

## 5.7

### Microprogramação: simplificando o projeto do controle

Para o controle do nosso subconjunto MIPS, uma representação gráfica da máquina de estados finitos, como na Figura 5, certamente é adequada. Podemos desenhar esse diagrama em uma única página e traduzi-lo para equações (veja o ■ Apêndice C) sem gerar muitos erros. Considere, em vez disso, uma implementação de todo o conjunto de instruções MIPS-32, que contém mais de 100 instruções (veja o ■ Apêndice A). Em uma implementação, as instruções levam de 1 a mais de 20 ciclos de clock. Claramente, a função de controle será muito mais complexa. Para esse projeto, provavelmente usaríamos uma linguagem de projeto de hardware, como Verilog (que é abordada em mais detalhes na ■ Seção 5.8) e teríamos o controle de estados finitos sintetizado; a próxima seção mostra como isso pode ser feito.

Considere, no entanto, um conjunto de instruções com várias centenas de instruções de classes amplamente variáveis, como a arquitetura IA-32. A unidade de controle poderia facilmente exigir milhares de estados com centenas de seqüências diferentes. Nesse caso, será impossível especificar a unidade de controle com uma representação gráfica. É provável que até mesmo o uso de uma abstração de estados finitos na qual o próximo estado precisa ser explicitamente especificado seja complicada.

Podemos usar algumas idéias da programação para ajudar a criar um método de especificar o controle que facilitará entender e projetar? Suponha que pensemos no conjunto de sinais de controle que precisam ser ativados em um estado como uma instrução a ser executada pelo caminho de dados. Para evitar confundir as instruções do conjunto de instruções MIPS com essas instruções de controle de baixo nível, estas últimas são chamadas de *microinstruções*. Cada microinstrução define o conjunto dos sinais de controle do caminho de dados que precisa ser ativado em um determinado estado. Executar uma microinstrução tem o efeito de ativar os sinais de controle especificados pela microinstrução.

Além de definir que sinais de controle precisam ser ativados, também precisamos especificar a seqüenciação – que microinstrução deve ser executada a seguir? Na máquina de estados finitos mostrada na Figura 5.38, o próximo estado é determinado em uma de duas maneiras diferentes. Algumas vezes, um único próximo estado segue o estado atual incondicionalmente. Por exemplo, o estado 1 sempre segue o estado 0, e a única maneira de atingir o estado 1 é por meio do estado 0. Em outros casos, a escolha do próximo estado depende da entrada. Isso é verdade no estado 1, que possui quatro estados sucessores diferentes.

Quando escrevemos programas, também temos uma situação análoga. Algumas vezes, um grupo de instruções deve ser executado seqüencialmente e, algumas vezes, precisamos desviar. Na programação, o padrão é a execução seqüencial, enquanto o desvio precisa ser indicado explicitamente. Ao descrever o controle como um programa, também consideramos que as microinstruções escritas em seqüência também são executadas em seqüência, enquanto o desvio precisa ser indicado explicitamente. O mecanismo de seqüenciação padrão ainda pode ser implementado usando uma estrutura como a da Figura 5.37; entretanto, normalmente é mais eficiente implementar o estado seqüencial padrão usando um contador. Veremos como essa implementação se parece no final desta seção.

A tarefa de projetar o controle como um programa que implementa as instruções de máquina em termos de instruções mais simples é chamada de *microprogramação*. A idéia básica é representar os valores ativados nas linhas de controle simbolicamente, de modo que o microprograma seja uma representação das microinstruções, exatamente como o assembly é uma representação das instruções de máquina. Ao escolher uma sintaxe para um assembly, normalmente representamos as instruções de máquina como uma série de campos (opcode, registradores e offset ou campo imediato); da mesma forma, iremos representar uma microinstrução sintaticamente como uma seqüência de campos cujas funções estão relacionadas.

Outra idéia importante do software é incorporada no controle microprogramado: o conceito de sub-rotinas. Considere por que isso pode fazer sentido: suponha que estamos implementando um grande conjunto de instruções com muitas instruções complexas. Nesse tipo de implementação, é provável que haja oportunidades para reutilizar seqüências de microcódigo para interpretar instruções semelhantes ou para implementar acesso e decodificação de operandos. O suporte a sub-rotinas no micro-

código permite compartilhar essas seqüências de microprograma sem precisar duplicar as microinstruções. Por esse motivo, as unidades de controle microcodificadas usadas para implementar microprogramas complexos (com centenas a milhares de microinstruções) normalmente fornecem suporte a sub-rotinas de microcódigo. Essas sub-rotinas são implementadas fornecendo uma pilha de endereços de retorno dentro da unidade de controle e usando registradores de rascunho para passar parâmetros.



### Definindo um formato de microinstrução

O microprograma é uma representação simbólica do controle que será traduzida por um programa para lógica de controle. Dessa forma, podemos escolher quantos campos uma microinstrução deve ter e que sinais de controle são afetados em cada campo. O formato da microinstrução deve ser escolhido de modo a simplificar a representação, tornando mais fácil escrever e entender o microprograma. Por exemplo, é útil ter um campo que controle a ALU e um conjunto de três campos que determine as duas origens para a operação da ALU, bem como o destino do resultado da ALU. Além da legibilidade, também gostaríamos que o formato de microprograma tornasse difícil ou impossível escrever microinstruções inconsistentes. Uma microinstrução é inconsistente quando ela exige que um determinado sinal de controle seja enviado para dois valores diferentes. Em breve veremos um exemplo de como isso poderia acontecer.

Para evitar um formato que permita microinstruções inconsistentes, podemos tornar cada campo da microinstrução responsável por especificar um conjunto de sinais de controle não sobrepostos. Para escolher como tornar essa partição dos sinais de controle para essa implementação em campos de microinstrução, é útil reexaminar duas figuras anteriores:

- A Figura 5.28, que mostra todos os sinais de controle e como eles afetam o caminho de dados.
- A Figura 5.29, que mostra a função de cada sinal de controle do caminho de dados.

Os sinais que nunca são ativados simultaneamente podem compartilhar o mesmo campo. A Figura 5.7.1 mostra como a microinstrução pode ser dividida em sete campos e define a função geral de cada campo. Os seis primeiros campos da microinstrução controlam o caminho de dados, enquanto o campo Sequencing (o sétimo campo) especifica como selecionar a próxima microinstrução.

As microinstruções normalmente são colocadas em uma ROM ou PLA (ambas descritas no  Apêndice B e usadas para implementar o controle no  Apêndice C), de modo que podemos atribuir endereços às microinstruções. Em geral, os endereços são seguidos seqüencialmente, da mesma maneira que escolhemos números seqüenciais para os estados na máquina de estados finitos. Três métodos diferentes estão disponíveis para escolher a próxima microinstrução a ser executada:

1. Incrementar o endereço da microinstrução atual para obter o endereço da próxima microinstrução. Esse comportamento seqüencial é indicado no microprograma colocando Seq no campo Sequencing. Como a execução seqüencial de instruções é encontrada freqüentemente, muitos sistemas de microprogramação tornam esse comportamento o padrão.


Nome do campo	Função do campo
ALU Control	Especificar a operação sendo realizada pela ALU durante este clock; o resultado é sempre escrito em SaídaALU.
SRC1	Especificar a origem para o primeiro operando da ALU.
SRC2	Especificar a origem para o segundo operando da ALU.
Register Control	Especificar leitura ou escrita para o banco de registradores e a origem do valor para uma escrita.
Memory	Especificar leitura ou escrita e a origem para a memória. Para uma leitura, especifica o registrador destino.
PCWrite Control	Especificar a escrita do PC.
Sequencing	Especificar como escolher a próxima microinstrução a ser executada.

**FIGURA 5.7.1 Cada microinstrução contém esses sete campos.** Os valores para cada campo são mostrados na Figura 5.7.2

2. Desviar para a microinstrução que inicia a execução da próxima instrução MIPS. Rotularemos essa microinstrução inicial (correspondente ao estado 0) como Fetch e colocaremos o indicador Fetch no campo Sequencing para indicar essa ação.
3. Escolher a próxima microinstrução com base na entrada da unidade de controle. O ato de escolher a próxima microinstrução com base em alguma entrada é chamado de *despacho*. As operações de despacho são implementadas criando-se uma tabela que contém os endereços das microinstruções de destino. Essa tabela é indexada pela entrada da unidade de controle e pode ser implementada em uma ROM ou em um PLA. Em geral, existem várias tabelas de despacho; para essa implementação, precisaremos de duas tabelas de despacho – uma para despachar do estado 1 e outra do estado 2. Indicamos que a próxima microinstrução deve ser escolhida por uma operação de despacho colocando Dispatch *i* (onde *i* é o número da tabela de despacho) no campo Sequencing.

A Figura 5.7.2 fornece uma descrição dos valores permitidos para cada campo da microinstrução e o efeito dos diferentes valores de campo. Lembre-se de que o microprograma é uma representação simbólica. Esse formato de microinstrução é apenas um exemplo dos muitos formatos possíveis.

Nome do campo	Valores para o campo	Função do campo com valor específico
Label	Qualquer string	Usado para especificar rótulos para controlar a seqüenciação de microcódigo. Os rótulos que terminam em 1 ou 2 são usados para despachar com uma tabela de desvios indexada com base no opcode. Outros rótulos são usados como destinos diretos na seqüenciação de microinstruções. Os rótulos não geram sinais de controle diretamente mas são usados para definir o conteúdo das tabelas de despacho e gerar controle para o campo Sequencing.
ALU Control	Add	Faz com que a ALU realize uma soma.
	Subt	Faz com que a ALU realize uma subtração; isso implementa a comparação para desvios.
	Func Code	Usa o campo funct da instrução para determinar o controle da ALU.
SRC1	PC	Usa o PC como a primeira entrada da ALU.
	A	O registrador A é a primeira entrada da ALU.
SRC2	B	O registrador B é a segunda entrada da ALU.
	4	Usa 4 para a segunda entrada da ALU.
	Extend	Usa a saída da unidade de extensão de sinal como a segunda entrada da ALU.
	Extshft	Usa a saída da unidade de deslocamento como a segunda entrada da ALU.
Register Control	Read	Lê dois registradores usando os campos rs e rt do IR como os números de registrador, colocando os dados nos registradores A e B.
	Write ALU	Escreve no banco de registradores usando o campo rd do IR como o número de registrador e o conteúdo de SaídaALU como os dados.
	Write MDR	Escreve no banco de registradores usando o campo rt do IR como o número de registrador e o conteúdo de MDR como os dados.
Memory	Read PC	Lê a memória usando o PC como o endereço; escreve o resultado em IR (e no MDR).
	Read ALU	Lê a memória usando SaídaALU como o endereço; escreve o resultado em MDR.
	Write ALU	Escreve na memória usando SaídaALU como o endereço e o conteúdo de B como os dados.
PCWrite Control	ALU	Escreve a saída da ALU no PC.
	ALUOut-cond	Se a saída Zero da ALU estiver ativa, escreve em PC o conteúdo do registrador SaídaALU.
	Jump Address	Escreve no PC o endereço de desvio da instrução.
Sequencing	Seq	Escolhe a próxima microinstrução seqüencialmente.
	Fetch	Vai para a primeira microinstrução para começar uma nova instrução.
	Dispatch <i>i</i>	Despacha usando a ROM especificada por <i>i</i> (1 ou 2).

**FIGURA 5.7.2 Cada campo da microinstrução possui um número de valores que ele pode assumir.** A segunda coluna fornece os possíveis valores permitidos para o campo, e a terceira coluna define o efeito desse valor. Cada valor de campo, com exceção do campo Label, é mapeado para uma definição específica das linhas de controle do caminho de dados; esse mapeamento é descrito no  Apêndice C, Seção C.5. Essa seção também mostra como o campo Label é usado para gerar as tabelas de despacho. Como veremos, a implementação do microcódigo irá diferir ligeiramente do controle da máquina de estados finitos, mas apenas de maneiras que não afetam a semântica das instruções.

**Detalhamento:** o formato básico de microinstrução pode permitir combinações que não podem ser suportadas dentro do caminho de dados. Em geral, um micromontador realizará verificações nos campos de microinstrução para garantir que essas inconsistências sejam sinalizadas como erros e corrigidas. Uma alternativa é estruturar o formato de microinstrução de modo a evitar isso, mas isso pode tornar a microinstrução menos legível. A maioria dos sistemas de microprogramação escolhe a legibilidade e exige que o montador do microcódigo detecte as inconsistências.

### Criando o microprograma

Agora vamos criar o microprograma para a unidade de controle. Iremos rotular as instruções no microprograma com rótulos simbólicos, que podem ser usados para especificar o conteúdo das tabelas de despacho (veja a Seção C.5 no Apêndice C para saber como as tabelas de despacho são definidas e montadas). Ao escrever o microprograma, há duas situações em que podemos querer deixar um campo da microinstrução em branco. Quando um campo que controla uma unidade funcional ou que faz com que o estado seja escrito (como o campo Memory ou o campo ALU dest) está em branco, nenhum sinal de controle deve ser ativado. Quando um campo *apenas* especifica o controle de um multiplexador que determina a entrada para uma unidade funcional, como o campo SRC1, deixá-lo em branco significa que não nos importamos com a entrada para a unidade funcional (ou com a saída do multiplexador).

A maneira mais fácil de entender o microprograma é desmembrá-lo em partes que lidam com cada componente da execução da instrução, assim como fizemos quando projetamos a máquina de estados finitos.

O primeiro componente de toda execução de instrução é buscar as instruções, decodificá-las e calcular o PC sequencial e o PC de destino de desvio. Essas ações correspondem diretamente às duas primeiras etapas descritas nas páginas 245 a 248. As duas microinstruções necessárias para essas duas primeiras etapas são mostradas a seguir:

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	pc	Extshft	Read			Dispatch 1

Para entender o que faz cada microinstrução, é mais fácil olhar o efeito de um grupo de campos. Na primeira microinstrução, os campos ativados e seus efeitos são os seguintes:

Campos	Efeito
ALU Control, SRC1, SRC2	Calcula PC + 4. (O valor também é escrito em SaídaALU, embora nunca seja lido de lá.)
Memory	Busca a instrução para IR.
PCWrite Control	Faz com que a saída da ALU seja escrita no PC.
Sequencing	Vai para a próxima microinstrução.

O campo Label, contendo o rótulo Fetch, será usado no campo Sequencing quando o microprograma quiser iniciar a execução da próxima instrução.

Para a segunda microinstrução, as operações controladas pela microinstrução são as seguintes:

Campos	Efeito
ALU Control, SRC1, SRC2	Armazena PC + extensão de sinal ( $IR[15:0] \ll 2$ em ALUOut.
Register Control	Usa os campos rs e rt para ler os registradores colocando os dados em A e B.
Sequencing	Usa a tabela de despacho 1 para escolher o próximo endereço de microinstrução.

Podemos pensar na operação de despacho como uma instrução *case* ou *switch* com o campo opcode e a tabela de despacho 1 usada para selecionar uma de quatro seqüências de microinstrução diferentes com um de quatro rótulos diferentes (todos terminados em “1”):

- Mem1 para instruções de acesso à memória
- Rformat1 para instruções tipo R
- BEQ1 para a instrução branch equal
- JUMP1 para a instrução jump

O microprograma para instruções de acesso à memória possui quatro microinstruções, como mostrado a seguir. A primeira microinstrução realiza o cálculo do endereço de memória. Uma sequência de duas instruções é necessária para completar um load (leitura da memória seguida de escrita no banco de registradores), enquanto o store exige apenas uma microinstrução após o cálculo do endereço de memória:

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch

Vejamos os campos da primeira microinstrução nessa sequência:

Campos	Efeito
ALU Control, SRC1, SRC2	Calcula o endereço de memória: Registro (rs) + extensão de sinal (IR[15:0]), escrevendo o resultado em SaídaALU.
Sequencing	Usa a segunda tabela de despacho para desviar para a microinstrução rotulada como LW2 ou SW2.

A primeira microinstrução na sequência específica que lw é rotulada como LW2, já que é alcançada por um despacho por meio da tabela 2. Essa microinstrução tem o seguinte efeito:

Campos	Efeito
Memory	Lê a memória usando SaídaALU como o endereço e escreve os dados no MDR.
Sequencing	Vai para a próxima microinstrução.

A próxima microinstrução completa a execução com uma microinstrução que tem os seguintes efeitos:

Campos	Efeito
Register Control	Escreve o conteúdo do MDR na entrada do banco de registradores especificada por rt.
Sequencing	Vai para a microinstrução rotulada como Fetch.

A microinstrução store, rotulada como SW2, opera de modo semelhante à microinstrução load, rotulada como LW2:

Campos	Efeito
Memory	Escreve na memória usando o conteúdo de SaídaALU como o endereço e o conteúdo de B como o valor.
Sequencing	Vai para a microinstrução rotulada como Fetch.

A sequência de microprograma para instruções tipo R consiste em duas microinstruções. A primeira realiza a operação da ALU (e é rotulada como Rformat1 para fins de despacho), enquanto a segunda escreve o resultado no banco de registradores:

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch

Você pode pensar que, como os campos dessas duas microinstruções não entram em conflito (ou seja, cada uma usa campos diferentes), você poderia combiná-las em uma única microinstrução. Sem dúvida, os otimizadores realizam essas operações quando compilam microcódigo. Nesse caso, entretanto, o resultado da instrução ALU é escrito no registrador SaídaALU, e o valor escrito não pode ser lido até o próximo ciclo de clock; portanto, não podemos combiná-las em uma única microinstrução. (Se você as combinar, acabará escrevendo a coisa errada no banco de registradores!) Você poderia tentar remover o registrador SaídaALU para permitir que as duas microinstruções fossem combinadas, mas isso exigiria estender o ciclo de clock para permitir que a escrita no banco de registradores ocorra no mesmo ciclo de clock da operação da ALU.

A primeira microinstrução inicia a operação da ALU:

Campos	Efeito
ALU Control, SRC1, SRC2	A ALU opera com os conteúdos dos registradores A e B, usando o campo Function para especificar a operação da ALU.
Sequencing	Vai para a próxima microinstrução.

A segunda microinstrução faz com que a saída da ALU seja escrita no banco de registradores:

Campos	Efeito
Register Control	O valor em SaídaALU é escrito na entrada do banco de registradores especificada pelo campo rd.
Sequencing	Vai para a microinstrução rotulada como Fetch.

Como a microinstrução executada imediatamente antes calculou o endereço de destino do desvio, a sequência de microprograma para branch, rotulada como BEQ1, exige apenas uma microinstrução:

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
BEQ1	Subt	A	B			ALUOut-cond	Fetch

Os campos ativados dessa microinstrução são os seguintes:

Campos	Efeito
ALU Control, SRC1, SRC2	A ALU subtrai os operandos em A e B para gerar a saída Zero.
PCWrite Control	Faz com que o PC seja escrito usando o valor já em SaídaALU, se a saída Zero da ALU for verdadeira.
Sequencing	Vai para a microinstrução rotulada como Fetch.

A sequência de microcódigo também consiste em uma microinstrução:

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
JUMP1						Jump Address	Fetch

Apenas dois campos dessa microinstrução são ativados:

Campos	Efeito
PCWrite Control	Faz com que o PC seja escrito usando o endereço de destino de desvio.
Sequencing	Vai para a microinstrução rotulada como Fetch.



O microprograma inteiro aparece na Figura 5.7.3. Ele consiste nas 10 microinstruções que aparecem acima. Esse microprograma corresponde à máquina de estados finitos de dez estados que projetamos anteriormente, já que ambos foram derivados da mesma seqüência de execução de cinco etapas para as instruções. Em máquinas mais complexas, a seqüência de microprograma pode consistir em centenas ou milhares de microinstruções e seria a representação adequada para o controle. Os caminhos de dados de máquinas mais complexas normalmente exigem registradores de rascunho adicionais usados para conter resultados intermediários ao implementar instruções multiciclo complexas. Os registradores A e B são como esses registradores de rascunho, mas os caminhos de dados para conjuntos de instruções mais complexos em geral têm um número maior desses registradores com um conjunto mais rico de interconexões com outros elementos do caminho de dados. Esses registradores estão disponíveis para o microprogramador e tornam ainda mais forte a analogia de implementar o controle como uma tarefa de programação.

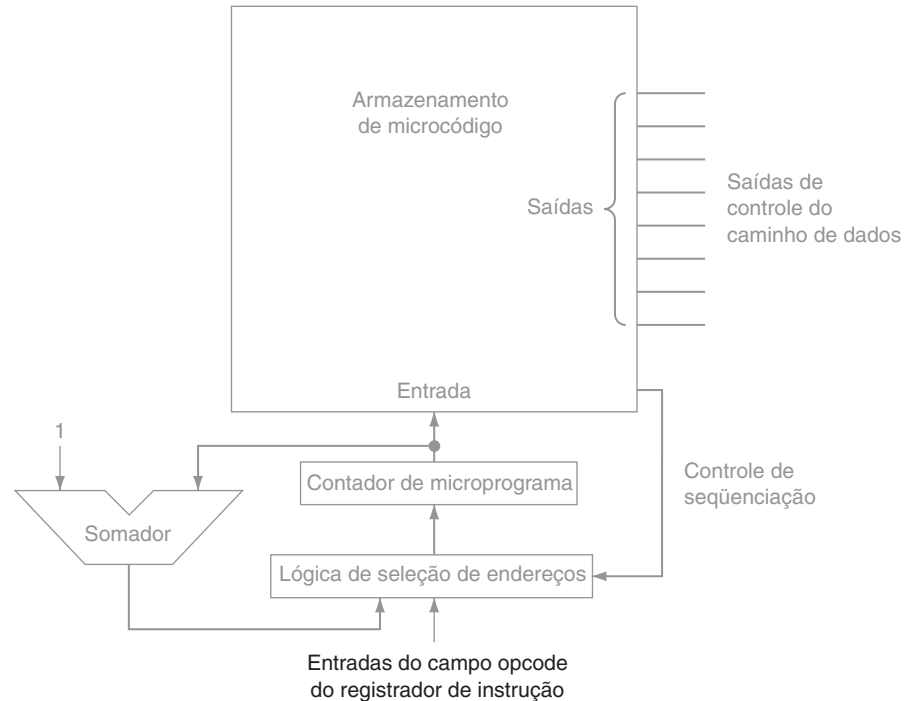
Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func Code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump Address	Fetch

**FIGURA 5.7.3 O microprograma para a unidade de controle.** Lembre-se de que os rótulos são usados para determinar os destinos para as operações de despacho. Dispatch 1 realiza um jump baseado no IR para um rótulo terminado em 1, enquanto Dispatch 2 realiza um jump baseado no IR para um rótulo terminado em 2.

## Implementando o microprograma

Traduzir um microprograma em hardware envolve dois aspectos: decidir como implementar a função de seqüenciação e escolher um método de armazenar a função de controle principal. O microprograma pode ser imaginado como uma representação textual de uma máquina de estados finitos e implementado exatamente da mesma maneira como implementaríamos uma máquina de estados finitos: usando uma PLA para codificar a função de seqüenciação e o controle principal (veja a Figura 5.37). Muitas vezes, no entanto, a implementação da função de seqüenciação e a implementação da função de controle principal são feitas diferentemente, sobretudo para microprogramas grandes.

A forma alternativa de implementação envolve armazenar a função de controle em uma memória ROM e implementar a função de seqüenciação separadamente. A Figura 5.7.4 mostra essa maneira diferente de implementar a função de seqüenciação: usando um incrementador para escolher a próxima microinstrução. Nesse tipo de implementação, o armazenamento do microcódigo determinaria os valores das linhas de controle do caminho de dados, além de *como selecionar* o próximo estado (em vez de *especificar* o próximo estado, como em nossa implementação da máquina de estados finitos). A lógica de seleção de endereços conteria as tabelas de despacho, implementadas em ROMs ou PLAs, e, sob o controle das saídas de seleção de endereços, determinaria a próxima microinstrução a ser executada. A vantagem dessa implementação da função de seqüenciação é que ela remove a lógica para implementar seqüenciação normal das microinstruções, implementando essa seqüenciação com um contador. Assim, nos casos onde haja longas seqüências de microinstruções, o seqüenciador explícito pode resultar em menos lógica no controlador de microcódigo.



**FIGURA 5.7.4 Uma implementação típica de um controlador de microcódigo usaria um incrementador explícito para calcular o próximo estado seqüencial padrão e colocaria o microcódigo em uma ROM.**

As microinstruções, usadas para definir o controle do caminho de dados, são montadas diretamente do microprograma. O contador de microprograma, que substitui o registrador de estado de um controlador que usa máquina de estados finitos, determina como a próxima microinstrução é escolhida. A lógica de seleção de endereços contém as tabelas de despacho, bem como a lógica para selecionar entre os próximos estados alternativos; a seleção da próxima microinstrução é controlada pelas saídas de controle de seqüenciação da lógica de controle. A combinação do contador de microprograma atual, do incrementador, das tabelas de despacho e da lógica de seleção de endereços forma um seqüenciador que seleciona a próxima microinstrução. O armazenamento de microcódigo pode consistir em uma ROM ou pode ser implementado por uma PLA. As PLAs podem ser mais eficientes em implementações VLSI, enquanto as ROMs podem ser mais fáceis de mudar. Mais detalhes sobre as vantagens dessas duas alternativas podem ser encontrados no final desta seção.

Na Figura 5.7.4, a função de controle principal poderia ser implementada em ROM, em vez de ser implementada em uma PLA. Com uma implementação em ROM, o microprograma é montado e armazenado no armazenamento de microcódigo e é endereçado pelo contador de microprograma, de modo muito semelhante a como um programa normal é armazenado na memória de programa e a próxima instrução é escolhida pelo contador de programa. Essa analogia com a programação é a origem da terminologia (microcódigo, microprogramação etc.) e o método inicial pelo qual os microprogramas foram implementados (veja a Seção 5.12).

Embora o tipo de seqüenciador mostrado na Figura 5.7.4 seja normalmente usado para implementar uma especificação de controle de microprograma, ele também pode ser usado para implementar uma especificação de estados finitos. A Seção C.4 do Apêndice C descreve em mais detalhes como gerar esse tipo de seqüenciador. A Seção C.5 descreve como um microprograma pode ser traduzido para essa implementação. Da mesma forma, o Apêndice C mostra como a função de controle pode ser implementada em uma ROM ou uma PLA e discute as compensações. No geral, o Apêndice C mostra como ir das representações simbólicas das máquinas de estados finitos ou microprogramas mostrados neste capítulo para os bits em uma memória ou as entradas em uma PLA. Se você estiver interessado na implementação detalhada ou no processo de tradução, poderá proceder ao Apêndice C.

A escolha de como representar o controle (diagrama de estados finitos *versus* microprograma) e como implementar o controle (PLA *versus* ROM e estado codificado *versus* seqüenciador explícito) são decisões independentes, afetadas pela estrutura da função de controle e pela tecnologia utilizada para implementar o controle.



## Ponderações sobre os métodos de controle

Muita coisa mudou desde que Wilkes [1953] escreveu o primeiro artigo sobre microprogramação. As mudanças mais importantes foram as seguintes:

- As unidades de controle são implementadas como parte integrante do processador, normalmente no mesmo die de silício. Elas não podem ser modificadas independentemente do restante do processador. Além disso, conforme as ferramentas de projeto auxiliadas por computador, a dificuldade de implementar uma ROM ou uma PLA é a mesma.
- A ROM, que era usada para conter as microinstruções, não é mais rápida do que a RAM, que contém o programa em linguagem de máquina. Uma implementação em PLA de uma função de controle normalmente é muito menor do que a implementação em ROM, que pode ter muitas entradas duplicadas ou vagas. Se a PLA for menor, ela será mais rápida.
- Os conjuntos de instruções se tornaram muito mais simples do que eram nas décadas de 1960 e 1970, levando a uma complexidade reduzida do controle.
- As ferramentas de projeto auxiliadas por computador melhoraram de modo que o controle pode ser especificado simbolicamente e, usando computadores muito mais rápidos, podem ser simuladas antes do hardware ser construído. Essa melhoria torna possível obter a lógica de controle correta sem a necessidade de correções futuras.

Essas mudanças tornaram imprecisas as distinções entre diferentes escolhas de implementação. Certamente, é útil usar uma especificação abstrata do controle. Como esse controle é então implementado dependerá de seu tamanho, da tecnologia utilizada e das ferramentas de CAD disponíveis.

Suponha que tivéssemos um conjunto de instruções com cinco classes diferentes; cada uma tendo algumas operações em comum e algumas delas exigindo fluxos de microprograma separados, como a seguir:

- Classe 1: 1 fluxo para todas as instruções.
- Classe 2: 10 fluxos separados.
- Classe 3: 25 fluxos separados.
- Classe 4: 1 fluxo para todas as instruções.
- Classe 5: 15 fluxos separados.

Considerando uma tabela de despacho separada para cada operação de despacho, quantas tabelas de despacho são necessárias e quantas entradas totais existem?

**Verifique  
você mesmo**