Assignment#4

Cache Design

과목명	컴퓨터구조
담당교수	이성원 교수님
학과	컴퓨터정보공학부
학년	3학년
학번	2019202009
이름	서여지
제출일	21.06.14(월)

KWANGWOON UNIVERSITY

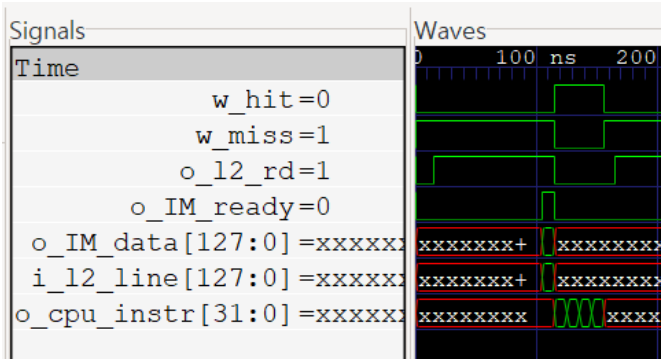
1. 실험내용

A. 실험1

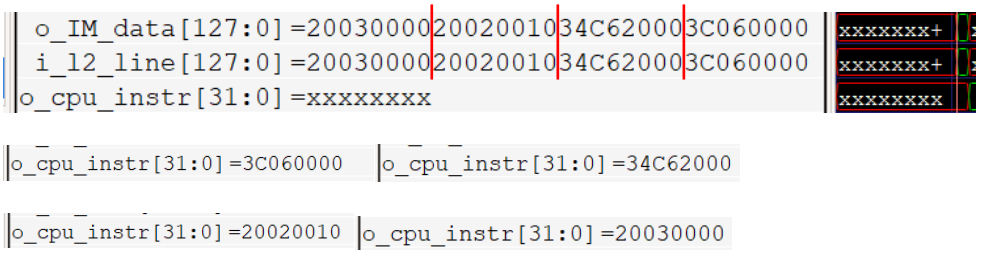
i. IM, DM hit/miss 동작 – cache access timing 분석

cache는 찾는 data가 있는지 확인하여 hit인 경우 해당 data word를 내보낸다. miss인 경우 MM을 1 block 읽어 cache에 저장한 뒤, 다시 cache를 확인하는 동작을 통해 hit를 확인한 뒤, data를 출력하는 과정으로 동작한다.

(1) instruction memory의 예를 들어 miss동작을 살펴보면 다음과 같다.

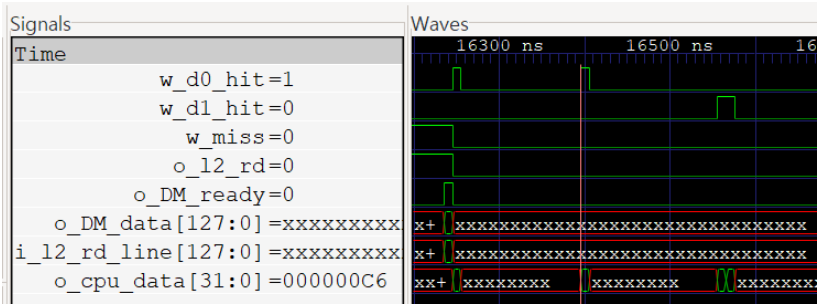


w_hit	IM의 hit여부	
w_miss	IM의 miss여부	
o_l2_rd	IM의 MM 읽기요청 여부	
o_IM_ready	MM에서 IM으로 보내는 data의 사용가능 여부	
o_IM_data	MM에서 IM으로 보내는 block data	4개의 명령어
i_l2_line	IM이 받은 block data	o_IM_data와 동일
o_cpu_instr	IM이 내보내는 명령어	o_IM_data의 일부



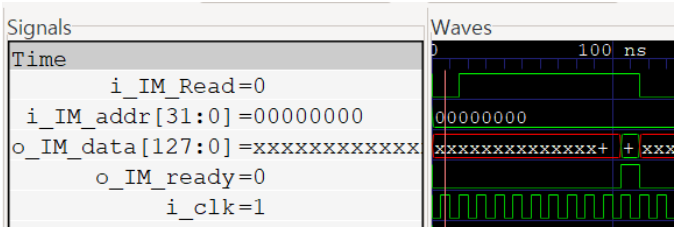
miss가 발생한 경우 o_l2_rd가 1이 되어 MM에 data를 요청한다. 이후MM은 주소값에 해당하는 block을 읽고, o_IM_ready를 1로 바꾸어 요청한 data를 모두 준비하여 IM이 block을 전달받는 것이 가능하다는 것을 알린다. 다시 IM은 MM이 전달하는 값을 읽어 저장한다. MM가 address에 해당하는 data block을 찾는 동안 IM은 대기한다.

(2) data memory의 예를 들어 hit동작과정을 살펴보면 다음과 같다.

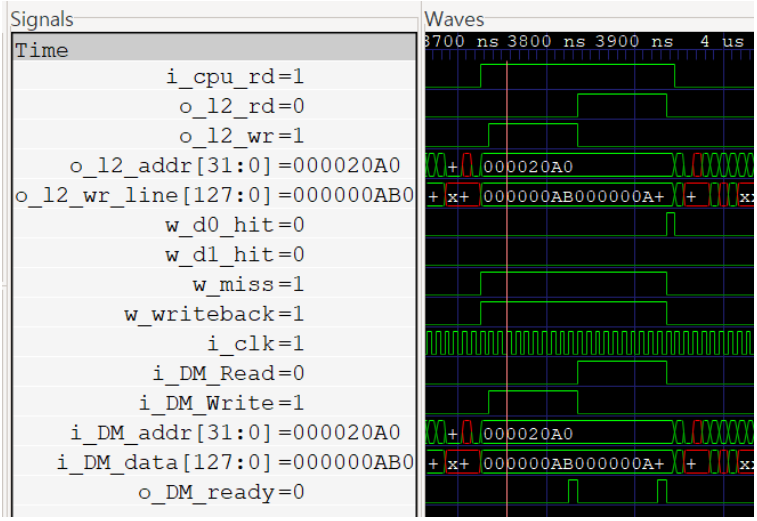


hit가 발생한 경우 DM은 해당하는 32bit data를 출력한다. MM에 접근하지 않으므로 data가 바로 출력된다.

ii. MM 동작



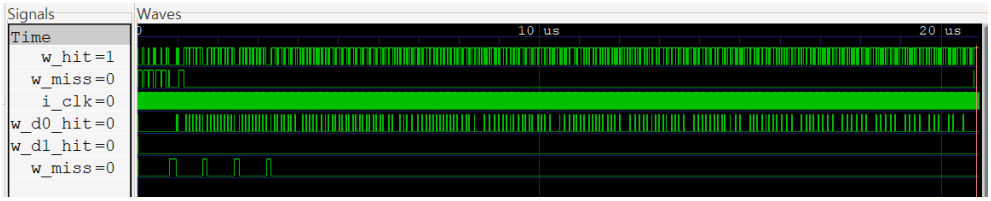
MM은 각각 IM과 DM에서 출력하는 read신호에 따라 address에 해당하는 block을 읽어 전달하는 동작을 한다. i_IM_Read가 1이 되면 MM은 i_IM_addr의 값에 해당하는 data를 읽어 o_IM_data를 통해 IM에 전달하고, IM이 해당 data를 읽을 수 있다는 것을 o_IM_ready를 통해 알린다. 위의 wave form을 통해 MM이 data를 읽는 것에 여러 cycle이 사용된 것을 확인할 수 있다. DM이 MM을 통해 값을 읽는 경우에도 동일하게 동작한다.



위는 DM에서 writeback이 발생한 경우이다. DM에서 o_l2_wr을 1값으로 변경하여 MM에 값을 저장한 뒤, 다시 읽는 것을 확인할 수 있다.

iii. BS와 RND프로그램의 cache 동작 차이 – 적합한 cache 제시

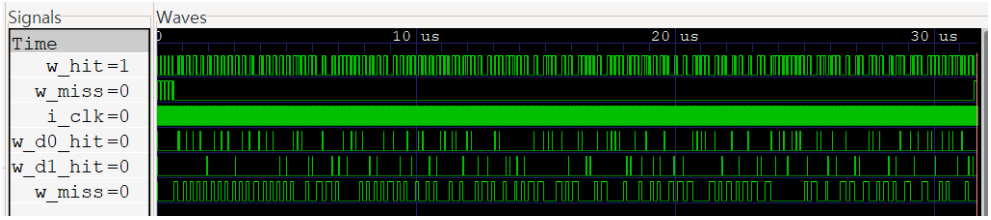
<BS 실행 결과>



bubble sort는 data를 순차적으로 읽어 사용하기 때문에, data cache에서 main memory에 존재하는 16개의 data를 처음 사용하기 위한 Compulsory miss가 4번 발생한다. 프로그램 전체에서 사용하는 16개의 data가 모두 DM 내부에 존재하기 때문에 다른 miss는 발생하지 않는다. 또한 연속된 4개의 data block만을 이용하고, cache는 16개의 set을 가지므로, set의 두 번째 block을 이용하지 않는다. 따라서 w_d1_hit가 1인 경우가 발생하지 않았다.

이 경우 bubble sort에 적합한 cache는 2-way set associative대신 direct mapping을 이용하여 cache의 속도를 약간 개선한 cache일 것이다. 또한 block size를 줄이는 것을 생각해볼 수 있다. 이 경우 compulsory miss가 증가하지만, 전체적인 cache read와 write속도를 높일 수 있기 때문에 전체적인 속도의 개선을 기대할 수 있을 것이다.

<RND 실행 결과>



그러나 random access program은 bubble sort와는 달리 data를 연속적으로 읽지 않고, 많은 수의 data에 접근한다. 따라서 compulsory miss와 더불어 capacity miss, conflict miss도 발생한다. 또한 set의 두 번째 block을 이용하여 w_d1_hit이 1로 나타나는 경우가 발생하는 것을 확인할 수 있다.

이 경우 spatial locality가 적다. block의 크기를 줄여서 cache 속도 개선을 기대할 수 있을 것이다. 기존의 2-way set associative대신 4-way를 이용하여 capacity miss와 conflict miss를 줄이는 것을 기대할 수 있다.

두 program 모두 instruction cache는 처음 명령어를 불러온 이후 cache에 저장된 명령어를 반복하여 사용한다. 따라서 프로그램 실행 초반에 compulsory miss가 발생하고 다른 miss는 발생하지 않는다.

iv. what if data size increase?

Bubble sort program에서 정렬하는 data의 수가 많아진다면 set의 두 번째 block도 이용할 수 있을 것이다. 그러나 연속적으로 데이터에 접근하는 프로그램의 특성상 directed mapping을 이용하는 것이 유리할 것이다.

B. 실험2

i. anagram, go, perl

anagram은 주어진 문자의 순서를 바꾸어 조합하는 프로그램이다. input의 문자를 읽어 words에 존재하는 순서로 재배열하여 그 결과를 출력한다. go는 바둑을 실행하는 프로그램이고, 처음 바둑을 두는 위치가 적힌 파일을 읽어 실행한다. perl은 프로그래밍 언어인 Perl언어의 interpreter프로그램이다. Perl 코드 내용이 적힌 파일을 읽어 실행한다.

ii. 알고리즘의 특징에 따른 cache access 차이

명령어 당 data에 접근하는 수에 따라 효율적인 cache가 다를 수 있다. 명령어에 의해 일어나는 miss가 data에 의해 일어나는 miss보다 적은 프로그램에서, data에 자주 접근하지 않는다면 separate cache를 사용하여 instruction cache의 속도를 높이는 것이 유리하다.

알고리즘의 spatial locality가 높을 경우 L2 cache의 크기를 줄이고, L1 cache의 크기를 증가시키는 것이 유리할 수 있다. L1 cache의 크기를 키우는 것을 통해 L2 cache에 접근하는 수를 줄일 수 있다. 반면 알고리즘의 Temporal locality가 높은 경우 cache에서 사용하는 bolck의 크기를 줄여 cache에 읽고 쓰는 시간을 줄이는 방법을 생각할 수 있다.

iii. 실험 결과

1. Unified vs splits

(1) anagram

cache의 크기가 커질수록 miss rate가 감소하였다. set이 64개에서 128개로 증가하였을 때 instruction cache의 miss rate가 크게 감소한 것을 확인할 수 있다.

# of sets	unified cache miss rate	unified cache AMAT	split cache		split cache AMAT
			inst. miss rate	data miss rate	
64	0.1390	28.8	0.1539	0.1577	(i) 31.78 (d)32.54
128	0.0614	13.8112	0.0304	0.1008	(i)7.3632 (d)22.006
256	0.0357	8.8042	0.0247	0.0690	(i)6.4247 (d)16.008
512	0.0182	5.2193	0.0095	0.0444	(i)3.2621 (d)11.114

(2) go

instruction cache는 unified cache와 유사한 miss rate를 갖고, data cache는 그보다 낮은 miss rate를 가지는 것을 확인할 수 있다.

# of sets	unified cache miss rate	unified cache AMAT	split cache		split cache AMAT
			inst. miss rate	data miss rate	
64	0.4043	81.86	0.4010	0.3975	(i)81.2 (d)80.5
128	0.3355	70.824	0.3409	0.2981	(i)71.9472 (d)63.0448
256	0.2675	58.9472	0.2825	0.2095	(i)62.192 (d)46.4006
512	0.2118	48.7741	0.2236	0.1358	(i)51.4288 (d)31.6762

(3) perl

instruction cache와 data cache의 AMAT이 큰 차이를 보이는 것을 확인할 수 있다.

# of sets	unified cache miss rate	unified cache AMAT	split cache		split cache AMAT
			inst. miss rate	data miss rate	
64	0.4339	87.78	0.4837	0.2654	(i)97.74 (d)54.08
128	0.3554	74.9632	0.4352	0.1979	(i)91.5616 (d)42.2032
256	0.2730	60.1370	0.3580	0.1418	(i)78.5242 (d)31.7558
512	0.1996	46.0294	0.2711	0.0943	(i)62.1150 (d)22.3398

2. L1/L2 Size

(1) anagram

L1 cache의 크기가 증가할수록 L1의 AMAT이 감소하여 L1I는 64에서, L1D는 32에서 최소를 기록하고, 이후 다시 AMAT이 증가하는 것을 확인할 수 있다.

L1I/L1D/L2U	inst.miss rate	data.miss rate	unified cache miss rate	AMAT
8/8/1024	0.4284	0.3737	0.0028	(i)9.8079 (d)8.6833
16/16/512	0.3065	0.2729	0.0065	(i)7.8296 (d)7.0853
32/32/256	0.1539	0.1577	0.0215	(i)5.1265

				(d)5.2264
64/64/128	0.0304	0.1008	0.1584	(i)2.8921 (d)6.9847
128/128/0	0.0247	0.0690		(i)6.949 (d)17.3139

(2) go

go program의 경우 L1 cache의 크기가 증가할수록 AMAT이 커지는 결과를 확인할 수 있었다.

L1I/L1D/L2U	inst.miss rate	data.miss rate	unified cache miss rate	AMAT
8/8/1024	0.4954	0.5824	0.0480	(i)15.6638 (d)18.239
16/16/512	0.4574	0.4922	0.0974	(i)19.8205 (d)21.2493
32/32/256	0.4010	0.3975	0.1807	(i)25.4307 (d)25.2182
64/64/128	0.3409	0.2981	0.2887	(i)30.9355 (d)27.1928
128/128/0	0.2825	0.2095		(i)67.2669 (d)50.1869

(3) perl

go program과 유사하게 L2 cache의 크기가 줄어들수록 AMAT의 크기가 커지는 것을 확인할 수 있다.

L1I/L1D/L2U	inst.miss rate	data.miss rate	unified cache miss rate	AMAT
8/8/1024	0.5452	0.4261	0.0367	(i)15.9058 (d)12.6496
16/16/512	0.5254	0.3472	0.0774	(i)20.4268 (d)13.8514
32/32/256	0.4837	0.2654	0.1597	(i)28.2551 (d)15.9913
64/64/128	0.4352	0.1979	0.2743	(i)37.7719 (d)17.7895
128/128/0	0.3580	0.1418		(i)84.9317 (d)34.3471

3. Associativity

(1) anagram

set associative를 이용한 경우 set의 way 수가 많아질수록 최적의

AMAT값을 갖는 cache의 set의 수가 적어졌다. cache의 크기가 커질수록 miss rate가 감소하지만, data를 읽고 쓰는데 필요한 시간이 길어지기 때문에 최적의 AMAT은 다음과 같이 나타났다.

# of sets	split cache mis srate/ AMAT							
	1-way		2-way		4-way		8-way	
64	0.1539 0.1577	31.78 32.54	0.0344 0.0781	8.0376 16.9524	0.0166 0.0540	4.4945 12.276	0.0023 0.0287	1.5494 7.1525
128	0.0304 0.1008	7.3632 22.006	0.0166 0.0547	4.5827 12.666	0.0036 0.0290	1.861 7.3578	0.0003 0.0119	1.1699 3.7304
256	0.0247 0.0690	6.4247 16.008	0.0077 0.0313	2.8022 8.0095	0.0003 0.0122	1.1928 3.871	0.0002 0.0109	1.1937 3.65
512	0.0095 0.0444	3.2621 11.114	0.0035 0.0143	1.9505 4.4288	0.0002 0.0109	1.2171 3.7216	0.0001 0.0108	1.2176 3.7721
1024	0.0075 0.0318	2.9246 8.6102	0.0014 0.0113	1.5274 3.89	0.0001 0.0108	1.2415 3.5559	0.0001 0.0107	1.2663 3.8982
2048	0.0020 0.0137	1.7033 4.5503	0.0001 0.0108	1.2658 3.9215	0.0001 0.0107	1.2911 3.9746	0.0001 0.0106	1.3169 4.0283

(2) go

go program을 실행한 결과 항상 가장 많은 set을 가진 cache의 성능이 가장 뛰어났다.

# of sets	split cache miss rate/ AMAT							
	1-way		2-way		4-way		8-way	
64	0.4010 0.3975	81.2 80.500	0.3336 0.2581	69.0744 53.6724	0.2690 0.1393	57.014 30.026	0.1982 0.0592	43.127 13.626
128	0.3409 0.2981	71.947 63.044	0.2715 0.1607	58.662 35.155	0.2006 0.0686	44.492 15.927	0.1531 0.0279	34.898 7.262
256	0.2825 0.2095	62.192 46.401	0.2102 0.0916	47.483 21.314	0.1541 0.0332	35.807 8.5973	0.1049 0.0121	25.229 3.9255
512	0.2236 0.1358	51.429 31.676	0.1587 0.0457	37.565 11.634	0.1054 0.0139	25.84 4.4238	0.0461 0.0040	12.2 2.1487
1024	0.1872 0.0798	44.969 19.841	0.1136 0.0221	28.304 6.4674	0.0503 0.0052	13.461 2.4829	0.0211 0.0013	6.4804 1.5642
2048	0.1474 0.0390	37.084 10.707	0.0700 0.0069	18.615 2.9535	0.0231 0.0015	7.1138 1.6455	0.0163 0.0003	5.5002 1.3686

(3) perl

perl program 실행 결과 anagram의 실행결과와 유사하게 가장 효율이 좋은 크기 이후로 다시 AMAT의 값이 증가하였다.

# of sets	split cache miss rate/ AMAT							
	1-way		2-way		4-way		8-way	
64	0.4837 0.2564	97.74 52.28	0.4330 0.1613	89.352 33.925	0.3348 0.0856	70.706 18.852	0.2038 0.0482	44.316 11.291
128	0.4352 0.1979	91.562 42.203	0.3518 0.1056	75.699 23.465	0.2164 0.0532	47.912 12.595	0.1064 0.0193	24.589 5.3638
256	0.0358 0.1418	8.8259 31.756	0.2431 0.0646	54.742 15.357	0.1196 0.0218	28.043 6.0316	0.0331 0.0015	8.7463 1.4921
512	0.2711	62.115	0.1393	33.113	0.0422	11.048	0.0032	1.9577

	0.0943	22.34	0.0292	7.848	0.0034	1.9661	0.0003	1.2653
1024	0.1836 0.0522	44.127 13.383	0.0649 0.0098	16.682 3.532	0.0088 0.0005	3.3593 1.3388	0.0002 0.0003	1.2911 1.316
2048	0.1101 0.0287	28.007 8.2002	0.0324 0.0017	9.2826 1.6629	0.0028 0.0003	1.9747 1.3418	0.0002 0.0003	1.3428 1.3689

4. Block size

(1) anagram

block size가 증가할수록 miss rate와 AMAT이 모두 감소하지만 block size가 512일때는 오히려 miss rate와 AMAT이 증가하였다.

block size	unified cache miss rate	AMAT
16	0.182	37.4
32	0.0055	2.184
64	0.0024	1.6008
128	0.0012	1.3948
256	0.0008	1.3570
512	0.0151	4.8909

(2) go

block size가 증가할수록 miss rate와 AMAT이 모두 감소하였다.

block size	unified cache miss rate	AMAT
16	0.2118	43.36
32	0.1077	23.4416
64	0.0489	11.6597
128	0.0217	6.0068
256	0.0106	3.65
512	0.0058	2.628

(3) perl

block size가 증가할수록 miss rate와 AMAT이 모두 감소하였다.

block size	unified cache miss rate	AMAT
16	0.1996	40.92
32	0.0913	20.0304
64	0.0396	9.6479
128	0.0195	5.5118
256	0.0147	4.6092
512	0.0039	2.1656

2. 분석 및 결과

A. anagram

실험 1에서 split cache에서 data cache의 AMAT이 instruction cache의 AMAT보다 상대적으로 높게 나타나기 때문에 anagram program에서 명령어당 접근하는 data의 수가 적을수록 split cache는 높은 효율을 보일 것이다. 실험 2에서 anagram program에서 data에 자주 접근한다면 L1의 크기를 32로 하는 것이 유리하고, 아닌 경우 64로 지정하여 instruction cache의 AMAT을 낮추는 것이 유리할 것이다. 실험 3에서 set이 32 혹은 64일 때 적절한 것은 8way 혹은 그 이상임을 알 수 있다. 실험 4에서 block size가 큰 것이 유리할 것이다.

위의 실험 결과를 통해 anagram은 나머지 두 프로그램인 go와 perl보다 spatial locality가 강하게 나타나는 것으로 추측할 수 있으며, 사용하는 data가 다른 두 프로그램에 비해 적을 것이다.

B. go

실험 1에서 instruction cache의 miss rate가 높은 것을 통해 go program의 수행에 branch가 빈번하게 쓰이는 것으로 추측할 수 있다. data cache의 AMAT이 instruction cache의 AMAT보다 낮기 때문에 이용하는 data의 수가 많을수록 split cache의 효율이 높아질 것이다. 실험 2에서 L2 cache의 크기가 큰 것이 더욱 유리한 것을 통해 data와 instruction의 spatial locality가 적을 것으로 추측할 수 있다. 실험3에서 작은 크기의 L1을 이용할 때 8way 혹은 그 이상이 AMAT을 줄이는 것을 확인할 수 있다. 실험 4에서 block size가 큰 것이 유리한 것을 확인할 수 있다.

실험 결과를 통해 go 프로그램은 다른 프로그램에 비해 사용하는 data의 수가 많으며, spatial locality가 낮게 적용되는 것으로 추측할 수 있다. 그렇기 때문에 프로그램에 적합한 캐시의 크기가 다른 두 프로그램보다 크게 나타났다.

C. perl

실험 1에서 go 프로그램과 마찬가지로 instruction cache의 miss rate가 높은 것을 통해 branch를 빈번하게 사용하는 것으로 추측할 수 있다. data에 자주 접근할수록 split cache의 효율이 높아질 것이다. 실험 2에서 L2 cache의 크기가 큰 것이 유리하고, 실험3에서 작은 L1에 적절한 것은 8 way 혹은 그 이상이었다. 마지막으로 실험 4에서 block size가 큰 것이 유리했다.

perl 프로그램은 실험에서 go와 유사한 결과가 나타났다. 그러나 실험3의 결과에서 anagram프로그램의 결과와 유사한 점을 통해 go 프로그램 보다는 적은 양의 cache를 이용하여 효율적으로 동작할 수 있을 것이라 추측하였다.

3. 고찰

Bubble sort와 random access 프로그램의 cache 동작을 확인하는 실험은 큰 어려움 없이 진행할 수 있었다. 그러나 bench mark 프로그램에 적합한 cache를 찾는 실험은 결과를 분석하는 과정에서 어려움을 겪었다. 각각 실험을 통해 얻은 miss rate와 AMAT

결과를 이해하고, 세 가지 프로그램의 특성에 맞는 cache를 찾는 과정에서 적절한 방법을 사용하지 못한 것 같다. 각 프로그램의 특성을 자세히 알지 못하기 때문에 AMAT을 단순하게 비교하여 적용한 점이 아쉬웠다. Cache의 기본 구조에 대해 이해한 뒤, 해당 실험을 진행하여 cache의 구성에 따라 그 성능이 변화하는 것을 확인할 수 있는 과제였다.