컴퓨터구조

Project #4

Cache Design

Class : **월 3 수 4**

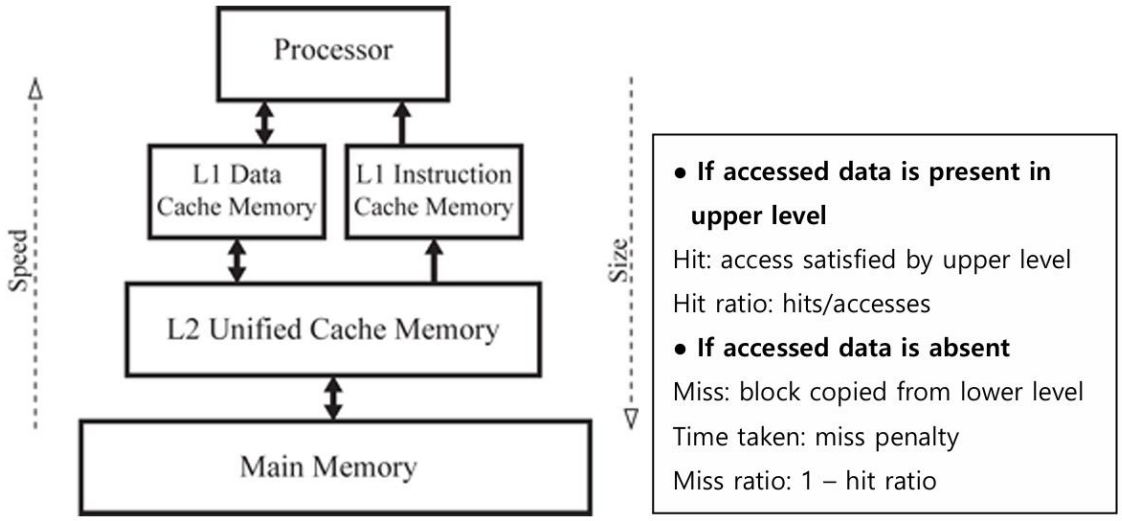
Professor : **이성원 교수님**

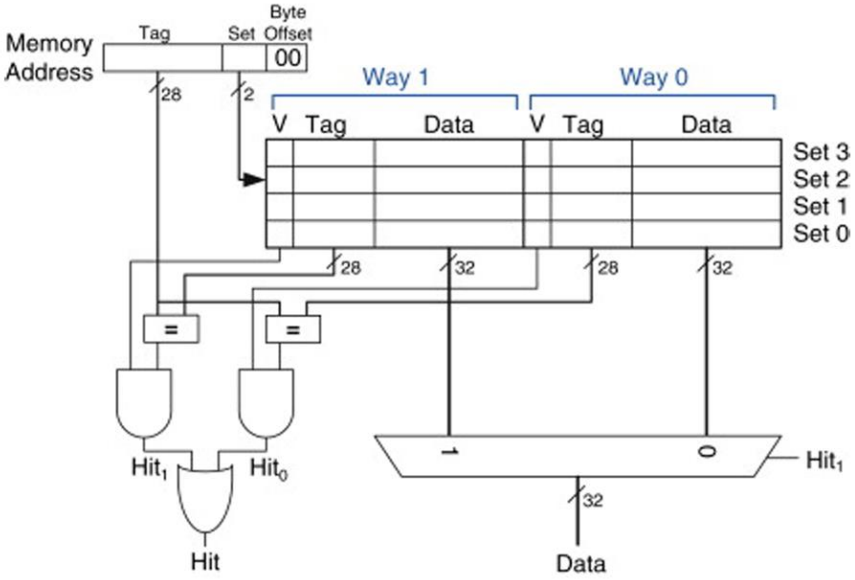
Student ID : **2020202031**

Name : **김재현**

1. Introduction

Cache is a smaller, faster memory which stores copies of the data from the most frequently used min memory locations





- Higher associativity increases the chance that the latest accessed block stays in the cache.

- Data accesses often show more irregular access patterns than instruction accesses do, which lead to more associativity.

2. Assignment

For the first phase, you have to check the timing and discuss about:

MIPS Instruction 의 크기는 4bytes 입니다.

IM 은 4byte 크기 4 개 즉, 16byte block 이 32 개 존재합니다. 따라서 offset size 는 log_2 16 = 4bits, Index size 는 log_2 32 = 5bits, tag size 는 32 - 5 - 4 = 23bits 입니다.

DM 은 4byte 크기 4 개 즉, 16byte block 이 16 개씩 2 way 로 존재합니다. 따라서 offset size 는 IM 과 마찬가지로 4bits, index size 는 log_2 16 = 4bits, tag size 는 32 - 4 - 4 = 24bits 입니다.

정리)

IM tag : 23bits index : 5bits, offset : 4bits,

DM tag : 24bits index : 4bits, offset : 4bits,

block size 가 16bits 이므로 MM 에서 cache 로 block 을 write 할 때는 0xXXXXXXX0 번지에서 block 을 가져옵니다.

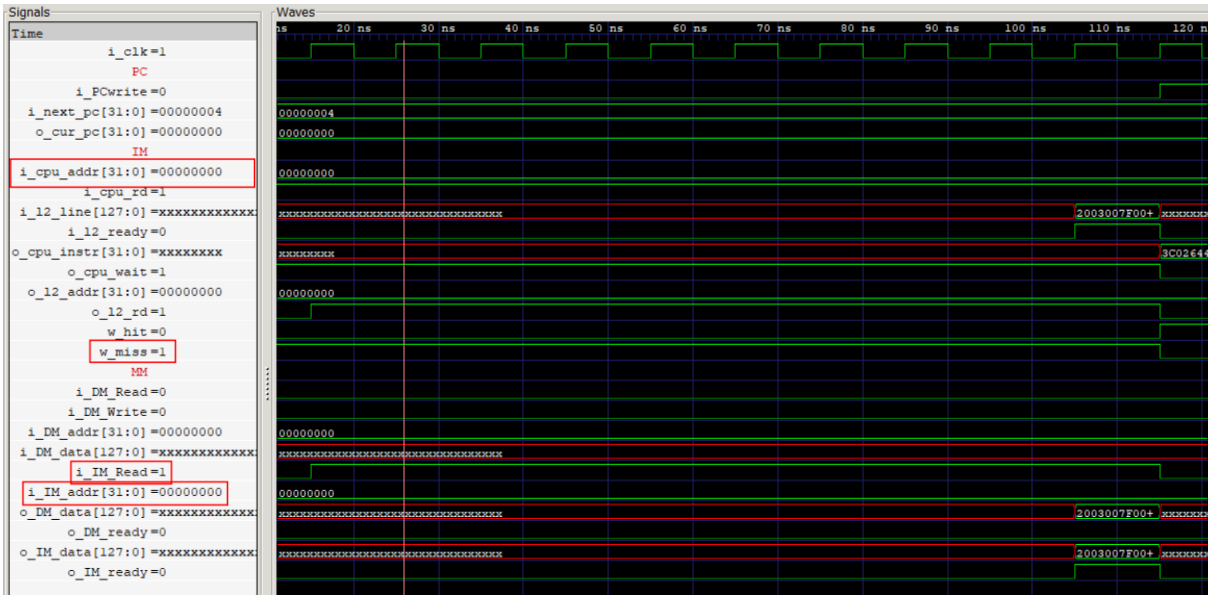
random_access 예제를 통해 설명하겠습니다.

■ How I/D L1 hit operates, how I/D L1 miss operates

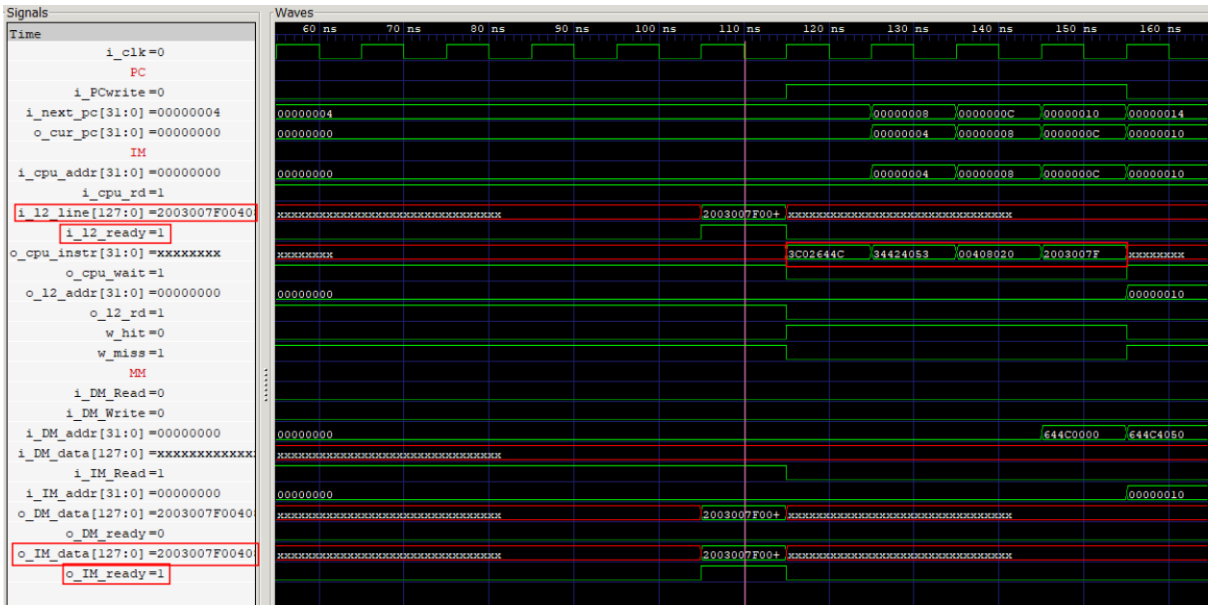
■ When and how main memory (L2) operates

IM hit, miss)

16 진수로 Instruction address 가 (xxxxx0aa, xxxxx1aa), (xxxxx2aa, xxxxx3aa), (xxxxx4aa, xxxxx5aa) ….. (xxxxxeaa, xxxxxoaa) pair 들은 index 가 같기 때문에 동시에 cache 에 존재할 수 없습니다. (x is don't care, e is even, o is odd, aa 는 임의의 수입니다.)

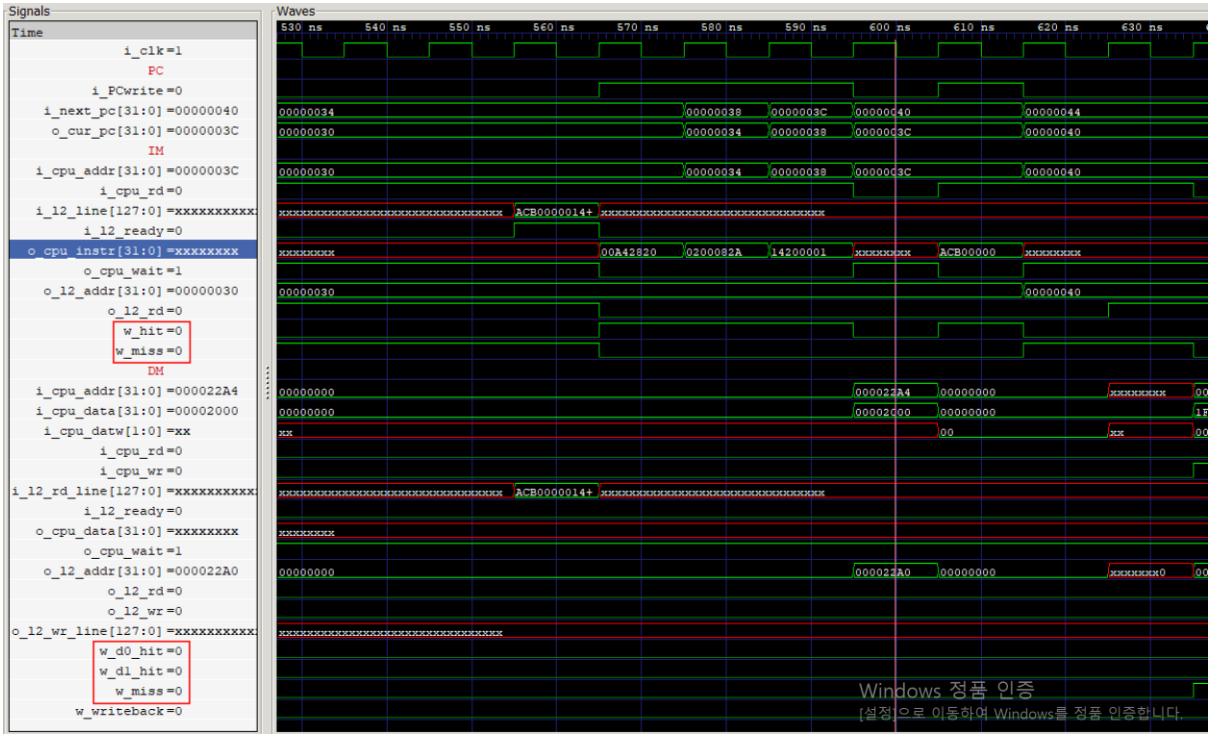


프로그램이 처음 실행되고 IM 0x0 번지에 valid 한 값이 존재하지 않으므로 miss 이고, 따라서 MM 의 IM_READ = 1, IM_addr = 0x0 으로 설정되며, MM 내에 존재하는 IM 0x0 번지 명령어 데이터를 탐색합니다.

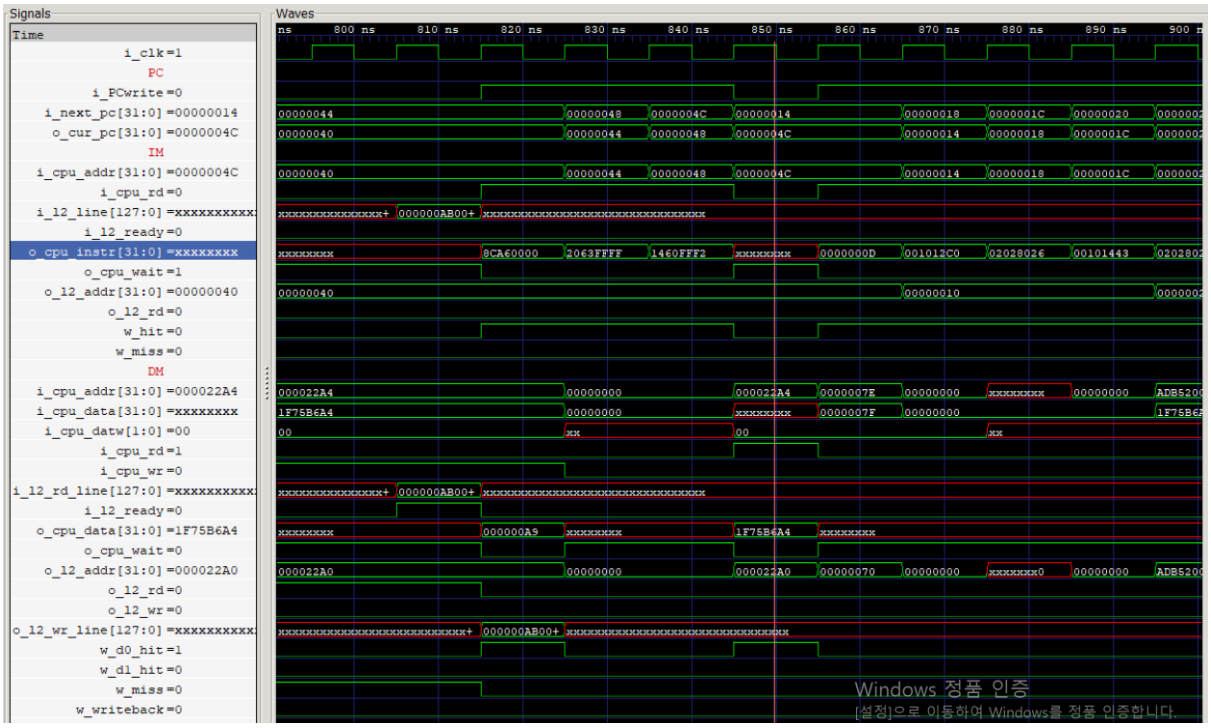


11clock 후에 MM 에서 IM 0x0 번지 데이터를 찾았으므로, IM_ready, l2_ready = 1 이 되고, 그 데이터 값은 2003007F... 입니다. 다음 clock 에서 IM 에 해당 데이터가 write 되므로 바로 0x0 명령어를 실행할 수 있게됩니다. 데이터는 2003007F....로 read 했지만 2003007F 이 마지막에 실행되는 것으로 보아 해당 시뮬레이션은 little endian 을 사용한다는 것을 알 수 있습니다. 한 block 의 크기가 16bytes 이므로 한

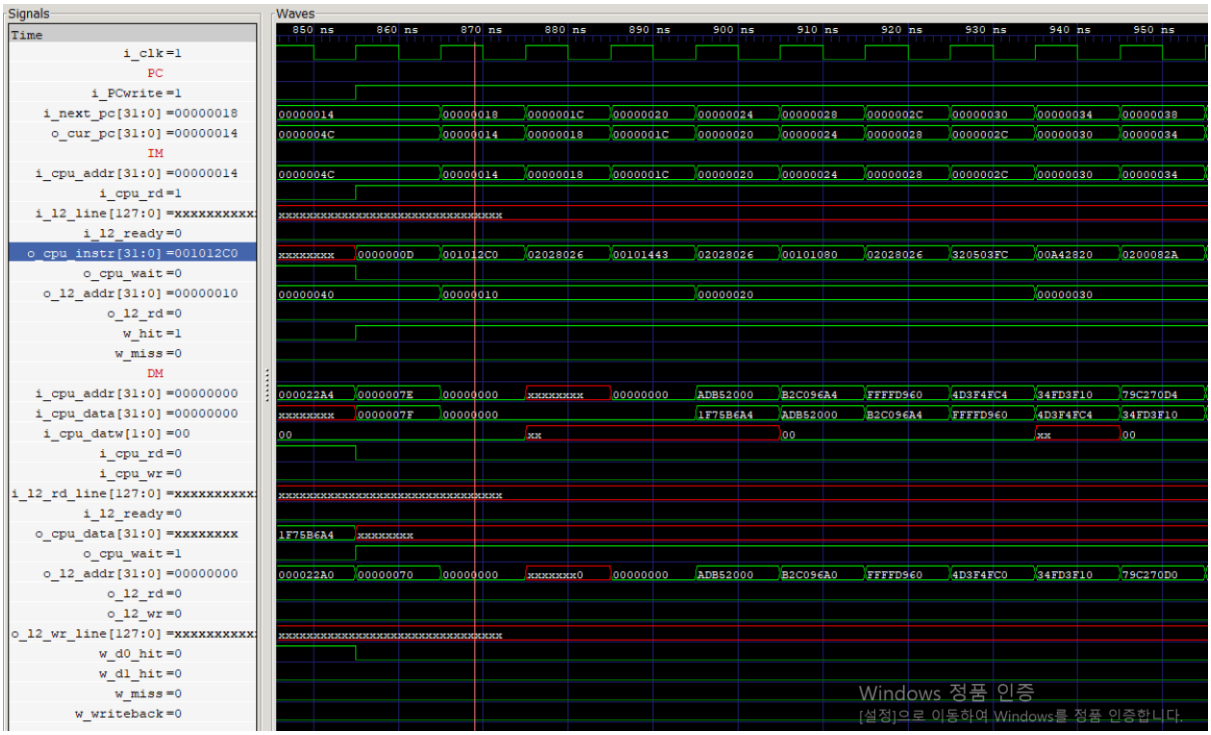
번에 불러올 수 있는 명령어는 4 개입니다. 따라서 4 개의 명령어를 수행한 후, 다음 PC 에서는 다시 MM 으로부터 값을 불러오기 위해 wait 상태가 됩니다.



다음은 IM 과 DM 이 모두 hit, miss 가 0 인 상황입니다. 해당 명령어의 opcode 를 살펴보면, 000101 로, bne 명령어임을 알 수 있습니다. bne 는 ID stage 에서 브랜치 여부를 계산하므로, 다음 clock 에서는 I/D memory 에 접근하지 않은 것임을 알 수 있습니다.

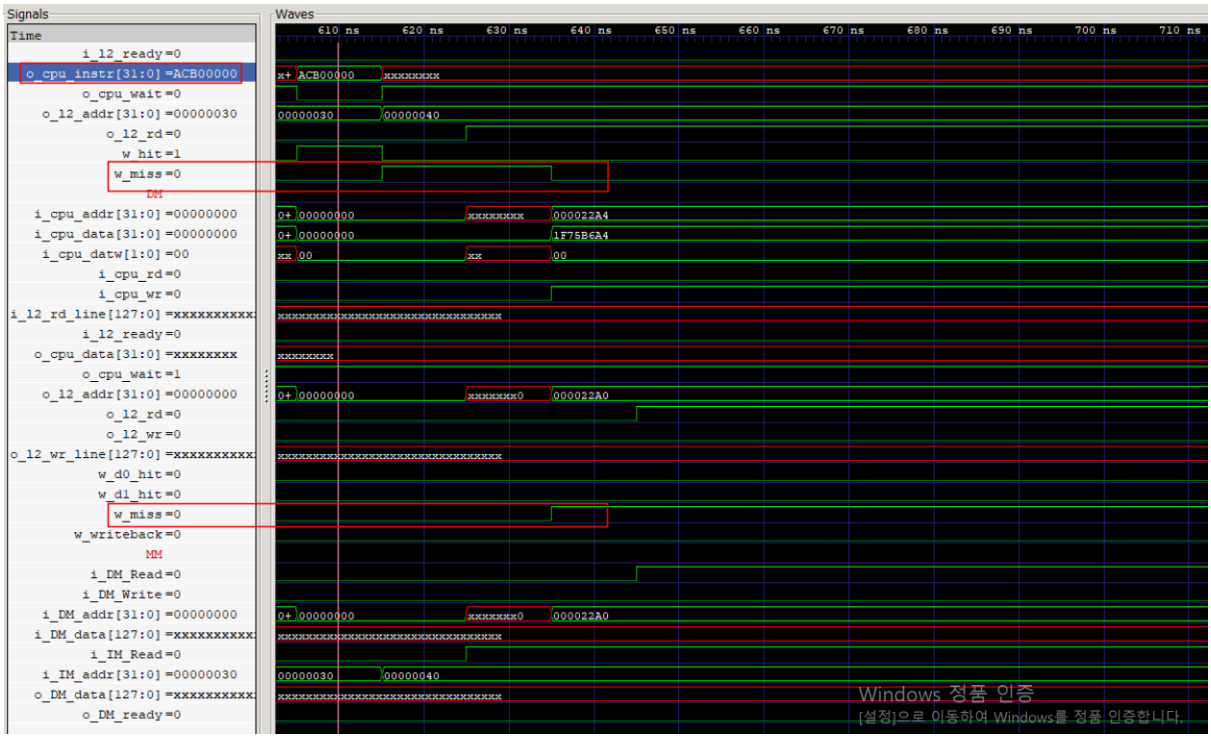


위 사진도 역시 bne 명령어이고, ID stage 에서 브랜치 여부를 계산한 결과, jump 로 판정되어 0x14 번지로 jump 한 것을 확인할 수 있습니다.

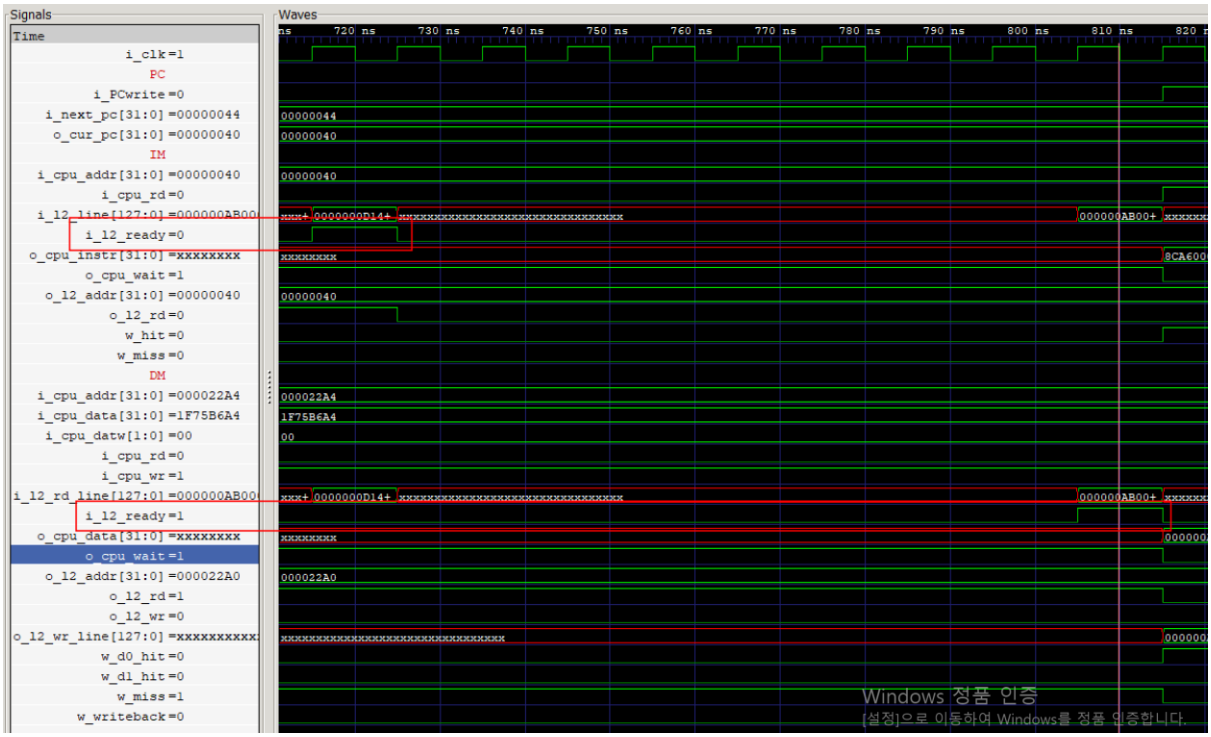


jump 이후론, 이미 IM 에 fetch 된 명령어들을 재사용하므로, 계속해서 hit 가 나오는 것을 알 수 있습니다.

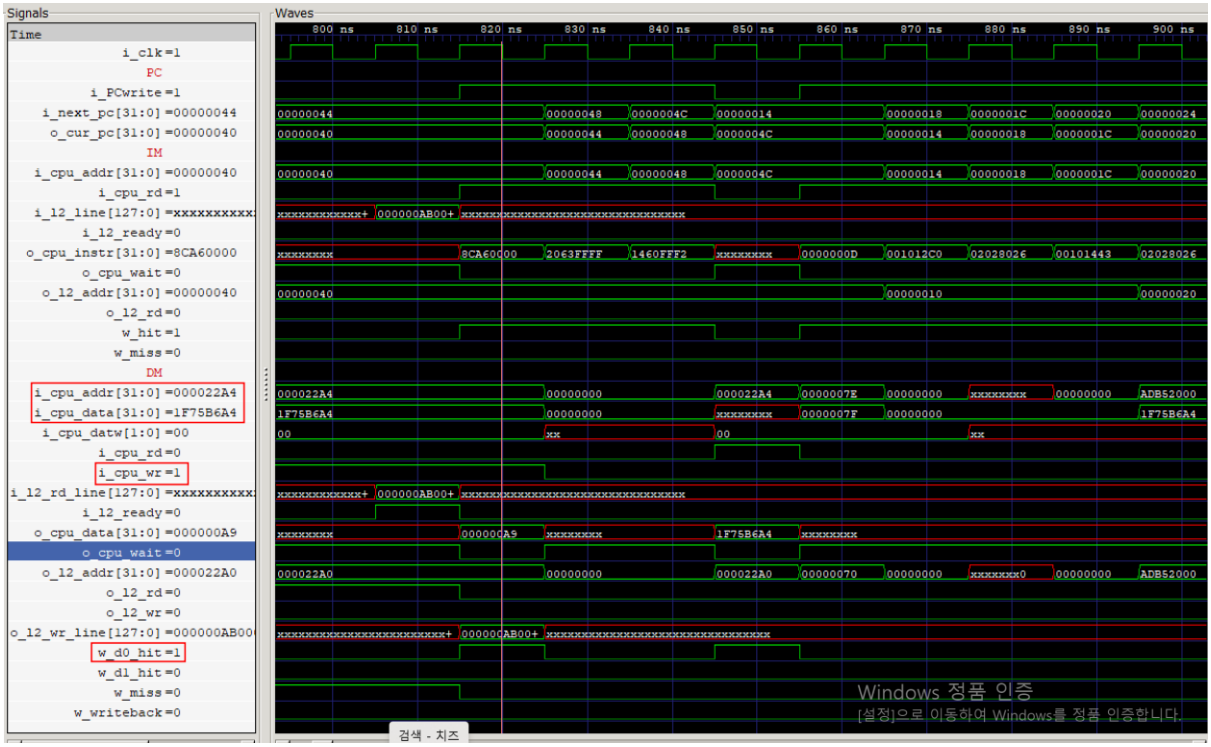
DM hit, miss)



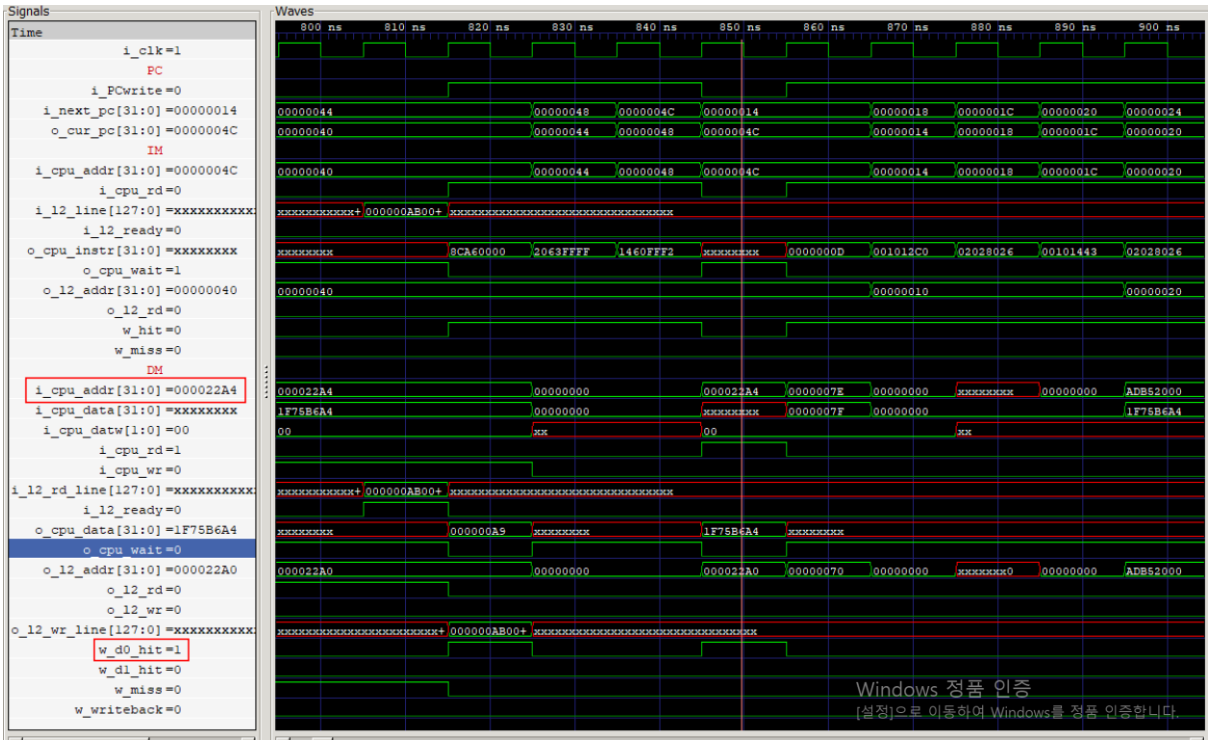
0x3C 번지 Instruction 은 sw 명령어입니다. sw 다음 0x40 번지 instruction 이 miss 기 때문에 MM 에서 탐색하다가 sw 명령어가 memory access stage 에서 miss 가 나오자 IM 의 hit 와 miss 가 둘 다 0 이 되고, DM 의 miss 가 1 이 됩니다. 하지만 이는 IM 의 탐색을 멈춘다는 의미는 아니고, IM 과 DM 의 탐색을 동시에 하되, DM 의 탐색이 끝날 때까지 wait 한다는 의미인 것을 확인할 수 있습니다.



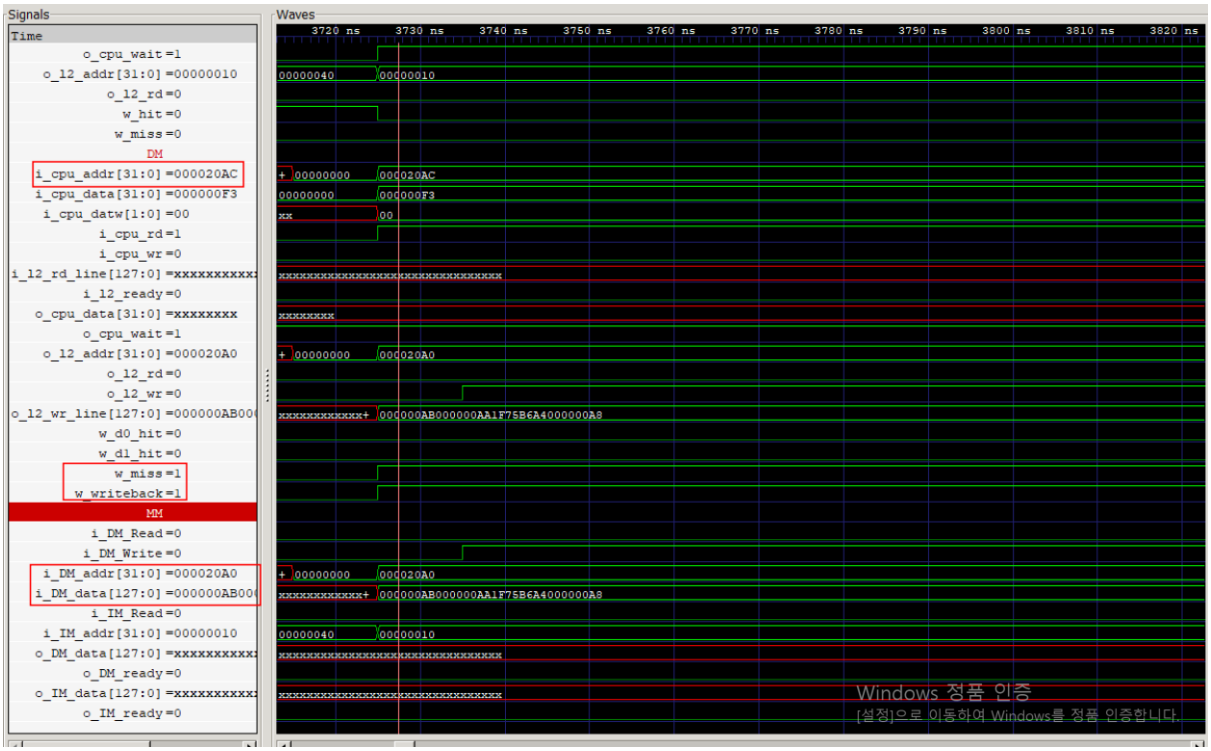
위 예제에서는 instruction 이 data 보다 먼저 탐색된 것을 확인할 수 있습니다.



DM 에 값이 저장된 후 sw 의 memory access stage 에서 write 가 시행되고 0x40 번지 instruction 이 fetch 되는 것을 확인할 수 있습니다.



0x40 번지 instruction 은 lw 명령어입니다. memory access stage 에서 sw 한 addr 와 같은 addr 의 값을 lw 하므로, DM hit 가 1 이 된 것을 확인할 수 있습니다.



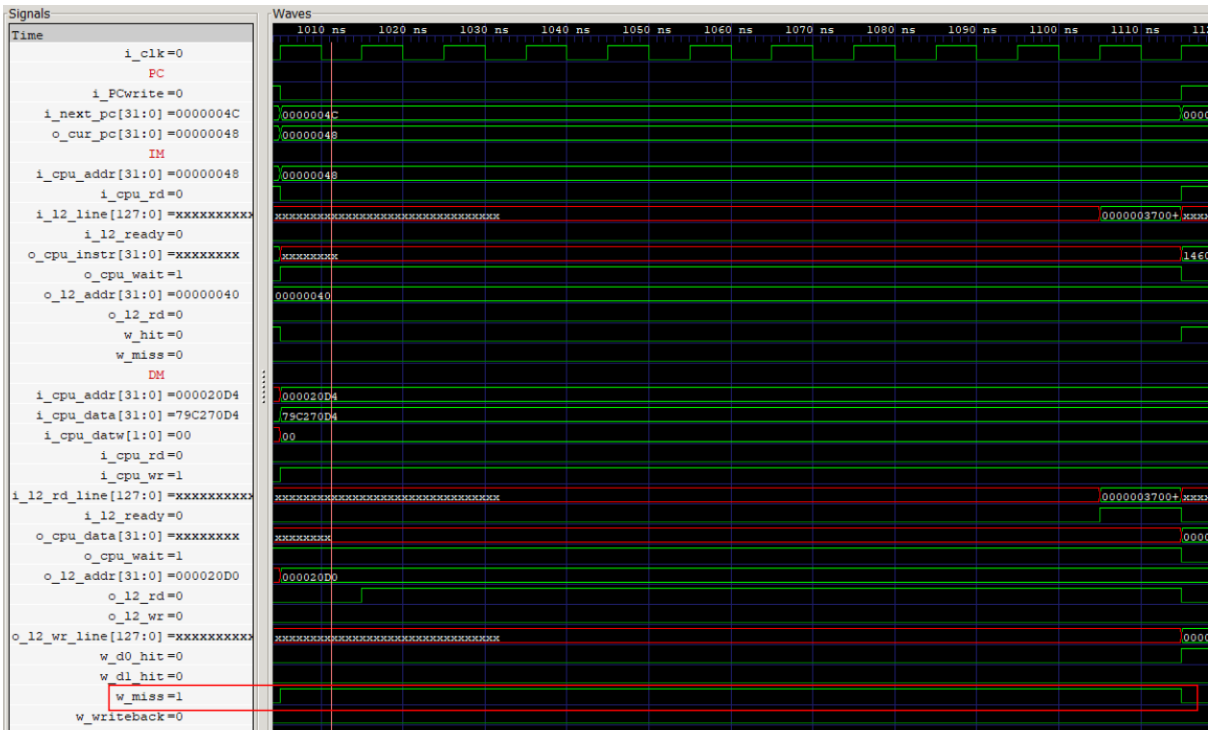
lw 명령어가 0x000020AC 번지 data 를 load 하려고 하는데 cache 에 해당 주소의 data 가 존재하지 않아서 miss 가 1 이 됩니다. 게다가 cache 의 1, 2way cache 에 이미 다른 주소의 유효한 값들이 존재하고 있어서 writeback 신호도 1 이 된 것을 확인할 수 있습니다.

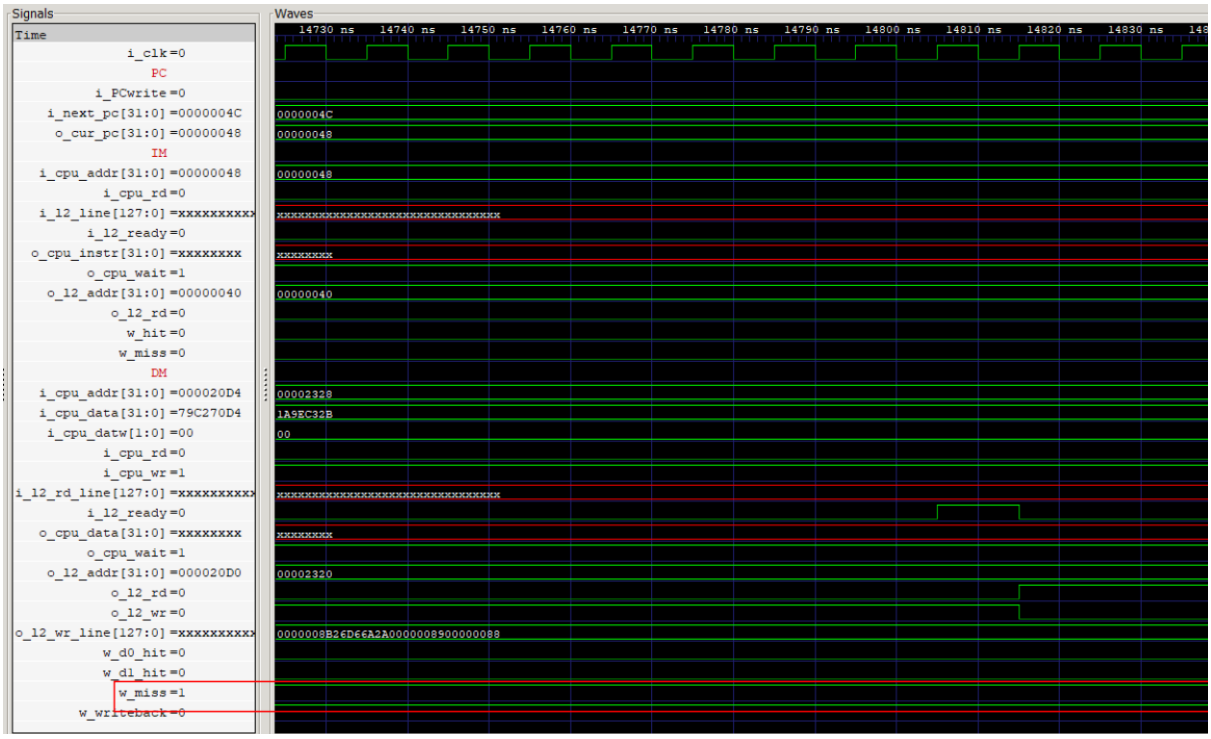
■ The differences between Bubble sort and Random access in terms of cache behaviors

bubble sort 의 경우 인접한 데이터들 간 크기 비교를 통해 sort 를 진행합니다.

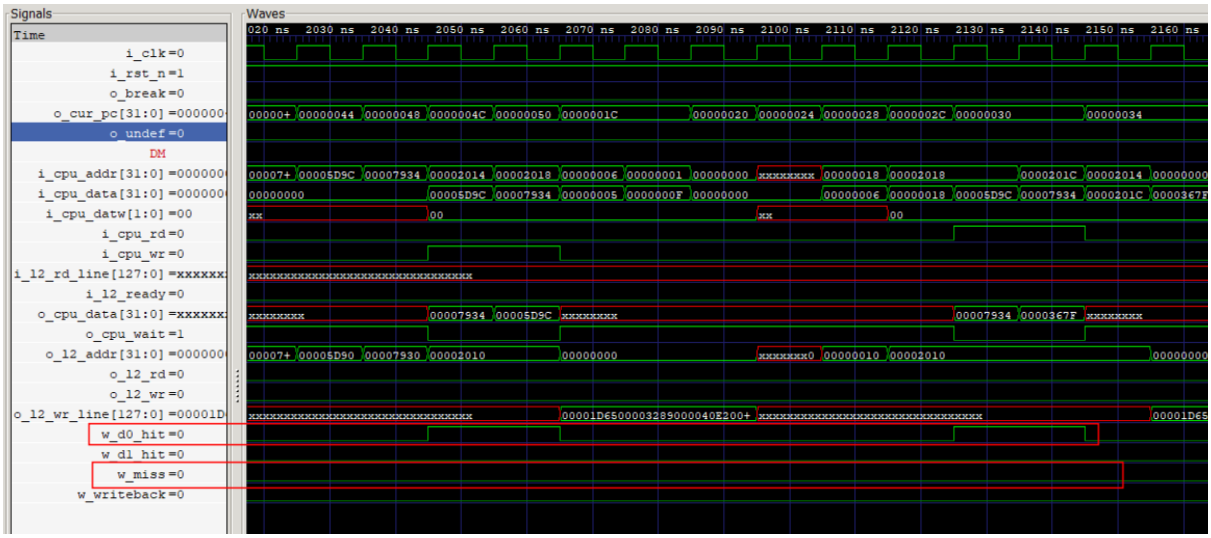
반면에 random access 는 random 한 주소의 데이터에 접근합니다.

따라서 bubble sort 가 random access 에 비해서 DM hit 가 더 많이 발생합니다.





random access 는 DM miss 가 계속해서 많은 빈도로 발생하는 것을 확인할 수 있습니다.



bubble sort 의 경우, 처음 한 사이클을 돌면, sort 에 필요한 데이터들이 이미 cache 에 모조리 들어가게 되므로, 한 사이클 이후엔 DM miss 가 발생하지 않는 것을 확인할 수 있습니다.

■ What if data size increase?

data size 가 block data size 라고 가정하겠습니다.

cache data size 가 동일하다고 가정하면, block data size 가 증가해도, offset size 가 증가하기 때문에 Tag size 는 동일합니다. 따라서 hit rate 자체는 알고리즘에 의존하게 됩니다.

block data size 가 늘어나면, MUX 의 크기가 늘어나기 때문에 data hit 이 발생했을 때, 해당 주소의 data 를 가져오는데 걸리는 시간이 증가하게 됩니다.

■ The original purpose of the benchmark programs given (not to measure the performance)

-li benchmark: used in Language interpreter. It is used for Lisp interpreter.

-vortex: used in Database. It is used for building and manipulating three interrelated databases.

-tomcatv: used in Geometric Translation. It is used for Generation of a two-dimensional vectorized mesh.

■ Discuss about the memory access patterns of the benchmark programs in the aspect of algorithm (for example: there is a matrix multiplication, so the program may show 2d array access)

Language interpreter 는 재귀적으로 접근합니다. 재귀는 곧 스택 구조이므로, 한 번 불러온 데이터를 반복해서 사용하게 됩니다. 따라서 li benchmark 는 non-random-access 방식으로 memory 에 접근합니다.

Database 는 방대한 양의 data 입니다. 이러한 Data 에서 내가 원하는 data 를 얻기 위해서는 random 한 위치의 data 에 접근할 수 있어야 합니다. 따라서 vortex benchmark 는 random-access 방식으로 memory 에 접근합니다.

Geometric Translation 은 행렬합, 행렬곱 등 행렬연산을 진행합니다. 행렬 연산은 주어진 몇 가지의 행렬만으로 진행되므로, non-random-access 방식으로 memory 에 접근합니다.

■ Perform experiments with several cache configurations and discuss about the results

SIMULATION 1 . Unified vs splits

li benchmark

# of Sets	Unified Cache Miss rate	Unified Cache AMAT	Split Cache		Split Cache AMAT
			Inst. Miss rate	Data Miss rate	
64	0.3108	63.16	0.3609	0.167	61.62165437
128	0.2687	54.74	0.299	0.1205	50.15964572
256	0.1325	27.5	0.2743	0.0862	44.64739138
512	0.0679	14.58	0.102	0.0643	19.15270949

vortex benchmark

# of Sets	Unified Cache Miss rate	Unified Cache AMAT	Split Cache		Split Cache AMAT
			Inst. Miss rate	Data Miss rate	
64	0.2057	42.14	0.2107	0.1258	38.94649736
128	0.1797	36.94	0.189	0.0917	33.99401876
256	0.1426	29.52	0.1746	0.0456	29.54824687
512	0.1018	21.36	0.1317	0.0393	22.77604659

tomcatv benchmark

# of Sets	Unified Cache Miss rate	Unified Cache AMAT	Split Cache		Split Cache AMAT
			Inst. Miss rate	Data Miss rate	
64	0.2286	46.72	0.2019	0.2133	42.13961059
128	0.1993	40.86	0.1954	0.2051	40.72633533
256	0.1771	36.42	0.1698	0.2003	36.99229149
512	0.16	33	0.1457	0.1948	33.41165613

세 개의 testbench 모두 cache size 가 증가함에 따라 AMAT 가 줄어드는 것을 확인할 수 있습니다. 그러나 unified cache 와 split cache 둘 다 cache size 의 증가가 AMAT 에 더 큰 영향을 주는 정도는 비슷했습니다..

SIMULATION 2 . L1/L2 size

li benchmark

L1I/L1D/L2U	Inst. Miss rate	Data Miss rate	Unified Cache Miss rate	AMAT
8 / 8 / 1024	0.5154	0.3148	0.0712	16.60013237
16 / 16 / 512	0.436	0.235	0.1605	20.59439961
32 / 32 / 256	0.3609	0.167	0.3668	29.29818826
64 / 64 / 128	0.299	0.1205	0.87	48.68485635
128 / 128 / 0	0.2743	0.0862		44.64739138

vortex benchmark

L1I/L1D/L2U	Inst. Miss rate	Data Miss rate	Unified Cache Miss rate	AMAT
8 / 8 / 1024	0.2782	0.2127	0.2019	16.82098799
16 / 16 / 512	0.225	0.1594	0.4265	22.98652978
32 / 32 / 256	0.2107	0.1258	0.6395	29.0614348
64 / 64 / 128	0.189	0.0917	0.873	33.10318025
128 / 128 / 0	0.1746	0.0456		28.2303322

tomcatv benchmark

L1I/L1D/L2U	Inst. Miss rate	Data Miss rate	Unified Cache Miss rate	AMAT
8 / 8 / 1024	0.2801	0.3896	0.3896	31.9996407
16 / 16 / 512	0.212	0.25	0.536	29.57677445
32 / 32 / 256	0.2019	0.2133	0.6524	31.95344301
64 / 64 / 128	0.1954	0.2051	0.7422	34.45751961
128 / 128 / 0	0.1698	0.2003		36.77809975

L1 cache 보다 L2 cache 가 클 때 AMAT 가 비교적 작은 것을 알 수 있습니다. 이는 L2 cache 가 작아질수록 L2 miss rate 가 높아지고, main memory 에 access 해야하는 횟수가 증가하기 때문인 것으로 생각됩니다. L2 access time 보다 main memory

access time 이 10 배 높기 때문에 되도록이면 main memory access 를 줄일 수 있도록 L2 cache size 를 키워야합니다.

SIMULATION 3. Associativity

li benchmark

# of Sets	Split Cache Inst.Miss rate / Data.Miss rate / AMAT			
	1-way	2-way	4-way	8-way
64	0.3609 / 0.1670 / 61.6216	0.2214 / 0.0836 / 37.0657	0.1452 / 0.0561 / 24.7287	0.0537 / 0.0453 / 11.2392
128	0.2990 / 0.1205 / 50.1596	0.1558 / 0.0624 / 26.5924	0.0643 / 0.0459 / 12.7631	0.0055 / 0.0369 / 3.9717
256	0.2743 / 0.0862 / 44.6473	0.1113 / 0.0487 19.5284	0.0192 / 0.0374 / 5.9248	0.0001 / 0.0316 2.8977
512	0.1020 / 0.0643 / 19.1527	0.0261 / 0.0396 / 7.0247	0.0003 / 0.0321 / 2.9555	0.0000 / 0.0280 / 2.6690
1024	0.0379 / 0.0500 / 36.1257	0.0015 / 0.0333 / 3.1955	0.0000 / 0.0280 / 2.6690	0.0000 / 0.0206 / 2.2279
2048	0.0087 / 0.0401 / 4.6117	0.0001 / 0.0284 / 2.7069	0.0000 / 0.0198 / 2.1802	0.0000 / 0.0007 / 1.0417

vortex benchmark

# of Sets	Split Cache Inst.Miss rate / Data.Miss rate / AMAT			
	1-way	2-way	4-way	8-way
64	0.2107 / 0.1258 / 38.9464	0.1904 / 0.0662 / 32.9453	0.1715 / 0.0336 / 28.4886	0.1505 / 0.0260 / 24.9505
128	0.1890 / 0.0917 / 33.9940	0.1733 / 0.0392 / 29.0363	0.1402 / 0.0261 / 23.4042	0.0494 / 0.0256 / 9.7044
256	0.1746 / 0.0456 / 29.5482	0.1355 / 0.0271 / 22.7457	0.0571 / 0.0257 / 10.8690	0.0154 / 0.0250 / 4.5541
512	0.1317 / 0.0393 / 22.7760	0.0719 / 0.0258 / 13.1029	0.0166 / 0.0250 / 4.7349	0.0142 / 0.0247 / 4.3586
1024	0.0864 / 0.0316 / 15.5732	0.0278 / 0.0251 / 6.4266	0.0148 / 0.0247 / 4.4489	0.0141 / 0.0247 / 4.3435
2048	0.0431 / 0.0275 / 8.8494	0.0171 / 0.0248 / 4.8003	0.0141 / 0.0247 / 4.3435	0.0141 / 0.0247 / 4.3435

tomcatv benchmark

# of Sets	Split Cache Inst.Miss rate / Data.Miss rate / AMAT			
	1-way	2-way	4-way	8-way
64	0.2019 / 0.2133 / 42.1396	0.1835 / 0.2006 / 38.8394	0.1625 / 0.1970 / 35.7988	0.1143 / 0.1928 / 29.0906
128	0.1954 / 0.2051 / 40.7263	0.1653 / 0.1975 / 36.2055	0.1273 / 0.1926 / 30.8111	0.0975 / 0.1902 / 26.6768
256	0.1698 / 0.2003 / 36.9922	0.1287 / 0.1941 / 31.0977	0.1007 / 0.1902 / 27.1036	0.0948 / 0.1846 / 25.9435
512	0.1457 / 0.1948 / 33.4116	0.1060 / 0.1905 / 27.8304	0.0956 / 0.1852 / 26.0902	0.0925 / 0.1846 / 25.6368
1024	0.1214 / 0.1938 / 30.1041	0.0997 / 0.1883 / 26.8436	0.0930 / 0.1846 / 25.7035	0.0923 / 0.1846 / 25.6101
2048	0.1079 / 0.1917 / 28.1638	0.0930 / 0.1846 / 25.7035	0.0923 / 0.1846 / 25.6101	0.0923 / 0.1846 / 25.6101

2:1 cache rule 에 의해 cache size N 인 1-way 가 cache size N/2 인 2-way 와 miss rate 가 비슷한 것을 확인할 수 있습니다.

sets 수와 associate 가 증가할수록 AMAT 가 감소하는 것을 확인할 수 있습니다.

sets 수가 작을 때는 associate 에 따른 AMAT 차이가 큽니다. 하지만 sets 수가 커질수록 associate 에 따른 AMAT 차이가 미미해집니다.

SIMULATION 4 . Block size

li benchmark

Block size	Unified Cache Miss rate	AMAT
16	0.0679	14.58
64	0.0146	3.92
128	0.0062	2.24
256	0.0031	1.62
512	0.0004	1.08

vortex benchmark

Block size	Unified Cache Miss rate	AMAT
16	0.1018	21.36
64	0.0244	5.88
128	0.0096	2.92
256	0.0066	2.32
512	0.002	1.4

tomcatv benchmark

Block size	Unified Cache Miss rate	AMAT
16	0.16	33
64	0.0462	10.24
128	0.0268	6.36
256	0.013	3.6
512	0.0077	2.54

block size 가 증가하면 AMAT 가 감소합니다. 하지만 block size 의 증가율에 비해 AMAT 의 감소율은 미미합니다. 따라서 block size 는 일정 수준까지만 증가시키는 것이 좋습니다.

■ Discuss about the best fit cache configuration for each benchmark program and compare and discuss about the differences among the configurations

cache size 가 극단적인 상황을 배재하고 생각해보면,

SIMULATION 2 에 의해 L1, MM 만 존재할 때보다, L1, L2, MM 3 계층으로 이뤄져 있을 때 AMAT 가 줄어드는 것을 확인할 수 있습니다. 또한 L1 보다 L2 의 크기가 클 때 더 효율적입니다.

SIMULATION 3 에 의해 적절한 cache size 에서는 associate 를 늘리는 것이 AMAT 감소에 도움을 주는 것을 확인할 수 있습니다.

SIMULATION 4 에 의해 block size 가 16->64 에서 AMAT 가 제일 크게 감소하는 것을 확인할 수 있습니다.

따라서 L1, L2, MM 로 이루어진 메모리에 block size 64, 8-way splited cache 를 사용하는 것이 유리하다고 할 수 있습니다.

고찰

IM 의 경우 branch 로 이전 주소의 instruction 을 반복해서 사용하는 경우가 많으므로 compulsory miss 를 제외하면 hit 가 나는 경우가 많았습니다.

DM 의 경우 IM 의 비해선 miss 가 많이 나서 LOWER LEVEL cache 로 접근하는 경우가 많았지만, benchmark 에 따라서 hit rate 면에서 차이가 있었습니다.

AMAT 관점에서 바라보면,

- unified cache 보다 splited cache 가 효율적입니다.
- L1, L2 MM 3 계층일 때 더 효율적입니다.

  비용 측면도 고려한다면, L1I / L1D / L2 는 16 / 16 / 512, 32 / 32 / 256 에서 제일 효율적입니다.
- associate 가 클수록 효율적입니다.

  하지만 sets 크기가 크다면, associate 를 늘림으로써 얻을 수 있는 효율이 미미해집니다.
- block size 가 클수록 효율적입니다.

  block size 가 16->64 로 증가할 때 AMAT 측면에서 이득을 가장 크게 볼 수 있고, 64 보다 크게 증가시킬 땐 AMAT 감소율이 미미해집니다.

Reference

컴퓨터구조실험 / 광운대학교 / 이혁준 교수님 / CA_Lab_12

컴퓨터구조실험 / 광운대학교 / 이혁준 교수님 / CA_Lab_13