Hierarquia de memória e a memória cache

MAC 344 - Arquitetura de Computadores
Prof. Siang Wun Song

Baseado parcialmente em W. Stallings -

Computer Organization and Architecture

Hierarquia de memória

Há vários tipos de memórias, cada uma com características distintas em relação a

- Custo (preço por bit).
- Capacidade para o armazenamento.
- Velocidade de acesso.

A memória ideal seria aquela que é barata, de grande capacidade e acesso rápido.

Infelizmente, a memória rápida é custosa e de pequena capacidade.

E a memória de grande capacidade, embora mais barata, apresenta baixa velocidade de acesso.

Hierarquia de memória

A memória de um computador é organizada em uma hierarquia.

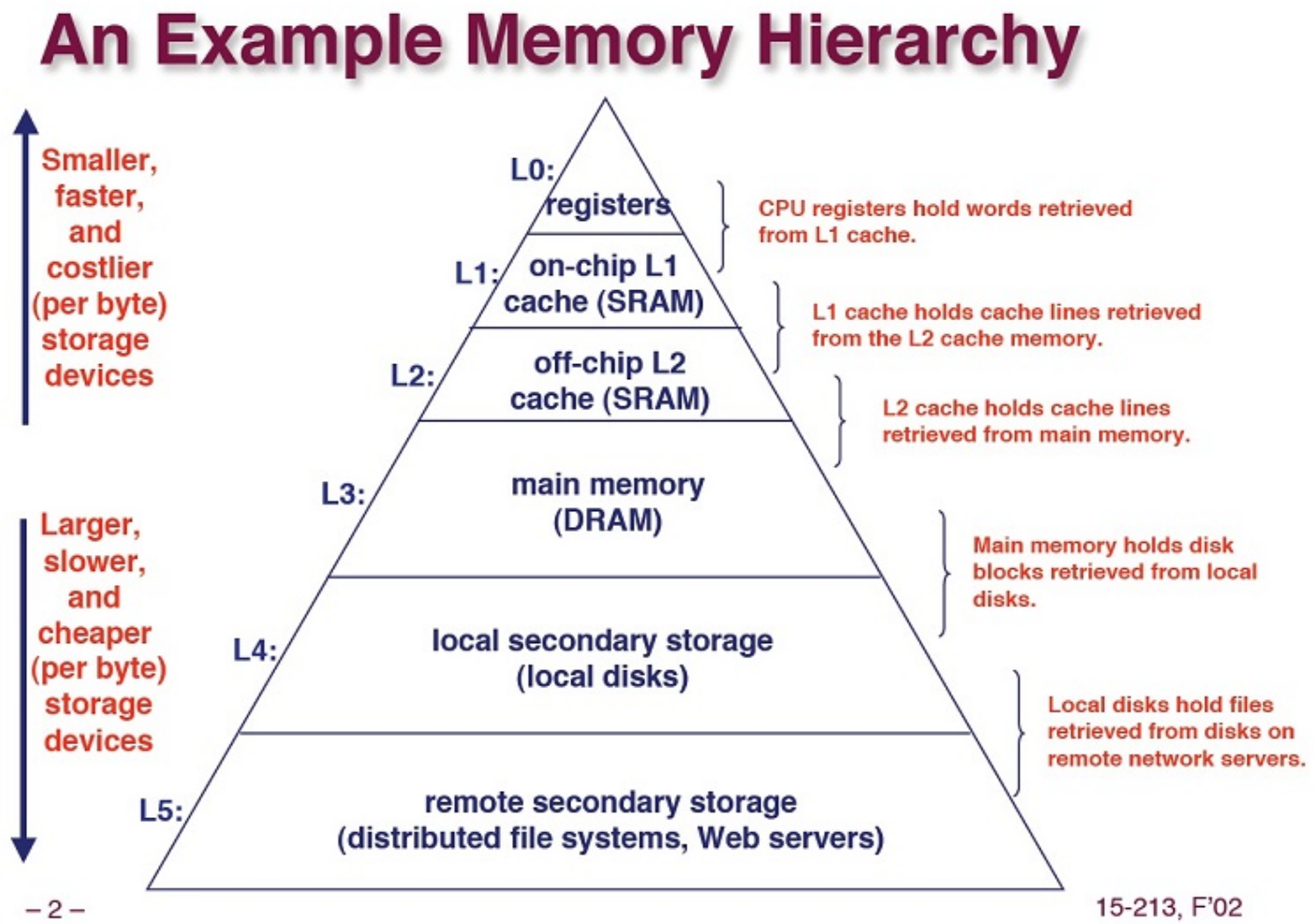
- Registradores: nível mais alto, são memórias rápidas dentro ao processador.
- Vários níveis de memória cache: L1, L2, etc.
- Memória principal (RAM).
- Memórias externas ou secundárias (discos, fitas).
- Outros armazenamento remotos (arquivos distribuidos, servidores web).

A seguir apresentamos o slide 2 de `http://web.cecs.pdx.edu/~jrb/cs201/lectures/cache.friendly.code.pdf`

Hierarquia de memória

slide 2 of `http:`

`//web.cecs.pdx.edu/~jrb/cs201/lectures/cache.friendly.code.pdf`



Hierarquia de memória

- Veremos hoje e nas próximas aulas: memória cache, memória interna e memória externa.
- No final desses estudos, iremos fazer uma pequena brincadeira: previsão do futuro do disco: Se o disco magnético HD vai ser substituído por SSD (*Solid State Drive*).

  Em 2005: *Samsung boss predicts death of hard drives*.

  `https://www.techworld.com/news/data/samsung-boss-predicts-death-of-hard-drives-4387/`
- Cada aluno(a), procura descobrir uma vantagem ou desvantagem de um dos dois tipos. Vamos enumerar em classe todas essas vantagens ou desvantagens.
- No final da discussão, vamos ver se chegamos a uma conclusão:
  - Se SSD vai derrubar completamente HD.
  - Caso positivo, em que ano isso irá ocorrer.
- Participação voluntária, quem não quiser, pode só assistir. Mas é mais divertido tomar algum partido nessa briga.

Hierarquia de memória

- Perigo de fazer previsões erradas:
  - "*I think there is a world market for maybe five computers.*" (Thomas Watson, Presidente da IBM, 1943.)
  - "*There is no reason anyone would want a computer in their home.*" (Ken Olsen, fundador da DEC, 1977.)
  - "*Apple is already dead.*" (Nathan Myhrvold, CTO Microsoft, 1997.)
  - "*Two years from now, spam will be solved.*" (Bill Gates, fundador da Microsoft, 2004.)

Hierarquia de memória

- Perigo de fazer previsões erradas:
  - "*I think there is a world market for maybe five computers.*" (Thomas Watson, Presidente da IBM, 1943.)
  - "*There is no reason anyone would want a computer in their home.*" (Ken Olsen, fundador da DEC, 1977.)
  - "*Apple is already dead.*" (Nathan Myhrvold, CTO Microsoft, 1997.)
  - "*Two years from now, spam will be solved.*" (Bill Gates, fundador da Microsoft, 2004.)

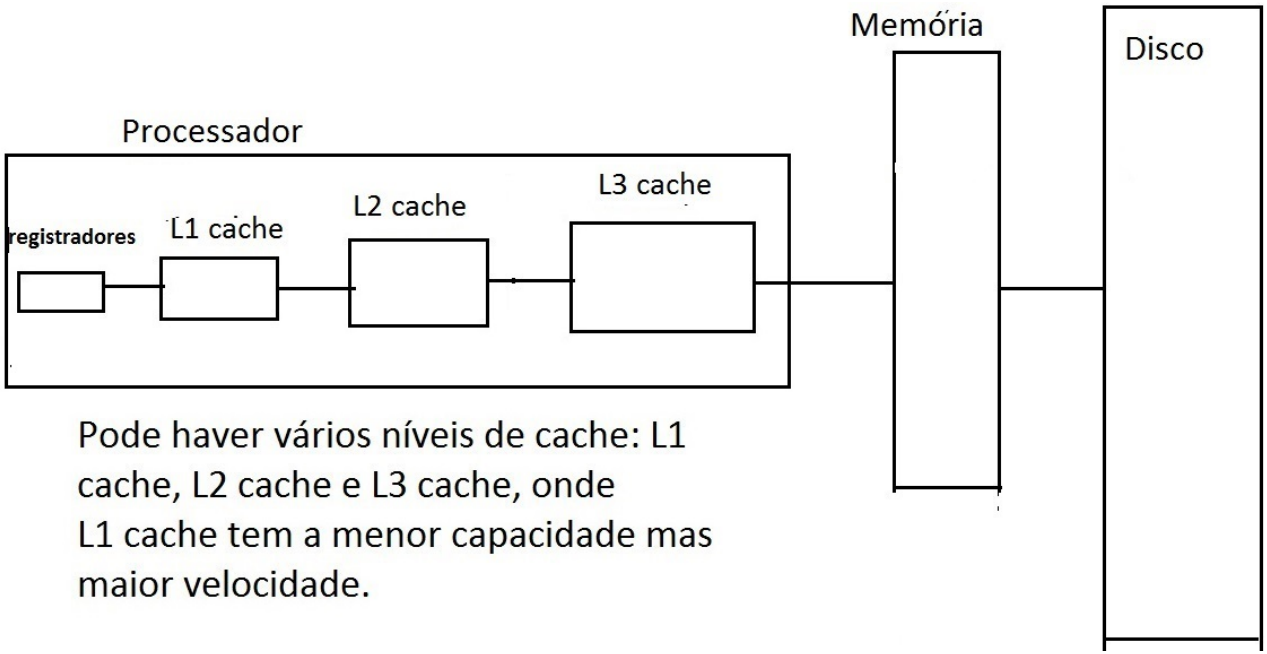
Hierarquia de memória

- Perigo de fazer previsões erradas:
  - "*I think there is a world market for maybe five computers.*" (Thomas Watson, Presidente da IBM, 1943.)
  - "*There is no reason anyone would want a computer in their home.*" (Ken Olsen, fundador da DEC, 1977.)
  - "*Apple is already dead.*" (Nathan Myhrvold, CTO Microsoft, 1997.)
  - "*Two years from now, spam will be solved.*" (Bill Gates, fundador da Microsoft, 2004.)

Hierarquia de memória

- Perigo de fazer previsões erradas:
  - "*I think there is a world market for maybe five computers.*" (Thomas Watson, Presidente da IBM, 1943.)
  - "*There is no reason anyone would want a computer in their home.*" (Ken Olsen, fundador da DEC, 1977.)
  - "*Apple is already dead.*" (Nathan Myhrvold, CTO Microsoft, 1997.)
  - "*Two years from now, spam will be solved.*" (Bill Gates, fundador da Microsoft, 2004.)

Memória cache



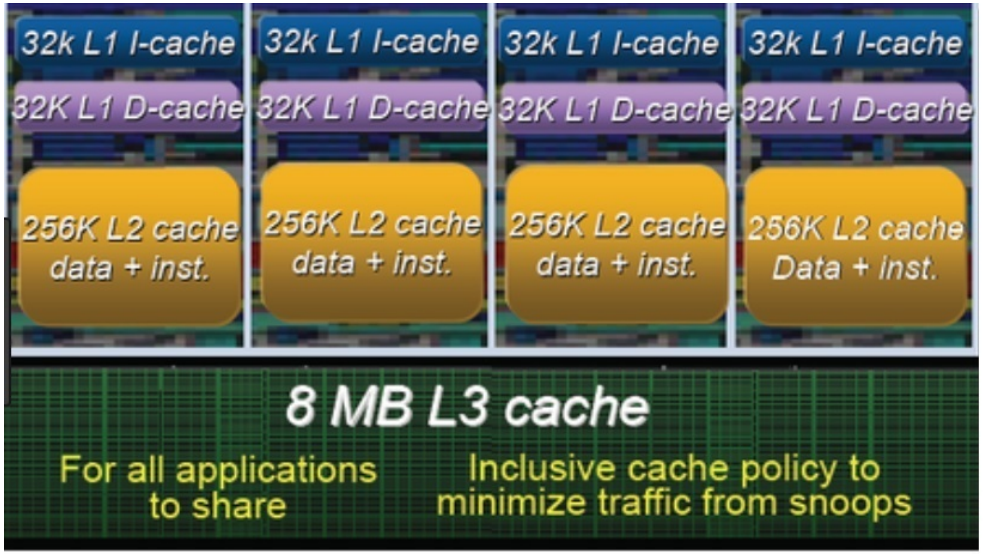
- Quando o processador precisa de um dado, ele pode já estar na cache (*cache hit*). Se não (*cache miss*), tem que buscar na memória (ou até no disco).
- Quando um dado é acessado na memória, um bloco inteiro (tipicamente 64 bytes) contendo o dado é trazido à memória cache. Blocos vizinhos podem também ser acessados (*prefetching*) para uso futuro.
- Na próxima vez o dado (ou algum dado vizinho) é usado, já está na cache cujo acesso é rápido.

Memória cache

Images source: Wikimedia Commons

- Analogia: se falta ovo (dado) na cozinha (processador), vai ao supermercado (memória) e compra uma dúzia (*prefetching*), deixando na geladeira (cache) para próximo uso.

Memória cache no Intel core i7



Intel core i7 cache (L3 cache também conhecida como LL ou Last Level cache)

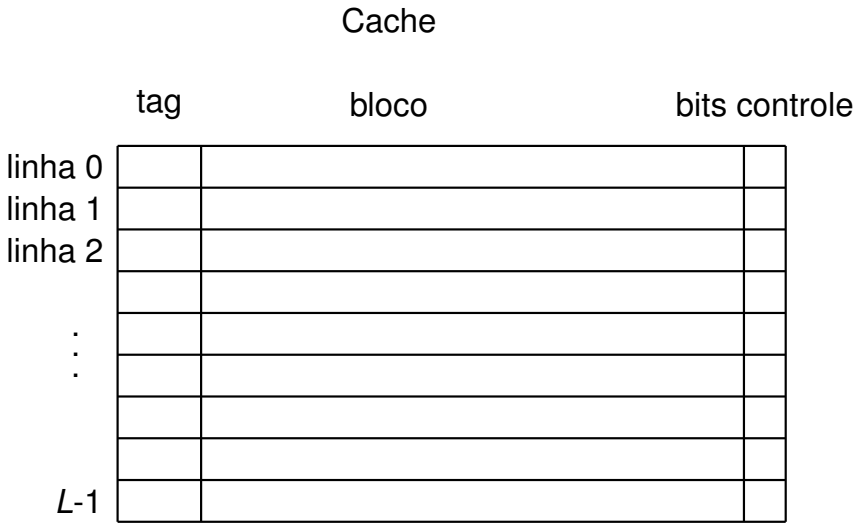
Hierarquia memória	Latência em ciclos
registrador	1
L1 cache	4
L2 cache	11
L3 cache	39
Memória RAM	107
Memória virtual (disco)	milhões

Memória cache

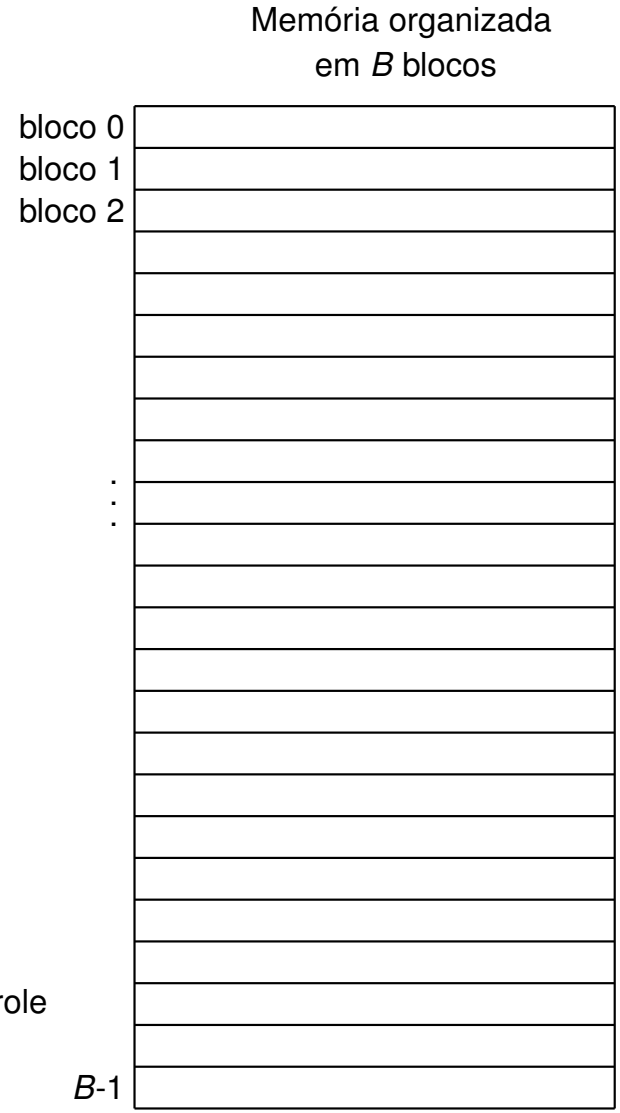
Evolução do uso da memória cache.

Processador	Ano fabr.	L1 cache	L2 cache	L3 cache
VAX-11/780	1978	16 KB	-	-
IBM 3090	1985	128 KB	-	-
Pentium	1993	8 KB	256 KB	-
Itanium	2001	16 KB	96 KB	4 MB
IBM Power6	2007	64 KB	4 MB	32 MB
IBM Power9 (24 cores)	2017	(32 KB I + 64 KB D) por core	512 KB por core	120 MB por chip

Memória cache



- Cache contém L linhas
- Cada linha contém um tag, um bloco de palavras da mem. e bits de controle
- tag contém info sobre o endereço do bloco na memória
- bits de controle informam se a linha foi modificada depois de carregada na cache



- Memória contém B blocos ($B >> L$)

Memória cache

- A L1 cache e L2 cache ficam dentro de cada *core* de um processador multicore e a L3 cache é compartilhada por todos os *cores*.
- A capacidade da cache é muito menor que a capacidade da memória. (Se a memória é virtual, parte dela fica no disco.)
- A memória cache contém apenas uma fração da memória.
- Quando o processador quer ler uma palavra da memória, primeiro verifica se o bloco que a contém está presente na cache.
- Se achar (conhecido como *cache hit*), então o dado é fornecido sem ter que acessar a memória.
- Se não achar (conhecido como *cache miss*), então o bloco da memória interessada é lido e colocada na cache.
- A razão entre o número de *cache hit* e o número total de buscas na cache é conhecida como *hit ratio*.

Memória cache

Fenômeno de localidade:

Quando um bloco de dados é guardado na memória cache a fim de satisfazer uma determinada referência a memória, é provável que haverá futuras referências à mesma palavra da memória ou a outras palavras do bloco.

Memória cache - Função de mapeamento

- Há menos linhas da memória cache do que número de blocos.
- É necessário definir como mapear blocos de memória a linhas da memória cache.
- A escolha da função de mapeamento determina como a cache é organizada.

Três funções de mapeamento:

- Mapeamento direto
- Mapeamento associativo
- Mapeamento associativo por conjunto

Memória cache - Mapeamento direto

É o mais simples: cada bloco da memória é mapeado, um a um, na sequência, a uma linha da cache, assim:
$i = j \mod L$ onde

- i = número da linha da cache
- j = número do bloco da memória
- L = número total de linhas na cache

Exemplo: Suponha uma memória de 16 blocos e uma cache de 4 linhas.
Então linha $i =$ bloco $j \mod 4$.

Bloco	Linha	Tag (parte vermelha)
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	0	0100
5	1	0101
6	2	0110
7	3	0111
8	0	1000
9	1	1001
10	2	1010
11	3	1011
12	0	1100
13	1	1101
14	2	1110
15	3	1111

Memória cache - Mapeamento direto

É o mais simples: cada bloco da memória é mapeado, um a um, na sequência, a uma linha da cache, assim:
$i = j \mod L$ onde

- i = número da linha da cache
- j = número do bloco da memória
- L = número total de linhas na cache

Suponha que a memória possui $B = 2^s$ blocos.
Suponha que a cache tem $L = 2^r$ linhas.

- Cada bloco da memória é mapeado diretamente a uma linha da cache.
- Bloco 0 é mapeado à linha 0 da cache, bloco 1 à linha 1, bloco $L - 1$ à última linha $L - 1$.
- Depois bloco L é mapeado à linha 0 da cache, bloco L+1 à linha 1, e assim sucessivamente.
- Quando um bloco de memória é introduzido numa linha da cache, o tag deve conter informação sobre esse bloco.

Memória cache - Mapeamento direto

Mostramos agora que basta ter um tag de $s - r$ bits para identificar qual bloco está mapeado a uma determinada linha.

- Suponha $s = 5$, i.e. uma memória de $B = 2^5 = 32$ blocos.
- Suponha $r = 2$, i.e. uma cache de $L = 2^2 = 4$ linhas.

Linha	Bloco mapeado à linha
0	00000 00100 01000 01100 10000 10100 11000 11100
1	00001 00101 01001 01101 10001 10101 11001 11101
2	00010 00110 01010 01110 10010 10110 11010 11110
3	00011 00111 01011 01111 10011 10111 11011 11111

Assim, somente $s - r = 5 - 2 = 3$ bits (os bits vermelhos) são necessários no tag para identificar qual bloco está mapeado.

Memória cache - Mapeamento direto exemplo

Suponha uma memória com 32 blocos e uma memória cache com 4 linhas, i.e.

- Seja $s = 5$ e uma memória de $B = 2^5 = 32$ blocos.
- Seja $r = 2$ e uma cache de $L = 2^2 = 4$ linhas.
- O tag terá $s - r = 5 - 2 = 3$ bits.

- Desenhe uma memória cache com 4 linhas e uma memória com 32 blocos.
- Acesse o bloco 6 da memória (vai dar cache miss, então introduza na cache).
- Acesse o bloco 8 da memória (vai dar cache miss, então introduza na cache).
- Acesse o bloco 6 da memória (vai dar cache hit).
- Acesse o bloco 4 da memória (vai dar cache miss, então introduza na cache expulsando o que estava lá).

Memória cache - Mapeamento direto

Apesar de simples, o mapeamento direto apresenta uma desvantagem.

- Cada bloco de memória é mapeado a uma posição fixa na cache.
- Se o programa faz referência repetidamente a palavras que estão em dois blocos de memória que são mapeados à mesma posição na cache, então esses blocos serão continuamente introduzidos e retirados da cache.
- O fenômeno acima tem o nome de *thrashing*, resultando em um número grande de *hit miss* ou baixa *hit ratio*.
- Para aliviar o problema de trashing: uma maneira é anotar quais blocos são retirados e em seguida necessários de novo. Cria-se uma memória cache chamada *cache vítima*, tipicamente de 4 a 16 linhas, onde são colocados tais blocos.

Memória cache - Mapeamento associativo

- No mampeamento associativo, cada bloco de memória pode ser carregado em qualquer linha da cache.
- Suponha a memória com $B = 2^s$ blocos.
- O campo tag de uma linha, de s bits, informa qual bloco está carregado na linha.
- Para determinar se um bloco já está na cache, os campos tag de todas as linhas são examinados simultaneamente. O acesso é dito associativo.
- A desvantagem do mapeamento associativo é a complexidade da circuitaria para possibiltar a comparação de todos os tags em paralelo.

Memória cache - Mapeamento associativo

Algorimos de substituição.

- Quando um novo bloco precisa ser carregado na cache, que já está cheia, a escolha da qual linha deve receber o novo bloco é determinada por um algoritmo de substituição.
- Diversos algoritmos de substituição têm sido propostos na literatura, e.g. LRU (*least recently used*) - a linha que foi menos usada é escolhida para ser substituída. Estuderemos esse assunto logo a seguir, nos próximos slides.

Memória cache - Mapeamento associativo por conjunto

É um compromisso entre o mapeamento direto e associativo, unindo as vantagens de ambos e reduz as desvantagens.

- A cache consiste de um número v de conjuntos, cada um com L linhas.
- Suponha i = número do conjunto.
- Suponha j = número do bloco na memória.
- Temos $i = j \mod v$. I.e. o bloco j é colocado no conjunto número i.
- Dentro do conjunto i, bloco j pode ser introduzido em qualquer linha. Usa-se acesso associativo em cada conjunto para verificar se um dado bloco está ou não presente.

Memória cache - Mapeamento associativo por conjunto

- A organização mais comum é usar 2 linhas em cada conjunto: $L = 2$. Resultados de simulação mostram que essa organização melhora significativamente o *hit ratio*.
- Essa organização recebe o nome de *Two-way set-associative cache*.

Algoritmos de substituição na cache

Quando a cache já está cheia e um novo bloco precisa ser carregado na cache, então um dos blocos ali existentes deve ser substituído, cedendo o seu lugar ao novo bloco.

- No caso de mapeamento direto, há apenas uma possível linha para receber o novo bloco. Não há portanto escolha.
- No caso de mapeamento associativo e associativo por conjunto, usa-se um algoritmo de substituição para fazer a escolha. Para maior velocidade, tal algoritmo é implementado em hardware.
- Dentre os vários algoritmos de substituição, o mais efetivo é o LRU (*Least Recently Used*): substituir aquele bloco que está na cache há mais tempo sem ser referenciado.

  > *Esse esquema é análogo a substituição de mercadoria na prateleira de um supermercado. Suponha que todas as prateleiras estão cheias e um novo produto precisa ser exibido. Escolhe-se para substituição aquele produto com mais poeira (pois foi pouco acessado.)*

- Há outros algoritmos de substituição como FIFO (*First In First Out*), *second chance*, aleatório, etc.

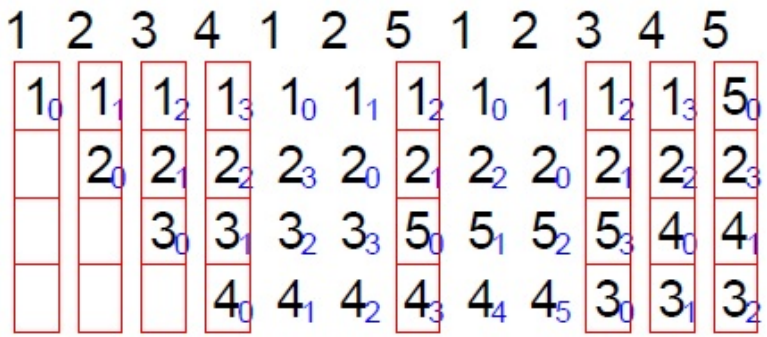
Algoritmo de substituição LRU

O algoritmo de substituição LRU substitui aquele bloco que está na cache há mais tempo sem ser referenciado.

- Numa cache associativa, é mantida uma lista de índices a todas as linhas na cache. Quando uma linha é referenciada, ela move à frente da lista.
- Para acomodar um novo bloco, a linha no final da lista é substituída.
- O algoritmo LRU tem-se mostrado eficaz com um bom *hit ratio*.

Source: Nesse link tem slides interessantes sobre algoritmos de substituição. Suponha cache com capacidade para 4 linhas. O índice pequeno denota o "tempo de permanência" na cache.

`https://www.cs.utah.edu/~mflatt/past-courses/cs5460/lecture10.pdf`



Algoritmo de substituição pseudo-LRU

Há uma maneira mais rápida de implementar um pseudo-LRU.

- Em cada linha da cache, mantém-se um *Use bit*.
- Quando um bloco de memória é carregado numa linha da cache, o *Use bit* é inicializado como zero.
- Quando a linha é referenciada, o *Use bit* muda para 1.
- Quando um novo bloco precisa ser carregado na cache, escolhe-se aquela linha cujo *Use bit* é zero.
- Esse esquema funciona especialmente bem em cache associativo por conjunto onde cada conjunto tem 2 linhas (*two-way set-associative cache*).

Cache write through e write back

Quando uma linha na cache vai ser substituída, dois casos devem ser considerados.

- Se a linha da cache não foi alterada (escrito nela), então essa linha pode ser simplesmente substituída.
- Se houve uma operação de escrita na linha da cache, então o bloco correspondente a essa linha na memória principal deve ser atualizado.

Escritas numa linha na cache pode gerar uma inconsistência na memória onde o bloco correspondente possui dados diferentes. Para resolver isso, há duas técnicas:

- *Write through*: Toda escrita à cache é também feita à memória.
- *Write back*: Escritas são somente feitas na cache. Mas toda vez que isso ocorre, liga-se um *dirty bit* indicando que a linha da cache foi alterada. Antes que a linha da cache é substituída, se *dirty bit* está ligado, então o bloco da memória correspondente é atualizado, mantendo-se assim a consistência.

Como foi o meu aprendizado?

Indique a(s) resposta(s) errada(s).

1. Com o avanço dos vários tipos de memórias, vai desaparecer por completo a hierarquia de memória.
2. A vantagem do mapeamento direto é a sua simplicidade.
3. Outra vantagem do mapeamento direto é o fenômeno de *thrasing*.
4. No mampeamento associativo, a linha que contém o bloco desejado é obtida rapidamente pois todos os tags são comparados em paralelo.
5. Outra vantagenm do mapeamento associativo é a simplicidade na implementação da circuitaria para o acesso associativo.
6. O mampeamento associativo por conjunto procura reunir as vantagens do mapeamento direto e o mapeamento associativo.
7. A organização mais usada no mapeamento associativo por conjunto é ter duas linhas em cada conjunto.
8. LRU é o algoritmo de substituição de linhas de cache efetivo e bastante usado.
9. Para escolher a linha ótima para ser substituída, é comum executar-se um algoritmo de substituição ótimo em que todas as possíveis linhas são testadas como objeto de substituição.
10. Write-through é uma técnica mais conservadora para manter a consistência. Mas write-back minimiza a escrita na memória e ainda mantém a consistência embora tardiamente.