Lab-7 (RISC-V Cache Simulator)

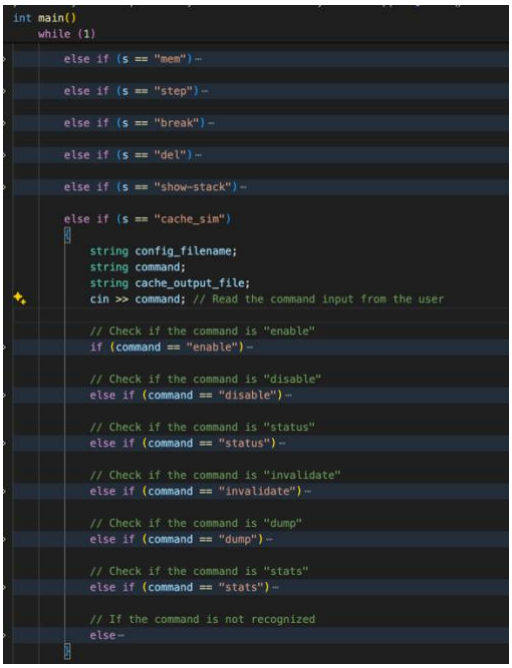
1. Implementation:

- I have implemented all three parts.
- I have created 5 vectors;

  valid : to store if the cache line is occupied or not.
  cache: to store contents of all cache lines.
  tags   : to store reference to memory in cache lines.
  time : to store timestamp of cache lines.
  clean : to know if a given cache line is in sync with memory.

```
vector<bool> valid;
vector<vector<char>> cache;
vector<int> tags;
vector<int> Time;
vector<bool> clean;
```

- Apart from the existing simulator menu for the user , I have added one more submenu to it ; the CACHE_SIM submenu.



- cache_sim enable: Enables cache simulation for D-cache within the simulator.
- cache_sim disable: Disables cache simulation.
- cache_sim status: prints the enable/disable status of the D-cache simulation. If cache simulation is enabled, it also prints the cache configuration values.
- cache_sim invalidate: Invalidates all entries of the D-cache.
- cache_sim dump: writes all the current D-cache entries to a file.
- cache_sim stats: prints the cache statistics for the executing code.

2. Working of Cache (READ):

i. Direct-Mapped Cache

For a **direct-mapped** cache (associativity = 1):

- **Cache Miss:**
  - If the tag does not match or the line is invalid, it:
    - Checks if the current line is dirty, and if so, writes it back to memory.
    - Loads a new block from memory into the cache.
    - Updates metadata (valid bit, tag, clean bit).

- **Cache Hit:**
  - Increments the hit counter and proceeds to the next block.

ii. Fully Associative Cache

For a **fully associative** cache (associativity = 0):

- Searches through the entire cache for a matching tag.
- **Cache Hit:**

- Increments the hit counter.
- Updates the access time for LRU if needed.

- **Cache Miss:**
  - Searches for an empty cache line or uses the replacement policy (FIFO, LRU, RANDOM) to replace a line.
  - Writes back the dirty line if needed and loads a new block from memory.

Replacement Policies:

- **FIFO/LRU:**
  - Finds the least recently used (LRU) or first-in line based on Time[].
- **RANDOM:**
  - Selects a random cache line for replacement.

iii. Set-Associative Cache

For a **set-associative** cache:

- **Cache Hit:**
  - Checks each line in the set for a matching tag.
  - Updates LRU time if required.
- **Cache Miss:**
  - Searches for an empty line in the set.
  - Uses the replacement policy (FIFO, LRU, RANDOM) to replace a line if no empty line is found.
  - Writes back the dirty line and loads a new block from memory.

Replacement Policies:

- **FIFO/LRU:** Chooses the line with the smallest access time.

- **RANDOM:** Selects a random line within the set.

3. Working of Cache (WRITE):

a. Cache Write Policies

The function supports two main cache write policies:

- **Write-Through (WT)**: Data is written directly to both the cache and the main memory on a write hit.

- **Write-Back (WB)**: Data is only written to the cache. The main memory is updated only when the cache line is evicted (i.e., when a dirty line is replaced).

b. Write-Through Policy (WT)

In this policy:

- The function first checks if the current cache line is valid and matches the tag.

- If a match is found (**write hit**):
    - It increments the hit count (hits).
    - Then the cache line is updated.

- If no match is found (**write miss**):
    - It generates output indicating a miss but does not update the cache.

- **Direct-Mapped Cache**:
  - Calculates the cache line index (level) and block tag.
  - Checks the tag for a hit or miss and proceeds accordingly.

- **Fully Associative Cache**:
  - Scans the entire cache for a matching tag.
  - If a match is found, updates the cache line.
  - Otherwise, outputs a miss without modifying the cache.

- **Set-Associative Cache**:
  - Computes the set index (set) and searches within the corresponding set.
  - If a match is found, it updates the cache line.
  - Otherwise, outputs a miss without modifying the cache.

c. **Write-Back Policy (WB)**

In this policy:

- The function checks if the cache line is valid and matches the tag.
- On a **write hit**:
  - Marks the cache line as dirty.
  - Updates the cache line with the new data.
  - If the tag does not match and the line is dirty, it writes back the dirty line to memory.
- On a **write miss**:
  - Updates the cache line with the new data, marks it as valid, and updates the tag.

- **Direct-Mapped Cache**:
  - Calculates the cache line index and tag.
  - If the line is valid and dirty but the tag does not match, it writes the dirty block back to memory before updating the cache.
  - Updates the cache with the new data.
- **Fully Associative Cache**:
  - Scans all lines for a matching tag.
  - If no match is found, it searches for an invalid line or evicts a line using the specified replacement policy (FIFO, LRU, RANDOM).
  - Updates the cache and memory as necessary, marking lines as dirty when modified.
- **Set-Associative Cache**:
  - Searches within the set for a matching tag.
  - If no match is found, it evicts a line using the specified replacement policy.
  - Writes the new data into the cache and updates metadata.

d. Replacement Policies

The function supports three cache replacement policies when eviction is necessary:

- **FIFO (First-In-First-Out)**: Replaces the line that has been in the cache the longest.
- **LRU (Least Recently Used)**: Replaces the least recently used line, using a Time array to track access time.
- **RANDOM**: Chooses a random cache line to replace.

4. Testing of my code:

   To test my code , I changed the block size, cache size , associativity , replacement policies and writeback policies in the sample testcases given to us.

```
.data
.dword 10, 20, 30, 40, 50
.text
lui x3, 0x10
ld x4, 0(x3)
ld x4, 8(x3)
ld x4, 16(x3)
ld x4, 24(x3)
ld x4, 32(x3)

256
16
0
FIFO
WT
```

```
.data
.dword 20, 30, 40, 50, 60
.text
lui x3, 0x10
sd x0, 0(x3)
ld x4, 8(x3)
ld x4, 0(x3)
ld x4, 16(x3)
ld x4, 24(x3)

256
16
8
FIFO
WB
```