As caches de segundo nível se tornaram comuns quando os projetistas descobriram que o silício limitado e as metas de altas velocidades de clock impedem que as caches primárias se tornem grandes. A cache secundária, que normalmente é 10 ou mais vezes maior do que a cache primária, trata muitos acessos que falham na cache primária. Nesses casos, a penalidade de falha é aquela do tempo de acesso à cache secundária (em geral, menos de dez ciclos de processador) contra o tempo de acesso à memória (normalmente mais de 100 ciclos de processador). Assim como na associatividade, as negociações de projeto entre o tamanho da cache secundária e seu tempo de acesso dependem de vários aspectos de implementação.

…foi inventado um sistema para fazer a combinação entre os sistemas centrais de memória e os tambores de discos aparecer para o programador como um depósito de nível único, com as transferências necessárias ocorrendo automaticamente.

Kilburn et al., *One-level storage system, 1962*

5.4 Memória Virtual

Na seção anterior, vimos como as caches fornecem acesso rápido às partes recentemente usadas do código e dos dados de um programa. Da mesma forma, a memória principal pode agir como uma "cache" para o armazenamento secundário, normalmente implementado com discos magnéticos. Essa técnica é chamada de **memória virtual**. Historicamente, houve duas motivações principais para a memória virtual: permitir o compartilhamento seguro e eficiente da memória entre vários programas e remover os transtornos de programação de uma quantidade pequena e limitada de memória principal. Quatro décadas após sua invenção, o primeiro motivo é o que ainda predomina.

memória virtual Uma técnica que usa a memória principal como uma "cache" para armazenamento secundário.

Considere um grupo de programas executados ao mesmo tempo em um computador. É claro que, para permitir que vários programas compartilhem a mesma memória, precisamos ser capazes de proteger os programas uns dos outros, garantindo que um programa só possa ler e escrever as partes da memória principal atribuídas a ele. A memória principal precisa conter apenas as partes ativas dos muitos programas, exatamente como uma cache contém apenas a parte ativa de um programa. Portanto, o princípio da localidade possibilita a memória virtual e as caches, e a memória virtual nos permite compartilhar eficientemente o processador e a memória principal.

Não podemos saber quais programas irão compartilhar a memória com outros programas quando os compilamos. Na verdade, os programas que compartilham a memória mudam dinamicamente enquanto estão sendo executados. Devido a essa interação dinâmica, gostaríamos de compilar cada programa para o seu próprio *espaço de endereçamento* – faixa distinta dos locais de memória acessível apenas a esse programa. A memória virtual implementa a tradução do espaço de endereçamento de um programa para os **endereços físicos**. Esse processo de tradução impõe a **proteção** do espaço de endereçamento de um programa contra outros programas.

endereço físico Um endereço na memória principal.

proteção Um conjunto de mecanismos para garantir que múltiplos processos compartilhando processador, memória ou dispositivos de E/S não possam interferir, intencionalmente ou não, um com o outro, lendo ou escrevendo dados no outro. Esses mecanismos também isolam o sistema operacional de um processo de usuário.

A segunda motivação para a memória virtual é permitir que um único programa do usuário exceda o tamanho da memória principal. Antigamente, se um programa se tornasse muito grande para a memória, cabia ao programador fazê-lo se adequar. Os programadores dividiam os programas em partes e, então, identificavam aquelas mutuamente exclusivas. Esses *overlays* eram carregados ou descarregados sob o controle do programa do usuário durante a execução, com o programador garantindo que o programa nunca tentaria acessar um overlay que não estivesse carregado e que os overlays carregados nunca excederiam o tamanho total da memória. Os overlays eram tradicionalmente organizados como módulos, cada um contendo código e dados. As chamadas entre procedimentos em módulos diferentes levavam um módulo a se sobrepor a outro.

Como você pode bem imaginar, essa responsabilidade era uma carga substancial para os programadores. A memória virtual, criada para aliviar os programas dessa dificuldade, gerencia automaticamente os dois níveis da hierarquia de memória representados pela memória principal (às vezes, chamada de *memória física* para distingui-la da memória virtual) e pelo armazenamento secundário.

Embora os conceitos aplicados na memória virtual e nas caches sejam os mesmos, suas diferentes raízes históricas levaram ao uso de uma terminologia diferente. Um bloco de memória virtual é chamado de *página,* e uma falha da memória virtual é chamada de **falta de página**. Com a memória virtual, o processador produz um **endereço virtual**, traduzido por uma combinação de hardware e software para um *endereço físico*, que, por sua vez, pode ser usado de modo a acessar a memória principal. A Figura 5.19 mostra a memória endereçada virtualmente com páginas mapeadas na memória principal. Esse processo é chamado de *mapeamento de endereço* ou **tradução de endereço**. Hoje, os dois níveis de hierarquia de memória controlados pela memória virtual são as DRAMs e os discos magnéticos (veja o Capítulo 1, Seção "Um lugar seguro para os dados"). Se voltarmos à nossa analogia da biblioteca, podemos pensar no endereço virtual como o título de um livro e no endereço físico como seu local na biblioteca.

falta de página Um evento que ocorre quando uma página acessada não está presente na memória principal.

endereço virtual Um endereço que corresponde a um local no espaço virtual e é traduzido pelo mapeamento de endereço para um endereço físico quando a memória é acessada.

tradução de endereço Também chamada de mapeamento de endereço. O processo pelo qual um endereço virtual é mapeado a um endereço usado para acessar a memória.

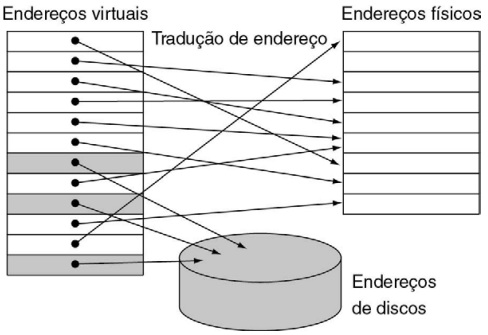

FIGURA 5.19 Na memória virtual, os blocos de memória (chamados de *páginas*) são mapeados de um conjunto de endereços (chamados de *endereços virtuais*) em outro conjunto (chamado de *endereços físicos*). O processador gera endereços virtuais enquanto a memória é acessada usando endereços físicos. Tanto a memória virtual quanto a memória física são desmembradas em páginas, de modo que uma página virtual é realmente mapeada em uma página física. Naturalmente, também é possível que uma página virtual esteja ausente da memória principal e não seja mapeada para um endereço físico, residindo no disco em vez disso. As páginas físicas podem ser compartilhadas fazendo dois endereços virtuais apontarem para o mesmo endereço físico. Essa capacidade é usada para permitir que dois programas diferentes compartilhem dados ou código.

A memória virtual também simplifica o carregamento do programa para execução fornecendo *relocação*. A relocação mapeia os endereços virtuais usados por um programa para diferentes endereços físicos antes que os endereços sejam usados no acesso à memória. Essa relocação nos permite carregar o programa em qualquer lugar na memória principal. Além disso, todos os sistemas de memória virtual em uso atualmente relocam o programa como um conjunto de blocos (páginas) de tamanho fixo, eliminando, assim, a necessidade de encontrar um bloco contíguo de memória para alocar um programa; em vez disso, o sistema operacional só precisa encontrar um número suficiente de páginas na memória principal.

Na memória virtual, o endereço é desmembrado em um *número de página virtual* e um *offset de página*. A Figura 5.20 mostra a tradução do número de página virtual para um *número de página física*. O número de página física constitui a parte mais significativa do endereço físico, enquanto o offset de página, que não é alterado, constitui a parte menos significativa. O número de bits no campo offset de página determina o tamanho da página. O número de páginas endereçáveis com o endereço virtual não precisa corresponder ao número de páginas endereçáveis com o endereço físico. Ter um número de páginas virtuais maior do que as páginas físicas é a base para a ilusão de uma quantidade de memória virtual essencialmente ilimitada.

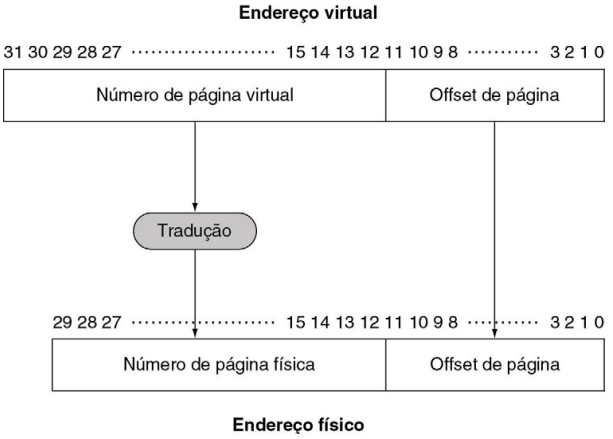

FIGURA 5.20 Mapeamento de um endereço virtual em um endereço físico. O tamanho de página é 2^{12} = 4KB. O número de páginas físicas permitido na memória é 2^{18}, já que o número de página física contém 18 bits. Portanto, a memória principal pode ter, no máximo, 1GB, enquanto o espaço de endereço virtual possui 4GB.

Muitas escolhas de projeto nos sistemas de memória virtual são motivadas pelo alto custo de uma falha, que, na memória virtual, tradicionalmente é chamada de falta de página. Uma falta de página levará milhões de ciclos de clock para ser processada. (A tabela na Seção 5.1 mostra que a memória principal é aproximadamente 100.000 vezes mais rápida do que o disco.) Essa enorme penalidade de falha, dominada pelo tempo para obter a primeira palavra para tamanhos de página típicos, leva a várias decisões importantes nos sistemas de memória virtual:

- As páginas devem ser grandes o suficiente para tentar amortizar o longo tempo de acesso. Tamanhos de 4KB a 16KB são comuns atualmente. Novos sistemas de desktop e servidor estão sendo desenvolvidos para suportar páginas de 32KB e 64KB, embora novos sistemas embutidos estejam indo na outra direção, para páginas de 1KB.
- Organizações que reduzem a taxa de faltas de página são atraentes. A principal técnica usada aqui é permitir o posicionamento totalmente associativo das páginas na memória.
- As faltas de página podem ser tratadas em nível de software porque o overhead será pequeno se comparado com o tempo de acesso ao disco. Além disso, o software pode se dar ao luxo de usar algoritmos inteligentes para escolher como posicionar as páginas, já que mesmo pequenas reduções na taxa de falhas compensarão o custo desses algoritmos.
- O write-through não funcionará para a memória virtual, visto que as escritas levam muito tempo. Em vez disso, os sistemas de memória virtual usam write-back.

As próximas subseções tratam desses fatores no projeto de memória virtual.

Detalhamento: Embora normalmente imaginemos os endereços virtuais como muito maiores do que os endereços físicos, o contrário pode ocorrer quando o tamanho de endereço do processador é pequeno em relação ao estado da tecnologia de memória. Nenhum programa único pode se beneficiar, mas um grupo de programas executados ao mesmo tempo pode se beneficiar de não precisar ser trocado para a memória ou de ser executado em processadores paralelos. Para computadores servidores e desktops, processadores de 32 bits já são problemáticos.

Detalhamento: A discussão da memória virtual neste livro focaliza a paginação, que usa blocos de tamanho fixo. Há também um esquema de blocos de tamanho variável chamado **segmentação**. Na segmentação, um endereço consiste em duas partes: um número de segmento e um offset de segmento. O registrador de segmento é mapeado a um endereço físico e o offset é *somado* para encontrar o endereço físico real. Como o segmento pode variar em tamanho, uma verificação de limites é necessária para garantir que o offset esteja dentro do segmento. O principal uso da segmentação é suportar métodos de proteção mais avançados e compartilhar um espaço de endereçamento. A maioria dos livros de sistemas operacionais contém extensas discussões sobre a segmentação comparada com a paginação e sobre o uso da segmentação para compartilhar logicamente o espaço de endereçamento. A principal desvantagem da segmentação é que ela divide o espaço de endereço em partes logicamente separadas que precisam ser manipuladas como um endereço de duas partes: o número de segmento e o offset. A paginação, por outro lado, torna o limite entre o número de página e o offset invisível aos programadores e compiladores.

segmentação Um esquema de mapeamento de endereço de tamanho variável em que um endereço consiste em duas partes: um número de segmento, que é mapeado para um endereço físico, e um offset de segmento.

Os segmentos também têm sido usados como um método para estender o espaço de endereçamento sem mudar o tamanho da palavra do computador. Essas tentativas têm sido malsucedidas devido à dificuldade e ao ônus de desempenho inerentes a um endereço de duas partes, dos quais os programadores e compiladores precisam estar cientes.

Muitas arquiteturas dividem o espaço de endereçamento em grandes blocos de tamanho fixo que simplificam a proteção entre o sistema operacional e os programas de usuário e aumentam a eficiência da paginação. Embora essas divisões normalmente sejam chamadas de "segmentos", esse mecanismo é muito mais simples do que a segmentação de tamanho de bloco variável e não é visível aos programas do usuário; discutiremos o assunto em mais detalhes em breve.

Posicionando uma página e a encontrando novamente

Em razão da penalidade incrivelmente alta decorrente de uma falta de página, os projetistas reduzem a frequência das faltas de página otimizando seu posicionamento. Se permitirmos que uma página virtual seja mapeada em qualquer página física, o sistema operacional, então, pode escolher substituir qualquer página que desejar quando ocorrer uma falta de página. Por exemplo, o sistema operacional pode usar um sofisticado algoritmo e complexas estruturas de dados, que monitoram o uso de páginas, para tentar escolher uma página que não será necessária por um longo tempo. A capacidade de usar um esquema de substituição inteligente e flexível reduz a taxa de faltas de página e simplifica o uso do posicionamento de páginas totalmente associativo.

Como mencionamos na Seção 5.3, a dificuldade em usar posicionamento totalmente associativo está em localizar uma entrada, já que ela pode estar em qualquer lugar no nível superior da hierarquia. Uma pesquisa completa é impraticável. Nos sistemas de memória virtual, localizamos páginas usando uma tabela que indexa a memória; essa estrutura é chamada de **tabela de páginas** e reside na memória. Uma tabela de páginas é indexada pelo número de página do endereço virtual para descobrir o número da página física correspondente. Cada programa possui sua própria tabela de páginas, que mapeia o espaço de endereçamento virtual desse programa para a memória principal. Em nossa analogia da biblioteca, a tabela de páginas corresponde a um mapeamento entre os títulos dos livros e os locais da biblioteca. Exatamente como o catálogo de cartões pode conter entradas para livros em outra biblioteca ou campus em vez da biblioteca local, veremos que a tabela de páginas pode conter entradas para páginas não presentes na memória. A fim de indicar o local da tabela de páginas na memória, o hardware inclui um registrador que aponta para o início da tabela de páginas; esse registrador é chamado de *registrador de tabela de páginas*. Por enquanto, considere que a tabela de páginas esteja em uma área fixa e contígua da memória.

tabela de páginas A tabela com as traduções de endereço virtual para físico em um sistema de memória virtual. A tabela, armazenada na memória, normalmente é indexada pelo número de página virtual; cada entrada na tabela contém o número da página física para essa página virtual se a página estiver atualmente na memória.

Interface hardware/ software

A tabela de páginas, juntamente com o contador de programa e os registradores, especifica o *estado* de um programa. Se quisermos permitir que outro programa use o processador, precisamos salvar esse estado. Mais tarde, após restaurar esse estado, o programa pode continuar a execução. Frequentemente nos referimos a esse estado como um *processo*. O

processo é considerado *ativo* quando está de posse do processador; caso contrário, ele é considerado *inativo*. O sistema operacional pode tornar um processo ativo carregando o estado do processo, incluindo o contador de programa, o que irá iniciar a execução no valor salvo do contador de programa.

O espaço de endereçamento do processo, e, consequentemente, todos os dados que ele pode acessar na memória, é definido pela sua tabela de páginas, que reside na memória. Em vez de salvar a tabela de páginas inteira, o sistema operacional simplesmente carrega o registrador de tabela de páginas de modo a apontar para a tabela de páginas do processo que ele quer tornar ativo. Cada processo possui sua própria tabela de páginas, já que diferentes processos usam os mesmos endereços virtuais. O sistema operacional é responsável por alocar a memória física e atualizar as tabelas de páginas, de modo que os espaços de endereço virtuais dos diferentes processos não colidam. Como veremos em breve, o uso de tabelas de páginas separadas também fornece proteção de um processo contra outro.

A Figura 5.21 usa o registrador de tabela de páginas, o endereço virtual e a tabela de páginas indicada para mostrar como o hardware pode formar um endereço físico. Um bit de validade é usado em cada entrada de tabela de páginas, exatamente como faríamos em uma cache. Se o bit estiver desligado, a página não está presente na memória principal e ocorre uma falta de página. Se o bit estiver ligado, a página está na memória e a entrada contém o número de página física.

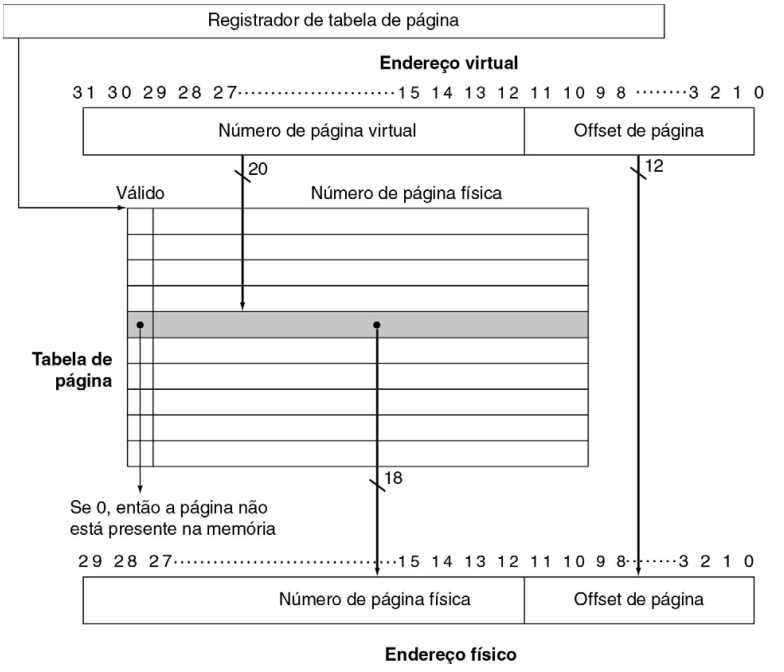

FIGURA 5.21 A tabela de páginas é indexada pelo número de página virtual para obter a parte correspondente do endereço físico. Consideramos um endereço de 32 bits. O endereço inicial da tabela de páginas é dado pelo ponteiro da tabela de páginas. Nessa figura, o tamanho de página é 2^{12} bytes, ou 4KB. O espaço de endereço virtual é 2^{32} bytes, ou 4GB, e o espaço de endereçamento físico é 2^{30} bytes, que permite uma memória principal de até 1GB. O número de entradas na tabela de páginas é 2^{20}, ou um milhão de entradas. O bit de validade para cada entrada indica se o mapeamento é legal. Se ele estiver desligado, a página não está presente na memória. Embora a entrada de tabela de páginas mostrada aqui só precise ter 19 bits de largura, ela normalmente seria arredondada para 32 bits a fim de facilitar a indexação. Os bits extras seriam usados para armazenar informações adicionais que precisam ser mantidas página a página, como a proteção.

Como a tabela de páginas contém um mapeamento para toda página virtual possível, nenhuma tag é necessária. Na terminologia da cache, o índice usado para acessar a tabela de páginas consiste no endereço de bloco inteiro, que é o número de página virtual.

Faltas de página

Se o bit de validade para uma página virtual estiver desligado, ocorre uma falta de página. O sistema operacional precisa receber o controle. Essa transferência é feita pelo mecanismo de exceção, que abordaremos posteriormente nesta seção. Quando o sistema operacional obtém o controle, ele precisa encontrar a página no próximo nível da hierarquia (geralmente o disco magnético) e decidir onde colocar a página requisitada na memória principal.

O endereço virtual por si só não diz imediatamente onde está a página no disco. Voltando à nossa analogia da biblioteca, não podemos encontrar o local de um livro nas estantes apenas sabendo seu título. Precisamos ir ao catálogo e consultar o livro, obter um endereço para o local nas estantes. Da mesma forma, em um sistema de memória virtual, é necessário monitorar o local no disco de cada página em um espaço de endereçamento virtual.

Como não sabemos de antemão quando uma página na memória será escolhida para ser substituída, o sistema operacional normalmente cria o espaço no disco para todas as páginas de um processo no momento em que ele cria o processo. Esse espaço do disco é chamado de **área de swap**. Nesse momento, o sistema operacional também cria uma estrutura para registrar onde cada página virtual está armazenada no disco. Essa estrutura de dados pode ser parte da tabela de páginas ou pode ser uma estrutura de dados auxiliar indexada da mesma maneira que a tabela de páginas. A Figura 5.22 mostra a organização quando uma única tabela contém o número de página física ou o endereço de disco.

área de swap O espaço no disco reservado para o espaço de memória virtual completo de um processo.

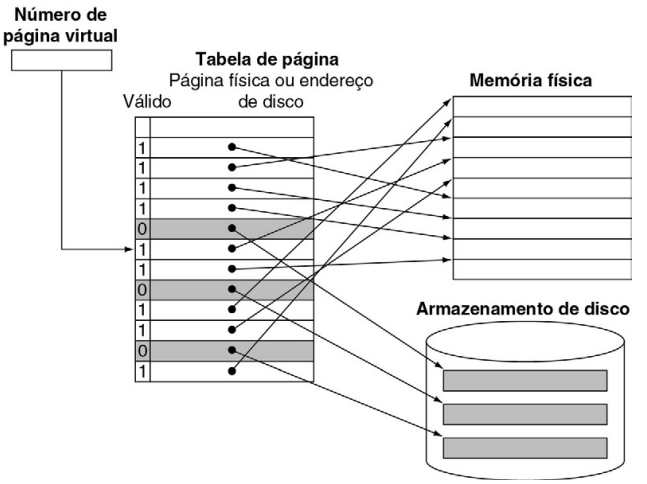

FIGURA 5.22 A tabela de páginas mapeia cada página na memória virtual em uma página na memória principal ou em uma página armazenada em disco, que é o próximo nível na hierarquia. O número de página virtual é usado para indexar a tabela de páginas. Se o bit de validade estiver ligado, a tabela de páginas fornece o número de página física (ou seja, o endereço inicial da página na memória) correspondente à página virtual. Se o bit de validade estiver desligado, a página reside atualmente apenas no disco, em um endereço de disco especificado. Em muitos sistemas, a tabela de endereços de página física e endereços de página de disco, embora sendo logicamente uma única tabela, é armazenada em duas estruturas de dados separadas. As tabelas duplas se justificam, em parte, porque precisamos manter os endereços de disco de todas as páginas, mesmo que elas estejam atualmente na memória principal. Lembre-se de que as páginas na memória principal e as páginas no disco são idênticas em tamanho.

O sistema operacional também cria uma estrutura de dados que controla quais processos e quais endereços virtuais usam cada página física. Quando ocorre uma falta de página, se todas as páginas na memória principal estiverem em uso, o sistema operacional precisa escolher uma página para substituir. Como queremos minimizar o número de faltas de página, a maioria dos sistemas operacionais tenta escolher uma página que supostamente não será necessária no futuro próximo. Usando o passado para prever o futuro, os sistemas operacionais seguem o esquema de substituição LRU (Least Recently Used – usado menos recentemente), que mencionamos na Seção 5.3. O sistema operacional procura a página usada menos recentemente, fazendo a suposição de que uma página que não foi usada por um longo período é menos provável de ser usada do que uma página acessada mais recentemente. As páginas substituídas são escritas na área de swap do disco. Caso você esteja curioso, o sistema operacional é apenas outro processo, e essas tabelas controlando a memória estão na memória; os detalhes dessa aparente contradição serão explicados em breve.

Interface hardware/ software

bit de referência Também chamado de **bit de uso**. Um campo que é ligado sempre que uma página é acessada e que é usado para implementar LRU ou outros esquemas de substituição.

Implementar um esquema de LRU completamente preciso é muito caro, pois requer atualizar uma estrutura de dados a *cada* referência à memória. Como alternativa, a maioria dos sistemas operacionais aproxima a LRU monitorando que páginas foram e que páginas não foram usadas recentemente. Para ajudar o sistema operacional a estimar as páginas LRU, alguns computadores fornecem um **bit de referência** ou **bit de uso**, que é ligado sempre que uma página é acessada. O sistema operacional limpa periodicamente os bits de referência e, depois, os registra para que ele possa determinar que páginas foram tocadas durante um determinado período. Com essas informações de uso, o sistema operacional pode selecionar uma página que está entre as referenciadas menos recentemente (detectadas tendo seu bit de referência desligado). Se esse bit não for fornecido pelo hardware, o sistema operacional precisará encontrar outra maneira de estimar que páginas foram acessadas.

Detalhamento: Com um endereço virtual de 32 bits, páginas de 4KB e 4 bytes por entrada da tabela de páginas, podemos calcular o tamanho total da tabela de páginas:

$$\text{Número de entradas da tabela de páginas} = \frac{2^{32}}{2^{12}} = 2^{20}$$

$$\text{Tamanho da tabela de páginas} = 2^{20} \text{ entradas da tabela de páginas} \; 2^{2} \frac{\text{bytes}}{\text{entrada da tabela de página}} = 4\,\text{MB}$$

Ou seja, precisaríamos usar 4MB da memória para cada programa em execução em um dado momento. Essa quantidade não é ruim para um único programa. Mas, e se houver centenas de programas rodando, cada um com sua própria tabela de página? E como devemos tratar endereços de 64 bits, que por esse cálculo precisaríamos de 2^{52} palavras?

Diversas técnicas são usadas no sentido de reduzir a quantidade de armazenamento necessária para a tabela de páginas. As cinco técnicas a seguir visam a reduzir o armazenamento máximo total necessário, bem como minimizar a memória principal dedicada às tabelas de páginas:

1. A técnica mais simples é manter um registrador de limite que restrinja o tamanho da tabela de páginas para um determinado processo. Se o número de página virtual se tornar maior do que o conteúdo do registrador de limite, entradas precisarão ser

incluídas na tabela de páginas. Essa técnica permite que a tabela de páginas cresça à medida que um processo consome mais espaço. Assim, a tabela de páginas só será maior se o processo estiver usando muitas páginas do espaço de endereçamento virtual. Essa técnica exige que o espaço de endereçamento se expanda apenas em uma direção.

2. Permitir o crescimento apenas em uma direção não é o bastante, já que a maioria das linguagens exige duas áreas cujo tamanho seja expansível: uma área contém a pilha e a outra contém o heap. Devido à essa dualidade, é conveniente dividir a tabela de páginas e deixá-la crescer do endereço mais alto para baixo, assim como do endereço mais baixo para cima. Isso significa que haverá duas tabelas de páginas separadas e dois limites separados. O uso de duas tabelas de páginas divide o espaço de endereçamento em dois segmentos. O bit mais significativo de um endereço normalmente determina que segmento – e, portanto, que tabela de páginas – deve ser usado para esse endereço. Como o segmento é especificado pelo bit de endereço mais significativo, cada segmento pode ter a metade do tamanho do espaço de endereçamento. Um registrador de limite para cada segmento especifica o tamanho atual do segmento, que cresce em unidades de páginas. Esse tipo de segmentação é usado por muitas arquiteturas, inclusive MIPS. Diferente do tipo de segmentação abordado na seção "Detalhamento" da Seção 5.4, essa forma de segmentação é invisível ao programa de aplicação, embora não para o sistema operacional. A principal desvantagem desse esquema é que ele não funciona bem quando o espaço de endereçamento é usado de uma maneira esparsa e não como um conjunto contíguo de endereços virtuais.

3. Outro método para reduzir o tamanho da tabela de páginas é aplicar uma função de hashing no endereço virtual de modo que a estrutura de dados da tabela de páginas precise ser apenas do tamanho do número de páginas *físicas* na memória principal. Essa estrutura é chamada de *tabela de páginas invertida*. É claro que o processo de consulta é um pouco mais complexo com uma tabela de páginas invertida porque não podemos mais simplesmente indexar a tabela de páginas.

4. Múltiplos níveis de tabelas de páginas também podem ser usados no sentido de reduzir a quantidade total de armazenamento para a tabela de páginas. O primeiro nível mapeia grandes blocos de tamanho fixo do espaço de endereçamento virtual, talvez de 64 a 256 páginas no total. Esses grandes blocos são, às vezes, chamados de segmentos, e essa tabela de mapeamento de primeiro nível é chamada de tabela de segmentos, embora os segmentos sejam invisíveis ao usuário. Cada entrada na tabela de segmentos indica se alguma página nesse segmento está alocada e, se estiver, aponta para uma tabela de páginas desse segmento. A tradução de endereços ocorre primeiramente olhando na tabela de segmentos, usando os bits mais significativos do endereço. Se o endereço do segmento for válido, o próximo conjunto de bits mais significativos é usado para indexar a tabela de páginas indicada pela entrada da tabela de segmentos. Esse esquema permite que o espaço de endereçamento seja usado de uma maneira esparsa (vários segmentos não contíguos podem estar ativos), sem precisar alocar a tabela de páginas inteira. Esses esquemas são particularmente úteis com espaços de endereçamento muito grandes e em sistemas de software que exigem alocação não contígua. A principal desvantagem desse mapeamento de dois níveis é o processo mais complexo para a tradução de endereços.

5. A fim de reduzir a memória principal real consumida pelas tabelas de páginas, a maioria dos sistemas modernos também permite que as tabelas de páginas sejam paginadas. Embora isso pareça complicado, esse esquema funciona usando os mesmos conceitos básicos da memória virtual e simplesmente permite que as tabelas de páginas residam no espaço de endereçamento virtual. Entretanto, há alguns problemas pequenos mas cruciais, como uma série interminável de faltas de página, que precisam ser evitadas. A forma como esses problemas são resolvidos é um tema muito detalhado e, em geral, altamente específico ao processador. Em poucas palavras, esses problemas são evitados colocando todas as tabelas de páginas no espaço de endereçamento do

sistema operacional e colocando pelo menos algumas das tabelas de páginas para o sistema em uma parte da memória principal que é fisicamente endereçada e está sempre presente – e, portanto, nunca no disco.

E quanto às escritas?

A diferença entre o tempo de acesso à cache e à memória principal é de dezenas a centenas de ciclos, e os esquemas write-through podem ser usados, embora precisemos de um buffer de escrita para ocultar do processador a latência da escrita. Em um sistema de memória virtual, as escritas no próximo nível de hierarquia (disco) levam milhões de ciclos de clock de processador; portanto, construir um buffer de escrita para permitir que o sistema escreva diretamente no disco seria impraticável. Em vez disso, os sistemas de memória virtual precisam usar write-back, realizando as escritas individuais para a página na memória e copiando a página novamente para o disco quando ela é substituída na memória.

Interface hardware/ software

Um esquema write-back possui outra importante vantagem em um sistema de memória virtual. Como o tempo de transferência de disco é pequeno comparado com seu tempo de acesso, copiar de volta uma página inteira é muito mais eficiente do que escrever palavras individuais novamente no disco. Uma operação write-back, embora mais eficiente do que transferir páginas individuais, ainda é onerosa. Portanto, gostaríamos de saber se uma página *precisa* ser copiada de volta quando escolhemos substituí-la. Para monitorar se uma página foi escrita desde que foi lida para a memória, um *bit de modificação* (dirty bit) é acrescentado à tabela de páginas. O bit de modificação é ligado quando qualquer palavra em uma página é escrita. Se o sistema operacional escolher substituir a página, o bit de modificação indica se a página precisa ser escrita no disco antes que seu local na memória possa ser cedido a outra página. Logo, uma página modificada normalmente é chamada de "dirty page".

Tornando a tradução de endereços rápida: a TLB

Como as tabelas de páginas são armazenadas na memória principal, cada acesso à memória por um programa pode levar, no mínimo, o dobro do tempo: um acesso à memória para obter o endereço físico e um segundo acesso para obter os dados. O segredo para melhorar o desempenho de acesso é basear-se na localidade da referência à tabela de páginas. Quando uma tradução para um número de página virtual é usada, ela provavelmente será necessária novamente no futuro próximo, pois as referências às palavras nessa página possuem localidade temporal e também espacial.

Assim, os processadores modernos incluem uma cache especial que controla as traduções usadas recentemente. Essa cache especial de tradução de endereços é tradicionalmente chamada de **TLB (translation-lookaside buffer)**, embora seria mais correto chamá-la de cache de tradução. A TLB corresponde àquele pequeno pedaço de papel que normalmente usamos para registrar o local de um conjunto de livros que consultamos no catálogo; em vez de pesquisar continuamente o catálogo inteiro, registramos o local de vários livros e usamos o pedaço de papel como uma cache da biblioteca.

TLB (Translation-Lookaside Buffer) Uma cache que monitora os mapeamentos de endereços recentemente usados para evitar um acesso à tabela de páginas.

A Figura 5.23 mostra que cada entrada de tag na TLB contém uma parte do número de página virtual, e cada entrada de dados da TLB contém um número de página física. Como não iremos mais acessar a tabela de páginas a cada referência, em vez disso acessaremos a TLB, que precisará incluir outros bits de status, como o bit de modificação e o bit de referência.

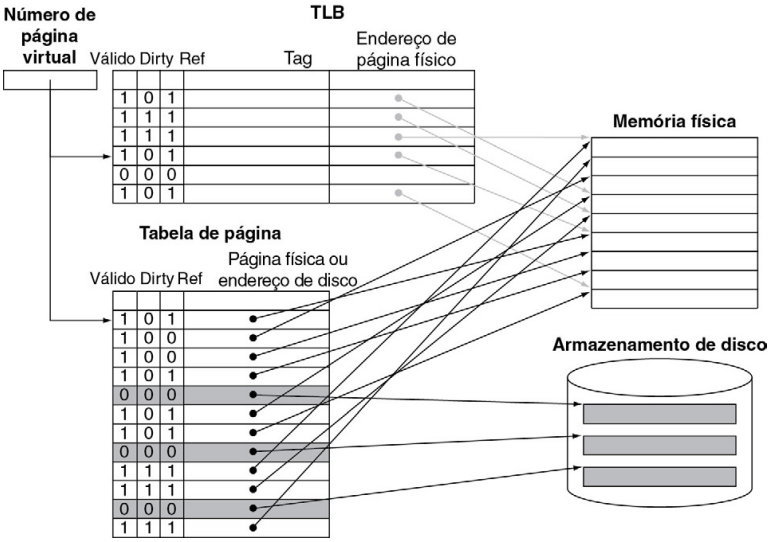

FIGURA 5.23 A TLB age como uma cache da tabela de páginas apenas para as entradas que mapeiam as páginas físicas. A TLB contém um subconjunto dos mapeamentos de página virtual para física que estão na tabela de páginas. Os mapeamentos da TLB são mostrados em destaque. Como a TLB é uma cache, ela precisa ter um campo tag. Se não houver uma entrada correspondente na TLB para uma página, a tabela de páginas precisa ser examinada. A tabela de páginas fornece um número de página física para a página (que pode, então, ser usado na construção de uma entrada da TLB) ou indica que a página reside em disco, caso em que ocorre uma falta de página. Como a tabela de páginas possui uma entrada para cada página virtual, nenhum campo tag é necessário; ou seja, ela *não* é uma cache.

Em cada referência, consultamos o número de página virtual na TLB. Se tivermos um acerto, o número de página física é usado para formar o endereço e o bit de referência correspondente é ligado. Se o processador estiver realizando uma escrita, o bit de modificação também é ligado. Se ocorrer uma falha na TLB, precisamos determinar se ela é uma falta de página ou simplesmente uma falha de TLB. Se a página existir na memória, então a falha de TLB indica apenas que a tradução está faltando. Nesse caso, o processador pode tratar a falha de TLB lendo a tradução da tabela de páginas para a TLB e, depois, tentando a referência novamente. Se a página não estiver presente na memória, então a falha de TLB indica uma falta de página verdadeira. Nesse caso, o processador chama o sistema operacional usando uma exceção. Como a TLB possui muito menos entradas do que o número de páginas na memória principal, as falhas de TLB serão muito mais frequentes do que as faltas de página verdadeiras.

As falhas de TLB podem ser tratadas no hardware ou no software. Na prática, com cuidado, pode haver pouca diferença de desempenho entre os dois métodos, uma vez que as operações básicas são iguais nos dois casos.

Depois que uma falha de TLB tiver ocorrido e a tradução faltando tiver sido recuperada da tabela de páginas, precisaremos selecionar uma entrada da TLB para substituir. Como os bits de referência e de modificação estão contidos na entrada da TLB, precisamos copiar esses bits de volta para a entrada da tabela de páginas quando substituirmos uma entrada. Esses bits são a única parte da entrada da TLB que pode ser modificada. O uso de write-back – ou seja, copiar de volta essas entradas no momento da falha e não quando são escritas – é muito eficiente, já que esperamos que a taxa de falhas da TLB seja pequena. Alguns sistemas usam outras técnicas para aproximar os bits de referência e de modificação, eliminando a necessidade de escrever na TLB exceto para carregar uma nova entrada da tabela em caso de falha.

Alguns valores comuns para uma TLB poderiam ser:

- Tamanho da TLB: 16 a 512 entradas.
- Tamanho do bloco: uma a duas entradas da tabela de páginas (geralmente 4 a 8 bytes cada uma).

- Tempo de acerto: 0,5 a 1 ciclo de clock
- Penalidade de falha: 10 a 100 ciclos de clock
- Taxa de falhas: 0,01% a 1%

Os projetistas têm usado uma ampla gama de associatividades em TLBs. Alguns sistemas usam TLBs pequenas e totalmente associativas porque um mapeamento totalmente associativo possui uma taxa de falhas mais baixa; além disso, como a TLB é pequena, o custo de um mapeamento totalmente associativo não é tão alto. Outros sistemas usam TLBs grandes, normalmente com pequena associatividade. Com um mapeamento totalmente associativo, escolher a entrada a ser substituída se torna difícil, pois é muito caro implementar um esquema de LRU de hardware. Além do mais, como as falhas de TLB são muito mais frequentes do que as faltas de página e, portanto, precisam ser tratadas de modo mais econômico, não podemos utilizar um algoritmo de software caro, como para as falhas. Como resultado, muitos sistemas fornecem algum suporte para escolher aleatoriamente uma entrada a ser substituída. Veremos os esquemas de substituição mais detalhadamente na Seção 5.5.

A TLB do Intrinsity FastMATH

Para ver essas ideias em um processador real, vamos dar uma olhada mais de perto na TLB do Intrinsity FastMATH. O sistema de memória usa páginas de 4KB e um espaço de endereçamento de 32 bits; portanto, o número de página virtual tem 20 bits de largura, como no alto da Figura 5.24. O endereço físico é do mesmo tamanho do endereço virtual. A TLB contém 16 entradas, é totalmente associativa e é compartilhada entre as referências de instruções e de dados. Cada entrada possui 64 bits de largura e contém uma tag de 20 bits (que é o número de página virtual para essa entrada de TLB), o número de página física correspondente (também 20 bits), um bit de validade, um bit de modificação e outros bits de contabilidade.

A Figura 5.24 mostra a TLB e uma das caches, enquanto a Figura 5.25 mostra as etapas no processamento de uma requisição de leitura ou escrita. Quando ocorre uma falha de TLB, o hardware MIPS salva o número de página da referência em um registrador especial e gera uma exceção. A exceção chama o sistema operacional, que trata a falha no software. Para encontrar o endereço físico da página ausente, a rotina de falha de TLB indexa a tabela de páginas usando o número de página do endereço virtual e o registrador de tabela de páginas, que indica o endereço inicial da tabela de páginas do processo ativo. Usando um conjunto especial de instruções de sistema que podem atualizar a TLB, o sistema operacional coloca o endereço físico da tabela de páginas na TLB. Uma falha de TLB leva cerca de 13 ciclos de clock, considerando que o código e a entrada da tabela de páginas estejam na cache de instruções e na cache de dados, respectivamente. (Veremos o código TLB MIPS posteriormente neste capítulo). Uma falta de página verdadeira ocorre se a entrada da tabela de páginas não possuir um endereço físico válido. O hardware mantém um índice que indica a entrada recomendada a ser substituída, escolhida aleatoriamente.

Existe uma complicação extra para requisições de escrita: o bit de acesso de escrita na TLB precisa ser verificado. Esse bit impede que o programa escreva em páginas para as quais tenha apenas acesso de leitura. Se o programa tentar uma escrita e o bit de acesso de escrita estiver desligado, uma exceção é gerada. O bit de acesso de escrita faz parte do mecanismo de proteção, que abordaremos em breve.

Integrando memória virtual, TLBs e caches

Nossos sistemas de memória virtual e de cache funcionam em conjunto como uma hierarquia, de modo que os dados não podem estar na cache a menos que estejam presentes na memória principal. O sistema operacional desempenha um importante papel na manutenção dessa hierarquia removendo o conteúdo de qualquer página da cache quando

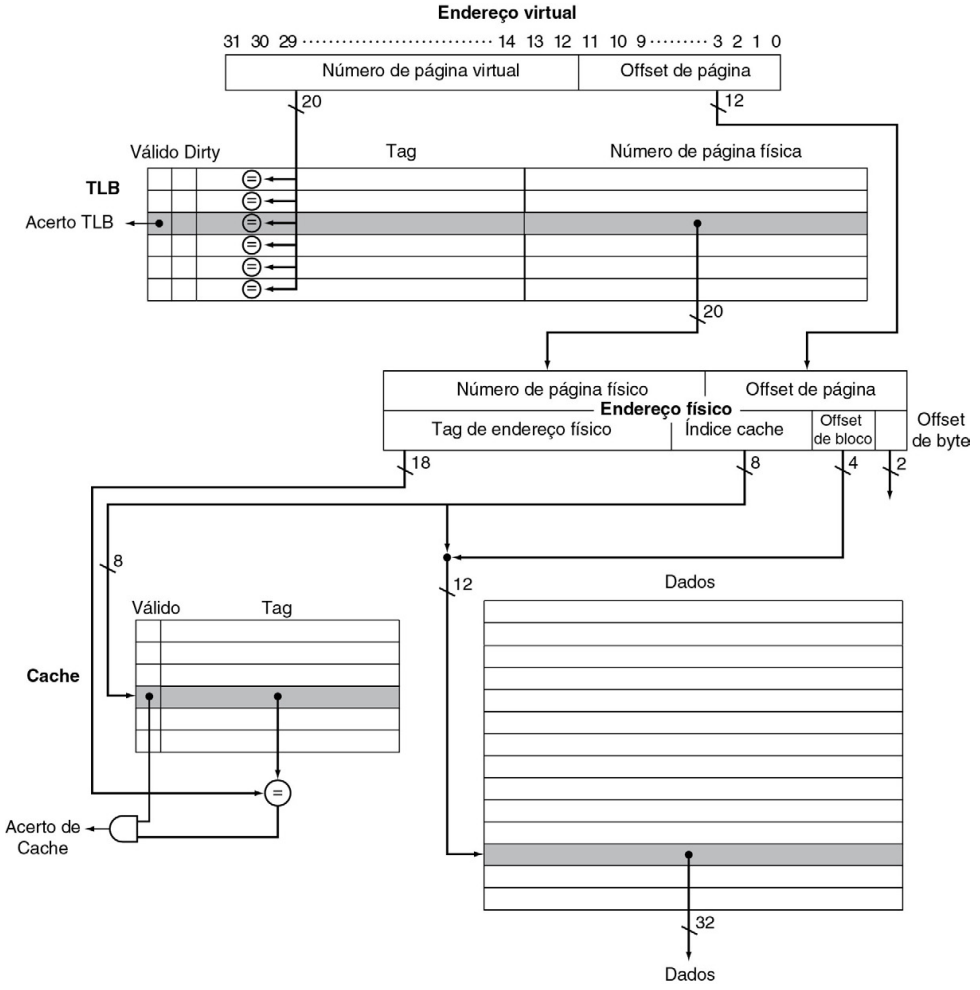

FIGURA 5.24 A TLB e a cache implementam o processo de ir de um endereço virtual para um item de dados no Intrinsity FastMATH. Essa figura mostra a organização da TLB e a cache de dados considerando um tamanho de página de 4KB. Este diagrama focaliza uma leitura; a Figura 5.25 descreve como tratar escritas. Repare que, diferente da Figura 5.9, as RAMs de tag e de dados são divididas. Endereçando a longa, mas estreita, RAM de dados com o índice de cache concatenado com o offset de bloco, selecionamos a palavra desejada no bloco sem um multiplexador 16:1. Embora a cache seja diretamente mapeada, a TLB é totalmente associativa. A implementação de uma TLB totalmente associativa exige que toda tag TLB seja comparada com o número de página virtual, já que a entrada desejada pode estar em qualquer lugar na TLB. (Ver memórias endereçáveis por conteúdo na seção "Detalhamento" na seção "Localizando um bloco na cache".) Se o bit de validade da entrada correspondente estiver ligado, o acesso será um acerto de TLB e os bits do número de página física acrescidos aos bits do offset de página formarão o índice usado para acessar a cache.

decide migrar essa página para o disco. Ao mesmo tempo, o sistema operacional modifica as tabelas de páginas e a TLB de modo que uma tentativa de acessar quaisquer dados na página migrada gere uma falta de página.

Sob as circunstâncias ideais, um endereço virtual é traduzido pela TLB e enviado para a cache em que os dados apropriados são encontrados, recuperados e devolvidos ao processador. No pior caso, uma referência pode falhar em todos os três componentes da hierarquia de memória: a TLB, a tabela de páginas e a cache. O exemplo a seguir ilustra essas interações em mais detalhes.

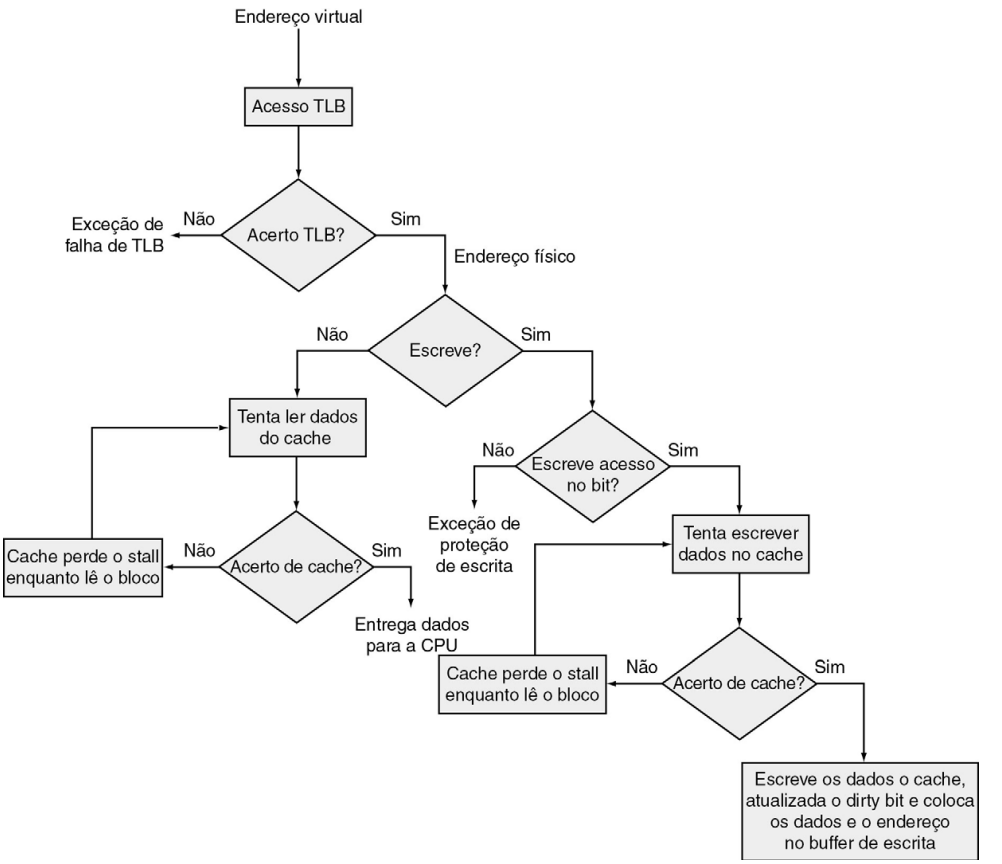

FIGURA 5.25 Processando uma leitura ou uma escrita direta na TLB e na cache do Intrinsity FastMATH. Se a TLB gerar um acerto, a cache pode ser acessada com o endereço físico resultante. Para uma leitura, a cache gera um acerto ou uma falha e fornece os dados ou causa um stall enquanto os dados são trazidos da memória. Se a operação for uma escrita, uma parte da entrada de cache é substituída por um acerto e os dados são enviados ao buffer de escrita se considerarmos uma cache write-through. Uma falha de escrita é exatamente como uma falha de leitura exceto que o bloco é modificado após ser lido da memória. Uma cache write-back requer que as escritas liguem um bit de modificação para o bloco de cache; além disso, um buffer de escrita é carregado com o bloco inteiro apenas em uma falha de leitura ou falha de escrita se o bloco a ser substituído estiver com o bit de modificação ligado. Observe que um acerto de TLB e um acerto de cache são eventos independentes, mas um acerto de cache só pode ocorrer após um acerto de TLB, o que significa que os dados precisam estar presentes na memória. A relação entre as falhas de TLB e as falhas de cache é examinada mais a fundo no exemplo a seguir e nos exercícios no final do capítulo.

EXEMPLO

Operação geral de uma hierarquia de memória

Em uma hierarquia de memória como a da Figura 5.24, que inclui uma TLB e uma cache organizada como mostrado, uma referência de memória pode encontrar três tipos de falhas diferentes: uma falha de TLB, uma falta de página e uma falha de cache. Considere todas as combinações desses três eventos com uma ou mais ocorrendo (sete possibilidades). Para cada possibilidade, diga se esse evento realmente pode ocorrer e sob que circunstâncias.

RESPOSTA

A Figura 5.26 mostra as circunstâncias possíveis e se elas podem ou não surgir na prática.

TLB	Tabela de página	Cache	Possível? Se sim, sob quais circunstâncias?
Acerta	Acerta	Erra	Possível, apesar da tabela de página nunca ser conferida se a TLB acertar.
Erra	Acerta	Acerta	TLB erra, mas entrada é encontrada na tabela de página; depois de nova tentativa, dado é encontrado no cache.
Erra	Acerta	Erra	TLB erra, mas entrada é encontrada na tabela de página; depois de nova tentativa, dado erra o cache.
Erra	Erra	Erra	TLB erra e é seguido de erro de página; depois de nova tentativa, dado deve errar o cache.
Acerta	Erra	Erra	Impossível: não pode haver tradução na TLB se a página não está presente na memória.
Acerta	Erra	Acerta	Impossível: não pode haver tradução na TLB se a página não está presente na memória.
Erra	Erra	Acerta	Impossível: dados não podem ser permitidos no cache se a página não está presente na memória.

FIGURA 5.26 As possíveis combinações de eventos na TLB, no sistema de memória virtual e na cache. Três dessas combinações são impossíveis e uma é possível (acerto de TLB, acerto de memória virtual, falha de cache), mas nunca detectada.

Detalhamento: A Figura 5.26 considera que todos os endereços de memória são traduzidos para endereços físicos antes que a cache seja acessada. Nessa organização, a cache é *fisicamente indexada* e *fisicamente rotulada* (tanto o índice quanto a tag de cache são endereços físicos em vez de virtuais). Nesse sistema, a quantidade de tempo para acessar a memória, considerando um acerto de cache, precisa acomodar um acesso de TLB e um acesso de cache; naturalmente, esses acessos podem ser em pipeline.

Como alternativa, o processador pode indexar a cache com um endereço que seja completa ou parcialmente virtual. Isso é chamado de **cache virtualmente endereçada** e usa tags que são endereços virtuais; portanto, esse tipo de cache é *virtualmente indexado* e *virtualmente rotulado*. Nessas caches, o hardware de tradução de endereço (TLB) não é usado durante o acesso de cache normal, já que a cache é acessada com um endereço virtual que não foi traduzido para um endereço físico. Isso tira a TLB do caminho crítico, reduzindo a latência da cache. Quando ocorre uma falha de cache, no entanto, o processador precisa traduzir o endereço para um endereço físico de modo que ele possa buscar o bloco de cache da memória principal.

cache virtualmente endereçada Uma cache acessada com um endereço virtual em vez de um endereço físico.

Quando a cache é acessada com um endereço virtual e páginas são compartilhadas entre programas (que podem acessá-las com diferentes endereços virtuais), há a possibilidade de **aliasing**. O aliasing ocorre quando o mesmo objeto possui dois nomes – nesse caso, dois endereços virtuais para a mesma página. Essa ambiguidade cria um problema porque uma palavra nessa página pode ser colocada na cache em dois locais diferentes, cada um correspondendo a diferentes endereços virtuais. Essa ambiguidade permitiria que um programa escrevesse os dados sem que o outro programa soubesse que eles foram mudados. As caches endereçadas completamente por endereços virtuais apresentam limitações de projeto na cache e na TLB para reduzir o aliasing ou exigem que o sistema operacional (e possivelmente o usuário) tome ações para garantir que o aliasing não ocorra.

aliasing Uma situação em que o mesmo objeto é acessado por dois endereços; pode ocorrer na memória virtual quando existem dois endereços virtuais para a mesma página física.

Uma conciliação comum entre esses dois pontos de projeto são as caches virtualmente indexadas (algumas vezes, usando apenas a parte do offset de página do endereço, que é um endereço físico, já que não é traduzida), mas usam tags físicas. Esses projetos, que são *virtualmente indexados mas fisicamente rotulados*, tentam unir as vantagens de desempenho das caches virtualmente indexadas às vantagens da arquitetura mais simples de uma **cache fisicamente endereçada**. Por exemplo, não existe qualquer problema de aliasing nesse caso. A Figura 5.24 considerou um tamanho de página de 4KB, mas na realidade ela tem 16KB, de modo que o Intrinsity FastMATH pode usar esse truque. Para isso, é preciso haver uma cuidadosa coordenação entre o tamanho de página mínimo, o tamanho da cache e a associatividade.

cache fisicamente endereçada Uma cache endereçada por um endereço físico.

Implementando proteção com memória virtual

Uma das funções mais importantes da memória virtual é permitir o compartilhamento de uma única memória principal por diversos processos, enquanto fornece proteção de memória entre esses processos e o sistema operacional. O mecanismo de proteção precisa garantir que, embora vários processos estejam compartilhando a mesma memória principal, um processo rebelde não pode escrever no espaço de endereçamento de outro processo do usuário ou no sistema operacional, intencionalmente ou não. O bit de acesso de escrita na TLB pode proteger uma página de ser escrita. Sem esse nível de proteção, os vírus de computador seriam ainda mais comuns.

Interface hardware/ software

Para permitir que o sistema operacional implemente proteção no sistema de memória virtual, o hardware precisa fornecer pelo menos três capacidades básicas resumidas a seguir.

modo supervisor Também chamado de **modo de kernel**. Um modo que indica que um processo executado é um processo do sistema operacional.

1. Suportar pelo menos dois modos que indicam se o processo em execução é de usuário ou de sistema operacional, normalmente chamado de processo **supervisor**, processo de **kernel** ou processo *executivo*.
2. Fornecer uma parte do estado do processador que um processo de usuário pode ler mas não escrever. Isso inclui o bit de modo usuário/supervisor, que determina se o processador está no modo usuário ou supervisor, o ponteiro para a tabela de páginas e a TLB. Para escrever esses elementos, o sistema operacional usa instruções especiais que só estão disponíveis no modo supervisor.
3. Fornecer mecanismos pelos quais o processador pode passar do modo usuário para o modo supervisor e vice-versa. A primeira direção normalmente é conseguida por uma exceção de **chamada ao sistema**, implementada como uma instrução especial (*syscall* no conjunto de instruções MIPS) que transfere o controle para um local dedicado no espaço de código supervisor. Como em qualquer outra exceção, o contador de programa do ponto da chamada de sistema é salvo no PC de exceção (EPC), e o processador é colocado no modo supervisor. Para retornar ao modo usuário da exceção, use a instrução *return from exception* (ERET), que retorna ao modo usuário e desvia para o endereço no EPC.

chamada ao sistema Uma instrução especial que transfere o controle do modo usuário para um local dedicado no estágio de código supervisor, chamando o mecanismo de exceção no processo.

Usando esses mecanismos e armazenando as tabelas de páginas no espaço de endereçamento do sistema operacional, o sistema operacional pode mudar as tabelas de páginas enquanto impede que um processo do usuário as modifique, garantindo que um processo do usuário só possa acessar o armazenamento fornecido pelo sistema operacional.

Também queremos evitar que um processo leia os dados de outro processo. Por exemplo, não desejamos que o programa de um aluno leia as notas enquanto elas estiverem na memória do processador. Uma vez que começamos a compartilhar a memória principal, precisamos fornecer a capacidade de um processo proteger seus dados de serem lidos e escritos por outro processo; caso contrário, a memória principal será um poço de permissividade!

Lembre-se de que cada processo possui seu próprio espaço de endereçamento virtual. Portanto, se o sistema operacional mantiver as tabelas de páginas organizadas de modo que as páginas virtuais independentes mapeiem as páginas físicas separadas, um processo não será capaz de acessar os dados de outro. É claro que isso exige que um processo de usuário seja incapaz de mudar o mapeamento da tabela de páginas. O sistema operacional pode garantir segurança se ele impedir que o processo do usuário modifique suas próprias tabelas de páginas. No entanto, o sistema operacional precisa ser capaz de modificar as tabelas de páginas. Colocar as tabelas de páginas no espaço de endereçamento protegido do sistema operacional satisfaz a ambos os requisitos.

Quando os processos querem compartilhar informações de uma maneira limitada, o sistema operacional precisa assisti-los, já que o acesso às informações de outro processo exige mudar a tabela de páginas do processo que está acessando. O bit de acesso de escrita pode ser usado para restringir o compartilhamento apenas à leitura e, como o restante da tabela de páginas, esse bit pode ser mudado apenas pelo sistema operacional. Para permitir que outro processo, digamos, P1, leia uma página pertencente ao processo P2, P2 pediria ao sistema operacional para criar uma entrada na tabela de páginas para uma página virtual no espaço de endereço de P1 que aponte para a mesma página física que P2 deseja compartilhar. O sistema operacional poderia usar o bit de proteção de escrita a fim de impedir que P1 escrevesse os dados, se esse fosse o desejo de P2. Quaisquer bits que determinam os direitos de acesso a uma página precisam ser incluídos na tabela de páginas e na TLB, pois a tabela de páginas é acessada apenas em uma *falha* de TLB.

Detalhamento: Quando o sistema operacional decide deixar de executar o processo P1 para executar o processo P2 (o que chamamos de **troca de contexto** ou *troca de processo*), ele precisa garantir que P2 não possa ter acesso às tabelas de páginas de P1 porque isso comprometeria a proteção. Se não houver uma TLB, basta mudar o registrador de tabela de páginas de modo que aponte para a tabela de páginas de P2 (em vez da de P1); com uma TLB, precisamos limpar as entradas de TLB que pertencem a P1 – tanto para proteger os dados de P1 quanto para forçar a TLB a carregar as entradas para P2. Se a taxa de troca de processos fosse alta, isso poderia ser bastante ineficiente. Por exemplo, P2 poderia carregar apenas algumas entradas de TLB antes que o sistema operacional trocasse novamente para P1. Infelizmente, P1, então, descobriria que todas as suas entradas de TLB desapareceram e precisaria pagar falhas de TLB para recarregá-las. Esse problema ocorre porque os endereços virtuais usados por P1 e P2 são iguais e precisamos limpar a TLB a fim de evitar confundir esses endereços.

troca de contexto Uma mudança no estado interno do processador para permitir que um processo diferente use o processador, o que inclui salvar o estado necessário e retornar ao processo sendo atualmente executado.

Uma alternativa comum é estender o espaço de endereçamento virtual acrescentando um *identificador de processo* ou *identificador de tarefa*. O Intrinsity FastMATH possui um campo ID do espaço de endereçamento (ASID) de 8 bits para essa finalidade. Esse pequeno campo identifica o processo que está atualmente sendo executado; ele é mantido em um registrador carregado pelo sistema operacional quando muda de processo. O identificador de processo é concatenado com a parte da tag da TLB, de modo que um acerto de TLB ocorra apenas se o número de página e o identificador de processo corresponderem. Essa combinação elimina a necessidade de limpar a TLB, exceto em raras ocasiões.

Problemas semelhantes podem ocorrer para uma cache, já que, em uma troca de processo, a cache conterá dados do processo em execução. Esses problemas surgem de diferentes maneiras para caches física e virtualmente endereçadas; além disso, uma variedade de soluções diferentes, como identificadores de processo, são usadas para garantir que um processo obtenha seus próprios dados.

Tratando falhas de TLB e faltas de página

Embora a tradução de endereços físicos para virtuais com uma TLB seja simples quando temos um acerto de TLB, o tratamento de falhas de TLB e de faltas de página é mais complexo. Uma falha de TLB ocorre quando nenhuma entrada na TLB corresponde a um endereço virtual. Uma falha de TLB pode indicar uma de duas possibilidades:

1. A página está presente na memória e precisamos apenas criar a entrada de TLB ausente.
2. A página não está presente na memória e precisamos transferir o controle para o sistema operacional a fim de lidar com uma falta de página.

Como saber qual dessas duas circunstâncias ocorreu? Quando processarmos a falha de TLB, iremos procurar uma entrada na tabela de páginas para ser trazida para a TLB. Se a entrada na tabela de páginas correspondente tiver um bit de validade que esteja desligado, a página correspondente não está na memória e temos uma falta de página em vez de uma

simples falha de TLB. Se o bit de validade estiver ligado, podemos simplesmente recuperar a entrada desejada.

Uma falha de TLB pode ser tratada por software ou por hardware, pois ela exigirá apenas uma curta sequência de operações que copia uma entrada válida da tabela de páginas da memória para a TLB. O MIPS tradicionalmente trata uma falha de TLB por software. Ele traz a entrada da tabela de páginas da memória e, depois, executa novamente a instrução que causou a falha de TLB. Na reexecução, ele terá um acerto de TLB. Se a entrada da tabela de páginas indicar que a página não está na memória, dessa vez ele terá uma exceção de falta de página.

Tratar uma falha de TLB ou uma falta de página requer o uso do mecanismo de exceção para interromper o processo ativo, transferir o controle ao sistema operacional e, depois, retomar a execução do processo interrompido. Uma falta de página será reconhecida em algum momento durante o ciclo de clock usado para acessar a memória. A fim de reiniciar a instrução após a falta de página ser tratada, o contador de programa da instrução que causou a falta de página precisa ser salvo. Assim como no Capítulo 4, o contador de programa de exceção (EPC) é usado para conter esse valor.

Além disso, uma falha de TLB ou uma exceção de falta de página precisa ser sinalizada no final do mesmo ciclo de clock em que ocorre o acesso à memória, de modo que o próximo ciclo de clock começará o processamento da exceção em vez de continuar a execução normal das instruções. Se a falta de página não fosse reconhecida nesse ciclo de clock, uma instrução load poderia substituir um registrador, e isso poderia ser desastroso quando tentássemos reiniciar a instrução. Por exemplo, considere a instrução `lw $1,0($1)`: o computador precisa ser capaz de impedir que o estágio de escrita do resultado do pipeline ocorra; caso contrário, ele não poderia reiniciar corretamente a instrução, já que o conteúdo de `$1` teria sido destruído. Uma complicação parecida surge nos stores. Precisamos impedir que a escrita na memória realmente seja concluída quando há uma falta de página; isso normalmente é feito desativando a linha de controle de escrita para a memória.

Registrador	Número de registradores CP0	Descrição
EPC	14	Onde reiniciar depois de exceção
Cause	13	Causa da exceção
BadVAddr	8	Endereço que causou exceção
Index	0	Local na TLB a ser lido ou escrito
Random	1	Local pseudorrandômico no TLB
EntryLo	2	Endereço de página físico e flags
EntryHi	10	Endereço de página virtual
Context	4	Endereço de tabela de página e número de página

FIGURA 5.27 Registradores de controle MIPS. Considera-se que estes estejam no coprocessador 0, e por isso são lidos com `mfc0` e escritos com `mtc0`.

Interface hardware/software

Entre o momento em que começamos a executar o tratamento de exceção no sistema operacional e o momento em que o sistema operacional salvou todo o estado do processo, o sistema operacional se torna particularmente vulnerável. Por exemplo, se outra exceção ocorresse quando estivéssemos processando a primeira exceção no sistema operacional, a unidade de controle substituiria o contador de programa de exceção, tornando impossível voltar para a instrução que causou a falta de página! Podemos evitar esse desastre fornecendo a capacidade de desabilitar e **habilitar exceções**. Assim que uma exceção ocorre, o processador liga um bit que desabilita todas as outras exceções; isso poderia acontecer ao mesmo tempo em que o processador liga o bit de modo supervisor. O sistema operacional, então, salva o estado apenas suficiente para lhe permitir se recuperar se outra exceção ocorrer – a saber, os registradores do contador de programa de exceção (EPC) e Cause. EPC e Cause são dois dos registradores de controle especiais que ajudam com exceções, falhas de TLB e faltas de página; a Figura 5.27 mostra o restante.

habilitar exceção Também chamado de "habilitar interrupção". Uma ação ou sinal que controla se o processo responde ou não a uma exceção; necessário para evitar a ocorrência de exceções durante intervalos antes que o processador tenha seguramente salvado o estado necessário para a reinicialização.

O sistema operacional, então, pode habilitar novamente as exceções. Essas etapas asseguram que as exceções não façam com que o processador perca qualquer estado e, portanto, sejam incapazes de reiniciar a execução da instrução interruptora.

Uma vez que o sistema operacional conhece o endereço virtual que causou a falta de página, ele precisa completar três etapas:

1. Consultar a entrada de tabela de páginas usando o endereço virtual e encontrar o local em disco da página referenciada.
2. Escolher uma página física a ser substituída; se a página escolhida estiver com o bit de modificação ligado, ela precisará ser escrita no disco antes que possamos definir uma nova página virtual para essa página física.
3. Iniciar uma leitura de modo a trazer a página referenciada do disco para a página física escolhida.

É claro que essa última etapa levará milhões de ciclos de clock de processador (assim como a segunda, se a página substituída estiver com o bit de modificação ligado); portanto, o sistema operacional normalmente selecionará outro processo para executar no processador até que o acesso ao disco seja concluído. Como o sistema operacional salvou o estado do processo, ele pode passar o controle do processador à vontade para outro processo.

Quando a leitura da página do disco está completa, o sistema operacional pode restaurar o estado do processo que causou originalmente a falta de página e executar a instrução que retorna da exceção. Essa instrução irá redefinir o processador do modo kernel para o modo usuário, bem como restaurar o contador de programa. O processo do usuário, então, reexecuta a instrução que causou a falta de página, acessa a página requisitada com sucesso e continua a execução.

As exceções de falta de página para acessos a dados são difíceis de implementar corretamente em um processador devido a uma combinação de três características:

1. Elas ocorrem no meio das instruções, diferente das faltas de página de instruções.
2. A instrução não pode ser completada antes que a exceção seja tratada.
3. Após tratar a exceção, a instrução precisa ser reinicializada como se nada tivesse ocorrido.

Tornar **instruções reinicializáveis**, de modo que a exceção possa ser tratada e a instrução possa ser continuada, é relativamente fácil em uma arquitetura como o MIPS. Como cada instrução escreve apenas um item de dados e essa escrita ocorre no final do ciclo da instrução, podemos simplesmente impedir que a instrução seja concluída (não escrevendo) e reinicializar a instrução no começo.

instrução reinicializável Uma instrução que pode retomar a execução após uma exceção ser resolvida sem que a exceção afete o resultado da instrução.

Vejamos o MIPS mais de perto. Quando uma falha de TLB ocorre, o hardware do MIPS salva o número de página da referência em um registrador especial chamado `BadVAddr` e gera uma exceção.

A exceção chama o sistema operacional, que trata a falha por software. O controle é transferido para o endereço 8000 0000_{hexa} (o local do **handler** da falha de TLB). A fim de encontrar o endereço físico para a página ausente, a rotina de falha de TLB indexa a tabela de páginas usando o número de página do endereço virtual e o registrador de tabela de páginas, que indica o endereço inicial da tabela de páginas do processo ativo. Para tornar essa indexação rápida, o hardware do MIPS coloca tudo que você precisa no registrador especial *`Context:`* os 12 bits mais significativos têm o endereço da base da tabela de páginas e os próximos 18 bits têm o endereço virtual da página ausente. Como cada entrada de tabela de páginas possui uma palavra, os últimos dois bits são 0. Portanto, as duas primeiras instruções copiam o registrador Context para o registrador temporário do kernel `$k1` e, depois, carregam a entrada de tabela de páginas desse endereço em `$k1`. Lembre-se de

handler Nome de uma rotina de software chamada para "tratar" uma exceção ou interrupção.

que `$k0` e `$k1` são reservados para uso do sistema operacional sem salvamento; um importante motivo dessa convenção é tornar rápido o handler de falha de TLB. A seguir está o código MIPS para um handler de falha de TLB típico:

```
FalhaDeTLB:
  mfc0  $k1,Context   # copia o endereço de PTE para temp $k1
  lw    $k1, 0($k1)   # coloca PTE em temp $k1
  mtc0  $k1,EntryLo   # coloca PTE no registrador especial EntryLo
  mfc0  $k1,Context   # coloca EntryLo na entrada de TLB em Random
  eret                # retorna da exceção de falha de TLB
```

Como mostrado anteriormente, o MIPS possui um conjunto especial de instruções de sistema que atualiza a TLB. A instrução *tlbwr* copia o registrador de controle *EntryLo* para a entrada de TLB selecionada pelo registrador de controle *Random*. *Random* implementa uma substituição aleatória e, portanto, é basicamente um contador de execução livre. Uma falha de TLB leva cerca de 12 ciclos de clock.

Observe que o handler de falha de TLB não verifica se a entrada de tabela de páginas é válida. Como a exceção para a entrada de TLB ausente é muito mais frequente do que uma falta de página, o sistema operacional carrega a TLB da tabela de páginas sem examinar a entrada e reinicializa a instrução. Se a entrada for inválida, ocorre outra exceção diferente, e o sistema operacional reconhece a falta de página. Esse método torna rápido o caso frequente de uma falha de TLB, com uma pequena penalidade de desempenho para o raro caso de uma falta de página.

Uma vez que o processo que gerou a falta de página tenha sido interrompido, ele transfere o controle para $8000\ 0180_{hexa}$, um endereço diferente do handler de falha de TLB. Esse é o endereço geral para exceção; a falha de TLB possui um ponto de entrada especial que reduz a penalidade para uma falha de TLB. O sistema operacional usa o registrador Cause de exceção a fim de diagnosticar a causa da exceção. Como a exceção é uma falta de página, o sistema operacional sabe que será necessário um processamento extenso. Portanto, diferente de uma falha de TLB, ele salva todo o estado do processo ativo. Esse estado inclui todos os registradores de uso geral e de ponto flutuante, o registrador de endereço de tabela de páginas, o EPC e o registrador Cause de exceção. Como os handlers de exceção normalmente não usam os registradores de ponto flutuante, o ponto de entrada geral não os salva, deixando isso para os poucos handlers que precisam deles.

A Figura 5.28 esboça o código MIPS de um handler de exceção. Note que salvamos e restauramos o estado no código MIPS, tomando cuidado quando habilitamos e desabilitamos exceções, mas chamamos código C para tratar a exceção em particular.

O endereço virtual que causou a falta de página depende se essa foi uma falta de instruções ou de dados. O endereço da instrução que gerou a falta está no EPC. Se ela fosse uma falta de página de instruções, o EPC contém o endereço virtual da página que gerou a falta; caso contrário, o endereço virtual que gerou a falta pode ser calculado examinando a instrução (cujo endereço está no EPC) para encontrar o registrador base e o campo offset.

Detalhamento: Essa versão simplificada considera que o stack pointer (sp) é válido. Para evitar o problema de uma falta de página durante esse código de exceção de baixo nível, o MIPS separa uma parte do seu espaço de endereçamento que não pode ter faltas de página, chamada **não mapeada** (unmapped). O sistema operacional insere o código para o ponto de entrada do tratamento de exceções e a pilha de exceção na memória não mapeada. O hardware MIPS traduz os endereços virtuais $8000\ 0000_{hexa}$ a $BFFF\ FFFF_{hexa}$ para endereços físicos simplesmente ignorando os bits superiores do endereço virtual, colocando, assim, esses endereços na parte inferior da memória física. Portanto, o sistema operacional coloca os pontos de entrada dos tratamentos de exceções e as pilhas de exceção na memória não mapeada.

não mapeada Uma parte do espaço de endereçamento que não pode ter faltas de página.

Detalhamento: O código na Figura 5.28 mostra a sequência de retorno da exceção do MIPS-32. O MIPS-I usa `rfe` e `jr` em vez de `eret`.

Salva Estado			
Salva GPR	addi	$k1,$sp, -XCPSIZE	# reserva espaço na pilha para o estado
	sw	$sp, XCT_SP($k1)	# salva $sp na pilha
	sw	$v0, XCT_V0($k1)	# salva $v0 na pilha
	...		# salva $v1, $ai, $si, $ti, ...na pilha
	sw	$ra, XCT_RA($k1)	# salva $ra na pilha
Salva Hi, Lo	mfhi	$v0	# copia Hi
	mflo	$v1	# copia Lo
	sw	$v0, XCT_HI($k1)	# salva valor de Hi na pilha
	sw	$v1, XCT_LO($k1)	# salva valor de Lo na pilha
Salva registradores de exceção	mfc0	$a0, $cr	# copia registrador Cause
	sw	$a0, XCT_CR($k1)	# salva valor de $cr na pilha
	...		# salva $v1,...
	mfc0	$a3, $sr	# copia registrador de status
Atribui novo valor a sp	sw	$a3, XCT_SR($k1)	# salva $sr na pilha
	move	$sp, $k1	# sp = sp - XCPSIZE
Habilita exceções aninhadas			
	andi	$v0, $a3, MASK1	# $v0 = $sr & MASK1, habilita exceções
	mtc0	$v0, $sr	# $sr = valor que habilita exceções
Chama handler de exceção em C			
Chama código em C	move	$a0, $sp	# arg1 = ponteiro para pilha de exceção
Atribui novo valor a $gp	jal	xcpt_deliver	# chama código em C para tratar exceção
	move	$gp, GPINIT	# $gp aponta para área do heap
Restaura estado			
Restaura a maioria dos registradores, Hi, Lo	move	$at, $sp	# valor temporário de $sp
	lw	$ra, XCT_RA($at)	# restaura $ra da pilha
	...		# restaura $t0, ..., $a1
	lw	$a0, XCT_A0($k1)	# restaura $a0 da pilha
Restaura registrador do status	lw	$v0, XCT_SR($at)	# carrega $sr antigo da pilha
	li	$v1, MASK2	# máscara para inabilitar exceções
	and	$v0, $v0, $v1	# $v0 = $sr & MASK2, inabilita exceções
	mtc0	$v0, $sr	# restaura o valor do registrador de sta
Retorna da exceção			
Restaura $sp e o restante dos registradores usados como registradores temporários	lw	$sp, XCT_SP($at)	# restaura $sp da pilha
	lw	$v0, XCT_V0($at)	# restaura $v0 da pilha
	lw	$v1, XCT_V1($at)	# restaura $v1 da pilha
	lw	$k1, XCT_EPC($at)	# copia $epc antigo da pilha
	lw	$at, XCT_AT($at)	# restaura $at da pilha
Restaura EPC e retorna	mtc0	$k1, $epc	# restaura $epc
	eret	$ra	# retorna para a instrução interrompida

FIGURA 5.28 Código MIPS para salvar e restaurar o estado em uma exceção.

Detalhamento: Para processadores com instruções mais complexas, que podem tocar em muitos locais de memória e escrever muitos itens de dados, tornar as instruções reiniciáveis é muito mais difícil. Processar uma instrução pode gerar uma série de faltas de página no meio da instrução. Por exemplo, os processadores x86 possuem instruções de movimento em bloco que tocam em milhares de palavras de dados. Nesses processadores, as instruções normalmente não podem ser reiniciadas desde o início, como fazemos para instruções MIPS. Em vez disso, a instrução precisa ser interrompida e mais tarde continuada no meio de sua execução. Retomar uma instrução no meio de sua execução normalmente exige salvar algum estado especial, processar a exceção e restaurar esse estado especial. Para que isso seja feito corretamente, é preciso haver uma coordenação cuidadosa e detalhada entre o código de tratamento de exceção no sistema operacional e o hardware.

Resumo

Memória virtual é o nome para o nível da hierarquia de memória que controla a cache entre a memória principal e o disco. A memória virtual permite que um único programa expanda seu espaço de endereçamento para além dos limites da memória principal. Mais

importante, a memória virtual suporta o compartilhamento da memória principal entre vários processos simultaneamente ativos, de uma maneira protegida.

Gerenciar a hierarquia de memória entre a memória principal e o disco é uma tarefa difícil devido ao alto custo das faltas de página. Várias técnicas são usadas para reduzir a taxa de falhas:

1. As páginas são ampliadas para tirar proveito da localidade espacial e para reduzir a taxa de falhas.
2. O mapeamento entre endereços virtuais e endereços físicos, que é implementado com uma tabela de páginas, é feito totalmente associativo para que uma página virtual possa ser colocada em qualquer lugar na memória principal.
3. O sistema operacional usa técnicas, como LRU e um bit de referência, para escolher que páginas substituir.

Como as gravações no disco são caras, a memória virtual usa um esquema write-back e também monitora se uma página foi modificada (usando um bit de modificação) para evitar gravar páginas não alteradas novamente no disco.

O mecanismo de memória virtual fornece tradução de endereços de um endereço virtual usado pelo programa para o espaço de endereçamento físico usado no acesso à memória. Essa tradução de endereços permite compartilhamento protegido da memória principal e oferece várias vantagens adicionais, como a simplificação da alocação de memória. Para garantir que os processos sejam protegidos uns dos outros, é necessário que apenas o sistema operacional possa mudar as traduções de endereços, o que é implementado impedindo que programas de usuário alterem as tabelas de páginas. O compartilhamento controlado das páginas entre processos pode ser implementado com a ajuda do sistema operacional e dos bits de acesso na tabela de páginas que indicam se o programa do usuário possui acesso de leitura ou escrita à página.

Se um processador precisasse acessar uma tabela de páginas residente na memória para traduzir cada acesso, a memória virtual seria muito dispendiosa e a cache não teria sentido! Em vez disso, uma TLB age como uma cache para traduções da tabela de páginas. Os endereços são, então, traduzidos do virtual para o físico usando as traduções na TLB.

As caches, a memória virtual e as TLBs se baseiam em um conjunto comum de princípios e políticas. A próxima seção aborda essa estrutura comum.

Entendendo o desempenho dos programas

Embora a memória virtual tenha sido criada para permitir que uma memória pequena aja como uma grande, a diferença de desempenho entre o disco e a memória significa que se um programa acessa rotineiramente mais memória virtual do que a memória física que possui, sua execução será muito lenta. Esse programa estaria continuamente trocando páginas entre a memória e o disco, o que chamamos de *thrashing*. O thrashing, embora raro, é um desastre quando ocorre. Se seu programa realiza thrashing, a solução mais fácil é executá-lo em um computador com mais memória ou comprar mais memória para o computador. Uma opção mais complexa é reexaminar suas estruturas de dados e algoritmo para ver se você pode mudar a localidade e, portanto, reduzir o número de páginas que seu programa usa simultaneamente. Esse conjunto de páginas é informalmente chamado de *working set*.

Um problema de desempenho mais comum são as falhas de TLB. Como uma TLB pode tratar apenas de 32 a 64 entradas de página ao mesmo tempo, um programa poderia facilmente ver uma alta taxa de falhas de TLB, já que o processador pode acessar menos de um quarto de megabyte diretamente: $64 \times 4\text{KB} = 0{,}25\text{MB}$. Por exemplo, as falhas de TLB normalmente são um problema para o Radix Sort. A fim de tentar amenizar esse problema, a maioria das arquiteturas de computadores agora suporta tamanhos de página variáveis. Por exemplo, além da página de 4KB padrão, o hardware do MIPS suporta

páginas de 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB e 256MB. Consequentemente, se um programa usa grandes tamanhos de página, ele pode acessar mais memória diretamente sem falhas de TLB.

Na prática, o problema é fazer o sistema operacional permitir que os programas selecionem esses tamanhos de página maiores. Mais uma vez, a solução mais complexa para reduzir as falhas de TLB é reexaminar as estruturas de dados e os algoritmos no sentido de reduzir o working set de páginas; dada a importância dos acessos à memória para o desempenho e a frequência de falhas de TLB, alguns programas com grandes working sets foram recriados com esse objetivo.

Verifique você mesmo

Associe o elemento da hierarquia de memória à esquerda com a frase correspondente à direita.

1. Cache L1	a. Uma cache para uma cache.
2. Cache L2	b. Uma cache para discos.
3. Memória principal	c. Uma cache para uma memória principal.
4. TLB	d. Uma cache para entradas de tabela de páginas.

5.5 Uma estrutura comum para hierarquias de memória

Agora você reconhece que os diferentes tipos de hierarquias de memória compartilham muita coisa em comum. Embora muitos aspectos das hierarquias de memória difiram quantitativamente, muitas das políticas e recursos que determinam como uma hierarquia funciona são semelhantes em qualidade. A Figura 5.29 mostra como algumas características quantitativas das hierarquias de memória podem diferir. No restante desta seção, discutiremos os aspectos operacionais comuns das hierarquias de memória e como determinar seu comportamento. Examinaremos essas políticas como uma série de questões que se aplicam entre quaisquer dos níveis de uma hierarquia de memória, embora usemos principalmente terminologia de caches por motivo de simplicidade.

Recurso	Valores típicos para caches L1	Valores típicos para caches L2	Valores típicos para memória paginada	Valores típicos para uma TLB
Tamanho total em blocos	250–2000	15.000–50.000	16.000–250.000	40–1024
Tamanho total em kilobytes	16–64	500–4000	1.000.000–1.000.000.000	0,25–16
Tamanho do bloco em bytes	16–64	64–128	4000–64.000	4–32
Penalidade da falha em clocks	10–25	100–1000	10.000.000–100.000.000	10–1000
Taxas de falha (global para L2)	2%–5%	0,1%–2%	0,00001%–0,0001%	0,01%–2%

FIGURA 5.29 Os principais parâmetros quantitativos do projeto que caracterizam os principais elementos da hierarquia de memória em um computador. Estes são valores típicos para esses níveis em 2008. Embora o intervalo de valores seja grande, isso ocorre parcialmente porque muitos dos valores que mudaram com o tempo estão relacionados; por exemplo, à medida que as caches se tornam maiores para contornar maiores penalidades de falha, os tamanhos de bloco também crescem.

Questão 1: onde um bloco pode ser colocado?

Vimos que o posicionamento de bloco no nível superior da hierarquia pode utilizar diversos esquemas, do diretamente mapeado ao associativo por conjunto e ao totalmente associativo. Como já dissemos, toda essa faixa de esquemas pode ser imaginada como variações em um

David A. Patterson
John L. Hennessy

ORGANIZAÇÃO E PROJETO DE COMPUTADORES

A INTERFACE HARDWARE/SOFTWARE

Tradução da 4ª Edição

ELSEVIER
CAMPUS