Organização de Computadores 2023
Relatório do Laboratório 2 – Group 040
(ist1103335, ist1103573, ist1100313)

4.1

```
void accessL1(uint32_t address, uint8_t *data, uint32_t mode)
```

Variáveis e Cálculos de Endereço:

- **'Tag'**, **'index'** e **'offset'** são calculados a partir do endereço fornecido. Estes valores são usados para determinar a localização do bloco de dados na cache.
- **'TempBlock'** é um Buffer temporário usado para trocar blocos entre a memória principal e a cache.
- **'Line'** é um ponteiro para a linha na cache que corresponde ao índice calculado.

Verificação de Miss:

- Esta condição verifica se a linha na cache (**'Line'**) é válida e se o campo de Tag coincide com o Tag calculado para o endereço recebido. Isso indica se o bloco de dados está presente na cache ou não. Se esta condição for verdadeira, significa que houve um miss.

Miss:

- Se houver um miss, isso significa que o bloco de dados não está presente na cache. Neste caso, é necessário ir buscar o bloco à memória principal (**'accessDRAM'**) e trazê-lo para a cache (**'TempBlock'**).
- Se a linha na cache já contém dados (indicado por **'Line->Valid'**) e esses dados foram modificados (**'Line->Dirty'**), então os dados antigos precisam ser escritos de volta na memória principal (**'accessDRAM'** com **'MODE_WRITE'**).
- Em seguida, os dados do **'TempBlock'** são copiados para a linha na cache (**'memcpy'**).
- A linha é marcada como válida (**'Line->Valid = 1'**), o Tag é atualizado (**'Line->Tag = Tag'**) e o bit de "Dirty" é reposto a 0 (**'Line->Dirty = 0'**).

Leitura e Escrita de Dados:

- Se o modo for leitura (**'MODE_READ'**), os dados são copiados da cache para a variável **'data'**. O tempo (**'time'**) é atualizado de acordo com o tempo de leitura da cache.
- Se o modo for escrita (**'MODE_WRITE'**), os dados são copiados de **'data'** para a cache. O tempo (**'time'**) é atualizado de acordo com o tempo de escrita da cache e o bit "Dirty" é marcado indicando que a linha na cache foi modificada.

Retorno de Miss:

- No final, a função retorna **'miss'**, que indica se houve um miss (1) ou não (0).

4.2

```
void accessL2(uint32_t address, uint8_t *data, uint32_t mode)
```

- A Cache L2 é definida da mesma forma da cache L1 por isso a explicação desta função é análoga ao 'accessL1' no 4.1.

  ```
  void accessL2(uint32_t address, uint8_t *data, uint32_t mode)
  ```

- A maior diferença passa pelo caso de um miss na cache L1. Aqui acessamos à cache L2 em vez de aceder à memória principal.

4.3

`void accessL2(uint32_t address, uint8_t *data, uint32_t mode)`

- A Cache L2 passa a ter um array de blocos em cada index da cache, cada um deles associado a um **'LRU_idx'**.
- O **'LRU_idx'** é um inteiro que representa a ordem temporal de acesso ao bloco em relação aos restantes na mesma linha. Assim, o valor 0 corresponde sempre ao **LRU**.

  `void use_block(CacheLine *line, int block_idx)`

  > Esta função coloca o respetivo bloco como o mais recente, ou seja, define o valor de **'LRU_idx'** igual ao maior tamanho possível. Decrementa ainda o valor de ''**LRU_idx'** dos mais recentes a esse para corresponderem a uma ordem mais antiga.

  `int get_LRU_block(CacheLine *line)`

  > Devolve o índice do bloco atual com 'LRU_index' igual a 0.

- Nesta versão, à semelhança das anteriores verificamos se a Tag do address está presente na linha de cache mas, agora temos de efetuar um loop porque as linhas estão divididas num número de blocos correspondente às vias de associatividade. Ao mesmo tempo, se nenhuma tag corresponder, também tenta encontrar um bloco vazio (bit de valid igual a zero).
- Quando ocorre um miss (não encontra a tag na linha), vamos buscar o bloco à memória principal e colocamo-lo no bloco vazio encontrado ou, em caso de todos estarem ocupados, no bloco correspondente ao LRU (LRU_idx igual a zero).
- Depois, colocamos este bloco como o último LRU e fazemos a operação requisitada (ler ou escrever)