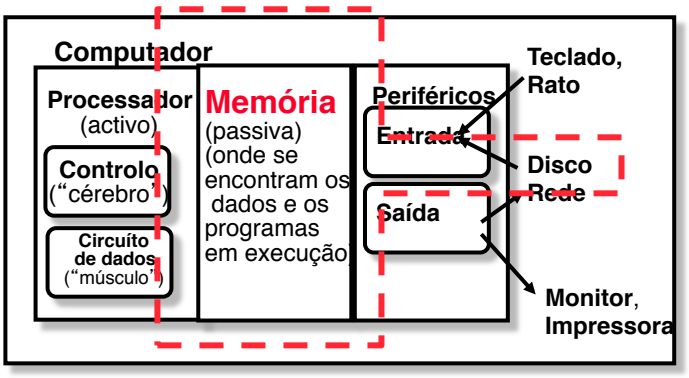
Arquitectura de Computadores

"Caches"

Docente: Pedro Sobral
http://www.ufp.pt/~pmsobral

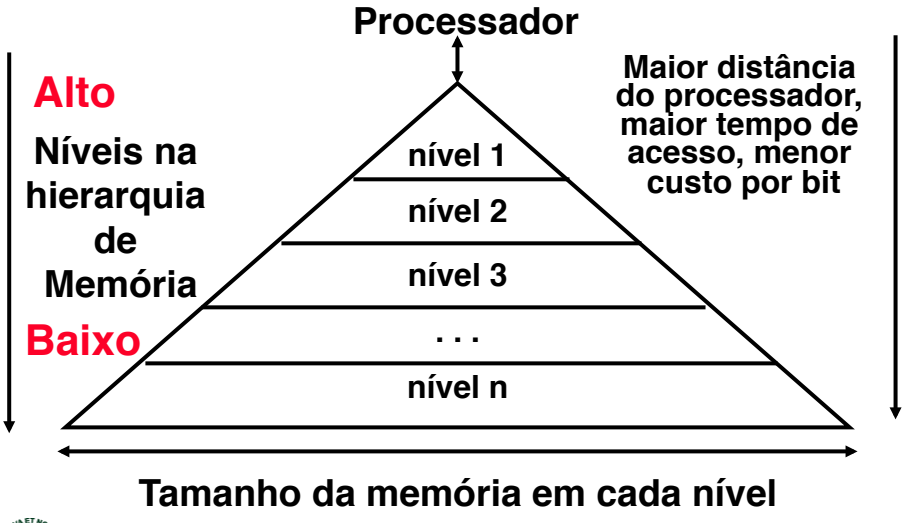
O computador



Hierarquia de memória (1/3)

- Processador
  - Executa instruções na ordem dos nanosegundos aos picosegundos
  - Guarda uma quantidade muito pequena de código e dados nos seus registos
- Memória
  - Com mais capacidade do que os registos, mas também limitada
  - Tempo de acesso na ordem dos ~50-100ns
- Disco
  - GRANDE capacidade (virtualmente ilimitada)
  - Muito lento: ~milisegundos

Hierarquia de memória (2/3)



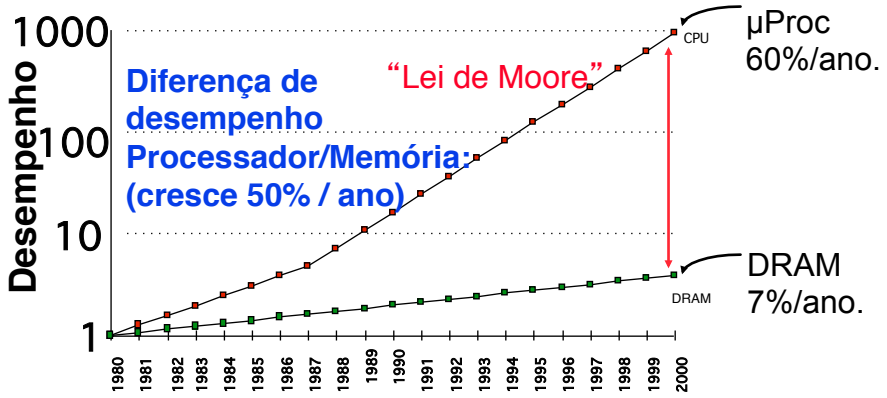
Hierarquia de memória (3/3)

- Cada nível que se encontra mais próximo do processador, tem que:
  - Ser de menor tamanho
  - Ser mais rápido
  - Conter um subconjunto dos dados do nível inferior (os dados usados recentemente)
- O nível inferior (geralmente o disco rígido) contém todos os dados disponíveis
- E os outros níveis?

Fazer "Cache" da memória

- Apresentamos três níveis na hierarquia de memória: processador, memoria, disco rígido.
- As diferenças nas velocidades de funcionamento entre o processador e a memória levaram à adição de um novo nível: a memória cache
- Implementada com tecnologia SRAM: muito rápida mas mais cara que a memória DRAM usada na memória principal.
  - "S" = Static, não necessita de refrescamento, ~10ns
  - "D" = Dynamic, necessita de refrescamento, ~60ns

Porquê usar Caches?



- 1989 Primeiro CPU da Intel com cache "on chip"
- 1998 Pentium III tem dois níveis de cache "on chip"

Analogia da hierarquia de memória: Biblioteca (1/2)

- Está a escrever um relatório (Processador) numa mesa da Biblioteca
- Biblioteca é equivalente ao disco rígido
  - Capacidade essencialmente ilimitada
  - Muito tempo necessário para obter um livro
- Mesa é a memória
  - Menor capacidade: significa que é necessário devolver um livro quando ela fica sem espaço
  - Mais simples e rápido encontrar um livro na mesa uma vez que já foi requisitado

Analogia da hierarquia de memória: Biblioteca (2/2)

- Os livros abertos na mesa são a cache
  - Menor capacidade: são poucos os livros que podemos ter abertos na mesa; quando não há espaço, é necessário fechar um livro.
  - Muito,muito mais rápido de obter a informação
- Ilusão criada: TODOS os livros da biblioteca abertos na mesa
  - Manter o maior número possível de livros usados recentemente abertos na mesa uma vez que é provável que voltem a ser necessários.
  - Manter o maior número possível de livros na mesa, uma vez que é muito mais rápido do que obtê-los da prateleira da biblioteca.

Bases da hierarquia de memória

- O disco rígido contém toda a informação.
- Quando o processador necessita de informação ela é copiada para níveis de memória superiores.
- A "cache" contém copias de dados em memória que estão a ser usados.
- A memória contém copias de dados do disco que estão a ser usados.
- Esta organização tem por base o Acesso Localizado : se estamos a usar uma certa informação agora, então provavelmente vamos voltar a usá-la brevemente

Desenho da "cache"

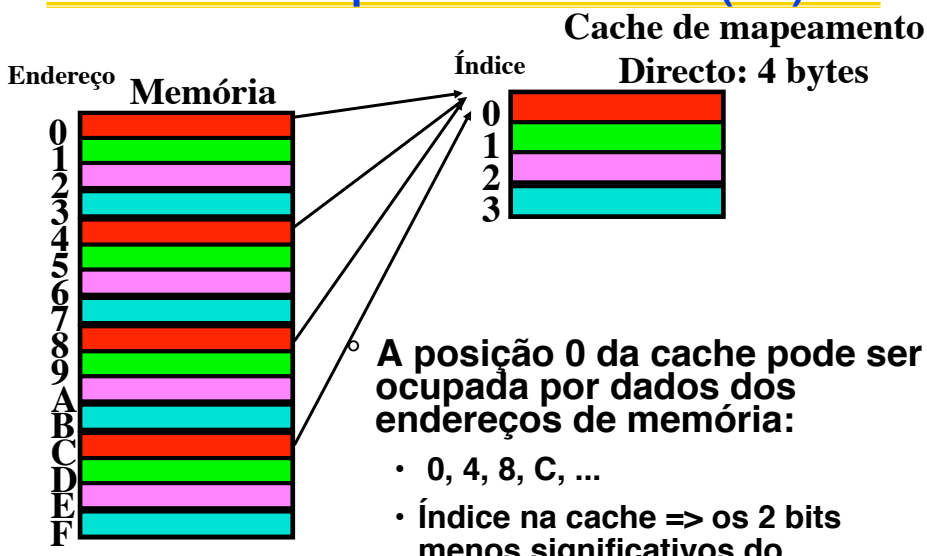
- Como se organiza a "cache"?
- Como mapear todos os endereços de memória na "cache"?

  (Recorde que a "cache" é um subconjunto da memória príncipal, portanto múltiplos endereços de memória são mapeados na mesma localização na "cache"…)
- Com sabemos que elementos da memória se encontram na cache?
- Como localizá-los rapidamente?

"Cache" de mapeamento directo (1/4)

- Numa cache de mapeamento directo, cada endereço de memória está associado com um possível bloco da cache
  - Portanto, basta verificar numa única posição da cache se lá se encontram os dados necessários
  - O bloco é a unidade de transferência entre a cache e a memória principal
- Há outros tipos de cache com desempenho superior ao mapeamento directo (caches associativas) mas por questões de tempo não as vamos estudar.

"Cache" de mapeamento directo (2/4)



- A posição 0 da cache pode ser ocupada por dados dos endereços de memória:
  - 0, 4, 8, C, ...
  - Índice na cache => os 2 bits menos significativos do endereço (*Cache Index*)

"Cache" de mapeamento directo (3/4)

- Uma vez que múltiplos endereços de memória são mapeados no mesmo índice da cache como saber qual está guardado?
- E se o tamanho do bloco é > 1 byte?
- Resposta: dividir o endereço de memória em 3 campos

tttttttttttttttt	iiiiiiiiii	dddd
Etiqueta "tag" para verificar se temos o bloco correcto	índice para seleccionar o bloco	desvio bytes no bloco

"Cache" de mapeamento directo (4/4)

- Todos os campos são inteiros sem sinal.
- Índice: especifica o índice da cache (qual a "linha" ou bloco da cache que devemos usar)
- Desvio: uma vez encontrado o bloco correcto, especifica qual o byte pretendido nesse bloco
- Etiqueta: os restantes bits depois do desvio e do índice são usados para distinguir entre todos os endereços que podem ser mapeados na mesma posição da cache

Terminologia das "caches"

- Quando tentamos ler a memória 3 coisas podem acontecer:

1. Sucesso na cache:
   o bloco da cache é válido e contém o endereço solicitado, portanto a informação é lida da cache
2. Falta na cache:
   O bloco indicado no endereço está vazio, portanto temos que o ler da memória principal
3. Falta na cache com substituição de bloco:
   No bloco indicado não se encontra o endereço solicitado portanto, o bloco existente é descartado e o bloco apropriado é copiado da memória principal

Cache de mapeamento directo: exemplo (1/3)

- Suponha que temos 16Kbytes numa cache de mapeamento directo com blocos de 4 palavras
- Qual será o tamanho da etiqueta, índice e deslocamento numa arquitectura de 32 bit?
- Deslocamento
  - Necessita de especificar o byte correcto dentro de um bloco.
  - Cada bloco contém 4 palavras de 32 bit

    = 16 bytes

    = 2^4 bytes
  - São necessários 4 bits para especificar o byte correcto

Cache de mapeamento directo: exemplo (2/3)

- Índice: (~índice num "vector de blocos")
  - Necessita de indicar a linha correcta na cache
  - cache contém 16 KB = $(2^4 * 2^{10}) = 2^{14}$ bytes
  - bloco contém 2^4 bytes (4 palavras)
  - # blocos/cache

    $$= \frac{\text{bytes/cache}}{\text{bytes/bloco}}$$

    $$= \frac{2^{14}\ \text{bytes/cache}}{2^4\ \text{bytes/bloco}}$$

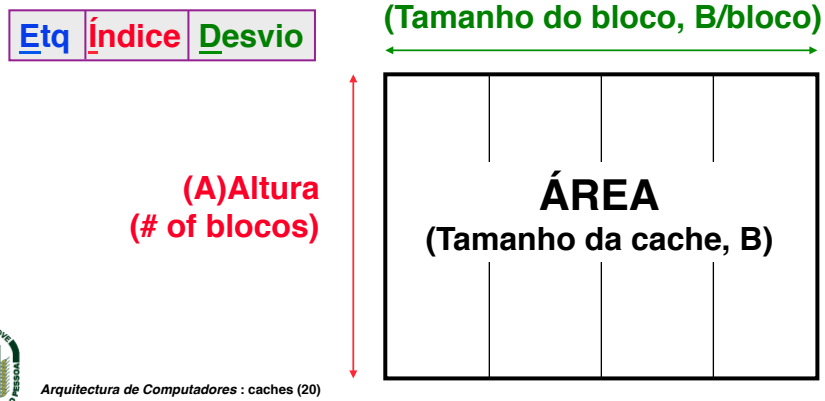
    $$= 2^{10}\ \text{blocos/cache}$$
  - São necessários 10 bits para especificar os blocos (linhas) da cache

Cache de mapeamento directo: exemplo (3/3)

- Etiqueta: usar os restantes bits como etiqueta
  - tam. etiqueta = tam. endereço - desvio - índice
    = 32 - 4 - 10 bits
    = 18 bits
  - Portanto a etiqueta é composta pelos 18 bits mais à esquerda do endereço de memória
- Porque não usar os 32 bits como etiqueta?
  - Todos os bytes num bloco necessitam de ter a mesma etiqueta.
  - O índice tem que ser o mesmo para todos os endereços de um bloco, portanto é redundante na verificação da etiqueta. Sendo assim, fica de fora para poupar memória (10 bits neste exemplo).

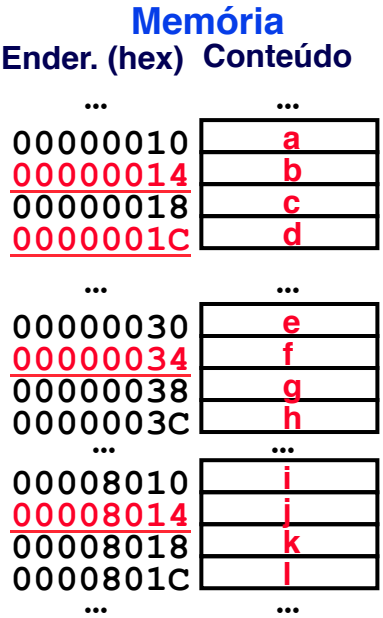
EID – Entender o tamanho da cache

ÁREA (tamanho da cache B) $2^{(A+L)} = 2^A * 2^L$
= Altura (# de blocos)
* Largura (tamanho de um bloco, B/bloco)



Exemplo: Acesso à cache de map. directo

- Ex.: 16KB de tamanho, map. directo, Blocos de 4 palavras
- Ler 4 endereços:
  1. 0x00000014
  2. 0x0000001C
  3. 0x00000034
  4. 0x00008014
- Valores em memória à direita:
  - Apenas se considera a existência da cache e memória na hierarquia



Exemplo: Acesso à cache de map. directo

- 4 Endereços:
  - 0x00000014, 0x0000001C, 0x00000034, 0x00008014
- 4 Endereços divididos (por conveniência) nos campos Etiqueta, Índice, Desvio

```
00000000000000000 0000000001 0100
00000000000000000 0000000001 1100
00000000000000000 0000000011 0100
00000000000000010 0000000001 0100
```

Etiqueta Índice Desvio

Cache de 16 Kbytes com blocos de 16bytes

- **Bit de validação:** determina se estão dados guardados na linha (quando o computador é inicializado todas as entradas estão a 0)

Índice	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

1. Ler 0x00000014

- 00000000000000000 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Vamos ler o bloco 1 (0000000001)

° 00000000000000000 0000000001 0100

Etq. Índice Desvio

Ind.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Não contém dados válidos!

° 00000000000000000 0000000001 0100

Etq. Índice Desvio

Ind.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Ler o bloco e actualizar a cache

° 00000000000000000 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Ler da cache no desvio, retorna *b*

° 00000000000000000 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

2. Ler 0x0000001C = 0…00 0..001 1100

° 00000000000000000 0000000001 1100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

O Índice é válido

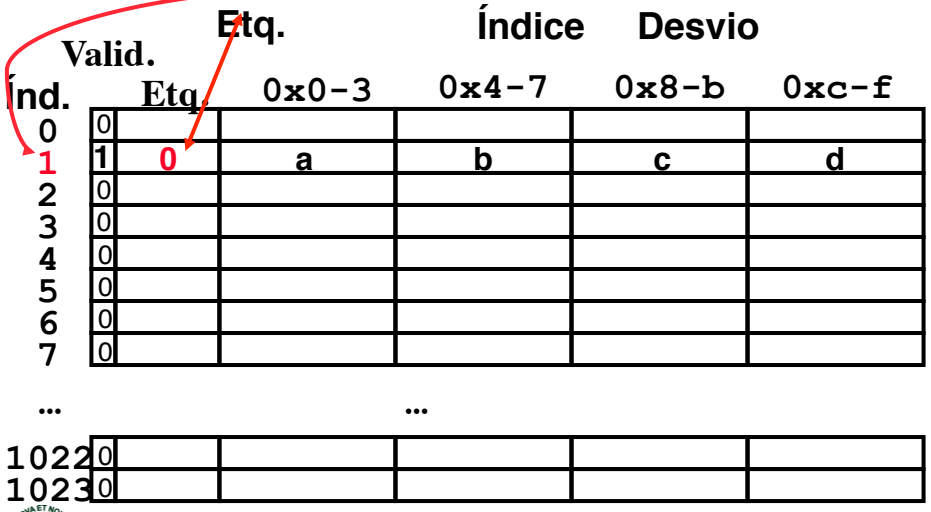
° 00000000000000000 0000000001 1100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

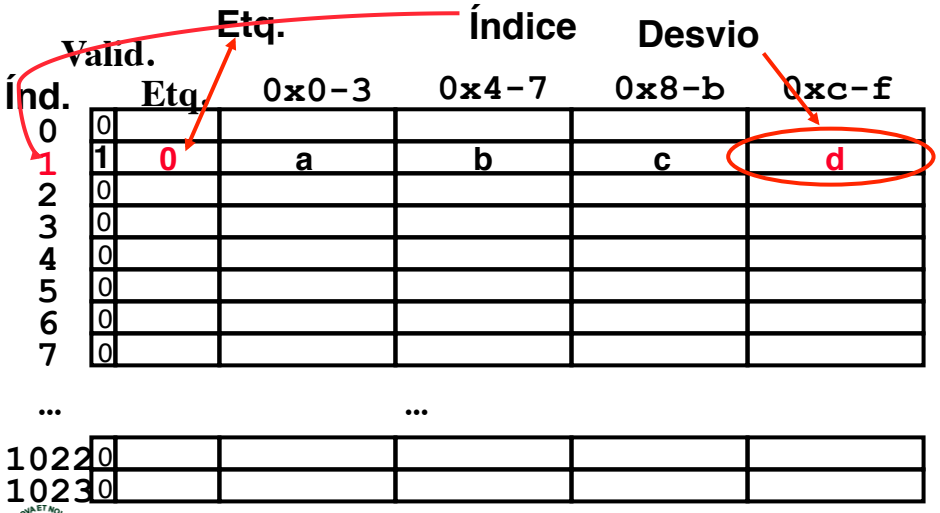
O índice é válido e a etiqueta é igual!

° 00000000000000000000 0000000001 1100



O índice é válido,a etiqueta é igual, retorna *d*

° 00000000000000000000 0000000001 1100



3. Ler 0x00000034 = 0…00 0..011 0100

° 00000000000000000 0000000011 0100

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

Portanto, ler o bloco 3

° 00000000000000000 0000000011 0100

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

Não contém dados válidos

° 00000000000000000 0000000011 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Ler o bloco, retorna *f*

° 00000000000000000 0000000011 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

4. Ler 0x00008014 = 0…10 0..001 0100

° 00000000000000010 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

Portanto, ler o bloco 1, que está válido

° 00000000000000010 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

Mas as etiquetas não são iguais! (0 != 2)

° 00000000000000010 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

falta, substituir o bloco 1 e a etiqueta

° 00000000000000010 0000000001 0100

Etq. Índice Desvio

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

E retornar *j*

° 00000000000000010 0000000001 0100

Etq. Índice Desvio

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Exercício: O que acontece?

° Escolha: sucesso, falta, falta com subst.
Valores retornados: a ,b, c, d, e, ..., k, l

° Ler endereço 0x00000030 ?
00000000000000000 0000000011 0000

° Ler endereço 0x0000001c ?
00000000000000000 0000000001 1100

Cache

Índ.	Valid.	Etq.	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...				...		

Respostas

- 0x00000030 : sucesso

  Índice = 3, Etq. ok, desvio = 0, valor = e

- 0x0000001c : falta com subst.

  Índice = 1, Etq. !=, subst. da memória,
  Desvio= 0xc, valor = d

- O valor lido é o mesmo, quer esteja na cache ou na memória :
  - 0x00000030 = e
  - 0x0000001c = d

Memória

Endereço	Conteúdo
...	...
00000010	a
00000014	b
00000018	c
0000001c	d
...	...
00000030	e
00000034	f
00000038	g
0000003c	h
...	...
00008010	i
00008014	j
00008018	k
0000801c	l
...	...

Avaliação de desempenho

- Desenhar o sistema de memória de forma a minimizar o *Tempo Médio de Acesso à Memória*
  - Minimizar:

  TMAM= Tempo do sucesso da cache +
  (Penalidade por falta da cache x Taxa de faltas)

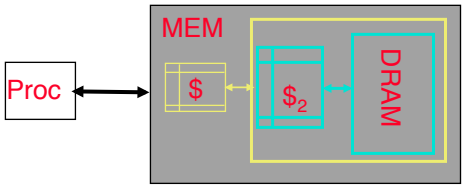
- Cria a ilusão de uma memória de grande capacidade , barata e rápida - em média

Exemplo

- Assuma
  - Tempo no caso de sucesso da cache = 1 ciclo
  - Taxa de faltas da cache = 5%
  - Penalidade por falta da cache = 20 ciclos
  - Calcular TMAM…
- Tempo médio de acesso à memória
  - = 1 + 0.05 x 20
  - = 1 + 1 ciclos
  - = 2 ciclos

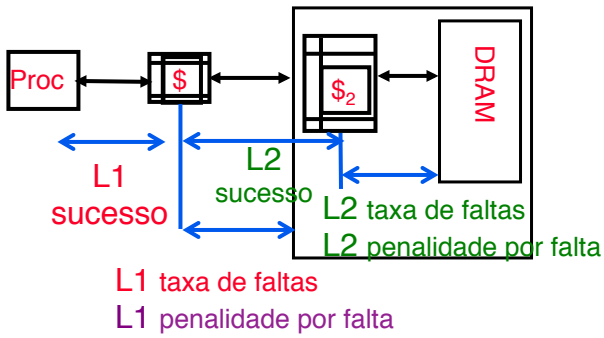
Diminuindo a penalidade por falta da cache

- Quando a utilização de caches começou a ser popular a penalidade por cada falta era ~10 ciclos de relógio do processador.
- Hoje em dia, com um processador a 2400 MHz (0.4ns por ciclo) e 80ns para aceder à DRAM ⇒ 200 ciclos de relógio do processador!



Solução: outra cache entre a memória e a cache do processador : Cache de Segundo Nível (L2)

Analisando uma cache multi-nível



TMAM =
Tempo sucesso L1 + Taxa de faltas L1* Penalidade por falta L1

Penalidade por falta L1 =
Tempo sucesso L2+ Taxa de faltas L2 * Penalidade por falta L2

Analisando uma cache multi-nível

Temos portanto:

TMAM =
Tempo sucesso L1 + (Taxa de faltas L1*
(Tempo sucesso L2+ Taxa de faltas L2 * Penalidade por falta L2))

Valores típicos

- L1
  - tamanho: dezenas de KB
  - Tempo de sucesso: um ciclo de relógio
  - Taxa de faltas: 1-5%
- L2:
  - tamanho: centenas de KB
  - Tempo de sucesso: alguns ciclos de relógio
  - Taxa de faltas: 10-20%
- A taxa de faltas do nível 2 é a fracção das faltas do nível 1 que também não estão presentes no nível 2.
  - Porquê tão elevada?

Exemplo: Com cache de nível 2

- Assuma
  - Tempo de sucesso L1 = 1 ciclo
  - Taxa de faltas L1 = 5%
  - Tempo de sucesso L2 = 5 ciclos
  - Taxa de faltas L2 = 15% (% faltas L1 que faltam)
  - Penalidade por falta L2 = 200 ciclos
- Penalidade por falta L1 = 5 + 0.15 * 200 = 35
- TMAM = 1 + 0.05 x 35= 2.75 ciclos

Exemplo: Sem cache de nível 2

- Assuma
  - Tempo de sucesso L1 = 1 ciclo
  - Taxa de faltas L1 = 5%
  - Penalidade por falta L1 = 200 ciclos
- TMAM = 1 + 0.05 x 200 = 11 ciclos

- 4x mais rápido com cache L2! (2.75 vs. 11)

Como escrever a memória em cache?

- "Write-through"
  - Actualizar a palavra no bloco da cache e a palavra correspondente em memória
- "Write-back"
  - Actualizar a palavra no bloco da cache
  - Permitir que a palavra correspondente em memória fique desactualizada.

  ⇒ adicionar um bit ('dirty bit') a cada bloco indicando que a memória necessita de ser actualizada quando o bloco for substituído

  ⇒ O SO actualiza a memória antes do I/O…
- O que se ganha com isto?

Concluindo…

- Gostariamos de ter a capacidade do disco rígido funcionando à velocidade do processador: infelizmente não é possível.
- Portanto criamos uma hierarquia de memória:
  - Cada nível sucessivamente inferior contém a "informação mais usada" pelo nível superior
  - Explora o ACESSO LOCALIZADO
  - Optimiza o caso comum, não se preocupa tanto com as excepções (príncipio usado no desenho do MIPS)
- Grande ideia!
  - Se uma tarefa é custosa mas queremos executá-la repetidamente…
  - executa-se uma vez e faz-se "caching"do resultado