Fundamentos de Arquitetura de Computadores

Tiago Alves

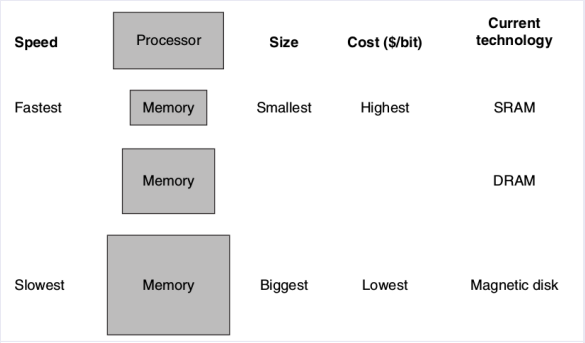
Faculdade UnB Gama
Universidade de Brasília

Visão Geral

- **Introdução**
- **Princípios Básicos da Cache**
- **Desempenho da Cache**
- Memória Virtual
- Hierarquia de Memória

Explorando a Hierarquia de Memória

Dados de 2012:



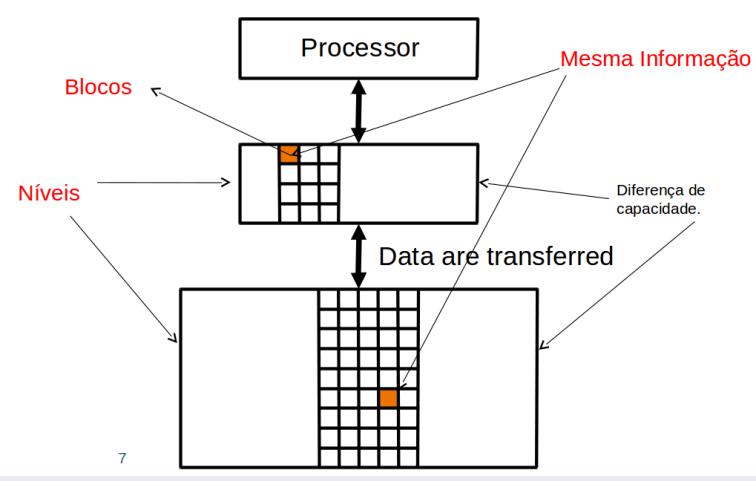
Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Evolução DRAM

Dados de 2012:

Year introduced	Chip size	$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 Kibibit	$1,500,000	250 ns	150 ns
1983	256 Kibibit	$500,000	185 ns	100 ns
1985	1 Mebibit	$200,000	135 ns	40 ns
1989	4 Mebibit	$50,000	110 ns	40 ns
1992	16 Mebibit	$15,000	90 ns	30 ns
1996	64 Mebibit	$10,000	60 ns	12 ns
1998	128 Mebibit	$4,000	60 ns	10 ns
2000	256 Mebibit	$1,000	55 ns	7 ns
2004	512 Mebibit	$250	50 ns	5 ns
2007	1 Gibibit	$50	45 ns	1.25 ns
2010	2 Gibibit	$30	40 ns	1 ns
2012	4 Gibibit	$1	35 ns	0.8 ns

Explorando a Hierarquia de Memória



Princípio da Localidade

Princípio arquitetural que justifica a utilização da hierarquia de memória.

Na ocorrência de uma **referência**:

- **localidade temporal**: o mesmo item tende a ser referenciado novamente em breve.
- **localidade espacial**: a mesma região (itens vizinhos) tendem a ser referenciada(os) novamente em breve.

Por que um código possui localidade?

- Consequência da simples e natural **estrutura** utilizada em algoritmos aplicados na solução de problemas.
  localidade temporal: loops. Instruções e dados com alta probabilidade de serem acessados repetidamente!
  localidade espacial: instruções são lidas sequencialmente (normalmente). Acessos a dados respeitam padrões sequenciais que implicam localidade espacial: arrays!

Princípio da Localidade

Foco inicial de estudos: dois níveis da hierarquia (superior e inferior).

Definições:

- **bloco**: unidade mínima de dados
- **acerto/hit**: os dados solicitados estão no nível superior.
- **falha/miss**: os dados solicitados não estão no nível superior.

Princípio da Localidade

Definições:

- **Taxa de Acertos**: É a fração dos acessos à memória encontrados no nível superior;
- **Taxa de Falhas**: É a fração dos acessos à memória não encontrados no nível superior;

  $$\text{Taxa de Falhas} = (1 - \text{Taxa de Acertos})$$

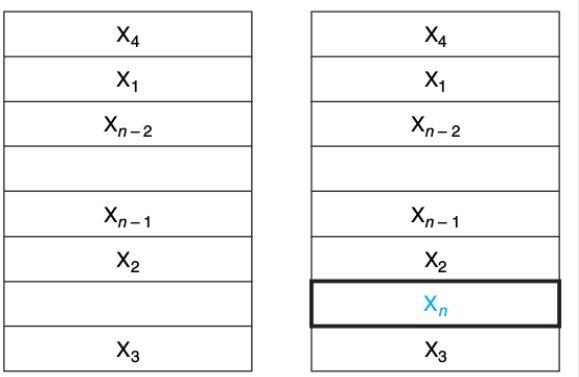
- **Tempo de Acerto**: Tempo necessário para acessar o nível superior da hierarquia de memória + o tempo para determinar o acerto (ou falha);
- **Penalidade de Falha**: É o tempo necessário para substituir o bloco do nível superior pelo bloco do nível inferior + o tempo de transferência para o processador.

Memória Cache

- Definição (dicionário): "Lugar seguro para esconder ou guardar coisas". (Fonte: Webster's New World Dictionary).
- Cache: denominação escolhida para representar o nível de hierarquia de memória entre o processador e a memória principal.
- O termo Cache tem sido utilizado para tirar proveito da localidade de acesso (Espacial e Temporal).
- Primeiros registros: aplicação inicial nos computadores de pesquisa da década de 1960.

Estratégia de estudos: Inicialmente trataremos a implementação de uma Cache simples: cada requisição do processador é uma word e os blocos também consistem em uma única word.

Memória Cache



a. Before the reference to X_n b. After the reference to X_n

Memória Cache

Dois problemas: (**Q:**)

- Como sabemos se um item de dados está na cache?
- Se estiver, como encontrá-lo?

Neste exemplo:

- O tamanho do bloco é uma word.
- Cache do tipo **diretamente mapeado**: para cada item de dados no nível inferior, existe exatamente um local onde ele pode estar.

  Logo, muitos itens no nível inferior compartilham locais no nível superior.

Memória Cache

Dois problemas: (**A:**)

- A cada word, atribui-se um endereço na memória: **mapeamento direto**.
- Aplicar uma Função de Mapeamento (do tipo função hash):
  - `posicao = (endereco_de_bloco) mod (numero_de_blocos_da_cache)`
  - Caso o número de entradas (posições) na cache seja uma potência de 2:

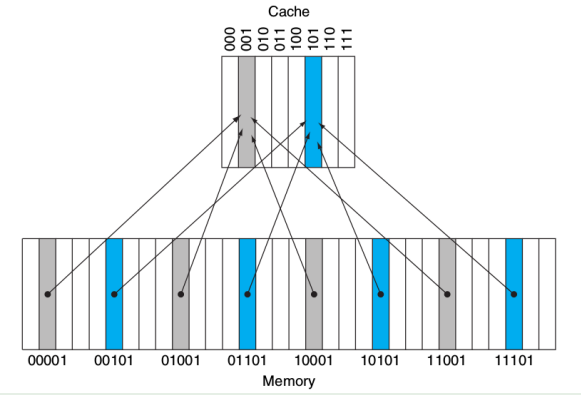
    Economiza-se na redução modular! Basta tomar os log_2(numero_de_blocos_da_cache) bits menos significativos do endereço.

Memória Cache

- Cache com capacidade para armazenar 8 words = `numero_de_blocos_da_cache`
- Endereço X é mapeado em posições da cache da seguinte forma: `End_X = X mod 8`.
- Mas 8 é potência de 2: 2^3. Logo $log_2 8 = 3$, a redução modular pode ser substituída por leitura dos 3 bits menos significativos do endereço.
- São esses bits que formarão o índice de endereçamento na Cache.

Memória Cache

Mapeamento direto para memória de 32 words e cache de 8 words.



Q: Como saber se os dados lidos da cache correspondem à word requisitada?

Memória Cache

- **Tags**: contém informações de endereços necessários para identificar se uma word na cache corresponde à word requisitada;

  Precisa somente da **parte superior do endereço** da word (2 bits MSB) dos 5 bits de endereço na Tag (ex. Anterior).

- **Bit de Validade**: indica se contém um endereço válido. Se NÃO estiver ativado, não pode haver correspondência para o respectivo bloco;

Acessando a Cache

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})

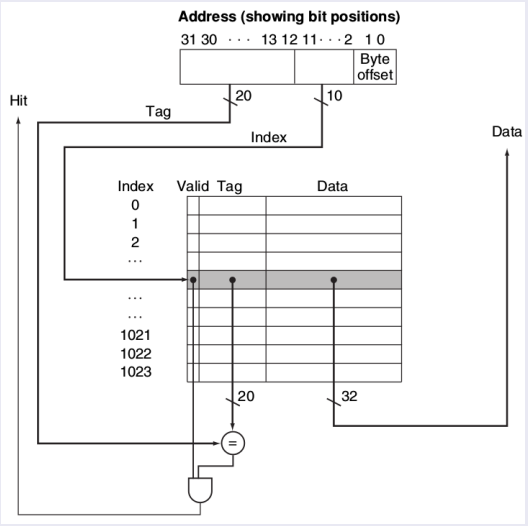
Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

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})

Cache com Mapeamento Direto: MIPS



Tamanho dos Campos da Cache

Na prática uma Cache tem várias words armazenadas:

- Considerando o endereço em bytes de 32 bits.
- Cache mapeada diretamente 2^n blocos com 2^m words, temos que:

  num_bits_cache $= 2^n \times$ (tamanho_bloco + tamanho_tag + tamanho_campo_validade)

Exemplo

Q: Quantos bits no total são necessários para uma cache em uma arquitetura MIPS diretamente mapeada com 16KB de dados e blocos de 4 words, considerando um endereço de 32 bits?

Exemplo

A:

- 16 KB = 4 Kwords = 2^{12} words com tamanho do bloco de 4 words (2^2) ou seja, 2^{10} blocos.
- Cada bloco possui $4 \times 32 = 128$ bits de dados mais uma tag que é $32 - 10 - 2 - 2$, mais um bit de validade, então:

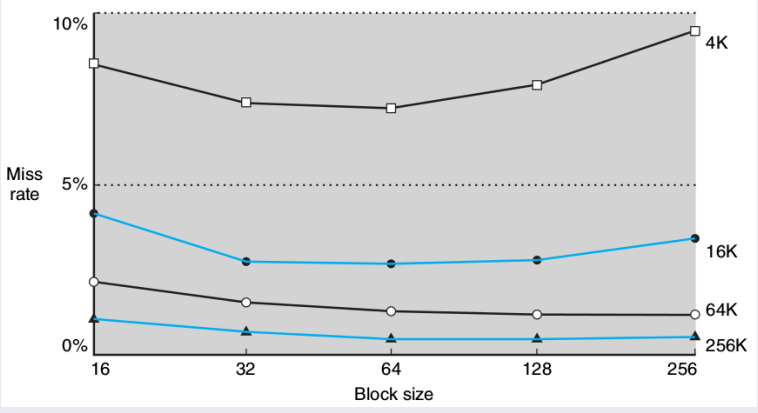
  tamanho_cache $= 2^{10} \times (128 + (32 - 10 - 2 - 2) + 1) =$ 147 Kbits

Acertos x Falhas

- Acertos de leitura: sempre desejáveis.
- Falhas de leitura: ocasiona um stall na CPU. Para saná-las, demanda-se que se busque o bloco da memória, que se distribua-o para o cache e que se reinicie e execução da instrução.
- Acertos de escrita:
  - **escrita direta**: podem substituir os dados no cache e na memória.
  - **escrita adiada na memória**: escrevem os dados apenas no cache.
- Falhas de escrita: lêem o bloco inteiro para o cache, depois escrevem a word.

Acertos x Falhas

Curva Taxa de Falhas vs. Tamanho do Bloco



Tratando Falhas na Cache

- Unidade de Controle precisa detectar a falha;
- Utilizamos a mesma estrutura de controle (Uniciclo, Multiciclo e Pipelining);

  Substituem-se Memórias pela Cache
- Cria-se uma unidade de controle exclusiva;
- Em caso de falha, gera-se um stall semelhante ao do pipeline e "congela" o conteúdo dos registradores temporários.

Tratando Falhas na Cache

1. Enviar PC original `PCatual - 4` para a memória;
2. Solicitar leitura da memória principal e esperar o término do acesso (vários ciclos!);
3. Escrever na entrada da Cache (atualizar tags e bit de validade);
4. Reiniciar a execução da instrução (1) e procurando a instrução na cache.

Tratando Escritas

- Risco de valores inconsistentes (valor da Cache $\neq$ memória principal);
- Usar o método write-through (escrever os dados tanto na Cache, quanto na memória principal).

  Processo lento (aprox. 100 ciclos!)
- Solução: Buffer de escrita (write buffer). Fila que contém os dados esperando para serem escritos na memória

  Problema: Escritas em rajadas (bursts)
- Solução Alternativa: Write-Back

  Quando ocorre uma escrita, o novo valor é somente escrito no bloco da Cache. Só é escrito na hierarquia inferior somente quando for substituído na Cache.
- E se ocorre um "Write-Miss"? Solução: Write-allocation!

Write-Through

- Durante um acerto de escrita o dado na cache pode ser atualizado.

  Mas os dados podem ser inconsistentes!
- Usando o Write through: atualiza-se também a memória.
- Processo demorado: se a CPI base igual a 1, 10% são tipo "store", o processo de escrever na memória leva em média 100 ciclos, então:

  CPI Efetiva $= 1 + 0.1 \times 100 = 11$

Write-Back

- Alternativa: Durante a escrita do dado, atualize somente o bloco na cache.

  Mantenha o controle de cada bloco escrito, evitando sobreposições
- Quando o bloco já escrito tiver de ser escrito novamente:
  - Escreva-o na memória;
  - Utilize um write buffer para permitir que o bloco seja primeiramente lido.

Alocação de Escrita (Write-Allocation)

O que aconteceria se ocorresse uma falha de escrita?

- Alternativa para o write-through
  - Alocação na falta: busque o bloco;
  - Escreva assim mesmo: não busque o bloco

    Desde que os programas frequentemente escrevem antes da leitura (ex.: inicialização)
- Para write-back: Usualmente, busque o bloco.

O Processador Intrinsity FastMATH

Características:

- Veloz
- Arquitetura MIPS
- 12 estágios de Pipeline
- Utiliza Cache de instruções separadas das Caches de dados
- Tamanho da Cache (16KB ou 4K Words) com blocos 16 words
- Possui write-through e write-back.

Taxa de Falhas do Intrinsity FastMATH

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

O Processador Intrinsity FastMATH

Arquitetura da Cache do Intrinsity FastMATH