Arquitetura de Alto desempenho

Projeto de hierarquia de memória

António Rui Borges

Resumo

- *Por que o gerenciamento de memória é tão importante*
  - *– Princípio da localidade*
  - *– Hierarquia de memória*
- *Cache*
  - *– Princípios de cache*
  - *– Desempenho do cache*
  - *– Otimização de cache*
- *Memória principal*
  - *– RAMs dinâmicas assíncronas*
  - *– RAMs dinâmicas síncronas •*

Leitura sugerida

Por que o gerenciamento de memória é tão importante - 1

Desde os primórdios da era da computação, os programadores sonham *com* quantidades de memória *rápida* para armazenar e executar seus programas, ou seja, sonham em ter tanta memória quanto julgarem necessária para armazenar códigos e dados que poderão então ser acessados na taxa máxima que o processador puder operar.

Embora a capacidade *da memória principal* tenha aumentado continuamente ao longo dos anos, o facto é que os programas tendem a expandir-se em tamanho e complexidade, preenchendo todo o espaço disponível – *Lei de Parkinson.*

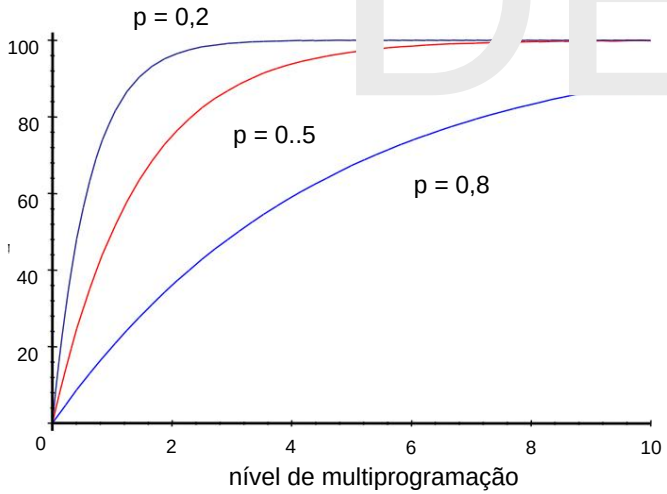
Isto é verdade não só porque os programas se tornaram mais complexos à medida que o desempenho dos sistemas informáticos melhora, lidando com problemas que não poderiam ter sido resolvidos anteriormente, mas também porque é crítico num ambiente multitarefa armazenar juntos na memória principal os espaços de endereçamento de muitos processos para que o processador seja mantido totalmente ocupado e a *resposta* associada e/ou os tempos *de resposta* são minimizados.

Por que o gerenciamento de memória é tão importante - 2

fração de ocupação do processador = 1 – p ^{n} (modelo simplificado)

p - fração do tempo que um processo espera bloqueado pela conclusão das operações de E/S, sincronização ou qualquer outra causa

n - número de processos que coexistem atualmente na memória principal



N. de processos em MP	*% de ocupação (P)*
4	59
8	83
12	93
16	97

p = 0,8

Princípio da localidade - 1

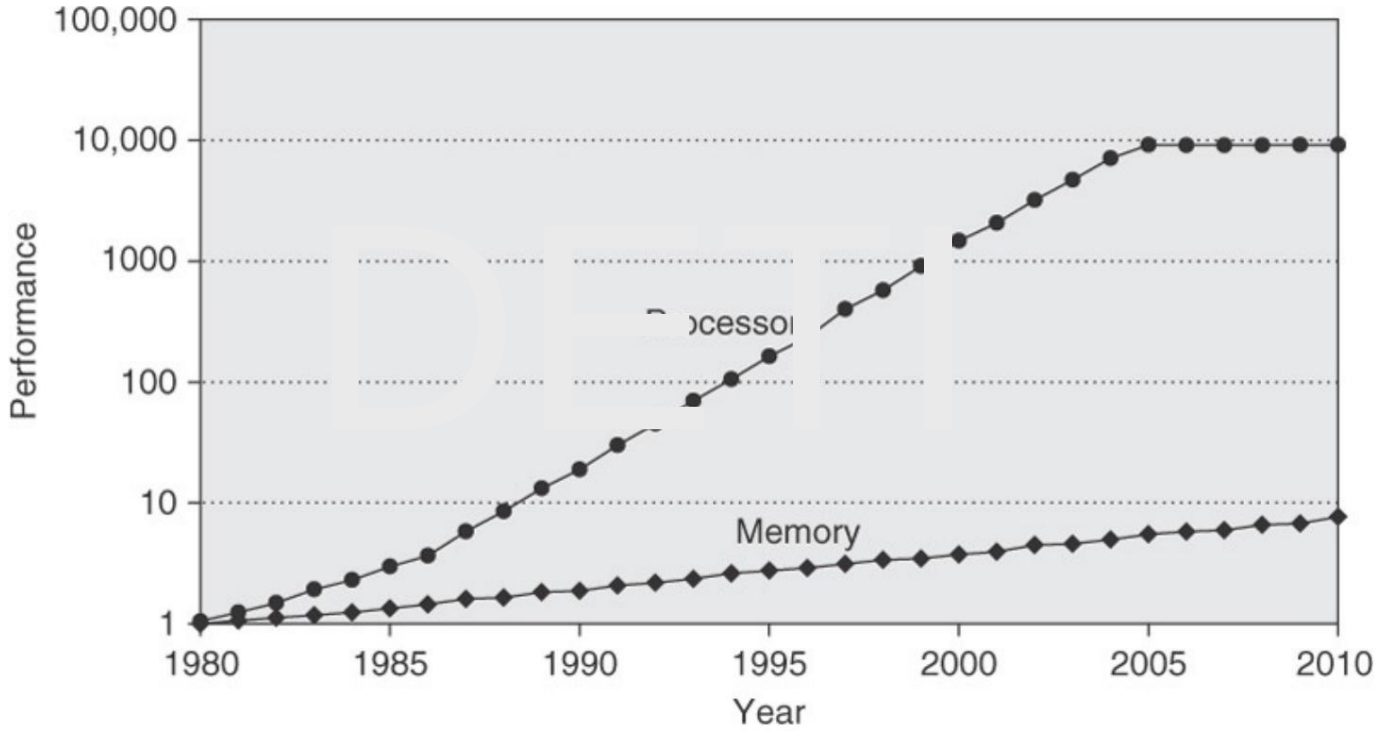
Não é possível, contudo, satisfazer literalmente o sonho dos programadores.

Pode ser demonstrado que, para uma determinada tecnologia de implementação e um determinado orçamento de energia, hardware mais simples pode ser fabricado mais rapidamente. À medida que a complexidade aumenta, os atrasos no tempo de propagação do sinal tornam-se maiores devido ao aumento da quantidade necessária de circuitos de interconexão e à diminuição da intensidade das correntes elétricas de acionamento.

Além disso, o problema tornou-se mais grave com o passar do tempo. Em vez de ser reduzida, a lacuna de desempenho do processador-DRAM aumentou gradualmente nas últimas décadas e até piorou desde a introdução dos processadores multicore, onde o pico de largura de banda agregado é proporcional ao número de processadores no núcleo.

Princípio da localidade - 2

Variação de desempenho ao longo do tempo de processador único versus memória Fonte: Computer Architecture: A Quantitative Approach



curva do processador – número médio de solicitações de memória por segundo

curva de memória – inverso da latência de acesso DRAM

Princípio da localidade - 3

Para lidar com este problema, a abordagem seguida pelos projetistas de sistemas de memória é baseada em um fato observacional derivado do rastreamento da execução do programa e conhecido como *princípio da localidade.* Afirma simplesmente que, durante períodos de tempo relativamente grandes, um programa tende a referenciar uma fração muito definida do seu espaço de endereçamento. Assim, pode-se falar de

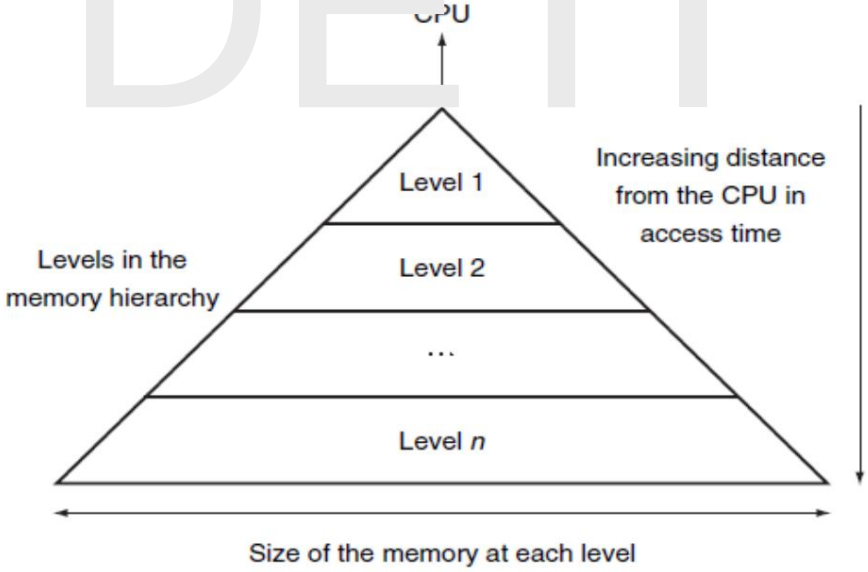
- *localidade espacial* – quando uma palavra de memória é referenciada uma vez, outra, que está armazenada nas proximidades, pode ser referenciada em breve
- *localidade temporal* – quando uma palavra de memória é referenciada uma vez, pode ser referenciado novamente em breve.

Não é muito difícil perceber porque é que os programas têm este tipo de comportamento. Tentar para afirmar as razões pelas quais isso acontece!

Hierarquia de memória - 1

Para aproveitar o princípio da localidade, a memória do computador é implementada como uma *hierarquia de memória,* ou seja, consiste em múltiplos níveis de memória com diferentes tamanhos e velocidades de acesso. Via de regra, os módulos de memória mais rápidos têm um preço por bit mais alto que os mais lentos e têm menor capacidade de armazenamento.

O objetivo é apresentar ao programador a quantidade de memória disponível na tecnologia mais barata (nível mais baixo), ao mesmo tempo que fornece acesso na velocidade oferecida pela tecnologia mais cara (nível mais alto).



Fonte: Organização e Design de Computadores: A Interface Hardware/Software

Hierarquia de memória - 2

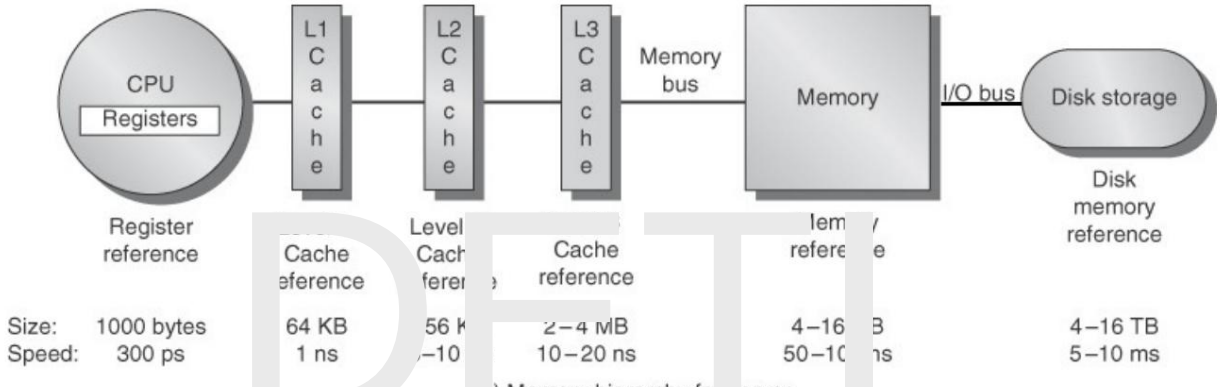
A hierarquia de memória pode ser dividida em níveis distintos

- *banco de registros* – memória interna do processador, o acesso é controlado por As instruções
- *cache* – memória externa ao processador, o acesso é controlado por *instruções de busca* e *transferência de dados ;* atualmente consiste em vários níveis de RAM estática, todos colocados dentro do circuito integrado do processador; o primeiro nível está localizado dentro do chip do processador e é dividido em unidades de instrução e de dados
- *memória principal* – memória externa ao processador; implementa o *conceito de programa armazenado* definindo o dispositivo onde as instruções e dados de um programa em execução são armazenados principalmente; consiste em RAM dinâmica
- *área de swap* – memória não volátil localizada em armazenamento em massa; funciona como uma extensão da memória principal para implementar uma organização de memória controlada pelo sistema operacional, geralmente uma *arquitetura de memória virtual paginada;* consiste principalmente em HDD ou dispositivos de memória flash.

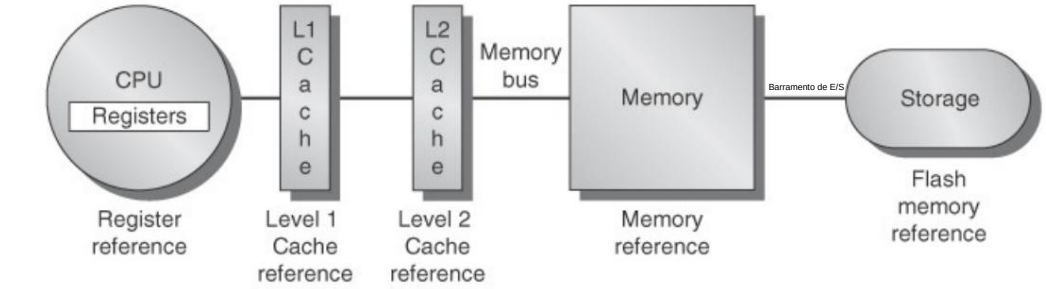
Hierarquia de memória - 3

Níveis de uma hierarquia de memória típica

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



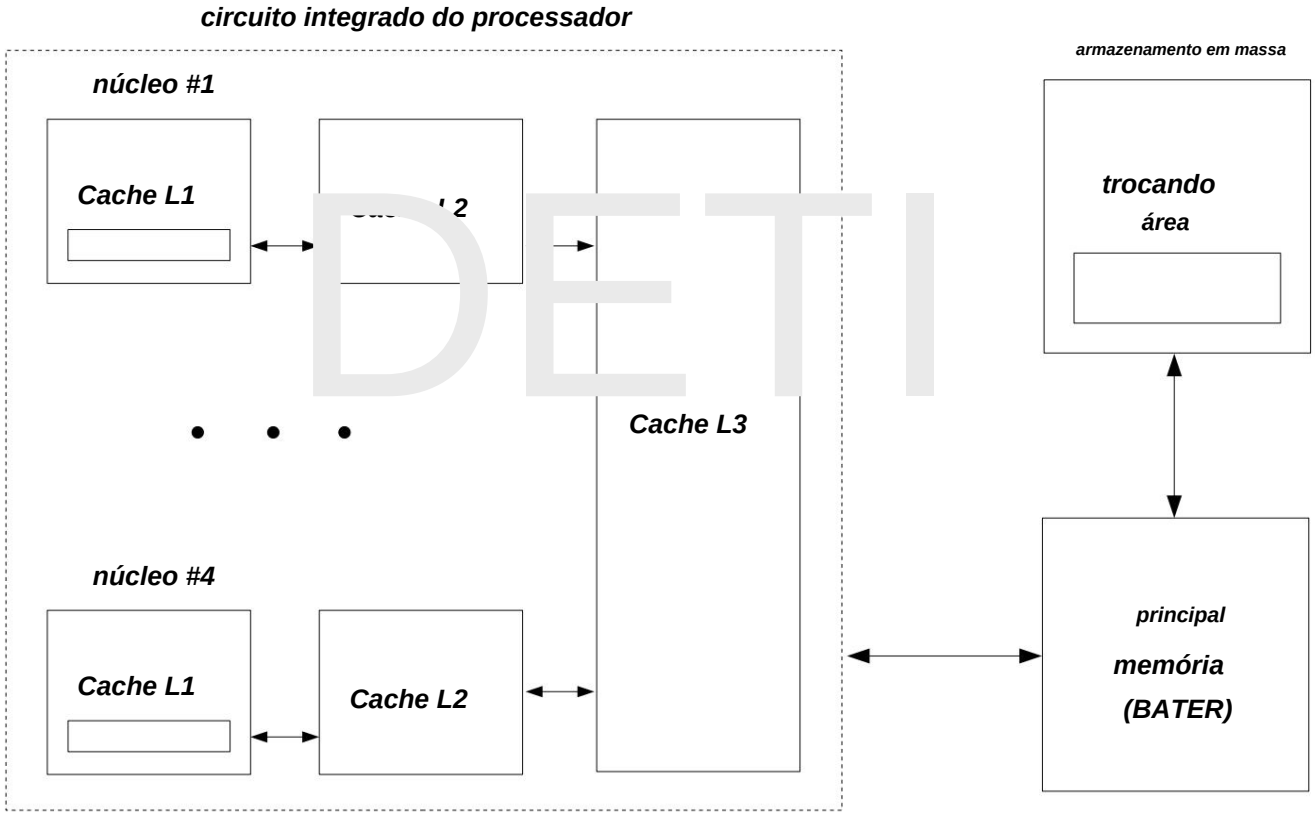
(a) Memory hierarchy for a server



(b) Memory hierarchy for a personal mobile device

Hierarquia de memória - 4

Hierarquia de memória típica para um sistema de processador de 4 núcleos



Hierarquia de memória - 5

Especificações técnicas de uma hierarquia de memória típica

Fonte: adaptado de Computer Architecture: A Quantitative Approach

Nome	*Tamanho típico*	*Tecnologia de implementação*	*Tempo de acesso (ns)*	*Largura de banda (MB/s)*	*Gerenciado por*	*Apoiado por*
registrar banco	menos 1 KB	design personalizado multiporta – CMOS	0,15 – 0,30	105 – 106	compilador	esconderijo
esconderijo	32 KB – 8 MB	CMOS on/off-chip SRAM	0,5 – 15	104 – 4x104	hardware	principal memória
principal memória	menos 512 KB	CMOS DRAM	30 – 200	5x103 – 2x104	sistema operacional	trocando área
trocando área	semicondutor maior de 1 TB	/ magnético	3x104 - 5x106	50 – 500	sistema operacional	armazenamento de longo prazo

Hierarquia de memória - 6

Em uma hierarquia de memória, quanto mais alto for um nível, mais próximo do processador ele estará localizado. As hierarquias de memória aproveitam a *localidade temporal,* mantendo instruções e dados acessados mais recentemente mais próximos do processador, e da *localidade espacial,* movendo blocos que consistem em múltiplas palavras de memória contíguas para níveis mais altos da hierarquia.

Na maioria dos sistemas, a memória constitui uma verdadeira hierarquia, ou seja, para que os dados estejam presentes no nível i, devem estar presentes primeiro no nível i+1, e todos os dados estão presentes no nível mais baixo.

Os conceitos subjacentes à construção de sistemas de memória afetam muitos outros aspectos além da arquitetura do computador. Isso inclui como o sistema operacional gerencia a memória e a E/S, como os compiladores geram código e até mesmo como os aplicativos usam os recursos do computador.

Como todos os programas passam grande parte do tempo acessando a memória, o subsistema de memória é um fator determinante para o desempenho. Portanto, os programadores precisam hoje em dia entender que a memória é hierárquica para melhorar o desempenho de seus programas.

Hierarquia de memória - 7

Embora uma hierarquia de memória possa consistir em vários níveis, a transferência de dados geralmente ocorre apenas entre dois níveis adjacentes de cada vez, de modo que é possível concentrar-se no que acontece entre qualquer par de níveis para obter uma visão geral das operações.

A quantidade mínima de dados que pode estar presente ou ausente em uma hierarquia de dois níveis é chamada de *bloco.* Dizemos que ocorre um *hit* quando os dados solicitados pelo processador aparecem em algum bloco de nível superior; caso contrário, a solicitação é chamada de *miss* e o nível inferior é então acessado para recuperar o bloco que contém os dados solicitados.

A *taxa de acerto* ou *taxa de acerto* é a fração de referências de memória aos dados encontrados no nível superior em todas as referências de memória. Por outro lado, a *taxa de falta* ou *razão de falta* é o seu complemento de 1.

Como o desempenho é a questão principal, o tempo necessário para atender acertos e erros é relevante. O *tempo de acerto,* sendo o momento de acesso ao nível superior, compreende também o tempo para afirmar se o acesso é um acerto ou um erro. A *penalidade por falta,* então, é o momento de substituir um bloco de nível superior pelo bloco que contém os dados solicitados pelo processador.

Princípios de cache - 1

Cache é o nome tradicionalmente dado ao(s) nível(s) de hierarquia de memória localizados entre o processador e a memória principal. Hoje em dia, porém, o termo tem um significado mais amplo: refere-se a qualquer dispositivo de armazenamento que é gerido de uma forma que tira partido do princípio da localidade.

Ao lidar com cache, o termo *linha* é usado especificamente para se referir à quantidade mínima de informações que são transferidas entre qualquer par de níveis de cache ou armazenadas no nível de cache mais baixo. *O bloco* é reservado para se referir aos próprios dados armazenados em uma linha de cache ou na memória principal.

Assumindo um cache de nível único, a resposta às seguintes perguntas ajudará a esclarecer a forma como um cache funciona

- onde está localizada uma linha no cache? *(colocação de linha)*
- como é encontrado se lá estivesse presente? *(identificação da linha)*
- qual linha deve ser alterada em caso de falha? *(substituição de linha)*
- o que acontece em uma operação de gravação? *(escrever estratégia).*

Princípios de cache - 2

Como a capacidade da memória principal é muito maior que o tamanho do cache, muitos blocos de memória se sobreporão no mesmo local do cache ao longo do tempo. Existem várias maneiras de fazer isso, mas deve-se ter em mente que os principais objetivos são manter o hardware simples e todo o procedimento de armazenamento/ acesso eficiente.

Neste sentido, o tamanho do bloco não deve ser completamente arbitrário. É importante que o número de bytes armazenados seja uma potência de 2 para que um *endereço de memória* possa ser dividido trivialmente em um *endereço de bloco* e um *deslocamento.*

Em geral, o cache pode ser organizado como

- *mapeado diretamente* – quando há um único local onde um *bloco* pode ser colocado
- *totalmente associativo* – quando um *bloco* pode ser colocado em qualquer lugar
- *conjunto associativo* – quando existe um agregado de lugares, denominado *conjunto,* onde um *bloco* pode ser colocado.

Mapeados diretamente para organizações totalmente associativas formam um continuum de níveis de associatividade de conjunto: o tamanho do conjunto para *mapeamento direto* é um e para *associatividade total* é igual ao número de linhas no cache.

Princípios de cache - 3

endereço de memória

endereço do bloco (*s* bits)	deslocamento (*w* bits)

Caracterização da memória principal/cache

comprimento do endereço = $s + w$

bits número de unidades endereçáveis na memória principal $^{s+w}$ bytes

= 2 número de blocos na memória principal = 2 ^{é}

número de linhas no cache = m = 2 tamanho do ^{R}

cache = 2 $^{r+w}$ bytes

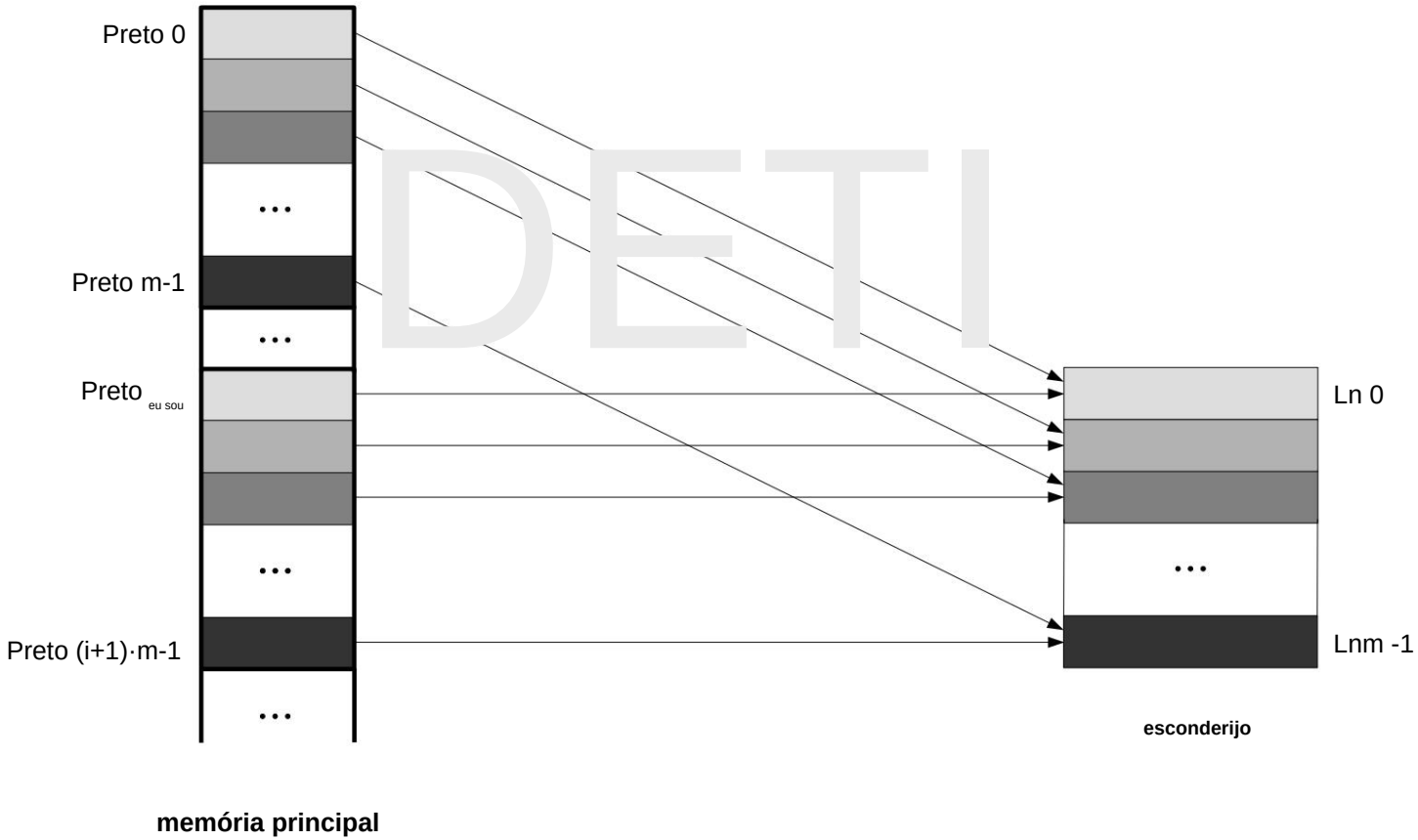
número de conjuntos no cache = v = 2 número ^{você}

de linhas por conjunto = $k = m / v$ = 2 ^{você}

Princípios de cache - 4

Mapeamento direto

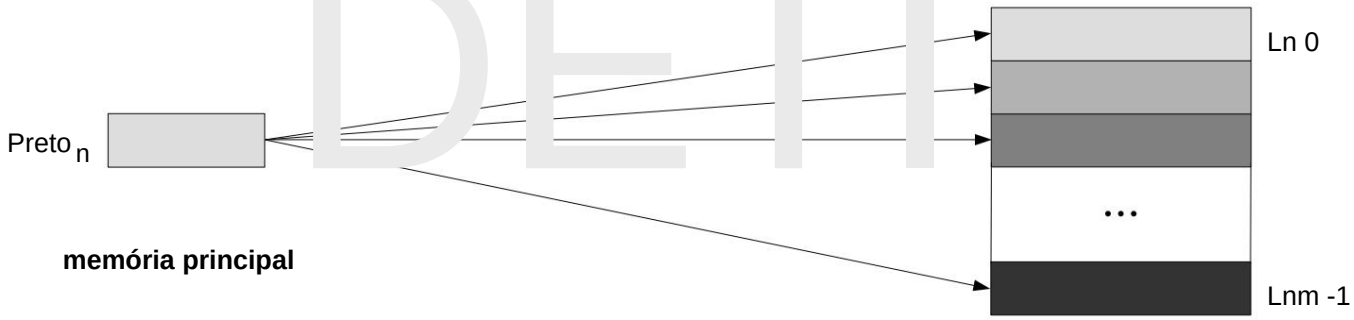
endereço da linha de cache = endereço do bloco *mod* número de linhas no cache



Princípios de cache - 5

Totalmente associatividade

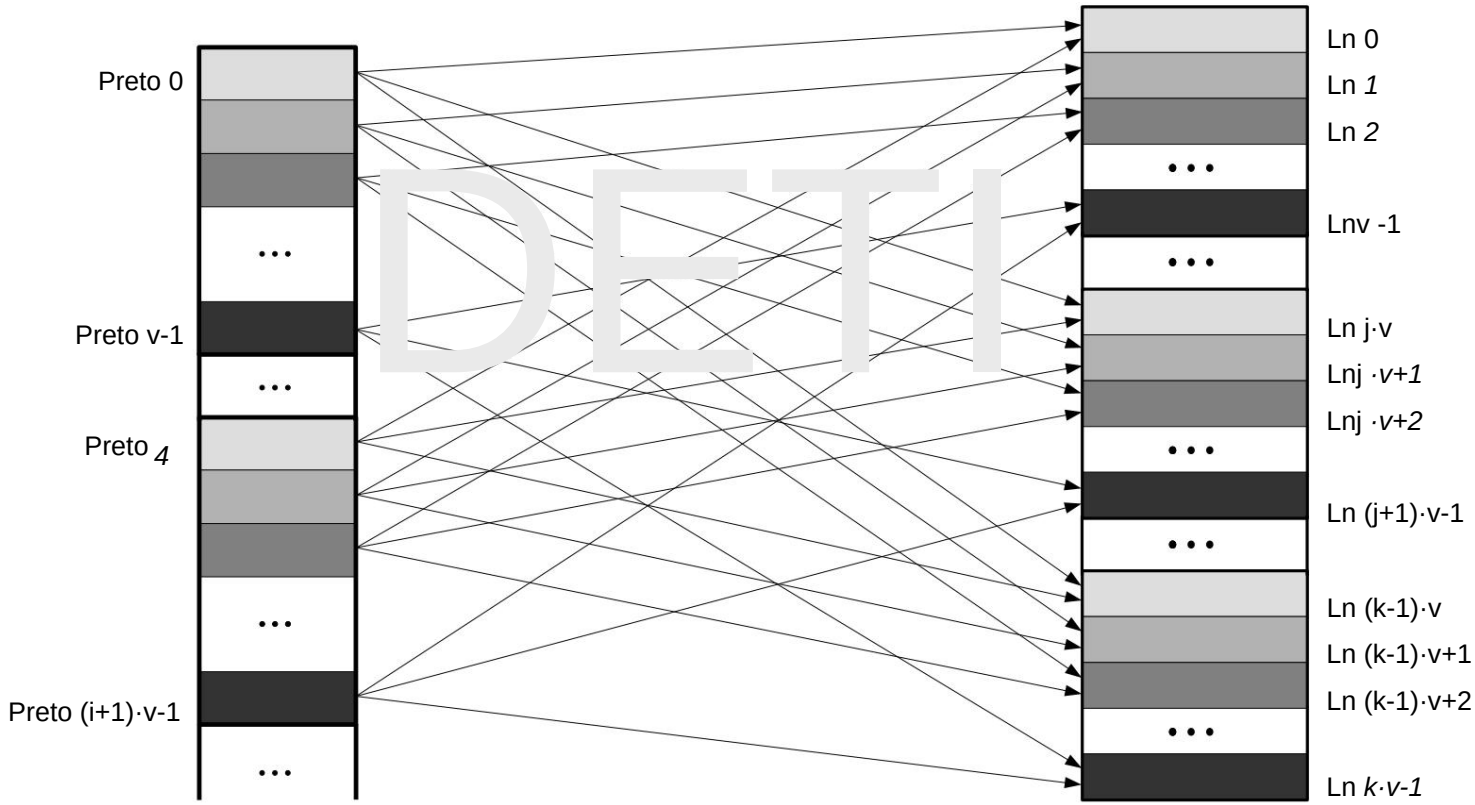
endereço da linha de cache = qualquer



Princípios de cache - 6

Definir associatividade

endereço da linha de cache = endereço do bloco *mod* número de conjuntos no cache



Princípios de cache - 7

Além do conteúdo de um bloco de memória principal, uma linha de cache também deve conter pelo menos parte de seu endereço, normalmente chamado de *campo de etiqueta de endereço do bloco.* O *campo tag* é armazenado em cada linha do cache para que possa ser verificado em relação à parte correspondente do endereço de memória gerado pelo processador para determinar se há uma correspondência quando ocorre um acesso. Via de regra, todas as *tags* possíveis são verificadas em paralelo para agilizar a busca. Além disso, como um registrador de dados nunca está vazio, um bit extra (chamado *bit de validação)* é necessário para afirmar se os dados atualmente armazenados na linha são significativos ou não.

Sempre que o *campo tag* de um endereço de bloco não for o endereço completo do bloco, os bits restantes do endereço formam um segundo campo, o *campo de índice,* cujo objetivo é selecionar um conjunto específico dentro do cache.

endereço de bloco

campo de tag *(su* bits)	campo de índice *(u* bits)

Princípios de cache - 8

Mapeamento direto

endereço de bloco

campo de tag *(sr* bits)	campo de índice *(r* bits)

Quando o cache é organizado através de *mapeamento direto,* existe uma única linha dentro do cache onde um determinado bloco de memória pode ser armazenado. Isso significa que o número de linhas por conjunto é igual a um, o número de conjuntos é igual ao número de linhas e o campo tag contido em cada linha tem comprimento mínimo.

Esta organização leva a implementações muito simples, rápidas e eficientes. A principal desvantagem é o risco de *thrashing:* um fenômeno que surge quando a fração do espaço de endereçamento referenciado pelo processador em um longo período de tempo contém grupos de dois ou mais endereços mapeados nas mesmas linhas de cache. Quando isso acontece, a taxa de acerto diminui bastante e a execução do programa torna-se bastante lenta porque nenhum benefício é obtido da localidade de referência.

Princípios de cache - 9

Totalmente associatividade

endereço de bloco

campo de tag (*s* bits)

Quando o cache é organizado por meio de *associatividade total,* todas as linhas dentro do cache ficam disponíveis para o armazenamento de um determinado bloco de memória. Isso significa que o número de linhas por conjunto é igual ao número de linhas no cache, o número de conjuntos é igual a um e o campo tag contido em cada linha tem comprimento máximo (o campo de índice não existe).

Esta organização leva à minimização da taxa de faltas, uma vez que em princípio um bloco de memória específico pode ser armazenado em qualquer uma das linhas da cache. A principal desvantagem é a complexidade de design que implica e, por isso, para uma determinada tecnologia de implementação e um determinado orçamento de energia, limita a velocidade de tomada de decisão sobre um acerto ou um erro.

Princípios de cache - 10

Definir associatividade

endereço de bloco

Quando o cache é organizado por meio de *associatividade de conjunto* (cache k-way), existem exatamente *k* linhas dentro do cache onde um determinado bloco de memória pode ser armazenado. Isso significa que o número de linhas por conjunto é igual a *k* e o número de conjuntos é igual a v.

Esta organização tenta alcançar *o melhor do mundo* retratado pelas duas organizações anteriores. Isso leva a implementações não muito complexas que ainda são rápidas e eficientes e evita o risco de *lixo ,* fornecendo alguma redundância ao local onde um bloco de memória pode ser armazenado.

Princípios de cache - 11

Quando ocorre uma falta, o controlador de cache deve selecionar uma linha cujo bloco será substituído pelos dados desejados. Com a organização *mapeada direta ,* o problema é trivial, pois apenas uma linha é verificada quanto a ocorrências e apenas o conteúdo desta linha pode ser modificado. Com organizações totalmente associativas e associativas em conjunto, há em princípio muitas, ou pelo menos algumas, linhas a serem consideradas. Se o *bit de validação* de qualquer uma dessas linhas for redefinido, uma delas poderá ser selecionada, mas depois de algum tempo todas elas conterão dados válidos e portanto uma decisão deverá ser tomada.

A estratégia óbvia, aquela que minimiza a taxa de falta, é escolher a linha dentro do grupo cujos dados não serão mais referenciados ou, se for, a referência acontecerá na maior distância do presente – o princípio da *otimalidade .* Infelizmente, esta regra não é causal, exigiria adivinhar o futuro e, portanto, não pode ser implementada na prática.

Princípios de cache - 12

As principais estratégias empregadas para seleção de linha são

- *aleatório* – um gerador pseudo-aleatório é usado para distribuir a substituição uniformemente entre as linhas candidatas; isso potencializa um comportamento reproduzível
- *menos recentemente utilizado* (LRU) – para aproximar o princípio da optimização confiamos no passado para prever o futuro; assim, a linha candidata é aquela que não foi referenciada por mais tempo
- *primeiro a entrar, primeiro a sair* (FIFO) – como LRU leva a uma implementação complexa, uma aproximação que é mais simples, mas ainda depende do passado para prever o futuro, é considerar a linha cujo conteúdo permaneceu por mais tempo de tempo no cache.

Princípios de cache - 13

Perdas de cache de dados por 1.000 instruções para a arquitetura Alpha (DEC) usando 10 benchmarks SPEC2000 (5 SPECint2000 e 5 SPECfp2000)

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

Definir associatividade (tamanho do bloco = 64 bytes)							
	2 vias 4 vias					8 vias	
Tamanho do cache	Aleatório	LRU	FIFO	Aleatório	LRU	FIFO	Aleatório	LRU	FIFO
16 KB	117,3	114,1	115,5	115,1	111,7	113,3	111,8	109,0	110,4
64 KB	104,3	103,4	103,9	102,1	102,3	102,4	103,1	100,5	99,7	100,3
256 KB	92,2	92,5	92,1	92,1	92,5	92,1	92,1	92,5

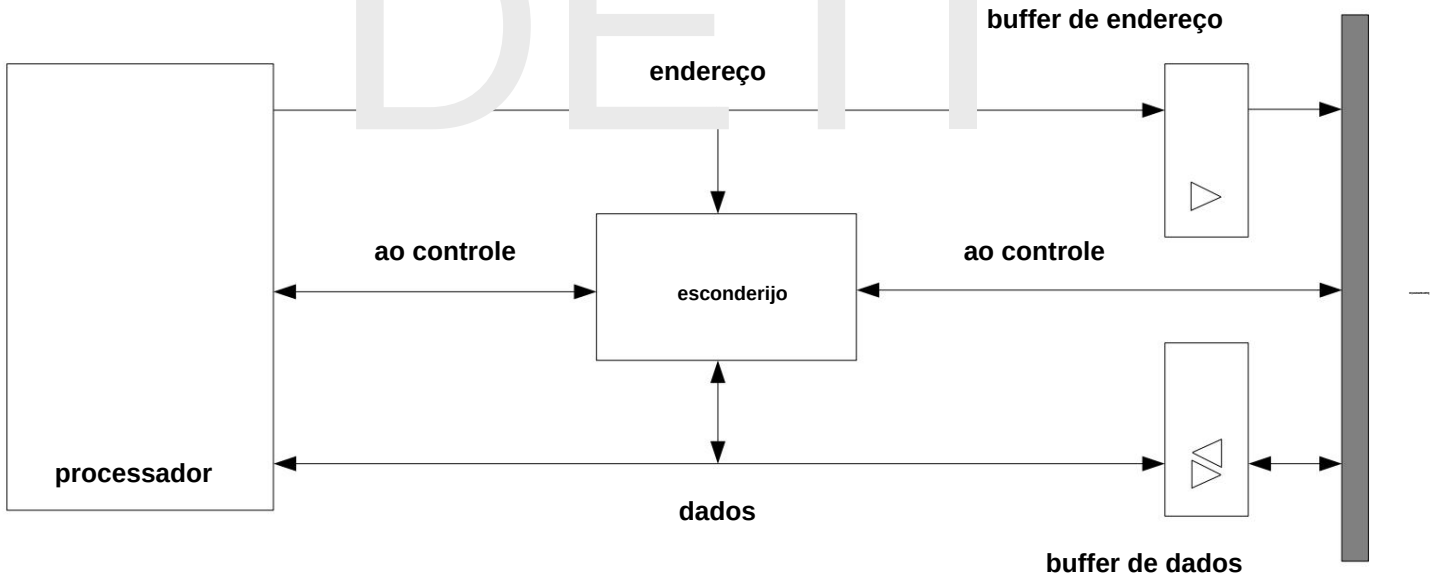
Princípios de cache - 14

As operações de *leitura* da memória dominam os acessos ao cache do processador. Isso ocorre porque todas as *buscas de instruções* são operações de leitura e a maioria das instruções não grava na memória. Na verdade, 10 benchmarks SPEC2000 (5 SPECint2000 / 5 SPECfp2000), rodando em um processador MIPS, sugerem uma média de 26%/30% de instruções *de carregamento* e de 10%/9% de instruções *de armazenamento* sobre todas as instruções executadas [Arquitetura do Computador: Uma abordagem quantitativa], o que leva a uma proporção de operações de *leitura* sobre *gravação* de 93:7/94:6.

Agilizar o *caso comum* significa otimizar caches para leituras, especialmente porque tradicionalmente os processadores esperam pela conclusão das leituras, mas não precisam esperar pela conclusão das gravações. Uma operação *de leitura* pode ser iniciada assim que o endereço do bloco que contém os dados referenciados for disponibilizado. Ao mesmo tempo, as tags das linhas candidatas são comparadas com o *campo de tag* do endereço do bloco para determinar se há um acerto. Nesse caso, a parte solicitada do bloco é imediatamente repassada ao processador; caso contrário, a transferência de bloco do nível inferior é iniciada e a operação de leitura é interrompida até que a transferência seja concluída.

Princípios de cache - 15

A fim de otimizar ainda mais os caches para operações de leitura, as organizações contemporâneas conectam o par processador-cache por meio de endereços, dados e linhas de controle. As linhas de endereço e de dados também se conectam a buffers de endereço e de dados que medeiam o acesso ao barramento do sistema a partir do qual a memória principal é acessada. Quando ocorre um acerto, os buffers de endereço e dados são desabilitados e toda a comunicação é interna. Quando ocorre uma falha, o endereço do bloco é carregado no barramento do sistema e os dados são retornados ao cache e ao processador.



Fonte: Organização de Computadores e Projeto de Arquitetura para Desempenho

Princípios de cache - 16

As operações *de gravação* são um pouco diferentes. A modificação do conteúdo do bloco só pode começar depois que a tag for verificada para determinar se há uma ocorrência. Assim, as operações de gravação geralmente demoram mais do que as operações de leitura. Além disso, embora o processador sempre especifique o tamanho e a localização dos dados, apenas para operações de escrita isso é crítico porque uma parte definida do conteúdo do bloco é alterada; para operações de leitura, o acesso a mais bytes de dados do que o necessário é irrelevante.

Existem duas *políticas* básicas de gravação

- *write-through* – os dados são gravados tanto na linha de cache quanto no bloco no nível mais baixo
- *write-back* – os dados são gravados na linha de cache; o conteúdo do bloco de linha só é gravado no nível inferior quando o bloco é substituído.

Quando a política de write-back é implementada, um recurso importante, conhecido como *bit sujo do bloco,* é comumente empregado para garantir que apenas os blocos modificados sejam transferidos de volta na substituição. Quando um bloco é transferido pela primeira vez para o cache, seu bit de status assume o valor *clean* para sinalizar que seu conteúdo não foi alterado; quando ocorre uma gravação, o bit de status assume o valor *sujo* e o bloco deve ser reescrito na substituição.

Princípios de cache - 17

Vantagens *de gravação* write-through

- é mais simples de implementar; como todas as operações de gravação resultam em uma gravação no nível inferior, as linhas de cache estão sempre limpas
- o conteúdo do bloco atualizado está sempre presente no nível inferior, o que simplifica a garantia da coerência dos dados
- desempenha um papel importante no projeto de caches multiníveis; para os níveis superiores, as gravações precisam apenas se propagar para o próximo nível inferior, em vez de chegar até a memória principal.

Vantagens *de write-back*

- operações de gravação para ocorrências ocorrem na velocidade do cache e múltiplas gravações no mesmo bloco requerem apenas uma gravação de volta para o nível inferior quando o bloco é substituído
- menos largura de banda de memória é usada, tornando-a atraente para multiprocessadores
- a energia também é economizada, tornando-o atraente para aplicações embarcadas.

Princípios de cache - 18

O processador para a conclusão das operações de gravação durante uma *gravação* write-through procedimento. Uma otimização comum para reduzir paralisações de gravação é implementar um *buffer de gravação,* que permite ao processador continuar assim que os dados são gravados no buffer, sobrepondo de fato a execução do processador à atualização da memória.

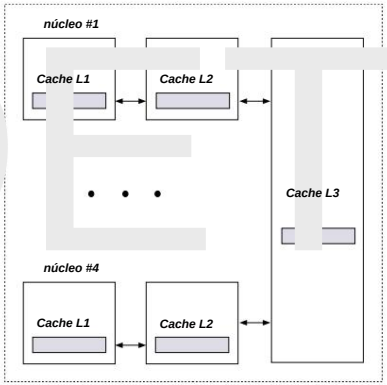
Uma *falha de gravação* pode ser tratada das seguintes maneiras

- *write allocate* – uma linha é alocada sempre que ocorre uma falha e então ocorre a operação de escrita; entretanto, se o tamanho do bloco for maior que os dados a serem gravados, o bloco deverá primeiro ser recuperado do nível inferior
- *alocação sem gravação* – o cache não é afetado por falhas, a operação de gravação ocorre apenas no nível inferior, o que significa que os blocos ficam fora do cache até que o processador leia os dados deles.

Qualquer uma *das políticas de falha de gravação* pode ser usada com qualquer uma das *políticas de gravação.* No entanto, *os caches de write-back* geralmente implementam a política *de alocação de gravação* na esperança de que as gravações subsequentes nesse bloco sejam capturadas pelo cache. Da mesma forma, *os caches write-through* implementam a política *de alocação sem gravação* porque todas as gravações devem ser escritas no nível inferior, portanto, nenhum ganho é obtido pela alocação do bloco.

Princípios de cache - 19

Para um processador multicore, onde múltiplas threads simultâneas podem estar competindo pelo acesso a dados compartilhados, protegidos ou não por regiões críticas, surge outro problema que tem a ver com a *coerência do cache.* Em tal situação, cópias do mesmo bloco de memória podem ser armazenadas em linhas de caches de nível 1 ou nível 2 associadas a diferentes processadores.



Para garantir que todos os processadores sempre vejam os mesmos dados, um *write-through* A política deve ser implementada para caches de nível 1 e nível 2 e quando ocorre uma *gravação ,* todas as cópias nesses níveis devem ficar *obsoletas* para que uma transferência do cache de nível 3 seja realizada se surgir uma *leitura* subsequente .

Princípios de cache - 20

Durante a execução de um programa, o processador não apenas busca instruções, mas também acessa a memória para carregar e armazenar operandos. Embora um cache único, *unificado* ou *misto* possa fornecer ambos, ele pode gerar um gargalo na comunicação. Uma situação típica em que isso acontece é quando um processador em pipeline está executando uma instrução *de carregamento* ou *armazenamento* e, ao mesmo tempo, está buscando a próxima. Para evitar este tipo de risco estrutural, é comum hoje em dia ter no nível mais alto dois caches: um dedicado a instruções e outro a dados.

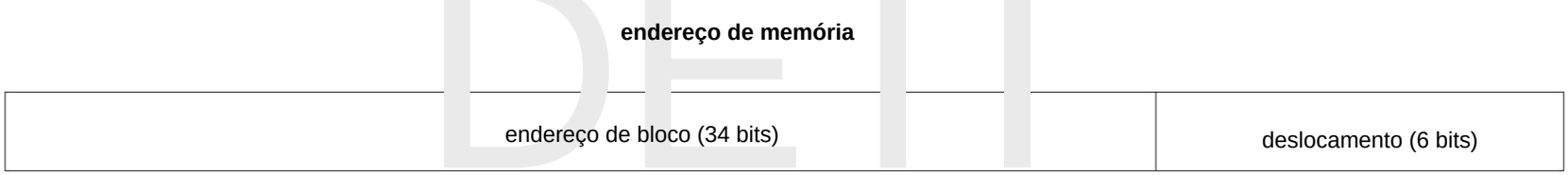
Assim, a largura de banda de comunicação com a memória pode ser duplicada através do uso de portas separadas.

Caches dedicados também permitem o ajuste individual de cada cache, especificando diferentes capacidades de armazenamento, tamanhos de bloco e associatividades para cada um.

O cache de dados do Opteron - 1

O cache de dados do Opteron tem tamanho de 64 KB e é organizado como um cache associativo de conjunto bidirecional, capaz de armazenar blocos de dados de 64 bytes. Ele apresenta uma estratégia de substituição de LRU e implementa uma política de *write-back* com *alocação de gravação* em caso de falha de gravação.

O Opteron apresenta um endereço de memória de 40 bits para a divisão do cache como segue.



comprimento do endereço = 40 bits

número de unidades endereçáveis na memória principal = 2^{40} bytes/2^{37} palavras

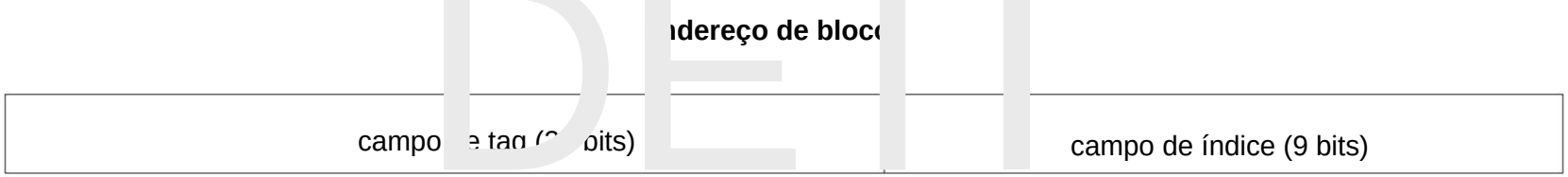
número de blocos na memória principal = 2^{34}

tamanho do bloco = 2^{6} = 64 bytes

O cache de dados do Opteron - 2

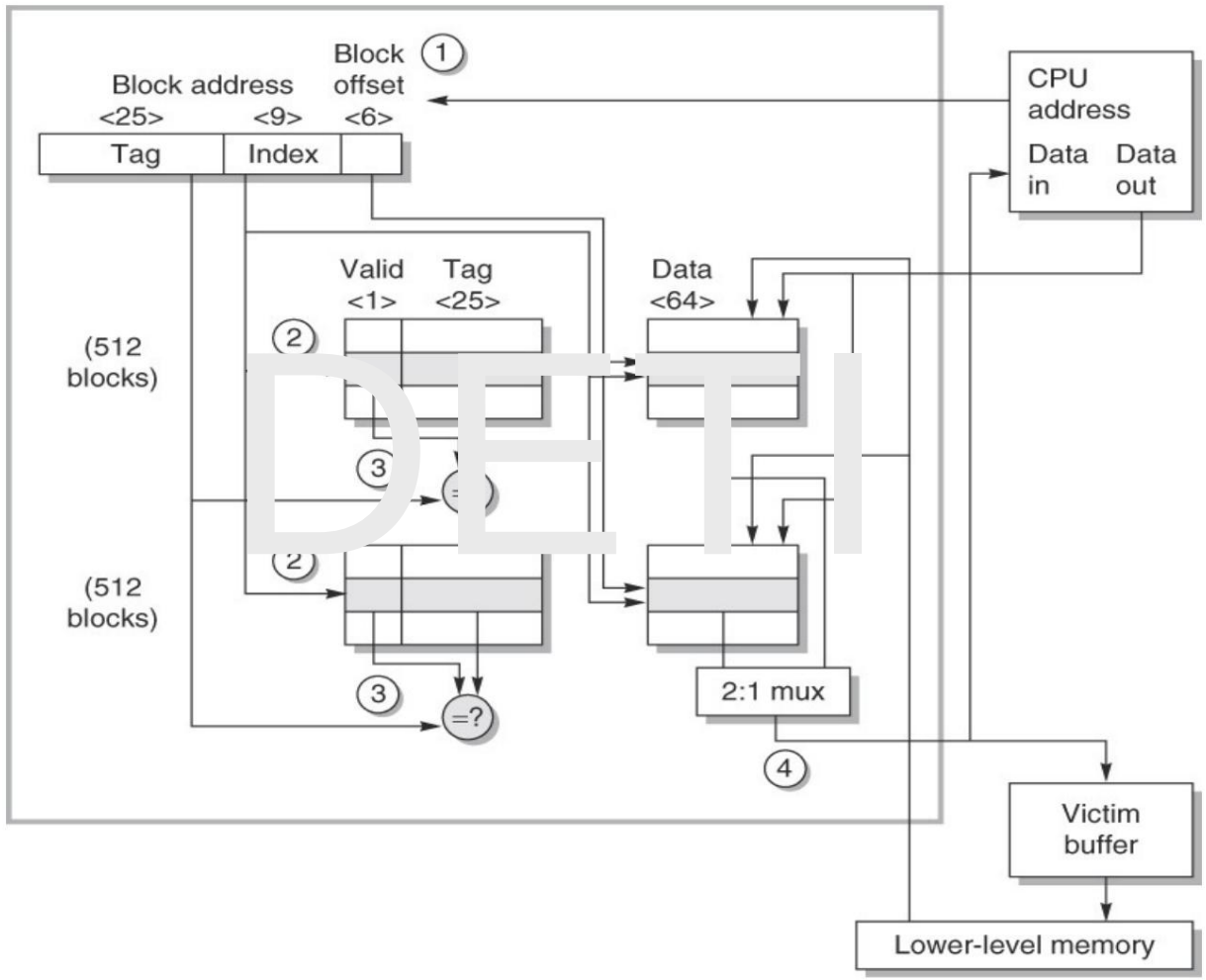
Para calcular o número de linhas no cache, fazemos

$$m = \frac{\text{tamanho do}}{\text{cache definir associatividadeÿtamanho do bloco}} = \frac{2^{16}}{2ÿ2^{6}} = 2^{9} = 512$$



Um ponto digno de nota é que, como o Opteron é um processador de 64 bits, cada acesso à memória tem no máximo 8 bytes de largura. Portanto, além dos 9 bits do *campo de índice* para selecionar o bloco de memória adequado, os 3 bits mais significativos do *deslocamento do bloco* também são usados para selecionar a palavra adequada dentro do bloco.

O cache de dados do Opteron - 3



Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

O cache de dados do Opteron - 4

Vários passos podem ser destacados num acesso à cache

- *etapa* 1 – o endereço de memória gerado pelo processador é dividido em duas partes: o *endereço do bloco,* os 34 bits mais significativos, que faz referência a um determinado bloco de memória, e o *deslocamento do bloco,* os 6 bits menos significativos, que faz referência a um byte específico dentro do bloco
- *passo* 2 – o endereço do bloco, por sua vez, também é dividido em duas partes: o *campo tag,* o mais significativo de 25 bits, que serve para determinar se o bloco está presente no cache, e o *campo de índice,* o menos significativo 9 bits, que faz referência ao conjunto dentro do cache onde o bloco pode ser armazenado
- *passo* 3 – se os *bits de validação* estiverem configurados, é realizada uma comparação entre os *tags* das duas linhas que pertencem ao conjunto referenciado e o *campo tag* do endereço do bloco
- *passo* 4 – quando ocorre um hit, o cache sinaliza ao processador para prosseguir com a leitura ou escrita dos dados; entretanto, como o Opteron é executado fora de ordem, a escrita só ocorre depois que o processador sinaliza que a instrução foi confirmada.

O Opteron permite 2 ciclos de clock para essas quatro etapas.

O cache de dados do Opteron - 5

Quando há uma falta, o cache envia um sinal ao processador informando que os dados ainda não estão disponíveis. Em seguida, o bloco necessário é lido no nível inferior da hierarquia, uma vez que o cache implementa uma política de *write-back* com *alocação de gravação* em caso de falha de gravação. A latência é de 7 ciclos de clock para os primeiros 8 bytes do bloco e, a seguir, de 2 ciclos de clock por 8 bytes para o restante do bloco.

Devido à associatividade de conjunto bidirecional apresentada pelo cache, devem começar com duas linhas onde o bloco pode ser armazenado. Se algum deles não contiver dados válidos, seu *bit de validação* é zerado, a linha é imediatamente selecionada; caso contrário, aplica-se a regra *menos utilizada recentemente ,* ou seja, a linha selecionada é aquela cujo *bit LRU* está zerado. Observe que a implementação da regra é, neste caso, quase trivial: após um acerto, o *bit LRU* da linha é setado e o *bit LRU* da linha homóloga no conjunto é reinicializado.

Antes de fazer isso, entretanto, por causa da política *de write-back ,* a *parte suja* da linha cujo bloco será substituído é verificada. Se estiver sujo, sua tag e dados serão primeiro removidos para o *buffer da vítima.* O cache fornece espaço para armazenar até oito *vítimas*
blocos que são posteriormente transferidos para o nível inferior em paralelo com outras atividades de cache. Se em algum momento, entretanto, o *buffer da vítima* estiver cheio, o processador terá que parar até que espaço seja disponibilizado.

O cache de dados do Opteron - 6

Formato de uma linha de cache

bit de validação	**etiqueta** (25 bits)	**parte suja**	**bit LRU**	**palavra de dados** (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)

Desempenho do cache - 1

Um método para avaliar *o desempenho do cache* é expandir a equação do *tempo de execução do processador.* A maneira de fazer isso é extrair da equação o número de ciclos de clock durante os quais o processador fica parado aguardando um acesso à memória, um parâmetro geralmente chamado de *ciclos de clock de travamento de memória.*

Então alguém consegue

Tempo de execução da CPU = (ciclos de clock da CPU + ciclos de clock de bloqueio de memória)ÿ
ÿtempo de ciclo do relógio.

Observe que a equação acima pressupõe que os *ciclos de clock variáveis da CPU* incluem o número de ciclos de clock para lidar com um acerto de cache e que o processador fica completamente paralisado durante uma falha de cache, o caso de um processador *de execução em ordem .*

Desempenho do cache - 2

O número de *ciclos de clock de bloqueio de memória* depende do *número de falhas* e no *custo por erro* ou *penalidade por erro*

ciclos de clock de travamento de memória = número de falhasÿ ciclos de clock de penalidade de falha =

= contagem de instruções ÿ $\frac{\text{sente falta}}{\text{instrução}}$ ÿperder ciclos de relógio de penalidade =

= contage de instruções $\frac{\text{acessos}}{\text{inst ção}}$ $\frac{\text{memória}}{\text{}}$

ÿtaxa de faltaÿciclos de relógio de penalidade de falta.

Obter o valor da *contagem de instruções* é simples se houver uma listagem da compilação do programa em linguagem assembly. Para processadores especulativos, entretanto, apenas as instruções comprometidas devem ser contadas. O valor dos *acessos à memória por instrução* pode ser estimado da mesma forma: toda instrução requer um acesso à memória para busca de instruções e, de acordo com sua semântica, pode ou não exigir um acesso aos dados.

Desempenho do cache - 3

A taxa de falhas pode ser medida com simuladores de cache que obtêm um rastreamento de endereço das instruções e referências de dados, simulam o comportamento do cache para determinar quais referências foram acertadas ou erradas e, no final, relatam seus totais. Muitos microprocessadores hoje fornecem hardware para contar tanto o número de falhas quanto de referências de memória em tempo real, o que facilita a tarefa.

A *penalidade de falha* variável é mais difícil de estimar. A memória por trás do cache pode estar ocupada no momento da falha devido a solicitações de memória anteriores, a uma atualização de memória ou mesmo a transferências de E/S que estão ocorrendo. O número de ciclos de clock também varia nas interfaces entre diferentes clocks do processador, do barramento ou da memória. Portanto, usar um valor único para *penalidade por falha* é uma simplificação.

Finalmente, como em muitos casos *a taxa de falta* e *a penalidade de falta* têm valores diferentes para operações de leitura e gravação, os ciclos de bloqueio de memória poderiam ser definidos levando este fato em consideração

ciclos de clock de parada de memória = contagem de instruções ÿ

$$\text{ÿÿ}_{i=r,w} \frac{\text{operações (i)}}{\text{instrução}} \text{ ÿtaxa de falta (i)ÿciclos de clock de penalidade de falta(i)} .$$

Desempenho do cache - 4

Suponha um sistema de computador onde o valor dos *ciclos de clock por instrução* (CPI) seja 1,0 quando todos os acessos à memória atingirem o cache. Os únicos acessos aos dados são instruções de carregamento e armazenamento, que representam 50% de todas as instruções executadas no benchmark em execução. Os *ciclos de clock de penalidade de falta* são 25 e a *taxa de falta* é de 2%.

Quão mais lento ele está executando em comparação com um sistema de computador com o mesmo processador e executando o mesmo programa sem falhas de cache ou, alternativamente, quão mais rápido ele está executando em comparação com um sistema de computador com o mesmo processador e executando o mesmo programa com 100 % de perdas de cache?

Taxa de falha de 0%

Tempo de execução CPU0mr = (ciclos de clock da CPU + ciclos de clock de bloqueio de memória)ÿ

ÿtempo de ciclo do relógio =

= (contagem de instruções ÿCPI + 0)ÿtempo do ciclo do relógio =

= 1,00ÿcontagem de instruções ÿtempo de ciclo do relógio

Desempenho do cache - 5

Taxa de falha de 2%

ciclos de clock de mem stall 2h contagem de instruções ÿ $\frac{\text{acessos à memória}}{\text{instrução}}$ =

ÿtaxa de faltaÿciclos de clock de penalidade de falta =

= contagem de instruções ÿ(1,0 + 0,5)ÿ0,02ÿ25 =

= 0,75ÿcontagem de instruções

Tempo de execução CPU2mr = (ciclos de clock da CPU + ciclos de clock de bloqueio de memória)ÿ

ÿtempo de ciclo do relógio

= = contagem de instruções ÿ(1,00+0,75)ÿtempo de ciclo do relógio =

= 1,75ÿcontagem de instruções ÿtempo de ciclo do relógio

Taxa de desempenho

$$\frac{\text{Tempo de execução CPU0mr}}{\text{Tempo de execução CPU2mr}} = \frac{1{,}00 \text{ÿcontagem de instruções ÿtempo de ciclo do relógio}}{1{,}75 \text{ÿcontagem de instruções ÿtempo de ciclo do relógio}} = 0{,}571.$$

Desempenho do cache - 6

Taxa de falha de 100%

ciclos de clock de mem stall $_{100\ senhor}$ = contagem de instruções ÿ acessos à memória = / instrução ÿ taxa de falha ÿ ciclos de clock de penalidade de falha = =

contagem de instruções ÿ (1,0 + 0,5) ÿ1,0ÿ25 =

= 37.5ÿcontagem de instrucões

CPU100 senhor = tempo de execução = (ciclos de clock da CPU + ciclos de clock de bloqueio de memória)ÿ

ÿtempo de ciclo do relógio =

= contagem de instruções ÿ(1,00+37,5)ÿtempo de ciclo de relógio = = 38,5ÿcontagem de instruções ÿtempo de ciclo de relógio

Taxa de desempenho

$$\frac{\text{Tempo de execução CPU100mr}}{\text{Tempo de execução CPU2mr}} = \frac{38{,}5 \text{ÿcontagem de instruções ÿtempo de ciclo do relógio}}{1{,}75 \text{ÿcontagem de instruções ÿtempo de ciclo do relógio}} = 22{,}0.$$

Desempenho do cache - 7

Como ilustra este exemplo, o comportamento do cache pode ter um impacto enorme no desempenho. As falhas de cache têm um impacto duplo em um processador com CPI baixo e clock rápido

- quanto menor for o CPI, maior será o impacto relativo da penalidade de falta independente do processador, medida como um número fixo de ciclos de clock
- mesmo que as hierarquias de memória de dois sistemas de computador sejam idênticas, o processador com a taxa de clock mais alta terá um número maior de ciclos de bloqueio de memória.

A importância do cache para processadores com baixo CPI e altas taxas de clock é assim maior e, portanto, maior é também o perigo de negligenciar o comportamento do cache na avaliação do desempenho destes sistemas computacionais.

Desempenho do cache - 8

Alguns projetistas preferem definir *a taxa de falhas* como o número de falhas por instrução, em vez do número de falhas por acesso à memória. Esses dois parâmetros estão relacionados por

$$\frac{\text{número total de erros}}{\text{contagem de instruções}} = \text{taxa de faltaÿ} \frac{\text{acessos à memória}}{\text{contagem de instruções}} .$$

A vantagem de usar *erros por instrução* para especificar a *taxa de erros* é que ele converte esse valor de mérito em um valor independente da implementação.
Processadores especulativos, por exemplo, buscam cerca de duas vezes mais instruções do que realmente são comprometidas. Assim, reduzindo artificialmente a taxa de faltas se medida como faltas por referência de memória.

A desvantagem é que *as falhas por instrução* dependem da arquitetura, o que significa que não faz sentido usá-lo para comparar duas arquiteturas bastante diferentes, como MIPS e Intel 80x86.

Desempenho do cache - 9

Em vez de se concentrar na *taxa de faltas* para avaliar o desempenho da hierarquia de memória, pode-se usar como melhor figura de mérito o *tempo médio de acesso à memória*

tempo médio de acesso à memória = tempo de acerto +

+ taxa de faltaÿciclos de relógio de penalidade de faltaÿtempo de ciclo de relógio.

O *tempo de acerto* variável é o tempo para acessar o cache em um acerto de cache e o outro variáveis, *taxa de falha* e *penalidade de falha,* têm o mesmo significado de antes.

Desempenho do cache - 10

Deve ser tomada uma decisão sobre a implementação de um cache de instruções de 16 KB e um cache de dados de 16 KB versus um cache unificado de 32 KB para um sistema de computador específico.

Os resultados da simulação em benchmarks adequadamente escolhidos indicaram os seguintes valores para o número de falhas por 1.000 instruções: 3,82 (cache de instruções de 16 KB), 40,9 (cache de dados de 16 KB) e 43,3 (cache unificado de 32 KB), onde 36% da totalidade de as instruções que foram executadas são transferência de dados. Suponha que um acerto leve 1 ciclo de clock e que a penalidade de falha seja 200. Suponha também que um acerto de carregamento ou armazenamento leve 1 ciclo de clock extra no cache unificado, pois há uma única porta para atender duas solicitações simultâneas e que o cache de dados e o cache unificado implementa uma política de gravação com um buffer de gravação (paralisações na gravação podem ser ignoradas).

$$\text{taxa de falta} = \frac{\dfrac{\text{erros por 1000 instruções}}{1000}}{\dfrac{\text{acessos à memória}}{\text{contagem de instruções}}}$$

Desempenho do cache - 11

O *cache de instruções* tem exatamente um acesso à memória por instrução

$$\text{taxa de falha de instrução de 16 KB} = \frac{3{,}82 \text{ÿ} 10^{\text{ÿ}3}}{1{,}00} = 0{,}004.$$

O *cache de dados* tem em média 0,36 acessos à memória por instrução

$$\text{taxa de perda de dados de 16 KB} = \frac{40{,}90 \text{ÿ} 10^{\text{ÿ}3}}{0{,}36} = 0{,}114.$$

O *cache unificado* tem em média 1,36 acessos à memória por instrução

$$\text{taxa de falta32 KB unificado} = \frac{43{,}30 \text{ÿ} 10 \text{ÿ} 3}{1{,}36} = 0{,}032.$$

Desempenho do cache - 12

A *taxa* efetiva de faltas dos caches combinados *de instrução* + *dados* é dada por

$$\text{taxa de falha16 KB de instruções + dados} = \frac{\text{instrução de acesso à mem}}{\text{mem acesso total}} \text{ ÿ taxa de falha de instrução de 16 KB} +$$

$$+ \frac{\text{mem acessa ados}}{\text{acesso à memória dadostotal}} \text{ ÿtaxa de falhas16 KB de} =$$

$$= \frac{,00}{1,36} \text{ ÿ0,004} + \frac{0,3}{1,36} \text{ ÿ0,114} = 0,033.$$

Observe que o cache unificado de 32 KB tem uma taxa de falhas um pouco menor do que a combinação de dois caches separados de instruções e dados de 16 KB!

Desempenho do cache - 13

Os *ciclos médios de clock de acesso à memória* são dados por

ciclos médios de clock de acesso à memória =

$$= \frac{\text{instrução de acesso à mem}}{\text{mem acesso total}} \text{ ÿ(tempo de acerto + instrução de taxa de falhaÿ ciclos de relógio de penalidade de falha) +}$$

$$+ \frac{\text{mem acess dado}}{\text{mem acess total}} \text{ ÿ(t certo + dos de t a de falhaÿ ciclos de clock de penalidade de falha)}$$

rendendo em cada caso

ciclos médios de clock de acesso à memória32 KB unificado =

$$= \frac{1,00}{1,36} \text{ ÿ(1,0+0,032ÿ200) +} \quad \frac{0,36}{1,36} \text{ ÿ(2+0,032ÿ200) = 7,66}$$

Desempenho do cache - 14

ciclos médios de clock de acesso à memória16 KB de instrução + dados =

$$= \frac{1,00}{1,36} \text{ ÿ}(1,0+0,004\text{ÿ}200) + \frac{0,36}{1,36} \text{ ÿ}(1+0,114\text{ÿ}200) = 7,62 .$$

Embora os dois caches separados de instruções e dados de 16 KB tenham uma taxa efetiva de faltas um pouco maior que o cache unificado de 32 KB, o fato é que quando se considera o tempo médio de acesso à memória, o resultado é invertido devido à existência neste caso de dois portas de memória por ciclo de clock, evitando assim o risco estrutural presente no cache unificado de porta única.

Desempenho do cache - 15

Qual é o impacto de duas organizações de cache diferentes no desempenho de um processador específico?

Suponha que o CPI com cache perfeito (sem perdas de cache) seja 1,6, que o ciclo de clock seja 0,35 ns e que haja 1,4 referências de memória por instrução para o benchmark em execução. O tamanho de ambos os caches é de 128 KB e ambos possuem tamanho de bloco de 64 bytes. O primeiro é organizado como mapeado direto e o segundo como associativo de conjunto bidirecional. Como ilustra o cache de dados do Opteron, um multiplexador deve ser adicionado para selecionar entre os dados do conjunto, dependendo da correspondência da tag. Como a velocidade do processador pode ser vinculada diretamente à velocidade de um acerto no cache, suponha que o tempo de ciclo do clock do processador seja esticado 1,35 vezes para acomodar o multiplexador de seleção do cache associativo definido. Em ambos os casos, o tempo de acerto é de 1 ciclo de clock, a penalidade de falta é de 65 ns e a taxa de falta é de 2,1% e 1,9%, respectivamente, para o mapa direto e os caches associativos de conjunto bidirecional.

Desempenho do cache - 16

tempo médio de acesso à memória128KB $_{\text{mapeado diretamente}}$ =

= ciclos de clock de acertoÿtempo de ciclo de clock + taxa de falhaÿpenalidade de falha =

= 1,0ÿ0,35 + 0,021ÿ65 = 1,72ns

tempo médio de acesso à memória128KB $_{\text{Conjunto associativo de 2 vias}}$ =

= ciclos de clock de acertoytempo efetivo do ciclo de clock + taxa de falhaÿpenalidade de falha =

= 1,0ÿ1,35 ÿ0,35 + 0,019ÿ65 = 1,71 ns

Observe que o cache associativo de conjunto bidirecional de 128 KB tem um tempo médio de acesso à memória um pouco menor do que o cache mapeado direto de 128 KB!

Desempenho do cache - 17

Tempo de execução da CPU128 KB mapeado diretamente = contagem de instruçõesÿCPIÿtempo de ciclo do relógio + + contagem de instruçõesÿ $\frac{\text{acessos à memória}}{\text{instrução}}$ ÿtaxa de falhaÿpenalidade de falha =

= (1,6ÿ0,35 + 1,4ÿ0,021ÿ65)ÿcontagem de instruções = 2,47ÿcontagem de instruções

Tempo de execução da CPU128 KB Conjunto de 2 vias associativo = contagem de instruçõesÿCPIÿtempo de ciclo de clock efetivo + + contagem de instruçõesÿ $\frac{\text{acessos à memória}}{\text{instrução}}$ ÿtaxa de falhaÿpenalidade de falha =

= (1,6ÿ1,35ÿ0,35 + 1,4ÿ0,019ÿ65)ÿcontagem de instruções = 2,49ÿcontagem de instruções

Embora o cache associativo conjunto bidirecional de 128 KB tenha um tempo médio de acesso à memória menor que o cache mapeado direto de 128 KB, o fato é que quando se considera o tempo de execução da CPU, o resultado é invertido devido ao estiramento do ciclo de clock em o caso do cache associativo.

Desempenho do cache - 18

Para um processador *de execução fora de ordem , a penalidade de falta* não pode mais ser definida como a latência total da falta na memória. Esta questão é relevante porque *fora de ordem*
sabe-se que os processadores toleram alguma latência devido a falhas de cache sem afetar seu desempenho.

O número de *ciclos de bloqueio de memória* pode ser redefinido para levar a uma nova definição de *penalidade de falta* como uma latência não sobreposta

$$\text{ciclos de parada de memória} = \text{contagem de instruções} \times \frac{\text{acessos à memória}}{\text{instrução}} \times$$

$$\times \text{taxa de falta} \times (\text{penalidade total de falta} - \text{latência de falta sobreposta}).$$

Diz-se que um *processador fica paralisado em um ciclo de clock* se não retirar o máximo número possível de instruções nesse ciclo. A *barraca* é atribuída à primeira instrução que não pôde ser retirada. *A latência,* por outro lado, pode ser medida desde o momento em que a instrução de memória é enfileirada na janela de instruções, ou desde o momento em que o endereço é gerado, até o momento em que ocorre a transferência de dados. Qualquer alternativa é válida desde que seja usada de forma consistente.

Desempenho do cache - 19

Considere que o processador do último exemplo possui um tempo de ciclo de clock 1,35 vezes maior para suportar execução fora de ordem e possui um cache mapeado diretamente. Suponha também que 30% da penalidade de falha de 65 ns possa ser sobreposta, reduzindo assim o ciclo médio de bloqueio da memória para 45,5 ns.

tempo médio de acesso à memória direto $_{\text{mapeado, OOO}}$ =

= ciclos de clock de acertoÿ tempo de ciclo de clock + taxa de falhaÿ penalidade de falha efetiva =

= 1,0ÿ1,35ÿ0,35 + 0,021ÿ45,5 = 1,43 ns

Tempo de execução da CPU direto $_{\text{mapeado, OO}}$ =

= contagem de instruções yCPIÿtempo de ciclo do relogio =

+ contagem de instruções ÿ $\frac{\text{acessos à memória}}{\text{instrução}}$ ÿtaxa de falhaÿpenalidade de falha efetiva =

= (1,6ÿ1,35ÿ0,35 + 1,4ÿ0,021ÿ45,5)ÿcontagem de instruções = 2,09ÿcontagem de instruções

Porém, devido à sua complexidade, os projetistas tendem a usar simuladores do processador fora de ordem e da memória ao avaliar os trade-offs na hierarquia de memória para avaliar se uma melhoria que ajuda a latência média da memória, também ajuda o programa. desempenho.

Otimização de cache - 1

A abordagem tradicional para melhorar o comportamento de um cache é minimizar a perda avaliar. As falhas podem ser modeladas em três categorias básicas

- *faltas compulsórias* – faltas que ocorrem mesmo que a cache tenha tamanho infinito; o primeiro acesso a qualquer byte ou palavra sempre se traduzirá em uma falha porque o bloco de memória onde reside o byte ou palavra deve ser trazido primeiro para o cache; eles também são conhecidos como *erros de partida a frio* ou *erros de primeira referência*

- *faltas de capacidade* – faltas que ocorrem em uma cache totalmente associativa; eles estão diretamente relacionados ao tamanho do cache, se o cache não for grande o suficiente, os blocos de memória serão descartados durante a execução do programa e posteriormente serão recuperados porque serão necessários novamente
- *faltas de conflito* – faltas que ocorrem especificamente devido à organização interna da cache; deixando de lado o caso de um cache totalmente associativo e considerando o cache mapeado direto como uma instância de um cache associativo de conjunto unidirecional, os blocos de memória podem ser descartados e posteriormente recuperados simplesmente porque muitos blocos são mapeados em alguns dos conjuntos; eles também são conhecidos como *erros de colisão.*

Otimização de cache - 2

Taxa de falha para a arquitetura Alpha (DEC) usando 10 benchmarks SPEC2000 (5 SPECint2000 e 5 SPECfp2000) – substituição de LRU – tamanho do bloco = 64 bytes

Fonte: adaptado de Computer Architecture: A Quantitative Approach

			Componentes da taxa de falta (valor/ % relativa do total)					
Tamanho da memória cache (KB)	Grau de Associatividade	Falta total Avaliar	Obrigatório		Capacidade		Conflito	
4	Unidirecional	0,098	0,0001	0,1%	0,070	72%	0,027	28%
4	2 maneiras	0,076	0,0001	0,1%	0,070	93%	0,005	7%
4	4 vias	0,071	0,0001	0,1%	0,070	99%	0,001	1%
4	8 vias	0,071	0,0001	0,1%	0,070	100%	0,000	0%
16	1 via	0,049	0,0001	0,1%	0,040	82%	0,009	17%
16	2 vias	0,041	0,0001	0,2%	0,040	98%	0,001	2%
16	4 vias	0,041	0,0001	0,2%	0,040	99%	0,000	0%
16	8 vias	0,041	0,0001	0,2%	0,040	100%	0,000	0%
64	1 via	0,037	0,0001	0,2%	0,028	77%	0,008	23%
64	2 vias	0,031	0,0001	0,2%	0,028	91%	0,003	9%
64	4 vias	0,030	0,0001	0,2%	0,028	95%	0,001	4%
64	8 vias	0,029	0,0001	0,2%	0,028	97%	0,001	2%
256	1 via	0,013	0,0001	0,5%	0,012	94%	0,001	6%
256	2 vias	0,012	0,0001	0,5%	0,012	99%	0,000	0%
256	4 vias	0,012	0,0001	0,5%	0,012	99%	0,000	0%
256	8 vias	0,012	0,0001	0,5%	0,012	99%	0,000	0%

Otimização de cache - 3

Como seria de esperar, *as faltas compulsórias* são independentes do tamanho da cache, *as faltas de capacidade* diminuem à medida que o tamanho da cache aumenta e *as faltas de conflito* diminuem à medida que o grau de associatividade aumenta.

Aumentar o tamanho do cache é importante. Se a memória de nível superior for muito pequena para conter a fração do código do programa e dos dados necessários ao princípio da localidade, então uma parte significativa do tempo será gasta na transferência de blocos de memória entre dois níveis adjacentes da hierarquia e ocorrerá *thrashing .* Conseqüentemente, o sistema do computador funcionará mais próximo da velocidade da memória de nível inferior, ou até mais lento devido à sobrecarga de falta. Por outro lado, implementar a associatividade total para eliminar as falhas de conflito é caro em termos de hardware e pode levar a uma taxa de clock lenta, diminuindo assim o desempenho geral.

O *modelo de erros* que acabamos de apresentar tem os seus próprios limites: dá uma visão do comportamento médio, mas não explica os erros individuais, nem incorpora a política de substituição. Muitas técnicas que reduzem as taxas de erro também aumentam o *tempo de acerto* e/ou a *penalidade de erro.* Portanto, é necessária uma abordagem equilibrada para tornar todo o sistema de computação mais rápido.

Otimização de cache - 4

Uma maneira direta de reduzir a taxa de falhas é ***aumentar o tamanho do bloco.*** Tamanhos de blocos maiores diminuirão *as falhas obrigatórias* devido à localidade espacial. Entretanto, se o tamanho do bloco for muito grande em relação ao tamanho do cache, a redução no número de linhas de cache tende a aumentar a *capacidade* e *as faltas de conflito* e a piorar a taxa total de faltas.

Taxa de falha versus tamanho do bloco para DECStation 5000 usando benchmarks SPEC92

Fonte: Gee, Hill, Pnevimatikatos, Smith, “Desempenho de cache do conjunto de benchmark SPEC92”, IEEE Micro 13:4, agosto de 1993

	Tamanho da memória cache			
Tamanho do bloco (bytes)	**4 *KB***	**16 *KB***	**64 *KB***	**256 *KB***
16	8,57%	3,94%	2,04%	1,09%
32	7,24%	2,87%	1,35%	0,70%
64	**7,00%**	**2,64%**	1,06%	0,51%
128	7,78%	2,77%	**1,02%**	**0,49%**
256	9,51%	3,39%	1,15%	**0,49%**

Otimização de cache - 5

Ao mesmo tempo, tamanhos de bloco maiores tendem a aumentar a *penalidade por falta,* pois o número de bytes a serem transferidos também aumenta.

penalidade de falta = ciclos de clock de latência de cacheÿtempo de ciclo de clock +

+ tamanho do bloco/largura de banda (ciclos de clock)ÿtempo de ciclo de clock.

Ciclos médios de clock de acesso à memória versus tamanho do bloco para DECStation 5000 usando benchmarks SPEC92

Fonte:Arquitetura de Computadores: Uma Abordagem Quantitativa

ciclos de clock = 1

ciclos de clock de latência de cache = 80

largura de banda = 8 bytes/ciclo de clock

		Tamanho da memória cache			
Tamanho do bloco (bytes)	***Faltar pênalti (ciclos de clock)***	**4 *KB***	**16 *KB***	**64 *KB***	**256 *KB***
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Otimização de cache - 6

Como a escolha do tamanho do bloco será afetada pela latência e pela largura de banda da memória de nível inferior, o projetista do cache deve considerar tanto a taxa de falta quanto a penalidade de falta ao decidir sobre o tamanho do bloco de cache.

Alta latência e alta largura de banda incentivam um tamanho de bloco grande porque o cache recebe muito mais bytes por falta para um pequeno aumento na penalidade de falta. Por outro lado, baixa latência e baixa largura de banda incentivam blocos menores porque o tempo economizado com um bloco maior não é significativo. É preciso lembrar que ter um grande número de pequenos blocos pode reduzir os erros de conflito.

As perdas de capacidade são obviamente reduzidas com o aumento do tamanho do cache. Porém, há um limite para isso porque o tempo de acerto é potencialmente maior devido ao aumento da complexidade lógica, tornando-o também uma solução de custo mais elevado e de maior potência. Apesar disso, sua técnica tem sido especialmente popular em caches localizados fora do chip do processador.

Otimização de cache - 7

Da mesma forma, embora uma maior associatividade reduza a taxa de erros, a média o tempo de acesso à memória nem sempre também é diminuído.

Ciclos médios de clock de acesso à memória versus associatividade para DECStation 5000 usando benchmarks SPEC92 – substituição de LRU – tamanho do bloco = 64 bytes

Fonte:Arquitetura de Computadores: Uma Abordagem Quantitativa

ciclos de clock = 1

tempo de ciclo de relógio e 2 vias 1,3ÿ tempo de ciclo de relógio de 1 via

tempo de ciclo de relógio e 4 vias = 1,4ÿ tempo de ciclo de relógio de 1 via

tempo de ciclo de relógio de 8 vias = 1,52ÿ tempo de ciclo de relógio de 1 via

	Associatividade			
Tamanho da memória cache **(KB)**	***1 via***	***2 vias***	***4 vias***	***8 vias***
16	2.23	2h40	2,46	2,53
32	2.06	14h30	2,37	2,45
64	1,92	2.14	2.18	2,25
128	1,52	1,84	1,92	2h00
256	1,32	1,66	1,74	1,82

Memória principal

A memória principal satisfaz principalmente as demandas dos caches e serve como interface de E/S tanto para a área de troca em uma organização de memória virtual quanto para os diferentes controladores de dispositivos, atuando como destino para dados de entrada e fonte para dados de saída.

Tradicionalmente, *a latência da memória,* que afeta a penalidade por falta de cache, é a principal preocupação do cache, enquanto *a largura de banda da memória* é a principal preocupação dos controladores de E/S, mas também é importante para caches multiníveis com seus tamanhos de bloco maiores.
Maior largura de banda pode ser alcançada usando múltiplos bancos de memória, ampliando o barramento de dados e introduzindo *o modo de transferência em rajada,* onde múltiplas palavras são transferidas no mesmo acesso.

A memória principal é construída em torno de chips DRAM. Atualmente, *a latência da memória* é expressa através de duas figuras de mérito

- *tempo de acesso* – intervalo de tempo entre a emissão de uma solicitação de leitura/gravação e o instante em que os dados associados se tornam disponíveis/são armazenados
- *tempo de ciclo* – intervalo mínimo de tempo entre solicitações de memória não relacionadas.

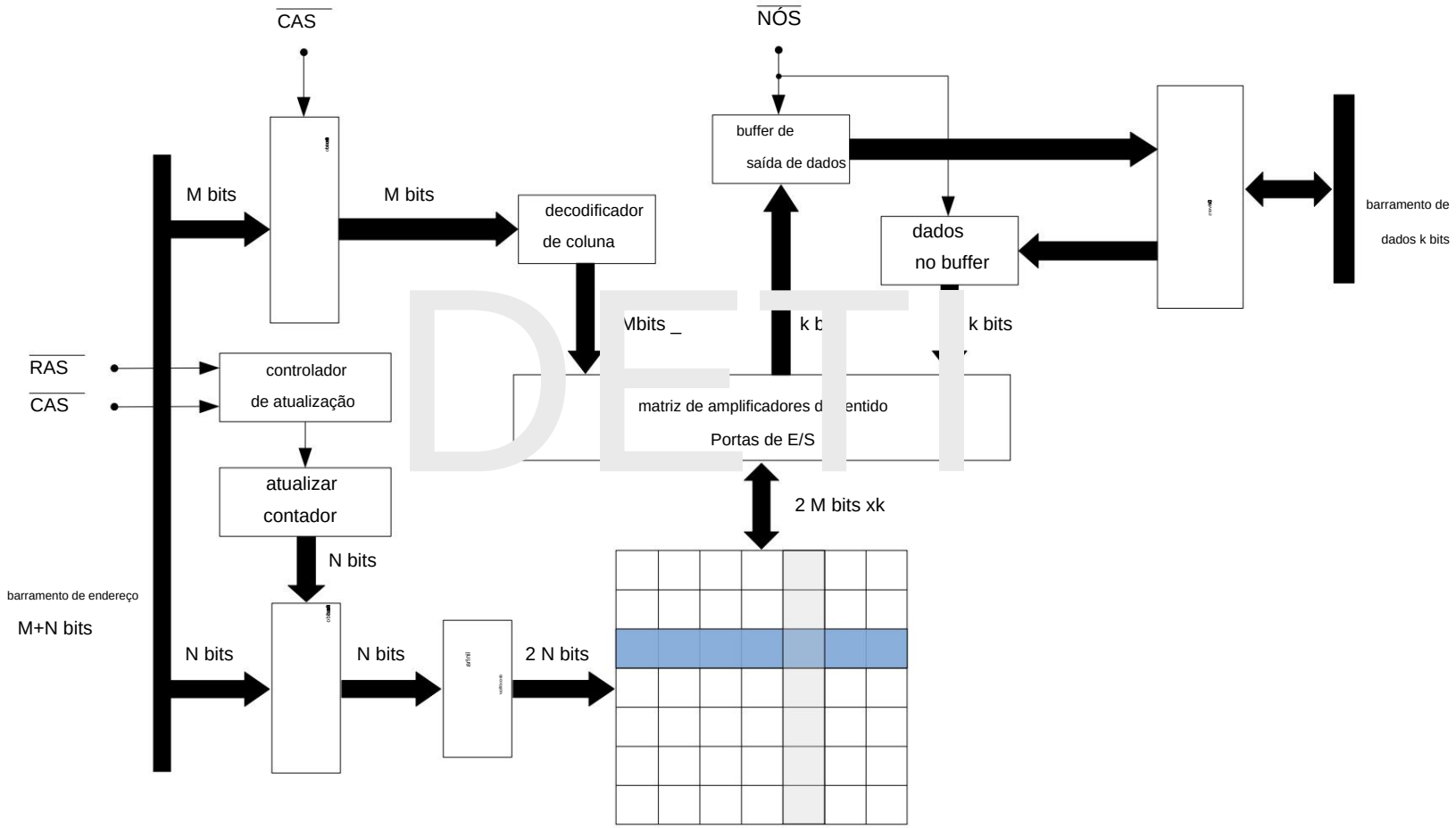
DRAM

À medida que a capacidade das RAMs dinâmicas cresceu, o custo do pacote com todos os pinos de endereço necessários tornou-se um problema. Para resolvê-lo, as linhas de endereço foram multiplexadas: parte do endereço é travada internamente durante a primeira fase de acesso à memória, com o sinal de controle *estroboscópico de acesso à linha ,* e o endereço restante é travado posteriormente, durante a segunda fase, com o sinal de controle *estroboscópico de acesso à coluna .*

Para armazenar mais bits por chip, um único transistor é usado para armazenar um bit.
A leitura deste bit destrói as informações armazenadas, o que significa que elas devem ser restauradas (escritas de volta) após cada leitura. Esta é uma das razões pelas quais o tempo de ciclo da memória é maior que o tempo de acesso à memória. Com a introdução de múltiplos bancos DRAM, que permitem ocultar a parte de reescrita do ciclo, este problema foi atenuado.

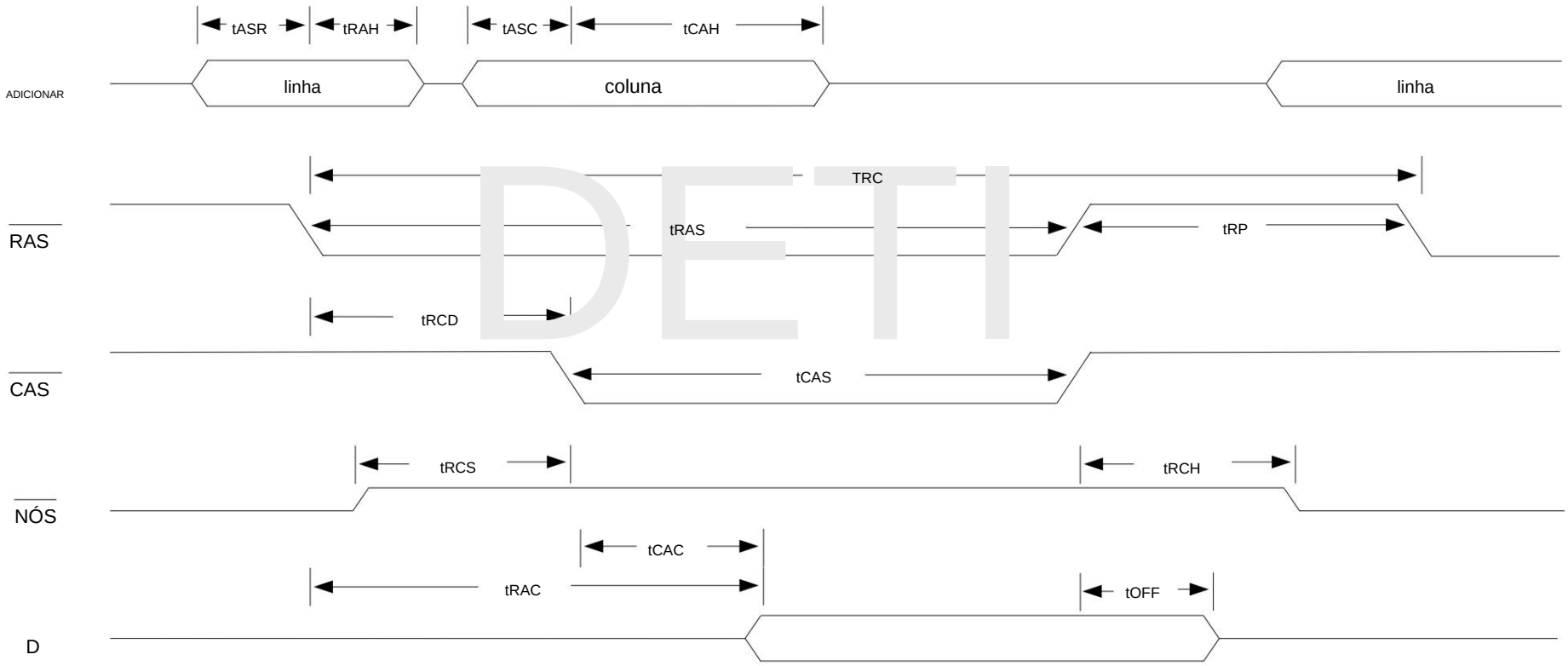
Além disso, para evitar a perda de informações quando um bit armazenado não é lido por um longo período de tempo, deve ocorrer uma atualização de carga. Todos os bits pertencentes à mesma linha podem ser atualizados simultaneamente apenas acessando uma coluna dessa linha. Conseqüentemente, devem ser fornecidos meios para que cada DRAM acesse cada uma de suas linhas dentro de um determinado período de tempo, normalmente algumas dezenas de milissegundos. Os controladores de memória possuem hardware para realizar esta tarefa.

DRAM assíncrona - 1



DRAM assíncrona - 2

Operação de leitura



DRAM assíncrona - 3

Operação de leitura

- o sinal *de habilitação de gravação* é declarado alto pelo *controlador de memória* para significar que uma operação de leitura está ocorrendo
- um endereço de N bits é colocado no barramento de endereço e travado no *buffer de endereço de linha* na borda descendente do *estroboscópio de acesso de linha* afirmado pelo *controlador de memória*

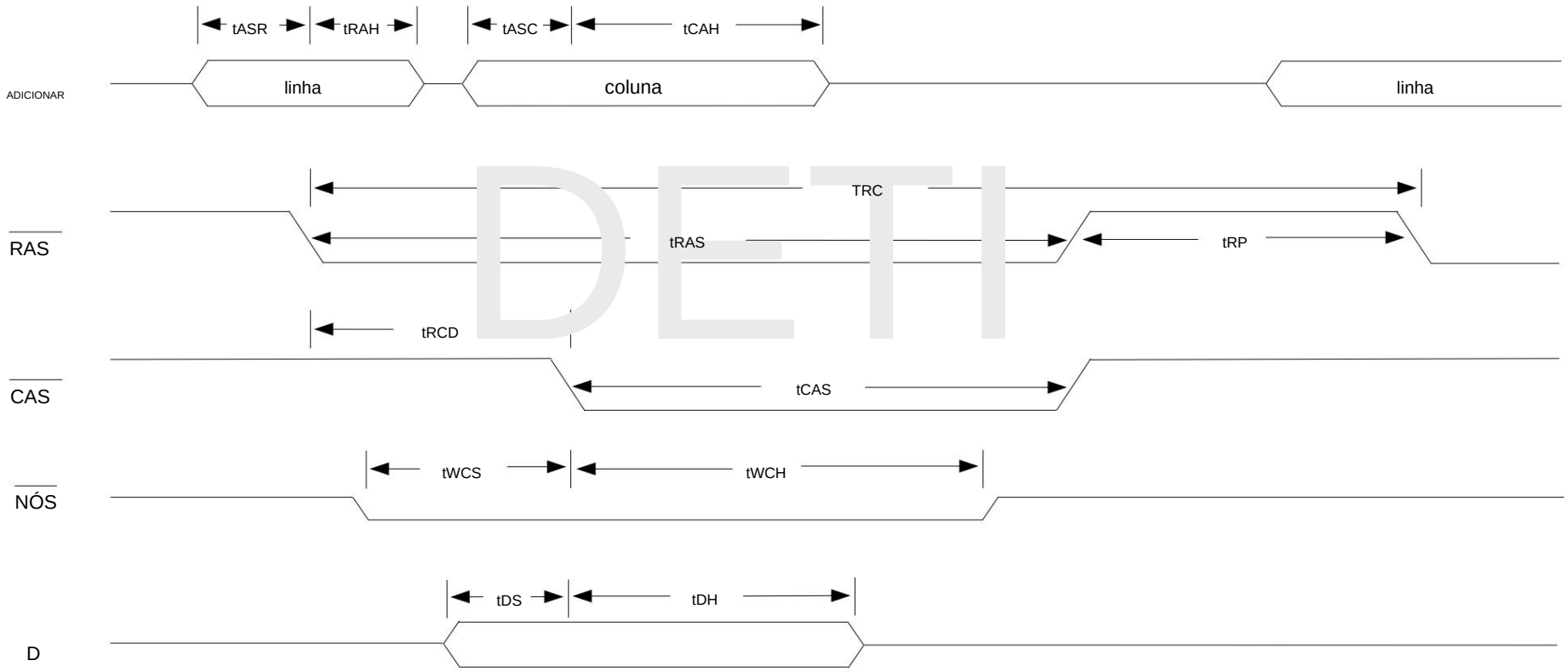
- os valores de dados armazenados na linha selecionada de células de memória são então detectados e mantidos no *conjunto de amplificadores de sentido* para serem posteriormente gravados de volta nas células de memória

- um endereço de M-bit é colocado em seguida no barramento de endereço e é travado no *buffer de endereço da coluna* na borda descendente do *estroboscópio de acesso à coluna* afirmado pelo *controlador de memória*

- os valores de dados mantidos na coluna selecionada da *matriz de amplificadores de detecção* são travados no *buffer de saída de dados* para acionar o barramento de dados

DRAM assíncrona - 4

Operação de gravação



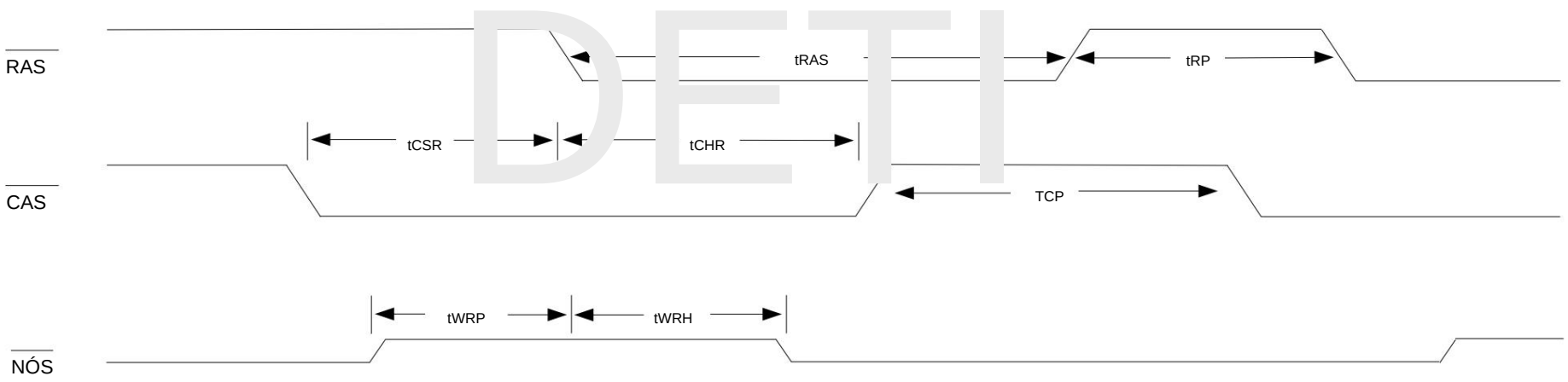
DRAM assíncrona - 5

Operação de gravação

- o sinal *de habilitação de gravação* é considerado baixo pelo *controlador de memória* para significar que uma operação de gravação está ocorrendo
- um endereço de N bits é colocado no barramento de endereço e travado no *buffer de endereço de linha* na borda descendente do *estroboscópio de acesso de linha* afirmado pelo *controlador de memória*
- os valores de dados armazenados na linha selecionada de células de memória são então detectados e mantidos no *conjunto de amplificadores de sentido* para serem posteriormente gravados de volta nas células de memória
- um endereço de M-bit é colocado em seguida no barramento de endereço e é travado no *buffer de endereço da coluna* na borda descendente do *estroboscópio de acesso à coluna* afirmado pelo *controlador de memória*
- os valores de dados presentes no barramento de dados são armazenados nos *dados no buffer* na borda descendente do *estroboscópio de acesso à coluna* e substituem os valores armazenados na coluna selecionada da *matriz de amplificadores de sentido* antes da gravação de volta nas células de memória

DRAM assíncrona - 6

Operação de atualização (CAS antes de RAS)



DRAM assíncrona - 7

Operação de atualização (CAS antes de RAS)

- o sinal *estroboscópico de acesso à coluna* é declarado baixo pelo *controlador de memória* antes do *estroboscópio de acesso à linha* e o sinal *de habilitação de gravação* é declarado alto, a lógica de controle dentro do chip interpreta esse arranjo como uma operação de atualização
- as entradas das linhas de endereço são ignoradas e o conteúdo do *contador de atualização* é armazenado no *buffer de endereço de linha* na borda descendente do *estroboscópio de acesso à linha*
- os valores de dados armazenados na linha selecionada de células de memória são então detectados e mantidos no *conjunto de amplificadores de sentido* para serem posteriormente gravados de volta nas células de memória

DRAM assíncrona - 8

A interface DRAM assíncrona foi modificada para melhorar o desempenho das operações de leitura e gravação na mesma linha de memória, evitando a necessidade de pré-carregamento e abertura repetida da linha para acesso a uma coluna diferente.

No *modo de página* DRAM, depois que uma linha é aberta mantendo *o estroboscópio de acesso à linha* baixo, a linha é mantida aberta e várias operações de leitura ou gravação são executadas em qualquer uma das colunas da linha. Cada acesso à coluna é iniciado afirmando *o estroboscópio de acesso à coluna* baixo e apresentando um endereço de coluna. Para operações de leitura, o sinal *de habilitação de gravação* também deve ser definido como alto. Para operações de gravação, o sinal *de habilitação de gravação* deve ser definido como baixo e os valores dos dados a serem gravados devem ser apresentados junto com o endereço da coluna.

A DRAM *de modo de página* foi posteriormente aprimorada com uma pequena modificação que reduziu ainda mais a latência, dando origem à chamada DRAM *de modo de página rápida ,* ou DRAM FPM.
Em uma DRAM *de modo de página , o strobe de acesso à coluna* é ativado após o endereço da coluna ser fornecido. Em uma DRAM FPM, o endereço da coluna pode ser fornecido enquanto *o estroboscópio de acesso à coluna* ainda está desativado. O endereço da coluna se propaga através do caminho de dados do endereço da coluna, mas não gera dados nos pinos de dados até que *o estroboscópio de acesso à coluna* seja ativado.

DRAM síncrona - 1

DRAM síncrona, ou SDRAM, muda radicalmente a forma como a interface da memória externa interage com o dispositivo. Os sinais de controle não têm mais efeito direto nas funções internas, mas um sinal de relógio controlado externamente é usado para gerenciar uma máquina de estados finitos integrada que atua sobre os comandos recebidos. Assim, os sinais RAS e CAS não atuam mais como flashes, mas passam a fazer parte do comando de entrada.

As operações internas são canalizadas para melhorar o desempenho. As operações iniciadas anteriormente são concluídas, enquanto novos comandos são recebidos e iniciam o processamento.

O conjunto bidimensional de células de memória é dividido em diversas seções de tamanhos iguais, mas independentes, chamadas de *bancos,* permitindo que o dispositivo opere um comando de acesso à memória, leitura ou gravação, em cada banco simultaneamente e agilize o acesso de forma intercalada.

O modo de transferência burst também é implementado para acessar múltiplas palavras de dados com o mesmo comando.

DRAM síncrona - 2

As gerações posteriores de SDRAMs melhoraram a largura de banda de transferência, permitindo que a transferência de dados ocorresse tanto na borda ascendente quanto na borda descendente do sinal de clock controlado externamente. Essa otimização é conhecida como *taxa de dados dupla* ou DDR.

DDR deu origem a uma sequência de padrões

- DDR, ou DDR1 – tem uma alimentação de tensão de 2,5 V e opera em frequências de clock de 133, 150 e 200 MHz
- DDR2 – tem uma alimentação de tensão de 1,8 V e opera em frequências de clock de 266, 333 e 400 MHz
- DDR3 – possui alimentação de tensão de 1,5 V e opera em frequências de clock de 533, 666 e 800 MHz
- DDR4 – possui alimentação de tensão de 1,2 V e opera em frequências de clock de 1066, 1333 e 1600 MHz.

DRAMs são geralmente vendidas em placas pequenas, chamadas *módulos de memória duplos em linha,* ou DIMMs, que contêm de 4 a 16 chips SDRAM e são normalmente organizadas para fornecer um comprimento de palavra de 64 bits de dados + bits *de código de correção de erros .*

DRAM síncrona - 3

SDRAM de 64 Mbits

(organizado em 4 bancos de 2.048 linhas x 1.024 colunas de células de memória de 8 bits cada)

CS RAS CAS NÓS BAn A10 An	Comando
Comando HXXXXXX inibido (o dispositivo não está selecionado)	
LHHHXXX sem operação	
LHHLXXX término em rajada: interrompe uma leitura em rajada ou uma gravação em rajada em andamento	
A coluna L do banco LHLH lê uma explosão de dados da linha ativa	
Coluna H do banco LHLH	leia uma explosão de dados da linha ativa atual com pré-carga (fechar a linha) quando terminar
A coluna L do banco LHLL grava uma explosão de dados da linha ativa atual	
Coluna H do banco LHLL	grava a explosão de dados da linha ativa atual com pré-carga (fechar a linha) quando
Banco LLHH linha	terminar, ativar (abrir) a linha para um comando de leitura ou gravação
LLHL banco LX pré-carregar ou desativar (fechar) a linha atual do banco selecionado	
Pré-carga LLHLXHX, ou desativar (fechar), a linha atual de todos os bancos atualiza uma linha de cada banco	
LLLHXXX	(atualização automática) usando um contador interno (todas as linhas devem ser pré-carregadas) registro
LLLL 00 modo	de modo de carregamento para configuração do chip: a palavra de controle está em A10-A0 as configurações mais significativas são para programar a latência CAS (2 ou 3 ciclos de clock) e comprimento de transferência burst (1, 2, 4 ou 8 palavras)

Gerações DRAM

Especificações de tempo

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

Produção ano	Tamanho do chip (pedaço)	DRAM tipo	Mais lento DRAM (ns)	O mais rápido DRAM (ns)	CAS/dados tempo de transferência (ns)	Ciclo tempo (ns)
1980	64K	DRAM	180	150	75	250
1983	256 mil	[illegible]	[illegible]	[illegible]	50	220
1986	1 milhão	DRAM	120	100	25	190
1989	4M	DRAM	[illegible]	80	20	165
1992	16 milhões	DRAM	80	60	15	120
1996	64 milhões	[illegible]	[illegible]	50	12	110
1998	128 milhões	SDRAM	70	50	10	100
2000	256 milhões	DDR1	65	45	7	90
2002	512 milhões	DDR1	60	40	5	80
2004	1G	DDR2	55	35		70
2006	2G	DDR2	50	30	5	60
2010	4G	DDR3	36	28	2,5 10	37
2012	8G	DDR3	30	24	0,5	31

Classificação DDR-SDRAM

Caracterização DIMM

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

Padrão	Taxa de clock (MHz)	M (transferência/s)	DRAM nome	MB/s/DIMM	DIMM nome
DDR	133	2[illegible]	[illegible]	2128	PC2100
DDR	150	3[illegible]0	DDR[illegible]00	2400	PC2400
DDR	200	4[illegible]0	DDR[illegible]400	3200	PC3200
DDR2	266	5[illegible]3	DDR[illegible]533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800
DDR4 1066-1600 2133-3200 DDR4-3200 17056-25600 PC25600					

Leitura sugerida

• *Arquitetura de Computadores: Uma Abordagem Quantitativa,* Hennessy JL, Patterson DA, 6ª Edição, Morgan Kaufmann, 2017 –
Apêndice B: *Revisão da Hierarquia de Memória* – Capítulo 2: *Projeto de Hierarquia de Memória* • *Organização e Arquitetura de Computadores: Projetando para Desempenho,* Stall 10ª Edição, Pearson Education, 2016
– Capítulo 4: *Memória Cache*
– Capítulo 5: *Memória Interna*
– Capítulo 6: *Memória Externa*
– *Capítulo* 8: *Suporte ao sistema operacional*