Relatório OC Lab 01 - Simple Cache Simulator

Eduardo Cardoso - 105894; Martim Rito - 106427; Tiago Santos - 106329
Grupo 50 - L05

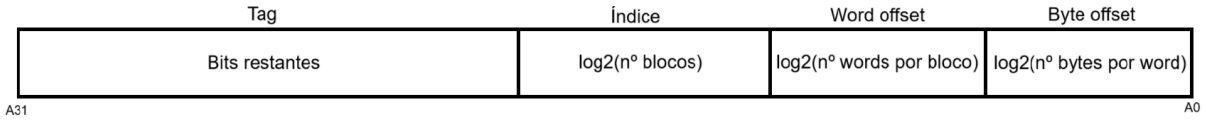
Tarefa 1 - Direct-Mapped L1 Cache

Para realizar esta tarefa foi necessário ajustar parte do código base, de forma à cache de nível 1 funcionar para múltiplas linhas.
Para realizar as operações necessárias ao funcionamento desta cache, utilizamos, de forma abstrata, constantes que indicam: o tamanho da cache, o tamanho de um bloco e o tamanho de uma palavra.
Comecemos por explicar de que forma é que calculamos a tag, índice e o offset dos bytes e das palavras, assim como os bits do endereço da memória que representam cada uma destas 4 componentes.

Cálculo dos números de bits do endereçamento:

- **Byte offset**: tendo em conta que a memória é endereçável ao byte, o número de bits para o byte offset, corresponde aos bits necessários para endereçar todos os bytes existentes numa palavra, ou seja: logBase2(nº bytes numa palavra).
- **Word offset**: a mesma lógica aplica-se aos bits para o word offset, mas com o número de palavras em cada bloco: logBase2(nº words num bloco).
- **Total offset**: assim, número total de bits para o offset = nº bits byte offset + nº bits word offset.
- **Índice**: como a cache L1 é direct mapped, o número de bits do índice corresponde ao número de bits necessários para endereçar todos os blocos da cache, ou seja, logBase2(nº total de blocos da cache).
- **Tag**: o resto dos bits do endereço (até ao 31 porque os endereços são de 32 bits) são utilizados para a tag.

Posto isto, o endereço de memória pode ser estruturado da seguinte forma:



Para extrair estas componentes do endereço de memória, realizamos certas operações sobre o endereço original:

- **Tag**: shift right do nº de bits do offset + nº de bits do índice, retirando assim os bits correspondentes ao índice e ao offset.
- **Índice**: shift right do nº de bits do offset total e em seguida resto da divisão inteira pelo nº de linhas da cache, de forma a retirar os bits da tag e extrair o índice.
- **Word offset**: resto da divisão inteira pelo nº de bytes num bloco para retirar os bits da tag e do índice e depois shift right do nº de bits do byte offset.
- **Byte offset**: resto da divisão inteira pelo tamanho de uma palavra (em bytes) de forma a excluir os bits do word offset, do índice e da tag.

Quando acedemos à cache L1, inicializamos a cache, se necessário (colocando o bit valid a 0 em cada linha para representar que as linhas estão inválidas/vazias).

Em seguida, calculamos o endereço do bloco que contém a palavra que queremos aceder (colocando os bits do offset a 0) e verificamos, através do índice, se este bloco está presente na cache.
Em caso de miss, ou seja, linha inválida ou tag diferente, vamos buscar o bloco pretendido à memória principal e escrevemo-lo na cache, substituindo o bloco que lá estava (se lá estiver).
É importante salientar que antes deste replace, e porque utilizamos a política de escrita write-back, verificamos se o bloco presente na cache está dirty, e se estiver, atualizamos a memória principal primeiro, escrevendo-o no seu endereço respectivo.
Por fim, procede-se então à operação pretendida (escrita ou leitura). Para tornar estas operações mais flexíveis e seguras, adicionamos a possibilidade de ler e escrever a partir de qualquer byte da palavra, sendo que só será escrito ou lido até ao fim da mesma. Em caso de escrita, atualizamos ainda o dirty bit para 1.

Tarefa 2 - Direct-Mapped L2 Cache

Nesta tarefa, acrescentamos uma cache de nível 2 para reduzir o nº de acessos à memória principal, cujo funcionamento segue o mesmo princípio da cache L1 mas age como intermediária entre a cache L1 e a memória principal, ou seja, em caso de miss na cache L1, em vez de aceder diretamente à memória principal, recorremos primeiro à cache L2.
Só em caso de miss na cache L2 é que a memória principal é acedida, e neste caso, o bloco é escrito tanto na cache L2, como na cache L1.
O caso de substituir um dirty block é estendido para funcionar com mais um nível de hierarquia, ou seja, se este ocorrer na cache L1, atualizamos L2 e, se ocorrer um write-miss, vamos buscar o bloco à memória principal e escrevemos essa palavra na cache L2. Se o replace de um dirty block ocorrer na cache L2, atualizamos a memória principal.

Tarefa 3 - Direct-Mapped L2 Cache

Nesta última tarefa mudamos a cache L2 de forma a ser 2-Way-Associative, isto é, estar organizada em conjuntos/sets de dois blocos (uma linha tem dois blocos).
Desta forma, um bloco da memória principal não é mapeado apenas para um bloco na cache L2, mas para um conjunto de 2 blocos da mesma.
Isto implica que vamos precisar de menos 1 bit para o índice, pois este referencia cada set e nº de sets da cache = nº de blocos da cache / 2, e que a tag aumenta em 1 bit
Para além disto, quando queremos aceder a um endereço, temos de verificar ambos os blocos do set.
Em caso de miss (não está em nenhum dos blocos) procedemos como nas tarefas anteriores, à exceção de que, se for necessário dar replace de um dos blocos porque ambos estão ocupados, escolhe-se segundo o método LRU, ou seja, o que não é usado há mais tempo.
As restantes operações são realizadas de forma idêntica à tarefa anterior, tendo em conta o número do bloco onde está a informação pretendida.

Conclusões

No nosso projeto, conseguimos ainda simular e perceber os tempos de acesso dos diferentes níveis da hierarquia de memória, o que nos permite testar o programa e tirar conclusões sobre o desempenho de cada uma das tarefas. Com base nos testes fornecidos e por nós desenvolvidos, conseguimos confirmar a importância de um segundo nível de cache, já que, para um mesmo programa, tivemos muito menos acessos à memória principal e um tempo de acesso à memória total bastante inferior do que quando tínhamos apenas um nível de cache.