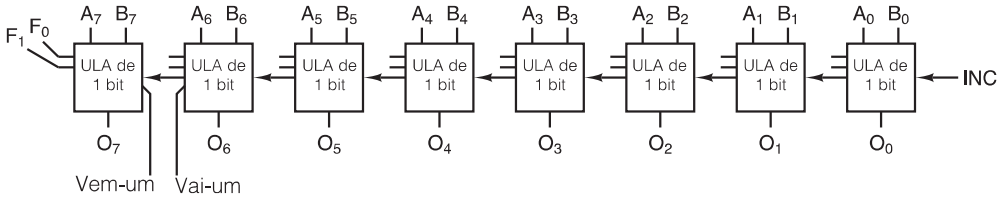
Figura 3.19 Oito segmentos (*slices*) de ULA de 1 bit conectados para formar uma ULA de 8 bits. Os sinais de habilitação e inversão não são mostrados por simplicidade.



3.2.4 *Clocks*

Em muitos circuitos digitais, a ordem em que os eventos ocorrem é crítica. Às vezes um evento deve preceder outro, às vezes dois eventos devem ocorrer simultaneamente. Para permitir que os projetistas consigam as relações de temporização requeridas, muitos circuitos digitais usam *clocks* para prover sincronização. Nesse contexto, um ***clock*** é um circuito que emite uma série de pulsos com uma largura de pulso precisa e intervalos precisos entre pulsos consecutivos. O intervalo de tempo entre as arestas correspondentes de dois pulsos consecutivos é denominado **tempo de ciclo de *clock***. Em geral, as frequências de pulso estão entre 100 MHz e 4 GHz, correspondendo a ciclos de *clock* de 10 nanossegundos a 250 picossegundos. Para conseguir alta precisão, a frequência de *clock* normalmente é controlada por um oscilador de cristal.

Muitos eventos podem ocorrer dentro de um computador durante um único ciclo de *clock*. Se eles devem ocorrer em uma ordem específica, o ciclo de *clock* deve ser dividido em subciclos. Uma maneira comum de prover resolução superior à do *clock* básico é aproveitar a linha de *clock* primária e inserir um circuito com um atraso conhecido, gerando assim um sinal de *clock* secundário deslocado em certa fase em relação ao primeiro, conforme mostra a Figura 3.20(a). O diagrama de temporização da Figura 3.20(b) dá quatro referências de tempo para eventos discretos:

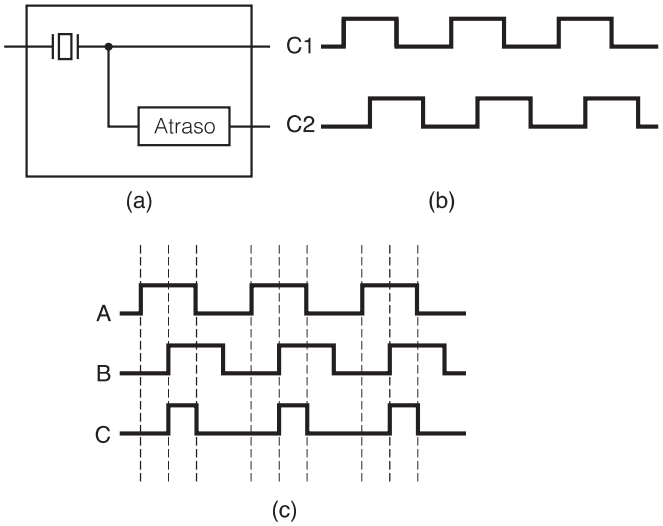
1. Fase ascendente de C1.
2. Fase descendente de C1.
3. Fase ascendente de C2.
4. Fase descendente de C2.

Vinculando diferentes eventos às várias fases, pode-se conseguir a sequência requerida. Se forem necessárias mais do que quatro referências de tempo dentro de um ciclo de *clock*, podem-se puxar mais linhas da linha primária, com diferentes atrasos, se for preciso.

Em alguns circuitos, estamos interessados em intervalos de tempo em vez de instantes discretos de tempo. Por exemplo, pode-se permitir que algum evento aconteça toda vez que C1 estiver alto, em vez de exatamente na fase ascendente. Outro evento só poderá acontecer quando C2 estiver alto. Se forem necessários mais de dois intervalos diferentes, podem ser instaladas mais linhas de *clock* ou pode-se fazer com que os estados altos dos dois *clocks* se sobreponham parcialmente no tempo. No último caso, podem-se distinguir quatro intervalos distintos: $\overline{\text{C1}}$ AND $\overline{\text{C2}}$, $\overline{\text{C1}}$ AND C2, C1 AND $\overline{\text{C2}}$ e C1 AND C2.

A propósito, *clocks* são simétricos, com o tempo gasto no estado alto igual ao tempo gasto no estado baixo, como mostra a Figura 3.20(b). Para gerar um trem de pulsos assimétrico, o *clock* básico é deslocado usando um circuito de atraso e efetuando uma operação AND com o sinal original, como mostra a Figura 3.20(c) como C.

Figura 3.20 (a) Um *clock*. (b) Diagrama de temporização para o *clock*. (c) Geração de um *clock* assimétrico.



3.3 Memória

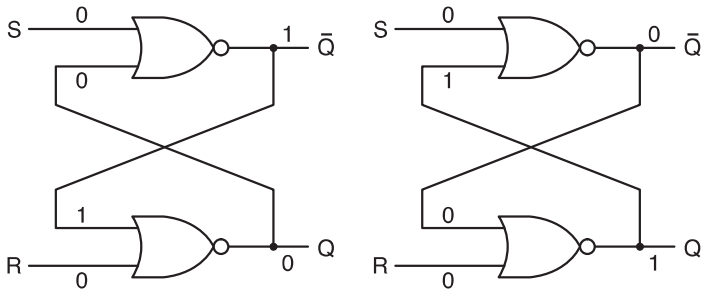
Um componente essencial de todo computador é sua memória. Sem ela não poderiam existir os computadores que conhecemos. A memória é usada para armazenar instruções a serem executadas e dados. Nas seções seguintes examinaremos os componentes básicos de um sistema de memória começando no nível da porta para ver como eles funcionam e como são combinados para produzir memórias de grande porte.

3.3.1 Memórias de 1 bit

Para criar uma memória de 1 bit ("*latch*"), precisamos de um circuito que "se lembre", de algum modo, de valores de entrada anteriores. Tal circuito pode ser construído com base em duas portas NOR, como ilustrado na Figura 3.21(a). Circuitos análogos podem ser construídos com portas NAND, porém, não vamos mais mencioná-los porque são conceitualmente idênticos às versões NOR.

O circuito da Figura 3.21(a) é denominado ***latch* SR**. Ele tem duas entradas, *S*, para ativar (*setting*) o *latch*, e *R*, para restaurá-lo (*resetting*), isto é, liberá-lo. O circuito também tem duas saídas, Q e $\overline{Q}$, que são complementares, como veremos em breve. Ao contrário de um circuito combinacional, as saídas do *latch* não são exclusivamente determinadas pelas entradas atuais.

Figura 3.21 (a) *Latch* NOR no estado 0. (b) *Latch* NOR no estado 1. (c) Tabela verdade para NOR.



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Para ver como isso ocorre, vamos supor que ambos, *S* e *R*, sejam 0, o que é verdade na maior parte do tempo. Apenas para polemizar, vamos supor que Q = 0. Como Q é realimentado para a porta NOR superior, ambas as suas entradas são 0, portanto, sua saída, $\overline{Q}$, é 1. O 1 é realimentado para a porta inferior que, então, tem entradas 1 e 0, resultando em Q = 0. Esse estado é no mínimo coerente e está retratado na Figura 3.21(a).

Agora, vamos imaginar que Q não seja 0, mas 1, com *R* e *S* ainda 0. A porta superior tem entradas de 0 e 1, e uma saída, $\overline{Q}$, de 0, que é realimentada para a porta inferior. Esse estado, mostrado na Figura 3.21(b), também é coerente. Um estado com as duas saídas iguais a 0 é incoerente, porque força ambas as portas a ter dois 0 como entrada, o que, se fosse verdade, produziria 1, não 0, como saída. De modo semelhante, é impossível ter ambas as saídas iguais a 1, porque isso forçaria as entradas a 0 e 1, o que resultaria 0, não 1. Nossa conclusão é simples: para *R* = *S* = 0, o *latch* tem dois estados estáveis, que denominaremos 0 e 1, dependendo de Q.

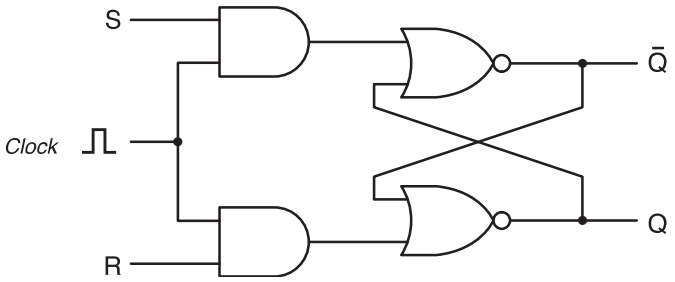
Agora, vamos examinar o efeito das entradas sobre o estado do *latch*. Suponha que *S* se torna 1 enquanto Q = 0. Então, as entradas para a porta superior são 1 e 0, forçando a saída $\overline{Q}$ a 0. Essa mudança faz ambas as entradas para a porta inferior serem 0, forçando a saída para 1. Portanto, ativar *S* (isto é, fazer com que seja 1) muda o estado de 0 para 1. Definir *R* em 1 quando o *latch* está no estado 0 não tem efeito algum porque a saída da porta NOR inferior é 0 para entradas de 10 e entradas de 11.

Usando raciocínio semelhante, é fácil ver que definir *S* em 1 quando em estado Q = 1 não tem efeito algum, mas definir *R* leva o *latch* ao estado Q = 0. Resumindo, quando *S* é definido em 1 momentaneamente, o *latch* acaba no estado Q = 1, pouco importando seu estado anterior. Da mesma maneira, definir *R* em 1 momentaneamente força o *latch* ao estado Q = 0. O circuito "se lembra" se foi *S* ou *R* definido por último. Usando essa propriedade podemos construir memórias de computadores.

Latches SR com *clock*

Muitas vezes é conveniente impedir que o *latch* mude de estado, a não ser em certos momentos especificados. Para atingir esse objetivo, fazemos uma ligeira modificação no circuito básico, conforme mostra a Figura 3.22, para obter um ***latch* SR com *clock***.

Figura 3.22 *Latch* SR com *clock*.



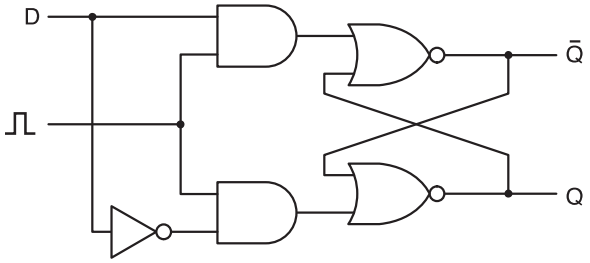
Esse circuito tem uma entrada adicional, o *clock*, que em geral é 0. Com o *clock* em 0, ambas as portas AND geram saída 0, independentemente de ser *S* e *R*, e o *latch* não muda de estado. Quando o *clock* é 1, o efeito das portas AND desaparece e o *latch* se torna sensível a *S* e *R*. Apesar de seu nome, o sinal do *clock* não precisa ser gerado por um *clock*. Os termos ***enable*** e ***strobe*** também são muito usados para indicar que a entrada do *clock* é 1; isto é, o circuito é sensível ao estado de *S* e *R*.

Até aqui evitamos falar no que acontece quando ambos, *S* e *R*, são 1, por uma boa razão: o circuito se torna não determinístico quando ambos, *R* e *S*, finalmente retornam a 0. O único estado coerente para *S* = *R* = 1 é Q = $\overline{Q}$ = 0; porém, assim que ambas as entradas voltam para 0, o *latch* deve saltar para um de seus dois estados estáveis. Se quaisquer das entradas cair para 0 antes da outra, a que permanecer em 1 por mais tempo vence, porque, quando apenas uma entrada for 1, ela força o estado. Se ambas as entradas voltarem a 0 ao mesmo tempo (o que é muito improvável), o *latch* salta aleatoriamente para um de seus estados estáveis.

Latches D com *clock*

Uma boa maneira de resolver a instabilidade do *latch* SR (causada quando $S = R = 1$) é evitar que ela ocorra. A Figura 3.23 apresenta um circuito de *latch* com somente uma entrada, *D*. Como a entrada para a porta AND inferior é sempre o complemento da entrada para a superior, nunca ocorre o problema de ambas as entradas serem 1. Quando $D = 1$ e o *clock* for 1, o *latch* é levado ao estado $Q = 1$. Quando $D = 0$ e o *clock* for 1, ele é levado ao estado $Q = 0$. Em outras palavras, quando o *clock* for 1, o valor corrente de *D* é lido e armazenado no *latch*. Esse circuito, denominado ***latch* D com *clock***, é uma verdadeira memória de 1 bit. O valor armazenado sempre estará disponível em Q. Para carregar o valor atual de *D* na memória, um pulso positivo é colocado na linha do *clock*.

Figura 3.23 *Latch* D com *clock*.



Esse circuito requer 11 transistores. Circuitos mais sofisticados (porém, menos óbvios) podem armazenar 1 bit com até seis transistores. Esses projetos costumam ser usados na prática. Esse circuito pode permanecer estável indefinidamente, desde que seja aplicada energia (não mostrado). Mais adiante, veremos os circuitos de memória que se esquecem rápido do estado em que estão, a menos que, de alguma forma, sejam "relembrados" constantemente.

3.3.2 *Flip-flops*

Em muitos circuitos é necessário ler o valor em determinada linha em dado instante, e armazená-lo. Nessa variante, denominada ***flip-flop***, a transição de estado não ocorre quando o *clock* é 1, mas durante a transição de 0 para 1 (borda ascendente), ou de 1 para 0 (borda descendente). Assim, o comprimento do pulso do *clock* não é importante, contanto que as transições ocorram rapidamente.

Para dar ênfase, vamos repetir qual é a diferença entre um *flip-flop* e um *latch*. Um *flip-flop* é **disparado pela borda**, enquanto um *latch* é **disparado pelo nível**. Contudo, fique atento, porque esses termos são muito confundidos na literatura. Muitos autores usam "*flip-flop*" quando estão se referindo a um *latch*, e vice-versa.

Há várias formas de projetar um *flip-flop*. Por exemplo, se houvesse alguma maneira de gerar um pulso muito curto na borda ascendente do sinal de *clock*, esse pulso poderia ser alimentado para um *latch* D. Na verdade, essa maneira existe, e o circuito para ela é mostrado na Figura 3.24(a).

À primeira vista, poderia parecer que a saída da porta AND seria sempre zero, uma vez que a operação AND de qualquer sinal com seu inverso é zero, mas a situação é um pouco diferente disso. O inversor tem um atraso de propagação pequeno, mas não zero, e é esse atraso que faz o circuito funcionar. Suponha que meçamos a tensão nos quatro pontos de medição *a*, *b*, *c* e *d*. O sinal de entrada, medido em *a*, é um pulso de *clock* longo, como mostrado na parte inferior da Figura 3.24(b). O sinal em *b* é mostrado acima dele. Observe que ele está invertido e também ligeiramente atrasado, quase sempre de alguns nanossegundos, dependendo do tipo de inversor utilizado.

O sinal em *c* também está atrasado, mas apenas pelo tempo correspondente à propagação (à velocidade da luz) do sinal. Se a distância física entre *a* e *c* for, por exemplo, 20 micra, então o atraso de propagação é 0,0001 ns, que decerto é desprezível em comparação com o tempo que o sinal leva para se propagar pelo inversor. Assim, para todos os efeitos e propósitos, o sinal em *c* é praticamente idêntico ao sinal em *a*.

Quando se efetua uma operação AND com as entradas para a porta AND, *b* e *c*, o resultado é um pulso curto, como mostra a Figura 3.24(b), onde a largura do pulso, Δ, é igual ao atraso da porta do inversor, em geral 5 ns ou menos. A saída da porta AND é exatamente esse pulso deslocado pelo atraso da porta AND, como mostrado na parte superior da Figura 3.24(b). Esse deslocamento de tempo significa apenas que o *latch* D será ativado com um atraso fixo após a fase ascendente do *clock*, mas não tem efeito sobre a largura do pulso. Em uma memória com tempo de ciclo de 10 ns, um pulso de 1 ns para informar quando ler a linha *D* pode ser curto o bastante, caso em que o circuito completo pode ser o da Figura 3.25. Vale a pena observar que esse projeto de *flip-flop* é atraente porque é fácil de entender, embora, na prática, sejam usados *flip-flops* mais sofisticados.

Figura 3.24 (a) Gerador de pulso. (b) Temporização em quatro pontos do circuito.

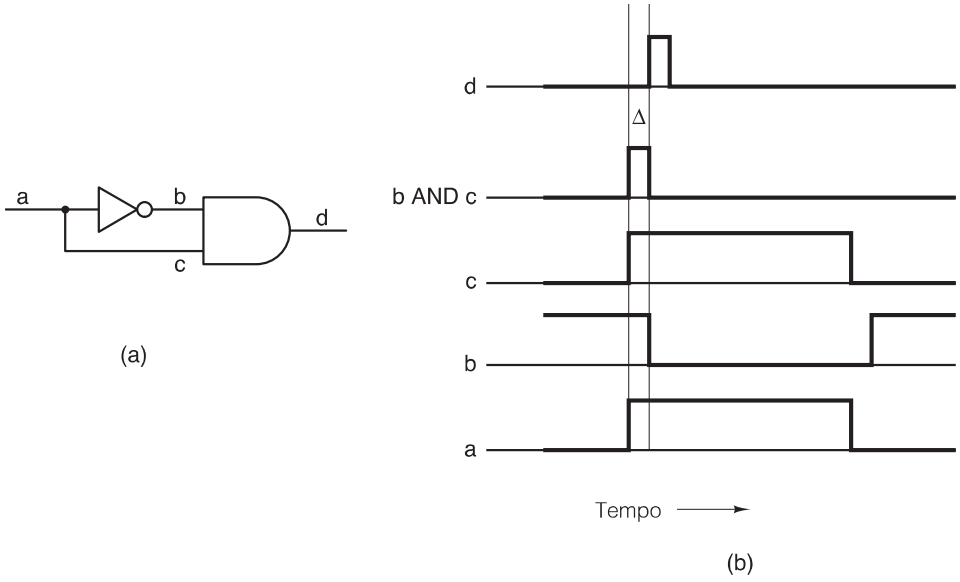
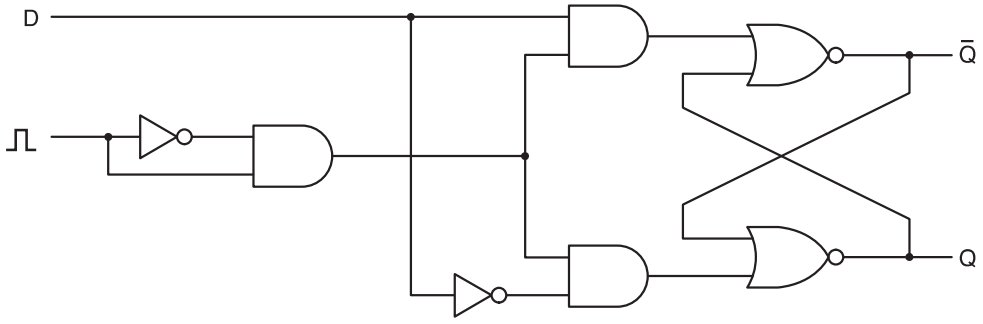


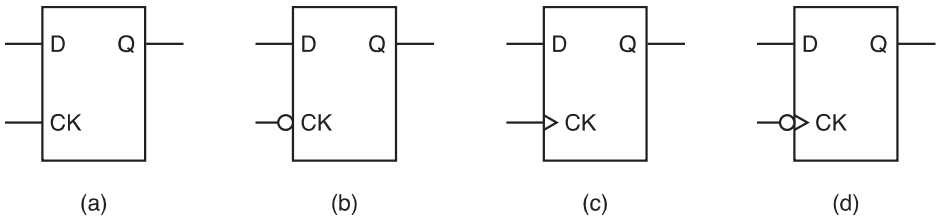
Figura 3.25 *Flip-flop* D.



Os símbolos padronizados para *latches* e *flip-flops* são mostrados na Figura 3.26. A Figura 3.26(a) é um *latch* cujo estado é carregado quando o *clock*, *CK*, é 1, ao contrário da Figura 3.26(b), que é um *latch* cujo *clock* costuma ser 1, mas cai para 0 momentaneamente para carregar o estado a partir de *D*. As figuras 3.26(c) e (d) são *flip-flops* em vez de *latches*, o que é indicado pelo símbolo em ângulo nas entradas do *clock*. A Figura 3.26(c) muda de estado na borda ascendente do pulso do *clock* (transição de 0 para 1), enquanto a Figura 3.26(d) muda de estado na borda

descendente (transição de 1 para 0). Muitos *latches* e *flip-flops* (mas não todos) também têm $\overline{Q}$ como uma saída, e alguns têm duas entradas adicionais *Set* ou *Preset* (que forçam o estado para $Q = 1$) e *Reset* ou *Clear* (que forçam o estado para $Q = 0$).

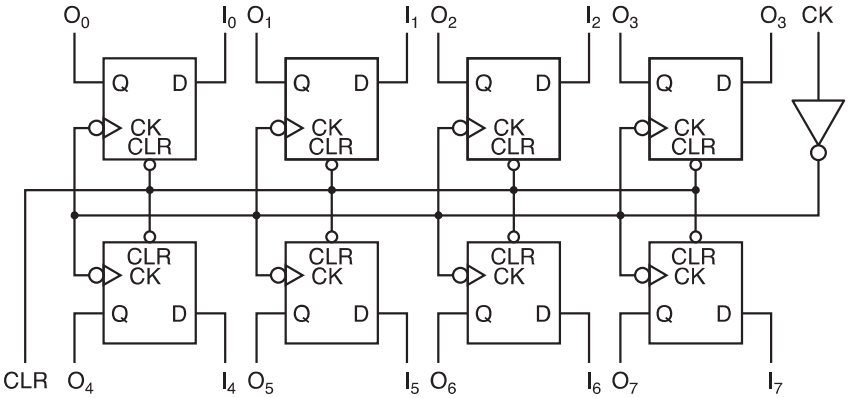
Figura 3.26 *Latches* e *flip-flops* D.



3.3.3 Registradores

Flip-flops podem ser combinados em grupos para criar registradores, que mantêm tipos de dados com comprimentos maiores do que 1 bit. O registrador na Figura 3.27 mostra como oito *flip-flops* podem ser ligados para formar um registrador armazenador de 8 bits. O registrador aceita um valor de entrada de 8 bits (*I*0 a *I*7) quando o *clock CK* fizer uma transição. Para implementar um registrador, todas as linhas de *clock* são conectadas ao mesmo sinal de entrada *CK*, de modo que, quando o *clock* fizer uma transição, cada registrador aceitará o novo valor de dados de 8 bits no barramento de entrada. Os próprios *flip-flops* são do tipo da Figura 3.26(d), mas as bolhas de inversão nos *flip-flops* são canceladas pelo inversor ligado ao sinal de *clock CK*, de modo que os *flip-flops* são carregados na transição ascendente do *clock*. Todos os oito sinais *clear* também são ligados, de modo que, quando o sinal *clear CLR* passar para 0, todos os *flip-flops* serão forçados a passar para o seu estado 0. Caso você queira saber por que o sinal de *clock CK* é invertido na entrada e depois invertido novamente em cada *flip-flop*, um sinal de entrada pode não ter corrente suficiente para alimentar todos os oito *flip-flops*; o inversor da entrada, na realidade, está sendo usado como um amplificador.

Figura 3.27 Um registrador de 8 bits construído a partir de *flip-flops* de único bit.

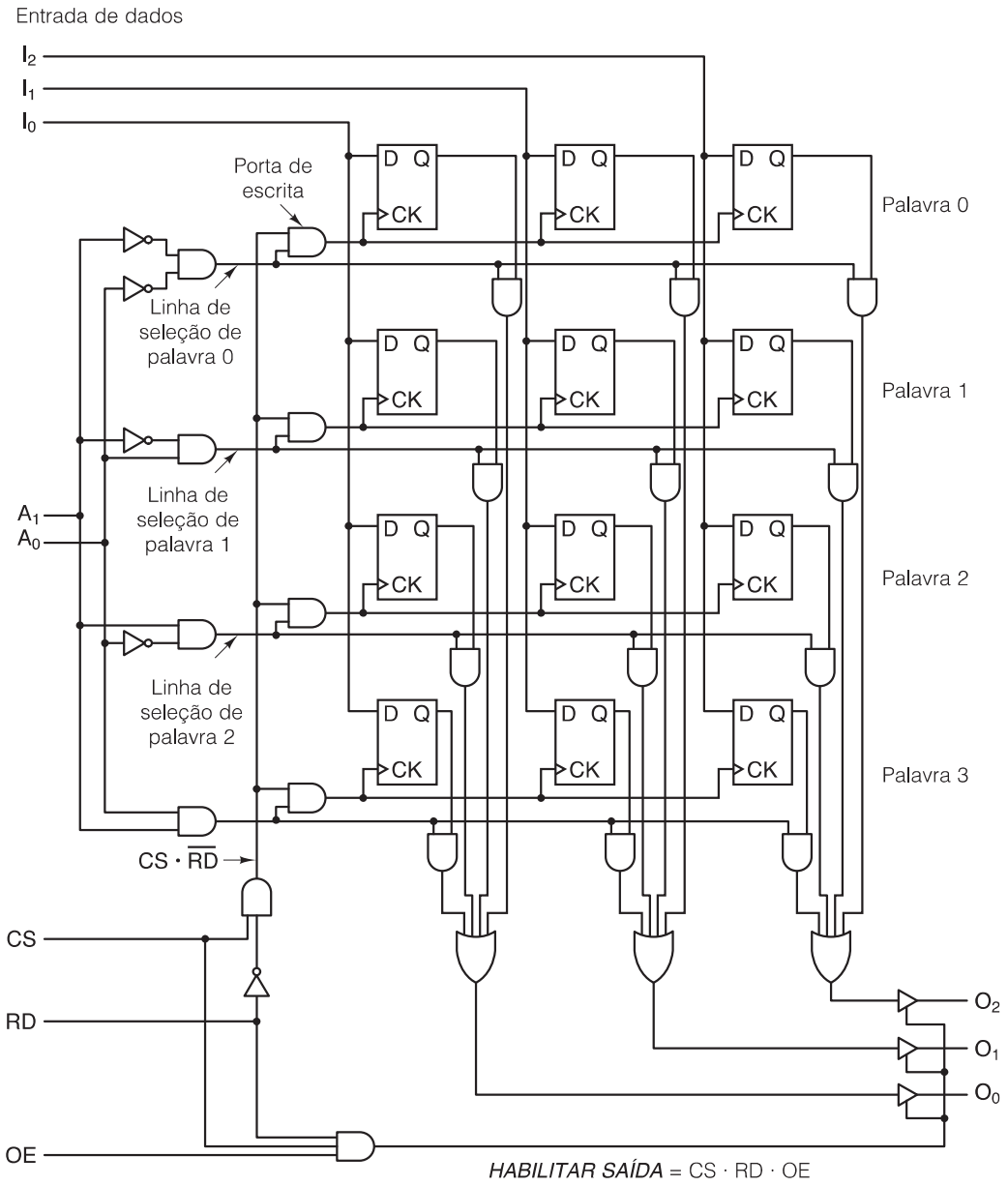


Quando tivermos projetado um registrador de 8 bits, poderemos usá-lo como um bloco de montagem para criar registradores maiores. Por exemplo, um registrador de 32 bits poderia ser criado pela combinação de dois registradores de 16 bits, unindo seus sinais de *clock CK* e sinais de *clear CLR*. Veremos os registradores e seus usos com mais detalhes no Capítulo 4.

3.3.4 Organização da memória

Embora agora tenhamos progredido de uma simples memória de 1 bit da Figura 3.23 para a de 8 bits da Figura 3.27, para construir memórias grandes é preciso uma organização diferente, na qual palavras individuais podem ser endereçadas. Uma organização de memória muito utilizada e que obedece a esse critério é mostrada na Figura 3.28. Esse exemplo ilustra uma memória com quatro palavras de 3 bits. Cada operação lê ou escreve uma palavra completa de 3 bits. Embora uma capacidade total de memória de 12 bits seja pouco mais do que nosso *flip-flop* octal, ela requer um número menor de pinos e, mais importante, o projeto pode ser estendido com facilidade para memórias grandes. Observe que o número de palavras é sempre uma potência de 2.

Figura 3.28 Diagrama lógico para uma memória 4 x 3. Cada linha é uma das quatro palavras de 3 bits. Uma operação de leitura ou escrita sempre lê ou escreve uma palavra completa.



Embora à primeira vista talvez pareça complicada, a memória da Figura 3.28 na verdade é bastante simples devido à sua estrutura regular. Ela tem oito linhas de entrada e três de saída. Três entradas são de dados: I_0, I_1 e I_2; duas são para o endereço: A_0 e A_1; e três são para controle: CS para *chip select* (selecionar chip), RD para distinguir entre ler e escrever e OE para *output enable* (habilitar saída). As três saídas são para dados: O_0, O_1 e O_2. É interessante notar que essa memória de 12 bits requer menos sinais que o registrador de 8 bits anterior. Este requer 20 sinais, incluindo alimentação e terra, enquanto a memória de 12 bits requer apenas 13 sinais. O bloco de memória requer menos sinais porque, diferente do registrador, os bits de memória compartilham um sinal de saída. Nessa memória, cada um dos 4 bits de memória compartilha um sinal de saída. O valor das linhas de endereço determina quais dos 4 bits de memória pode receber ou enviar um valor.

Para selecionar esse bloco de memória, a lógica externa deve estabelecer CS alto e também RD alto (1 lógico) para leitura e baixo (0 lógico) para escrita. As duas linhas de endereço devem ser ajustadas para indicar qual das quatro palavras de 3 bits deve ser lida ou escrita. Para uma operação de leitura, as linhas de entrada de dados não são usadas, mas a palavra selecionada é colocada nas linhas de saída de dados. Para uma operação de escrita, os bits presentes nas linhas de entrada de dados são carregados na palavra de memória selecionada; as linhas de saída de dados não são usadas.

Agora, vamos examinar atentamente a Figura 3.28 para ver como isso funciona. As quatro portas AND de seleção de palavras à esquerda da memória formam um decodificador. Os inversores de entrada foram instalados de modo que cada porta é habilitada (saída é alta) por um endereço diferente. Cada porta comanda uma linha de seleção de palavra, de cima para baixo, para as palavras 0, 1, 2 e 3. Quando o chip é selecionado para uma escrita, a linha vertical rotulada $\text{CS} \cdot \overline{\text{RD}}$ estará alta, habilitando uma das quatro portas de escrita, dependendo de qual linha de seleção de palavra esteja alta. A saída da porta de escrita comanda todos os sinais CK para a palavra selecionada, carregando os dados de entrada nos *flip-flops* para aquela palavra. Uma escrita é efetuada apenas se CS estiver alto e RD estiver baixo, e, ainda assim, somente a palavra selecionada por A_0 e A_1 é escrita; as outras palavras não são alteradas.

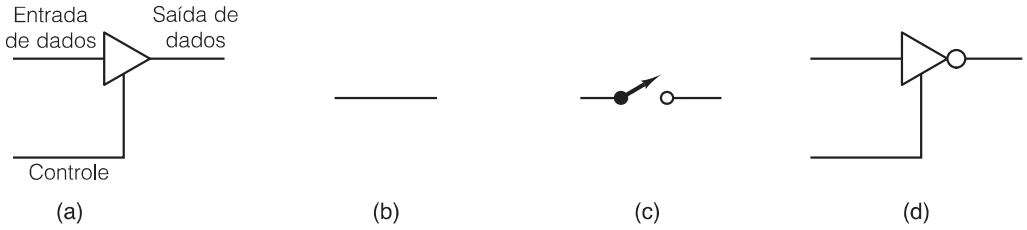
Ler é semelhante a escrever. A decodificação de endereço é idêntica à da escrita. Mas agora a linha $\text{CS} \cdot \overline{\text{RD}}$ está baixa, portanto, todas as portas de escrita estão desabilitadas e nenhum dos *flip-flops* é modificado. Em vez disso, a linha de seleção de palavra que for escolhida habilita as portas AND vinculadas aos Q bits da palavra selecionada. Portanto, a palavra selecionada entrega seus dados às portas OR de quatro entradas na parte inferior da figura, enquanto as outras três palavras produzem 0s. Em consequência, a saída das portas OR é idêntica ao valor armazenado na palavra selecionada. As três palavras não selecionadas não dão nenhuma contribuição à saída.

Embora pudéssemos ter projetado um circuito no qual as três portas OR fossem diretamente ligadas às três linhas de saída de dados, essa operação às vezes causa problemas. Em particular, mostramos que as linhas de entrada de dados e as linhas de saída de dados são diferentes, porém, nas memórias em si, as mesmas linhas são usadas. Se tivéssemos vinculado as portas OR às linhas de saída de dados, o chip tentaria produzir dados, isto é, forçar cada linha a um valor específico, mesmo nas escritas, interferindo desse modo com os dados de entrada. Por essa razão, é desejável ter um meio de conectar as portas OR às linhas de saída de dados em leituras, mas desconectá-las completamente nas escritas. O que precisamos é de um comutador eletrônico que possa estabelecer ou interromper uma conexão em poucos nanossegundos.

Felizmente, esses comutadores existem. A Figura 3.29(a) mostra o símbolo para o que denominamos ***buffer* não inversor**, que tem uma entrada e uma saída de dados e uma entrada de controle. Quando a entrada de controle estiver alta, o *buffer* age como um fio, como mostra a Figura 3.29(b). Quando a entrada de controle estiver baixa, ele age como um circuito aberto, como mostra a Figura 3.29(c); é como se alguém desconectasse a saída de dados do resto do circuito com um alicate de corte. Contudo, ao contrário do que aconteceria no caso do alicate de corte, a conexão pode ser restaurada logo em seguida, dentro de alguns nanossegundos, apenas fazendo o sinal de controle ficar alto novamente.

A Figura 3.29(d) mostra um ***buffer* inversor**, que funciona como um inversor normal quando o controle estiver alto, e desconecta a saída do circuito quando o controle estiver baixo. Ambos os tipos de *buffers* são

Figura 3.29 (a) *Buffer* não inversor. (b) Efeito de (a) quando o controle está alto. (c) Efeito de (a) quando o controle está baixo. (d) *Buffer* inversor.



dispositivos de três estados, porque podem produzir 0, 1, ou nenhum dos dois (circuito aberto). *Buffers* também amplificam sinais, portanto, podem comandar muitas entradas simultaneamente. Às vezes, eles são usados em circuitos por essa razão, mesmo quando suas propriedades de comutação não são necessárias.

Voltando ao circuito de memória, agora já deve estar claro para que servem os três *buffers* não inversores nas linhas de saída de dados. Quando CS, RD e OE estiverem todos altos, o sinal *output enable* também está alto, habilitando os *buffers* e colocando uma palavra nas linhas de saída. Quando qualquer um dos CS, RD ou OE estiver baixo, as saídas de dados são desconectadas do resto do circuito.

3.3.5 Chips de memória

O bom da memória da Figura 3.28 é que ela pode ser ampliada com facilidade para tamanhos maiores. Em nosso desenho, a memória é 4 × 3, isto é, quatro palavras de 3 bits cada. Para ampliá-la para 4 × 8, basta adicionar cinco colunas de quatro *flip-flops* cada, bem como cinco linhas de entrada e cinco linhas de saída. Para passar de 4 × 3 para 8 × 3, devemos acrescentar quatro linhas de três *flip-flops* cada, bem como uma linha de endereço A_2. Com esse tipo de estrutura, o número de palavras na memória deve ser uma potência de 2 para que haja o máximo de eficiência, mas o número de bits em uma palavra pode ser qualquer um.

Como a tecnologia de circuitos integrados se ajusta bem à fabricação de chips cuja estrutura interna é um padrão bidimensional repetitivo, chips de memória são uma aplicação ideal para ela. À medida que a tecnologia melhora, o número de bits que podem ser colocados em um chip continua crescendo, normalmente por um fator de dois a cada 18 meses (lei de Moore). Os chips maiores nem sempre tornam os menores obsoletos devido aos diferentes compromissos entre capacidade, velocidade, energia, preço e conveniência da interface. Em geral, os chips maiores disponíveis no momento são vendidos por preços mais elevados, portanto, são mais caros por bit do que os antigos, menores.

Há vários modos de organizar o chip para qualquer tamanho de memória dado. A Figura 3.30 mostra duas organizações possíveis para um chip de memória mais antigo de 4 Mbits de tamanho: 512 K × 8 e 4.096 K × 1. (A propósito, os tamanhos de chips de memória costumam ser citados em bits em vez de bytes, e por isso adotaremos essa convenção.) Na Figura 3.30(a), são necessárias 19 linhas de endereço para endereçar um dos 2^{19} bytes e oito linhas de dados para carregar e armazenar o byte selecionado.

Cabe aqui uma observação sobre tecnologia. Em alguns pinos, a alta tensão provoca uma ação. Em outros, é a baixa tensão que causa uma ação. Para evitar confusão, preferimos manter a coerência e dizer sempre que o sinal é **afirmado** (em vez de dizer que fica alto ou baixo), o que significa que foi disparado para provocar alguma ação. Assim, para alguns pinos, afirmá-lo significa estabelecê-lo alto. Para outros, significa estabelecer o pino baixo. Os nomes de sinais de pinos afirmados baixos são distinguidos por uma barra superior. Assim, um sinal com rótulo CS é ativado alto, mas um sinal com rótulo $\overline{\text{CS}}$ é ativado baixo. O oposto de afirmado é **negado**. Quando nada de especial estiver acontecendo, os pinos são negados.

Agora, vamos voltar ao nosso chip de memória. Uma vez que um computador costuma ter muitos chips de memória, é preciso um sinal para selecionar o chip necessário no momento em questão, de modo que ele responda e todos os outros não. O sinal $\overline{CS}$ (*chip select* – seleção de chip) existe para essa finalidade e é ativado para habilitar o chip. Além disso, é preciso uma maneira de distinguir entre leituras e escritas. O sinal $\overline{WE}$ (*write enable* – habilitar escrita) é usado para indicar que os dados estão sendo escritos, e não lidos. Por fim, o sinal $\overline{OE}$ (*output enable* – habilitar saída) é afirmado para comandar os sinais de saída. Quando ele não é afirmado, a saída do chip é desconectada do circuito.

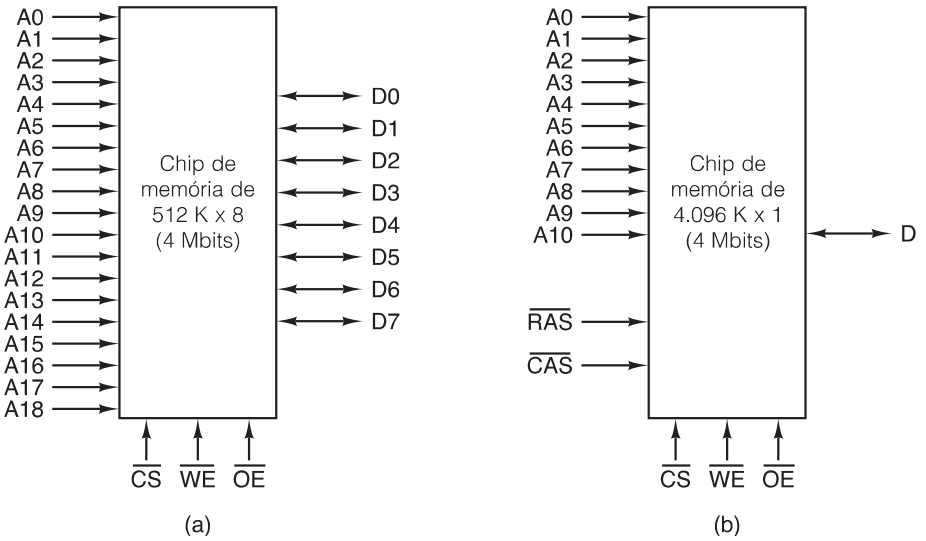
Na Figura 3.30(b), é usado um esquema de endereçamento diferente. Esse chip é organizado internamente como uma matriz 2.048 × 2.048 de células de 1 bit, o que dá 4 Mbits. Para endereçar o chip, em primeiro lugar uma linha é selecionada ao se colocar seu número de 11 bits nos pinos de endereço. Então o $\overline{RAS}$ (*row address strobe* – *strobe* de endereço de linha) é afirmado. Em seguida, um número de coluna é colocado nos pinos de endereço e o $\overline{CAS}$ (*column address strobe* – *strobe* de endereço de coluna) é afirmado. O chip responde aceitando ou entregando um bit de dados.

Chips de memória de grande porte costumam ser construídos como matrizes $n \times n$ endereçadas por linha e coluna. Essa organização reduz o número de pinos requerido, mas também torna mais lento o endereçamento do chip, já que são necessários dois ciclos, um para a linha e outro para a coluna. Para recuperar um pouco da velocidade perdida por esse projeto, alguns chips de memória podem receber um endereço de linha acompanhado por uma sequência de endereços de coluna para acessar bits consecutivos em uma linha.

Anos atrás, os maiores chips de memória costumavam ser organizados como os da Figura 3.30(b). À medida que as palavras de memória cresciam de 8 bits até 32 bits e mais, os chips de 1 bit começaram a ser inconvenientes. Construir uma memória com uma palavra de 32 bits usando chips de 4.096 K × 1 requer 32 chips em paralelo. Esses 32 chips têm capacidade total de no mínimo 16 MB, ao passo que usar chips de 512 K × 8 requer somente quatro chips em paralelo e permite memórias pequenas, de até 2 MB. Para evitar ter 32 chips para memória, grande parte dos fabricantes lançou famílias com 4, 8 e 16 bits de largura. A situação com as palavras de 64 bits é pior ainda, é claro.

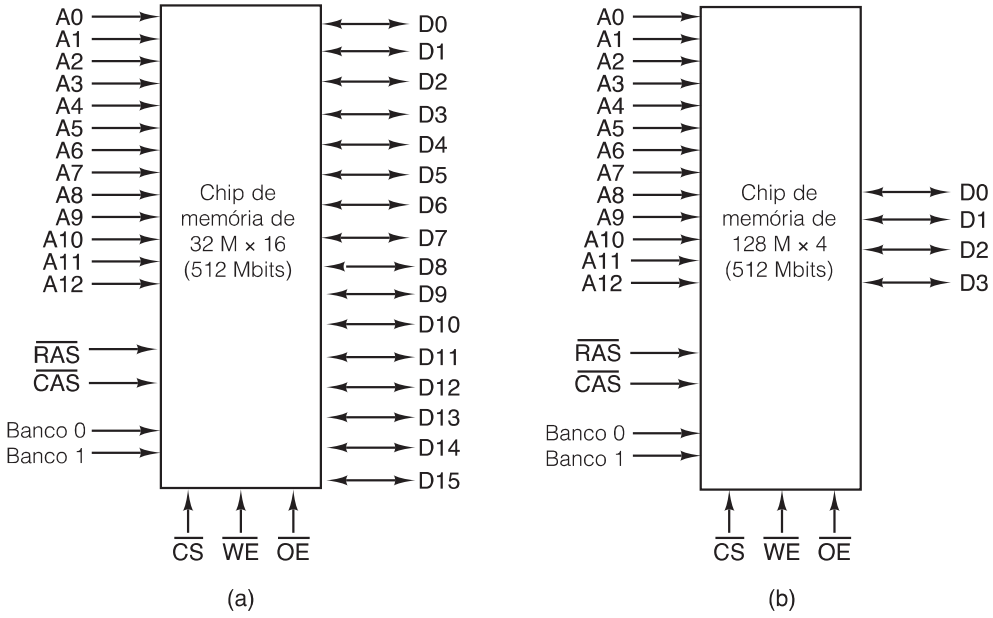
Dois exemplos de chips modernos de 512 Mbits são dados na Figura 3.31. Esses chips têm quatro bancos de memória internos de 128 Mbits cada, o que requer duas linhas de seleção de banco para escolher um banco. O projeto da Figura 3.31(a) é de um chip de 32 M × 16 com 13 linhas para o sinal $\overline{RAS}$, 10 linhas para o sinal $\overline{CAS}$ e 2 linhas para a seleção de banco. Juntos, esses 25 sinais permitem o endereçamento de cada uma das 2^{25} células

Figura 3.30 Dois modos de organizar um chip de memória de 4 Mbits.



internas de 16 bits. Em comparação, a Figura 3.31(b) apresenta um projeto de 128 M × 4 com 13 linhas para o sinal $\overline{\text{RAS}}$, 12 linhas para o sinal $\overline{\text{CAS}}$ e 2 linhas para a seleção de banco. Nesse caso, 27 sinais podem selecionar quaisquer das 2^{27} células internas de 4 bits a serem endereçadas. A decisão sobre o número de linhas e de colunas que um chip tem é tomada por razões de engenharia. A matriz não precisa ser quadrada.

Figura 3.31 Dois modos de organizar um chip de memória de 512 Mbits.



Esses exemplos demonstram duas questões separadas e independentes para o projeto do chip de memória. A primeira é a largura da saída (em bits): o chip entrega 1, 4, 8, 16 ou algum outro número de bits de uma vez só? A segunda é se todos os bits de endereço são apresentados em pinos separados de uma vez só ou se as linhas e colunas são apresentadas em sequência, como nos exemplos da Figura 3.31. Um projetista de chips de memória tem de responder a ambas as perguntas antes de iniciar o projeto do chip.

3.3.6 RAMs e ROMs

Todas as memórias que estudamos até aqui podem ser escritas e lidas. Elas são denominadas memórias **RAM** (Random Access Memory – memória de acesso aleatório), um nome suspeito porque todos os chips de memória têm acesso aleatório. No entanto, o termo já é muito utilizado para que o mudemos agora. RAMs podem ser de duas variedades, estáticas e dinâmicas. Nas estáticas (**Static RAMs – SRAMs**), a construção interna usa circuitos similares ao nosso *flip-flop* D básico. Uma das propriedades dessas memórias é que seus conteúdos são conservados enquanto houver fornecimento de energia: segundos, minutos, horas e até mesmo dias. As RAMs estáticas são muito rápidas. Um tempo de acesso típico é da ordem de um nanossegundo ou menos. Por essa razão, elas são muito usadas como memória *cache*.

RAMS dinâmicas (Dynamic RAMs – DRAMs), ao contrário, não usam *flip-flops*. Em vez disso, uma RAM dinâmica é um arranjo de células, cada uma contendo um transistor e um pequenino capacitor. Os capacitores podem ser carregados ou descarregados, permitindo que 0s e 1s sejam armazenados. Como a carga elétrica tende a vazar, cada bit em uma RAM dinâmica deve ser **renovado** (recarregado) com alguns milissegundos de intervalo para evitar que os dados desapareçam. Como a lógica externa é que tem de cuidar da renovação, as RAMs dinâmicas precisam de uma interface mais complexa do que as estáticas, embora em muitas aplicações essa desvantagem seja compensada por suas maiores capacidades.