Organização de Computadores 2023/2024
Lab 2 – Grupo 011
(Leonor Fortes - ist1102695, Isabela Pereira - ist1102703, Beatriz Paulo - ist1102823)

Objetivo

O objetivo deste projeto é desenvolver uma hierarquia de memória completa, implementado uma cache L1 e L2, nos exercícios 4.1 e 4.2 respetivamente, e posteriormente implementar a cache L2 com uma associatividade de 2 vias (exercício 4.3), garantindo a sua interação com a RAM.

Exercício 4.1

Começamos por modificar a cache simples fornecida, de forma a suportar múltiplas linhas implementando um vetor de linhas na estrutura da cache (calculado por o tamanho da cache L1 a dividir pelo tamanho de cada bloco), e acrescentámos também um vetor de dados a cada linha.

O código começa por inicializar os campos de cada linha de cache a 0. Quando fazemos o acesso a L1 calculamos cada secção da *address* (*tag, index* e *offset*), o *offset* fazendo o resto da divisão inteira da *address* pelo tamanho de bloco (últimos 6 *bits* da *address*), o *index* pela *address* a dividir pelo tamanho de cada bloco com a união do nº de linhas de L1 menos 1 (*bits* 6-13) e a *tag* dividindo o *address* pelo tamanho da cache (18 *bits* mais significativos).

Selecionamos a linha de cache com o *index* calculado anteriormente, e verificamos se esta se encontra na cache - *bit* validade (válido=1) e se a *tag* presente é igual à calculada. Se isto não se verificar temos de aceder à DRAM com um endereço de memória calculado anteriormente (*MemAddress: address* com offset a 0), alocando a informação recolhida num bloco temporário. No caso do *miss* do endereço tiver acontecido devido ao facto da linha se encontrar *dirty* então escrevemos a informação atual da linha na RAM. Posteriormente, alocamos a informação do bloco temporário na linha de cache associada ao *index* e atualizamos os restantes campos da mesma. Por fim, é executada a ação referente ao modo de acesso à cache L1, leitura ou escrita.

Exercício 4.2

Esta implementação tem por base o exercício o 4.1, apenas com a adição de uma segunda cache L2. O ficheiro “L2Cache.h” contém, adicionalmente, a estrutura da cache L2. Começamos por inicializar esta nova cache da mesma forma da anterior. Em seguida, atualizamos o tamanho de cada informação da memória RAM (WORD_SIZE para BLOCK_SIZE).

No acesso à cache L1 calculamos novamente a *offset, index* e *tag* como anteriormente descrito, verificamos, outra vez, a existência do bloco na cache, se isto não se verificar, acedemos então à cache L2 (à qual passamos o input da *address* original).

Em relação ao acesso à cache L2, criamos uma função (accessL2) que vai executar o cálculo do *index* e da *tag*, da *address* fornecida pela cache L1, e vai verificar a sua

presença na cache secundária. Depois, reutilizamos então o método de verificação do bloco em cache (usado previamente).

No caso de o acesso à L1 nos dar a informação de que o bloco a que queremos aceder está *dirty,* vamos então escrever essa informação do bloco na cache L2 e posteriormente escrever a informação retirada de L2, remetente à *address* fornecida inicialmente, no bloco da cache L1. Por fim, é executada a ação referente ao modo de acesso à cache L1, leitura ou escrita.

Exercício 4.3

Esta implementação tem como base a solução do exercício 4.2. Começamos por implementar a estrutura da cache L2, tendo em conta a associatividade. Acrescentamos também um inteiro LRU (Least Recently Used) a fim de verificar qual das entradas do set foi utilizada mais recentemente. O número de linhas da L2 tem também em conta as vias de associatividade (2 vias).

O acesso à cache L1 continua com a mesma implementação. Alteramos o acesso à cache L2 na medida em que o acesso à estrutura da linha de cache passou a ser um ponteiro dando a possibilidade de verificar ambas as entradas das vias de associatividade. Verifica a presença da *address* na linha que é suposto, se não se encontrar em nenhuma das vias da linha, passamos a verificar qual das entradas foi usada menos recentemente (sendo 0 a que foi usada à mais tempo e 1 a mais recente). Tendo encontrado a entrada, repetimos o processo da função criada anteriormente (accessL2 na 4.2). Isto é a verificação da entrada (se está *dirty* ou não) e os acessos à RAM necessários. Se o que procuramos estiver na cache L2 fazemos a leitura ou escrita (de acordo com o modo referido) e alteramos qual das vias do bloco foi usada mais recentemente.