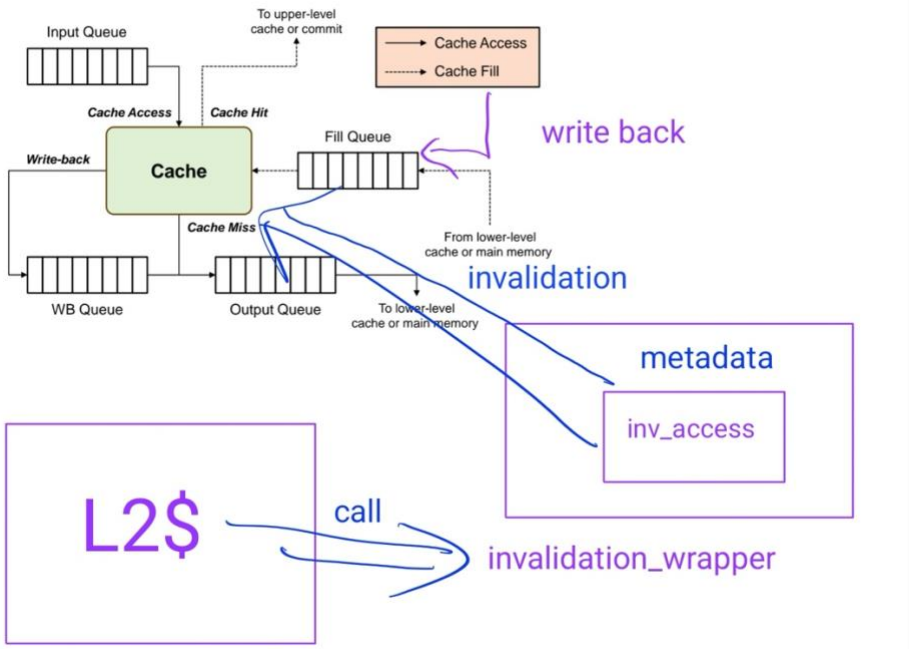
LAB04 REPORT: CACHE

2020-17400

HOJOONKIM

My implementation works with various queue sizes on PART4. It even functions without any limit on the queue size. The following diagram illustrates my overall implementation.



I declared invalidation_wrapper function and inv_access function to protect access of cache metadata.

PART1

	direct-mapped	2-way	4-way	8way
L1 Hit Rate	95.1 %	96.4078 %	96.8525 %	97.1274 %
# of accesses	1000000			
# of hits	951000	964078	968525	971274
# of misses	49000	35922	31475	28726
# of writes	80669			
# of writebacks	5743	4377	2950	2783

PART2

CPI	#of cycles	#of insts	#of memory insts
10.7347	8367825	779515	220485

HIT rate	#of access	#of hits	#of misses	#of writes	#of writebacks
96.8525%	1000000	968525	31475	80669	2950

I implemented L1U cache with main memory. It was difficult to implement WB_address calculation function.

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART2.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  10.7347
number of cycles: 8367825
number of insts: 779515
number of memory insts: 220485
------------------------------
L1U Hit Rate: 96.8525 %
------------------------------
number of accesses: 1000000
number of hits: 968525
number of misses: 31475
number of writes: 80669
number of writebacks: 2950
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

PART3–A

CPI	#of cycles	#of insts	#of memory insts
13.0721	10189890	779515	220485

	L1U cache	L2 cache
HIT rate	89.8369%	68.9681%
#of access	1000000	101631
#of hits	898369	70093
#of misses	101631	31538
#of writes	80669	0
#of writebacks	18399	3008
#of back invalidations	29	0
#of writebacks-backinv	13	0

I implemented L1U cache, L2 cache with main memory. It was difficult to implement invalidation logic. I made a wrapper function “invalidation_wrapper” to wrap “inv_access” function to access and modify cache meta data such as LRU stack, valid bits.

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART3A.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  13.0721
number of cycles: 10189890
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 89.8369 %
------------------------------
number of accesses: 1000000
number of hits: 898369
number of misses: 101631
number of writes: 80669
number of writebacks: 18399
number of back invalidations: 29
number of writebacks due to back invalidations: 13
------------------------------
L2 Hit Rate: 68.9681 %
------------------------------
number of accesses: 101631
number of hits: 70093
number of misses: 31538
number of writes: 0
number of writebacks: 3008
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

PART3–B

CPI	#of cycles	#of insts	#of memory insts
12.644	9856165	779515	220485

	L1I cache	L1D cache	L2 cache
HIT rate	93.4227%	88.474%	58.1177%
#of access	779515	220485	76684
#of hits	728244	195072	44567
#of misses	51271	25413	32117
#of writes	0	80669	0
#of writebacks	0	9251	3126
#of back invalidations	380	1406	0
#of writebacks backinv	0	538	0

I pretty much used schemes of PART3-A.

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART3B.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  12.644
number of cycles: 9856165
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 93.4227 %
------------------------------
number of accesses: 779515
number of hits: 728244
number of misses: 51271
number of writes: 0
number of writebacks: 0
number of back invalidations: 380
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 88.474 %
------------------------------
number of accesses: 220485
number of hits: 195072
number of misses: 25413
number of writes: 80669
number of writebacks: 9251
number of back invalidations: 1406
number of writebacks due to back invalidations: 538
------------------------------
L2 Hit Rate: 58.1177 %
------------------------------
number of accesses: 76684
number of hits: 44567
number of misses: 32117
number of writes: 0
number of writebacks: 3126
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

PART4

Since it can handle the situation well, no limit was set on the queue size. I implemented MSHR on both L1$, L2$.

CPI	#of cycles	#of insts	#of memory insts
1.28306	1000166	779515	220485

	L1I cache	L1D cache	L2 cache
HIT rate	93.5301%	88.684%	58.4196%
#of access	779515	220485	75384
#of hits	729081	195535	44039
#of misses	50434	24950	31345
#of writes	0	80669	0
#of writebacks	0	9198	3055
#of back invalidations	269	1213	0
#of writebacks backinv	0	494	0

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART4.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  1.28306
number of cycles: 1000166
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 93.5301 %
------------------------------
number of accesses: 779515
number of hits: 729081
number of misses: 50434
number of writes: 0
number of writebacks: 0
number of back invalidations: 269
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 88.684 %
------------------------------
number of accesses: 220485
number of hits: 195535
number of misses: 24950
number of writes: 80669
number of writebacks: 9198
number of back invalidations: 1213
number of writebacks due to back invalidations: 494
------------------------------
L2 Hit Rate: 58.4196 %
------------------------------
number of accesses: 75384
number of hits: 44039
number of misses: 31345
number of writes: 0
number of writebacks: 3055
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

Figure 1 NOLIMITED IN_QUEUE FILL_QUEUE

SCHEME

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART4.cfg
ERROR should be WB!!!

TEST!!!
mid: 2
mtype: 2
msize: 5
mincycle: 2
mrdycycle: 130
mcycle: 130
mlvl: 2
m_prev null0


ERROR should be WB!!!

TEST!!!
mid: 3
mtype: 2
msize: 5
mincycle: 3
mrdycycle: 131
mcycle: 131
mlvl: 2
m_prev null0
```

Figure 2Errors raised in multi request on module PART3B

I registered error codes in the region that requests could not enter. For instance, request that are returned from lower-level cache or memory could not cache hit in the second fill request. However, as I activated multiple requests, my code raised error since while in request is in memory or lower cache, request on same memory could be issued. Makes first access miss but second access hit. Which is inconsistent and wrong situation according to our single request cache. It also means that there is some wrong order in requests which might result in inconsistent and OoO-memory.

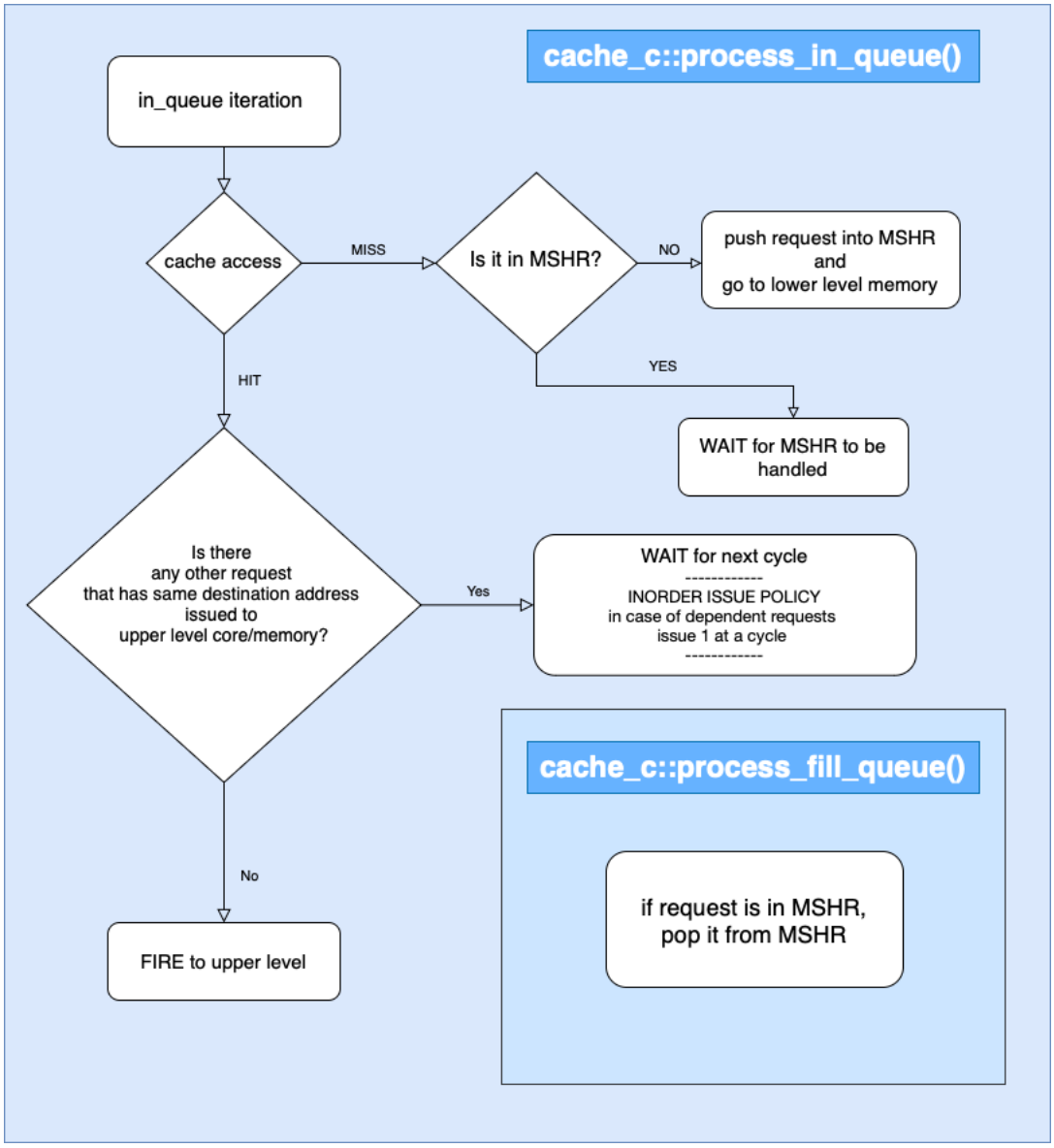


Figure 3 MSHR scheme

I implied MSHR to handle this issue.

- The address is the value obtained by masking the lower 6 bits which is the block size.
- Multiple issue is available if address is different.
- For same address, 1 issue is available at a time.

I strictly followed that rule. The underlying operation of the CACHE for my MSHR to function is as follows:

- Queue works in FIFO policy, so request with the same destination address are handled in the order they are fired to cache, with the earliest request being processed first.
- It is not an essential feature, but fill_queue_process() is called before in_queue_ process().

As I implemented this scheme, there was 1 more problem left. I found some of WB request entering wrong code region. so theoretically, it should always result in a cache hit. However, I found out that 6 out of 9198 WB request sent to L2 cache result in cache miss. The causes are as follows:

- I implemented error handling in all the code regions where specific status requests could not enter.
- While the WB request is being handled, another request can access and evict it. However, excluding that case, according to the definition of an inclusive cache, the WB request must result in cache hit in most situations.
- In the case of LD/ST/INST request, if they were miss at first access (input queue) they result in miss in second access (fill queue, after returning from accessing the lower-level memory). Which is valid and consistent situation.

I wrote verification code to check if WB miss is due to other cache access type of LD/ST/INST, not by another WB request.

	modified position	access type	Access time	valid
WB miss #1	2	LD	322330	1
WB miss #2	2	LD_INST	419390	1
WB miss #3	3	LD_INST	578438	1
WB miss #4	2	LD	604802	1
WB miss #5	0	LD_INST	648987	1
WB miss #6	0	LD	813006	1

METADATA of cache set that WB request is accessing

(only most recent block among set)

I checked access time to know if multiple cache eviction happened or not in a single cycle. I figured out that there were no multiple evictions in those cache sets. We could check from access type that another LD /INST request evicted WB request's target block.

If that is the case, while the WB request is being handled, another earlier accessed request of LD/ST/INST can access and evict it.

I decided to send WB request to memory if it results in miss, as it must be written back to main memory. If there was another request in the in_queue or fill_queue prior to WB that specific block would not have been evicted. So, I must write it back into memory to handle future requests.

For additional information, my MSHR could handle 'different destination address – same cache block' requests. If they are issued sequentially, one could be evicted. In that case, my MSHR could just fire it into lower-level cache or memory and run miss handling again. Orders of dependent requests are not harmed.

QUEUE SIZE variation example

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART4.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  1.28417
number of cycles: 1001029
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 93.5301 %
------------------------------
number of accesses: 779515
number of hits: 729081
number of misses: 50434
number of writes: 0
number of writebacks: 0
number of back invalidations: 270
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 88.6831 %
------------------------------
number of accesses: 220485
number of hits: 195533
number of misses: 24952
number of writes: 80669
number of writebacks: 9198
number of back invalidations: 1213
number of writebacks due to back invalidations: 494
------------------------------
L2 Hit Rate: 58.4207 %
------------------------------
number of accesses: 75386
number of hits: 44041
number of misses: 31345
number of writes: 0
number of writebacks: 3055
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

Figure 4 IN_QUEUE=100 FILL_QUEUE=100

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART4.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  2.86065
number of cycles: 2229918
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 93.4358 %
------------------------------
number of accesses: 779515
number of hits: 728346
number of misses: 51169
number of writes: 0
number of writebacks: 0
number of back invalidations: 317
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 88.5938 %
------------------------------
number of accesses: 220485
number of hits: 195336
number of misses: 25149
number of writes: 80669
number of writebacks: 9253
number of back invalidations: 1315
number of writebacks due to back invalidations: 527
------------------------------
L2 Hit Rate: 58.3322 %
------------------------------
number of accesses: 76318
number of hits: 44518
number of misses: 31800
number of writes: 0
number of writebacks: 3096
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

Figure 5 IN_QUEUE=10 FILL_QUEUE=10

```
sysprog@sysprog-lab:~/comporg-lab4-k296070$ ./memory_sim ./traces/sample.trace ./configs/PART4.cfg
Processed 100000 instructions
Processed 200000 instructions
Processed 300000 instructions
Processed 400000 instructions
Processed 500000 instructions
Processed 600000 instructions
Processed 700000 instructions
------------------------------
Performance Stats
------------------------------
CPI:  3.84347
number of cycles: 2996040
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 93.4335 %
------------------------------
number of accesses: 779515
number of hits: 728328
number of misses: 51187
number of writes: 0
number of writebacks: 0
number of back invalidations: 324
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 88.548 %
------------------------------
number of accesses: 220485
number of hits: 195235
number of misses: 25250
number of writes: 80669
number of writebacks: 9183
number of back invalidations: 1394
number of writebacks due to back invalidations: 525
------------------------------
L2 Hit Rate: 58.2624 %
------------------------------
number of accesses: 76437
number of hits: 44534
number of misses: 31903
number of writes: 0
number of writebacks: 3097
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

Figure 6 IN_QUEUE=5 FILL_QUEUE=5