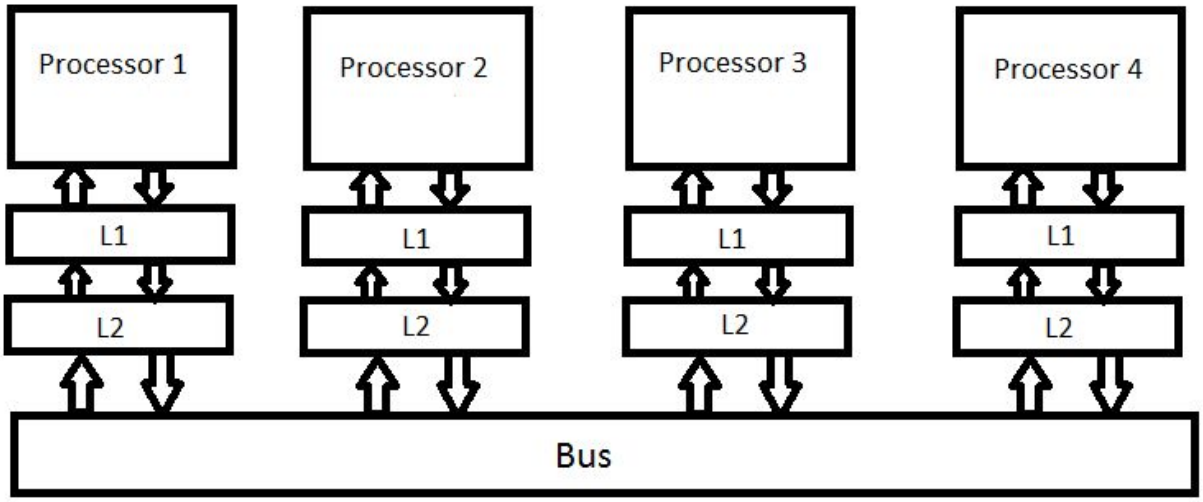
ECE 585
Final Project
Simulation of
Last level Cache

By:
Amit Solapurkar
Kalyani Chawak
Soma Sai Charitha Yenuga
Haranadh Chintapalli

Portland State
UNIVERSITY

1. Block Diagram



2. Assumptions

The assumptions are as follows:

- For the design purposes, L2 cache is assumed to be the last level cache and L1 cache is assumed to be the upper level cache.
- LRU bits of a empty/evicted lines are assumed to be 7.
- The MESI state for evicted/empty lines is assumed to be Invalid.
- After sending a message to L1 cache line about eviction, it is assumed that L1 can evict the line (considering L2 has 64 bytes cache line and L1 has 32 bytes cache line)
- All the addresses are assumed to be 32 bits long only.

3. Design

All the functions are declared in a header file called 'cache.h'. The arguments are taken from command line (name of trace file and define statements). Makefile is written for compiling and simulating the source code. An example of address is as follows:

Address in hex format: 85124940
In binary format: 1000 0101 0001 0010 0100 1001 0100 0000
Tag bits : 1000 0101 000 (11 bits)
Index bits : 1 0010 0100 1001 01 (15 bits)
Byte select : 00 0000 (6 bits)

Cases are taken for different operands and the functionality for them is written accordingly. Functionalities of individual functions are explained below:

1) Least Recently Used (LRU)

- This function is used to evict the least recently used cache line in a set. LRU bits are maintained for each cache line in a set (values 0 to 7).

2) MESI protocol

- When CPU read request is sent, L1 level cache encounters a MISS and request is sent to L2 cache. The L2 level cache checks for the requested address, if present , the cache would generate Cache HIT (if tag matches) else Cache MISS is generated. When it is a MISS, number of Cache Misses gets incremented.Then bus operation occurs by sending address and read request. Snooped response of NOHIT, HIT and HITM occurs from other caches. If snooped response is a HIT or HITM, LRU and MESI bits are updated and state is changed to Shared. If it is a NOHIT, the address is fetched from memory and state is changed to Exclusive.

- When CPU WRITE request occurs, the number of CPU writes is incremented. If Cache write HIT occurs, increment the number of Cache hits, update LRU bits and bus operation is Invalid. If Cache MISS occurs, bus operation is Read for Ownership (RFO). If the snooped response is NOHIT, update LRU bits and change MESI state to Modified. If it is HIT/HITM, update LRU bits, and change MESI state to Modified.

- When CPU read request occurs from L1 instruction cache, the operation is same as read request to L1 data cache.

- When snoop invalid request occurs , first check if that address is present and then invalidate the same. Change the MESI state to Invalid.
- When CPU snoop read request occurs and cache line is present in invalid state, then NOHIT is issued. Else if present in Shared state, HIT is issued. If present in Exclusive state, HIT is issued and MESI state is updated to shared. If present in Modified state, HITM is issued and bus operation of Write Back is performed and moved to Shared state.
- When Snoop write request is issued , no operations are performed on modelled cache, since other caches are Modified. snoop write occurs only if RFO and eviction occurs.
- When snoop RFO occurs, and L2 cache is in Invalid state, it would issue NOHIT. If present in Shared or Exclusive state, state is changed to Invalid state. If present in Modified state, HITM is issued and bus operation is Write Back is issued and state is changed to Invalid.

3) Getting Snoop Result

- The last 4 bits of the tag are taken and compared to hexadecimal numbers. For numbers 0 to 5, HIT is returned. HITM is returned for numbers 6 to 9. NOHIT is returned for numbers A-F.

4) Getting tag and index bits

- The tag and index bits are obtained by left shifting the address bits by 21 and 6 respectively.

5) Cache Performance Statistics

- The number of CPU Reads, CPU Writes, Cache HITs and Cache MISSes are incremented depending on the Operands and Bus Operations.
- Hit Ratio is the ratio of number of HITs to total number of HITs and MISSes.

4. Testing

Testing has been done for the functionalities. The test cases are as follows:

1) Basic cache code Functionality

- The counters of Number of Reads, Writes, Hits, Misses must be set to zero, the LRU bit must be set to to 0x7, tag bits set to zero, MESI state is set to Invalid for resetting/initialising the cache.
- The cache size must be evaluated by providing 256K addresses. The trace files are generated using a separate C code.
- The cache line size must be evaluated by accessing the 64 bytes. The trace files are generated by using a separate C code.
- The set associativity must be checked by accessing the 8 lines in a set. They must be accessed without eviction.
- If any other operand is given other than the valid operands as input, error output should be generated.
- For maintaining inclusivity, the L2 cache must send a message to L1 cache regarding evictions.
- A message should be generated for the Write Back policy.
- If operand 9 is given as an input, the state of only the valid lines must be printed.
- If no trace file is given as an input, error output should be generated.
- The code must be able to handle spaces and tabs in the trace file.

2) LRU policy

- Test cases were written for checking whether the LRU function is being called and all the bits are initialised to 7.
- After a particular line x is accessed, the LRU bits for that line should change to 0 (MRU).
- The lines having LRU bits less than those of line x should increment. The bits greater than those of line x should remain as is.

- If the LRU bits for line x are 7, then the line should be evicted at the next cache MISS.

3) MESI

- The MESI function is accessed when it is called from the main function.
- Addresses are generated and checked as per the functionalities mentioned above in the design.
- Check the subsequent trace activity and tabulate the result.

5. Expected Test Results

Expected test results for MESI trace file are as follows:

CPU Read	Address	Snooped Response	MESI State	BusOp	Hit/Miss
0	85124940	HITM	SHARED	READ	MISS
0	85124940	NA	SHARED	NA	HIT
1	85924940	HITM	MODIFIED	RFO	MISS
0	85B24940	NOHIT	EXCLUSIVE	READ	MISS
2	85A24940	NOHIT	EXCLUSIVE	READ	MISS
1	85A24940	NA	MODIFIED	NA	HIT
3	85A24940	NOHIT	INVALIDATE	NA	
4	85A24940	NA	INVALIDATE	NA	
5	85B24940	NA	EXCLUSIVE	NA	
1	85A24940	NA	MODIFIED	RFO	MISS
4	85A24940	HITM	SHARED	WRITEBACK(PRINT)	
2	85A24940	X	SHARED	X	HIT
6	84A24940	X	INVALIDATE	X	
0	84724940	HIT	SHARED	READ	MISS
1	85C24940	X	MODIFIED	INVALIDATE	HIT
0	85C24940	NOHIT	EXCLUSIVE	X	MISS
0	85C24940	X	EXCLUSIVE	X	HIT
1	85C24940	X	MODIFIED	X	HIT
1	85C24940	X	MODIFIED	X	HIT
0	85C24940	X	MODIFIED	X	HIT
0	84324940	HIT	SHARED	READ	MISS
4	84324940	HIT	SHARED	X	
0	85D24940	NOHIT	EXCLUSIVE	READ	MISS
4	85D24940	HIT	SHARED	X	
0	84624940	HIT	SHARED	X	MISS
6	84624940	X	INVALIDATE	X	HIT
1	84624940	HIT	MODIFIED	RFO	MISS
6	84624940	HITM	INVALIDATE	WRITEBACK	

6. Readme

Run using a bash script

clean , build & run using "TRACE_FILE_NAME"
>> *source run.sh <TRACE_FILE_NAME>*

clean , build & run using "TRACE_FILE_NAME" and DEBUG defined
>> *source run_debug.sh <TRACE_FILE_NAME>*

clean , build & run using "TRACE_FILE_NAME" and SILENT defined
>> *source run_silent.sh <TRACE_FILE_NAME>*

Results are captured in <TRACE_FILE_NAME>_result_timestamp file

Manual compile and run options

To remove previously created object files
>> *make clean*

To build the project
>> *make*

To build the project with define SILENT(no BusOp prints)
>> *make CFLAGS=-DSILENT*

To build the project with define DEBUG (to get the all debug prints)
>> *make CFLAGS=-DDEBUG*

To run the L2 cache functionality using "TRACE_FILE_NAME" file as input file
>> *./cachebuild <TRACE_FILE_NAME>*

To capture the result in a file
>> *./cachebuild <TRACE_FILE_NAME> > result_file*

Trace file generation options

Generates trace file containing all sequential addresses
>> ./gen_trace_all_addr.sh

Generates trace file to test eviction
>> ./gen_trace_evict.sh

Generates trace file to check LRU bits updation
>> ./gen_trace_lru_test.sh