deverá ser elaborada "a mão", a partir de operações simples. Toda a programação consistirá na manipulação de bits ou bytes isolados de memória, empregando-se funções lógicas simples como AND, OR e NOT, além de aritmética binária elementar.

Esse é um dos motivos por que o desenvolvimento de programas nessa linguagem se torna uma tarefa lenta; o outro reside no fato de o programador ser obrigado a saber a localização de tudo que está armazenado na memória. Em BASIC, sempre que se encontra uma instrução como LET A = 5, é função do interpretador achar um espaço na memória para armazenar a variável. Além disso, sempre que A for referida de novo no programa o interpretador recordará o local onde devem ser procurados os dados necessários. Ao iniciar a programação em código de máquina, você descobre que tem de especificar um endereço (uma posição de memória) para cada conjunto de dados a armazenar. E cabe também a você garantir que não haja superposição acidental com outros conjuntos de dados.

Examinemos em que consiste o código de máquina. (Todos os exemplos serão relacionados a CPUs de 8 bits de capacidade, como a do Z80 e do 6502; dispositivos de 16 bits operam de modo semelhante, mas processam o dobro de bits em cada operação.) O microprocessador conecta-se à memória do computador por meio de dois busses (vias): o bus de endereços e o de dados (ver p. 144). Há também um elemento denominado bus de controle que fornece apenas sinais de cronometragem à CPU e não é utilizado pelo programador.

O bus de endereços tem capacidade de 16 bits e a atribuição de um padrão de bits a esse bus possibilita à CPU selecionar qualquer dos 65.536 bytes em seu "mapa de memória" (ver p. 329). Em microcomputadores comuns, algumas dessas posições estarão na RAM, outras na ROM, algumas nos chips especiais de entrada/saída e ainda existirão as que não serão utilizadas. Se a CPU quiser ler determinada posição de memória (uma das linhas no bus de controle indica se deve haver leitura ou registro), o byte sele-

cionado coloca seus conteúdos no bus de dados, na forma de um padrão de 8 bits. De modo semelhante, a CPU pode registrar um padrão de 8 bits em qualquer posição escolhida. A CPU não sabe quais as partes da memória em que estão a ROM e a RAM; desse modo, determinar o endereço correto é outra responsabilidade importante do programador.

No interior do microprocessador, há talvez meia dúzia de "registros", semelhantes a posições individuais de memória, utilizados para o armazenamento de resultados temporários e execução de funções aritméticas lógicas e binárias. A maior parte desses registros corresponde a 1 byte de memória, embora alguns tenham 16 bytes de capacidade. Um dos tipos mais recentes de registro é o chamado contador do programa (PC, Program Counter), que possui na memória o endereço da instrução em código de máquina em execução. Você pode imaginá-lo como semelhante ao número de linha num programa em BASIC.

Outro registro muito importante (mas, agora, com apenas 8 bits de extensão) é o "acumulador" Como o nome indica, esse registro acumula totais (isto é, bytes que podem se somar ou subtrair). Na verdade, esse é, em geral, o único registro que pode executar qualquer tipo de procedimento aritmético. Assim, um código de máquina muito simples seria especificado da seguinte forma:

1) Carregar o acumulador com os conteúdos da posição de memória $3F80. Os endereços em código de máquina são quase sempre desenvolvidos em hexadecimais (ver p. 179), indicados pela anteposição de um sinal especial, quase sempre o $.

2) Acrescentar ao acumulador os conteúdos da posição de memória $3F81, levando em conta que o resultado pode ser maior do que a capacidade de armazenamento num único byte — neste caso também haverá um "bit de transporte".

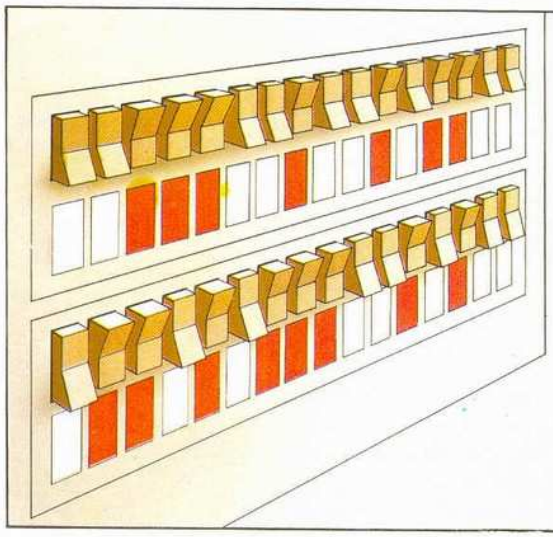
3) Armazenar os novos conteúdos do acumulador (isto é, o resultado) na posição de memória $0493.

Cada um desses procedimentos representa uma instrução em código de máquina e o programa normalmente seria desenvolvido da seguinte forma:

```
LDA $3F80 (LoaD Accumulator, carregar o
          acumulador)
ADC $3F81 (ADd with Carry, somar com transporte)
STA $0493 (STore Accumulator, armazenar o
          acumulador)
```

Os comentários entre parênteses, da mesma forma que as instruções REM do BASIC, não resultam em operações. A primeira entrada em cada linha é denominada "opcode" (código de operação), e indica o tipo de operação. A segunda coluna apresenta o "operand" (operando), ou seja, os detalhes ou as posições dos dados que serão operados. O microprocessador costuma apresentar várias dezenas de códigos de operação (isto é, pode realizar várias dezenas de tipos de operações simples) e cada código de operação fornecido à máquina ocupa apenas 1 byte de memória.

O código de operação pode, desse modo, ser especificado por um número entre 0 e 255 (ou, mais adequadamente, na base hexadecimal de $00 a $FF). Todavia, enquanto o programa é desenvolvido, costuma-se tornar a listagem mais legível pelo emprego de três letras mnemônicas, como LDA, ADC, STA.



Luzes cintilantes
A imagem de grandes painéis luminosos vistos nos computadores dos filmes veio do "painel frontal" existente em muitos minis. Esse painel consistia numa linha de luzes e interruptores representando os busses de endereços e de dados da CPU. Antes da introdução dos teclados todos os programas em código de máquina tinham de ser fornecidos sob a forma binária por esse procedimento.

Cada operando apresentado consiste num número hexadecimal de $0000 a $FFFF e utiliza até 2 bytes da memória do programa. No entanto, alguns operandos têm apenas 1 byte de comprimento e alguns códigos de operação não possuem quaisquer operandos. O pequeno programa que apresentamos deve ocupar um total de apenas 9 bytes, sem incluir as três posições de memória ($3F80, $3F81 e $0493), com as quais o operará. Para esse exercício simples, o seguinte programa em BASIC chega ao mesmo resultado, mas ocupa quase 50 bytes e realiza a operação pelo menos cem vezes mais lentamente, devido ao tempo gasto pelo interpretador para traduzi-la:

```
10 A = PEEK (16256)
20 A = A + PEEK (16257)
30 POKE 1171,A
```

Observação: As posições utilizadas neste programa podem não ser adequadas a sua máquina.

No próximo fascículo examinaremos o procedimento para fornecer o código de máquina ao microcomputador e processá-lo, bem como os diferentes modos de apresentar a linguagem.

LDA
LoaD Accumulator (carregar o acumulador)
Transfere o conteúdo de uma posição isolada de memória (byte) para o registro do acumulador interno.

STA
STore Accumulator (armazenar o acumulador)
Executa o procedimento oposto ao do LDA.

ADC
ADd with Carry (soma com transporte)
Soma os conteúdos de uma posição de memória aos que estão no acumulador, criando 1 bit de transporte.

SBC
SuBtract with Carry (subtração com transporte)
Realiza função inversa à do ADC.

JMP
JuMP (salto)
Transfere a operação do programa para uma nova posição. É semelhante, quanto ao modo de operar, ao GOTO do BASIC.

Códigos de operação
Estes são apenas alguns dos opcodes (códigos de operação ou de instrução) que um microprocessador comum pode executar.

Linha de montagem

Diversos são os modos pelos quais os programas podem ser expressos. Os códigos de máquina incluem desde a linguagem binária até a Assembly.

Uma das dificuldades conceituais que os principiantes experimentam com relação à linguagem de máquina está nas várias formas pelas quais os programas podem se apresentar. Qualquer dado armazenado na memória do computador assume, em última análise, a forma de números binários de 8 bits. Contudo, ao serem relacionados, esses dados ocupam muito espaço, tornam-se difíceis de ler e de memorizar, e induzem a erros de digitação. Para evitar esses inconvenientes, empregamos quase sempre os números hexadecimais (hex), que permitem a representação dos conteúdos de qualquer byte como número de dois dígitos. Os hex possibilitam ainda que os endereços no limite da memória do computador (0 a 65535, em decimal) sejam representados por quatro dígitos.

Em geral, números hex escritos no papel são precedidos por um sinal $, para distingui-los de números decimais, embora esse sinal não fique retido na memória do computador. Além disso, quando um código de operação possui um operando de 2 bytes (por exemplo, LDA $3F80), eles são fornecidos ao equipamento em ordem inversa — ao último byte segue-se o primeiro. Assim, no exemplo apresentado, os 3 bytes seriam AD (que corresponde à representação do código de operação LDA na linguagem 6502) seguido por 80 e depois por 3F. Esse procedimento facilita as operações do processador, mas pode confundir o usuário.

O programa em código de máquina é quase sempre impresso como um "despejo hex" — longa lista de valores hexadecimais de dois dígitos. Além disso, obtém-se um endereço de início (em hex ou em decimal). O primeiro valor hex é carregado nessa posição; o segundo, na seguinte; e assim por diante. Obtém-se o carregamento por meio do comando POKE da linguagem BASIC. Se o endereço de início for $1000 (4096, em decimal) e o despejo hex for

AD (173, em decimal)
80 (128, em decimal)
3F (63, em decimal)

o programa poderá ser carregado com três instruções em BASIC:

```
POKE 4096,173
POKE 4097,128
POKE 4098,63
```

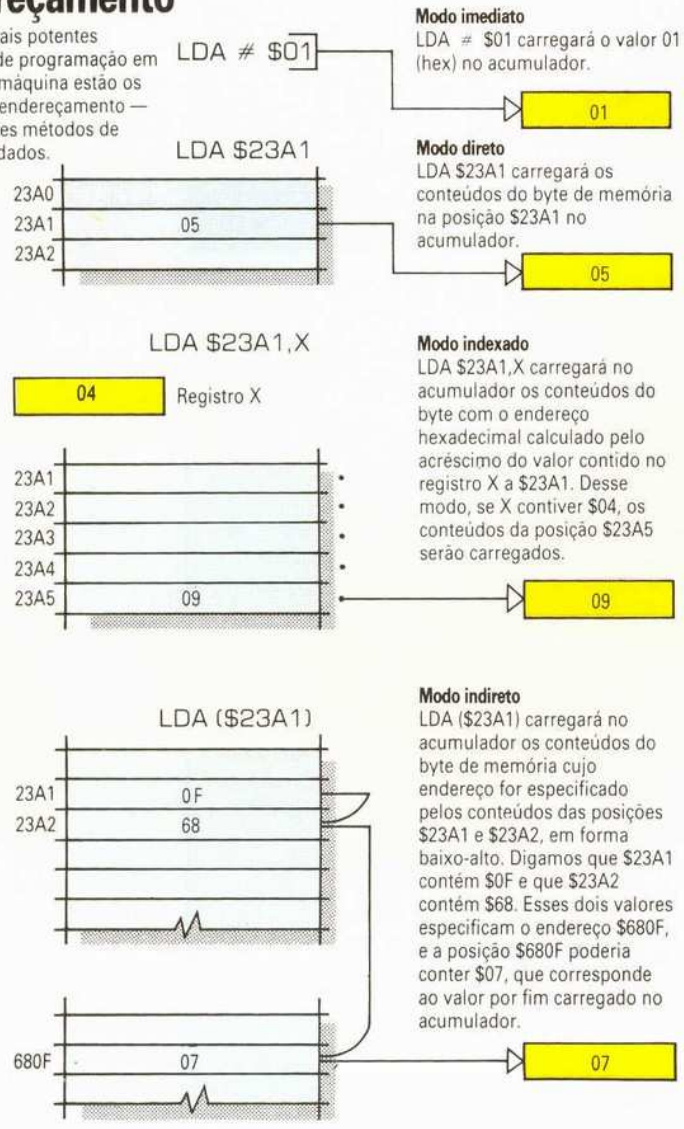
Observe que temos de converter todos os valores de hex para decimal, antes de utilizá-los na instrução POKE. No interior do equipamento eles serão armazenados em código binário.

Para despejos mais longos é freqüente o emprego de um pequeno programa em BASIC denominado "carregador de código de máquina". Esse procedimento requer o endereço de início e, em seguida, os valores hex. À medida que é fornecida, a pequena rotina em BASIC converte o valor hex em decimal e armazena-o por meio do comando POKE na posição seguinte. Como alternativa, o despejo hex pode ser lido pelo programa a partir das instruções DATA.

Uma vez fornecido o código de máquina, pode-se abandonar o programa carregador em BASIC. Con-

Modos de endereçamento

Entre os mais potentes conceitos de programação em código de máquina estão os modos de endereçamento — os diferentes métodos de recuperar dados.



Por extenso/ abreviado

O programa em código de máquina pode assumir diferentes formas. Em geral, desenvolve-se pelo programador como linguagem Assembly, que emprega os sinais mnemônicos para os códigos de operação e títulos para os operandos, assim:

```
LDA PESO
ADC COMBUSTÍVEL
STA PESO
```

Devemos, contudo, especificar os endereços desses títulos. Por exemplo:

```
COMBUSTIVEL = $03EE
PESO = $031F
```

O pacote assembler transformará isso num despejo hex, pelo emprego de unidade de discos. A "pseudolinguagem assembly", usada abaixo, é de leitura mais difícil, mas com freqüência pode ser fornecida a um pacote denominado "spot assembler", que não necessita de discos.

```
LDA $031F
ADC $03EE
STA $031F
```

O despejo hex consiste num endereço de início (à esquerda) e na seqüência de valores hex de dois dígitos como aparecerão na memória. Observe que um operando como $031F é armazenado em ordem inversa (1F03) e que o valor hex correspondente substitui os códigos de operação:

```
19C4 AD 1F 03 6D EE 03 8D 1F 03
```

vém carregar o código de máquina num ponto da memória em que ele não seja "atropelado" pelo programa em BASIC, nem eliminado por instruções dessa linguagem, como NEW.

A maioria dos microcomputadores possui comando BASIC indicativo de quando se deve parar de operar nessa linguagem e começar a executar o programa em código de máquina. Uma forma desse comando se apresenta como SYS 4096 (RETURN), que corresponde a "transferir o controle ao sistema a partir da posição decimal 4096". CALL $E651 significa "chamar a rotina de código de máquina a partir da posição hex E651".

A sub-rotina ou programa em código de máquina executa esse sistema ou rotina (que pode ou não apresentar resultados visíveis, dependendo do tipo de programa). Se estiver corretamente desenvolvido e incorporar o procedimento apropriado de encerramento, o controle será transferido de novo para a linguagem BASIC. Incidentalmente, isso implica a possibilidade de chamar sub-rotinas de código de máquina a partir de vários pontos na operação do programa em BASIC, sempre que se fizer necessária a execução de função em alta velocidade.

Uma das dificuldades da programação em linguagem de máquina está em que, se você errar, o computador não responderá com uma simpática e útil mensagem tipo SYNTAX ERROR. A probabilidade maior é que o programa "engasgue": o equipamento deixa de responder às instruções que você digitar. Isso não danifica o computador, mas você terá de zerá-lo (ou desligá-lo e depois ligar outra vez). E isso, em geral, implica fornecer novo programa desde o princípio. Você não pode, portanto, experimentar em código de máquina como faz em BASIC — a operação do programa deve ser conferida por completo em papel antes de ser fornecida ao computador.

O "monitor do código de máquina" (que não tem nenhuma relação com a tela do monitor) auxilia bastante o fornecimento e a conferência do código de máquina. Ele está incorporado na ROM de alguns computadores, mas o mais comum é sua aquisição em pacotes tipo cartucho. O monitor do código de máquina constitui um sistema operacional simples, que apresenta na tela os conteúdos de qualquer setor da memória. Esses valores (hex) podem ser alterados ou substituídos por sobreposição; desse modo, o monitor é a melhor forma de requisitar um despejo hex. Além disso, quase sempre possibilita o carregamento e a gravação em cassete de programas em código de máquina, sem necessidade do programa carregador em BASIC. Os programas de recursos de código de máquina mais desenvolvidos — o código de máquina correspondente aos pacotes ferramentas em BASIC (ver p. 444) — mostram os conteúdos de cada registro interno do processador.

Os despejos hex constituem eficiente modo de representação do código de máquina, mas não são fáceis de ser lidos. A menos que você memorize os correspondentes hexadecimais de vários códigos de operação, será quase impossível distingui-los dos operandos. Assim, desenvolve-se a maior parte dos programas com o auxílio de sinais mnemônicos de três letras, que apresentamos na página 449. Esses auxiliares da memória são então traduzidos para hex com o emprego de códigos do manual do microprocessador.

Uma forma mais sofisticada de monitor de código de máquina — o "spot assembler" — permite a digitação do programa em sinais mnemônicos, executando automaticamente as conversões.

Isso nos leva à última forma pela qual o código de máquina pode ser representado — a linguagem Assembly, que não só utiliza os sinais mnemônicos para códigos de operação como também manipula nomes (ou títulos), em vez de números hex para os operandos. Desse modo, se a posição $07B2 contiver o número de mísseis disparados num jogo, podemos carregá-lo no acumulador com a instrução:

```
LDA MISSIL
```

No começo do programa temos de especificar a posição de MISSIL = $07B2 e que esta deve, de início, conter o valor $09 (nove mísseis).

Depois de desenvolver esse programa em linguagem Assembly (o "código fonte" do programa) processamos um programa utilitário denominado assembler (montador), que opera todo o código, substituindo os sinais mnemônicos e os títulos por seus correspondentes hexadecimais. Cria, desse modo, uma nova versão, o "código objeto". Este pode então ser fornecido à memória do computador e executado. O processo não é muito diferente da compilação (ver p. 84), embora nesse caso haja correspondência unívoca entre o código fonte e o código objeto.

Por constituir um nível de linguagem mais alto que o código de máquina, a linguagem Assembly apresenta maior operacionalidade sem perda de qualidade no desempenho. Contudo, os pacotes assembler em geral só funcionam com unidades de disco e, desse modo, não podem ser utilizados por todos os computadores.

Códigos

Abaixo, alguns códigos de operação que se encontram em microprocessadores comuns.

JSR

Jump SubRoutine
(sub-rotina de desvio)
Equivale à função GOSUB do BASIC. JSR $354D alterará os conteúdos do contador do registro do programa (PC) para fazê-lo executar o código a partir de $354D.

RTS

ReTurn from Subroutine
(retorno da sub-rotina) Ao encontrar RTS, o processador voltará à posição a partir da qual a sub-rotina foi chamada (isto corresponde a RETURN em BASIC). RTS não possui operando porque o endereço de retorno terá sido automaticamente armazenado numa área especial da memória chamada Stack.

BMI

Branch if MInus
(desviar, se negativo) É uma das várias formas de desvio condicional em código de máquina (em BASIC, IF-THEN GOTO é um desvio condicional). Se o resultado da última operação for um valor negativo no acumulador, a execução do programa saltará para um endereço especificado. BPL especifica Branch if PLus (desviar, se positivo).

LDX

LoaD X register
(carregar o registro X) X é outro registro de 1 byte no interior do processador; não executa procedimentos aritméticos do mesmo modo que o acumulador, mas é utilizado para "endereçamento indexado" (ver quadro da p. 464). LDX carrega um valor em X e STX (STore X, armazenar X) o armazena de novo na memória.

INX

INcrement X
(aumentar X) Pelo acréscimo de uma unidade ao valor de X (DEX — DEcrement X, diminui X) e pela utilização do endereçamento indexado, pode-se passar gradativamente por uma série de posições na memória, executando o mesmo procedimento em cada uma.

As próximas gerações

Com a introdução da tecnologia de VLSI, estamos a ponto de entrar na quarta geração de computadores. Mas os japoneses já se preparam para a quinta.

Homens velhos não fazem revoluções, segundo o dito popular, e o diretor do projeto japonês para a criação da quinta geração de computadores parece ter levado isso ao pé da letra. Ao escolher quarenta cientistas nas dez maiores empresas privadas e nos laboratórios do governo para trabalharem com ele no Instituto de Tecnologia para Computadores de Nova Geração, em Tóquio, o dr. Kazuhiro Fuchi selecionou apenas colaboradores com idade inferior a 35 anos. O instituto, fundado em 14 de abril de 1982, com uma verba de 600 milhões de dólares (a ser despendida em dez anos), é uma *joint-venture* entre o governo e as indústrias. Empresas como a Fujitsu, Sharp e Toshiba participam desse ambicioso projeto, que pretende ultrapassar o estágio atual da tecnologia de computadores e criar máquinas muito mais avançadas.

A própria cunhagem do termo "quinta geração" tem sua origem nos progressos conseguidos no passado, levando em conta as futuras possibilidades de criação. A primeira geração de computadores caracterizou-se pelo uso de válvulas termiônicas, que se tornaram obsoletas após a invenção do transistor. Os computadores de segunda geração (os de transistores simples) foram, por sua vez, superados pelas máquinas que usavam tecnologia de Integração em Larga Escala (Large Scale Integration, LSI) e possibilitavam a montagem de muitos componentes dentro de um único chip. Atualmente (meados da década de 80), estamos no fim dessa terceira geração e até a década de 90 deveremos chegar à quarta de chips do tipo VLSI (Very Large Scale Integrated), que conterão 10 milhões de transistores por chip, ao passo que hoje o limite não passa de 250.000.

Comparado à IBM — International Business Machines, que gasta por ano quantia superior a 2 bilhões de dólares em pesquisa e desenvolvimento, o investimento japonês parece insignificante. Contudo, o desembolso de capital japonês não visa só ao lucro.

Sociedade pós-industrial

O interesse da ciência mudou nos últimos cem anos, passando da utilização de energia em forma bruta (com a eletricidade e o motor de combustão interna) ao estudo da riqueza mais intangível — a informação. Terras, mão-de-obra, capital e indústria talvez tenham sido fontes de poder no passado, mas o futuro favorecerá aqueles que controlarem a informação. Conhecimento e processamento de informações serão as chaves da sociedade pós-industrial. Assim, essa nova sociedade precisará de um mecanismo com raciocínio automático aplicável a qualquer problema real ou área da atividade humana, com a precisão matemática de um computador. O mecanismo que está sendo construído pelos japoneses chama-se Knowledge and Information Processing System — KIPS (Sistema de Processamento de Informação e Conhecimento).

Os seres humanos têm muita habilidade para converter sinais sensoriais em formas cognitivas — por exemplo, percebem de relance a situação de uma partida de xadrez —, mas, quando se trata de tomar decisões que dependem de grande volume de dados, as limitações logo se evidenciam. As regras do xadrez são explicáveis em poucos minutos; contudo, o jogo pode se mostrar tão complexo que os grandes mestres só conseguem ver até uma dúzia de lances adiante. No entanto, em princípio, todo problema ao qual se aplica o raciocínio divide-se numa série de etapas simples, que por sua vez se resolvem mediante regras de inferência. Esse conjunto de regras, conhecido como lógica predicativa, aplica-se a todos os problemas, mas nas decisões cotidianas simples não temos consciência do processo.

Linguagem lógica

O PROLOG — uma abreviação de Programming Logic (Lógica de Programação) — foi desenvolvido no começo da década de 70 pelo grupo de Inteligência Artificial da Universidade de Marselha, na França. Com base em alguns dos princípios da lógica humana, essa linguagem provavelmente será usada nos computadores de quinta geração. Ela facilita o trabalho de criação e consulta de bancos de dados além de adequar-se a aplicações educacionais.

Para um especialista não basta um bom cérebro — no caso de um médico, são necessários muitos anos de experiência para acumular conhecimentos. Da mesma forma, um KIPS deve dispor de um banco de dados no qual as regras de inferência possam operar. Além do mais, o sistema precisa ser extremamente "user-friendly" (de fácil uso), para que sua operação não requeira uma equipe de especialistas. Uma máquina KIPS com a qual você possa manter uma conversa na linguagem de sua escolha deve resultar de pesquisas no campo da inteligência artificial — área de estudos bem controvertida.

As metas estabelecidas pelos japoneses englobam grande variedade de tecnologias relacionadas ao computador: hardware, software, interfaces, sistemas especializados (ver p. 72) e os problemas da inteligência artificial.

O objetivo do projeto japonês vai além dos avanços da tecnologia dos chips. Com o aumento da densidade dos transistores em circuitos integrados, os elétrons têm menos distância para percorrer entre os componentes e, portanto, os circuitos operam com

maior rapidez. Todavia, os japoneses sabem que só velocidade não basta, e esta é a razão de despenderem tanto esforço em software. Num jogo de xadrez, digamos, há tantas seqüências possíveis de movimentos (aproximadamente 10^{120}) que o tempo necessário para explorá-las excede o período de vida que resta ao Sol. O projeto tem por objetivo a produção de uma máquina que faça 100 milhões de inferências lógicas por segundo, ou seja, 100 milhões de LIPS (Logical Inferences Per Second).

Outro meio de aumentar a velocidade seria colocar algumas funções de software no design de um chip, em vez de carregá-las na memória e processá-las através de um chip de uso geral. Essa mudança na diferença entre hardware e software é um dos aspectos mais interessantes do projeto japonês. Já existem memórias "associativas" que possuem circuitos de busca lógica embutidos nos elementos da memória. Esses dispositivos podem localizar parte dos dados apenas por seu significado, sem haver necessidade de especificar um endereço da memória.

Avanços desse tipo irão acelerar a interação dos processadores lógicos com os bancos de dados. A colocação de rotinas de programas no computador pelo processo de hard-wiring data da época do ENIAC (ver p. 140), mas as máquinas de quinta geração divergirão da arquitetura de Von Neumann num aspecto fundamental — elas vão apresentar muitos processadores diferentes trabalhando ao mesmo tempo (em paralelo), em vez de dispor apenas de uma unidade central de processamento. Isso exige maior cuidado na cronometragem e controle das operações internas, mas elimina a restrição da velocidade que a execução seqüencial de instruções impõe.

A linguagem interna escolhida para o KIPS é a PROLOG, desenvolvida na França e na Inglaterra e baseada na lógica predicativa. O KIPS terá a capacidade de comunicar-se em diversas línguas com os usuários.

Tradução de fala humana é outra meta do projeto, com o objetivo imediato de 95% de precisão. Hoje, a capacidade de reconhecer palavras soltas pronunciadas por diversos locutores perde de longe para o sucesso alcançado pela fala sintetizada. No entanto, a NEC Corporation of Japan já conseguiu criar a máquina que reconhece a voz de um indivíduo. Mas o projeto tem uma limitação: cada palavra deve ser previamente registrada por ele, a fim de que o computador possa "lembrar-se" do padrão da fala e reconhecê-lo.

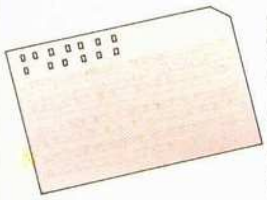
Comunidade de informação

Quanto à palavra escrita, o projeto prevê um programa de dicionários japonês-inglês de 100.000 palavras, com margem de erro de apenas 10%.

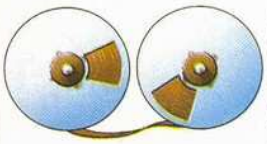
O Japão tem precedentes de sucesso em pesquisas de longo prazo: o projeto PIPS (Pattern Information Processing Systems) dos anos 70 mostrou-se útil no desenvolvimento de bancos de dados visuais e nas interfaces do tipo "user-friendly". Um KIPS precisa ser capaz de olhar para uma imagem e extrair as características e os contornos salientes para fazer uma avaliação preliminar. No metrô de Tóquio já existe uma máquina que faz o seguinte: controla os usuários através de uma câmara e produz um gráfico do fluxo de passageiros.

A tecnologia da informação movimentou, em 1983, 88 bilhões de dólares nos Estados Unidos e, com a tendência de declínio do emprego de mão-de-obra na indústria manufatureira, como na agricultura no começo do século (de 40% do total da força de trabalho para 3% hoje), a comunidade passará a ser uma sociedade de informação. Diante desse fato, o Japão faz de seu projeto de quinta geração algo muito ambicioso. O plano é otimista e inclui uma série de descobertas "programadas" que podem se concretizar ou não. Por exemplo: a descoberta da fissão nuclear controlada.

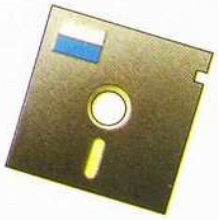
Jogo de gerações



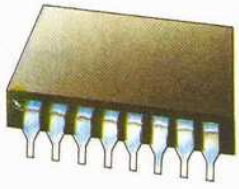
A primeira geração de computadores eletrônicos foi desenvolvida em torno da tecnologia da válvula termiônica. Eles possibilitavam pouca memória on-line, e os dados eram geralmente armazenados em cartões perfurados.



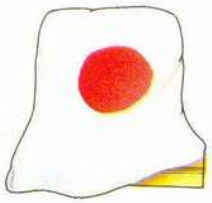
A segunda geração evoluiu a partir do transistor, que aumentou a capacidade de memória, embora o armazenamento off-line (em fita magnética) ainda fosse usado.



Na terceira geração, a invenção do circuito integrado aumentou extraordinariamente a capacidade do computador e foi responsável pelo surgimento do microcomputador — caracterizado pela unidade de disco flexível.



Atualmente, estamos passando da terceira para a quarta geração, com a tecnologia dos chips VLSI. A memória RAM será tão grande que o armazenamento off-line se tornará irrelevante.



A quinta geração de computadores, que está sendo desenvolvida no Japão, relaciona-se mais ao software que ao hardware. No entanto, baseia-se na suposição de que a memória disponível para o usuário será tão grande que o tamanho do programa perderá importância.