Universidade Federal do Rio Grande do Sul
Instituto de Informática

Organização de Computadores

Aula 16

Memória cache
primeira parte

Memória cache
primeira parte

1. Tendências tecnológicas
2. Hierarquia de memória
3. Princípo de localidade
4. Impacto no desempenho
5. Organizações de memória cache

1. Tendências tecnológicas

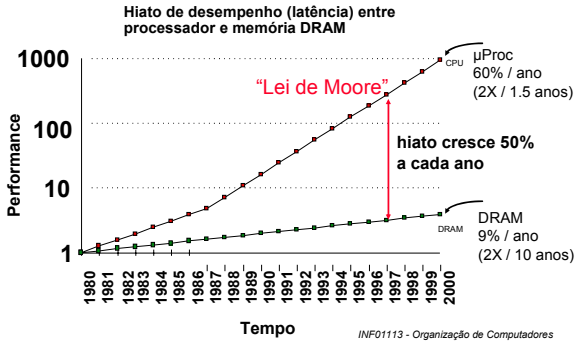
	Capacidade	Velocidade (latência)
Lógica:	2x em 3 anos	2x em 3 anos
DRAM:	4x em 3 anos	2x em 10 anos
Disco:	4x em 3 anos	2x em 10 anos

DRAM

Ano	Tamanho	Tempo acesso
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns

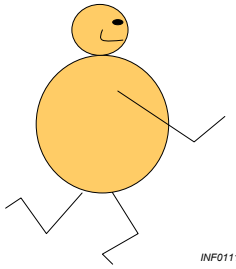
1000:1! 2:1!

Tendências tecnológicas



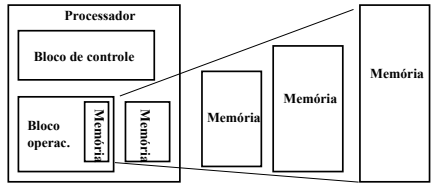
O que queremos?

- Um lutador de sumo que corre e ganha os 100m rasos!



2. Hierarquia de memória

- objetivo: oferecer ilusão de máximo tamanho de memória, com mínimo custo e máxima velocidade
- cada nível contém cópia de parte da informação armazenada no nível superior seguinte

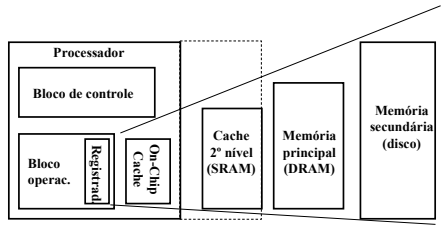


Velocidade:	Mais rápida	Mais lenta
Tamanho:	Menor	Maior
Custo:	Mais alto	Mais baixo

Tecnologias na hierarquia de memória

- Acesso randômico
  - tempo de acesso é o mesmo para todas as posições
  - DRAM: Dynamic Random Access Memory
    - alta densidade, baixa potência, barata, lenta
    - dinâmica: precisa de um "refresh" regular
  - SRAM: Static Random Access Memory
    - baixa densidade, alta potência, cara, rápida
    - estática: conteúdo dura "para sempre"(enquanto houver alimentação)
- Acesso "não-tão-randômico"
  - tempo de acesso varia de posição para posição e de tempos em tempos
  - exemplos: disco, CD-ROM
- Acesso sequencial
  - tempo de acesso varia linearmente com a posição (p.ex. fita)

Hierarquia de memória



Velocidade (ns):	0,1	1	2-5	10-20	10.000.000 (10 ms)
Tamanho (bytes):	100	16 K	512 K	256 M	Gs

Hit e *miss*

- Hit: dado aparece em algum bloco no nível superior (junto ao processador)
  - Hit Ratio: a fração de acessos à memória resolvidos no nível superior
  - Hit Time: tempo de acesso ao nível superior, que consiste de tempo de acesso à memória RAM + tempo para determinar hit/miss
- Miss: dado precisa ser buscado de um bloco no nível inferior
  - Miss Ratio = 1 – (Hit Ratio)
  - Miss Penalty: tempo gasto para substituir um bloco no nível superior + tempo para fornecer o bloco ao processador
- Hit Time << Miss Penalty

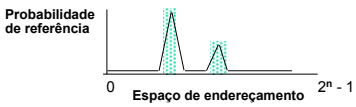
Hierarquia de memória

Como a hierarquia é gerenciada?

- Registradores <-> memória
  - pelo compilador
- cache <-> memória principal
  - pelo hardware
- memória principal <-> disco
  - pelo hardware e pelo sistema operacional (memória virtual)
  - pelo programador (arquivos)

3. Princípio de localidade

- Hierarquia de memória funciona devido ao <u>princípio de localidade</u>
  - todos os programas repetem trechos de código e acessam repetidamente dados próximos



- <u>localidade temporal</u>: posições de memória, uma vez acessadas, tendem a ser acessadas novamente no futuro próximo
- <u>localidade espacial</u>: endereços em próximos acessos tendem a ser próximos de endereços de acessos anteriores

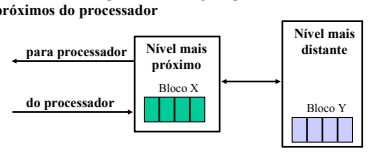
Exemplo:

- 1,2,3,4,5,6,7,6,7,6,7,6,7,492, 493,494,6,7,6,7...

Princípio de localidade

Como explorar o princípio de localidade numa hierarquia de memória?

- **Localidade Temporal**
  - **=> Mantenha itens de dados mais recentemente acessados nos níveis da hierarquia mais próximos do processador**
- **Localidade Espacial**
  - **=> Mova blocos de palavras contíguas para os níveis da hierarquia mais próximos do processador**



Localidade temporal

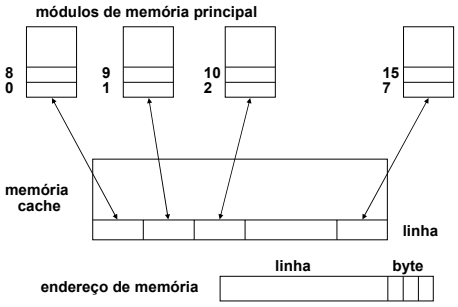
- **usualmente encontrada em laços de instruções e acessos a pilhas de dados e variáveis**
- **é essencial para a eficiência da memória cache**
- **se uma referência é repetida N vezes durante um laço de programa, após a primeira referência a posição é sempre encontrada na cache**

Tc = tempo de acesso à cache
Tm = tempo de acesso à memória principal
Tce = tempo efetivo de acesso à cache

$$Tce = \frac{N\,Tc + Tm}{N} = Tc + \frac{Tm}{N}$$

se Tc = 1 ns, Tm = 20 ns, N = 10 ⇨ Tce = 3 ns
N = 100 ⇨ Tce = 1,2 ns

Localidade espacial



Localidade espacial

- **memória principal é entrelaçada**
- **uma linha é transferida num único acesso entre a memória principal e a cache, através de um largo barramento de dados**
- **casamento entre tempo de acesso da cache e da memória principal**

M = número de módulos da memória principal

Tc = tempo de acesso à cache

Tm = tempo de acesso à memória principal

Ideal: M Tc = Tm

Localidade espacial

- **tempo médio de acesso a um byte, na primeira referência**
  = 2 M Tc / M = 2 Tc
- **se cada byte é referenciado N vezes na cache, então o tempo efetivo (médio) de acesso *Tce* a cada byte é**

$$Tce = \frac{2\,Tc + (N-1)\,Tc}{N} = \frac{(N+1)\,Tc}{N}$$

se **Tc = 1 ns**
Tm = 20 ns
N = 10 **N = 100**
⇩ ⇩
então **Tce = 1,1 ns** **Tce = 1,01 ns**

4. Impacto no desempenho

Medindo o impacto do *hit ratio* no tempo efetivo de acesso

Tc = tempo de acesso à memória cache
Tm = tempo de acesso à memória principal
Tce = tempo efetivo de acesso à memória cache, considerando efeito dos misses

Tce = Tc + (1 – h) Tm

se **Tc = 1 ns, Tm = 20 ns**

h =	**0.85**	**0.95**	**0.99**	**1.0**
	⇩	⇩	⇩	⇩
então Tce =	**4 ns**	**2 ns**	**1.2 ns**	**1 ns**

Impacto no desempenho

Tempo gasto com um *cache miss*, em número de instruções executadas

1º Alpha	340 ns / 5.0 ns =	68 clks x 2 instr.	ou	136 instruções
2º Alpha	266 ns / 3.3 ns =	80 clks x 4 instr.	ou	320 instruções
3º Alpha	180 ns / 1.7 ns =	108 clks x 6 instr.	ou	648 instruções

1/2 X latência x 3 X freqüência clock x 3 X instruções/clock ⇒ ≈ 5 X

Impacto no desempenho

- Supondo um processador que executa um programa com:
  - CPI = 1.1
  - 50% aritm/lógica, 30% load/store, 20% desvios
- Supondo que 10% das operações de acesso a dados na memória sejam *misses* e resultem numa penalidade de 50 ciclos

CPI = CPI ideal + nº médio de stalls por instrução
= 1.1 ciclos + 0.30 acessos à memória / instrução
x 0.10 misses / acesso x 50 ciclos / miss
= 1.1 ciclos + 1.5 ciclos
= 2. 6

CPI ideal	1.1
Data misses	1.5
Instr.misses	0.5

- 58 % do tempo o processador está parado esperando pela memória!
- um miss ratio de 1% no fetch de instruções resultaria na adição de 0.5 ciclos ao CPI médio

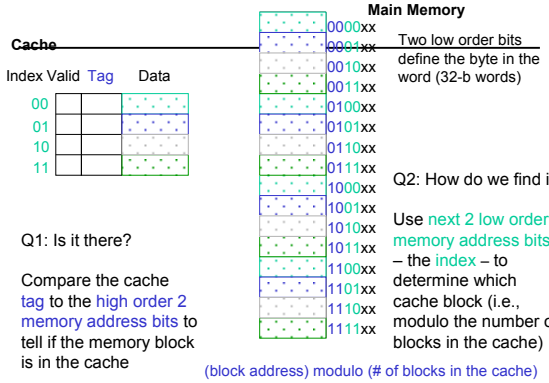
5. Organizações de memória cache

- processador gera endereço de memória e o envia à cache
- cache deve
  - verificar se tem cópia da posição de memória correspondente
  - se tem, encontrar a posição da cache onde está esta cópia
  - se não tem, trazer o conteúdo da memória principal e escolher posição da cache onde a cópia será armazenada
- *mapeamento* entre endereços de memória principal e endereços de cache resolve estas 3 questões
  - deve ser executado em hardware
- estratégias de organização (mapeamento) da cache
  - mapeamento completamente associativo
  - mapeamento direto
  - mapeamento set-associativo

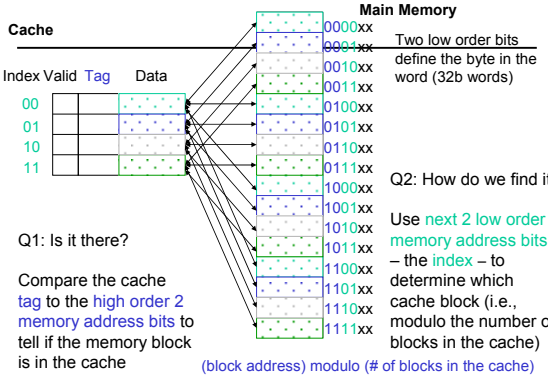
Cache

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?
- Direct mapped
  - For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level must share locations in the upper level
  - Address mapping:
    (block address) modulo (# of blocks in the cache)
  - First consider block sizes of one word

Caching: A Simple First Example



Caching: A Simple First Example



Direct Mapped Cache

- **Consider the main memory word reference string**

0 1 2 3 4 3 4 14

Start with an empty cache - all blocks initially marked as not valid

0	1	2	3

4	3	4	15

Direct Mapped Cache

- **Consider the main memory word reference string**

0 1 2 3 4 3 4 15

Start with an empty cache - all blocks initially marked as not valid

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

01 4

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

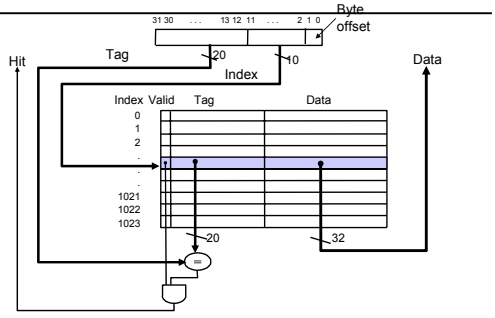
01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

11 15

- **8 requests, 6 misses**

MIPS Direct Mapped Cache Example

- **One word/block, cache size = 1K words**



What kind of locality are we taking advantage of?