Relatório LAB 1 OC - Grupo 41

Nuno Pelágio (106640), João Teixeira (97226), Miguel Pereira (79740)

Hierarquia de memória (cache)

Primeira parte. Implementação da cache L1 de mapeamento direto, utilizando as seguintes estruturas de dados:

i) ***CacheLine*:** Representa uma linha da cache. Contém um vetor para os dados, a respetiva *Tag* e os bits de validez (*Valid*) e de marcação (*Dirty*), necessário para suportar a política de escrita *write-back.*

```
typedef struct CacheLine {
  uint8_t Valid;              // Indica se a linha contém dados válidos
  uint8_t Dirty;              // Indica se os dados foram modificados (write-back)
  uint32_t Tag;               // Identifica o bloco armazenado
  uint8_t Data[BLOCK_SIZE];   // Armazena os dados do bloco
} CacheLine;
```

ii) ***Cache*:** Agrupa todas as linhas da cache num vetor, com uma linha em cada posição.

```
typedef struct Cache {
  CacheLine lines[L1_LINES];   // Array de linhas da cache L1
} Cache;
```

Segunda parte. Implementação da cache L2 de mapeamento direto, integrada com a cache L1. A cache L2 possui uma estrutura semelhante à da L1, mas com maior número de linhas.

Adicionado um nível à hierarquia, quando um bloco não está presente na cache L1 (*miss*), vai-se procurar primeiro à cache L2, em vez de ir diretamente à *DRAM:*

Como podemos ver neste excerto do código em que na primeira parte o acesso era à *DRAM*, substitui-se a função *accessDRAM()* pela função *accessL2()*.

```
if (!Line->Valid || Line->Tag != Tag) {        // if block not present - miss
   accessL2(L2Address, TempBlock, MODE_READ); // get new block from L2
```

Agora, o acesso à *DRAM* através da função *accessDRAM()* passa para dentro da função *accessL2()*.

Terceira parte. Modificação da cache L2 para ter um mapeamento associativo de 2 vias.

i) Acrescentou-se um parâmetro *Time* à estrutura *CacheLine*, necessário para implementar a política de substituição *LRU (Least Recently Used*):

```
typedef struct CacheLine {
  (...)
  uint32_t Time;              // Marca o tempo de acesso para LRU
} CacheLine;
```

ii) Criou-se uma estrutura de dados ***Set***, que guarda num vetor as duas 2 linhas de um set da cache 2-way.

```
typedef struct Set {
  CacheLine lines[WAYS];
} Set;
```

iii) Alterou-se a estrutura de dados ***CacheL2*** para passar a agrupar todas as linhas da cache em sets, guardados num vetor com tamanho metade do número de linhas total.

```
typedef struct CacheL2 {
  Set set[L2_LINES/WAYS];      // Conjuntos para a cache L2
} CacheL2;
```

iv) Implementou-se ainda a função ***calculateLRU(Set *set)*** que recebe um pointer para um set, calcula o maior tempo de entre os 2 blocos e retorna um pointer para essa linha.

Conclusão

O simulador de cache desenvolvido demonstrou um comportamento correto em todos os testes executados. A estrutura do código permite ajustar facilmente as constantes definidas em **Cache.h** para explorar diferentes configurações de cache, o que contribui para um melhor entendimento das dinâmicas de acesso à memória em arquiteturas de cache de vários níveis. A experiência adquirida durante o projeto reforçou a compreensão dos conceitos teóricos de hierarquias de memória.