AOC - Pipeline e Hierarquia de memória - Prova 1 AOC 2 - 22/09/2018

André Marcelino de Souza Neves

Setembro de 2018

1 Caminho de dados Pipeline

1.1 Introdução

1.2 Controle

1.3 Hazards: Forwarding and Stalling

1.4 Hazards de Controle

1.5 Exceptions

2 Hierarquia de memória

2.1 Tecnologias de memória

2.2 Mapeamento direto

Uma estrutura de cache em que cada local de memória é mapeado exatamente para um local na cache.

2.2.1 Blocos únicos (apenas uma word)

Dado um endereço de word (ou endereço do bloco), o endereço de cache a usar é:

$$(BlockAddress)\%(CacheSize) \tag{1}$$

O valor da tag é calculado por:

$$(BlockAddress)/(CacheSize) \tag{2}$$

Onde:

- BlockAddess - Endereço do **bloco**
- CacheSize - Número de blocos na cache

A tag é armazenada juntamente ao dado, dentro da cache. Isso é necessário pois a função de mapeamento não é injetora, ou seja, podem ter dois blocos mapeados para um mesmo local de cache. Além disso, é necessário armazenar o bit de validade.

2.2.2 Cálculo do tamanho da cache

Em MIPS, o endereçamento é feito em bytes. Como cada palavra possui 4 bytes, a modificação dos últimos dois bits de um endereço alteram apenas o byte que será acessado, de uma mesma palavra.

Se temos endereços de 32 bits, pela afirmação acima, 30 bits são usados para armazenar palavras.

Consideremos blocos de cache, que podem armazenar uma ou mais palavras. Para facilitação da convenção, o número de palavras dentro de um bloco deve ser potência de 2. Seja 2^m o número de palavras por bloco, então são usados m bits para o offset de endereço dentro do bloco.

Isso mostra que dado um endereço de memória de 32 bits, 2 bits são usados para offset de byte de uma palavra, e com isso sobram 30 bits para endereçamento de palavras. Se temos blocos de 2^m palavras, então temos m bits para offset de endereço da palavra dentro do bloco. Logo, o endereço do nosso bloco tem $32 - 2 - m = 32 - (m + 2)$ bits.

A expressão acima é útil para saber o tamanho do endereço de cada bloco. Se nossa cache terá a capacidade para armazenar 2^n blocos, temos n bits para endereçar nosso bloco em relação à cache. O endereço é será calculado pela equação 1:

$$\text{BlockAddress } \% \ 2^n$$

Para essa operação, basta utilizar os n bits menos significativos do endereço do bloco, e os bits restantes são usados para identificar o restante do endereço, sendo usado como TAG. Com isso, o tamanho da tag em bits é:

$$32 - (m + 2) - n$$

$$32 - (m + n + 2)$$

O tamanho total da cache, em bits, é:

$$NumBlocos * (TamanhoBloco + TamanhoTag + BitValidade)$$

Onde:

- NumBlocos = 2^n
- TamanhoBloco = $2^m * 32$
- TamanhoTag = $32 - (m + n + 2)$
- BitValidade = 1

Com isso:

$$TamanhoCache = 2^n * (2^m * 32 + 32 - (m + n + 2) + 1)$$

$$TamanhoCache = 2^n * (2^m * 32 + 31 - m - n)$$

2.2.3 Endereçamento de blocos com várias palavras

Partindo do endereçamento por bytes:

O endereço de um bloco é:

$$\frac{ByteAddress}{BytesPerBlock} \tag{3}$$

O conjunto de endereços dos bytes contidos nesse bloco é de:

$$\left[\frac{ByteAddress}{BytesPerBlock} * BytesPerBlock\right] \tag{4}$$

Até:

$$\left[\frac{ByteAddress}{BytesPerBlock} * BytesPerBlock + BytesPerBlock - 1\right] \tag{5}$$

O mesmo vale para um contexto que use palavras em vez de bytes

2.3 Lidando com falhas de cache

Falha da cache de instruções

- PC = PC - 4
- Iniciar a leitura na memória principal e aguardar o processo
- Escrever o dado na cache, juntamente com a TAG

2.4 Escrita

2.4.1 Write-through

Ao atualizar um dado na cache, uma inconsistência é gerada, já que o valor na memória principal é diferente.

Sempre escrever o dado na cache e na memória, ao sobrescrever um dado. Isso causa perca de performance, por ter que esperar a memória escrever o dado.

Se 10% das instruções são stores e se o CPI sem falha de cache é 1.0, em cada falha de escrita gastaremos 100 ciclos a mais. Como apenas 10% das intruções gravam dados, o novo CPI é:

$$1 + 10\% * 100 = 11CPI$$

Solução: write buffer. Armazenadar o dado pendente de ser gravado na memória. Os dados são gravados no buffer enquanto se espera o delay de gravação na memória principal.

Em uma gravação, após escrever o dado na cache e no buffer, a execução pode ser continuada. Se o buffer estiver cheio, a execução é pausada até houver um espaço livre.

2.4.2 Write-back

Alternativa ao Write-through. Quando ocorre uma escrita, o novo valor é escrito apenas na cache, e tal bloco é marcado como "sujo". O dado será de fato gravado na memória apenas ao ter o bloco da cache substituido.

2.5 Mapeamento direto: (Copiado do livro, resumir depois)

Resumo:

- **Mapeamento direto**: Existe um local definido para localizar cada palavra.
- **Write-through**: Para cada escrita na cache implica em atualizar no mesmo momento a memória.
- **Write-back**: Não atualiza a memória imediatamente, apenas no momento em que o bloco da cache for substituido.
- **Blocos grandes**: Tira vantagem da localidade espacial, além de reduzir o tamanho da TAG. Um bloco maior diminui a taxa de falha, mas aumenta a penalidade. Isso porque, ao ocorrer falha, mais palavras terão que ser obtidas na memória, e a performance fica dependente da largura de banda (razão entre quantidade de palavras lidas por unidade de tempo)

We began the previous section by examining the simplest of caches: a direct-mapped cache with a one-word block. In such a cache, both hits and misses are simple, since a word can go in exactly one location and there is a separate tag for every word. To keep the cache and memory

consistent, a write-through scheme can be used, so that every write into the cache also causes memory to be updated. The alternative to write-through is a write-back scheme that copies a block back to memory when it is replaced; we'll discuss this scheme further in upcoming sections.

To take advantage of spatial locality, a cache must have a block size larger than one word. Th e use of a larger block decreases the miss rate and improves the efficiency of the cache by reducing the amount of tag storage relative to the amount of data storage in the cache. Although a larger block size decreases the miss rate, it can also increase the miss penalty. If the miss penalty increased linearly with the block size, larger blocks could easily lead to lower performance.

To avoid performance loss, the bandwidth of main memory is increased to transfer cache blocks more effi ciently. Common methods for increasing bandwidth external to the DRAM are making the memory wider and interleaving. DRAM designers have steadily improved the interface between the processor and memory to increase the bandwidth of burst mode transfers to reduce the cost of larger cache block sizes.

2.6 Falhas de leitura e performance

No acesso à cache, podem acontecer falhas ao acessar dados e ao acessar instruções.

2.6.1 Equações

A medição de performance considera o número de ciclos gastos para acessar a cache normalmente, caso ocorresse 100% de hit, acrescido dos ciclos gastos para buscar dados nos níveis inferiores:

$$\text{Clocks} = \text{Frequência} * \text{tempo}$$
$$[\frac{Clocks}{s}] * [\text{s}] = [\text{Clocks}]$$

$$\text{Clocks} = \frac{Tempo\_total}{Tempo\_de\_ciclo}$$
$$[\frac{s}{\frac{s}{Clock}}] = [\text{Clock}]$$

$$\text{Ciclos de clock} = \text{Clocks padrão} + \text{Clocks de stall}$$

$$\text{Tempo de CPU} = \text{Ciclos de clock} * \text{tempo de ciclo (período)}$$
$$[\text{Clocks}] * [\frac{s}{Clock}] = [\text{s}]$$

$$\text{Clocks de stall} = \text{Stalls de leitura} + \text{Stalls de escrita}$$

$$\text{Stalls de leitura} = \text{Leituras} * \text{Taxa de falha} * \text{Penalidade de falha}$$

Stalls de escrita = Estritas * Taxa de falha * Penalidade de falha + Stalls do buffer de escrita

Onde:

- Leituras: Número de leituras de memória feita no programa
- Taxa de falha: Do número total de leituras, quantas falham (percentual)
- Penalidade de falha: Ciclos gastos para cada falha

Multiplicar **Leituras * Taxa de falha** resulta no número real de operações de leitura que levam à falha, e multiplicar tal resultado por **Penalidade de falha** resulta no número total de ciclos gastos para realizar os stalls

2.6.2 Exemplo

- Dados:
  - Miss rate (cache de instruções): 2%
  - Miss rate (cache de dados): 4%
  - CPI padrão: 2
  - Penalidade por falha: 100 ciclos
  - Instruções de dados: 36%
- Cálculos (Considere I o número de instruções):
  - Ciclos de penalidade de acesso à instruções: 2% * 100 * I = 2,00 * I
  - Ciclos de penalidade de acesso à dados: 36% * 4% * 100 * I = 1,44 * I
  - Ciclos totais: CPI_Padrao * I + Penalidade_instrucoes + Penalidade_dados = 2,00 * I + 2,00 * I + 1,44 * I = 5,44 * I
  - CPI : $\frac{Ciclos}{I}$ = 5,44

2.6.3 Tempo médio e ciclos médios

- Ciclos = CPI Padrão * I + Taxa de falhas * I * Clocks de penalidade por falha
- CPI Total = CPI padrão + Taxa de falhas * Clocks de penalidade por falha
- AMAT = Tempo padrão (por execução perfeita de instrução) + Taxa de falhas * Tempo de penalidade por falha
- Ciclos = CPI Total * I
- Tempo total = Ciclos * Tempo de ciclo = $\frac{Ciclos}{Frequencia}$
- Tempo total = Tempo * CPI Padrão * I + Tempo * Taxa de falhas * I * Clocks de penalidade por falha

- AMAT = $\frac{Tempo\_total}{I}$ = CPI * Tempo de ciclo
- AMAT = Tempo * CPI Padrão + Tempo * Taxa de falhas * Clocks de penalidade por falha

2.7 Cache com mapeamento associativo

Cache mapeada diretamente tem um conjunto para cada bloco

2.8 Performance com mapeamento associativo

Exemplo (livro)
memória principa : 100ns Clock rate : 4 Ghz CPI de 1 sem stall
taxa de miss por instrução na L1: 2
Cache L2:
5 ns de tempo reduzir a taxa de miss global para 0.5
penalidade para acessar a memória

$$100ns * 4Ghz = 400ciclos$$

penalidade para acessar a cache nível 2

$$5ns * 4Ghz = 20ciclos$$

Cada aumento no grau de associatividade diminui um bit no índice de mapeamento, e aumenta um bit na tag