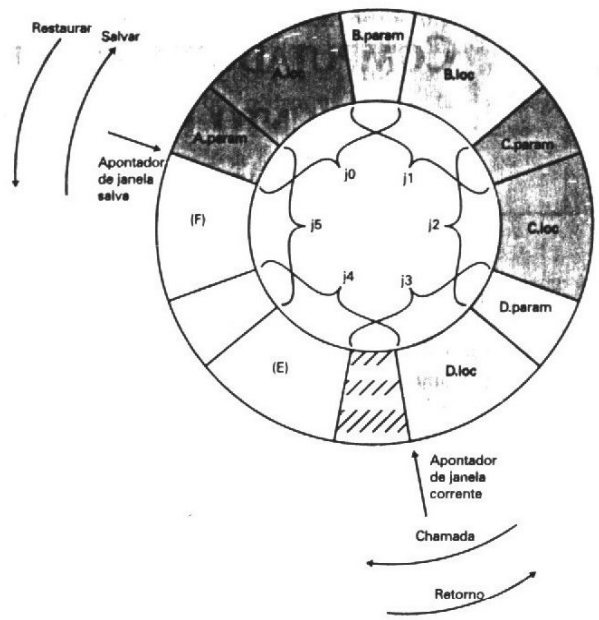

- Estudos sobre o comportamento da execução de programas em linguagens de alto nível orientaram o projeto de um novo tipo de arquitetura para processadores: o computador com um conjunto reduzido de instruções (*reduced instruction set computer* — RISC). Há a predominância de comandos de atribuição, o que sugere que as transferências de dados simples devam ser otimizadas. Além disso, a existência de muitos comandos condicionais (IF) e de laços de repetição sugere que também deve ser otimizado o mecanismo subjacente de controle do seqüenciamento das instruções, de modo que possibilite um uso eficiente de *pipelines*. Estudos sobre padrões de referência a operandos sugerem que é possível obter melhor desempenho, mantendo um número moderado de operandos em registradores.
- Esses estudos motivaram o surgimento de máquinas RISC, com as seguintes características básicas: (1) um conjunto limitado de instruções com formato fixo; (2) um grande número de registradores ou o uso de um compilador que otimize o uso de registradores; e (3) um enfoque na otimização do uso da *pipeline* de instruções.
- O conjunto de instruções simples de uma máquina RISC favorece o uso eficiente de *pipelines* porque o número de operações por instrução é menor e mais previsível. Uma arquitetura com um conjunto reduzido de instruções também favorece o uso da técnica de atraso de instruções de desvio (*delayed branch*), na qual instruções de desvio são trocadas de posição com outras instruções para melhorar a eficiência de uso da *pipeline*.

Desde o desenvolvimento do primeiro computador com programa armazenado na memória, por volta de 1950, houve poucas inovações significativas nas áreas de arquitetura e organização de computadores. Alguns dos maiores avanços desde o nascimento do computador foram:

- **O conceito de família de computadores:** introduzido pela IBM com o Sistema/360, em 1964, e seguido de perto pela DEC, com o PDP-8. O conceito de família de computadores desvincula uma arquitetura de máquina de suas implementações. São fabricados diversos computadores com características de preço e desempenho diferentes, que apresentam, para o usuário, a mesma arquitetura. As diferenças de preço e de desempenho são devidas às diferentes implementações da mesma arquitetura.
- **Unidade de controle microprogramada:** sugerida por Wilkes, em 1951, e introduzida pela IBM na linha S/360, em 1964. A microprogramação facilita a tarefa de projetar e implementar a unidade de controle e oferece suporte para o conceito de família de computadores.
- **Memória cache:** introduzida comercialmente pela primeira vez no IBM S/360 modelo 85, em 1968. A adição desse componente na hierarquia de memória melhorou sensivelmente o desempenho.
- ***Pipeline*:** mecanismo para introduzir paralelismo na natureza essencialmente seqüencial de programas em linguagens de máquina. Alguns exemplos são *pipelines* de instruções e processamento vetorial.
- **Múltiplos processadores:** essa categoria abrange grande número de organizações diferentes, com objetivos distintos.

A essa lista deve ser ainda adicionada uma das inovações mais interessantes e, potencialmente, mais importantes: a arquitetura de computadores com um conjunto reduzido de instruções (RISC). A arquitetura RISC constitui um desvio dramático na história das tendências de arquiteturas de processadores. Uma análise da arquitetura RISC traz à tona muitas das questões importantes relativas à organização e à arquitetura de computadores.

Embora os sistemas RISC tenham sido definidos de várias maneiras por diferentes grupos de projetistas, a maioria dos projetos compartilha os seguintes elementos básicos:

- Um grande número de registradores de propósito geral ou o uso de tecnologias de compilação na otimização do uso de registradores
- Um conjunto de instruções simples e limitado
- Enfoque na otimização da *pipeline* de instruções

A Tabela 12.1 compara diversos sistemas RISC com outros sistemas.

Este capítulo começa com uma breve revisão sobre alguns resultados referentes a conjuntos de instruções e, em seguida, examina cada um dos três tópicos citados anteriormente. Logo depois, são descritos dois dos projetos de máquinas RISC mais bem-documentados.

Tabela 12.1 Características de alguns processadores CISC, RISC e superescalares

	Computador com um conjunto complexo de instruções (CISC)			Computador com um conjunto reduzido de instruções (RISC)		Computador superescalar		
Característica	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Ano de desenvolvimento	1973	1978	1989	1987	1991	1993	1996	1996
Número de instruções	208	303	235	69	94	225		
Tamanho de uma instrução (bytes)	2-6	2-57	1-11	4	4	4	4	4
Modos de endereçamento	4	22	11	1	1	2	1	1
Número de registradores de propósito geral	16	16	8	40-520	32	32	40–520	32
Tamanho da memória de controle (Kbits)	420	480	246	—	—	—	—	—
Tamanho da cache (Kbytes)	64	64	8	32	128	16-32	32	64

12.1 CARACTERÍSTICAS DA EXECUÇÃO DE INSTRUÇÕES

Uma das formas de evolução mais evidentes associada aos computadores foi a das linguagens de programação. Com a queda do custo do hardware, o custo relativo do software aumentou. Simultaneamente, com a diminuição crônica do número de programadores disponíveis para o desenvolvimento de programas, o custo do software aumentou também em termos absolutos. Assim, o custo do software passou a ser maior, no ciclo de vida de um sistema, do que o do hardware. Além de apresentar um alto custo, o sistema de software é, em geral, pouco confiável: é comum que tanto programas de sistema quanto aplicações continuem a exibir novos erros, mesmo após vários anos em operação.

A resposta dos pesquisadores e da indústria a esses problemas foi desenvolver linguagens de programação de alto nível cada vez mais poderosas e complexas. Linguagens de alto nível possibilitam ao programador expressar algoritmos de maneira mais concisa, abstraindo detalhes de máquina, muitas das quais oferecem suporte à programação estruturada e ao projeto orientado a objetos.

Infelizmente, essa solução fez surgir outro problema, conhecido como *gap semântico*: a enorme distância semântica entre as operações disponíveis em linguagens de alto nível e as operações disponibilizadas pelo hardware de computadores. Os sintomas dessa distância incluem ineficiência na execução de programas, tamanho excessivo dos programas e grande complexidade dos compiladores. Os projetistas responderam a esses problemas buscando desenvolver arquiteturas que diminuíssem a distância entre instruções de linguagens de alto nível e instruções de máquina. Características básicas dessas arquiteturas incluem grandes

conjuntos de instruções, dúzias de modos de endereçamento e implementação de diversos comandos de linguagens de alto nível no hardware da máquina. Um exemplo disso é a instrução de máquina CASE do VAX. Esses complexos conjuntos de instruções tinham os seguintes objetivos:

- Facilitar a tarefa do desenvolvedor de compiladores.
- Melhorar a eficiência da execução de programas, com a implementação de seqüências de operações complexas em microcódigo.
- Oferecer suporte para linguagens de alto nível cada vez mais complexas.

Enquanto isso, durante vários anos foram feitos diversos estudos para determinar as características e os padrões de execução de instruções de máquina geradas por programas em linguagens de alto nível. Os resultados desses estudos inspiraram alguns pesquisadores a buscar uma abordagem diferente: tornar a arquitetura que oferece suporte a linguagens de alto nível mais simples, e não mais complexa.

Para entender essa linha de raciocínio adotada pelos defensores da arquitetura RISC, começamos com uma breve revisão das características da execução de instruções. Os aspectos da computação que devem ser examinados são:

- **Operações realizadas:** essas operações determinam as funções que devem ser realizadas pelo processador e sua interação com a memória.
- **Operandos usados:** os tipos de operandos usados e a freqüência de uso de cada um determinam como a memória deve ser organizada para armazená-los e os modos de endereçamento que devem ser disponíveis para acessá-los.
- **Organização das instruções para execução:** determina o controle e a organização da *pipeline*.

No final desta seção, resumimos os resultados da análise desses aspectos em diversos programas em linguagens de alto nível. Esses resultados são baseados em medições feitas dinamicamente, isto é, coletadas durante a execução de programas, contabilizando o número de ocorrências de alguma característica ou o número de vezes em que se verificou uma determinada propriedade ou condição. Medidas feitas estaticamente, ao contrário, são obtidas examinando exclusivamente o texto do programa fonte. Elas não fornecem informações úteis sobre desempenho, pois não são ponderadas com relação ao número de vezes que cada comando é executado.

Operações

Vários estudos foram realizados para analisar o comportamento de programas de alto nível. A Tabela 4.9, discutida no Capítulo 4, mostra os resultados fundamentais obtidos nesses estudos. Existe grande consenso entre os resultados obtidos com diferentes linguagens e aplicações. O tipo de comando predominante é o comando de atribuição, que sugere que transferências de dados simples são de grande importância. Existe também grande predominância de comandos condicionais e de laços de repetição (IF, LOOP). Esses comandos são implementados em linguagem de máquina que usa algum tipo de comparação e instruções de desvio. Isso sugere que o mecanismo de controle do seqüenciamento das instruções é muito importante.

Os resultados obtidos são instrutivos para o projetista de conjunto de instruções da máquina, indicando os tipos de comandos que ocorrem com maior freqüência e que, portanto,

devem ser implementados de maneira 'ótima'. Entretanto, eles não revelam quais comandos consomem maior parte do tempo de execução de um programa típico. Ou seja, dado um programa compilado para linguagem de máquina, quais comandos da linguagem fonte causam a execução da maioria das instruções de linguagem de máquina?

Para obter essa informação essencial, ou seja, para determinar o número médio de instruções de máquina e referências à memória causadas por tipo de comando de linguagem de alto nível, um conjunto de programas do estudo conduzido por Patterson (1982a), descritos no Apêndice 4A, foi compilado e executado nas máquinas VAX, PDP-11 e Motorola 68000. A segunda e a terceira colunas da Tabela 12.2 mostram a freqüência relativa de ocorrência de várias instruções de linguagens de alto nível em uma variedade de programas; os dados foram obtidos observando as ocorrências durante a execução dos programas e não apenas contando o número de vezes que esses comandos ocorrem no código fonte. Portanto, essas estatísticas constituem freqüências dinâmicas. Para obter os dados apresentados nas colunas 4 e 5 (ponderados para instrução de máquina), cada valor na segunda e na terceira colunas é multiplicado pelo número de instruções de máquina produzido pelo compilador. Esses resultados são então normalizados. Desse modo, as colunas 4 e 5 mostram a freqüência de ocorrência relativa, ponderada pelo número de instruções de máquina por comando da linguagem de alto nível. De maneira análoga, a sexta e a sétima colunas são obtidas multiplicando a freqüência de ocorrência de cada tipo de comando pelo número relativo de referências à memória causado pelo comando. Os dados das colunas 4 a 7 fornecem medidas que correspondem ao tempo gasto efetivamente na execução dos vários tipos de comandos. Os resultados sugerem que a operação de chamada/retorno de procedimento é a que consome mais tempo em programas típicos em linguagens de alto nível.

Tabela 12.2 Freqüência dinâmica relativa ponderada de operações de linguagens de alto nível (Patterson, 1982a)

	Ocorrência dinâmica		Ponderada por instrução de máquina		Ponderada por referência à memória	
	Pascal	C	Pascal	C	Pascal	C
Comando de atribuição	45	38	13	13	14	15
Comando de repetição	5	3	42	32	33	26
Chamada de procedimento	15	12	31	33	44	45
Comando condicional	29	43	11	21	7	13
Desvio incondicional	—	3	—	—	—	—
Outros	6	1	3	1	2	1

Você deve certificar-se de que compreendeu o significado da Tabela 12.2. Essa tabela indica a importância relativa dos vários tipos de comandos de uma linguagem de alto nível, quando essa linguagem é compilada para uma arquitetura de conjunto de instruções típica. Os resultados seriam possivelmente diferentes para outros tipos de conjuntos de instruções de máquina. Entretanto, esse estudo fornece resultados representativos para arquiteturas modernas de computadores com um conjunto complexo de instruções (*complex instruction set computer* — CISC). Eles podem, assim, fornecer uma orientação na busca de formas mais eficientes para oferecer suporte a linguagens de alto nível.

Operandos

Um menor número de estudos tem sido realizado para determinar a ocorrência dos diversos tipos de operandos, apesar da importância dessa informação. Existem diversos aspectos significativos.

O estudo de Patterson (1982a), mencionado anteriormente, também procurou determinar a freqüência de ocorrência dinâmica de classes de variáveis (Tabela 12.3). Resultados que foram coerentes entre programas em Pascal e C mostram que a maioria das referências é para variáveis escalares simples, e mais de 80% dessas referências eram para variáveis locais (internas a um procedimento). Além disso, referências a vetores ou registros/estruturas requerem uma referência anterior a seus apontadores ou índices, que são, novamente, variáveis escalares locais. Portanto, existe predominância de referências a variáveis escalares, e essas referências são altamente localizadas.

O estudo de Patterson examinou o comportamento dinâmico de programas em linguagens de alto nível, independentemente da arquitetura subjacente. Como foi discutido anteriormente, para examinar mais profundamente o comportamento de programas é necessário lidar com arquiteturas reais. O estudo apresentado em Lunde (1977) examinou a ocorrência dinâmica de instruções do DEC-10, descobrindo que cada instrução usa, em média, 0,5 operando na memória e 1,4 operando em registradores. Resultados semelhantes são relatados em Huck (1983), para programas C, Pascal e FORTRAN no S/370, PDP-11 e VAX. Embora esses dados dependam fortemente tanto da arquitetura quanto do compilador utilizados, eles mostram efetivamente a freqüência de acesso a operandos.

Esses últimos estudos sugerem a importância de uma arquitetura que facilite o acesso rápido a operandos, pois essa operação é realizada freqüentemente. O estudo de Patterson sugere que um primeiro candidato à otimização é o mecanismo de armazenamento e acesso a variáveis escalares locais.

Tabela 12.3 Porcentagem dinâmica de operandos

	Pascal	C	Média
Constante inteira	16	23	20
Variável escalar	58	53	55
Vetor/registro	26	24	25

Chamadas de procedimentos

Vimos que chamadas e retornos de procedimentos são bastante comuns na execução de programas feitos em linguagens de alto nível. Evidências (Tabela 12.2) sugerem que essas operações são as que consomem mais tempo de execução do código obtido pela compilação de programas. É útil considerar, portanto, mecanismos para implementar essas operações de maneira eficiente. Dois aspectos são significativos: o número de parâmetros e variáveis usadas em um procedimento e o nível de aninhamento de procedimentos.

Um estudo feito por Tanenbaum (1978) mostrou que em 98% das chamadas a procedimentos, durante a execução de um programa, o número de argumentos é inferior a seis e que em 92% dessas chamadas são usadas menos que seis variáveis escalares locais. Resultados semelhantes foram relatados pela equipe de projetistas da arquitetura RISC de Berkeley (Katevenis, 1983), indica a Tabela 12.4. Esses resultados mostram que o número de palavras requeridas por ativação de procedimento não é grande. Estudos relatados anteriormente indicavam que grande parte das referências a operandos são para variáveis escalares locais. Esses estudos mostram que essas referências são, de fato, confinadas a um número relativamente pequeno de variáveis.

O mesmo grupo de Berkeley observou também o padrão de chamadas e retornos de procedimentos em linguagens de alto nível. Eles descobriram que é raro ocorrer uma longa seqüência ininterrupta de chamadas de procedimentos, seguida pela seqüência correspondente de retornos. Ao contrário, eles perceberam que um programa permanece confinado a uma janela bastante estreita de profundidade de chamada de procedimentos. Isso é mostrado na Figura 4.29, discutida no Capítulo 4. Esses resultados reforçam a conclusão de que referências a operandos são altamente localizadas.

Tabela 12.4 Argumentos e variáveis escalares locais de procedimentos

Porcentagem de execução de chamadas de procedimentos com	Compiladores, interpretadores e editores de texto	Pequenos programas não-numéricos
>3 argumentos	0–7%	0–5%
>5 argumentos	0–3%	0%
>8 palavras de argumentos e variáveis escalares locais	1–20%	0–6%
>12 palavras de argumentos e variáveis escalares locais	1–6%	0–3%

Implicações

Diversos grupos de projetistas de computadores observaram os resultados como os que acabamos de relatar e concluíram que a tentativa de projetar uma arquitetura com um conjunto de instruções mais próximo das instruções de linguagens de alto nível não era a estratégia de projeto mais efetiva. Ao contrário, um suporte mais eficaz para linguagens de alto nível poderia ser obtido por meio da otimização do desempenho das características responsáveis por maior consumo de tempo de execução de programas típicos escritos em linguagens de alto nível.

A partir da generalização do trabalho de diversos pesquisadores, podemos perceber que as arquiteturas RISC se caracterizam principalmente por três elementos. Primeiramente, pelo uso de um grande número de registradores ou de técnicas de compilação que visam otimizar a utilização de registradores. Isso visa otimizar referências a operandos. Os estudos discutidos anteriormente mostram que existem diversas referências a operandos em cada instrução de uma linguagem de alto nível e que a proporção de comandos com movimentação de dados (atribuição) é grande. Esse fato, juntamente com a localidade de referências e a predominância de referências a variáveis escalares, sugere que o desempenho pode ser melhorado com a redução do número de referências à memória, à custa de maior número de referências a registradores. Diante da alta localidade das referências, parece ser útil empregar um conjunto maior de registradores.

Em segundo lugar, é necessária cuidadosa atenção no projeto de *pipelines* de instruções. Em virtude da alta taxa de ocorrência de instruções de desvio condicional e de chamada a procedimento, uma *pipeline* de instruções simples seria ineficiente. Esse fato é comprovado pela grande proporção de instruções buscadas antecipadamente que nunca são executadas.

Finalmente, parece ser mais indicado empregar um conjunto de instruções simplificado (reduzido). Esse ponto não é tão óbvio quanto os outros, mas deverá tornar-se mais claro ao longo da discussão a seguir.

12.2 USO DE UM GRANDE BANCO DE REGISTRADORES

Os resultados resumidos na Seção 12.1 mostram como é importante um acesso rápido a operandos. Vimos que existe grande proporção de comandos de atribuição em programas de linguagens de alto nível, muitos deles da forma simples A ← B. Há também um número significativo de acessos a operandos para cada comando de uma linguagem de alto nível. Juntando esses resultados ao fato de a maioria dos acessos ser de acessos a variáveis escalares locais, fica clara a importância do armazenamento de valores em registradores.

A razão é que, entre os dispositivos de armazenamento disponíveis, os registradores constituem o dispositivo que oferece acesso mais rápido; mais rápido mesmo que a memória principal ou a memória cache. Um banco de registradores é fisicamente pequeno, fica contido na mesma pastilha da ULA e da unidade de controle e usa endereços de tamanho muito menor que endereços da memória cache ou da memória principal. Para tirar proveito disso, é necessário que se adote uma estratégia para garantir que os operandos usados mais freqüentemente sejam mantidos em registradores, minimizando operações de transferência de dados entre os registradores e a memória.

Duas abordagens básicas são possíveis, uma baseada em software e a outra em hardware. A abordagem baseada em software consiste em delegar ao compilador a responsabilidade de otimizar o uso de registradores. O compilador tenta alocar nos registradores as variáveis que serão mais usadas durante um determinado período de tempo. Essa abordagem requer o uso de algoritmos sofisticados de análise de programas. A abordagem baseada em hardware consiste simplesmente em usar um número maior de registradores, de modo que mais variáveis possam ser armazenadas em registradores por um maior período de tempo.

Nesta seção, discutiremos a abordagem de hardware. Essa abordagem foi usada pela primeira vez pelo grupo de projetistas da arquitetura RISC de Berkeley (Patterson, 1982a), no primeiro produto comercial RISC, o processador Pyramid (Ragan-Kelley, 1983), e atualmente é utilizada na popular arquitetura SPARC.

Janelas de registradores

O uso de um grande número de registradores tem como objetivo diminuir a necessidade de acesso à memória. A tarefa do projeto é organizar os registradores de modo que esse objetivo seja atingido.

Como a maioria das referências a operandos é para variáveis escalares locais, a abordagem óbvia é armazenar em registradores os valores dessas variáveis, sendo talvez alguns poucos registradores reservados para variáveis globais. O problema é que a definição de *local* muda com cada chamada e retorno de procedimento, operações que ocorrem com freqüência. Em cada chamada, variáveis locais mantidas em registradores devem ser armazenadas na memória, de maneira que os registradores possam ser reutilizados pelo procedimento chamado. Além disso, devem ser passados parâmetros para o procedimento chamado. No retorno do procedimento, os valores salvos anteriormente devem ser restaurados (carregados de volta nos registradores) e, possivelmente, os resultados devem ser passados do procedimento para o programa que o chamou.

A solução para isso é baseada em dois outros resultados relatados na Seção 12.1. Primeiramente, o fato de um procedimento típico ter um número pequeno de parâmetros e variáveis locais (Tabela 12.4). Em segundo lugar, o fato de a profundidade de ativações de procedimentos variar dentro de uma faixa relativamente estreita (Figura 4.29). Para explorar essas propriedades, são empregados vários conjuntos pequenos de registradores, cada qual alocado para um procedimento diferente. Uma chamada de procedimento faz com que o processador use uma janela de registradores diferente de tamanho fixo, em vez de salvar os valores contidos em registradores na memória. Janelas alocadas para procedimentos adjacentes são sobrepostas, para permitir a passagem de parâmetros.

Esse conceito é mostrado na Figura 12.1. Em qualquer instante de tempo, apenas uma janela de registradores é visível, sendo endereçada como se fosse o único conjunto de registradores disponível (por exemplo, endereços 0 a $N - 1$). A janela é dividida em três áreas de tamanho fixo. Registradores de parâmetros são usados para armazenar os valores dos parâmetros passados do procedimento que efetuou a chamada para o procedimento corrente, assim como os resultados a serem retornados. Registradores locais são usados para variáveis locais, conforme foram designadas pelo compilador. Registradores temporários são usados para trocar parâmetros e resultados com o procedimento de nível inferior seguinte (procedimento chamado pelo procedimento corrente). Os registradores temporários de um nível são, fisicamente, os mesmos que os registradores de parâmetro do próximo nível inferior. Essa sobreposição permite que a passagem de parâmetros seja feita sem a necessidade de movimentação real de dados.

Figura 12.1 Sobreposição de janelas de registradores.

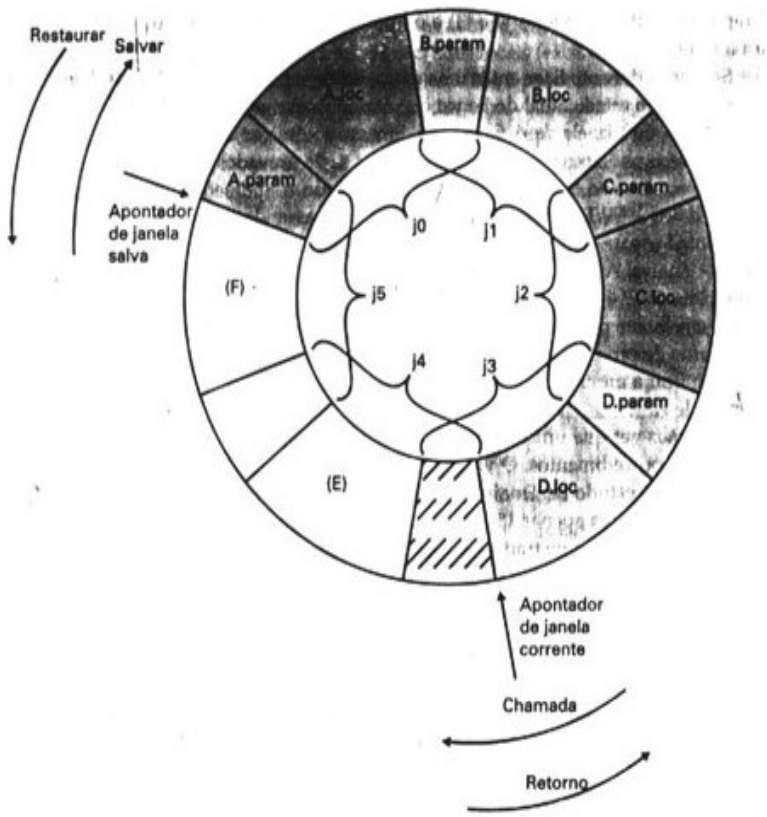

Figura 12.2 Organização circular de janelas de registradores sobrepostas.

Para manipular qualquer possível padrão de chamadas e retornos de procedimentos, o número de janelas de registradores teria de ser ilimitado. Em vez disso, as janelas de registradores podem apenas ser usadas para armazenar dados de algumas poucas ativações de procedimento mais recentes. Dados de ativações mais antigas devem ser salvos na memória e restaurados mais tarde, quando diminuir a profundidade de aninhamento de chamadas. Portanto, o banco de registradores é, de fato, organizado como um arranjo circular de janelas sobrepostas.

Essa organização é mostrada na Figura 12.2, que representa um banco de registradores com seis janelas organizadas de forma circular. A figura mostra a utilização das janelas de registradores em uma chamada de procedimento com nível de aninhamento igual a 4 (*A* chamou *B*; *B* chamou *C*; *C* chamou *D*), sendo *D* o procedimento ativo. O apontador para a janela corrente (*current window pointer* — CWP) indica a janela alocada ao procedimento corrente. Uma referência a um registrador em uma instrução de máquina usa esse apontador como base para determinar o registrador físico usado na instrução. O apontador para a janela salva identifica a janela armazenada na memória mais recentemente. Se o procedimento *D* chama um procedimento *E*, os argumentos para *E* são colocados nos registradores temporários de *D* (na

sobreposição entre as janelas j4 e j3) e o apontador de janela corrente é movido uma janela para a frente.

Se o procedimento *E* faz então uma chamada a um procedimento *F*, essa chamada não pode ser executada no estado atual do banco circular de janelas de registradores. Isso porque a janela de *F* se sobrepõe à janela de *A*. Se o procedimento *F* começar a carregar seus registradores temporários, preparando uma chamada, esses dados serão gravados sobre registradores de parâmetros do procedimento *A* (*A.param*). Portanto, quando o apontador de janela corrente (*current window pointer* — CWP) é incrementado (módulo 6), passando a apontar a mesma janela indicada pelo apontador para a janela salva (*saved window pointer* — SWP), ocorre uma interrupção e a janela de *A* é salva. Apenas as duas primeiras partes (*A.param* e *A.loc*) precisam ser armazenadas. O apontador SWP é então incrementado e a chamada ao procedimento *F* prossegue. Uma interrupção semelhante pode ocorrer em uma instrução de retorno. Por exemplo, após a ativação de *F*, no retorno do procedimento *B* para o procedimento *A*, o apontador CWP é decrementado, passando a indicar a mesma janela que SWP. Isso causa uma interrupção que resulta na restauração da janela de *A*.

Podemos ver que um banco de registradores de *N* janelas pode sustentar apenas $N - 1$ ativações de procedimentos. O valor de *N* não precisa ser grande. Como foi mencionado no Apêndice 4A, o estudo de Tmair (1983) mostra que, com oito janelas, é preciso salvar ou restaurar uma janela em apenas 1% das chamadas/retornos. Os computadores RISC de Berkeley usam 8 janelas, de 16 registradores cada uma. O computador Pyramid emprega 16 janelas de 32 registradores cada uma.

Variáveis globais

O esquema de janelas descrito oferece uma organização eficiente para armazenar variáveis escalares locais em registradores. Entretanto, esse esquema não atende à necessidade de armazenar variáveis globais, que são usadas por mais de um procedimento. Duas opções podem ser usadas. A primeira seria o compilador alocar na memória todas as variáveis declaradas como globais em um programa escrito em uma linguagem de alto nível, e todas as instruções de máquina que usam essas variáveis deverão acessar os operandos na memória. Isso é simples, tanto do ponto de vista do hardware como do software (compilador). Entretanto, esse esquema não é eficiente para variáveis globais usadas com muita freqüência.

Uma alternativa é incorporar no processador um conjunto de registradores globais. Haveria um número fixo de registradores desse tipo, disponíveis para todos os procedimentos. Um esquema de numeração unificado poderia ser usado para simplificar o formato das instruções. Por exemplo, referências aos registradores 0 a 7 poderiam mencionar registradores globais únicos e referências aos registradores 8 a 31 corresponderiam a registradores físicos na janela corrente. Isso implica algum aumento de complexidade do hardware, para lidar com essa divisão de endereçamento a registradores. Além disso, o compilador deve decidir que variáveis globais devem ser alocadas nesses registradores.

Grande banco de registradores *versus* memória cache

Um conjunto de registradores organizado em janelas atua como uma área de memória de armazenamento temporário pequena e de acesso rápido, que é usada para manter um subconjunto das variáveis que provavelmente devem ser usadas mais freqüentemente. Sob esse ponto de vista, o banco de registradores atua como uma memória cache. Surge, portanto, a questão de saber se não seria melhor e mais simples usar uma memória cache e o pequeno conjunto tradicional de registradores.

A Tabela 12.5 compara características dessas duas abordagens. O banco de registradores organizado em janelas mantém todas as variáveis escalares locais (exceto no caso raro em que esse número de variáveis exceda o tamanho da janela) dos $N - 1$ procedimentos ativados mais recentemente. A memória cache contém uma seleção das variáveis escalares usadas mais recentemente. O banco de registradores deve economizar mais tempo porque retém os valores de todas as variáveis escalares locais. Por outro lado, a cache faz uso mais eficiente da memória porque reage à situação dinamicamente. Além disso, caches geralmente tratam todas as referências à memória da mesma maneira, incluindo instruções e outros tipos de dados. Portanto, o uso da memória cache permite uma economia de tempo nesses casos, que não são favorecidos pelo uso de um banco de registradores.

Tabela 12.5 Características de organização de computadores com um grande banco de registradores e com uma memória cache

Grande banco de registradores	Cache
Todas as variáveis escalares locais	Variáveis escalares locais usadas recentemente
Variáveis individuais	Blocos de memória
Variáveis globais designadas pelo compilador	Variáveis globais usadas recentemente
Operações de salvamento/restauração baseadas na profundidade de aninhamento de procedimentos	Operações de salvamento/restauração baseadas no algoritmo de substituição de cache
Endereçamento de registrador	Endereçamento de memória

Um banco de registradores pode fazer um uso ineficiente de espaço de armazenamento, pois nem todo o procedimento requererá todo o espaço da janela alocada a ele. Por outro lado, a memória cache sofre outro tipo de ineficiência: os dados são lidos da memória cache em blocos. Enquanto os registradores contêm apenas as variáveis em uso, a cache pode manter um bloco que contenha alguns ou muitos dados que não serão usados.

A cache é capaz de manipular variáveis globais, assim como variáveis locais. Normalmente, existe um grande número de variáveis escalares globais, mas apenas algumas delas são usadas freqüentemente (Katevenis, 1983). O mecanismo de gerenciamento da memória cache é capaz de descobrir dinamicamente as variáveis usadas com freqüência, para mantê-las na cache. Um banco de registradores baseados em janelas, acrescido de registradores globais, também é capaz de reter valores de algumas variáveis escalares globais. Entretanto, é difícil para um compilador determinar as variáveis globais que serão usadas mais freqüentemente.

Com o banco de registradores, a transferência de dados entre os registradores e a memória é determinada pela profundidade de aninhamento de procedimentos. Como essa profundidade normalmente varia dentro de uma faixa estreita, o acesso à memória é relativamente raro. A maioria das memórias cache é associativa por conjunto, com um tamanho de conjunto pequeno. Portanto, existe perigo de que outros dados ou instruções substituam variáveis usadas freqüentemente.

Com base na discussão apresentada até aqui, a escolha entre banco de registradores e memória cache não parece óbvia. Entretanto, existe uma característica em que a abordagem de banco de registradores é claramente superior, o que sugere que um sistema baseado em cache seja significativamente mais lento. Essa distinção entre as duas abordagens se revela pela sobrecarga existente no mecanismo de endereçamento de cada abordagem.