CAPÍTULO 8
CONTROLADORES

No Cap. 7 introduzimos algumas idéias úteis e importantes concernentes a sistemas lógicos seqüenciais, isto é, sistemas com memória. Para ilustrar o projeto de sistemas seqüenciais, usamos o exemplo do detector de seqüências. Um tipo de sistema seqüencial que tem uma faixa de aplicabilidade muito mais ampla é o *controlador*. Os controladores são sistemas seqüenciais que fornecem níveis lógicos apropriados em tempos apropriados para controlar uma seqüência de operações lógicas simples que, juntas, executam uma operação complicada. Consideremos primeiro como estas operações lógicas simples podem ser efetuadas.

8.1 TRANSFERÊNCIAS DE REGISTRADORES

As operações elementares, lógicas ou aritméticas, que podem ser efetuadas sobre palavras lógicas são muito simples. Uma operação básica típica e a mais importante consiste simplesmente na transferência do conteúdo de um registrador para um segundo registrador. Um mecanismo de uma tal transferência é mostrado na Fig. 8.1-1, onde temos dois registradores, o registrador A e o registrador B. Os registradores são considerados aqui como consistindo de um arranjo de inúmeros latches estáticos (RS) de set-reset, como na Fig. 4.2-1. Somente um latch do registrador A, o latch A_i, e um do registrador B, o latch B_i, estão mostrados. Os outros latches estão representados pelos pontos que aparecem em ambos os lados dos latches. Duas portas AND estão interpostas entre os latches. Estas portas AND podem ser habilitadas elevando-se o *terminal de controle* assinalado "Move A para B" ao nível lógico 1. Em tal habilitação, S de B_i assumirá o valor Q de A_i, e R assumirá o valor $\overline{Q}$ de A_i. O latch B_i assumirá, portanto, o estado de A_i. Quando o terminal de controle "Move A para B" retorna ao nível lógico 0, tanto R quanto S tornar-se-ão 0 e o estado de A_i será retido em B_i. Em resumo, a colocação do terminal de controle em seu nível ativo gera um

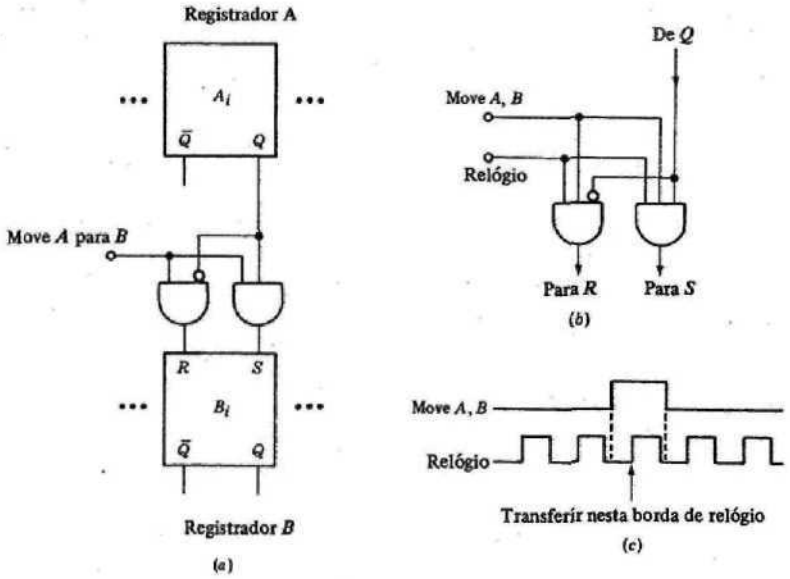


Fig. 8.1-1 (*a*) Elevação do terminal de controle "Move A para B" para lógica 1 brevemente movimenta o conteúdo do registrador A para o registrador B. (*b*) Um relógio é acrescentado para operação síncrona. (*c*) Temporização da operação de movimentar.

comando a que o circuito responde. O terminal de controle é em conseqüência muitas vezes denominado *terminal de comando*. Notar que neste processo a palavra armazenada no registrador A permanecerá inalterada, mas, naturalmente, qualquer palavra armazenada no registrador B antes da transferência será perdida.

Se, por exemplo, os registradores A e B forem registradores de dezesseis bits, então são requeridas dezesseis linhas de conexão, bem como dezesseis pares de portas AND. A inversão exigida na porta AND que precede a entrada R seria desnecessária se tivéssemos escolhido acrescentar uma segunda linha de conexão da saída $\bar{Q}$ de A_i. Escolhemos não fazer isto com base em que fisicamente a implementação de uma inversão muitas vezes é mais simples do que o acréscimo de uma conexão.

Mais importante, notamos que transferir o conteúdo do registrador A para o registrador B requer simplesmente que em curto tempo habilitemos um arranjo de portas mudando o nível lógico de um terminal de controle. Se estivermos tratando com um sistema síncrono (o caso usual), podemos igualmente querer que a transferência ocorra em sincronismo com o relógio. Tal transferência síncrona pode ser efetuada acrescentando-se uma entrada de relógio às portas AND, conforme mostrado na Fig. 8.1-1*b*. A Fig. 8.1-1*c* mostra uma forma-de-onda de relógio e uma mudança do nível de habilitação em "Move A para B" que seleciona um ciclo de relógio particular durante o qual ocorre a transferência.

Suponhamos que temos muitos registradores e requeremos a facilidade de efetuar uma transferência de qualquer registrador para qualquer outro registrador. Se fornecermos conexões

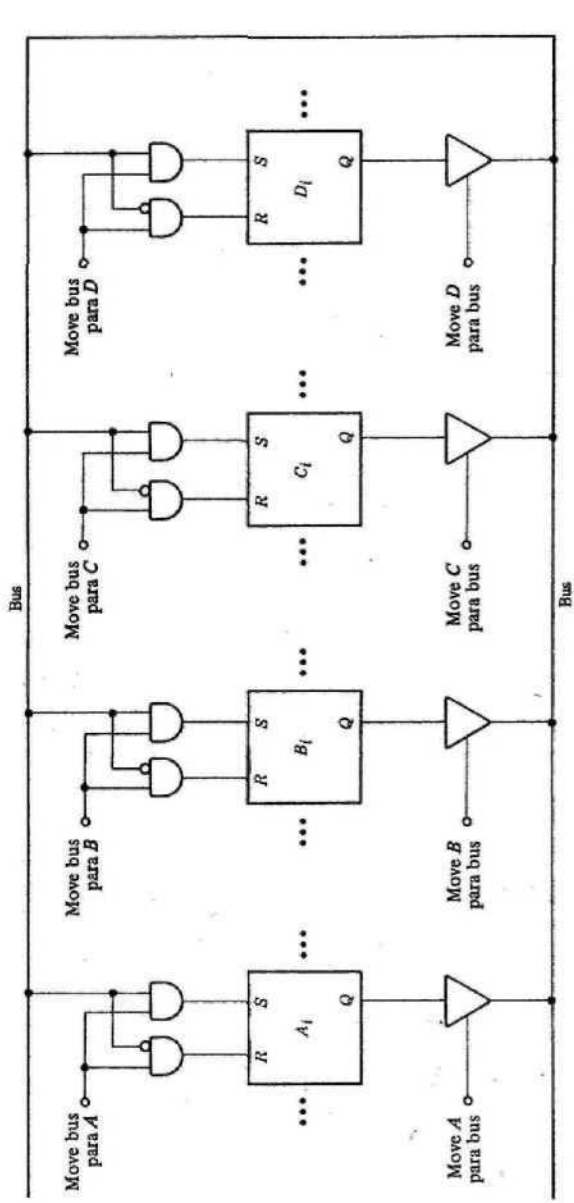


Fig. 8.1-2 Inúmeros registradores compartilham de um bus comum. Elevando-se para 1 lógico um controle de entrada e um controle de saída de cada vez, as transferências entre registradores são realizadas.

individuais de cada registrador para todos os outros, o número de tais conexões pode igualmente descontrolar-se. Num tal caso podemos também querer usar um esquema de multiplexação (ver Seção 3.19) em que um bus (*linha de transmissão de dados*) comum é empregado. O bus é mostrado na Fig. 8.1-2 e serve como uma interconexão entre muitos registradores. Somente quatro registradores são indicados, e para cada registrador somente um latch é mostrado. Correspondentemente, apenas um fio do bus é mostrado. Se os registradores fossem, por exemplo, registradores de dezesseis bits, haveria dezesseis fios separados no bus. Uma vez que toda informação deve ser transferida sobre um único bus, é possível fazer somente uma transferência de cada vez. Se, por exemplo, quisermos fazer uma transferência de A para C, elevaríamos para 1 lógico o nível no terminal de controle "Move A para o bus" e o estado do registrador A seria colocado no bus. (Na Fig. 8.1-2 usamos a conexão de três-estados para acoplar as saídas de muitos registradores a um único bus. Alternativamente, a conexão de *coletor aberto* da Seção 3.19 poderia ser usada.) Se agora também colocarmos em nível lógico 1 o terminal marcado "Move bus para C"; a transferência será realizada. Mas, novamente, o ponto importante a observar é que a operação de transferência é obtida pelas portas de habilitação, neste caso dois conjuntos de portas, fazendo linhas disponíveis sobre as quais o nível lógico 1 é mudado para o nível de habilitação.

8.2 OUTRAS OPERAÇÕES

Complementação

Uma segunda operação lógica simples comumente requerida consiste na complementação de uma palavra. Aqui temos em mente a substituição de uma palavra num registrador por uma nova palavra, sendo cada bit da nova palavra o complemento do bit correspondente da palavra original. A Fig. 8.2-1 mostra a palavra originalmente no registrador A; o complemento deve aparecer no registrador B. Esta transferência e complementação é realizada simplesmente elevando-se para 1 lógico os terminais "Complemento" e "Move A para B". Se for empregado um relógio, como indicado, "Complementa" e "Move A para B" terão que estar em 1 lógico quando o relógio também se eleva para 1 lógico, e a transferência ocorrerá sincronamente com a borda de relógio. Na Fig. 8.2-1*a* o registrador pode consistir simplesmente em um arranjo ou conjunto de latches.

Suponhamos, por outro lado, que não queremos usar um segundo registrador em que o complemento deve aparecer. Em vez disto, queremos usar apenas um registrador e queremos que a palavra original seja substituída por seu complemento. Depois propomo-nos a ler do registrador (para ver qual bit está em cada flip-flop) e, no mesmo ciclo de relógio, a escrever no flip-flop, isto é, escrever no flip-flop o complemento do bit que acabamos de ler. Para este propósito requeremos que os flip-flops no registrador sejam do tipo especial (mestre-escravo etc., conforme discutido no Cap. 4), o que permite leitura e escrita simultâneas. Na Fig. 8.2-1*b* usamos os flip-flops JK no registrador. Quando a linha "Complementa" estiver em 1, o flip-flop chaveará na transição de gatilho de relógio, substituindo desta maneira cada bit por seu complemento.

Deslocamento

A Fig. 8.2-2 mostra um arranjo que possibilita uma transferência do registrador A para o registrador B. Dependendo de se S_O, S_L ou S_R está em lógica 1, o deslocamento será direto ou será acompanhado por um deslocamento numa direção ou em outra. São mostrados três flip-flops adja-

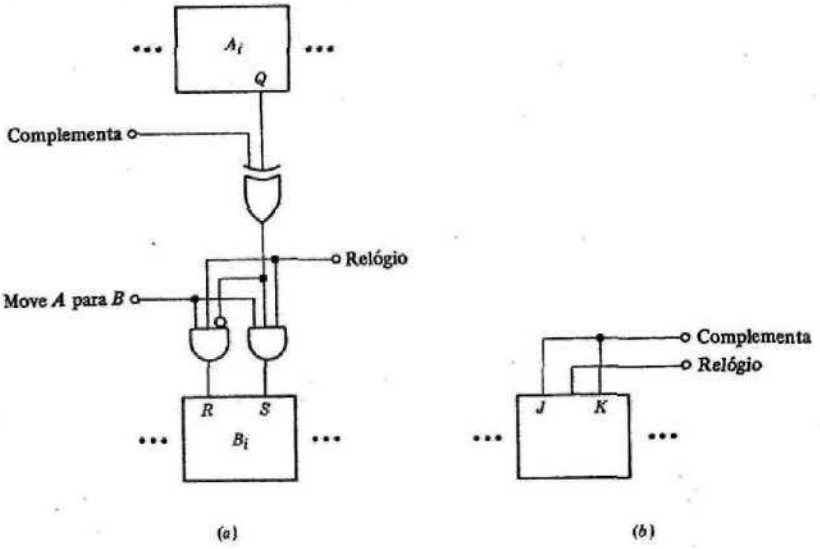


Fig. 8.2-1 (*a*) Um arranjo que permite uma transferência do registrador A para o registrador B do conteúdo de A (Complemento = 0) ou o complemento de A bit por bit (Complemento = 1). (*b*) Um arranjo que possibilita a complementação sem transferência.

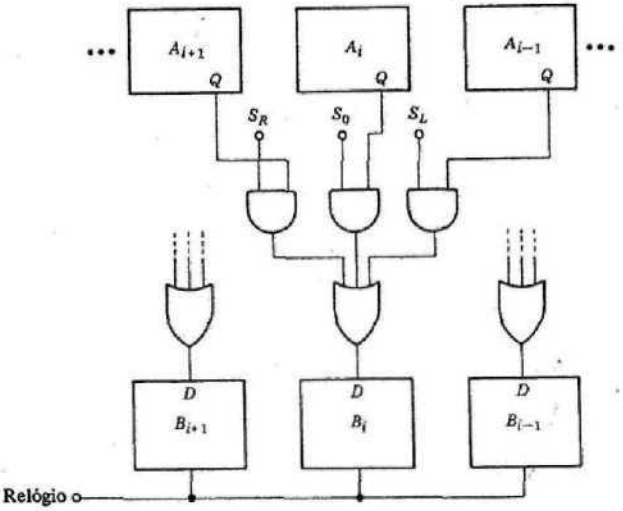


Fig. 8.2-2 Um arranjo que possibilita a transferência do registrador A para o registrador B sem deslocamento ($S_R S_O S_L = 10$), com deslocamento à esquerda ($S_R S_O S_L = 001$) ou com deslocamento à direita ($S_R S_O S_L = 100$).

centes do registrador A e três do registrador B. O registrador B é aqui suposto compreender os flip-flops do tipo D. Há uma estrutura de quatro portas (três portas AND e uma porta OR) associada à entrada D de cada flip-flop do registrador B, mas somente uma estrutura é mostrada completamente. Se somente S_O for $S_O = 1$, o conteúdo de A_i será transferido para B_i na ocorrência da borda de gatilho de relógio. Se somente S_L for $S_L = 1$, haverá um deslocamento à esquerda e transferência. $S_R = 1$ alcançará um deslocamento à direita e transferência. Naturalmente, num deslocamento à esquerda teremos que fazer provisão especial para o bit que deve ser deslocado para o flip-flop mais à direita, uma vez que este bit deve ser fornecido por uma fonte externa. Um comentário semelhante se aplica ao bit a ser deslocado para o flip-flop mais à esquerda num deslocamento à direita. Em qualquer caso, e mais importante, notamos novamente que a operação a ser realizada é conseguida mantendo em lógica 1, durante um ciclo de relógio, um ou outro dos terminais de controle S_O, S_L ou S_R.

Suponhamos que não necessitamos da facilidade de transferir e deslocar uma palavra, mas somente de deslocá-la, mantendo-a no mesmo registrador. Depois devemos requerer nada mais do que o registrador de deslocamento à direita-à esquerda já descrito na Seção 4.18.

Incrementação e Decrementação

Muitas vezes devemos armazenar um número num registrador e incorporar no registrador a facilidade de alterar o número armazenado por +1 ou por -1. Estas operações são chamadas *incrementação* e *decrementação*. Um registrador que responderá ao comando de *altera contagem*, mostrado na Fig. 8.2-3, consiste em um contador incrementador-decrementador (de qualquer tipo, assíncrono ou síncrono) com a modificação de que o relógio não é aplicado diretamente à entrada de relógio do contador. Em vez disto, uma porta AND é acrescentada e o relógio é passado através dessa porta acrescentada. A outra entrada da porta AND é uma entrada de habilitação chamada *incremento*. Se o terminal de altera contagem for mantido em lógica 1 na direção de um ciclo de relógio, o contador incrementará ou decrementará seu registro de 1, dependendo do nível lógico do controle de modos.

Reajuste e Ajuste (Resetting and Setting)

Suponhamos que necessitamos da facilidade de limpar um registrador (colocar Q de cada flip-flop em $Q = 0$) ou ajustar (set) um registrador (cada $Q = 1$). Um flip-flop individual de um tal registrador é mostrado na Fig. 8.2-4. Um flip-flop JK está sendo empregado aqui. Se a entrada

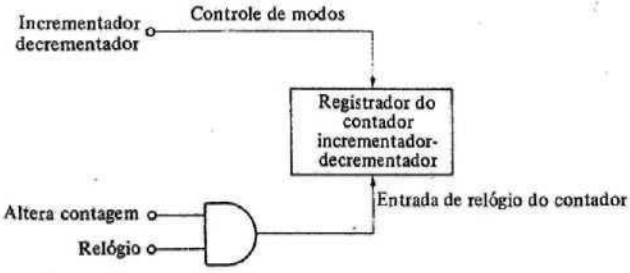


Fig. 8.2-3 O registrador do contador incrementará ou decrementará sua contagem, dependendo do nível lógico do controle de modo, se o terminal "Altera contagem" for mantido em lógica 1 na duração de um ciclo de relógio.

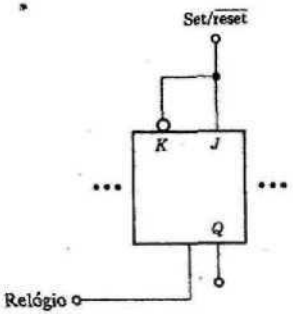


Fig. 8.2-4 Um flip-flop *JK* usado como parte de um registrador e dotado da facilidade de ser limpo ou ajustado (set).

de set/reset estiver em lógica 1, então $J = 1$ e $K = 0$, de modo que em uma borda de gatilho de relógio o flip-flop se ajustará. Se a entrada de set/reset estiver em lógica 0, teremos $J = 0$ e $K = 1$, de modo que o flip-flop se reajustará (reset).

8.3 REGISTRADOR SENSÍVEL A COMANDOS MÚLTIPLOS

Vimos que podemos construir um registrador que responderá a um *comando* ou a outro. O comando é transportado colocando-se o nível lógico de um certo *terminal de controle* no nível que habilita uma porta ou um conjunto de portas. Num sistema síncrono que usa um relógio, o comando geralmente será executado na borda de gatilho de uma forma-de-onda de relógio. Suponhamos que num certo sistema digital requeremos a facilidade de comandar inúmeras operações. Podemos então escolher efetuar operações individuais em registradores separados ou construir um registrador que seja capaz de responder a inúmeros comandos diferentes. A primeira alternativa oferece a vantagem da flexibilidade; a segunda alternativa pode-nos possibilitar a economia de algum hardware.

Como exemplo de um registrador que pode responder a inúmeros comandos, projetemos um registrador que é capaz de responder a cinco comandos. Estes cinco comandos e o símbolo que deve ser associado a cada comando estão listados na Tab. 8.3-1. Portanto, nosso registrador deve ter cinco terminais de controle *W*, *R*, *I*, *C* e *Z*. A qualquer instante apenas um destes terminais deve ser mantido em lógica 1, enquanto todos ou outros devem estar em lógica 0. Se, por exemplo,

Tabela 8.3-1 Comandos aos quais responde o registrador

Comando	Símbolo
1. Escrever a palavra do bus no registrador (*write*)	*W*
2. Ler a palavra do registrador no bus (*read*)	*R*
3. Incrementar o registrador	*I*
4. Complementar o registrador	*C*
5. Limpar o registrador de modo que todos os *Q*s sejam zero (reset)	*Z*

tivermos $W = 1$, então na borda de gatilho da forma-de-onda de relógio a palavra sobre o bus (barramento) deve ser introduzida no registrador. Os n flip-flops no registrador são FF_0, FF_1, ..., FF_i, ..., FF_{n-1}. Uma linha B_i do bus de n linhas é mostrada, e a lógica associada exatamente a um flip-flop FF_i é mostrada na Fig. 8.3-1. Decidimos aqui arbitrariamente que os flip-flops JK devem ser usados. A fim de introduzir a lógica na linha B_i do cabo no FF_i em uma transição de gatilho de relógio e, quando $W = 1$, requeremos que

$$J_i = B_i W \qquad \text{e} \qquad K_i = \bar{B}_i W \tag{8.3-1}$$

(Usamos o símbolo B_i tanto para representar a i-ésima linha do bus quanto o nível lógico na linha.) Para transferir o bit em FF_i para o bus, isto é, para ler o flip-flop sobre o bus quando $R = 1$, requeremos que

$$B_i = Q_i R \tag{8.3-2}$$

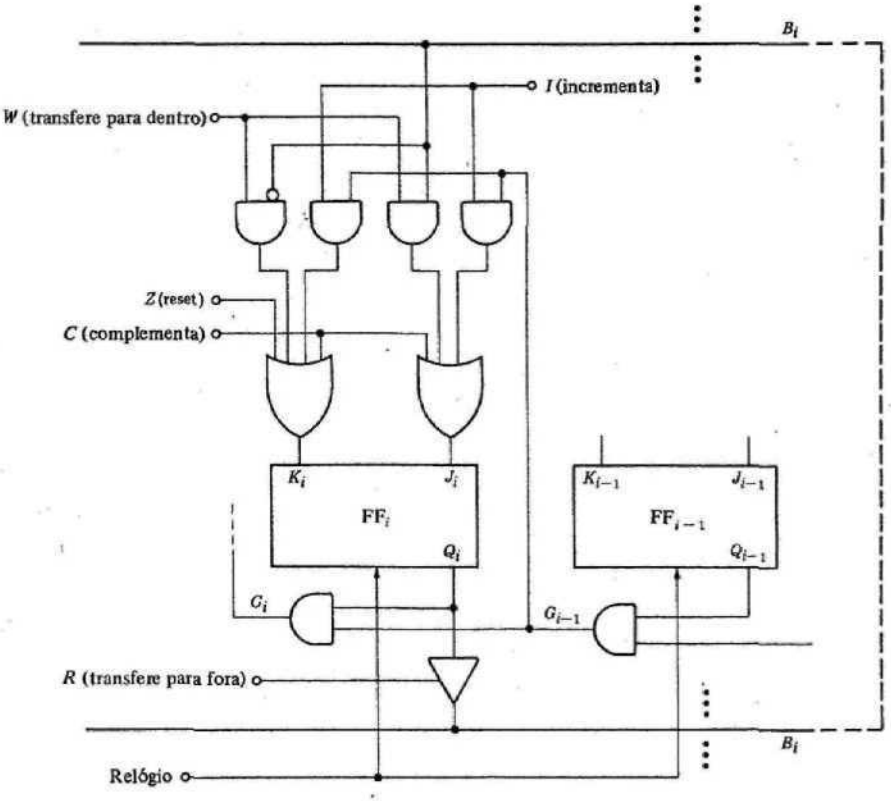


Fig. 8.3-1 Um estágio de um registrador que responderá a cinco comandos.

Notar que esta transferência ocorrerá logo que $R = 1$ e a sincronização (temporização) da transferência não depende do relógio. Naturalmente, se a palavra assim colocada no bus deve ser colocada noutro registro, esta colocação ocorrerá em uma transição de gatilho de relógio.

Para incrementar o registrador em $I = 1$, interconectamos estágios de flip-flop da maneira como fizemos em um contador. Escolhemos arbitrariamente usar um contador assíncrono, de ondulação (contador por pulsação). O contador de ondulação requer que as entradas J e K de cada flip-flop sejam ligadas juntas e depois conectadas à saída Q do flip-flop precedente (ver Fig. 4.27-1). Portanto, requeremos que no i-ésimo flip-flop

$$J_i = K_i = G_{i-1}I \tag{8.3-3}$$

A Eq. (8.3-3) aplica-se a todos os flip-flops, exceto ao primeiro flip-flop FF_0, que não tem estágio precedente. Neste caso especial, requeremos

$$J_0 = K_0 = 1 \cdot I \tag{8.3-4}$$

Para fazer com que um flip-flop complemente, isto é, chaveie, quando $C = 1$, estabelecemos

$$J_i = K_i = C \tag{8.3-5}$$

Finalmente, para fazer o flip-flop limpar (reset) quando $Z = 1$ mas não ser afetado por Z quando $Z = 0$, facilmente verificamos na tabela verdade da Fig. 4.11-1 que requeremos

$$K_i = Z \qquad J_i = 0 \tag{8.3-6}$$

Em conjunto, lembrando que apenas uma das variáveis de controle W, R, I, C e Z está em lógica 1 a qualquer instante, temos das Eqs. (8.3-1) a (8.3-3), (8.3-5) e (8.3-6) que

$$J_i = B_iW + G_{i-1}I + C \qquad K_i = \bar{B}_iW + G_{i-1}I + C + Z \tag{8.3-7}$$

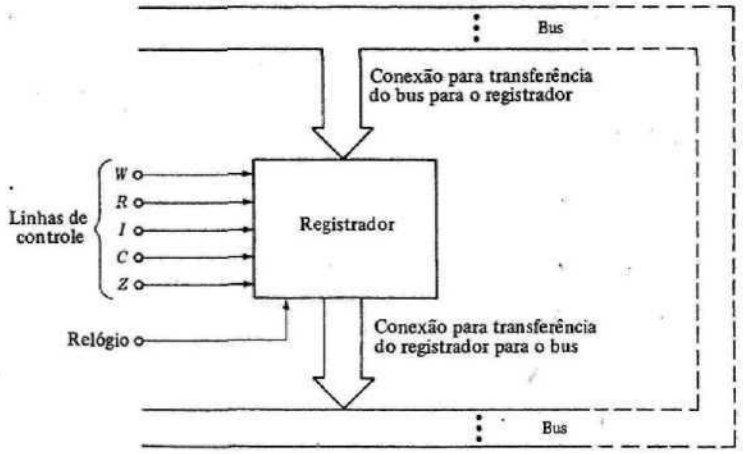


Fig. 8.3-2 Uma representação funcional do registrador cujo único estágio é dado na Fig. 8.3-1.

e

$$B_i = Q_i R \tag{8.3-8}$$

O primeiro flip-flop FF_0 é especial quanto a que os termos $G_{i-1}I$ na Eq. (8.3-7) são substituídos simplesmente pelos termos $1 \cdot I = I$, como indicado na Eq. (8.3-4).

Na Fig. 8.3-1 são mostradas as portas associadas a FF_i requeridas pelas Eqs. (8.3-7) e (8.3-8). Naturalmente, uma estrutura idêntica de portas (não mostrada) é requerida para cada flip-flop no registrador.

Um diagrama em blocos do registrador mostrando suas conexões ao bus é mostrado na Fig. 8.3-2. As cinco linhas de comando de controle estão indicadas, como também a entrada de relógio.

8.4 UM CONTROLADOR SIMPLES

Vimos que é viável construir de maneira direta circuitos combinacionais e registradores que nos permitirão efetuar operações aritméticas ou lógicas simples com as palavras e armazenar o resultado. Uma vez que tenha sido construída uma peça de hardware apropriada, a operação é efetuada pelo simples expediente de elevar à lógica 1 (ou, se quisermos, abaixar para lógica 0) alguma linha de controle que depois serve para habilitar uma porta com um conjunto de portas. Por este ato de habilitação é dado um *comando* a que a peça de hardware então responde. As operações elementares, cuja implementação de hardware examinamos, incluem a transferência *para* e *de* um bus, incrementando, complementando e deslocando. De maneira similar, podemos construir hardware para efetuar outras operações elementares. Suponhamos, por exemplo, que um registrador de n bits que mantenha a palavra $R_{n-1}, \ldots, R_i, \ldots, R_0$ aceite uma palavra de entrada $A_{n-1}, \ldots, A_i, \ldots, A_0$ e depois mude seu registro para $R'_{n-1}, \ldots, R'_1, \ldots, R'_i$, em que cada novo bit é $R'_i = R_i A_i$. Um tal registro efetua a operação AND. Registradores que respondem a comandos para efetuar outras operações lógicas são igualmente possíveis. Assim podemos ter $R'_i = R_i + A$, $R'_i = R_i \oplus A_i$ etc. (ver Probs. 8.2-1, 8.2-2).

Se necessitarmos efetuar inúmeras operações aritméticas e lógicas diferentes com palavras, podemos escolher construir um registrador que possa ser comandado para efetuar todas as operações requeridas. Por exemplo, podemos projetar um registrador que possibilite transferências para e de um bus, complementação, incrementação, deslocamento, restabelecimento, operação lógica AND, operação lógica OR etc., ou podemos decidir usar inúmeros registradores, cada um possibilitando individualmente menor número de operações, porém capazes, entre eles, de efetuar todas as operações exigidas. No primeiro caso, um menor número de transferências de registrador a registrador será requerido. No último caso, quanto mais numerosos os registradores, mais simples eles serão individualmente, e o sistema pode prestar-se mais facilmente à modificação. (Tais registradores, conforme havíamos descrito, são chamados *registradores operativos*, em contraste com os *registradores de armazenamento*, que servem apenas para armazenar uma palavra.) Alternativamente, podemos decidir usar registradores de armazenamento em vez de registradores operativos utilizando circuitos combinacionais que respondem a comandos, como a ULA descrita na Seção 5.13. Em qualquer caso, o começo do projeto de um sistema digital consiste em uma decisão (pelo menos numa tentativa) das peças componentes de hardware a serem usadas, de que comandos os componentes devem ser capazes de responder e de como os componentes devem ser interligados. Estes aspectos característicos do sistema são denominados *arquitetura do*

sistema. Uma vez que a arquitetura tenha sido estabelecida, podemos construir um *controlador*, que fornecerá comandos na seqüência correta às linhas de controle dos componentes, conforme requerido, para fazer com que o sistema execute sua função.

Exceto nos casos muito mais simples, não há procedimento de projeto que conduza à melhor arquitetura. As arquiteturas são selecionadas pelos projetistas na base da experiência e no simples bom senso. Dois projetistas podem desenvolver dois sistemas individuais de arquiteturas diferentes *que desempenham a mesma função. Pode ser impossível julgar um projeto superior ao outro sem* que haja ambigüidade. Um projeto pode ter méritos em certas direções; o outro pode ter vantagens em outras áreas. A situação é um tanto análoga à que prevalece nos negócios (ou em outras instituições humanas). Consideremos, por exemplo, inúmeras companhias industriais que fabricam bugigangas, todas tendo quase o mesmo rendimento bruto anual. Todas elas necessitam de sistemas de escrituração contábil. O resultado final em cada caso deve ser o mesmo. Elas devem todas guardar os registros e fazer cálculos de modo que possam arrecadar suas contas a receber, saldar suas contas a pagar, pagar seus empregados por hora de trabalho executado etc. Mas o grupamento de funções em departamentos pode bem diferir de companhia para companhia, e dentro de um departamento a divisão de trabalho entre os empregados também pode ser diferente. Ao comparar duas companhias, é muito improvável que haja uma correspondência precisa um a um entre departamentos ou entre os serviços de empregados individuais.

Voltemos agora ao sistema simples cuja arquitetura é dada na Fig. 8.4-1. Aqui temos em

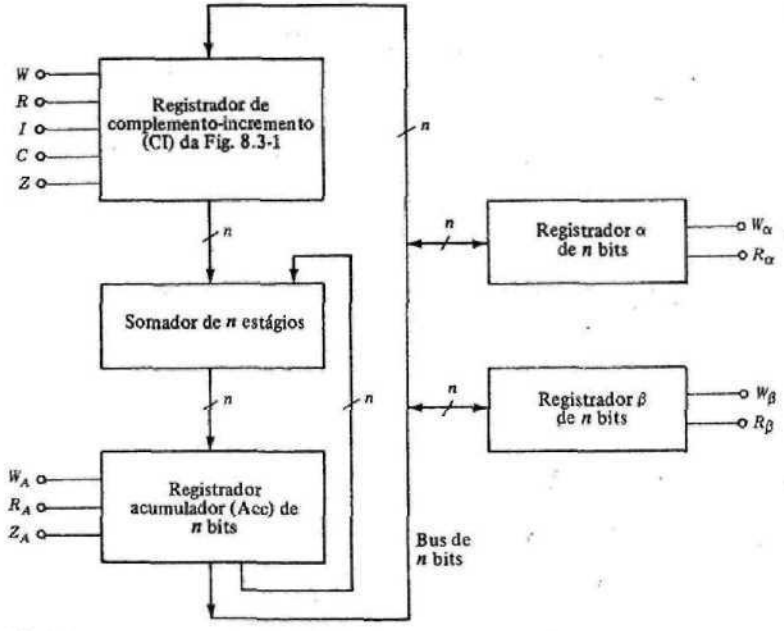


Fig. 8.4-1 Uma arquitetura para combinar aritmeticamente o conteúdo dos registradores α e β.

mente um sistema para calcular o valor da soma ou diferença aritméticas de dois números binários de n bits. Especificamente, queremos calcular as somas e diferenças $\alpha + \beta, \alpha - \beta, -\alpha + \beta$ e $-\alpha - \beta$ dos números nos registradores α e β. O mecanismo pelo qual estes números são introduzidos nos registradores α e β não está indicado na figura. Admitamos que estes números sejam introduzidos assincronamente através dos terminais de *set-direto* e *reset-direto* dos flip-flops individuais que constituem o registrador. Se um flip-flop tiver que ter um 0 introduzido nele, seu terminal de *reset-direto* é trazido brevemente para lógica 1. Se um 1 deve ser introduzido, o terminal de *set-direto* é trazido brevemente para lógica 1. Quando um número tiver sido introduzido desta maneira, todos os terminais *diretos* são permitidos retornar à lógica 0, de modo que o flip-flop possa agora responder à transição de gatilho de relógio conforme solicitado pelos terminais de controle. A quantidade $\pm\alpha \pm \beta$ é finalmente preparada no registrador acumulador e também no registrador α ou no β.

Todos os registradores e o somador acomodam n bits. O registrador (CI) de complementa-incrementa está conectado ao somador, o somador está conectado ao acumulador e o acumulador está conectado de volta ao somador, todos por conexões de n linhas. Uma vez que estas conexões de n bits são *dedicadas*, isto é, cada uma serve a uma única função de transferência, elas não são buses. Há também um bus de n bits para e a partir do qual podemos transferir os conteúdos do registrador α e do registrador β. Esta viabilidade de transferência de duas vias é indicada pela seta bidirecional. Uma forma-de-onda de sincronização comum, não mostrada, é aplicada a todos os registradores. Quando $W_\alpha = 1$, na borda de gatilho de relógio uma palavra é transferida do bus para o registrador α; assim, uma palavra é "escrita" no registrador. Quando $R_\alpha = 1$, uma palavra é "lida" do registrador no bus. Comentários similares se aplicam à resposta do registrador β a W_β e R_β. Quando $W_A = 1$, a saída do somador será registrada no acumulador, e quando $R_A = 1$, o conteúdo do acumulador será colocado no bus. O conteúdo do acumulador está permanentemente conectado à conexão de n bits que retorna ao somador. Não há nenhum controle sobre esta conexão. O acumulador é limpo (levado a zero) na transição de gatilho de relógio se $Z_A = 1$.

Consideremos como o sistema da Fig. 8.4-1 pode ser orientado para achar $\alpha + \beta$ e para armazenar o resultado no registrador α. Inúmeras operações elementares, chamadas *microoperações*, são necessárias, cada uma requerendo um ciclo de relógio. A seqüência está mostrada na Tab. 8.4-1.

Notar que nesta seqüência de operações elementares tanto α quanto β foram passados através de CI. Naturalmente, não há necessidade de se fazer assim, e o tempo de dois ciclos de relógio foi desperdiçado desta maneira. Ainda, por causa da arquitetura de nosso sistema, não tivemos nenhuma escolha. Se tivéssemos partido de uma arquitetura mais elaborada, que fornecesse acesso direto ao somador, teríamos sido capazes de desviar o registrador CI. Notar que não há necessidade de limpar o registrador CI.

Projetaremos agora um controlador que tornará disponíveis os níveis lógicos requeridos para pôr em seqüência nossa máquina através de suas etapas. É evidente que o controlador deve ser uma máquina seqüencial com seis estados, uma vez que têm que ser efetuadas seis operações separadas. No entanto, no caso presente encontramos uma situação que não surgiu no Cap. 7, quando projetamos detectores de seqüências. Num detector de seqüências estávamos satisfeitos em permitir que a máquina funcionasse continuamente desde que considerássemos que a seqüência de entrada também era contínua. No caso presente queremos ser capazes de interromper ou parar a máquina após a última etapa na seqüência. Do contrário, a máquina continuará de um lado para outro de sua seqüência de operações. Após uma volta, o novo registro no registrador α será $\alpha' = \alpha + \beta$. Após a volta seguinte, teremos $\alpha'' = \alpha' + \beta$ etc. E, se usarmos um relógio com

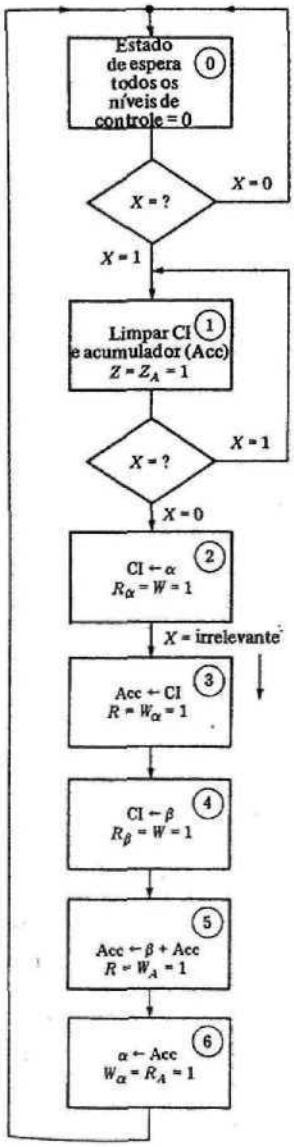
Tabela 8.4-1

Ciclo de relógio	Linhas de controle a serem postas em lógica 1	Comentário
1	Z_A	Registrador do acumulador limpo de qualquer número que possa ter restado de uma operação anterior
2	R_α, W	*Ler α no bus e escrever palavra no bus dentro do registrador* CI
3	R, W_A	Conteúdo de CI passado pelo somador (outra entrada do somador é zero) e registrado no acumulador
4	R_β, W	Conteúdo de β transferido para CI
5	R, W_A	β acrescentado ao acumulador
6	R_A, W_α	Conteúdo do acumulador transferido para o registrador α

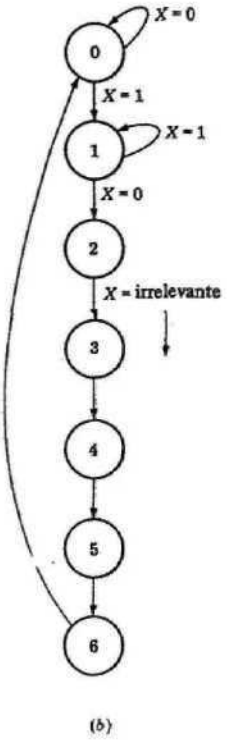
uma freqüência comumente usada em sistemas eletrônicos digitais (100 kHz ou mais alta), não teremos mesmo tempo de ler o registrador α antes que ele mude.

Portanto, aos seis estados correspondentes às seis micro operações acrescentemos um sétimo, em que o controlador irá parar e *esperar* quando a seqüência tiver sido completada. Este estado extra nos dará tempo para ler o resultado de nossa computação e para colocar novos números nos registradores α e β. Depois consideramos que, para tirar o controlador fora do estado de espera, há uma entrada X no controlador proveniente de alguma fonte externa. Esta entrada pode ser fornecida por uma chave de botão de calcar. Quando o botão for calcado, $X = 1$; do contrário, $X = 0$. Quando $X = 0$, o controlador, tendo iniciado sua seqüência, continuará até que ele alcance o estado de *espera*. A seqüência seguinte não começará até que o botão tenha sido calcado de modo que X novamente se torne $X = 1$. Conforme vemos agora, no entanto, não resolvemos nosso problema completamente. Suponhamos que, tendo calcado o botão, deixamos de liberá-lo antes que a seqüência tenha sido completada. Então o controlador pode atravessar diversos ciclos antes que finalmente chegue a se restabelecer no estado de *espera* (a uma taxa de relógio de 100 kHz, a seqüência de seis etapas é concluída em apenas 60 μs). Em conjunto, somos levados a um controlador descrito pelo fluxograma da Fig. 8.4-2*a*.

Há sete estados que numeramos de 0 a 6. O número do estado é dado no círculo no canto superior direito dos quadrículos de estado na Fig. 8.4-2*a*. No estado 0, o estado de espera, todas as entradas de controle na unidade aritmética da Fig. 8.4-1 estão em lógica 0. Enquanto $X = 0$, o controlador permanece no estado 0. Quando o botão for calcado e $X = 1$, o controlador vai para o estado 1, enquanto $Z_A = 1$, de modo que o registrador do acumulador está limpo. O controlador não sai do estado 1 até que o botão tenha sido liberado. Na liberação do botão o controlador vai para o estado 2, onde $R_\alpha = W = 1$, de modo que o conteúdo do registrador α se deslocará para o registrador CI. Depois do estado 2, o progresso para o estado 3 e o restante da seqüência prossegue independentemente de se $X = 0$ ou $X = 1$. O controlador finalmente termina no estado 0 e permanece ali até que o botão seja calcado novamente. O diagrama de estados é dado na Fig. 8.4-2*b*.



(a)



(b)

Fig. 8.4-2 (*a*) O fluxograma de um controlador que usa o sistema da Fig. 8.4-1 através das microoperações requeridas para somar os conteúdos de R_α e R_β e armazenar a soma em R_α. (*b*) O diagrama de estados.

8.5 IMPLEMENTAÇÃO DO CONTROLADOR

Uma vez que o controlador da seção anterior tem sete estados, necessitamos de um circuito seqüencial de três flip-flops. Decidimos arbitrariamente usar flip-flops do tipo D. Uma tabela de transição é dada na Fig. 8.5-1*a*. Na primeira coluna, para identificação de estados, listamos os estados por sua numeração na Fig. 8.4-2*a*. Arbitrariamente fizemos a atribuição de estado que está indicada na segunda coluna. Fizemos simplesmente, para cada estado, uma atribuição $Q_2Q_1Q_0$ que, quando lida como um número binário, é igual ao número de identificação decimal. Os estados seguintes, para $X = 0$ e $X = 1$, são tomados diretamente do fluxograma ou diagrama de estados da Fig. 8.4-2.

O controlador é uma máquina Moore. As saídas são totalmente dependentes do estado do controlador, isto é, dos níveis lógicos em Q_2, Q_1 e Q_0. A única entrada X não tem influência direta sobre as saídas, uma vez que X serve apenas para determinar se o controlador está em seqüência através de seus estados ou se é mantido em repouso. As saídas listadas nas colunas restantes podem também ser lidas diretamente do fluxograma.

Na Fig. 8.5-1*b* construímos mapas K para as excitações D_2, D_1, D_0 dos três flip-flops. A leitura de cada mapa é dada diretamente sob o mapa. Na Fig. 8.5-1*c* desenhamos o diagrama de circuito daquela parte do controlador que determina a ação de pôr em seqüência de estado a estado. (A parte do controlador que gera as muitas saídas requeridas não é mostrada.) As três estruturas de porta que geram as três excitações D_2, D_1 e D_0 não são mostradas explicitamente, mas estão indicadas pelas caixas (quadrículos) designadas por "lógica". As entradas nas caixas lógicas são Q_2, Q_1 e Q_0 e a variável, X. As saídas das caixas lógicas são expressas na figura como funções booleanas. Naturalmente, se escolhermos, podemos substituir as três caixas lógicas usando portas individuais com uma ROM. As entradas na ROM, isto é, o endereço, seriam então $Q_2Q_1Q_0$ e X e as saídas, isto é, a palavra lida da ROM, seriam D_2, D_1 e D_0. Finalmente, a parte de geração de seqüências do controlador (Fig. 8.5-1*c*) é representada como um bloco na Fig. 8.5-1*d*. Os únicos terminais explicitamente em evidência são o relógio, a entrada X e os terminais dos flip-flops requeridos para implementar o decodificador que irá gerar as saídas.

O decodificador de geração das saídas é dado na Fig. 8.5-2. Como exemplo, notamos da tabela da Fig. 8.5-1*a* que Z e Z_A estão em lógica 1 quando e somente quando $Q_2 = 0$, $Q_1 = 0$ e $Q_1 = 1$. Por conseguinte, um único AND com entradas conforme indicado gera Z e Z_A. Notamos que R e W_A estão em lógica 1 quando o estado for $Q_2Q_1Q_0 = 011$ e também quando $Q_2Q_1Q_0 = 101$. Por conseguinte, aqui duas portas AND e uma porta OR são requeridas conforme mostrado. O restante do decodificador é lido de maneira idêntica à da Fig. 8.5-1*a*, que, no que concerne às saídas, é uma tabela verdade que mostra a relação das saídas com Q_2, Q_1 e Q_0. As Figs. 8.5-1*c* e 8.5-2, juntas, constituem o controlador completo. Quando o controlador é usado em conjunto com a arquitetura mostrada na Fig. 8.4-1, temos uma máquina que somará dois números binários.

As formas-de-onda da máquina de adição estão mostradas na Fig. 8.5-3. Admitimos aqui que os flip-flops das seqüências respondem à transição (borda) crescente negativa da forma-de-onda de relógio. A forma-de-onda de X representa a operação da chave. Em um ponto arbitrário, num ciclo de relógio, a chave está fechada e X vai para lógica 1. A chave permanece fechada durante um número indeterminado de ciclos de relógio e depois abre, novamente num ponto arbitrário no ciclo de relógio. As formas-de-onda mostradas são aquelas na saída do decodificador e também a "forma-de-onda" de $\bar{Q}_2\bar{Q}_1\bar{Q}_0$. Esta última não é realmente gerada em qualquer lugar no sistema e foi incluída de modo que será evidente quando a seqüência estiver no estado de espera.

Estado presente	Estado presente	Estado seguinte		Saídas								
	$Q_2Q_1Q_0$	$X=0$	$X=1$	Z	Z_A	R_α	W	R	W_A	R_β	W_α	R_A
0	000	000	001	0	0	0	0	0	0	0	0	0
1	001	010	001	1	1	0	0	0	0	0	0	0
2	010	011	011	0	0	1	1	0	0	0	0	0
3	011	100	100	0	0	0	0	1	1	0	0	0
4	100	101	101	0	0	0	1	0	0	1	0	0
5	101	110	110	0	0	0	0	1	1	0	0	0
6	110	000	000	0	0	0	0	0	0	0	1	1

(a)

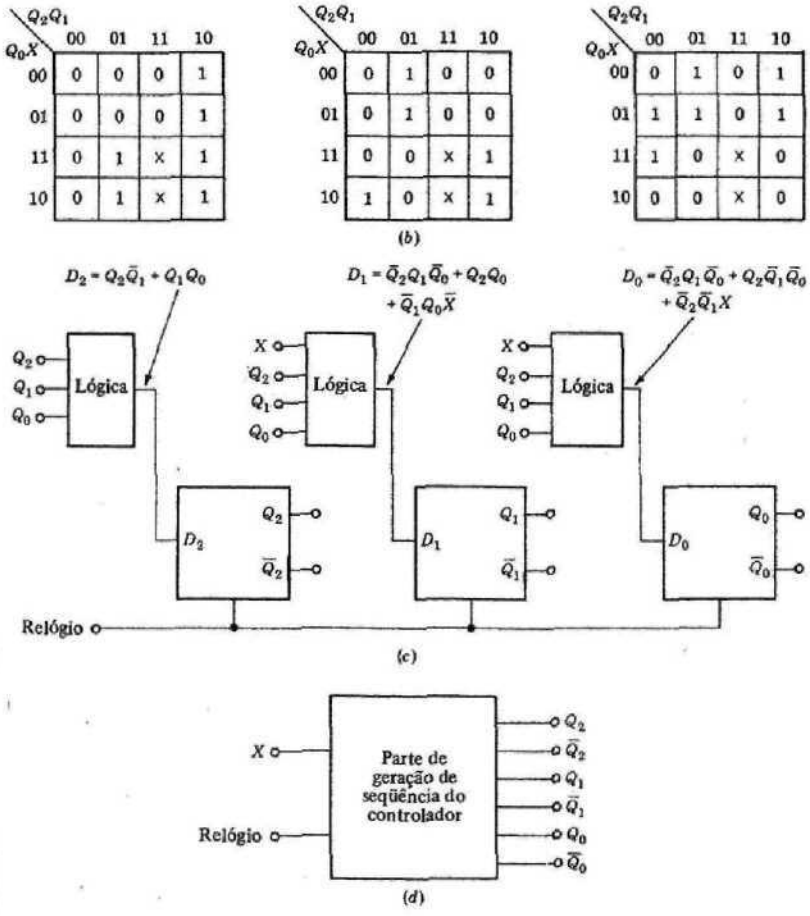


Fig. 8.5-1 O projeto do controlador cujos fluxograma e tabela de estado são dados na Fig. 8.4-2: (*a*) tabela de transição; (*b*) mapas K para as excitações de entrada dos flip-flops; (*c*) diagrama lógico da parte de geração de seqüência do controlador; e (*d*) símbolo do sistema lógico em (*c*).

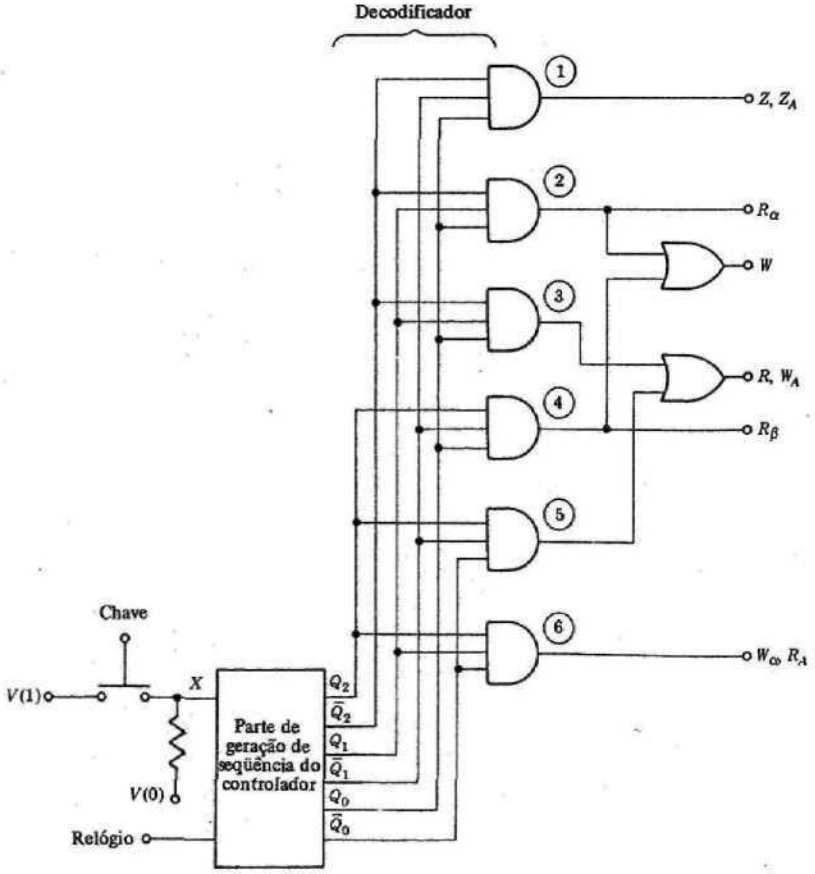


Fig. 8.5-2 O decodificador a ser usado com o gerador de seqüências da Fig. 8.5-1.

As formas-de-onda começam num instante em que $X = 0$ (chave ainda não calcada) e o sistema está no estado de espera. No instante da primeira transição de relógio crescente negativa após a chave ter sido calcada, o sistema sai do estado de espera (estado 0) e vai para o estado seguinte (estado 1), em que os estados seguintes Z e Z_A são 1. Este estado 1 persiste até que o tempo da próxima transição de relógio negativa depois de X volte a $X = 0$. Além das formas-de-onda de habilitação Z e Z_A, todas as outras formas-de-onda crescem até o nível de habilitação de um único ciclo de relógio em um instante. Algumas formas-de-onda fazem esta excursão para a lógica 1 apenas uma vez quando as seqüências completam um ciclo através de seus estados; outras fazem-na duas vezes.

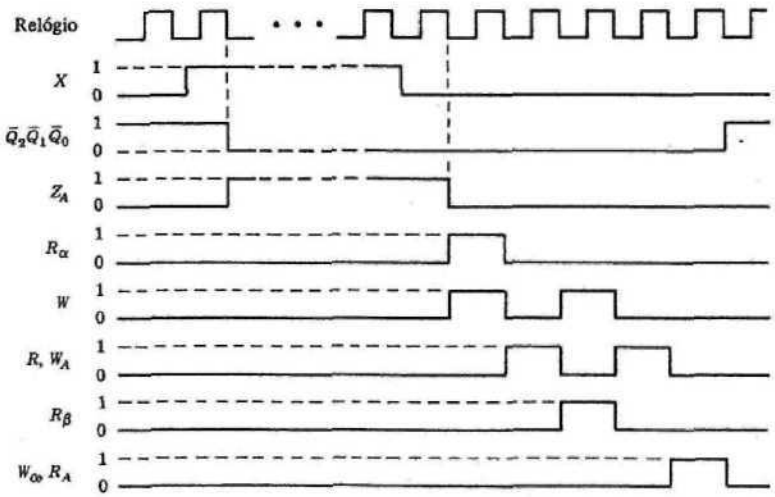


Fig. 8.5-3 Formas-de-onda de relógio e formas-de-onda geradas nas saídas do decodificador da Fig. 8.5-2.

8.6 O CONTROLADOR DO REGISTRADOR DE DESLOCAMENTOS

O controlador das seções anteriores é projetado para requerer um número mínimo de flip-flops. Este critério de projeto de minimização do número de estados, de modo que o número de flip-flops seja mínimo, também foi observado no Cap. 7, onde projetamos detectores de seqüências. No Cap. 7 notamos, no entanto, que havia um método alternativo de projeto usando registradores de deslocamento. O projeto do registrador de deslocamentos usa mais flip-flops, mas geralmente menos lógica, isto é, menor número de portas para gerar funções lógicas. E enquanto, no final, um projeto de registrador de deslocamentos possa não ser tão econômico em hardware quanto um projeto de mínimo de flip-flops, o projeto do registrador de deslocamentos tem pelo menos um grande mérito, que é o de ser mais ordenado e sistematizado no sentido de que podemos determinar fácil e precisamente o que faz cada flip-flop. Tal não é geralmente o caso num projeto com o mínimo de número de estados.

Estas considerações levam-nos a examinar um projeto de registrador de deslocamentos no caso presente. Observemos primeiro que, enquanto o controlador projetado nas seções precedentes usa apenas três flip-flops, ele usa muita lógica nas caixas lógicas da Fig. 8.5-1 e no decodificador da Fig. 8.5-2. Segundo, notemos que as formas-de-onda da Fig. 8.5-3 são rememorativas das formas-de-onda encontradas geralmente nos circuitos registradores de deslocamentos. É, em princípio pelo menos, muito óbvio como usar um registrador de deslocamentos para construir uma seqüência para nosso controlador de máquina de somar. Temos apenas que construir um contador em anel do registrador de deslocamento de seis estados e providenciar para que, no início, o primeiro flip-flop esteja ajustado (set) e todos os outros reajustados (reset). Depois, com cada ciclo de relógio, a condição de set ($Q = 1$) se moverá ao longo do registrador e, ciclo a ciclo,

teremos disponíveis, em sucessão nas saídas dos flip-flops, as formas-de-onda de habilitação de um ciclo de relógio que requeríamos. Em verdade, num seqüencializador do registrador de deslocamentos, o problema de interromper a seqüência para tornar um estado de espera disponível é facilmente resolvido. Necessitamos simplesmente evitar a conexão do último flip-flop de volta ao primeiro para completar o anel. Então, quando a condição de set for deslocada para fora do último flip-flop, a seqüência se interrompe automaticamente.

Há, no entanto, um detalhe a ser tratado na seqüência do registrador de deslocamentos que

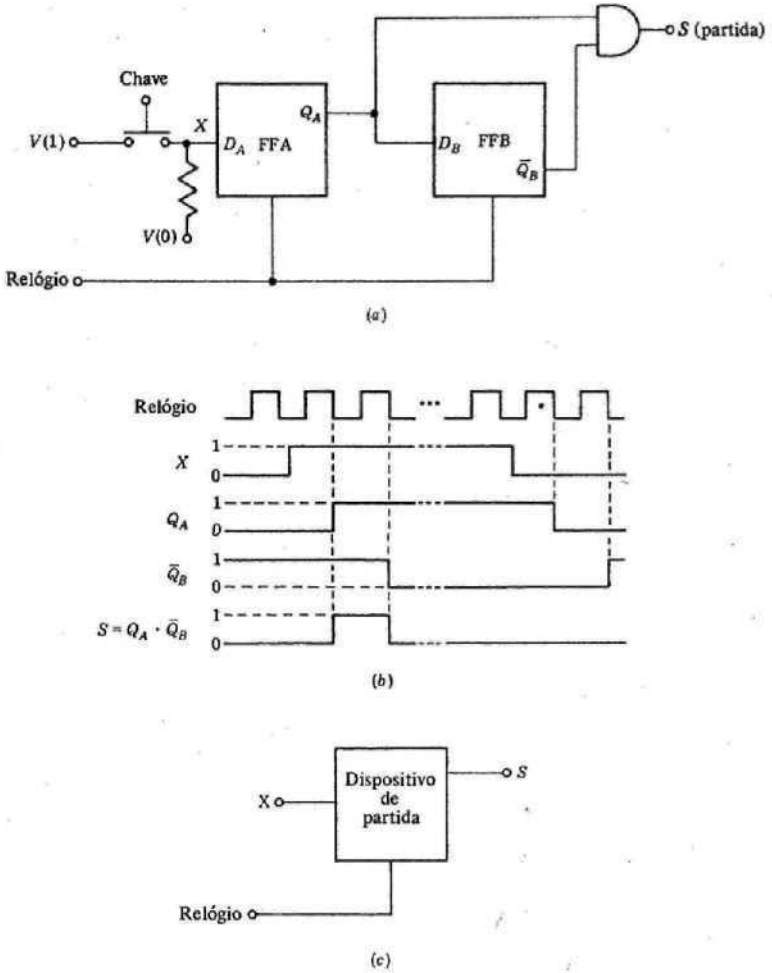


Fig. 8.6-1 (*a*) Uma estrutura lógica que, no fechamento da chave, gera uma transição de S para o nível lógico 1 que persiste durante um ciclo de relógio. (*b*) Formas-de-onda. (*c*) Símbolo do dispositivo de partida.

não surgiu na seqüência com mínimo de estados. Ali não tínhamos nenhum interesse a respeito do estado em que a seqüência podia estar quando fosse aplicada energia de alimentação para acionar ou ligar o sistema. Não importa qual o estado inicial, a seqüência, depois de alguns ciclos de relógio, achar-se-á no estado de espera. Na seqüência do registrador de deslocamentos, por outro lado, necessitamos de um modo de assegurar que no início um e somente um registrador de deslocamento está na condição de set em qualquer instante e que o flip-flop assim ajustado é o primeiro flip-flop na cadeia de flip-flops. Um circuito que nos permitirá preparar e iniciar (*start*) adequadamente uma seqüência de registrador de deslocamento é mostrado na Fig. 8.6-1*a*. Ele envolve uma chave e usa a mesma forma-de-onda de relógio que o seqüencializador. As formas-de-onda são dadas na Fig. 8.6-1 *b* na hipótese de que os flip-flops do tipo D respondem à transição crescente negativa da forma-de-onda de relógio. Quando a chave estiver aberta, $X = 0$ e $Q_A = 0$ enquanto $\overline{Q}_B = 1$. Em um tempo arbitrário num ciclo de relógio, a chave é calcada e X torna-se $X = 1$. Na próxima transição negativa de relógio, Q_A torna-se $Q_A = 1$, e um ciclo de relógio, mais tarde, $\overline{Q}_B$ torna-se $\overline{Q}_B = 0$. Durante exatamente um ciclo de relógio, $S = Q_A \cdot \overline{Q}_B = 1$. A abertura da chave mais tarde, não importa quando, não tem nenhum efeito sobre S. Em conjunto, então, o fechamento e a subseqüente abertura da chave gera lógica 1 em S durante um ciclo, independentemente de quanto tempo a chave possa permanecer fechada. O circuito na Fig. 8.6-1*a* será representado pelo símbolo na Fig. 8.6-1*c*.

O controlador inteiro com esta unidade de dispositivo de partida é mostrado na Fig. 8.6-2. Quando a chave está aberta, $X = 0$ e $S = 0$. Depois de alguns ciclos de relógio o registrador inteiro de deslocamentos terá voltado a zero (estará limpo) e todas as saídas estarão em lógica 0. O fechamento da chave ajustará $S = 1$ durante um ciclo de relógio, e esta condição de set propagar-se-á para baixo no registrador de deslocamentos. É bem verdadeiro que usamos aqui muito mais flip-flops que no controlador de mínimo de estados, e em conseqüência há muitos estados não utilizados; mas observemos que muito menos lógica é usada no presente caso. Observemos também que aqui há uma correspondência de um para um entre o flip-flop e os estados através dos quais o controlador completa o ciclo. E finalmente observemos quanto mais simples seria reparar defeitos no presente sistema.

8.7 RESPOSTA CONDICIONAL DE CONTROLADORES

O controlador das seções precedentes seguia uma seqüência fixa de produção de ciclos através dos estados, fazendo disponível uma seqüência fixa de níveis lógicos de habilitação para efetuar uma seqüência fixa de microoperação. No caso mais geral queremos um controlador para acompanhar diferentes seqüências sob diferentes circunstâncias. Para orientar o controlador, haverá disponível para ele inúmeras entradas X_0, X_1 etc. Em alguns casos, uma entrada, por exemplo X_0, virá de uma fonte externa tanto ao controlador, quanto ao processador que está sendo controlado, e o nível lógico de X_0 será determinado pela intervenção humana. Por exemplo, em nosso sistema acima, o resultado final da adição é, como uma última etapa, transferido de volta ao registrador R_α, mas podemos querer ser capazes de escolher se o resultado será transferido para R_α ou para R_β. E podemos depois arranjar, por exemplo, para que, quando pusermos $X_0 = 0$, a transferência seja para R_α, e quando pusermos $X_0 = 1$, a transferência seja para R_β. Em outros casos, uma entrada, por exemplo X_1, pode ser determinada por uma conseqüência ou resultado do processamento. Por exemplo, suponhamos que queremos o resultado final transferido para R_α, se o resultado revelar-se um número positivo, e transferido para R_β se revelar um número negativo. Supo-

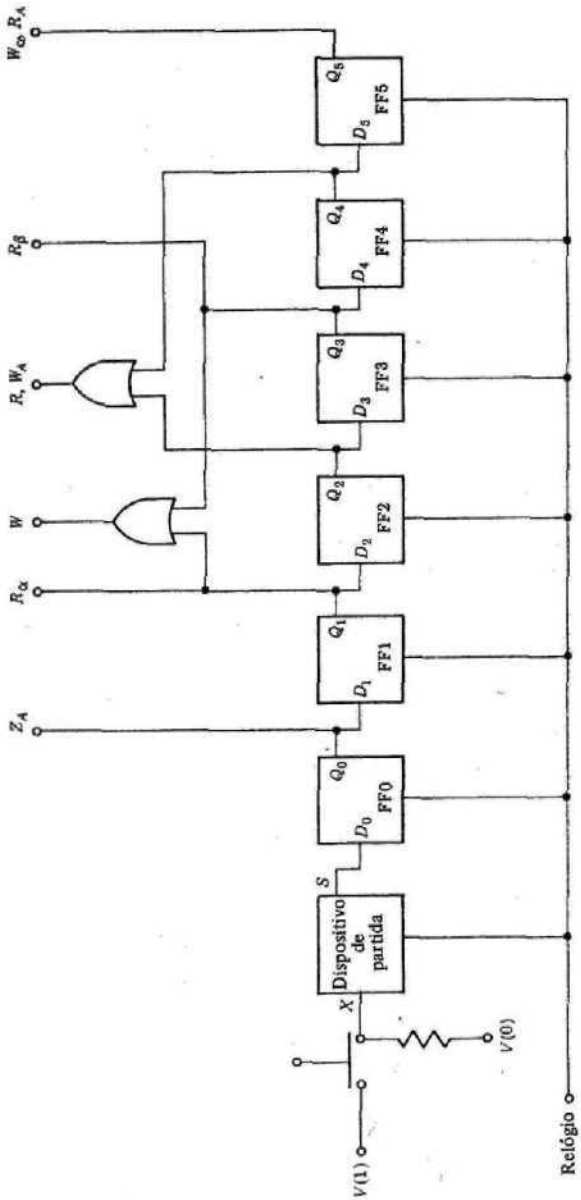


Fig. 8.6-2 Um controlador de registrador de deslocamento que pode servir como uma substituição para o controlador com "mínimo de estados" das Figs. 8.5-1 e 8.5-2.

nhamos, ainda, que números negativos sejam representados em forma de complemento de dois de modo que o bit de sinal seja 0 ou 1, conforme o número for positivo ou negativo. Então a entrada X_1 do controlador seria o bit de sinal no registrador do acumulador, e quereríamos arranjar uma transferência para R_α ou R_β dependendo de X_1 ser 0 ou 1. Estas entradas, X_0, X_1 etc., sejam de uma fonte inteiramente externa, sejam de um resultado do processamento, são precisamente as entradas X_0, X_1 etc. na Fig. 7.5-6, que representava um circuito seqüencial generalizado. As entradas geradas são chamadas *entradas de realimentação.*

Num projeto de controlador de mínimo de estados, sempre preparamos uma tabela de estados. Numa tal tabela de estados há uma coluna de estados seguintes correspondente a cada combinação possível de entradas. Se houver uma entrada, há duas de tais colunas, uma para $X = 0$ e outra para $X = 1$. Se houver duas entradas, há quatro colunas; três entradas, oito colunas, e assim por diante. Depois de ter sido deduzida a tabela de estados, e após havermos decidido a respeito do tipo de flip-flop a ser usado e termos feito uma atribuição de estado, o restante do projeto que leva ao controlador é bastante automático. Por outro lado, são requeridos alguns comentários para esclarecer como as entradas X_0, X_1 etc., muitas vezes chamadas *entradas condicionais* ou *modificadores*, são manipuladas nos controladores de registradores de deslocamento.

Suponhamos, por exemplo, que queremos ser realmente capazes de selecionar se R_α ou R_β deve ser o repositório final de nosso resultado e que providenciamos para este propósito um *seletor de registrador final* (FRS) de entrada de controle. Então podemos ver na Fig. 8.7-1 como esta entrada de controle é feita eficiente. Todo o controlador da Fig. 8.6-5 permanece inalterado, exceto na saída do último flip-flop na cadeia, uma vez que somente a última microoperação é afetada. A última saída é modificada conforme mostrado pelo acréscimo de duas portas AND. Se FRS = 1, W_α tornar-se-á $W_\alpha = 1$ durante o último ciclo da seqüência e o resultado será escrito em R_α. Se RFS = 0, R_β será escolhido.

Suponhamos, por outro lado, que queremos selecionar o registrador R_α ou R_β não na base da entrada externa, mas na base do sinal do resultado. Então na Fig. 8.7-1, substituiríamos RFS pela conexão ao flip-flop do acumulador que guarda ou retém o bit de sinal. É viável fazer assim porque, durante o intervalo em que deve ser feita a transferência final para R_α ou R_β, o acumulador ainda não está limpo e o bit de sinal está disponível.

Consideremos a seguir um caso em que há uma entrada f de realimentação no controlador, e suponhamos que é exigido que o valor lógico de f durante o k-ésimo ciclo de relógio tem que selecionar a microoperação a ser executada durante o $(k+1)$-ésimo ciclo de relógio. Um modo de se conseguir esta finalidade consiste em acrescentar um flip-flop (e não parte da cadeia do registrador de deslocamentos), no qual armazenamos o valor de f de modo que ele será encontrado quando necessário. Então, como na Fig. 8.7-1, executaríamos uma ou outra microoperação

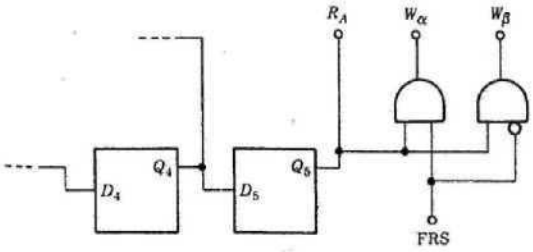


Fig. 8.7-1 Uma modificação do controlador da Fig. 8.6-2 que permite uma resposta do controlador determinada pelo nível lógico de um FRS (seletor do registrador final) de entrada.

trador de complementar-incrementar (CI), que realmente não serve a nenhum propósito na formação da soma. Se quisermos formar $R_\alpha - R_\beta$, temos apenas que inverter o sinal do conteúdo de R_β, formando, desta maneira, $-R_\beta$, e depois adicionar $R_\alpha + (-R_\beta)$. Admitimos, como antes, que números negativos devem ser representados em forma de complemento de dois. O complemento de dois é gerado complementando-se cada bit individual do número e depois incrementando-se o número de 1. Daí, em conjunto, para formar $R_\alpha - R_\beta$, primeiro transferimos R_α através de CI para o registrador do acumulador como antes; depois transferimos R_β para CI, onde, então, primeiro *complementamos* e depois *incrementamos* antes da transferência de CI para o acumulador.

Para realizar a subtração, necessitamos modificar o controlador para proporcionar duas microoperações adicionais, que por sua vez exigem dois estados adicionais. Se projetarmos um controlador com mínimo de estados, a tabela de estados terá dois estados adicionais. Uma vez que nosso controlador original tinha sete estados, o novo controlador terá nove estados e quatro flip-flops serão necessários. Além disto, teremos que modificar a lógica que usa a seqüência através de seus passos, e teremos que acrescentar lógica ao decodificador. Se projetarmos um controlador do registrador de deslocamentos, teremos que acrescentar dois flip-flops à cadeia do registrador de deslocamentos. Os detalhes de cada um destes projetos é deixado como exercício ao estudante.

A seguir, suponhamos que queremos formar $-R_\alpha + R_\beta$. Partindo então novamente do controlador original da Fig. 8.5-2 ou Fig. 8.6-2, acrescentaríamos duas microoperações, desta vez para formar o negativo do número em R_α. Se quiséssemos $-R_\alpha - R_\beta$, teríamos que acrescentar quatro microoperações.

Poderíamos também decidir que quatro diferentes controladores para realizar as quatro operações $R_\alpha + R_\beta$, $R_\alpha - R_\beta$, $-R_\alpha + R_\beta$ e $-R_\alpha - R_\beta$ não são realmente exigidos. Em vez disto poderíamos projetar um simples controlador que pudesse efetuar qualquer uma das quatro operações dependendo da *instrução* que déssemos a ele. Um tal controlador é dado na Fig. 8.8-1. Com o propósito de guardar a *instrução*, acrescentamos um *registrador de instrução de dois bits*. Consideremos aqui que a instrução está colocada no registrador de instruções pela manipulação manual usando chaves, exatamente como registramos os números a serem combinados, isto é, os operandos, nos registradores R_α e R_β. Ao controlador original da Fig. 8.6-2 também acrescentamos quatro estados adicionais. Nestes estados os números no registrador CI serão complementados e incrementados, conforme requerido, para trocar o sinal dos operandos. A complementação no registrador CI é efetuada quando o terminal C na Fig. 8.8-1 (conectado a C na Fig. 8.4-1) vai para lógica 1. A incrementação é efetuada quando o terminal I vai para lógica 1. Quando as linhas de instruções de bit são $C_0 = C_1 = 0$, todas as portas AND na Fig. 8.8-1 estão desativadas, C e I estão sempre em lógica 0, e o resultado líquido é que a operação total efetuada é para gerar a soma $R_\alpha + R_\beta$. A operação total para todas as quatro instruções possíveis é dada na Fig. 8.8-2.

Instrução		Operação total
C_1	C_0	
0	0	$R_\alpha + R_\beta$
0	1	$-R_\alpha + R_\beta$
1	0	$R_\alpha - R_\beta$
1	1	$-R_\alpha - R_\beta$

Fig. 8.8-2 O código C_1C_0 de instruções para as quatro instruções do controlador da Fig. 8.8-1.

8.9 UM COMPUTADOR SIMPLES

O *controlador* da Fig. 8.8-1, operando em conjunto com *registradores de armazenamento* e *registradores controláveis* organizados na arquitetura da Fig. 8.4-1, nos permitirá combinar dois números aritmeticamente por adição e subtração. Para usar a máquina, colocaríamos números em R_α e R_β, presumivelmente de forma manual usando chaves, uma vez que não tomamos outras providências. Depois, manualmente colocaríamos uma instrução no registrador de instruções (IR). Depois disto, tudo prossegue automaticamente. Necessitamos somente calcar o botão de partida e, depois de certo tempo, acharemos nosso resultado em R_α.

Suponhamos, agora, que queremos tornar nossa máquina mais elaborada. Primeiro, queremos ser capazes de lidar com mais do que dois números, ou operandos. Assim, podemos efetuar uma adição de apenas dois operandos, mas queremos ser capazes de selecionar a partir de um grande número deles, ou seja, dispondo de um grande número de operandos, combinar muitos deles. Uma modificação óbvia que é depois exigida consiste em substituir os registradores R_α e R_β por um grande conjunto de registradores. Este grande conjunto de registradores é, naturalmente, uma memória, como descrita no Cap. 6. Se quisermos ser capazes de mudar operandos facilmente, é requerida uma RAM'

No controlador da Fig. 8.8-1 inúmeras etapas da seqüência tratam do operando armazenado em R_α, e inúmeras etapas tratam do operando em R_β. É evidente que a obediência a este padrão levaria a uma seqüência muito longa se muitos operandos estivessem envolvidos. Cada novo operando acrescenta etapas à seqüência. Reconhecemos, no entanto, que cada operando é realmente submetido às mesmas microoperações. O operando é transferido de um registrador de armazenamento (ou melhor, de um local na memória) para o registrador CI e daí através do somador para o acumulador. Se necessitarmos mudar o sinal do operando, complementamos e incrementamos no registrador CI. Do contrário, o registrador CI não faz nada. Esta seqüência de microoperações é repetida sem cessar para cada operando. Daí parece que realmente podemos dispor de um controlador que tem uma seqüência curta, uma seqüência que processará apenas um operando. No entanto, com este controlador de seqüências, necessitaríamos de algum mecanismo para ajustar a *instrução ao controlador, isto é, mudar o sinal do operando ou não mudar o sinal, quando cada* novo operando for processado. Um modo de ajustar a instrução é simplesmente interromper a seqüência de controle depois que ele processa cada operando e em seguida mudar manualmente a instrução antes de permitir que a seqüência processe o operando seguinte. Mas, já que temos uma memória, podemos armazenar nela a informação concernente a como cada operando deve ser processado e a operação inteira pode ser feita de forma automática. Com estas considerações em mente, voltemos agora à Fig. 8.9-1; ela expõe um sistema que nos permitirá (com alguma intervenção manual) combinar aritmeticamente um grande número de operandos automaticamente.

O sistema tem uma memória RAM. Para sermos explícitos, admitamos uma memória com 64 palavras, cada palavra tendo oito bits. Um local da memória é endereçado por um endereço de seis bits ($2^6 = 64$). A memória tem uma entrada de HABILITA (*ENABLE*) e uma entrada de *leitura/escrita* (*read/write*). Quando *HABILITA* (*ENABLE*) = 1, a memória está conectada ao bus (oito bits), e quando *HABILITA* (*ENABLE*) = 0, o bus está isolado da memória. Quando *HABILITA* (*ENABLE*) = 1, a memória lerá uma palavra sobre o bus ou escreverá uma palavra na memória, dependendo de *leitura/escrita* ser 1 ou 0. O local da memória a partir do qual uma palavra é lida ou no qual uma palavra é escrita é determinado pelos seis bits de endereço. A seta de ponta dupla que conecta a memória ao bus indica que a transferência de informações entre o bus e a memória é bidirecional.

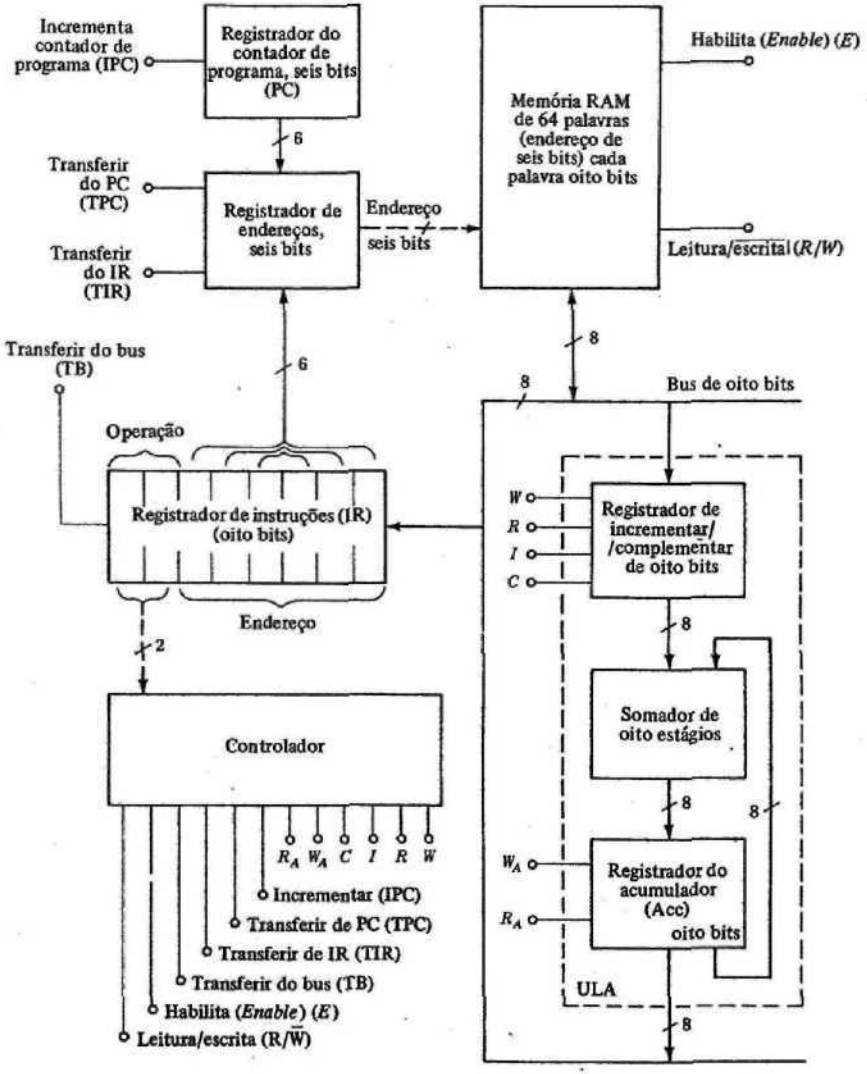


Fig. 8.9-1 A arquitetura de um "computador" que permitirá a combinação aritmética de um grande número de operandos.

Na Fig. 8.9-2 ilustramos o que tipicamente pode estar contido na memória que serve à nossa máquina um tanto simples. Na Fig. 8.9-2*a* escrevemos por extenso em palavras e números decimais o conteúdo de alguns dos locais na memória. Nos locais 0 a 6 escrevemos um *programa* de instruções.

Local da memória		Local da memória	Subtrair de Acc / Local 59 da memória
0	Subtrair de Acc conteúdo de local 59 da memória	0 0 0 0 0 0	1 0 1 1 1 0 1 1
1	Somar ao Acc conteúdo de local 60 da memória	0 0 0 0 0 1	0 1 1 1 1 1 0 0
2	Subtrair de Acc conteúdo de local 61 da memória	0 0 0 0 1 0	1 0 1 1 1 1 0 1
3	Somar ao Acc conteúdo de local 62 da memória	0 0 0 0 1 1	0 1 1 1 1 1 1 0
4	Somar ao Acc conteúdo de local 63 da memória	0 0 0 1 0 0	0 1 1 1 1 1 1 1
5	Transferir conteúdo de Acc para local 39 da memória	0 0 0 1 0 1	1 1 1 0 0 1 1 1
6	Parar (*stop*)	0 0 0 1 1 0	0 0 x x x x x x
• • •		• • •	
59	49	1 1 1 0 1 1	0 0 1 1 0 0 0 1
60	−79	1 1 1 1 0 0	1 0 1 1 0 0 1 1
61	−52	1 1 1 1 0 1	1 1 0 0 1 1 0 0
62	121	1 1 1 1 1 0	0 1 1 1 1 0 0 1
63	82	1 1 1 1 1 1	0 1 0 1 0 0 1 0

(*a*) (*c*)

Código	Instrução
00	Parar (*stop*)
01	Somar a Acc
10	Subtrair de Acc
11	Transferir conteúdo de Acc para

(*b*)

Fig. 8.9-2 (*a*) Um possível conteúdo da memória na Fig. 8.9-1 escrito para transportar o intento do conteúdo da memória. (*b*) Um código de instruções. (*c*) O conteúdo real de bits binários dos locais da memória.

No outro extremo da memória armazenamos alguns operandos. Se pudermos arranjar para que a máquina execute as instruções dadas na ordem listada, a máquina computará a quantidade $-(49) + (-79) - (-52) + (121) + (82) = +127$, como pode ser visto observando os conteúdos dos locais 59 a 63 da memória. A máquina então transferirá esta soma acumulada do registrador do acumulador para o local 39 da memória e depois interromperá e esperará pela intervenção humana. Conforme veremos, há um propósito na colocação das instruções em locais sucessivos na memória. Por outro lado, não é importante que os operandos (49, −79 etc.) estejam localizados em locais sucessivos. Nem é importante que tenhamos colocado as instruções nas extremidades opostas da memória. Tudo o que é requerido é que as instruções sejam colocadas (em locais sucessivos) numa parte da memória e os operandos sejam colocados em outro lugar. A ordem indicada na Fig. 8.9-2 é sugerida principalmente por um respeito característico à escrituração contábil bem organizada. Os locais da memória não indicados na figura têm naturalmente algum conteúdo. (Um local limpo da memória com um conteúdo 00 . . . 00 tem um conteúdo apesar disto.) Mas o conteúdo destes locais da memória é irrelevante para nossa presente discussão. Propomos armazenar nosso resultado no local 39 (não mostrado). O conteúdo do local 39 é desconhecido e irrelevante. Quando for executada a instrução que grava o resultado no local 39, o conteúdo anterior deste local será perdido.

Notemos que usamos quatro instruções: somar, subtrair, transferir e parar. Arbitrariamente, usamos o código de dois bits da Fig. 8.9-2*b* para representar estas instruções. Na Fig. 8.9-2*c* reescrevemos a memória em forma binária. Nos locais da memória que retêm operandos, simplesmente substituímos números decimais por números binários. Em locais da memória que retêm instruções, arbitrariamente arranjamos para que os dois bits mais à esquerda representassem a instrução e os seis bits restantes especificassem o local que retém o operando. Esta atribuição de significância de bit é expressamente indicada para o primeiro local da memória. Os locais 0 a 5 da memória retêm, todos, instruções que exigem uma operação e se referem a um local de memória específico quando o operando é armazenado. O local 6 da memória requer uma operação, mas, uma vez que nenhum operando é envolvido, não nos interessamos pelos seis bits de endereço.

Observemos que as palavras da memória têm oito bits de comprimento. Se propusermos representar números negativos em forma de complemento de dois, a faixa de números que podemos acomodar estende-se de $+127$ a -128. Devemos, então, ser cuidadosos para assegurar que quando nossa máquina acumula os números que estamos combinando, nunca exijamos que a soma acumulada se estenda para fora da faixa permitida. Os números introduzidos nos locais 59 a 63 da memória foram selecionados arbitrariamente, exceto que observamos esta restrição a respeito da faixa.

Retornando à Fig. 8.9-1, notamos que a seção na caixa (quadrículo) desenhada é o sistema da Fig. 8.4-1 sem os registradores R_α e R_β. (Estes registradores foram substituídos pela parte da memória que armazena os operandos.) Este pequeno sistema efetua aritmética (somando e incrementando) e lógica (complementando) e é com razão denominado Unidade Lógica e Aritmética (ULA), embora difira em muitos aspectos da ULA da Fig. 5.13-1. Por simplificação, deixamos de fora o terminal Z_A de *limpar*, uma vez que o acumulador pode ser limpo usando as instruções da Fig. 8.9-2*b* (Prob. 8.9-1).

Continuando nosso exame da Fig. 8.9-1, notamos a presença de um registrador contador de programa (PC), de um registrador de instruções (IR), de um registrador de endereços da memória (MAR) e, finalmente, do controlador que é para colocar nosso pequeno computador à prova e que temos ainda que projetar. O propósito detalhado de cada registrador será discutido em breve. No momento, observemos simplesmente os caminhos disponíveis para transferências de palavras

(ou de partes de palavras) e as facilidades que foram incorporadas a cada registrador. Notemos aqui que há um bus de oito bits a que a memória tem uma conexão bidirecional. A ULA pode aceitar uma palavra do bus sobre seu lado de entrada e fornecer uma palavra ao bus de seu lado de saída. O registrador de instruções pode aceitar uma palavra do bus. O registrador de endereços da memória tem duas entradas de controle que podem ser usadas para transferir para o MAR a palavra de seis bits no contador de programa ou os seis bits mais à direita do registrador de instruções. Os dois bits mais à esquerda do registrador de instruções são tornados disponíveis (não transferidos) para o controlador. Por conseguinte, esta conexão de dois bits é indicada por uma linha tracejada ao invés de linha cheia. A única operação que o contador de programas pode executar é a operação de incrementar. Finalmente, o controlador tem uma linha de saída de controle correspondente a cada linha de entrada de controle de cada registrador e da memória. Quaisquer microoperações que sejam efetuadas serão executadas sempre que a correspondente linha de controle for para o nível de habilitação. Com exceção da memória, todos os registradores e o controlador são sincronizados (clocados), e o momento real em que um registrador aceita uma transferência e incrementa é o instante de tempo da transição de gatilho do relógio. A linha da forma-de-onda de relógio, que é distribuída em todo o sistema da Fig. 8.9-1, não está indicada no desenho.

8.10 OPERAÇÃO DO COMPUTADOR

Para ver como opera nosso computador, consideremos que nossa memória é carregada como na Fig. 8.9-2*c*. Podemos imaginar que para efetuar este carregamento a memória foi temporariamente desconectada do sistema e que as entradas (endereços, dados, *habilita* (*enable*), leitura/escrita) foram aplicadas manualmente. Admitamos também que, no início, o contador do programa (PC) e o registrador do acumulador estão limpos. (Não importa se os outros registradores estão limpos inicialmente.) Listaremos agora, nas Tabs. 8.10-1 e 8.10-2, ciclo de relógio por ciclo de relógio, a seqüência de microoperações através das quais o controlador deve regular o andamento do sistema para que este tome nota da primeira instrução, execute seu intento e prepare para ler a próxima instrução. Quando for viável efetuar mais do que uma microoperação durante o curso de um ciclo de relógio, tiraremos vantagem deste aspecto característico a fim de economizar tempo. Deve ser notado especialmente que as únicas operações especificadas na tabulação são aquelas que serão afetadas pelo fato de que uma ou mais saídas de controle do controlador vão para o nível de habilitação.

Nas operações listadas na Tab. 8.10-1 trouxemos as primeiras instruções da memória para dentro do registrador de instruções. Esta parte do ciclo de operações da máquina é chamada *ciclo de busca* (*fetch cycle*). Agora que a primeira instrução está disponível, a máquina prosseguirá para responder às instruções. O ciclo das operações pelas quais esta resposta é executada é chamado *ciclo de executar*. As operações do ciclo de busca são, naturalmente, executadas sob o controle do controlador, mas a operação do controlador durante o ciclo de busca é independente dos dois bits mais à esquerda no registrador de instruções, que estão disponíveis para o controlador. Em verdade, já que não limpamos o registrador de instruções, nem mesmo soubemos quais eram estes bits. No entanto, agora que buscamos esta primeira instrução e a registramos no registrador de instruções, o processamento a partir deste ponto seguirá um curso que depende da

Tabela 8.10-1 Ciclo de busca (fetch cycle)

Ciclo de relógio	Descrição simbólica da operação	Linha de controle a ser habilitada
1. Transferir conteúdo do contador de programa para o registrador de endereços da memória	PC → MAR	TPC
2. Transferir instrução endereçada (no local 000000) para o registrador de instruções por (1) habilitação da memória para conectar a memória ao bus, (2) colocação de $R/\overline{W}$ em 1 para ler memória, e (3) transferência da palavra no bus para o registrador de instruções; incrementar contador de programa para preparar para fazer aparecer a próxima instrução quando a primeira instrução tiver sido concluída	M → IR ("M" representa palavra de memória endereçada)	E, $R/\overline{W}$, TB
	PC + 1 → PC	IPC

instrução transportada pelos dois *bits de operação* da instrução. Uma vez que a primeira instrução requer *subtração*, a *execução* da instrução prossegue conforme mostrado na Tab. 8.10-2.

A máquina agora executou as primeiras instruções. Providenciaremos para que, correspondentemente, o controlador que projetamos tenha completado toda sua seqüência e retornado a seu início. Por conseguinte, a próxima operação a ser executada é novamente transferir o contador do programa para o registrador de endereços da memória. Mas lembremos que incrementamos o contador de programas. Em conseqüência, a instrução buscada será a segunda instrução (no local 00001). A segunda instrução será executada como a primeira, exceto que, uma vez que a adição é requerida ao invés da subtração, as operações de complementar e incrementar serão des-

Tabela 8.10-2 Ciclo de executar

Ciclo de relógio	Descrição simbólica da operação	Linha de controle a ser habilitada
3. Transferir parte do endereço do registrador (seis bits à direita) para o registrador de endereços da memória (endereço é 59)	IR (ADD) → MAR	TIR
4. Transferir palavra endereçada da memória para o bus e do bus para o registrador CI	M → BUS . BUS → CI	E, $R/\overline{W}$, W
5. Complementar CI	CI → CI	C
6. Incrementar CI	CI + 1 → CI	I
7. Registrar saída do somador no registrador do acumulador	Adder → Acc	W_A

viadas. Assim vemos que, quando o controlador se põe em seqüência, ele busca e depois executa, busca depois executa etc. A operação de busca é sempre a mesma; as operações durante o executar dependem, naturalmente, da instrução.

Ao construir o sistema simples da Fig. 8.9-1 tomamos inúmeras decisões (algumas das quais arbitrárias e algumas baseadas na experiência). Estas decisões referem-se ao número e à função dos registradores, aos tipos de interligações entre os registradores e entre os registradores e a memória, ao número de palavras na memória, ao número de bits por palavra, ao número e tipo de operações que a ULA pode efetuar etc. Estes aspectos constituem a arquitetura e a organização do computador. Uma vez que tenham sido estabelecidas uma arquitetura e uma organização, permanece a tarefa do projeto do controlador. (O controlador de nossa máquina é projetado na próxima seção.)

Há, naturalmente, muitas arquiteturas e organizações que podem ser admitidas por uma peça da maquinaria de computação. Depois de muitos anos de experimentação com uma ampla faixa de possibilidades, a maquinaria de computação da época atual geralmente incorpora alguns aspectos comuns, alguns dos quais devem ser vistos em nosso computador simples e que destacamos agora.

Notemos primeiro que a máquina tem uma memória, em que *armazenamos* de início todas as instruções requeridas para executar uma computação. Por esta razão, a máquina é denominada *computador de programa armazenado*. Uma vez que a máquina está plenamente instruída, não teríamos que interrompê-la para proporcionar outra direção para sua computação. A memória armazena não somente as instruções mas também os operandos e os resultados da computação. Daí, teremos que fazer freqüentes referências à memória para ler dela e escrever nela. O local endereçado é achado no *registrador de endereços da memória* (MAR) e não é surpreendente que haja acesso ao MAR de inúmeras direções. Em nosso caso, podemos chegar ao MAR a partir do contador de programas e do registrador de instruções. (Em computadores mais sofisticados, meios de acesso adicionais diretos e indiretos ao MAR são fornecidos.)

Observemos a seguir que nossa máquina tem uma unidade lógica e aritmética em que todas as operações aritméticas e lógicas são efetuadas. O resto da máquina (além do controlador) consiste *em nada mais que um conjunto de registradores. Mesmo a memória nada mais é que uma coleção* de registradores de armazenamento. E os registradores de armazenamento não fazem nada. Eles constituem apenas o "papel de escrita" digital em que escrevemos as palavras digitais que necessitamos guardar para futura referência. Nossa ULA também é um tanto simples. Ela complementa, incrementa e adiciona. ULAS mais elaboradas proporcionam estas funções e também efetuam operações lógicas (AND, OR etc.), deslocam à esquerda ou à direita etc.

Observemos que na manipulação da máquina providencia-se para que inúmeros assuntos sejam "entendidos". Quando uma instrução é executada, é "entendido" *que a próxima instrução está no próximo local da memória. Daí, o local da próxima instrução não necessita ser especificado*. Novamente, quando deve ser efetuada uma adição, a instrução especifica somente um dos operandos envolvidos na adição. Está "entendido" que o outro operando está no registrador do acumulador. Finalmente, a instrução não dá nenhuma indicação de onde o resultado da adição deve ser armazenado. Está "entendido" que o resultado deve ser deixado no acumulador. Todos estes entendimentos efetuam uma considerável economia no comprimento das palavras. Graças a estes entendimentos que foram incorporados à máquina, uma instrução necessita apenas especificar uma operação e um endereço de um operando. Sem estes entendimentos uma instrução teria que especificar a operação, a fonte do primeiro operando, a fonte do segundo operando, a casa onde o resultado deve ser armazenado e a fonte da próxima instrução. Naturalmente,

uma tal instrução elaborada exigiria muito mais bits do que as instruções mais simples possíveis, devido aos entendimentos.

8.11 PROJETO DO CONTROLADOR DO COMPUTADOR

Um projeto do controlador requerido na Fig. 8.9-1 é mostrado na Fig. 8.11-1. Neste projeto, a fim de obter simplificação, apresentamos um deliberado e extremo descaso pela economia de hardware. Como sempre, o *elemento de partida* e todos os flip-flops são comandados por uma forma-de-onda de sincronização comum (não mostrada explicitamente). O fechamento da chave dá partida à seqüência do registrador de deslocamentos. Quando a saída de cada flip-flop sucessivo cresce alternadamente para lógica 1, é executada uma microoperação. Estabelecemos em palavras quais microoperações são executadas em cada etapa e indicamos entre parênteses quais terminais de controle devem ser tornados ativos para que as microoperações sejam executadas. As duas primeiras etapas na seqüência *buscam* (*fetch*) a instrução e carregam-na dentro do registrador de instruções. Esta parte de busca da operação do controlador é a mesma, independentemente da instrução buscada. Depois de ter sido concluída a parte de busca da seqüência, começa a parte de *executar*. A operação a ser executada é transmitida ao controlador como um código de operação de dois bits. No controlador estes bits são aplicados a um decodificador com quatro linhas de saída. Quando o código de operação for 10, é pretendida a subtração. A linha de saída do decodificador 10 sozinha está em lógica 1 e apenas a porta AND superior está habilitada. A seqüência do registrador de deslocamentos continua através da fila superior de flip-flops, arranjando desta maneira para executar a seqüência apropriada à subtração. Um código 01 de operação habilita a segunda porta AND da parte superior. A seqüência resultante é a mesma da subtração, exceto que são descartadas as microoperações de *complementar* e *incrementar*. Um código 11 de operação exige uma transferência do conteúdo do registrador do acumulador de volta para a memória. Aqui são requeridas duas microoperações. A primeira, como nas operações de subtração e adição, transfere o endereço do registrador de instruções para o registrador de endereços da memória. A segunda *escreve* na memória em vez de ler da memória, daí a inversão de nível lógico aplicada ao controle $R/\overline{W}$ da memória.

Se a instrução for *subtrair*, *somar* ou *transferir*, a seqüência de executar, quando concluída, leva imediatamente de volta através da porta OR ao começo do ciclo de busca. A próxima instrução é buscada, e começa uma nova execução etc. Se, por outro lado, a instrução for 00, o que significa *parar* (*stop*), não há retorno ao ciclo de busca. A operação pára depois da saída da porta AND mais inferior ter estado em lógica 1 durante um ciclo de relógio. Esta saída "acabada" pode ser usada, se quisermos, para indicar que o computador concluiu seu trabalho e agora requer outra intervenção manual.

A porção de executar do controlador da Fig. 8.11-1 usa flip-flops um tanto mais livremente do que o necessário. Uma porção alternativa de executar mostrada na Fig. 8.11-2 é mais econômica em flip-flops, embora às expensas de exigir mais portas. São usados cinco flip-flops, permitindo uma seqüência de executar com outras tantas cinco microoperações. Em cada caso, exceto quando a instrução for parar (*halt*), a primeira etapa da seqüência exige que o terminal TIR seja elevado ao nível ativo. Na segunda etapa, as instruções *somar* e *subtrair* exigem uma microoperação comum, enquanto a instrução *armazenar* requer uma microoperação alternativa. A menos que a instrução seja subtrair, as terceira e quarta etapas da seqüência são descartadas. Se for requerida a subtração, as terceira e quarta microoperações são *complementar*

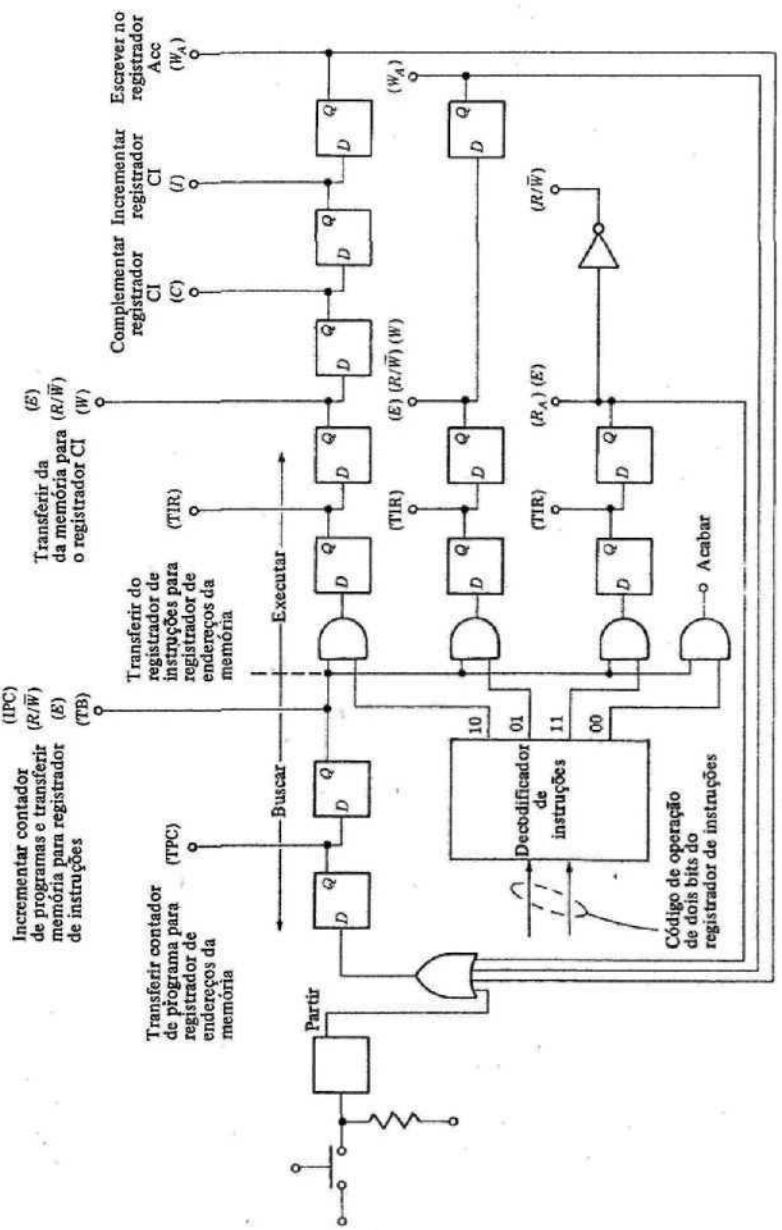


Fig. 8.11-1 O controlador do "computador" da Fig. 8.9-1.

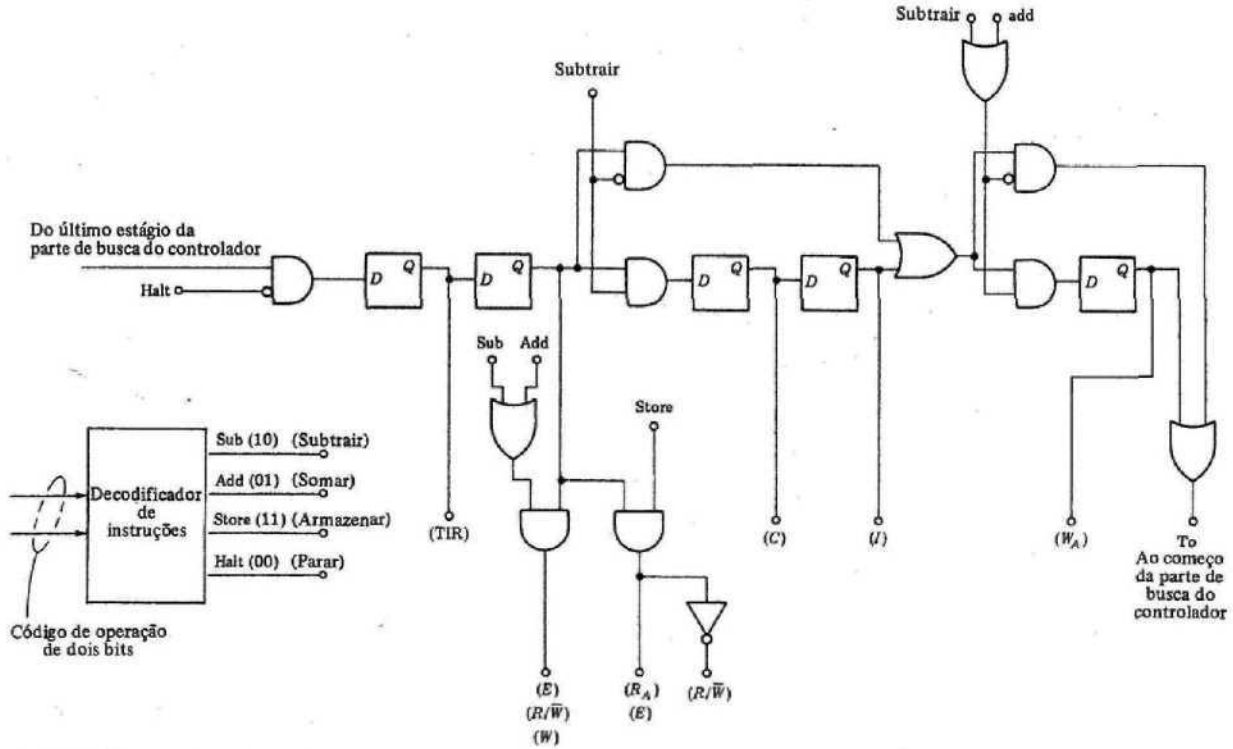


Fig. 8.11-2 Um controlador alternativo para o controlador da Fig. 8.11-1 e um que usa menor número de flip-flops, porém maior número de portas.

e *incrementar*. Finalmente, outra vez, se for requerida *somar* ou *subtrair*, há uma ativação comum da entrada W_A, conforme proporcionado pela quinta etapa na seqüência. Do contrário, esta quinta etapa é descartada.

8.12 INTERRUPÇÃO

Uma vez posto a funcionar, o controlador da Fig. 8.11-1 corre em todas as direções. Ele executa uma seqüência de busca e depois uma seqüência de execução, volta para uma seqüência de busca etc. repetidas vezes até que, finalmente, ele encontra uma instrução de parar (*halt*). O controlador opera o processador digital de idealização um tanto simples da Fig. 8.9-1, que é capaz de fazer não muito mais do que somar e subtrair uma coluna de números. No entanto, o modo de operação deste controlador (buscar, executar, buscar etc.) é de importância fundamental porque os controladores na maioria dos sofisticados computadores digitais operam neste mesmo modo.

Freqüentemente, acontece que é necessário *interromper* esta operação cíclica ou repetitiva de buscar-executar e convocar o controlador para gerar uma seqüência de sinais de comando de habilitação para executar alguma operação especial. A fonte que requer a operação especial é descrita como *requerendo uma interrupção*, e a resposta correspondente do controlador é descrita como *respondendo a um pedido de serviço.*

A Fig. 8.12-1 mostra como o controlador pode ser aumentado para incorporar a capacidade de responder a um tal pedido de serviço. Se o latch estático estiver no estado reset com $\bar{Q} = 1$, a porta AND designada G_N em operação normal é habilitada, e a porta de uma operação de interrupção G_I é desabilitada. Depois do botão de partida ter sido calcado, começará uma seqüência normal, primeiro pela porção de busca do controlador, depois pela porção de executar do controlador, depois de volta à busca, e assim por diante. Suponhamos que num certo instante um sinal seja recebido indicando que a seqüência de serviço é requerida. O sinal que requer a interrupção necessita apenas ser uma breve excursão para lógica 1 aplicada à entrada set do latch. Quando o latch se ajusta ou se estabelece ($Q = 1, \bar{Q} = 0$), será habilitada G_I em vez de G_N. Suponhamos que a primeiríssima etapa da seqüência de busca já tenha começado antes do sinal de interrupção ser recebido. Então a seqüência de busca continuará e será seguida pela seqüência de executar. Assim, a instrução que está no processo de ser buscada e executada não será interrompida, mas quando a execução tiver sido concluída, desde que G_I esteja habilitada e G_N não, a colocação em seqüência prosseguirá através do gerador de seqüências auxiliar para fazer o serviço da interrupção. Quando a seqüência prossegue assim, serão gerados sinais de comando para ativar quaisquer microoperações que sejam requeridas. Providenciamos para que o último sinal de comando, além de qualquer microoperação que ele efetue, também venha a reajustar (reset) o latch. Com o latch reajustado, o controlador voltará para operação normal.

O latch na Fig. 8.12-1 é requerido para armazenar a informação de que alguma fonte fez um pedido de uma interrupção porque ela requer prestação de serviço. Tal armazenamento é requerido porque o sinal de entrada de interrupção pode ser de curta duração e nenhuma resposta pode ser feita pelo controlador até que ele tenha completado o tratamento ou procedimento com a instrução no processo a ser executado. "Registradores" de um bit, como o latch, que não registram uma palavra mas servem para armazenar uma certa parte de informação de um bit para alertar o controlador em alguma situação, são chamados *flags* (bandeiras). Assim, a chamada de uma interrupção *eleva o flag de interrupção* e, quando a interrupção for atendida, o flag será abaixado.

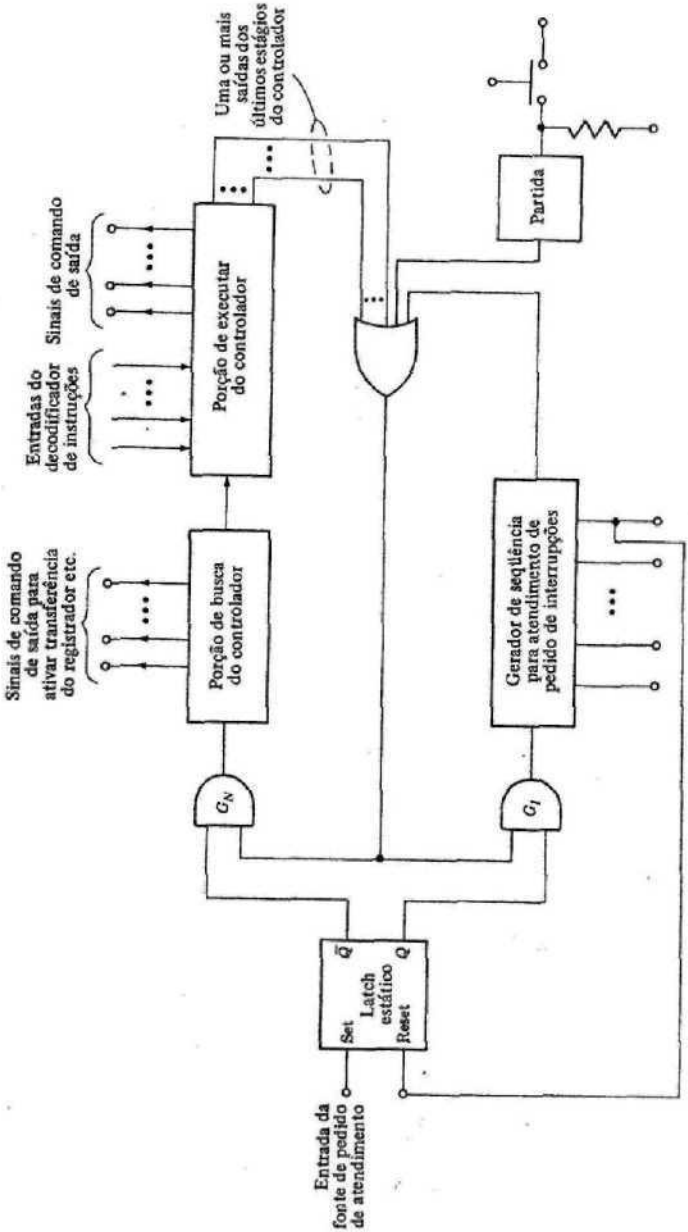


Fig. 8.12-1 Uma modificação do controlador que permite resposta a um pedido de *interrupção*.

Geralmente, um processador digital, seja ele um computador digital ou mesmo algum processador de simplicidade comparável à nossa máquina da Fig. 8.9-1, não está sozinho. Inevitavelmente, ele irá operar em conjunto com algum equipamento que é exterior ao próprio processador. Em muitos casos, este equipamento exterior serve como o recurso pelo qual o processador pode comunicar-se com os seres humanos. Este equipamento pode consistir, por exemplo, em uma máquina de escrever eletromecânica através da qual as pessoas podem *introduzir* dados e instruções no processador e através da qual o processador pode *lançar* seus resultados numa forma que é convenientemente entendida pelas pessoas. Este hardware externo é então denominado equipamento de entrada-saída (I/O); quase todo processador necessita incorporar uma capacidade de controlar este hardware de entrada-saída em seu controlador. Freqüentemente, a facilidade de interrupção de um controlador é requerida precisamente para auxiliar na manipulação do hardware de I/O.

Um simples exemplo de como uma chamada de interrupção pode ser atendida está indicado na Fig. 8.12-2. Aqui consideramos que nosso processador de somar-subtrair da Fig. 8.9-1 está combinando uma longa lista de números, e a taxa de processamento (determinada pela taxa de relógio) é bastante lenta para nós para que razoavelmente tenhamos um interesse, de vez em quando, em testar para ver qual é a acumulação total presente no acumulador. Além disto, a nosso pedido, queremos arranjar para que o registro no acumulador seja datilografado numa máquina de escrever. Admitimos a disponibilidade de uma máquina de escrever eletromecânica que responderá, conforme exigimos, à ativação lógica de linhas de entrada.

Na Fig. 8.12-2 reproduzimos, da Fig. 8.9-1, somente o bus e a ULA e, da ULA, somente o acumulador. O restante do processador não é relevante para nossos propósitos atuais. Para conseguir nosso propósito, calcamos um botão de partida, que fornecemos para indicar ao processador que queremos que ele interrompa sua operação normal e datilografe o último registro do acumulador. O acumulador deve transferir para o registrador externo. Aqui os quatro bits menos significativos e os quatro bits mais significativos são separadamente decodificados por dois decodificadores hexadecimais. Cada decodificador proporciona dezesseis linhas de saída, somente uma de cada dezesseis estando em lógica 1, dependendo das significações numéricas dos códigos hexadecimais de entrada. Suponhamos uma máquina de escrever com os dezesseis caracteres hexadecimais e admitamos igualmente que, quando o terminal "Datilografar MSB" ("Datilografar Bit Mais Significativo") for elevado à lógica 1, a máquina de escrever ativará a tecla correspondente à linha ativa na saída do decodificador do MSB. Uma entrada correspondente à entrada da máquina de escrever do LSB é admitida. E, finalmente, admitimos uma entrada que desloca o carro de um espaço de modo que cada número hexadecimal de dois dígitos pode ser separado dos outros. Conforme também é mostrado na figura, para realizar o que é requerido, o gerador de seqüências para atendimento do pedido de interrupção deve ter quatro sinais de comando. Cada vez, então, que calcamos o botão de interrupção, o processador completará o tratamento com sua instrução presente e irá para a seqüência de atendimento para responder nosso pedido.

Uma dificuldade é imediatamente evidente no esquema da Fig. 8.11-2. Se construirmos nosso gerador de seqüências como os exemplos apresentados de início, teremos permitido apenas um intervalo de relógio para cada operação (transferência, datilografia e deslocamento). A transferência do acumulador para o registrador externo é facilmente conseguida num intervalo de relógio, mas, a menos que a taxa de relógio seja muito lenta, um intervalo de relógio não será adequado para as respostas mecânicas requeridas da máquina de escrever. Um modo de resolver dificuldades devidas à resposta lenta consiste em aumentar de um para muitos o número de

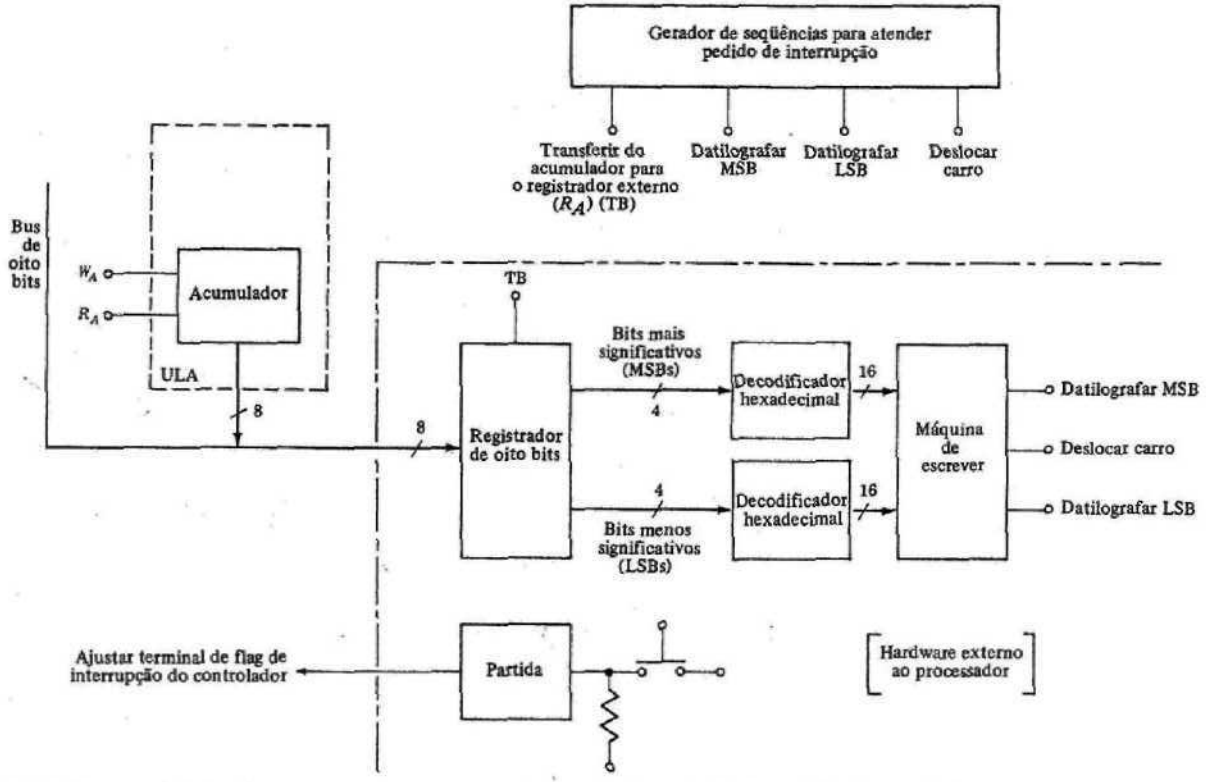


Fig. 8.12-2 Um exemplo. O atendimento de uma interrupção que solicitou uma leitura de saída do conteúdo do acumulador.

flip-flops num gerador de seqüências entre dois pontos a partir dos quais os sinais de comando são atendidos. Este método, naturalmente, é desperdiçador de hardware. Em qualquer caso, este problema de fazer a interface entre sistemas que não operam com a mesma velocidade é um problema que reaparece regularmente. Na seção seguinte nós trataremos deste assunto.

8.13 CONFIRMAÇÃO DE CONEXÃO (HANDSHAKING)

Um método pelo qual podemos arranjar comunicação entre dois sistemas que não operam sincronicamente e muitas vezes com velocidades não relacionadas e amplamente diferentes é chamado *confirmação de conexão* (*handshaking*). Consideremos o assunto especificamente em relação à transmissão de dados de um sistema de transmissão para um sistema de recepção.

No handshaking, o transmissor torna disponível para o receptor, além dos dados a serem transmitidos, uma linha que transporta um sinal lógico chamado "*dados válidos*" (DAV). O receptor torna disponível para o transmissor uma linha que transporta um sinal lógico chamado "*dados aceitos*" (DAC). O arranjo é ilustrado na Fig. 8.13-1. Presumivelmente, o transmissor transmitirá palavra após palavra ao receptor sobre um bus de dados de n bits. Quando DAV = 1, significa que quaisquer processamento e operações que necessitam ser efetuados para colocar uma nova palavra no bus de dados foram concluídos e a palavra no bus está estabelecida, estável e pronta para ser lida pelo receptor; assim, a palavra é *válida*. Quando DAV = 0, a palavra não é válida. Uma seqüência de níveis lógicos no DAV desde 1 até 0 e de volta a 1 acompanha cada mudança de palavra.

Quando DAC = 0, significa que o receptor está completamente pronto para receber a palavra de dados sobre o bus, mas o fará somente se achar que DAV = 1. Se DAC = 1, o receptor não aceitará os dados; assim, o receptor está, efetivamente, desconectado do bus. Transmissor e receptor comunicam-se sobre estas linhas DAV e DAC. O receptor diz ao transmissor quando a palavra de dados foi aceita, de modo que o transmissor possa colocar a próxima palavra na linha. O transmissor diz ao receptor quando uma nova palavra foi de maneira estável estabelecida no bus de modo que ela possa ser lida.

Suponhamos que partimos durante um intervalo em que DAV = 1 e DAC = 0, isto é, uma palavra está sendo transmitida e no processo de ser aceita. Então devemos admitir que DAV não se torna DAV = 0 até que tenhamos DAC = 1. Além disto, quando, com DAV = 1 e DAC = 0, uma palavra é aceita, esta palavra não deve ser aceita novamente até que DAV mude para DAV = 0 e depois novamente de volta a DAV = 1. De outro modo, o receptor aceitará a mesma

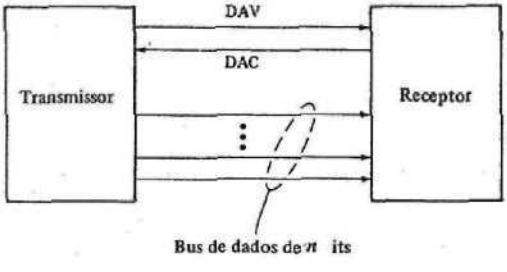


Fig. 8.13-1 A operação de um transmissor e receptor de dados é coordenada pelo uso de linhas DAV e DAC de confirmação de conexão.

Fig. 8.12-2 Um exemplo. O atendimento de uma interrupção que solicitou uma leitura de saída do conteúdo do acumulador.

Tabela 8.13-1 Seqüência de níveis lógicos DAV e DAC quando uma palavra é transmitida e feita a preparação para a palavra seguinte

Intervalo de tempo	Transferir DAV	Receber DAC	Comentário
1	1 válido	0 aceita	Palavra transferida
2	1 válido	1 não aceita	Certificar-se de que os dados permanecem válidos até depois que o receptor se desconecta
3	0 não-válido	1 não aceita	Palavra sendo mudada
4	0 não-válido	0 aceita	Nenhuma transferência porque DAV = 0
5	1	0	Nova palavra transferida

palavra duas vezes. Com um tal sistema haverá seguramente intervalos em que o transmissor deve *esperar* para que o receptor complete algum processamento ou em que o receptor deve *esperar* pelo transmissor. Na verdade, o ponto preciso de todo o arranjo consiste em assegurar que o transmissor ou receptor *esperará* quando requerido. A seqüência que as linhas DAV e DAC atravessam quando uma palavra é transferida e são feitas preparações para transferência da palavra seguinte é mostrada na Tab. 8.13-1.

Como ilustração do uso das linhas DAV e DAC, consideremos a situação na Fig. 8.13-2, onde temos duas memórias de acesso aleatório de organização idêntica e é nosso propósito transferir o conteúdo da memória A (M_A) para a memória B (M_B). Cada palavra deve ocupar o mesmo endereço em B conforme ocupou quando em A. Permanentemente, ajustamos a entrada E_A de habilitar e a $(R/\overline{W})_A$ em $E_A = (R/\overline{W})_A = 1$ de modo que haja sempre uma palavra extraída de M_A, sendo esta palavra determinada por um contador cuja contagem constitui o endereço da palavra selecionada. Em M_B ajustamos $(R/\overline{W})_B = 0$ de modo que M_B esteja permanentemente no modo de escrita, e a operação de escrita ocorrerá quando o terminal de habilitação for para $E_B = 1$.

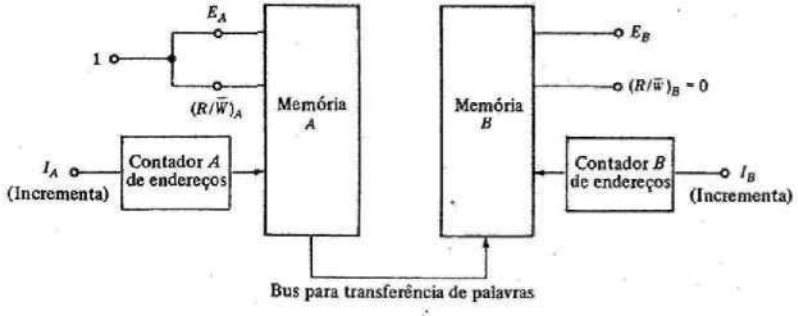


Fig. 8.13-2 Duas RAMs idênticas estão conectadas para efetuar a transferência do conteúdo da memória A para a memória B.

Se o sistema inteiro fosse operar como um simples sistema síncrono, isto é, comandado por uma única forma-de-onda comum de sincronização, o projeto de um controlador para efetuar a transferência de M_A para M_B seria bem elementar. Na verdade, requereríamos apenas um único contador de endereços que forneceria um endereço comum tanto para M_A quanto para M_B. Um controlador de dois estados faria o serviço. Num estado o controlador ajustaria $E_B = 1$ para ler a palavra em M_B. No segundo estado o contador de endereços seria incrementado. O controlador realizaria a seqüência através dos estados escrever, incrementar, escrever, incrementar etc. Se o número de estados do contador for o mesmo que o número de palavras nas memórias, não importará qual seja o endereço inicial no contador. Além disto, se o controlador continuar a funcionar

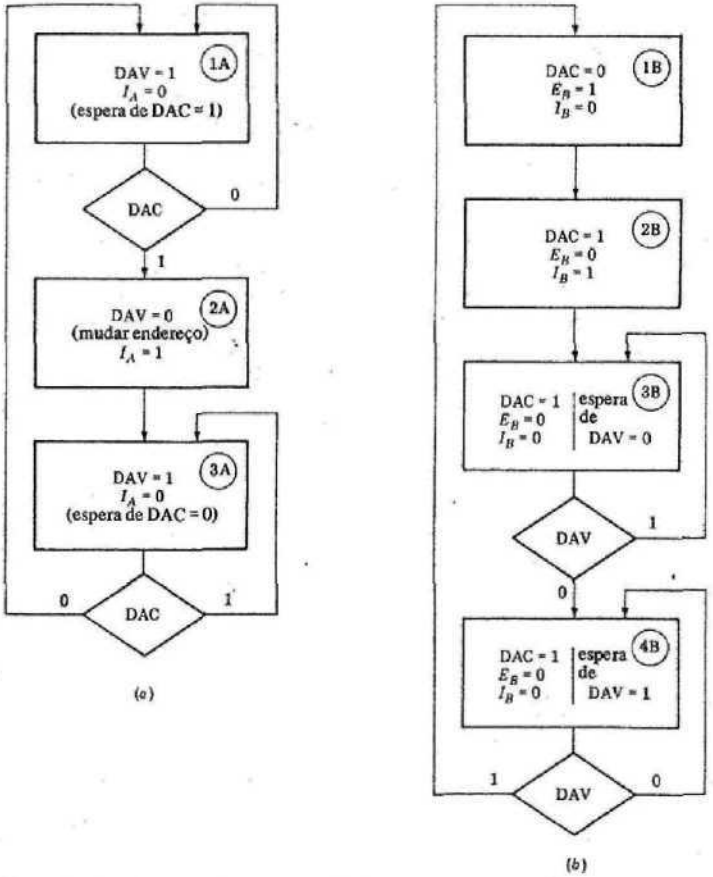


Fig. 8.13-3 O fluxograma de um controlador para operar o sistema da Fig. 8.13-2 e para tornar disponíveis os sinais DAV e DAC.

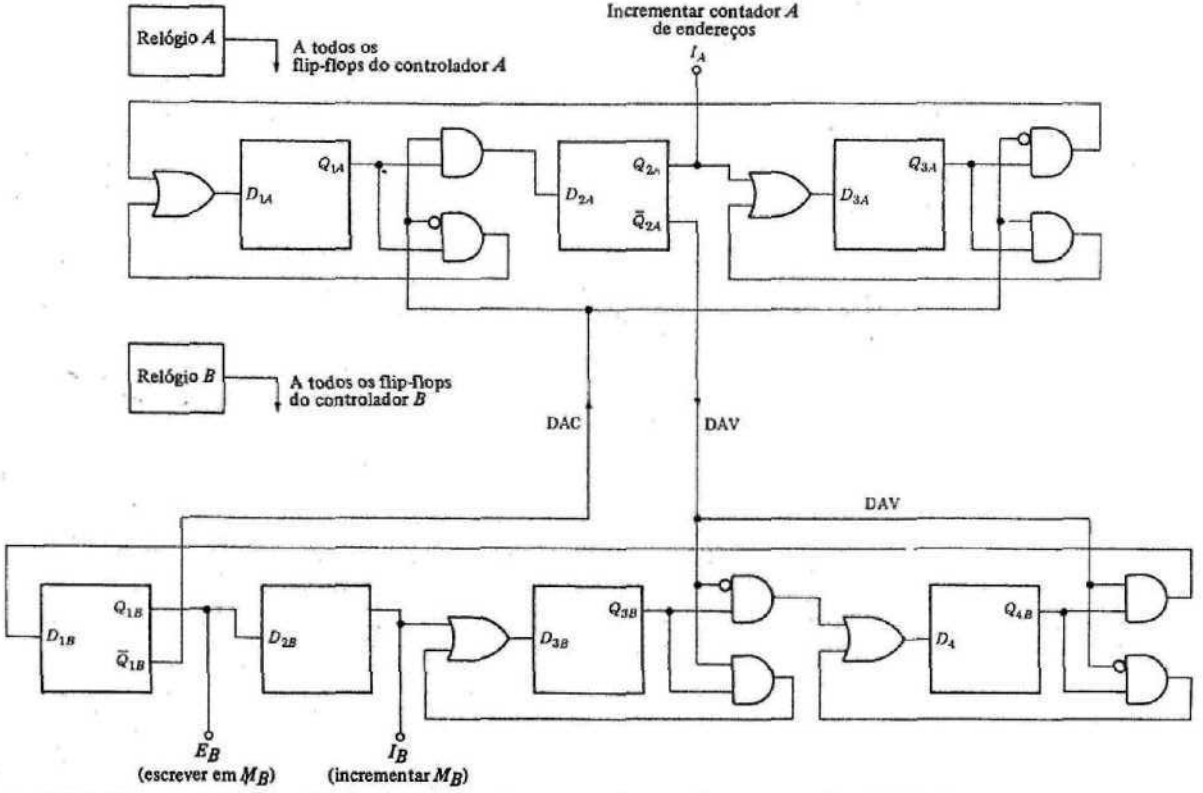


Fig. 8.13-4 Um controlador do registrador de deslocamentos correspondente ao fluxograma da Fig. 8.13-3. São mostradas as interconexões das linhas DAV e DAC.

depois do serviço ser concluído, não haverá prejuízo, uma vez que simplesmente estaremos escrevendo as mesmas palavras nos mesmos lugares.

A seguir suponhamos que as duas porções do sistema A e B devam ser comandadas pelas duas formas-de-onda $C1_A$ e $C1_B$ de sincronização não-relacionadas, separadas e independentes com freqüências f_A e f_B. Pode ser que $f_A > f_B$ ou vice-versa. Pode ser mesmo que f_A e f_B variem com o tempo de maneira não relacionada uma à outra. Os dois sistemas são conectados *somente* pelo bus sobre o qual a palavra é transferida, e interligaremos os dois controladores separados agora requeridos pelas linhas DAC e DAV. Consideremos agora o projeto dos dois controladores.

O fluxograma do controlador A está mostrado na Fig. 8.13-3*a*. Definimos o primeiro estado, $1A$, como sendo um estado em que o contador A de endereços está quiescente em um endereço fixo e a palavra endereçada está sobre o bus. A palavra sobre o bus é a palavra que queremos ler; nenhum de seus bits está no processo de modificação, de modo que a palavra é válida e temos DAV = 1. Este primeiro estado também definimos como sendo um estado que é atingido em conseqüência do fato de que $C1_A$ num instante da transição de gatilho de C1 a linha DAC do controlador B para o controlador A estava DAC = 0. Este estado então é o estado durante o qual uma palavra válida é aplicada a um receptor de aceitação de modo que a palavra será escrita em M_B. A escrita real requer apenas que o controlador coloque E_B em lógica 1 durante este estado.

Desde que DAC permaneça DAC = 0, o transmissor deve guardar a mesma palavra no bus e, correspondentemente, manter DAV = 1. Quando DAC tornar-se DAC = 1, isto é, quando o receptor tenha se desconectado do bus, o controlador do transmissor pode ir para o estado seguinte, $2A$, onde DAV será DAV = 0 e o contador A de endereços estará avançado. Depois do endereço ter sido modificado, o transmissor novamente tem uma palavra válida sobre o bus e, portanto, o controlador vai para um estado $3A$, em que novamente DAV = 1. Neste estado o controlador espera até que DAC novamente se torne DAC = 0, em cujo tempo o controlador retorna a seu estado inicial. Note particularmente que, uma vez que um incremento de endereço tenha sido feito, a próxima mudança de endereço não ocorre até que DAC primeiro se torne DAC = 0 e, depois, DAC = 1. Desta maneira o controlador se certifica de que o receptor aceitou a palavra transmitida antes que uma nova palavra seja colocada no bus.

O fluxograma do controlador B é mostrado na Fig. 8.13-3*b*. O estado $1B$ corresponde ao estado $1A$. Quando os controladores A e B estiverem nestes estados, os contadores de endereço não estão sendo avançados, os dados são válidos, E_B está em $E_B = 1$ e uma transferência de palavra está em processo. Note que o controlador A não pode sair do estado $1A$ até que o controlador B tenha saído do estado $1B$. Note, ainda, que o controlador não retornará ao estado $1B$ até que primeiro DAV tenha ido para DAV = 0 e depois retornado a DAV = 1. Desta maneira está assegurado que a próxima palavra transferida será uma palavra nova.

Um controlador correspondente ao fluxograma da Fig. 8.13-3 é mostrado na Fig. 8.13-4.

Fig. 8.13-4 Um controlador do registrador de deslocamentos correspondente ao fluxograma da Fig. 8.13-3. São mostradas as interconexões das linhas DAV e DAC.