**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*) na estrutura L1\_Cache, que agora armazena múltiplas linhas em vez de apenas uma. Cada linha de cache contém campos como *Valid*, *Dirty*, *Tag* e um *Block\_Data* para gerenciar informações sobre cada linha da cache.

Na função *accessL1*, começamos por percorrer a cache e inicializamos todas as suas linhas. Em seguida extraímos os bits do offset, do index e da tag do endereço fornecido. Para calcular o offset, utilizamos a operação de resto com o tamanho do bloco, obtendo assim os bits menos significativos do endereço. O índice é determinado removendo os bits do offset através de um deslocamento à direita e, em seguida, aplicando a operação de resto com o número de linhas da cache. Por fim, a tag é calculada deslocando os bits do endereço para a direita em uma quantidade equivalente à soma do número de bits do offset e do índice.

Após calcular essas informações, verificamos se ocorre um *miss*. Em caso de *hit*, o acesso (leitura ou escrita) é realizado diretamente. Caso contrário, o bloco correspondente é trazido da RAM para um bloco temporário. Se o bloco a ser substituído estiver *Dirty*, efetua-se um *Write-back* deste último para RAM antes de se escrever o novo bloco na cache. Para finalizar o tratamento de um miss, alteramos os bits de *Valid* e *Dirty* para um e zero, respetivamente, e a *Tag* é ajustada. Finalmente, o acesso desejado é realizado, seja em caso de *hit* ou *miss*, já que o bloco correspondente estará na cache em ambos os casos.

**4.2**

**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: sets ⇒ 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*).