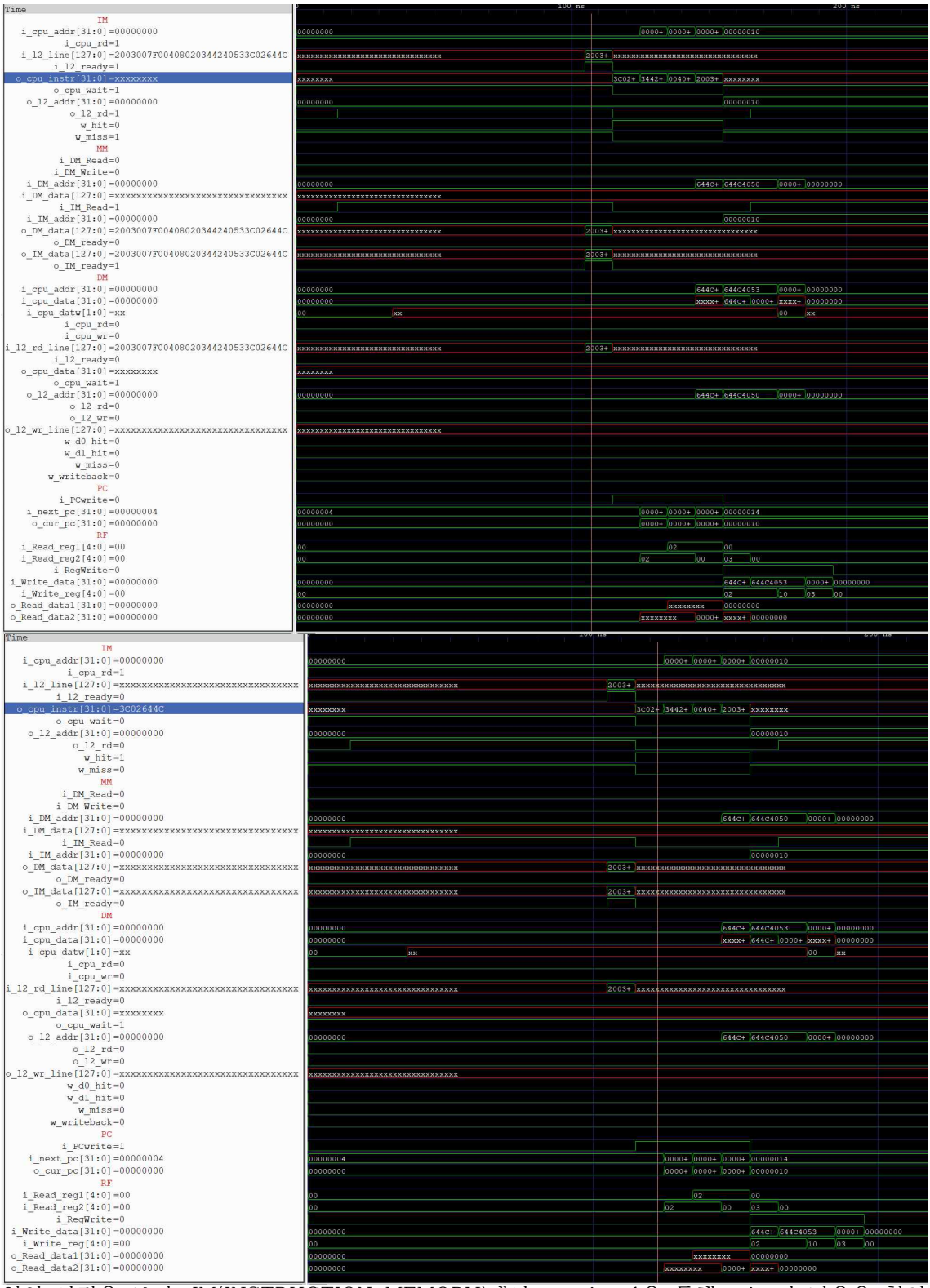
Project #4
<Cache Design>

과목명: 컴퓨터구조실험
학번: 2021202045
이름: 김예은
담당 교수님: 이성원 교수님
분반: 컴구실 (수) 분반

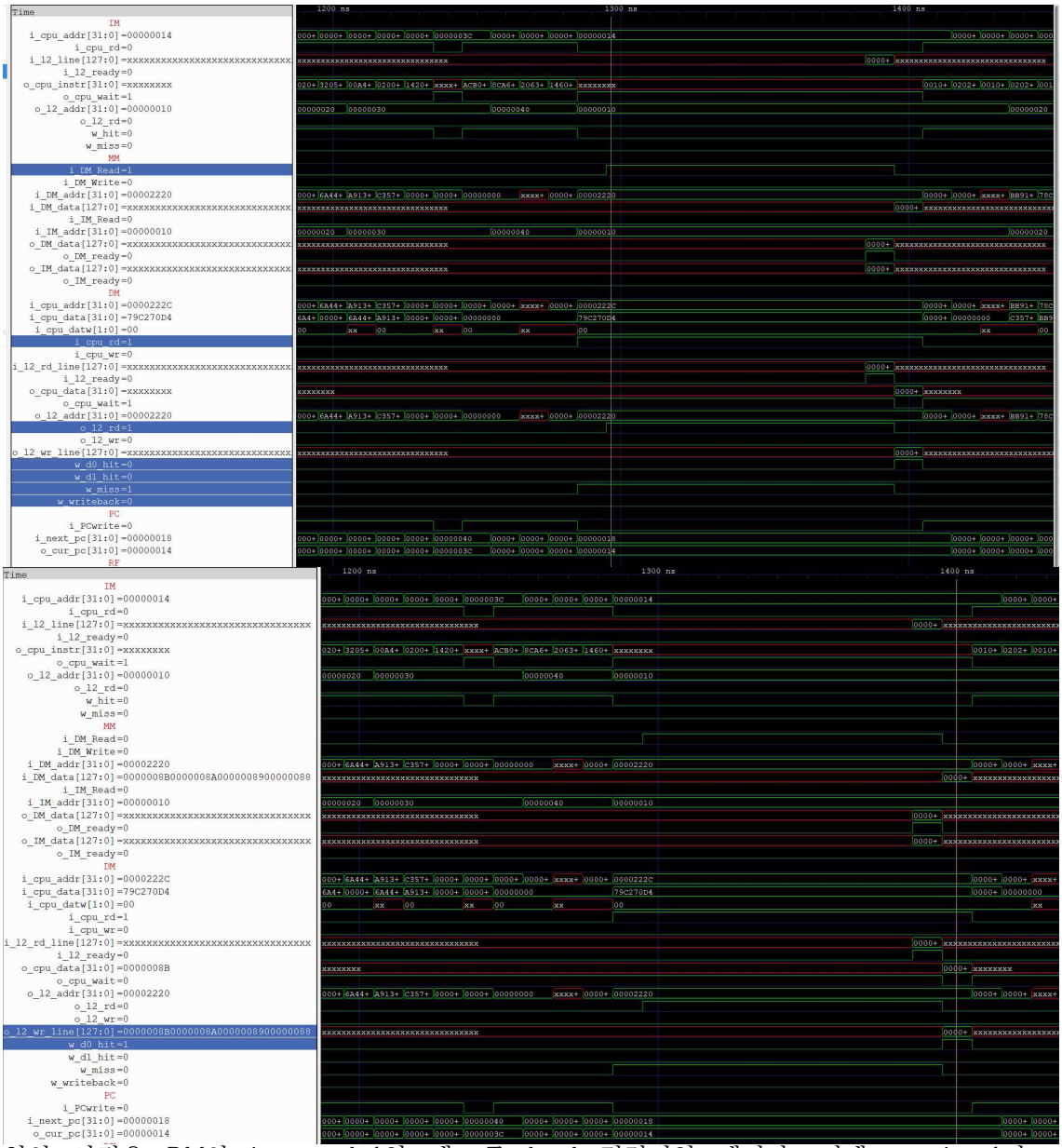
Ⅰ. GTKWAVE 분석(Random Access 및 Insertion Sort)

(1) Random Access

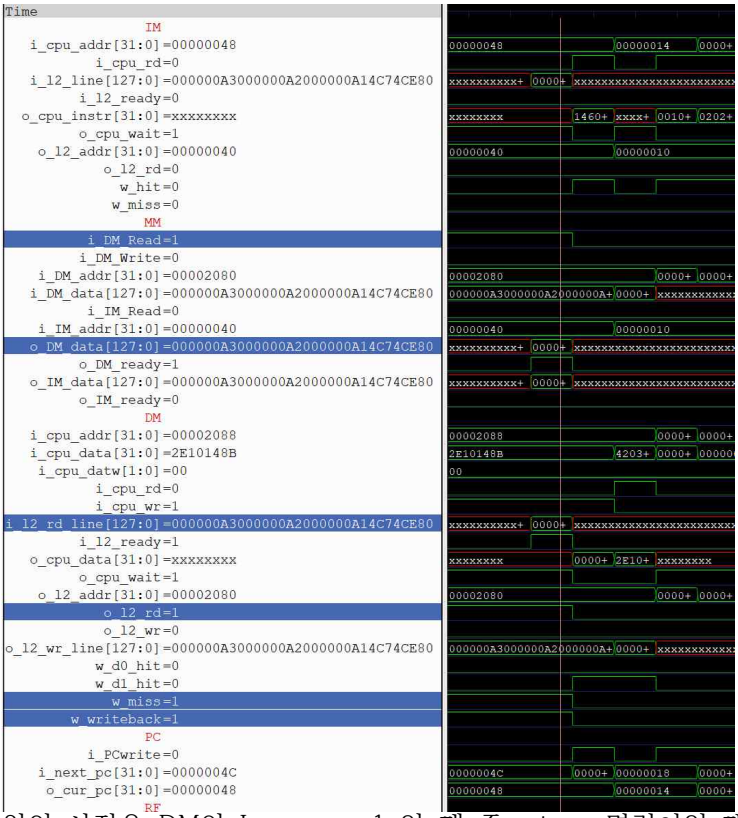


위의 사진을 보면, IM(INSTRUCTION MEMORY)에서 w_miss=1을 통해 miss가 났음을 확인할 수 있다. 따라서, o_l2_rd=1이 되면서 l2 cache로 읽어오려 하였으나 MM의

I_IM_Read=1이 확인되어, main memory까지 가서 값을 읽어와 l2 cache로 가져온 것을 i_l2_line을 통해 확인할 수 있다. 총 128bit 중 0x3C02644C부분을 가져왔으며, o_cpu_wait=0이 되면서 o_cpu_insr에서 main memory에서 가져와 준비된 값을 읽는 것을 확인할 수 있다. 그 때의 w_hit=1이 된다. cache에 가져왔기 때문이다. 아무래도 l2 cache에서도 찾을 수 없어 main memory까지에 접근해야했기 때문에 데이터가 준비되기까지 한 눈에 봐도 긴 cycle이 걸렸음을 확인할 수 있다.

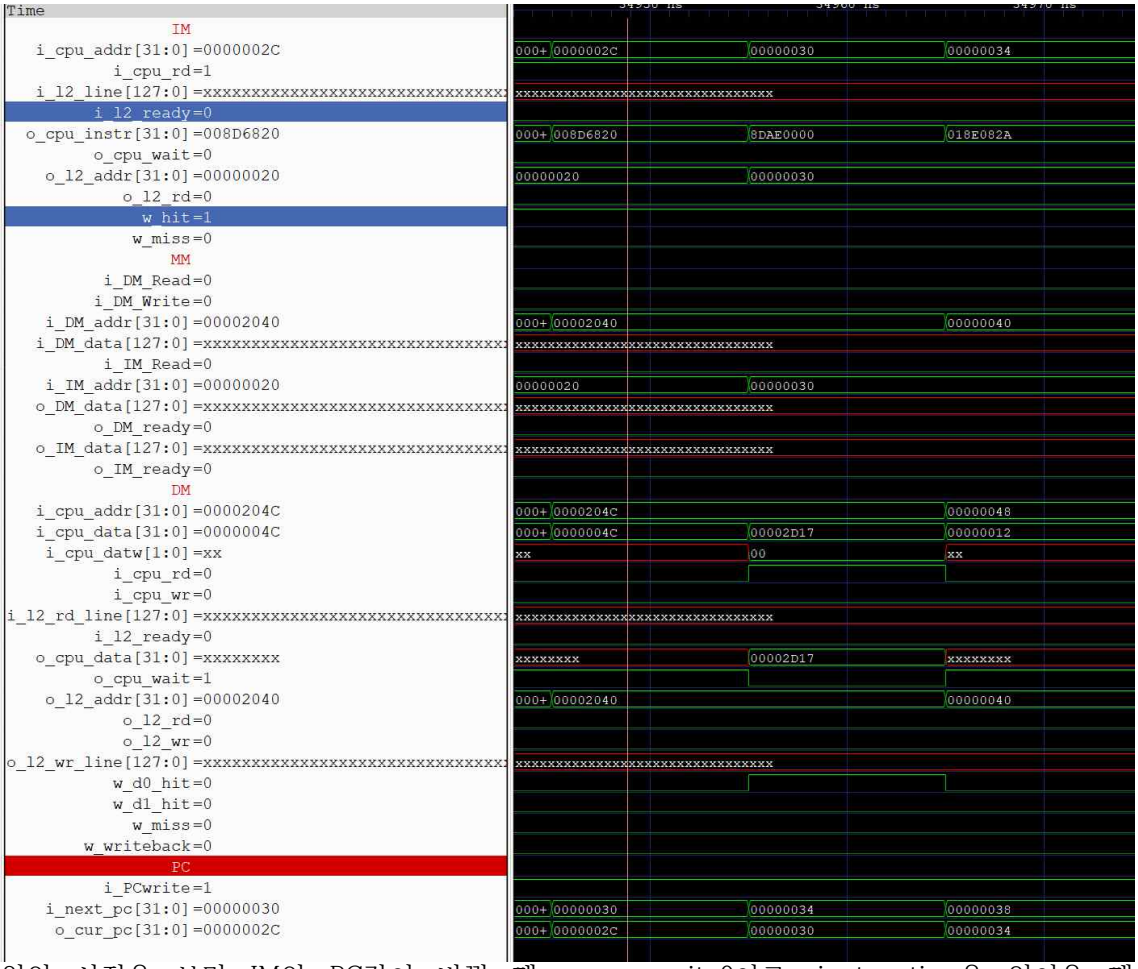


위의 사진은 DM의 i_cpu_rd=1일 때, 즉 load 명령어일 때이다. 이때 w_miss=1이고, o_l2_rd=1인 것을 보아 l2 캐시에서 읽어오려 했으나 MM의 I_DM_Read=1인 것을 보아 main memory까지 접근한 것으로 확인할 수 있다. 긴 clock을 지난 다음, o_l2_wr_line을 통해 값이 준비된 것을 확인할 수 있다. 이때 w_d0_hit=1이 되었다.

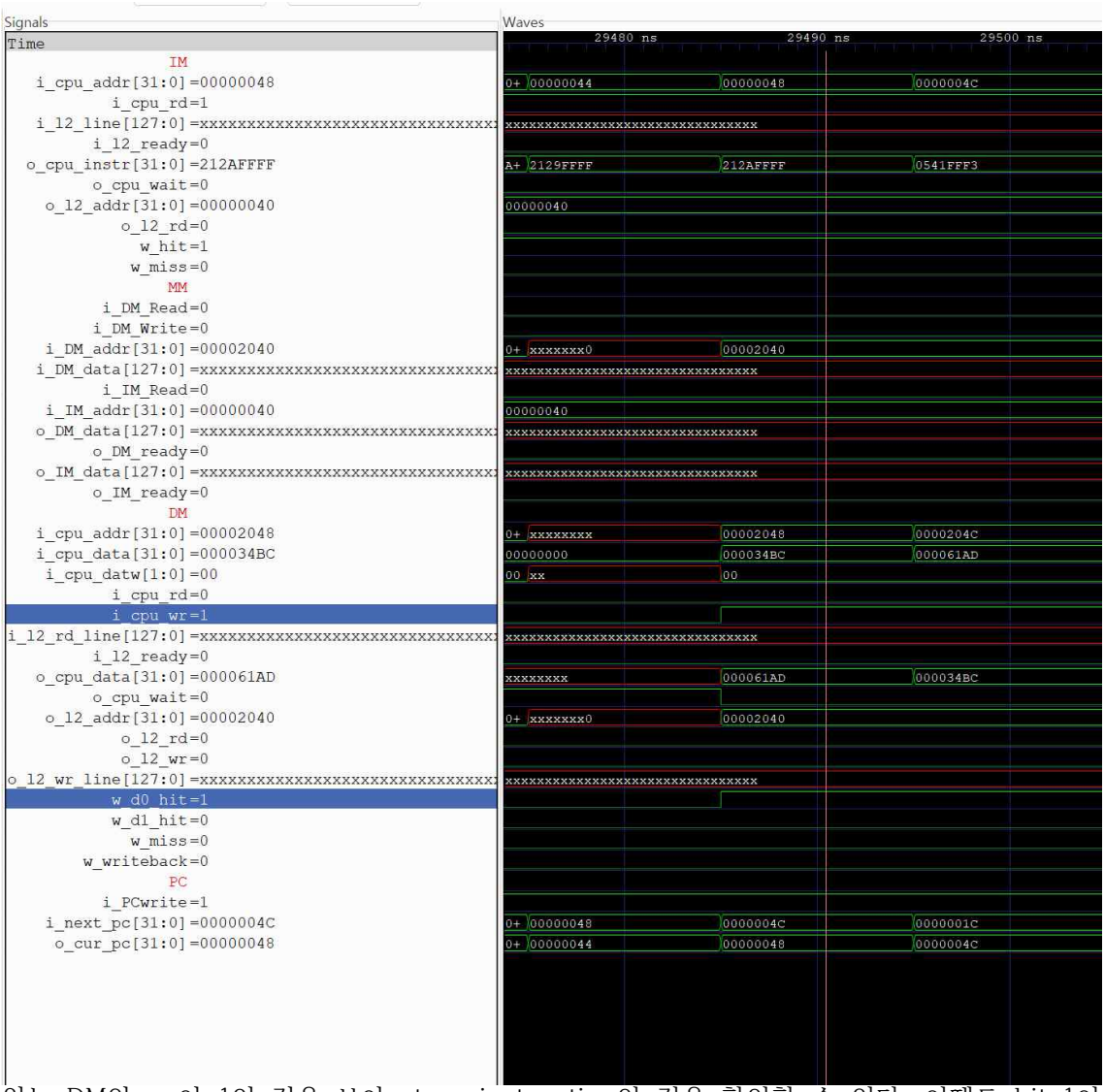


위의 사진은 DM의 I_cpu_wr=1 일 때 즉, store 명령어일 때이다. 이때 w_miss=1이 나오고, o_l2_rd=1이다. MM의 I_DM_Read=1이므로, 이 또한 main memory까지 접근하였음을 알 수 있다. 이때 o_DM_data값이 MM에서 추출되는 것을 확인할 수있고, 다음 clock에서 값이 준비되면서, w_d1_hit=1로 바뀌는 것을 확인함으로써, 해당 값이 cache까지 잘 올라왔음을 확인할 수 있다. 또한 w_writeback=1이므로, writeback 방식을 이용하는 것 또한 파악할 수 있다. write back이란, cache에 있는 값을 수정하고자 할 때, 캐시에 있는 값만 수정하는 것을 말한다. write through와 대조되는 개념이다.

(2) Insertion Sort



위의 사진을 보면 IM의 PC값이 바뀔 때 o_cpu_wait=0이고, instruction을 읽어올 때 w_hit=1이 된다. 즉, 캐시에 instr이 있어서 캐시에서 가져온 것으로 보인다. 위의 random access에서는 대부분 hit가 없고, miss 였는데 그 조차도 MM까지가서 값을 읽어와서 cycle 수가 길게 늘어났던 거에 비해 hit가 되면 위처럼 pc 하나당 cycle latency가 짧은 것을 확인 할 수 있다.

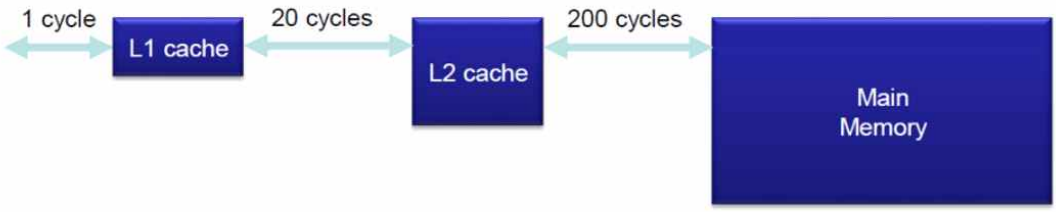


위는 DM의 wr이 1인 것을 보아 store instruction인 것을 확인할 수 있다. 이때도 hit=1이 되면서 l1 cache에 있는 값을 바로 읽어 store명령어를 진행한 것을 확인할 수 있다. l1 cache에 있었으므로, cycle latency가 길게 늘어나지 않는다.

random access와 insertion sort의 gtkwave분석 결과, random access에 비해 insertion sort의 miss가 발생하는 경우 확실히 적다는 것을 확인할 수 있었다. random access는 miss가 나면 거의 대부분 l2 cahce에서 해결되는 것이 아니라 Main Memory에 접근하여 값을 가져오기 때문에, miss penalty가 매우 큰 것도 확인할 수 있었다. random이기 때문에 temporal locality와 spatial locality의 장점이 살아나지 않는 것 같다.

Ⅱ. SIMULATION OF CACHE DESIGN(TOTAL 4)

L1 cache access time is 1 cycle
L2 cache access time is 20 cycles
Main memory access time is 200 cycles



본 프로젝트에서 위의 access time을 hit time으로 가정한다. L2 cache를 setting하지 않는 경우의 simulation에서 L1에서 miss가 나면 miss penalty를 220 cycle이 아닌 20 cycle로 가정하여 AMAT을 계산한다.

If increase twice cache size, cycle time increase 4%

If increase twice associativity, cycle time increase 2%

위의 프로젝트 설명에 따라 cache size가 두배 늘어날 때 hit time을 0.04 더해주었고, associativity가 두배 늘어날 때, hit time을 0.02 더해주었다. 해당 강의 묻고 답하기의 조교님 답변에 따라 위의 두 조건이 동시에 발생했을 때(simulation 3)는 hit time에 0.06을 더해 hit time이 늘어나는 상황을 가정하여 AMAT을 계산해주었다. 기존의 4%를 더하든, 0.04를 더하든 상관없다고 하셔서 우선 계산의 복잡도를 줄이기 위해 기존의 4%가 아닌 0.04를 더하는 방향으로 hit time을 늘려주었다.

(1) simulation 1 : unified vs split

① peri

Simulation 1. Unified vs splits

- Block size = 16, Associativity = 1
- Split cache
  - ✓ Each of instruction cache, Data cache : Number of sets / 2
- Simulate the performance of the cache under the following condition
  - ✓ ./sim-cache –config configfile path application path
  - ✓ Ex) ~/simplesim-3.0/sim-cache -config ~/simplesim-3.0/config/mycache.cfg bin/vortex.little.ss inputs/vortex.raw 2> traces/vortex.trace

# of Sets		Unified cache Miss rate	Unified cache AMAT	Split cache (0.7725) Inst. Miss rate	Split cache (0.2275) Data Miss rate	Split cache AMAT
1	64	0.37158	9.4316	0.4052	0.2790	7.3861 (7.138+1.248)
1.04	128	0.29067	7.8932	0.3527	0.1805	7.35 (6.346+1.004)
1.08	256	0.23050	6.477	0.2797	0.1386	6.064 (5.232+0.832)
1.12	512	0.1817	5.874	0.2371	0.1079	4.937 (4.229+0.708)

$\frac{1}{1+0.2275}$ (I-cache Amat) + $\frac{0.2275}{1+0.2275}$ (D-cache Amat)

$\frac{1}{1+0.2275} \fallingdotseq 0.814$ $\frac{0.2275}{1+0.2275} \fallingdotseq 0.216$

② ijpeg

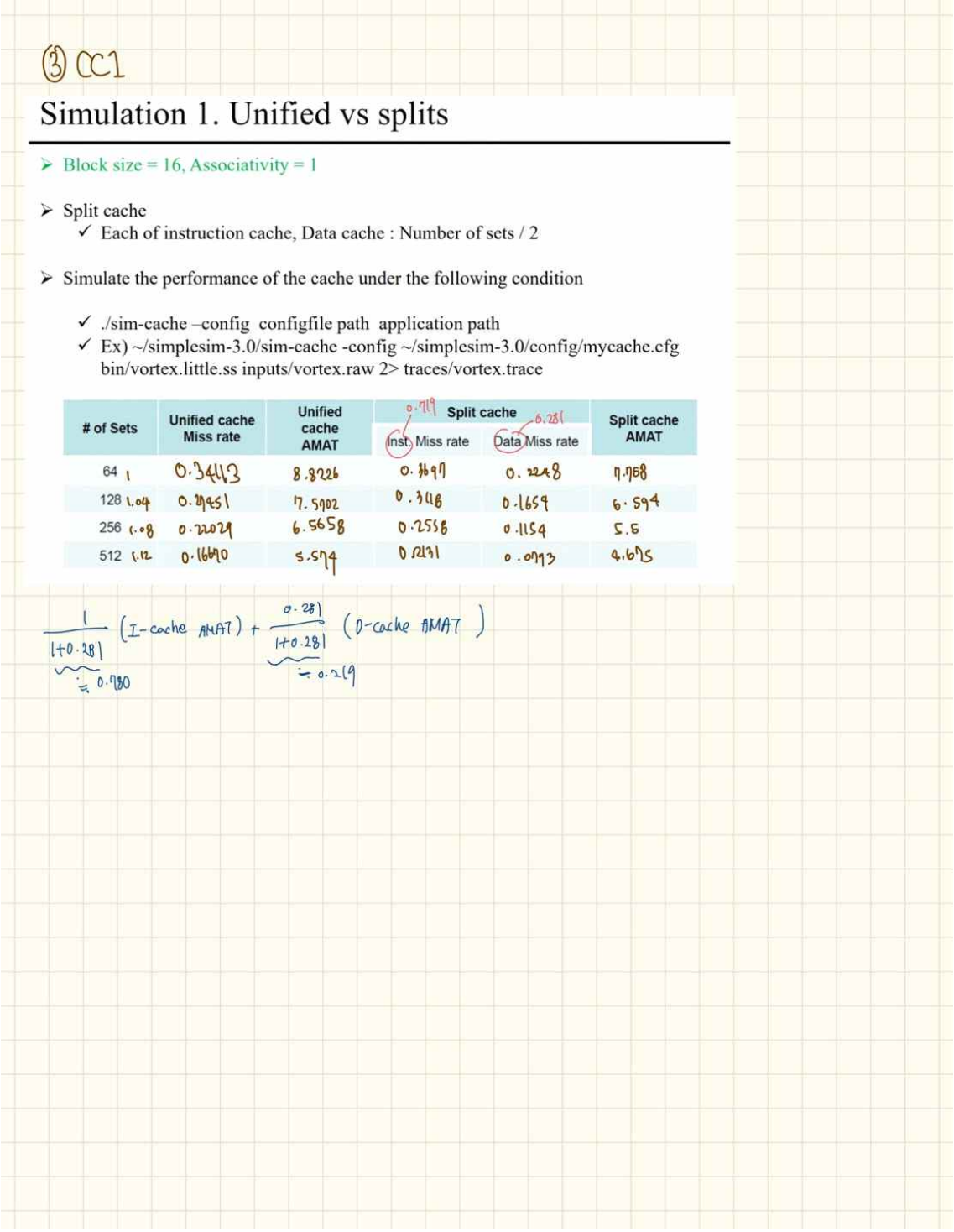
Simulation 1. Unified vs splits

- Block size = 16, Associativity = 1
- Split cache
  - ✓ Each of instruction cache, Data cache : Number of sets / 2
- Simulate the performance of the cache under the following condition
  - ✓ ./sim-cache –config configfile path application path
  - ✓ Ex) ~/simplesim-3.0/sim-cache -config ~/simplesim-3.0/config/mycache.cfg bin/vortex.little.ss inputs/vortex.raw 2> traces/vortex.trace

# of Sets		Unified cache Miss rate	Unified cache AMAT	Split cache (0.782) Inst. Miss rate	Split cache (0.218) Data Miss rate	Split cache AMAT	
1	64	0.18802	5.7604	0.2586	0.2911	6.166	(5.067+1.099)
1.04	128	0.10056	4.0912	0.1089	0.2194	3.614	(2.642+0.972)
1.08	256	0.04571	3.0142	0.0212	0.1483	1.959	(1.235+0.724)
1.12	512	0.0322	2.864	0.0047	0.0791	1.481	(0.997+0.484)

$\frac{1}{1+0.218}$ (I-cache AMAT) + $\frac{0.218}{1+0.218}$ (D-cache Amat)

$\frac{1}{1+0.218} \fallingdotseq 0.821$ $\frac{0.218}{1+0.218} \fallingdotseq 0.179$



③ CC1

Simulation 1. Unified vs splits

- Block size = 16, Associativity = 1
- Split cache
  - ✓ Each of instruction cache, Data cache : Number of sets / 2
- Simulate the performance of the cache under the following condition
  - ✓ ./sim-cache –config configfile path application path
  - ✓ Ex) ~/simplesim-3.0/sim-cache -config ~/simplesim-3.0/config/mycache.cfg bin/vortex.little.ss inputs/vortex.raw 2> traces/vortex.trace

# of Sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst. Miss rate	Data Miss rate	
64 1	0.3413	8.8226	0.3697	0.2248	7.758
128 1.04	0.2951	7.5102	0.3118	0.1659	6.594
256 1.08	0.2209	6.5658	0.2558	0.1154	5.6
512 1.12	0.1690	5.574	0.2131	0.0773	4.675

$\frac{1}{1+0.281}$ (I-cache AMAT) + $\frac{0.281}{1+0.281}$ (D-cache AMAT)

$\frac{1}{1+0.281} \fallingdotseq 0.780$, $\frac{0.281}{1+0.281} \fallingdotseq 0.219$

1번 simulation의 경우, unified(폰 노이만 구조) cache일 때와 split(havard) cache일 때 block 사이즈와 associativity는 고정하고, set 수만 늘려 AMAT을 구하는 실험을 하였다. 우선 위의 시뮬레이션에서는 L2 cache가 없다는 가정하에 돌리는 실험이다. hit time은 위에서 설명했듯이, set 수가 2배 늘어날 때마다 0.04를 더해주었고, miss penalty는 20으로 AMAT을 계산해주었다. UNIFIED CAHCE의 경우 load나 store가 hit time에서 extra cycle을 두 배로 잡아먹기 때문에 hit time을 두 배로 넣어 계산해주었다. 그 결과, PERL의 경우, SET수

가 두배로 늘어날수록 차이가 근소해지긴 하나 split cache가 unified cache보다 AMAT이 작은 것을 확인할 수 있다. IJPEG의 경우, SET 수가 64일 때는 split cache의 AMAT이 unified cahce보다 크더니 set수가 두배씩 늘얼 때 split cache의 AMAT이 unified cache보다 확 작아지는 것을 확인할 수 있다. set수가 늘어나면서 ijpeg 내에서 instruction access와 memory access가 중복되어 delay가 생기는 것으로 예상할 수 있다. CC1의 경우 split일 때의 AMAT이 unified cache의 AMAT보다 작은 것을 확인할 수 있다. 위의 세 개의 시뮬레이션을 통해 공통적으로 SET수가 512일 때 즉, CAHCE 사이즈가 가장 클 때, miss rate가 제일 작고, 그에 따라 AMAT이 가장 작은 값을 가지는 것을 확인할 수 있다. cache size가 커짐에 따라 hit time은 증가하겠지만, miss rate가 줄면서 AMAT이 작아진다. 또한, ijpeg의 set 수가 64일 때를 제외하고는(본인 생각으로는 split cache는 set수가 64/2=32로 cache size가 너무 작아져서 오히려 miss rate가 unified cahce보다 더 발생하는 것으로 보인다.) unified cache보다 split cache의 AMAT이 작다는 것을 확인할 수 있다. 따라서 모든 벤치마크에 대해 split cache가 더 적합하다는 것을 확인할 수 있다.(예외: ijpeg의 set수가 64일 때 제외)

(2) simulation 2 : L1/L2 size

① perl

Simulation 2. L1/L2 size

- Block size = 16, Associativity = 1
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition
  - ✓ Do calculate, once each for instruction-miss rate, data miss rate, inst & data unified cache and AMAT

L1I/L1D/L2U	Inst.Miss rate (0.925)	Data.Miss rate (0.215)	Unified Cache Miss rate	AMAT
8/8/1024	0.4579	0.375	0.3074	36.853
16/16/512	0.4342	0.2763	0.4153	42.723
32/32/256	0.4052	0.239	0.5764	51.046
64/64/128	0.3527	0.1805	0.8107	58.596
128/128/0	0.2997	0.1386	0	6.145

② ijpeg

Simulation 2. L1/L2 size

- Block size = 16, Associativity = 1
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition
  - ✓ Do calculate, once each for instruction-miss rate, data miss rate, inst & data unified cache and AMAT

L1I/L1D/L2U	Inst.Miss rate (0.782)	Data.Miss rate (0.218)	Unified Cache Miss rate	AMAT
8/8/1024	0.4912	0.4193	0.031	13.532
16/16/512	0.3733	0.3511	0.0429	18.811
32/32/256	0.2586	0.2571	0.1261	12.762
64/64/128	0.1089	0.2194	0.4098	14.188
128/128/0	0.0212	0.1483	0	2.089

③ cc1

Simulation 2. L1/L2 size

- Block size = 16, Associativity = 1
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition
  - ✓ Do calculate, once each for instruction-miss rate, data miss rate, inst & data unified cache and AMAT

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.4327	0.3761	0.245	82.693
16/16/512	0.4014	0.2954	0.391	40.167
32/32/256	0.3697	0.2248	0.5967	98.165
64/64/128	0.3118	0.1654	0.8267	82.957
128/128/0	0.2558	0.1154	0	8.660

0.719 0.281

AMAT : %inst (hit time + missrate x misspenalty) + %ds (hit time + missrate x misspenalty)

misspenalty : L2 cache의 AMAT

: hit time + miss rate x misspenalty

: 200 cycle

이번 시뮬레이션 2는 L2 cache를 MM과 L1 cahce size 사이에 배치하여 L1의 set수를 늘림으로써 사이즈를 늘리고 L2의 사이즈를 줄이며 AMAT을 확인하는 실험이다. L1 cache는 split cache로, L2 cache는 unified cache로 설정한다. PERL의 경우, L1의 사이즈가 늘고, L2의 사이즈가 줄어들수록 AMAT이 늘어나는 것을 확인할 수 있다. 본 시뮬레이션에서 마지막 칸은 L2 cahce size가 0이되면서, L2 cache가 아예 없을 때의 영향을 분석하라는 칸인 것을 보인다. 이론 상으로는 L2 cache가 없으면 L1 cache에서 miss가 날 때 바로 main

memory에 접근해야하므로 miss penalty가 커진다. 하지만 본 프로젝트에서 L2 cache가 없다면 miss penalty를 220이 아니라 20으로 두고 계산해야하기 때문에 오히려 L2 cahce가 없어야 AMAT이 작아지는 아이러니한 현상이 발생한다. 따라서 마지막 L2 cache가 0일 때가 AMAT이 가장 작으니 L2 cache가 없어야한다는 결과를 내리지 않기 위해, 해당 시뮬레이션 표의 마지막 칸은 분석하지 않도록 한다. IJPEG의 경우, L1의 캐시가 각각 set수가 64이고, L2캐시가 SET수가 256일 때 AMAT이 가장 작아졌다가 바로 L1은 두배로 커지고 L2가 두배로 작아지면 다시 AMAT이 커지는 것을 확인할 수 있다. cc1의 경우 L2 캐시 사이즈가 가장 클 때가 AMAT이 가장 작고, L2캐시사이즈가 작아질수록 AMAT이 늘어나는 것을 확인할 수 있다. 또한, IJPEG의 AMAT보다 CC1과 PERL의 AMAT이 유난히 큰 것을 보아 IJPEG는 branch 명령어가 다른 두 개의 벤치마크에 비해 없다는 것으로 분석할 수 있다. 왜냐하면 branch 명령어가 많으면 temporal, spatial locality가 적용되지않으므로, cache에서 값을 찾을 확률이 적기 때문이다. 따라서 AMAT이 높아질 텐데, 이때 L2 cache 사이즈가 커지면서 miss penalty를 조금이라도 줄일 수 있다. 따라서 L2 캐시 사이즈가 클수록 AMAT이 작은 이유(PERL과 CC1)를 찾을 수 있다. PERL의 경우 L2 cache set가 1024일 때, ijpeg의 경우 L2 set가 256일 때, cc1의 경우 L2 SET수가 1024일 때 가장 AMAT이 작다.

(3) simulation 3 : Associativity

① perl

Simulation 3. Associativity

- Block size = 16
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition

# of Sets	Split Cache Miss rate / AMAT			
	1-way	2-way	4-way	8-way
1 64	0.3913 8.426	0.2807 6.634	0.2038 5.86	0.1371 3.802
1.04 128	0.3044 5.220	0.2088 5.936	0.1414 3.908	0.0859 2.814
1.08 256	0.2304 5.680	0.1491 4.162	0.0899 2.918	0.0449 2.238
1.12 512	0.1816 4.152	0.1026 3.192	0.0492 2.198	0.0282 1.784
1.16 1024	0.1493 4.146	0.07137 2.454	0.0324 1.948	0.0220 1.66
1.2 2048	0.0951 3.102	0.0405 1.901	0.0230 1.7	0.0142 1.684

② ijpeg

Simulation 3. Associativity

- Block size = 16
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition

# of Sets	Split Cache Miss rate / AMAT			
	1-way	2-way	4-way	8-way
1 64	0.1380 4.716	0.0831 2.682	0.0234 1.508	0.0078 1.216
1.04 128	0.1006 3.052	0.0619 1.998	0.0082 1.294	0.0038 1.176
1.08 256	0.0457 1.994	0.0181 1.482	0.0040 1.2	0.0024 1.194
1.12 512	0.0322 1.764	0.0093 1.296	0.0027 1.214	0.0021 1.228
1.16 1024	0.0157 1.774	0.0070 1.24	0.0022 1.248	0.0021 1.262
1.2 2048	0.0084 1.368	0.0023 1.266	0.0021 1.24	0.0015 1.29

③ CC1

Simulation 3. Associativity

- Block size = 16
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition

# of Sets	Split Cache Miss rate / AMAT			
	1-way	2-way	4-way	8-way
64	0.3441 7.82	0.2999 6.018	0.1859 4.82	0.1354 3.768
128	0.2945 6.53	0.1961 4.988	0.1397 3.854	0.0965 2.83
256	0.2203 5.436	0.1471 4.042	0.0906 2.992	0.043 2
512	0.1667 4.459	0.0999 3.12	0.0477 2.09	0.0194 1.528
1024	0.1205 3.597	0.0583 2.346	0.021 1.682	0.0086 1.992
2048	0.0798 2.976	0.0300 1.882	0.0099 1.438	0.0061 1.382

위에서 갈색은 hit time을 나타내는 숫자이므로 무시해도 된다.

PERL의 경우, set수를 고정했을 때 way수가 늘어날수록 대체로 AMAT이 작아진다. way를 고정하고 set수를 늘리면, 대체적으로 AMAT이 작아진다. 해당 벤치마크에서 가장 적절한 CACHE는 2-way의 set수가 2048일 때이다. 아무래도 way수가 너무 커지면 mux 사이즈가 커지고, 오히려 hit time이 늘어나기 때문인 거 같다. 그래도 associativity가 늘어날수록 miss rate가 줄어든 것만큼은 확실하다. ijpeg의 경우 또한 associatibity가 늘어날수록 확실히 miss rate가 줄어드는 것을 확인할 수 있으며 대체적으로 set수가 늘어날수록 AMAT이

줄어든다. n-way는 늘어날수록 AMAT이 줄어들다가 8-way에서 다시 AMAT이 증가하는 경우도 보인다. IJPEG에서는 4-way이고, set수가 256일 때 가장 AMAT이 작은 것을 확인할 수 있다. CC1의 경우는 앞의 두경우와 달리 하나의 예외도 없이 set 수가 늘어날수록, way 수가 늘어날수록 AMAT이 작아지는 것을 확인할 수 있다. 따라서 8-way이고, set수가 2048일 때 가장 작은 AMAT을 가진다. hit time이 늘어나는 것보다, cache의 사이즈를 키워서 miss rate를 줄이는 게 중요한 벤치마크인듯하다.

(4) simulation 4 : Block Size

①ler1

Simulation 4. Block size

- Number of Sets = 512, Associativity = 1
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition

Block size	Unified cache Miss rate	AMAT
16	0.1816	4.632
64	0.0493	1.996
128	0.0254	1.508
256	0.0153	1.306
512	0.0092	1.144

②tiff

Simulation 4. Block size

- Number of Sets = 512, Associativity = 1
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition

Block size	Unified cache Miss rate	AMAT
16	0.0322	1.644
64	0.0093	1.186
128	0.0014	1.028
256	0.0008	1.016
512	0.0006	1.012

(3) CC1

Simulation 4. Block size

- Number of Sets = 512, Associativity = 1
  - ✓ Use sim-cache to simulate the performance of the cache under the following condition

Block size	Unified cache Miss rate	AMAT
16	0.1669	9.334
64	0.0410	1.82
128	0.0206	1.412
256	0.0118	1.236
512	0.0064	1.128

이번 시뮬레이션은 set수와 associativity를 고정하고, block size를 늘림으로써 AMAT을 계산하여 비교하는 실험이다. 이때 cache는 L1 cache만 사용하며, unified cache이다. PERL,IJPEG,CC1 모든 벤치마크의 경우 block size가 늘어날 대마다 miss rate가 줄어드는 것을 확인할 수 있다. block size가 크면 들어있는 data가 많아지기 때문에 cache hit가 날 확률이 높아지기 때문이다. miss rate가 줄어듦에 따라 AMAT도 작아진다. 따라서 모든 벤치마크에 대해 가장 block size가 클 때가 가장 적합한 cache design이라고 볼 수 있다. block size가 커지면 spacial locality의 장점을 가질 수 있고, compulsory miss를 줄일 수

있다는 장점이 있다. 하지만 만약 replace가 되면, 많이 replace가 되어 miss rate가 늘어날 수 있는 단점도 있다.

Ⅲ. 프로젝트 전체 요약 및 고찰

우선 첫 번째로 gtkwave 분석을 통해, random access와 insertion sort 알고리즘을 분석하는 것이 1번 과제였고, 2번 과제로는 위의 네가지의 조건에 따라 cache design을 바꾸며 각각 벤치마크에 대해 어떤 식으로 적용되는지, unified/split, associativity, l2의 size, block size, cache size가 AMAT에 어떤 영향을 끼치는 지 직접 벤치마크를 돌려보며 확인하는 프로젝트이다. 프로젝트를 진행하면서, IJPEG의 경우 branch 명령어가 다른 두 벤치마크에 비해 작은 비율로 이루어졌다는 것을 확인할 수 있었고, 각각의 data cache access 및 instruction cache access를 통해 CC1의 경우가 data cache에 가장 access를 많이하는 것을 확인할 수 있었다. 이번 프로젝트를 진행하면서 AMAT을 계산하며 은근 hit time이 영향을 많이 준다는 것을 확인할 수 있었다. 왜냐하면 맨처음에 unified vs split(simulation 1) 실험을 진행했을 때, unified cache가 AMAT이 더 작게 나와서 이론과 다른 결과에 당황하여 AMAT 계산이 잘못된건가 했었다. 그런데 unified cache일 때는 load/store가 extra cycle을 더 잡아먹는 다는 것을 깨닫고 hit time을 두배로 넣어주니 split cache가 역시 AMAT이 더 작게나왔다. 이 부분을 다시 깨닫게 되었고, 역시 unified cache는 instruction fetch와 data cache access의 중복 때문에 split이 더 효율적이라는 것을 알게 되었다. 맨처음에 또한 AMAT을 구할 때 set수와 associativity의 증가에 따른 hit time 증가를 생각하지 못했는데 hit time이 늘어나는 것에 따라 분석 결과가 살짝 살짝씩 달라져서 뭔가 흥미로웠다. 확실히 hit time을 다 1로 뒀을 때는 너무 결과가 뻔한거 아닌가 했는데 hit time 증가에 따라 오히려 way 수가 작아야 AMAT이 더 작은 상황이 발생하면서 더 최적화된 cache design하기가 어렵다는 생각도 하였다. cache design 프로젝트를 하면서 컴퓨터가 돌아가는 구조에 대해 더 잘 알게 되었다. 또한 branch 명령어의 비율도 중요하다는 것을 알았다. 이것이 내가 실제로 코딩하는 데에 얼마나 큰 영향을 줄지는 모르겠지만, loop나 branch 명령어가 하드웨어가 수행하기에 빠르지 않다는 점을 알고, 잘 코딩해야겠다.