Lab4 Report_2019-10737_CHOI

전기정보공학부
2019-10737
최동호

Computer Architecture

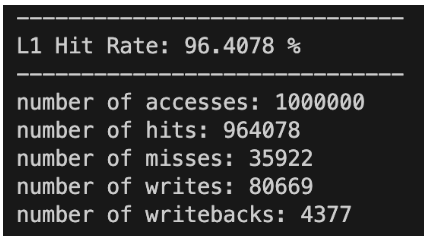
Lab 4: Memory System Simulation

Note

to TA
Part3부터는 main branch가 아니라 testbench branch를 보시면 됩니다.

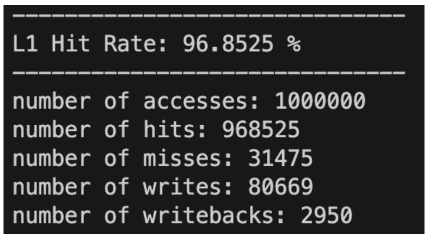
Part1

16384, 2, 64



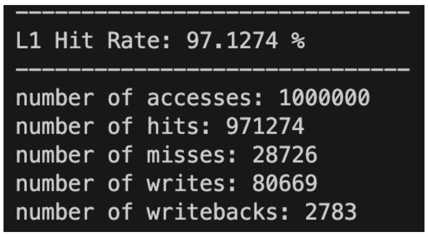
```
-------------------------------
L1 Hit Rate: 96.4078 %
-------------------------------
number of accesses: 1000000
number of hits: 964078
number of misses: 35922
number of writes: 80669
number of writebacks: 4377
```

16384, 4, 64



```
-------------------------------
L1 Hit Rate: 96.8525 %
-------------------------------
number of accesses: 1000000
number of hits: 968525
number of misses: 31475
number of writes: 80669
number of writebacks: 2950
```

16384, 8 64

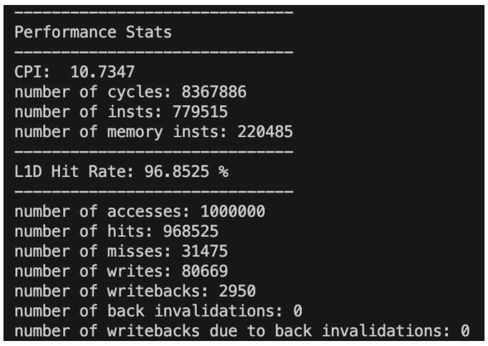


```
-------------------------------
L1 Hit Rate: 97.1274 %
-------------------------------
number of accesses: 1000000
number of hits: 971274
number of misses: 28726
number of writes: 80669
number of writebacks: 2783
```

Part1 에서 기본적인 access 함수를 구현하였고, 결과는 각각 위와 같다.
1-B 에서는 LRU 스택을 std::list를 써서 구현하였다.

Part2

16384 4 64



```
------------------------------
Performance Stats
------------------------------
CPI:  10.7347
number of cycles: 8367886
number of insts: 779515
number of memory insts: 220485
------------------------------
L1D 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
```

Part2 에서는 L1 - Memory의 hierarchy를 구현하였다.
이때 주의했던 점은

1. cache_base::access() 함수의 접근하는 것을 중복으로 세지 않도록 하는 것이었다.

2. dram의 done_func로 l1의 fill()을 호출하였다.

Part3

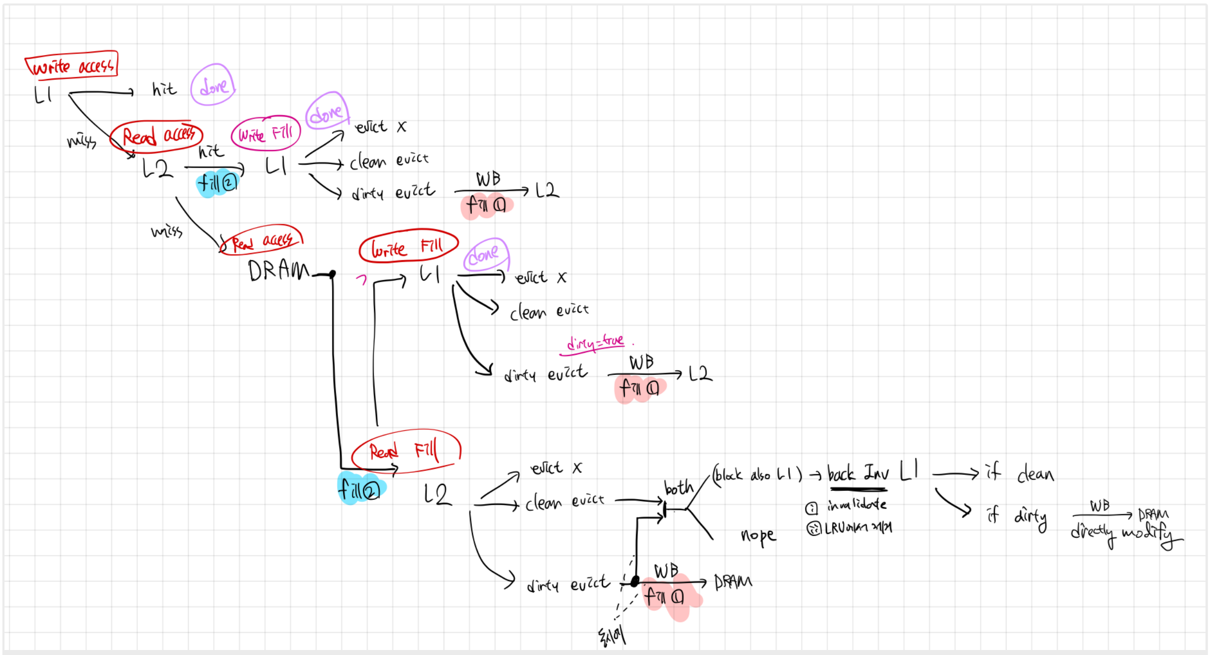


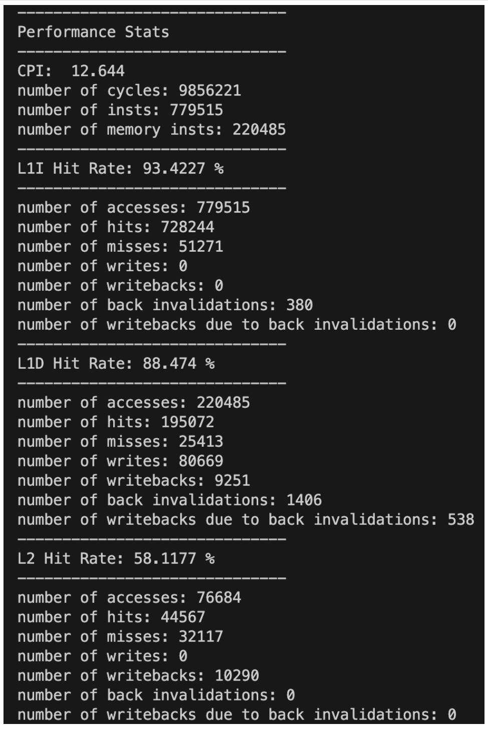
Figure 1. Memory Hierarchy Flow

Part3-A

Stats	Implementation Details

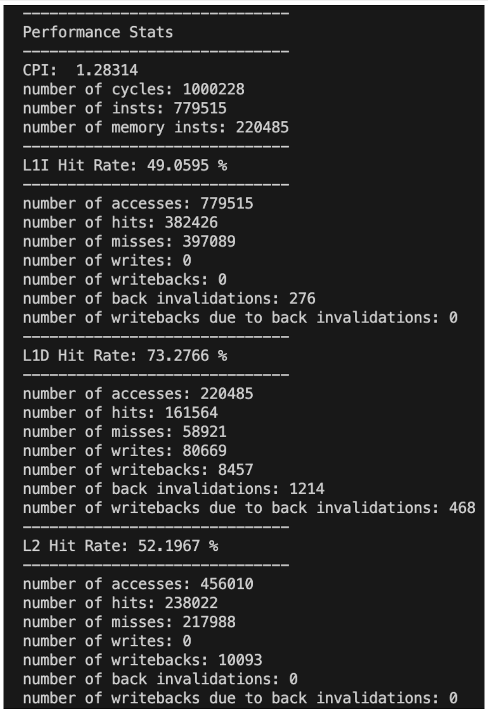
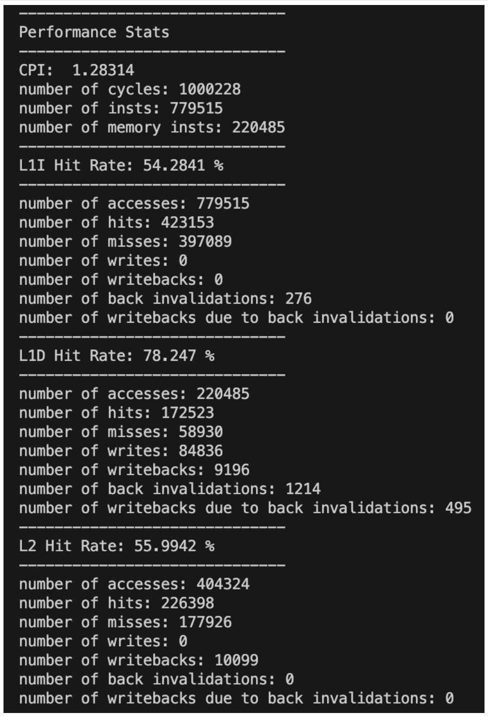
`-------------------------------`
`Performance Stats`
`-------------------------------`
`CPI:  13.0722`
`number of cycles: 10189946`
`number of insts: 779515`
`number of memory insts: 220485`
`-------------------------------`
`L1U Hit Rate: 89.8369 %`
`-------------------------------`
`number of accesses: 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: 31279`
`number of back invalidations: 0`
`number of writebacks due to back invalidations: 0`
	- **Unified L1 Cache (L1U)**: 2KB, 2-Way SA, LRU, 64B Line Size, 4-cycle latency
	- **Unified L2 Cache (L2)**: 16KB, 4-Way SA, **Inclusive**, LRU, 64B Line Size, 10-cycle latency
	
	전반적으로 Figure 1의 다양한 flow의 경우를 모두 고려하였다. 특히 주의한 부분은 다음과 같다.
	1. cache에 upper level의 WB를 하는 경우 반드시 `hit` 이 보장되어야 한다.
	2. miss로 인하여 lower level or memory로 부터 req를 받는 경우 반드시 `!hit` 이어야 한다.
	3. L2 cache 에서 Write Miss 인 경우, access 할 때 Read로 해야하고, 또한 memory로 부터 fill 시에도 Read로 해줘야 한다.
	4. back invalidation 이 일어난 상황에서는 L1 cache에 direct로 처리하도록 구현하였다.

Part3-B

Stats	Implementation Details
	- **L1 Instruction Cache (L1I)**: 2KB, 2-Way SA, LRU, 64B Line Size, 4-cycle latency - **L1 Data Cache (L1D)**: 2KB, 2-Way SA, LRU, 64B Line Size, 4-cycle latency - **Unified L2 Cache (L2)**: 16KB, 4-Way SA, **Inclusive**, LRU, 64B Line Size, 10-cycle latency L1I, L1D 분리하는 데 있어서, 크게는 Part3-A와 같다. 하지만 신경 쓴 디테일은 다음과 같다. 1. Memory에서 L2로 fill을 해줄 때 L1I, L1D 각각 forwarding 한다. 2. L1I는 read-only cache 이므로 Write을 하지 않도록 조심하고, 또한 체크하였다. 3. back invalidation에서는 L1I, L1D cache에 각각 체크하고 해주며 이때 L1I는 clean eviction만 일어나도록 한다.

```
------------------------------
Performance Stats
------------------------------
CPI:  12.644
number of cycles: 9856221
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: 10290
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

Part4

Before	After
	

Before:

```
------------------------------
Performance Stats
------------------------------
CPI:  1.28314
number of cycles: 1000228
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 49.0595 %
------------------------------
number of accesses: 779515
number of hits: 382426
number of misses: 397089
number of writes: 0
number of writebacks: 0
number of back invalidations: 276
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 73.2766 %
------------------------------
number of accesses: 220485
number of hits: 161564
number of misses: 58921
number of writes: 80669
number of writebacks: 8457
number of back invalidations: 1214
number of writebacks due to back invalidations: 468
------------------------------
L2 Hit Rate: 52.1967 %
------------------------------
number of accesses: 456010
number of hits: 238022
number of misses: 217988
number of writes: 0
number of writebacks: 10093
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

After:

```
------------------------------
Performance Stats
------------------------------
CPI:  1.28314
number of cycles: 1000228
number of insts: 779515
number of memory insts: 220485
------------------------------
L1I Hit Rate: 54.2841 %
------------------------------
number of accesses: 779515
number of hits: 423153
number of misses: 397089
number of writes: 0
number of writebacks: 0
number of back invalidations: 276
number of writebacks due to back invalidations: 0
------------------------------
L1D Hit Rate: 78.247 %
------------------------------
number of accesses: 220485
number of hits: 172523
number of misses: 58930
number of writes: 84836
number of writebacks: 9196
number of back invalidations: 1214
number of writebacks due to back invalidations: 495
------------------------------
L2 Hit Rate: 55.9942 %
------------------------------
number of accesses: 404324
number of hits: 226398
number of misses: 177926
number of writes: 0
number of writebacks: 10099
number of back invalidations: 0
number of writebacks due to back invalidations: 0
```

Multiple Instructions Issue

- Before 과 같이 L1I, L1D, L2 에서 모두 hit rate이 크게 감소하였다.
- Part3 까지는 로직 상 하나의 request가 commit 되어야 다음 request를 받는 구조였으므로, req 끼리의 dependency를 고려할 필요는 없었다.
- 하지만 지금은 앞선 request가 commit 되지 않은 상태에서 다음 request가 같은 address를 갖고 중복적으로 캐시를 참조하는 상황이 생겼음을 깨달았다.

Implementation Details

- 앞선 request가 miss인데 아직 처리되지 못한 상황에서 후의 request가 똑같이 miss인 경우, 두 중복된 req를 하나로 합칠 수 있다고 판단하였다.
    - L1에서 miss가 발생한 경우, L1 in_queue 에서 memory hierarchy의 `in_flight_reqs` 를 참조하여 같은 address가 있으며 이전에 이미 miss인 경우,
    - 두 req를 한번에 처리할 수 있다.
- 다음은 처리한 로직의 디테일이다.

older request from in_flight_req	younger request (now L1 cache)	Handling in L1 cache (younger one)
Read	Read	1. do not forward out_queue 2. delete request 3. L1 `num_hits` ++ 4. do not touch L1 LRU
IF	IF	1. do not forward out_queue 2. delete request

older request from in_flight_req	younger request (now L1 cache)	Handling in L1 cache (younger one)
		3. L1 `num_hits` ++ 4. do not touch L1 LRU
Write	Read	1. do not forward out_queue 2. delete request 3. L1 `num_hits` ++ 4. do not touch L1 LRU
Write	Write	1. do not forward out_queue 2. delete request 3. L1 `num_hits` ++ 4. do not touch L1 LRU 5. L1 `num_writes` ++
Read	Write	1. do not forward out_queue 2. delete request 3. L1 `num_hits` ++ 4. do not touch L1 LRU 5. L1 `num_writes` ++ 6. older request's access_type: Read => Write

- 이로 인해 약 5%의 성능개선을 보였다.