Arquitetura de Computadores I

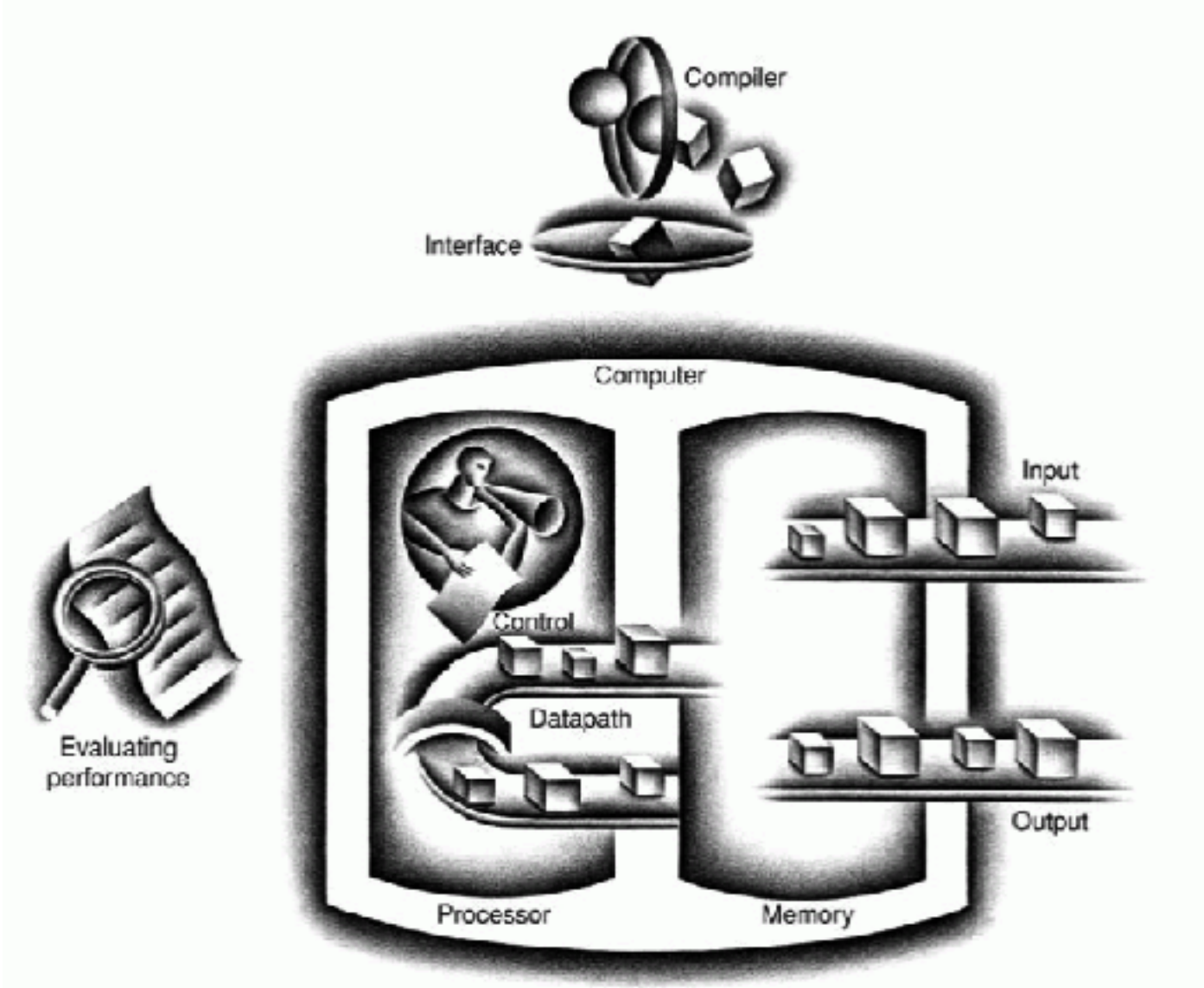
Cap. 07 – Hierarquia de Memória
(Aula 01 e 02)

Prof. M.Sc. Bruno R. Silva

Plano de aula

- Introdução
- Princípios básicos de cache
- Mapeamento direto
- Blocos

A Grande Figura!



Os programadores desejam quantidades de memória ilimitadas e rápidas!
Como “atender” a este requisito?

Com uma ilusão!

Princípio da Localidade

O princípio da localidade diz que os programas acessam uma parte relativamente pequena do seu espaço de endereçamento em qualquer instante do tempo.

- **Localidade Temporal**: Se um item é referenciado, ele tenderá a ser referenciado novamente em breve
  - Exemplos: Loops acessão instruções e dados de modo representativo mostrando alta localidade temporal

- **Localidade Espacial**: Se um item é referenciado, os itens cujos endereços estão próximos tenderão a ser referenciados em breve
  - Exemplo: Em geral instruções são acessadas sequencialmente
    Acessos a elementos de um array ou um registro

Hierarquia de Memória

- Tiramos vantagem do princípio da localidade, implementando a memória de um computador como uma hierarquia de memória.
  - Consiste em múltiplos níveis de memória de diferentes tamanhos e velocidades

- As memórias mais rápidas são mais caras por bit do que as memórias mais lentas e, portanto, são menores.

- Hoje existem 4 tecnologias de construção das hierarquias de memória.
  - Memória principal é implementada em DRAM (Dinamic Random Access Memory)
  - Caches usam SRAM (Static Random Access Memory)
  - Armazenamento secundário utilizando Discos Magnéticos ou Memória Flash.

Hierarquia de Memória

- DRAMs é mais barata por bit do que as SRAMs, embora seja substancialmente mais lenta.
  - DRAM usa significantemente menos área por bit de memória, portanto têm maior capacidade para a mesma área de silício.

Memory technology	Typical access time	$ per GB in 2004
SRAM	0.5–5 ns	$4000–$10,000
DRAM	50–70 ns	$100–$200
Magnetic disk	5,000,000–20,000,000 ns	$0.50–$2

Hierarquia de Memória

- A memória mais rápida está próxima ao processador e a memória mais lenta e barata está abaixo dele.
  - O objetivo é oferecer ao usuário o máximo de memória disponível na tecnologia mais barata, enquanto se fornece acesso na velocidade oferecida pela memória mais rápida.

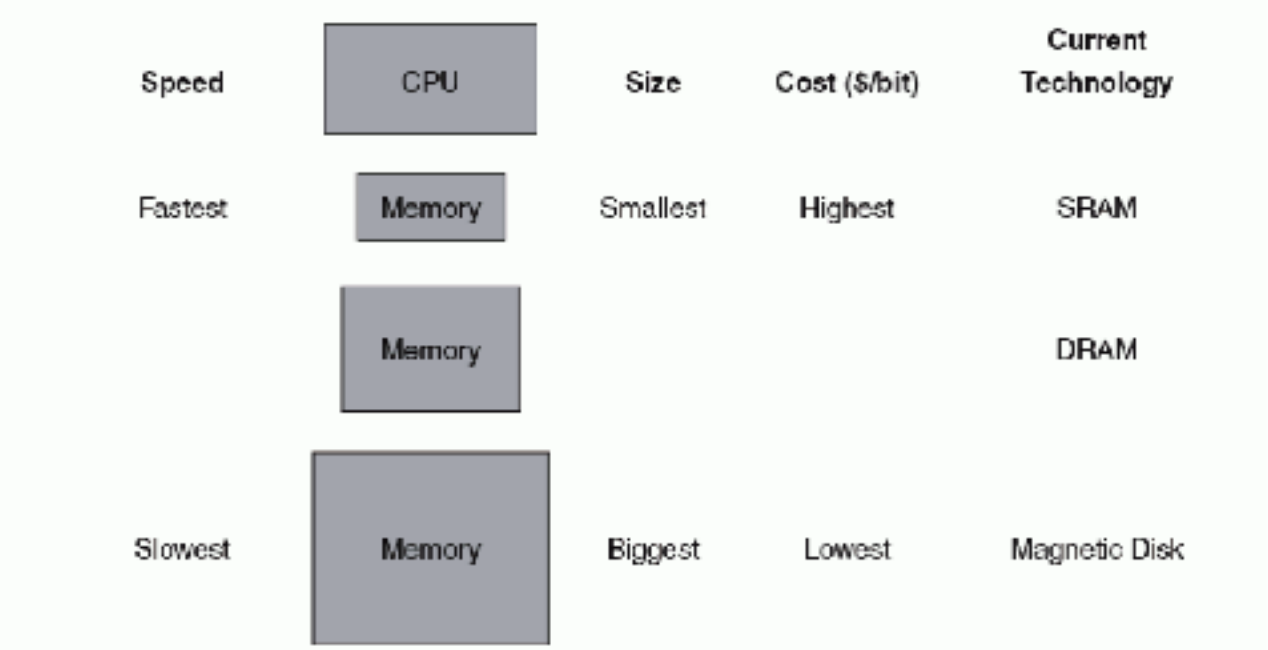


FIGURE 7.1 The basic structure of a memory hierarchy. By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory.

Hierarquia de Memória

- Um nível mais próximo ao processador em geral é um subconjunto de qualquer nível mais distante
- Todos os dados são armazenados no nível mais baixo.
- Conforme nos afastamos do processador, os níveis levam cada vez mais tempo para serem acessados.
- Uma hierarquia de memória pode consistir de múltiplos níveis, Mas os dados são copiados apenas entre dois níveis adjacentes ao mesmo tempo

Hierarquia de Memória

- A unidade de informação mínima que pode estar presente ou ausente na hierarquia de dois níveis é denominada um bloco ou uma linha

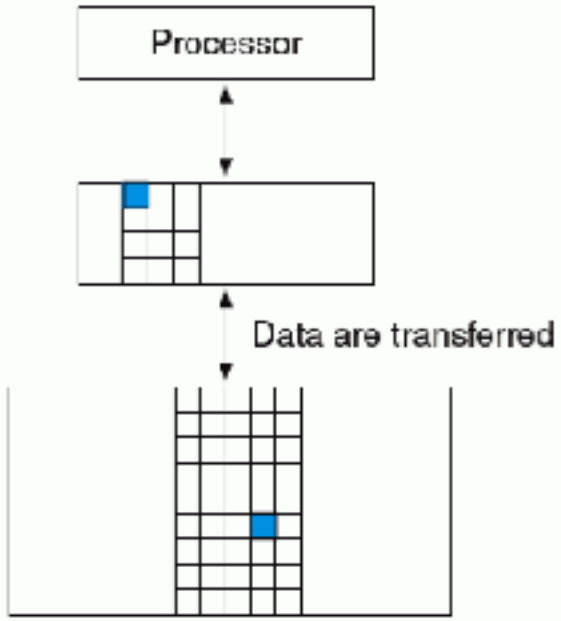


FIGURE 7.2 Every pair of levels in the memory hierarchy can be thought of as having an upper and lower level. Within each level, the unit of information that is present or not is called a *block*. Usually we transfer an entire block when we copy something between levels.

Hierarquia de Memória

- Se os dados requisitados pelo processador aparecerem em um bloco no nível superior, isso é chamado uma **ACERTO**

- Se os dados não forem encontrados no nível superior, a requisição é chamado uma **FALHA**
  - O nível inferior em uma hierarquia é, então, acessado para recuperar o bloco com os dados requisitados.

- A **taxa de acertos** é uma medida de desempenho da hierarquia de memória.

- A **taxa de falhas** (`1 – taxa de acerto`) é a proporção dos acessos à memória não encontrados no nível superior.

Hierarquia de Memória

- O **tempo de acerto** é o tempo para acessar o nível superior da hierarquia de memória, que inclui o tempo necessário para determinar se o acesso é um acerto ou uma falha.

- A **penalidade de falha** é o tempo para substituir um bloco no nível superior pelo bloco correspondente do nível inferior, somado ao tempo para transferir esse bloco para o processador

- Quais das seguintes afirmações costuma ser verdadeiras?
  - As caches tiram proveito da localidade temporal
  - Em uma leitura, o valor retornado depende de quais blocos estão na cache
  - A maioria do custo da hierarquia de memória está no nível mais alto

Hierarquia de Memória

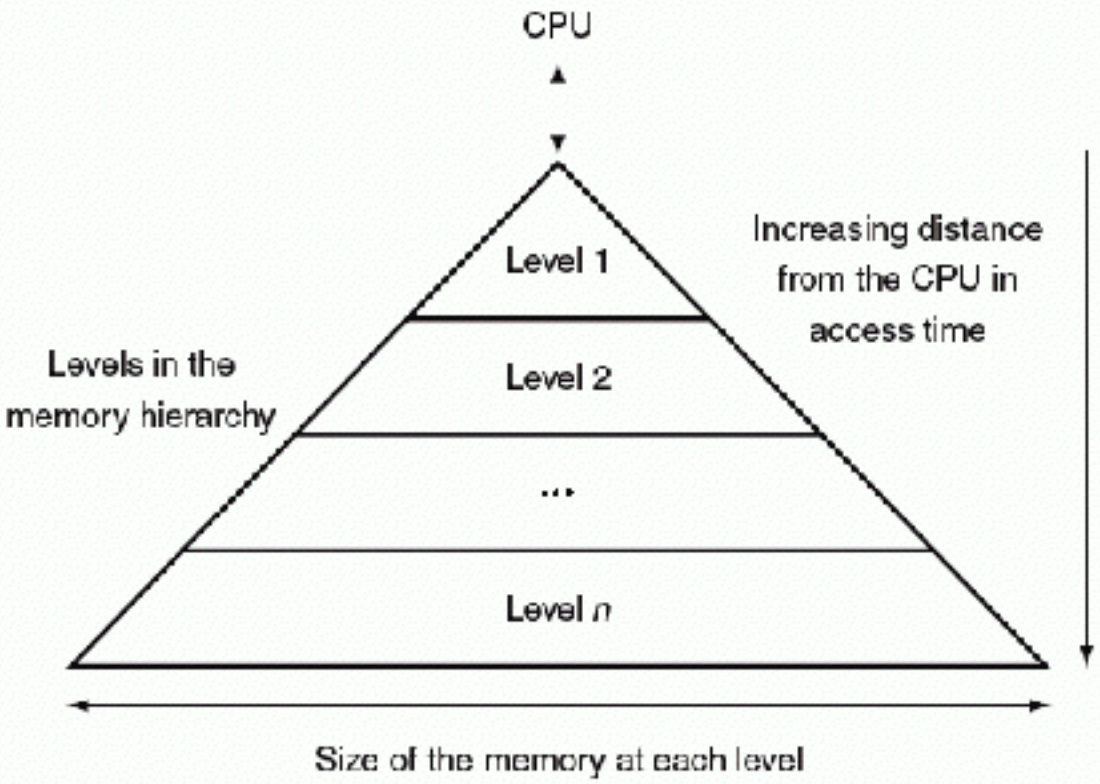


FIGURE 7.3 This diagram shows the structure of a memory hierarchy: as the distance from the processor increases, so does the size. This structure with the appropriate operating mechanisms allows the processor to have an access time that is determined primarily by level 1 of the hierarchy and yet have a memory as large as level *n*. Maintaining this illusion is the subject of this chapter. Although the local disk is normally the bottom of the hierarchy, some systems use tape or a file server over a local area network as the next levels of the hierarchy.

Hierarquia de Memória

- Os programas apresentam localidade temporal e espacial
  - As hierarquias de memória tiram proveito da *localidade temporal* mantendo mais próximos do processador os itens de dados acessados mais recentemente
  - As hierarquias de memória tiram proveito da *localidade espacial* movendo blocos consistindo em múltiplas words contíguas na memória para níveis superiores na hierarquia.

Se a taxa de acerto for bastante alta, a hierarquia de memória terá um tempo de acesso efetivo próximo ao tempo de acesso do nível mais alto (e mais rápido) e um tamanho igual ao do nível mais baixo (e maior).

- Na maioria dos sistemas, a memória é uma hierarquia verdadeira
  - Dados não podem estar presentes no nível `i` a menos que também estejam no nível `i+1`

Princípio Básico de Cache

- Cache foi o nome escolhido para representar o nível da hierarquia de memória entre o processador e a memória principal
  - Apareceram inicialmente nos computadores de pesquisa no início da década de 1960

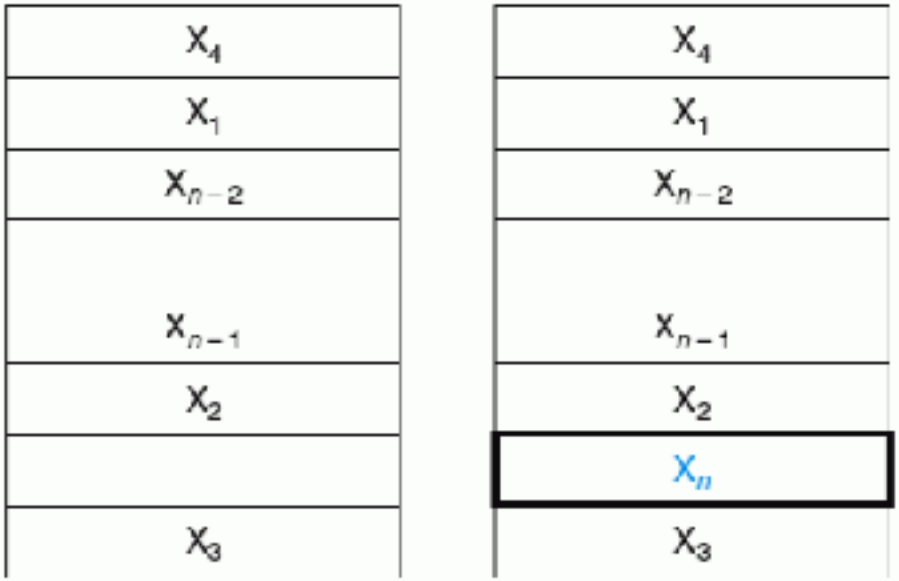


FIGURE 7.4 The cache just before and just after a reference to a word X_n that is not initially in the cache. This reference causes a miss that forces the cache to fetch X_n from memory and insert it into the cache.

Princípio Básico de Cache

Como saber se o item de dado está na cache?

Se estiver, como encontrá-lo?

Primeira alternativa - Mapeamento direto

```
(Endereço de bloco) módulo (Número de blocos de cache na cache)
```

Se o número de entradas na cache for uma potência de dois, então, o módulo pode ser calculado simplesmente usando os $\log_2$ bits menos significativos sobre o tamanho da cache em blocos

Princípio Básico de Cache

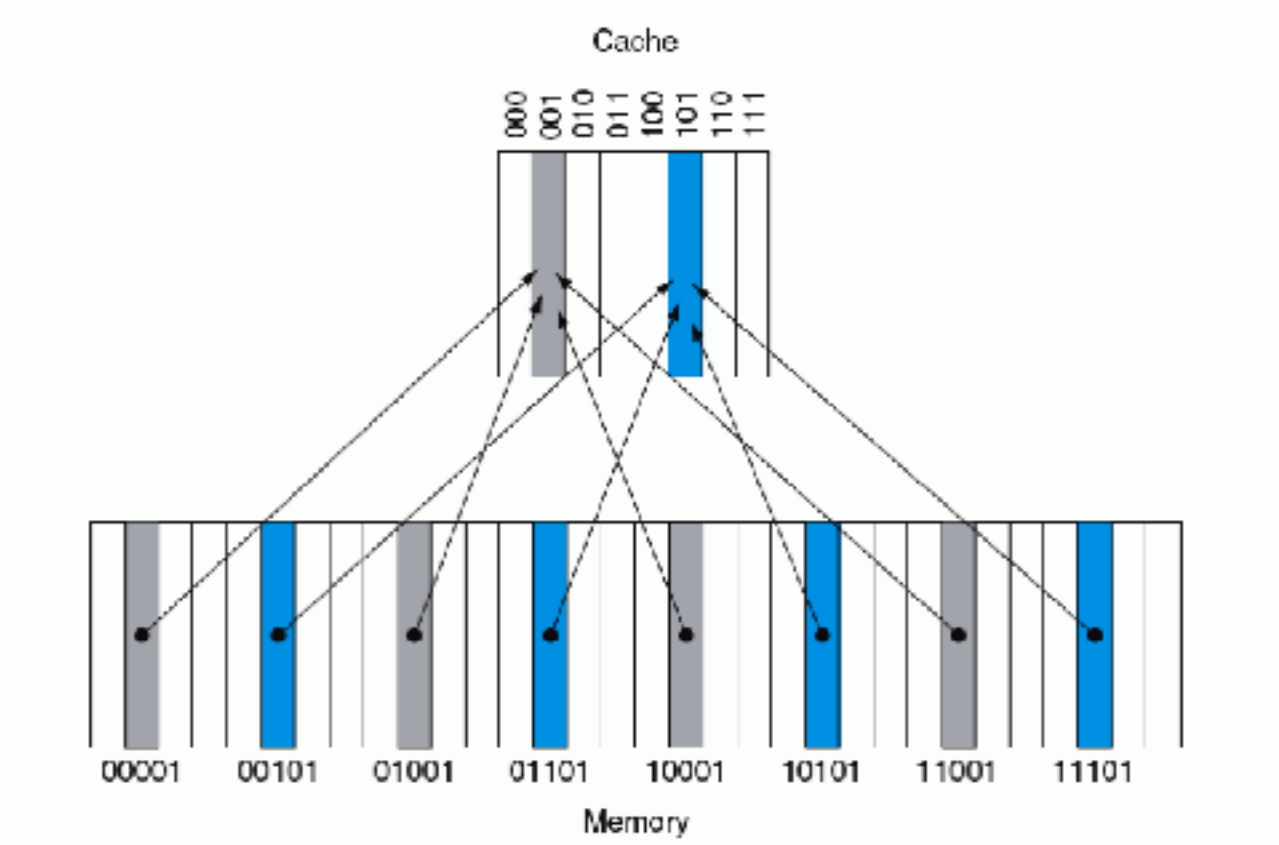


FIGURE 7.5 A direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations. Because there are eight words in the cache, an address X maps to the cache word X modulo 8. That is, the low-order $\log_2(8) = 3$ bits are used as the cache index. Thus, addresses 00001_{two}, 01001_{two}, 10001_{two}, and 11001_{two} all map to entry 001_{two} of the cache, while addresses 00101_{two}, 01101_{two}, 10101_{two}, and 11101_{two} all map to entry 101_{two} of the cache.

Princípio Básico de Cache

Como cada local da cache pode armazenar o conteúdo de diversos locais da memória diferentes, como podemos saber se os dados na cache correspondem àquela word requistada?

- É incluído um conjunto de tags na cache. Elas contém informações de endereço para identificar se a word na cache corresponde à word requisitada.
- A tag precisa apenas conter a parte superior do endereço, correspondente aos bits que não são usados como índice para a a cache.

- Quando o processador é iniciado, a cache não tem dados válidos e os campos de tag não terão significado e mesmo após a execução de muitas instruções algumas entradas de cache poderão estar “vazias”. Como identificar estes casos?
  - Inclusão do **bit de validade.** Se o bit não estiver ligado, não poderá haver uma correspondência com este bloco.

Acessando uma cache

- Uma cache de mapeamento direto com oito words. Como há oito blocos na cache (neste exemplo, cada bloco possui apenas 1 word), os 3 bits menos significativos de um endereço de memória, fornecem o número do bloco
  - O processador requisita os seguintes endereços: **10110** (falha), **11010** (falha), **10110** (acerto), **11010** (acerto), **10000**(falha), **00011** (falha), **10000** (acerto) e **10010**(falha)

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Acessando uma cache

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

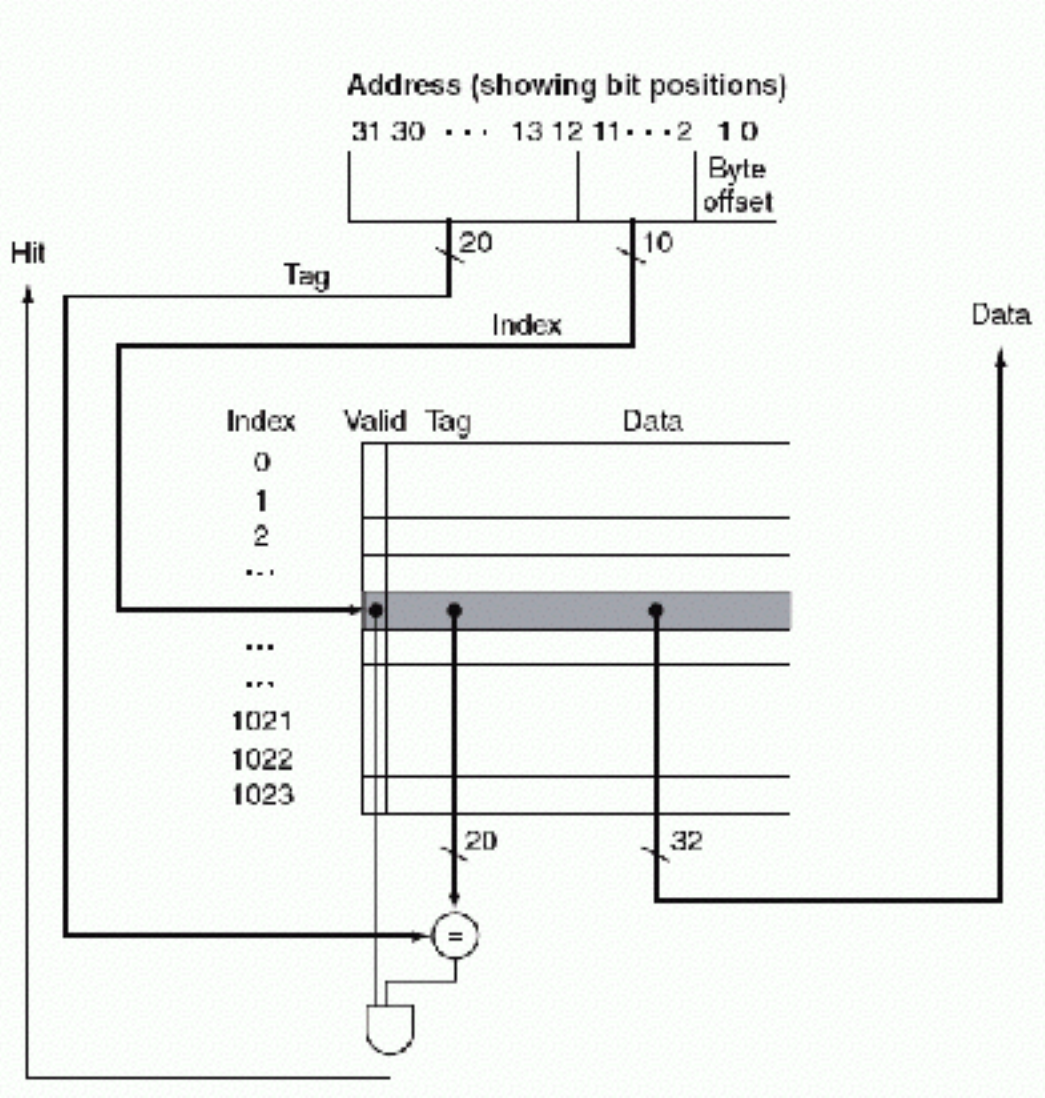
Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

Acessando uma cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

Acessando uma cache



Tamanho de uma cache

- O Número de bits necessários para uma cache é uma função do tamanho da cache e do tamanho do endereço, pois a cache inclui o armazenamento para os dados e as tags.
- O tamanho do bloco mencionado anteriormente era de uma word, mas normalmente é de várias words.
- Considerando o endereço em bytes de 32 bits, uma cache diretamente mapeada de 2^n blocos de tamanho com 2^m words (2^{m+2} bytes) por bloco.
  - Ela exigirá um campo tag cujo tamanho é 32 – (n+m+2) bits, pois n bits são usados para o índice, m bits são usados para a word dentro do bloco e 2 bits são usados para a parte do byte do endereço.
  - O número total de bits de uma cache diretamente mapeada é

    2^n x (tamanho do bloco + tamanho da tag + tamanho do campo de validade).

  - Como o tamanho do bloco é 2^m words (2^{m+5} bits) e o tamanho do endereço é 32 bits, o número de bits nessa cache é

    2^n x (2^m x 32 +(32 − n − m - 2)+ 1) = 2^n x (2^m x 32 + 31 − n - m).

Tamanho de uma cache

- Ex. Quantos bits no total são necessários para uma cache diretamente mapeada com 16KB de dados e blocos de 4 words, considerando um endereçamento de 32 bits?
  - 16KB = 4K words ou 2^{12} words
  - Com um tamanho de bloco de 4 words (2^2), são 2^{10} blocos
  - Cada bloco possui 4 x 32 ou 128 bits de dados mais uma tag que é de 32 – 10 - 2 - 2 bits, mais um bit de validade.
  - Portanto o tamanho da cache em bits é
    - 2^{10} x (128+(32-10-2-2)+1) = 2^{10} x 147 = 147 kbits
    - ou 18,4 KB para uma cache de 16KB

Mapeando um endereço para um bloco de cache multiword

We saw the formula on page 474. The block is given by

$$\text{(Block address) modulo (Number of cache blocks)}$$

where the address of the block is

$$\frac{\text{Byte address}}{\text{Bytes per block}}$$

Notice that this block address is the block containing all addresses between

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block}$$

and

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block} + (\text{Bytes per block} - 1)$$

Mapeando um endereço para um bloco de cache multiword

- Ex. Considere uma cache com 64 blocos e um tamanho de bloco de 16 bytes (4 words). Para qual número de bloco o endereço em bytes 1200 é mapeado?

- O endereço em bytes 1200 é o endereço de bloco 1200/16 = 75

- O bloco 75 é mapeado para o número de bloco de cache (75 mod 64) = 11.

- Este bloco 11 mapeará todos os endereços entre 1200 e 1215

Explorando a localidade espacial

- Blocos maiores exploram a localidade espacial para diminuir a taxa de falhas. Aumentando o tamanho do bloco normalmente diminui a taxa de falhas.

- Mas existe um limite, pois aumentando o tamanho do bloco para uma fração significativa do tamanho da cache, poderá fazer a taxa de falhas aumentar.
  - Devido a compentição entre os blocos.
    - Um bloco será retirado da cache antes que muitas de suas words tenham sido acessadas
  - Blocos muito grande aumenta o custo da penalidade de falha.
    - Sem alterações caras no sistema de memória, leva mais tempo trazer blocos grandes faltantes do nível inferior do que blocos pequenos faltantes

Explorando a localidade espacial

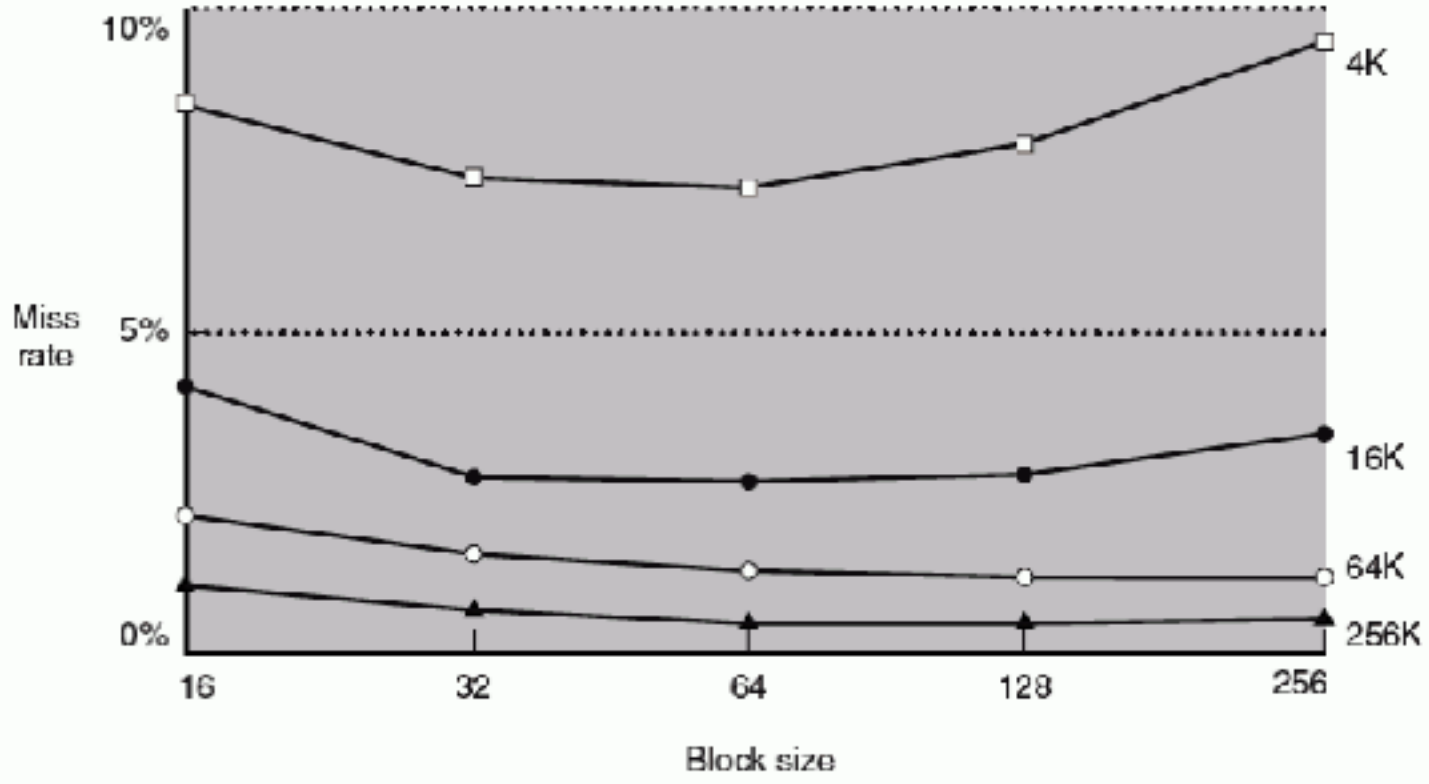


FIGURE 7.8 Miss rate versus block size. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. (This figure is independent of associativity, discussed soon.) Unfortunately, SPEC2000 traces would take too long if block size were included, so these data are based on SPEC92.

Tratando falhas na cache

- Como a unidade de controle lida com uma falha?
  - Ela precisa detectá-la
  - Ela precisa processá-la, buscando os dados requisitados da memória

- E se a cache reportar uma acerto?
  - O computador continua usando os dados como se nada tivesse acontecido

Portanto podemos utilizar o mesmo controle desenvolvido no capítulo 5, onde as memórias são simplesmente substituídas por caches.

Tratando falhas na cache

Modificar o controle para tratar acerto é fácil

As falhas no entanto, exigem trabalho maior

- O tratamento de falha da cache é feito com a unidade de controle do processador e com um controlador separado que inicia o acesso à memória e preenche novamente a cache.

- O processamento de uma falha da cache cria um **stall** (parada)
  - Basicamente congelando o conteúdo dos registradores temporários e visíveis ao programador, enquanto esperamos a memória.

Tratando falhas na cache

Como falhas de instrução podem ser tratadas pelo caminho de dados

- Se um acesso à instrução resultar em falha, o conteúdo do IR será inválido.
- O endereço da instrução que gerou a falha na cache de instruções é igual ao valor do PC – 4.
- Um vez, tendo o endereço, instruímos a memória principal a realizar uma leitura
- Esperamos a memória responder (já que o acesso levará vários ciclos)
- Escrevemos as words na cache

Tratando falhas na cache

- Enviar o valor do PC original (PC atual – 4) para a memória
- Instruir a memória principal a realizar uma leitura e esperar que a memória complete seu acesso
- Escrever na entrada da cache
  - Colocando os dados da memória na parte dos dados da entrada selecionada pelos bits menos significativos do endereço do bloco
  - Escrever os bits mais significativos do endereço (vindo da ALU) no campo TAG
  - Ligar o bit de validade
- Reiniciar a execução da instrução
  - O que buscará novamente a instrução, desta vez encontrando-a na cache

Exercícios p/ correção na próxima aula!

7.2, 7.3, 7.4, 7.9, 7.10, 7.11, 7.12

Tratando Escritas

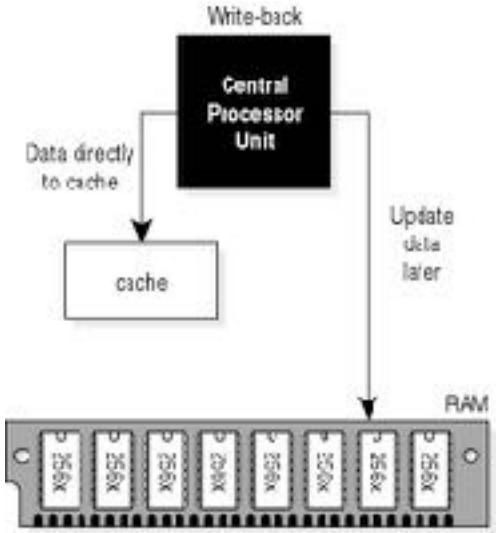
- Suponha que uma instrução store escreva dados apenas na cache de dados (sem alterar a memória principal)
  - Neste caso, dizemos que a cache e a memória estão inconsistentes
- A maneira mais simples de manter a consistencia é sempre escrever os dados na memória e na cache
  - Este esquema é chamado write-through
- E quando correr uma falha de escrita?
  - Primeiro buscamos as words do bloco da memória. Após se buscado e colocado na cache, podemos substituir a word que causou a falha no bloco da cache e também escrevemos a word na memória principal usando o endereço completo.

Tratando Escritas

- Embora o projeto anterior trate das escritas de maneira muito simples, ele não oferece um desempenho muito bom.
  - Com um esquema de **write-through**, toda escrita faz com que os dados sejam escritos na memória principal.
  - Ex. Se 10% das instruções são stores. Se o CPI sem falhas de cache fosse 1. Gastar 100 ciclos extras em cada escrita levaria um CPI de 1 + 100 * 10% = 11, reduzindo o desempenho em mais de 10%

- Uma solução é usar um buffer de escrita (write buffer)
  - Ele armazena os dados enquanto estão esperando para serem escritos na memória.
  - Entretanto se o buffer de escrita estiver cheio quando o processador atingir uma escrita, ele precisará sofrer um stall até que haja uma posição vazia no buffer
  - Naturalmente, se a velocidade em que a memória pode completar escritas for menor do que a velocidade em que o processador está gerando escritas, nenhuma quantidade de buffer pode ajudar.

Tratando Escritas



- A alternativa para um esquema write-through é um esquema chamado write-back.
  - Quando ocorre uma escrita, o novo valor é escrito apenas no bloco da cache.
  - O bloco modificado é escrito no nível inferior da hierarquia quando elé é substituído.
  - São mais eficientes, especialmente quando os processadores geram escritas mais rápido do que elas podem ser tratadas pela memória principal
  - Entretanto é mais complexo de implementar que um esquema write-through

Tratando escritas

- Polícita nas falhas de escrita
  - Buscar na falha (buscar na escrita) em caches write-through
    - Aloca um bloco de cache para o endereço que falhou e busca o restante do bloco para a cache antes de escrever os dados.
    - Uma alternativa seria alocar o bloco na cache, mas não buscar os dados (não buscar na escrita) ou mesmo não alocar o bloco (não alocar na escrita). Também chamadas de write-around.

Tratando escritas

- Implementação eficiente das escritas em caches write-back
  - Implementar stores eficientes em uma cache write-back é mais complexo do que em uma cache write-through.
    - Precisamos escrever o bloco na memória se os dados na cache estiverem modificados e tivermos uma falha de cache.
    - Stores ou exigem 2 ciclos (um ciclo para verificaro a certo e outro para efetivamente escrever na cache) ou exige um buffer chamado buffer de store
  - Em caches write-through, as escritas sempre podem ser feitas em um ciclo
  - Muitas caches write-back incluem buffers de escrita usados para reduzir a penalidade de falha quando um bloco precisa ser substituído.

Exemplo

- Processador Intrinsity FastMATH
  - Um processador que usa a arquitetura MIPS e uma implementação de cache simples
  - Cache de instruções e dados separada de 16KB ou 4K words com blocos de 16 words.

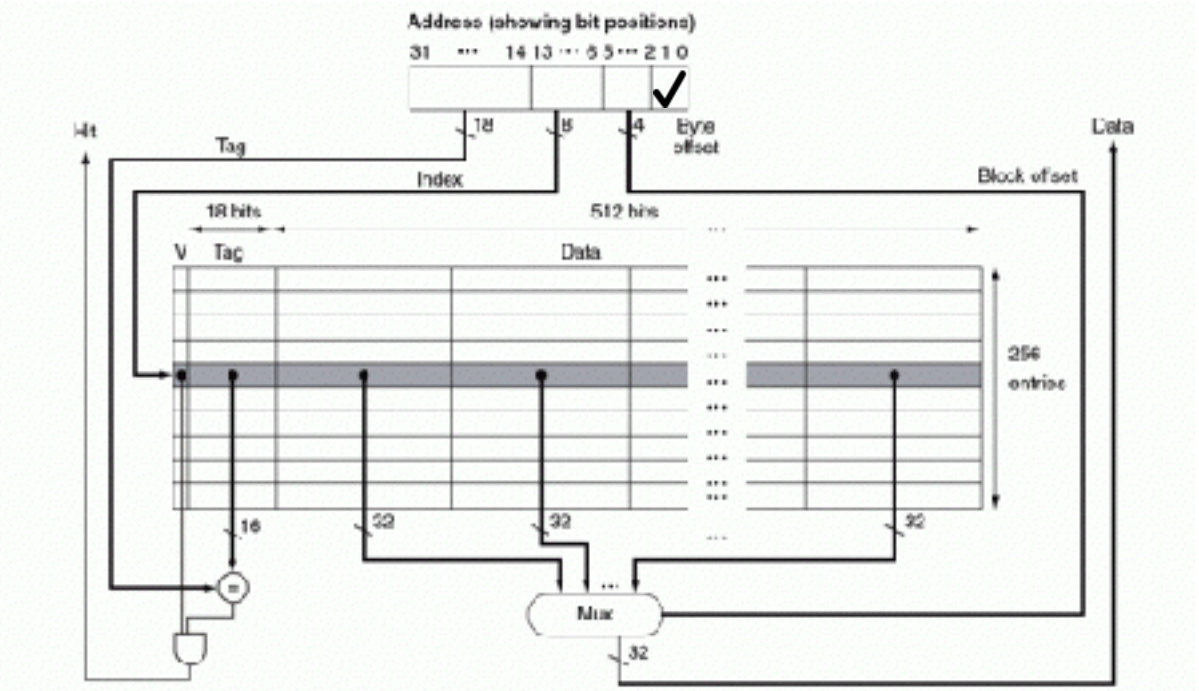


FIGURE 7.9 The 16 KB caches in the Intrinsity FastMATH each contain 256 blocks with 16 words per block. The tag field is 18 bits wide and the index field is 8 bits wide, while a 4-bit field (bits 5–2) is used to index the block and select the word from the block using a 16-to-1 multiplexor. In practice, to eliminate the multiplexor, caches use a separate large RAM for the data and a smaller RAM for the tags, with the block offset supplying the extra address bits for the large data RAM. In this case, the large RAM is 32 bits wide and must have 16 times as many words as blocks in the cache.

Exemplo

- As etapas para uma requisição de leitura para qualquer uma das caches são as seguintes:
  - Enviar o endereço para a cache apropriada, o endereço vem do PC (para uma instrução) ou da ALU (para dados).
  - Se a cache sinalizar acerto, a word requisitada estará disponível na linha de dados.
  - Um campo índice de bloco é usado para controlar o multiplexador que seleciona a word requisitada dentro das 16 words possíveis.

Exemplo

- Taxa de falhas de instruções e dados para o processador Intrinsity FastMATH utilizando o benchmark SPEC2000.

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

FIGURE 7.10 Approximate instruction and data miss rates for the Intrinsity FastMATH processor for SPEC2000 benchmarks. The combined miss rate is the effective miss rate seen for the combination of the 16 KB instruction cache and 16 KB data cache. It is obtained by weighting the instruction and data individual miss rates by the frequency of instruction and data references.

- Uma cache combinada com um tamanho total igual à soma das duas caches divididas normalmente terá uma taxa de acertos melhor.
  - Caches combinadas não dividem rigidamente o número de entradas que podem ser usadas por instruções daquelas que podem ser usadas para dados
  - Entretanto, a vantagem de suportar acesso à instruções e dados simultaneamente na cache dividida, logo suplanta a desvantagem da taxa de falhas maior.

Projetando o sistema de memória para suportar caches

- Embora seja difícil reduzir a latência para buscar a primeira word da memória, podemos reduzir a penalidade de falha se aumentarmos a largura de banda da memória para a cache.
  - Isso permite que blocos maiores sejam usados enquanto mantemos uma baixa penalidade de falhas

- O processador é normalmente conectado à memória por meio de um barramento.
  - A velocidade de clock do barramento geralmente é muito mais lenta do que a do processador.
    - Isso afeta a penalidade de falha

- Exemplo
  - 1 ciclo de clock de barramento de memória para enviar o endereço
  - 15 ciclos de clock de barramento de memória para cada acesso a DRAM iniciado
  - 1 ciclo de clock de barramento de memória para enviar uma word de dados
  - Se tivermos um bloco de cache de 4 words e um banco de DRAMs com a largura de uma word, a penalidade de falha seria:
    - 1 + 4*15 + 4*1 = 65 ciclos
  - O número de bytes transferidos por ciclo de clock de barramento para uma única falha seria de 4*4/65 = 0.25

Projetando o sistema de memória para suportar caches

- Três opções para projetar o sistema de memória:
  1. Memória possui uma word de largura e todos os acessos são feitos sequencialmente.

  2. Aumentar a largura de banda para a memória alargando a memória e os barramentos entre o processador e a memória

  3. Aumentar a largura de banda alargando a memória mas não o barramento de interconexão.
     - Pagamos o custo para transmitir 1 word, mas podemos evitar pagar o custo da latência de acesso mais de uma vez.

Projetando o sistema de memória para suportar caches

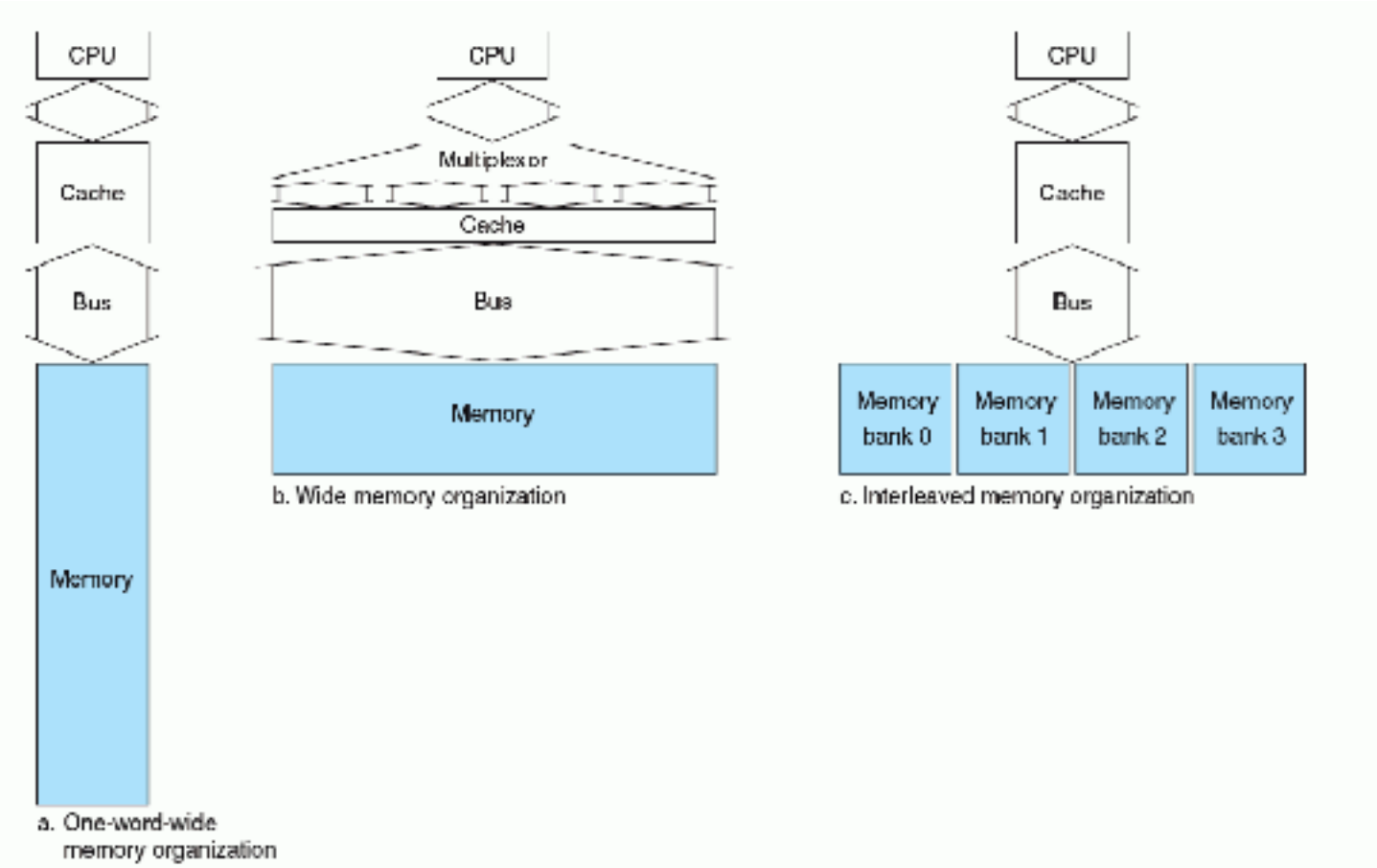


FIGURE 7.11 The primary method of achieving higher memory bandwidth is to increase the physical or logical width of the memory system. In this figure, memory bandwidth is improved two ways. The simplest design, (a), uses a memory where all components are one word wide; (b) shows a wider memory, bus, and cache; while (c) shows a narrow bus and cache with an interleaved memory. In (b), the logic between the cache and processor consists of a multiplexor used on reads and control logic to update the appropriate words of the cache on writes.

Projetando o sistema de memória para suportar caches

- No segundo caso
  - Com a largura de memória principal de duas words, a penalidade de falha cai de 65 ciclos de clock de barramento de memória para 1 + 2*15+2*1 = 33 ciclos de clock de barramento de memória.
  - A largura de banda para uma única falha é, então, 0.48 bytes por ciclo de clock de barramento e 0.94 bytes por ciclo de clock de barramento quando a memória tem quatro words de largura.
  - Os maiores custos desta melhoria são o barramento mais largo e o possível aumento no tempo de acesso à cache devido ao multiplexador e à lógica de controle entre o processador e a cache.

Projetando o sistema de memória para suportar caches

- No terceiro caso:
  - Em vez de tornar todo o caminho entre a memória e a cache mais largo, os chips de memória podem ser organizados em bancos para ler ou escrever múltiplas words em um único tempo de acesso.
  - Cada banco poderia ter 1 word de largura, mas enviar um endereço para vários bancos permite que todos eles leiam simultaneamente.
  - Este esquema é chamado de *Intercalação*
  - Exemplo:
    - Com quatro bancos, o tempo para obter um bloco de quatro words consistiria em 1 ciclo para transmitir o endereço e a requisição de leitura para os bancos, 15 ciclos para que todos os quatro bancos acessem a memória e 4 ciclos para enviar as quatro words de volta para a cache
    - Isso produz uma penalidade de falha de 1 + 1*15 + 4 + 1 = 20 ciclos de clock de barramento de memória.
    - Essa é uma largura de banda efetiva por falha de 0.80 bytes por clock, ou cerca de 3 vezes a largura de banda para a memória e barramento de uma word de largura.
  - Cada banco pode escrever independentemente , quadruplicando a largura de banda de escrita e gerando menos stalls em uma cache write-through.

Projetando o sistema de memória para suportar caches

- Os chips de memória são organizados para produzir vários bitds de saída, normalmente de 4 a 32 bits.

- A organização da RAM pode ser descrita como d x w, onde d é o número de locais endereçáveis (a profundidade) e w é a saída (ou a largura de cada local)

Year introduced	Chip size	$ per MB	Total access time to a new row/column	Column access time to existing row
1980	64 Kbit	$1500	250 ns	150 ns
1983	256 Kbit	$500	185 ns	100 ns
1985	1 Mbit	$200	135 ns	40 ns
1989	4 Mbit	$50	110 ns	40 ns
1992	16 Mbit	$15	90 ns	30 ns
1996	64 Mbit	$10	60 ns	12 ns
1998	128 Mbit	$4	60 ns	10 ns
2000	256 Mbit	$1	55 ns	7 ns
2002	512 Mbit	$0.25	50 ns	5 ns
2004	1024 Mbit	$0.10	45 ns	3 ns

FIGURE 7.12 DRAM size increased by multiples of four approximately once every three years until 1996, and thereafter doubling approximately every two years. The improvements in access time have been slower but continuous, and cost almost tracks density improvements, although cost is often affected by other issues, such as availability and demand. The cost per megabyte is not adjusted for inflation.

Verifique você mesmo!

- Quais dos seguintes princípios de projeto de cache normalmente são válidos?
  - Quanto mais curta for a latência de memória, menor será o bloco de cache
  - Quanto mais curta for a latência de memória, maior será o bloco de cache
  - Quanto maior fora a largura de banda de memória, menor será o bloco de cache
  - Quanto maior for a largura de banda da memória, maior será o bloco de cache

Medindo e melhorando o desempenho da cache

- O tempo de CPU pode ser dividido nos ciclos de clock que a CPU gasta executando o programa e os ciclos de clock que gasta esperando o sistema de memória.
- Normalmente, consideramos que os custos dos acessos à cache que são acertos são parte dos ciclos de execução normais da CPU. Portanto,

```
Tempo de CPU = (ciclos de clock de execução da CPU + ciclos
   de clock de stall de memória) * tempo de ciclo de clock

Ciclos de clock de stall de memória = ciclos de stall de
          leitura + ciclos de stall de escrita

Ciclos de stall de leitura = leituras/programa * taxa de
    falhas de leitura * penalidade de falha de leitura
```

Medindo e melhorando o desempenho da cache

- As escritas são mais complicadas:

- Para um esquema write-through, temos duas origens de stalls:
  - As falhas de escrita, que normalmente exigem que busquemos o bloco antes de continuar a escrita
  - Os stalls do buffer de escrita, que ocorrem quando o buffer de escrita está cheio ao ocorrer uma escrita.

```
Ciclos de stall de escrita = (escritas/programa * taxa de
   falhas de escrita * penalidade de falhas de escrita) +
                stalls do buffer de escrita
```

Medindo e melhorando o desempenho da cache

- Stalls do buffer de escrita dependem da sincronização das escritas, e não apenas da frequência.
- Não é possível fornecer uma equação simples para calcular estes stalls
  - Felizmente, nos sistemas com buffer de escrita razoável (quatro ou mais words) e uma memória capaz de aceitar escritas em uma velocidade que excede significativamente a frequência de escrita média em programas (por um fator de duas vezes), os stalls do buffer de escrita serão pequenos e podemos ignorá-los
- Esquemas write-back também possuem stalls potenciais extras surgindo da necessidade de escrever um bloco de cache novamente na memória quando o bloco é substituído.
- Na maioria das organizações de cache write-back, as penalidades de falha de leitura e escrita são iguais (o tempo para buscar o bloco da memória)
- Se considerarmos que os stalls do buffer de escrita são insignificantes, podemos combinar as leituras e escritas usando uma única taxa de falhas e a penalidade falha.

Ciclos de clock de stall de memória = acessos à memória/programa * taxa de falhas * penalidade de falha

Medindo e melhorando o desempenho da cache

Também podemos fatora isso como:

```
Ciclos de clock de stall de memória = Instruções/programa *
                falhas/instrução * penalidade de falha
```

Medindo e melhorando o desempenho da cache

- Exemplo
  - Taxa de falhas de cache de instruções p/ um programa = 2%
  - Taxa de falhas de caceh de dados = 4%
  - Processador possuir CPI de 2 sem qualquer stall de memória e a penalidade de falha é de 100 ciclos para qualquer falha.

Quanto mais rápido um processador executaria com uma cache perfeita que nunca falhasse? (use a frequência de instruções do SPECint2000 do Cap. 3)

P/ instruções temos: ciclos de falha = I * 2% * 100 = 2*I

Temos 36% de load/stores

P/ dados tempos: ciclos de falha = I*36%*4%*100 = 1.44*I

O total de ciclos de stall de memória é de 3.44*I. Isto é mais de 3 ciclos de stall de memória por instrução. Portanto o CPI com stall de memória é de 2 + 3.44 = 5.44.

Como não há mudança alguma na contagem de instruções ou na velocidade de clock, a taxa dos tempos de execução da CPU é

Tempo de CPU com stall / Tempo de CPU com cache perfeita = (I * CPIstall * ciclo de clock) / I * CPIperfeito * ciclo de clock

CPIstall/CPIperfeito = 5.44/2 = 2.72

Medindo e melhorando o desempenho da cache

- O que acontece se o processador for tornando mais rápido, mas o sistema de memória não?

- Suponha que aceleremos o computador do exemplo anterior reduzindo seu CPI de 2 para 1, sem mudar a velocidade de clock. O sistema com falhas de cache então teria um CPI de 1 + 3.44 = 4.44 e o sistema com cache perfeita seria e 4.44/1 = 4.44 vezes mais rápido.

  - A quantidade de tempo de execução gasto em stall de mem´roai sube então de 3.44/5.44 = 63% para 3.44 / 4.44 = 77%

Atividade:

P/ próxima aula:

Ler o Apêndice B, seção B.8 sobre a arquitetura interna das DRAMs
Ler as páginas 374 à 385

Referências

- Hennessy, J. L., Patterson, D. A. *Organização e projeto de computadores: A Interface hardware/software*. 3 ed, Rio de Janeiro, LTC, 2005.