4.1. L1 Cache com Mapeamento Direto

Para a implementação da cache L1, começamos por adicionar `L1_LINE_COUNT` (256) linhas à struct da Cache. Na inicialização da cache L1, em vez de apenas colocar o valor de Valid a 0 e init a 1, também colocamos o valor de Dirty e de Tag a 0 para cada CacheLine e inicializamos as linhas colocando as palavras com valor 0.

Para acomodar mais linhas, adicionamos uma variável para o offset e alteramos o valor de index e tag. Uma vez que cada bloco possui 16 palavras de 4 Bytes, o offset tem $\log_2(4*16)$ = 6 Bits, o Index passa a ter $\log_2(L1\_LINE\_COUNT)$ = 8 bits e os restantes bits (32-14 = 18 bits) passam para a tag.

De seguida, em vez de calcularmos apenas o valor da Tag e address do bloco em memória, também calculamos o offset (usando shift-left e shift-right de 32 - 6 = 26 bits) e o index (removendo o offset e aplicando AND com a mask 0xFF, pois o index tem 8 bits). O endereço do bloco na memória principal é o address dado após a anulação do offset. Assim, obtemos os parâmetros da linha a aceder.

No acesso à cache, utilizamos o index e o offset para saber onde ler/escrever e, em vez de apenas adicionar o tempo de acesso e alterar o valor Dirty (no caso de write), também colocamos o valor da tag igual à tag calculada e o valor de Valid a 1.

4.2. L2 Cache com Mapeamento Direto

Para a cache L2 com Mapeamento DIreto, utilizamos o mesmo método que o anterior, com algumas alterações:

Alteramos o tamanho em relação à cache L1: L2 possui `L2_LINE_COUNT` (512) linhas. Pois tem o dobro do tamanho da cache anterior, há um bit extra de index (9 bits), o que leva à alteração da a mask usada para o cálculo do index - 0xFF para 0x1FF. Com isto, a tag diminui 1 bit, ficando com 17 bits.

Também alterou-se o que aconteceria na cache L1 em caso de miss: a cache L1 agora acede à cache L2 e só em caso de miss na cache L2 é que se acede à memória primária.

Quanto ao acesso, funciona de forma igual à implementação anterior.

4.3. L2 Cache com 2 vias de associatividade

Para implementar a cache L2 com 2-way associativity, acrescentamos mais uma dimensão de tamanho 2 ao array de cache lines (para cada linha ter dois blocos) e adicionamos à estrutura CacheLine um atributo "Time" que guarda o tempo da última atualização do bloco

em questão, para nos permitir descobrir qual foi o bloco que foi usado menos recentemente (LRU). Este atributo é inicializado com o valor zero, junto dos outros atributos.

Como a cache agora tem 2 vias de associatividade, dividimos o index por 2, voltando assim a ter 8 bits. No acesso à cache, começamos por verificar se um dos dois blocos da cache line do index dado pelo address é o bloco que estamos à procura, guardando se é o primeiro ou segundo bloco.

Caso o bloco não seja encontrado nesta linha, verificamos os atributos "Time" dos dois blocos, guardando o índice do bloco que não é acedido à mais tempo. Depois, lemos da memória primária para um bloco temporário, escrevemos na memória primária o conteúdo do bloco de L2 se este era Valid e Dirty e de seguida lemos do bloco temporário para o bloco de L2. Por fim, atualizamos os atributos deste bloco (Valid = 1, Tag = tag, Dirty = 0 e Time = time) e, se o modo de acesso era Read, escrevemos em *data* o conteúdo do bloco L2, incrementando o tempo.

Caso o bloco seja encontrado, escrevemos ou lemos no respectivo bloco, atualizando o tempo de acesso ao bloco e os seus atributos (Valid = 1, Tag = tag e Time = time). Em caso de escrita colocamos Dirty igual a 1, colocando como 0 caso contrário.

Explicação dos acessos:

- `[index * BLOCK_SIZE + offset]`

Para aceder à L1 e à L2 de mapeamento direto, o index * BLOCK_SIZE indica-nos a linha do bloco que queremos aceder, somando depois o offset para mover dentro do bloco.

- `L2Cache[index * 2 * BLOCK_SIZE + (i * BLOCK_SIZE)]`

Para aceder à cache L2 com duas vias de associatividade, como cada linha tem dois blocos, o primeiro byte do bloco que queremos é index * 2 * BLOCK_SIZE. A variável 'i' é 0 ou 1, sendo utilizada para somar 0 se queremos o primeiro bloco ou BLOCK_SIZE se queremos o segundo.