

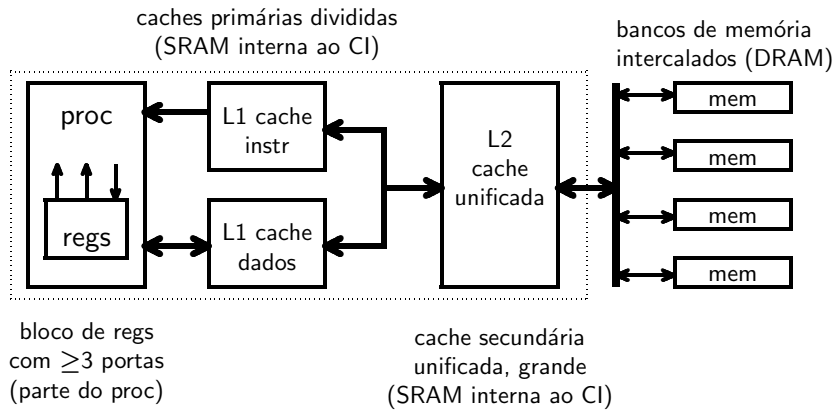
revisão – Revisão de Caches

- Localidade Temporal
 - * bloco poderá ser referenciado novamente no futuro próximo
- Localidade Espacial
 - * blocos vizinhos poderão ser referenciados no futuro próximo
- Taxa de Acertos = $\text{núm_acertos} / \text{núm_referências}$
 - * Taxa de Faltas = $1 - \text{Taxa_de_acertos}$
- TMAM = $\text{TempoDeAcerto} + (\text{TaxaDeFaltas} \times \text{PenalidadePorFalta})$
 - melhorar um termo geralmente piora outro/s
- 3 Cs: faltas compulsórias, por conflitos, por capacidade
 - * compulsórias: **são compulsórias...**
 - * conflitos: $\text{ender} \neq \text{s}$ mapeiam no mesmo bloco **+associatividade**
 - * capacidade: $|\text{conj de dados}| > |\text{cache}|$; **+capacidade**

UFPR BCC CI212 2016-2— projeto de caches (ii)

1

Hierarquia de Memória



$$T_{\text{mam}} = T_{L1} + F_{L1} \cdot [T_{L2} + F_{L2} \cdot (T_m + F_m \cdot ?)]$$

UFPR BCC CI212 2016-2— projeto de caches (ii)

2

And now for something completely different¹

Até agora, somente estudamos caches acessadas para leitura

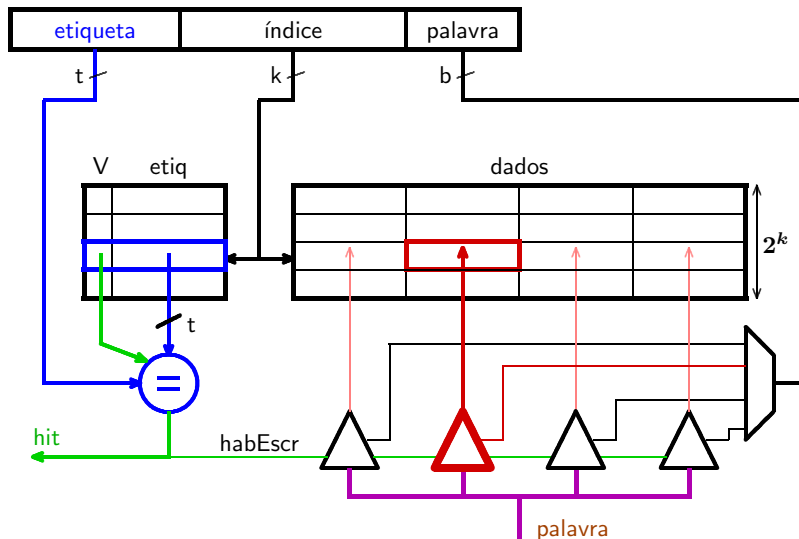
O que acontece nas escritas?

¹ with thanks to Monty Python

UFPR BCC CI212 2016-2— projeto de caches (ii)

3

Implementação da Escrita



UFPR BCC CI212 2016-2— projeto de caches (ii)

4

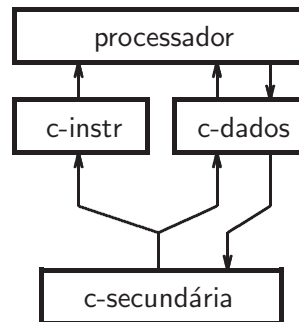
Políticas de escrita

Escrita forçada (*write-through*)

propaga escrita até memória
 escritas completam na
 velocidade da **memória**

Escrita preguiçosa (*write-back*)

acumula escritas na cache
 — bit **sujo** para lembrar
 escritas completam na
 velocidade da **cache**
 — falta em leitura implica na
 escrita da vítima, se bit sujo
 falta → escritaSujo+leitura



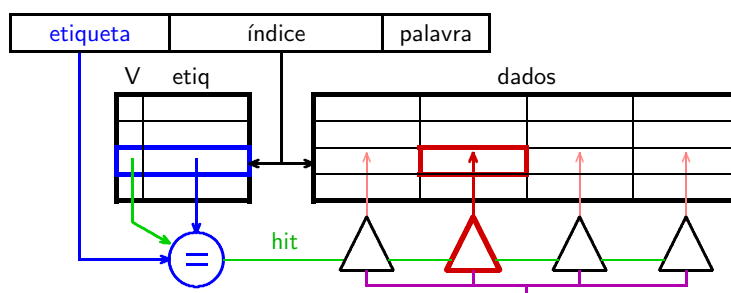
UFPR BCC CI212 2016-2— projeto de caches (ii)

5

Políticas de escrita – Escrita Forçada

Escritas completam na **velocidade da memória**
 porque toda escrita é propagada até a memória
 → memória **sempre** contém cópia atualizada

temporização: compara-etiqueta || escreve-dado **um ciclo**
 se acerto → tudo pronto
 se falta → sobrescreve bloco antigo (memória atualizada em //)

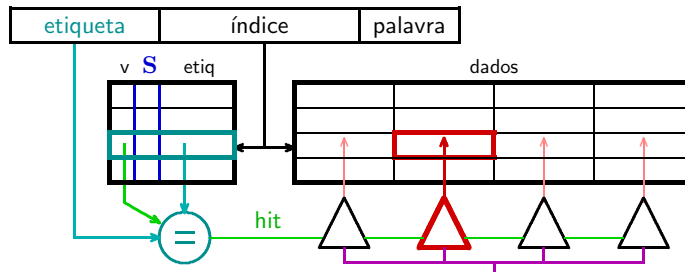


UFPR BCC CI212 2016-2— projeto de caches (ii)

6

Políticas de escrita – Escrita Preguiçosa

Escritas completam na **velocidade da cache**;
 acumula escritas na cache → a cada escrita, bit **sujo** é ligado
falta: ao substituir, bloco vítima sujo enviado para atualizar memória
 falta na leitura pode causar escrita se bloco vítima está sujo
temporização: compara-etiqueta ; escreve-dado **dois ciclos**
 se acerto → só escreve no próximo ciclo
 se falta → **não pode sobrescrever sem atualizar memória**



UFPR BCC CI212 2016-2— projeto de caches (ii)

7

Políticas de escrita – Escrita Preguiçosa

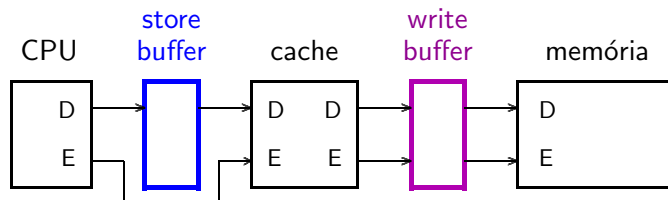
Pode reduzir tempo de escrita com um **store buffer**

temporização: compara-etiqueta || escreve-dado-no-store buffer
 atualiza cache no próximo ciclo vago, em acerto ou em falta

Usa **dois buffers**:

store buffer “antes da cache” para minimizar tempo de escrita

write buffer “depois da cache” para minimizar tempo de substituição



UFPR BCC CI212 2016-2— projeto de caches (ii)

8

Implementação de Escrita Forçada

A cada instrução `sw $r, desloc($i)`
 propaga conteúdo de `$r` até memória
 → `sw` custa referência à memória (10-100 ciclos)
 10% das referências são escritas
 → muito tráfego entre cache e memória
 MAS cache e memória sempre consistentes

```
for (i=0; i<100; i++) /* i → 100 × {lw; add; sw} */
                    /* 100 acessos ao segundo nível */
    res[i] = vetor[i] * const;
```

UFPR BCC CI212 2016-2— projeto de caches (ii)

9

Implementação de Escrita Preguiçosa

A cada instrução `sw $r, desloc($i)`
atualiza bloco na cache e marca-o como sujo
→ `sw` custa referência à cache (se acerto, 2 ciclos)
10% das referências são escritas
→ pouco tráfego entre cache e memória
MAS cache e memória ficam **IN**consistentes
falta de leitura pode custar *escrita* ; *busca*

```
for (i=0; i<100; i++) /* i → 100 × {lw; add; sw} */  
                        /* 2 acessos a segundo nível */  
    res[i] = vetor[i] * const;
```

O que fazer quando ocorre uma falta na escrita?

- **Aloca espaço na escrita** *write-allocate*
 - ★ espaço é alocado na cache para bloco faltante, e então é atualizado
 - ★ se uma palavra no bloco foi atualizada, outras também o serão...
 - ★ se cache com escrita preguiçosa, falta provoca até duas transações:
 - 1) se bloco está sujo, expurga-o
 - 2) carrega bloco faltante
- **Não aloca espaço na escrita** *no-write-allocate*
 - ★ não é alocado espaço na cache para bloco faltante
 - ★ se não ocorreu falta de leitura, bloco pode não ser necessário...
 - ★ bloco faltante é atualizado diretamente na memória
 - ★ fila de escrita é imprescindível
- **Combinações comuns**
 - ★ escrita forçada & não-alocação de espaço (na falta) *no-fetch-on-write*
 - ★ escrita preguiçosa & alocação de espaço (na falta) *fetch-on-write*

Fila de escrita I

Referências de escrita bloqueiam processador até que acesso ao nível mais baixo da hierarquia complete

10% das referências são escritas;
escritas na cache secundária custam ≈ 10 ciclos
 $T_{\text{mem}} = 0.9 * 1 + 0.1 * 10 = 1.9$ ciclos

solução: *fila de escrita* *write-buffer*
para desacoplar velocidade do processador da velocidade da memória

- * imprescindível com escrita forçada
- * com escrita forçada, fila tem largura \geq uma palavra
- * com escrita preguiçosa, fila tem largura de um bloco porque todo o bloco é marcado sujo e precisa ser atualizado na memória

Fila de escrita II

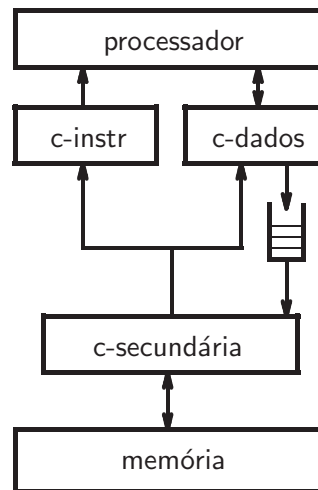
Fila reduz tempo médio da referência de escrita:

escrever na interface da fila custa 1 ciclo

cada elemento da fila contém um par <endereço, bloco*>

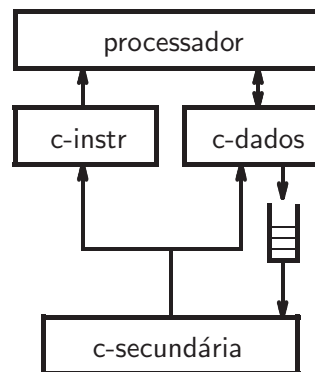
* palavra⁺ (forçada), bloco (preguiçosa)

controlador de cache efetua atualização da L2/memória



Fila de escrita III

- Processador executa `sw $5,24($8)`
 - * em MEM, processador insere na fila par <(24+\$8), \$5> ←EF
 - * se há espaço na fila, escrita completa em um ciclo
 - * senão, processador bloqueia até abrir espaço na fila, enquanto escrita na cabeça da fila é propagada até L2
- Fila tem capacidade para 1-16 registros
fila enche na entrada de funções com muitos parâmetros...



Desempenho de Caches I

$$\text{TempoCPU} = (\text{ciclosEmExecução} + \text{ciclosParados}) \times \text{duraçãoDoCiclo}$$

$$\text{ciclosParados} = (\text{ciclosDeLeituraParados} + \text{ciclosDeEscritaParados})$$

$$\text{ciclosDeLeituraParados} = \text{leituras/programa} \times \text{taxaFaltasLeitura} \times \text{penalidadeLeitura}$$

$$\text{ciclosDeEscritaParados} = \text{escritas/programa} \times \text{taxaFaltasEscrita} \times \text{penalidadeEscrita} + \text{esperaNaFilaDeEscrita}$$

Desempenho de Caches II

$$\text{TempoCPU} = (\text{ciclosEmExecução} + \text{ciclosParadosPorMemória}) \times \text{duraçãoDoCiclo}$$

Combinando escritas e leituras numa única taxa:

ciclosParadosPorMemória

$$\begin{aligned} &= \text{referências/programa} \times \text{taxaDeFaltas} \times \text{penalidadePorFalta} \\ &= \text{instruções/programa} \times \text{faltas/instrução} \times \text{penalidadePorFalta} \end{aligned}$$

Desempenho de Caches – exemplos

Exemplo 1: GCC em máquina com memória perfeita atinge $\text{CPI}=2.0$.

Taxa de faltas em instruções é 5%; taxa de faltas em dados é 10%; penalidade por falta é 12 ciclos; 33% das instruções são LDs e STs. Determine relação entre velocidades com memória perfeita e real.

$$\text{ciclosPerdidosInstruções} = \text{IC} * 5\% * 12 = 0.6 \text{ IC}$$

$$\text{ciclosPerdidosDados} = \text{IC} * 33\% * 10\% * 12 = 0.4 \text{ IC}$$

$$\begin{aligned} \text{Número total de ciclos perdidos em faltas} &= 0.6 + 0.4 \text{ IC} = 1.0 \text{ IC} \\ \text{CPI}_{\text{real}} &= 2.0 + 1.0 = 3.0 \end{aligned}$$

$$\text{tempoCPU}_{\text{mem_Real}} = \text{IC} * \text{CPI}_{\text{real}} * \text{ciclo} = 3.0 * \text{IC} * \text{ciclo}$$

$$\text{tempoCPU}_{\text{mem_Ideal}} = \text{IC} * \text{CPI}_{\text{ideal}} * \text{ciclo} = 2.0 * \text{IC} * \text{ciclo}$$

$$3.0 / 2.0 = 1.5$$

Desempenho de Caches – exemplos

Ex. 2: GCC em máquina com memória perfeita atinge $\text{CPI}=2.0$.

Taxa de faltas em instruções é 5%; taxa de faltas em dados é 10%; penalidade por falta é 12 ciclos; 33% das instruções são LDs e STs. Determine relação entre velocidades com memória perfeita e real.

Número total de ciclos perdidos em faltas =

$$\text{IC} * (5\% * 12 + 0.33 * 10\% * 12) = 1.0 \text{ IC}; \quad \text{CPI}_{\text{lenta}} = 2.0 + 1.0 = 3.0$$

Nova CPU com relógio 2 vezes mais rápido, penalidade é 24 ciclos.

Número total de ciclos perdidos em faltas =

$$\text{IC} * (5\% * 24 + 0.33 * 10\% * 24) = 2.0 \text{ IC}; \quad \text{CPI}_{\text{rapida}} = 2.0 + 2.0 = 4.0$$

$$\text{Ganho} = \text{tempo}_{\text{lenta}} / \text{tempo}_{\text{rapida}} =$$

$$(\text{IC} * \text{CPI}_{\text{lenta}} * \text{ciclo}) / (\text{IC} * \text{CPI}_{\text{rapida}} * \text{ciclo} / 2) =$$

$$(3.0 / (4.0 * 1/2)) = 3/2 = 1.5 \ll 2.0 !!$$

resumo – Caches

- Localidade
 - ★ Temporal: bloco será referenciado novamente no futuro próximo
 - ★ Espacial: blocos vizinhos serão referenciados no futuro próximo
- 3 Cs: faltas compulsórias, por conflitos, por capacidade
- Políticas de escrita
 - ★ escrita forçada, escrita preguiçosa
 - ★ fila de escrita desacopla velocidades da CPU daquela da L2/memória
 - ★ nas faltas em escrita: aloca espaço, não-aloca espaço
- $\text{TempoCPU} = (\text{ciclosEmExecução} + \text{ciclosParadosMemória}) \times \text{duraçãoDoCiclo}$
 $\text{ciclosParadosPorMemória}$
 - = referências/programa \times taxaDeFaltas \times penalidadePorFalta
 - = instruções/programa \times faltas/instrução \times penalidadePorFalta