ECE 585
WINTER 2017
LAST LEVEL CACHE SIMULATOR
PROJECT REPORT

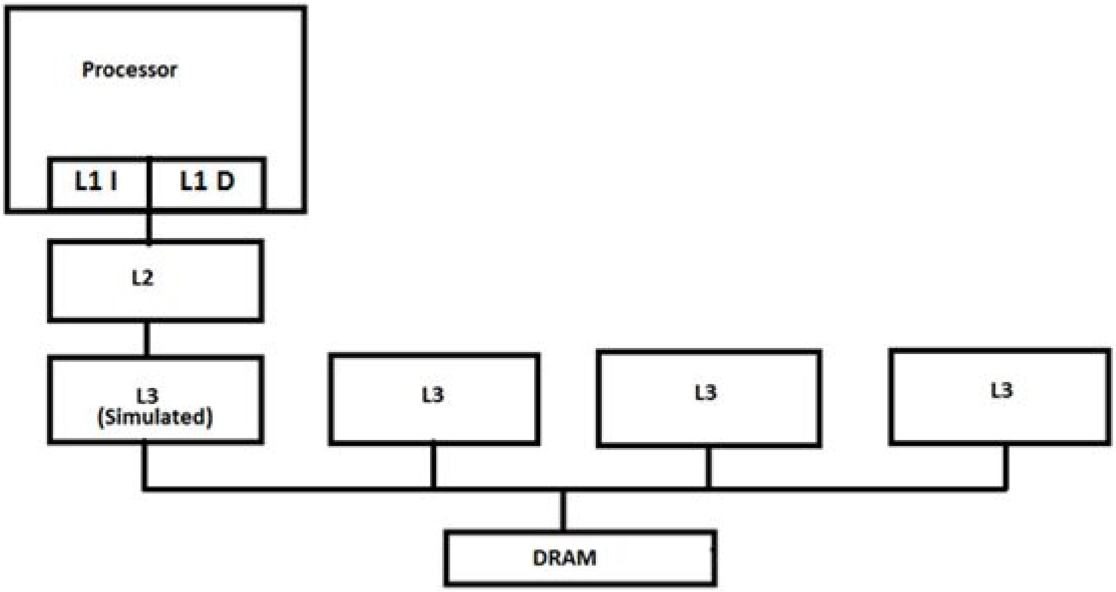
Amit Thobbi, Chandar Baskaran, Nikhil Naik, Shreyas Sankaran

Design Specification

The objective is to simulate the behavior of a unified Last Level Cache for a 32 bit processor. The cache can be used with three other processors in a shared memory configuration that communicates via a shared Front Side Bus (FSB). MESI protocol is used to ensure cache coherence. The simulated Last level cache has a total capacity of 16MB, uses 64 byte lines, and is 8 way set associative. It uses a write allocate and true Least Recently Used (LRU) replacement policy which is implemented using counter method. The last level cache maintains inclusivity with higher level caches. The actual data transfers are not simulated in this design.

The simulated Last level cache is backed by a DRAM based memory subsystem that reads and writes 64 bytes at a time.

Assuming the last-level cache is L3, a system block diagram is shown below:



Last Level Cache System Diagram

Next higher level cache Details

The processor's next higher level cache uses 32 byte lines and is 4-way set associative. It has split instruction and data caches. This cache is not modeled in the design.
Assumptions:

- The number of sets in this level and all higher levels are such that inclusivity is possible in the system.
- On an eviction of cache line in last level cache, a message is sent to next higher level cache. Since last level cache line size is 64 bytes and next higher level cache line size is 32 bytes, for one eviction in last level cache, multiple evictions are possible in higher level cache. The multiple evictions, if any, based on message from Last level cache are taken care by the higher level cache.

Last-level Cache Details

Cache Events:

The last-level cache, apart from servicing the requests from its own processor, it also snoops the operations of the other last -level caches in the system and responds to their requests.
All events to be handled by the last-level cache are listed below:

- CPU Data read
- CPU Data Write
- CPU Instruction read
- Snooped Invalidate
- Snooped Read
- Snooped Write
- Snooped Read with Intent to Modify (RWIM)
- Clear cache and reset all states
- Print cache contents and state of valid lines

The simulator tracks the cache hit and cache miss counts for all the requests from its own processor. For all the snoop requests, the last-level cache provides a snoop result depending on availability of the cache line and the state of cache line.

The operations to be performed for each of the events is shown below.

CPU Data Read:

- Check in cache.
- If hit, provide data to CPU.
- If miss, check if other cache have the data. If data is sourced by other cache, mark the state as SHARED. If none, of other cache have data, perform the bus operation from memory and mark the state and EXCLUSIVE.

CPU Data Write:

- Check in cache.
- If cache hit, check the state of line. If SHARED, indicate other caches to invalidate their copies. If EXCLUSIVE or MODIFIED, mark it as MODIFIED.
- If cache miss, perform bus operation of RWIM.

CPU Instruction Read:

- Steps are same as for CPU Data read.

Snooped Invalidate:

- If cache hit, mark the cache line as INVALID and send a message to next higher level cache to invalidate its copies.

Snooped Read:

- If cache hit and in modified state, put snoop result of HITM and writeback to memory and mark cache line to SHARED.
- If cache hit, put the snoop result of HIT and mark the cache line as SHARED.
- If cache miss, put snoop result of NO HIT.

Snooped Write:

- No actions to be performed on snooping write request.

Snooped RWIM:

- If cache hit, put the snoop result of HIT/HITM depending on cache state line.In case of HITM writeback to memory.Then Invalidate the state of cache line and send a message to higher level cache too.
- If cache miss, put snoop result of NO HIT.

Clear cache and reset all states:

- Clear the cache hit and miss counts.
- Set all the cache line states to INVALID.

Print cache content and state of valid lines:

- Print Tag and state of all valid lines.

Other actions:

- Maintaining a sequence for LRU cache way for each set. Cache line from least recently used way is evicted to make place for a new cache line.
- All evictions are to be reported to the next higher level cache, so that the cache line can be evicted from the higher level cache as well. This is required to maintain inclusivity.

Assumptions:

- On bus operation of RWIM or CPU read, if other last level caches have the same line in Modified state, it is expected that the other cache will write-back the line to memory.
- In case of snooped RWIM, if snooping cache has the same line in Exclusive or Shared state, it is expected to source the line and invalidate its copy.

Last-level Cache Simulator Code Design

Addressing:

Address width: 32-bit

Number of Sets: 32K

Ways per set: 8

Cache line Size: 64 bytes

32-bit address from the CPU is split as below:

Tag (11 bits)	Index (15 bits)	Byte-Offset (6 bits)

Parameters in the code:

Parameter	Description	Configurable	Value
TOTAL_CACHE_SIZE	Specifies the total size of the cache	Yes	Defined (16777216)
CACHE_LINE_SIZE	specifies the size of the cache line in bytes	Yes	Defined (64)
WAYS_PER_SET	specifies the number of ways	Yes	Defined (8)
NO_OF_SETS	Number of Cache Sets	No	Calculated
BYTE_BITS	Number of bits of 32-bit address allocated for byte offset	No	Calculated
INDEX_BITS	Number of bits of 32-bit address allocated for index	No	Calculated
TAG_BITS	Number of bits of 32-bit address allocated for tag	No	Calculated

Data Structures:

For each cache line 11 bit Tag and MESI Coherence state is to be stored. The data structure can be defined as:

```
typedef struct info

{

        unsigned int tag;

        unsigned int state : 2;

}cache_info_t;

cache_info_t cache[NO_OF_SETS][WAYS_PER_SET];
```

Similarly LRU counter values for each cache line is to be maintained. The data structure can be defined as:

unsigned int lru_sequence[NO_OF_SETS][WAYS_PER_SET];

Code Modules:

The code can be broadly classified into 4 modules:

1. Trace decoder: Reads the trace file line by line and decodes (converts from ascii to hex value) the event and address. Based on the event calls the appropriate event handler.
2. Cache Operations: There are 8 different events which the cache needs to respond to. They are listed in the table below. Once common operation required for most of these events is to first check the availability of data in cache and determine if it is a cache hit or cache miss. MESI states are handled in this module too.
3. Notification operations: Perform bus operation or snooping or put snoop result or send a message to higher level cache
4. LRU policy: Maintains LRU counters for ways in a set. Promotes a way to LRU position or MRU position. Returns LRU way for a set when required.

The functions required in each of these modules are listed in the table below:

Module	Function	Description
Trace Decoder	AsciiToHex	Converts the ascii string to hexadecimal value.
	HandleTraceCommand	Calls appropriate functions for all the trace events
Cache Operations	CPUReadData	Handles CPU Data Read request
	CPUWriteData	Handles CPU Data Write request
	CPUReadInstruction	Handles CPU Instruction Read request
	SnoopedInvalidate	Handles Snooped invalidate request
	SnoopedRead	Handles Snooped Read request
	SnoopedWrite	Handles Snooped Write request
	SnoopedReadModify	Handles Snooped RWIM request
	ClearCacheAndReset	Clear Cache counts and resets the states of all lines
	PrintCacheInfo	Prints the information of all valid lines in the cache.
	CheckInL3	Used to check whether the data for requested address is present in cache.
	UpdateCacheTag, UpdateCacheState	Update the cache line tag and state.
Notification Operations	GetSnoopResult	Gets the snoop result from other processors.
	BusOperation	Emulates bus operation.
	PutSnoopResult	Puts snoop result for snoop requests.

	MessageToL2Cache	used to send a message to L2
LRU	InitializeLRU	Initializes all the LRU counters for all the ways of each set.
	MakeLRU	Update the way as Least recently used (LRU).
	MakeMRU	Update the way as Most recently used (MRU).
	GetLRU	Returns the Least Recently Used (LRU) way.

Definition for GetSnoopResult function:

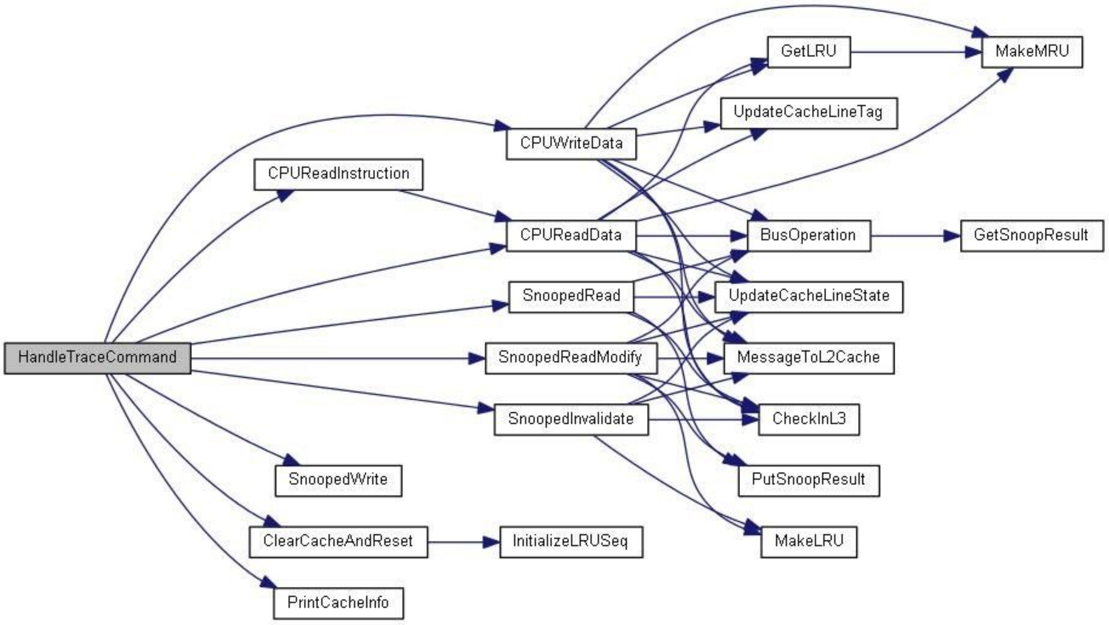
Bit 0 and Bit 1 of the Tag part of the address are evaluated to generate the snoop result. The mapping is as follows:

Value of (Tag Bit 1,Bit 0)	Result
0	HIT
1	HIT
2	HITM
3	NO HIT

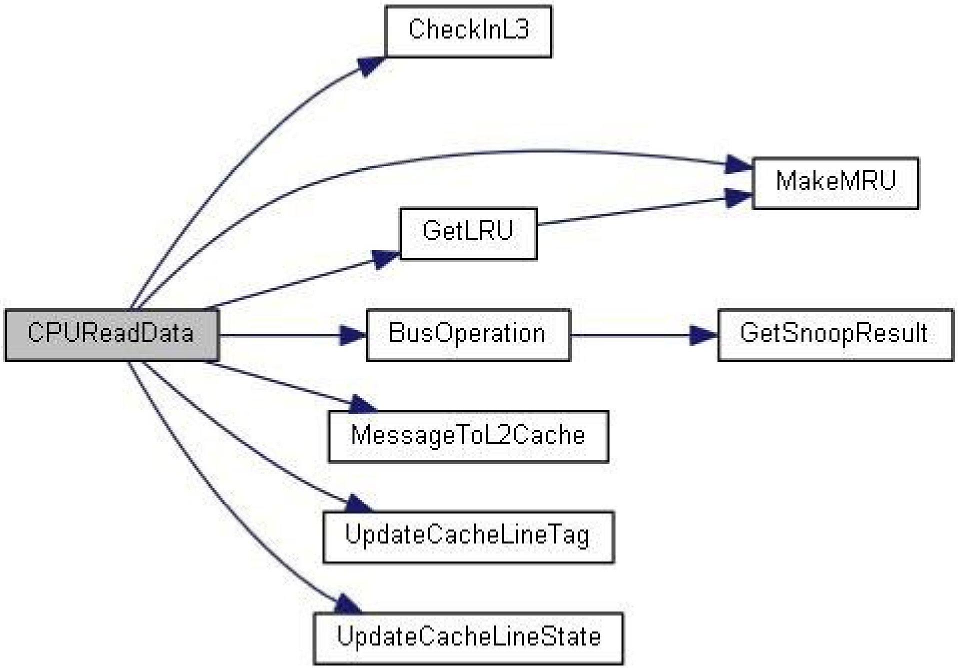
Code Flow:

Doxygen generated code flow snippets are shown below:

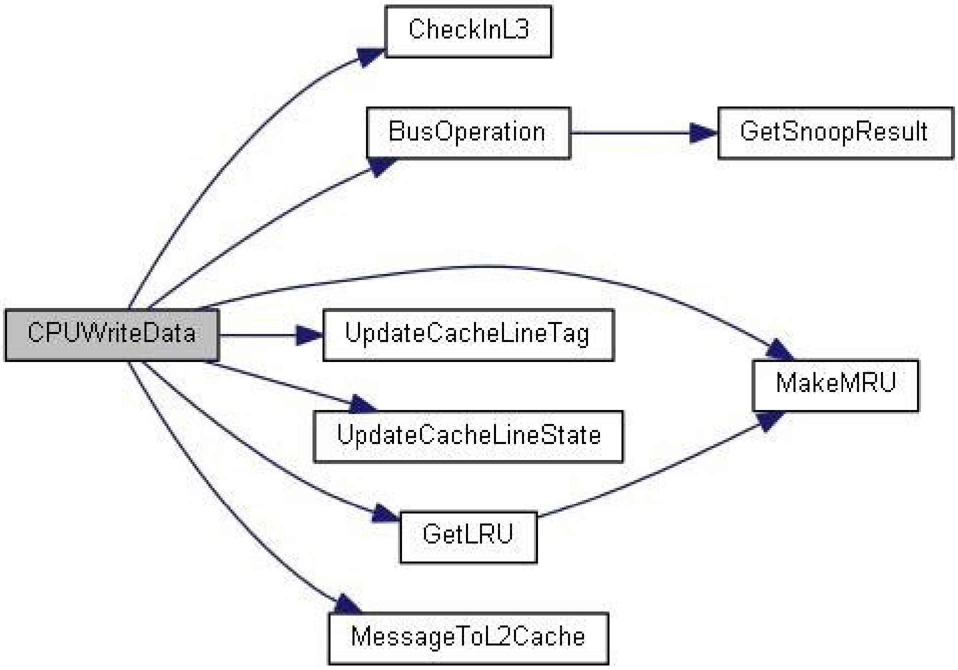
HandleTraceCommand:



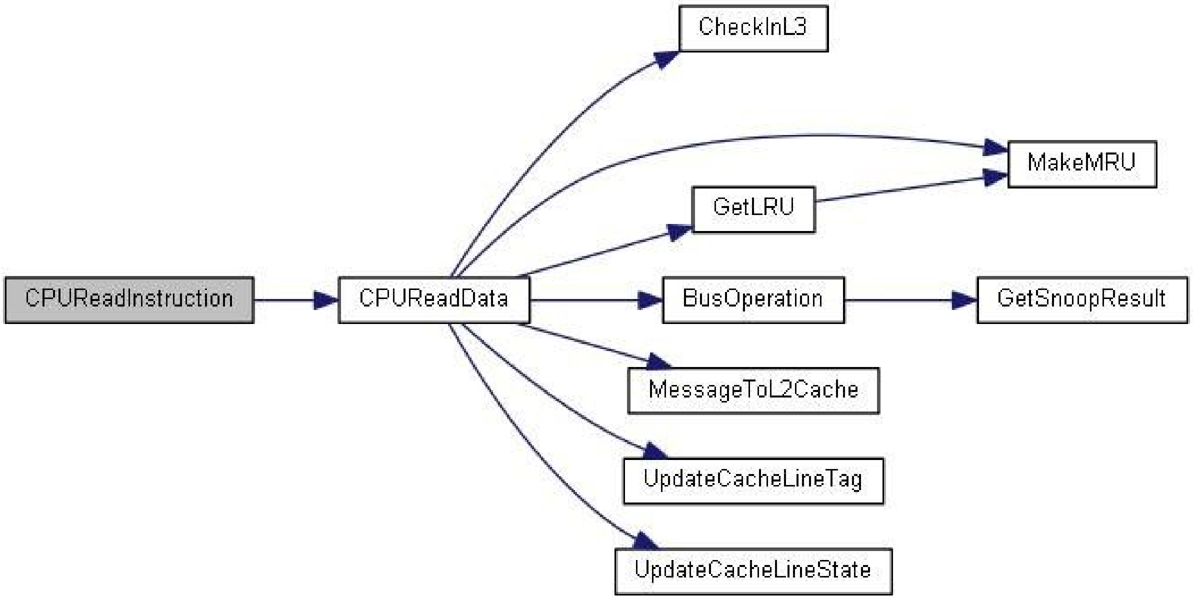
CPUReadData:



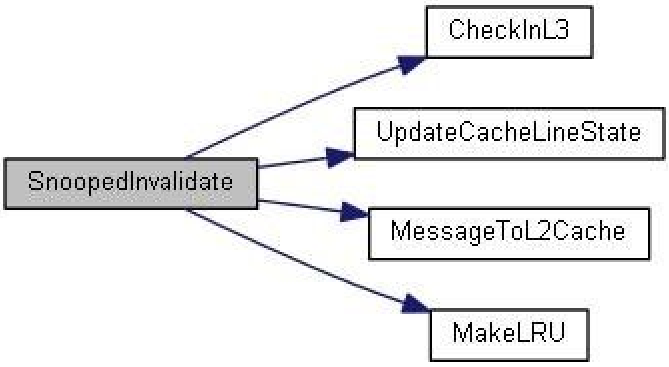
CPUWriteDate:



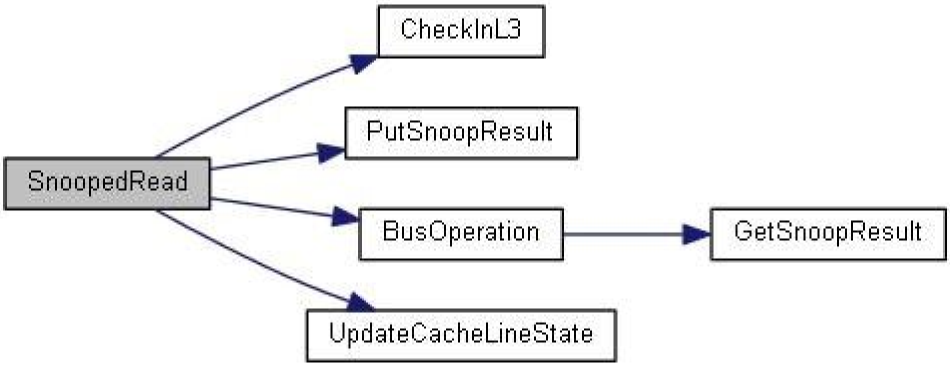
CPUReadInstruction:



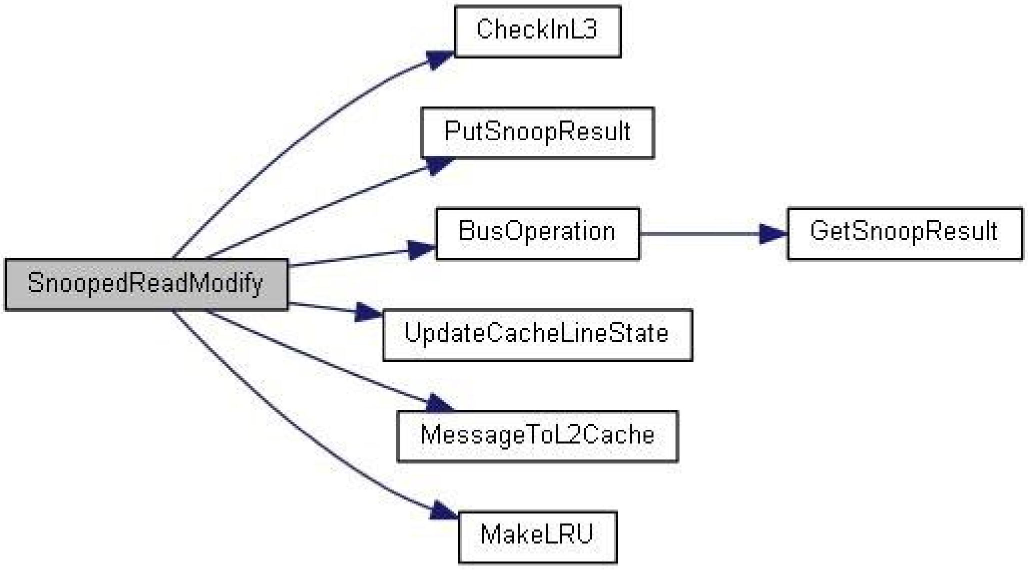
SnoopedInvalidate:



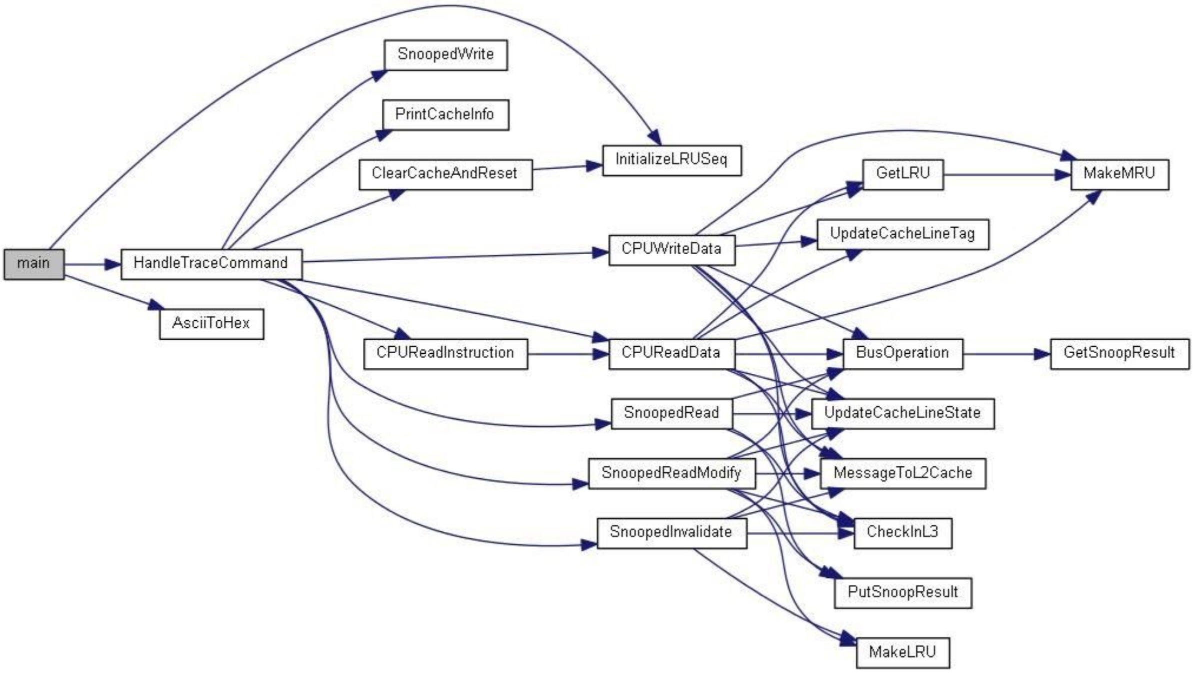
SnoopedRead:



SnoopedReadModify:



Main:



User Interface

Building executable:

To build the executable from Cygwin shell using gcc, the following can be used:

1) Building cacheapp.exe:
   ```
   make all
   ```
2) Cleaning cacheapp.exe:
   ```
   make clean
   ```

Run Command:

To run the executable:

1) From Windows command line, give the following command:
   ```
   cacheapp.exe -f <trace file name>
   E.g: cacheapp.exe -f cc1.din
   ```
2) From Cygwin command line, give the following command:
   ```
   ./cacheapp.exe -f <trace file name>
   E.g: ./cacheapp.exe -f cc1.din
   ```

Output:

The bus operations are output as below:

```
BusOp: 2, Address : d000, Snoop Result : 1
```

where

BusOp = 1 for Read, 2 for Write, 3 for Invalidate, 4 for Read with Intent to Modify

Address = Address in Hexadecimal format

Snoop Result = 0 for No Hit, 1 for Hit, 2 for Hit Modified

Similarly the snoop result output is as below:

```
SnoopResult: Address a000, SnoopResult : 1
```

The message to higher level cache is as below (format: L2 <Bus Op> <Address>):

```
L2: 3 b004
```

Simulator Validation

Test Cases:

Test No.	Module	Test Case	Expected Result	Test Result
1	Trace decoder	Get the arguments from the command line	Successfully read file name.	Passed.
2	Trace decoder	Open and Read Trace File line	Successfully open and read the file.	Passed.
3	Trace decoder	Decode event and address	Successfully decode event and address.	Passed.
4	Trace decoder	Loop through till the end of file and call appropriate event handler for each event	Successfully complete reading the file and decode every line.	Passed.
5	LRU	For a index, fill ways of set to convert invalid ways to valid ways	Most recent way accessed way becomes MRU.	Passed.
6	LRU	All ways are valid and a way is randomly accessed	The accessed way becomes MRU.	Passed.
7	LRU	Some/all ways are valid and a valid way is evicted	The evicted way becomes LRU. A message is sent to higher level cache on eviction.	MRU was not getting updated on eviction.Fixed by adding a function.
8	LRU	All ways are valid and another way is to be written	The LRU way is replaced and way becomes MRU	Passed.
9	Cache Operations	Verify get snoop result algorithm	NOHIT/HIT/HITM is returned as expected based on function defined provided earlier.	Passed.
10	Cache Operations	CPU Read- test all MESI possibilities	Cache hit/miss is as expected. If cache miss, the MESI state transition happens correctly depending on snoop result. On obtaining LRU, if a cache line is evicted a message is	Error: Failed to go into shared state in case of HITM.

			sent to higher level cache.	
11	Cache Operations	CPU Write- test all MESI possibilities	Cache hit/miss is as expected. If cache hit, depending on MESI state Invalidate bus operation is performed. If cache miss bus operation of RWIM is performed..	Passed.
12	Cache Operations	Snoop Read	If cache hit, put appropriate snoop result of HIT/HITM. If cache miss put snoop result of NOHIT.	Passed.
13	Cache Operations	Snoop Write	Do nothing.	Passed.
14	Cache Operations	Snoop Invalidate	If cache hit, invalidate the line and send a message to higher level cache. If cache miss put snoop result of NOHIT.	Error: Message to L2 cache was missing in case of evicting a shared line.
15	Cache Operations	Snoop RWIM	If cache hit, invalidate the line and send a message to higher level cache. If cache hit upto snoop result of NOHIT.	Error: PutSnoopResult() was missing in case the line is in Shared or exclusive state.
16	Cache Operations	Clear cache- at the end of trace file	Cache hit/miss counts are zero at the end of file. LRU counters are initialized again. All cache lines are invalidated.	Error: Got “Divide by zero” error when tried to print cache hit ratio after clearing the counts. Fixed the bug by adding a check.
17	Cache Operations	Print cache info	Print Tag, Way, Index, State and Address of all valid lines.	Added a print indicating no valid lines when there are no valid lines for better representation.

Test Report:

There are two trace files used for testing events. Trace1.din covers test cases 5 to 8. Trace2.din covers test cases 10 to 15. The output of these trace files is given at the end of test report. The test configuration for each of the test cases is mentioned below.

The format of trace file: **event address**

Test Case 5: Filled the cache and verify the LRU counter value.

Test Configuration: 2 sets, 4 ways, 4 byte cache line size

Trace Used:(part of Trace1.din)

```
0 a000
0 b000
1 c000
1 d000
0 a004
0 b004
1 c004
1 d004
```

```
Index   Way  Tag      State  Address  LRU Number
00000000  00  006656   M     d000    0
00000000  01  006144   M     c000    1
00000000  02  005632   S     b000    2
00000000  03  005120   S     a000    3
00000001  00  006656   M     d004    0
00000001  01  006144   M     c004    1
00000001  02  005632   S     b004    2
00000001  03  005120   S     a004    3
```

Test Case 6: Access a random way to verify LRU count.

Test Configuration: 2 sets, 4 ways, 4 byte cache line size

Trace Used:(part of Trace1.din)

```
0 a000
0 b004
```

```
Index   Way  Tag      State  Address  LRU Number
00000000  00  006656   M     d000    1
00000000  01  006144   M     c000    2
00000000  02  005632   S     b000    3
00000000  03  005120   S     a000    0
00000001  00  006656   M     d004    1
00000001  01  006144   M     c004    2
00000001  02  005632   S     b004    0
00000001  03  005120   S     a004    3
```

Test Case 7: Evict a valid way.
Test Configuration: 2 sets, 4 ways, 4 byte cache line size.
Trace Used:(part of Trace1.din)

```
6 a000
6 b004
SnoopResult: Address a000, SnoopResult : 1
L2: 3 a000
SnoopResult: Address b004, SnoopResult : 1
L2: 3 b004
Index    Way  Tag     State  Address  LRU Number
00000000 00   006656  M      d000     0
00000000 01   006144  M      c000     1
00000000 02   005632  S      b000     2
00000001 00   006656  M      d004     0
00000001 01   006144  M      c004     1
00000001 03   005120  S      a004     2
```

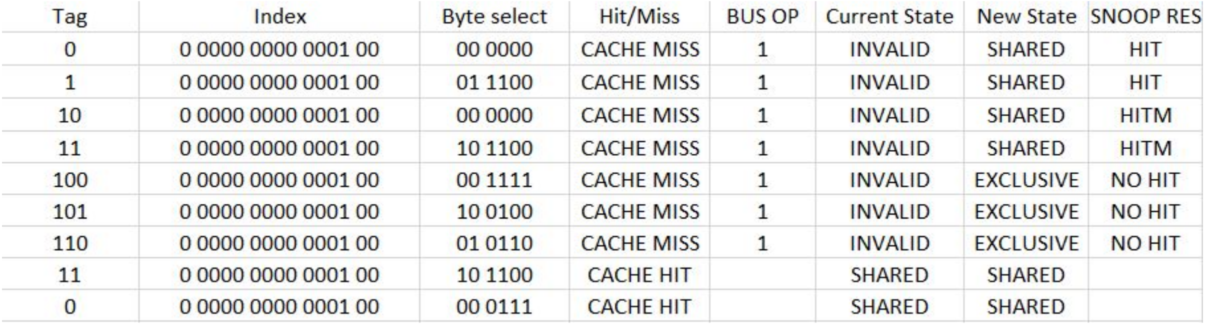
Test Case 8: Fill cache and replace a modified way in cache
Test Configuration: 2 sets, 4 ways, 4 byte cache line size.
Trace Used:(part of Trace1.din)

```
0 c000
0 c004
0 e000
0 e004
1 f000
1 f004
0 a000
BusOp: 2, Address : d000, Snoop Result : 1
L2: 3 d000
Index    Way  Tag     State  Address  LRU Number
00000000 00   005120  S      a000     0
00000000 01   006144  M      c000     3
00000000 02   007680  M      f000     1
00000000 03   007168  S      e000     2
00000001 00   006656  M      d004     3
00000001 01   006144  M      c004     2
00000001 02   007168  S      e004     1
00000001 03   007680  M      f004     0
```

Test Case 10: CPU Read- test all MESI possibilities
Test Configuration: 32K sets, 4 ways, 64 byte cache line size. The operations are performed on one index only. So in essence the test configuration is 1 set, 4 ways, 64 byte cache line.
Trace Used:(part of Trace2.din)

```
0 100
```

0 20011C
0 400100
0 60012C
0 80010F
0 A00124
0 C00126
0 60012C
0 107



Tag	Index	Byte select	Hit/Miss	BUS OP	Current State	New State	SNOOP RES
0	0 0000 0000 0001 00	00 0000	CACHE MISS	1	INVALID	SHARED	HIT
1	0 0000 0000 0001 00	01 1100	CACHE MISS	1	INVALID	SHARED	HIT
10	0 0000 0000 0001 00	00 0000	CACHE MISS	1	INVALID	SHARED	HITM
11	0 0000 0000 0001 00	10 1100	CACHE MISS	1	INVALID	SHARED	HITM
100	0 0000 0000 0001 00	00 1111	CACHE MISS	1	INVALID	EXCLUSIVE	NO HIT
101	0 0000 0000 0001 00	10 0100	CACHE MISS	1	INVALID	EXCLUSIVE	NO HIT
110	0 0000 0000 0001 00	01 0110	CACHE MISS	1	INVALID	EXCLUSIVE	NO HIT
11	0 0000 0000 0001 00	10 1100	CACHE HIT		SHARED	SHARED	
0	0 0000 0000 0001 00	00 0111	CACHE HIT		SHARED	SHARED	

Test Case 11: CPU Write

Test Configuration: 32K sets, 4 ways, 64 byte cache line size. The operations are performed on one index only. So in essence the test configuration is 1 set, 4 ways, 64 byte cache line.
Trace Used:(part of Trace2.din)

1 400100

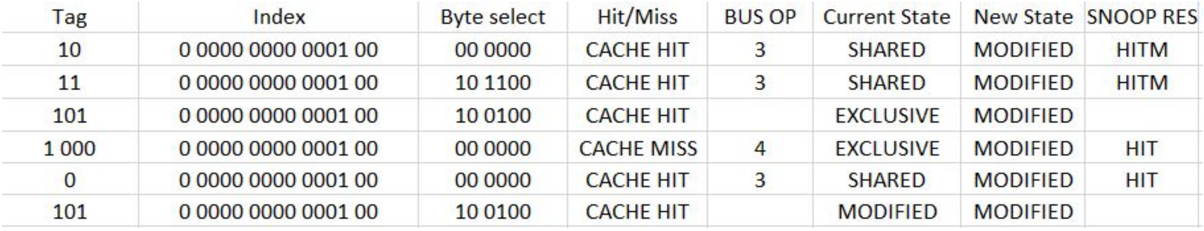
1 60012C

1 A00124

1 1000100

1 100

1 A00124



Tag	Index	Byte select	Hit/Miss	BUS OP	Current State	New State	SNOOP RES
10	0 0000 0000 0001 00	00 0000	CACHE HIT	3	SHARED	MODIFIED	HITM
11	0 0000 0000 0001 00	10 1100	CACHE HIT	3	SHARED	MODIFIED	HITM
101	0 0000 0000 0001 00	10 0100	CACHE HIT		EXCLUSIVE	MODIFIED	
1 000	0 0000 0000 0001 00	00 0000	CACHE MISS	4	EXCLUSIVE	MODIFIED	HIT
0	0 0000 0000 0001 00	00 0000	CACHE HIT	3	SHARED	MODIFIED	HIT
101	0 0000 0000 0001 00	10 0100	CACHE HIT		MODIFIED	MODIFIED	

Test Case 12: Snooped Read

Test Configuration: 32K sets, 4 ways, 64 byte cache line size. The operations are performed

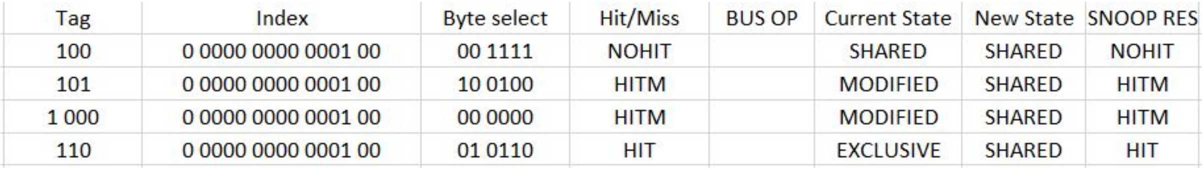
on one index only. So in essence the test configuration is 1 set, 4 ways, 64 byte cache line.
Trace Used:(part of Trace2.din)

4 80010F

4 A00124

4 1000100

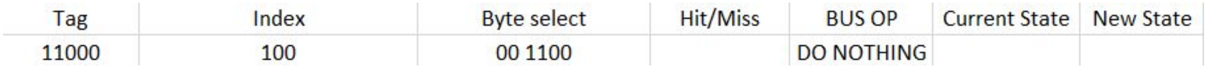
4 C00126



Tag	Index	Byte select	Hit/Miss	BUS OP	Current State	New State	SNOOP RES
100	0 0000 0000 0001 00	00 1111	NOHIT		SHARED	SHARED	NOHIT
101	0 0000 0000 0001 00	10 0100	HITM		MODIFIED	SHARED	HITM
1 000	0 0000 0000 0001 00	00 0000	HITM		MODIFIED	SHARED	HITM
110	0 0000 0000 0001 00	01 0110	HIT		EXCLUSIVE	SHARED	HIT

Test Case 13: Snooped Write

Test Configuration: 32K sets, 4 ways, 64 byte cache line size. The operations are performed on one index only. So in essence the test configuration is 1 set, 4 ways, 64 byte cache line.
Trace Used:(part of Trace2.din)
5 300010C



Tag	Index	Byte select	Hit/Miss	BUS OP	Current State	New State
11000	100	00 1100		DO NOTHING		

Test Case 14: Snooped Invalidate

Test Configuration: 32K sets, 4 ways, 64 byte cache line size. The operations are performed on one index only. So in essence the test configuration is 1 set, 4 ways, 64 byte cache line.

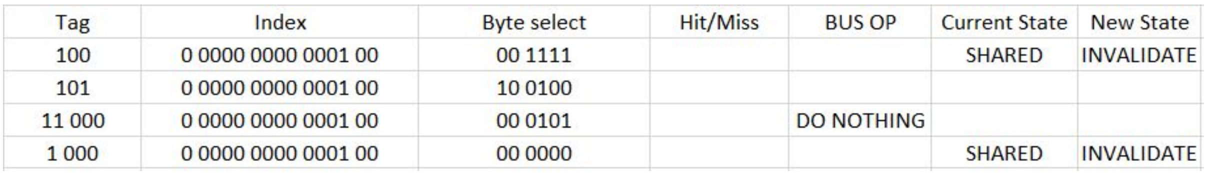
Trace Used:(part of Trace2.din)

3 80010F

3 A00124

3 3000105

3 1000100



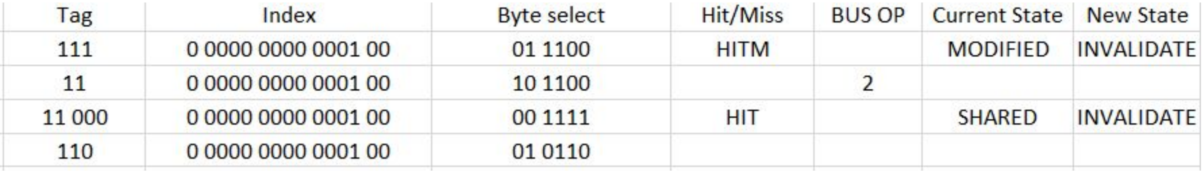
Tag	Index	Byte select	Hit/Miss	BUS OP	Current State	New State
100	0 0000 0000 0001 00	00 1111			SHARED	INVALIDATE
101	0 0000 0000 0001 00	10 0100				
11 000	0 0000 0000 0001 00	00 0101		DO NOTHING		
1 000	0 0000 0000 0001 00	00 0000			SHARED	INVALIDATE

Test Case 15: Snooped RWIM

Test Configuration: 32K sets, 4 ways, 64 byte cache line size. The operations are performed on one index only. So in essence the test configuration is 1 set, 4 ways, 64 byte cache line.

Trace Used: (part of Trace2.din)

```
 6 E0011C
 6 60012C
 6 300010F
 6 C00126
```



Tag	Index	Byte select	Hit/Miss	BUS OP	Current State	New State
111	0 0000 0000 0001 00	01 1100	HITM		MODIFIED	INVALIDATE
11	0 0000 0000 0001 00	10 1100		2		
11 000	0 0000 0000 0001 00	00 1111	HIT		SHARED	INVALIDATE
110	0 0000 0000 0001 00	01 0110				

Test Trace Files:

Trace1.din and usage statistics:

```
0 a000
0 b000
1 c000
1 d000
0 a004
0 b004
1 c004
1 d004
0 a000
0 b004
6 a000
6 b004
0 c000
0 c004
0 e000
0 e004
1 f000
1 f004
0 a000
9
```

Output:

```
Implementing Last-level Unified Cache shared with 4 processors
Total Cache Size in Bytes:              32
```

```
Cache Line Size:            4
Ways per set:                      4
No. of Sets:                2
Trace File Name:            Trace1.din
BusOp: 1, Address : a000, Snoop Result : 1
BusOp: 1, Address : b000, Snoop Result : 1
BusOp: 4, Address : c000, Snoop Result : 1
BusOp: 4, Address : d000, Snoop Result : 1
BusOp: 1, Address : a004, Snoop Result : 1
BusOp: 1, Address : b004, Snoop Result : 1
BusOp: 4, Address : c004, Snoop Result : 1
BusOp: 4, Address : d004, Snoop Result : 1
Index   Way   Tag      State  Address
00000000  00   006656        M       d000
00000000  01   006144        M       c000
00000000  02   005632        S       b000
00000000  03   005120        S       a000
00000001  00   006656        M       d004
00000001  01   006144        M       c004
00000001  02   005632        S       b004
00000001  03   005120        S       a004
Index   Way   Tag      State  Address
00000000  00   006656        M       d000
00000000  01   006144        M       c000
00000000  02   005632        S       b000
00000000  03   005120        S       a000
00000001  00   006656        M       d004
00000001  01   006144        M       c004
00000001  02   005632        S       b004
00000001  03   005120        S       a004
SnoopResult: Address a000, SnoopResult : 1
L2: 3 a000
SnoopResult: Address b004, SnoopResult : 1
L2: 3 b004
Index   Way   Tag      State  Address
00000000  00   006656        M       d000
00000000  01   006144        M       c000
00000000  02   005632        S       b000
00000001  00   006656        M       d004
00000001  01   006144        M       c004
00000001  03   005120        S       a004
BusOp: 1, Address : e000, Snoop Result : 1
```

```
BusOp: 1, Address : e004, Snoop Result : 1
L2: 3 b000
BusOp: 4, Address : f000, Snoop Result : 1
L2: 3 a004
BusOp: 4, Address : f004, Snoop Result : 1
BusOp: 2, Address : d000, Snoop Result : 1
L2: 3 d000
BusOp: 1, Address : a000, Snoop Result : 1
Index    Way  Tag      State  Address
00000000  00  005120     S      a000
00000000  01  006144     M      c000
00000000  02  007680     M      f000
00000000  03  007168     S      e000
00000001  00  006656     M      d004
00000001  01  006144     M      c004
00000001  02  007168     S      e004
00000001  03  007680     M      f004
BusOp: 2, Address : c000, Snoop Result : 1
L2: 3 c000
BusOp: 1, Address : 0, Snoop Result : 1
Cache misses = 14
Cache hits = 4
Cache reads = 12
Cache writes = 6
Cache hit ratio = 0.2222
```

Trace2.din and usage statistics:

```
0 100
0 20011C
0 400100
0 60012C
0 80010F
0 A00124
0 C00126
0 60012C
0 107
1 400100
1 60012C
1 A00124
1 1000100
1 100
1 A00124
0 400100
```

```
4 80010F
4 A00124
4 1000100
4 C00126
5 300010C
3 80010F
3 A00124
3 3000105
3 80010F
3 A00124
3 1000100
3 E0011C
6 E0011C
6 60012C
6 300010F
6 C00126
0 5000124
6 5000124
```

Output:

```
Implementing Last-level Unified Cache shared with 4 processors
Total Cache Size in Bytes:          8388608
Cache Line Size:                    64
Ways per set:                       4
No. of Sets:                        32768
Trace File Name:                    Trace2.din
BusOp: 1, Address : 100, Snoop Result : 1
BusOp: 1, Address : 200100, Snoop Result : 2
BusOp: 1, Address : 400100, Snoop Result : 0
BusOp: 1, Address : 600100, Snoop Result : 0
L2: 3 100
BusOp: 1, Address : 800100, Snoop Result : 1
L2: 3 200100
BusOp: 1, Address : a00100, Snoop Result : 2
L2: 3 400100
BusOp: 1, Address : c00100, Snoop Result : 0
L2: 3 800100
BusOp: 1, Address : 100, Snoop Result : 1
L2: 3 a00100
BusOp: 4, Address : 400100, Snoop Result : 0
L2: 3 c00100
BusOp: 4, Address : a00100, Snoop Result : 2
L2: 3 100
```

```
BusOp: 4, Address : 1000100, Snoop Result : 1
BusOp: 2, Address : 400100, Snoop Result : 0
L2: 3 400100
BusOp: 4, Address : 100, Snoop Result : 1
BusOp: 2, Address : 600100, Snoop Result : 0
L2: 3 600100
BusOp: 1, Address : 400100, Snoop Result : 0
SnoopResult: Address 800100, SnoopResult : 0
SnoopResult: Address a00100, SnoopResult : 2
BusOp: 2, Address : a00100, Snoop Result : 2
SnoopResult: Address 1000100, SnoopResult : 2
BusOp: 2, Address : 1000100, Snoop Result : 1
SnoopResult: Address c00100, SnoopResult : 0
L2: 3 a00100
L2: 3 1000100
SnoopResult: Address e00100, SnoopResult : 0
SnoopResult: Address 600100, SnoopResult : 0
SnoopResult: Address 3000100, SnoopResult : 0
SnoopResult: Address c00100, SnoopResult : 0
BusOp: 1, Address : 5000100, Snoop Result : 1
SnoopResult: Address 5000100, SnoopResult : 1
L2: 3 5000100
Cache misses = 14
Cache hits = 3
Cache reads = 11
Cache writes = 6
Cache hit ratio = 0.1765
```

For 8 way associativity:

```
Cache misses = 9
Cache hits = 8
Cache reads = 11
Cache writes = 6
Cache hit ratio = 0.4706
```