Relatório Laboratório 1

Grupo 19: Maria Ramos (105875), Enzo Nunes (106336), Guilherme Campos (105875)

Introdução

Este projeto, parte da disciplina de Organização de Computadores, tem como objetivo desenvolver uma hierarquia de memória que inclua uma *cache* L1 mapeada diretamente e uma *cache* L2 também mapeada diretamente. Por fim, a tarefa consiste em modificar a *cache* L2, transformando-a numa *cache* associativa por conjunto de 2 vias, aplicando políticas de substituição como *Least Recently Used* (LRU) e *write-back*.

Exercício 1

Neste primeiro exercício, o objetivo é desenvolver uma *cache* de nível 1 (L1) diretamente mapeada. Nesta, cada bloco de memória é mapeado para uma única linha da *cache*, ou seja, cada endereço de memória principal (DRAM) tem um único lugar específico onde pode ser armazenado. A *cache* é dividida em várias linhas, e cada linha pode armazenar um bloco da memória. O endereço de memória é dividido em três partes principais: bits de *offset*, que indicam a posição dentro de um bloco, os bits de *index*, que especificam em qual linha da cache o bloco deve ser armazenado, e os bits de *tag* que verificam se o bloco armazenado naquela linha é o que está a ser procurado.

Para além disso, numa cache mapeada diretamente**,** se outro bloco de memória for mapeado para uma linha já preenchida, o bloco anterior é substituído. Quanto à política de escrita e leitura, utilizamos um método *write-back*, o que implica que os dados modificados na *cache* não são imediatamente escritos de volta à memória principal, apenas quando são removidos.

A *cache* L1 tem $2^8 = 256$ linhas, exigindo 8 bits para o *index*, e cada bloco possui 16 *words* ($16 \times 4 = 2^6 = 64$ bytes), necessitando de 6 bits para o *offset* e, de resto, 18 bits para a *tag*.

A verificação de *hit* ou *miss* ocorre quando a *tag* do endereço de memória requisitado pelo CPU é comparada com a *tag* armazenada na linha da *cache*, com base no *index*. Se a linha não é válida ou a *tag* é diferente, temos um *cache miss* e temos de aceder à *DRAM* para trazer os dados para a *cache*, marcando a linha como válida e atualizando a *tag*.

A política de *write-back* implica que ao se modificar alguma linha da *cache*, esta deve ser marcada como *dirty*. Assim, se for necessário remover esta linha da *cache*, esta deve ser escrita de volta para a DRAM primeiro.

Exercício 2

O segundo exercício consiste na criação de uma *cache* de nível 2 (L2), também diretamente mapeada. A *cache* L2 servirá como um segundo nível de armazenamento, complementando a *cache* L1 desenvolvida anteriormente. Este segundo nível permite uma maior capacidade de armazenamento, mas menor velocidade de acesso.

A *cache* L2 tem $2^9 = 512$ linhas, com 9 bits para o index, cada bloco possui 16 *words* ($16 \times 4 = 2^6 = 64$ bytes), necessitando de 6 bits para o *offset* e, de resto, 17 bits para a *tag*, um bit a menos que em L1 dado o aumento de um bit no *index*.

Em caso de *miss* na L1, o programa verifica a presença do bloco na L2. Se este não estiver na L2, o bloco é procurado na DRAM e copiado para L2 e L1. Se houver *miss* na L1 e *hit* na L2, o bloco é transferido para a L1. Caso haja necessidade de substituir uma linha *dirty* em qualquer uma das *caches*, esta é escrita de volta para a DRAM primeiro.

Exercício 3

O terceiro exercício tem como objetivo modificar a *cache* L2 criada no exercício anterior para ser uma *cache* associativa por conjunto 2-vias.

Diferente de uma *cache* mapeada diretamente, uma cache associativa por conjunto permite que cada conjunto contenha múltiplas vias, reduzindo conflitos ao armazenar mais de um bloco de memória no mesmo conjunto. Em uma *2-Way Set Associative Cache*, cada conjunto possui 2 vias. Se ambas as vias estiverem ocupadas, utilizamos a política de substituição LRU (Least Recently Used). Este método prioriza substituir a linha que não foi utilizada por mais tempo.

A estrutura do endereço também muda: o número de bits de *offset* permanece em 6, pois o tamanho dos blocos não muda. Para manter o número de linhas (512) e tendo em conta que temos duas linhas por conjunto, teremos $512/2 = 256$ sets, necessitando de 8 bits para o index. Os 18 bits restantes compõem a *tag*.

A lógica da *cache* L1 é idêntica à do exercício anterior. Porém, a lógica de *hits* e *misses* na *cache* L2, associativa por conjunto, funciona de maneira ligeiramente diferente da *cache* diretamente mapeada. Quando um bloco é procurado em L2, encontra-se o *set* correspondente através do *index*. De seguida, compara-se a *tag* do bloco requisitado com as *tags* dos dois blocos presentes no conjunto. Se o bloco de dados não estiver em nenhuma das duas vias do conjunto, temos um *cache miss* e ele precisa ser procurado na DRAM e carregado na *cache*. O bloco carregado substitui o bloco da linha escolhida pela política LRU.

A política de substituição LRU é implementada utilizando contadores em cada linha. A linha que tem o maior contador é a que será substituída, visto que indica que ela foi a menos recentemente utilizada. Quando um set é modificado, o contador da linha que foi modificada é zerado e o da outra é incrementado.

Teste - Próxima palavra do bloco

Além dos testes providenciados pelo professor, realizámos um teste que procura escrever dois valores na mesma linha, em blocos diferentes. Inicialmente, tivemos um *bug* que provocou que o valor fosse apagado de um dos blocos. O problema estava relacionado com o endereço para onde a *Cache* estava a escrever os blocos durante o *write-back.* Com o *bug* resolvido, a *cache* passou a funcionar corretamente.