# CACHE IMPLEMENTATION DETAILS

- **Phase 3 code is in the file sim_cache.cpp**
- **The cache parameters specifications with a sample input are present in CacheParameters.pdf**
- **cache.cpp has cache implementation only , this can be used to test the cache alone (use this file for easy reference of the  code)**

***This document gives details about the cache policies, brief description of code and their implementation. It also explains stalls that are caused by the changes made to the simulator with examples**

Two levels of cache with the following properties has been added to the pipeline simulator :

- Both levels follow **LRU** replacement policy
- **Non Inclusive policy** is implemented
- **write back , write allocate** for both levels
- No write buffer (so stores wait until the write operation is complete and hence may cause stalls like the loads due to the data access delays )

## A BRIEF EXPLANATION OF THE CODE

### class block

- It is an augmentation of **an integer ( word = 4 bytes) array** to store a block of words (size of a block is user defined), tag , age (stores the age of the block), dbit (dirty bit), vbit ( valid bit ) and methods to access and modify these variables.

### class set

- Contains an **array of blocks**( the size of this array is the associativity and is given by the user), and methods to set parameters

- Important Methods :
    - **searchTag** : searches for the block with given tag
    - **getFreeBlk**
    - **setage, getMaxAge, getLRU :** These three methods help identify the LRU block in a set

      LRU is implemented as follows
      Consider a set with blocks A B C D

*The following tables are not diagrammatic representations of sets. they just give the status of valid bits and age of the blocks

Initially...cache empty

| | A | B | C | D |
|---|---|---|---|---|
| vbit | 0 | 0 | 0 | 0 |
| age | N/A | N/A | N/A | N/A |

A is asked

| | A | B | C | D |
|---|---|---|---|---|
| vbit | 1 | 0 | 0 | 0 |
| age | 0 | N/A | N/A | N/A |

D accessed

| | A | B | C | D |
|---|---|---|---|---|
| vbit | 1 | 0 | 0 | 1 |
| age | 1 | N/A | N/A | 0 |

B accessed

| | A | B | C | D |
|---|---|---|---|---|
| vbit | 1 | 1 | 0 | 1 |
| age | 2 | 0 | N/A | 1 |

D accessed

| | A | B | C | D |
|---|---|---|---|---|
| vbit | 1 | 1 | 0 | 1 |
| age | 3 | 1 | N/A | 0 |

Now if asked for LRU, A is returned

*As age is an integer the upper limit to it is INT_MAX

## class Cache

- Methods :
    - **setCacheParam**
    - **find :** finds a block by resolving addresses into tag,index and offset
    - **Insert :** gets tag,index,offset for the block to be inserted Goes to the appropriate set using index and searches for a free block, if free block is not available Uses LRU and chooses a block for eviction, Now the chosen block is filled with the incoming block contents and the tag,dirty bit valid bit ,age are updated accordingly
    - **update :** used only when a block which is evicted from a cache level above to it is dirty and present in the current level of cache, that block contents are updated in the current level

## class CacheCtrl

contains methods

to check and set the cache parameters

to manage eviction of blocks

- evictToNext ( method for an intermediate cache level )
- evictToMem( method for the last level of cache)
- updateMem (to write values into Main memory)

to implement reads and writes

- read ( address is given as an input )
- write ( input : value to be written and destination address)

Using all the above mentioned methods cache is organised by the CacheCtrl

**CACHE ORGANISATION (Non inclusivity, write back and write allocate)**

For 2 levels of cache, cache management is done by cache controller

Any request goes to L1 cache first

1. Hit - read or write ( set age, incase of a write dirty bit is set )
2. Miss - go to L2 cache

In L2 cache

1. Hit
    - read or write( set age, incase of a write dirty bit is set )
    - **copy the block** (in case of write, block is copied after write (write allocate)) **into** a Free block or LRU block (if free block is not  available) in **L1**
    - **A copy of the block still remains in L2**
    - Now for the copied block in L1
        - update tag
        - incase of write allocation it is enough to set the dirty bit in L2 cache, no need in L1 cache.

        - For any block valid bit needs to be set only when the block is being used for the first time
        - Age of the block is set (it is made MRU) only when processor accesses the block not when the cache controller accesses it , So here age remains same
2. Miss - go to Main memory

**On Miss** in both the levels the block is brought from the main memory( if it is a write request first a write is performed in the main memory itself and then the block is brought(write allocate) , so no need to set the dirty bit) and **allocated in both L1 and L2**. And the block parameters tag, age , dbit and vbit are updated accordingly

**When a block is being evicted from the L1 cache :** Write back policy
If it is not dirty nothing is done
If it is dirty L2 is checked for the evicted block. If L2 has it, then its contents are updated with evicted blocks contents and the L2 cache block is set as dirty. else updation of contents is done directly in main memory

**When the block is being evicted from L2 cache :** Write back policy
Updation of contents is done in main memory incase of a dirty block

When the process ends both cache levels are cleared and if any block is dirty it is updated in the main memory

## INCORPORATING CACHE INTO THE SIMULATOR

In the previous version of simulator all the 5 MIPS pipeline stages are assumed to be taking 1 cycle to complete, which is actually not true for the

MEM stage , memory access without cache takes 10s to 100s of cycles depending on the speed of the memory. This leads to extra stalls in the pipeline
In this version this the memory access time is considered and cache is included so that at least some stalls can be avoided by speeding up the data service

**L1 cache latency, L2 cache latency and Memory latency are understood as follows :**

L1 cache latency : L1 hit time
L2 cache latency : L1 miss + L2 hit time
Memory latency : L1 miss + L2 miss + memory access time

All the overhead caused by evictions, allocations (whatever happens to manage the cache except servicing the data to the process) are ignored

**STALLS are explained with some examples as follows :**

1 stall = 1 cycle (same convention as in the previous version of the simulator )

L1 access latency  =  2 cycles
L2 access latency  =  4 cycles
Memory access latency = 10 cycles

The above values are used for examples.

For any load/store :

On a L1 cache hit the pipeline stalls by 1 cycle (i.e 1 stall)
    L2 cache hit the pipeline stalls by 2 cycles ( 2 stalls)
    Both miss 9 cycles (9 stalls)

on the top of any other stalls caused by the hazards

Following are few examples :

Example 1 :

lw $t1,0($s0)        // L2 hit

3 STALLS (access delay)
1 STALL (data hazard)

add $t2,$t1,$t3    // no memory access so MEM stage completes in 1 Cycle

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 |
|---|---|---|---|---|---|---|---|---|---|---|
| IF | ID/RF | EX | MEM | | | | WB | | | |
| | STALL | STALL | STALL | STALL | IF | ID/RF | EX | MEM | WB | |

Example 2 :

lw $t1,0($s0)       // L1 hit
1 STALL (delayed data access)
add $t2,$t3,$t4   // no memory access
1 STALL ( RAW on the registers indicated)
beq $t1,$0, target  // no memory access
1 STALL
*branch target and decision both made in ID/R.  reason for the stall due to data hazard in this case is explained in phase 2 readme

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 |
|---|---|---|---|---|---|---|---|---|
| IF | ID/RF | EX | MEM | | WB | | | |
| | STALL | IF | ID | EX | MEM | WB | | |
| | | | STALL | IF | ID/RF | EX | MEM | WB |

Example 3 :

lw $t1,0($s0) // Memory access
9 STALLS
2 STALLS ( data hazard)
beq $t1,$0, target

| C1 | C2 | C3 | 9 cycles | C13 | C14 | C15 | C16 | C17 |
|---|---|---|---|---|---|---|---|---|
| IF | ID/RF | EX | MEM | | WB | | | |
| | STALL | STALL | 9 STALLS | IF | ID/RF | EX | MEM | WB |

Example 4:

sw $t1,0($s0) // L2 hit
3 STALLS
add $t2,$t1,$t2

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 |
|---|---|---|---|---|---|---|---|---|
| IF | ID/RF | EX | MEM | | | | WB | |
| | STALL | STALL | STALL | IF | ID/RF | EX | MEM | WB |

*Two minor changes have been done to the previous version of simulator
And uploaded as pipeline2.cpp. Changes are indicated with “// changed” at
that line in the code.