Capítulo 2

Conceitos Básicos de Arquitetura de Processador

Este capítulo descreve a arquitetura básica de um processador. Podemos considerar que um processador é organizado em duas unidades, a **seção de processamento** e a **seção de controle**. Este capítulo descreve os principais componentes em cada uma destas unidades. Dentro deste contexto, o capítulo também descreve como um processador executa as instruções de um programa. No capítulo seguinte, aborda-se um tema central na arquitetura de um processador, qual seja, um exemplo de um caminho de dados e seu controle.

2.1 A Seção de Processamento

A seção de processamento é formada basicamente pela **unidade lógica e aritmética** (ALU) e por diversos **registradores**. Estes componentes normalmente estão organizados conforme mostra a Figura 2.1.

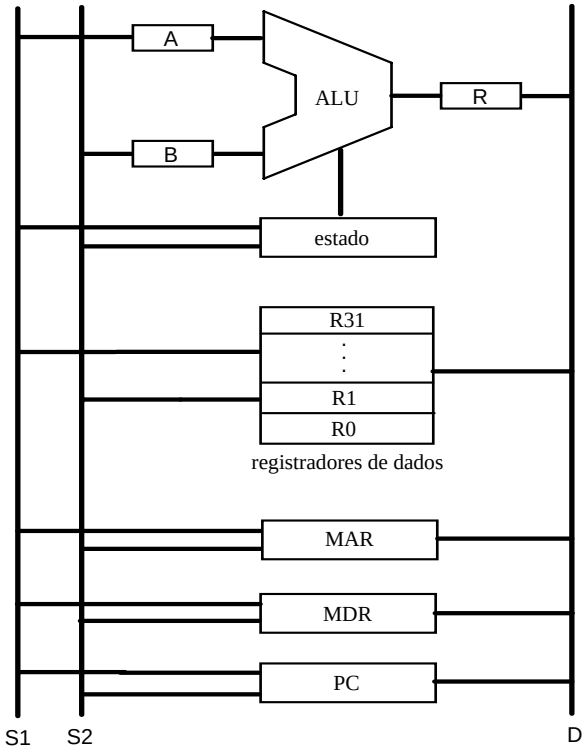

Figura 2.1. *Componentes da seção de processamento.*

A ALU realiza as operações aritméticas, tais como adição e subtração, e operações lógicas, tais como *and*, *or*, *not*. Podemos dizer então que a ALU é o componente da arquitetura que, de fato, manipula os dados. Os registradores são utilizados para armazenar informações internamente no processador. Um registrador pode ser utilizado tanto para acesso de **leitura** quanto para acesso de **escrita**: uma informação é armazenada no registrador em uma operação de escrita, enquanto a informação contida no registrador é recuperada em uma operação de leitura.

A Figura 2.1 mostra aqueles registradores normalmente encontrados na seção de processamento. Os diversos registradores possuem um uso bem definido dentro da arquitetura, e de uma maneira geral podem ser classificados em três tipos: **registradores de uso geral**, **registradores de uso específico** e **registradores auxiliares**. Os registradores de uso geral normalmente são usados para armazenar dados que serão processados pela ALU, bem como resultados produzidos pela ALU. Na seção de processamento mostrada na Figura 2.1, existem 32 registradores de uso geral, denominados R0,...,R31. Coletivamente, estes registradores são chamados de **conjunto de registradores de dados** (*data register file*).

O **registrador de estado** (*status register*) associado à ALU é um registrador de uso específico, e contém informações sobre o resultado produzido pela ALU. Este registrador possui *bits* sinalizadores que são ativados ou desativados[1] de acordo com o tipo de resultado produzido pela ALU. Por exemplo, o registrador de estado pode ter um *bit* denominado Z, o qual é ativado quando o resultado for nulo e desativado quando o resultado for não-nulo. Também é comum encontrar no registrador de estado um *bit* chamado N que é ativado se o resultado for negativo, sendo desativado se o resultado for positivo.

Um outro exemplo de registrador de uso específico é o **contador de programa** (*program counter*). O contador de programa contém o endereço da locação de memória onde se encontra a próxima instrução a ser executada pelo processador.

Os registradores auxiliares normalmente são usados para armazenamento temporário. Este é o caso dos registradores A e B, que armazenam os operandos de entrada da ALU, enquanto estes estão sendo processados. Antes de cada operação da ALU, os operandos são transferidos dos registradores de dados ou da memória principal para estes registradores temporários. O resultado produzido pela ALU é temporariamente armazenado no registrador R até ser transferido para o seu destino, que pode ser um registrador de dados ou a memória principal. A Figura 2.1 mostra dois outros registradores temporários, denominados MAR (*Memory Address Register*) e MDR (*Memory Data Register*). O registrador MAR armazena o

[1] Considera-se aqui que um *bit* é ativado quando ele recebe o valor lógico 1, sendo desativado ao receber o valor lógico 0.

endereço da locação de memória onde será feito o acesso, ao passo que o registrador MDR armazena temporariamente a informação transferida de ou para a locação de memória endereçada por MAR. Em geral, registradores auxiliares não são visíveis, no sentido que um programador de baixo nível não dispõe de instruções para acessar diretamente tais registradores.

A interligação entre a ALU e os registradores é feita através de três vias, chamadas **barramentos internos**. Os barramentos internos S1 e S2 permitem a transferência de dados dos registradores para a ALU. O barramento interno D permite a transferência do resultado produzido pela ALU, temporariamente armazenado no registrador R, para outro registrador.

Uma arquitetura de processador é uma **arquitetura de *n bits*** quando todas as operações da ALU podem ser realizadas sobre operandos de até *n bits*. Normalmente, em uma arquitetura de *n bits* os registradores de dados e os barramentos internos também são de *n bits*, de forma a permitir que os dados sejam armazenados e transferidos de forma eficiente.

2.2 A Execução de Instruções

Tendo examinado os componentes e a organização da seção de processamento, podemos agora analisar como as instruções são executadas. A execução de uma instrução envolve a realização de uma seqüência de passos, que podemos chamar de **passos de execução**. Em geral, a execução de uma instrução envolve quatro passos, como mostra a Figura 2.2.

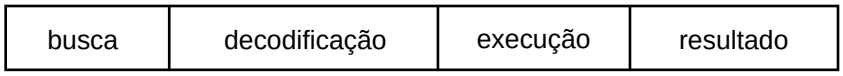

Figura 2.2. *Passos na execução de uma instrução.*

No primeiro passo, denominado **busca**, o processador realiza o acesso ao código binário da instrução, armazenado na memória principal. A etapa seguinte é a **decodificação** da instrução, na qual as informações contidas no código da instrução são interpretadas. Em algumas arquiteturas, neste passo também são acessados os dados usados pela instrução. Após a decodificação, a execução da instrução entra no terceiro passo, denominado **execução**, no qual a operação indicada pela instrução (por exemplo, uma operação na ALU) é efetuada. Finalmente no quarto passo, chamado **resultado**, é armazenado em um registrador ou na memória principal o resultado produzido pela instrução.

Cada passo de execução envolve a realização de várias **operações básicas**. As operações básicas acontecem dentro da seção de processamento, sob a coordenação da seção de controle. Existem quatro tipos principais de operações básicas:

- transferência de dados entre os registradores e a ALU;
- transferência de dados entre os registradores;
- transferência de dados entre os registradores e a memória;
- operações aritméticas e lógicas realizadas pela ALU.

Esta sub-divisão dos passos de execução em operações básicas pode ser visualizada na Figura 2.3.

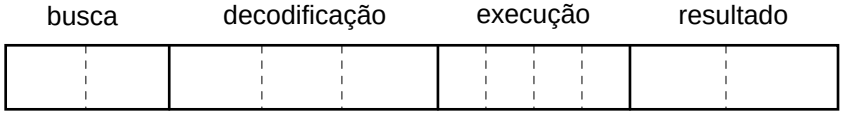

Figura 2.3. *Divisão dos passos de execução em operações básicas.*

A sub-divisão dos passos sugerida pela Figura 2.3 é apenas um exemplo ilustrativo. Os passos de execução não possuem necessariamente o mesmo número de operações básicas em todas as instruções. O que diferencia cada instrução é justamente o **número** e o **tipo** de operações básicas executadas em cada passo.

Para tornar mais claro o mecanismo de execução de instruções, considere uma arquitetura com uma seção de processamento idêntica à da Figura 2.1. Considere também que a arquitetura oferece quatro tipos de instruções: instruções aritméticas e lógicas, instruções de desvio incondicional e condicional, e instruções de acesso à memória. Exemplos destes tipos de instruções aparecem no quadro da Figura 2.4.

Tipo	Exemplo	Descrição
lógica/aritmética	`ADD R1,R2,Rd`	O conteúdo dos registradores R1 e R2 devem ser somados e o resultado armazenado em Rd
desvio incondiconal	`JMP dst`	A próxima instrução a ser executada deve ser a que se encontra na locação de memória com endereço dst
desvio condicional	`JZ dst`	A próxima instrução a ser executada deve ser a que se encontra no endereço dst, caso o *bit Z* do registrador de estado esteja ativado
acesso à memória	`LOAD end,R1` `STORE end,R1`	O dado armazenado na locação de memória com endereço end deve ser transferido para o registrador R1 (LOAD)

Figura 2.4. *Exemplos de instruções.*

A Figura 2.5 apresenta um exemplo hipotético relacionando, para cada tipo de instrução, as operações básicas que acontecem dentro de cada passo de execução. Nesta figura, Ry ← Rx representa a transferência do conteúdo do registrador Rx para o registrador Ry. M[R] denota o conteúdo da locação de memória cujo endereço está no registrador R. Finalmente, IR (*Instruction Register*) representa um registrador especial que recebe o código da instrução a ser executada, enquanto PC é o contador de programa.

	Aritméticas e Lógicas	**Desvios Incondicionais**	**Desvios Condicionais**	**Acessos à Memória**
Busca	MAR ← PC MDR ← M[MAR] IR ← MDR PC++	MAR ← PC MDR ← M[MAR] IR ← MDR PC++	MAR ← PC MDR ← M[MAR] IR ← MDR PC++	MAR ← PC MDR ← M[MAR] IR ← MDR PC++
Decodificação	decod A ← Rs1 B ← Rs2	decod	decod	decod
Execução	R ← A op B	PC ← destino	cond se (cond) PC ← destino	MAR ← end MDR ← Rs (E) M[MAR] ← MDR (E) MDR ← M[MAR] (L)
Resultado	Rd ← R			Rd ← MDR (L)

Figura 2.5. *Operações básicas na execução de instruções.*

O passo de busca é idêntico para todos os tipos de instruções e envolve quatro operações básicas: (1) o conteúdo do contador de programa é transferido para o registrador de endereço de memória MAR; (2) é realizado o acesso à memória, usando o endereço contido em MAR; (3) o código de instrução recebido da memória, temporariamente armazenado em MDR, é transferido para o registrador de instrução IR (este registrador faz parte da seção de controle, como pode ser visto na seção 2.3); (4) contador de programa é incrementado, passando a indicar a próxima instrução onde será feito o acesso e executada.

Concluído o passo de busca, inicia-se o passo de decodificação da instrução armazenada em IR. A interpretação do código da instrução é indicado por **decod** na Figura 2.5. Como mencionado, em algumas arquiteturas este passo também inclui o acesso aos operandos da instrução. Na Figura 2.5, isto é indicado pela transferência do conteúdo dos registradores de dados Rs1 e Rs2 para os registradores temporários A e B, respectivamente, no caso de instruções aritméticas e lógicas.

O próximo passo é o de execução. As operações básicas neste passo dependem inteiramente do tipo de instrução que está sendo executada. No caso das instruções aritméticas e lógicas, este passo corresponde à execução pela ALU da operação indicada na instrução, utilizando como operandos o conteúdo dos registradores A e B, com armazenamento do resultado no registrador R.

Em instruções de desvio incondicional, apenas uma operação básica é realizada: o endereço destino é carregado no contador de programa. Como o contador de programa indica a próxima instrução a ser executada, isto resulta em um desvio para a instrução armazenada no endereço destino. Em desvios condicionais, o contador de programa é modificado somente se a condição de desvio for verdadeira. A avaliação da condição de desvio é indicada por **cond**, na Figura 2.5. O endereço destino é carregado no contador de programa apenas se a condição de desvio testada for verdadeira.

Em instruções de acesso à memória, o passo de execução inicia-se com a transferência do endereço da locação onde será feito o acesso para o registrador MAR. As demais operações básicas dependem se o acesso é de escrita (indicadas por E) ou de leitura (indicadas por L). No caso de uma escrita, são executadas duas operações básicas: a transferência do dado a ser escrito para o registrador MDR e a escrita na memória propriamente dita. No caso de uma leitura, é realizado o acesso à locação de memória e o dado obtido é armazenado no registrador MDR.

O último passo é o de armazenamento do resultado. Em instruções aritméticas e lógicas, o resultado armazenado no registrador R é transferido para o registrador destino indicado na instrução. Em instruções de leitura à memória, o dado que se encontra no registrador MDR é transferido para o registrador destino.

É importante salientar que o quadro na Figura 2.5 é extremamente simplificado. Por exemplo, na prática um acesso à memória não é feito por uma única operação básica como indicado. Ele normalmente requer várias operações básicas. No entanto, este quadro reflete corretamente como a execução de uma instrução é logicamente organizada.

2.3 A Seção de Controle

Como mencionado anteriormente, as operações básicas que ocorrem dentro da seção de processamento são todas comandadas pela seção de controle. Ao efetuar a busca da instrução, a **unidade de controle** interpreta a instrução de modo a identificar quais as operações básicas que devem ser realizadas, e ativa os sinais de controle que fazem uma operação básica de fato acontecer.

A Figura 2.6 apresenta um diagrama em blocos da seção de controle e, de forma bastante simplificada e ilustrativa, a sua interligação com a seção de processamento. Como mostra a figura, a seção de controle é formada basicamente pela **unidade de controle** e pelo **registrador de instrução**, ou IR (*Instruction Register*). A interpretação do código da instrução e a ativação dos sinais de controle são realizadas pela unidade de controle. Os sinais de controle que são ativados, bem como a seqüência com que são ativados, depende de cada instrução em particular.

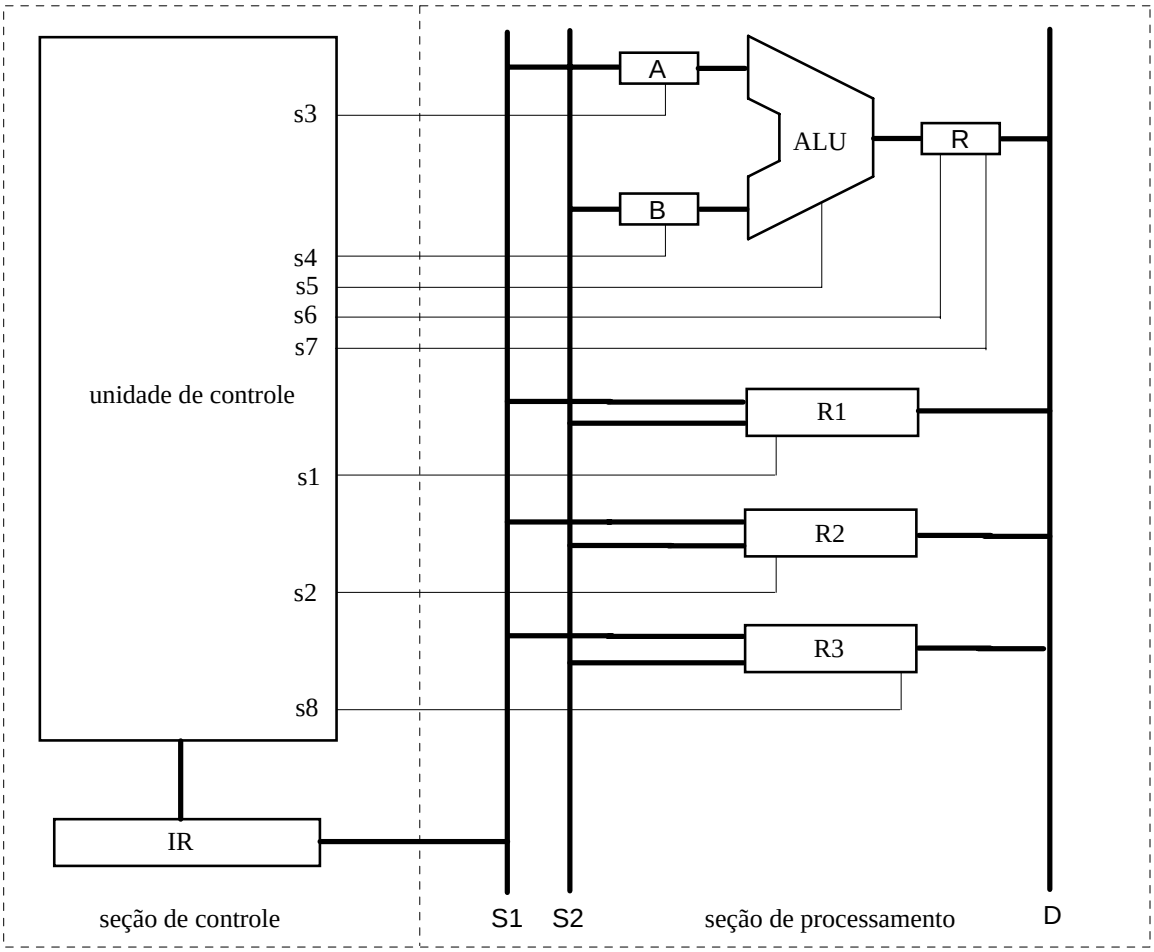

Figura 2.6. *A seção de controle e a parte de processamento.*

A título de exemplo suponha a execução da instrução `ADD R1,R2,R3.` Essa e todas as demais instruções são coordenadas pela unidade de controle. Para tanto, a seção de processamento na Figura 2.6 foi representada apenas com os três registradores de dados

envolvidos na execução desta instrução. A execução desta instrução requer as seguintes operações básicas:

(1) transferência do conteúdo do registrador de dados R1 para o registrador temporário A;

(2) transferência do conteúdo do registrador de dados R2 para o registrador temporário B;

(3) adição dos dados armazenados nos registradores A e B e armazenamento do resultado no registrador R;

(4) transferência do conteúdo do registrador R para o registrador R3.

A seqüência de ativação dos sinais de controle, com as operações básicas correspondentes é apresentada na Figura 2.7.

operação básica	sinal de controle	descrição da operação básica
(1) (2)	s1, s2	coloca o conteúdo de R1 e R2 nos barramentos S1 e S2, respectivamente.
	s3,s4	armazena a informação presente nos barramentos S1e S2 em A e B, respectivamente.
(3)	s5	seleciona a operação de soma na ALU.
	s6	armazena o resultado produzido pela ALU em R.
(4)	s7	coloca o conteúdo de R no barramento D.
	s8	armazena a informação presente no barramento D em R3.

Figura 2.7. *Ativação dos sinais de controle e operações básicas correspondentes.*

Este é um exemplo típico de como a unidade de controle coordena a execução das operações básicas na seção de processamento. Para cada registrador existem sinais que controlam a leitura e a escrita do registrador. Outros sinais indicam à ALU a operação aritmética ou lógica que deve realizada. Para qualquer outro componente na seção de processamento existem os sinais de controle necessários. Para executar uma operação básica, a unidade de controle simplesmente ativa os sinais apropriados na seqüência correta.

O Sinal de *Clock*

Pode-se observar pelo exemplo citado que a ordem na qual os sinais de controle são ativados é crítica. Alguns sinais devem obrigatoriamente preceder outros (s3, por exemplo, não pode ser ativado antes de s1), enquanto que outros sinais podem ser ativados simultaneamente (s1 e s2, por exemplo). Mais ainda, para garantir um tempo suficiente para a transmissão da

informação através dos barramentos internos, em alguns casos deve ser observado um intervalo de tempo mínimo entre a ativação de dois sinais.

Para atender as relações de tempo requeridas na ativação dos sinais de controle, a unidade de controle opera em sincronismo com um **sinal de *clock*.**

A execução de uma instrução consome um certo número de ciclos de *clock*. O número de ciclos de *clock* por instrução não é o mesmo para todas as instruções, já que cada instrução pode envolver um número diferente de operações básicas em cada passo de execução.

O tamanho do ciclo de *clock* é um dos fatores que determinam diretamente o desempenho de um processador. Quanto menor o tamanho do ciclo de *clock*, menor será o tempo de execução das instruções, e assim maior será o número de instruções executadas por unidade de tempo. Ao longo das décadas de 70 e 80, procurava-se diminuir o tamanho do ciclo de *clock* com novas tecnologias que permitissem velocidades de operação cada vez maiores. No entanto, as tecnologias de integração foram se aproximando dos limites impostos pela própria física, tornando esta evolução mais lenta e elevando os custos.

Por este motivo, a redução do ciclo de *clock* passou a ser considerada sob o ponto de vista arquitetural. Atualmente, procura-se diminuir o ciclo de *clock* não somente através de novas tecnologias, mas também através de simplificações na arquitetura, de modo que ela possa ser implementada através de circuitos mais simples e inerentemente mais rápidos.

Implementação da Unidade de Controle

Existem basicamente duas maneiras de implementar uma unidade de controle. A primeira delas é usando **lógica aleatória** (o termo original é *hardwared control*). A outra forma é usando **microprogramação**. A Figura 2.8 mostra a estrutura típica de uma unidade de controle implementada com lógica aleatória.

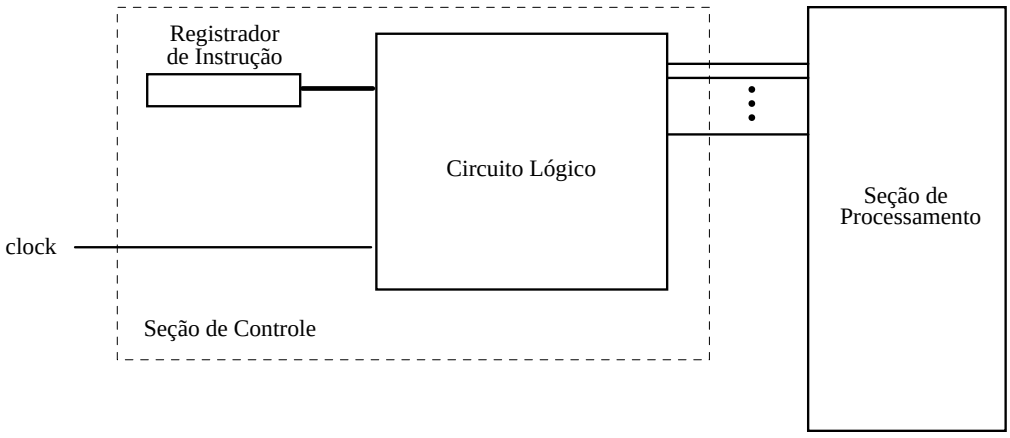

Figura 2.8. *Organização de uma unidade de controle implementada com lógica aleatória.*

Na implementação por lógica aleatória, a unidade de controle é formada por um único circuito lógico, cuja entrada é o código de instrução armazenado em IR e cujas saídas são os próprios sinais de controle que comandam as operações básicas na seção de processamento. De acordo com o código da instrução, a cada ciclo de *clock* este circuito ativa os sinais de controle que comandam as operações básicas que devem ser realizadas naquele ciclo.

A desvantagem desta forma de implementação é que a complexidade do circuito de controle, em termos do número de dispositivos lógicos, aumenta rapidamente com o número de instruções oferecidas pela arquitetura e com o número de operações básicas que devem ser realizadas na execução de uma instrução. Em arquiteturas com instruções funcionalmente complexas, o projeto de uma unidade de controle com lógica aleatória torna-se muito difícil e propenso a erros.

A técnica de microprogramação corrige esta desvantagem da implementação com lógica aleatória. Nela, cada instrução oferecida pela arquitetura, que passa a ser chamada de **macroinstrução**, é na realidade executada por uma seqüência de instruções primitivas, extremamente simples, chamadas **microinstruções**. Os próprios *bits* das microinstruções são usados para ativar e desativar os sinais de controle que comandam as operações básicas. A seqüência de microinstruções que executa uma macroinstrução forma uma **microrotina**. Usando de uma interpretação mais simples, podemos considerar que a execução de uma macroinstrução consiste na chamada de uma microrotina, feita pela unidade de controle. As microinstruções da microrotina executam as operações básicas associadas à macroinstrução.

A Figura 2.9 mostra a estrutura de uma unidade de controle implementada com a técnica de microprogramação.

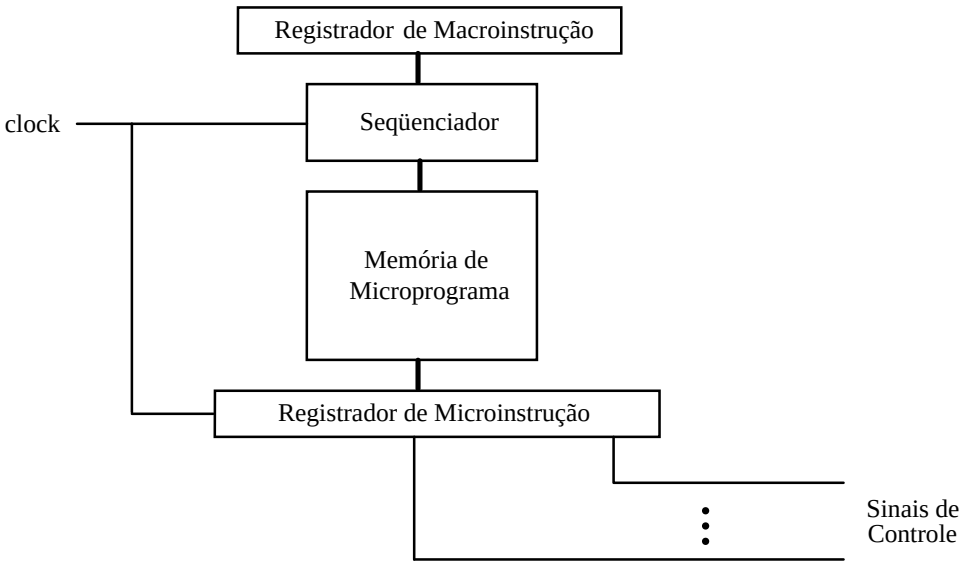

Figura 2.9. *Organização de uma unidade de controle microprogramada.*

As microrotinas encontram-se armazenadas na **memória de microprograma**. Quando o código da macroinstrução é armazenado no registrador de (macro)instrução, o **seqüenciador** interpreta este código e determina o endereço de entrada da microrotina que executa aquela macroinstrução. O seqüenciador fornece, a cada ciclo de *clock*, o endereço da próxima microinstrução a ser executada. Após o acesso à microinstrução, ela é armazenada no **registrador de microinstrução**. Alguns *bits* da microinstrução são usados diretamente para comandar os sinais de controle para a seção de processamento. Outros *bits* são utilizados pelo seqüenciador para determinar a próxima microinstrução a ser executada.

A execução de uma microinstrução envolve o acesso da microinstrução na memória de microprograma, o armazenamento no registrador de microinstrução e a realização das operações básicas comandadas pelos *bits* da microinstrução. Uma nova microinstrução é executada a cada ciclo de *clock*. Quando a execução de uma microrotina é concluída, uma nova macroinstrução é acessada na memória principal e armazenada no registrador de instrução, e iniciada a execução de uma nova microrotina.

A vantagem da microprogramação está no fato de que a implementação das instruções reduz-se basicamente à escrita das microrotinas que serão gravadas na memória de microprograma. Esta vantagem se torna especialmente significativa quando a arquitetura oferece um grande número de instruções e estas instruções são complexas, ou seja, a sua execução envolve um grande número de operações básicas. Neste caso, o projeto de um circuito lógico complicado, como aconteceria na implementação com lógica aleatória, é substituído pela escrita das microrotinas, uma tarefa comparativamente bem mais simples.

2.4 Exceções

Uma das partes mais difíceis do controle é a implementação das exceções e das interrupções porque elas mudam o fluxo normal da execução das instruções. Uma exceção é um evento inesperado gerado dentro do próprio processador. Como exemplo podemos citar o *overflow* aritmético. Uma interrupção também é um evento inesperado só que gerado externamente ao processador, Como exemplo podemos citar as interrupções geradas por um dispositivo de entrada e saída para se comunicar com o processador.

A detecção das condições excepcionais faz parte do caminho crítico de uma máquina. Elas estão relacionadas a temporização, influenciam na determinação do período de *clock* e em conseqüência no desempenho.

As ações básicas que a máquina precisa realizar são salvar os endereços da instrução envolvida no evento e transferir o controle para o sistema operacional num determinado endereço. O sistema operacional pode então realizar as ações apropriadas. A manipulação de

uma exceção pelo sistema operacional pressupõe que ele conheça a razão da exceção, além da instrução que a causou.

Existem dois métodos conhecidos para comunicar ao sistema operacional a razão de uma exceção. Um deles inclui um registrador de causa, que tem um campo adicional para indicar o motivo da exceção. O outro utiliza um vetor de interrupções em que o endereço para o qual o controle é transferido é determinado pela causa da exceção.

2.5 Resumo

Este capítulo mostra que a arquitetura de um processador é organizada em uma seção de processamento e uma seção de controle. A seção de processamento contém a unidade aritmética e lógica, os registradores de dados e outros registradores com funções específicas. A ALU e os registradores são interconectados pelos barramentos internos.

A execução de uma instrução pode ser dividida em passos. Dentro de cada passo são executadas operações básicas, tais como uma operação da ALU, a transferência de dados entre um registrador e a ALU, entre um registrador e a memória ou ainda entre dois registradores. Estas operações são todas sincronizadas pelo sinal de *clock*. Uma operação básica ocorre dentro do intervalo de tempo de um ciclo de *clock*. A execução completa de uma instrução consome alguns ciclos de *clock*. Quanto menor o ciclo de *clock*, menor será o tempo de execução da instrução, e maior será o desempenho.

Na seção de controle encontra-se o componente que comanda todas as operações básicas. A unidade de controle interpreta a instrução e ativa os sinais de controle apropriados que comandam as operações básicas que devem ser realizadas para aquela instrução. Uma maneira de implementar a unidade de controle é através de um circuito monolítico. No entanto, o projeto deste circuito pode tornar-se difícil. Com a técnica de microprogramação, uma (macro)instrução é executada por uma seqüência de microinstruções, ou microrotina. O estado dos sinais de controle é dado pelos próprios *bits* das microinstruções. Uma microinstrução é executada a cada ciclo de *clock*. Com a microprogramação, a implementação da unidade de controle transforma-se basicamente na tarefa de escrever as microrotinas.