Relatório do Laboratório 1:

'Simple Cache Simulator'

Realizado por:

106559 – Francisco Nascimento | 106794 – Tiago Santos | 107301 – João Caçador

Introdução

O objetivo deste laboratório consiste no desenvolvimento de uma hierarquia de memória. Para tal implementamos uma cache L1 e L2 diretamente mapeada e de seguida uma cache L2 de 2 vias. Ambas as caches são endereçáveis ao byte e ambas usam políticas de *Write-Back* e *LRU*.

4.1

Nesta tarefa, foi feita uma evolução do SimpleCache para a implementação de uma cache L1. A principal alteração foi a adição de um array de linhas de cache (*CacheLine*), de tamanho = BLOCK_SIZE, na estrutura L1_Cache (presente no ficheiro header).

Na função *accessL1*, inicializamos todas as linhas da cache. Tendo em conta que $\text{L1\_SIZE} = 256 * \text{BLOCK\_SIZE}$, concluímos que necessitamos de 8 bits ($2^8 = 256$) para o index; como $\text{BLOCK\_SIZE} = 2^6$ precisamos de 6 bits de offset (byte adressable). Por último precisamos de 32-8-6 = 18 bits para a Tag.

Em caso de um *miss,* o bloco correspondente é trazido da memória para um bloco temporário. Se o bloco a ser substituído estiver *Dirty*, efetua-se um *Write-back* deste último para RAM. Para finalizar o tratamento de um miss, alteramos os bits de *Valid* e *Dirty* para 1 e 0, respetivamente, e a *Tag* é ajustada. Em caso de *hit* o acesso é feito diretamente (através do index e offset). O restante código do ficheiro *SimpleCache.c* mantém-se inalterado.

4.2

A segunda tarefa tinha como objetivo estender a hierarquia de caches construída na primeira tarefa, implementando uma cache L2 (segundo nível), de modo a que os endereços não encontrados na primeira cache sejam primeiro procurados na cache L2, de maneira a evitar, com mais eficácia, que seja preciso carregá-los da memória principal.

Acrescentamos por isso uma função *acessL2*, com estrutura lógica semelhante à função

accessL1 do exercício anterior, pois a segunda cache na hierarquia de nível 2 tem um comportamento semelhante à cache L1 na hierarquia de nível 1. A única diferença reside no facto de a cache L2 possuir 512 linhas em vez das 256. Isto resulta em 9 bits de index $(512 = 2^9)$ em vez de 8. A Tag passa assim a ter 32-9-6 = 17 bits. Como o tamanho do bloco não muda, permanecem os 6 bits de offset da cache L1.

Por último, apenas precisamos de alterar o comportamento da função *accessL1* em caso de miss. Em vez de ser trazido o bloco correspondente da memória, o pedido de acesso é agora reencaminhado para a cache L2 através da função *acessL2*. Os bits *Valid* e *Dirty* são repostos da mesma forma que no exercício anterior.

4.3

Para a terceira tarefa, convertemos a cache L2 desenvolvida no exercício anterior numa cache associativa de 2 vias. Para tal, começámos por criar uma data structure denominada *Set_Blocks* que consiste num array de *CacheLine*'s de tamanho 2 (associatividade da cache – definida com constante no ficheiro header); a data struct que representa L2 foi também modificada para passar a ter um array de *Set_Blocks* cujo tamanho é o número de sets. Por fim, alterámos também a data struct *CacheLine* para esta passar a incluir o atributo *Time* necessário para implementar a política de *LRU*.

O número de sets, neste caso particular, corresponde a metade do número de blocos da anterior cache L2 diretamente mapeada (já que associatividade = 2), pelo que concluímos que necessitamos de menos 1 bit para o Index: $\frac{2^9}{2\ vias}$ sets $\Rightarrow$ 9 – 1 = 8 bits de Index. A forma como a Tag é calculada também é alterada para refletir esta mudança (o número de bits para o offset mantém-se).

Mantendo a função *accessL1* inalterada, adaptámos a função *accessL2*. Para um Index em L2, percorremos as 2 vias por forma a encontrar a Tag correspondente à adress que procuramos. Caso haja uma correspondência e a linha seja válida, temos um hit.

Em caso de estarmos perante um miss, utilizamos uma política de substituição de bloco *LRU* para obtermos a via que deve ser substituída. Esta abordagem utiliza o atributo *Time* de maneira a obter a via utilizada há mais tempo. O restante código mantém-se inalterado em relação ao desenvolvido no exercício anterior (nomeadamente o código responsável pela escrita e leitura da cache e pela política de *Write-Back*).